DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2019

Automatic tag suggestions using a deep learning recommender system

DAVID MALMSTRÖM

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Automatic tag suggestions using a deep learning recommender system

David Malmström

Master in Computer Science
Date: January 2, 2020
Supervisor: Johan Gustavsson
Examiner: Viggo Kann
School of Electrical Engineering and Computer Science
Host company: Cantemo AB
Swedish title: Automatiska taggförslag med hjälp av ett rekommendationssystem baserat på djupinlärning

Abstract

This study was conducted to investigate how well deep learning can be applied to the field of tag recommender systems. In the context of an image item, tag recommendations can be given based on tags already existing on the item, or on item content information. In the current literature, there are no works which jointly models the tags and the item content information using deep learning. Two tag recommender systems were developed. The first one was a highly optimized hybrid baseline model based on matrix factorization and Bayesian classification. The second one was based on deep learning. The two models were trained and evaluated on a dataset of user-tagged images and videos from Flickr. A percentage of the tags were withheld, and the evaluation consisted of predicting them. The deep learning model attained the same prediction recall as the baseline model in the main evaluation scenario, when half of the tags were withheld. However, the baseline model generalized better to the sparser scenarios, when a larger number of tags were withheld. Furthermore, the computations of the deep learning model were much more time-consuming than the computations of the baseline model. These results led to the conclusion that the baseline model was more practical, but that there is much potential in using deep learning for the purpose of tag recommendation.

Keywords: Deep Learning; Recommender System; Collaborative Filtering; Matrix Factorization

Sammanfattning

Den här studien genomfördes i syfte att undersöka hur effektivt djupinlärning kan användas för att konstruera rekommendationssystem för taggar. När det gäller bildobjekt så kan taggar rekommenderas baserat på taggar som redan förekommer på objektet, samt på information om objektet. I dagens forskning finns det inte några publikationer som presenterar ett rekommendationssystem baserat på djupinlärning som bygger på att gemensamt använda taggarna och objektsinformationen. I studien har två rekommendationssystem utvecklats. Det första var en referensmodell, ett väloptimerat hybridsystem baserat på matrisfaktorisering och bayesiansk klassificering. Det andra systemet baserades på djupinlärning. De två modellerna tränades och utvärderades på en datamängd med bilder och videor taggade av användare från Flickr. En procentandel av taggarna var undanhållna, och utvärderingen gick ut på att förutsäga dem. Djupinlärningsmodellen gav förutsägelser av samma kvalitet som referensmodellen i det primära utvärderingsscenariot, där hälften av taggarna var undanhållna. Referensmodellen gav dock bättre resultat i de scenarion där alla eller nästan alla taggar var undanhållna. Dessutom så var beräkningarna mycket mer tidskrävande för djupinlärningsmodellen jämfört med referensmodellen. Dessa resultat ledde till slutsatsen att referensmodellen var mer praktisk, men att det finns mycket potential i att använda djupinlärningssystem för att rekommendera taggar.

Nyckelord: Djupinlärning; Rekommendationssystem; Collaborative Filtering; Matrisfaktorisering

Contents

1 Introduction

Recommender systems are algorithms or computer programs that provide users with suggestions of objects that might be of interest. For example, a recommender system could be recommending the next film to watch, or suggesting what items could be bought in addition to one that is currently being bought ("other users also bought this"). Usually, the recommendations are based on information about the user or the item that is recommended, or both. Recommender systems are presently being used by most information-based companies, for example Google, Amazon or Netflix [48, 53]. This means that the recommender systems are encountered many times per day by most people. The field of researching and developing recommender systems concerns itself with finding and evaluating the most efficient methods. As the applications and objectives, of a recommender system can vary greatly, there exists many different types of models. Some of them might be more specialized at finding new, more unknown items, while other might primarily recommend the most popular items. Compared to other information technology research fields such as databases or search engines, the field of recommender systems emerged relatively late with dramatic increase of interest in recent years [53]. One prominent inspiration for the development of recommender systems comes from the observation that many of the daily choices that a person makes come from suggestions from other people in some form, and the wish to make this automated. Instead of delivering a list of all possible options, recommender systems narrow down the amount of choices that a user is given. Therefore, they are also an effective way of dealing with the problem of information overload [14].

The research area of *tag recommender systems* concerns itself with being able to recommend appropriate tags to users. Tags can be described as words added freely by users to objects. An example of the use of tags would be at a website with user-uploaded photos, where a user uploads a photo of a beach. The user might add any relevant tag of choice to the photo, for example *ocean*. This helps other users find the photo, for example while browsing ocean-related photos using a search filter.

Deep learning is a family of machine learning methods that can learn complex representations of data. This project investigates the potential of using deep learning for a tag recommender system.

1.1 Motivation

As mentioned in section 1, recommender systems are effective at reducing the amount of information displayed. An application example would be the vast amount of video that is uploaded every day, 65 years of video content, just on

YouTube, as of February 2017[1]. Given a list of all this content, how could a sound decision regarding what to watch possibly be made? One option could be to look through the first dozen videos and then pick the most interesting video among them. With millions and millions of objects available, the chances that the set of videos that are easily surveyable contains something of actual interest would be very low. This type of problem exists for a great deal of applications. It is therefore paramount that some kind of effective filtering can be made, which is where a recommender system would come in.

The main idea of tags is to identify those words that represent the objects best and associate the objects with these. Having a proper coverage of tags on the items in a database will facilitate organization, description, browsing and retrieval [41]. The problem is though, that the process of manually adding tags to items could be repetitive and tedious, which might discourage users from actually doing it. Refraining from adding some important tags could be detrimental to the functional usage of the items. An example would be to forget to add the word *vacation* to a photo of a beach (in the case where it would be appropriate). An organization or retrieval of photos related to vacation would now not utilize this photo. Tag recommender systems can be used to help users in the process of tagging, suggesting relevant tags for the user to add, thus increasing the quality of the operations making use of the tags.

Deep learning is a growing field that is constantly being applied for new, untried, purposes, displaying good results in a variety of application [37]. It is therefore of interest to see how deep techniques can be used for tag recommendation.

1.2 Project Overview

This project was conducted at Cantemo AB in Stockholm. Cantemo provides a cloud-based video hub where users among other things can add their own tags to videos and images. Cantemo is not interested in a multiple user recommender system where recommendations can be tailored to a particular user. Therefore, a limitation of the project is that collaborative patterns and recommendations based on what other users have done will not be included. This shifts the focus of the project towards content-based recommender system methods. Recommender systems can be divided into two different types, collaborative filtering methods and content-based methods. The collaborative filtering methods are based around trying to find users with similar interests, and recommending items based on these users. One technique is called matrix factorization. The content-based methods makes recommendations based on similarities between items and their content.

[1] `https://mashable.com/2017/02/27/youtube-one-billion-hours-of-video-daily/`

This can be established using a classifier, for example a naive Bayes classifier in the case of tag recommender systems. To achieve the best results, collaborative filtering and content-based methods are often combined into what is called a hybrid recommender system. In tag recommender systems, collaborative filtering can be used to model the item-tag relationship, meaning that a hybrid tag recommender system can include a collaborative filtering approach without involving any users. The aim of this project is to investigate the potential of deep learning for tag recommender systems. A deep learning tag recommender system was developed alongside a baseline hybrid tag recommender system. The baseline model was using the conventional techniques of matrix factorization and Bayesian classification. The performances of the recommender systems were measured using the metric recall, and the results were compared.

1.3 Problem definition

How good is the prediction recall of a deep learning tag recommender system compared to a hybrid tag recommender system based on Bayesian classification and matrix factorization, if the recommender systems are not based on user data?

2 Background

In this section, the theory relevant for this project is presented. After that, the ethical and societal aspects as well as the sustainability of tag recommender systems are discussed. Finally, the related works in history as well as the state-of-the-art are summarized.

2.1 Theory

In this subsection, the underlying theory of the fundamental concepts treated in this thesis is described. First, the basic recommender system methods such as collaborative filtering and content-based filtering are explained, as well as important implementations of them, for example the collaborative filtering method matrix factorization. Then, the theory of neural networks and deep learning is discussed. Complications in the recommender system domain relevant to the project subject are also introduced, such as cold-start, personalization and implicit data. Finally, the methods of evaluating and optimizing recommender systems are described.

2.1.1 Collaborative filtering

Collaborative filtering is a recommender system technique which bases the recommendations on creating a database of the ratings of items by users. Suppose that person A has a similar preference on certain items as person B. The idea is then that person A is more likely to appreciate an item which person B has liked in the past, instead of a completely random item. The user-item rating database is often constructed as a matrix with the users on one axis and the ratings of the users on the other.

There are two different types of collaborative filtering, memory-based collaborative filtering and model-based collaborative filtering:

- *Memory-based* collaborative filtering methods simply uses the raw ratings of items by users and computes similarities between items or users using techniques such as Pearson correlation or cosine similarity [56, 25]. Recommendations are then made based on the most similar items or users. Often times however, the users will not have nearly rated all the items in the database, limiting the performance of this approach.

- *Model-based* collaborative filtering methods uses data mining algorithms, for example clustering or Bayesian networks to infer the missing ratings in the user-item matrix and use these to provide recommendations [56]. Another widely used model-based method for predicting the missing ratings is matrix factorization, which is further described in section 2.1.4.

Collaborative filtering can either be based on similarities between users or similarities between items. The user-based collaborative filtering methods can be described as the following: When a new user Mary is encountered, the method uses the preference database to look up similar users, and provides recommendations to Mary based on what these users liked. The item-based collaborative filtering methods on the other hand, uses similarities between items rather than users. Suppose Mary has liked an item previously. Then the method will look up what other items are similar in the sense that the same people have rated them similarly, and recommend these to Mary. The main advantage with this method is the performance of it when there are many users with high activity in the system. In that scenario the item-based approach is much faster than the user-based approach, because the search for neighboring users would be very time-consuming. A more detailed description of user-based collaborative filtering and item-based collaborative filtering are given in [25] and [56], respectively.

2.1.2 Content-based filtering

Content-based filtering is a recommender system technique which as the name implies takes the content of the items into account when generating the recommendations. The method creates representations of the items along with a profile of the user's interests. As the types of items can vary greatly depending on the application of the recommender system, there are many different ways to create the item representations. An example would be if an item contained a text description (movie synopsis), then a representation of the item could be the TF-IDF-vector of the text [49]. The profile of a user is constructed using the history of the user's interactions with the recommender system. Usually the profile is in the form of a classification or regression model where the training data is the user interactions [1]. Further, more detailed, information regarding the content-based filtering can be found in [46].

2.1.3 Hybrid recommender systems

The techniques of two or more recommender systems can be combined in various ways to create a more complex system, which is called a hybrid recommender system. A common practice is to combine the methods of a collaborative filtering recommender system with the methods of a content-based filtering recommender system. These make use of both collaborative user data and the information that the item contents can provide. The combination of several techniques allows for better performance, possibly at the cost of increased computation as the system gets more complex. Further information regarding hybrid recommender systems can be found in [11].

2.1.4 Matrix factorization

As mentioned in the collaborative filtering section, section 2.1.1, matrix factorization is a model-based collaborative filtering algorithm. This means that the goal is to use the information of the known ratings in the user-item matrix to fill in the user-item matrix pairs which do not have any ratings. Matrix factorization is a technique where the user-item matrix is decomposed into two smaller matrices, one representing the users and one representing the items. These two matrices have a common dimension called the latent dimension, which is typically much smaller than the number of users or number of items. The latent dimension is the size of the latent vector space, which represents theoretical discriminatory features that control the correlations between the users and the item ratings. The decomposition is done using dimensionality reduction techniques such as Single Value Decomposition or Principal Component Analysis [9]. The multiplication of the two decomposed matrices approximates the original ratings, filling in the blanks [33]. Generally, it is not straight-forward to interpret what actually constitutes the dimensions of the latent vector space. To help with the intuition in the case of a person-movie ratings database, one could imagine the latent vector space as different types of film genres that the users like less or more. It should however be stressed that this does not necessarily have to be the case at all.

2.1.5 Naive Bayes classifier

A naive Bayes classifier is a probabilistic model for performing classification [44]. Given n features and a classification problem $\mathbf{x} = \{x_1, ..., x_n\}$, assuming strong independence among the features, the probability of class C_k can be expressed as

$$P(C_k|\mathbf{x}) = \frac{1}{Z} P(C_k) \prod_{i=1}^{n} P(x_i|C_k).$$

Z is a scale factor with a constant value for the different classes. At classification time, Z is disregarded and the class with the highest value is chosen. To calculate the values of $P(x_i|C_k)$, an assumption regarding the distribution of the features has to be made. Common options include a normal distribution for continuous data, a multinomial distribution letting the features be frequencies and a Bernoulli distribution treating the features as Boolean variables.

2.1.6 Artificial neural networks

Artificial neural networks (ANN) are computing systems inspired by biological neural networks [30]. The computing systems can learn to perform tasks and computations by considering examples. An ANN can among other things be trained to perform pattern classification, clustering and function approximation of any arbitrary function to any degree of accuracy [4]. An ANN is constructed of artificial

neurons, which are computational models taking an input, performing some linear computation with it and outputting a result, depending on some threshold function. An ANN can be seen as a weighted directed graph with the neurons being the nodes and the connections between the neuron input and output being the edges. ANN:s are divided into two types; feed-forward networks, in which the graphs have no loops, and recurrent (or feedback) networks, where loops in the graphs are allowed and are used to provide feedback. A common type of the feed-forward networks is the multi-layer perceptron, where the neurons are grouped in layers where the edges only go in one direction from one layer to another. In a multi-layer perceptron, all neurons in a layer have a connecting edge to all neurons in the previous and following layers. The layers are therefore said to be fully connected. The first layer is called the input layer and the last the output layer. All layers in between are called hidden layers. In feed-forward networks, the weights on the neurons are usually determined using the method of backpropagation [55]. Backpropagation is the process of defining an error function (often also called a loss function) for the output of the network and finding the weights that optimizes it using an optimization algorithm such as gradient descent. This is what is referred to when a feed-forward neural network is said to be trained. An important parameter in many optimization algorithms is the learning rate, which essentially controls how much the optimization algorithm shifts the weights at each update. The way the error function is defined can be varied quite considerably, the most important thing being that it has some properties that makes the optimization possible. Common functions include the mean squared error loss for regression type problems, and cross-entropy loss for classification problems. The choice of error function is linked to the output function of the final layer of the neural network, and care has to be taken to ensure that they work well together. Common output functions include a sigmoid function like the logit for binary classification or the softmax function for classification into several classes [20].

