Dataiku's Solution to Yandex's Personalized Web Search Challenge

Paul Masurel
Software engineer at Dataiku
paul.masurel@dataiku.com

Kenji Lefèvre-Hasegawa
Ph.D. in math
kenjil@free.fr

Christophe Bourguignat
Engineer
christophe.bourguignat@yahoo.fr

Matthieu Scordia
Data Scientist at Dataiku
matthieu.scordia@dataiku.com

ABSTRACT

Our team won the first prize[1] at the Personalized Web Search Challenge organized by Yandex, on Kaggle. This paper explains our solution as well as our workflow as a team of data scientists.

Keywords

Kaggle, Yandex, Web-search, Learning-to-rank.

1. INTRODUCTION

At the end of 2013, Yandex organized a Learning-to-Rank competition on Kaggle. Dataiku[2] offered to sponsor our team for this contest. Our solution reached first prize and writing this workshop paper was one of the Winners' duty.

The reader is warned that this paper should not be considered as proper research on Machine Learning.

We are a team of engineers, gourmet of low hanging fruits. Hence all the algorithms, their implementations, as well as almost all of the features in this paper have been directly picked from available bibliography.

Hopefully, this paper will contain sufficient information to reproduce our results, and we will try to make it transparent when our choices were time-driven rather than following proper scientific method.

2. RELATED WORKS

In recent years, personalization of various aspects of search has been the subject of many papers. Adapting search results to the user can be sometime done by taking into account its straightforward specificities (e.g. location, prefered language). Others try to model the user's interests to predict his future searches result's relevance. In the review below, we won't cover exhaustively the strategies that the information retrieval researchers community have produced to capture searcher's interests. Instead we will focus on the papers that we have used while competing for Kaggle/Yandex Challenge. Most papers cited here were suggested for reading by the challenge organizer.[24]

Clicks' feedback

Clicks are very informative to improve the quality of search both for evaluating the quality of a ranking and for modeling user's interests. The satisfied-clicks (SAT-click) introduced by Fox et al. [12] have now become a standard in the quality evaluation of personalized search systems. On the prediction side, ranking by learning from the implicit feedback gathered from clickthrough data outperforms static search engines [14] [18].

When to personalize

In [22], Teevan et al. have shown that not all queries can benefit from personalization. For instance, queries with very low click entropy, i.e. queries for which most people click on the same returned URL, won't be improved by personalization. An easy example fo such queries is given by navigational queries (straightforward queries aimed to find a unique URL e.g. "new york times"). Up to 25% of all queries are of navigational nature [4]. For other queries, results pertinence is more hard to interpret and different people might find different results relevant for a same query. Teevan et al. [22] give some features based on the query alone that can predict such situation.

Personalizing search have proved very effective dealing with user's repeated searches. It has been shown [8] that well over half of all web pages that a person visits is in fact a revisitation. Actually search engine are very often used to refind a document that the searcher has visited in the past. This fact supports the use of the user's past activity to track repeated combinations of searcher's queries, results and clicks to improve search results on future similar queries.

Past interaction timescales

Repeated patterns in user's past interactions with respect to topics, URL and domain can be used to model user's

[1]First Prize, but second place! First place was held by Yandex team, which was however declared out-of-competition.
[2]Dataiku is a software startup. Our product, the Data Science Studio, is a software platform for data scientists. Taking part in this contest was an opportunity to test it.

WSCD workshop '13 New York, USA

interests. Supposing that for a given user, his interests are constant enough, then *long term* past activity [21] is handy to personalize his search. On the other hand, if the searcher has a sudden change in his interests, then long term data isn't helpful and it is more accurate to see what the user has done recently, for example by checking searcher's current session past activity (*short term*). See [7] [6] [17] [23] for session impact studies. Identifying these sudden changes in search activities regarding the user's former long term past interactions is done in [11].

How deep these different timescaled collected feedbacks interact to model user's interests is little known. Teevan et al. give in [3] a unified framework to represent user's interests (topical and URL) handling both timescales (short and long). Allowing the ranker to learn weights for short term features, long term features and the combination of these improves the result significantly. They also showed that long term features have weights that decrease as the search session goes.

Search behaviors beyond clicks

Most paper cited above limit users' past activity to clicks. But one can actually capture a lot more informations beyond clicks. For example, the reason why the searcher did not click on a URL is also valuable information. Thus click omition is also a very precious feedback.

But the likelihood that a user will click or not on a URL is hard to interpret, as it is the result of many causes. A url shown at the top of the result page is more likely to be seen. This means that searcher's feedbacks are highly biased by the influence of the rank and that they can't be taken as an absolute signal. Also the user is influenced by the relevance of the snippet computed by the search engine page [10]. Fortunately, relative preferences derived from search behavior can be fairly accurate. [15].

