ICML Exploration
Exploitation challenge: Keep it simple!

Olivier Nicol, Jérémie Mary, Philippe Preux

JMLR: Workshop and Conference Proceedings vol 26 (2012) 62–85 On-line Trading of Exploration and Exploitation 2

ICML Exploration & Exploitation Challenge: Keep it simple!

Olivier Nicol OLIVIER.NICOL@INRIA.FR

Jérémie Mary JEREMIE.MARY@INRIA.FR

Philippe Preux PHILIPPE.PREUX@INRIA.FR

LIFL (UMR CNRS 8022) & INRIA Lille Nord Europe
Université de Lille
59650 Villeneuve d'Ascq
France

Editor: Dorota Głowacka, Louis Dorard and John Shawe-Taylor

Abstract

Recommendation has become a key feature in the economy of a lot of companies (online shopping, search engines...). There is a lot of work going on regarding recommender systems and there is still a lot to do to improve them. Indeed nowadays in many companies most of the job is done by hand. Moreover even when a supposedly smart recommender system is designed, it is hard to evaluate it without using real audience which obviously involves economic issues. The ICML Exploration & Exploitation challenge is an attempt to make people propose efficient recommendation techniques and particularly focuses on limited computational resources. The challenge also proposes a framework to address the problem of evaluating a recommendation algorithm with real data. We took part in this challenge and achieved the best performances; this paper aims at reporting on this achievement; we also discuss the evaluation process and propose a better one for future challenges of the same kind.

Keywords: Recommendation - Bayesian - Exploration - Exploitation - Evaluation - Click prediction

1. Introduction

The ICML Exploration & Exploitation challenge considers the problem of predicting clicks of visitors on items presented on a website, based on website logs. The challenge relies on data provided by Adobe. The dataset represents a website activity regarding visitors' appeal towards a set of options. The options correspond to a set of news, one being displayed in a small box on the visited web page. The aim is to propose interesting news to the visitor; their interest for a given news is asserted by a click on it, which provides the visitor further information on the subject.

Let us list a few characteristics of this challenge:

- This is an online process, that is, the data in the dataset are ordered by time.

- Participants do not have access to the data set: the evaluation of proposed algorithm is performed by the organisers. This lack of availability of data also limits the optimisation of the hyper parameters.
- Name and purpose of the attributes are unknown; so it is impossible to rely on some expert knowledge. We only know that all the attributes are somehow characterising the visitor except one: the id of the option.
- We do not have access to the domain of definition of the discrete attributes.
- We do not know the distribution of the value of the continuous attributes. Some of the values are very small, while some others are very large.
- There are missing values in the data. Some of the attributes are never available.
- We have no way to know if we see a visitor for the first time or not (no id accessible during the evaluation process).

In a nutshell, our approach is based on an additive model updated at after each data handling in a Bayesian way. After an overview of the algorithm, we present how we progressively improved our performances. This improvement was mostly obtained thanks to our continuous work on the feature construction issue, and, more surprisingly, by a continuous simplification of the model. Indeed, we ended up making our predictions ignoring completely the displayed option. More importantly for an "Exploration & Exploitation" challenge, all our attempts to take into account any underlying dynamics, or to explore carefully also led to a decrease of the performance. We then exhibit a few reasons why we think simplicity was the best option in this challenge, but that it might not be the case in real recommendation systems. Actually, we think that the bias towards simple algorithms comes from, at least partially, the evaluation process of the challenge. Finally, we conclude by proposing a better procedure to evaluate recommendation systems on that kind of data set.

2. Formalisation of the task and organisation of the challenge

We consider the following problem. A display d is a point in a space $C \times D \times O$ characterising an option presented to a visitor where:

- C is a compact subspace of $\mathbb{R}^{99}$.
- D is a space of dimension 20 of discrete values.
- a point $u \in C \times D$ characterises a visitor.
- O is the option space: $O = \{1, \ldots, K\}$, $K = 6$ in the challenge.

We have a data set $\mathcal{D}$ made of $N = 18 \cdot 10^6$ records ordered by time. To each display d_i is associated a reward $r_i \in \{0, 1\}$, meaning whether the option was clicked or not. As already mentioned, there are lots of missing values for the discrete and continuous attributes, but the option and the reward are known for all data.

A batch b is a group of 6 consecutive tuples (display, reward). During the evaluation process of a program $\mathcal{P}$, the batches are given one at a time in chronological order. At iteration t, $\mathcal{P}$ receives b_t without the associated rewards. $\mathcal{P}$ chooses the element of b_t which it considers the most likely to be associated with a reward of 1. The reward of this element (and only this one) is then revealed to $\mathcal{P}$ and added to the current score. The goal is obviously to score as much as possible.

Computational resources are limited: approximately 1GB of memory for the process of the whole data set, and 100 ms on a 2GHz processor to process each batch.

For debugging purposes, we were initially given 60 lines of the data set. Those 60 data should not be assumed to be representative of the whole data set in terms of attribute values, and their distribution. In particular, the number of successful displays is grossly over-represented in these 60 lines with regards to the whole data set. Thus, it is clear that one should neither train his/her algorithm with these 60 data, nor even learn precise information about attribute values with these 60 data.

In the first part of the challenge, each participant was evaluated on the same set of data, made of the first $3 \cdot 10^6$ lines (so on $5 \cdot 10^5$ batches) of the data set. This set of data is unknown: the evaluation is performed on a server managed by the organisers of the challenge. Each participant is allowed to repeatedly submit his/her program to get his/her score. There are computational limits so that the frequency at which each participant can submit a code is not very high; it depends on the server load, that is, the number of participants having submitted their code; in our experience, the time to perform one evaluation of one program ranged from less than 1 hour to more than a whole day. As our algorithm, and probably most of submitted algorithms, is stochastic, the score obtained is varying at each submission: chance plays a role.

At the end of the first round of the challenge, the $3 \cdot 10^6$ data were revealed to let the participants improve their algorithm. The resulting program was then run only once on the whole data set made of N data, just before the workshop was held. Our algorithm was designed using only the data revealed after the first part of the challenge. After the second part, as the whole data set was revealed, we made use of it to better understand our results.

3. Our approach

We investigated the use of ideas based on adPredictor to predict click rates in online advertisement by Graepel et al. (2010). This solution is inspired from the TrueSkill$^{\text{TM}}$ algorithm by Herbrich et al. (2007) used to rank online players on the Xbox gaming network. The main idea is to consider the probability of click of a visitor u on an option o as the sum of some contributors having different levels of uncertainty and which are updated in a Bayesian way.