2.1.7 Deep learning

In many applications of machine learning methods, including ANN:s, the developers have to carefully engineer a feature extractor. The purpose of the feature extractor is to transform the raw data into a suitable internal representation which the learning subsystem, often a classifier, can make use of. Constructing such a feature extractor could take a substantial amount of effort and time. One way to get around this problem is to use deep learning methods [37]. These methods learn the different types of content representation not from design by humans, but through a general-purpose learning procedure. They are representation learning methods with multiple levels of representation.

A deep neural network (DNN) is an ANN with multiple hidden layers. A greater

amount of hidden layers allows for implementations of more intricate functions of its inputs. Deep learning methods have seen great success recently, as stated in [37], beating other state-of-the-art machine-learning techniques in for example the fields of image and speech recognition and natural language understanding. As the required amount of engineering by hand is considerably smaller for deep learning techniques than many other techniques, they can more easily take advantage of the recent increase in amount of data. The main successes have been in supervised learning, but it is perhaps in unsupervised learning that deep learning will be the most important in the long term. The reason for this is that there exists much more unlabelled data than labelled data. One type of an unsupervised DNN is an autoencoder. At training time, the autoencoder is a combined encoder-decoder pair. The first part, the encoder, breaks down the input data into a smaller representation. The second part, the decoder, takes this representation and tries to reconstruct the original input as the output. The autoencoder is thus trying to learn the identity function.

As the field of deep learning has grown considerably recently, a large amount of new discoveries concerning how to make them more effective have emerged. The use of the threshold (also called activation) function rectified linear unit (ReLU) alleviates the need for unsupervised pre-training for sparse DNN:s, as described by Glorot et al. [18]. Srivastava et al. [61] developed a technique they named Dropout. They found out that using special layers that drops units and their connections randomly during the process of training works as a regularizer and prevents the networks from overfitting. Another technique is batch normalization, which is the method of performing batch-wise normalization of input to hidden layers, allowing faster training with higher performance and at the same time providing regularization [29]. The batch refers to that in the backpropagation, weights are updated according to an average value over a batch of training examples instead of after each individual training example. This is called mini-batch gradient descent [54]. An optimization algorithm for backpropagation that has produced good results and become very popular recently is Adam [31, 54], which is a variation of stochastic gradient descent with an adaptive learning rate.

2.1.8 Tag recommender system

A tag recommender system is a recommender system which recommends tags to the user. In this context, a tag is defined as a word freely added to an object by a user. Usually there is no limitation on the amount of tags that can be added to an object. Adding tags provide descriptions and organization of objects, and facilitates searching among objects. A tag recommender system generally uses a database of objects with tags to generate new tags, either for new objects without any tags or for objects which already have tags. There are two main sub-problems in the

tag recommendation problem [5]. There is the object-centered problem, and the personalized problem. The *object-centered problem* is about analyzing a specific object. Tag recommender systems based on the object-centered approach aims to suggest tags that are relevant to the object and will recommend the same tags for an object regardless of the user. The *personalized problem* consists of taking the target user into account as well, in addition to making suggestions fitting the object. This means that the recommendations can vary depending on the history of the interactions with the recommender system, and that different users might get different recommendations on the same object. The personalized problem is what will be treated in this thesis, and can be formulated as follows (from [5]):

Given a set of input tags I_o associated with the target object o, generate a list of candidates C_o ($C_o \cap I_o = \emptyset$), sorted according to their relevance to both user u and object o, and recommend the k candidates in the top positions of $C_{o,u}$.

The set of input tags I_o can be empty. This is known as the cold-start problem, which is described further in section 2.1.9. As with most types of recommender systems, tag recommender methods can be divided into the two different approaches described in section 2.1.1 and section 2.1.2, content-based methods and collaborative filtering methods [28].

The content-based tag recommendation implicates no special additions to the usual content-based methods, recommendations are made based on what content can be extracted from the item. An example could be a multi-label classifier classifying the content information into one or several labels (the tags to be recommended). To achieve this, one binary classifier (for example a support vector machine or a naive Bayes classifier) per tag can be used. The recommender system chooses the tags with highest confidence scores from their respective classifiers when making predictions. This is called the one-vs.-rest approach of multiclass classification and was for example used for tag recommendation in Illig et al. [28].

The collaborative filtering methods for tag recommendation are interesting in the sense that the tags provides an extra dimension compared to the standard binary user-item relationship that exists in regular collaborative filtering. The ternary relationship between users, items and tags results in that the usual techniques have to be replaced or modified, for example using tensor factorization instead of matrix factorization [50]. In many practical applications however, tags are not unique to users, but are instead public and shown to everyone. In these scenarios it would not be possible or make much sense to model the user-item relations. However, instead of immediately resorting to content-based methods, focus can be shifted towards modelling only the item-tag relationships, often referred to as the tag co-occurrence.

Tag co-occurrence is the idea that if a tag frequently appears attached to the same item as another tag, the two tags are most likely related and can be added to

the same items. Tag recommender systems based on exploiting tag co-occurrence are often also called *associative tag recommenders* and have consistently been producing state-of-the-art performances [43]. The idea to model the item-tag relationships in a collaborative filtering fashion has been explored. That is, using the collaborative filtering algorithms, but with the typical user-item matrix being replaced by a item-tag matrix. The idea is that this kind of model can efficiently capture the tag co-occurrence patterns among tags that have been previously added in order to provide new suggestions [65, 66]. This way of modelling co-occurrence is of the same nature as the way it is modelled in the tensor factorization method [50] mentioned above. The difference is that in the latter case an extra dimension representing the users is included.

Tag recommender systems can be combined into one more complex hybrid tag recommender system, just as the regular hybrid recommender system. Figure 2.1 shows a hybrid tag recommender system taking both tag information and item content information into account.

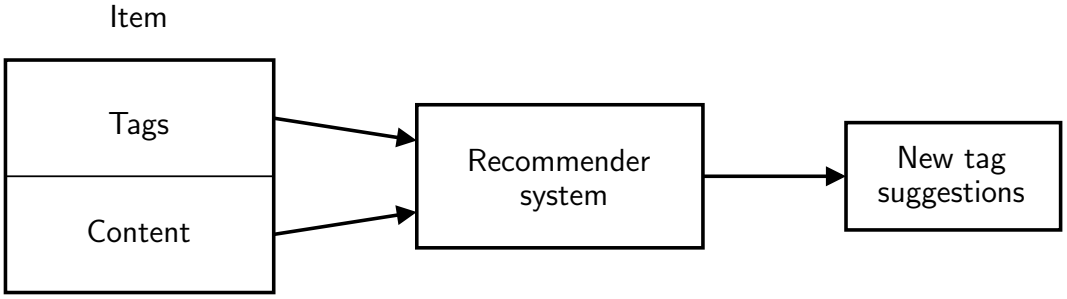


Figure 2.1: The hybrid tag recommendation procedure. Tag information (from already existing tags) and content information are both taken into account to produce new tag suggestions for an item.

2.1.9 Cold-start

A common problem for recommender systems is the cold-start problem, also called the out-of-matrix recommendation problem. This problem can emerge in several ways depending on the context. In the case of collaborative filtering based on ratings, the cold-start problem refers to the situation when the recommendations are to be based on a new item which no one has yet rated. It can also refer to the situation when a new user who has not yet rated anything is introduced. Pure collaborative filtering can not do anything in these cases [57]. Content-based methods are often not subject to these problem, since they do not rely on ratings. However, there might still be complications if they are based on what a user has done in the past. The cold-start problem is generally solved by taking into account additional information about users and items [16]. For collaborative filtering methods, this is commonly done by adding a content-based part of some

sort to the recommender system, effectively creating a hybrid system as described in section 2.1.3.

The cold start problem in tag recommender systems needs to be addressed separately. In this case, cold start refers to that an item, which are to be provided relevant tag recommendations to, does not have any previously added tags. It is easy to realize that for associative tag recommender systems, which rely purely or heavily on the tags that have been added previously to the item, this will be a problem.

2.1.10 Personalization in tag recommender systems

As mentioned in section 2.1.8, a personalized tag recommender system takes the target user information into account, as well as making the suggestions fit the particular object that the recommendations are target at. Personalized tag recommender systems uses a user's past tagging behaviour to recommend new tags [62]. Another way to formulate a definition is to say that a personalized recommender system will not always give same recommendations to two different users. It is however not necessary for a system to have several users in order to be personalized, the suggestions could be based on the history of the one unique user. Most collaborative tag recommender systems are personalized by default. This is because they take the user information into account when making suggestions based on what similar users have liked (user-based) or based on the items that the user has liked (item-based). A personalized approach will typically outperform a non-personalized approach [50]. However the quality of the recommendations can degrade if the user changes their tag preferences but keeps getting recommendations based on their own behaviour [62].

2.1.11 Recommendations using implicit data

When describing the methods of collaborative filtering in section 2.1.1, the user preference modelling was performed using item ratings, where for example an item gets a high rating if a user likes it and a low rating if a user dislikes it. This system works well and describes both the user's positive and negative preferences in a convenient way. But what happens if the recommendation problem at hand does not contain any ratings? An example could be a photo browsing website where users do not rate the photos, but just do or do not click on them when they appear as suggestions in the user interface. This type of preference feedback is referred to as *implicit* feedback, while the other kind with the ratings is referred to as *explicit* feedback [27]. Implicit feedback could be many different things. In addition to a click it could for example be a like, a search word or even mouse movements.

The task of using this implicit feedback to infer user preference is not straight-

forward. Hu et al. [27] mentions some prime characteristics of implicit feedback:

- There is no negative feedback. It does not mean that a user disliked a video just because it did not click on it, the user might just have missed that it was there for example. However, this type of feedback must be taken into account, because the only other feedback available is the positive examples. Using only the positive examples which will greatly misrepresent the user profile, often leading to severe bias of the recommendations [24].

- The implicit feedback is noisy, in the sense that it can not be known what the user thought about an item even if the user clicked on it. The user might have disliked it afterwards, or in the example of an online store and a purchase; might just have bought an item for a friend.

- A large amount of implicit feedback on an item does not necessarily imply that the item is more liked than an item which only has a lesser amount of implicit feedback. This is contrary to the way that a higher explicit rating always is better than a lower explicit rating.

Based on these characteristics, Hu et al. proposed an approach to generating recommendations on implicit datasets. The approach is a variant of matrix factorization where missing data (items not interacted with) is taken into account in the factorization process. The positive data is multiplied with a confidence level factor in order to give it a higher weight than the missing data. This confidence level factor is determined using hyperparameter optimization, which is further described in section 2.1.13.

In the case of tag recommender systems, the problem of implicit data is very frequent. Typically, the only feedback that the system receives is if a user has used a tag or not, i.e. only the positive feedback [51]. Once again, one approach to handle this is to use the missing data as negative data.

2.1.12 Evaluation of recommender systems

The performance of a recommender system can be evaluated in different ways using performance metrics. The choice of which performance metric to use has to be carefully considered, as its effectiveness depends on the application. A common way to evaluate a recommender system is to treat the system as a classifier, modelling the items to be recommended as classes. The performance metrics for classification can then be used to evaluate the recommender system, as mentioned in [3] by Amatriain et al. Other ways to measure the performance of recommender systems include rank-based metrics like Mean Reciprocal Rank, or rating/score-based metrics like Mean Average Error. An example of a performance metric for classification would be accuracy, which Amatriain et al. states is the most

commonly used metric for model performance. In the paper it is defined as "the ratio between the instances that have been correctly classified (as belonging or not to the given class) and the total number of instances". This metric is troublesome if the number of instances per class is imbalanced, leading to that the classes with much fewer instances might be ignored. Other common metrics include precision or recall. Precision is a measurement of the classifiers' ability to not positively label a sample that is negative. Recall is a measurement of the classifiers' ability to find all positive samples. In particular, tag recommendation can be thought of as multi-label classification, with the labels being the tags. Suppose a prediction of a number of tags for an item were made. The precision would then be the number of correct predictions divided by the total number of predictions made. The recall would be the number of correct predictions divided by the the number of relevant tags.

2.1.13 Hyperparameter optimization

For many machine learning applications including recommender systems, there are some parameters of the model that can be adjusted. These parameters could for example affect the speed at which the model learns the data, or the performance of the model at the point of evaluation. Examples of hyperparameters could be the number of latent factors in matrix factorization or the size or number of hidden layers in a deep neural network. One model can have several hyperparameters, and it is of great interest to pick the best combination of them in order to get as high performance as possible. Therefore, trying out different combinations of hyperparameters and evaluating how well they work is a crucial process of the construction of a model. The optimization of the hyperparameters can be done manually, adjusting the parameters according to the evaluation results as they come. In many cases, especially when the amount of hyperparameters are many, this process is a bit tedious and extensive. The process is traditionally automated by using what is called grid search. This is an exhaustive search for the best parameters over a predefined subset of the hyperparameter space, changing the parameters incrementally and methodologically. However, Bergstra and Bengio [7] has published results suggesting that using a randomized search for the parameters over a subset of the hyperparameter space is more effective. Another way to optimize the hyperparameters is to view the tuning of the learning algorithm as the tuning of a black-box function. This allows the use of Bayesian optimization [60]. This is the process of attempting new hyperparameter configurations based on the results from previous attempted hyperparameter configurations, calculating the new configurations using Bayesian statistics.

When performing the optimizations of the hyperparameters, in order to get a trustworthy estimate of its performance, it is important to ensure that the evalu-

ation of the model is made on data that represent the data that the model would be used on in practice. A common way to do this is to use cross-validation, where the dataset used for training is split into two parts, one for training the model and one for evaluating the performance. In order to avoid bias in the part that is chosen for evaluation, a common variation is k-fold cross-validation. This is the process of separating the training dataset into k folds and for each fold, train the model on the remainder of the dataset. The model is then evaluated on the fold, and in the end the performance is averaged over each fold [32].

