Technische Universität Berlin

Bernstein Center for Computational Neuroscience

Master Thesis

Meta Learning in Recommendation Systems

Author:
Jonas Seiler

1. Supervisor:
Prof. Dr. Manfred Opper
2. Supervisor:
Dr. Andreas Ruttor

June 14, 2013

Eidesstattliche Versicherung

Die selbständige und eigenhändige Ausfertigung versichert an Eides statt

.....
(Datum/Date)

.....
(Ort/Place)

Statutory Declaration

I declare in lieu of oath that I have written this thesis myself and have not used any sources or resources other than stated for its preparation

.....
(Unterschrift/Signature)

Most modern recommendation systems use several recommendation algorithms. In order to create useful recommendations, the different recommenders must be combined. We will implement such a combination method in a real world recommendation system, which has to recommend millions of items per day. Due to this large amounts of recommendations, we will rely on simple Bayesian Model Averaging (BMA), i.e. a weighted average. Under mild assumptions, we will derive a simple algorithm to calculate the weights for the BMA (Mean Ensemble Threshold) and compare it to a Soft-Max normalization. The weight calculation will rely on Multi-Armed Bandits (MAB) to address the Exploration-Exploitation trade off problem. We will compare different MABs. On average, the UCB1 MAB performed best, however the best result was reached by estimating an upper confidence bound based on a binomial distribution. The Soft-Max normalization performed best if the hyperparameter are well tuned. MET where able to outperform the benchmark approach as well, when UCB1 as MAB where used. Thus, using BMA in combination with MAB is useful to combine recommendation algorithms. Further, we discuss the use of contextual information or Bayesian Model Combination to improve the performance.

Contents

1. Introduction

Recommendation systems are currently used for various purposes including e-commerce, news articles, advertisements, movies or music. In the past, several recommendation algorithms have been proposed to improve the quality of recommendations. However, there are only little attempts that try to combine several recommendation algorithm to improve the recommendation performance. The idea to combine several learner to create better learning algorithms is not new though. In the literature this is usually known as meta learning or ensemble techniques [9, 11]. The purpose of this thesis is to review, implement and evaluate different methods for meta learning in the field of recommendation systems. The actual algorithms will be tested in a web based online evaluation with real data used in news article recommendation. This means the user will be presented with a selection of different news articles at the end of a news article, which she or he might want to read. Moreover, the algorithms have to be able to handle huge amount of data (thousands request per minute). The main limiting resources of our learning algorithms is not the number of samples but the computational power. Thus, our approach needs to be fast and scalable.

1.1. Problem Definition

There are R many recommenders given. A recommender $\mathrm{f}_r(\vec{x_t}) : \mathbb{R}^d \rightarrow [0,1]^m$ will map some input vector $\vec{x_t} \in \mathbb{R}$ (d number of input features) at time $t \in \{1, 2, ...\}$ to a real valued vector ($m \geq 0$ is the number of items) indicating the believe of recommender r that its recommendation for each item (i.e. news articles) will be accepted. In practice it can be the case that a recommender does not return any recommendation or only for a subset of items. The task is to find a function that will map the tuple $\mathrm{g} : (\mathrm{F}_t, x_t) \rightarrow [0,1]^m$where F is the set of all recommenders (may change over time). Thus, the purpose of $\mathrm{g}(\mathrm{F}_t, x_t)$ is to mix the recommendation of single recommenders to increase its performance (i.e. building an ensemble). In the end, the top n recommendation will be shown to the user. Thus, we are forced to make a decision, even if all recommenders would not recommend any items. For those n items we will receive a reward $r_{j,t} \in \{0,1\}$with $j \in I_t$ and I_t is the set of items recommended at time t. I_t^*will denote the optimal set of items at time t. Thus, the function $\mathrm{g}(\mathrm{F}_i, x_i)$ should be constructed such that

$$\lim_{t \rightarrow \infty} \frac{\sum_{t=1}^{T} \sum_{j \in I_t} r_{j,t}}{\sum_{t=1}^{T} \sum_{j \in I_t^*} r_{j,t}} = 1 \tag{1.1}$$

Moreover, the algorithm has to handle the Exploration vs. Exploitation Trade off (EET). This means, it needs to deal with the stochastic nature of rewards and has to consider an eventual learning period of the recommender itself.

1.2. Literature Review

We will review different meta learning strategies first. It will become clear that there is a need to incorporate methods to handle the EET. Thus, we will also review methods from this field in the section thereafter.

1.2.1. Meta Learning

The most widely used meta learning techniques are boosting, bagging and Bayesian model averaging (also known as Bayes optimal classifier (BOC)) [11].

Boosting and bagging belong to the same class of algorithms. They will try to combine various models, but not necessarily all models, into a new, potentially better model [11, 10]. Contrary to this is model averaging, which, as the name suggest, tries to average over all models. According to Davidson and Fan [10], model averaging is desirable *"when there is significant model uncertainty and averaging out model uncertainty is desirable"*.

Bagging (Bootstrap aggregating) tries to learn many classifier, each will be trained on a bootstrapped subsample of the data and combined via model averaging [6]. Boosting (e.g. Adaboost) also trains multiple classifier iteratively, but weights training samples differently [27, 26]. Thus, classifier trained later will be tweaked to handle misclassified instances of those before. For our purposes, both methods are not suitable in that form, as we do not want to train new classifier every time, that would be to time consuming. Thus, in its essence, we cannot use boosting and bagging.

The idea of BOC is to weight each model by its confidence [10]. The definition is given in (1.2).

$$\begin{aligned} \mathrm{P}(y_i|Y) &= \sum_{j=1}^{R} \mathrm{P}(y_i, M_j|Y) \\ &= \sum_{j=1}^{R} \mathrm{P}(y_i|M_j, Y)\mathrm{P}(M_j|Y) \\ &\overset{\text{cond.ind.}}{=} \sum_{j=1}^{R} \mathrm{P}(y_i|M_j)\mathrm{P}(M_j|Y) \end{aligned} \tag{1.2}$$

where $\mathrm{P}(y_i, |Y)$ is the likelihood observing a new data point y_i given the past observations, $\mathrm{P}(y_i|M_j)$ is the likelihood observing a new instance given the model (M_j) and $\mathrm{P}(M_j|Y)$ is the likelihood of the model. Note the step from the second to the third line, where conditional independence between the new instance and observed data given the model is assumed. Thus, this is only valid if the data generation is fully explained by the

model. Examples of algorithms performing model averaging are Random Forest (RF) and Parametric Bootstrap Model Averaging (PBMA) [10].

In an online system we face the problem that we cannot be sure that our estimates of $P(y_i|M_j)$ and $P(M_j|Y)$ are correct, because we cannot be sure that the learning algorithms have been converged yet or that the estimate of the model prior is correct (enough). Thus, this uncertainty has to be taken into account in the final decision process. In the next section we will review different methods how this problem is usually tackled.

1.2.2. Exploration - Exploitation Trade Off

The exploration versus exploitation trade off refers to a problem an agent faces during a decision process. The agent has to decide to which degree it will choose an action with the highest payoff (exploitation) or receiving more information on the payoff of other actions (exploration). A simple approach would be to choose ϵ percent of the time an action randomly and otherwise follow the model. Because we usually assume that our model performs better over time, we might start with an ϵ of 1 and decrease it over time. The former strategy is known as ϵ-greedy while the later one is called ϵ decreasing greedy [30]. Obviously, they have the draw back that one has to optimize the hyperparameters (ϵ and the decreasing strategy) somehow and that its completely ignores other sources of information that might be relevant (e.g. the variance of the payoff for an action). Estimating the hyperparameters is usually done using a grid search procedure, which is very time consuming. Moreover, we cannot rely on an offline evaluation of the hyperparameters because in an online system the actions do effect the choices of the user. We would have to simultaneously search the grid (not point by point but in parallel) in an A/B testing fashion, to eliminate the possibility that performance difference might be attributed to some other covariates (e.g. the performance is usually better at weekends). Thus, we would have to split up the traffic into equal chunks and perform a grid search. Large parts of the grid would not result in an acceptable performance and the procedure needs to be repeated every time the model definition changes or a recommender is added or deleted. Thus, this procedure would be very expensive in the end. That is why try to minimize the amount of hyperparameters that we have to fit as much as possible.

