User Engagement as Evaluation: a Ranking or a Regression Problem?

Frédéric Guillou, Romaric Gaudel, Jérémie Mary, Philippe Preux

User Engagement as Evaluation: a Ranking or a Regression Problem?

Frédéric Guillou
INRIA Lille - Nord Europe
frederic.guillou@inria.fr

Romaric Gaudel
University of Lille
romaric.gaudel@inria.fr

Jérémie Mary
University of Lille
jeremie.mary@inria.fr

Philippe Preux
University of Lille
philippe.preux@inria.fr

LIFL UMR CNRS 8022 & INRIA Lille - Nord Europe
40 avenue Halley 59650 Villeneuve d'Ascq, FR

ABSTRACT

In this paper, we describe the winning approach used on the RecSys Challenge 2014 which focuses on employing user engagement as evaluation of recommendations. On one hand, we regard the challenge as a ranking problem and apply the LambdaMART algorithm, which is a listwise model specialized in a Learning To Rank approach. On the other hand, after noticing some specific characteristics of this challenge, we also consider it as a regression problem and use pointwise regression models such as Random Forests. We compare how these different methods can be modified or combined to improve the accuracy and robustness of our model and we draw the advantages or disadvantages of each approach.

Categories and Subject Descriptors

[**Information systems**]: Information retrieval—*Retrieval tasks and goals; Recommender systems*; [**Computing methodologies**]: Machine learning—*Learning paradigms; Supervised learning; Learning to rank*; [**Computing methodologies**]: Machine learning—*Machine learning approaches; Classification and regression trees*; [**Computing methodologies**]: Machine learning—*Machine learning algorithms ; Ensemble methods*

Keywords

Recommender Systems; Learning to rank; LambdaMART; Random Forests

1. INTRODUCTION

Standard approaches for recommender systems [11, 4] usually focus on predicting ratings and interests of users, and then recommend high-scoring items. Such approaches have however shown some flaws [8], as achieving a low prediction error for ratings does not mean the recommended items would match user's taste. Compared with traditional approaches focusing on estimating ratings in the most accurate way, the Learning to Rank (LTR) approach only endeavors to predict an accurate ranking, and is nowadays an active topic in the field of Machine Learning due to the rapid growth of web search engines and recommendation systems.

Given a set of query-item pairs, described by a set of input features and a numerical score (or the rank among the other items) indicating the relevance of each item as a response to the given query, the LTR methods aim at learning from these data a model that would find the most appropriate ranking for a set of potentially unseen items for a potentially unseen query.

The RecSys challenge [10] investigates a specific recommendation task: find items with the highest user engagement. Specifically, the objective is to rank a set of tweets after the (unknown) number of time they will be favorited or retweeted [6]. This problem comes within the scope of a LTR problem, since each user can be considered as a query, and each tweet associated to each user as an item. Will regressions approaches handily win that challenge?

Our approach built for this challenge uses a combination of several ranking models. Firstly, we apply a LambdaMART algorithm [2], which is a listwise model for LTR, trained on a cleared version of the train dataset. After observing the shape of the data and noticing some specific characteristics, we also consider the problem as a standard regression problem and investigate a way to take these characteristics into account into our model by training Random Forests [1] on a full dataset, modified with respect to these characteristics. We compare several approaches to combine ranking or simple regression models and examine pros and cons of these two approaches for this type of problem.

The rest of this paper is organized as follows. Section 2 describes the dataset and the rules of the challenge. In Section 3, we briefly review LambdaMART method and Random Forest and explain how we use or combine them in our model. Our experimental protocols and results on the challenge are then described in Section 4. We open some discussion about the challenge and results in Section 5 and then conclude in Section 6.

RecSys '14 , Foster City, Silicon Valley, USA

http://dx.doi.org/10.1145/2668067.2668073.