2.2 Societal aspects, sustainability and ethics

Recommender systems are encountered many times on a daily basis by most people, as mentioned in section 1. Because of this, they have a great influence on society as a whole, through their ways of narrowing down the information presented.

With the web 2.0, the amount of choices available in society has been increasing considerably. An example would be the limited range of shoes in a physical store, compared to the thousands of choices offered in an online store. Over-choice makes it hard to get an overview and decide what is actually a good choice, especially if the choices are very different. In a meta-analysis, Chernev et al. [14] concludes that no definitive conclusion has been made in general regarding if large assortments benefit or are detrimental to choice. However, studies have concluded that displaying a smaller set of high-quality results gives a higher choice satisfaction compared to displaying a larger set of item of the same high quality [10, 38, 22, 58]. In other words, many things to choose from makes the choice harder and the satisfaction lower, even though the same item was chosen in the end. Because of this, it can be argued that developing more effective recommender systems results in more satisfied people, a socially sustainable development. A company centering parts of their service on a recommender system would then benefit economically by this as the service might get more popular the more satisfied people are.

Another societal aspect of recommender systems is information segregation. The technological changes of the 21th century have made it possible for members of society to access all the information they want, almost at any given time. In theory this should diversify the information that people take part in, since much more information from more diverse places is available. However, a current concern is that members of society through all of the personalized recommender systems are exposed to the same types of information over and over. This phenomena is usually called "filter bubbles" or "echo chambers". Recent simulations indicate that homogeneity among user behavior is increased when recommender systems are used [12]. However, there has been much discussion whether the information segregation actually is happening. Recent studies points towards that the concern might be exaggerated and that effects from both sides, while existing, are relatively

modest [15, 21]. If recommender systems are a cause for information segregation, an increase in recommendation effectiveness could harm the way information is distributed in society, possibly segregating groups of people and stirring up conflicts. In this case, it can then be said that recommender systems are a cause for unsustainable social development.

Another concern when it comes to personalized recommender systems, as expressed by Lam et al. [36], is that it might be possible to figure out information about a user through the recommendations. This breaks the user's integrity. The information could then be used in malicious and unethical ways without the consent or knowledge of the user.

Machine learning and especially deep learning methods can be very computationally demanding, with training times being long even on the most efficient computers. The fact that running powerful computers consumes a considerable amount of energy raises the issue of how ecologically sustainable it is to develop and run these algorithms. For a sustainable future, care needs to be taken to ensure that the methods are as efficient as possible and that the energy is renewable. There are some research being done concerning this, for example presented in the annual Low-Power Image Recognition Challenge [2]. The main purpose of this challenge is said to be to develop models that can be used in battery-powered devices without consuming all of the limited resources. However, the solutions can certainly be used with the purpose of limiting the energy consumed by society in general.

Lastly, treating the question of whether more effective recommender systems cause economically sustainable development, the previously mentioned company benefiting economically from improved recommendations can be taken as an example. It can be argued that more effective recommender systems yields economically sustainable development for this company in the sense that economic growth is provided. However, there is another definition of economically sustainable development: that economic growth is realized without a negative impact on social and ecological sustainable development. If this is what is meant, then economically sustainable development can not be established as confidently. As mentioned before, neither social nor ecological sustainable development can be guaranteed when more effective recommender systems are developed, due to the possibilities of information segregation and increased power consumption.

2.3 Related work

What is considered to be the first recommender system [52] is a manual collaborative filtering system proposed in 1992 [19]. This caused the start of the development of automatic recommender systems. Herlocker et al. presented a user-based automated collaborative filtering method in 1999 [25], and Sarwar et

al. proposed an item-based model in 2001 [56]. During this time, different types of content-based methods have been used on their own or as a complement to the collaborative methods [46, 11]. The next big leap in recommender systems came when the Netflix prize [6] was announced in 2006. One million dollars was to be awarded to the first one to make the company's recommender system 10% more accurate. The large prize was however not the only impressive thing; for the competition, Netflix released a dataset of over 100 million ratings by over 480 thousand randomly chosen users for nearly 18 thousand movie titles. These two things caused a surge of research focused at the recommender system problem. One of the main results was that matrix factorization methods for recommender systems evolved and surpassed the more classical nearest-neighbor techniques in performance and popularity [33]. Matrix factorization become the standard approach to latent factor model-based recommendation [23]. Around the same time, in 2008, Hu et al. published their work on implicit feedback datasets [27], which came to be of big influence on the cases when rating feedback was not available.

With the advancements of machine learning and deep neural networks, several improvements and innovations have been made recently. A survey of deep learning based recommender systems is given in [72]. The article states that "Hence, in today's research climate (and even industrial), there is completely no reason to not use deep learning based tools for development of any recommender system". In 2014, Wang et al. proposed in the paper called Collaborative Deep Learning for Recommender Systems (CDL) [67] a deep learning hybrid system. It uses deep representation learning for the content information but still uses matrix factorization for the collaborative filtering effect. A similar approach is taken by Cheng et al. [13] in 2016. In 2017, He et al. [23] described a technique called Neural Collaborative Filtering (NCF) that uses neural networks to model the latent user and item features in collaborative filtering. They showed that matrix factorization can be interpreted as a specialization of the neural network that they developed. It was also showed that a deeper network yields better performance. Significant improvement over state-of-the-art methods was achieved on two datasets, one with explicit feedback and one with implicit feedback. The authors claim that the deep learning modelling of the collaborative filtering effect has not been done previously, and motivates the success of the model by saying that the usual inner product of the latent matrices of matrix factorization is not sufficient to capture the complex structure of the user interaction data. It is stated in the paper that the framework is simple and generic and not limited to the models presented in the paper. For example, in order to avoid the cold-start problem for collaborative filtering methods, content-based features can be included as input of the model. Since this is a rather recent paper, it could be considered the best current way to solve the general recommender system problem.

A state-of-the-art recommender system that is not based on neural networks is LightFM by M. Kula [35]. LightFM is a hybrid matrix factorization system which represents users and items as linear combinations of the item content, thus avoiding the cold-start/sparse data problems. As another investigation of the state-of-the-art methods for recommender systems in general, the results of the most recent ACM RecSys Challenge, the 2018 version [71], were examined. This year, the competition focused on the task of automatic music playlist continuation. More precisely, the task concerns itself with recommending a playlist of up to 500 music tracks that fit the characteristics of a playlist of arbitrary length with some additional metadata. The recommendations were based on the structure of music playlists in a database of a million user-created playlists from the Spotify[2] platform. Most of the best placing entries were based on either a neural network design, a matrix factorization design or a combination of them both. The first-placing team used a combination approach of matrix factorization, neural networks and learning-to-rank techniques. The second-placing team, "hello world!" [70], used an approach purely based on neural networks, using an auto-encoder, that they call Multimodal Collaborative Filtering (MMCF). This approach simultaneously analyses the playlist and its categorical contents (artists and albums of the songs). This circumvents the cold start problem as well as the popularity bias problem caused by relying too heavily on song co-occurrence. Unlike pure collaborative filtering which only extends playlists profiled at training time, this system creates a new playlist containing recommended songs.

In the more specific research area of tag recommender systems, collective co-occurrence-based methods were described independently in 2008 by Sigurbjörnsson and van Zwol, Heyman et al. as well as Garg and Weber [59, 26, 17]. These methods recommends an extension of tags to an already existing list of tags, for example on an item. They were all purely based on tag co-occurrence, not taking any item content into consideration, meaning that the methods alone can not handle a cold start scenario. In 2009, as a response to these associative tag recommendation models, Krestel et al. [34] used Latent Dirichlet Allocation (LDA) [8] to model the latent topics in the items, generating new tags based on these. This yielded better results than the three papers from 2008 and it is mentioned in the paper that combining the two different methods could further improve the recommendations. Rendle et al. [50, 51] described in 2009 and 2010 a method of generating tag recommendations using tensor factorization of the tag-user-item tensor. This method is thus collaborative in the user-item sense but also taking into account the tag-co-occurrence relationships with the user-tag and tag-item dimensions. It is however not taking any content-information into account.

As for content-based tag recommendations, Illig et al. presented in 2011 [28] a

[2] `https://spotify.com/`

comparison of several text-based classifiers in a cold-start scenario. Among other methods, they were using a support vector machine (SVM) as well as a multinomial naive Bayes classifier with the one-vs.-rest approach mentioned in section 2.1.8. In this study, the SVM had the best performance, with the multinomial naive Bayes classifier not far behind. Naive Bayes classifiers were also used in [17] and [69] as baseline methods, the latter paper motivating the use of it by good performance for text classification and low computational complexity. In 2009, Lu et al. [42] proposed a content-based cold-start method for the social bookmarking site Delicious[3], where tags were suggested based on the cosine-similarities of the websites. Another comparison of methods in the cold-start scenario was performed by Martins et al. [43]. In that paper, their own system, specialized on cold-start, is evaluated and compared against four baseline models based on either tag co-occurrence or content information. One of the baseline methods is [39], which won the competition ECML PKDD Discovery Challenge 2009[4]. It combines six basic tag recommendation methods into one, starting by generating tags from the title of an item and then through these tags generating more tags using tag co-occurrence. A deep learning content-based method for images was described by Nguyen et al. [45] in 2017, where the image features were extracted using a convolutional neural network and combined with user information to create a personalized recommendation. Tag recommender system research is also being conducted for software information websites such as StackOverflow and Freecode [68, 40, 69]. Here, each item is a post with a question and answer, the content being the texts.

Recently, it has become more common to create models which combines the content-based approach with the tag co-occurrence approach. An example of this is the method described by Wang et al. [65] in 2013, which they call CTR-SR. CTR-SR uses the *collaborative topic regression* (CTR) model described in [64] to combine a collaborative filtering model based on the item-tag relationship for the tag co-occurrence with an LDA model for the item-content. The method yielded satisfactory results on a dataset consisting of scientific articles. Wang et al. then proposed an improvement of this method in 2015 [66], using the deep learning model *stacked denoising autoencoder* (SDAE) [63] instead of LDA to model the item-content. Matrix factorization was used for the tag co-occurrence part. They call this model a *relational stacked denoising autoencoder* (RSDAE) and states that it outperforms the state-of-the-art.

In the current state-of-the-art of deep learning tag recommender systems, there are no systems at the moment which model the tag co-occurrence using a deep learning collaborative filtering approach as described in [23] (NCF). As described

[3]Delicious is now discontinued.

[4]`https://www.kde.cs.uni-kassel.de/wp-content/uploads/ws/dc09/`

above, the RSDAE model uses matrix factorization, and the deep learning tag recommender presented in [45] by Nguyen et al did not model the tag co-occurrence at all. Therefore, it is of interest to see how well a deep learning approach to modelling tag co-occurrence would perform, since it yielded such good results for collaborative filtering in general. Furthermore, most research has been focused on recommending tags on datasets where a describing text is available (for example article abstracts and film synopses [66, 65] or software-related questions [40, 69, 68]). More research on other types of content-information would therefore be of interest. Finally, because of the success of combining content information and tag co-occurrence information, it would be relevant to find out how this relationship can be jointly modelled using deep learning.

When it comes to evaluation, various types of performance metrics are used in the surveyed papers. Recall@M, recall of the top M recommended items, is most commonly used, for example in CTR, CDL and RSDAE [64, 67, 66], among several others [65, 34, 69, 26, 43]. NCF used hit-rate@10 as well as normalized discounted cumulative gain (NDCG), which was also used by MMCF [23, 70]. Other used metrics are the Jaccard coefficient [59], the precision [17, 26] and the mean receiver operating characteristics area under the curve (ROC AUC) [35, 50].

3 Methodology

In this project, an investigation of the the potential of using deep learning for recommending tags was performed. Two models for generating tag suggestions were developed, a deep learning model and a baseline model. These models were trained and evaluated on a dataset where each item contains a number of tags along with additional content information.

In this section, an overview and a justification of the methodology is given. Then, the materials used in the project are described. After that, the procedures of the project are listed in the order that they were performed. The main procedures were *implementation of models*, *hyperparameter optimization* and *final testing*. The models are described in depth. Finally, the evaluation approach is described.

3.1 Overview

The goal of the research question formulated in section 1.3 was to investigate the potential of a deep learning tag recommender system. The methodology to do this was to implement two different models and compare the performance of them. The models were defined as the following:

- The *deep learning model* was a deep learning hybrid (content- and tag co-occurrence-based) tag recommender system. The tag co-occurrence part was modelled using a collaborative filtering-like approach similar to the one described in [66]. The content and tag co-occurrence features were used in a deep learning recommender system where the architecture was based on that of NCF [23]. Adam was used as the optimization algorithm and binary cross-entropy loss was used as the error function of the optimization.

- The *baseline model* was a hybrid model combining content and tag co-occurrence. The tag co-occurrence part was modelled in a matrix factorization manner using a method described by Hu et al in [27]. The content-based part was a one-vs.-rest multinomial naive Bayes classifier.

These models were trained and evaluated on a Flickr dataset containing photo and video items with tags and other content information. Specifically for the evaluation, hyperparameter optimization was performed using k-fold cross-validation and recall@M will be used as a performance metric. The research hypothesis, based on the problem definition formulated in section 1.3, was:

The deep learning model will give tag predictions with a higher recall@M than the baseline model.

3.2 Justification

The collaborative filtering way of modelling the tag co-occurrence was motivated by the success of the technique in [50, 65, 66], as described in section 2.1.8. Furthermore, the scenario of extending a list of tags could be considered fairly similar to the automatic music playlist recommendation scenario from the RecSys competition [71], mentioned in section 2.3. The success of the song co-occurrence modelling using neural networks in the MMCF model [70] could therefore be seen as further support for the hypothesis that the tag co-occurrence could successfully be modelled using a neural collaborative filtering technique.

