Summary of Master Thesis

Our Master Thesis belongs to the specialization branch Machine Intelligence. It focuses on the research area of Recommender Systems which is the concept of recommending a user *u* interesting objects/items *i* from a list of unseen items. More specifically we focus on Collaborative Filtering which bases its recommendations on the previous interactions between users and items e.g. a matrix of |users| x |items| encapsulating feedback such as “thumbs up”/”thumbs down” or numerical ratings/stars.

Matrix factorization is a popular algorithm for making recommendations as part of collaborative filtering. The concept behind is that a rating matrix is sparse as every user has not rated every item. The sparse matrix can be decomposed into two submatrices known as latent factor matrices which when multiplied together attempts to reconstruct the values in the original sparse matrix. A side effect of this reconstruction is that the previously unknown values are now filled out based on the latent factors. The filled out values serve as predictions for a user’s interactions and are used as the basis of recommendations.

Our Master thesis is an extension from our own work during the 9th semester which showed that the recommendation performance of Matrix factorization can be improved through the introduction of additional data. More specifically we extended the concept of CoFactorization introduced in 2016 where Matrix Factorization is augmented by the use of co-occurrence matrices, containing Shifted Positive Pointwise Mutual Information (SPPMI) values.
The original paper on CoFactorization from 2016 was extended in 2018 and we based our 9th semester work on the remaining possible extensions outlined in their future work section. Unfortunately the last extension proved to decrease performance – a finding that we posit is the result of a “cap” being reached. The latent factor matrices are not able to represent both the original matrix and all the additional SPPMI extensions without suffering in general performance. This discovery fueled the pivot of focus for our 10th semester.

As a direct result of the above finding we have focused our 10th semester on another avenue of increasing performance: the concept of ensembling. Ensembling is based on combining multiple individual recommender systems to aggregate their recommendations and thus improve recommendation performance. Ensembling was proven as a valid strategy for recommendation by the Netflix Prize in 2009, where the winner was an ensemble of 108 constituent methods. The simplest type of ensemble takes the “confidence” values e.g. the values in a dense matrix output by matrix factorization, of each constituent recommender system and averages them across all constituent recommenders.
It has since been attempted to make this combination of recommender systems more intelligent than pure averaging – a simple extension is the introduction of weights for each item-user combination for each recommender. Assigning a unique and fitting weight to every user-item combination in each single recommender that is being ensembled is an enormous task – for that reason we attempt to assign a fitted weight to each individual user on each recommender instead. In order to make our proposed method more general we constrain our use of additional data to be the meta data that can be extracted solely from the rating matrix – primarily rating counts and co-occurrences. That means no use of timestamps or demographic data.

Due to our positive experiences with SPPMI matrices for encapsulating co-occurrences last semester we have implemented two simple recommender systems based on K Nearest Neighbours, which use SPPMI as the similarity value instead. The recommender based on user-user co-occurrence performs well, while the recommender based on item-item co-occurrence performs poorly. For that reason we use the user-based recommender, the KNN recommender and a matrix factorization recommender as the three constituent recommenders we will try to ensemble.

Our initial proposed method to assign unique weights to each user is based on a neural network. The network takes a user's meta data as input and maps it into a set of weights – one weight for each constituent recommender. This process is a lot faster to train than a unique weight for each user-item combination in each constituent recommender.
Unfortunately the neural network approach did not yield any useful results. We spent a long time attempting to tune the network using alternative architectures and loss functions both pointwise and listwise. Eventually we discarded the idea of using a neural network and settled on the best performing architecture and loss function to report in our paper.

As an alternative to the neural network approach we propose a clustering approach. The concept behind is that we use the meta data of each user as a basis for clustering. We then assign a fitted set of weights to each cluster and use this weight for every user in the cluster.
Where the network could fit the weights by mapping the meta data, we utilize a grid search for the fitting of each cluster's weights. We search over every possible combination of weights that fulfill the requirements that all weights sum up to 1 and the step size is 0.05. This approach is fully parallelizable and for that reason we employ threading as part of our weight fitting.

The clustering approach performs better than the baseline predictors it combines, but fails to beat the naïve combination on all but one dataset. For this one dataset it would seem that the naïve combination is impeded by one of the constituent recommenders performing extraordinarily poor.
Despite not finding any approaches with groundbreaking performance we have spent our semester researching current trends in recommender systems and experimented a lot with implementations of our ideas.

Meta Ensembling for Recommender Systems

Frederik Kirk Kristensen
fkrist14@student.aau.dk
Aalborg University
Aalborg, Denmark

Mathias Rosgaard Klemmensen
mklemm14@student.aau.dk
Aalborg University
Aalborg, Denmark

ABSTRACT

In this paper we extend upon ensembling of recommender systems by customizing the weights used in linear stacking. We propose the use of meta data readily available in rating matrices as the base of customization. We introduce two novel approaches, one based on neural networks, and one based on clustering of users, to map meta data into a set of weights. We furthermore introduce two KNN based predictors to use when ensembling. We extensively test our proposed approaches against multiple baselines and beat singular recommenders with our ensembles but fail to beat the baseline ensemble.

