# Exploiting Performance Estimates for Augmenting Recommendation Ensembles

Gustavo Penha
Delft University of Technology
Delft, Netherlands
g.penha-1@tudelft.nl

Rodrygo L. T. Santos
CS Dept., UFMG
Belo Horizonte, MG, Brazil
rodrygo@dcc.ufmg.br

## ABSTRACT

Ensembling multiple recommender systems via stacking has shown to be effective at improving collaborative recommendation. Recent work extends stacking to use additional user performance predictors (e.g., the total number of ratings made by the user) to help determine how much each base recommender should contribute to the ensemble. Nonetheless, despite the cost of handcrafting discriminative predictors, which typically requires deep knowledge of the strengths and weaknesses of each recommender in the ensemble, only minor improvements have been observed. To overcome this limitation, instead of engineering complex features to predict the performance of different recommenders for a given user, we propose to directly estimate these performances by leveraging the user's own historical ratings. Experiments on real-world datasets from multiple domains demonstrate that using performance estimates as additional features can significantly improve the accuracy of state-of-the-art ensemblers, achieving nDCG@20 improvements by an average of 23% over not using them.

## CCS CONCEPTS

• **Information systems** → **Collaborative filtering**; **Learning to rank**; • **Computing methodologies** → **Ensemble methods**.

## KEYWORDS

Recommender Systems, Performance Prediction, Performance Estimation, Ensembling, Learning to Rank

**ACM Reference Format:**
Gustavo Penha and Rodrygo L. T. Santos. 2020. Exploiting Performance Estimates for Augmenting Recommendation Ensembles. In *Fourteenth ACM Conference on Recommender Systems (RecSys '20), September 22–26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3383313.3412264

## 1 INTRODUCTION

Recommender systems aim to suggest items (e.g., movies, books) to users to assist in their decision-making process. When faced with the huge amount of available options, combined with a possible lack of experience or knowledge from the user, recommender systems become extremely useful. With the steady interest in the subject from both academia and industry throughout the years, several recommendation approaches have been proposed, each with different strengths and weaknesses. For instance, collaborative recommenders typically excel in data-rich scenarios, while content-based and knowledge-based recommenders are often preferred in item and user cold-start scenarios, respectively [2].

Hybrid recommenders are designed to leverage the power of different base recommenders in order to make more robust recommendations [1]. Ensembling, a particular hybridization technique, is commonly used in machine learning tasks such as classification to enhance generalization by combining various hypotheses learned by different base models. This technique has led to state-of-the-art machine learning models such as extreme gradient boosting [12]. Figure 1 illustrates an ensemble of recommender systems. In the figure, given a user $u$, we want to estimate the relevance of unseen items (white circles labeled $a$, $b$, $c$, $\cdots$). A standard recommendation ensembler would take the scores produced by $k$ base recommenders $RS_1$, $\cdots$, $RS_k$ as input (gray circles labeled $\hat{r}_{ui}^{(1)}$, $\cdots$, $\hat{r}_{ui}^{(k)}$ for each item $i \in \{a, b, c, \cdots\}$).

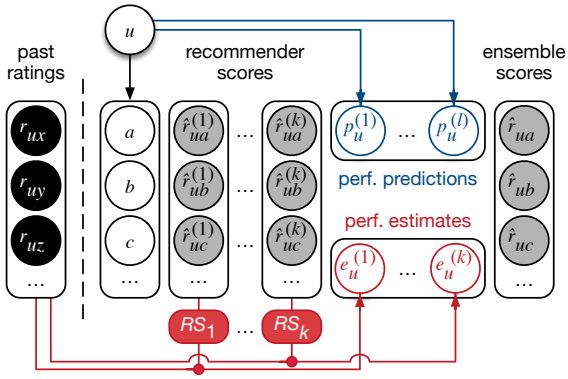


**Figure 1: Ensemble augmentation overview. At recommendation time, candidate items (white circles) are scored for a given user by multiple base recommenders (gray circles on the left), which are to be ensembled into a final score (gray circles on the right). Current approaches augment the ensemble input space with performance predictions (blue outlined circles) based on characteristics of the target user, regardless of the ensembled recommenders. We propose to leverage past historical ratings by the user (black circles on the left) to augment the ensemble with performance estimates (red outlined circles), one per base recommender.**


*RecSys '20, September 22–26, 2020, Virtual Event, Brazil*

ACM ISBN 978-1-4503-7583-2/20/09…$15.00
https://doi.org/10.1145/3383313.3412264

In addition to base recommender scores, recommendation ensembling has been shown to benefit from performance predictors [4, 6], i.e., features that are indicative of the performance of each base recommender for the target user (blue outlined circles in Figure 1, labeled $p_u^{(1)}, \cdots, p_u^{(l)}$, for a total of $l$ predictors). For instance, Bellogín et al. [6] adapted the well-known clarity score [14], originally formulated as a measure of query ambiguity in adhoc search, to measure the coherence of the user's historical ratings in a recommendation setting. Despite their promise, handcrafting performance predictors to improve ensembling requires a deep understanding of the (often many) recommenders to be combined, as well as intuition about how each predictor relates to the performance of each recommender in the ensemble. Moreover, performance predictors have been shown to be better indicators of the inherent difficulty of a user (regardless of any particular recommender) than of the relative effectiveness of different base recommenders [31].

To ease the burden of handcrafting predictors and at the same time improve their prediction accuracy, we introduce a novel class of predictors called *performance estimates*. In particular, unlike adhoc search, where performance must be *predicted* given the lack of user supervision, collaborative recommendation offers an inexpensive alternative for directly *estimating* the performance of different recommenders, namely, the user's own historical feedback. As also illustrated in Figure 1, given a user $u$ and $k$ base recommenders $RS_1, \cdots, RS_k$, we compute $k$ performance estimates (red outlined circles in Figure 1, labeled $e_u^{(1)}, \cdots, e_u^{(k)}$), one per base recommender, as the outcome of a standard evaluation metric (e.g., root mean squared error, normalized discounted cumulative gain) by assessing the scores produced by each recommender (gray circles) against the available user feedback (black circles). In contrast to performance predictors, performance estimates can be readily computed for any number of base recommenders in the ensemble while requiring no deep understanding of each individual recommender nor of when they are expected to outperform one another.

Comprehensive experiments using real-world datasets of different domains demonstrate the effectiveness of the proposed performance estimates when combined with plain recommender systems scores for improving current pointwise ensemblers from the literature. Moreover, we show that performance estimates are robust to the choice of ensembler, achieving state-of-the-art recommendation accuracy when leveraged by pairwise and listwise ensemblers.