In order to simplify this complexity, we formalize the search interactions by using the *Examination hypothesis* that states that the user has examined a snippet and taken into account its attractivity to decide whether to click or not, together with the *Cascade hypothesis*[9] that states that all URLs above a clicked URL have been examined and all URLs below the lowest clicked URL haven't been examined. Following [19], we will call the former *skipped* URLs and the latter *missed* URLs.

Learning from all repeated results

Shokouhi et al. [19] demonstrate that results repeatedly shown to a user are not only due to the use of the search engine to find again a document visited in the past. In fact, repetitions often occur in the process of searching for informations. Shokouhi et al. showed that depending on the user past interactions with repeated results, and the details of the session, URLs must sometimes be promoted when newly encountered, while in some other cases they should be demoted. Using the skip click miss-based model of user's interests they positively reranked results for the user. By doing so they successfully fought the amnesia of search engine on repeated results.

3. CHALLENGE DESCRIPTION

3.1 Datasets and task

For this contest, Yandex supplied two search log datasets, both sharing the same format. These logs are organized into sessions. In a session, the user may perform one or more queries. For each query, the log format shows the search query entered by the user, a list of query terms, a timestamp, and a list of $(url, domain)$ pairs sorted according to a non-personalized ranking algorithm. Finally we know on which urls the user clicked.

- The **train** dataset contains 27 days of data (16GB) or about 35 millions sessions, used to learn about user preferences.
- The **test** dataset contains 800,000 sessions picked in the next 3 following days. Each of these *test sessions* stops at a *test query*, for which no click information is given. The goal of the contest is to improve the order in which url are displayed to the user in the test sessions.

Every piece of information in this dataset : url, domains, user, queries, and query terms were replaced by an ID. While this heavy anonymization makes it difficult to grasp proper insight from the data, doing otherwise was not an option for Yandex. In 2006, AOL released a log where only user ids were anonymized [1]. The websphere rapidly started identifying some of the users appearing in the log, just by looking at the queries they performed. The story ended up with a class action against AOL.

3.2 Quality metric

Yandex uses the time elapsed between a click and the next action (click or query), or dwell-time, to associate a score of 0, 1, or 2 for each click. If the dwell-time is less than a given threshold (50 units of time given by Yandex) or if the url has not been clicked at all the url score is 0. If dwell-time is between 50 and 300 units of time, the score is 1. Finally if the dwell-time is more than 300 or if it is the last click of the session the associated score is 2. When a URL is clicked many times the highest relevance is retained.

The score of the solution is the mean of the Normalized Discounted Cumulative Gain [13] or NDCG over the test sessions. NDCG is defined as follows.

For a given test query, the search results consists on a list of 10 urls for which we have estimated a list of as many relevancies $r \in \{0, 1, 2\}^{10}$. Re-ranking consists on applying a permutation $\sigma \in \mathcal{S}_{10}$ where $\mathcal{S}_{10}$ is the symmetrical group on 10 elements.

The DCG associated to a permutation σ and a list of relevance r is then defined by

$$DCG(\sigma, r) = \sum_{i=1}^{10} \frac{2^{r_{\sigma_i}} - 1}{\log_2(i+1)} \quad (1)$$

The DCG reach its maximum value $DCG_0(r)$ when the permutation σ sorts the url by decreasing relevancies. The NDCG is the normalized version of DCG_0. It takes values in $[0, 1]$.

$$NDCG(\sigma, r) = \frac{DCG(\sigma)}{DCG_0(r)} \quad (2)$$

4. TEAM'S METHODOLOGY

4.1 Our Data Flow / Work Flow

We split up the 27 days of historical data into :

- **A history dataset** : 24 days of historical data similar to Yandex **train** dataset. It is used to learn our user preferences, and build our feature dataset.
- **A learning dataset** : 3 days of test data similar to Yandex **test** dataset. This dataset can be used for our supervized learning as it is labeled. These three days amount for $1,216,672$ sessions.
- **Satisfaction labels** for our 3 days test dataset.

Once the split is done, a set of recipe takes the train and test dataset and build a feature dataset. This feature dataset consists on a table with $12,166,720$ rows. For each row in the dataset we have a session id, a URL, and all computed features. The list of features we investigated are described in Sec.4.2.

The feature dataset can then be joined again with labels and used for supervised learning.

The separation of the label dataset was a solid way for us to make sure that the features were not tainted in any way by the labels. Also, with this setup submitting solution for the contest was easier : once the model was learnt, we could use Yandex original **train** and **test** datasets in place of our **history** dataset and **learning** dataset, create a feature matrix for these datasets, and use the learnt model to compute our personalized order.