3.1. The model of the clicks

The model for the click probability of a visitor on an option is made up of k discrete features built on $C \times D \times O$. The construction of these features will be discussed in section 4. Before that, let us precise that for each possible value of each feature, we estimate a Gaussian distribution $\mathcal{N}(m, \sigma)$. Subsequently, values from this distribution are drawn on demand, acting as weights; each value of a given feature has a weight which is drawn from a Gaussian distribution each time this is required; we name such weights "Gaussian weights", and our algorithm estimates their parameters (mean, and variance). For a given display d, each feature of $C \times D \times O$ takes a value (which may be NA). We say that these values are *active* for d. The Gaussian weight associated to this value is called a *contributor* and reflects its contribution to the click probability of d. This contribution is more or less uncertain depending on σ. We note $active(d)$ the *active contributors* of a display d. Given a *contributor* c, m_c (resp. σ_c) is the mean (resp. the standard deviation) of the associated Gaussian weight.

3.2. Making the decision on a batch

When a batch b is received, a score $s(d)$ is computed for each display $d \in b$ as follows:

$$s(d) = \sum_{a \in active(d)} X(m_a, \alpha \cdot \sigma_a) \tag{1}$$

where $X(m, \sigma)$ is a realisation of a $\mathcal{N}(m, \sigma)$ and α a real parameter. Then the display with the maximum score is chosen.

3.3. Online learning

After an option has been chosen for a display d, the algorithm is notified whether d led to a click or not, by way of the binary reward. The active contributors of d are updated in a positive way if a click occurred, and in a negative way otherwise. This comes close to the update of the weights of a perceptron, but in the present case, the weights are not scalar but Gaussian, and this necessitates smarter updates. We use the following notations:

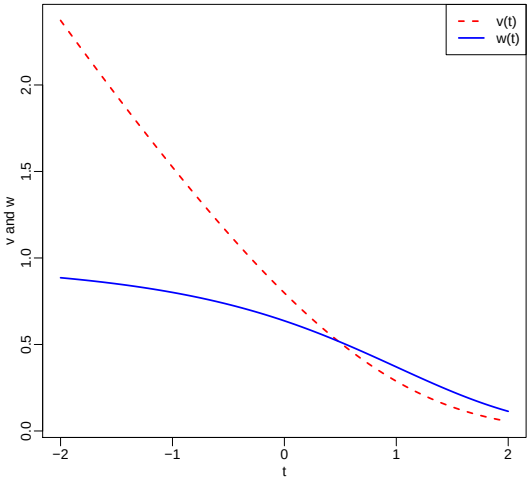
- $y = 1$ if there is a click, -1 otherwise.
- $\Sigma^2 = \beta^2 + \sum_{a \in active(d)} \sigma_a^2$, with β a real parameter
- $\Lambda = \frac{y \cdot \sum_{a \in active(d)} m_a}{\Sigma}$

Σ quantifies the uncertainty on the score and Λ its correctness. The numerator of the fraction defining Λ may be understood as follows: if a score is positive we expect a click and if it is negative we do not; so this numerator is positive if the prediction was correct, and the larger this product, the more correct the prediction. Then, this product is divided by the uncertainty, which means that the more uncertain the prediction, the less it can be considered as really meaningful since we did know that it was uncertain. The update rules for the parameters of the active contributors are as follows:

$$\tilde{m}_a = m_a + y \cdot \frac{\sigma_a}{\Sigma} \cdot v(\Lambda)$$

$$\tilde{\sigma}_a^2 = \sigma_a^2 \cdot \left(1 - \frac{\sigma_a^2}{\Sigma^2} \cdot w(\Lambda)\right),$$

where v and w are real valued functions. Many different functions can be used for v and w. With respect to our tests, as long as the shape is similar, the results are quite robust to variations on v and w. So we decided to use the same functions as Graepel et al. (2010) (see Fig. 1).



$$v(t) = \frac{d\Phi(t)}{dt} \cdot \frac{1}{\Phi(t)}$$

$$w(t) = -\frac{dv(t)}{dt}$$

with $\Phi(t)$ the cumulative distribution function of $\mathcal{N}(0,1)$

Figure 1: The functions v and w.

The main idea of these update equations is that the update is small if the outcome is not surprising (*i.e.*, the prediction was correct). This can easily be seen from the shape of v and w (see Fig. 1). Indeed when Λ (the correctness) is negative, $v(\Lambda)$ is large and so is the correction to m whereas if Λ is positive, then almost no update is performed. The same thing is true for the multiplicative update of σ as when Λ grows, $w(\Lambda)$ gets closer to 1. To see why this is important, let us consider a bad contributor c active along with a lot of good contributors in a display. If we observe a click which is not surprising given the number of good contributors, should we change a lot our estimation of c? Obviously the answer is no since it is very likely that the click has resulted from the contributions of the good ones and not from a bad estimation of c.

This update strategy offers some similarities with TD-learning by Sutton (1988): the more we are surprised by the consequences of a decision, the larger the correction. This strategy usually leads to faster convergence rates of learning.

Another interesting idea is the use of the uncertainty. It makes the learning of the model much more robust than the learning strategy of a simple perceptron for example. Indeed, let us consider the case in which we are sure that a display is bad (low uncertainty on the contributors), and in which unexpectedly, we get a click. This is very surprising, but since

the update also depends on the uncertainty, we will not make a big update because of an event that is most certainly noise.

Note that the update equations were computed through the approximate message passing algorithm. The reader who would like to have a deeper understanding of these equations and of the functions v and w in addition to the intuition we gave here should refer to Herbrich et al. (2007) and Graepel et al. (2010).

4. Our performances along the course of the challenge

The goal of the challenge is to score as much as possible on the 5.10^5 first batches (3.10^6 displays). The final score is the only output we get when an algorithm is submitted; so it is the only criterion we will use to compare different approaches in this section. This section describes how we improved our program along the course of the challenge. For a pseudo-code implementation of our final algorithm, see Appendix B.

4.1. Early results

A purely uniform random strategy scores slightly lower than 1200 on 5.10^5 batches; a strategy that always chooses the same option scores equally. If we model the number of clicks with a Poisson law of parameter $\lambda = 1200$, a significant difference (with risk 5%) is 57 points. This order of magnitude should be kept in mind when trying to compare two different scores.

The algorithm presented in section 3 needs discrete features; so during the first trials, we just ignored the continuous ones. Moreover as the main goal of the challenge was to do recommendation, it is quite natural to try to catch the preferences of the visitors in the features. So we simply built 20 features, each one of them being the Cartesian product of a discrete feature $f_d \in D$ and the option feature. We handled the missing values by just assigning them to a specific contributor.

