PERFORMANCE PREDICTION FOR ENHANCING ENSEMBLE LEARNING

GUSTAVO PENHA

PERFORMANCE PREDICTION FOR ENHANCING ENSEMBLE LEARNING

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

ADVISOR: RODRYGO SANTOS

Belo Horizonte

2018

Penha, Gustavo

P399p Performance Prediction for Enhancing Ensemble Learning / Gustavo Penha. — Belo Horizonte, 2018
xxii, 69 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Rodrygo Santos

1. Computação – Teses. 2. Recuperação da informação. – Teses. 3. Sistemas de recomendação – Teses. I. Orientador. II. Título.

CDU 519.6*73(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Performance Prediction for Enhancing Ensemble Learning

GUSTAVO PENHA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]

PROF. RODRYGO LUIS TEODORO SANTOS - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]

PROF. MARCOS ANDRÉ GONÇALVES
Departamento de Ciência da Computação - UFMG

[illegible]

PROF. MARCELO GARCIA MANZATO
Departamento de Ciências da Computação - USP

Belo Horizonte, 31 de agosto de 2018.

Acknowledgments

There are several people that I am indebted to and I must express my gratitude for their support during my MSc. First I am grateful for my family and their unconditional support for my choices. Second, my advisor Rodrygo Santos, who guided my ideas and directly helped me turning them into this dissertation with his instructions, recommendations and experience. Also, I would like to thank every coworker at Hekima, with whom I shared my ideas, results and had a lot of fruitful discussions. Most of all, I am specially grateful to my wife Sara, who shared with me the good and bad times during this process. This dissertation is dedicated to you.

"I know not all that may be coming, but be it what it will, I'll go to it laughing"

(Herman Melville)

Resumo

A combinação de modelos em aprendizado de máquina é uma técnica útil para melhorar a eficácia de sistemas em tarefas como classificação, busca e recomendação. O empilhamento de modelos, por exemplo, aprende a pesar e combinar previsões de diversos modelos base para alcançar eficácias melhores. Uma limitação do empilhamento é que em sua formulação básica não há informações sobre o contexto em que instâncias levam um modelo a ter um melhor do que outros, ponderando-as com base apenas no eficácia geral de cada modelo base. Nesta dissertação, inspirado por trabalhos em Predição de Performance, propomos usar modelos auxiliares capazes de prever a eficácia de cada modelo no empilhamento para uma nova instância. As abordagens atuais baseiam-se no desenvolvimento de atributos para prever a eficácia de sistemas e usá-los como atributos adicionais para a camada de empilhamento, que tem o ônus de compreender quando cada modelo tem eficácia melhor que os outros. De maneira diferente, nossas abordagens nas tarefas de busca e recomendação facilitam o trabalho da camada de empilhamento com um conjunto mais discriminativo de atributos. Para a tarefa de busca, demonstramos por meio de simulações que existe uma barreira de acurácia que deve ser superada para que a predição de eficácia se torne útil. Além disso, mostramos que os preditores de eficácia de consultas aprendidos por máquina para cada modelo base são capazes de ultrapassar essa barreira quando usados como meta-atributos para empilhar modelos individuais de ranqueamento via *learning to rank*. Para a tarefa de recomendação, nós propomos estimar diretamente a eficácia de cada um dos modelos base para um usuário, considerando seu conjunto histórico de avaliações, em vez de criar atributos discriminativos para prever essa eficácia. Experimentos em conjuntos de dados do mundo real de vários domínios demonstram que o uso de estimativas de eficácia como atributos adicionais melhoram significativamente a eficácia dos *ensembles* com base no aprendizado *pointwise*. Além disso, com *ensembles pairwise* e *listwise*, a utilização das estimativas de eficácia atinge performance do estado-da-arte.

Palavras-chave: Combinação de modelos, Predição de Eficácia, Sistemas de Recomendação, Recuperação de Informação.

Abstract

Ensembles of models in machine learning has proved to be a useful technique for improving the effectiveness of systems in tasks such as classification, ad-hoc retrieval and recommendation. Stacking, for instance, learns to weight and combine base models predictions in order to achieve higher performances. One limitation of stacking is that in its basic formulation it has no information on the context of instances that make a model perform better than others, weighting them based only on the overall model performance. In this dissertation, inspired by work on Performance Prediction, we propose to use auxiliary models capable of predicting the performance of each model in the ensemble for new instances. Current approaches are based on handcrafting meta-features for predicting the performance of systems and using them as additional features for the stacking layer, which has the burden of understanding when each model outperforms others. Unlike them, our novel approaches in both search and recommendation facilitates the stacking layer job with a discriminative set of features. For ad-hoc retrieval, we demonstrate through simulations that there is a prediction accuracy bar that must be overcome for query performance prediction to become useful. Moreover, we show that machine-learned query performance predictors for each base model are able to pass this bar when leveraged as meta-features for stacking individual ranking models via learning to rank. For recommendation, we propose to directly estimate the performance of base models for a user given his historical set of ratings, instead of handcrafting discriminative features for predicting it. Experiments on real-world datasets from multiple domains demonstrate that using performance estimates as additional features can significantly improve the accuracy of current ensemblers based on pointwise learning. Moreover, when used with pairwise and listwise ensemblers, exploiting performance estimates achieves state-of-the-art recommendation effectiveness.

Keywords: Ensembling, Performance Prediction, Recommender Systems, Information Retrieval.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The huge scale of content, services and products available on the World Wide Web has motivated the development of systems capable of retrieving relevant items to users given his information needs. This area of research is known as Information Retrieval (IR) and two related tasks are search – ranking relevant documents given a query – and recommendation – ranking items, e.g movies, given a user profile –. Providing rapid and accurate response to enormous user requests is a challenge faced by the methods proposed to tackle these tasks. Several ranking models have been proposed to generate rankings where the most relevant items are in the first positions, which users see first. Among them, a promising category of methods that yields state-of-the art effectiveness are the ones based on the learning to rank framework [Liu et al., 2009]. In addition to using a machine-learned strategy to generate the final ranking, this class of methods is also suited to combine different base ranking strategies, which bears resemblance to stacking [Breiman, 1996b] from machine learning.

An equally challenging task, relevant to this dissertation, is knowing when systems will fail to perform well given an information request, which is an area of research known as Performance Prediction (PP) [Cronen-Townsend et al., 2002]. Being able to accurately predict the performance of a method has several applications: avoid user dissatisfaction by omitting the response of the system, invoke alternative methods or actively using such information in order to combine multiple models for improving the effectiveness of the system.

Several approaches have been proposed to tackle the problem of predicting the performance of systems in IR for both search [Hauff, 2010; Katz et al., 2014; Shtok et al., 2016] and recommendation [Bellogín et al., 2011; Ekstrand and Riedl, 2012; Matuszyk and Spiliopoulou, 2014], and they are in essence query and user features respectively that might correlate well with the performance of a given system. For instance, a query

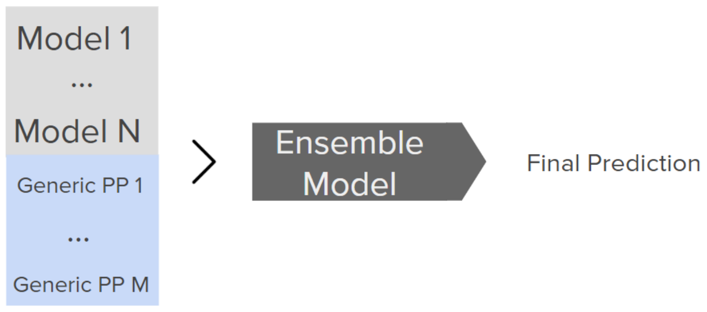


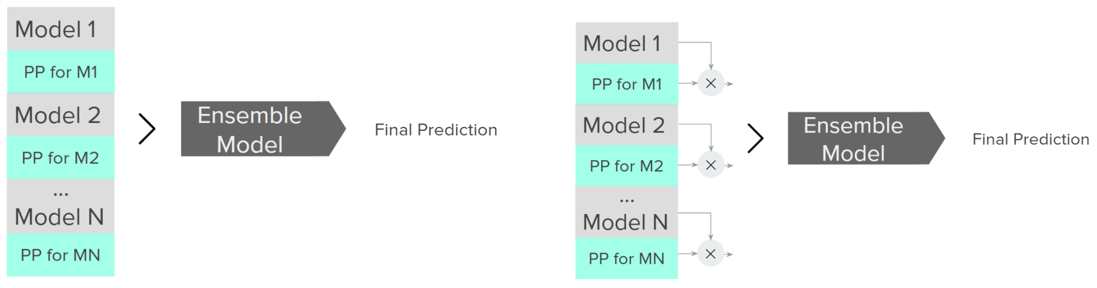
Figure 1.1: Baseline framework for exploiting performance predictions. This architecture is known as STREAM in the recommender systems literature [Bao et al., 2009] and was also explored in ad-hoc retrieval by Santos et al. [2010].

performance predictor for a vectorial model based on the TF-IDF representation might be the query length: the higher the query length in terms of words, the more complex the query might be and TF-IDF might not be able to retrieve relevant documents. Another example, in the field of recommender systems, is the number of ratings a user has: the more ratings, a collaborative filtering based model might perform better and with less ratings a content-based model might perform better.

One direction that has shown to be promising is leveraging performance predictors for the combination of models, using them as additional features to a final model that is a function of predictions of base models to the final prediction. This architecture, known as STREAM in the recommender systems literature[Bao et al., 2009], is described in the diagram of Figure 1.1. In ad-hoc search, Macdonald et al. [2012] demonstrated that query performance prediction can be used as additional features to a learning to rank model in order to improve the final results. In recommendation, both first and second place entries [Koren, 2009; Sill et al., 2009] to the Netflix challenge [Bennett et al., 2007] were stacking strategies that used at least one additional user performance predictor.

However, the expectation that improved performance prediction would lead to significant improved results for their respective tasks (ad-hoc retrieval and recommendation) has not yet been fully realized by previous work. [Raiber and Kurland, 2014] argued that the reason improvements in the performance prediction task have not fully translated to improvements in the search task is due to the intrinsic difficulty of predicting performance, which boils down to the relevance estimation task.

In this dissertation we demonstrate that indeed an accuracy bar must be passed in order to make performance predictors useful for search and recommendation. Moreover, in this dissertation we describe two strategies – ML-QPP in search and a novel approach for performance prediction in recommendation called performance estimates



(a) Unweighted (b) Weighted

Figure 1.2: Proposed framework for exploiting accurate performance predictions for the combination of different models. The core idea is that we have one effective prediction accuracy for each model in the ensemble, which facilitates the ensemble model job of weighting the models predictions, in an instance-wise manner. The weighted variation, as we explore in this dissertation, further increases the sensitivity of models predictions, by multiplying them by their predicted effectiveness.

– for making accurate performance predictions and a unified framework on how to explore accurate performance predictions to effectively improve the results of state-of-the-art ensembles of models in both search and recommendation.

1.1 Dissertation Statement

The statement of this dissertation is that effective performance prediction of different base models can be exploited to improve the results of their combination. In particular, having one performance prediction per base model as additional feature to the stacking layer provides a fine-grained comprehension of conditions – query or user for search and recommendation respectively – when each model performs better or worse than others, facilitating the combiner model job. Furthermore, multiplying each base model prediction by its respective performance prediction further increases the sensibility of this set of features, attaining higher effectiveness.

1.2 Dissertation Contributions

The key contributions of this dissertation are three-fold:

We propose a framework for utilizing concepts of performance prediction for the combination of models.

Unlike previous work, that exploit the information of features that correlate with the performance of systems by simply adding them – M performance predictors for

N models– to the ensemble input layer, we propose to have one accurate performance prediction for each model in the ensemble – N performance predictions for N models respectively. The intuition is that we take the burden of learning which performance prediction correlates with each prediction and how it should be used from the ensemble and pass it to the performance prediction techniques.

2. We demonstrate that performance prediction can be used to improve the results for ad-hoc retrieval as additional features to a combiner model, when they achieve certain accuracy bar.

Traditional approaches for query performance prediction are single features that measure the difficulty of the query for a given system. We demonstrate that this approach falls short in improving ad-hoc retrieval, performing similarly to a very noisy performance prediction. Inspired by previous work on combining query performance predictors for improved accuracy [Hauff et al., 2009], we show in this dissertation that having one machine-learned query performance predictor for each model in the ensemble is sufficient to pass the accuracy bar needed for improving ad-hoc retrieval.

3. We introduce a novel approach to performance prediction in the context of recommender systems, called performance estimation which exploits users historical ratings to estimate the effectiveness of different recommenders instead of trying to predict it based on user features.

Recent approaches to performance prediction for recommender systems that provide a measure of difficulty have shown moderate success when leveraged as additional features to the ensemble, for example the number of user ratings [Bao et al., 2009]. In this dissertation we advocate for the use of performance estimation instead of the standard approach of hand-crafting accurate user performance predictors. The key idea is to use each user past ratings in order to both train and evaluate the performance – hence directly *estimating* – of each recommender method in the ensemble. The assumption that past performance of models for users is indicative of their future performance has been evaluated in this dissertation, demonstrating that indeed performance estimation provides are more accurate and can replace user performance prediction features in the recommender systems area.

1.3 Dissertation Outline

The remainder of this dissertation is organized as follows:

- Chapter 2 describes background on Information Retrieval, including both ranking and recommendation methods, including a formal definition of the tasks and their organization into categories. This is followed by an introduction to Performance Prediction and current approaches for it in both ad-hoc search and recommendation. In addition, the combination of methods via ensembling is described and how it has been applied in Information Retrieval, including efforts to leverage Performance Prediction to enhance the accuracy of the ensemble. The chapter ends describing how the approaches proposed in the dissertation differ from the current literature.

- Chapter 3 begins with an overview of query performance predictions and how approaches are currently limited and have not shown improved ad-hoc retrieval performance. Three strategies are then defined for such task, including machine-learned query performance predictors. In addition, this chapter evaluates the usage of this approach to improve the combination of ranking functions via learning to rank.

- Chapter 4 defines a novel approach for predicting the performance of recommendation methods, and also how to effectively use them in the combination of recommender systems. This chapter also evaluates the approach in several domains, shedding light upon different facets of the proposed approaches: their effectiveness, robustness to different ensembling strategies, complementarity to other performance predictors, generalization and discriminative power. Furthermore, the chapter finishes by doing breakdown analysis across users, how different ensembles compare to each other and also the importance of specific meta-features for the ensemble.

- Chapter 5 provides a summary of the contributions and conclusions made throughout the chapters of this dissertation. Other future directions are also presented, regarding alternative applications of the statements and framework provided here.

Chapter 2

Related Work

2.1 Information Retrieval

Information retrieval (IR) is the broad area of computer science concerned with storing in a collection and retrieving items relevant to an information need. The most known and visible applications are search web engines, however several other systems and tasks can be categorized as applications of this area, such as recommenders and text classifiers. In this chapter we describe the ranking and the recommendation tasks from IR, including methods designed to tackle such problems.

2.1.1 Ranking Methods

The objective of ranking models is to create a function that predicts the relevance of a query and document combination. They are one of the core modules of search engines such as Google and Bing, responsible to sort retrieved documents in order of relevance to the user. The relevance of a document for a user query might be influenced by other contextual variables, such as location and time.

More formally, the ranking task - also called in this dissertation as ad-hoc retrieval or search - is to calculate a relevance function f that given a query q and a document d will estimate the score $s = f(q, d)$ of such combination. Ranking documents according to this function in decreasing order will generate a ranking of documents for a given query.

2.1.1.1 Categorization of methods

Ranking methods are usually grouped in the following categories regarding the evidence it leverages from the query q and document d:

- **Query-dependent**: models that score documents according to its estimated relevance to a query, examples are vector space models such as $tf - idf$ which scores a document using the similarity of the query vector representation and the document vector representation [Salton et al., 1975]. Inside this category of models, we also have probabilistic relevance ones, which instead of representing query and documents algebraically, they do so in a probabilistic manner. Examples are the Binary Independence Model [Robertson and Jones, 1976], 2-Poisson [Poisson, 1837] and BM25 [Robertson et al., 1995]. Moreover, language modelling and divergence from randomness are two other classes of models that are also query-dependent [Santos, 2013].

- **Query-independent**: in addition to estimating the relevance of a document to a query, a ranking model might only answer specific quality criteria of documents. One example of this class is PageRank [Page et al., 1999], which uses the link structure formed in the web to score documents regarding their importance on this graph.

- **Machine-learned**: models that use machine learning principles have also been applied to the ranking problem. Liu et al. [2009] argues that the machine learning framework tackles problems faced by researchers in IR such as automatically tuning parameters, combining multiple models, and avoiding over-fitting. Approaches that uses the learning to rank framework can be categorized according to their choice of input and output representation and their underlying model structure: pointwise, pairwise and listwise. A tree based example of this category is LambdaMart [Wu et al., 2008].

2.1.2 Recommendation Methods

The objective of Recommender Systems (RS) is to create a **utility function** that predicts how much a user will like an item. User behavior is modeled by such systems based on their **explicit feedback**, as in ratings for movies, or **implicit feedback**, as in product views.

