Multimedia Tools and Applications manuscript No.
(will be inserted by the editor)

Offline Optimization for User-specific Hybrid Recommender Systems

Simon Dooms · Toon De Pessemier · Luc Martens

Received: date / Accepted: date

Abstract Massive availability of multimedia content has given rise to numerous recommendation algorithms that tackle the associated information overload problem. Because of their growing popularity, selecting the best one is becoming an overload problem in itself. Hybrid algorithms, combining multiple individual algorithms, offer a solution, but often require manual configuration and power only a few individual recommendation algorithms. In this work, we regard the problem of configuring hybrid recommenders as an optimization problem that can be trained in an offline context. Focusing on the switching and weighted hybridization techniques, we compare and evaluate the resulting performance boosts for hybrid configurations of up to 10 individual algorithms. Results showed significant improvement and robustness for the weighted hybridization strategy which seems promising for future self-adapting, user-specific hybrid recommender systems.

Keywords Recommender systems · Hybrid · Algorithms · RMSE · Optimization

1 Introduction

The availability of multimedia content nowadays, is booming exponentially in a wide variety of domains. Through the Internet, users have access to unlimited music resources (e.g., *Spotify*, *Pandora*, etc.), video platforms (e.g., *YouTube*, *Dailymotion*, etc.), image galleries (e.g., *flickr*, *Instagram*, etc.) and many more. Accessing these resources has become so easy that users are quickly overloaded with information, and often find it hard to select relevant content. As a solution, mul-

S. Dooms - T. De Pessemier - L. Martens
Wica, iMinds-Ghent University
G. Crommenlaan 8 box 201, 9050 Ghent, Belgium
Tel.: +32-09-33-14908
Fax: +32-09-33-14899
E-mail: simon.dooms@intec.ugent.be
E-mail: toon.depessemier@intec.ugent.be
E-mail: luc.martens@intec.ugent.be

timedia tools or applications are increasingly integrating recommender systems to support their users.

Recommender systems are systems designed to tackle the problem of information overload by automatically selecting interesting content for users based on their personal preferences. The domain of recommender systems has been an active research topic for over twenty years now, and still continues to expand. Years of research contributions by hundreds of researchers, have led to an abundance of recommendation algorithms that can be used to combat information overload. Recommendation algorithms come in all sorts and sizes, from really simple ones as *SlopeOne* [20] to extensive mathematical-based ones as *SVD++* [18]. Algorithms can be based on collaborative filtering principles, content-based, knowledge-based, demographic-based, etc. More recently also context-based [2,15] and social-based [30] algorithms are starting to show up. Each and every one of these algorithms have their own advantages, downsides and optimal use cases and scenarios. Also the availability of recommendation frameworks (or platforms) that offer out-of-the-box recommendation solutions is on the rise, these include MyMedialite [13], LensKit [12], Mahout[1], Lucene[2], Duine[3], etc. It seems that, by eagerly tackling the information overload problem with new methods, the recommender domain in itself is becoming overloaded with available algorithms and thus making it more difficult (for recommender system administrators) to select the right algorithm for the job.

Instead of selecting one algorithm, sometimes multiple algorithms are combined into a so-called *hybrid* recommender system. Hybrid recommender systems have long been popular, and are widely used in many real-world applications because of their obvious advantages over individual recommendation algorithms. By combining and integrating different types of recommendation algorithms, hybrid recommenders are able to overcome the drawbacks associated with each of them individually [1,3,7]. A problem that most hybrid recommenders nowadays are facing, is that they are inherent static in nature. Very often they are trained or manually tweaked before deployment, but at runtime their configuration remains the same. Their static nature prevents them from being deployed in other scenarios, with other algorithms, or for other (types of) users.

Because hybrid systems are cumbersome to configure (often done manually), the number of incorporated individual algorithms is usually rather limited to two or three algorithms at most. Since hybrid systems inherit the properties of its individual components, it seems more interesting however to have hybrid recommenders composed of dozens of algorithms instead of just a few. Ideally, a hybrid recommender system would include all existing recommendation algorithms and be capable of intelligently deciding what algorithm (or what combination of algorithms) generates the most interesting results for any given user in a system.

In this work, we strive towards the ideal hybrid recommender system which automatically fine-tunes itself based on given individual recommendation algorithms and user input data. We focus specifically on two hybridization techniques i.e., hybrid switching and weighted hybridization and include up to 10 individual algorithms in our experiments. The contributions of this paper include:

[1] http://mahout.apache.org
[2] http://lucene.apache.org
[3] http://www.duineframework.org

1. Formulation of the hybrid recommendation configuration problem as an optimization problem.
2. The introduction of an offline optimization procedure for a user-specific switching and weighted hybrid strategy.
3. A training and test dataset division strategy that allows both offline training, and realtime updates (on individual user-level) for online recommendation scenarios.
4. Comparison and evaluation of the optimization potential (in terms of RMSE) of both the switching and weighted hybrid strategies.

The remainder of the paper is organized as follows. In Section 2 we discuss the current state of the art for hybrid recommender systems. We then present an overview of our system's architecture in Section 3. Section 4 details the transformation from configuring a hybrid system to solving an optimization problem and discusses evaluation. In the two following sections 5 and 6, we show how both the switching and the weighted hybridization techniques can be implemented so that they allow for optimization. Section 7 presents results for both of the hybridization techniques, using the performance of individual algorithms (and naive hybrid systems) as baseline for comparison. Finally, the implications of the results are put into perspective in the discussion (Section 8) and conclusion (Section 9).

2 Related Work

Burke et al. [9] was one of the first to categorize hybrid recommender systems in function of their combining strategies: *weighted, switching, mixed, feature combination, cascade, feature augmentation,* and *meta-level.* Every combining strategy comes with its own specific properties and consequences.

Two of the most commonly combined recommendation algorithms are the content-based and the collaborative filtering approach [4,14,23], because they tend to complement each other in various ways. Very often only two algorithms are combined using a simple combining strategy, e.g. [27] where the predictions of two types of collaborative filtering systems are combined linearly (*weighted*) using the formula $P = \alpha \times P_{algo_1} + (1 - \alpha) \times P_{algo_2}$.

A recent hybridization technique is feature-weighted linear stacking (FWLS) [29], which continues on the original concept of stacking [32], where multiple recommendation models are stacked (or *blended*) together. The most famous application of stacking is the winning entry of the Netflix Prize[4], where the *BellKor's Pragmatic Chaos* team stacked more than hundred different models together into one blend (in fact even a blend of blends) [19,25,31]. The advantage of the stacking technique is that individual components are loosely coupled, which allows easy integration of new algorithms and tuning of the end results by adjusting the individual coefficients (i.e., weights) of the models. These coefficients are usually determined by taking into account so called *meta-features*, which are metrics describing some specific properties of the dataset at hand. In FWLS, the coefficients associated with the models are parametrized as linear functions of the meta-features. The STREAM (Stacking Recommendation Engines with Additional Meta-Features) system [5] experimented with eight different meta-features,

[4] http://www.netflixprize.com

and ultimately considered the number of user ratings and item ratings the most interesting.

Individual weights of the models are usually determined by fitting some sort of linear regression to optimize general recommendation metrics such as RMSE. Recent research however, is moving away from generalizing users, and towards a *per user* focus. Ekstrand et al. [11] showed that recommenders fail (and succeed) on different items and users (although their focus was on switching hybrids). Hybrid recommender systems would obviously benefit from being able to predict which recommendation algorithm works best for which user. The importance of individuality of users was also noted by the work of Kille et al. [16], in which they tried to model the difficulty of generating recommendations for individual users.