Another way to tackle this question is at the heart of the multi armed bandit (MAB) problem. There is a slot machine (bandit) with multiple arms. The goal is to determine which arm results in the highest payoff by iterative playing (it is assumed that the samples are i.i.d) [30, 21]. One of the most successful strategies is estimating an upper confidence interval (UCI) for each arm and then playing the arm with the highest interval [20, 1]. There are different ways to tackle the MAB problem than estimating UCI (i.e. one could use ϵ-greedy as well). However, using an UCI will become beneficial when we try to combine it with model averaging later. One of the most common algorithm is the UCB1 [1]. The definition of UCB1 is given in (1.3).

$$\bar{r_j} + \sqrt{\frac{2\log(n)}{n_j}} \tag{1.3}$$

where $\bar{r_j}$ is the average reward for the arm j, n is the number of overall plays and $n_j$is the number of times arm j has been played. If the second term vanishes, we will fully exploit the current model. This happens when we observe an arm infinity times often because $0 \leq \lim_{n_j \to \infty} ln(\sum_{i=1}^{N} n_i)/n_j \leq \lim_{n_j \to \infty} (ln(N) + ln(n_j))/n_j = 0$. There are several other algorithm available to calculate the UCI. The draw back of UCB1 is that it does not explicitly exploit the fact that our rewards are binary. Again, if we assume that our samples are i.i.d., then the amount of success follows a Binomial distribution. Thus, we can exploit this knowledge to calculate more accurate UCI [16], an UCI for that case is given in (1.4).

$$\frac{\frac{s_j}{n_j} + \frac{z^2_{\alpha/2}}{2n} + \frac{z_{\alpha/2}}{\sqrt{n_j}} \sqrt{(\frac{s_j}{n_j})(1 - \frac{s_j}{n_j}) + \frac{z^2_{\alpha/2}}{4n_j}}}{1 + \frac{z^2_{\alpha/2}}{n_j}} \tag{1.4}$$

where s_j is the number of success for arm j and $z_{\alpha/2}$ is the value of a standard normal variable with the probability $\alpha/2$. One can also look at this problem from a Bayesian perspective. Defining the probability that a recommendation is accepted using the Bayesian theorem is:

$$\mathrm{P}(\gamma_{j,t} = y | Z_t) = \frac{\mathrm{P}(Z_t | \gamma_{j,t} = y)\mathrm{P}(\gamma_{j,t} = y)}{\int_0^1 \mathrm{P}(Z_t | \gamma_{j,t} = y)\mathrm{P}(\gamma_{j,t} = y) dy} \tag{1.5}$$

where $Z_t = \{s_{1,t}, n_{1,t}, ..., s_{R,t}, n_{R,t}\}$ is the set of success and trials at time t for all recommender. As mentioned above, assuming a Binomial distribution for $\mathrm{P}(Z_t | \gamma_{j,t} = y) := \binom{n_{j,t}}{s_{j,t}} y^{s_{j,t}} (1-y)^{n_{j,t} - s_{j,t}}$and using the beta distribution as the prior distribution $\mathrm{P}(\gamma_{j,t} = y) = \frac{1}{B(\alpha_{j,t}, \beta_{j,t} - \alpha_{j,t})} y^{\alpha_{j,t}-1} (1-y)^{\beta_{j,t} - \alpha_{j,t} - 1}$(uniform for $\alpha = 1, \beta = 2$), because it is the conjugate prior of the Binomial distribution, one gets:

$$\mathrm{P}(\gamma_{j,t} = y | Z_t) = \frac{1}{\mathrm{B}(x_{j,t}, n_{j,t} - x_{j,t})} y^{s_{j,t} + \alpha_{j,t} + 1} (1-y)^{n_{j,t} - s_{j,t} + (\beta_{j,t} - \alpha_{j,t}) + 1} \tag{1.6}$$

Thus, $\gamma_{j,t+1} \sim \mathrm{Beta}(\alpha_{j,t} + s_{j,t}, \beta_{j,t} - \alpha_{j,t} + n_{j,t} - x_{j,t})$, which basically means that we just have to sum up the number of trials (β) and success (α) over time for each recommender. Note that the expected value of the Beta distribution is

$$\mathrm{E}(\gamma_{j,t}) = \frac{\alpha_{j,t}}{\beta_{j,t}} \tag{1.7}$$

which is just the probability of seen a success (also called Click Through Rate CTR in the case of news article recommendations).

As it turns out, there is a very easy way to determine which arm has to be played at every point t. The method is known as Optimistic Bayesian Sampling (OBS)[19] and is defined in (1.8)

$$\begin{aligned} Q_{t,j} &\sim \mathrm{Beta}(\alpha_{j,t}, \beta_{j,t} - \alpha_{j,t}) \\ k &= \underset{j}{\mathrm{argmax}} \ \max(Q_{t,j}, \mathrm{E}(\gamma_{j,t})) \end{aligned} \tag{1.8}$$

The first line means that we draw a sample from a beta distribution, but only use this sample if its higher than the expected value of this beta distribution. If the maximum operation is disregarded, the method is called Local Thompson Sampling [19].

We might further improve our estimate by using additional contextual information. Bandits using contextual information are therefore called contextual bandits. One approach comes from Graepel, Candela, Borchert and Herbrich [13]. They basically suggest to perform a Bayesian Probit regression, with a prior normal distribution over the weights and the further assumptions that the contextual information must be binary and only one information could be supplied at a time. This results in a very simple update formula and has the advantage that the input vectors are not correlated, resulting in a posterior distribution of the weights that is still Gaussian and uncorrelated. Next, they sample weights from this Gaussian distribution and calculate the prediction using the sampled weights and the actual weights (as equivalent to (1.7)) and use this to perform OBS as in (1.8). Because the Gaussian is uncorrelated, drawing samples from it is in the complexity of $O(w)$ (w being the number of weights), otherwise it would be $O(w^3)$ in its time complexity. However, enforcing to have only one binary contextual information set is a very strong limitation.

Li, Chu, Langford and Schapire suggested an algorithm called LinUCB (or an extension Hybrid LinUCB [18]). It tries to predict the reward using a ridge regression, which is again $O(w^3)$ or $O(n^2)$(n being the number of samples) in its time complexity. Given that we have to deal with ~ 50 Mio samples per day and potentially thousands of features, this is a very complex task. Also the memory constrains are obviously extreme with this approach. The reason for that time complexity is the need to invert the square of the design matrix. This matrix (and its inverse) is also needed to calculate the UCI. Thus, its not possible to update the UCI in every decision process and the performance of LinUCB would be very much dependent on the update speed of this matrix.

2. Method

As we saw in the previous section that the draw back of most used techniques is that they are not practical for an online use with millions of data points. Thus, we will mainly rely on simple and fast methods. The outline of this section is as follows. First, we will describe how to efficiently estimate the performance of a recommender. Next, we will derive a method how we can use this information to build an ensemble of recommenders. We will also discuss in the end under which conditions these method will produce useful results or rather how recommender might be constructed such that ensembles can result in a better performance.

2.1. Performance Estimation of Recommender

As one can see in (1.1), the purpose of the algorithm is to optimize the choice of items. Because the choice of the items depends on the recommender, its natural to define the rewards in terms of the choice of the recommender. We will then use this definition in combination with the (MAB) problem to tackle the exploration vs. exploitation trade off. The reward of the recommender i at time t is defined as follows:

$$\gamma_{i,t} = \frac{1}{|I_t \bigcap \Gamma_{i,t}|} \sum_{j \in (I_t \bigcap \Gamma_{i,t})} r_{j,t} \tag{2.1}$$

where $\Gamma_{i,t}$is the set items at time t recommended by i, remember that I_t is the set of items being recommended at time t. Thus, the union of both is the set of items that have actually been shown to the user and that have been selected by the recommender i. This definition does not punish recommenders that have not recommended an item in the final set of items. This is useful because some recommenders might deliver good recommendation but not all the time (may be for technical reasons). If $\gamma_{i,t}$ is averaged over time, one gets the CTR (see (1.7)), which is often used as a success measurement of news article recommender [18, 13].

$$ctr_i = \gamma_i = \frac{1}{\sum_{t=1}^{T} |I_t \bigcap \Gamma_{i,t}|} \sum_{t=1}^{T} \sum_{j \in (I_t \bigcap \Gamma_{i,t})} r_{j,t} \tag{2.2}$$

(2.2) is just the amount of accepted recommendation (basically clicks on the link) over the number of recommendation actually given and shown by i (called impressions). However, as explained in section 1.2.2, this highly depends on how often each recommender had the chance to contribute. Thus, we will use MAB and replace 2.2 by some UCI in the end.

2.2. Ensemble Learning

From the review in section 1.2.1 we conclude that we will use model averaging because of its simplicity and computational efficiency. Thus, we need to define the two terms in equation (1.2). $\mathrm{P}(y_i|M_j)$ is given by the recommender itself. We just have to define a value for $\mathrm{P}(M_j|Y)$, the model weight. This gives us basically (2.3).

$$\begin{aligned} \mathrm{P}(y_{t,j}|Y) &= \sum_{r=1}^{R} \mathrm{P}(y_{t,j}|M_r)\mathrm{P}(M_r|Y) \\ &= \sum_{r=1}^{R} \mathrm{f}_{r,j}(x_t)\beta_r \qquad (2.3) \\ 1 &= \sum_{r=1}^{R} \beta_r; \beta_r \geq 0 \qquad (2.4) \end{aligned}$$

There is a slight change in the notation here. The $y_{t,j}$ indicates the value (click / no click) at time t for item j and accordingly $\mathrm{f}_{r,j}(\vec{x_t})$ is the prediction of recommender r at for item j given the input vector $\vec{x}$ at time t and we are looking for values for $\beta_r = \mathrm{P}(M_r|Y)$. Applying Bayes rule we get

$$\mathrm{P}(M_r|Y) \propto \underbrace{\mathrm{P}(Y|X; M_r)}_{\text{likelihood}} \underbrace{\mathrm{P}(M_r)}_{\text{prior}} \qquad (2.5)$$

So we need to define a likelihood function and a prior. First, we will have a look at the prior.