Other variables might influence the utility of an item, such as the time of the day, location and even what the user is currently doing. This kind of information is defined as **context**. Another important aspect of RS is the **domain** in which it works. The item domain might vary from scientific articles to locations to travel. This scope can influence decisions on the modeling of items.

More formally, the RS task is usually defined in two different ways [Aggarwal, 2016b] :

1. *Prediction task*: This approach is to predict the rating value for a user-item combination. For m users and n items, the training data is an incomplete $m \times n$ matrix, where the observed values are used for training and the missing values are predicted by the model. This is also referred as *matrix completion problem.*
2. *Ranking task*: In practice, we only need to rank items for specific users, determining the top-k items the user will most likely find useful. This is also referred as *top-k recommendation problem.*

2.1.2.1 Relationship to Supervised Learning

We could view the *prediction task* of RS as a generalization of classification and regression models. In common supervised learning tasks, the class variable can be viewed as an attribute with missing values, whereas in the *prediction task* from RS any column is allowed to have missing values (any entry could be the class variable). Such relationship is crucial, as many principles from the area of classification and regression can be applied to Recommender System research. This relationship is illustrated in Figure 2.1, extracted from [Aggarwal, 2016b].

2.1.2.2 Categorization of methods

Recommender System models are usually grouped in the following categories:

- Collaborative Filtering Models: The main intuition behind Collaborative Filtering (CF) methods is that we can use the collaborative power of the community of users to infer which ones have similar tastes and then complete the missing values of the ratings matrix based on this correlation. This group of methods can be further divided into memory-based methods (predictions based on the neighborhoods) and model-based methods (a model is learned by an optimization procedure based on the training set).
- Content-Based Recommender Systems: In Content-Based (CB) RS attributes of items and users are used, in combination with the rating information, in order to deliver predictions.
- Knowledge-Based Recommender Systems: The recommendation is not based on ratings, but rather on the match of the user requirements for the items and the item descriptions.

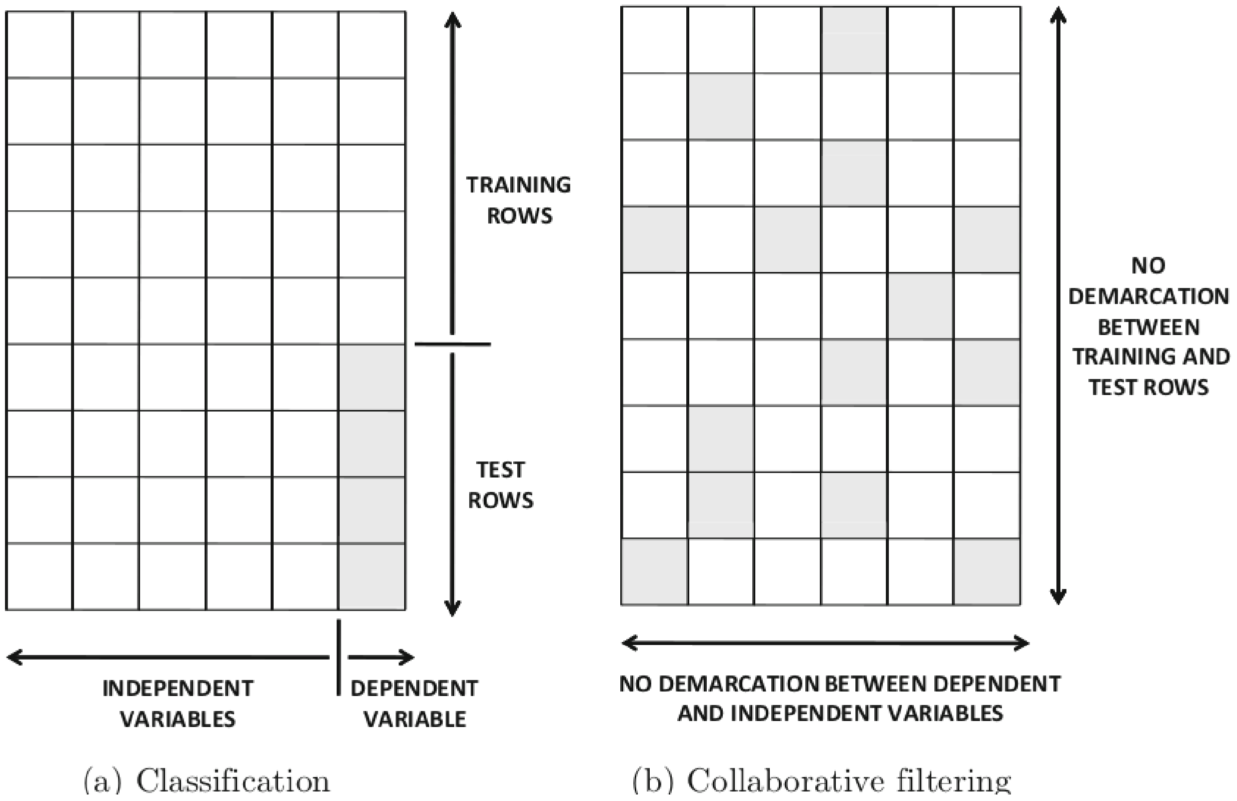

TRAINING ROWS
TEST ROWS
INDEPENDENT VARIABLES
DEPENDENT VARIABLE
(a) Classification
NO DEMARCATION BETWEEN TRAINING AND TEST ROWS
NO DEMARCATION BETWEEN DEPENDENT AND INDEPENDENT VARIABLES
(b) Collaborative filtering

Figure 2.1: Relationship of Recommender Systems to supervised learning, extracted from [Aggarwal, 2016b].

- Demographic Recommender Systems: Such methods learn mappings from specific demographic groups to buying or rating propensities.
- Hybrid and Emsemble-Based Recommender Systems: Various aspects of RS methods are combined in order to achieve better performance. We will further discuss such methods in the next section.

2.1.2.3 Hybridization

One of the first surveys on different types of Hybrid RS was made by Burke [2002], which categorizes them into seven different categories:

- Weighted: The predictions of multiple RS are combined together to produce a single recommendation.
- Switching: The system decides between RS techniques depending on the situation.
- Mixed: Multiple recommendations from different methods are displayed at the same time.

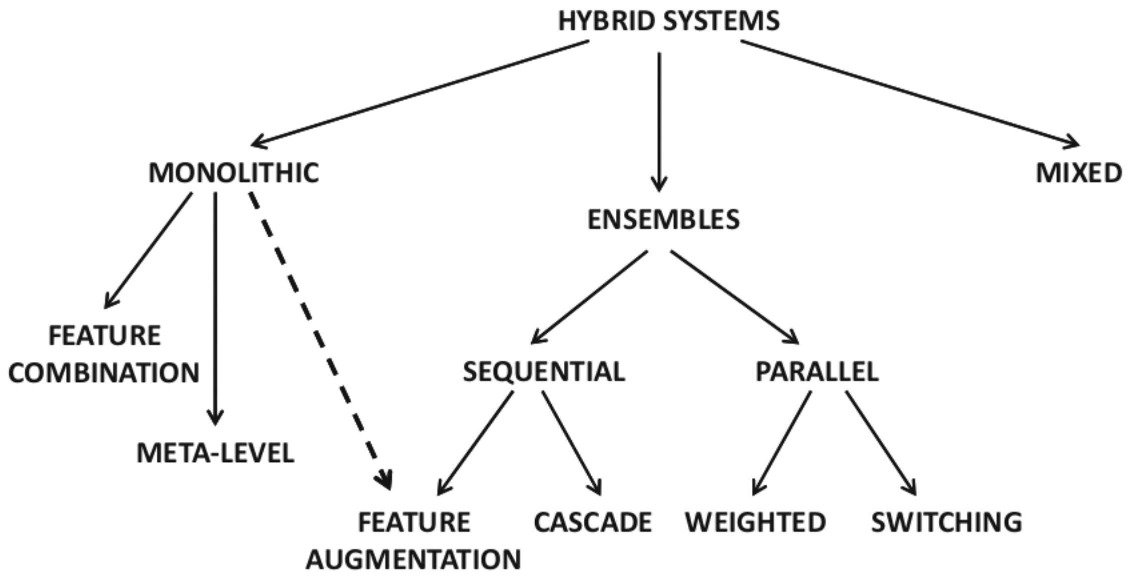


Figure 2.2: Taxonomy of hybrid systems extracted from [Aggarwal, 2016a]

- Cascade: Recommenders refine recommendations given by another technique.
- Feature augmentation: The outputs of RS are used as features for another method.
- Meta-level: A model that was learned is used as input by another recommender.
- Feature Combination: Multiple data sources are used by a single RS.

Aggarwal [2016a] made a recent effort into categorization of types of RS ensembles, by revisiting Burke [2002] and adding depth to it. His comparison involves a parallel between ensembling in classification, which is an area of machine learning that has successfully applied ensembling, and ensembling in RS. The taxonomy proposed is illustrated in Figure 2.2.

Recently, works on RS that combine multiple methods have been proposed, and the two winning entries for the Netflix Competition were ensembles [Sill et al., 2009; Koren, 2009]. Jahrer et al. [2010] combine predictions from 18 RS in the Netflix dataset using several blending algorithms. The best single algorithm was a Feedforward Neural Network (also called Multilayer Perceptron) [Goodfellow et al., 2016], and the best ensemble was obtained by combining Linear Regression [Montgomery et al., 2015] with polynomial feature engineering, Feedforward Neural Network and Bagged Gradient Boosted Decision Trees (combines Bagging [Breiman, 1996a], Gradient Boosting [Friedman, 2002] and Random Subspace selection [Breiman, 2001]). An improvement over Feature-Weighted Linear Stacking [Sill et al., 2009] was reported.

Dooms et al. [2015] approaches to hybridization include a switching algorithm and a weighted one. Combining predictions from 10 RS using the error from training time to estimate the RS's weights, they obtained improvements over using a single algorithm.

The category of hybrids most relevant to this dissertation is weighted ensembles, which has empirically shown the best effectiveness improvements for the recommendation task.

2.2 Performance Prediction

Performance prediction has been an established research topic in Information Retrieval (IR) [Cronen-Townsend et al., 2002], and the task is to predict the performance[1] of a query in a specific configuration of an Information Retrieval system. The approaches to the task have been classified into pre-retrieval features (predictions are made before the retrieval stage) and post-retrieval features (uses the rankings produced by the retrieval engine).

The difficulty of a query with respect to a certain system can be measured in different dimensions such as specificity, ranking sensitivity, ambiguity, term relatedness and using query/document/retrieval perturbations or language models [Hauff, 2010]. Several approaches have been proposed to improve the effectiveness for the performance prediction task, however, the expectation that improved performance prediction would lead to improved adhoc retrieval has not yet been fulfilled Raiber and Kurland [2014].

Even though, this topic still receives attention of researchers in IR: for measuring users satisfaction [Dan and Davison, 2016], improving the quality of predictors [Arguello et al., 2017; Shtok et al., 2016; Katz et al., 2014].

This problem was first explored in the field of Recommender Systems by Bellogín et al. [2011], who adapted query clarity features from [Cronen-Townsend et al., 2002] and used them to predict performance of RS. The evaluation procedure involves calculating correlation of such predictors to the errors of RS, reaching a maximum of 0.5 Pearson correlation.

[1]A more adequate term would be effectiveness prediction, since what is being predicted is the system capacity to make correct predictions and not other aspects such as efficiency. However, due to adoption of the term by the literature, throughout this dissertation we maintain the nomenclature of Performance Prediction (PP).

Since then, this topic has been further explored by researchers in the area. Ekstrand and Riedl [2012] proposed a switching hybridization algorithm using a Logistic Regression based on three user features, resulting in minor improvements over a single state-of-the-art model. Ghazanfar and Prügel-Bennett [2014] compared clustering-based Collaborative Filtering methods and proposed a hybridization of CCF and Content-based methods using the weights learned in the clustering step of CCF. However, it did not outperform single Clustering-based Collaborative Filtering methods.

Gras et al. [2015] explored user features that correlate with low accuracy in RS, in order to identify users who are outliers and consequently get poor recommendations. The features used are Abnormality (User average ratings distance to items he rated), average Pearson correlation with top-K similar neighbours, AbnormalityCR (Abnormality using standard deviation of items) and AbnormalityCRU (AbnormalityCR centered by user average rating). Results show Pearson correlations up to 0.55 between features and error metrics. Griffith et al. [2012] proposed a Decision Tree based regression model (M5) to predict the performance of a RS. The features are first selected (from a total of 11 features) for each dataset and used as input to the regressor. Results show good correlation between prediction from the regression model and ground-truth (0.8 Pearson correlation).

The first part of this dissertation statement is *"**Effective performance prediction of different base models** can be exploited to improve the results of their combination"*, which depends upon the literature on performance prediction to obtain accurate results. In order to improve the effectiveness on the task, in the recommendation scenario, we propose a novel approach for performance prediction. Unlike previous work that is based upon manually creating user features that might correlate with the performance of methods, our proposal is to directly estimate it using the user historical set of ratings. On the other hand, for the ad-hoc retrieval scenario, we resort to a technique that combines multiple query performance predictions by using a machine-learned approach [Hauff et al., 2009]. In both cases, we obtain one performance prediction for each model in the system, which is a key aspect of our proposed framework, as discussed in Section 2.4.

2.3 Ensemble Methods

Ensemble is the area of machine learning concerned with the combination of the output of several base learners in order to improve the generalization and robustness compared to using only one learner. Bootstrap aggregating (bagging) [Breiman, 1996a] and boosting [Kearns, 1988] 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 weak learner.

Even though such methods have been explored in the recommender systems literature with some success [Bar et al., 2012], in this dissertation we focus on the class of ensembles that are able to combine models created from different hypothesis spaces, namely, stacking [Breiman, 1996b], which can be applied to both the search and recommendation problem. 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, as we see in the diagram in Figure 2.3. This idea has also been successfully used in the field of recommender systems for producing hybrid recommendations [Aggarwal, 2016a; Burke, 2002]. For instance, both the winner and second place of the Netflix competition [Koren, 2009; Sill et al., 2009] employed a blending of multiple recommender systems in their solutions.

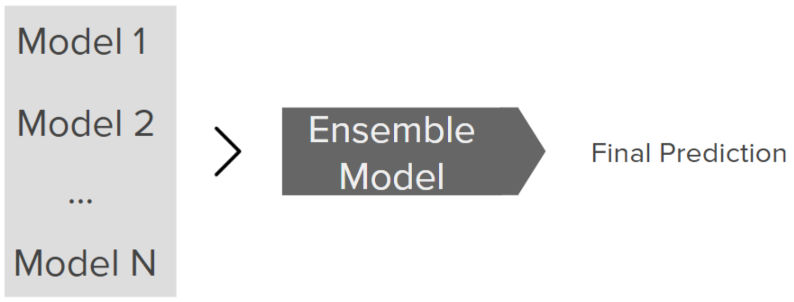


Figure 2.3: The framework for combing multiple models using machine learning. Different ensemble models can be used for this task, including models that optimize pointwise, pairwise and listwise objectives.

Strategies to combine different rankings have also been proposed, and they can be divided into those that require training data and those that do not. The area concerned with methods that do not require training a model is generally known as rank aggregation [Aslam and Montague, 2001]. Rank aggregation techniques have also been applied to combine the output of multiple recommenders [Valcarce et al., 2017]. The other research strand for the combination of multiple rankings, which requires training data, is known as learning to rank [Liu et al., 2009]. The adoption of supervised machine learning for combining different ranking models has led to state-of-the-art results [Wu et al., 2008].

2.4 Leveraging Performance Prediction in Ensembles

Using performance predictors for the task of ad-hoc retrieval has been done in the literature in the following manners: used as additional features for the ensemble [Macdonald et al., 2012], i.e. increasing the learning to rank input size with n query performance predictions, used as the weight assigned to the list in a linear fusion combination of lists [Raiber and Kurland, 2014] and selecting the best predicted model [Berger and Savoy, 2007]. When using them to directly weight or select between base models, the performance prediction strategy is outperformed by using the cross-validation error of each model. When used as additional features to the ensemble, the effectiveness improvements are inexpressive, as we also show in this dissertation.

In the field of recommender systems, using performance predictors as additional features for ensembling has been extensively explored. Bao et al. [2009] formalized a framework called STREAM 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 the base recommenders. Intuitively, the ensemble should learn which performance predictors are adequate for each recommender score. This framework was further studied by Jahrer et al. [2010], who leveraged different pointwise ensemblers such as neural networks and bagged gradient boosted decision trees, resulting in improved performance. The second place solution in the Netflix competition, FWLS [Sill et al., 2009], leveraged additional features in a different manner, by making the Cartesian product of the score of base recommenders and of performance predictors as the input space for a linear regression ensembler. Fortes et al. [2017] compared different ensembling strategies (stacking, FWLS and STREAM) through several evaluation metrics to answer whether performance predictors are really useful for such hybridization approaches. They concluded that even though the ensemble is not optimized for metrics such as nDCG, the usage of performance predictors still proves to be beneficial.