In [21], the authors describe their system *SemanticMovie* which integrates multiple recommendation approaches (*recommender agents* in their words) into a single agent *ensemble*. Combining weights are generated by default, but can be manually adjusted per user. Automatically adjusting weights (through a *learning component*) according to user feedback was however deferred to future work. The topic of automatically adjusting user-specific weights for dynamic ensembles has been touched by Bellogín et al. [6,7]. From an information retrieval perspective, they proposed adaptations of query performance techniques to define performance predictors in recommender systems. Using these predictors, recommendation strategies can then be dynamically fine-tuned. The selection of predictors and individual recommenders to use in the ensemble is limited by specific constraints e.g. the predictors should correlate positively with the performance of not all but some recommenders. Our work on the other hand, considers a machine learning approach where the weights of the ensemble can be iteratively trained where algorithms (considered 'black boxes') of any type can be combined.

In our work we integrate the most relevant ongoing topics of hybrid recommender research. We focus on dynamically optimizing user-specific hybrid systems through a machine learning perspective. We compare the popular hybrid switching approach with the weighted hybrid strategy and show how both can be optimized, yielding a performance boost versus individual recommendation algorithms or static hybrid systems.

3 General Architecture

Our general architecture corresponds to a common hybrid recommender system layout as depicted in Fig. 1. The original ratings dataset is divided into a training and test set, which are then respectively used as input for the hybrid recommender and for the final evaluation of the system. The hybrid recommender system consists of multiple recommendation algorithms which run in parallel on the provided data and are finally aggregated into hybrid recommendation results.

In this work, we employ the MovieLens 100K dataset which is a popular and commonly used dataset in recommender research. It contains 100K ratings by 943 users of 1682 movies. Every rating is a numerical value ranging from 1 (i.e., awful) to 5 (i.e., must see), expressing the interest of the user in a particular movie. For more information on this dataset we refer to recommender systems literature (e.g. [8,24]). Working with the MovieLens dataset has the advantage that every user in the system will have at least rated 20 items. Since we are tackling the hybridization

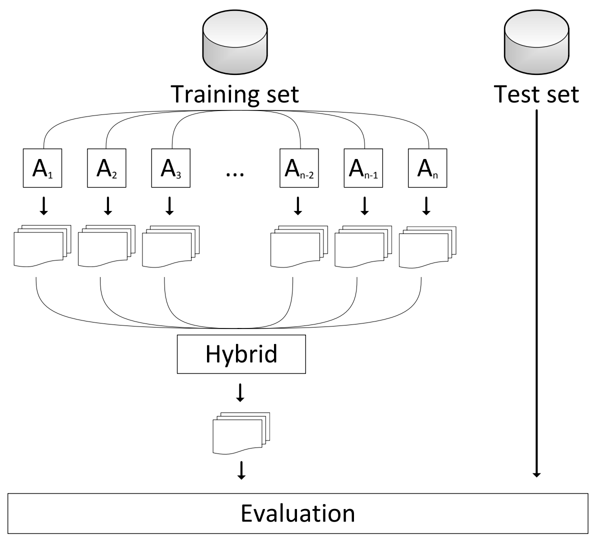

Fig. 1 The high-level architecture of the hybrid system. Rating data is split in a training and test set. Multiple recommendation algorithms then generate recommendations, which are combined into one set of recommendations and finally evaluated based on the test set.

problem on a user-specific level, we also want to evaluate user-specific and so train and test sets will contain all users but only a ratio of their ratings. For each user we adopt a split-ratio of 60% (train) and 40% (test) ratings. Because every user has rated at least 20 items, every user will have a minimum of 8 ratings in the test set.

To serve as individual recommendation algorithms in the hybrid system, we selected the following 10 algorithms from the rating predictors available in the MyMediaLite[5] framework [13]. For each of these algorithms, default settings were used as set in MyMediaLite version 3.09.

- BiasedMatrixFactorization
- MatrixFactorization
- FactorWiseMatrixFactorization
- SigmoidSVDPlusPlus
- BiPolarSlopeOne
- SlopeOne
- UserItemBaseline
- UserKNN
- Constant1
- Constant5

The output of each of these single algorithms is directed to the 'Hybrid' module, where results are collected and aggregated. We specifically focus (and compare) two common hybridization strategies: a *switching* approach, where only the best algorithm is selected, and a *weighted* approach where all algorithms contribute to the final recommendations according to a specific weight.

[5] http://mymedialite.net

In the end, the system is evaluated according to a k-fold evaluation procedure. The training dataset is used throughout the system to calculate the individual and finally the hybrid recommendation values. Subsequently, the results are compared with the test set. This process is repeated 10 times (i.e., 10 folds). The specific evaluation metric that is calculated and used throughout this work, is recommendation accuracy in the form of RMSE as detailed in the next section.

4 Offline optimization for hybrid recommenders

In this work, we attempt to optimize hybrid recommender systems in an offline setting. To be able to handle the problem of selecting or composing an optimal hybrid recommender, we consider our problem as an optimization task.

$$\underset{x}{minimize}\ f(x)$$

Here, the goal is to minimize a defined objective function $f(x)$ by providing it with an optimal input x. For the hybrid recommendation use case, the objective function could be an evaluation metric which we want to optimize (e.g., accuracy, diversity, serendipity, etc.) and the input x the hybrid recommender. The better the recommender, the better values the objective function will come up with.

Since we are working in an offline setting, we need to select an objective function (i.e., evaluation metric) that can be evaluated offline, or in other words, without requiring additional user input. For the purpose of demonstrating the offline optimization procedure described in this work, we choose to optimize the accuracy metric: *Root Mean Squared Error* (RMSE). This method is defined as

$$RMSE = \sqrt{\frac{1}{|\tau|} \sum_{(u,i)\in\tau} (\hat{r}_{ui} - r_{ui})^2}$$

where system-predicted ratings $\hat{r}_{ui}$ are compared with true ratings r_{ui} contained in a certain test set τ of user-item pairs (u, i) [28]. We adopt RMSE because it is one of the most popularly used evaluation metrics in the recommender systems domain and is easily computed in an offline context. Recent research [22, 26, 10, 17] shows that although recommendation accuracy is principal to achieve user satisfaction, it is not the only important metric and often metrics as diversity, transparency or trust should be considered as well. We note however, that our offline optimization strategy is not limited to RMSE, and can incorporate any desired metric as long as it can be calculated offline on a given set of ratings.

4.1 Evaluating optimization

Aside from defining the objective function (i.e., RMSE), we also need data to calculate $f(x)$ given a certain x. Two available datasets are the training set and the test set. The test set is clearly unusable since we want to avoid *tuning to the test set* ; the test set should only be used in the final evaluation of the complete system and not in intermediate optimizing iterations, to guarantee evaluation fairness. We therefore focus on using the training set as input for our optimization task. In other words, we intend to optimize our hybrid system in terms of RMSE on the training set. We then hypothesize that, a decrease of RMSE (lower RMSE is better) on the training set will result in a similar decrease on the test set. This may not be the case if the optimization that was *learned* from this procedure is not generalizable i.e., too specific to the training set.

To prevent overfitting the training set, we do not train our optimization on the full training set, but rather on 10 distinct subsets of this training set. Such a subset divides the training set into a smaller subtraining set and subtest set, much like the general 10-fold evaluation procedure of the system, with a user-specific ratio of 60/40 (Fig. 2). The objective function results are then averaged (arithmetic mean) over these 10 subdatasets. A downside of this optimization procedure, is that it requires to run the individual algorithms on 11 different datasets: 10 subfolds of the training set (to optimize the hybrid), and once on the full training set (for the final recommendations).

Once the individual algorithms are trained and ready, our evaluation strategy allows very fast updating and processing of new feedback given by the user. Usually, when a user provides new ratings, the recommender needs to be retrained to take this new data into account. Since training the algorithms can take a long time, very often online recommender systems recalculate their models only a few times a day. With our evaluation strategy, new ratings can be dynamically integrated by adding them to the subtest datasets. That way the hybrid recommender can take them into account without retraining the individual algorithms. This strategy paves the path towards online hybrid recommender systems that respond realtime to new user feedback.