We made sure to cherry-pick test sessions using the same rules as Yandex to make sure we didn't introduce any bias in our selection. Even then, we couldn't reproduce the baseline of this contest on our **learning** dataset. Yandex announces a NDCG of 0.79056 for its original ranking, while we measure a NDCG of 0.798 on our test set. This difference cannot be explain by the simple variance of the test set selection. Seasonality would also tend to confirm the gap rather. As of today we still cannot explain this difference.

Dataiku's Data Science Studio organizes dataflow as a graph of datasets and recipes to build new datasets (See Fig.1). Note that in order to work in parallel, we built all families of features in as many independant recipes, before merging all of them together into one big feature file.

4.2 Features

4.2.1 *Non-Personalized Rank*

The main feature is most probably the *non-personalized rank*, the rank at which the url is displayed in the original result list. It contains two extremely valuable pieces of information. First, it is our only proxy to pagerank, query-document similarity, and all other information Yandex could have used to compute its original ranking.

The second piece of information is a side-effect of an approximation made in this re-ranking exercise. The contest score is computed against the clicks that have been observed while using the personalized rank, and user behavior is very strongly influenced by the order of the urls in the result page. This introduces a very important bias in the whole exercise that we need to take in account in our solution. Some of the features even have the specific role of inhibiting or promoting the importance of the non-personalized rank.

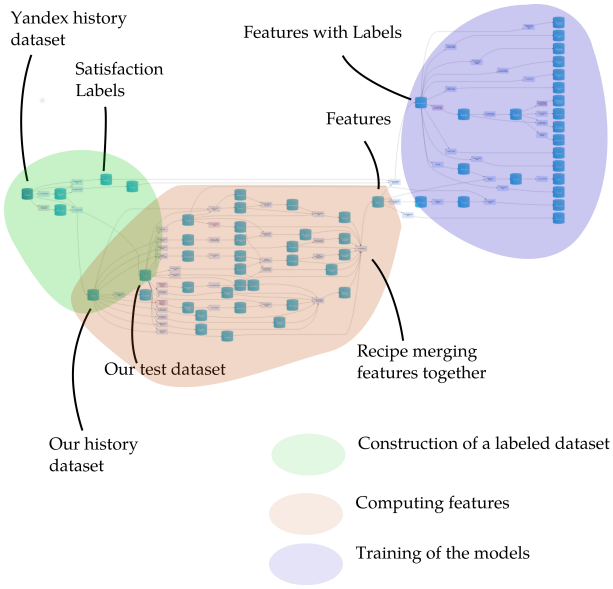

Figure 1: Our Dataflow (screenshot of Dataiku's data science studio). On the left, Yandex history dataset is split to construct our history dataset, our learning dataset and our labels. Feature are constructed in the middle, and merged together. Finally features and labels are re-merged for training.

4.2.2 *Aggregate features*

For each test query, and for each url displayed in these result list, we will consider a specific set of predicates to filter the logs and produce 11 features[3] on this subset.

The predicate we considered are defined by the conjunction of conditions on

- the url (same url or same domain),
- the user submitting the query (anyone, or same user),
- the submission timescale (current session, anterior historic, all historic),
- the queries submitted (any query, same query, superset of terms, subset of terms)

We didn't compute the whole possible cube of predicates. The specific combinations we investigated are described in the two following subsections *User-specific aggregate features* and *Anonymous aggregate features*.

For all the predicates considered, we will compute a vector of the 11 features described below.

Miss, Skip, Click0, Click1, Click2 conditional probabilities

Any URL returned by Yandex consecutively to a query submission is either

- Clicked, with a satisfaction 0, 1, or 2
- *Skipped*, when the snippet has been examined by the user but URL was not clicked,

[3] 1 overall counter + 5 conditional probabilities + 4 MRR values + 1 snippet quality

- *Missed*, when is the snippet was not examined by the user.

We use the *Cascade Hypothesis* which states that URLs under the last clicked URL are all missed and that URLs above any clicked URL is either skipped or clicked.

Note that this classification is stricly finer than the $\{0, 1, 2\}$ relevance labels used by yandex in the NDCG score. Namely their label 0 corresponds to the union of Miss, Skip and Click0, their label 1 to Click1 and 2 to Click2.

Given a filtering predicate $\mathcal{P}$, we compute a consistent set of conditional probabilities for the outcome of the display of a URL.

Examples of such filtering predicates could be

- Display of url u, upon query q, that took place earlier within session S,
- Display of any url belonging to domain d, that took place earlier before session S,
- Display of any url belonging to domain d, that took place earlier before session S,
- Display of any url, upon query q

Ideally the probability could be estimated by the ratio of the number of times we observed an outcome ℓ in the subset of the logs that match predicate $\mathcal{P}$.

But depending on the predicate we may lack data to estimate this probability accurately. The subset considered may even be empty, and frequency undefined.