KEYWORDS

recommender systems, ensembling, meta data

1 INTRODUCTION

Recommender Systems (RSs) are used to model users of a system, in order to recommend items that are of interest to a given user [12, p. 1]. Collaborative Filtering (CF) is a method which generates recommendations based solely on patterns of interaction, e.g. usage or rating, between users and items, without the need for any external data [12, p. 77]. This interaction between users and items can be captured in a $|\mathcal{U}| \times |\mathcal{I}|$ matrix R. As a user rarely interacts with all available items in a system, matrix R is sparse.

In this paper we propose novel approaches to improve upon RSs. Recent research has shown that individual recommenders such as the well known Matrix Factorization (MF) approach can be improved by the inclusion of additional data for use during factorization[10, 14]. Our earlier work has shown that the addition of further data does not necessarily guarantee improved recommendations [6]. For that reason we use the concept of ensembling, that is combining the output of multiple recommenders, rather than the improvement of a single recommender.

The Netflix prize showed that ensembling of multiple methods can yield greater performance in recommendation than singular methods [7]. The Netflix prize used a method that assigns a custom ensemble to each user-item pair. While this leads to better performance it comes at the price of a computational overhead. We propose using additional data (meta data) to perform ensembling on a per user basis instead of user-item pairs to reduce the computation. We introduce and test a neural network that can learn a mapping between meta data describing an individual user, and a set of weights for use when performing a linear combination of constituent recommenders. This is closely related to the concept of meta learning [15].
We also suggest a simpler alternative by using clustering algorithms to segregate the users of a recommender system based on their meta data. Each user cluster can then have a customized ensemble assigned to it.

Finally, as part of our ensembling experiments we introduce two new baseline recommenders: User Positive Co-occurrence (UPC) and Item Positive Co-occurence (IPC) which are both based on KNN, but using Shifted Positive Pointwise Mutual Information (SPPMI) as their similarity score.

To summarize: in this paper we make the following contributions:

- We present Neural Network Meta Stacking (NNMS), a neural network model that maps user meta data to differentiate ensemble weights on a per user basis.
- We present a model, Meta Clustered Stacking (MCS), that uses clustering to segregate users into clusters based on user meta data in order to differentiate ensemble weights on a per cluster basis.
- We introduce two new KNN based models: User Positive Co-occurrence (UPC) and Item Positive Co-occurrence (IPC).
- We perform an experimental evaluation to show the effectiveness of our proposed methods.

Section 2 will summarize the notation and terminology used in this paper. Section 3 will describe both the baseline methods we ensemble, and our approaches for ensembling. Section 4 will detail our experimental evaluation of our proposed ensembling approaches. Section 5 will conclude upon our findings while Section 6 will propose extensions to our work.

2 PRELIMINARIES

The notation used in this paper is summarized in Table 1. The terminology used is presented in the following text.
User. The entity that has interacted with items in the rating matrix. A user is represented as a single integer number.
Item. The entity that is being recommended and rated by users. An item is represented as a single integer number.
Rating Matrix. The $|\mathcal{U}| \times |\mathcal{I}|$ matrix R containing the interactions

Table 1: Notation used in this paper.

Notation	Description
R	Rating matrix of size $\|\mathcal{U}\| \times \|\mathcal{I}\|$
$\mathcal{U}$	Set of all users. Individual users are denoted as u
$\mathcal{I}$	Set of all items. Individual items are denoted as i
$\mathcal{P}$	Set of all recommenders
$\mathcal{J}$	An ordered list of items served as recommendations
M	Number of items to consider from the front of $\mathcal{J}$
α	Latent user factors. Rows are indicated with α_u
β	Latent item factors. Rows are indicated with β_i
λ	Regularization parameter

between users and items.
Confidence. The value assigned to a user-item combination by a recommmender. The user-item combinations with highest confidence values are used as recommendations.
Meta Data. The side information available. Some datasets contain additional data describing users or items in the rating matrix e.g. demographic data, or item descriptions. In this paper we constrain meta data to be data that can be extracted solely from the rating matrix. Examples of such data are a user's total number of rated items, the average popularity of a user's rated items, or a user's average co-occurrence value with other users.

3 RECOMMENDATION MODELS

First we present the concept of SPPMI matrices, then we give a description of the three base recommenders used throughout this paper: K-Nearest Neighbours, User-Positive Co-occurrence, and Matrix Factorization. Lastly we present the different types of ensemble methods used in this paper: Linear Stacking, List Position Stacking, Neural Network Meta Stacking, and Meta Clustered Stacking.

3.1 SPPMI Matrices

Co-occurrence is a concept that is often used in CF. Co-occurrence can be measured using simple counts as was done by Kamehkhosh et al. [5] for their association rules mining. Alternatively Liang et al. [10] extended upon the work of Levy and Goldberg [9] and used Shifted Positive Pointwise Mutual Information (SPPMI) for measuring co-occurrence. SPPMI is calculated from ordinary Pointwise Mutual Information (PMI) which is defined in Equation 1 [9]. PMI calculates the co-occurrence between item i and item j using probabilities P for the items appearing together in a user's ratings and individually.

$$PMI(i,j) = \frac{P(i,j)}{P(i)P(j)} \tag{1}$$

To turn PMI values into SPPMI values, the PMI values are shifted by $log(k)$ and all negative values are removed as seen in Equation 2 [9]. In this paper we use $k = 1$ which is similar to the work of Tran et al. [14].

$$SPPMI(i,j) = Max(PMI(i,j) - log(k), 0) \tag{2}$$

Liang et al. [10] stored the SPPMI co-occurrence between items in a matrix and used this matrix to regularize a weighted matrix factorization based RS. Tran et al. [14] extended upon this idea and used multiple SPPMI matrices for co-occurrence between both items and users – discerning between positive co-occurrence and negative co-occurrence.
An important aspect of SPPMI matrices is that they can be obtained from the rating matrix R by counting occurrences to estimate probabilities, and as such are available for use in any CF dataset.