The main contributions of this paper are twofold:

(1) We propose performance estimates as a highly discriminative and inexpensive set of features, which significantly improve the performance of recommendation ensembles, while mitigating the need for feature engineering.
(2) We demonstrate the effectiveness of the proposed features in different recommendation domains, as well as the robustness of these features to the choice of ensembler.

## 2 RELATED WORK

Below we provide an overview of ensemble methods, followed by a discussion on performance prediction and on related attempts to exploit these features in recommender systems hybridization.

### 2.1 Ensemble Methods

Ensembling is the field of machine learning concerned with the combination of several base learners to improve their generalization and robustness compared to using only one learner. Bootstrap aggregating (bagging) [8] and boosting [21] are ensembles that combine base models from the same hypothesis space. Bagging modifies the input data for each learner, using bootstrap samples, and then takes the average of the various models for each new sample. Boosting, on the other hand, incrementally constructs models by focusing more on training examples where previously learned models have failed, combining them using a closed formula which takes into account the error of each base learner being combined.

While such methods have been explored in the recommender systems literature with some success [5], here we focus on the class of ensembles that are able to combine models based on different hypothesis spaces, namely, stacking [9]. This technique has been extensively used in the machine learning community. The method is based on the combination of different base models by training a final model, also known as meta-learner or second-level model, which makes new predictions based on the predictions of the base models. This idea can be seamlessly adapted to combine multiple recommender systems, and has been successfully used in the field of recommender systems for producing hybrid recommendations [1, 10]. For instance, both the winner and second place of the Netflix competition [22, 32] employed a stack of recommenders.

Strategies to combine different rankings have also been proposed. Strategies that do not require training a model generally fall into the realm of rank aggregation [3], which has been applied to combine the output of multiple recommenders [33]. The other research strand for the combination of multiple rankings, which requires training data, is known as learning to rank [25]. Existing approaches can be broadly categorized as pointwise, pairwise and listwise, according to their choice of input and output representation and their underlying model structure [25]. While ensemblers based on pointwise learning have been used to leverage performance predictors before [4], to the best of our knowledge, no previous work attempted to augment pairwise and listwise ensemblers.

### 2.2 Performance Prediction

The study of features that can indicate the extent to which a given system will perform effectively for a particular instance (in our case, a user) is called performance prediction. Performance predictors can enable decisions such as whether to deliver system outputs or how to combine multiple systems accordingly. This is an established research area in information retrieval [14], where the performance of a search system in response to a specific query is predicted. Such features have been classified into pre-retrieval and post-retrieval predictors, according to the available data for the prediction [11, 19].

Performance prediction for recommendation was first explored by Bellogín et al. [6]. In particular, they proposed multiple adaptations of the query clarity score predictor, originally proposed by Cronen-Townsend et al. [14], to predict the performance of recommender systems for different users, reaching a maximum correlation of 0.5 with the actual nDCG attained by these systems. In a similar vein, Gras et al. [17] explored predictors such as AbnormalityCR to identify users who are outliers and consequently get poor

recommendations, reaching correlations of up to 0.55 with RMSE. Griffith et al. [18] proposed a decision tree-based strategy using multiple predictors and attained a high correlation of 0.8 between the actual and predicted performance of multiple recommenders.

High prediction accuracy does not automatically translate to high ranking effectiveness. Indeed, in a recent study, Raiber and Kurland [31] questioned the usefulness of performance prediction as a mechanism for improving ranking effectiveness for adhoc search. Their study demonstrated that, in the absence of relevance feedback, performance prediction equates to the more fundamental problem of relevance estimation, which explains the difficulty of the task. In contrast to adhoc search, personalized recommendation enables the exploration of the target user's historical feedback. As a result, rather than predict, we propose to directly estimate the performance of different recommenders, under the assumption that past performance is indicative of future performance.

## 2.3 Prediction-enhanced Ensembling

Augmenting ensembles with performance predictors has been explored in the recommendation literature. Bao et al. [4] proposed the STREAM (Stacking Recommendation Engines with Additional Meta-Features) framework for exploiting such features for stacking recommenders. The ensemble leverages additional features by making the input space the concatenation of performance predictors and the scores of base recommenders, as illustrated in Figure 1.

Intuitively, STREAM learns which performance predictors are adequate for each base recommender. This framework was further improved by Jahrer et al. [20], who leveraged pointwise ensemblers such as neural networks and bagged gradient boosted decision trees. To release the ensembler from the burden of learning feature associations, Sill et al. [32] introduced FWLS (Feature-Weighted Linear Stacking), which augments recommender scores by computing their Cartesian product with respect to multiple predictors. However, the input space of FWLS grows proportionally to $kl$, where $k$ is the number of base recommenders and $l$ is the number of performance predictors. Fortes et al. [15] further demonstrated the benefits of ensembling augmentation strategies such as FWLS and STREAM compared to non-augmented stacking. However, their results did not show significant differences between FWLS and STREAM.

Our approach differs from STREAM and FWLS in two fundamental ways. First, instead of traditional performance predictors, we augment a recommendation ensemble with performance estimates, which more accurately describe the performance of each base recommender. Second, as illustrated in Figure 1, because each performance estimate is tied to a single base recommender, we produce an arguably more discriminative input space. Indeed, as shown in Section 5, our approach significantly improves upon pointwise ensemblers augmented with performance predictors, such as STREAM and FWLS. Moreover, our approach is robust to the choice of ensembler, attaining state-of-the-art recommendation effectiveness with pairwise and listwise ensemblers.

# 3 PERFORMANCE ESTIMATES

The idea of using performance predictors for improving ensembles is fairly intuitive. For instance, consider a certain collaborative recommender $RS_1$ which performs best when users have rated plenty of items, and a content-based recommender $RS_2$ which can handle better users with a small amount of ratings. A performance predictor quantifying the number of historical ratings of the target user can give a higher weight to $RS_1$ and demote the contribution of $RS_2$ if the user has a prolific history. While intuitive, this approach has two key shortcomings. First, effective performance prediction is a difficult task and better suited for predicting the inherent difficulty of different users for a fixed recommender rather than to predict the performance of different recommenders for a fixed user. Indeed, Raiber and Kurland [31] demonstrated that accurate performance prediction boils down to accurate relevance estimation. Second, even if perfect performance prediction could be achieved, new predictors must be engineered every time a new recommender is added to the ensemble, which is in itself a difficult task. In the following, we formalize our proposed solution to address both of these shortcomings by directly *estimating* (as opposed to *predicting*) the performance of different recommenders in the ensemble.