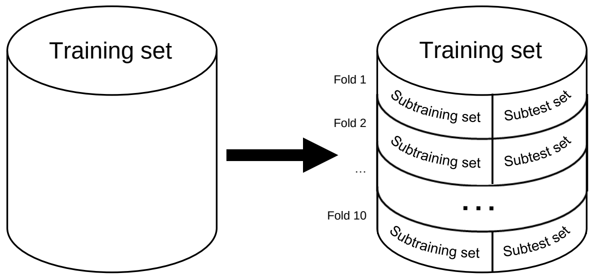

Fig. 2 The training set is split into 10-folds with a 60/40 subtraining set and subtest set, which are then used to optimize the hybrid configuration.

With a defined objective function and data to train (and evaluate) on, we can now proceed to the optimization of our hybrid recommender system. In the next sections we explore both the *switching* and *weighted* hybrid strategies.

5 Hybrid switching strategy

The *hybrid switching* technique entails the switching between different recommendation algorithms by means of a switching strategy [9]. It is a very easy and straightforward method since in the end only one algorithm will contribute to the final recommendations. The selection of the 'best' recommendation algorithm for the situation often depends on objective metrics such as metadata availability (e.g., content-based when item features are available), the number of total ratings (e.g., collaborative filtering when a large number of items have been rated), etc. However, since we are optimizing hybrid recommender systems at user-level, the switching strategy should also be implemented at user-level so that for every user, a single best algorithm can be selected.

Our user-specific switching selection strategy, starts with the determination of a default algorithm. This default algorithm serves as a fallback option when no clear 'best' algorithm could be detected for a user. The default algorithm is determined by evaluating RMSE values for all algorithms over all users on the subtest datasets, and so will be the same for every user.

Next, for every user a 'best' algorithm is detected among the available individual recommendation algorithms. Best in this case, is defined as providing the best (averaged out) RMSE values on the 10 subtest datasets. Aside from best RMSE, also the variance among the 10 (i.e., one for every subtest dataset) RMSE values is taken into account. This variance serves as a confidence value indicating the stability of the RMSE values among the different subtest datasets. A high variance indicates divergent RMSE values and so the average may not be a good prediction of the RMSE value the algorithm will finally deliver. Our experiments have shown that a reasonable method of coping with this situation, is imposing a cutoff variance threshold that, when reached, automatically discards the algorithm from the selection process (for the current user). This way only algorithms which show good and constant RMSE values across the subtest datasets will be compared and ranked. When all, or all but one, algorithm is discarded, the default algorithm is selected for the current user.

Although our experiments showed good results implementing our discrete cut-off threshold, it may be interesting to explore more continuously-based solutions. In future work, we plan on experimenting with confidence intervals for the scores of each algorithm to integrate variance values in a more flexible way.

Algorithm. Best Switching Selection Strategy

```
    default_algo ← determine_default_algorithm()
    for user in users
    algo_selection_list ← {empty}
        for algorithm in algorithms
            RMSE, variance ← evaluate_algorithm(user, algorithm)
            if variance < VARIANCE_THRESHOLD
                algo_selection_list ← algo_selection_list + (RMSE, algorithm)
            end if
        end for
        if length of algo_selection_list > 2
            user_algorithm ← select algorithm with lowest RMSE
        else
            user_algorithm ← default_algo
        end if
    end for
```

PROCEDURE - determine_default_algorithm()

```
    all_algo_values ← {empty}
    for algorithm in algorithms
```

```
        algo_values ← {empty}
        for user in users
            RMSE ← evaluate_algorithm(user, algorithm)
            algo_values ← algo_values + (algorithm, RMSE)
        end for
        RMSE ← average of algo_values
        all_algo_values ← all_algo_values + (algorithm, RMSE)
    end for
    algorithm ← select best from all_algo_values
    return algorithm

PROCEDURE - evaluate_algorithm(user, algorithm)
    RMSE_list ← {empty}
    for subtest_set in subtest_sets
        RMSE ← {RMSE of real ratings in subtest_set and predicted ratings by
algorithm}
        RMSE_list ← RMSE_list + RMSE
    end for
    RMSE ← average of RMSE_list
    variance ← variance of RMSE_list
    return RMSE, variance
```

For the experiments described in this work, we used a cutoff variance threshold value of 0.2.

6 Weighted hybrid strategy

Another hybridization technique we employ in this work, is the *weighted hybrid* recommendation technique. With this method the scores of individual recommendation algorithms are combined together into a single hybrid recommendation score [9]. Individual algorithms can be associated with different weights to allow fine-grained control over the contribution of the individual algorithms to the final score. Weights can be set in such a way that this weighted technique produces the same results as hybrid switching (i.e., if only one algorithm contributes to the final score). The real power of the weighted technique however, lies in the ability to join multiple algorithms together and form a new hybrid algorithm. We therefore hypothesize that this technique may yield better or at least equal results as the hybrid switching technique.

An additional advantage of the weighted hybrid strategy towards other strategies like Meta-Level or Feature Augmentation is its black box approach. Individual algorithms are considered *black boxes*, which are served input data and produce output data without revealing any internal processing information. Since the final score takes only the output of the algorithms into account, new algorithms can easily be added to the system without the need for structural changes. In this work, we intend to optimize a hybrid system built on many (i.e., 10) different algorithms, and so the black box approach seems a valuable advantage.

The core challenge of the weighted hybrid technique is to find appropriate weights for the individual algorithms. We define this problem in the form of an optimization task. We adopt the notation from related work [6] where a *dynamic ensemble recommender* was defined as

$$g(u,i) = \gamma_{a_1} * g_{a_1}(u,i) + \gamma_{a_2} * g_{a_2}(u,i) + ... + \gamma_{a_n} * g_{a_n}(u,i)$$

where γ is the weighting factor for the individual algorithms a that weighs the recommendation values $g_a(u,i)$ for a user u and item i. Since our approach is a user-specific one, we want to optimize the weights of the algorithms specifically for every user so that every user may benefit from a personalized hybrid recommender system. User-specific weights can be added to the optimization task in the following way.

$$g(u,i) = \gamma_{a_1}(u) * g_{a_1}(u,i) + \gamma_{a_2}(u) * g_{a_2}(u,i) + ... + \gamma_{a_n}(u) * g_{a_n}(u,i)$$

We can now denote the objective function as

$$f(\gamma(u))$$

where

$$\gamma(u) = (\gamma_{a_1}(u), \gamma_{a_2}(u), ..., \gamma_{a_n}(u))$$

with n the total number of recommendation algorithms. Through an optimization process we seek to minimize the objective function (i.e., RMSE). This metric can again be measured offline by evaluating the subtest sets and averaging out the values as was done to optimize the hybrid switching strategy.

We limit possible weight values to the interval [0,1] so that values can be easily interpreted and compared. A final recommendation score for a user u and item i, given the weight vector and individual recommendation scores, can then be calculated by means of an average weighted formula.

$$g(u,i) = \frac{\gamma_{a_1}(u) * g_{a_1}(u,i) + \gamma_{a_2}(u) * g_{a_2}(u,i) + ... + \gamma_{a_n}(u) * g_{a_n}(u,i)}{\gamma_{a_1}(u) + \gamma_{a_2}(u) + ... + \gamma_{a_n}(u)}$$

In our weighted hybrid optimization procedure, $\gamma(u)$ will be optimized such that $g(u,i)$ minimizes RMSE on the subtest datasets. In this procedure we iteratively try to improve the individual weights of the weight vector. To reduce the number of iterations required for this optimization, we start the procedure by selecting the best start vector out of a number of randomly generated weight vectors. Together with the random vectors, we compare all the *individual weight vectors* for every single recommendation algorithm in the system. We define an individual weight vector as a vector that allows only a single algorithm to contribute to the final score e.g, $\gamma(u) = (1,0,0,0,0,0,0,0,0,0)$. Doing so, forces the hybrid to take also individual algorithms into account.