3.2 Base Predictors

The base predictors are singular recommenders, which in this paper will be used both as a baseline and as a constituent of our ensembles.

3.2.1 K-Nearest Neighbours. (KNN) is a simple predictor based on similarity between either users or items, that has been used for more than a decade [11]. For this paper we focus on item-based KNN. When using item-based KNN, a user u is modelled as a list of liked items. For each liked item i in this list, the k items most similar to i are found along with their respective similarity values. The similarity values are summed in case the same item is found multiple times. The summed similarity values found for items of user u are normalized for future ensembling purposes using the formula in Equation 3. x is the similarity value of a single item while u_{max} is the highest similarity value found between u's items. The normalization does not change the ordering of items with a score above 1, but makes ensembling of multiple predictors straightforward. Recommendations can be made based on the items with the highest similarity values to the user.

$$Normalize(x, u_{max}) = \begin{cases} 0.0 & \text{if } x \leq 1.0 \\ \frac{Log_2(x)}{Log_2(u_{max})} & \text{if } x > 1.0 \end{cases} \tag{3}$$

Similarity is frequently measured using the cosine similarity metric.

3.2.2 User Positive Co-occurrence. (UPC) is similar to KNN, but uses SPPMI values as the similarity measure and measures similarities between users rather than items. The recommendations for user u are built by finding all users similar to u according to the SPPMI matrix. Each item liked by a related user is assigned the related user's SPPMI value as its similarity score. Once again, the similarity values are summed in case the item has been found multiple times, and the normalization approach of Equation 3 is used. Recommendations are made by finding the items with the highest similarity sums.

3.2.3 Item Positive Co-occurrence. (IPC) is another predictor based on KNN and SPPMI values as similarity measure. It measures the similarities between items on a per user basis. For an item i liked by a user u all items have their similarity score increased by the SPPMI between i and the item. Items that are assigned multiple similarity scores have their scores summed. IPC reuses the normalization approach seen in Equation 3 and recommendations are made by selecting the items with the highest similarity scores.

3.2.4 Matrix Factorization. (MF) is a popular predictor based on the concept of latent factor models which won popularity during the Netflix prize [8]. The premise is that the rating matrix R is sparse, but can be factorized into two latent factor matrices α and β, which when multiplied back together results in a dense rating matrix with approximations of the previously unknown ratings. The unknown rating $r_{u,i}$ which user u gives item i is approximated as seen in Equation 4.

$$\hat{r}_{u,i} = \alpha_u^T \beta_i \tag{4}$$

The factorization into latent factor matrices α and β is usually achieved by minimizing the regularized squared error shown in Equation 5.

$$\mathcal{L}_{MF} = \frac{1}{2} \sum_{u,i|r_{u,i} \neq 0} (R_{u,i} - \alpha_u^T \beta_i)^2 + \frac{1}{2}(\lambda_\alpha \sum_u \|\alpha_u\|^2 + \lambda_\beta \sum_i \|\beta_i\|^2) \tag{5}$$

The minimization is performed using either stochastic gradient descent or alternating least squares [8]. In this paper we use the implementation of Tran et al. [14].

3.3 Ensembling Methods

Ensembling methods take multiple baseline predictors and aggregate them to obtain a singular result.

3.3.1 Linear Stacking. (LS) as described by Breiman [1] is the baseline of our ensemble methods. It is given by Equation 6.

$$LS(u) = \sum_{p=1}^{\mathcal{P}} (w_p \cdot g_p(u)) \tag{6}$$

$LS(u)$ is the final prediction function, w_p is a weight assigned to recommender p, and g_p is the prediction function for recommender p. w can either contain $|\mathcal{P}|$ optimized weights, or can be set to 1 to weight each recommender equally. We refer to the latter as Simple Linear Stacking (SLS) and use this as a baseline for ensembling.

3.3.2 List Position Stacking. (LPS) is a novel approach of this paper. Instead of taking a confidence vector for all items like LS, it uses the user's top-M items from each recommender. This M can be adjusted as a hyper parameter, but for this paper we use $M = 100$. Then these results are aggregated as shown in Equation 7.

$$LPS(u) = \sum_{i=1}^{\mathcal{J}} \sum_{p=1}^{\mathcal{P}} (w_p \cdot \frac{1}{log_2(Pos(i)+1)}) \tag{7}$$

We still use the weighting of each recommender as done in LS, but instead of using the confidence for each item, we instead derive this from the position (Pos) of item i in $\mathcal{J}$ given p. Here log_2 is used to make the decline in confidence smoother when increasing the position of i.
LPS ensures that all models used in the ensemble have comparable confidence values.

3.3.3 Neural Network Meta Stacking. (NNMS) is a novel approach introduced in this paper. For both LS and LPS the weights are static for all users. NNMS still use the concept of these two ensemble methods, but instead of static weights we attempt to have a neural network assign a custom set of weights to each user depending on their meta features.
Figure 1 shows the structure of this model. On the figure we see the first layer of the model, labeled as "Meta Data". This is an input layer taking a meta vector for a given user. This is then fed through a number of hidden layers before outputting a set of weights, equal to the number of recommenders in the ensemble. These weights are then used with either LS or LPS on the output of the recommenders to give the final prediction for a given user.