## 3.1 Estimating Performance

Let $\mathcal{U}$, $\mathcal{I}$, $\mathcal{R}$, and $\mathcal{T}$ denote a set of users, items, possible rating values (e.g. 1-5 stars), and discrete rating timestamps, respectively. Moreover, let $\mathcal{D}_{\kappa_1}^{\kappa_2} = \{(u, i, r, t) \mid u \in \mathcal{U},\ i \in \mathcal{I},\ r \in \mathcal{R},\ t \in \mathcal{T},\ \kappa_1 < t \leq \kappa_2\}$ be the set of ratings recorded in the left-open time interval bounded by timestamps $\kappa_1$ and $\kappa_2$. A recommender system can be defined as a function $s(u, i) : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{R}$. For a user $u$, this function produces a recommendation list $\mathcal{L}_u^s = \text{sort}_{s(u,i)}\{i \in \mathcal{I}\}$ as a permutation of all available items $\mathcal{I}$. Given a recommendation list $\mathcal{L}_u^s$ produced by recommender $s$ at time $\tau$ and assessed by user $u$ within some (typically short) time interval $\delta$, the true performance $m_u$ of the system can be measured by:

$$m_u = \Delta(u, \mathcal{L}_u^s, \mathcal{D}_\tau^{\tau+\delta}), \tag{1}$$

where $\Delta$ could be any evaluation metric, including business metrics such as number of clicks or purchases, error metrics such as root mean squared error (RMSE) or ranking-based metrics such as normalized discounted cumulative gain (nDCG).

In reality, for a recommendation list $\mathcal{L}_u^s$ displayed at time $\tau$, no user feedback will be available until after this time. Hence, at time $\tau$, system performance can only be approximated. A performance predictor computes such an approximation based on characteristics of the target user $u$ (for pre-retrieval predictors), or of the produced recommendation list $\mathcal{L}_u^s$ (for post-retrieval predictors). Without loss of generality, a performance predictor $p_u$ is computed as:

$$p_u = \Pi(u, \mathcal{L}_u^s), \tag{2}$$

where $\Pi$ could compute, e.g., the amount of ratings in the historical profile of user $u$ [32] or its deviation from a random or most-popular list of items [6]. Given the engineering effort spent in producing discriminative performance predictors for different recommenders and these predictors' inherently limited accuracy, we instead propose a simple yet effective alternative. In particular, we compute a performance estimate $e_u$ according to:

$$e_u = \Delta(u, \mathcal{L}_u^s, \mathcal{D}_{\kappa_1}^{\kappa_2}), \tag{3}$$

where the set of ratings used for the performance estimation are a subset of $\mathcal{D}$ limited by two timestamps $\kappa_1$ and $\kappa_2$ before the recommendation time $\tau$, $0 \leq \kappa_1 < \kappa_2 \leq \tau$ and, similarly to Equation (1),

$\Delta$ could be any evaluation metric. The key insight here is that the true performance of recommender $s$ can be directly approximated by its *past* performance, by leveraging user $u$'s historical feedback.

Because each base recommender in the ensemble is also trained using the target user's historical feedback, its performance estimate computed on the same data may be overly optimistic. For this reason, we instead compute performance estimates using validation data set aside from the base recommender training, in the hope of improving their generalizability to unseen test data. In this case, base recommender training uses $\mathcal{D}_{\kappa_1}^{\kappa_2}$, whereas performance estimates use $\mathcal{D}_{\kappa_3}^{\kappa_4}$, with $0 \leq \kappa_1 < \kappa_2 \leq \kappa_3 < \kappa_4 \leq \tau$.

## 3.2 Leveraging Performance Estimates

To leverage our introduced performance estimates, we propose to tackle recommendation ensembling as a learning to rank task. As illustrated in Figure 1, for each user–item pair $\langle u, i \rangle$, we are given the scores of $k$ base recommenders (*RS*) previously trained on historical ratings $\mathcal{D}_{\kappa_1}^{\kappa_2}$, with $0 \leq \kappa_1 < \kappa_2 \leq \tau$, where $\tau$ once again denotes the recommendation time. In addition, we are also given $l$ performance predictors (here collectively referred to as *PP*) and $k$ performance estimates (*PE*) as features for the ensemble.

Our goal is to learn a hypothesis function $h : \mathcal{X} \rightarrow \mathcal{Y}$ mapping the input space $\mathcal{X}$ onto the output space $\mathcal{Y}$. Our input space $\mathcal{X}$ comprises learning instances of the form $\vec{x} = \Phi(u, i)$, where $\Phi$ is a feature extractor defined over the user-item pair $\langle u, i \rangle$. In practice, we could represent each learning instance $\vec{x}$ as a $(k+l+k)$-dimensional vector, such that $\vec{x} = (\{\hat{r}_{ui}^{(j)}\}_{j=1}^{k}, \{p_u^{(j)}\}_{j=1}^{l}, \{e_u^{(j)}\}_{j=1}^{k})$, where $\hat{r}_{ui}^{(j)}$, $p_u^{(j)}$, and $e_u^{(j)}$ denote the $j$-th recommender score, performance predictor, and performance estimate, respectively. In turn, our output space $\mathcal{Y}$, in its most basic form, equates to the set of possible rating values $\mathcal{R}$. In Section 5, we experiment with representative learning to rank approaches from the pointwise, pairwise, and listwise families,[1] encompassing both linear and non-linear hypotheses, using different input spaces in order to compare our approach with alternative ensemble augmentation strategies.

We propose two variants for leveraging performance estimates. Our first variant corresponds to the raw performance estimates defined in Equation (3). Because such performance estimates are defined at the user level, all learning instances $\vec{x} = \Phi(u, i)$ associated with a given user $u$ have the same performance estimates for all items $i \in \mathcal{I}$. Such user-dependent, item-agnostic features bear resemblance to query-dependent, document-agnostic features, which have been shown to be useful for learning non-linear hypotheses in adhoc search, such as boosted regression trees [30]. Nevertheless, to provide alternative, item-dependent performance estimates, we consider a weighted variant of the raw $e_u$, by multiplying them by the corresponding recommender score $\hat{r}_{ui}$:

$$e_{ui} = e_u \times \hat{r}_{ui}, \tag{4}$$

where $e_u$ (a function of user $u$ and recommender $s$) is given by Equation (3) and $\hat{r}_{ui}$ denotes the score produced by recommender $s$ for the $\langle u, i \rangle$ pair. If the unweighted variant from Equation (3) is used, the ensemble has to learn the relation between each performance estimate and the score of each base recommender. In contrast, the weighted variant in Equation (4) further increases the sensitivity of our approach by automatically boosting scores produced by recommenders with a high performance estimate. In the following sections, we assess the effectiveness of all variants of our proposed performance estimates for augmenting recommendation ensembles.