The second part of this dissertation statement is *"Effective performance prediction of different base models* ***can be exploited to improve the results of their combination****"*. In both ad-hoc retrieval and recommendation, the state-of-the-art approach for doing such exploitation is to use the framework of simply adding M

performance prediction features to the ensemble input space, which is the framework described in Figure 1.1. As we discussed in the Chapter 1, this framework has the following limitations:

- The burden of learning the relationship between each performance predictor and model predictions is given to the ensemble which has the difficult task of comprehending how and which model prediction relates to each performance prediction feature.
- There are M performance predictions that are not directly related to the N models in the ensemble.

In order to overcome both limitations we propose to have one accurate performance prediction for each model in the ensemble, which results in the framework described in Figure 1.2.

2.5 Summary

Unlike approaches for performance prediction in recommender systems that create features to predict user performance for a system, we propose to use a new set of features called performance estimates, which we empirically show in our experiments that dismiss the need of handcrafting additional discriminative performance predictors. Moreover, we show that having one performance estimate for each model in the ensemble achieves state-of-the art performance in the recommendation task. Additionally, in the search scenario, we differ from other approaches to improve ad-hoc retrieval by having one machine-learned query performance predictor for each base model in the ensemble, which we show that achieves an accuracy bar for performance prediction that translates to statistical improvement on the ad-hoc retrieval task if used as additional features for learning to rank.

By using our proposed framework for exploiting accurate performance predictors we demonstrate, in the experiments conducted in this dissertation, significant improvements compared to the state-of-the-art baseline framework from the literature in two different tasks from information retrieval.

Chapter 3

Leveraging Performance Prediction for Ad-hoc Retrieval

Query performance prediction concerns the design of features that are indicative of the effectiveness of a given ranking model for a particular query in the absence of actual relevance information [Hauff, 2010]. Query performance predictors (QPP) have been categorized into two approaches, according to the available data for prediction: pre-retrieval and post-retrieval. For example, Averaged Inverse Document Frequency (AvIDF) is a pre-retrieval QPP that measures query specificity as the average IDF of all query terms. Intuitively, a ranking model might achieve higher effectiveness when query specificity is higher. Multiple pre-retrieval QPP have also been combined and shown to yield improved prediction accuracy [Hauff et al., 2009].

One key application of QPP is to weight the results from different ranking models according to the predicted effectiveness of each model on a per-query basis. For instance, QPP have been tested as triggering mechanisms of specialized ad-hoc retrieval models, such as query expansion [Cronen-Townsend et al., 2004], as well as to combine ad-hoc and diversification models [Santos et al., 2010]. More recently, Macdonald et al. [2012] investigated the usefulness of QPP as additional features for learning to rank via gradient boosted regression trees. Despite considerable effort on the subject, the promise of leveraging QPP for improving ad-hoc retrieval has not been entirely fulfilled. Raiber and Kurland [2014] argued that this is due to the inherent difficulty of the performance prediction task, which boils down to the standard relevance estimation task in information retrieval.

In this chapter, we investigate the limits of QPP for improving ad-hoc retrieval. Section 3.1 discusses the baseline strategy of learning to rank, as well as using additional QPP and ML-QPP. Section 3.2 introduces the experimental methodology used for our experiments and Section 3.3 evaluates the research questions. The results of this evaluation attest the effectiveness of machine-learned performance predictions for enhancing learning to rank for adhoc-retrieval, which has been a struggle in the area.

3.1 Learning to Rank with QPP

Raiber and Kurland [2014] provided a theoretical argument for the limited usefulness of QPP for ad-hoc retrieval, demonstrating the equivalence between predicting the performance of a query and estimating the relevance of documents for this query—arguably, the holy grail of information retrieval. To further illustrate their point, they empirically demonstrated the limited effectiveness of linearly fusing a set of ranking models weighted by various QPP, which could not consistently improve compared to the single best ranking model in the set alone. In this chapter, we revisit their investigation by simulating QPP with various levels of accuracy as input to a non-linear learning to rank approach.

3.1.1 Baseline Strategy

Without loss of generality, we assume a standard learning to rank setup and aim to learn a ranking function $f : X \to Y$, mapping the input space X onto the output space Y. Our input space comprises query–document pairs encoded as feature vectors $\vec{x} \in X$. In our experiments in Section 3.3, we consider a 46-dimensional space comprising standard query-dependent and query-independent ranking models from LETOR 4.0 [Qin and Liu, 2013] (e.g., TF-IDF, BM25, PageRank) as features in our baseline representation. For a full description of the models in LETOR 4.0 see Table A.1. Our output space comprises graded relevance labels $y \in Y$ assigned to each query–document pair. As a learning approach, following Macdonald et al. [2012], we use LambdaMART as a representative of the state-of-the-art. This baseline is displayed at the diagram from Figure 3.1.

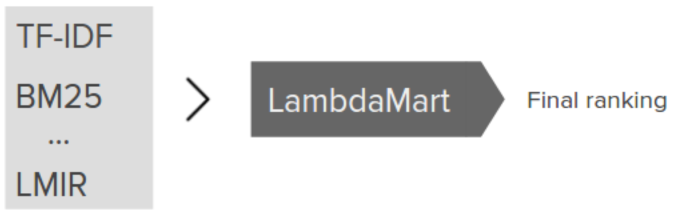


Figure 3.1: Baseline strategy for combining LETOR 4.0 ranking functions.

3.1.2 Leveraging QPP

To test the usefulness of QPP for ad-hoc retrieval, we leverage a total of 267 QPP from the literature, including pre-retrieval and post-retrieval ones, as summarized in Table 3.1. For a complete description, please refer to Appendix B or Hauff [2010]. QPP that generated 5 variants, such as AvIDF, are extracted from the 5 streams in each document (body, anchor, title, url and whole document). AvQC used [0.95, 0.90 0.85, 0.80] as similarity cutoffs (hence 4 features). AvLCH and AvWUP are similar to Av-Path, varying only the similarity metric between terms, which are Leacock-Chodorow Similarity and Wu-Palmer Similarity (provided by the WordNet-Similarity v1.04 package). Finally, post-retrieval QPP provide different summaries (sum, max, std, avg) for each of the 46 LETOR features based on their top 10 retrieved documents. This stronger baseline is displayed at the diagram from Figure 3.2.

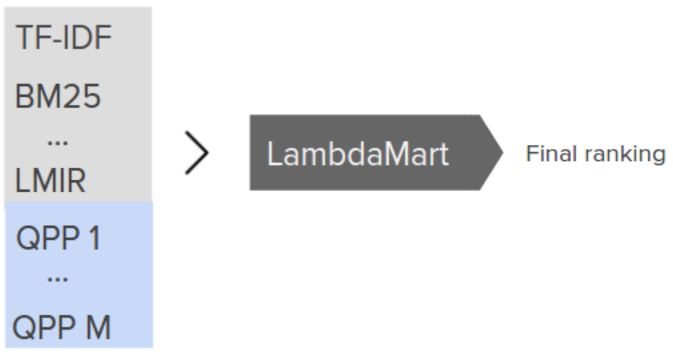


Figure 3.2: Baseline strategy that leverages query performance predictors for combining LETOR 4.0 ranking functions.

To leverage QPP, we extend the 46-dimensional input space X of the baseline learning to rank strategy described in Section 3.1.1 to encompass three groups of QPP from Table 3.1: pre-retrieval (+83 features), post-retrieval (+184 features), and all QPP (+267 features).

3.1.3 Leveraging ML-QPP

Hauff et al. [2009] proposed a machine-learned approach to combine multiple pre-retrieval QPP into a single, stronger QPP. Raiber and Kurland [2014] extended this approach to also combine post-retrieval QPP using Ranking SVMs. The intuition behind machine-learned query performance prediction is straightforward: predicting the performance of a ranking model for a query is better when multiple QPP are combined, as opposed to using a single QPP.

Category	**Stage**	**QPP**	**Total**
Specificity	pre-retrieval	MatchingDocs	1
Specificity	pre-retrieval	TokenCount	1
Specificity	pre-retrieval	TermCount	1
Specificity	pre-retrieval	AvQL	1
Specificity	pre-retrieval	AvIDF	5
Specificity	pre-retrieval	MaxIDF	5
Specificity	pre-retrieval	DevIDF	5
Specificity	pre-retrieval	AvICTF	5
Specificity	pre-retrieval	SCS	5
Specificity	pre-retrieval	AvSCQ	5
Specificity	pre-retrieval	SumSCQ	5
Specificity	pre-retrieval	MaxSCQ	5
Specificity	pre-retrieval	QS	5
Sensitivity	pre-retrieval	SumVAR	5
Sensitivity	pre-retrieval	AvVAR	5
Sensitivity	pre-retrieval	MaxVAR	5
Ambiguity	pre-retrieval	AvQC	4
Ambiguity	pre-retrieval	AvP	1
Ambiguity	pre-retrieval	AvNP	1
Term Relatedness	pre-retrieval	AvPMI	5
Term Relatedness	pre-retrieval	MaxPMI	5
Term Relatedness	pre-retrieval	AvPath	1
Term Relatedness	pre-retrieval	AvLCH	1
Term Relatedness	pre-retrieval	AvWUP	1
Retrieval Summary	post-retrieval	SumLETOR	46
Retrieval Summary	post-retrieval	MaxLETOR	46
Retrieval Summary	post-retrieval	StdLETOR	46
Retrieval Summary	post-retrieval	AvLETOR	46
TOTAL			267

Table 3.1: Standard QPP used in our experiments [Hauff, 2010].

Each ML-QPP combines multiple QPP to predict the effectiveness of a single ranking model, as measured by some standard ranking evaluation metric, such as average precision (AP), reciprocal rank (RR), or normalized discounted cumulative gain (nDCG). Formally, given a ranking model m and a target evaluation metric e, we aim to learn a regression function $ML-QPP_m^e : V \to W$. Each learning instance $\vec{v} \in V$ represents a single query q, encoded as a k-dimensional feature vector $\vec{v} = [QPP_1, \ldots, QPP_k]$. Each feature QPP_i is computed by a standard QPP given the query q itself (for pre-retrieval QPP) or the ranking R produced for this query by model m (for post-retrieval QPP). In our investigation, we set $k = 267$ and use all QPP in Table 3.1 as features for learning one $ML-QPP_m^e$ per ranking model m and evaluation metric e. To this end, each learning label $w \in W$ denotes the actual effectiveness of the ranking R produced by model m, as measured by metric e given the ground-truth associated with query q. In Section 3.3, we experiment with several regressors for learning effective ML-QPP. This procedure is summarized in the diagram from Figure 3.3.

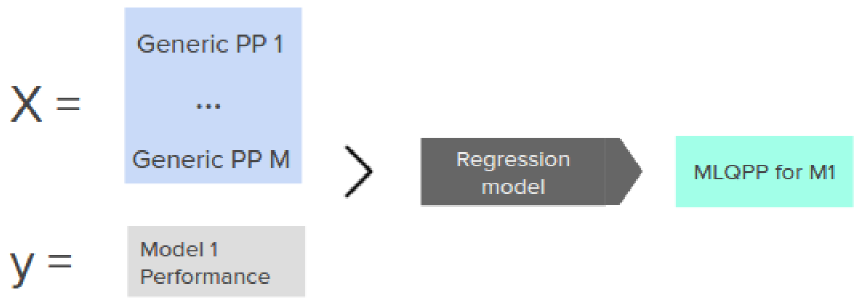


Figure 3.3: Overview of our proposed machine-learned query performance predictor (ML-QPP). The input space for a regression model are the 267 features described in Table 3.1, the target is the performance (as measured by nDCG@20 for example) for a specific model (TF-IDF for instance). The prediction of this regression model is a single ML-QPP which is used as part of the input space for the ensembling model.

To leverage ML-QPP, our input space X in Section 3.1.1 is extended with 46 $ML-QPP_m^e$, one per each of the 46 ranking models m in the baseline representation. While exploiting QPP in the ensemble adds the time cost of calculating all the query performance prediction features, ML-QPP increases this time cost only by the corresponding model prediction time given a new instance – the regression model for generating ML-QPP can be trained offline.

Because ML-QPP are defined at the query level, all learning instances for the learning to rank model associated with a given query q will have the same values for all documents. Motivated by this, we provide an alternative, document-dependent ML-QPP, by multiplying each $ML-QPP_m^e$ by the score produced by ranking model m for each query–document pair. This weighted variant further increases the sensitivity of our approach by boosting relevance estimates produced by ranking models with a high predicted performance. The unweighted and weighted variations of our approach are displayed at the diagram from Figure 3.4.

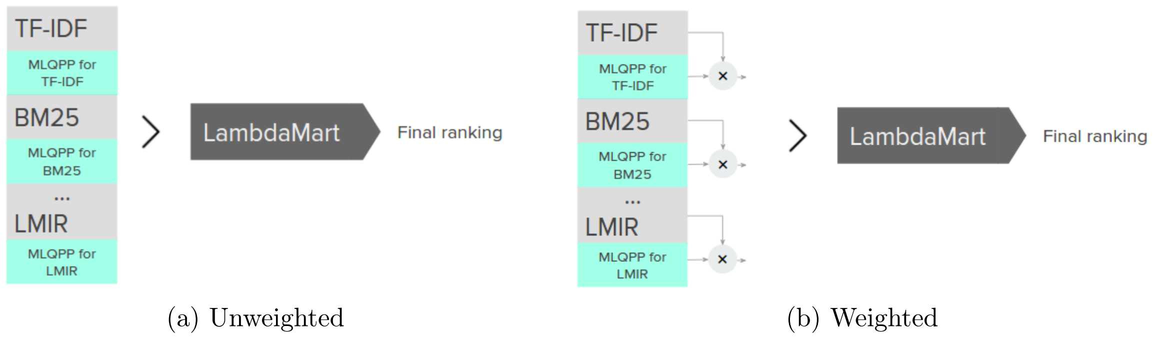


Figure 3.4: The proposed framework for exploiting performance predictors instantiated in the ad-hoc retrieval scenario, using ML-QPP as the approach for performance prediction.

3.2 Experimental Setup

To assess the usefulness of query performance prediction for ad-hoc retrieval, we address the following research questions:

Q1. What are the limits of QPP for improving ad-hoc retrieval?

Q2. Can we improve ad-hoc retrieval with ML-QPP?

As previously described, our evaluation uses the publicly available LETOR 4.0 dataset [Qin and Liu, 2013]. It uses the GOV2 document corpus and the query set from the TREC 2008 Million Query track. As a further preprocessing step, we removed queries with no relevant documents in the ground-truth, reducing the total number of queries from 784 to 564. For learning ML-QPP, we use several regressors: Random Forest (RF), Extreme Gradient Boosting (XGB), Gradient Boosting (GB), AdaBoost (ADA), Neural Network (MLP), Support Vector Machines (SVR) and Linear Regression (LR). The regressor used to learn each ML-QPP in our experiments is shown in parentheses (e.g., ML-QPP (RF) was generated using Random Forest). We used scikit-learn v0.19.1 implementation, except for XGB, which comes from the Python package xgboost v0.6a2. For learning to rank, we used LambdaMART as implemented in RankLib v2.8. After an initial exploration of their hyperparameter spaces we chose default settings for LambdaMART and small variations for the regressors which are available online.[1]

LambdaMART is trained to optimize nDCG@20 through a five-fold cross validation using the partitions provided by LETOR. To make sure ML-QPP are not overfitted to the training data, we perform a three-fold cross validation inside each of the five LETOR partitions. In each round of this inner cross-validation, two-thirds of the data are used to train the regressors and the remaining third is used to calculate the predictions that will be leveraged as additional features by LambdaMART. In addition to reporting retrieval effectiveness via nDCG@20, we report prediction accuracy using mean absolute error (MAE).[2] For a fair comparison, we normalize all predictions in the range 0–1 using min-max normalization to lie in the same range as the target evaluation metric e. In all experiments, we use three different metrics as targets to be predicted: normalized discounted cumulative gain (nDCG), average precision (AP) and reciprocal rank (RR). Both retrieval as well as prediction accuracy improvements are statistically validated through paired two-sided Student's t-tests using a 95% confidence level.

3.3 Experimental Evaluation

In the following, we address both questions posed in Section 3.2 to assess the usefulness of QPP and ML-QPP for ad-hoc retrieval.

3.3.1 Usefulness of QPP

To address *Q1*, we assess how much query performance predictors can improve ad-hoc retrieval when used as additional features for learning to rank. To this end, we perform a simulation using QPP of various levels of prediction accuracy, from perfect to random. In particular, each simulated α-QPP^e_m incorporates randomness proportional to parameter α. Precisely, we have:

$$\alpha\text{–}QPP^e_m = (1-\alpha)\,w + \alpha\,\tilde{w}, \tag{3.1}$$

[1]https://github.com/Guzpenha/performance-prediction-for-enhancing-ensemble-learning

[2]Hauff et al. [2009] have shown that error metrics such as MAE are better indicators of prediction accuracy in cross-validation settings compared to the more typical use of correlations between predicted and actual performance.

where w denotes the actual effectiveness of ranking R produced by model m, as measured by metric e given the ground-truth for query q, and $\tilde{w} \sim U(0, 1)$ denotes uniformly sampled random noise.