Table 1: Training data statistics

Metric	Value
Tweets	170,285
Unique users	22,079
Unique items	13,618
Tweets with zero engagement	162,107 (95.2%)
Unsuccessful users	17,502 (79.27%)

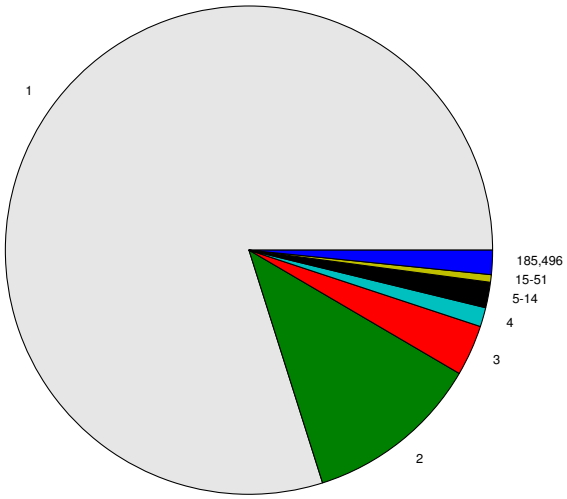


Figure 1: Distribution of the user engagement of successful tweets in the training dataset.

2. RECSYS CHALLENGE 2014: DATA AND PROTOCOL

For this challenge, a large dataset is split chronologically in three subsets: a training set, a test set, and an evaluation set. The percentage for each of these subsets is respectively around 80%, 10% and 10%. Contestants of the Recsys Challenge are provided on the training and test sets in order to build their models and algorithms, while the evaluation set is kept for the final evaluation in order to decide on the winner of the challenge. The dataset is an extended version of the MovieTweetings dataset [6], so data originate from users of the IMDb app, where they can rate movies and share the rating on Twitter. Tweets and information related to them were collected by querying the Twitter API on a daily basis for tweets containing the keywords 'I rated IMDb'.

As input features for each tweet, the challenge dataset contains metadata of the tweets as provided by the Twitter API. These metadata include information about the user, the movie or the tweet itself. The retweets and favorites counts were also included in the metadata, in order to evaluate tweets by their engagement and rank them.

2.1 Data Characteristics and Statistics

Each dataset contains tweets represented as follow: twitter user id, IMDb item id, rating given by the user to the movie, scraping timestamp, tweet data. We present in Table 1 some statistics about the training dataset. We denote a tweet which obtains a non-zero engagement as a *successful* tweet, and a user who have had at least one successful tweet as a *successful* user. These statistics give us some important information for our model which is described in Section 3. First of all, we notice the number of successful tweets is really low, i.e. most tweets have no success and do not get any retweet or are put as favorite by other users. As a consequence, most users in the dataset only have tweets with an engagement of 0. Successful tweets receive an average engagement of 4.41 but 80% of these tweets have only a user engagement of 1 (cf. Figure 1). Overall, most of the tweets have an engagement of either 0 or 1, which means the challenge is almost a binary classification problem. In fact, as shown in [9], the binary oracle, i.e. the classifier that only gives the score 1 to tweets with positive engagement and the score 0 to remaining tweets, gets an overall result of 0.9877 on the test set.

Table 2: About retweets in the training dataset

Metric	Value
Number of retweets	1,808
Percentage among successful tweets	22.10%
"Artificial" successful users	28.44%

Secondly, part of these tweets are retweets (cf. Table 2). The engagement of a retweeted tweet is a specific case in the dataset, since such tweets share their retweet count with the original tweet (but do not share their favorite count). Every retweet necessarily has a strictly positive engagement. Given that both the retweet and favorite counts of the original tweet are available on the metadata, this distinctive feature has to be taken into account in the model. These tweets have not been originally posted by the user that we evaluate, so the user engagement of retweets can cause a distorted evaluation. On the 4,577 users who are considered successful, only 3,321 have successful tweets that have been posted by themselves, the rest of these users can be considered as "artificial" successful users since they only received a positive user engagement by retweeting, but not on their own tweets.