After the selection of the start vector, this weight vector will be improved one weight at a time. Since this optimization procedure must run for every user in the system, we want an optimization procedure that produces qualitative results in a very short time frame (almost realtime). For this, we have implemented a standard binary search procedure where improved weight values will be searched in iteratively diminishing intervals both upwards and downwards as Fig. 3 illustrates. We have experimented with other optimization procedures such as a custom genetic algorithm and publicly available optimization tools (such as SciPy[6]), but found

[6] http://docs.scipy.org/doc/scipy/reference/optimize.html

our standard binary search procedure to produce the best or equal results in less time. To reduce the risk of ending up in locally-optimal RMSE solutions, instead of selecting only one start vector (from the randomly generated ones), multiple vectors could be optimized simultaneously. We experimented with different settings and even implemented a back-tracking mechanism to allow the optimization procedure to escape locally-optimal solutions. We found that this approach did not yield significantly improved results and therefore we did not include this in our algorithm presented below. We hypothesize that the randomization of the start vector in itself already prevents ending up in the most obvious local RMSE solutions.

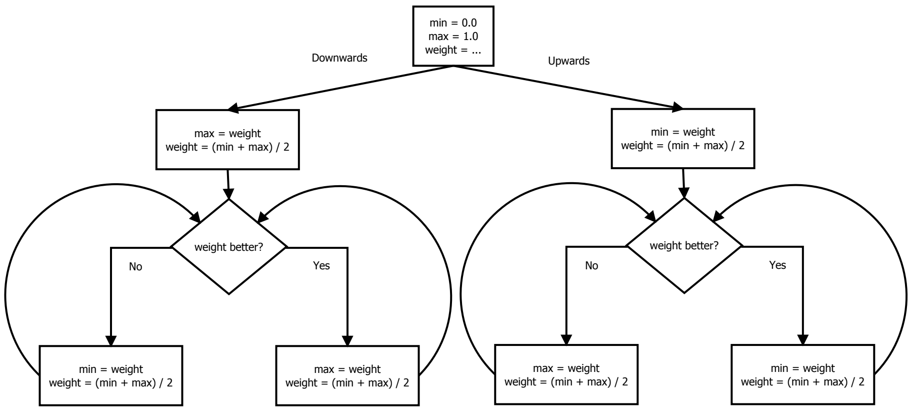

Fig. 3 The binary search procedure to improve the weights in the weight vector.

We note that although the procedures searches for an improvement one weight at a time, the evaluation (i.e., is the new weight value better?) will be performed on the weight vector as a hole. If for example, the start weight vector is $\gamma(u) = (0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ and we seek an improvement of the first weight in upwards direction, we will evaluate if the weight vector $\gamma(u) = (0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ yields improvement. Improvement, as stated earlier, is defined here as yielding a reduced RMSE value on the subtest datasets. We again take the variance of the generated RMSE values into account to model for confidence. When a cutoff threshold for variance is reached, the evaluation discards the weight vector and the weight suggestion is considered 'not better'.

The binary search procedure is repeated for every weight in two directions until no more improvements can be found or a fixed number of iterations have passed.

Algorithm. Weighted Average Strategy

```
    $weights\_vectors \leftarrow$ {random weight vectors, and all individual weight vectors}
    $weights\_vector \leftarrow$ select the best from $weights\_vectors$
    $iterations \leftarrow 0$
    $current\_RMSE \leftarrow$ evaluate_weights_vector($weights\_vector$)
    $previous\_RMSE \leftarrow current\_RMSE + 1$ //start worse than $current\_RMSE$
    while ($iterations <$ MAX_ITERATIONS) and ($previous\_RMSE > current\_RMSE$)
```

```
        previous_RMSE ← current_RMSE
        weight_vector ← optimize_weights_vector(weights_vector)
        current_RMSE ← evaluate_weights_vector(weights_vector)
        iterations ← iterations + 1
    end while

PROCEDURE - optimize_weights_vector(weights_vector)
    old_RMSE ← evaluate_weights_vector(weights_vector)
    for weight in weights_vector
        new_RMSE, new_weight ← upwards binary search for improved weight
        if new_RMSE < old_RMSE
            return weights_vector with new_weight
        end if
        new_RMSE, new_weight ← downwards binary search for improved weight
        if new_RMSE < old_RMSE
            return weights_vector with new_weight
        end if
    end for
    return weights_vector

PROCEDURE - evaluate_weights_vector(weights_vector)
    RMSE_list ← {empty}
    for subtest_set in subtest_sets
        RMSE ← {RMSE of real ratings in subtest_set and predicted ratings by
calculating the weighted avarage score with weights_vector}
        RMSE_list ← RMSE_list + RMSE
    end for
    RMSE ← average of RMSE_list
    variance ← variance of RMSE_list
    if variance > VARIANCE_THRESHOLD
        return worst case RMSE, so this weights_vector will be discarded
    else
        return RMSE, variance
    end if
```

For the experiments described in this work, we used the same variance cutoff threshold value as for the hybrid switching strategy (i.e., 0.2). The maximum number of iterations was set to 500 and 1000 random weight vectors were used to boost the start weight vector.

7 Results

In this section, we evaluate the performance of the offline optimization of the two hybridization techniques discussed in this work. We first evaluate the performance (in terms of rating prediction accuracy) of the 10 individual recommendation algorithms that were selected from the MyMediaLite framework. We then continue to evaluate and compare the hybrid switching and weighted hybrid strategies. While evaluations for the sake of our optimization procedures used the subtest datasets, all the results in this section have been calculated on the true test set.

7.1 Individual algorithms

We evaluated the RMSE values of the 10 individual algorithms, averaged over all users (and 10 folds). Default parameter settings where applied as set in MyMediaLite version 3.09. Table 1 shows the results of the RMSE evaluation.

Method	*RMSE*
UserKNN	0.9458
UserItemBaseline	0.9473
SlopeOne	0.9476
BiasedMatrixFactorization	0.9576
BiPolarSlopeOne	0.9759
MatrixFactorization	0.9767
FactorWiseMatrixFactorization	0.9817
SigmoidSVDPlusPlus	1.2808
Constant5	1.7420
Constant1	2.7912

Table 1 The RMSE values for the individual algorithms averaged out over all users according to a 10-fold evaluation on the test set. Results are sorted from low to high RMSE.

As the table shows, many of the individual algorithms show a similar performance. Exceptions to this are the *SigmoidSVDPlusPlus*, *Constant5* and *Constant1* algorithms, which perform considerably worse. Closer inspection of the results of *SigmoidSVDPlusPlus*, revealed high variance in results so that for some users the algorithm performed considerably worse than for others (a result we have already encountered in our previous work [10]). We hypothesize the *SigmoidSVDPlusPlus* algorithm needs more data to increase its relevancy or needs at least better fine-tuned parameters. For the *Constant* algorithm (which predicts a constant value of 1 or 5), poor RMSE results were to be expected.

To observe the relationships of the algorithms among each other, we plotted the RMSE values for the 7 most similar algorithms showing their respective 95% confidence intervals in Fig. 4. From the figure it now more clearly shows that *UserKNN*, *UserItemBaseline* and *SlopeOne* can be considered to have equal performance as well as *BiPolarSlopeOne*, *MatrixFactorization* and *FactorWiseMatrixFactorization*. The *BiasedMatrixFactorization* resides somewhere in between. These results are confirmed as we calculate the statistical significant differences among the algorithms with a Wilcoxon Signed-Rank Test (Table 2).