[1] For pairwise and listwise learners, the input and output spaces are suitably redefined to consider instance pairs or instance lists, respectively.

# 4 EXPERIMENTAL SETUP

In this section, we detail the setup that supports our investigations in Section 5. We aim to answer the following research questions:

- *Q1.* How effective are *PE* for ensembling recommenders?
- *Q2.* How robust are *PE* to the choice of ensembler?
- *Q3.* How do error and ranking-based metrics compare for *PE*?

## 4.1 Datasets

We use two publicly available datasets covering three different domains: Yelp,[2] for point-of-interest recommendation, and Amazon,[3] for book and electronics recommendation. These datasets provide large-scale timestamped rating data, which allows for a consistent evaluation by respecting the chronology of the recorded user interactions.[4] In particular, we divide each dataset into base recommenders training (30% earliest ratings) and validation (next 30%), ensembler training (next 30%) and test (last 10%), as described in Figure 2. Note that we separate training of base recommenders and ensemblers to make sure ensemblers will not leverage the output of overfitted recommenders. Likewise, we set aside validation data to estimate the performance of a recommender on data points other than those used for training it. Also note that data partitioning is performed globally for each dataset and not at the user level, so as to simulate a more realistic setting with users with different amounts of training and test. After this process, to enable a consistent evaluation of performance estimates, we retain only users who have ratings in all four partitions.[5]

The statistics of the resulting datasets after preprocessing are described in Table 1.

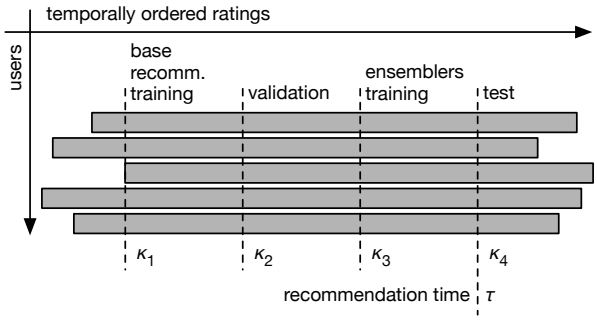


**Figure 2: Data splitting performed in our experiments. Split points $\kappa_1$, $\kappa_2$, $\kappa_3$, and $\kappa_4$ are global across users.**

[2] https://www.yelp.com/dataset/challenge
[3] http://jmcauley.ucsd.edu/data/amazon/
[4] This design prevents future ratings by a user from leaking into her training data.
[5] Cold-start users are out of the scope of this investigation as we focus on personalized recommendation scenarios. Under a cold-start scenario, estimates could be calculated from users with similar demographics for example. We leave this study as future work.

To compute performance estimates, we choose RMSE and nDCG as representative of error and ranking-based metrics, respectively.[6] Our default setup computes *PE* using RMSE on the validation data. Investigations of the impact of the evaluation metric (RMSE vs. nDCG) used for estimation are discussed in Section 5.3.

**Table 1: Statistics of the datasets in our evaluation after filtering users without ratings in all four partitions.**

| Dataset | # users | # items | # ratings | density |
|---|---|---|---|---|
| Yelp | 4,014 | 6,989 | 229,809 | 0.8 % |
| Amazon Books | 44,136 | 673,601 | 2,156,842 | 0.00007 % |
| Amazon Electronics | 16,814 | 86,822 | 337,158 | 0.0002 % |

## 4.2 Performance Predictors

For comparison with our proposed performance estimates, we implemented a total of 14 performance predictors from the literature, as summarized in Table 2. They measure aspects of the user rating behavior that might indicate the performance of a recommender system. For instance, a high rarity of rated items (low $PP_{10}$) might indicate that the user has a unique taste and might be difficult for the recommender system to provide effective predictions. The predictors can be calculated on the user dimension, e.g. *Abnormality* ($PP_6$), or the item, e.g. *Item support* ($PP_{12}$). In this paper, we resort only to features based on the rating matrix $\mathcal{D}^{\kappa_2}_{\kappa_1}$, but other sources could also be used, such as contextual data (e.g., rating location) and content information (e.g., item category).

**Table 2: Performance predictors used in our experiments.**

| *PP* | Description | Ref. |
|---|---|---|
| $PP_1$ | The log of the number of distinct ratings dates | [32] |
| $PP_2$ | The log of the number of user ratings | [32] |
| $PP_3$ | The standard deviation of the user ratings | [32] |
| $PP_4$ | Regularized mean support for the user items | [32] |
| $PP_5$ | User support: number of ratings | [20] |
| $PP_6$ | Abnormality | [17] |
| $PP_7$ | AbnormalityCR | [17] |
| $PP_8$ | User average rating value | [18] |
| $PP_9$ | User standard deviation of rating values | [18] |
| $PP_{10}$ | Average number of ratings for the user items | [18] |
| $PP_{11}$ | Average of ratings from items rated by the user | [18] |
| $PP_{12}$ | Item support: number of ratings | [18] |
| $PP_{13}$ | Average rating value of item | [20] |
| $PP_{14}$ | Item standard deviation of rating values | [20] |

## 4.3 Base Recommenders

To test our approach, we produce ensembles of eight classical collaborative recommenders, organized into three broad classes:

**Simple models.** *NormalPredictor* assumes the prediction is generated by a normal distribution, and it estimates its parameters using maximum likelihood estimation. *DebiasedAverage* predictions are given solely by the overall ratings mean and the user and item deviations from this overall average.

**Neighborhood models.** *KNNBaseline* is a classical item-based nearest-neighbor recommender [23, Equation (3)]. A variant that takes into account the mean rating of each user (*KNNWithMeans*) and a variant that does not (*KNNBasic*) are also used. *CoClustering* [16] also uses similarity measurements between users and between items.

**Latent factor models.** *SVD* denotes the matrix factorization algorithm described by Koren et al. [24], which is closely related to singular value decomposition, hence the name. *NMF* stands for non-negative matrix factorization [28], which is similar to SVD with non-negative user and item factors.

For these base recommenders, we use the implementations provided in the Surprise v1.0.5 package.[7] We performed a randomized search by sampling five times within the domain defined for each hyperparameter [7]. We keep in the ensemble all five generated models. This way, along with the hyperparameter-free recommenders *NormalPredictor* and *DebiasedAverage*, we produced a total of $6 \times 5 + 2 = 32$ base recommenders for ensembling.