A motivation for the choice of using deep learning was that it is a growing field where models are constantly being tried for new purposes, displaying good results in a variety of applications, as mentioned in section 2.1.7. Researching where deep learning is effective can lead to discoveries of new state-of-the-art methods. As deep learning methods are currently being adopted with promising results in the general field of recommender systems [72, 23], it might be successful for tag recommendation as well. Furthermore, the success of using deep learning to model the inner product in matrix factorization in NCF [23] for general recommender systems motivated basing the neural network architecture on the architecture in NCF. It is of interest to see if this methodology generalizes well to tag recommender systems. NCF did not use the deep learning techniques of dropout or batch normalization, making it interesting to see if these would have a positive impact on the performance of the model. Batch normalization can be used to increase the training speed and slightly reduce overfitting, while dropout can significantly reduce overfitting. Adam [31, 54] was used in the deep learning model because of the good results it has been producing, as mentioned in section 2.1.7. Cross-entropy loss was used because the tag recommendation problem is modelled as classification. Both Adam and binary cross-entropy loss were also used in NCF.

The choice of using a variant of matrix factorization in the baseline model was motivated by that matrix factorization was used in the well-performing hybrid systems CTR-SR and RSDAE, both by Wang et al. [65, 66]. The method described in the paper by Hu et al. [27] was used since it handles the implicit feedback scenarios. The decision to use a naive Bayes classifier for the content part was based on the good results it got in [28] by Illig et al., and the fact that it has been used as a baseline in previous works [17, 69]. The reason for using a multinomial distribution when the features are in fact Boolean was that it has been shown that a binary multinomial naive Bayes model often outperforms a Bernoulli naive Bayes model [44]. Moreover, both [28] and [69] used a multinomial naive Bayes classifier for Boolean features. The content-based part of the baseline model could also be used separately to provide recommendations, meaning that a comparison between using and not using tag co-occurrence is possible to perform.

The Flickr dataset was used because it is public and similar to the dataset that was available and used internally at the principal Cantemo. Both datasets contained photo and video items that were annotated with user-tags. These tags can be used and predicted by the recommender systems in the tag co-occurrence sense. The items in both datasets also contained content information in the form of tags automatically added using computer vision, denoted by "autotags" in the Flickr dataset. The autotags represented the item content information used by the content-based methods, reinforcing the recommendations of new tags. The Flickr dataset is further described in section 3.3. Evaluation using k-fold cross-validation during the hyperparameter optimization was performed in order to ensure that the hyperparameters did not get biased by the choice of validation set. Training on a training set and making predictions on a validation set emulates the scenario where data for some items is already available and it is desirable to use this data to predict data for a number of new items. This is similar to what would happen in practice with a database of items when new ones are added. The choice of the recall@M as a performance metric was partly motivated by the fact that it is so widely used in the literature, facilitating comparisons. Recall@M is well suited for the application of recommending tags, measuring how well the algorithms can show as many relevant suggestions as possible, giving a potential user several high-quality options. It was also desired that it is possible to limit the amount of suggestions taken into account, so that not too many options were given. This is because having too many options is not necessarily a good thing, as mentioned in section 2.2. Therefore, values of M was restricted to the range of $0 - 20$ in the investigation.

3.3 Dataset

The dataset that was used in the project to test and evaluate the models was a subset of the Yahoo Flickr Creative Commons 100 million dataset[5] (YFCC100M). The YFCC100M dataset is a collection of 100 million Flickr photos and videos. All items in the dataset are associated with the Creative Commons[6] license. The photos and videos come with various types of descriptive metadata, for example user id, creation date, geoposition or different types of tags. Specifically of interest for this project was the tags, and in particular two different kinds of tags. Firstly, the "user-tags", tags that users had added manually when uploading the media to Flickr. Secondly, the "autotags", tags generated by the Flickr Vision team using a deep-learning approach analyzing the images or videos specifically for this dataset. The user-tags are what will be recommended by the recommender system. The autotags are what will be used as the item content information, reinforcing the

[5] `https://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67`

[6] `https://creativecommons.org/`

recommendations of user-tags. In the context of figure 2.1, the user-tags of the dataset represent the tags, and the autotags represent the content. The top five most common autotags and user-tags can be seen in table 3.1.

Table 3.1: Top 5 autotags and user-tags of the YFCC100M dataset and their count.[7]

Autotag	Count
outdoor	32,968,167
indoor	12,522,140
face	8,462,783
people	8,462,783
building	4,714,916

User-tag	Count
nikon	1,195,576
travel	1,195,467
usa	1,188,344
canon	1,101,769
london	996,166

The difference in the amount of counts between the autotags and the user-tags can be explained by that the set of unique autotags is much smaller than the set of unique user-tags.

3.4 Hardware resources

For the training and hyperparameter optimization of the deep learning model and general development of the models, a desktop computer with an Nvidia GeForce GTX 1070 8 GB graphics card, an Intel Core i5-6600K processor and 16 GB RAM was used. For the hyperparameter optimization of the baseline model, a virtual server computer with an Intel Xeon CPU X5670 2.93GHz (utilizing two cores only) and 16 GB RAM was used.

3.5 Software resources

In this project, all development was done using Python 3.6.7. The main Python libraries along with their version numbers were: Tensorflow 1.12.0, scikit-learn 0.21.2, Keras 2.2.4, pandas 0.24.2, Matplotlib 3.1.1, seaborn 0.9.0.

3.6 Preprocessing

Preprocessing of the dataset was the first procedure that was performed in this project. From the 100 million picture and video items of the database, the first 5 million items were preprocessed to a suitable input dataset for the models. The preprocessing consisted of:

[7] From `https://code.flickr.net/2014/10/15/the-ins-and-outs-of-the-yahoo-flickr-100-million-creative-commons-dataset/`

1. For each item, removing all data except for the *photo ids*, the *autotags* and the *user-tags*.

2. Removing items that contain an already seen (duplicate) set of user-tags.[8]

3. Removing user-tags that were camera brands (*Canon*, *Nikon* etc.) and year labels.[9]

4. Filtering out the 1,000 most used autotags and the 2,000 most used user-tags.[10]

5. Removing items that had less than 6 autotags and 6 user-tags.[11]

6. Randomly choosing 20,000 out of the remaining items to make up the final dataset, making sure that points 4 and 5 were fulfilled.[12]

The lists of autotags and user-tags of each item were then transformed into binary vectors. The vectors had a size of 1,000 in the case of the autotags and a size of 2,000 in the case of the user-tags. The non-zero elements of the vectors represent the presence of a particular tag for an item.

The resulting dataset that was used in the project was therefore a dataset of 20,000 items with each item containing only an item index, a list of autotags and a list of user-tags. There were 1,000 autotags and 2,000 user-tags. Each item had an average of 13.56 autotags and an average of 8.97 user-tags, and at least 6 autotags and 6 user-tags.

Of the 20,000 items, the first 2,000 formed the test set. The test set was held out during all model selection and hyperparameter optimization, and was only used for the final testing of the models. The set of the remaining 18,000 items was used as a development set.

3.7 Implementation of models

Following the preprocessing, the models were implemented. Descriptions of how the models are constructed can be seen in the following sections, section 3.8 and 3.9.

[8]This was done in [17] for a Flickr dataset with the the reasoning being that Flickr allows batch uploading with the same set of tags for all photos. This results in that some tags could get strongly associated as groups, making the systems always suggest the rest of the tags when some of them are present.

[9]This was done as these tags do not have anything in particular to do with the pictures, and were very common.

[10]This was done to avoid tags appearing on too few items.

[11]This was done to have enough tags on each item.

[12]This was done due to resource constraints.

Both models were hybrid recommender systems that takes an item with a certain number of tags as input and outputs a list of new tag suggestions. The systems were hybrid systems in the sense that they were using both tag co-occurrence among user-tags as well as item content information (in the shape of autotags) to generate the tag candidate lists. The first recommender system was the baseline model, using Bayesian classification for the content part and matrix factorization for the tag co-occurrence part. The other recommender system was the deep learning model, where both the content part and the tag co-occurrence part were implemented using artificial neural networks.

3.8 Baseline model

The baseline model was a hybrid tag recommender system (see figure 2.1), using a method called implicit collaborative filtering (Implicit CF) to handle the user-tags and a statistical approach denoted by Naive Bayes for the autotags. The layout of the model can be seen in figure 3.1. Implicit CF and Naive Bayes were both complete tag-recommendation models by themselves, the difference being that the Implicit CF model took only user-tags as input while the Naive Bayes model took only autotags as input. Both the Implicit CF model and the Naive Bayes model outputted a list of the top M user-tag predictions along with their score. In the baseline model, these resulting lists were then combined into a final list which formed the output, in a manner which is described in section 3.8.3. When fitting the baseline model, the Implicit CF model and the Naive Bayes model were both fitted separately. The Implicit CF model and the Naive Bayes model are further described in sections 3.8.1 and 3.8.2.

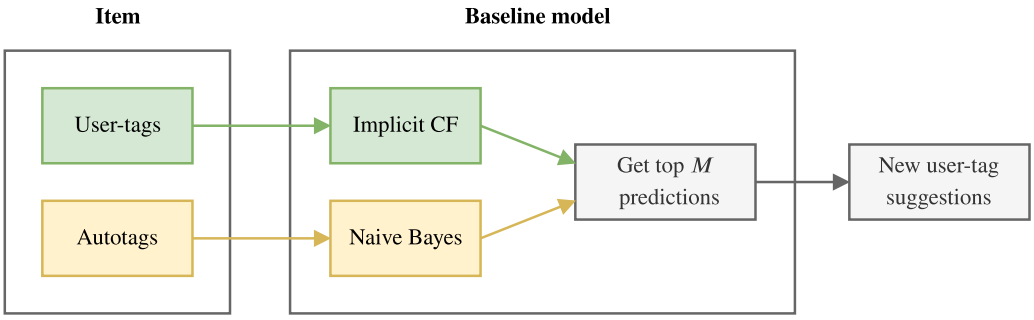


Figure 3.1: The baseline model. The user-tags and the autotags are fed into an implicit collaborative filtering (CF) model and a naive Bayes model, respectively. The M user-tag predictions with the highest score from both models are then used as the output for the entire baseline model.

3.8.1 Implicit CF

The model handling the user-tags was based on the "Alternating Least Squares"-model from the Implicit[13] Python library. This model was in turn based on the method described in [27], which is a collaborative filtering method for implicit feedback datasets. The "Alternative Least Squares"-model was fitted by inputting the binary user-tag vectors of all items in order. For the predictions, the model took the user index, and outputted a list of all user-tags with their scores. The Implicit CF model was defined to take the same input, but to output a list of the top M user-tags along with their score. The model had three tuneable parameters of relevance; the number of latent factors n_l, a regularization factor r, and the confidence level factor c. These hyperparameters were also hyperparameters of the baseline model, since the Implicit CF model was a part of the baseline model.

3.8.2 Naive Bayes

The model handling the autotags was a multinomial naive Bayes classifier made to handle multiclass classification using the one-vs.-rest-strategy. The actual implemetation of the model is based on the scikit-learn [47] Python library using the MultinomialNB and OneVsRestClassifier classes. The model was fitted using the autotags as input and the user-tags as output. For the predictions, the Naive Bayes model takes the autotag vectors as input and outputs the top M list of user-tags along with their prediction probabilities. There is only one parameter to tune in this model, the smoothing parameter for the multinomial naive Bayes classifier, called α. Analogously to the hyperparameters of the Implicit CF model, this smoothing parameter is also a hyperparameter for the baseline model.

3.8.3 Aggregating top predictions

The baseline model aggregates the top M predictions of the Implicit CF model and the Naive Bayes model. The procedure is straightforward, choosing the M tags with the highest prediction scores. However, to be able to tune the relative influence of the two models, the scores from the Naive Bayes model is multiplied with a scalar. This scale factor parameter, denoted by s_f, is thus another hyperparameter of the baseline model, in addition to the hyperparameters that it inherits from the Implicit CF model and the Naive Bayes model.

3.9 Deep learning model

The deep learning model was a hybrid recommender system based on the paper "Neural Collaborative Filtering" by He et al. [23]. The authors' model (NCF),

[13] `https://implicit.readthedocs.io/en/latest/index.html`

publicly available on Github[14], was reworked to handle items and tags instead of users and items. This was a rather straightforward translation, the main differences being the evaluation methodology. Content information, which was not used in the NCF model, was also incorporated into the deep learning model in the form of autotags. The deep learning model was developed using the Python deep learning library Keras[15]. The model can be seen in figure 3.2. As input, the deep learning model took an item, its autotags and a user-tag candidate. The model had two different internal parts, GMF and MLP. These two parts were, similarly to the Implicit MF and the Naive Bayes parts of the baseline model, stand-alone recommender systems themselves, whose outputs were combined in the deep learning model. The GMF and MLP models are further described in section 3.9.1 and 3.9.2. The deep learning model fed its input into the MLP and GMF models. After that, it concatenated their outputs n_m and n_g into one vector and fed it into a dense layer with a sigmoid activation function, yielding the output. This output was a confidence estimate score of whether the user-tag belongs to the item, in the form of a number between 0 and 1. Here, 1 meant most confident and 0 meant least confident. To produce a list of predictions for an item, the confidence estimate was calculated for every user-tag that was not already seen on that item. The list of the top-M tags with the highest confidence values was then chosen.

3.9.1 GMF

The GMF (Generalized Matrix Factorization) model was an artificial neural network model that took an item and a user-tag as input and outputted a confidence value. Generation of a prediction tag list works in the same way as for the deep learning model, described above. The top-M user-tags with the highest confidence values are the predicted tags. The GMF model can be seen in figure 3.3. The model used what in Keras are called embedding layers and flatten layers to transform the high-dimensional information of the item and user-tag number into binary vectors of dimension n_g. Batch normalization and dropout with a dropping fraction of 0.2 were applied to the vectors separately. These vectors were then element-wise multiplied, and the result was fed into a dense 1-unit layer with a sigmoid activation function, yielding a score between 0 and 1. In figure 3.3, the part inside the rectangular box labeled "GMF" was the neural network part that was used in the deep learning model, referred to as the GMF box in figure 3.2. The last part of the GMF model, i.e. the dense layer with a sigmoid activation, was utilized only when the model was used as a stand-alone recommender system. The dimension, n_g, can be seen as the main hyperparameter for the GMF model.