Figure 3.5 shows the nDCG@20 attained by LambdaMART when using the 46-dimensional baseline representation in Section 3.1.1 augmented with 46 α-QPP^e_m, one per feature m, for $\alpha \in [0, 1]$. The figure comprises three plots corresponding to $e \in \{AP, nDCG, RR\}$. Each plot also includes three horizontal lines from bottom to top contrasting the effectiveness of the baseline representation (1) without augmentation, (2) augmented with all 267 QPP from Table 3.1, and (3) augmented with the 10 best performing QPP from Table 3.1.

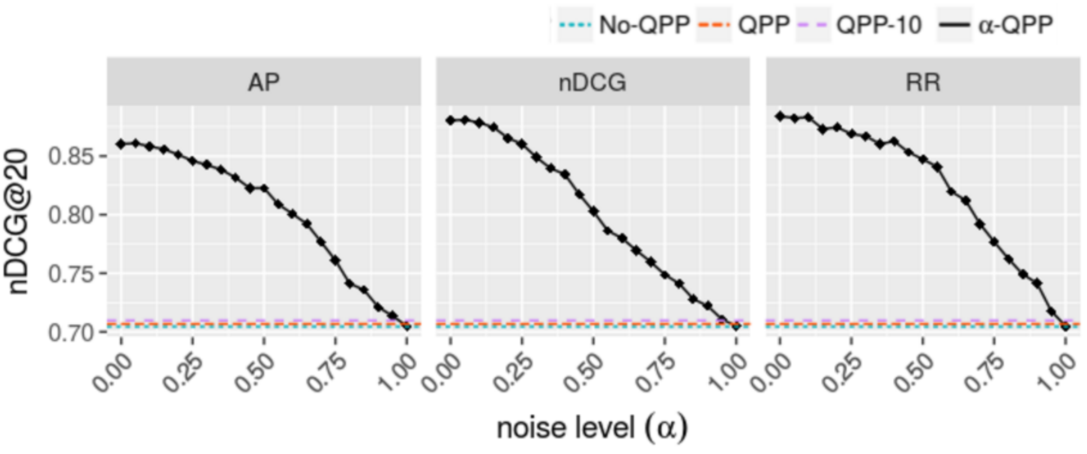


Figure 3.5: nDCG@20 attained by machined-learned ranking models augmented with performance predictors ($\alpha - QPP^e_m$) targeting AP, nDCG, or RR with various amounts of noise α. Note that $\alpha - QPP^e_m$ with $\alpha = 0$ denotes a perfect performance predictor whereas $\alpha = 1$ a completly random predictor. Horizontal lines from bottom to top denote machine-learned ranking models with no prediction augmentation (No-QPP, our baseline strategy), augmentation with all 267 predictors from Table 3.1 (QPP, stronger baseline strategy, which leverages QPP), or with the top 10 predictors from Table 3.1 (QPP-10).

From Figure 3.5, we first observe that query performance prediction has the potential to substantially improve upon our baseline, non-augmented representation (No-QPP), as demonstrated by the results attained by α-QPP^e_m with $\alpha = 0$ (i.e., perfect predictions). Nonetheless, existing QPP fall short in realizing this potential, with only marginal gains compared to the baseline representation. Indeed, QPP delivers an ad-hoc retrieval performance comparable to the almost random α-QPP^e_m with $\alpha \approx 0.98$. One way of improving this result could be to select from the complete set of 267 QPP from Table 3.1 only the most accurate ones. However, even when the top 10 QPP are selected—the QPP-10 variant—no noticeable improvement is observed. Recalling

Q1, this empirical result confirms the theoretical findings of Raiber and Kurland [2014] about the inherent difficulty of query performance prediction and reveals an accuracy bar that must be overcome for QPP to be useful for ad-hoc retrieval.

3.3.2 Usefulness of ML-QPP

The results in Section 3.3.1 showed that current QPP cannot significantly improve a machine-learned stack of ranking models, even when only the most accurate QPP are selected. Instead of selecting the most accurate QPP, our proposed ML-QPP aim to ensemble multiple QPP into a single, more accurate query performance predictor. To address question *Q2*, we assess both the prediction accuracy of ML-QPP as well as their usefulness for ad-hoc retrieval.

3.3.2.1 Performance Prediction Accuracy

To assess query performance prediction accuracy, we contrast the predicted vs. actual performance of each ranking model m measured by evaluation metric e over our fixed set of 564 queries. Following standard practice [Hauff et al., 2009], we consider mean absolute error (MAE) as a measure of accuracy. Since we have a total of 46 ranking models whose performance must be predicted, we report the average MAE across all models for each of three target metrics, namely, AP, nDCG, and RR.

Figure 3.6 summarizes the results of this investigation, contrasting the 10 most accurate QPP from Table 3.1 against ML-QPP instantiations learned by different regressors, as discussed in Section 3.2. As shown in the figure, most of the considered regressors are able to improve prediction accuracy when compared with the best performing solo QPP, with ensembling regressors RF, XGB, and GB delivering particularly accurate predictions. The best average improvements were obtained by using RF to predict all three ranking evaluation metrics: AP (11.27%), nDCG (1.6%), and RR (2.55%). Moreover, ML-QPP produce significantly improved performance predictions compared to the single best QPP in 75% of all 46×3 tested combinations of ranking model m and evaluation metric e.

3.3.2.2 Ad-hoc Retrieval Effectiveness

Results in the previous section demonstrated the accuracy of ML-QPP for query performance prediction. However, as pointed out by Raiber and Kurland [2014], the expectation that improved performance prediction would translate to improved retrieval has not yet been realized. To address Q2, we assess the usefulness of ML-QPP when

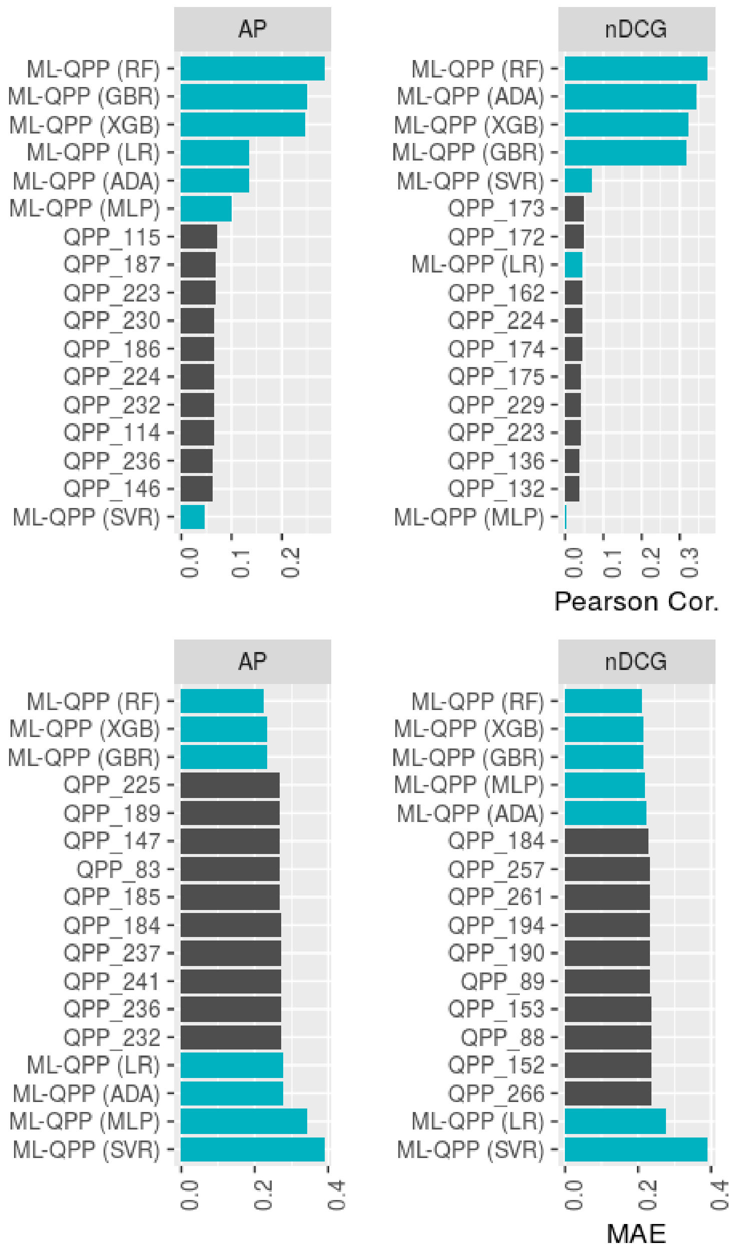


Figure 3.6: Prediction accuracy (measured by MAE and Pearson Correlation averaged across 46 ranking models) of ML-QPP using various regressors in contrast to the 10 most accurate QPP from Table 3.1.

leveraged as additional features for learning to rank. Table 3.2 shows the results of this investigation in terms of nDCG@20. Similarly to Figure 3.5, we compare LambdaMART models built using different input spaces: a baseline space comprising 46 LETOR features as well as augmented spaces adding either QPP or ML-QPP variants as extra features. For QPP, we consider three variants: (1) pre-retrieval QPP (+83 features), (2) post-retrieval (+184 features), and (3) all QPP (+267 features). For ML-QPP, as discussed in Section 3.1.3, we include results using multiple regressors under two variants: (1) unweighted, computed at the query-level, and (2) weighted, computed at the document-level.

Table 3.2 shows that using QPP as additional features is not enough to significantly improve over the baseline representation using LETOR features. This observation is consistent regardless of which variant of QPP (pre-retrieval, post-retrieval, all) is used. On the other hand, when using the proposed ML-QPP as additional features, we observe significant improvements in many settings, notably for ensemble-based regressors: XGB, RF, and GB. Recalling *Q2*, these results attest the usefulness of ML-QPP compared to existing QPP for ad-hoc retrieval, with significant improvements in many cases. Contrasting the two ML-QPP variants, while there is no clear winner, the weighted variant seems particularly effective for XGB and RF, with the weighted variant of $ML - QPP^{RR}(RF)$ delivering the overall best nDCG@20 in our investigation: 0.7237. Nevertheless, this result is still far behind the theoretical best result attainable should we have perfect predictors (0.8840, as given by α-QPP$^{RR}(RF)$ in Figure 3.5 with $\alpha = 0$), which points out interesting directions for further research, as discussed in the next section.

3.4 Summary

In this chapter, we addressed the claim of our dissertation statement, by showing that effective performance prediction of different base models (ranking functions in this case) can be exploited to improve the results of their combination (learning to rank) in the context of query performance predictors and ad-hoc retrieval. While past theoretical results had suggested an inherent limitation of QPP for ad-hoc retrieval, we empirically demonstrated that such a limitation is rather due to insufficient accuracy of existing pre- and post-retrieval QPP. By simulating predictors with various levels of noise, we showed that there is a prediction accuracy bar that must be overcome for QPP to become useful features. As a proof-of-concept, we demonstrated the usefulness of ML-QPP, which ensemble multiple QPP into stronger predictors. Despite the positive

Features	**nDCG@20**	
LETOR features	0.7045	
	Unweighted	Weighted
$+QPP(pre)$	0.7067	
$+QPP(post)$	0.7053	
$+QPP(all)$	0.7067	
$+ML-QPP^{AP}(LR)$	**0.7158**	0.7109
$+ML-QPP^{AP}(MLP)$	0.7083	0.7116
$+ML-QPP^{AP}(SVR)$	0.7062	0.7064
$+ML-QPP^{AP}(XGB)$	**0.7131**	**0.7142**
$+ML-QPP^{AP}(RF)$	0.7105	**0.7173**
$+ML-QPP^{AP}(GB)$	0.7137	0.7133
$+ML-QPP^{AP}(ADA)$	0.7118	0.7104
$+ML-QPP^{nDCG}(LR)$	0.7134	0.7133
$+ML-QPP^{nDCG}(MLP)$	0.7109	0.7063
$+ML-QPP^{nDCG}(SVR)$	0.7062	0.7077
$+ML-QPP^{nDCG}(XGB)$	0.7096	**0.7167**
$+ML-QPP^{nDCG}(RF)$	0.7084	0.7151
$+ML-QPP^{nDCG}(GB)$	**0.7178**	**0.7162**
$+ML-QPP^{nDCG}(ADA)$	0.7102	0.7097
$+ML-QPP^{RR}(LR)$	0.7114	0.7077
$+ML-QPP^{RR}(MLP)$	**0.7130**	0.7108
$+ML-QPP^{RR}(SVR)$	0.7070	0.6996
$+ML-QPP^{RR}(XGB)$	0.7064	**0.7184**
$+ML-QPP^{RR}(RF)$	0.7079	**0.7237**
$+ML-QPP^{RR}(GB)$	0.7104	0.7067
$+ML-QPP^{RR}(ADA)$	0.7125	0.7058

Table 3.2: nDCG@20 attained by machine-learned ranking models augmented with existing QPP as well as our proposed ML-QPP. Significant improvements compared to the baseline representation using LETOR features are in bold.

results, our simulation also showed that there is much room for improvement, provided that even more accurate predictors can be built.

Chapter 4

Leveraging Performance Prediction for Recommender Systems

Recommender systems aim to suggest items, e.g. movies, books and places, 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 [Aggarwal, 2016b].

Hybrid recommenders are designed to leverage the power of different base recommenders in order to make more robust recommendations [Aggarwal, 2016a]. 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. It has led to the creation of state-of-the-art machine learning models such as extreme gradient boosting [Chen and Guestrin, 2016].

In addition to the scores produced by different recommenders, recommendation ensembling has been shown to benefit from leveraging performance predictors, i.e., features that are indicative of the performance of each base recommender for the target user [Bao et al., 2009; Bellogín et al., 2011]. As illustrated in Figure 4.1, performance predictors have been traditionally categorized as pre-retrieval or post-retrieval, depending on whether they are calculated based on some property of the target user or of the set of items recommended for the user, respectively. For instance, Bellogín et al. [2011] adapted the well-known clarity score [Cronen-Townsend et al., 2002], originally formu-

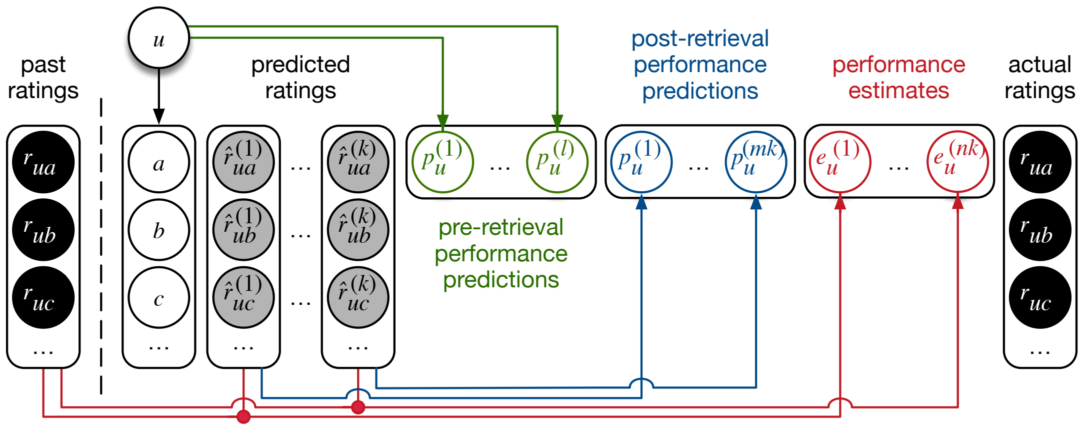


Figure 4.1: At recommendation time, an ensemble may combine scores from multiple base recommenders (gray circles) to learn the actual ratings (black circles on the right) that would be assigned by the target user to a set of recommended items (white circles). Current approaches augment the ensemble to leverage performance predictions based on the user (green outlined circles) or the set of recommended items (blue outlined circles). 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).

lated as a measure of query ambiguity in adhoc search, to measure the coherence of the user's historical ratings in a recommendation setting. They showed that the adapted predictors have a moderate correlation with the performance of some collaborative recommenders. However, 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 user feature relates to the performance of each recommender in the ensemble. Moreover, performance predictors are often better indicators of the inherent difficulty of a user (regardless of any particular recommender) rather than of the relative effectiveness of different recommenders [Raiber and Kurland, 2014].

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. As also illustrated in Figure 4.1, we introduce performance estimates for a given recommender as the outcome of a standard evaluation metric given the user's historical ratings and the corresponding predictions from the recommender. 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.

The remainder of this chapter describes the approach for predicting the performance of users in recommender systems as well as how to effectively use this information to enhance recommendation effectiveness. In particular, Section 4.1 formalizes the approach of *performance estimation*, how to use it in ensembles of recommender systems and the variants of this set of features, Section 4.2 describes the experimental methodology of our experiments and in Section 4.3 we evaluate our approach in light of the research questions posed. Comprehensive experiments using real-world datasets of four different domains demonstrate the effectiveness of performance estimates when combined with plain recommender 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 also when leveraged by pairwise and listwise ensemblers.