Note that running the algorithm on $D \times O$ is equivalent as having 6 models, one for each option, using only D. Indeed, two contributors of two different features associated with different options can never be active together. This is actually the approach we took in the beginning but we changed a bit later to be able to handle both features resulting from a Cartesian product with O and features built otherwise without having to duplicate them.

The first attempts scored equally as random. Our first improvement occurred when we set the value of β to 100 instead of 1 (as it was arbitrary set when first executed). Our score was approximately 1500 (25% more than random). In that setting, our score went up to 1610 (34% more than random) after optimising β and the value of the priors on the contributors. It was a good start as we performed significantly better than the random strategy using only 21 out of the 120 features of (C, D, O) (in fact we were using (D, O)).

4.2. Discretisation

4.2.1. Cartesian products

¿From this point, the most natural idea to get better performances out of this model was to include the continuous features. Since we need discrete ones, we began with a discretisation of the continuous values in 5 buckets. Cutting values were fixed using the first seen values using an EM algorithm (Mc Lachlan and Krishnan, 1996) to cluster the 64 first values in 5 Gaussian weights. As with the discrete features, the *null* value (for missing values) has its own bucket. Each bucket of each feature was then associated to a different contributor.

We note C_d the discretised version of C. Using this, we tried to run the algorithm with different versions of the model:

- Still with the idea of trying to find the favourite option of each visitor, we first tried to do a Cartesian product of each feature with the option. So with $C_d \times O, D \times O$, our algorithm scored around 1550. It was very disappointing since it is worse than with only $D \times O$.
- As adding $C_d \times O$ made the performances decrease, we tried to add the continuous features without any Cartesian product. With $C_d, D \times O$, our algorithm scored around 1900 (58% more than random).
- As removing the Cartesian product with O for C_d drastically improved the performances, we also tried C_d, D but our algorithm only scored around 1600.

We tried to combine (with a Cartesian product) only a few features from C_d with the option but no matter which ones we picked, we found no contribution scoring more than 1900. We also tried to identify online which features from C_d were actually correlated with the option using ANOVA and then to dynamically combine them with the option. It did not work either. According to these results, it seems clear that the features from D are characterising the visitor preferences whereas the features from C are characterising the general behaviour of the visitor facing an option.

4.2.2. A better discretisation

We were trying different approaches in parallel to the one we present in this paper and we had noticed that using only one cutting value per feature was pretty efficient (so two buckets per feature). Nevertheless trying to find the good cutting value using only the 60 given lines was pretty hopeless. That is why we decided to build several 2-buckets discretisations for each continuous feature and then choose the best one.

More formally, for each feature $C_i \in C$ we have a set S_i of possible cutting values each of which corresponds to a 2-buckets discretisation. The possible cutting values are chosen before running the algorithm looking at the 60 lines of the data set we have been given ($\forall C_i \in C \quad |S_i| \approx 20$ in order to cover C_i quite exhaustively). For each cutting value v_{ij} of S_i we have 2 Gaussian weights $w_{ij}^{(1)}$ and $w_{ij}^{(2)}$ (one for values lower than v_{ij} and one for greater values).

In principle, for a given display, each feature has one *active* Gaussian weight (see section 3.1). To compute the score of this display, we sum these *active* Gaussian weights (see section 3.2). Here for each feature $C_i \in C$ we have several active weights, one per discretisation/cutting value. So for each $C_i \in C$ we only consider as *active* the one weight associated to the cutting value which maximises the following criterion:

$$booleanToInt\left(\sigma_{ij}^{(1)} \leq T(t) \quad AND \quad \sigma_{ij}^{(2)} \leq T(t)\right) \times d(w_{ij}^{(1)}, w_{ij}^{(2)})$$

where:

- $\sigma_{ij}^{(k)}$ is the standard deviation of the weight $w_{ij}^{(k)}$.
- $booleanToInt(x)$ is equal to 1 if x is true and 0 otherwise.
- $d(w_1, w_2)$ is the probability that the Gaussian weight with the greatest mean between w_1 and w_2 is actually greater than the other. It is a metric of how far apart the two weights are.
- $T(t)$ is a threshold function depending on time. The one we used is just a linearly decreasing function of time $T(t) = \gamma \cdot t$ with a minimum value (time is actually the batch index).

During the update, we update the *active* weights of all the discretisations as in section 3.3.

To sum up, for each feature $C_i \in C$ instead of building only one complex discretisation we build $|S_i|$ discretisations with 2 discrete values (so with only on cutting value). We then choose the cutting value with the two most separated weights if they are both accurately enough estimated. This approach slightly outperformed the EM based discretisation mentioned above (see Sec. 4.2.1). Indeed, it scored around 1940 (62% more than random).

Note that while running, the algorithm could add new values to some S_i if it observed too much values outside of the minimum and maximum observed in the 60 given lines. However when we were able to run the algorithm by ourselves after the end of phase 1, we noticed that it was almost never the case, hence the good results we observed building discretisations only observing these data.

4.2.3. A simpler discretisation

To check whether we were learning interesting things with our two previous approaches, we tried to build an offline fixed discretisation. To do so, we looked at the density plot of the continuous features for the 60 lines of the data set given during phase 1 (see Fig. 2). We did not use the frequency of clicks as there were too few of them. To build the discretisation, we just assigned one interval per peak on the plot. We identified 12 features as different from the others and from each other like the two ones in the left part of Fig. 2. Their discretisations were built one by one. For the 87 others, we made three groups:

- A group where all the values in the 60 lines were missing. We just kept the approach that tries to find only one cutting value. We found out during phase 2 that these features were actually *null* in all data.
- A group where all the features looked almost exactly like the one on the right of Fig. 2. They represent one third of all the continuous features and we only built a unique discretisation for all of them.
- A group where the value of the features is 0 most of the time but not always. They represent half of all the continuous features and as the previous groups we only built a unique discretisation for all of them.

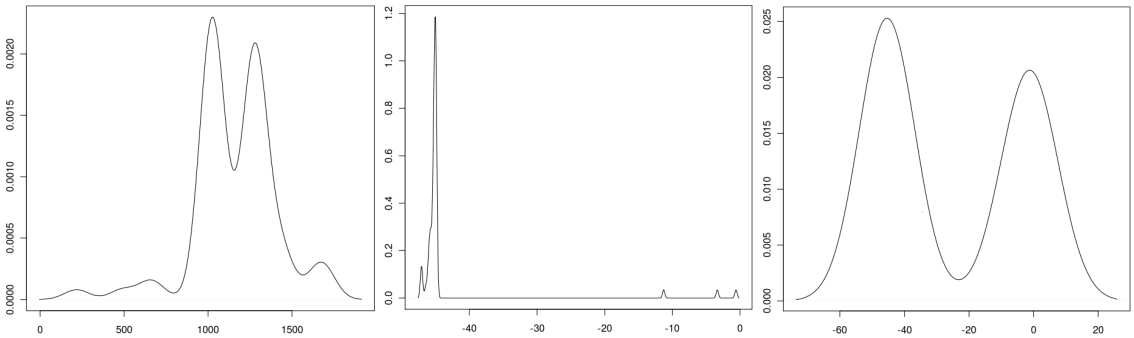


Figure 2: The density plot of 3 continuous features for the 60 lines of the data set given during phase 1. The two on the right are on a logarithmic scale.