To elude this problem, we use *additive smoothing* with the assumption [4] that all URLs have been missed once. That is we defined our estimated probability to reach outcome ℓ knowing that we match predicate $\mathcal{P}$ by

$$agg(\ell, \mathcal{P}) = \hat{P}(outcome = \ell | \mathcal{P}) = \frac{Count(\ell, \mathcal{P}) + p_\ell}{Count(\mathcal{P}) + \sum_{\ell'} p_{\ell'}}$$

Where $Count(\mathcal{P})$ is the number of url displays matching predicate $\mathcal{P}$, $\mathcal{P}$, $Count(\ell, \mathcal{P})$ is the number of times such url display's outcome was ℓ and p_ℓ represents our prior, and is defined by $p_\ell = 1$ if l is Miss and 0 otherwise.

Miss, Skip, Click, Shown MRR

The problem with the conditional probabilities alone is that they do not take into account the bias induced by the rank of the displays. Surely a URL returned in top rank has very high probability to be skipped or clicked whereas the same URL appearing at bottom rank has high probability to be missed. To give the opportunity to our ranker to capture the relative importance of the estimated probability above, we to need complete our set of features by the *Mean reciprocal rank* of the Miss, Skip, Click occurences. The MRR is defined as the reciprocal value of the harmonic mean on the rank of the URL. We used additive smoothing with a prior belief equivalent to assuming that any URL has been displayed at least once with a virtual rank which inverse value equals 0.283, as it is the expectancy of the MRR when the URL rank follows a uniform distribution.

[4]This prior has been chosen arbitrarily.

$$\widehat{MRR}(\ell, \mathcal{P}) = \frac{\sum_{r \in \mathcal{R}_{\ell,\mathcal{P}}} \frac{1}{r} + 0.283}{Count(\ell, \mathcal{P}) + 1}$$

with ℓ a label in Miss, Skip or Click, $\mathcal{R}_{\ell,\mathcal{P}}$ being the list of ranks of all the displays matching predicate $\mathcal{P}$ and with outcome ℓ.

Note that instead of working with Click0, Click1 and Click2, we merged these classes back into one single class, as once an url is clicked, the displayed rank is not likely to explain its satisfaction.

In addition to this, we compute $\widehat{MRR}(\mathcal{P})$ the estimated MRR in the subset of the log. This information is somehow averaging how the ranker of Yandex is returning the URL.

Snippet quality score

Along with the rank's bias, the click is highly impacted by the perceived relevance of a document when user reads the snippet in the page returned by Yandex.

Given a filtering predicate and a URL, we now construct a score reflecting the quality of the snippet of u by using implicit attractivity given by clicks feedback. The idea is to give a high score to the first clicked documents and a bad score to skipped documents. Suppose document u appears in the result set of a query q. We define the score $score_q(u)$ the following way. Since we follow the *examination hypothesis*, we can't score missed documents and we set $score_q(u) = 0$ when $u \in$ Miss. When u is clicked at click position p (the order in which documents are consequently clicked on the result page), we set $score_q(u) = 1/p$. In order to not give a score twice, we remove the redundant clicks by considering only scores of the first click on each document to define the position p. Skipped documents will be penalized by a negative score, opposite of the score of the last clicked document on the result page : $score_q(u) = -min(score_q(u) | u \in \text{Click})$. For example, if a document has been skipped on a result page with three clicked documents, we will attribute a score of $-1/3$. Now given a filtering predicate $\mathcal{P}$, the snippet quality score is defined as the ratio of all score gain or loss on the number of click and miss situation involving u:

$$sq_{agg}(u, \mathcal{P}) = \frac{\sum score_q(u)}{\text{Count}(\ell \in \{missed, skipped\}, \mathcal{P})}$$

where the sum is on all queries of the subset of the log matching $\mathcal{P}$, where u appears in their results.

User-specific aggregate features

Given a user, a query q and a URL u, personalization features are defined as the features for the 12 aggregate compounds determined by any combination of the following conditions:

- The query is issued by the same user,
- The timescales are current session, anterior historic or all historic,
- The url is either the same or the domain is the same,

- The query matches : all queries, the exact query q, any subquery of q, any superquery of q.

Subqueries (resp. superqueries) of a given query are queries that contains a subset (resp. superset) of its terms.

The feature based on the filtering predicate involving any subquery of q, or any superquery of q hasn't proved to improve our models. So we eventually worked with $1 \times 3 \times 2 \times 2 \times 11 = 132$ personalization features. Note that our features are close in spirit to features of R-cube [19] and of [3].

Anonymous aggregate features

For each test session, given the test query q and a URL u and its domain d in the hit results, anonymous aggregate compound are also computed for the three following predicates :

1. The domain is d
2. The url is u.
3. The url is u, and the query is q

That is $3 \times 11 = 33$ anonymous features.