2.2 About User Engagement as Evaluation

This challenge focuses on a different way of evaluating models and recommendations. The goal of the algorithm is to determine the best ranking for each user after the engagement that each of his tweets receives. The user engagement for one tweet is calculated as the sum of the number of times it has been retweeted and the number of times it has been marked as favorite. In order to measure the performance of this ranking, the information retrieval measure used is the Normalized Discounted Cumulative Gain (NDCG) [7], computed on the top 10 elements for each user. The overall evaluation is obtained by averaging the NDCG@10 of each user.

However, following the previous section about characteristics of the dataset, some details have to be mentioned concerning the use of the NDCG measure. Firstly, most of the users don't have any successful tweet among their tweets. For these users, any ranking of the tweets is equivalent since all items have the same relevance. Secondly, if we restrict ourselves to remaining users who have at least one successful tweets, it is also possible that any ranking will give the

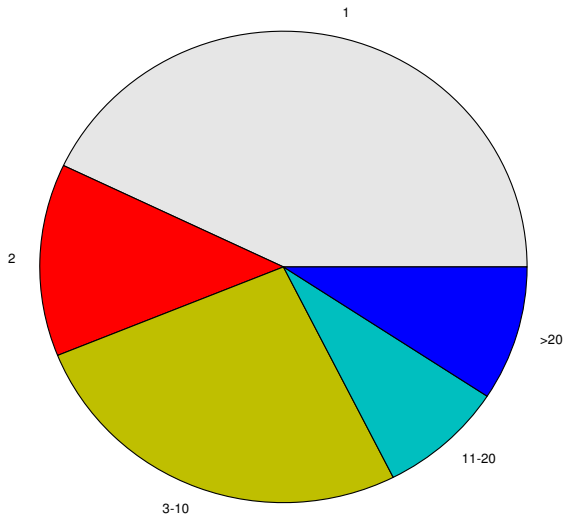


Figure 2: Distribution of the number of tweets per user in the training dataset.

maximum NDCG score. This case can happen if the user only has one tweet in total (cf. Figure 2), or if all of his tweets have the same positive user engagement, for example a user with three tweets with an engagement of 1. Then the ranking provided by the model would not have any consequence on the NDCG score for this user, and the NDCG would always be 1.0 since any ranking would be considered perfect.

Overall, 2,792 users provide a clean ranking (after removal of the retweet effect). While these users represent only 13% of the total number of users, they still gather 44% of the whole tweets, and 8% of their tweets are true successful tweets.

2.3 Input Features for the Model

We use several features that seem relevant to predict the ranking or the user engagement of a tweet. Each of these features is contained in the dataset or is extracted separately to enhance the model. Each of these features enters into one of three categories: user features, movie features, and tweet features.

User features are given in the original dataset. These features include the number of followers, the number of friends, the number of tweet put as favorites, the number of statuses posted and the number of lists in which the user is included. User features can change through time, since a user might follow more people or get more followers, and feature values for each tweet are the one extracted at the exact time the tweet was posted.

As for the movie features, we extract some features from IMDb website such as the IMDb rating, the IMDb votes, the budget of the movie or its release date. Since people constantly rate movies on IMDb, the IMDb rating or the number of votes keeps changing, and the values taken in our model are the one available at the time we extracted it. However, they are still significant in that these values provide the model a way to distinguish between popular or unpopular movies, as well as between acclaimed or criticized movies.

Finally, we include also the features that are related to the tweet itself, such as the rating given to movie by the user, the date the tweet was posted, the time difference between the release date of the movie and the post of the tweet, or other information about hashtags, lang, retweet, image inside the tweet...

3. METHOD

After a brief description of each method used, we discuss how to combine different rankers, and then explain our approach.