With this simple method of discretisation, our algorithm scored 1950 (63% more than random) which was already more than the previous approach. We then focused on the two largest groups of features and after merging and splitting a few intervals, we were able to bring our score up to 2000 (67% more than random) and sometimes a bit more because of the variance of the algorithm. This has remained our best score for a while.

4.3. Dynamics

The batches were served in chronological order during the challenge to let us identify the dynamics of the system. When we observe the update equation for σ we notice that whatever happens, σ always decreases making the model unable to handle the fact that some weights may vary over time. None of the approaches we tried allowed us to improve our performances. We try to explain why in section 5. However we present here two of these approaches.

4.3.1. Microsoft's idea

In Graepel et al. (2010), the following update rule is proposed to be used at each time step and for each weight of each feature:

$$\tilde{\sigma}_i^2 = \frac{\sigma_{i(p)}^2 \cdot \sigma_i^2}{(1-\epsilon) \cdot \sigma_{i(p)}^2 + \epsilon \cdot \sigma_i^2}$$

$$\tilde{m}_i = \sigma_i^2 \cdot \left((1-\epsilon)\frac{m_i}{\sigma_i^2} + \epsilon \cdot \frac{m_{i(p)}}{\sigma_{i(p)}^2} \right)$$

The idea behind these update equations is to let the weights evolve back to their prior in order to slowly forget the influence of past data. This allows the model to adapt to the dynamics. In the equations, the bigger ϵ, the faster the algorithm forgets. Unfortunately the smaller ϵ, the better the algorithm performed in the challenge. The better performances were achieved with $\epsilon = 0$.

4.3.2. Two or more models

The previous approach is not fully satisfying as it cannot take into account the fact that some weights may evolve at different speeds. Some other may also not evolve at all. Moreover, the convergence of the weights towards their prior makes the model learn slower as we are always forgetting a little.

To deal with these issues, we propose to use m models. At each time step only the oldest model is used to make a prediction and all of them are updated. Every τ time steps, we destroy the oldest model and replace it by a new one. This new model can be initialised in several ways. The simplest one would be to always give to each weight the same prior. Then we would handle the dynamics by making prediction with a model which has been learning during between $\tau \cdot (m-1)$ and $\tau \cdot m$ time steps. We could also try to build the priors using the history of their values. Are these values very stable? unstable? evolving following some kind of trend? We tried such estimators but as nothing worked, we will not go into the details. However we will see in section 5 that we have good reasons to think that this failure was due to the challenge itself. We will study this idea more carefully in some future work.

4.4. Exploration

Throughout the challenge we had been using the update equation (1) with $\alpha = 1$. Actually this parameter did not even exist. As none of our approaches seemed to do better than 2000 we were starting to explore different paths. Just to see what happened we submitted an algorithm which was not taking into account the standard deviations in the prediction phase (it was actually summing up the means of the weights). Surprisingly this approach scored 2130 (130 points more than with exploration and 78% more than random). Then, we tried to optimize the value of α but the best value was 0 (no exploration at all). A value up to 0.05 was not making any difference though.

We obviously tried decreasing exploration approach as $\epsilon_n - greedy$ but it did not improve the performances. The only one that seemed to slightly improve the algorithm (in terms of performance and variance) is the following: for the first 5000 batches (1% of the total) instead of choosing the display with the best score, we choose the display d maximising:

$$v(d) = \sum_{a \in active(d)} \sigma_a^2$$

which is the variance of the sum of the active Gaussian weights. During phase 2, by running the two approaches 100 times, we were able to check whether the performances were really improved or not. Here are the results: using $s(d)$ for the 5000 first batches: $mean = 2130$ $variance = 1086$, while using $v(d)$ for the 5000 first batches: $mean = 2140$ $variance = 587$. That is how our final score 2170 for phase 1 was achieved.

4.5. Further score improvements

For phase 2 we got access to the data of phase 1. Then we could run a lot more simulations to optimise the algorithm. Trying to optimise α, β and the prior on the weights led to minor improvements. However to be able to present the results, we did again a few experiments. In one of them, we tried to use C_d, D as a model which is the raw feature space with our discretisation over C and without the option. We ran the algorithm 100 times with these features and it achieved a mean score of 2215 and a variance of 190. The best score we got during this experiment is 2240 (85% more than random) which is much better than our previous approach. This is the algorithm that won the second phase of the challenge. In summary, the simplest version of the model ended up having the better performances and being the more stable.

We had tried this approach before and it had scored badly on the server of the challenge. We can only try to guess why: a bug on the server's side, a mistake in the file submission on our side... We could have avoided that kind of thing by submitting each algorithm more than once but we would have lost a lot of time as one submission usually returned after several hours. This is an interesting idea to think about in case of a future challenge. The simplest thing would be to run the algorithm 5 or 10 times at each submissions. However if computational resources are limited, it becomes an issue. To address this problem, more data could be given to the challengers so that they can test their algorithm by themselves before submitting it. Restrictions on the number of submissions per day could even be imposed in that case to easily allow a sharing of resources between challengers.

5. Understanding the results

The purpose of the challenge was to design an algorithm capable of doing three main things:

- identify the preferences of a visitor to recommend something to him/her
- balance exploitation and exploration
- adapt to the dynamics of system (some options may for example become less popular for some visitors)

However, in the challenge we had to choose between 6 couples $(visitor, option)$ and what we did is basically visitor selection. Indeed our best approach only used C and D — the visitor features — to make its decisions. Moreover as presented in the last section the approach which performs the best neither explores, nor adapts.

5.1. Why did visitor selection work so well?

What we did is identifying the general behaviour of visitors in front of an option without paying attention to evolutions or exploring anything. Then when a batch comes we select the visitor who clicks the most in general regardless to the displayed option. We can try to explain why that works so well in two ways.

5.1.1. Intuition

Trying to find the click probability of a visitor in general is a lot easier than trying to find it for each option. Indeed each time a visitor is seen, the model can be updated whereas to find the preferences, no information is obtained for non displayed options. When working on the logs of a big french company we had also noticed that some variables are very important as far as clicks are concerned. For instance some pages have higher click probabilities than others and the time of the day matters (visitors do not click at night). C, D might have contained that kind of features making our task even easier. Moreover half of the visitors in the company's logs never click. Identifying all of them in the challenge already doubles the click probability (which is basically the maximum score we achieved).