4.1 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 a lot 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, performance predictors are 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 [2014] demonstrated that accurate performance prediction boils down to accurate relevance estimation. Second, even if performance predictors were accurate enough, 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.

4.1.1 Estimating Performance

Let U, I, R, and T denote a set of users, items, possible rating values, and discrete rating timestamps, respectively. Moreover, let $D_{\kappa_1}^{\kappa_2} = \{(u, i, r, t) \mid u \in U,\ i \in I,\ r \in R,\ t \in T,\ \kappa_1 < t \leq \kappa_2\}$ be the set of ratings recorded in the left-open time interval bounded by timestamps κ_1 and κ_2. A recommender system can be defined as a function $s(u, i) : U \times I \to R$. For a user u, this function produces a recommendation list $L_u^s = sort_{s(u,i)}\{i \in I\}$ as a permutation of all available items I. The performance of the system can be assessed in different ways. For a recommendation produced at time τ, the true performance TP of the system is given by:

$$TP \equiv \Delta(u, L_u^s, D_\tau^{\tau+1}), \tag{4.1}$$

where Δ 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 L_u^s displayed at time τ, no user feedback will be available until time $\tau + 1$. As a result, at time τ, system performance can only be approximated. A performance predictor PP computes such an approximation based on characteristics of the target user u (in the case of pre-retrieval predictors), or of the produced recommendation list L_u^s (in the case of post-retrieval predictors). Formally, we have:

$$PP \equiv \Pi(u, L_u^s), \tag{4.2}$$

where Π could compute, for instance, the amount of ratings in the historical profile of user u [Sill et al., 2009] or the deviation of the recommended list L_u^s from a random or most-popular list of items [Bellogín et al., 2011].

Given the considerable 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 PE according to:

$$PE \equiv \Delta(u, L_u^s, D_{\kappa_1}^{\kappa_2}), \tag{4.3}$$

where $0 \leq \kappa_1 < \kappa_2 \leq \tau$ and, similarly to Equation 4.1, Δ 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. The approach is also described in the diagram from Figure 4.2.

4.1.2 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 4.1, in the first layer of our proposed architecture, for each user–item pair $\langle u, i\rangle$, we are given the scores of k base recommenders (RS) previously trained on historical ratings $D_{\kappa_1}^{\kappa_2}$, with $0 \leq \kappa_1 < \kappa_2 \leq \tau$, where τ once again denotes the recommendation time. In addition,

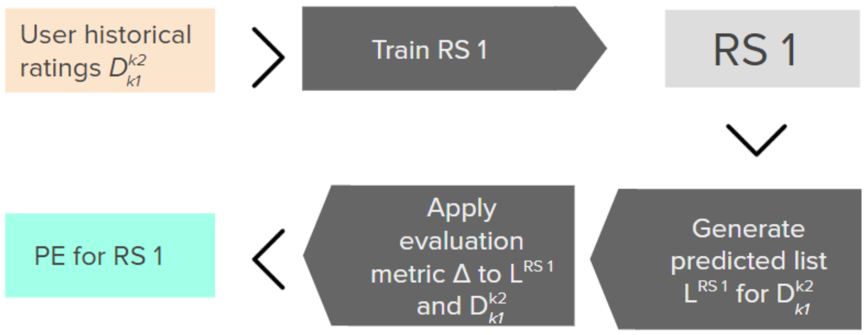

User historical ratings D_{k1}^{k2}
Train RS 1
RS 1
Generate predicted list $L^{RS\,1}$ for D_{k1}^{k2}
Apply evaluation metric Δ to $L^{RS\,1}$ and D_{k1}^{k2}
PE for RS 1

Figure 4.2: Fluxogram for estimating the performance of recommender systems for users using their historical set of ratings.

we are also given l pre-retrieval and mk post-retrieval performance predictors (here collectively referred to as PP) and nk performance estimates (PE) as meta-features for the ensemble. In our analysis in Section 4.3, we experiment with one performance estimate per base recommender, which effectively makes $n = 1$. Notice however that the total number of performance predictors $l + mk$ does not necessarily correspond to that of base recommenders, as they are engineered independently.

In the second layer of our architecture, our goal is to learn a hypothesis function $h : X \to Y$ mapping the input space X onto the output space Y. Our input space X comprises learning instances of the form $\vec{x} = \Phi(u, i)$, where Φ is a meta-feature extractor defined over the user-item pair $\langle u, i \rangle$. In practice, we could represent each learning instance $\vec{x}$ as a $(k + (l + mk) + nk)$-dimensional vector, such that $\vec{x} = (\{RS_j\}_{j=1}^{k}, \{PP_j\}_{j=1}^{l+mk}, \{PE_j\}_{j=1}^{nk})$, where RS_j, PP_j, and PE_j denote the j-th recommender score, performance predictor, and performance estimate, respectively.

However, in order to evaluate the effectiveness of each set of features separately, we use two baselines (recommender systems only, recommender systems and performance predictors) and our proposed approach (recommender systems and performance estimates). The first baseline is equivalent to stacking, where the input space is composed by the predicted ratings from the k recommender systems in the ensemble, $\vec{x} = (\{RS_j\}_{j=1}^{k})$, described visually in Figure 4.3.

Second, a stronger baseline, equivalent to the state-of-the-art STREAM [Bao et al., 2009] for exploiting performance predictions in ensembles is equivalent to setting the input space as $\vec{x} = (\{RS_j\}_{j=1}^{k}, \{PP_j\}_{j=1}^{l+mk})$, where the $l+mk$ performance predictors are appended to the input space, displayed in Figure 4.4.

Finally, the input space for our proposed set of features, which has the same dimensionality of the recommender systems being combined (k), is defined as $\vec{x} = (\{RS_j\}_{j=1}^{k}, \{PE_j\}_{j=1}^{nk})$, described in the diagram from Figure 4.5.

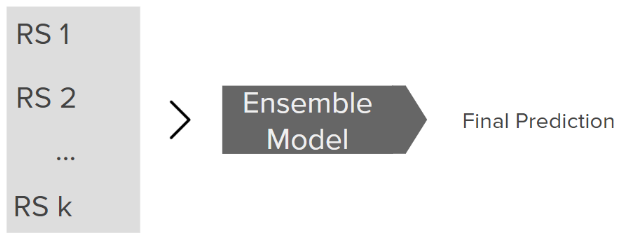


Figure 4.3: The first baseline for combing multiple recommender systems using machine learning, called *stacking* in its basic formulation and referred as *learning to rank* when applied to generating lists of ranked items instead of pointwise regression. We call this baseline as **RS**, as we only use the recommender systems outputs.

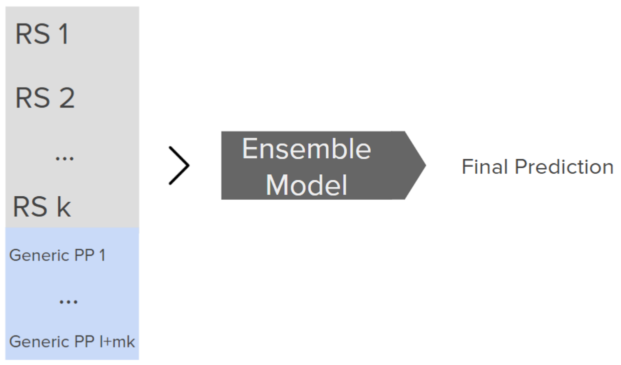


Figure 4.4: The state-of-the-art baseline for combing multiple recommender systems using machine learning, called STREAM by Bao et al. [2009]'s formalization of the framework. Performance predictors are used as additional features in the input space for the ensemble. We call this baseline as **PP**, referring to the fact that we use both recommender systems predictions, RS, and PP.

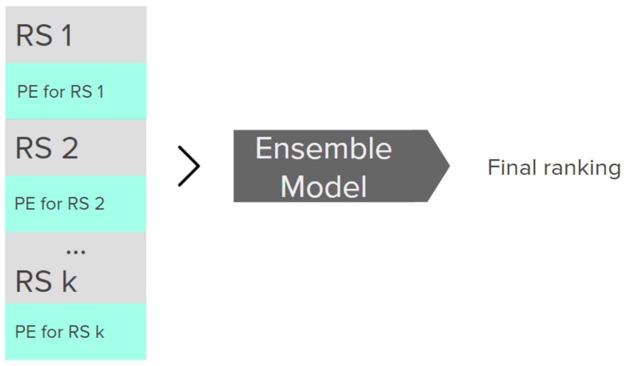


Figure 4.5: The proposed framework for exploiting Performance Predictions in the combination of recommender systems. Unlike previous approaches, we have one ***estimation*** of performance for each recommender in the ensemble. We refer to our approach as **PE**.

In turn, our output space Y, in its most basic form, equates to the set of possible rating values R. In Section 4.3, we experiment with representative learning to rank approaches from the pointwise, pairwise, and listwise families,[1] encompassing both linear and non-linear hypotheses.

4.1.3 Performance Estimation Variants

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 compute two variants of our performance estimates. Our first variant is estimated using the same ratings available as training to the base recommender, i.e., both base recommender training and performance estimates use $D_{\kappa_1}^{\kappa_2}$, with $0 \leq \kappa_1 < \kappa_2 \leq \tau$. In turn, our second variant is estimated using validation data set aside from the base recommender training, in the hope of improving its generalization capabilities to unseen test data. In this case, base recommender training uses $D_{\kappa_1}^{\kappa_2}$, whereas performance estimates use $D_{\kappa_3}^{\kappa_4}$, with $0 \leq \kappa_1 < \kappa_2 < \kappa_3 < \kappa_4 \leq \tau$.

In addition to variants for computing performance estimates, we also propose two variants for leveraging them. In particular, because 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 PE values for all items $i \in I$. Such user-dependent, item-agnostic meta-features bear resemblance to query-dependent, document-agnostic features, which have been shown to be useful for learning non-linear hypotheses, such as boosted regression trees [Macdonald et al., 2012]. Nevertheless, to provide alternative, item-dependent performance estimates, we consider a weighted variant of the raw PE meta-features defined in Equation 4.3, by multiplying them by the corresponding recommender score $s(u, i)$. Formally, we have:

$$PE_w \equiv PE \times s(u, i), \tag{4.4}$$

where PE (a function of user u and recommender s) is given by Equation 4.3 and $s(u, i)$ denotes the score produced by recommender s for the $\langle u, i \rangle$ pair. If the unweighted variant from Equation 4.3 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 4.4 further increases the sensitivity of our approach by automatically boosting scores produced by recommenders with a high performance estimate. The weighted variation is displayed at the diagram from Figure 4.6. In the

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

following sections, we assess the effectiveness of all variants of our proposed performance estimates for ensembling recommendations.

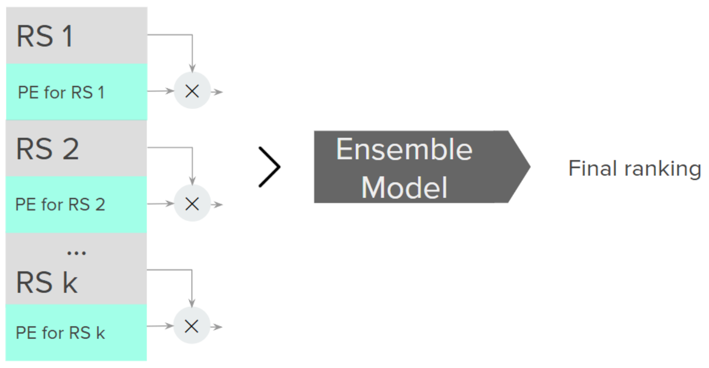


Figure 4.6: The weighted variation of our proposed approach for using performance estimates in ensembles of recommender systems.

4.2 Experimental Setup

In this section, we detail the setup that supports our investigations in Section 4.3. We aim to answer the following research questions:

Q0. How effective are PE for performance prediction?

Q1. How effective are PE for ensembling recommenders?

Q2. How robust are PE to the choice of ensembler?

Q3. How complementary are PE to PP?

Q4. How generalizable are PE to unseen data?

Q5. How do error and ranking-based metrics compare for PE?

4.2.1 Datasets

Our experiments use two publicly available datasets from four different domains: Yelp,[2] for point-of-interest recommendation, and Amazon,[3] for book, movie, and electronics recommendation. In common, these datasets provide large-scale timestamped rating

[2]https://www.yelp.com/dataset/challenge
[3]http://jmcauley.ucsd.edu/data/amazon/

data, which allows for a more realistic evaluation procedure 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%), ensembling training (next 30%) and test (last 10%), as described in Figure 4.7.

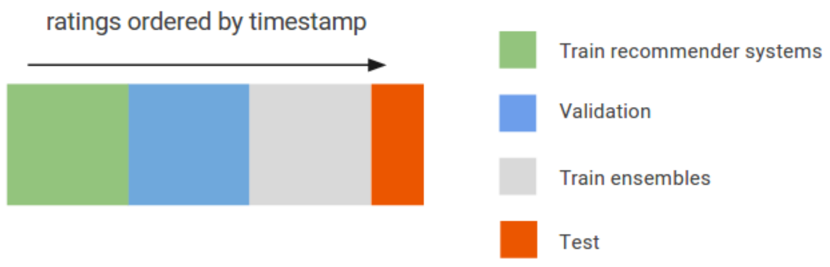


Figure 4.7: The data division used in our evaluation procedure. Ratings cutting points are based on the ordered ratings *timestamp*. The validation set is used to evaluate the generalization power of performance estimates, in order to address *Q4*, and could be dismissed in a production scenario.

Note that we separate training of base recommenders and ensemblers to make sure ensemblers will not build upon overfitted recommenders. Likewise, we set aside validation data to address *Q4*, which concerns the generalization power of PE to unseen data. 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 4.1.

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 Movies	3,266	61,837	245,707	0.001 %
Amazon Electronics	16,814	86,822	337,158	0.0002 %

Table 4.1: Statistics of the four datasets used in our evaluation after filtering users without ratings in all four partitions.

4.2.2 Performance Predictors and Estimates

We implemented a total of 14 performance predictors previously proposed in the literature, as summarized in Table 4.2. For computing performance estimates, we chose

[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 improving ensembles for personalized recommendation scenarios.

RMSE and nDCG as representative of error and ranking-based evaluation metrics, respectively.[6] Unless othwerwise stated, we compute PE using RMSE on the base recommender's validation data. Investigations of the impact of the source of estimation data and the evaluation metric used for estimation are discussed in Sections 4.3.5 and 4.3.6, respectively.

PP	**Description**	**Ref.**
PP_1	The log of the number of distinct ratings dates	[Sill et al., 2009]
PP_2	The log of the number of user ratings	[Sill et al., 2009]
PP_3	The standard deviation of the user ratings	[Sill et al., 2009]
PP_4	Regularized mean support for the user items	[Sill et al., 2009]
PP_5	User support: number of ratings	[Jahrer et al., 2010]
PP_6	Abnormality	[Gras et al., 2015]
PP_7	AbnormalityCR	[Gras et al., 2015]
PP_8	User average rating value	[Griffith et al., 2012]
PP_9	User standard deviation of rating values	[Griffith et al., 2012]
PP_{10}	Average number of ratings for the user items	[Griffith et al., 2012]
PP_{11}	Average of ratings from items rated by the user	[Griffith et al., 2012]
PP_{12}	Item support: number of ratings	[Griffith et al., 2012]
PP_{13}	Average rating value of item	[Jahrer et al., 2010]
PP_{14}	Item standard deviation of rating values	[Jahrer et al., 2010]

Table 4.2: Performance predictors used for experiments.

4.2.3 Base Recommenders and Ensemblers

To test our approach, we produce ensembles of the following nine classical collaborative recommenders from the literature:

Basic 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 simple item-based nearest-neighbor recommender [Koren, 2010, Equation (3)]. A variation that takes into account the mean rating of each user called KNNWithMeans, and a final variation that has no

[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.

bias or user mean in the prediction formula called KNNBasic are also used. Co-Clustering [George and Merugu, 2005] also uses similarity measurements between users and between items.

Latent factor models. SVD denotes the matrix factorization algorithm described by Koren et al. [2009], which is closely related to singular value decomposition, hence the name. NMF stands for non-negative matrix factorization [Luo et al., 2014], which is similar to SVD with non-negative user and item factors.

For base recommenders, we used the implementations in Surprise v1.0.5.[7] To speed up hyperparameter tuning, we performed a randomized search by sampling five times within the domain defined for each hyperparameter [Bergstra and Bengio, 2012]. This process resulted in the configurations described in Table 4.3. Along with the hyperparameter-free recommenders $NormalPredictor$ and $DebiasedAverage$, we produced a total of $6 \times 5 + 2 = 32$ recommenders for ensembling.