Using the method of maximum entropy we get

$$\begin{aligned} -\sum_{r=1}^{R} \mathrm{P}(\beta_r) \log(P(\beta_r) &= \max \qquad (2.6) \\ \sum_{r=1}^{R} \mathrm{P}(\beta_r) &= 1 \qquad (2.7) \\ \sum_{r=1}^{R} E_{\beta_r}^{\mathrm{Reg}} P(\beta_r) &= E_0 \qquad (2.8) \end{aligned}$$

Where $E_{\beta_r}^{\mathrm{Reg}}$ is some kind of regularization that will enforce some prior knowledge. Solving (2.6) with the constrains in (2.7) and (2.8) using Lagrangian multipliers this results in

$$\mathrm{P}(\beta_r) \propto \exp(-\alpha E_{\beta_r}^{\mathrm{Reg}}) \qquad (2.9)$$

where α is some normalization constant. Because we do not have any prior knowledge and $\mathrm{P}(\beta_r)$ is a discrete distribution we want a regularization function that enforces a

uniform distribution over the weights, because it is the maximum entropy distribution in that case [24]. The regularization function can then be

$$E_{\beta_r}^{\text{Reg}} = \frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2 \tag{2.10}$$

Where $\lambda > 0$ is some regularization constant. This can be shown again using Lagrangian multipliers and constrain in (2.7) (proof see Appendix A.2)

Defining the likelihood function is less specific, there are in principle several possibilities. In general we define it as

$$\mathrm{P}(y_{t,j}|x_i;\vec{\beta}) \propto \exp(-\gamma \mathrm{e}(y_{t,j}, \vec{x}_t, \vec{\beta})) \tag{2.11}$$

where $\gamma > 0$ is again some normalization constant and $\mathrm{e}(y_{t,j}, \vec{x}_t, \vec{\beta})$ is some error function. In particular, we will have a look at error functions that are linear in $\vec{\beta}$, i.e. there are in the form

$$\mathrm{e}(y_t, \vec{x_i}, \vec{\beta}) = \sum_{r=1}^{R} \beta_r \mathrm{h}(y_t, \mathrm{f}_r(\vec{x}_t)) \tag{2.12}$$

where $\mathrm{h}(y_{t,j}, \mathrm{f}_r(\vec{x}_t))$ is some function that should measure the distance (i.e. error function) between the input and output. We choose this error function because we want to use the dot product as error function. We will later see that this will allow us to use the performance measurement defined in section 2.1 naturally. Putting (2.11), (2.10) and (2.9) into (2.5) and assuming that the samples are i.i.d. That gives us:

$$\begin{aligned} \mathrm{P}(M_r|Y) &= \prod_{t=1}^{T} \exp(-\gamma \mathrm{e}(y_{t,j}, x_t, \vec{\beta})) \exp(-\frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2) \\ &= \exp(\sum_{t=1}^{T} -\gamma \mathrm{e}(y_{t,j}, x_t, \vec{\beta}) - \frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2) \end{aligned} \tag{2.13}$$

We want to seek for some $\vec{\beta}$ that maximizes (2.13) or equivalently that minimizes the negative log of (2.13). The result is given in Lemma 1.

Lemma 1. *Let $\gamma \neq 0$ and $\forall r \, \forall j, r \neq j : \sum_{t=1}^{T} h(y_t, f_r(\vec{x_t})) \neq \sum_{t=1}^{T} h(y_t, f_j(\vec{x}_t))$ then the function $-\log(P(M_r|Y)) = \gamma \sum_{t=1}^{T} \sum_{r=1}^{R} \beta_r h(y_t, f_r(\vec{x}_t)) + \frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2$ has exactly one minimum w.r.t $\vec{\beta}$, $\forall r \beta_r \geq 0$, $|\vec{\beta}| = 1$ at*

$$\phi_r = -\gamma \sum_{t=1}^{T} \mathrm{h}(y_t, \mathrm{f}_r(\vec{x}_t)) - \alpha \tag{2.14}$$

$$\alpha = -\gamma \sum_{t=1}^{T} \frac{\sum_{r=1}^{R} \mathrm{h}(y_t, \mathrm{f}_r(\vec{x}_t))}{R} \tag{2.15}$$

$$\beta_r \propto \begin{cases} \phi_r & \text{if } \phi_r > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

Proof. see Appendix A on page 32 □

Corollary 2. *If all recommender perform identical i.e* $\forall r \forall j : \sum_{i=1}^{n} h(y_t, f_r(\vec{x}_t)) = \sum_{i=1}^{n} h(y_t, f_j(\vec{x}_t))$ *then weights are*

$$\beta_r = \frac{1}{R} \tag{2.17}$$

where R is the number of recommender.

It is actually easy to see that the result is independent of the choice of γ. Though, it is useful for the interpretation of Lemma 1 to assume that $\gamma = \frac{1}{T}$ (see proof) and it will limit the values of α and ϕ which is important for numerical reasons.

Corollary 3. *If the assumptions in Lemma 1 are met and* $\gamma = \frac{1}{T}$ *then*

$$\rho_r = \begin{cases} \mathrm{E}[\mu_r] - \mu_r & \text{if } \mathrm{E}[\mu_r] - \mu_r > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

$$\mu_r = \mathrm{E}[\mathrm{h}(\vec{y}, \mathrm{f}_r(\vec{x})] \tag{2.19}$$

$$\beta_r = \frac{\rho_r}{\sum_{r=1}^{R} \rho_r} \tag{2.20}$$

Proof. see on page 32 □

From Corollary 3 we can see that only recommenders will be used that perform better than the mean ensemble or all recommenders if they perform exactly the same. We will refer to this algorithm as Mean Ensemble Threshold (MET) algorithm. Note that (A.1) is very similar to the Elastic net regularization [32]. The main difference is that we require the weight to be positive and use the dot product as error function and not the mean squared difference.

There are two problems with this formulation. The first problem is the again the exploration versus exploitation trade-off. In case a recommender performs slightly less than the mean ensemble, it will be ignored. The second problem is a practical one. In the actual system the value $\mathrm{f}_r(\vec{x}_t)$ is actually not stored. It is in principle possible, but due to time constrains we were not able to change the system in this regrades. However, because we kept the definition of the error function in (2.12) general. We will just assume an error function that is suitable for us and compatible with the performance measurement in (1.2.2). The error function is defined in (2.21)

$$\mathrm{h}(y_t, \mathrm{f}_r(x_t)) = -\sum_{j \in I_t} y_j \tag{2.21}$$

$$\mu_r = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j \in (I_t \bigcap \Gamma_{r,t})} y_{j,t} \tag{2.22}$$

where $y_{j,t} \in \{0, 1\}$ indicates if the recommendation has been accepted and then its mean value is actually the CTR defined in (2.2). This basically means we use the dot product $\mathrm{h}(y_t, \mathrm{f}_r(x_t)) = -\sum_{j \in I_t} y_t \mathrm{f}_r(x_j)$ and set $\mathrm{f}_r(x_j) = 1$. That is why we used a linear error function in the first place. However, in order to tackle the exploration - exploitation trade off (EET) we will use the UCI instead. In the case of OBS there is no UCI we could use. One could probably derive one from it, but we will motivate another method.

In order to derive the expression in (2.5), we used a frequentist approach to find a solution. However, we might also have look at the problem from a Bayesian perspective. Again, assuming a series of success and failures, that this series of success/failures is Binomial distributed and using a beta distribution as prior, we end up with (1.6) , then, using the OBS scheme again, we get an unnormalized weight value.

$$\mu_{r,t} = \max(Q_{t,r}, \mathrm{E}(\gamma_{r,t})) \tag{2.23}$$

where $\mathrm{E}(\gamma_{r,t})$ is the ctr of recommender r and time t and $Q_{r,t}$ is a sample from a beta distribution as in (1.8). Then, we use the normal procedure as in (3) to determine the final weight.

2.2.1. Alternative Weighting

There are, of course, various ways to calculate the final weight β_r. Because we use MAB to estimate UCI to solve EET, we want to present another way of calculating the weights that resembles the selection procedure of the MAB. In the MAB, always the arm with the highest UCI will be played. There is no mixing of arms. Thus, we suggest using a soft-max function to calculate the weights.

$$\beta_r = \frac{\exp(\alpha\gamma_r)}{\sum_{r=1}^{R} \exp(\alpha\gamma_r)} \tag{2.24}$$

where γ_r is the output of the MAB for recommender r and $\alpha \gg 1$. This will enforce that the best recommender will get a weight towards one and the other towards zero, except they are all performing equal.

The choice of α should in general depend on the model uncertainty of the best model uncertainty (believe that the best model is not the true model), which would be maximal if all models perform identical [10]. Thus, α should be high if there is one model that outperforms the others by far and lower the more similar they are.