Intuitively the general behaviour of someone on the Internet is very unlikely to change dramatically as opposed to his preferences. That is why trying to find some kind of dynamics was hopeless. So having to identify something pretty easy to characterise and very stable, the reason why almost no exploration was needed is rather clear. To picture a bit more that the task was not that hard see Fig. 3 and 4 where we see that we very quickly improve the performances and stabilise them. Note that the low click rate around batch $170,000$ must be due to the data as it is also there for the random strategy.

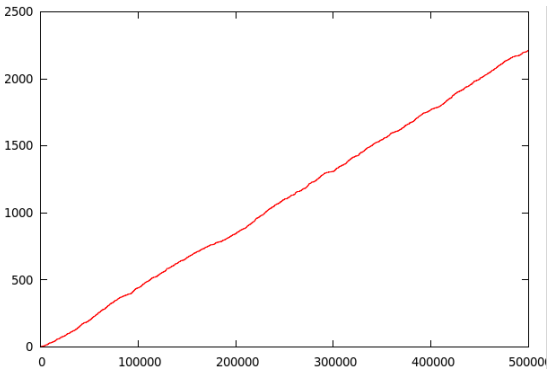


Figure 3: Evolution of our score during phase 1. The x-axis is the number of the current batch.

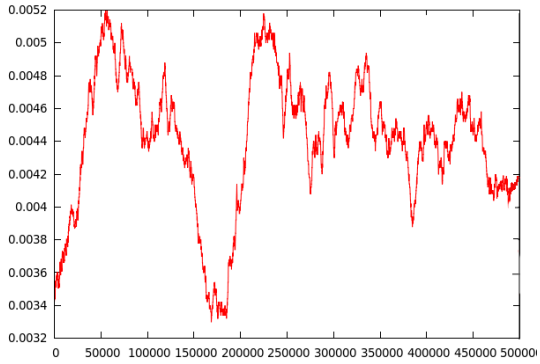


Figure 4: Evolution of the click rate during phase 1.

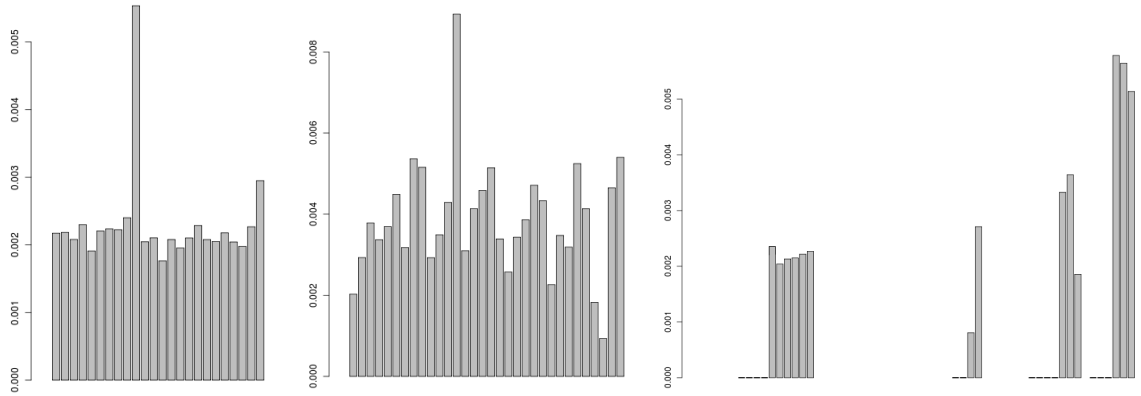


Figure 5: Click frequency against the value of 3 different features. The two features on the left are discrete and the values appearing less than 5000 times in the data set have been removed. The one on the right is continuous and plotted on a log scale. For that latter feature, please note that the high frequencies on the right only represents 0.5% of the values.

5.1.2. Features

In a more pragmatic way we can explain why we performed so well without using the options (or any other Cartesian product) by looking at the click frequency plot of some features. Looking at Fig. 5, we can clearly identify patterns both for the discrete and the continuous features. The one in the middle of Fig. 5 is particularly informative. We tried to run a very simple UCB strategy as described by Auer et al. (2002) with one arm per value of this feature and scored 1450. It means that using only one feature from D allows us to perform 21% better than a purely uniform random strategy!

5.2. Crossed effects

Does this mean that there are not any crossed effects between features in this data set? The answer is no. We performed an analysis of variance (ANOVA) and found a lot of crossed effects between O and some other features. We also found crossed effects between features of D, C. During phase 1 we had tried to learn them online and during phase 2, we knew about the Cartesian products. However we still have not been able to exploit them. We can only try to give an intuition to explain that. The general behaviour of visitors seems to be something very stable and very well characterised by the set of features we have been given. Trying to enhance the model with things like preferences which are much more unstable and much harder to identify is very tricky. In our experiments we have even noticed that it adds disturbance to the model making it less efficient in learning and using the behaviour of visitors.

6. More experiments

6.1. Click rate prediction

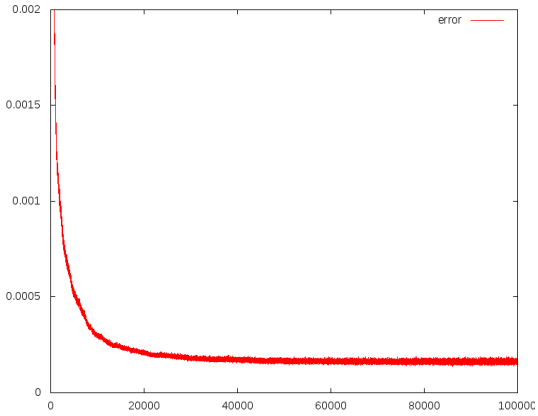


Figure 6: Mean squared error on the probability over time (computed on 100 randomly generated displays).

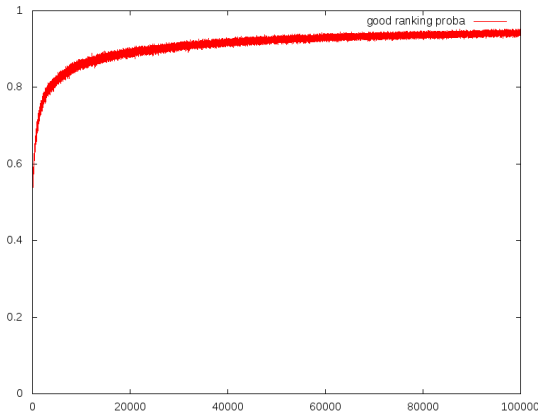


Figure 7: Probability that two predictions are correctly ranked over time (computed on 100 couples of randomly generated displays).