[14] `https://github.com/hexiangnan/neural_collaborative_filtering`
[15] `https://keras.io/`

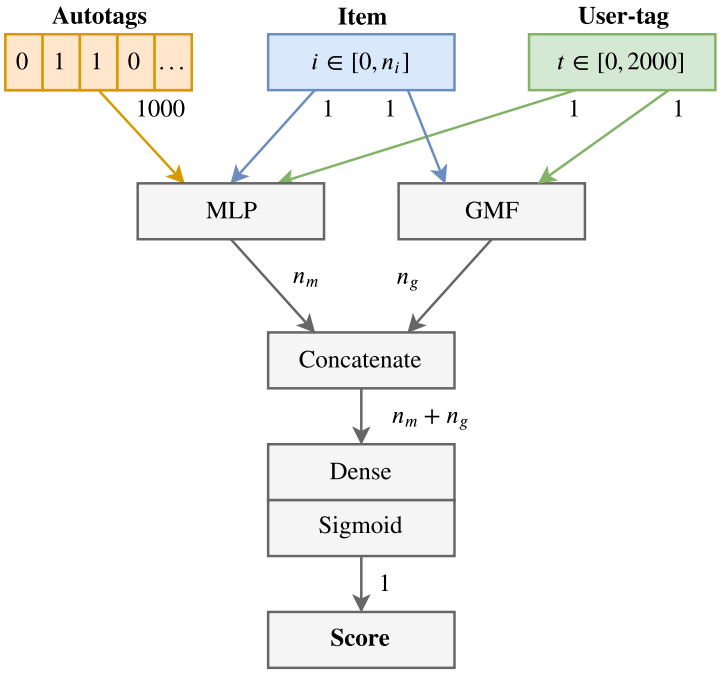


Figure 3.2: The deep learning model. An item, its autotags and a user-tag is fed into the system which outputs a confidence level score between 0 and 1. n_i is the number of items in the dataset. The GMF and MLP boxes are artificial neural network models themselves and can be seen in figure 3.3 and 3.4, respectively. The numbers and symbols next to the arrows represent the dimensions of the vectors in the model. n_g and n_m are the dimensions of the outputs from the GMF and MLP models.

3.9.2 MLP

The Multi-Layered Perceptron (MLP) model was an artificial neural network model that took an item, its autotags and a user-tag as input and outputted a confidence value. Generation of a prediction tag list worked in the same way as for the deep learning model and the GMF model. The model can be seen in figure 3.4. The item and user-tag integer input were transformed in the embedding layers and then flattened, as in the GMF model to a dimension of $n_{m'}$. A dropout of 0.2 was applied to the vectors, which were then concatenated along with the autotag input to form a binary vectors of size $1000 + 2n_{m'}$. These vectors were then fed into m dense layers with ReLU activation functions and a 0.2-dropout layer. The number of nodes in these layers were $n_1, n_2, ..., n_m$. Finally, the vectors were fed into a dense layer with one unit and a sigmoid activation function. Similarly to the GMF model, the part inside the big box labeled "MLP" was the part that was used in the deep learning model. The purpose of the dense layer and sigmoid activation outside of the box was to be able to train the MLP model separately. The main

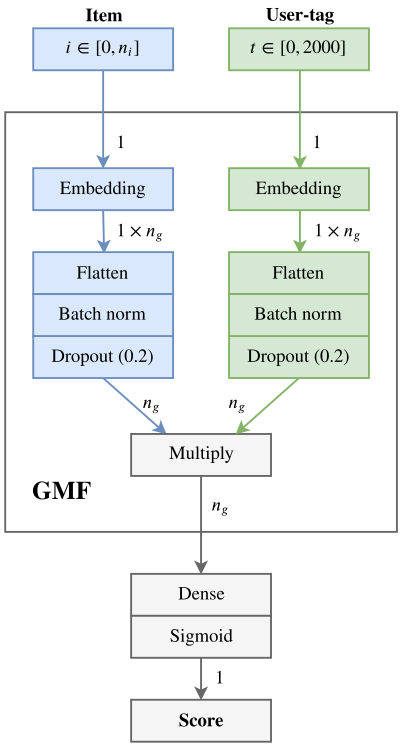


Figure 3.3: The GMF model. n_i is the number of items in the dataset. The numbers and symbols next to the arrows represents the dimensions of the vectors in the model. n_g is the dimension of the output of the embedding layers.

hyperparameters of the MLP model were m (the number of dense layers), $n_{m'}$ and $n_1, n_2, ..., n_m$. During the experimentation with the model architecture, the use of batch normalization was also tested. The layers with batch normalization was placed before the first dropout, right after the flattening layers.

3.9.3 Training

The deep learning model was trained in two different ways. The first way was to train the model completely from scratch. The second way was to first train the MLP and GMF models separately and save the weights. The weights were then loaded into the deep learning model and the training was started from that point. More specifically, the weights in the layers of the GMF and MLP parts of the deep learning model were inserted as they were, directly into the deep learning model at their corresponding positions. For the last layer (the dense layer in figure 3.2), the weights of the last layers of the two smaller models were concatenated in the same way that the respective vectors were concatenated. These weights were multiplied by a scalar each. These two scalars are further described in section 3.12.

The deep learning model was trained by feeding it an item number, the autotags belonging to that item, a user-tag and a label. The label represented the presence of that user-tag on that item, either 1 if it existed, or 0 if not. All positive examples

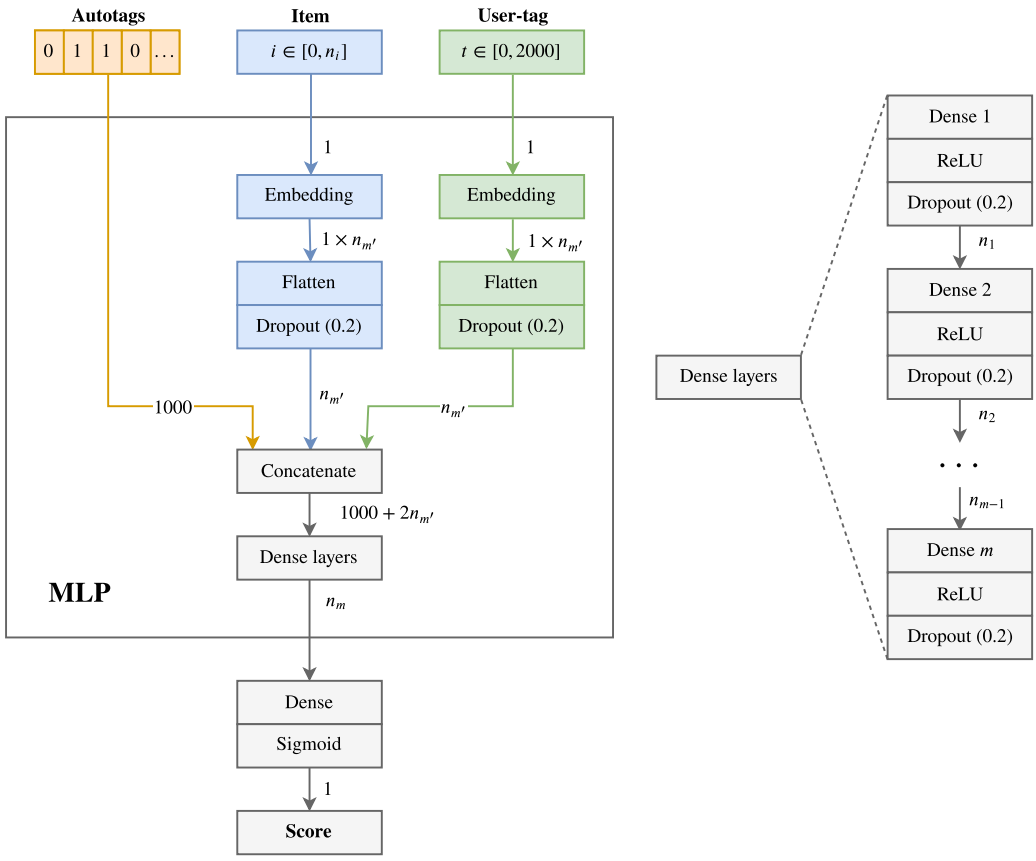


Figure 3.4: The MLP model. n_i is the number of items in the dataset. The numbers and symbols next to the arrows represents the dimensions of the vectors in the model. $n_{m'}$ is the dimension of the output of the embedding layers. $n_1, n_2, ..., n_m$ is the number of units of the corresponding dense layer.

(a user-tag existing on an item) in the dataset were used, but similarly to what was done in [23] by He et al., not all negative examples were used; four negative examples were uniformly sampled for each positive example.

The GMF model and the MLP model were trained in the same way as the deep learning model, with the exception that the GMF model did not take autotags as input. All neural network models were trained using the optimizer Adam, with the objective to minimize binary cross-entropy loss. If nothing else is mentioned in the results section, the initial learning rate of the Adam optimizer was 0.001. The batch size used was 256.

3.10 Hyperparameter optimization

When the implementation had been completed, the optimal sets of hyperparameters were determined for the two models. k-fold cross-validation was performed on the development set (see the end of section 3.6) and the hyperparameters with the highest average performance were chosen. In particular, 10-fold cross-validation

was performed on the baseline model and 5-fold cross-validation was performed on the deep learning model. During the hyperparameter optimization, M, the number of user-tags in the tag suggestion lists, were set to $M = 3$ for both models. The hyperparameter search for the baseline model was mainly done through Bayesian optimization using the Scikit-Optimize Python library[16]. For the baseline model, around a thousand experiments were made. The hyperparameter search for the deep learning model was done using manual tuning, adjusting the hyperparameters manually after obtaining some results. For the deep learning model, over two hundred experiments were made.

For both the baseline model and the deep learning model, the main approach was to first separately optimize the independent parts of the models, i.e. the Implicit MF, Naive Bayes, GMF and MLP models. After this was done, the hyperparameters of the main model were then further optimized using or taking inspiration of the best hyperparameters for the parts. To summarize, the hyperparameters for the baseline model were:

- From the Implicit CF model; the number of latent factors n_l, the amount of regularization r and the confidence level factor c.
- From the Naive Bayes model; the amount of smoothing α.
- The scale factor s_f.

The hyperparameters for the deep learning model were:

- From the GMF model; the embedding size n_g.
- From the MLP model; m, the number of dense layers, the embedding size $n_{m'}$ and $n_1, n_2, ..., n_m$, the number of units of the corresponding dense layer.

The parameter tuning for MLP model was more complex than for the other models, as the "Dense layers"-part could contain different numbers of layers, all with different number of units. For both the MLP model and the GMF model, the hyperparameters were first tuned without any dropout or batch normalization layers. After the hyperparameters were determined, dropout or batch normalization layers were added to the models, or both. In the case of the MLP model, the hyperparameters were tuned a bit further. Hyperparameter optimization was not performed directly on the deep learning model, as it did not have any additional hyperparameters in addition to the ones it inherited from MLP and GMF.

[16] `https://scikit-optimize.github.io/`

3.11 Final testing

When the hyperparameter optimization was finished, the models with the best parameters were tested and evaluated on the testing set. Both the full models and the independent parts were tested. The deep learning model was tested using pretrained weights from the GMF and MLP models as well as from scratch. A range of different values of M for the top-M suggested user-tag lists were used. The models were evaluated on datasets with varying amounts of user-tags per item in the training sets. This is further described in the evaluation approach section, section 3.12.

3.12 Evaluation approach

Evaluation of the performance of the models was done both in the hyperparameter optimization procedure and in the final testing procedure. When evaluating, a part of the dataset was used as validation data or testing data. This part is denoted by the *validation/testing fold*. In the case of the k-fold cross-validation, the validation fold is one of k folds of the development set. In the case of the final testing, the testing fold is the test set. The items in the validation/testing fold, along with their autotags, were included in the training of the model, but only a percentage p of the user-tags per item were kept. The held-out user-tags were the user-tags that were to be predicted by the model in the performance evaluation. See figure 3.5.

In other words, in the case of the k-fold cross validation, the models were trained on all of the items in the development set, trying to predict the held-out user-tags in the validation folds. In the case of the final performance testing, the models were trained on the whole dataset, i.e. the test set combined with the development set, trying to predict the held-out user-tags in the testing fold. The development of the models as well as the hyperparameter optimization was done with $p = 50\%$. In the final testing procedure, the models were evaluated using three different values of the percentage p; $p = 0\%$, $p = 10\%$ and $p = 50\%$. The percentage $p = 0\%$ represents the cold-start scenario, and when doing the partitioning for the $p = 10\%$ dataset, it was ensured that each item had at least one user-tag. The performance metric used in the evaluation was the recall@M of the set of M predicted user-tags. The relevant user-tags were all the user-tags on the item, both the ones kept for training as well as the held-out ones.

In addition to the evaluation procedures described above, the neural network models need some further specification:

- During cross-validation, the models were trained for the same number of epochs (passes of the training dataset through the models). The score from the epoch with the highest score is used as the score representing that fold.

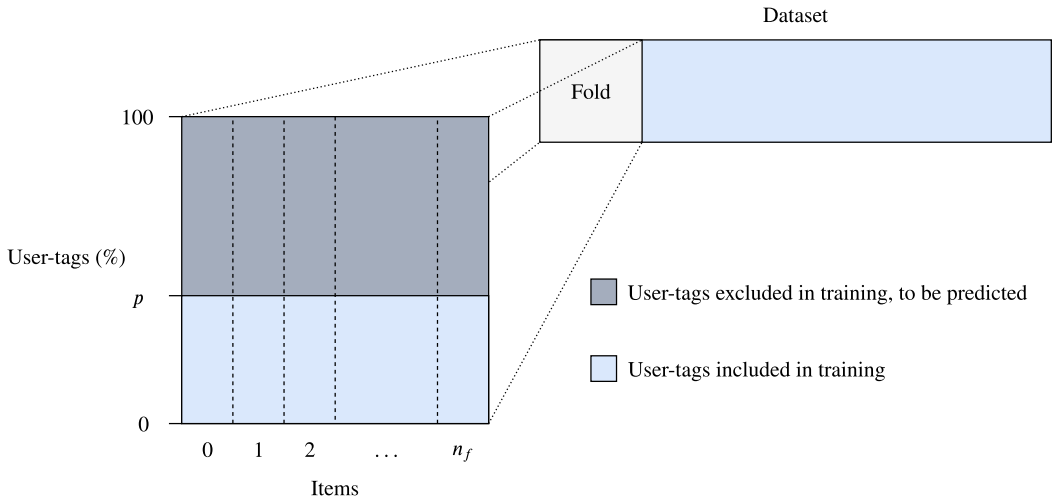


Figure 3.5: The partitioning of user-tags in the validation/testing fold. For each item in the validation/testing fold, the set of user-tags were partitioned into two parts. One part, p percent of the user-tags, were used in the training of the models. The other part, $1-p$ percent of the user-tags, were used as a ground truth value when using the models to predict new tags for a particular item. n_f is the number of items in a validation/testing fold.