3.3.4 Meta Clustered Stacking. (MCS) is another novel approach introduced in this paper. This method takes a similar approach to NNMS in that it tries to optimize the weights for stacking. Figure 2 shows the structure of this model. The first part of the model is finding K number of clusters (buckets) for the users based on their meta data. For this paper we will use K-means clustering to divide the users into clusters with the number of clusters K given as a hyperparameter. Then we find the optimal weights for each cluster during training. To do this we perform a grid search and use the set of weights with the best results in NDCG@20 on the validation dataset. Once these weights are found predictions are done by taking a user's meta vector, finding the cluster for the given user, and using stacking with the cluster's set of custom weights.

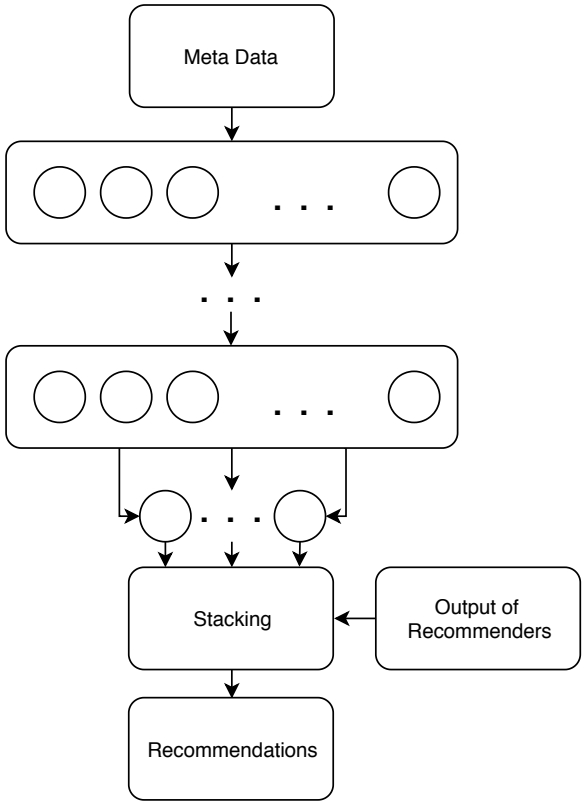

Figure 1: Neural Network Meta Stacking structure

4 EXPERIMENTAL EVALUATION

We experimentally evaluate the proposed ensembling methods as well as the described baselines. We base our choices of evaluation metrics and dataset preprocessing procedure on the previous work of Liang et al. [10] and Tran et al. [14], in order to compare our evaluation with the results they obtained.

4.1 Datasets

To test our proposed models and baselines we use three datasets for recommendation research:

- MovieLens-10M (ml10m) [3]: A timestamped dataset of rating interactions between users and movies.
- MovieLens-20M (ml20m): A larger version of ml10m.
- Goodbooks-10k (gb10k) [16]: A dataset of rating interactions between users and books. Does not contain timestamps.

We follow the same procedure for preprocessing as Liang et al. [10] and Tran et al. [14]. This procedure can be summarized as binarizing the data, only keeping ratings ≥ 4. An iterative process removes user or items with < 5 rating interactions in the kept data. Finally, the ratings are split into three subsets: *train*, *validation*, and *test*.

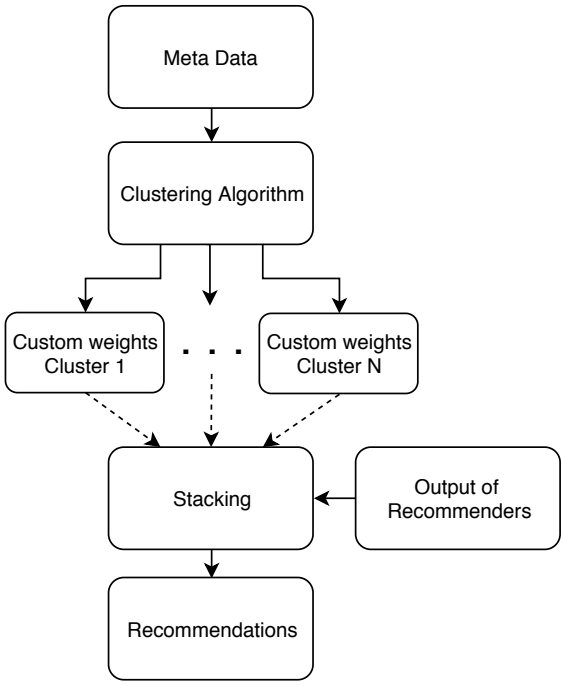

Figure 2: Meta Clustered Stacking structure

Table 2: Statistics after preprocessing for each dataset used.

	ml10m	ml20m	gb10k
# users	58057	111146	53366
# items	7223	9888	9999
# ratings	4.1M	8.2M	4.1M
% sparsity	99.03 %	99.25 %	99.23 %

The splitting is performed following a 70/10/20 split, guaranteeing at least 1 rating for a given user or item in each subset. When timestamps are available, the test subset consists of the most recent ratings. Table 2 summarizes the statistics of each dataset after preprocessing.