As 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 dissertation as well as their implementations are publicly available.[10]

4.2.4 Evaluation Procedure

We evaluate all ensembles in a top-20 recommendation task. To this end, we report nDCG@20 on the test partition of each dataset. Following Lopes et al. [2016], 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. [2010]. According to this personalized notion of

[7]`http://surpriselib.com/`
[8]`http://scikit-learn.org/`
[9]`https://sourceforge.net/p/lemur/wiki/RankLib/`
[10]`https://github.com/Guzpenha/performance-prediction-for-enhancing-ensemble-learning`

Base recommender	**Configurations**
$KNN_{A1}, KNN_{B1}, KNN_{C1}$	k = 33
$KNN_{A2}, KNN_{B2}, KNN_{C2}$	k = 28
$KNN_{A3}, KNN_{B3}, KNN_{C3}$	k = 29
$KNN_{A4}, KNN_{B4}, KNN_{C4}$	k = 20
$KNN_{A5}, KNN_{B5}, KNN_{C5}$	k = 14
SVD_1	lr = 0.0048, reg = 0.0443
SVD_2	lr = 0.0059, reg = 0.0547
SVD_3	lr = 0.0057, reg = 0.0625
SVD_4	lr = 0.0076, reg = 0.0648
SVD_5	lr = 0.0090, reg = 0.0662
NMF_1	factors = 66, epochs = 54
NMF_2	factors = 56, epochs = 65
NMF_3	factors = 58, epochs = 73
NMF_4	factors = 40, epochs = 75
NMF_5	factors = 28, epochs = 76
$CoClustering_1$	uc = 3, ic = 3, epochs = 37
$CoClustering_2$	uc = 3, ic = 3, epochs = 48
$CoClustering_3$	uc = 3, ic = 4, epochs = 40
$CoClustering_4$	uc = 2, ic = 4, epochs = 41
$CoClustering_5$	uc = 1, ic = 4, epochs = 40
$NormalPredictor$	-
$DebiasedAverage$	-

Table 4.3: Configurations of base recommenders obtained via random search within the range of each of their hyperparameters. KNN_A, KNN_B, and KNN_C denote KNNBaseline, KNNBasic, and KNNWithMeans, respectively.

relevance, an item is considered highly relevant if rated above average by the target user, somewhat relevant if rated 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.

4.3 Experimental Evaluation

In this section, we empirically evaluate our approach in light of the research questions posed in Section 4.2.

4.3.1 Performance Prediction Effectiveness

To address *Q0*, we evaluate our proposed approach in the performance prediction task. Formally, we define the task as predicting users nDCG@20 given a recommender system. Therefore, the evaluation metric Δ for performance prediction is nDCG@20, for all 32 recommender systems, given access only to the training set of ratings. To this end, we compare the RMSE of PE (our proposed approach) against the performance predictors (PP) described in Table 4.2. In order to make the comparison fair, we normalize PP in the range of nDCG using min/max transformation.

Figure 4.8 summarizes the results of this investigation. We observe that for all datasets and recommender systems PE outperforms the best PP, with statistical significance using Student's paired t-test (confidence level of 0.95), with gains up to 6.4%. The results attest the effectiveness of performance estimation at the performance prediction task, showing promising potential to enhance the ensemble of recommender systems, which we evaluate in the next section.

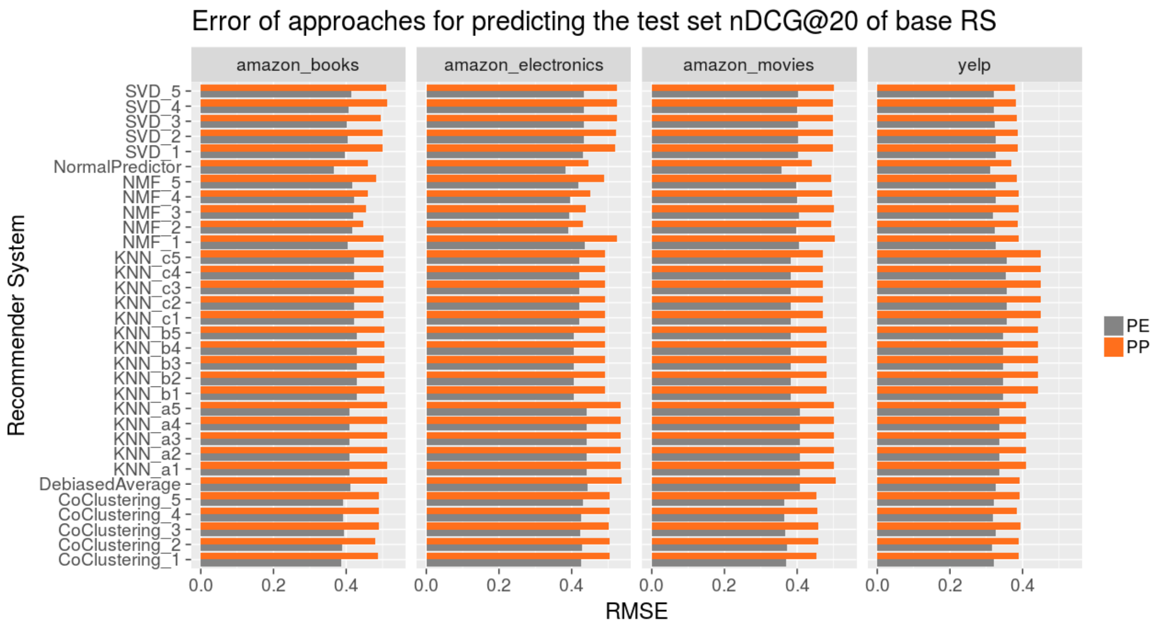


Figure 4.8: Error of our proposed performance estimates (PE) for the performance prediction task for multiple recommender systems, using nDCG@20 as Δ for the effectiveness measurement. It consistently outperforms performance predictors (PP) from the literature.

4.3.2 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 4.3 (denoted $+PE_u$ for clarity) as well as their weighted version from Equation 4.4 (denoted $+PE_w$).

Table 4.4 shows the results of this investigation. Bold values denote the best result in each row, while superscript letters denote statistically significant improvements over the corresponding method. Compared to *RS*, either $+PE_u$ or $+PE_w$ significantly improve for 6 out 8 tested ensembles (exceptions are LambdaMART and SVM) for the Yelp dataset. Similarly, for Amazon Books, significant improvements are observed for 5 out of 8 ensembles (exceptions are LambdaMART, SVM and Gradient Boosting), whereas for Amazon Movies and Electronics, 7 out of 8 ensembles are significantly improved (exceptions are LambdaMART and Gradient Boosting, respectively). Compared to using $+PP$, either $+PE_u$ or $+PE_w$ improve in most cases: 25 out of 32 ensembles (78%), with nDCG@20 gains up to 79% (with a mean gain of 26%) when compared to using only *RS* and with gains up to 63% (with a mean gain of 16%) when compared to using *RS* and *PP*. Recalling question *Q1*, the results attest the effectiveness of performance estimates at improving recommendation ensembles, encouraging their usage as a replacement for handcrafted performance predictors.

4.3.3 Robustness to Ensembling Strategy

Results in the previous section demonstrated the effectiveness of performance estimates (*PE*) as an addition to base recommender scores (*RS*) and as an alternative to performance predictors (*PP*). To address *Q2*, we further assess the robustness of *PE* to different ensembling strategies. In particular, we note from Table 4.4 that the unweighted variant PE_u, which provides user-dependent, item-agnostic performance estimates, tends to perform best (7 out of 7 cases) with pointwise ensemblers (gradient boosting, random forest, SVM, neural networks). This result suggests that these non-linear models are somehow capable of leveraging such 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 (13 out of 18 cases) for pairwise (RankBoost) and listwise ensemblers (AdaRank, ListNet). A key distinction

	baselines		**proposed approaches**	
Ensemble	RS (a)	$+PP$ (b)	$+PE_w$ (c)	$+PE_u$ (d)
		Yelp		
AdaRank	0.427	0.468^{ad}	**0.668**^{abd}	0.427
LambdaMART	0.592^{c}	**0.597**^{c}	0.465	0.591^{c}
ListNet	0.456^{bd}	0.415	**0.679**^{abd}	0.451^{b}
RankBoost	0.500	0.510^{ad}	**0.556**^{abd}	0.499
GradBoosting	0.437	0.435	0.438	**0.450**^{abc}
SVM	0.434	**0.479**^{acd}	0.431	0.443^{ac}
NeuralNetwork	0.435^{c}	0.432	0.426	**0.442**^{c}
RandomForest	0.429	0.429	**0.451**^{ab}	0.450^{ab}
		Amazon Books		
AdaRank	0.451	0.532^{ad}	**0.765**^{abd}	0.457
LambdaMART	**0.477**^{cd}	0.475^{cd}	0.393	0.461^{c}
ListNet	0.444	0.572^{ad}	**0.725**^{abd}	0.467^{a}
RankBoost	0.470	0.565^{ad}	**0.688**^{abd}	0.479^{a}
GradBoosting	0.508	0.513^{d}	**0.519**^{d}	0.500
SVM	0.502^{c}	**0.520**^{acd}	0.452	0.501^{c}
NeuralNetwork	0.485	0.477	0.510^{ab}	**0.513**^{ab}
RandomForest	0.458	0.471^{a}	**0.520**^{abd}	0.486^{ab}
		Amazon Movies		
AdaRank	0.487	0.485	**0.689**^{abd}	0.489
LambdaMART	0.591	**0.615**^{acd}	0.578	0.587
ListNet	0.549	0.625^{ad}	**0.714**^{abd}	0.548
RankBoost	0.567	0.605^{ad}	**0.637**^{abd}	0.565
GradBoosting	0.497	0.520^{a}	**0.567**^{abd}	0.525^{a}
SVM	0.498	0.516^{ac}	0.489	**0.521**^{ac}
NeuralNetwork	0.498	0.521^{a}	0.510	**0.531**^{ac}
RandomForest	0.482	0.500^{a}	**0.536**^{ab}	0.517^{ab}
		Amazon Electronics		
AdaRank	0.457	0.586^{ad}	**0.819**^{abd}	0.460
LambdaMART	0.496	0.588^{ad}	**0.687**^{abd}	0.489
ListNet	0.496^{d}	0.598^{ad}	**0.816**^{abd}	0.462
RankBoost	0.454	0.580^{ad}	**0.635**^{abd}	0.455
GradBoosting	**0.512**^{bc}	0.488	0.486	0.510^{bc}
SVM	0.444	0.459^{a}	**0.471**^{a}	0.468^{a}
NeuralNetwork	0.480	0.492^{c}	0.472	**0.493**^{ac}
RandomForest	0.445	0.451	0.463^{a}	**0.478**^{abc}

Table 4.4: @20 results for different ensemblers leveraging different combinations of meta-features: base recommenders scores only (RS, first baseline), added performance predictors ($+PP$, state-of-the-art baseline for using performance predictions), and added performance estimates ($+PE_w$ and $+PE_u$, our proposed approaches). Bold values denote the best result in each row, while superscript letters denote statistically significant improvements over the corresponding method.

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 notable exception of LambdaMART, which is significantly improved only for Amazon Electronics, these results further attest the robustness of performance estimates for different ensemblers.

4.3.4 Complementarity to PP

Our previous results demonstrated the effectiveness of using PE, yielding better nDCG@20 results than using PP. However, another possible use of PE, instead of replacing PP entirely, is to complement them, enabling further performance improvements. In this section, we address *Q3*, by investigating the complementarity of PP and PE.

Our experiments show that, for most ensembles and dataset combinations, expanding the ensembling input space to also include PP is not better than using solely PE_w for 22 out of 32 ensemblers. However, for the PE_u variant, 18 out of 32 ensemblers leveraged PP information in a beneficial way. Given that unweighted performance estimates carry less information, it is not surprising that adding more user dimensions will help improve ensembling. Recalling *Q3*, these results attest the complementarity of PE_u with respect to PP. Nevertheless, this effect is not as strong for PE_w.

4.3.5 Generalization Power

Supervised learning models are usually optimized via empirical risk minimization with the hope that the learned hypothesis will generalize to unseen test data. However, as models might overfit to the training examples, performance estimates computed on these examples might be optimistic. In this section, we address *Q4*, by assessing the generalization power of PE computed using the base recommender's training set in contrast to using a separate validation set. Table 4.5 describes the results of this experiment, once again including both PE_u and PE_w variants of our approach.

Surprisingly, there are several cases where PE calculated on training examples are not worse than the ones calculated on the validation set, and sometimes even better. There are a total of 40 out of 64 cases of ensembles where this happens, suggesting that estimates taken on the training set are good enough for most of the datasets and models. This fact makes PE even more convenient, as having a separate validation set (which makes the base recommenders' training set smaller) is not always necessary to achieve performance improvements. Recalling *Q4*, we did not find strong evidence

Ensemble	PE_w^v (a)	PE_w^t (b)	PE_u^v (a)	PE_u^t (b)
		Yelp		
AdaRank	0.668	**0.686**^{a}	**0.427**	**0.427**
LambdaMART	**0.465**	0.456	**0.591**	0.587
ListNet	**0.679**^{b}	0.640	0.451	**0.454**
RankBoost	**0.556**^{b}	0.542	**0.499**	**0.499**
GradBoosting	**0.438**^{b}	0.424	**0.450**^{b}	0.441
SVM	**0.431**	0.429	**0.443**^{b}	0.431
NeuralNetwork	0.426	**0.443**^{a}	**0.442**^{b}	0.435
RandomForest	**0.451**	0.444	**0.450**^{b}	0.438
		Amazon Books		
AdaRank	**0.765**	**0.765**	**0.457**	**0.457**
LambdaMART	0.393	**0.395**^{a}	0.461	**0.478**^{a}
ListNet	**0.725**	0.723	0.467	**0.474**^{a}
RankBoost	**0.688**^{b}	0.595	**0.479**	**0.479**
GradBoosting	**0.519**^{b}	0.470	0.500	**0.517**^{a}
SVM	0.452	**0.527**^{a}	0.501	**0.514**^{a}
NeuralNetwork	0.510	**0.526**^{a}	**0.513**	0.512
RandomForest	**0.520**^{b}	0.479	0.486	**0.494**
		Amazon Movies		
AdaRank	0.689	**0.698**	**0.489**	**0.489**
LambdaMART	**0.578**^{b}	0.509	0.587	**0.592**
ListNet	**0.714**^{b}	0.682	0.548	**0.553**^{a}
RankBoost	**0.637**^{b}	0.627	**0.565**	**0.565**
GradBoosting	**0.567**	0.561	**0.525**^{b}	0.514
SVM	**0.489**	0.484	**0.521**^{b}	0.501
NeuralNetwork	**0.510**^{b}	0.486	**0.531**^{b}	0.512
RandomForest	0.536	**0.537**	**0.517**^{b}	0.511
		Amazon Electronics		
AdaRank	**0.819**^{b}	0.653	**0.460**	**0.460**
LambdaMART	**0.687**^{b}	0.374	**0.489**^{b}	0.478
ListNet	**0.816**^{b}	0.542	0.462	**0.490**^{a}
RankBoost	**0.635**^{b}	0.625	**0.455**	**0.455**
GradBoosting	**0.486**	0.478	0.510	**0.514**
SVM	0.471	**0.485**^{a}	0.468	**0.484**^{a}
NeuralNetwork	0.472	**0.482**^{a}	**0.493**	0.492
RandomForest	**0.463**	**0.463**	**0.478**^{b}	0.462

Table 4.5: nDCG@20 results of ensembles leveraging PE calculated using training (superscript t) or separate validation (superscript v) examples. All combinations implicitly include the score of base recommenders (RS). Bold values denote the best result in each row, while superscript letters denote statistically significant improvements over the corresponding method. We observe that using a separated validation set from the training set of ratings for calculating performance estimates was not mandatory, as they reach similar results.

in support of using a separate validation set for computing PE, which suggests that measuring PE on the recommender systems training set generalizes well to unseen data.

4.3.6 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 on the top-k recommendation 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 *Q5*, by contrasting the discriminative power of performance estimates computed using either RMSE or nDCG as representative of error and ranking-based metrics, respectively. Table 4.6 shows the results of this investigation, once again for both the PE_u and PE_w variants.

Contrary to our prior belief, Table 4.6 shows that PE based on RMSE is at least as effective (and sometimes outperforms) estimates based on nDCG on 41 out of 64 cases. This counter-intuitive result can be observed for both unweighted and weighted variants and could be explained by the fact that nDCG does not distinguish between items with the same relevance level (rating). Let us assume, for instance, that a user rated three items with $[5, 3, 2]$ stars respectively. If one of the recommender systems in the ensemble predicted $[3, 2, 0]$ for this user's items, the PE_{NDCG} would be 1.0, which is the best possible value, while the PE_{RMSE} for this example would be 3 (0 being the best possible value), capturing the numeric errors made by the model for this user. Recalling question *Q5*, in contrast to nDCG, RMSE provides a more fine-grained assessment of such tied items, which could help explain its improved discriminative power for ensembling multiple recommenders.