In Graepel et al. (2010) the following way to infer click probability from the model is proposed:

$$p(d) = \Phi(\frac{s(d)}{\sqrt{\beta^2 + \sigma^2}})$$

with the same notations as in section 3.3.

We experimented this equation on a toy example. In this example, we assume that reality is as assumed by the model: each click probability is the result of a sum of contributors. In the following experiment we use 10 features which can take 5 possible values. For each feature f_i, the values of the 5 contributors are as follows:

$$\{10^{-4}.p_i^0,\ 10^{-4}.p_i^1,\ 10^{-4}.p_i^2,\ 10^{-4}.p_i^3,\ 10^{-4}.p_i^4\}$$

The real parameters p_i are uniformly drawn in $[1, 5]$ at the beginning of the experiment.

At each time step a random display is presented with its reward to update the model (the values of the features are drawn uniformly). The squared error (Fig. 6) on the predictions converges but its value remains very high when it stabilises. Indeed the squared error stabilises around 0.002. The average over the click probabilities is around 0.04 and $\sqrt{0.0002} \approx 0.014$ (35% of 0.04).

However as it can be seen on Fig. 7 the order of the probabilities is very well learnt and it is learnt very fast. The model reaches a rate of 80% of well ranked probabilities after less than $1,000$ steps and then goes up to 95%. Hence the success of the algorithm when it comes to choose between displays during the challenge. Also note that when the error on the predictions stops improving after $30,000$ time steps, the probability of good ranking keeps improving until the end of the experiment.

6.2. Influence of the parameters

The model has a few parameters and we propose here to study their influence on the performance of our algorithm. We will not talk about α since the best option in the challenge was to do no exploration at all ($\alpha = 0$).

6.2.1. Beta

β impacts the learning speed. Fig. 8 shows that the algorithm is not very sensitive to its value. Any value between 300 and 600 achieves almost the best performances. Then when β grows we do not learn fast enough and the performances decrease until stabilising around 1630. If β gets closer to 0 we try to learn too fast and performances dramatically decrease until 1200 (random strategy).

6.2.2. Priors

Another important parameter is the prior we take for the Gaussian weights. As we had no prior knowledge about the data, we had to give default values. As far as the mean is concerned, if it is between -1 and 1 it does not change anything. For larger values, we eventually learn correctly but it takes more time, hence a decrease in the performances.

Optimising the standard deviation (σ) was more interesting. As we can see on Fig. 9 we can find an optimal value to give to each weight which is around 20. However Fig. 10 and 11 (the latter being a zoom on the best scores of the former) show that we can do better. If we initialise with different values the standard deviation of the discrete features (σ_d) and the standard deviation of the continuous features (σ_c), we can win something like 30 points. It seems to mean that learning from D is easier than learning from C. We tried to split C and give different values of σ to each resulting group but nothing significant came out. Note that on Fig. 10 even though we can find optimal values for σ_d and σ_c the algorithm remains very stable. Only very low values for σ (less than 2) or very high values (more than 80) leads to scores inferior to 2000.

7. Evaluation protocol

As already mentioned, the evaluation protocol used during the challenge is problematic because there exists a much stronger link between the visitor and the click probability, than between the option $\times$ visitor and the click rate. So if we are given a batch with the possibility to choose a couple (visitor, option), we should focus only on the visitor.

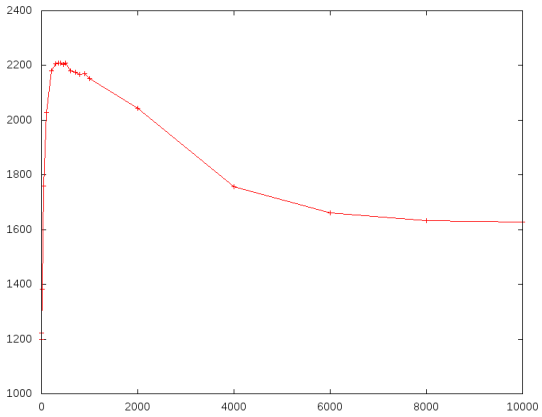


Figure 8: Score in the challenge against β.

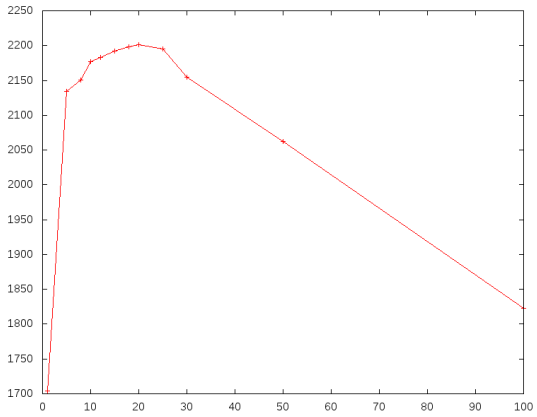


Figure 9: Score in the challenge against σ.

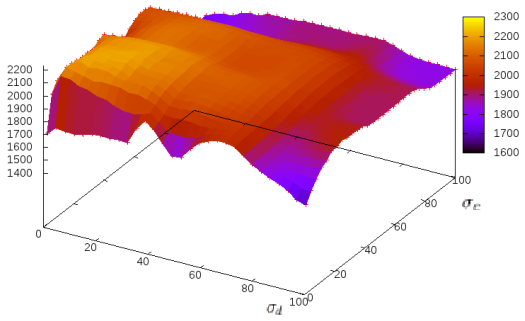


Figure 10: Score in the challenge against σ_d and σ_c.

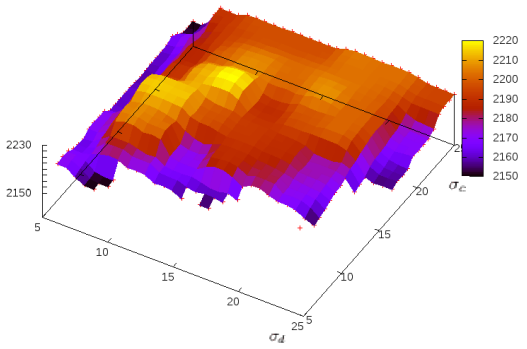


Figure 11: Zoom on the best scores in the challenge against σ_d and σ_c.

Of course it is impossible in this situation to know what would have been the performance of an algorithm because we can not go backward in time or find an exactly identical situation. One good solution to compare two algorithms would be to use them online with real visitors at the same time but it would be very expensive (in terms of missed clicks) and would necessitate a big visitor flow to conduct experiments.