## 4.4 Ensemblers and Input Spaces

For the learning to rank methods for ensembling, we used the following pointwise regressors implemented in Scikit-learn v0.19.1:[8] gradient boosting, random forest, support vector machines (SVM) and neural network. For pairwise and listwise models, we used the following implementations from RankLib v2.1-patched:[9] LambdaMART, ListNet, AdaRank and RankBoost. For each ensemble, we selected the best configuration of hyperparameters by performing a grid search through a 5-fold cross-validation on the partition for ensemble training in each dataset. This process was performed for each dataset, ensemble and set of features used. The full hyperparameters configuration used for all ensembles in this paper as well as their implementations are publicly available.[10]

To assess the effectiveness of performance estimates for ensemble augmentation, we contrast our proposed input space to two representative baselines from the literature. Our first baseline input space is composed solely by the predicted ratings from the $k$ base recommenders in the ensemble, i.e., $\vec{x} = (\{\hat{r}_{ui}^{(j)}\}_{j=1}^{k})$, which is equivalent to non-augmented stacking. Our second baseline input space augments the $k$ base recommenders with $l$ performance predictors, i.e., $\vec{x} = (\{\hat{r}_{ui}^{(j)}\}_{j=1}^{k}, \{p_u^{(j)}\}_{j=1}^{l})$, which is equivalent to STREAM [4], a state-of-the-art ensemble augmentation approach described in Section 2. We contrast these two baselines with ensembles augmented with $k$ performance estimates, one per base recommender, such that $\vec{x} = (\{\hat{r}_{ui}^{(j)}\}_{j=1}^{k}, \{e_u^{(j)}\}_{j=1}^{k})$. Ensemble augmentation using both *PP* and *PE* did not show further improvements compared to using *PE* alone and are hence omitted for brevity.

[6] Results with other error (e.g., MAE, MSE) and ranking-based metrics (e.g., MRR, MAP) showed a high correlation (above 86%) with those reported here and are hence omitted.

[7] http://surpriselib.com/

[8] http://scikit-learn.org/

[9] https://sourceforge.net/p/lemur/wiki/RankLib/

[10] https://github.com/Guzpenha/PerformanceEstimates

### 4.5 Evaluation Procedure

We evaluate all ensembles in a top-20 recommendation task by reporting nDCG@20 on the test partition of each dataset. Following Lopes et al. [26], instead of predefining a relevance scale based on absolute rating values, for each user $u$ in a dataset, we discretize her test ratings into a 3-level relevance scale after correcting for the user bias $\bar{r}_u$. Precisely, we define relevance level 2 if $r_{ui} \geq \bar{r}_u$, 1 if $r_{ui} < \bar{r}_u$, and 0 for 50 randomly selected unseen items, which we assume are not relevant for the user, following Cremonesi et al. [13]. As a result, an item is considered highly relevant if rated above average by the target user, somewhat relevant if rated positively yet below average, and not relevant if it did not attract the user's attention. To compare our approach to baselines we conducted paired two-sided Student's $t$-tests with Bonferroni correction (when comparing more than two models) using a 95% confidence level.

## 5 EXPERIMENTAL EVALUATION

In this section, we empirically evaluate our approach in light of the research questions posed in Section 4.

### 5.1 Ensembling Effectiveness

To address *Q1*, we assess the extent to which performance estimates can improve the effectiveness of recommendation ensembles. To this end, we contrast the effectiveness of ensembles using only the scores of base recommenders (*RS*) to those that integrate *RS* with either performance predictors (+*PP*) or our proposed performance estimates (+*PE*). For performance estimates, we consider both their unweighted version from Equation (3) (denoted $+PE_u$ for clarity) as well as their weighted version from Equation (4) (denoted $+PE_w$).

Table 3 shows the results of this investigation. The best values in each row are in bold, while a superscript letter denotes a statistically significant improvement over the method in the corresponding column. Compared to *RS*, either $+PE_u$ or $+PE_w$ significantly improve for 6 out 8 ensembles (exceptions are LambdaMART and SVM) for the Yelp dataset. For Amazon Books, significant improvements are observed for 5 out of 8 ensembles (exceptions are LambdaMART, SVM and Gradient Boosting), whereas for Amazon Electronics, 7 out of 8 ensembles are significantly improved (exception is Gradient Boosting). Compared to using +*PP*, either $+PE_u$ or $+PE_w$ improve in most cases: 20 out of 24 ensembles (83%), with nDCG@20 gains up to 79% (with a mean gain of 23%) compared to using only *RS* and with gains up to 64% (with a mean gain of 14%) compared to using *RS* + *PP*. Recalling question *Q1*, this attests the effectiveness of performance estimates for augmenting recommendation ensembles as a replacement for handcrafted performance predictors.

### 5.2 Robustness to Ensembling Strategy

The previous results demonstrated the effectiveness of performance estimates (*PE*) as an alternative to performance predictors (*PP*) for augmenting recommendation ensembles. To address *Q2*, we assess the robustness of *PE* to different ensembling strategies. We note from Table 3 that the unweighted variant $PE_u$, which provides item-agnostic performance estimates, is particularly effective for pointwise ensemblers (gradient boosting, random forest, SVM, neural networks), significantly outperforming the weighted variant $PE_w$ in 7 out of 12 cases. This suggests that non-linear pointwise models are capable of leveraging unweighted estimates as a mechanism to adapt the learned ensemble to the specificities of different users, regardless of any particular item. In contrast, the weighted variant $PE_w$, which discriminates performance estimates for different items, is often more effective (10 out of 12 cases) for pairwise (RankBoost) and listwise ensemblers (AdaRank, ListNet). A key distinction of these ensemblers, which might explain their preference for the weighted variant, is their pursuit of an accurate *relative* ordering of items (as opposed to an accurate *absolute* item relevance estimation). Recalling question *Q2*, with the exception of LambdaMART, which is significantly improved only for Amazon Electronics, these results further attest the robustness of performance estimates for different ensemblers.