4.1.1 Meta data. For all 3 datasets we extract a meta vector for each user based on the training set. We extract 3 features for each user:

- Number of liked items
- Average SPPMI values for user similarity
- Average SPPMI values between liked items

4.2 Metrics

All metrics used for evaluation in this paper are calculated given an ordered list of recommendations, $\mathcal{J}$, and a value M. Only the first M items in $\mathcal{J}$ are considered as part of evaluation.
The first metric used for evaluation is *Recall* which is defined in Equation 8 [10].

$$Recall@M(u, \mathcal{J}) = \frac{\sum_{i=1}^{M} Liked(u,\ i)}{Min(M,\ Number\ of\ liked\ items\ of\ u)} \quad (8)$$

The *Liked* function used in the numerator returns 1 when user u has liked item i in the test dataset and 0 otherwise. The denominator evaluates to the minimum between M and the number of items u has liked in the test set. The result is a metric indicating how many items in the top of $\mathcal{J}$ are relevant recommendations for u.

The second metric is Normalised Discounted Cumulative Gain (NDCG) which is the normalised version of DCG. DCG is defined in Equation 9 [10].

$$DCG@M(u, \mathcal{J}) = \sum_{i=1}^{M} \frac{Gain(u,\ i)}{log(Pos(i) + 1)} \quad (9)$$

The *Gain* function used in the calculation of DCG measures how much user u gains from being recommended item i. This allows for different scores of items in the test set. In this paper we use the *Liked* function in place of *Gain*. The *Gain* in the numerator is being discounted by the expression in the denominator which reduces the gain as the item moves further away from the top of $\mathcal{J}$.
The normalised version of DCG is obtained by dividing with a perfect ordering. When *Liked* is used as the *Gain* function, a perfect ordering has all items liked by user u in the test set at the front of $\mathcal{J}$. The resulting NDCG metric indicates how useful the recommendations in $\mathcal{J}$ are, while penalizing useful recommendations placed further down in the list.

The third evaluation metric is Mean Average Precision (MAP). Equation 10 defines AP which is calculated for an individual user.

$$AP@M(u, \mathcal{J}) = \frac{\sum_{i=1}^{M} Precision@i(u, \mathcal{J})}{Min(M,\ Number\ of\ liked\ items\ of\ u)} \quad (10)$$

The *Precision* function used can also be formulated as Hit Rate – how many items were liked in the list. An important aspect of Equation 10 is that the *Precision* function takes a suffix indicating how many items of $\mathcal{J}$ to consider. This means that the Precision is calculated for a list of length 1, 2, ..., M and averaged. The mean of all users' APs can be calculated and this is the MAP metric. MAP is similar to NDCG as it also penalizes the ordering of items in $\mathcal{J}$.

In keeping with the work of Liang et al. [10] and Tran et al. [14] we will use {5, 20, 10} as the values of M for each metric respectively. Similar to MAP being the mean across all users, we will report the mean across all users for Recall and NDCG too.

4.3 Experimental Approach

The experimental approach can be separated into two phases. Training of baseline models and training of ensemble methods. The training phases are conducted separately across datasets. When hyperparameters are part of a model we mention how we obtain their values.

4.3.1 KNN. The training of KNN is performed solely using the training set. The k neighbours to consider can be seen as a hyperparameter, but as we have sufficient memory, and want the best performance of KNN, we set k to consider all neighbours.

4.3.2 UPC. This has the same hyperparameters as KNN, and uses the same configuration.

4.3.3 IPC. This has the same hyperparameters as KNN and UPC, and uses the same configuration. Preliminary testing shows that the performance of IPC is not competitive with other baseline predictors and for that reason we do not use it as part of our ensembles.

4.3.4 WMF. The training of WMF requires the use of both the training set and the validation set. The ratings in the training set are being used when optimizing the factorization. After each optimization step the NDCG@10 is calculated on the validation set. The optimization continues until NDCG@10 does not improve. The number of latent factors to use k is a hyperparameter. The regularization weight λ is a hyperparameter too. For ml20m and ml10m we use the same parameters used by Tran et al. [14], and for gb10k we find them using a grid search, searching between $k = \{30, 40, 50, 60, 70, 80, 90, 100\}$ and $\lambda = \{0.01, 0.1, 1, 10\}$.

4.3.5 NNMS. We use the baseline predictors KNN, UPC, and WMF to map the meta data for a given user into a set of three weights. The neural network architecture used to produce our results is built using Keras and consists of 3 hidden dense layers with 128, 64 and 32 units respectively. All three hidden dense layers use rectified linear unit as their activation function. The layer producing the final three weights is a dense layer using softmax as activation to ensure the weights sum up to 1. The three weights are eventually used for stacking using LS.
The training of the network uses the validation set and a pointwise loss function defined by He et al. [4].
We choose to omit LPS as the complexity of the method makes it non-differentiable within the Keras framework. Additionally a pointwise loss function is not possible for LPS, as a full list of predictions is necessary to make any prediction.
As the results were unimpressive we experimented with alternative architectures and different loss functions including the listwise loss function from ListNet defined by Cao et al. [2]. With different architectures and loss functions tested, all with poor results, we ended our tests of the method and chose the above architecture. However we do not believe that this is the optimal structure.

4.3.6 MCS. We use the same three baseline predictors as NNMS. The amount of buckets to use is determined through a grid search over {1, 5, 10, 20, 40, 50, 60, 80, 100}. Each bucket of users finds its optimal set of weights by using the set of weights with the highest performance on the validation set for NDCG@20. Weights are considered between 0 and 1 with all weights in a set always summing up to 1. Weights are changed in increments of 0.05 which gives a total of 231 weight combinations to consider when three baseline predictors are used.