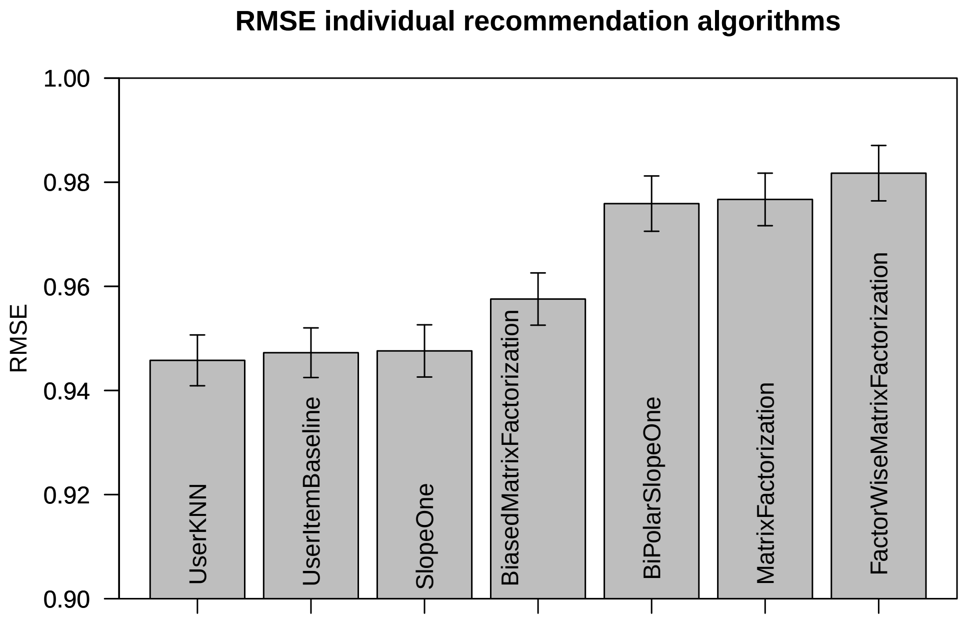

Fig. 4 The RMSE values for the 7 most similar individual recommendation algorithms, averaged out over all users according to a 10-fold evaluation on the test set.

	1	2	3	4	5	6	7	8	9	10
1: UserKNN	-	.302	.864	**	**	**	**	**	**	**
2: UserItemBaseline		-	.396	**	**	**	**	**	**	**
3: SlopeOne			-	**	**	**	**	**	**	**
4: BiasedMatrixF...				-	**	**	**	**	**	**
5: BiPolarSlopeOne					-	.589	.255	**	**	**
6: MatrixFactorization						-	.551	**	**	**
7: FactorWiseMatrixF...							-	**	**	**
8: SigmoidSVD++								-	**	**
9: Constant5									-	**
10: Constant1				(**: $p < .05$)						-

Table 2 Pair-wise p-values of the null hypothesis, that the two systems have an equal performance, as computed by a Wilcoxon Signed-Rank Test.

7.2 Hybrid switching

For the evaluation of the hybrid switching strategy, we performed multiple experiments, each time increasing the number of individual algorithms that were used in the hybrid recommender. The algorithms are added in order of their individual performance, from good (low RMSE) to bad (high RMSE). We define the following hybrid systems in the experiment:

$$\begin{aligned}
BS2 &:= UserKNN + UserItemBaseline\\
BS3 &:= BS2 + SlopeOne\\
BS4 &:= BS3 + BiasedMatrixFactorization\\
BS5 &:= BS4 + BiPolarSlopeOne\\
BS6 &:= BS5 + MatrixFactorization\\
BS7 &:= BS6 + FactorWiseMatrixFactorzation\\
BS8 &:= BS7 + SigmoidSVD{+}{+}\\
BS9 &:= BS8 + Constant5\\
BS10 &:= BS9 + Constant1
\end{aligned}$$

BS stands for 'Best Switching' and the index refers to the number of individual algorithms participating in the hybrid recommender. For every one of these setups, we optimize the hybrid into selecting the best individual algorithm for every user specifically. The results, as calculated on the test set, are shown in Fig. 5.

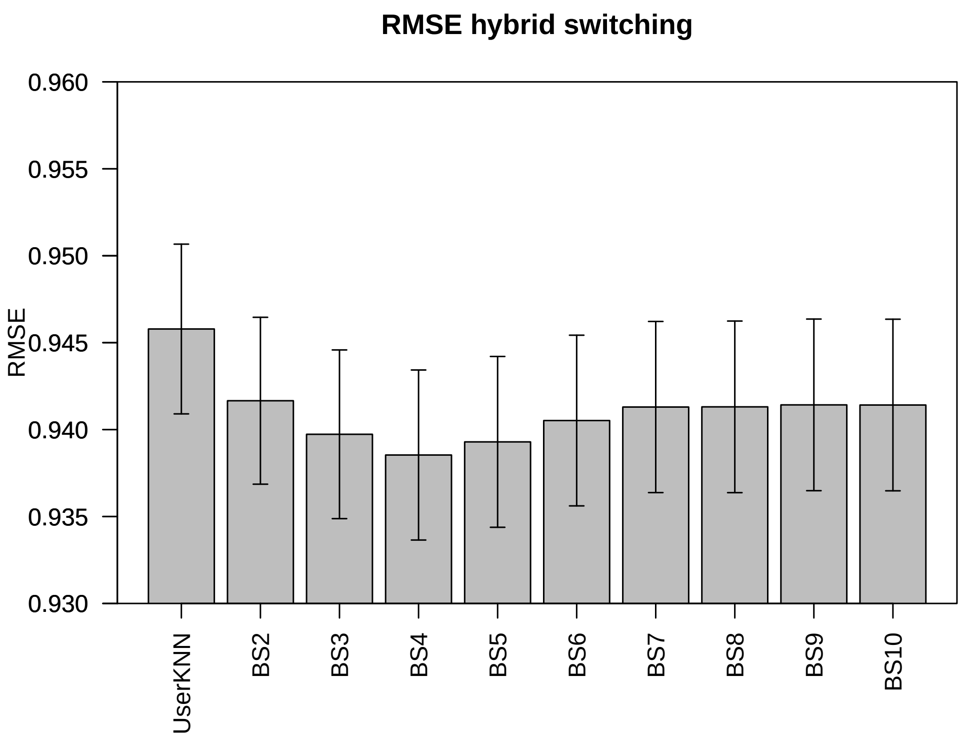

Fig. 5 The RMSE results for multiple experiments with a hybrid switching setup. The index in the x-axis labels refers to the number of individual algorithms participating in the hybrid recommender.

The result of the best individual algorithm (i.e., *UserKNN*) was added to the plot as a baseline for comparison. The figure demonstrates how each of the hybrid recommenders in the experiment achieves better results (i.e., lower RMSE values) in comparison with the best individual algorithm. Adding individual algorithms improves the result of the hybrid up to a number of 4 algorithms (i.e., *BS4*), results then slightly deteriorate and finally stabilize in the end.

From the results it is clear that our user-specific hybrid switching strategy performs better than simply selecting the best individual algorithm and using that for every user. It is however interesting that the results do not continue to improve when additional algorithms are added. The reason for this, is that the systems *BS5*, *BS6* and *BS7* add variations of already included algorithms (i.e, *SlopeOne* and *BiasedMatrixFactorization*) rather than adding completely new algorithms as is the case for *BS2*, *BS3* and *BS4*. These variations perform considerably worse than their original algorithm, and so adding them only brings noise into the system. The performance of the last systems *BS8*, *BS9* and *BS10* remains stable because for these hybrid recommenders the performance of the added algorithms (i.e., *SigmoidSVD++*, *Constant1*, and *Constant5*) is so bad (or their variance so high) that they will never be selected as 'best' algorithm for a user. When we inspect, for *BS10*, the number of times each algorithm was chosen (Fig. 6), we find our argument confirmed.