**Table 3: nDCG@20 for ensemblers leveraging different features: base recommenders scores only (*RS*), added performance predictors (+*PP*), and added performance estimates ($+PE_w$ and $+PE_u$). A dashed line separates pointwise ensemblers (bottom) from pairwise and listwise ensemblers (top).**

| | baselines | | proposed approaches | |
|---|---|---|---|---|
| **Ensemble** | *RS* (a) | +*PP* (b) | $+PE_w$ (c) | $+PE_u$ (d) |
| Yelp | | | | |
| AdaRank | 0.427 | $0.468^{ad}$ | $\mathbf{0.668}^{abd}$ | 0.427 |
| LambdaMART | $0.592^{c}$ | $\mathbf{0.597}^{c}$ | 0.465 | $0.591^{c}$ |
| ListNet | $0.456^{bd}$ | 0.415 | $\mathbf{0.679}^{abd}$ | $0.451^{b}$ |
| RankBoost | 0.500 | $0.510^{ad}$ | $\mathbf{0.556}^{abd}$ | 0.499 |
| GradBoosting | 0.437 | 0.435 | 0.438 | $\mathbf{0.450}^{abc}$ |
| SVM | 0.434 | $\mathbf{0.479}^{acd}$ | 0.431 | $0.443^{ac}$ |
| NeuralNetwork | $0.435^{c}$ | 0.432 | 0.426 | $\mathbf{0.442}^{c}$ |
| RandomForest | 0.429 | 0.429 | $\mathbf{0.451}^{ab}$ | $0.450^{ab}$ |
| Amazon Books | | | | |
| AdaRank | 0.451 | $0.532^{ad}$ | $\mathbf{0.765}^{abd}$ | 0.457 |
| LambdaMART | $\mathbf{0.477}^{cd}$ | $0.475^{cd}$ | 0.393 | $0.461^{c}$ |
| ListNet | 0.444 | $0.572^{ad}$ | $\mathbf{0.725}^{abd}$ | $0.467^{a}$ |
| RankBoost | 0.470 | $0.565^{ad}$ | $\mathbf{0.688}^{abd}$ | $0.479^{a}$ |
| GradBoosting | 0.508 | $0.513^{d}$ | $\mathbf{0.519}^{d}$ | 0.500 |
| SVM | $0.502^{c}$ | $\mathbf{0.520}^{acd}$ | 0.452 | $0.501^{c}$ |
| NeuralNetwork | 0.485 | 0.477 | $0.510^{ab}$ | $\mathbf{0.513}^{ab}$ |
| RandomForest | 0.458 | $0.471^{a}$ | $\mathbf{0.520}^{abd}$ | $0.486^{ab}$ |
| Amazon Electronics | | | | |
| AdaRank | 0.457 | $0.586^{ad}$ | $\mathbf{0.819}^{abd}$ | 0.460 |
| LambdaMART | 0.496 | $0.588^{ad}$ | $\mathbf{0.687}^{abd}$ | 0.489 |
| ListNet | $0.496^{d}$ | $0.598^{ad}$ | $\mathbf{0.816}^{abd}$ | 0.462 |
| RankBoost | 0.454 | $0.580^{ad}$ | $\mathbf{0.635}^{abd}$ | 0.455 |
| GradBoosting | $\mathbf{0.512}^{bc}$ | 0.488 | 0.486 | $0.510^{bc}$ |
| SVM | 0.444 | $0.459^{a}$ | $\mathbf{0.471}^{a}$ | $0.468^{a}$ |
| NeuralNetwork | 0.480 | $0.492^{c}$ | 0.472 | $\mathbf{0.493}^{ac}$ |
| RandomForest | 0.445 | 0.451 | $0.463^{a}$ | $\mathbf{0.478}^{abc}$ |

### 5.3 Discriminative Power

Thus far, we have used RMSE as the evaluation metric for calculating performance estimates. As an absolute error metric, RMSE does not directly detect mistaken item swaps, nor swaps in higher (and hence more important) ranking positions. Given our focus

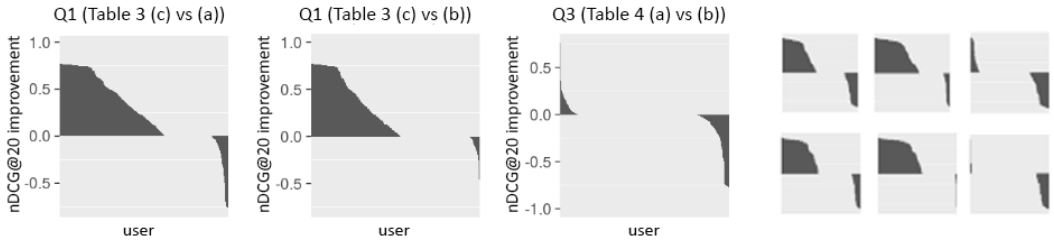


**Figure 3: nDCG@20 improvement for ListNet across Yelp users (miniaturized rows on the right show similar distributions for Amazon Books (top) and Electronics (bottom)). Each plot addresses a research question, with the plot title indicating the settings compared in each case (e.g., the first plot compares columns (c) and (a) of Table 3).**

on the ranking task, a natural question is whether producing performance estimates using a ranking-based metric, such as nDCG, could be more discriminative and, as a result, improve ensembling. In this section, we address *Q3*, by contrasting the discriminative power of performance estimates computed using either RMSE (denoted $PE^e$) or nDCG (denoted $PE^g$) as representative of error and ranking-based metrics, respectively. Table 4 shows the results of this investigation, once again for both the $PE_u$ and $PE_w$ variants.

Table 4 shows that $PE^e$ is at least as effective as (and sometimes outperforms) $PE^g$ on 34 out of 48 cases (71%). This result can be observed for both weighted (16 out of 24 cases) and unweighted (18 out of 24 cases) variants of *PE* leveraged by both pointwise (15 out of 24 cases) and pairwise/listwise ensemblers (19 out of 24 cases). A possible explanation for the inferior discriminative power of nDCG in this scenario is that it does not distinguish between items with the same relevance level. For instance, consider a user who rated three items with [5, 3, 2] stars respectively. If one of the base recommenders in the ensemble scored these items [3, 2, 0], its $PE^g$ would be 1.0, which is the best possible value, while its $PE^e$ would be 3 (0 being the best possible value), capturing the absolute scoring errors made by the recommender. Recalling *Q3*, in contrast to nDCG, RMSE provides a more fine-grained assessment of such tied items, which could help explain its improved discriminative power as a performance estimate for ensembling recommenders.

## 5.4 Breakdown Analyses

In this section, we perform three additional analyses to provide further insight into the investigations conducted thus far. Firstly, to assess the extent to which our observations for the previously stated research questions hold,[11] we perform a breakdown of improvements across users. To this end, we selected ListNet as a representative of the several ensemblers used in our experiments. Figure 3 plots nDCG@20 improvements across all users on Yelp (distributions on the other datasets are strikingly similar, and are miniaturized in the figure). Each plot addresses a different research question, with the plot title indicating the settings compared in each case (e.g., the first plot compares columns (c) and (a) of Table 3).

From Figure 3, we observe that question *Q1* is answered positively for the majority of users (90%), corroborating the reported effectiveness of our proposed performance estimates compared to using only recommender scores (first plot) as well as to using performance predictors (second plot). Regarding question *Q3*, the third plot shows neutral nDCG@20 improvements for the majority of users, confirming that using a ranking evaluation metric for performance estimation does not necessarily have a positive effect.