3.1 LambdaMART Model

LambdaMART [2] uses a listwise approach: it considers the whole lists of items as instances in learning, and tries to optimize directly a performance measure.
LambdaMART has been created by combining two previous algorithms, MART and LambdaRank. MART is a pointwise approach based on a boosted tree model in which the output of the model is a linear combination of the output of a set of regression trees. It can be viewed as performing gradient descent in function space using regression trees. LambdaRank is a method based on neural networks, which expresses gradients based on the ranks of the documents, and modifies the weight in the neural net according to these gradients. The λ terms in LambdaRank can be seen as rules defining how to change the ranks of items in a ranked list in order to optimize the performance. Gradients of costs to optimize directly a performance measure are hard to compute since these measure are non-differentiable. Instead, the λ terms are considered to be gradients with contributions from all other items that have a different relevance label. LambdaMART combines both approaches by using the idea of λ terms from LambdaRank and MART's boosted regression trees. LambdaMART models have shown great efficiency in ranking problems and won the Yahoo! Learning to Rank Challenge [3].

3.2 Random Forests

Random Forest [1] is a kind of ensemble learning algorithm which combines predictions from an ensemble of random trees. Bagging is used to reduce the correlation between each pair of random trees in the ensemble. Compared to LambdaMART, the Random Forest method belongs to pointwise methods as it is a regression model. Each of the trees in the ensemble forest votes for the output value, and the predicted output is then determined by all the trees in the ensemble. This method has demonstrate high performance and has been applied successfully in various different fields, including LTR competitions [5].

3.3 Description of the Approach

Here we describe the models we use in our overall approach, and how we combined them. At first we built a LambdaMART model on a modified train dataset, in which we removed users who would have the same NDCG without regard to the ranking given by the algorithm. As a consequence, the training dataset for LambdaMART is very small. We tried to artificially augment the dataset as showed in [3] by sampling some percentage of tweets for each user and inserting these data in the training set. For example,

instead of learning the ordering A>B>C for three tweets, sampling from these tweets and removing B help the model not to overfit and manage to learn that A>C without the condition of B lying between them. However, training on augmented models did not show any improvement on the evaluation. This is due to the characteristics of the data where only very few tweets, are successful for a given user, i.e. in most cases, the ranking problem is reduced to identify that one or two tweets will generate higher engagement than other tweets. These characteristics are highlighted by the almost perfect score of the binary oracle.

Such result encourages to also consider simpler algorithms, such as regression models. In Section 4 we present the results obtained with Linear Regression, and with Random Forests. One advantage of regression models is that they allow to easily correct the effect of retweeted tweets: (i) remove the retweet count of the original tweet from the features, (ii) while learning the model, modify the user engagement by subtracting the retweet count of the original tweet, and (iii) add the retweet count of the original tweet to the result returned by the learned model. In other words, the user engagement of any tweet that was not originally posted by the user, is reduced to its "true" user engagement, by dropping the retweet count that actually belongs to the user who posted the original tweet. At the contrary, the effect of such cleaning is less clear while using LambdaMART. In fact, LambdaMART model returns a value that does not target the engagement score as it focuses on the ranking of tweets. Hence, adding the retweet count of the original tweet to that value is meaningless.

Finally, we also explore various ways to combine these models. It is usually not an easy task to decide which method to keep in the final model if several methods perform similarly well, and combining rankers can be a way to minimize the chance of achieving poor results on an eventual new test dataset, because the diversification of methods will provide more robustness to the final model. Since the values returned by LambdaMART and regression models are not at the same scale, we standardize both predictions. After this step, the combination of rankers can be done in various ways. We apply both a simple averaging method, and a method to perform an optimal linear combination of rankers described in [12]. Given a pair of rankers who respectively gave the score s_1^i and s_2^i for a item i, the idea is to combined them convexly as:

$$s_{comb}^i = \alpha s_1^i + (1-\alpha) s_2^i \quad (1)$$

where a parameter α is sweeping from 0 to 1. By enumerating all values of α for which the NDCG will change, it is possible to identify the value of this parameter for which the two rankers are linearly optimally combined.

4. EXPERIMENTS

In this section we partially answer two questions:

- While the challenge is a Learn To Rank problem, should we learn a ranking model or not?
- Is there an efficient way to combine ranking models with other models?

These questions are answered in the light of the results obtained on the data of the challenge. We consider both (i) the NDCG score obtained by the models discussed in the paper and (ii) the importance of features in the learned forests.