4.3 Cumulative feature

If a url with a low rank is likely to be clicked, latter urls' likelihoods to be clicked are likely to be missed. It is therefore interesting to consider the values of the features of the urls of lower ranks. We therefore compute what we called cumulated feature by summing up the value of some of the aggregate features for url of lower ranks.

For instance, considering a filtering predicate $\mathcal{P}$, for an url with a non-personalized rank of 5, for the cumulated feature associated to clicks with satisfaction 2, we compute the sum

$$cum(\mathcal{P}, click_2, 5) = \sum_{rk=1}^{rk=4} agg(\ell = click_2, \mathcal{P}(url_{rk}))$$

4.4 Query's features

Some queries are very unambiguous and are therefore not good candidates for re-ranking. For this reason, we also included query click rank entropy as described in [22].

It is also interesting to assess the complexity of a query. We therefore also used

- the number of terms in the query,
- the number of times it was queried for,
- its average position in the session,
- its average number of occurence in a session,
- the MRR of its clicks

4.5 User click habits

Users have different habits when using a search engine. It is very important for instance to measure how much a user is influenced by the non-personalized rank. For this reason we included both the user-rank click entropy, as well as three counters[5] of how many times the user clicked on urls in ranks within respectively $\{1, 2\}$, $\{3, 4, 5\}$ and $\{6, 7, 8, 9, 10\}$. We also compute the user's number of terms average, as well as the average number of different terms used in a session.

Finally we added the total number of queries issued by the user.

[5]These counters have not been smoothed, or normalized by lack of time.

4.6 Session features

4.7 Collaborative filtering attempt

Precedent features do not give any clue about results for which the user had no interaction in the past[6]. We therefore had great expectations in enriching the user history by taking in account the history of similar users.

We investigated rapidly a technique commonly used in collaborative filtering : the values given by the FunkSVD algorithm applied on a matrix of interaction between users and domains. It amounted for a marginal increase of respectively 5.10^{-5} and 2.10^{-5} on the public and the private leaderboard of the contest.

As of today, we do not know whether to blame our lack of effort for this poor result or whether collaborative features are actually too loose to allow for re-ranking.

We redirect the reader to [20] for a more serious approach of personalizing web search using matrix factorization.

4.8 Seasonality

We noticed a strong workweek/weekend seasonality, in research keywords, domain clicks, and even NDCG score of Yandex unpersonalized rank. This helped identify that Day 1 was a Tuesday. We exploited rapidly this information by adding a domain seasonality feature.

We compute a factor expressing the propensity of a domain to be clicked during weekend :

$$weekend(domain) = \frac{5\,(1 + nb\ of\ clicks_{weekend})}{2\,(1 + nb\ of\ click_{workweek})}$$

A ratio of 1 indicates no seasonality. The greater the ratio, the more the domain is clicked during weekends. We then created a feature that takes for value $weekend(domain)$ during weekend, and its inverse during the workweek.

5. LEARNING-TO-RANK

5.1 Point-wise approach

5.1.1 Principle

We first approached this problem using a point-wise approach.

Point-wise algorithms typically rely on either regression or classification methods. Our point-wise approach was based on the classification of the display of one url into one of the three classes used by Yandex's scoring system.

- click of satisfaction 0 or no click
- click of satisfaction 1
- click of satisfaction 2

Sorting out urls according to the score of their respective classes optimizes NDCG in case of a perfect classifier. However assuming a perfect classifier is a bit too strong as our

[6]At least at the domain level

feature are most likely not even sufficient to reach it. As pointed out by [16], such a way to sort urls is highly unstable. Instead of using our classifier for classification per-se, we use it to produce probability estimates.

In order to rank our URLs we made the assumption that we had trained a Bayes classifier : in other words, we assume that our classifier outputs perfect probabilities conditional to our set of features.

Under this hypothesis, we show in Appendix A how to sort the result urls in order to maximize the NDCG metric. We rapidly settled for Random Forests (RF) as the classification algorithm.

We also tried to improve this approach by classifying on five classes rather than three. Indeed, there are actually three different ways a URL display can score 0.

- Missed URLs
- Skipped URLs
- Clicked with a satisfaction 0

These three cases maps to their own specific domain in the feature space. It seems intuitively easier to linearize our data for classification on the resulting five classes rather than on the original three classes. The benefits when using Random Forests was however uncertain. Our experiments showed a small unconcluding increase in our score when classifying on 5 classes, and we chose to keep this approach without further investigation.

5.1.2 Optimizing the hyperparameter

The minimum number of samples per leaf was chosen directly to optimize the NDCG. This lead us to larger values for our hyperparameter than typical classification problems (between 40 to 180 depending on the set of features selected).

Empirically NDCG appeared to be an unimodal function of our hyperparameter, which made it possible to use the golden section algorithm in place of the usual grid search.