2.2.2. Remarks on MET

BOC will work best is the model uncertainty is high and otherwise it might perform poorly [10]. One can easily see that is also true for the MET algorithm. If there is one algorithm that outperforms the others by far i.e. $\mu_r \to -\infty$ and $\forall j : j \neq r \, \mu_j \to 0$ then $\mathrm{E}[\mu] \to -\infty$ which implies that the algorithm r will get a weight of 1 and all other a weight of 0. Thus, the ensemble would not perform better than its best algorithm. This behavior is typically for Bayesian model averaging, because it belongs to the class of soft model selection algorithms [22]. In the case that there is one algorithm that

out performs the others by far. It might be a good strategy to create several different version of the original algorithm and uses these to build an ensemble in order to increase the model uncertainty. There is also an alternative to model averaging that one might consider, model combination. We will discuss this alternative in (4.1).

3. Results

The whole system was implemented in a web based news article recommendation system, supplied by Plista GmbH. In all experiments seven different recommender where used. We implemented three non-contextual bandits, the UCB1 algorithm (1.3), UCB-Binomial algorithm (1.4) and the OBS (1.8). Each algorithm is implemented using a soft-max weighting (with an α value of 100 or 5000) and the MET algorithm. We use A/B testing to determine the best algorithm. This minimizes the possibility that a difference in the performances might be attributed to other covariates, like difference in recommender performance, change in the amount of computational resources or others. The current method is to use fix weights that have been adjusted manually by expert knowledge, we will refer to this as default method. All data where aggregated in 30 minutes intervals and stored. This is necessary due to memory constrains.

3.1. CTR

In Figure 1 the performance (i.e. CTR) of the recommender over time is displayed. The CTR reported here is the so called widget CTR. A widget is a set of news articles that are shown to the user (usually at the end of the news article). Because a user does not click multiple times at different items in a widget, we count one presented widget as one impression but we count every click. Thus, in theory a CTR greater than one would be possible.

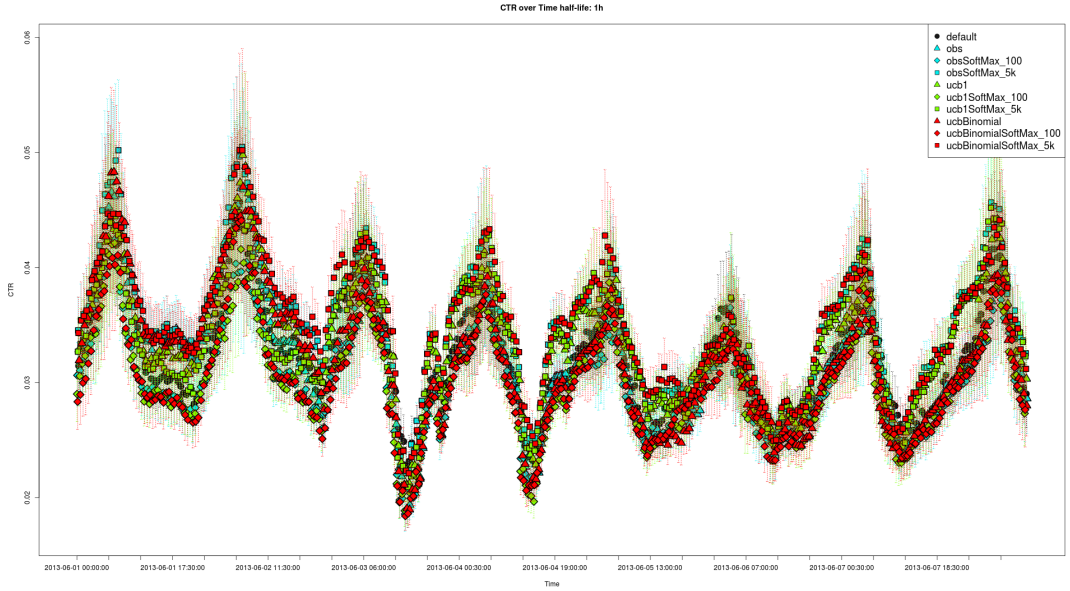
We convolved the raw data with an exponential kernel to smooth the data (we padded the data with the respective mean value). We further average over the the three different MAB types (see Figure 2) as well as over the different normalization techniques (see Figure 3). The half life value always defines the half life time used for the exponential kernel, the decay factor is then $\tau = \frac{\log(2)}{t_{\frac{1}{2}}}$. In table 1 we present the final aggregated results at the end of the experiment. The best approach used the UCB Binomal Bandit with a SOFT-MAX normalization (with $\alpha = 5000$) and resulted in a performance increase compared to the default approach by 8.4%.

Bandit	Normalization	Impressions	Clicks	CTR
Default	-	6506142	200228	3.078%
OBS	MET	8671819	258766	2.984%
OBS	SOFT-MAX 100	7372962	212454	2.882%
OBS	SOFT-MAX 5k	8459904	279449	3.303%
UCB1	MET	8413960	265513	3.156%
UCB1	SOFT-MAX 100	9059800	255414	2.819%
UCB1	SOFT-MAX 5k	7997488	251703	3.147%
UCB Binomial	MET	9054229	272349	3.008%
UCB Binomial	SOFT-MAX 100	8881730	248564	2.799%
UCB Binomial	**SOFT-MAX 5k**	**6959977**	**232252**	**3.337%**

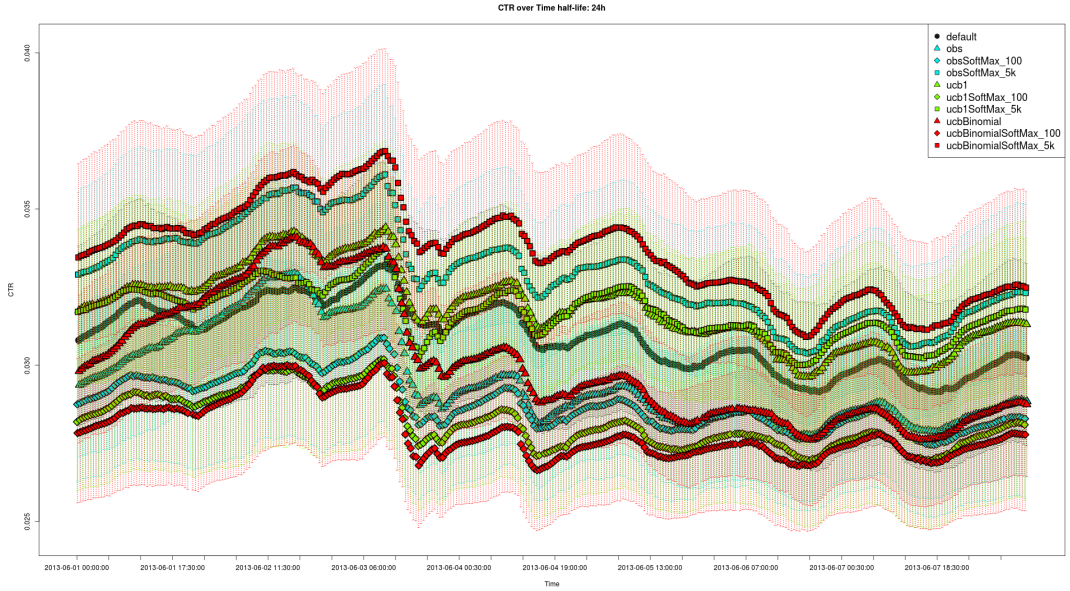
Table 1.: Shows the aggregated result for each approach. Default is the currently used method, where the weights have been adjusted by a human expert, but they remained constant. In bold, the best result.

3.2. Recommender Weights

In order to determine which recommenders mainly drive the recommendation output we will investigate the development of the weights (i.e $\vec{\beta}$ in 2.16) over time. The development of the weight over time can be seen in Figure 4. The corresponding performance of each recommender can be seen in Figure 4.



(a) Uses a exp. kernel with 1h half-life time



(b) exp. kernel with 24h half-life time

Figure 1.: The development of the CTR over time. It begins on a Friday and ends on a Saturday the next week. OBS is Optimistic Bayesian Sampling ucbBinomial is an upper confidence interval estimation assuming a binomial distribution. SoftMax_100 means SoftMax Normalization was used and α is 100, SoftMax_5k means SoftMax Normalization was used and α is 5000, default is as described in the text above.