Table 3: NDCG@10 on test dataset

Model	NDCG@10
Retweet	0.806
λ-MART	0.838
RF / WrapRF	0.823 / 0.858
Lin / WrapLin	0.806 / 0.843
Mean(λ-MART, Retweet)	0.876
Mean(λ-MART, WrapRF, WrapLin)	0.874
OptAvg(λ-MART, WrapRF, WrapLin)	0.876
OptAvg(λ-MART, RF, Retweet)	0.878

Hyper-parameters of simple models are selected through 10-fold cross-validation on the training dataset, with NDCG as objective function. These parameters include the number of trees, the number of leaves in each tree, and the learning rate of LambdaMART. Parameters found were 500 for the number of trees, 10 for the number of leaves and a learning rate of 0.05. We chose for Random Forest algorithm to use 2000 trees with a maximum depth of 5.

4.1 Experimental Results

While this strategy is subject to overfitting, models are compared after their NDCG@10 score on the test set. Results are given in Table 3 where *λ-MART*, *RF* and *Lin* stand respectively for LambdaMART, Random Forest and linear regression. *Retweet* is the model which predicts as a score the number of retweets of the original tweet (0, when the tweet is an original one, not a retweet).

The suffix *Wrap* indicates that the corresponding regression model is used on dataset for which retweet effect has been cleaned. Specifically, during the training phase, the number of retweets of the original tweet is removed from the features and is subtracted to the user engagement. During the test phase, this number is added to the engagement predicted by the learned model.

Finally, *Mean* stands for the meta-model which uses the average predicted score to rank tweets, and *OptAvg* stands for the meta-model using the best linear combination[1]. Notice that these best linear combinations are selected after the NDCG@10 score on the test set.

The first remark on the results is the importance of the retweet score. For example, this score alone is enough to reach an NDCG of 0.806. Similarly, the combination of Retweet with other models (through linear combination or wrapping) increases the NDCG score of these models to the extent of 0.035. Finally, any of the best models uses the retweet score.

The second important remark is that a ranking model is also needed to achieve the best NDCG score. LambdaMART is used by any model with an NDCG greater than 0.87, and the simple solution *Mean(λ-MART, Retweet)* has almost the best NDCG score. However, the wrapping strategy allows regression models to outclass LambdaMART. This suggests that ranking model are more promising as soon as the data are cleaned from the retweet effect.

[1]The best linear combinations are
0.48×λ-MART + 0.08×WrapRF + 0.44×WrapLin and
0.3×λ-MART + 0.12×RF + 0.58×Retweet

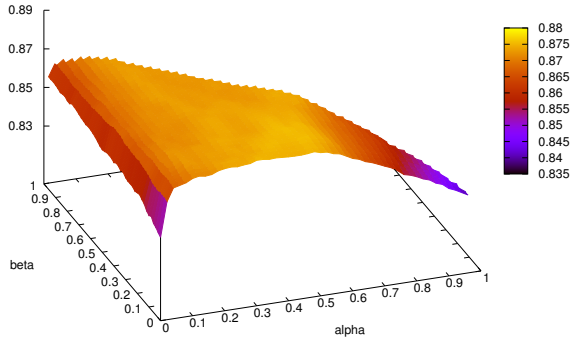


Figure 3: Test NDCG@10 after the linear combination of LambdaMART, WrapRF and WrapLin. Their weight is respectively α, β and $1 - \alpha - \beta$.

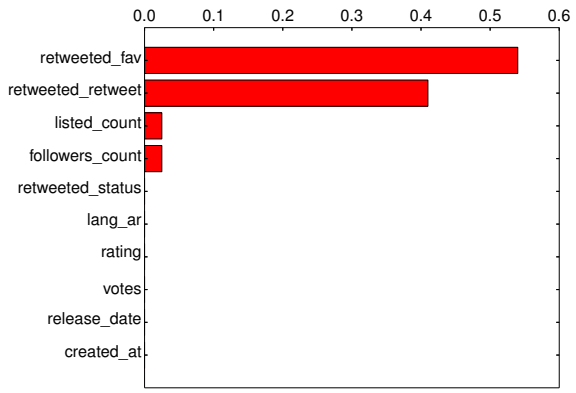
Notice also that the linear combination of simple models works surprisingly well. LambdaMART is a ranking model, but it is based on a boosted Forest which predicts the inclination for tweet to have a better score than an other one. Then, LambdaMART mixes well with Random Forests or with Retweet.