4.3.7 Breakdown Analyses

In this section, we perform three additional breakdown analyses to provide further insight into the investigations conducted thus far. Firstly, to assess the extent to which our observations made in light of each of 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 4.9 plots

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

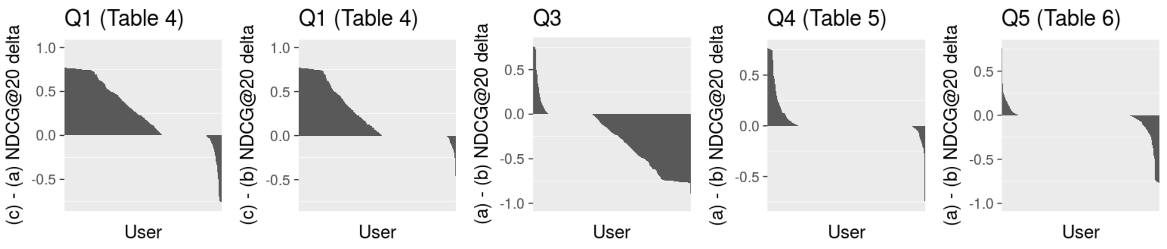


Figure 4.9: nDCG@20 improvement for ListNet across users on Yelp. Each plot analyzes the indicated research question, with symbols in parentheses on the y-axis referencing the columns in the source table used to compute the improvement (e.g., the first plot compares columns (c) and (a) of Table 4.4).

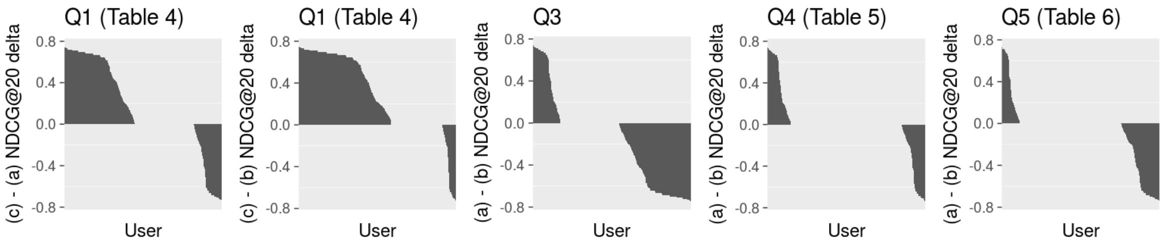


Figure 4.10: nDCG@20 improvement for ListNet across users on Amazon Books.

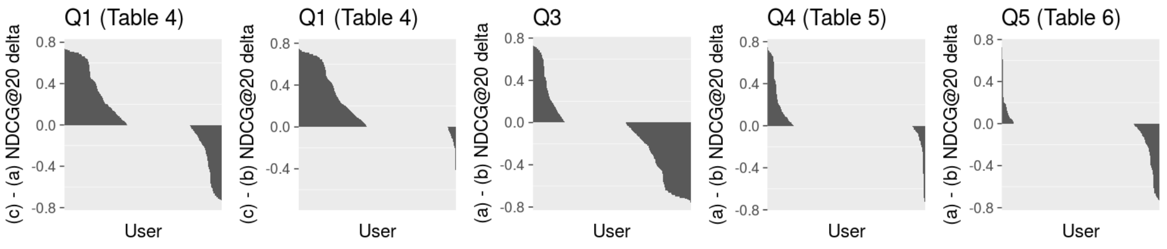


Figure 4.11: nDCG@20 improvement for ListNet across users on Amazon Movies.

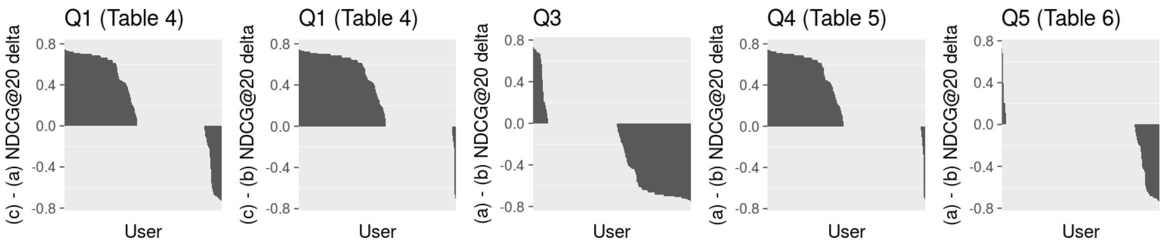


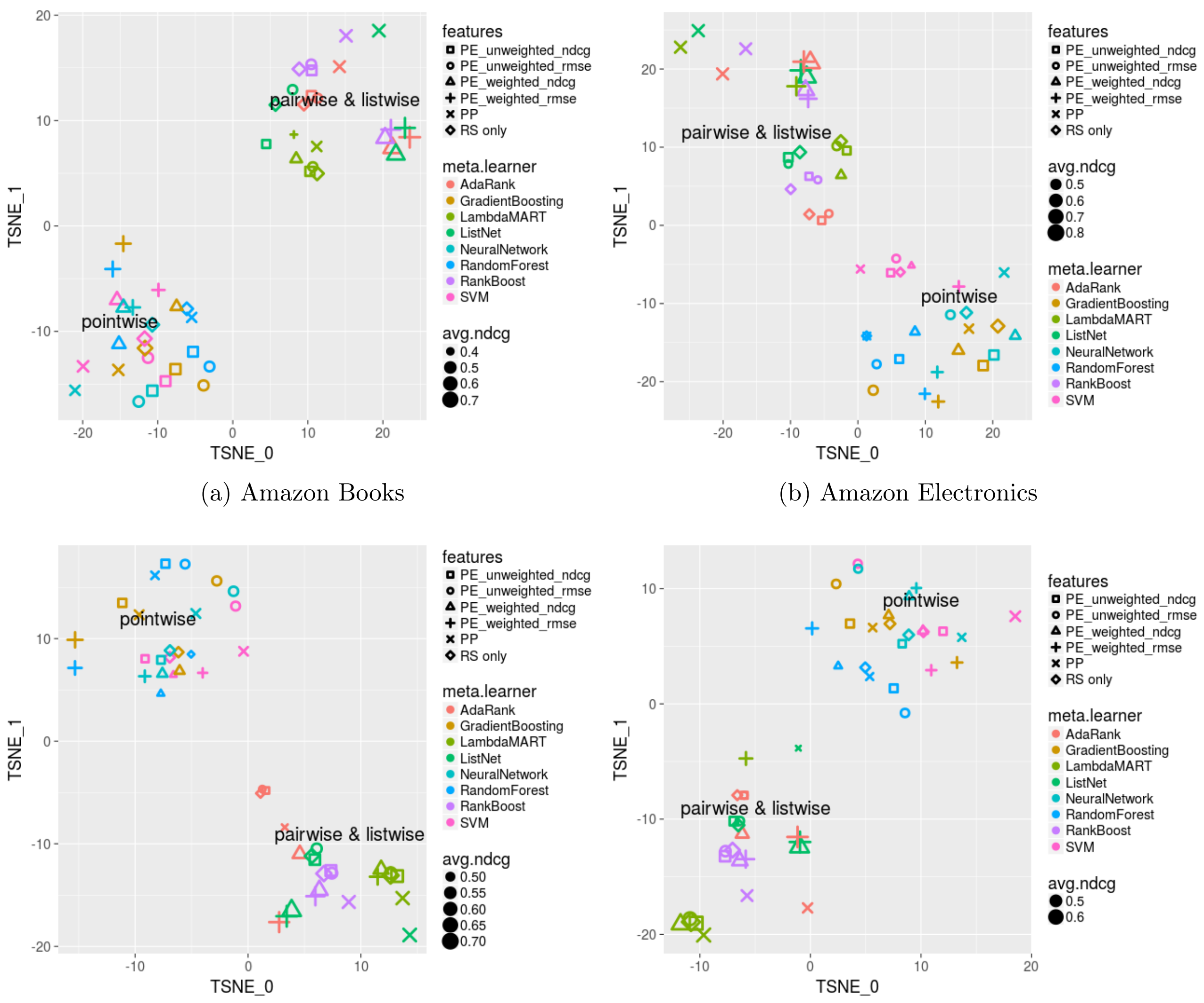
Figure 4.12: nDCG@20 improvement for ListNet across users on Amazon Electronics.

nDCG@20 improvements across all users on the Yelp dataset (distributions on the other datasets are strikingly similar as we see in Figures 4.10, 4.11 4.12). Each plot conveys improvements for a different research question, with letters in parentheses on the y-axis indicating the settings compared in each case (e.g., the first plot compares columns (c) and (a) of Table 4.4).

From Figure 4.9, we observe that *Q1* is answered positively, or results in a tie, 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 *Q3*, the third plot confirms that PE_w is more effective on its own than when combined with PP, with the combination hurting the ensembling performance for most users. As for *Q4* and *Q5*, the fourth and fifth plots show neutral nDCG@20 improvements for the majority of users, confirming that neither a separate validation set (*Q4*) nor using a ranking evaluation metric for performance estimation (*Q5*) have a positive effect. We conclude that our proposed approach increases recommendation accuracy for most users while incurring minimum risk of decreasing recommendation quality.

Another question that arises is how each ensemble, given a set of features, performs compared to all others. To shed light on this matter, we reduced the dimensionality of ensemble results (concerning all users) using t-SNE [Maaten and Hinton, 2008] into two dimensions. Each point in this new embedded space denotes how the ensemble performs in terms of nDCG@20, and proximity to other ensembles indicates that they have similar results for the same users. Figure 4.13 shows the output of this process for the Amazon Books dataset, revealing that pointwise methods and listwise/pairwise methods generated 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 performance estimates (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 smaller 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_w^e (triangles labeled `PE_weighted_rmse`) and PE_w^g (plus symbols labeled `PE_weighted_ndcg`).

Finally, we analyze the importance of specific meta-features for the ensembles. To get a rough idea as to the importance ranking for each set of meta-features, we used random forest combined with Gini importance [Louppe et al., 2013] as the sorting criteria, training three distinct models for each input space, $RS + PP$, RS, and $RS + PE$. For this experiment, we chose the weighted variation of PE, calculated on



(a) Amazon Books

(b) Amazon Electronics

(c) Amazon Movies

(d) Yelp

Figure 4.13: Dimensionality reduction of ensemble configurations using t-SNE, with input dimensions denoting the nDCG@20 performance for unique users. We observe clusters being formed by the type of ensemble (pointwise and pairwise/listwise) as well as by the set of features used (shapes).

the validation set and using RMSE for the estimations due to the previous research questions findings. Due to space limitations, in Table 4.7, we report the results only for Amazon Books. As expected, results of the feature importance analysis indicate that the performance estimates (PE) in the top-10 meta-features correspond to recommenders that are also highly ranked themselves as meta-features for the ensemble (RS). As for performance predictors (PP), the two most important meta-features are user abnormality formulations [Gras et al., 2015], which capture how atypical the user preferences are.

$RS + PP$		RS		$RS + PE$	
feat.	imp.	feat.	imp.	feat.	imp.
PP_7	0.031	RS_SVD_5	0.086	PE_NP	0.035
PP_6	0.029	RS_BO	0.085	PE_SVD_5	0.026
PP_{12}	0.029	RS_SVD_3	0.074	PE_SVD_1	0.023
PP_{10}	0.025	RS_SVD_1	0.069	PE_BO	0.022
PP_8	0.024	RS_SVD_4	0.069	PE_SVD_4	0.022
PP_9	0.017	RS_NMF_4	0.066	PE_NMF_3	0.022
PP_{14}	0.017	RS_NMF_3	0.050	PE_NMF_5	0.021
PP_3	0.017	RS_NP	0.050	PE_SVD_2	0.021
PP_1	0.016	RS_NMF_5	0.048	PE_NMF_1	0.020

Table 4.7: Meta-feature importance on Amazon Books. We abbreviate DebiasedAverage 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).

Using the aforementioned criteria for defining meta-feature importance, we performed a final experiment that adds meta-features incrementally to the input space from the most important ones to the least, training an ensemble for each resulting configuration. We do that for both our proposed set of features (PE), and for the baseline that leverages performance predictions (PP) from the literature. Results in Figure 4.14 indicate that, for PP, ensemble accuracy was not increased when using the top ranked meta-features according to the Gini coefficient. However, PP_1, the 10th best ranked meta-feature in this group, provided a boost in performance. For PE, after a certain amount of meta-features is added, performance reaches a plateau, indicating that not all performance estimates (and correspondingly, recommender scores) are needed to achieve the best accuracy. This experiment shows that in a real-world scenario, not all recommenders and their performance estimates are necessary in production, only the 5 or 10 most important ones.

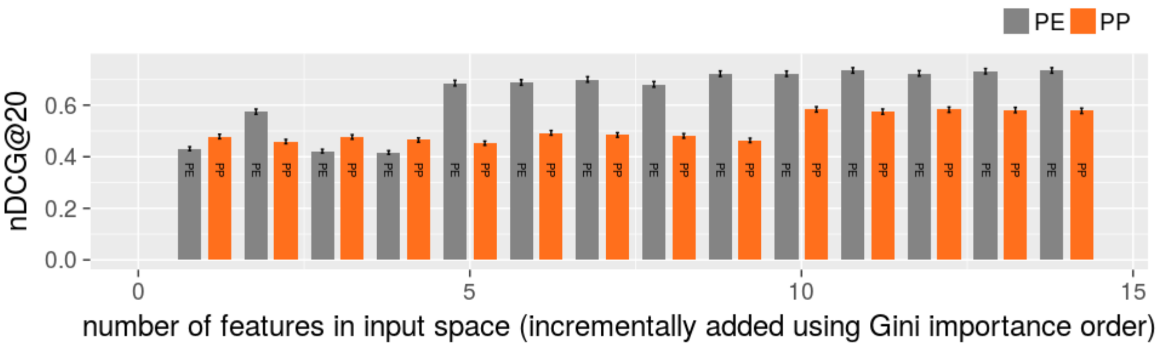


Figure 4.14: nDCG@20 results after the incremental addition of the most important meta-features by the Gini importance using ListNet ensemble on Amazon Books. Error bars indicate 95% confidence intervals for the means.

4.4 Summary

In this chapter, we addressed the claim from our dissertation statement again, in the recommendation scenario this time, by showing that effective performance prediction (obtained here using our novel approach called performance estimation) of different base models (recommender systems) can be exploited to improve the results of their combination. Performance estimates are directly computed on the historical feedback provided by the target user by standard evaluation metrics, such as RMSE or nDCG. As a result, such meta-features are highly discriminative of the performance of different recommenders for the target user and incur virtually no engineering cost when new recommenders are added to the ensemble. Through a thorough evaluation using datasets in four different domains, we demonstrated the effectiveness of performance estimates at improving ensembles produced by representative pointwise, pairwise, and listwise learning to rank approaches.

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 Movies		
AdaRank	0.564	**0.689**^{a}	**0.489**	**0.489**
LambdaMART	**0.579**^{b}	0.578	**0.592**	0.587
ListNet	0.678	**0.714**^{a}	**0.556**^{b}	0.548
RankBoost	0.616	**0.637**^{a}	**0.565**	**0.565**
GradBoosting	0.499	**0.567**^{a}	0.508	**0.525**^{a}
SVM	0.482	**0.489**	0.495	**0.521**^{a}
NeuralNetwork	0.503	**0.510**	0.501	**0.531**^{a}
RandomForest	0.482	**0.536**^{a}	0.507	**0.517**^{a}
		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**

Table 4.6: nDCG@20 results for ensemblers leveraging performance estimates computed using either RMSE (superscript e) or nDCG (superscript g). All combinations implicitly include the score of base recommenders (RS). Bold values denote the best result in each row, while superscript letters denote statistically significant improvements over the corresponding method. Surprisingly, RMSE outperforms nDCG as the evaluation metric Δ for performance estimation.

Chapter 5

Conclusion

The combination of models in machine learning has shown great results for several different tasks and domains. A common approach for weighting the predictions of several models, called stacking, adds another layer on top of the predictions containing a final model that makes the final prediction based on the other models outputs. This strategy has been applied to the Information Retrieval area through learning to rank models, which typically combines several ranking functions to generate a final ranking. This approach has been the state-of-the art for ad-hoc retrieval and also recommendation ensembling, and in combination with Performance Prediction, has been an extremely active area of research.

In this dissertation, we proposed to tackle the problem of effectively using performance predictors for the combination of models. In particular, we argued that having one effective performance predictor for each different base model, can be exploited to improve the results of their combination. By having an accurate performance prediction for each base model, the ensemble has an easier task of combining their outputs, being able to discriminate the context that make a base model outperform others.

Throughout this dissertation, we described and validated this claim in the domains of ad-hoc retrieval and recommendation. In the remainder of this chapter, we summarize our contributions in Section 5.1, conclusions in Section 5.2 and finally we point to directions for future work in Section 5.3.

5.1 Summary of contributions

In this section we summarize the contributions made throughout this dissertation.

A novel approach for performance prediction in RS In Chapter 4 we proposed a novel set of features for predicting the performance of recommender systems for users. It is based on the premise that past performance of a model for a user indicates its future performance. Using historical ratings of each user, we directly estimate the performance of each base model in the ensemble instead of engineering features to predict such performance. In Section 4.1.1 we formally define this set of features, that are then thoroughly evaluated in Section 4.3.