During the final testing, in order to know when to stop training the neural network models, user-tags from a small part of the development set was also withheld. The neural network model from the epoch with the highest score on these user-tags was then used to do the final prediction on the testing fold.

- In the deep learning model, in the case of loading pretrained weights from the GMF and MLP models, the weights of the last layer (the dense layer in figure 3.2) is filled with the concatenated weights from the final dense layers of the GMF and MLP models. Before the concatenation, the weights of these layers are multiplied by the fractions corresponding to the percentages p, in the case of the GMF weights, and $p-1$, in the case of the MLP weights.

- As mentioned in section 3.9, to produce a list of recommended user-tags, the neural network models calculates a confidence estimate on all unseen user-tags. This could be quite time-consuming, so another faster evaluation scheme was used during the cross-validation. Instead of making predictions on all unknown user-tags, predictions were made on a randomly sampled subset of the unknown user-tags, making sure that the "true" unknown user-tags were included in this set. Evaluation was then continued exactly as described in this section. This evaluation scheme is denoted by "fast-eval". It should be noted that using fast-eval results in a higher recall score,

since it is more likely that the models recommend the relevant tags if they have fewer tags to choose from. Therefore, the performance of the neural network models evaluated using the fast-eval scheme was never compared to the performance of the models evaluated without it. In the project, fast-eval was set to use 100 user-tags per prediction. Fast-eval was never used during the final testing of the model. If not specified, the results in this report is not obtained using fast-eval.

4 Results

In this project, two tag recommender systems were developed, a baseline model and a deep learning model. The baseline model was a hybrid tag recommender system based on two parts, matrix factorization and probabilistic classification. The two parts were independent tag recommender systems themselves, and are called the Implicit CF model and the naive Bayes model, respectively. The deep learning model was a tag recommender system fully based on neural networks, consisting of two independent neural networks named the GMF model and the MLP model. Together, the deep learning model, the GMF model and the MLP model is denoted by the neural network models. The goal was to determine if the deep learning model could generate better tag recommendations than the baseline model. A dataset of images and photos from Flickr were preprocessed and divided into two parts, a training set and a testing set. The models were developed and optimized on the training set and tested on the testing set.

In this section, the results of the study are presented. The results are divided into two subsections, corresponding to the respective procedure in the methodology section they derive from, which are the hyperparameter optimization and the final testing. In the hyperparameter optimization subsection, the best hyperparameters for the models are presented. In the case of the neural network models, tables containing parts of the experimentation to get to the results are also shown. In the final testing subsection, the evaluation results of the models with the best parameters are displayed. An example tag recommendation query is presented, together with tag suggestions made by the two recommender systems. Then, the final performance of all of the models are compared for different values of the percentage p (as defined in section 3.12). The time it takes to train and make predictions with the models is shown. The final training process of the neural network models is also displayed.

4.1 Hyperparameter optimization

In this section, the results of the hyperparameter optimization of the models and the model parts are listed in the order that they were performed. The main result for each model is the configuration of parameters with the highest k-fold cross-validation performance. For the neural network models, tables with experiments using different sets of hyperparameters are shown.

Implicit CF

The hyperparameter optimization for the Implicit CF model resulted in the follow-

ing parameters: $n_l = 178$, $r = 8.6$ and $c = 89$. This configuration had the 10-fold cross-validation average recall@3 of 0.2191.

Naive Bayes

The hyperparameter optimization for the Naive Bayes model yielded the parameter $\alpha = 3.5$, which had the 10-fold cross-validation average recall@3 of 0.081.

Baseline

Using the parameters resulting from the hyperparameter optimization of the Implicit CF and Naive Bayes models, the hyperparameter search for the final parameter of the baseline model resulted in $s_f = 0.38$, with a 10-fold cross-validation average recall@3 of 0.2193.

GMF

The hyperparameter optimization for the GMF model, *without* the use of any dropout or batch normalization, resulted in $n_g = 94$ with a 5-fold cross-validation fast-eval average recall@3 of 0.3449. Some of the experimentation to reach this result can be seen in table 4.1.

MLP

The hyperparameter optimization resulted in the following hyperparameters for the MLP model: $m = 3$, $n_{m'} = 256$, $n_1 = 750$, $n_2 = 350$ and $n_3 = 96$. This configuration, combined with dropout as depicted in figure 3.4, yielded a 5-fold cross-validation fast-eval average recall@3 of 0.3922. See table 4.2 for the experiments to reach these results. Table 4.2a shows the initial optimization of the model parameters, without any dropout or batch normalization. Here it can be seen that $n_{m'} = 256$ together with $n_m = 96$ is a good configuration. Table 4.2b displays further optimization using dropout and batch normalization, as described in section 3.9.2. It can be noted that more layers gives better performance. Comparing table 4.2a and table 4.2b shows that dropout and batch normalization also improves performance considerably.

Deep learning

The deep learning model did not have any hyperparameters other than the ones in the GMF and MLP models, meaning that no additional optimization was performed.

Table 4.1: GMF dense layer optimization, using fast-eval, without any dropout or batch normalization. The recall@3 is the average recall using 5-fold cross-validation. The variable n_g is the dimension of the output of the embedding layers. The context of n_g can be seen in figure 3.3.

n_g	Recall@3
54	0.3302
64	0.3344
74	0.3384
84	0.3417
94	**0.3449**
104	0.3434

Table 4.2: MLP dense layer optimization, using fast-eval. The recall@3 is the average recall using 5-fold cross-validation. Both the number of units in each layer and the number of layers are optimized. $n_{m'}$ is the dimension of the output of the embedding layers. $n_1, n_2, ..., n_m$ is the number of units in the corresponding dense layers. The context of the variables can be seen in figure 3.4.

(a) Initial optimization, without any dropout or batch normalization.

$1000 + 2n_{m'}$	n_1	n_2	n_3	Recall@3
1128	32	-	-	0.3208
1128	64	-	-	0.3231
1128	96	-	-	0.3171
1128	128	-	-	0.3130
1128	32	16	-	0.2636
1128	64	32	-	0.2646
1128	96	64	-	0.2674
1128	128	128	128	0.2683
1256	32	-	-	0.3283
1256	128	-	-	0.3239
1256	128	64	-	0.2668
1256	256	-	-	0.3213
1512	32	-	-	0.3272
1512	64	-	-	0.3282
1512	**96**	**-**	**-**	**0.3286**
1512	128	-	-	0.3243
1512	256	-	-	0.3253
1512	350	-	-	0.3219

(b) Further optimization using dropout (drop) or dropout and batch normalization (dropbn). LR is the initial learning rate of the Adam optimizer.

Variant	LR	$1000 + 2n_{m'}$	n_1	n_2	n_3	Recall@3
drop	0.002	1512	96	-	-	0.3821
dropbn	0.002	1512	96	-	-	0.3757
drop	0.002	1512	350	96	-	0.3880
drop	0.001	1512	750	350	96	0.3906
drop	**0.002**	**1512**	**750**	**350**	**96**	**0.3922**

4.2 Final testing

In this section, the results from final testing are shown. The final testing was performed on the two models with their best parameters as determined in section 4.1. The main result from the final testing of the two models is graphs of the recall@M for different values of the user-tag per item percentage p. These graphs can be seen in figure 4.1. In figure 4.1a it can be observed that the performances of the deep learning model and the baseline model are nearly identical in the $p = 50\%$ scenario. It can be noted that in this scenario, the Naive Bayes model performed considerably worse than the other models. In figure 4.1b and 4.1c it can be seen that the baseline model performed better in the $p = 10\%$ and $p = 0\%$ scenarios. The models that were mainly based on user-tags, the Implicit CF model and the GMF model, performed the worst in the $p = 0\%$ scenario. The Naive Bayes model (and thus the baseline model) was clearly the best at making predictions when $p = 0\%$.

An example query along with predictions made by the two models can be seen in figure 4.2. In its top-7 list, it can be seen that the baseline model predicted two out of seven of the ground truth tags. The deep learning model predicted three out of seven. The time it takes to train the final models to convergence on the training set of 18,000 items is displayed in table 4.3. The time it takes to make predictions on 2,000 items with these models is also shown. All displayed times were measured when the calculations were made on the desktop computer mentioned in section 3.4. For the neural network models, the training time included evaluating the performance at each epoch in order to know when to stop the training. This took from half to two thirds of the total training time. It can be seen in the table that the time it took for the neural network models to train is more than 1,000 times longer than the time it took for the baseline model. Similarly, it took the neural network model more than 10 times longer to make predictions compared to the baseline model.

The process of training the neural network models can be seen in figure 4.3. The training loss per epoch is displayed in figure 4.3a. The deep learning model achieved a much lower loss than its individual parts, the GMF and MLP models. It can be seen that the trend for the loss function for all models was to decrease over time. In figure 4.3b the validation recall@10 per epoch is displayed. The deep pretrained model had a high performance from the start which decreased slightly, while the other models increased in recall with each epoch. It can be observed that the GMF and MLP models converged at similar speed towards roughly the same values of both loss and recall@10, with the GMF model being slightly faster. Compared to the two sub-models, the "Deep from scratch" model converged to a lower value of recall@10, but at considerably lower loss value.

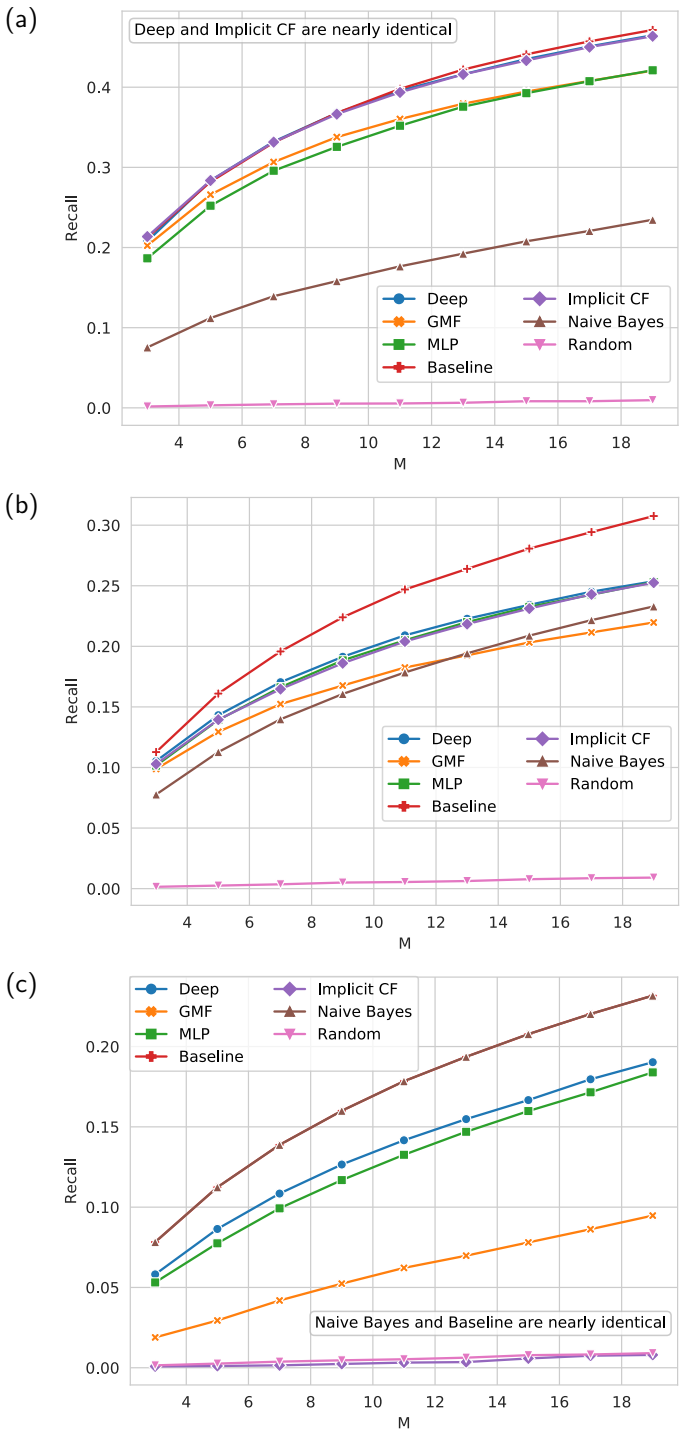


Figure 4.1: Recall@M for different values of the percentage of user-tags kept per item p, plotted for all the different model types. The performance of random tag recommendations is also displayed. (a) shows $p = 50\%$, (b) shows $p = 10\%$ and (c) shows $p = 0\%$. $p = 0\%$ corresponds to the cold start scenario.

User-tags	cars, course, france, pentax, race, sport
Autotags	car, car bumper, car hood, car part, car tire, ferrari, outdoor, sports car, tire, truck, van, vehicle

(a) The user-tags and autotags that are used as input to the models.

(b) The image[18] corresponding to the tags. The image is included for reference and was not used as an input to the models.