Finally, Figure 3 gives the NDCG after the linear combination of LambdaMART, WrapRF and WrapLin. We observe a large plateau of NDCG around the arithmetic mean. Moreover, the careful selection of the linear combination of simple models only increases the NDCG to an extent of a few thousandth. This tiny increase in NDCG, and the large plateau of equivalent linear combination indicates that the arithmetic mean is a safer approach. A similar analysis of OptAvg(λ-MART, RF, Retweet) leads to the same conclusion.

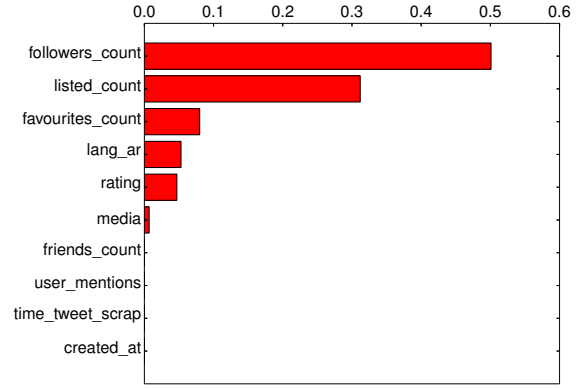
4.2 Relevant Features

The relevance of features has been measured on different sets of Random Forests to observe which effect has the removal of one or several features. In order to measure the relevance, we can use the relative depth at which the feature appears in a tree. A feature which appears at the top is contributing more to the final prediction as there is a larger fraction of input samples going through this node. The estimate used to measure the importance of a feature is thus the expected fraction of the samples to which this feature will contribute. Figure 4a presents the feature relevance on original data: we keep all input features, including the retweet count of the original tweet, and we do not modify the user engagement of the tweet. We notice in this case the features related to the original tweet, such as the number of favorites or retweet of the original tweet are the features contributing the most to the prediction. This contribution again highlights the importance of the original retweet count to build a efficient model.

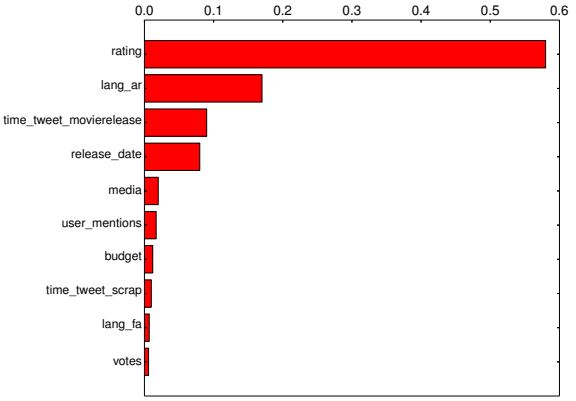
Figure 4b exhibits the feature relevance on data cleaned from the retweet effect: the retweet count of the original tweet is removed from input features and user engagement is modified as mentioned previously. We observe that after removal of the retweet effect, the Random Forest makes a



(a) Relevance of features using all features and unmodified user engagement.



(b) Relevance of features after cleaning the retweet effect.



(c) Relevance of features after cleaning the retweet effect and removing user features.

Figure 4: Relevance of features.

prediction about the user engagement mostly based on user features, such as the number of followers, the number of favorites, or the number of lists in which the user is. A user who has a higher number of followers or is included in more lists is more likely to receive higher engagement on his tweets. However, focusing mostly on user features sounds inappropriate for the challenge, as the tweets to rank are always emitted by the same user. While comparing two tweets, the only difference in user properties comes from the fact that both tweets were not emitted at the same date.