We didn't observe any improvement of our score over 24 trees. Optimizing Random Forests typically took one hour on our 12 cores computer, which was fast enough to explore a great number of feature combinations.

5.1.3 Feature importances

Feature selection in Random Forests is still puzzling for us. Highly correlated features will share a high importance score in all the solutions we could find, making it difficult to notice that the noisiest ones should be removed.

We tried two approaches to test and select our features.

- Top-bottom approach : Starting from a high number of features, we iteratively removed subsets of features. This approach led to the subset of 90 features (See the Table in Appendix) that were used for the winning solution.
- Bottom-up approach : Starting from a low number of features, we incrementally added the features that produced the best marginal improvement. That approach gave us the subset of 30 features that lead to the best solution with the point-wise approach.

5.2 LambdaMART

LambdaMART is presented in [5] as the result of iterative improvement over Ranknet and then LambdaRank.

RankNet introduced the idea of aiming at a fictional score function, for which the probability that a page should be ranked better than another is the logistic function of the difference of their scores.

The gradient of the resulting cross-entropy cost can then be used to train a neural network to approach such a score function. Given a query, the analytical expression of this gradient can be broken down as a sum of as many force trying to pull/push each pair of urls in order.

In a second implementation of Ranknet, the algorithm speed was greatly improved by factorizing the contribution of all individual forces to the score of each urls. RankNet aims at minimizing the number of inversions and therefore belongs to the pair-wise category of Learning-To-Rank algorithms.

LambdaRank is an adaptation of RankNet which consists of tweaking the gradient expression to minimize an information retrieval score directly, or NDCG in our case.

Finally LambdaMart is using Gradient Boosted Trees in place of neural networks.

For our winning solution we used the implementation in Ranklib [2] trained on 90 features. The depth of the trees was set to 10 and its influence was not investigated. The final model contained 1165 trees.

5.3 Results

The results, pros and cons of both approaches are displayed in Table.5.3.

We unfortunately did not have time to try test other classification algorithm. Especially, we would have liked to confirm the results of [16], in which a point-wise approach with gradient boosted trees gave similar results as LambdaMART.

During our first experiments with LambdaMART, we got similar results as with our point-wise approach. But as the number of features grew, it started showing better performances than the random forests.

Unfortunately, LambdaMART relies on Gradient Boosted Trees whose training is by nature not parallelizable and takes a great amount of time to train a new model. For this reason we kept using the RF point-wise approach for exploratory purposes.

While NDCG metric makes it possible to compare two solutions, it is a little difficult to actually figure out whether users of Yandex actually see any difference. In Fig.3, we tried to visualize the actual search sessions satisfaction. Personalized ranking shows little improvement. Fig.4 makes the difference more visible by sorting the different sessions in lexicographical order.

We can however assume that both NDCG and this visualisation are suffering from the bias explained in Sec.4.2.1, and that an actual release would lead to much more visible results.

APPENDIX

A. MAXIMIZING NDCG WITH A BAYES CLASSIFIER

In this Appendix, we try to find out a strategy to find the reorder our urls given a Bayes classifier. Given a specific query, the list of their relevance scores $r \in \{0, 1, 2\}^1 0$ are unknown to us, and can hence be considered as random variables.

	Point-wise RF	LambdaMART
NDCG pub.	0.80394	0.80647
NDCG pri.	0.80458	0.80714
Rank	5th	2nd
#features	30	90
Parallelizable	Yes	No
Training time	$<$ 1 hour	30 hours
Other	24 trees. Min of 104 samples per-leaf.	1165 trees of 10 leaves

Figure 2: Performance comparison of the point-wise and list-wise approaches. In order to avoid overfitting, Kaggle discloses a public leaderboard, but uses a different test set at the end of the contest. Ranks were the same in the public and the private leaderboard.

Assuming we have trained a classifier to predict the probability ditribution of the relevance conditional to our set of feature $(p(r_i = s)|X)_{0\leq i\leq 10, s\in\{0,1,2\}}$.

We want to find the permutation $\sigma_0 \in \mathcal{S}_{10}$ that maximize our NDCG's expectancy. The normalizing denominator is constant, hence we focus on maximizing the DCG instead.

$$DCG(\sigma_0, r) = \max_{\sigma\in\mathcal{S}_{10}} E\left[\sum_{i=1}^{10} \frac{2^{r_{\sigma_i}} - 1}{\log_2(i+1)}\right] \quad (3)$$

We make the point-wise hypothesis and assume that, for each URL, we summarized all the relevant information in our feature vector X. The flaw of the point-wise hypothesis is that the relevance of previous URLs actually have a great effect on the probability for a URL to be clicked. For instance, if the URL at rank 1 was relevant enough, the probability for the user to click on URL at rank 2 anyway is likely to be lower. Fortunately, this effect is likely to have a low effect on the order of the URLs.