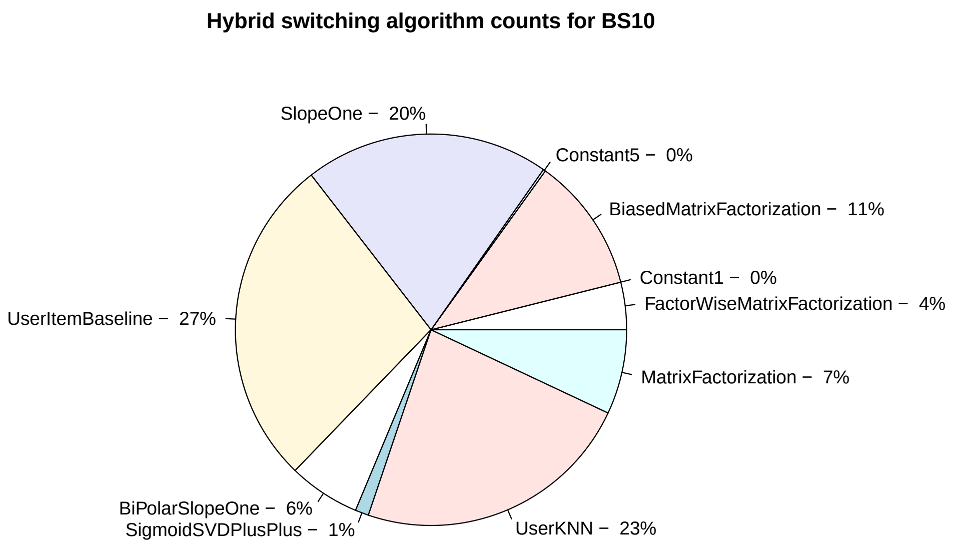

Fig. 6 The number of times each algorithm was selected as 'best' algorithm, counted over all users for hybrid setup *BS10*.

When comparing the RMSE results for the hybrid switching systems, only two systems were found to have a statistical significant difference with $p < 0.05$: the *UserKNN* algorithm and the *BS4* hybrid system, although on average all hybrid systems seem better than *UserKNN*. In conclusion we state that, the best performing hybrid switching strategy (i.e., *BS4*) yields a significant improvement towards the individual algorithms approach, but the participating algorithms in the hybrid must be carefully chosen in order to obtain good results.

7.3 Weighted hybrid

For the evaluation of the weighted hybrid strategy, experiments were iteratively performed with an increasing number of individual recommendation algorithms participating in the hybrid system. Every hybrid now optimizes the weight vectors indicating the importance of the individual algorithms on a user-specific basis. The final prediction scores for every user are compared against the true ratings in the test set.

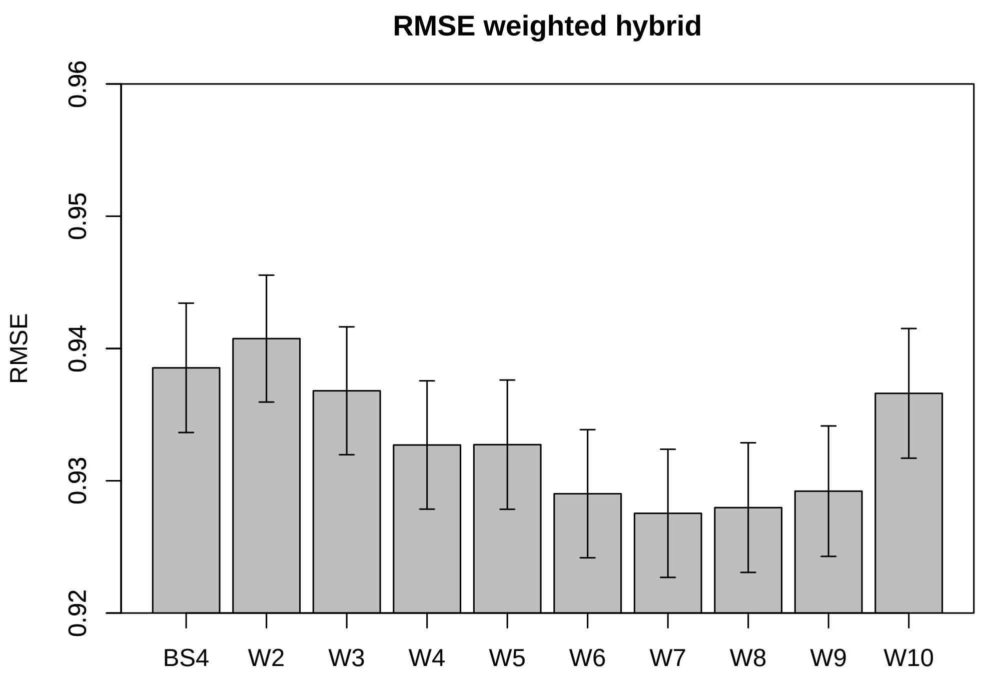

Fig. 7 The RMSE results for multiple experiments with a weighted hybrid setup. The index in the x-axis labels refers to the number of individual algorithms participating in the hybrid.

Fig. 7 shows the RMSE results for multiple hybrid configurations with a varying number of participating individual algorithms. Table 3 depicts the statistical significance of the difference between the algorithms. Algorithms are added in the same order as detailed in the previous subsection (e.g., $W3 = W2 + \mathit{SlopeOne}$). As a comparison baseline, the best result of the hybrid switching strategy (i.e., *BS4*) was added to the plot.

The results show that most of the weighted hybrid configurations (except for *W2*) perform better than the *BS4* baseline. Moreover, the performance increases (or at least remains stable) when new algorithms are added. Exceptions are the *W9*

	BS4	W2	W3	W4	W5	W6	W7	W8	W9	W10
BS4	-	.327	.781	.105	.114	**	**	**	**	.622
W2		-	.207	**	**	**	**	**	**	.140
W3			-	.177	.192	**	**	**	**	.831
W4				-	.967	.331	.150	.122	.222	.257
W5					-	.311	.138	.112	.207	.277
W6						-	.636	.562	.799	**
W7							-	.913	.829	**
W8								-	.746	**
W9									-	**
W10					(**: $p < .05$)					-

Table 3 Pair-wise p-values of the null hypothesis, that the two systems have an equal performance, as computed by a Wilcoxon Signed-Rank Test.

and *W10* configurations, which show a decreased performance (increased RMSE) towards the previous configurations. This is caused by the algorithms that are added in those configurations, namely *Constant5* and *Constant1*. The *Constant* algorithm predicts always the same recommendation score (i.e., here either 1 or 5). When considering the weighted average formula used for this hybrid strategy (Section 6), adding a *Constant* algorithm in the equation will function as a general weighting factor for the final prediction value. This weighting factor can either boost or decrease the final recommendation score. When the *Constant5* algorithm is used in the calculation of the final prediction score, all predicted scores (for that user) will be slightly increased or similarly decreased when applying *Constant1*. The reason that *Constant1* has a bigger (negative) impact on performance becomes clear when we inspect the distribution of the rating values for our dataset. Fig. 8 shows how this distribution is slightly skewed towards the higher values of the rating scale and therefore decreasing the final recommendation score will (on average) worsen performance more than increasing the score.

To continue the evaluation of the weighted hybrid strategy discussed in this paper, we take a closer look at the weight vectors produced by our optimization procedure. Specifically, we are interested in how the algorithms (on average) contribute to the final prediction scores, what algorithms are used, and how many algorithms (i.e., algorithms with non-zero weights) are usually combined. We focus on the weights generated by the *W7* hybrid configuration, which showed the best performance.

Inspecting the complete set of weights produced by our optimization procedure for all users, we logged for each algorithm the number of times a non-zero weight was generated. Fig. 9 shows a pie chart detailing the normalized (percentage) counts for the individual recommendation algorithms.

The figure shows an approximately even distribution of the values, which indicates that the system involves every algorithm about the same number of times in the final prediction score. It seems that the weighted hybrid strategy is able to make use of all given algorithms, without degrading the performance when variations of the same algorithm are present (as was the case for the hybrid switching strategy).