A framework for using performance prediction in ensembling In Chapters 3 and 4 we use the same structure for leveraging performance predictors. As defined in Sections 3.1 and 4.1.2, we have one performance prediction for each base model, that is used as additional input features for the ensemble. Moreover, we also proposed a variant that further increases the sensibility of the features, by multiplying each base model output by its respective performance prediction. We validate the effectiveness of this framework doing experiments in both ad-hoc retrieval and recommendation, in Sections 3.3.2.2 and 4.3.2 respectively.

5.2 Summary of conclusions

In this section we summarize the conclusions drawn from the comprehensive evaluation done in this dissertation. In particular we validate the statement of this dissertation, presented in Section 1.1.

On the effectiveness of PP for combining ranking functions In Chapter 3 we investigate ensembles of ranking functions enhanced with query performance prediction features. Our goal was to understand the limitations of using query performance predictions to actually improve ad-hoc retrieval. To this end, we first evaluated through a noise analysis in Section 3.3.1 the potential of using query performance predictions, showing that current approaches fall short in improving the baseline strategy and that improved effectiveness on the performance prediction task improved ad-hoc retrieval in this simulation. Moreover, to attest that performance prediction are able to improve the combination of ranking functions, we used improved performance predictors by having one machine-learned query performance predictor for each ranking function in the ensemble. In Section 3.3.2 we first show that ML-QPP are better at the performance prediction task than raw QPP, moreover we show that this improvement translates to improved ad-hoc retrieval effectiveness, showing evidence in favor of our dissertation statement.

On the effectiveness of PP for combining recommender systems In Chapter 4 we investigated ensembles of recommender systems enhanced with user performance prediction features. We argued that having one accurate user performance prediction for each base recommender improves the performance of the ensemble. To this 7end, we proposed a new set of features for the performance prediction task, which we called performance estimates. First, in Section 4.3.2, we evaluated how our PE compared to common performance predictors from the literature, when used as additional features to the ensemble. The results were favourable to our dissertation statement, showing that accurate performance predictions can be used to improve the combination of models, reaching nDCG@20 gains up to 79%. In addition, we evaluated several aspects of the proposed set of features such as its generalization power in Section 4.3.5, suggesting that we do not need a separated validation set to calculate performance estimates. Moreover, in Section 4.3.6 we concluded that RMSE was a better evaluation metric than NDCG for calculating the performance prediction, reaching higher improvements for the ensembling task. Finally in Section 4.3.7, we provided breakdown analyses across users, different ensembling techniques and feature importances, providing further insight into the investigations conducted on performance estimates.

5.3 Future work

There are few directions for future research inspired directly by the results of this dissertation. First, we plan to explore the effectiveness of Performance Prediction for combining multiple classifiers in the context of text classification, using the successful framework proposed here. Using the same setup, we could investigate whether one effective performance predictor for each classifier in the stacking improves the effectiveness of the ensemble. However, the area of predicting the performance for text classification models has not yet been explored in the information retrieval literature. This way, an open problem is to first understand if current approaches for predicting the performance of retrieval methods for queries are able to accurately predict the performance of classification methods for documents, and then assess their usefulness for improving stacking of multiple classifiers.

Another direction for future research is on extending the framework for exploiting performance predictions in the level of a single base model instead of the stacking layer. In this dissertation we explored how to use such predictions to enhance the combination of models learned from different hypothesis spaces, which is called stacking. Another possibility is to design a new machine learning model that uses performance prediction in its core. Boosting, for instance, leverages the information of train error estimates for the models being combined in order to focus on harder training instances and also to weight base estimators accordingly for the final prediction. On the other hand, bootstrap aggregating (bagging) applies several weak learners to samples of the input space and combines them by taking the average prediction. We plan to devise and evaluate a tree-based machine learning model that uses auxiliary performance prediction models - as we proposed in this dissertation in stacking strategies - for each tree in the ensemble.

Bibliography

Aggarwal, C. C. (2016a). Ensemble-based and hybrid recommender systems. In *Recommender Systems*, pages 199--224. Springer.

Aggarwal, C. C. (2016b). *Recommender Systems*. Springer.

Arguello, J., Avula, S., and Diaz, F. (2017). Using query performance predictors to reduce spoken queries. In *Proc. of ECIR*, pages 27--39. Springer.

Aslam, J. A. and Montague, M. (2001). Models for metasearch. In *Proc. of SIGIR*, pages 276--284. ACM.

Bao, X., Bergman, L., and Thompson, R. (2009). Stacking recommendation engines with additional meta-features. In *Proc. of RecSys*, pages 109--116. ACM.

Bar, A., Rokach, L., Shani, G., Shapira, B., and Schclar, A. (2012). Boosting simple collaborative filtering models using ensemble methods. *arXiv preprint arXiv:1211.2891*.

Bellogín, A., Castells, P., and Cantador, I. (2011). Predicting the performance of recommender systems: an information theoretic approach. In *Proc. of ICTIR*, pages 27--39. Springer.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA.

Berger, P.-Y. and Savoy, J. (2007). Selecting automatically the best query translations. In *Large Scale Semantic Access to Content (Text, Image, Video, and Sound)*, pages 287--300. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(Feb):281--305.

Breiman, L. (1996a). Bagging predictors. *Machine learning*, 24(2):123--140.

Breiman, L. (1996b). Stacked regressions. *Machine learning*, 24(1):49--64.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5--32.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Model. User-Adapted Interac.*, 12(4):331--370.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proc. of SIGKDD*, pages 785--794. ACM.

Cremonesi, P., Koren, Y., and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *Proc. of RecSys*, pages 39--46. ACM.

Cronen-Townsend, S., Zhou, Y., and Croft, W. B. (2002). Predicting query performance. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 299--306. ACM.

Cronen-Townsend, S., Zhou, Y., and Croft, W. B. (2004). A framework for selective query expansion. In *Proc. of CIKM*, pages 236--237.

Dan, O. and Davison, B. D. (2016). Measuring and predicting search engine users' satisfaction. *ACM Computing Surveys (CSUR)*, 49(1):18.

Dooms, S., De Pessemier, T., and Martens, L. (2015). Offline optimization for user-specific hybrid recommender systems. *Multimedia Tools and Applications*, 74(9):3053--3076.

Ekstrand, M. and Riedl, J. (2012). When recommenders fail: predicting recommender failure for algorithm selection and combination. In *Proc. of RecSys*, pages 233--236. ACM.

Fortes, R. S., de Freitas, A. R. R., and Gonçalves, M. A. (2017). A multicriteria evaluation of hybrid recommender systems: on the usefulness of input data characteristics. In *Proc. of ICEIS*, pages 623--633.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367--378.

George, T. and Merugu, S. (2005). A scalable collaborative filtering framework based on co-clustering. In *Proc. of ICDM*, pages 4--pp. IEEE.

Ghazanfar, M. A. and Prügel-Bennett, A. (2014). Leveraging clustering approaches to solve the gray-sheep users problem in recommender systems. *Expert Syst. Appl.*, 41(7):3261--3275.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gras, B., Brun, A., and Boyer, A. (2015). Identifying users with atypical preferences to anticipate inaccurate recommendations. In *Proc. of WEBIST*.

Griffith, J., O'Riordan, C., and Sorensen, H. (2012). Investigations into user rating information and predictive accuracy in a collaborative filtering domain. In *Proc. of SAC*, pages 937--942. ACM.

Hauff, C. (2010). *Predicting the effectiveness of queries and retrieval systems*. PhD thesis, University of Twente.

Hauff, C., Azzopardi, L., and Hiemstra, D. (2009). The combination and evaluation of query performance prediction methods. In *Proc. of ECIR*, pages 301--312.

Jahrer, M., Töscher, A., and Legenstein, R. (2010). Combining predictions for accurate recommender systems. In *Proc. of SIGKDD*, pages 693--702. ACM.

Katz, G., Shtock, A., Kurland, O., Shapira, B., and Rokach, L. (2014). Wikipedia-based query performance prediction. In *Proc. of SIGIR*, pages 1235--1238. ACM.

Kearns, M. (1988). Thoughts on hypothesis boosting. *Unpublished manuscript*, 45:105.

Koren, Y. (2009). The Bellkor solution to the Netflix grand prize. *Netflix prize documentation*, 81:1--10.

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

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8).

Liu, T.-Y. et al. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225--331.

Lopes, R., Assunção, R., and Santos, R. L. (2016). Efficient Bayesian methods for graph-based recommendation. In *Proc. of RecSys*, pages 333--340. ACM.

Louppe, G., Wehenkel, L., Sutera, A., and Geurts, P. (2013). Understanding variable importances in forests of randomized trees. In *Proc. of NIPS*, pages 431--439.

Luo, X., Zhou, M., Xia, Y., and Zhu, Q. (2014). An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Trans. Ind. Inf.*, 10(2):1273--1284.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *J. Mach. Learn. Res.*, 9(Nov):2579--2605.

Macdonald, C., Santos, R. L. T., and Ounis, I. (2012). On the usefulness of query features for learning to rank. In *Proc. of CIKM*, pages 2559--2562. ACM.

Matuszyk, P. and Spiliopoulou, M. (2014). Predicting the performance of collaborative filtering algorithms. In *Proc. of WIMS*, page 38. ACM.

Montgomery, D. C., Peck, E. A., and Vining, G. G. (2015). *Introduction to linear regression analysis*. John Wiley & Sons.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Poisson, S. D. (1837). *Recherches sur la probabilite des jugements en matiere criminelle et en matiere civile, precedees des regles generales du calcul des probabilites*. Bachelier.

Qin, T. and Liu, T.-Y. (2013). Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*.

Raiber, F. and Kurland, O. (2014). Query-performance prediction: setting the expectations straight. In *Proc. of SIGIR*, pages 13--22. ACM.

Robertson, S. E. and Jones, K. S. (1976). Relevance weighting of search terms. *Journal of the Association for Information Science and Technology*, 27(3):129--146.

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., Gatford, M., et al. (1995). Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613--620.

Santos, R. L. T. (2013). *Explicit web search result diversification*. PhD thesis, University of Glasgow.

Santos, R. L. T., Macdonald, C., and Ounis, I. (2010). Selectively diversifying web search results. In *Proc. of CIKM*, pages 1179--1188.

Shtok, A., Kurland, O., and Carmel, D. (2016). Query performance prediction using reference lists. *ACM Transactions on Information Systems (TOIS)*, 34(4):19.

Sill, J., Takács, G., Mackey, L., and Lin, D. (2009). Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460*.

Valcarce, D., Parapar, J., and Barreiro, Á. (2017). Combining top-n recommenders with metasearch algorithms. In *Proc. of SIGIR*, pages 805--808. ACM.

Wu, Q., Burges, C. J., Svore, K. M., and Gao, J. (2008). Ranking, boosting, and model adaptation. Technical report, Microsoft Research.

Appendix A

Letor 4.0 models

Here we describe the features that compose LETOR 4.0, as they are described in [Qin and Liu, 2013].

Column in Output	Description
1	TF(Term frequency) of body
2	TF of anchor
3	TF of title
4	TF of URL
5	TF of whole document
6	IDF(Inverse document frequency) of body
7	IDF of anchor
8	IDF of title
9	IDF of URL
10	IDF of whole document
11	TF*IDF of body
12	TF*IDF of anchor
13	TF*IDF of title
14	TF*IDF of URL
15	TF*IDF of whole document
16	DL(Document length) of body
17	DL of anchor
18	DL of title
19	DL of URL
20	DL of whole document
21	BM25 of body
22	BM25 of anchor
23	BM25 of title
24	BM25 of URL
25	BM25 of whole document
26	LMIR.ABS of body
27	LMIR.ABS of anchor
28	LMIR.ABS of title
29	LMIR.ABS of URL
30	LMIR.ABS of whole document
31	LMIR.DIR of body
32	LMIR.DIR of anchor
33	LMIR.DIR of title
34	LMIR.DIR of URL
35	LMIR.DIR of whole document
36	LMIR.JM of body
37	LMIR.JM of anchor
38	LMIR.JM of title
39	LMIR.JM of URL
40	LMIR.JM of whole document
41	PageRank
42	Inlink number
43	Outlink number
44	Number of slash in URL
45	Length of URL
46	Number of child page

Table A.1: Ranking models contained in LETOR 4.0.

Appendix B

Implemented QPP

For the experiments in Chapter 3 we used several query performance predictors, that we explain in detail here how they were calculated. All implementations can be further analyzed using the code from `https://github.com/Guzpenha/performance-prediction-for-enhancing-ensemble-learning`.

MatchingDocs Number of matching documents for the query, extracted directly from MetaFeature file of LETOR 4.0.

TokenCount Number of terms in the query, using space as the separator between terms.

TermCount Number of unique terms in the query, using space as the separator between terms.

AvQL Average character size of terms in the query.

AvIDF Average inverse document frequencies of the query terms for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features.

MaxIDF Maximum inverse document frequencies of the query terms for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features.

DevIDF Standard deviation of inverse document frequencies of the query terms for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features.

AvICTF Average inverse collection term frequency of query terms for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $AvICTF = \frac{1}{m}\sum_{i=1}^{m}[log_2(termcount) - log_2(tf(q_i))]$

SCS Simplified Clarity Score of query terms for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $SCS \approx log_2\frac{1}{m} + \frac{1}{m}\sum_{i=1}^{m}[log_2(termcount) - log_2(tf(q_i)]$

SumSCQ Sum of collection query similarity for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $SumSCQ = \sum_{i=1}^{m}(1 + ln(cf(q_i))) \times ln(1 + \frac{doccount}{df(q_i)})$

AvSCQ Average of collection query similarity for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $AvSCQ = \frac{1}{m} \times \sum_{i=1}^{m}(1 + ln(cf(q_i))) \times ln(1 + \frac{doccount}{df(q_i)})$

MaxSCQ Maximum of collection query similarity for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $MaxSCQ = max((1 + ln(cf(q_i))) \times ln(1 + \frac{doccount}{df(q_i)}))$

QS Query scope uses the number of documents - stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document) - that matches the query terms, resulting in 5 features. $QS = -log\frac{N_q}{doccount}$

SumVAR SumVAR is the sum of the query term weight deviations (TF-IDF) for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $SumVAR = \sum_{i=1}^{m}\sqrt{\frac{1}{df(q_i)}\sum_{d \in N_{qi}}(w(q_i,d) - AVGw_{qi})^2}$

AvVAR AvVAR is the average of the query term weight deviations (TF-IDF) for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $AvVAR = \frac{1}{m} \times \sum_{i=1}^{m}\sqrt{\frac{1}{df(q_i)}\sum_{d \in N_{qi}}(w(q_i,d) - AVGw_qi)^2}$

MaxVAR MaxVAR is the maximum value of the query term weight deviations (TF-IDF) for each stream in LETOR 4.0 (streams are: body, anchor, title, URL and whole document), resulting in 5 features. $AvVAR = max(\sum_{i=1}^{m}\sqrt{\frac{1}{df(q_i)}\sum_{d \in N_{qi}}(w(q_i,d) - AVGw_qi)^2})$

AvQC The average set coherence over all query terms, we used the similarity files available in LETOR 4.0, *"Large_simi.txt"* and the following similarity thresholds: 0.8, 0.85, 0.9 and 0.95. $SetCoherence(N_{q_i}) = \frac{\sum_{i \neq j \in 1,\ldots,n}\sigma(d_i,d_j)}{n(n-1)}$, where $\sigma(d_i, d_j)$ is 1 when the similarity between documents are over the similarity threshold, otherwise is 0.

AvP	Average number of senses for terms in the query, using WordNet function wordnet.synsets.
AvNP	Average number of noun senses for terms in the query, using WordNet function wordnet.synsets.
AvPMI	Averaged pointwise mutual information for query terms for each stream in LETOR 4.0. The pointwise mutual information is the division of the probability that the two terms occur togheter in a document by the probability of the terms occuring togheter by chance.
MaxPMI	Maximum pointwise mutual information for query terms for each stream in LETOR 4.0. The pointwise mutual information is the division of the probability that the two terms occur togheter in a document by the probability of the terms occuring togheter by chance.
AvPath	Relatedness between two terms of the query by the reciprocal of the number of nodes on the shortest path of the IS-A hierarchy between the two corresponding synset nodes. The implementation uses WordNet function wordnet.path_similarity.
AvLCH	Relatedness between two terms of the query by the Leacock-Chodorow similarity. The implementation uses WordNet function wordnet.lch_similarity.
AvWUP	Relatedness between two terms of the query by the Wu-Palmer similarity. The implementation uses WordNet function wordnet.wup_similarity.
SumLETOR	Sum for each one of 46 LETOR features for the top 10 retrieved documents.
MaxLETOR	Max for each one of 46 LETOR features for the top 10 retrieved documents.
StdLETOR	Standard deviation for each one of 46 LETOR features for the top 10 retrieved documents.
AvLETOR	Average for each one of 46 LETOR features for the top 10 retrieved documents.