An other possibility would have been to use rejection sampling on the option. The process would be:

- Pick the first non used row (u, o, r) from $\mathcal{D}$ (u is the visitor, o the displayed option, r the click or no click information).
- Input u to the recommendation algorithm and let it choose an option o_a,
- If o_a and o are identical, then give the reward r to the algorithm, otherwise discard this visitor.
- loop until no more rows in $\mathcal{D}$.
- The score of the algorithm is the average click rate for non discarded visitors.

This method based on rejection would only use $1/K$ records to evaluate the algorithm, and would act as a "time accelerator"; it is assumed that(options are uniformly distributed in $\mathcal{D}$. Similar ideas have been used by Li et al. (2010). Here it is acceptable because there is only 6 possible options and a large number of records (so dropping 5/6 of them is not a big deal). But sometimes we can not afford such a method.

7.1. A new method for offline evaluation

In this section, we propose a new way to evaluate recommendation algorithm on a task such as the one considered in this challenge. In the next section, we show that this approach is sound and actually evaluates what it is meant for.

With the data set $\mathcal{D}$, we can create another data set $\mathcal{D}_1$ by only taking the lines of $\mathcal{D}$ associated with a reward of 1. Then we can design an online classification problem in which the classes are the options. So the goal is to map each visitor of $\mathcal{D}_1$ to the right option. The only way to perform well in that case is to find connections between visitor preferences and the options: simple visitor selection is no longer relevant. By trying to solve this problem online, we might also be able to identify some kind of dynamics in the preferences or to benefit from exploration. We call that problem P_1.

In problem P_1, a given visitor may have more than one class if he/she has been shown different options and that he/she has clicked on at least two of them. In fact one can consider that each visitor u belongs to K classes. Each time we encounter visitor u during the online process and try to classify him/her as class (or option) o the probability to be correct is given by:

$$p_{ou} = p(o|u \in \mathcal{D}_1)$$

Note that $p_{ou} = 0$ if u has never clicked on o in the data set $\mathcal{D}$ and that:

$$\forall u \quad \sum_{o \in Options} p_{ou} = 1$$

We call that kind of classification problem a *stochastic* classification problem.

We can also design a harder stochastic classification problem where we have to map each visitor of $\mathcal{D}$ to the right option. When we consider a visitor which is in $\mathcal{D}_1$, the reward is given as in P_1. Mapping an option to a visitor which is not in $\mathcal{D}_1$ always lead to a reward of 0. So in this problem that we call P_2 we have visitors with no known class. One can think of them as noise added to problem P_1. In the evaluation process, no difference is made between a misclassified visitor, and a visitor without class: in both cases, the reward is 0 and that is all. This is what makes the problem harder.

7.2. Theoretical result

Theorem 1 *With the notation introduced in section 2. We denote:*

- f^* *the optimal function that maps a visitor to an option for the recommendation problem.*
- g^* *the optimal function that maps a visitor to an option for* P_1
- h^* *the optimal function that maps a visitor to an option for* P_2

Under the assumption that the options were allocated uniformly while creating $\mathcal{D}$*, we have the following result:*

$$\forall u \in \mathcal{D}_1 \quad f^*(u) = g^*(u) = h^*(u)$$

This means that if we manage to design a good mapping function for P_1 or P_2 (two problems whose proposed solutions are easy to evaluate), we have theoretical guarantees that it will also perform well on the real recommendation problem.

The proof in provided in Appendix A. The uniformity assumption is true in the case of the Adobe data set. It should be possible to extend this result (up to a renormalisation) to the case were the probability of distribution is not uniform but known and with a non null probability of having any option attributed to visitor u.

8. Conclusion and future work

We presented a Bayesian approach to the recommendation problem. This approach led us to win the 2011 ICML Exploration & Exploitation challenge. Based on a data set provided by Adobe, we have shown that on this particular dataset, the click probability is much more related to the visitor than to the option. Moreover this behaviour is very stable over time. This means that not all visitors are equal, and that there are much more "valuable" visitors than others. In the advertising context, this means that some part of the optimisation problem of advertising display is to allocate some of this "premium" visitors to some

well chosen ads. To perform this optimisation, a mixture of bandit algorithm and linear programming was proposed by Girgin et al. (2010).

We also proposed to use an other evaluation protocol to use collected data to compare the performance of new online algorithms. Further work will investigate the link between the Thompson sampling step of adPredictor and the global performance. As we outperform the Thompson sampling using likelihood maximisation, and do even better doing something in between, we are interested in the non-asymptotic behaviour of Thompson sampling and will try to exhibit better strategies (especially in the small probability case).

Acknowledgments

This work was supported by Ministry of Higher Education and Research, Nord-Pas-de-Calais Regional Council and FEDER through the "Contrat de Projets Etat Region (CPER) 2007-2013", the ANR project Lampada (ANR-09-EMER-007) and the CRE-INRIA-Orange Labs. During this work, O. Nicol was supported by a PhD grant of the University of Lille, and J. Mary acknowledges a partial secondment from INRIA.

References

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. volume 47, pages 235–256, Hingham, MA, USA, May 2002. Kluwer Academic Publishers. URL http://dx.doi.org/10.1023/A:1013689704352.

S. Girgin, J. Mary, Ph. Preux, and O. Nicol. Advertising campaigns management: Should we be greedy? In *The 10th IEEE International Conference on Data Mining (ICDM-2010)*, pages 821–826, 2010. URL http://www.grappa.univ-lille3.fr/~{}mary/paper/icdm-2010.pdf.

Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-2010)*, pages 13–20, Haifa, Israel, June 2010. Omnipress. URL http://www.icml2010.org/papers/901.pdf.

Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill$^{\text{tm}}$: A bayesian skill rating system. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS-2006)*, pages 569–576. MIT Press, 2007.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, WWW'2010, pages 661–670, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: http://doi.acm.org/10.1145/1772690.1772758. URL http://doi.acm.org/10.1145/1772690.1772758.

Geoffrey J. Mc Lachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley-Interscience, 1st edition, November 1996. ISBN 0471123587. URL http://www.worldcat.org/isbn/0471123587.

Richard S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, pages 9–44. Kluwer Academic Publishers, 1988. URL http://webdocs.cs.ualberta.ca/~sutton/papers/sutton-88-with-erratum.pdf.

