# Optimizing engagement in online news

Andrei Oghină

Master of Science Thesis

University of Amsterdam

University of Amsterdam, Faculty of Science

# Optimizing engagement in online news

Master of Science Thesis

For the degree of Master of Science in Artificial Intelligence at
University of Amsterdam

Andrei Oghină

Supervisor: prof. Maarten de Rijke
Mentors: dr. Manos Tsagkias, Maximiliano Neustadt

February 20, 2013

Faculty of Science · University of Amsterdam

The work in this thesis was supported by Hyves B.V. Their cooperation is hereby gratefully acknowledged.

University of Amsterdam

# Abstract

As the internet is becoming the primary source of information, news websites are competing to provide the best stories. Each day, dozens of high-impact events make the headlines. In the same time, mobile browsing is exploding, bringing the news *paper* to a previously unthinkable small size. Given this informational avalanche and shrinking formats, it becomes crucial to develop methods that quickly detect the most engaging stories.

In this master thesis, I apply reinforcement learning methods to detect engaging stories in real time, on a medium-sized news website. I then cluster visitors based on their click histories and evaluate their age and gender stereotyping. I investigate how this insight can be applied for news personalization. Finally, I adapt and test the methods on a mobile news website, to study the impact of the mobile setting on news engagement optimization.

Experimental results show how a content-agnostic reinforcement learning method can achieve click-through rate lifts of up to 92% over a random (or most recent article) baseline, when applied to the top story position of a news website. Due to skewed user demographics and other limitations, user segmentation doesn't improve performance on the tested website. However, clustering based on click histories outline a young female user group with distinctive interests. The tested methods prove to be medium-invariant: when applied on a mobile website, they boost a 108% lift in click-through rate comparing to a random serving scheme, and a 20% increase when comparing to an informed manual approach, where members of an editorial team place good articles in the top story position.

Through this work, I highlight the potential of content-agnostic, online reinforcement learning methods for solving engagement optimization problems, and encourage further research and new applications based on such methods.

# Table of Contents

# Acknowledgements

I thank my supervisor, prof. Maarten de Rijke, and my academic mentor, dr. Manos Tsagkias, for giving me the freedom to explore such an interesting topic, and for their kind guidance during the writing of this thesis.

I'm grateful to Maximiliano Neustadt from Hyves, for sparking the idea of this work, and for his continuing support throughout my internship.

University of Amsterdam Andrei Oghină

February 20, 2013

To my late grandparents: Elena & Gheorghe Vranciu, Angela & Bogdan Oghină.

# Chapter 1

# Introduction

The internet is clearly emerging as a predominant source of information. It has long surpassed printed newspapers, and is currently closing in on television.[1] As more people head online for their daily informational intake, this transformation of the news landscape comes with both unique challenges and opportunities.

Major online news outlets serve tens of millions of monthly visitors.[2] When a new story is published on the main page, its instant exposure is huge. Without the delay and practical limitations of print, the number of potential stories is virtually unlimited. This can easily lead to informational overload, since human attention span is relatively constant. Therefore, the question is: how to pick the most informative and engaging stories?

News websites have been around for more than two decades. However, until recently, they still shared a similar, manual publishing process with their printed counterparts. While the details vary by publication, an editorial team is usually responsible with filtering news, writing stories, and selecting which items appear on the main page. Most websites also have a *top story* - a highlighted article which is the equivalent of the *front page story* featured by traditional newspapers. This headline is usually hand-picked by the editorial team.

In online news, probably the most obvious gains are accessibility and recency, as users can read stories instantly once they are published, from any place in the world. However, there are other important advantages in this setting, two of which are particularly relevant for this work. First, in the online environment, it is easy to monitor user behaviour in real-time, for instance tracking which articles are clicked more often. Secondly, because web pages are dynamic, content can be reordered and displayed in various ways. Moreover, personalization is possible, by highlighting or ordering stories differently for individual users.

While most online news publishers still have a manual editorial process, some of the big websites are making efforts to automatically optimize user engagement, taking advantage of the online setting. Probably the most notable examples are Google News (Das et al., 2007)

[1] http://www.people-press.org/2012/09/27/in-changing-news-landscape-even-television-is-vulnerable/
[2] http://www.quantcast.com/cnn.com

and Yahoo! News (Li et al., 2010), which both use automated personalization techniques. While their approaches differ, there is a common ground in learning from user behaviour.