[11] *Q2* is not analyzed, as we fix the ensembler for this investigation.

**Table 4: nDCG@20 for ensemblers leveraging performance estimates computed using either RMSE ($PE^e$) or nDCG ($PE^g$). All combinations implicitly include the score of base recommenders (*RS*). A dashed line separates pointwise ensemblers (bottom) from pairwise and listwise ensemblers (top).**

| Ensemble | $PE^g_w$ (a) | $PE^e_w$ (b) | $PE^g_u$ (a) | $PE^e_u$ (b) |
|---|---|---|---|---|
| Yelp | | | | |
| AdaRank | 0.479 | **0.668**$^a$ | **0.427** | **0.427** |
| LambdaMART | **0.599**$^b$ | 0.465 | 0.590 | **0.591** |
| ListNet | 0.649 | **0.679**$^a$ | 0.450 | **0.451** |
| RankBoost | 0.538 | **0.556**$^a$ | **0.499** | **0.499** |
| GradBoosting | 0.430 | **0.438**$^a$ | 0.444 | **0.450** |
| SVM | 0.430 | **0.431** | 0.433 | **0.443**$^a$ |
| NeuralNetwork | **0.428** | 0.426 | 0.432 | **0.442**$^a$ |
| RandomForest | 0.417 | **0.451**$^a$ | 0.440 | **0.450**$^a$ |
| Amazon Books | | | | |
| AdaRank | 0.667 | **0.765**$^a$ | **0.457** | **0.457** |
| LambdaMART | **0.450**$^b$ | 0.393 | **0.465** | 0.461 |
| ListNet | 0.657 | **0.725**$^a$ | 0.434 | **0.467**$^a$ |
| RankBoost | 0.645 | **0.688**$^a$ | **0.479** | **0.479** |
| GradBoosting | 0.461 | **0.519**$^a$ | **0.504** | 0.500 |
| SVM | **0.519**$^b$ | 0.452 | 0.487 | **0.501**$^a$ |
| NeuralNetwork | **0.513** | 0.510 | 0.505 | **0.513** |
| RandomForest | 0.498 | **0.520**$^a$ | **0.488** | 0.486 |
| Amazon Electronics | | | | |
| AdaRank | 0.784 | **0.819**$^a$ | **0.460** | **0.460** |
| LambdaMART | 0.462 | **0.687**$^a$ | 0.472 | **0.489**$^a$ |
| ListNet | 0.746 | **0.816**$^a$ | **0.491**$^b$ | 0.462 |
| RankBoost | **0.688**$^b$ | 0.635 | **0.455** | **0.455** |
| GradBoosting | **0.491** | 0.486 | **0.518** | 0.510 |
| SVM | 0.436 | **0.471**$^a$ | 0.458 | **0.468**$^a$ |
| NeuralNetwork | **0.473** | 0.472 | **0.511**$^b$ | 0.493 |
| RandomForest | 0.457 | **0.463** | 0.473 | **0.478** |

Overall, we conclude that our approach increases recommendation accuracy for most users while incurring minimum risk of a decreased accuracy for individual users.

Another question is how different ensembles benefit from different feature sets. To shed light on this, we represent each ensemble as an $m$-dimensional vector $\vec{v}$, where $m$ is the number of test users in the underlying dataset, and $\vec{v}_i$ denotes the nDCG@20 attained by the ensemble for the $i$-th user. To better visualize how different ensembles perform, we project their vector representation onto a 2-dimensional embedded space using t-SNE [29], such that neighboring ensembles perform similarly (in terms of nDCG@20) across the user base. Figure 4 shows the output of this process for all three datasets. From the figure, we note that pointwise methods and listwise/pairwise methods generate two well-separated clusters, indicating that they behave very differently for each user and have close intra-similarity (e.g. RankBoost and AdaRank are close in the visualization). We further observe that the choice of $PE$ (unweighted or weighted, visualized as different shapes) also induces sub-clusters in the embedded space, sometimes independent of the choice of ensembler (visualized as different colors). An example can be seen in the plot for Amazon Electronics, with sub-clusters induced for $PP$ (cross symbols) on the top left and another sub-cluster to its right for $PE^e_w$ (triangles labeled `PE_weighted_rmse`) and $PE^g_w$ (plus symbols labeled `PE_weighted_ndcg`).

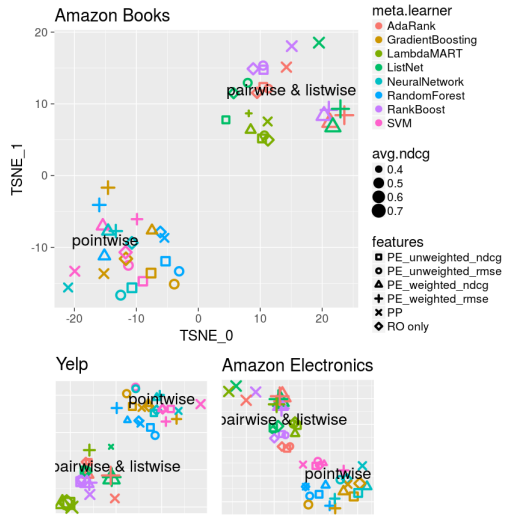


**Figure 4: Dimensionality reduction of various ensembles using t-SNE, with input dimensions denoting their nDCG@20 performance for users on Amazon Books (similar results are observed for the other two datasets, as shown in the miniaturized plots at the bottom).**

Finally, we analyze the importance of different features for the ensembles. To get a rough idea as to the importance ranking for each set of features, we used random forest combined with Gini importance [27] as the sorting criteria, training three distinct models for each input space, $RS + PP$, $RS$, and $RS + PE$. For this experiment, we chose the $PE^e_w$ variant given our findings for the previously investigated research questions. Due to space limitations, in Table 5, we report the results only for Amazon Books, which shows similar trends as the other datasets. From the table, we observe that the two most important performance predictors ($PP_7$ and $PP_6$) are user abnormality formulations [17], which indicate how atypical the user preferences are. In contrast, the top-10 performance estimates often correspond to recommenders that are also highly ranked themselves as features for the ensemble (e.g., $SVD_5$, $SVD_1$, $BO$), which further demonstrates the discriminative power of $PE$ for identifying effective base recommenders.