Aside from how much the algorithms are used, it is also interesting to know how many algorithms usually contribute to the final prediction score for a user. Again inspecting the complete set of weight vectors produced for all users, we

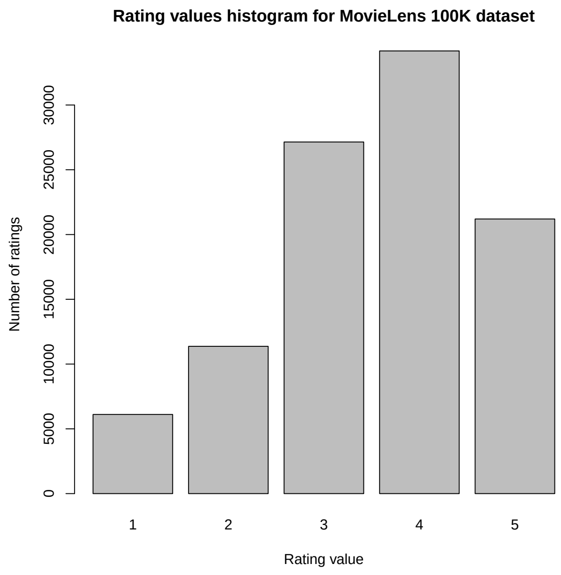

Fig. 8 The distribution of the ratings for the MovieLens 100K which shows a slight skew towards higher rating values.

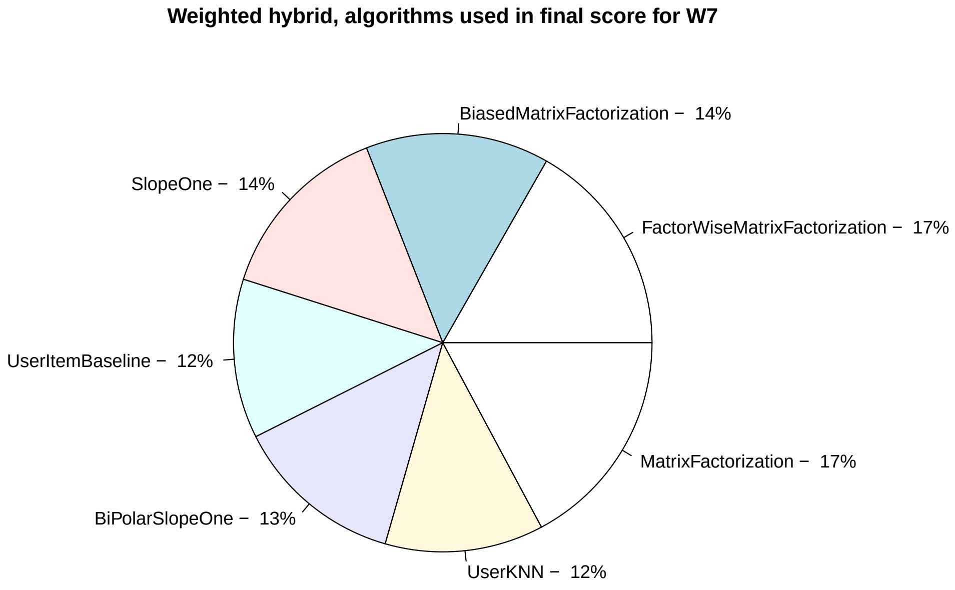

Fig. 9 The distribution of the usage of each algorithm in the weight vectors for all users. The results are normalized in percentage.

logged the amount of non-zero weights in each weight vector. Fig. 10 shows the resulting histogram for the *W7* hybrid configuration.

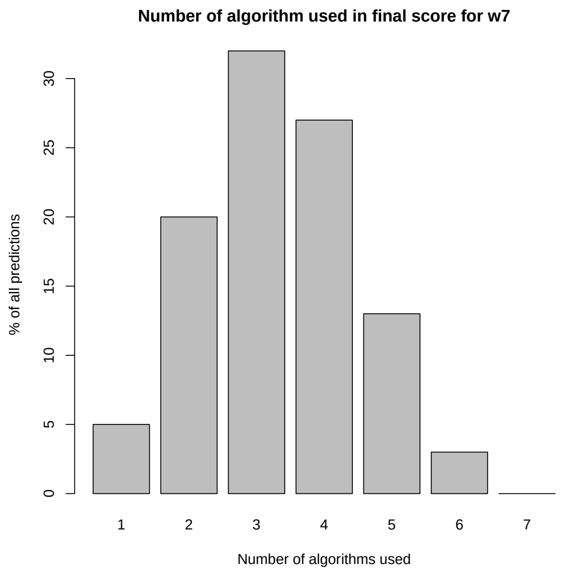

Fig. 10 The histogram of the number of non-zero algorithms used for all weight vectors for all users (normalized in percentage).

From the histogram we learn that the weighted hybrid strategy is usually combining multiple algorithms together. For some users, the results of as many as 6 algorithms are combined into the final recommendation score. For others, on the other hand, only a single algorithm is used. It is interesting to see that the weighted hybrid strategy does indeed revert to a hybrid switching strategy when it seems appropriate.

A final interesting aspect of the generated weight values, is the value itself. Table 4 displays for every algorithm (used in *W7*) the average weight value over all weight vectors for all users without counting the zero weights.

Algorithm	Average Weight
UserKNN	0.426
UserItemBaseline	0.421
SlopeOne	0.389
BiasedMatrixFactorization	0.282
BiPolarSlopeOne	0.257
MatrixFactorization	0.249
FactorWiseMatrixFactorization	0.201

Table 4 The average weight values (in range [0,1]) over all users for every algorithm used by the weighted hybrid configuration $W7$.

The order of the averaged out weights for the algorithms, matches the order of the performance results for the individual algorithms. This confirms that the results of our offline optimization strategy do in fact correlate with the final results as obtained by the test set.

In conclusion, Fig. 11 overviews the final results of our hybrid strategy evaluation. The three result values indicate the results as obtained by the three respective recommendation strategies: best individual algorithm (same for all users), user-specific best switching, and user-specific weighted hybrid. For each of these systems the best results are shown in the graph i.e., *UserKNN* (best individual), *BS4* (best switched), and *W7* (best weighted). The differences between the results (all of which are found to be statistical significant $p < 0.05$) confirm the original hypothesis that a user-specific hybrid switching strategy will yield better results than an individual algorithm and a weighted hybrid system will outperform even the hybrid switching strategy.

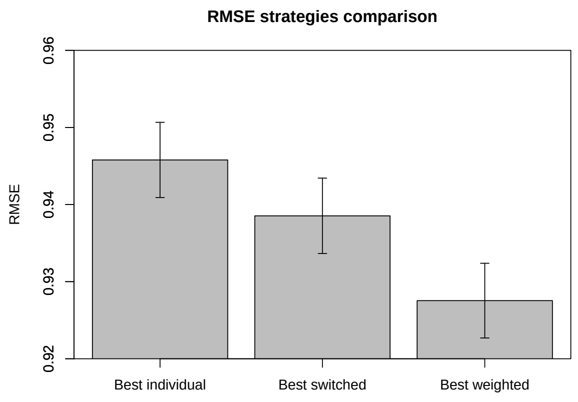

Fig. 11 A comparison of the best RMSE results obtained by the three systems compared in our evaluation: individual algorithms, a switching approach and a weighted hybrid approach. All of the differences were found to be statistical significant.

7.4 Computational performance and user requirements

Although all of the experiments described in this work were carried out by a supercomputer infrastructure, the presented hybrid recommender could run on commodity computers as well. The main computational burden lies with the training of the individual recommendation algorithms. Once they are trained, the hybrid optimization procedure can optimize the specific weights for a user in a matter of seconds. Training the individual algorithms can be done in the background, or even repeated in timed intervals while the hybrid recommender is capable of delivering realtime responses to user feedback.

Before starting the optimization process for a user, the available data of the user must be carefully considered. If a user has provided only a few ratings, then the evaluation procedure might optimize too hard on too few ratings which will badly influence the perceived recommendation quality. Ratings are split with a 60/40 ratio in subtraining and subtest sets, so a total of 20 ratings (i.e., 8 in the subtest sets) for a user should be considered a minimum. Even the individual recommendation algorithms would fail at recommending interesting items for users with less than 20 ratings. In that case, a default set of weights for the hybrid recommender could be used, or the users could be served non-personalized recommendation lists like 'popular items'.