Despite the potentially significant impact on both user experience and commercial gain, the work in this field is rather limited, for multiple reasons. For instance, Agarwal et al. (2008) mention the technical and cultural challenges in convincing experienced editorial teams to accept automated serving schemes, and the necessity of conducting tests on live traffic for statistical validity. Few companies have the resources, traffic and openness needed for such research, and business information sensitivity prevents open data sets to become available.

More recently, mobile news consumption is exploding, with the percentage of Americans who regularly get news on a mobile device doubling in two years, reaching 15%.[3] The screen size constrains of these small devices have already changed website design paradigms, and make it even more important for news websites to pick the most engaging headlines.

## 1-1 Contribution

For this thesis, I have the opportunity to explore content optimization methods on live traffic from the news section of the Hyves social network.[4] To the best of my knowledge, this is the first time such experiments are reported on a medium-sized news website. I hope that my results can pave the way for openly available content optimization tools that can benefit both online publishers and their readers. That's why I report not only results, but also architectural details on how to plug such algorithms on existing news platforms.

Based on a content-agnostic, behavioural approach, I use multi-armed bandit models to detect which stories generate most engagement. For tuning, I devise a method to generate artificial user interactions, to be used for offline evaluation. The goal is to maximize the number of clicks on the top story. This also leads to higher user satisfaction, and ultimately to an increase in user visits (Liu et al., 2010).

Since Hyves is a social network, I have access to user features such as age and gender. This allows me to investigate the extent to which clicking behaviour can be stereotyped with gender and age segmentation. I employ the results to attempt a novel approach to personalization, combining click-based user clustering with multi-armed bandit models.

I am also given the chance to run experiments on a different, mobile news website. This allows me to adapt the methods to a different setting, and gather valuable data on how the constrains of the mobile world impact on the proposed serving schemes, and ultimately on user behaviour.

I consider the following research questions:

1. Is click-based reinforcement learning a suitable method capable of lifting user engagement, when applied to top story selection on a medium-sized news website?
2. Can top article clicking patterns be stereotyped using traditional gender and age segmentation, and is this insight useful for achieving news personalization?

[3] http://www.people-press.org/2012/09/27/section-2-online-and-digital-news-2/
[4] http://hyves.nl

3. Are these methods applicable to a mobile news website, and do the particularities of a mobile setting influence the performance of click-based reinforcement learning top story selection methods?

## 1-2 Overview

This thesis is organized as follows. In Chapter 2, I present relevant related work, covering the areas of recommendations, news filtering and personalization, click-based user clustering, reinforcement learning and multi-armed bandits. In Chapter 3, I describe how the multi-armed bandit problem can be used for modelling user engagement and selecting the best top story on a news website. Then, in Chapter 4, I use cluster analysis to group users based on their click histories, and apply this insight for news personalisation. To test how the methods generalize to a different user setting, I apply them to a mobile website in Chapter 5, and devise a platform than allows easy integration on virtually any news website. Finally, in Chapter 6 I present my conclusions and discuss possible future work.

# Chapter 2

# Related Work

News filtering and personalization is often formulated as a recommendation problem. As opposed to traditional search, where users have a clear information need, in a recommendation setting they come with a *show me something interesting* attitude (Das et al., 2007). As you can imagine, there are endless possibilities to address such an open question, and an entire new family of methods is required to cope with this broad mindset.

## 2-1 Collaborative and content-based filtering

One of the most successful and commonly used technologies for generating recommendations is collaborative filtering (Sarwar et al., 2001). It works by providing recommendations based on the opinions of other like-minded users. Depending on the scenario, the users can be shop customers that purchase products, film lovers that rate movies, or readers that click on articles. Notice that all these settings are fundamentally similar, with users expressing explicit (ratings) or implicit (clicks) opinions on some type of items.

The user-based collaborative filtering approach computes the similarity between users, based on their past activities (similar movie ratings, or clicks on the same articles). After finding the nearest neighbours of an active user, it combines their preferences to generate recommendations. To address the scalability limitations of this method, Amazon has adopted an item-centric approach (Linden et al., 2003), named item-based collaborative filtering. The intuition is that the user would be interested in items similar to the ones liked (purchased, clicked) earlier. A similar-items table is pre-computed offline, based on the users that have clicked, liked or rated them. The resulting online performance is high, since the only required operation is to retrieve and merge the most similar items, given a user's profile.