**Table 5: Feature importance on Amazon Books. We abbreviate BaselineOnly as BO and NormalPredictor as NP. Each set of features was used in three different models for this analysis ($RS + PP$, $RS$, and $RS + PE$, respectively).**

| $RS + PP$ | | $RS$ | | $RS + PE$ | |
|---|---|---|---|---|---|
| feature | imp. | feature | imp. | feature | imp. |
| $PP_7$ | 0.031 | $SVD_5$ | 0.086 | $PE\_NP$ | 0.035 |
| $PP_6$ | 0.029 | $BO$ | 0.085 | $PE\_SVD_5$ | 0.026 |
| $PP_{12}$ | 0.029 | $SVD_3$ | 0.074 | $PE\_SVD_1$ | 0.023 |
| $PP_{10}$ | 0.025 | $SVD_1$ | 0.069 | $PE\_BO$ | 0.022 |
| $PP_8$ | 0.024 | $SVD_4$ | 0.069 | $PE\_SVD_4$ | 0.022 |
| $PP_9$ | 0.017 | $NMF_4$ | 0.066 | $PE\_NMF_3$ | 0.022 |
| $PP_{14}$ | 0.017 | $NMF_3$ | 0.050 | $PE\_NMF_5$ | 0.021 |
| $PP_3$ | 0.017 | $NP$ | 0.050 | $PE\_SVD_2$ | 0.021 |
| $PP_1$ | 0.016 | $NMF_5$ | 0.048 | $PE\_NMF_1$ | 0.020 |

## 6 CONCLUSION

We proposed performance estimates as a novel class of features for augmenting recommendation ensembles. Performance estimates are computed on the target user's historical feedback by standard evaluation metrics, such as RMSE or nDCG. As a result, they are highly discriminative of the performance of different recommenders and incur no engineering cost when new recommenders are added to the ensemble. A thorough evaluation using datasets in three different domains demonstrated the effectiveness of performance estimates at improving ensembles produced by representative pointwise, pairwise, and listwise learning to rank approaches.

As future work, we plan to investigate interactions between different features (including base recommender scores, performance predictors, and performance estimates). This may help further understand the circumstances under which ensembles fail for individual users, as a means to further improve their robustness. In a similar vein, we plan to investigate approaches for feature selection, which may have a positive impact on the efficiency of ensembles in large-scale deployments. Finally, we plan to evaluate the effectiveness of our approach for datasets with other types of interactions such as implicit feedback and also in predicting the performance of recent deep neural recommenders.

## ACKNOWLEDGMENTS

Work partially funded by project MASWeb (FAPEMIG APQ-01400-14) and by the authors' individual grants from CNPq and FAPEMIG.

# REFERENCES


[1] Charu C Aggarwal. 2016. Ensemble-based and hybrid recommender systems. In *Recommender Systems*. Springer, 199–224.

[2] Charu C Aggarwal. 2016. *Recommender Systems*. Springer.

[3] Javed A Aslam and Mark Montague. 2001. Models for metasearch. In *SIGIR*. ACM, 276–284.

[4] Xinlong Bao, Lawrence Bergman, and Rich Thompson. 2009. Stacking recommendation engines with additional meta-features. In *RecSys*. ACM, 109–116.

[5] Ariel Bar, Lior Rokach, Guy Shani, Bracha Shapira, and Alon Schclar. 2012. Boosting simple collaborative filtering models using ensemble methods. *arXiv preprint arXiv:1211.2891* (2012).

[6] Alejandro Bellogín, Pablo Castells, and Iván Cantador. 2011. Predicting the performance of recommender systems: an information theoretic approach. In *ICTIR*. Springer, 27–39.

[7] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, Feb (2012), 281–305.

[8] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24, 2 (1996), 123–140.

[9] Leo Breiman. 1996. Stacked regressions. *Machine learning* 24, 1 (1996), 49–64.

[10] Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User Model. User-Adapted Interac.* 12, 4 (2002), 331–370.

[11] David Carmel and Elad Yom-Tov. 2010. Estimating the query difficulty for information retrieval. *Synthesis Lectures on Information Concepts, Retrieval, and Services* 2, 1 (2010), 1–89.

[12] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *SIGKDD*. ACM, 785–794.

[13] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*. ACM, 39–46.

[14] Steve Cronen-Townsend, Yun Zhou, and W Bruce Croft. 2002. Predicting query performance. In *SIGIR*. ACM, 299–306.

[15] Reinaldo Silva Fortes, Alan R. R. de Freitas, and Marcos André Gonçalves. 2017. A multicriteria evaluation of hybrid recommender systems: on the usefulness of input data characteristics. In *ICEIS*. 623–633.

[16] Thomas George and Srujana Merugu. 2005. A scalable collaborative filtering framework based on co-clustering. In *ICDM*. IEEE, 4–pp.

[17] Benjamin Gras, Armelle Brun, and Anne Boyer. 2015. Identifying users with atypical preferences to anticipate inaccurate recommendations. In *WEBIST*.

[18] Josephine Griffith, Colm O'Riordan, and Humphrey Sorensen. 2012. Investigations into user rating information and predictive accuracy in a collaborative filtering domain. In *SAC*. ACM, 937–942.

[19] Claudia Hauff. 2010. *Predicting the effectiveness of queries and retrieval systems*. Ph.D. Dissertation. University of Twente.

[20] Michael Jahrer, Andreas Töscher, and Robert Legenstein. 2010. Combining predictions for accurate recommender systems. In *SIGKDD*. ACM, 693–702.

[21] Michael Kearns. 1988. Thoughts on hypothesis boosting. *Unpublished manuscript* 45 (1988), 105.

[22] Yehuda Koren. 2009. The Bellkor solution to the Netflix grand prize. *Netflix prize documentation* 81 (2009), 1–10.

[23] Yehuda Koren. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4, 1 (2010), 1.

[24] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[25] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[26] Ramon Lopes, Renato Assunção, and Rodrygo LT Santos. 2016. Efficient Bayesian methods for graph-based recommendation. In *RecSys*. ACM, 333–340.

[27] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. 2013. Understanding variable importances in forests of randomized trees. In *NIPS*. 431–439.

[28] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Trans. Ind. Inf.* 10, 2 (2014), 1273–1284.

[29] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9, Nov (2008), 2579–2605.

[30] Craig Macdonald, Rodrygo L. T. Santos, and Iadh Ounis. 2012. On the usefulness of query features for learning to rank. In *CIKM*. ACM, 2559–2562.

[31] Fiana Raiber and Oren Kurland. 2014. Query-performance prediction: setting the expectations straight. In *SIGIR*. ACM, 13–22.

[32] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. 2009. Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460* (2009).

[33] Daniel Valcarce, Javier Parapar, and Álvaro Barreiro. 2017. Combining top-n recommenders with metasearch algorithms. In *SIGIR*. ACM, 805–808.