Appendix A. Proof of the theorem

Let us remind the notations:

- $\mathcal{D}$ is the whole data set.
- $\mathcal{D}_1$ is the data set containing only the lines of $\mathcal{D}$ associated with a reward of 1.
- f^* is the optimal function that maps a visitor to an option for the recommendation problem.
- P_1 is the stochastic classification problem where we have to map each visitor of $\mathcal{D}_1$ to the correct option. g^* is its optimal mapping function.
- P_2 is the stochastic classification problem where we have to map each visitor of $\mathcal{D}$ to the correct option. Classifying a visitor not present in $\mathcal{D}_1$ only leads to a reward of 0. h^* is its optimal mapping function.

A.1. g^* and h^*

It is straight forward to see that if $u \in \mathcal{D}_1$ we have:

$$h^*(u) = g^*(u)$$

Indeed rewards are earned the same way in P_1 and P_2 and the only lines that matter in P_2 are the ones from $\mathcal{D}_1$. To complete h^* we can map anything to the visitors who are not in $\mathcal{D}_1$ since they do not produce rewards anyway (because all visitors not in $\mathcal{D}_1$ correspond to a non click row, and in P_2 there is no way to score even providing a different option).

A.2. f^* and g^*

An optimal mapping function for the recommendation problem is as follows:

$$f^*(u) = \text{argmax}_{o \in Options} p\left(Click = 1 | \left(u, o\right)\right)$$

For the stochastic classification problem in $\mathcal{D}_1$, an optimal mapping function can be written as follows:

$$g^*(u) = \text{argmax}_{o \in Options} p(o | u \in \mathcal{D}_1)$$

Let us call $N_{(u,o)}$ the number of times option o was displayed to visitor u in $\mathcal{D}$ and $C_{(u,o)}$ the number of times u clicked on o in $\mathcal{D}$. We have the two following equalities:

$$p(o|u \in \mathcal{D}_1) = \frac{\mathbb{E}_{(u,o)\in\mathcal{D}}[C_{(u,o)}]}{\sum_{o' \in Options} \mathbb{E}_{(u,o')\in\mathcal{D}}[C_{(u,o')}]} \tag{2}$$

$$\mathbb{E}_{(u,o)\in\mathcal{D}}[C_{(u,o)}] = \mathbb{E}_{(u,o)\in\mathcal{D}}[N_{(u,o)}].p\left(Click = 1|(u, o)\right) \tag{3}$$

If the options were allocated uniformly while creating $\mathcal{D}$ - which is the case in the data set used for the challenge according to the information we have been given by Adobe - then

$$\forall u \quad \mathbb{E}_{(u,o_1)\in\mathcal{D}}[N_{(u,o_1)}] = \mathbb{E}_{(u,o_2)\in\mathcal{D}}[N_{(u,o_2)}] = ... = \mathbb{E}_{(u,o_k)\in\mathcal{D}}[N_{(u,o_k)}], \quad k = |Options| \tag{4}$$

Replacing (2) in (1) and then simplifying using (3) we obtain:

$$\begin{aligned} p(o|u \in \mathcal{D}_1) &= \frac{\mathbb{E}_{(u,o)\in\mathcal{D}}[N_{(u,o)}] \,.\, p(Click = 1|(u,o))}{\sum_{o'\in Options} \mathbb{E}_{(u,o')\in\mathcal{D}}[N_{(u,o')}] \,.\, p\,(Click = 1|(u,o'))} \\ &= \frac{p\,(Click = 1|(u,o))}{\sum_{o'\in Options} p\,(Click = 1|(u,o'))} \end{aligned}$$

Finally, using this result in the expression of g*, we get:

$$\begin{aligned} g^*(u) &= \text{argmax}_{o\in Options} \frac{p\,(Click = 1|(u,o))}{\sum_{o'\in Options} p\,(Click = 1|(u,o'))} \\ &= \text{argmax}_{o\in Options} p\,(Click = 1|(u,o)) \\ &= f^*(u) \end{aligned}$$

So we do have:

$$\forall u \in \mathcal{D}_1 \quad f^*(u) = g^*(u) = h^*(u)$$

□

Appendix B. Pseudo code of the algorithm

B.1. The winning algorithm

Note that before using the algorithm, a set of discrete features is to be built. For each possible value of each feature, we keep track of a Gaussian weight (its mean m and standard deviation σ). For a given display, each feature takes on one value. The weights corresponding to these values are said to be *active* for this display.

Function chooseDisplay(D: SetOfDisplays): Display

parameter: *T: the number of initial pure exploration steps*
parameter: α
foreach *Display* $d_i \in D$ **do**
  $s[i] \leftarrow 0$
  foreach $a \in active(d_i)$ **do**
    if $\#iterations \geq T$ **then**
      draw x from $\mathcal{N}(m_a, \sigma_a \cdot \alpha)$
    else
      $x \leftarrow \sigma_a^2$
    end
    $s[i] \leftarrow s[i] + x$
  end
end
$maxIndex \leftarrow \text{argmax}_i s[i]$
return $d_{maxIndex}$

Procedure updateModel(d: Display, click: Boolean)

parameter: β
if *click* **then**
  $y \leftarrow 1$
else
  $y \leftarrow -1$
end
$uncertaintySq \leftarrow \beta^2 + \sum_{a \in active(d)} \sigma_a^2$
$uncertainty \leftarrow \sqrt{uncertaintySq}$
$correctness \leftarrow \frac{y \cdot \sum_{a \in active(d)} m_a}{uncertainty}$
foreach $a \in active(d)$ **do**
  $m_a \leftarrow m_a + y \cdot \frac{\sigma_a}{uncertainty} \cdot v(correctness)$
  $sq \leftarrow \sigma_a^2 \cdot \left(1 - \frac{\sigma_a^2}{uncertaintySq} \cdot w\left(correctness\right)\right)$
  $\sigma_a \leftarrow \sqrt{sq}$
end

Algorithm 1 Evaluation of the two previous functions during the challenge

foreach $Batch\ b : B$ **do**
  $d \leftarrow chooseDisplay(b)$
  $r \leftarrow observeReward(d)$
  $updateModel(d, r)$
end

B.2. More than one model

This approach is the one presented in section 4.3.2. We note $updateModel_M$ the procedure updating model M and $chooseDisplay_M$ the function choosing a display using model M.

Algorithm 2 using more than one model to handle the dynamics

$modelList \leftarrow emptyList$
$i \leftarrow 0$
add a new model to $modelList$
foreach $Batch\ b : B$ **do**
  $M \leftarrow firstElement(modelList)$
  $d \leftarrow chooseDisplay_M(b)$
  $r \leftarrow observeReward(d)$
  foreach $Model\ M \in modelList$ **do**
    $updateModel_M(d, r)$
  end
  $i \leftarrow i + 1$
  if $i = \tau$ **then**
    if $size(modelList) = m_{max}$ **then**
      remove the last element of $modelList$
    end
    add a new model at the beginning of $modelList$
    $i = 0$
  end
end