4.3.7 Regularized Matrix Embedding. (RME) is a recommender based on WMF regularized by the SPPMI values for positive user co-occurrence, and positive/negative item co-occurrence [14]. We include RME as an additional baseline to showcase the performance of a complex singular recommender. For ml10m and ml20m we will use the settings Tran et al. [14] present in their paper, while we will do a grid search for gb10k, using the same grid search parameters as for WMF.

4.3.8 Extended RME. (ERME) is our extension upon RME to include negative item co-occurrence [6]. We will again use the same settings found for ml10m and ml20m while doing a grid search for gb10k reusing the grid search parameters of WMF.

4.4 Research Questions

When all recommenders are trained we will evaluate them using the metrics defined in Section 4.2. The results will then be used to answer the following research questions:

(1) How does the baselines and the ensemble methods perform on our chosen datasets?
(2) Does UPC improve performance when included in ensembles?
(3) Can the concept of customized weights improve upon simple linear stacking?
(4) Is linear stacking or list position better for ensembling?
(5) What impact does the amount of buckets have when using MCS?
(6) Is per user basis (NNMS) or per cluster basis (MCS) better for customizing weights?

4.5 Results

We now present the results of our experimentation and use the obtained results to answer our research questions.

4.5.1 RQ1: Baseline and Ensemble performance. The results obtained through the experiments described in Section 4 can be seen in Tables 4, 5 and 6 for the ml10m, ml20m, and gb10k datasets respectively. In the case of MCS we only highlight the results of the best performing bucket counts and indicate the count value as a suffix using @ e.g. MCS@5 for MCS with 5 buckets.
Looking at the ml10m and ml20m datasets it can be seen that the three baselines used for ensembling are similar in performance, with KNN having the highest score across all metrics. It can also be seen that a more complex single recommender such as RME and ERME can improve performance further. When ensembling is utilized the best performing model in both datasets is SLS, beating the singular recommenders (although barely losing to RME by 0.0002 on NDCG@20 in the ml10m dataset).
Looking at the goodbooks dataset we see that the three constituent recommenders of our ensemble are differing by a larger margin in performance. KNN is still the best of the three recommenders. Due to the larger margin in performance we see that SLS performs worse than KNN, but introducing customized weights shows that ensembles are still able to improve performance. However the best performing model on this dataset is the more complex singular model RME.
Overall we see that ensembles achieve better performance than its constituent recommenders across all three datasets. For ml10m and ml20m we are able to outperform all singular methods with a SLS approach, but for goodbooks10k RME is the best performing model. We believe that this is due to the weak performance of WMF and UPC on this dataset.

4.5.2 RQ2: UPC ensemble performance. We introduced UPC to include diverse singular recommenders in our ensembles. In order

to verify that the inclusion of UPC is beneficial we have tried ensembling with SLS and MCS@1 (LS) all pairwise combinations of KNN, UPC, and WMF. We present the results of this ensembling on ml10m and ml20m in Table 3. We choose to omit goodbooks10k as the performance of UPC is very weak on this dataset and therefore using UPC in the ensemble will not prove useful. The results show that for both datasets and for both ensembling methods the inclusion of UPC is useful, as the performance is best when all three models are included.

4.5.3 RQ3: Customized weight performance. The customization of weights had a different impact across the three datasets. Looking at ml10m in Table 4 the customized weights model that performed best was MCS@5 (LS), which matched the simple linear stacking on Recall@5, but performed worse on NDCG and MAP indicating a worse ordering of items. A similar pattern can be seen for ml20m in Table 5 where the models with customized weights perform worse across all metrics.
For goodbooks10k, seen in Table 6, we have a more unique set of results as the simple linear stacking performs worse than one of its constituent predictors, KNN. The best performing model on this dataset is MCS@1 (LS) and MCS@5 (LS) beating both SLS and KNN by a significant amount. This highlights that when the gap in performance between models used in an ensemble increases, weighting the constituent models differently becomes important, as the gb10k results show that there is still performance to be gained when combining the strong KNN model with weaker models.

4.5.4 RQ4: Linear Stacking and List Position Stacking comparison. Looking at all three tables LPS is outperformed by LS. This is expected as LPS was conceived as a method to use when an individual recommender is unable to output a meaningful confidence value, but as all three baseline predictors used for ensembling output meaningful confidences LS is expected to have the best performance.
It is worth noting that even though LPS performs worse than LS, it still outperforms all constituent recommenders on the ml10m and ml20m datasets.
These results suggest that LS is the better choice compared to LPS on these datasets, with the chosen constituent recommenders.

4.5.5 RQ5: MCS bucket count impact. In order to evaluate the impact of the choice of buckets for MCS, we have plotted its performance as a function of bucket count in Figure 3. It can be seen from the graphs that the choice of buckets have a negligible impact on the performance of the MCS model. The most distinguishable impact of bucket counts can be seen on the ml10m dataset where 5 buckets beat 1 bucket by 0.0047 on Recall@5. While no value varies by more than 0.0015 on the ml20m dataset, and no value varies by more than 0.0009 on gb10k. Generally no pattern can be seen between bucket count and performance.