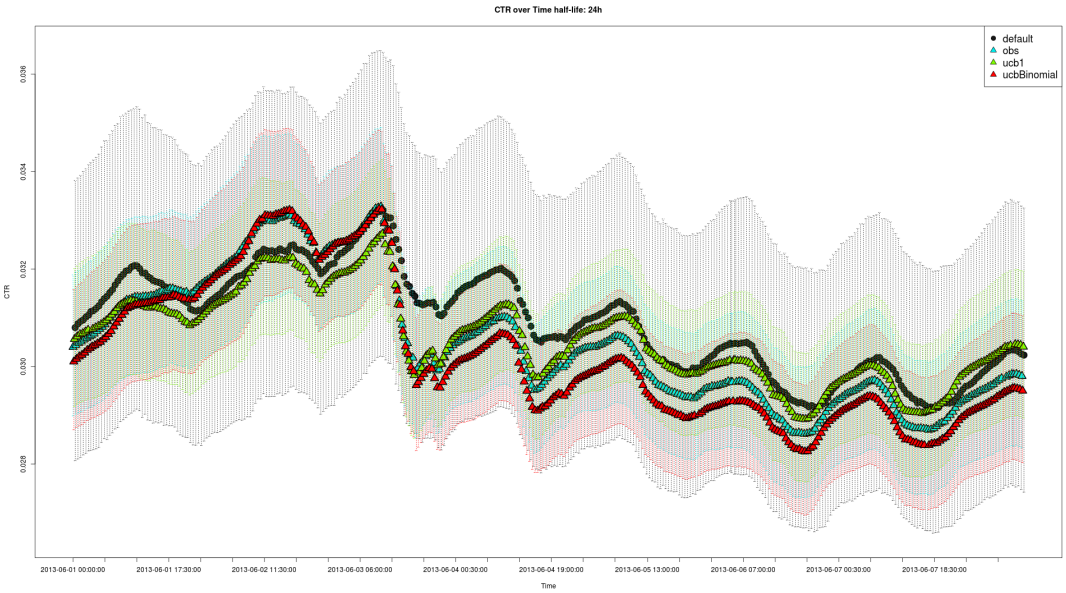


Figure 2.: The CTR over time aggregated for each bandit. OBS is Optimistic Bayesian Sampling ucbBinomial is an upper confidence interval estimation assuming a binomial distribution default is as described in the text above.

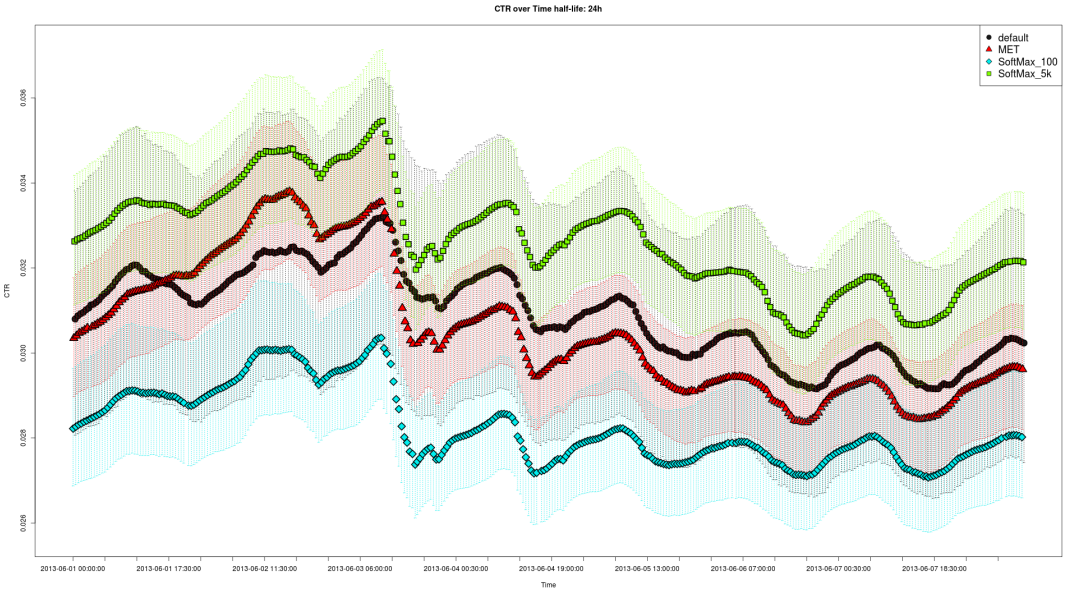


Figure 3.: The CTR over time for different normalization methods. SoftMax_100 means SoftMax Normalization was used and α is 100, SoftMax_5k means SoftMax Normalization was used and α is 5000, default is as described in the text above.

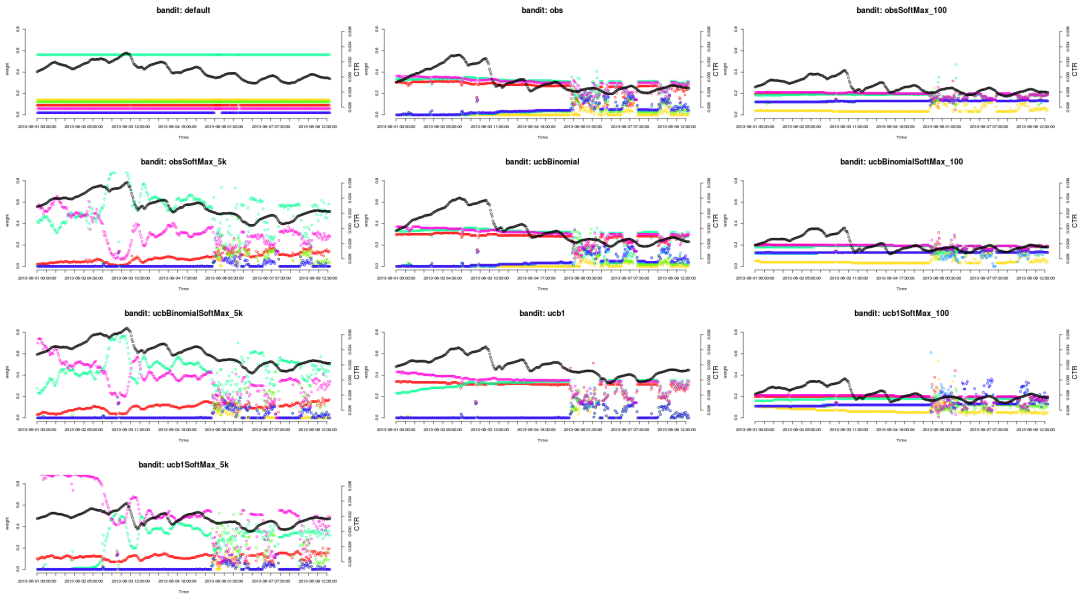


Figure 4.: Weights of different recommender algorithms over time. Colored: this is the weights for different recommenders and in black this is the CTR of the ensemble. The left Y-axis represents the weights and the right Y-axis the CTR. The recommenders: Turquoise/Yellow: A Collaborative Filtering Algorithm Magenta/Red: Semantic Fit Recommender Blue: Top List Recommender. The increased variability at the end in the weights were caused by a database bug in the live-system. However, it effected every approach in the same way. The weights were not smoothed, the CTR were smoothed by an exponential kernel with half life time of 24h.

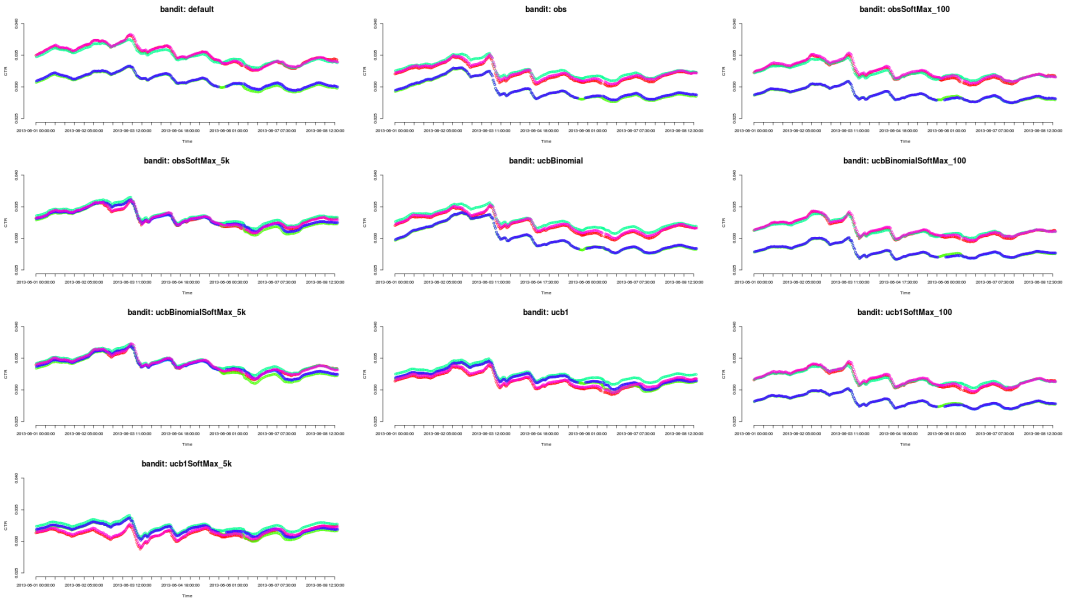


Figure 5.: Show the CTR over time for the different recommenders. Note that in the case that the ensemble performed well, the difference between the recommenders is lower. In order to preserve visibility we scaled the Y-Axis to zoom in, this has the consequence that one recommender that performed very poorly at less than 1% is not shown. The colors match the recommender in Figure 4. The data was not smoothed.