Baseline	Deep	Ground truth
paris	**car**	l2
racing	auto	**car**
action	paris	june
run	sports	le
competition	**racing**	**racing**
running	**voiture**	sunday
car	blanc	**voiture**

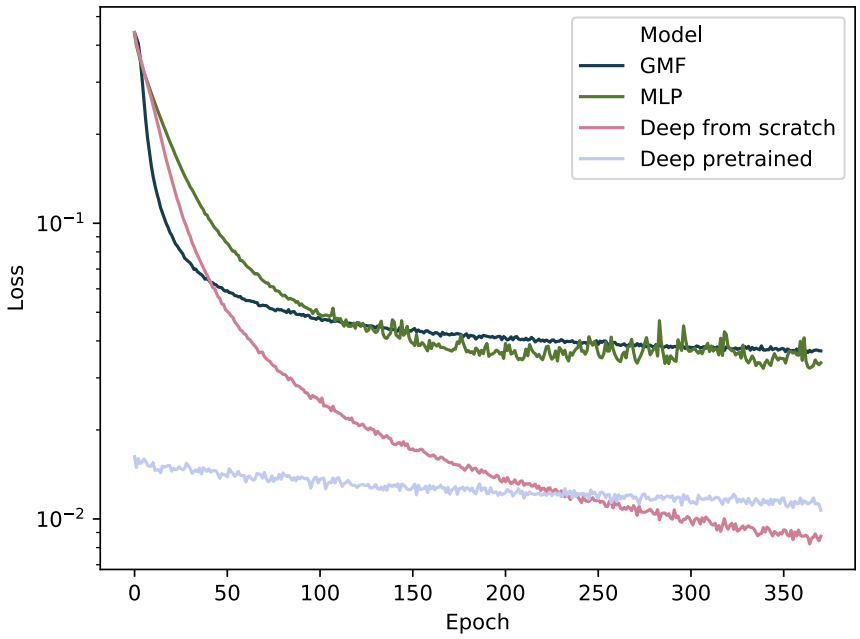
(c) Top 7 user-tag predictions made by the baseline model and the deep learning model, compared to the ground truth user-tags. The user-tags are presented in the way the two models rank them, with the most highly ranked user-tags at the top.

Figure 4.2: An example query (a), the actual picture (b) and the predictions made by the models (c).

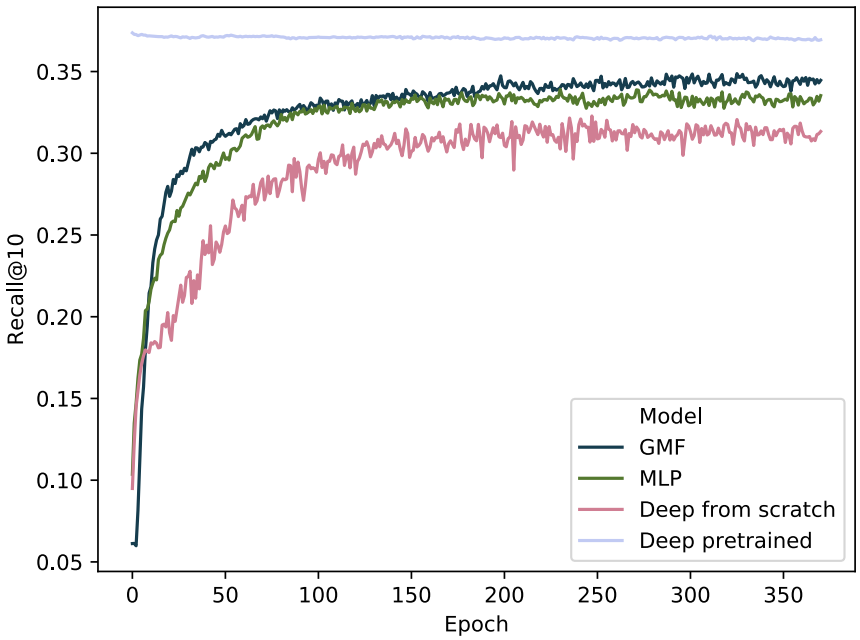
[18] *24h of Le Mans 2011* by Thomas DESFORGES is licensed under CC BY-NC-SA 2.0.
Image link: `https://www.flickr.com/photos/41258005@N08/5970102161`
Author link: `https://www.flickr.com/photos/tomasphotography/`

Table 4.3: The time it took to train and make predictions with the models using the desktop computer defined in section 3.4. The training times are how long it takes to train the models until convergence on the training set (18,000 items). The prediction times are how long time it takes to make predictions on 2,000 items.

Model	Training time (s)	Prediction time (s)
Implicit CF	6.5	0.9
Naive Bayes	5.6	1.4
Baseline	12.2	3.0
GMF	$1.5 \cdot 10^4$	28.4
MLP	$2.6 \cdot 10^4$	40.5
Deep learning	$4.1 \cdot 10^4$	51.9



(a) Training loss per epoch.



(b) Validation recall@10 per epoch.

Figure 4.3: Training loss and validation recall for the GMF model, the MLP model and the deep learning model. The deep learning model is shown being trained both from scratch and using pretrained GMF and MLP parts.

5 Discussion

Two tag recommender systems have been developed. One is a deep learning model built upon two standalone neural networks, the GMF model and the MLP model. The other system is a baseline hybrid model, also containing two parts. The first part is based on a multinomial naive Bayes classifier called the naive Bayes model. The second part is based on implicit collaborative filtering, and is named Implicit CF. The two recommender systems were implemented with the objective to answer the research question of which method has the best performance. In other words, can the recent success of deep learning in numerous fields also be applied to the field of tag-recommendation, and beat the conventional methods? The purpose of the recommender systems was to provide suggestions of tags to photo or video items based on two things: what tags users have added previously, *user-tags*, and item content information in the form of tags automatically generated using computer vision, *autotags*. To test the performance of the two recommender systems, a percentage p of user-tags per item was held out from a set of test items. The goal of the recommendation task was then to feed these items (i.e. the autotags and the remaining user-tags) to the models and predict the withheld user-tags. The amount of held-out user tags per item was varied, with half being the primary scenario ($p = 50\%$). In this scenario, both recommender systems were able to provide recommendations of high relevance, with the performance of the two being very similar. In the case of the scenarios with fewer user-tags per item, the baseline model outperformed the deep learning model.

In this section, the results are discussed in different ways. First, the research question is addressed. After that, the neural network models and their optimization, as well as the baseline model, are treated. A discussion of the results in the sparser scenarios follows, and the problem of cold-start recommendations is mentioned. Then, the concrete predictions made by the models are discussed. Finally, the possibility to make comparisons to other studies is treated.

5.1 Research question

The answer to the research question is not perfectly clear. The two models performed very similarly in the primary scenario, which both models were trained and optimized for. This was seen in figure 4.1a. This was surprising, as the deep learning model is a more complex model with more internal parameters. It takes the autotags and the user-tags into account concurrently, and should thus be able to infer more complicated patterns. It is important to note that in this project, over a thousand configuration experiments were performed using the baseline model, compared to the two hundred experiments that were performed using the deep learning model. This suggests that the baseline model is optimized to a fuller

degree than the deep learning model. To add to this, the deep learning model is in general more configurable than the baseline model. Because of this, it should actually need even more experiments than the baseline model to reach its best performance. Here, *more configurable* refers to that, while the baseline model has five main hyperparameters that can be tuned, the deep learning model has, in addition to the five main hyperparameters mentioned in section 3.10, many more things that can be changed. Some examples would be the amount of dropout layers or the dropout rate, the presence or placement of batch normalization layers, the initial learning rate, the choice of learning rate optimizer or the batch size. In addition to this, the main architecture of the deep learning model could be changed, for example by adding or removing fully-connected layers, adding almost endless possibilities. In the baseline model, one thing that could be changed is the way that the top candidate lists from the naive Bayes model and the Implicit CF model are combined. However, if more variation is desired, these two sub-models would have to be replaced or complemented by some other technique. With these things in mind, the fact that the deep learning model achieved a similar result as the very well-optimized baseline model means that there is potential in using deep learning techniques for tag recommendation. It can be concluded that tag co-occurrence can be successfully modelled using deep learning, both separately as well as jointly together with content information.

5.2 Neural network optimization

On the subject of the optimization of the neural network models, it was seen in table 4.2 that introducing dropout and batch normalization greatly improved performance. This has most likely to do with the overfitting prevention effects of the two techniques. As batch normalization allows the increase of the learning rate, the models are also able to become better in a shorter time. A loss graph and a validation recall@10 graph of the training processes of the neural network models were also shown in the results section in figure 4.3. In the graphs, it can be seen that the GMF model and the MLP model both train really well, reaching high recall@10 at relatively high loss. Combining the two then gives a substantial amount of lesser loss and higher recall@10, as seen for the "Deep pretrained" model plot. It is hard to make any improvements of the recall@10 at this point, as seen in the plots, the loss goes down slowly, but so is the recall@10, indicating slight overfitting. Signs of greater overfitting is seen in the training plots for the "Deep from scratch" model. The loss gets very low, lower than the loss for the deep pretrained model, while the recall@10 is lower than that of all of the other plotted models. To prevent this overfitting, the amount of regularization could be increased, for example by increasing the dropout rate or adding dropout or batch normalization layers to the non-GMF/MLP parts of the deep learning model. The

observation that the pretrained model performed better than the from scratch model was also seen in the Neural Collaborative Filtering paper [23]. The reason for this is probably that it is easier to avoid overfitting when training the parts of the model separately. It should be noted that it would have been better to perform the hyperparameter optimization of the neural network models using 10-fold cross-validation instead of 5-fold, and without the fast-eval scheme. Unfortunately there was not enough time for that since the training and prediction times of the neural network models were very long, as was seen in table 4.3.

5.3 Baseline model

While potential could be seen in the deep learning model, the results of the primary scenario definitely means that there is potential in using the baseline model. The model is much simpler than the deep learning model, and still achieving the same performance. Considering that the time to train the baseline model and make predictions is many times less than that of the deep learning model (table 4.3), the baseline model should be the alternative to go for in practice. Furthermore, as noted in section 4.2, the naive Bayes model has a considerably lower performance than the other models in the primary scenario. This could potentially have negatively affected the performance of the baseline model by a substantial amount. There are several possible explanations to why the performance of the naive Bayes model is worse. It should first be noted that the comparison to the other models might be a bit unfair, since all the other ones have access to the user-tags as input. It could simply be the case that the information in the co-occurrence of the user-tags is strictly superior to what can be extracted from the autotags. Disregarding this possibility for a moment, the explanation that comes to mind is that the naive Bayes model is a very simple model with a strong independence assumption that most certainly is not true. Because of this, the naive Bayes model might not be able to take advantage of the autotags to their fullest potential. An interesting extension to this study would therefore be to try out other types of models in place of the naive Bayes model to see how they and the baseline model would perform. Techniques of interest include LDA as in [65], an autoencoder as in [70, 66] or a deep neural network as in [45].

5.4 Sparse scenarios

Moving on to examining the results of the scenarios $p = 10\%$ and $p = 0\%$, it was seen that the baseline model clearly dominates the deep learning model (figure 4.1b and 4.1c). It is evident that the baseline model is greatly aided by the effectiveness of the naive Bayes model in the cold-start setting, as it is not at all dependent on the amount of user-tags available. Meanwhile, all of the parts and hyperparameters

of the neural network models have been optimized for the $p = 50\%$ scenario, which means that they have a strong dependence on the existence of at least some user-tags. Furthermore, the ability to train the MLP model and the deep learning model in the cold-start scenarios is greatly limited. This has to do with that during the training, all of the user-tags in the training set are used, in order to maximize the amount of information that can be extracted. In other words, the training loss is calculated in a setting where no user-tags are withheld, corresponding to what would have been the percentage $p = 100\%$ in the validation/testing fold. This results in a discrepancy when the number of user-tags per item in the validation set is low, as in the scenarios $p = 0\%$ and $p = 10\%$. In these scenarios, the models are still trained in the setting with an abundance of user-tags, but are validated and tested in a setting with a sparsity of user-tags. This is fine in the case of the GMF model since the user-tags is the only input the model has; the lower amount of data in the validation/test set naturally results in a lower performance. The problems arise when trying to use the MLP or deep learning models for the sparse scenarios. Both these models have an additional input, the autotags. The natural thing to do when training the models for these scenarios would be to adjust the model parameters so that the autotags gets to have more influence over the predictions. However, since the setting of training will always include all user-tags, the increase of autotag influence will never happen. Because of this, the models will perform significantly worse on the scenarios with lower user-tag percentage p, than if they only took the autotags as input. There is not much that can be done to mitigate this and make the training work for the more user-tag sparse scenarios. One thing that could be tried is to feed the models a sparser training dataset. Unfortunately, this would require many more items (a much larger dataset) in order to provide the same number of user-tags in total as the regular training dataset. The best solution to avoid this problem might be to handle the autotags and the user-tags completely separately, as in the baseline model. Unfortunately, this approach misses out on the patterns that can be inferred when the information from the user-tags and the autotags is learnt simultaneously. As mentioned previously, being able to identify these more complicated patterns should be one of the strong points of the deep learning model.

5.5 Cold-start recommendations

The problem of cold-start recommendations, $p = 0\%$, needs to be further addressed. At some point, in practical applications of the recommender systems, items that were not in the original training set will have to be treated. It could be that tag predictions need to be made for these items. These predictions are called out-of-matrix predictions. It could also be that it is desirable that the model learns the information contained within the new items without having to retrain the

models on the full training data again. This is called online learning. Many times, both out-of-matrix predictions and online learning are desired at the same time. It is very simple to do this for the naive Bayes model. As the model depends on just autotags for input, out-of-matrix predictions can be made by feeding the model the autotags of a new item. Online learning is also easily realized, since the scikit-learn library that the model is based on supports partial fit functions, designed for this very purpose. For the Implicit CF model and the neural network models, neither out-of-matrix predictions nor online learning are possible, the whole models have to be retrained when adding one or several items. It is clear that to do this for every new item could be very time-consuming. However if new items are added continuously to a system, a practical approach would be to wait for some items to accumulate and then perform a batch update of the model after some time. The time to wait would depend on how long time it takes to train. Looking at the training times in table 4.3, this is possible to do for the baseline model, with the option to retrain as often as every 12 seconds. However, for a practical purpose, this is unthinkable for the deep learning model, as it takes $4.1 \cdot 10^4$ seconds, a little bit more than 11 hours, to train it.