4.5.6 RQ6: MCS and NNMS comparison. Answering this question is not possible as NNMS does not give a representative image of a weight optimization on a per user basis. From the two sets of results obtained it would appear that MCS is the best approach. This is however not an interesting finding as RQ3 showed that SLS has better performance than MCS.

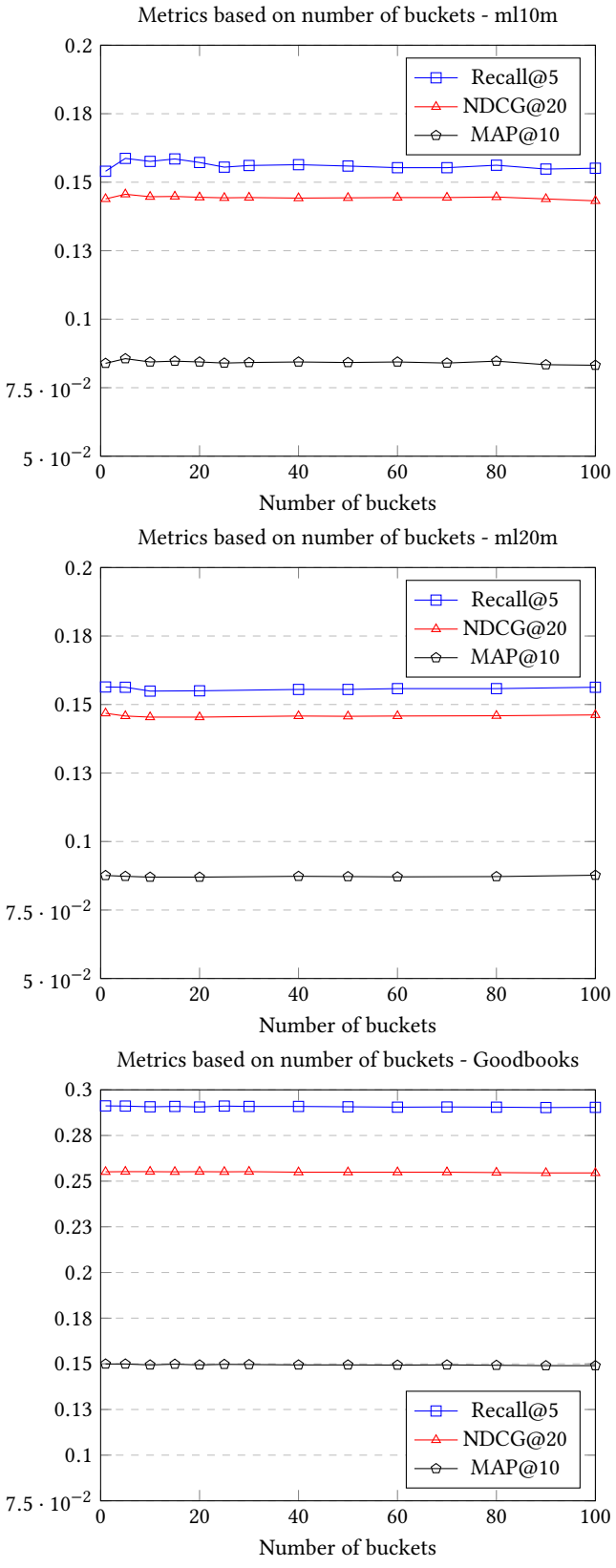

Figure 3: Metrics as a result of varying bucket count

Table 3: Pairwise combinations of recommenders in ensembles on the ml10m (left) and ml20m (right) dataset

Model	Predictors	Recall@5	NDCG@20	MAP@10
SLS	KNN+UPC	0.1479	0.1364	0.0817
SLS	KNN+WMF	0.1468	0.1386	0.0789
SLS	UPC+WMF	0.1487	0.1402	0.0798
SLS	All three	0.1587	0.1468	0.0867

Model	Predictors	Recall@5	NDCG@20	MAP@10
SLS	KNN+UPC	0.1476	0.1360	0.0832
SLS	KNN+WMF	0.1521	0.1420	0.0838
SLS	UPC+WMF	0.1542	0.1431	0.0841
SLS	All three	0.1587	0.1483	0.0894

Model	Predictors	Recall@5	NDCG@20	MAP@10
MCS@1(LS)	KNN+UPC	0.1451	0.1332	0.0787
MCS@1(LS)	KNN+WMF	0.1528	0.1417	0.0821
MCS@1(LS)	UPC+WMF	0.1532	0.1424	0.0817
MCS@1(LS)	All three	0.1540	0.1439	0.0839

Model	Predictors	Recall@5	NDCG@20	MAP@10
MCS@1(LS)	KNN+UPC	0.1438	0.1340	0.0806
MCS@1(LS)	KNN+WMF	0.1551	0.1446	0.0856
MCS@1(LS)	UPC+WMF	0.1554	0.1448	0.0853
MCS@1(LS)	All three	0.1564	0.1468	0.0876

Table 4: Results for the ml10m dataset.

Model	Recall@5	NDCG@20	MAP@10
WMF	0.1292	0.1244	0.0655
UPC	0.1322	0.1232	0.0723
IPC	0.0265	0.0272	0.0101
KNN	0.1419	0.1321	0.0780
RME	0.1534	**0.1470**	0.0843
ERME	0.1497	0.1420	0.0811
SLS	**0.1587**	0.1468	**0.0867**
NNMS	0.1417	0.1318	0.0773
MCS@5(LS)	**0.1587**	0.1456	0.0856
MCS@80(LPS)	0.1512	0.1402	0.0799

Table 5: Results for the ml20m dataset.