4. Discussion

We showed that BMA in combination with MAB can be used efficiently to combine the recommendation of several algorithms. It turned out the correct choice of the normalization is vital for a successful model averaging (see Figure 3). On the other hand, the choice of the MAB approach is less important (see Figure 2).

It is not surprising that the normalization process is very important for the overall performance. BMA works well if the best model uncertainty is high [10]. However, in our case this is not the case. There are two recommenders (i.e the Collaborative Filtering (CF) approach (Turquoise) and the Semantic Recommender (Magenta)) that outperform the others. Thus, it turned out to be benefical to give extraordinary high weights to these two approaches compared to the others. One has to keep in mind that BMA works as a "soft-selection" algorithm [22]. Consequently, if there are two algorithms that outperform the others, these should be selected, which is the case in the SOFT-MAX normalization with high α (see Figure 4). We might overcome this behavior by using actual model combination instead of model averaging, we will discus this in Section 4.1. On the other hand, finding the best parameter α is part of the hyperparameter selection process, which is a little bit tougher in online system. We will discus this issue in 4.4.

Giving a high weight to the best recommenders had the consequence that in the end most recommender performed better in the SOFT-MAX 5k normalization compared to the others (see Figure 5). However, this is rather a consequence of not including the estimated item likelihood from each recommender (see 2.22). Items that had been estimated likely by the CF or Semantic recommender were chosen. If this item has also been recommended by another recommender, regardless how likely, it got rewarded if this item was clicked in the end. Thus, it will be beneficial to expand the current system to include the item likelihoods, as assumed in the MET algorithm.

There is another very interesting result, that can be seen in Figure 2. The UCB1 Bandit performs worse at the weekend than the other, but better during the week. This is explained by the fact that UCB1 gives, in general, a higher weight to the Semantic recommender and the UCB Binomial or OBS Bandit give higher weights to the CF approach. Thus, the semantic recommender seems to work better during the week compared to the CF recommender. The reason is most likely that the CF algorithm has to be able to update itself very fast to be able to give good recommendations. However, during the week the over all computational work load is higher, thus it can update itself less often and performs slightly worse (this is partially also the explanation why the recommendations are better during the night than day). This highlights two important problem, our data is not stationary and using covariates to predict the CTR can be beneficial, which is not very surprising. The former will be discussed in Section 4.5 and the later in Section 4.2.

4.1. Model Combination

The method of Bayesian Model Averaging (BMA) will only work well if there is one and only one true Data Generating Model among the ensemble [23]. BMA should rather be seen as a procedure for model selection [22]. Monteith, Carroll, Seppi and Martinez suggest using Bayesian Model Combination (BMC). This means one will average ensembles of recommender, not single recommender. BMC is given in (4.1)

$$\mathrm{P}(y_{t,j}|Y) = \sum_{e \in E} \mathrm{P}(y_{t,j}|e)\mathrm{P}(e|Y) \tag{4.1}$$

where E is the set of ensembles. According to Monteith et al., the hope is that: "*it [the ensemble] can augment the hypothesis space with hypotheses that its individual members may not be able to even represent on their own*". They used a very simple method to create the ensembles. The base classifier (in our case the recommender) where weighted and their results averaged. The weights where determined by drawing from a Dirichlet distribution with uniform alpha values. The posterior for each ensemble was then calculated and the most probable weights where used to update the alpha values of the Dirichlet distribution. However, estimating the performance of the ensembles would be again susceptible to fluctuation and using an UCI might be reasonable. This idea of combining methods with MABs is not new though. Fusa-Fekete and Kégl used MAB in combination with boosting to gain performance [8, 7]. They *split* the base classifier into some subsets and estimated their performance using MAB, then they only applied boosting onto the classifier in the best subset. BMC is not restricted to linear weighting of base classifier. One could introduce a variety of base function. In fact, one might even optimize over the structure of the model (e.g. by evolutionary algorithms).

4.2. Contextual Bandit

From section 2.1 and 2.2 we saw that the CTR can be used to estimate the model likelihood. Thus, a more reliable estimate of the CTR would be a vital improvement. Formally, we want to estimate the probability that the recommendation by recommender r will be accepted (i.e. the probability of a click). This is a typical classification problem. The probability belonging to the class "click" is then the CTR. They are several different methods one might use for the classification in general, like support vector machines (SVM), linear regression or Gaussian Processes (GP). Especially, GPs seem very promising for this task, because they are also able to give variance estimates, that can be used to calculate UCI [14]. However, SVMs and GPs are not very fast in training and their computational cost probably exceed their use. Thus, a simple logistic regression might be used.

4.2.1. Efficient Logistic Regression

Because we have to deal with thousands of data points per minute, we have to use streaming solutions. Thus, Stochastic Gradient Descent (SGD) is a possibility. The

model is given in (4.2), the log likelihood function (Binomial distribution) in (4.3)and the derivative in (4.4).

$$\mathrm{f}_\Theta(x) = \frac{1}{1+\exp(-\Theta^T x)} \tag{4.2}$$

$$\mathrm{L}(\Theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i \log(\mathrm{f}_\Theta(x_t)) + (1-y_i)\log(1-\mathrm{f}_\Theta(x_i)) \tag{4.3}$$

$$\frac{d\mathrm{L}(\theta)}{d\theta_j} = \frac{1}{N}\sum_{i=1}^{N}(\mathrm{f}_\theta(x_i) - y_i)x_j \tag{4.4}$$

The usual SGD procedure can be enhanced usually by using the Hessian matrix $\frac{d^2\mathrm{L}(\theta)}{d\theta d\theta^T} \mid_{\theta=\theta*}$ as well ($\Theta^*$is the optimal parameter) [5]. However, computing this matrix is very costly, instead one can use average stochastic gradient descent (ASGD) [5, 31]. ASGD uses also an average model parameter $\bar{\Theta} = \frac{1}{T}\sum_{j=1}^{T}\theta_j$. Polyak and Juditsky showed that $\bar{\Theta}$ converges towards Θ^* as fast as second order SGD [25]. Vanilla ASGD is,

$$\Theta_t = (1-\lambda\gamma_t)\Theta_{t-1} - \gamma_t \frac{d\mathrm{L}(\Theta_{t-1})}{d\Theta_{t-1}} \tag{4.5}$$

$$\bar{\Theta} = \frac{1}{T}\sum_{j=1}^{T}\theta_j \tag{4.6}$$

where γ_t is some learning rate that converges slowly over time, λ is a regularization coefficient and $\bar{\Theta}$ is the average coefficient. The time complexity is linear in the number of features (i.e. dimension of Θ). In [31] Xu showed that the time complexity can be reduced even further to be linear in the number of inputs that are different from zero. It holds always if the gradient is linear in the input vector. Thus, its also applicable to a logistic regression. The algorithm is given below in Algorithm 4.1.

4.2.2. Exploration

An approach is to use the EXP4 algorithm by Auer [2]. However, its computational complexity might be exponential in the numbers of features. Another approach is the LinUCB algorithm which requires an occasional offline learning phase that is very demanding in time and memory. The online phase has a computational complexity that is cubic in the number of features. Thus, this is not really applicable to our problem. Because the computational power is the greatest limitations, a simple ϵ-decreasing Greedy strategy might be useful, where ϵ_t is the probability of choosing an action random. It is defined in 4.7.

$$\epsilon_t = \frac{1}{t^d} \tag{4.7}$$

where d is some hyperparameter, which determines how fast epsilon decays.

Algorithm 4.1 Logistic Regression with average stochastic gradient exploiting sparse vectors.

`initialize` $t = 1, \vec{g} = 0, \vec{\Theta}_0 = \bar{\Theta}_0 = random, \alpha_0 = 1, \beta_0 = 1, \tau_0 = 0, u = \bar{\Theta}_0, \bar{u}_0 = \bar{\Theta}_0$

`while t < T:`

$g_j = \frac{dL(\Theta)}{d\Theta_j}$

$\gamma_t = \gamma_0(1 + \phi\gamma_0 t)^{-c}$

$\alpha_t = \frac{\alpha_{t-1}}{1-\lambda\gamma_t}$

$\beta_t = \frac{\beta_{t-1}}{1-\frac{1}{t}}$

$\tau_t = \tau_{t-1} + \frac{1}{t}\frac{\beta_t}{\alpha_t}$

`for each` $x_{t,h} > 0$:

$u_{t,j} = u_{t-1,j} - \alpha_t\gamma_t g_j$

$\bar{u}_{t,j} = \bar{u}_{t,j} + \tau_{t-1}\alpha_t\gamma_t g_j$

$\bar{\Theta}_j = \frac{\tau_t u_{t,j} + \bar{u}_{t,j}}{\beta_t}$

$\Theta_j = \frac{1}{\alpha_t} u_{t,j}$

4.2.3. Performance

In order to speed up the learning process, it is possible to just randomly discard samples. Instead of a random selection, samples might be chosen by some criterion. This processes is usually called active learning (AL). AL is usually used when the acquisition of a label is very expensive (i.e. by a human expert) and one wants to pick a sample from a pool of unlabeled instances that is in some sense most informative [29]. In our case, the labeling is not expensive at all. Still, we can use the same principles but we have to keep in mind that the method of our choice needs to be faster than actually learning the sample (which was in the time complexity of O(d) where d is the number of inputs different from zero). There are several methods to determine if a new sample should be accepted or discarded. A very simple approach would be to choose a sample that has a high uncertainty (i.e. high entropy), also called uncertainty sampling [29], shown in (4.8).