Another idea to tackle the cold-start problem would be to restructure the way the neural network models take their input. Could the way that data is inputted to the naive Bayes model be used for neural network models? That is, to input all autotags and user-tags of an item to the models at the same time, instead of per item per user-tag as it is now. This would make the models agnostic to the item index, i.e. which item is currently treated, and would allow for instant out-of-matrix predictions. Online learning would still not be possible, but the model could be trained in the background when it has time, in the time span of for example once a day. It would be interesting to make further experiments using a setup like this, the main question being how much prediction performance would be lost, if any. A hypothesis is that some good predictions based on the tag co-occurrence could be lost doing this. The performance of the new approach could be compared to the results in this paper, to see how much of an impact the change would have.

5.6 Concrete predictions

Looking at the example query, figure 4.2, the models correctly predicted two or three out of seven ground-truth user-tags. It should be noted that the number of correctly predicted tags varies greatly depending on the item that is queried, with some items being much harder to treat than others. What is consistent however, which can be seen in the figure, is that the tags that were wrongly predicted by the models are most often relevant and related to the inputted user-tags anyway. In the example, the predicted tags are all related to cars, France and some kind of race, albeit a running race. This matches well with the general topics of the query.

It could also happen that the wrongly predicted tags are completely acceptable substitutes for the correct ones, for example if *car* is predicted instead of *cars*. Sometimes, some of the goal user-tags do not have anything apparent to do with the inputted user-tags and autotags, making them quite hard to predict. In the example query, the ground truth user-tags *june* and *sunday* are examples of this, there is nothing in the input user-tags or autotags that directly justifies the presence of these tags. It should be noted that the tag co-occurrence parts of the baseline model and the deep learning model are exactly what can manage to predict these tags anyway (even though this is not the case in the example). This can happen if there are other items in the dataset which have this non-apparent combination of tags. For example, if races are consistently performed on a sunday, it would be a good thing to predict the tag "sunday" in the presence of information regarding races.

5.7 Comparisons to other studies

It is difficult to make direct comparisons of the results in this study to the results of other papers since the way experiments are set up varies considerably. Problems lie in the differing evaluation methods and metrics, but also in the datasets. Even though recall was chosen as the evaluation metric partly to facilitate making comparisons, it did not make a big difference because of the other aspects. For example, of the studies that treat tag recommendation surveyed in this paper, none was using the YFCC100M dataset. Furthermore, a study performing a similar analysis on a comparable dataset could not be found. For example, Sigurbjörnsson and van Zwol [59] used a Flickr dataset for tag recommendation, but it only contained 332 photos in total. RSDAE by Wang et al. [66] did tag recommendation using a large dataset, but not on photo or video items. This results in that the content information that was used was different than the one used in this study. Nguyen et al. [45] used a public Flickr dataset for tag recommendation that at the time of this project could not be found. Disregarding the direct performance comparisons, it is very common to see that the deep learning models outperform the baseline models in other works. Two examples would be the previously mentioned RSDAE [66], and the method by Nguyen et al. [45]. A result like in this thesis where the baseline model and the deep learning model performed the same, or when the baseline model performed better, could not be found. The reason for this is probably that negative results tend to not be published or get traction in the same way as positive results. This is a shame, since there could be many things to learn from a well-written paper with negative results, for example concerning what approaches not to try.

6 Conclusions

A deep learning tag recommender system was developed and compared to a highly optimized baseline tag recommender system based on Bayesian classification and matrix factorization. The deep learning model attained a similar prediction recall to the baseline model in the main evaluation scenario. It was concluded that the deep learning model has more optimization potential compared to the baseline model. It was also concluded that deep learning architectures other than the one used in this project could be explored and yield good results. However, the deep learning model did not generalize to sparser environments as well to as the baseline model. The computations of the deep learning model was also many times slower than the computations of the baseline model. This led to the conclusion that in practice, the baseline model is better than the deep learning model. The main conclusions are that there is potential in the use of deep learning for the purpose of recommending tags, and that deep learning can be used to successfully model tag co-occurrence both separately and jointly together with content information.

References

[1] C. C. Aggarwal. Content-Based Recommender Systems. In *Recommender Systems*, pages 139–166. Springer International Publishing, Cham, 2016.

[2] S. Alyamkin, M. Ardi, A. Brighton, A. C. Berg, Y. Chen, H.-P. Cheng, B. Chen, Z. Fan, C. Feng, B. Fu, et al. 2018 low-power image recognition challenge. *arXiv preprint arXiv:1810.01732*, 2018.

[3] X. Amatriain, A. Jaimes, N. Oliver, and J. M. Pujol. Data Mining Methods for Recommender Systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 39–71. Springer US, Boston, MA, 2011.

[4] I. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, Dec. 2000.

[5] F. M. Belém, J. M. Almeida, and M. A. Gonçalves. A survey on tag recommendation methods. *Journal of the Association for Information Science and Technology*, 68(4):830–844, 2017.

[6] J. Bennett, S. Lanning, and N. Netflix. The Netflix Prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[7] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[9] D. Bokde, S. Girase, and D. Mukhopadhyay. Matrix Factorization Model in Collaborative Filtering Algorithms: A Survey. *Procedia Computer Science*, 49:136–146, 2015.

[10] D. Bollen, B. P. Knijnenburg, M. C. Willemsen, and M. Graus. Understanding choice overload in recommender systems. In *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*, page 63, Barcelona, Spain, 2010. ACM Press.

[11] R. Burke. Hybrid Web Recommender Systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321, pages 377–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[12] A. J. B. Chaney, B. M. Stewart, and B. E. Engelhardt. How Algorithmic Confounding in Recommendation Systems Increases Homogeneity and Decreases Utility. *Proceedings of the 12th ACM Conference on Recommender Systems - RecSys '18*, pages 224–232, 2018. arXiv: 1710.11214.

[13] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & Deep Learning for Recommender Systems. *arXiv:1606.07792 [cs, stat]*, June 2016. arXiv: 1606.07792.

[14] A. Chernev, U. Böckenholt, and J. Goodman. Choice overload: A conceptual review and meta-analysis. *Journal of Consumer Psychology*, 25(2):333–358, 2015.

[15] S. Flaxman, S. Goel, and J. M. Rao. Filter Bubbles, Echo Chambers, and Online News Consumption. *Public Opinion Quarterly*, 80(S1):298–320, 2016.

[16] Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme. Learning Attribute-to-Feature Mappings for Cold-Start Recommendations. In *2010 IEEE International Conference on Data Mining*, pages 176–185, Dec. 2010.

[17] N. Garg and I. Weber. Personalized, interactive tag recommendation for flickr. In *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08*, page 67, Lausanne, Switzerland, 2008. ACM Press.

[18] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[19] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, Dec. 1992.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] M. Haim, A. Graefe, and H.-B. Brosius. Burst of the Filter Bubble?: Effects of personalization on the diversity of *Google News*. *Digital Journalism*, 6(3):330–343, Mar. 2018.

[22] G. A. Haynes. Testing the boundaries of the choice overload phenomenon: The effect of number of options and time pressure on decision difficulty and satisfaction. *Psychology & Marketing*, 26(3):204–212, 2009.

[23] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural Collaborative Filtering. *arXiv:1708.05031 [cs]*, Aug. 2017. arXiv: 1708.05031.

[24] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval - SIGIR '16*, pages 549–558, Pisa, Italy, 2016. ACM Press.

[25] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 230–237, New York, NY, USA, 1999. ACM. event-place: Berkeley, California, USA.

[26] P. Heymann, D. Ramage, and H. Garcia-Molina. Social tag prediction. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '08*, page 531, Singapore, Singapore, 2008. ACM Press.

[27] Y. Hu, Y. Koren, and C. Volinsky. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Pisa, Italy, Dec. 2008. IEEE.

[28] J. Illig, A. Hotho, and G. Stumme. A comparison of contentbased tag recommendations in folksonomy systems. In *In KPP '07: Postproceedings of the International Conference on Knowledge Processing in Practice, 2009*, 2011.

[29] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, Feb. 2015. arXiv: 1502.03167.

[30] A. K. Jain and a. K. M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, Mar. 1996.

[31] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Dec. 2014. arXiv: 1412.6980.

[32] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[33] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42:30–37, Aug. 2009.

[34] R. Krestel, P. Fankhauser, and W. Nejdl. Latent dirichlet allocation for tag recommendation. In *Proceedings of the third ACM conference on Recommender systems - RecSys '09*, page 61, New York, New York, USA, 2009. ACM Press.

[35] M. Kula. Metadata Embeddings for User and Item Cold-start Recommendations. *arXiv:1507.08439 [cs]*, July 2015. arXiv: 1507.08439.

[36] S. K. Lam, D. Frankowski, and J. Riedl. Do You Trust Your Recommendations? An Exploration of Security and Privacy Issues in Recommender Systems. In G. Müller, editor, *Emerging Trends in Information and Communication Security*, volume 3995, pages 14–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[37] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[38] M. R. Lepper. When choice is demotivating: Can one desire too much of a good thing? *Journal of Personality and Social Psychology*, 79(6):995–1006, 2000.

[39] M. Lipczak, Y. Hu, Y. Kollet, and E. Milios. Tag Sources for Recommendation in Collaborative Tagging Systems. In *Proceedings of the 2009th International Conference on ECML PKDD Discovery Challenge - Volume 497*, ECMLPKDDDC'09, pages 157–172, Aachen, Germany, Germany, 2009. CEUR-WS.org. event-place: Bled, Slovenia.

[40] J. Liu, P. Zhou, Z. Yang, X. Liu, and J. Grundy. FastTagRec: fast tag recommendation for software information sites. *Autom Softw Eng*, 25(4):675–701, Dec. 2018.

[41] P. Lops, M. de Gemmis, G. Semeraro, C. Musto, and F. Narducci. Content-based and collaborative techniques for tag recommendation: an empirical evaluation. *Journal of Intelligent Information Systems*, 40(1):41–61, Feb. 2013.

[42] Y.-T. Lu, S.-I. Yu, T.-C. Chang, and J. Y.-j. Hsu. A Content-Based Method to Enhance Tag Recommendation. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, page 6, 2009.

[43] E. F. Martins, F. M. Belém, J. M. Almeida, and M. A. Gonçalves. On cold start for associative tag recommendation. *Journal of the Association for Information Science and Technology*, 67(1):83–105, 2016.

[44] V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with naive bayes-which naive bayes? In *CEAS*, volume 17, pages 28–69. Mountain View, CA, 2006.

[45] H. T. H. Nguyen, M. Wistuba, J. Grabocka, L. R. Drumond, and L. Schmidt-Thieme. Personalized Deep Learning for Tag Recommendation. In *Advances in Knowledge Discovery and Data Mining*, volume 10234, pages 186–197. Springer International Publishing, Cham, 2017.

[46] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The Adaptive Web: Methods and Strategies of Web Personalization. Volume 4321 of Lecture Notes in Computer Science*, pages 325–341. Springer-Verlag, 2007.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] I. Portugal, P. Alencar, and D. Cowan. The Use of Machine Learning Algorithms in Recommender Systems: A Systematic Review. *arXiv:1511.05263 [cs]*, Nov. 2015. arXiv: 1511.05263.

[49] J. Ramos and others. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142, 2003.

[50] S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 727, Paris, France, 2009. ACM Press.

[51] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining - WSDM '10*, page 81, New York, New York, USA, 2010. ACM Press.

[52] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, Mar. 1997.

[53] F. Ricci, editor. *Recommender systems handbook*. Springer, New York, 2011. OCLC: ocn373479846.

[54] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[55] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

[56] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the tenth international conference on World Wide Web - WWW '01*, pages 285–295, Hong Kong, Hong Kong, 2001. ACM Press.

[57] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '02*, page 253, Tampere, Finland, 2002. ACM Press.

[58] A. M. Shah and G. Wolford. Buying Behavior as a Function of Parametric Variation of Number of Choices. *Psychological Science*, 18(5):369–370, 2007.

[59] B. Sigurbjörnsson and R. van Zwol. Flickr tag recommendation based on collective knowledge. In *Proceeding of the 17th international conference on World Wide Web - WWW '08*, page 327, Beijing, China, 2008. ACM Press.

[60] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.

[61] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[62] M. M. Uddin, M. T. Hassan, and A. Karim. Personalized versus non-personalized tag recommendation: A suitability study on three social networks. In *2011 IEEE 14th International Multitopic Conference*, pages 56–61, Dec. 2011.

[63] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11:38, 2010.

[64] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international*

conference on Knowledge discovery and data mining - KDD '11, page 448, San Diego, California, USA, 2011. ACM Press.

[65] H. Wang, B. Chen, and W.-J. Li. Collaborative topic regression with social regularization for tag recommendation. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[66] H. Wang, X. Shi, and D.-Y. Yeung. Relational Stacked Denoising Autoencoder for Tag Recommendation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 3052–3058. AAAI Press, 2015. event-place: Austin, Texas.

[67] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative Deep Learning for Recommender Systems. *arXiv:1409.2944 [cs, stat]*, Sept. 2014. arXiv: 1409.2944.

[68] Y. Wu, Y. Yao, F. Xu, H. Tong, and J. Lu. Tag2word: Using Tags to Generate Words for Content Based Tag Recommendation. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*, pages 2287–2292, Indianapolis, Indiana, USA, 2016. ACM Press.

[69] X. Xia, D. Lo, X. Wang, and B. Zhou. Tag recommendation in software information sites. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 287–296, May 2013.

[70] H. Yang, Y. Jeong, M. Choi, and J. Lee. MMCF: Multimodal Collaborative Filtering for Automatic Playlist Continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018 on - RecSys Challenge '18*, pages 1–6, Vancouver, BC, Canada, 2018. ACM Press.

[71] H. Zamani, M. Schedl, P. Lamere, and C.-W. Chen. An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation. *arXiv:1810.01520 [cs]*, Oct. 2018. arXiv: 1810.01520.

[72] S. Zhang, L. Yao, A. Sun, and Y. Tay. Deep Learning based Recommender System: A Survey and New Perspectives. *arXiv:1707.07435 [cs]*, July 2017. arXiv: 1707.07435.

TRITA -EECS-EX-2019:845