Model	Recall@5	NDCG@20	MAP@10
WMF	0.1372	0.1292	0.0723
UPC	0.1343	0.1228	0.0728
IPC	0.0248	0.0259	0.0095
KNN	0.1410	0.1318	0.0793
RME	0.1556	0.1465	0.0868
ERME	0.1523	0.1431	0.0848
SLS	**0.1587**	**0.1483**	**0.0894**
NNMS	0.1423	0.1321	0.0790
MCS@1(LS)	0.1564	0.1468	0.0876
MCS@20(LPS)	0.1519	0.1420	0.0829

Table 6: Results for the gb10k dataset.

Model	Recall@5	NDCG@20	MAP@10
WMF	0.2170	0.2089	0.1027
UPC	0.1532	0.1450	0.0671
IPC	0.0978	0.0928	0.0419
KNN	0.2766	0.2394	0.1422
RME	**0.2958**	**0.2680**	**0.1524**
ERME	0.2654	0.2426	0.1324
SLS	0.2712	0.2430	0.1361
NNMS	0.1606	0.1514	0.0707
MCS@1(LS)	0.2912	0.2551	0.1499
MCS@5(LS)	0.2911	0.2552	0.1499
MCS@40(LpS)	0.2819	0.2501	0.1429

5 CONCLUSION

We introduce NNMS and MCS, novel approaches to adjust weights of ensembling methods, based on meta data about users. The NNMS approach utilizes neural networks to map meta data about a user to a set of personalized weights for ensembling. MCS utilizes a clustering algorithm to assign users to clusters, which can then be assigned a customized set of weights. We also introduce UPC and IPC, derivatives of KNN using SPPMI values as their similarity score, for the purpose of ensembling diverse baseline predictors. Our final ensembles did not use IPC as its performance was not competitive. However our results showed that for both ml10m and ml20m the inclusion of UPC in ensembles yielded better performance.

As part of our experimental evaluation we show that simple ensembles can outperform complex single predictors such as RME. This indicates that the idea of customizing weights is promising, but unfortunately our attempts are unable to show any meaningful improvement using NNMS or MCS.

With the current information we can not argue whether NNMS or MCS can be used to further improve recommendations over simple linear stacking, but we posit that it merits future work.

6 FUTURE WORK

The first area we suggest for future work in this paper is further research on the neural networks approach. The current network structure does not present any meaningful results, we do however still find that the overall idea is sound. The problem is how to cast the problem of weighting the constituent recommenders into a solvable problem for a neural network. We suggest using a listwise loss function and have the neural network learn a stacking function unique for each user, instead of using LS with a set of weights given by the network as we have presented.

The MCS implementation introduced in this paper uses grid search after clustering to find the optimal weights. This will be an issue moving forward especially when used with bigger sets of recommenders, as the set of possible weights grows exponentially. To be able to scale MCS and make it more broadly applicable,

a method for learning these weights e.g. optimization of a loss function, would be necessary. We suggest optimizing this using a pairwise approach similar to the work of He et al. [4], as this is easier to differentiate. However for the final evaluation we suggest the metrics presented in this paper.

As a last improvement we suggest the inclusion of further meta data. We constrained our meta data to be the data solely available from the rating matrix, and if future research wishes to keep this constraint, Sill et al. [13] present an interesting list of meta data for both users and items. Alternatively, some datasets have external meta data available e.g. movie tags in the ml10m dataset. It is a reasonable assumption that more relevant data will lead to increased performance for both NNMS and MCS.

ACKNOWLEDGMENTS

We would like to thank our supervisor, Peter Dolog, for his guidance and constructive criticism throughout the project.

REFERENCES

[1] Leo Breiman. 1996. Stacked regressions. *Machine learning* 24, 1 (1996), 49–64.
[2] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. ACM, 129–136.
[3] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.
[4] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.
[5] Iman Kamehkhosh, Dietmar Jannach, and Malte Ludewig. 2017. A Comparison of Frequent Pattern Techniques and a Deep Learning Method for Session-Based Recommendation.. In *RecTemp@ RecSys*. 50–56.
[6] Mathias Klemmensen and Frederik Kirk Kristensen. 2019. *Co-Occurrence Embeddings: Enriching existing data for Recommender Systems*. Technical Report. Aalborg University - Department of Computer Science.
[7] Yehuda Koren. 2009. The bellkor solution to the netflix grand prize. (2009).
[8] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.
[9] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.
[10] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. 2016. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *Proceedings of the 10th ACM conference on recommender systems*. ACM, 59–66.
[11] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.
[12] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. *Recommender Systems Handbook* (2nd ed.). Springer Publishing Company, Incorporated.
[13] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. 2009. Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460* (2009).
[14] Thanh Tran, Kyumin Lee, Yiming Liao, and Dongwon Lee. 2018. Regularizing Matrix Factorization with User and Item Embeddings for Recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 687–696.
[15] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A meta-learning perspective on cold-start recommendations for items. In *Advances in neural information processing systems*. 6904–6914.
[16] Zygmunt Zajac. 2017. Goodbooks-10k: a new dataset for book recommendations. http://fastml.com/goodbooks-10k. *FastML* (2017).