Using this hypothesis, Eq.3 becomes

$$DCG(\sigma_0, r) = \max_{\sigma\in\mathcal{S}_n} \sum_{i=1}^{10} \frac{1}{\log_2(i+1)} \mathbf{E}\left[2^{r_{\sigma_i}} - 1 \mid X\right] \quad (4)$$

In the DCG, the denominator $\log_2(i+1)$ is decreasing. The max is therefore obtained by sorting the urls by decreasing values of $E\left[2^r - 1 \mid X\right]$.

Thanks to our Bayes classifier, we can compute the conditional probability and expand this expectancy. In conclusion, given a Bayes classifier, an optimal way[7] to re-rank our urls is by decreasing values of

$$p(r = 1 \mid X) + 3p(r = 2 \mid X)$$

B. ACKNOWLEDGMENTS

We thank Dataiku for allocating time and servers for our team. Yandex for organizing this contest, and supplying

[7]This solution is not-unique as any increasing function of this score, leads to the same NDCG.

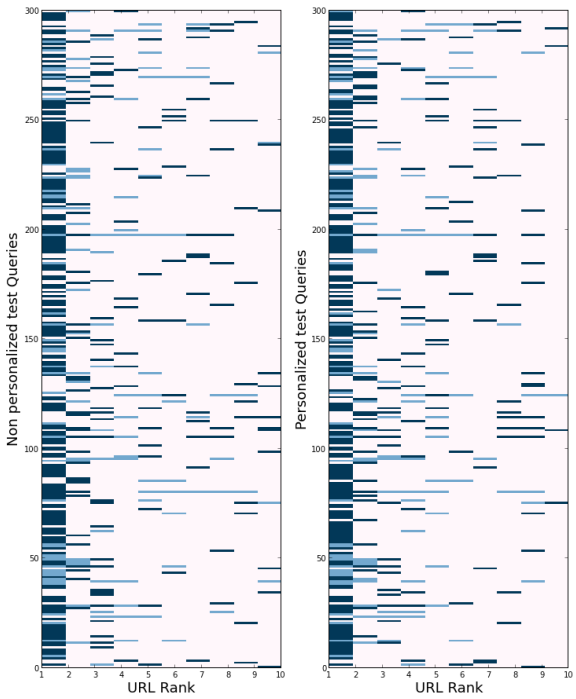

Figure 3: One thousands queries re-ranked. This figure shows the place of the clicks for 300 sessions. Darker gray stands for satisfaction 2, light gray stands for satisfaction 1. Left figure is showing Yandex original order (NDCG=0.79056). The right figure is showing our re-ranked solution (NDCG=0.80647).

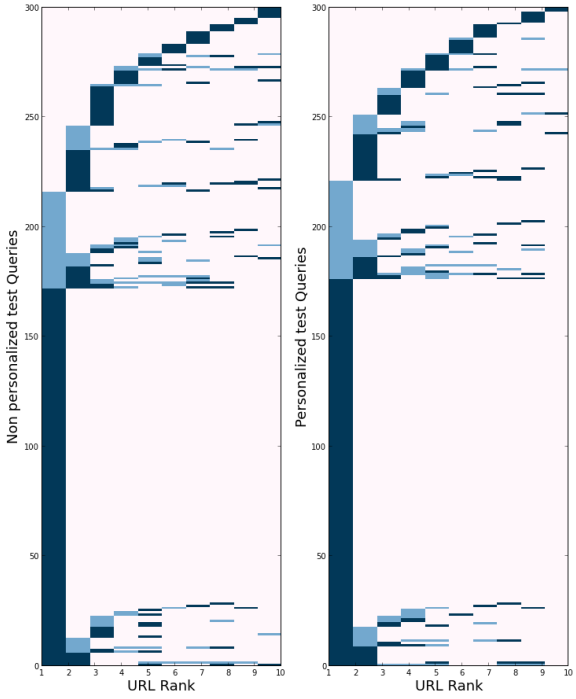

Figure 4: 300 sessions re-ranked. Same as Fig.3 except that sessions have been sorted lexicographically with their clicks to make histograms appear.

this very valuable dataset. We also thank Ranklib for its implementation of LambdaMART, scikit-learn for its implementation of Random Forests.

C. REFERENCES

[1] Aol search data leak. `http://en.wikipedia.org/wiki/AOL_search_data_leak`.

[2] Ranklib. http://sourceforge.net/p/lemur/wiki/RankLib/.

[3] P. N. Bennett, R. W. White, W. Chu, S. T. Dumais, P. Bailey, F. Borisyuk, and X. Cui. Modeling the impact of short- and long-term behavior on search personalization. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 185–194, New York, NY, USA, 2012. ACM.