In the last case (cf. Figure 4c), we evaluate the importance of all features except user features. Since the NDCG score is computed for each user and user features seem not to change a lot throughout time, it would be interesting to observe which movie or tweet features are influencing the regression prediction. Remark that the most relevant features sound reasonable: the rating given by the user in the tweet, one language feature (arabic), the time passed between the release of the movie and the tweet post, the date at which the movie was released.

5. DISCUSSIONS

We discuss on this section some important remarks about the dataset. Several temporal aspects can influence the model prediction and evaluation. Firstly, the time interval from which the tweets have been posted for both the train set and the test/evaluation set are strongly different, since the train set contains tweets posted over a period of 10 months while the two others contains only a time interval of approximately one month. This leads to some questions about the effects of some features. For example, a user has more chance throughout a long period of time to gather new followers who could bring potentially a higher engagement on his future tweets. We observed during the study of relevant features that user features were bringing a lot of information to the Random Forest, and such difference of time interval between the datasets can have a large impact on the prediction.

Moreover, if we look at movies corresponding to tweets with the highest engagement, we mostly find popular movies which have been released recently at the time the tweet was extracted. Taking a small interval of time for a dataset can also modify the structure of data on this part since popular movies are often released at a specific period of time during the year. Regarding how the different datasets have been built, such information cannot be taken into account.

A sequential approach of Twitter problems would probably be more appropriate since both the evolution of a user or the release of movies can be seen as sequential problems.

6. CONCLUSIONS

Recommendation problems are *learn to rank* problems: you have to identify the best items. However, the greatest part of recommendation systems aim at predicting the importance of each item, which corresponds to a regression problem. The current challenge gives us the opportunity to build a recommendation system which directly identifies the best items.

Our analysis on that challenge leads to two main conclusions. Firstly, the best approach builds upon a ranking strategy: LambdaMART. However LambdaMART only reaches its full potential after the removal of the retweet effect. Secondly, LambdaMART and Random Forest are close enough to be combined through linear regression.

Given the high success and potential of Twitter or IMDb, it is surprising to work with so few relevant tweets on the dataset (tweets that have been retweeted or put as favorite at least one time). If we leave aside the retweet effect, the scarcity of original successful tweets makes it difficult to be certain about the robustness of any approach on this dataset. We hope that the end of the challenge will allow to build a bigger dataset, from which a deeper analysis could be done.

7. ACKNOWLEDGMENTS

This work was supported by Ministry of Higher Education and Research, Nord-Pas-de-Calais Regional Council, FEDER through the *Contrat de Projets Etat Region (CPER) 2007-2013*, and Hermes project of the *Pôle de compétitivité PICOM*.

8. REFERENCES

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

[2] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[3] C. J. Burges, K. M. Svore, P. N. Bennett, A. Pastusiak, and Q. Wu. Learning to rank using an ensemble of lambda-gradient models. In *Yahoo! Learning to Rank Challenge*, pages 25–35, 2011.

[4] J. Cai, E. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[5] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Yahoo! Learning to Rank Challenge*, pages 1–24, 2011.

[6] S. Dooms, T. De Pessemier, and L. Martens. Movietweetings: a movie rating dataset collected from twitter. In *Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys*, volume 2013, 2013.

[7] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[8] A. Karatzoglou, L. Baltrunas, and Y. Shi. Learning to rank for recommender systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 493–494, New York, NY, USA, 2013. ACM.

[9] D. Loiacono, A. Lommatzsch, and R. Turrin. Recsys challenge 2014: Learning to rank. 2014.

[10] A. Said, S. Dooms, B. Loni, and D. Tikk. Recommender systems challenge 2014. In *Proceedings of the eighth ACM conference on Recommender systems*, RecSys '14, New York, NY, USA, 2014. ACM.

[11] G. Takács and D. Tikk. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 83–90, New York, NY, USA, 2012. ACM.

[12] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.