$$x^* = \underset{x}{\operatorname{argmax}} - \sum_i \mathrm{P}(y_i|x;\Theta)\log(\mathrm{P}(y_i|x;\Theta)) \tag{4.8}$$

Besides that it is unclear what actually 'high' means, uncertainty sampling does not evaluate how representative a sample is and thus might be prone to outliers [29]. There are other methods [e.g. expected model change, expected error reduction] that can measure the information content of a new sample. However, there all suffer the same limitation as uncertainty sampling [29]. A good method should also take into account if a new sample is representative. Methods of this kinds are variance reduction or

density weighting methods. Thus, one could modify (4.8) such that it incorporates this information, this gives rise to (4.9)

$$\begin{aligned} x^* &= \underset{x}{\operatorname{argmax}} - \sum_i \mathrm{P}(y_i|x;\Theta)\mathrm{P}(x)\log(\mathrm{P}(y_i|x;\Theta)\mathrm{P}(x)) \\ &= \underset{x}{\operatorname{argmax}} - \sum_i \mathrm{P}(y_i,x;\Theta)\log(\mathrm{P}(y_i,x;\Theta)) \end{aligned} \tag{4.9}$$

which is just the joint entropy. This prohibits that very unlikely inputs (i.e. outliers) are queried. However, estimating $\mathrm{P}(x)$ has usually the draw back that it is computationally to complex (at least in our case) to be used. Still, using very conservative decision rules, (4.8) might still be applicable, especially because its success is in general very much problem dependent [29, 28].

4.3. Excluding Recommender from the Ensemble

It might be useful to exclude recommender from the recommendation process. In practice, the usual recommendation process is as follows: A recommendation request will be received. The ensemble of recommender is then ask to recommend some items. Usually there is a time limit of a couple of milliseconds, then all results will be collected and the aforementioned method for mixing will be applied and the best n items will be displayed to the user. It is important to understand that there is a fixed amount of computational power available that has to be used to train the recommender, as well as to actually create the recommendation. Thus, if for some reasons a recommender will barely deliver results. One might be better of not to ask the recommender in the first place and rather use the computational power to update the other recommender. Thus, we would like to estimate the probability that a recommender will actually deliver a result (i.e. Success Rate Probability SRP). Again, we face the EET problem. Because we want to estimate a success/failure series again, we are able to apply the same MAB as before to estimate an UCI on the SRP and then only ask a recommender if it reaches a certain threshold (i.e. in 25% of the time its able to deliver a result). However, there is one important difference to the problem before. While it was reasonable to assume that the overall performance of a recommender will be somewhat stable over time. In this case, the SRP might change rapidly. A simple solution to that problem is to just use recent data (i.e. the last two days) in order to estimate the UCI of the SRP. This is a very reasonable approach because the must likely cause of a low SRP is a bug in the system, which will usually fixed rapidly (i.e. in the order of days).

There is also the alternative that a recommender will not return results due to limited computational resources. This might very well happen, especially due to certain peaks in the amount of recommendation requests. However, a simple non-contextual bandit would not be able to detected those. This is a general problem of non-contextual MAB. This might effect the mixing of recommender in section 2.2 as well, because it is likely that the UCI on the CTR has a couple of covariates. These two problems motivate the next section. We will explore how to use contextual information in order to increase our prediction of the CTR, while handling thousands of request per minute.

4.4. Hyperparameter Selection

We already outlined that it is in general tough to determine hyperparameters in an online system. The usual method to determine hyperparameters is by using grid search procedure. In an online system, we would need to do this in an A/B testing fashion. Which means that we would need to split up the traffic into several chunks then searching the performance. If we assume that performance is continuous w.r.t. the hyperparameters, then we could try to estimate the grid. A very naive approach would be to split up the grid into very coarse spaces and then search the best area more closely. If the number of hyperparameters is huge, then it is probably a good idea to perform a greedy search. Meaning, one optimizes one hyperparameter after each other and greedily selects the best value. Thus, it is not necessary to search all combination of hyperparameters. In general one will benefit from using MAB here as well to estimate the performance of each model. If the hyperparameter is huge, it might be beneficial to use a random search instead of a grid search [4].

Another method is to use Sequential Model-Based Global Optimization (SMBO) [3]. It uses a regression model on estimating the performance of the ensemble given the hyperparameters and tries to predict a region of interest for parameter optimization. GP's as well as tree based regression models have been suggested as regression models [4]. However, SMBO assumes a stationary environment. Because the CTR is usually highly time dependent (see Figure 1), this has to be taken into account. Moreover, the computational overhead is significant, though there is effort to bound these methods in time [15]. Still, it remains unclear how much this approach would outperform a random search or grid search.

4.5. Non-Stationarity

An important assumption of the proposed MABs is stationarity. This is obviously wrong. Especially errors in the implementation (and their fixes) will abruptly change the performance of recommenders. A simple approach would be to use some discounting or some sliding window. Garivier and Moulines showed that using these techniques and assuming a sudden change in performance, that UCB1 would still match a logarithmic regret bound [12]. The draw back of these method is that one has to somehow adjust the hyperparameters again. The problem is that a data driven approach would probably not the best choice, because we expect a change in the performance to be caused by some changes in the implementation of the recommender or sever malfunctions (or a change of user behavior). Because these events are rather rare, a data driven approach is likely to fail. However, the amount of data in this task is rather huge. Thus, it might be a good idea to choose a rather high discounting (or short windows a couple of days), as the recommender UCI will converge fast (i.e. < 3 Mio samples).

Instead of using some sort of decay, one could use two or more different time scales simultaneously (e.g. a short and a long one) and use a MAB with a medium time scale to determine which time scale currently works best.

There is also the possibility to use evolutionary algorithm to solve this problem [17]. However, it is rather unlikely that the increase in computational complexity will result in a notable performance increase.

4.6. Conclusion

BMA in combination with MAB is a very fast but also efficient way of combining (or rather selecting) recommenders. In order to further improve the performance, it is important to consider model combination, contextual information and non-stationarity in the data.

Acknowledgements

I'd like to thank my whole family that made it possible for me to go study. Moreover a special thanks to Plista as well as my supervisors, Manfred Opper and Andreas Ruttor that gave me the chance and freedom to work on a topic of my choice, as well as for the support.

Bibliography

[1] Peter Auer, N Cesa-Bianchi, and P Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, pages 235–256, 2002.

[2] Peter Auer, Nicolo Cesa-bianchi, Yoav Freund, and Robert E Schapire. Gambling in a rigged casino : The adversarial multi-armed bandit problem. Technical report, 1998.

[3] James Bergstra, R Bardenet, Y Bengio, and B Kégl. Algorithms for hyper-parameter optimization. *NIPS 2011*, (Nips):1–9, 2011.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305, 2012.

[5] L Bottou. Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT'2010*, (x), 2010.

[6] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

[7] R Busa-Fekete and B Kégl. Accelerating adaboost using ucb. *KDDCup (JMLR W&CP)*, pages 1–12, 2009.

[8] R Busa-Fekete and B Kégl. Fast boosting using adversarial bandits. *27th International Conference on Machine Learning*, (Icml), 2010.

[9] R Caruana, A Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. *Proceedings of ICML'04*, 2004.

[10] Ian Davidson and Wei Fan. When efficient model averaging out-performs boosting and bagging. *Knowledge Discovery in Databases: PKDD 2006*, (1), 2006.

[11] TG Dietterich. Ensemble methods in machine learning. *Multiple classifier systems*, 2000.

[12] Aurèlien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*, pages 1–24, 2008.

[13] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsofts's Bing Search Engine. (April 2009), 2010.

[14] S Grünewälder, JY Audibert, Manfred Opper, and John Shawe-taylor. Regret bounds for gaussian process bandit problems. *Thirteenth International Conference on Artificial Intelligence and Statistics*, 6, 2010.

[15] Frank Hutter and HH Hoos. Time-bounded sequential parameter optimization. 2010.

[16] Leslie Pack Kaelbling. *Learning in Embedded Systems*. PhD thesis, 1990.

[17] D.E. Koulouriotis and a. Xanthopoulos. Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems. *Applied Mathematics and Computation*, 196(2):913–922, March 2008.

[18] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *Nineteenth International Conference on World Wide Web*, 2012.

[19] BC May, Nathan Korda, Anthony Lee, and DS Leslie. Optimistic Bayesian sampling in contextual-bandit problems. *Annals of Applied Probability*, pages 1–24, 2012.

[20] Benedict C May and David S Leslie. Simulation Studies in Optimistic Bayesian Sampling in Contextual-Bandit Problems. 1:1–29, 2011.