[4] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, Sept. 2002.

[5] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research, 2010.

[6] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen, and Q. Yang. Context-aware query classification. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 3–10, New York, NY, USA, 2009. ACM.

[7] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 875–883, New York, NY, USA, 2008. ACM.

[8] A. Cockburn, S. Greenberg, S. Jones, B. Mckenzie, and M. Moyel. Improving web page revisitation: Analysis, design, and evaluation. *IT & Society*, 1(3), 2003.

[9] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 87–94, New York, NY, USA, 2008. ACM.

[10] G. Dupret and C. Liao. A model to estimate intrinsic document relevance from the clickthrough logs of a web search engine. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 181–190, New York, NY, USA, 2010. ACM.

[11] C. Eickhoff, K. Collins-Thompson, P. N. Bennett, and S. Dumais. Personalizing atypical web search sessions. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 285–294, New York, NY, USA, 2013. ACM.

[12] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, Apr. 2005.

[13] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, Oct. 2002.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

[15] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst.*, 25(2), Apr. 2007.

[16] P. Li, C. J. C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS*, 2007.

[17] L. Mihalkova and R. Mooney. Learning to disambiguate search queries from short sessions. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD '09, pages 111–127, Berlin, Heidelberg, 2009. Springer-Verlag.

[18] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 239–248, New York, NY, USA, 2005. ACM.

[19] M. Shokouhi, R. W. White, P. Bennett, and F. Radlinski. Fighting search engine amnesia: Reranking repeated results. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 273–282, New York, NY, USA, 2013. ACM.

[20] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen. Cubesvd: A novel approach to personalized web search. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 382–390, New York, NY, USA, 2005. ACM.

[21] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 449–456, New York, NY, USA, 2005. ACM.

[22] J. Teevan, S. T. Dumais, and D. J. Liebling. To personalize or not to personalize: Modeling queries with variation in user intent. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 163–170, New York, NY, USA, 2008. ACM.

[23] B. Xiang, D. Jiang, J. Pei, X. Sun, E. Chen, and H. Li. Context-aware ranking in web search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 451–458, New York, NY, USA, 2010. ACM.

[24] Yandex. Personalized web search challenge. `http://www.kaggle.com/c/yandex-personalized-web-search-challenge/details/prizes`, October 2013.

D. FEATURES USED IN THE FINAL MODEL

Table 1: Feature used in the winning model

feature family	number of features	description
unpersonalized rank	1	See Sec.4.2.1
aggregate features., same user, same domain (skipped, missed, click2, snippet quality)	4	See Sec.4.2.2
aggregate features., same user, same domain, anterior sessions (click2, missed, snippet quality)	3	See Sec.4.2.2
aggregate features. same user, same url (click mrr, click2, missed, snippet quality)	4	See Sec.4.2.2
aggregate features. same user, same url, anterior sessions (click2, missed, snippet quality)	3	See Sec.4.2.2
aggregate features. same user, same domain, same query (missed, snippet quality, missed MRR)	3	See Sec.4.2.2
aggregate features. same user, same domain, same query, anterior sessions (snippet quality)	1	See Sec.4.2.2
aggregate features. same user, same url, same query (MRR, click2, missed, snippet quality)	4	See Sec.4.2.2
aggregate features. same user, same url, same query, anterior sessions (MRR, click MRR, miss MRR, skipped MRR, missed, snippet quality)	6	See Sec.4.2.2
aggregate features. same user, same url, same query, test session (Missed MRR)	1	See Sec.4.2.2
Number of times the user performed the query	1	
aggregate features., any user, same domain (all 11 features)	11	See Sec.4.2.2
aggregate features., any user, same url (all 11 features)	11	See Sec.4.2.2
aggregate features., any user, same url, same query (all 11 features)	11	See Sec.4.2.2
user click entropy	1	See Sec.4.5
user click12, click345 and click678910	3	See Sec.4.5
user overall number of queries	1	See Sec.4.5
user query length average	1	See Sec.4.5
user session number of terms average	1	See Sec.4.5
query click rank entropy	1	See Sec.4.4
query length	1	See Sec.4.4
query average position in session	1	See Sec.4.4
query average occurences in session	1	See Sec.4.4
query occurences	1	See Sec.4.4
query clicks MRR	1	See Sec.4.4
query average number of clicks / skips	2	See Sec.4.4
cumulated feature. same domain, same query (click2, and skip)	2	See Sec.4.3
cumulated feature. same url, same query (skip, click1, click2)	3	See Sec.4.3
cumulated feature. same user, same url (skip, click1, clicks2)	3	See Sec.4.3
terms variety	1	Number of different terms used during the test session
collaborative filtering svd	1	See Sec.4.7
domain seasonality factor	1	See Sec.4.8