8 Discussion

This work aims to be a step in the direction of modern hybrid recommender systems that automatically fine-tune themselves to fit users on a individual basis. In this work, we have provided a way in which two hybrid systems (best switching and weighted average) can be user-specifically optimized in an offline setting. Although the evaluation of our methods focused on the popular accuracy metric RMSE, other offline calculable metrics can be optimized for. The evaluation intended to show the success of the optimization procedure, and the specific advantages of the weighted hybrid procedure over a hybrid switching approach. Implementing the weighted average strategy in a hybrid recommender allows for easy adding new algorithms or variations of existing algorithms without fundamentally disrupting user experience. The quality of offline optimization depends however on the quality of the available data and will therefore always be fundamentally limited in potential. In future work, we intend to expand our optimization to incorporate online user interactions and thereby unlock the true potential of dynamically adapting hybrid recommender systems.

9 Conclusion

We started this work by noting the existence of vast numbers of recommendation algorithms available to tackle the information overload problem. Combining multiple algorithms together, seemed a sensible approach to harvest the union of their merits. However, combining algorithms into hybrid recommender systems can be cumbersome, often manual configuration is required such that the recommender can not be easily re-used for other scenarios.

In this work, we considered the configuration of a hybrid system as an optimization problem which generates hybrid recommender systems that are automatically fine-tuned towards individual users. Focus was on the commonly used *switching* and *weighted* hybridization techniques. We demonstrated an evaluation approach that allowed the hybrid recommender system to optimize recommendations offline, while allowing realtime integration of new user feedback which can be very useful for online recommendation scenarios. Results showed that the *switching* strategy was highly sensitive to the used individual algorithms i.e., best results when most different algorithms are used. The *weighted* strategy, on the other hand, was more robust and, even with the simple binary search optimization procedure, it obtained significantly better results by blending the individual algorithms into complex and improved user-specific ensembles.

Acknowledgements The described research activities were funded by a PhD grant to Simon Dooms of the Agency for Innovation by Science and Technology (IWT Vlaanderen). This work was carried out using the Stevin Supercomputer Infrastructure at Ghent University, funded by Ghent University, the Hercules Foundation and the Flemish Government - department EWI.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on **17**(6), 734–749 (2005)
2. Adomavicius, G., Tuzhilin, A.: Context-aware recommender systems. In: Recommender systems handbook, pp. 217–253. Springer (2011)
3. Aksel, F., Birturk, A.: An adaptive hybrid recommender system that learns domain dynamics. In: Int. workshop handling concept drift in adaptive information systems: importance, challenges and solutions (HaCDAIS-2010) at the european conf. machine learning and principles and practice of knowledge discovery in databases, p. 49 (2010)
4. Balabanović, M., Shoham, Y.: Fab: content-based, collaborative recommendation. Communications of the ACM **40**(3), 66–72 (1997)
5. Bao, X., Bergman, L., Thompson, R.: Stacking recommendation engines with additional meta-features. In: Proc. 3rd ACM conf. recommender systems, pp. 109–116. ACM (2009)
6. Bellogín, A.: Predicting performance in recommender systems. In: Proc. 5th ACM conf. Recommender systems, pp. 371–374. ACM (2011)
7. Bellogín, A.: Performance prediction and evaluation in recommender systems: an information retrieval perspective. Ph.D. thesis, Universidad Autonoma de Madrid (2012)
8. Bobadilla, J., Serradilla, F., Bernal, J.: A new collaborative filtering metric that improves the behavior of recommender systems. Knowledge-Based Systems **23**(6), 520–528 (2010)
9. Burke, R.: Hybrid recommender systems: survey and experiments. User modeling and user-adapted interaction **12**(4), 331–370 (2002)
10. Dooms, S., De Pessemier, T., Martens, L.: A user-centric evaluation of recommender algorithms for an event recommendation system. In: Workshop on human decision making in recommender systems (Decisions@RecSys'11) and user-centric evaluation of recommender systems and their interfaces - 2 (UCERSTI 2) affiliated with 5th ACM conf. recommender systems, pp. 67–73 (2011)
11. Ekstrand, M., Riedl, J.: When recommenders fail: predicting recommender failure for algorithm selection and combination. In: Proc. 6th ACM conf. recommender systems, pp. 233–236. ACM (2012)
12. Ekstrand, M.D., Ludwig, M., Konstan, J.A., Riedl, J.T.: Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In: Proc. 5th ACM conf. recommender systems, pp. 133–140. ACM (2011)
13. Gantner, Z., Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: MyMediaLite: A free recommender system library. In: Proc. 5th ACM conf. recommender systems (2011)
14. Han, E.H.S., Karypis, G.: Feature-based recommendation system. In: Proc. 14th ACM int. conf. information and knowledge management, pp. 446–452. ACM (2005)
15. Hussein, T., Linder, T., Gaulke, W., Ziegler, J.: Hybreed: A software framework for developing context-aware hybrid recommender systems. User Modeling and User-Adapted Interaction pp. 1–54 (2012)
16. Kille, B., Albayrak, S.: Modeling difficulty in recommender systems. In: Workshop on recommendation utility evaluation: beyond RMSE (RUE 2011), p. 30 (2012)
17. Knijnenburg, B.P., Willemsen, M.C., Gantner, Z., Soncu, H., Newell, C.: Explaining the user experience of recommender systems. User Modeling and User-Adapted Interaction **22**(4-5), 441–504 (2012)
18. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proc. 14th ACM int. conf. knowledge discovery and data mining (SIGKDD), pp. 426–434. ACM (2008)
19. Koren, Y.: The bellkor solution to the netflix grand prize. Netflix prize documentation (2009)
20. Lemire, D., Maclachlan, A.: Slope one predictors for online rating-based collaborative filtering. Society for Industrial Mathematics **5**, 471–480 (2005)
21. Lommatzsch, A., Kille, B., Kim, J.W., Albayrak, S.: An adaptive hybrid movie recommender based on semantic data. In: Proc. 10th conf. open research areas in information retrieval, pp. 217–218. centre de hautes etudes internationales d'informatique documentaire (2013)
22. McNee, S.M., Riedl, J., Konstan, J.A.: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: extended abstracts on Human factors in computing systems (CHI), pp. 1097–1101. ACM (2006)

23. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review **13**(5-6), 393–408 (1999)
24. Peralta, V.: Extraction and integration of movielens and imdb data. Tech. rep., Technical Report, Laboratoire PRiSM, Université de Versailles, France (2007)
25. Piotte, M., Chabbert, M.: The pragmatic theory solution to the netflix grand prize. Netflix prize documentation (2009)
26. Pu, P., Chen, L., Hu, R.: A user-centric evaluation framework for recommender systems. In: Proc. 5th ACM conf. recommender systems, pp. 157–164. ACM (2011)
27. Salehi, M., Pourzaferani, M., Razavi, S.A.: Hybrid attribute-based recommender system for learning material using genetic algorithm and a multidimensional information model. Egyptian Informatics Journal (2013)
28. Shapira, B.: Recommender systems handbook. Springer (2011)
29. Sill, J., Takács, G., Mackey, L., Lin, D.: Feature-weighted linear stacking. arXiv preprint arXiv:0911.0460 (2009)
30. Song, Y., Zhang, L., Giles, C.L.: Automatic tag recommendation algorithms for social recommender systems. ACM Transactions on the Web (TWEB) **5**(1), 4 (2011)
31. Töscher, A., Jahrer, M., Bell, R.M.: The bigchaos solution to the netflix grand prize. Netflix prize documentation (2009)
32. Wolpert, D.H.: Stacked generalization. Neural networks **5**(2), 241–259 (1992)