[21] R E Mcinerney. Multi-Armed Bandit Bayesian Decision Making PRS Transfer Report Supervised by : Stephen Roberts Hilary Term 2010 Multi-Armed Bandit Bayesian Decision Making. Technical report, 2010.

[22] Thomas P. Minka. Bayesian model averaging is not model combination, 2000.

[23] Kristine Monteith, James L. Carroll, Kevin Seppi, and Tony Martinez. Turning Bayesian model averaging into Bayesian model combination. *The 2011 International Joint Conference on Neural Networks*, pages 2657–2663, July 2011.

[24] Sung Y. Park and Anil K. Bera. Maximum entropy autoregressive conditional heteroskedasticity model. *Journal of Econometrics*, 150(2):219–230, June 2009.

[25] B. T. Polyak and A. B. Juditsky. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, July 1992.

[26] G Rätsch, T Onoda, and KR Müller. Soft margins for AdaBoost. *Machine learning*, pages 287–320, 2001.

[27] Robert E Schapire. Theoretical View of Boosting. In *Computational Learning Theory: Fourth European Conference*, pages 1–10, 1999.

[28] Andrew I. Schein and Lyle H. Ungar. Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265, August 2007.

[29] Burr Settles. Active Learning Literature Survey. *SciencesNew York*, 15(2):201–221, 2010.

[30] Joannès Vermorel and Mehryar Mohri. Multi-armed Bandit Algorithms and Empirical Evaluation. In *ECML*, pages 437–448, 2005.

[31] Wei Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490*, pages 1–19, 2011.

[32] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, April 2005.

Appendix A.

A.1. Proof Lemma 1

Proof. We use the method of Lagrangian multipliers to shown that(2.13) has only one extrema and that it is has the minimum defined in(1)

$$\begin{aligned} -\log(\mathrm{P}(M_r|Y)) &= \gamma \sum_{t=1}^{T}\sum_{r=1}^{R} \beta_r \mathrm{h}(y_t, \mathrm{f}_r(x_t)) + \frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2 \\ \sum_{r=1}^{R} \beta_r &= 1 \\ \beta_r &\geq 0 \end{aligned}$$

Thus, the objective function we seek to minimize is

$$\begin{aligned} \mathrm{L}(\vec{\beta}, \alpha, \lambda, \vec{\rho}) &= \gamma \sum_{t=1}^{T}\sum_{r=1}^{R} \beta_r \mathrm{h}(y_t, \mathrm{f}_r(\vec{x}_t)) + \frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2 + \\ &\quad \alpha(\sum_{r=1}^{R} \beta_r - 1) + \sum_{r=1}^{R} \rho_r \beta_r \stackrel{!}{=} \min \qquad \text{(A.1)} \\ \frac{d\mathrm{L}(\vec{\beta}, \alpha, \lambda, \vec{\rho})}{d\beta_j} &= \gamma \sum_{t=1}^{T} \mathrm{h}(y_t, \mathrm{f}_j(\vec{x}_t)) + \lambda\beta_j + \alpha - \rho_j = 0 \qquad \text{(A.2)} \\ \beta_j &= \frac{-\gamma \sum_{t=1}^{T} \mathrm{h}(y_t, \mathrm{f}_j(\vec{x}_t)) - \alpha + \rho_j}{\lambda} \qquad \text{(A.3)} \\ \frac{d^2\mathrm{L}(\vec{\beta}, \alpha, \lambda, \vec{\rho})}{d\beta_j^2} &= \lambda \qquad \text{(A.4)} \end{aligned}$$

Using the Karush–Kuhn–Tucker conditions it follows that $\beta_r\rho_r = 0$ must be fulfilled, applying this to (A.3) yields $0 = \rho_j^2 - \rho_j(\gamma \sum_{t=1}^{T} \mathrm{h}(y_t, \mathrm{f}_j(\vec{x}_t)) + \alpha)$. Solving this quadratic equations gives two solutions $\rho_j = \{0, \gamma \sum_{t=1}^{T} \mathrm{h}(y_t, \mathrm{f}_j(\vec{x}_t)) + \alpha\}$.We know that $\rho_j = \begin{cases} 0 & \text{if}\, \beta_j > 0 \\ \gamma \sum_{t=1}^{T}(\mathrm{h}(y_t, \mathrm{f}_j(\vec{x}_t)) + \alpha) & \text{otherwise} \end{cases}$ (otherwise it would not be a valid solution). Using the constrain that$\sum_{r=1}^{R} \beta_r = 1$ together with (A.3) we see that $\lambda = \sum_{r=1}^{R}(-\gamma \sum_{t=1}^{T} \mathrm{h}(y_t, \mathrm{f}_r(\vec{x}_t)) - \alpha) - \rho_r$ is the normalization constant. Rewriting yields

$$\lambda = \sum_{r\in\{j:\beta_j>0\}}^{R}(-\gamma\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))-\alpha)+ \underbrace{\sum_{r\in\{j:\beta_j\leq 0\}}^{R}(-\gamma\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))-\alpha)+(\gamma\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_j(\vec{x}_t))+\alpha)}_{0} \quad (A.5)$$

$$\lambda = -\gamma\sum_{r\in\{j:\beta_j>0\}}^{R}(\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))+\alpha) \quad (A.6)$$

Plugging (A.6) back into (A.3) and using again $\sum_{r=1}^{R}\beta_r = 1$. We get:

$$-\gamma\sum_{r\in\{j:\beta_j>0\}}^{R}(\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))+\alpha) = \sum_{r=1}^{R}(-\gamma\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))-\alpha)-\rho_r$$

$$\begin{aligned}-\gamma\sum_{r\in\{j:\beta_j>0\}}^{R}\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))-\alpha|\{j:\beta_j>0\}| &= \sum_{r=1}^{R}(-\gamma\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))-\alpha R+\\ &\quad \gamma\sum_{r\in\{j:\beta_j>0\}}^{R}\sum_{t=1}^{T}\mathrm{h}(y_t,f_r(\vec{x}_t))\\ &\quad +\alpha|\{j:\beta_j>0\}| \quad (A.7)\\ \alpha &= \frac{-\gamma\sum_{r=1}^{R}(\sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t))}{R}\\ \alpha &\stackrel{!}{=} -\mathrm{E}(\mathrm{h}(\vec{y},\vec{f}(\vec{x})) \quad (A.8)\end{aligned}$$

The last line is valid for $\gamma = \frac{1}{T}$, $T\to\infty$. Now we have values for $\vec{\rho}$ and α and can put everything together. The unnormalized $\beta_r$value for $\gamma = \frac{1}{T}$, $T\to\infty$ is

$$\beta_r \propto \mathrm{E}(\mathrm{h}(\vec{y},\vec{f}(\vec{x}))-\mathrm{E}_r(\mathrm{h}(\vec{y},\mathrm{f}_r(\vec{x}))+\rho_r \quad (A.9)$$

where $\mathrm{E}_r(x)$ refers to the expectation over the ensemble and $\mathrm{E}(x)$ is the expectation over time and ensemble. Thus, whenever the mean error of a recommender exceeds the mean error of all recommenders β_r would be negative, in this case we see that ρ_r must be greater than 0, which results in $\beta_r = 0$. Normalizing those values then results in (2.16). The argument also holds for other values of γ.

So far we showed the position of the local extrema. Now we will show that it is a minimum. We showed that the second derivative (A.4) is just the normalization constant in (A.6). Therefore we have to show that $\lambda > 0$. Using (A.9) we see $\lambda = \sum_{r\in\{j:\beta_j>0\}}\beta_r > 0$. However, there is the possibility that $\{j:\beta_j>0\} = \emptyset$ if all $\beta_j = 0$, which happens if all recommender perform exactly the same. In this case the function would be at a saddle point. However, because we assumed that $\forall r\,\forall j, r\neq j : \sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_r(\vec{x}_t)) \neq \sum_{t=1}^{T}\mathrm{h}(y_t,\mathrm{f}_j(\vec{x}_t))$ this will not happen. Though, this gives rise to corollary 2. □

A.2. Proof of Regularization Function

Lemma. The minimum of the function $E_{\beta_r}^{\text{Reg}} = \frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2$ w.r.t. $\vec{\beta}$, $\sum_{r=1}^{R} \beta_r = 1$ is the discrete uniform distribution.

Proof.

$$\begin{aligned}
\mathrm{L}(\lambda, \vec{\beta}, \alpha) &= \frac{1}{2}\lambda \sum_{r=1}^{R} \beta_r^2 + \alpha(\sum_{r=1}^{R} \beta_r - 1) \\
\frac{d\mathrm{L}}{d\beta_r} &= \lambda\beta_r + \alpha \stackrel{!}{=} 0 \qquad \text{(A.10)} \\
\frac{d^2\mathrm{L}}{d\beta_r^2} &= \lambda > 0 \\
\sum_{r=1}^{R} \beta_r &= 1 \text{ inserting into (A.10)} \\
-\sum_{r=1}^{R} \frac{\alpha}{\lambda} &= 1 \text{ inserting back into (A.10)} \\
\beta_r &= \frac{1}{R}
\end{aligned}$$

□