The traditional collaborative filtering approach is widely successful, reportedly accounting for 30% of Amazon's revenue[1], and a 200% increase in click-through rate for YouTube's "related videos", comparing to just showing popular items (Davidson et al., 2010). However,

[1] http://blog.kiwitobes.com/?p=58

the method assumes the item-set suffers minimal churn. This assumption holds well for online shops, film websites or YouTube, where recommending an item which is a few months old is typical. But this is not the case in online news, where one of the biggest challenges is the dynamic article pool (Agarwal et al., 2008). An article has an average lifetime of just a few hours. Moreover, even during this short period, its performance, in terms of click-through rate, can change dramatically, an vary for different user groups. This makes the temporal dynamics of interests, which has been previously studied in the context of movie ratings (Koren, 2010), much more problematic for the news setting.

Das et al. (2007) describe an approach for generating recommendations for users of Google News[2]. They use collaborative filtering, but acknowledge that the high item churn present in the news setting makes the traditional approach unsuitable for the task. Instead, they begin by clustering the users based on their click-histories. Once the users are clustered, the recommendations are generated based on the number of clicks on each article, decayed by time, of all users in the same cluster with the active user. They observe a 38% increase in click-through rate for their best method, compared with a popularity-based baseline.

The collaborative-filtering approach to news personalization comes with an obvious limitation: there's no mechanism for new articles to get recommended. This is known as the first-rater problem (Good et al., 1999). News recommendations are particularly sensitive to this issue, due to the high item churn. The system has to wait for hours to collect enough clicks from other parts of the website, to be able to recommend a new story. For this, it has to model in the position bias (users tend to click more on articles displayed more prominently). But what if the system doesn't have access to clicks from other story placements?

Conceptually, Das et al. (2007) cluster the users based on the overlap of their click histories. In practice, they use two probabilistic clustering techniques: PLSI (Hofmann, 2004) and LSH (Indyk and Motwani, 1998). These methods are efficient and scalable, because they do not require computing the similarities between all user pairs. There are various evaluation measures to compare different clusterings, and Vinh et al. (2009) give a good overview of the challenges and solutions for this task.

In their follow-up work, Liu et al. (2010) describe the drawbacks of collaborative filtering applied to news, and propose a solution for improving the recommendations of Google News. They adopt a hybrid method which combines collaborative filtering with content-based information filtering, one of the first approaches used for news filtering (Carreira et al., 2004). Text classification is used to assign categories to each article, and then the topical preferences of users are learned. The collaborative filtering and content-based scores are combined to generate recommendations, allowing new articles to surface based solely on their topic relevance for the active user. This hybrid method improves with 30% over the pure collaborative filtering approach.

Content-based filtering can therefore sometimes improve performance, especially when used in combination with collaborative filtering (Good et al., 1999), but it also makes the system language-dependent and content-focused. Such an approach couldn't be applied to other domains, such as images, videos or music, making it less attractive. Even for news, there are many signals that contribute to a story's success, besides its textual content. For instance, in our setting, the prominent image or video associated with each article plays an

[2] http://news.google.com/

important role, as the content is usually quite short. Even when content is king, *there is wide variability in article performance sharing a common set of feature values*, making it *difficult to build good models based solely on offline data* (Agarwal et al., 2008). Motivated from these findings, and because a content-based method has been already tested on the available setup, I do not look into content-based methods.

## 2-2 Online models

To overcome the limitations of content-based methods and collaborative filtering, I chose to take an alternative, content-agnostic approach that tracks article performance using online models. More precisely, I plan to combine, for the first time, online models (Li et al., 2010) with user clustering methods based on click histories (Das et al., 2007), in an attempt to bridge the two recommendation approaches for serving medium-sized news websites. For this, I take a closer look to modelling user behaviour in the context of online learning.

In recent years, researchers started to exploit the tremendous value of implicit feedback coming from user interactions, such as clicks. Most of the attention is focused on utilizing user feedback to improve document rankings. This family of methods is collectively known as *online learning to rank*. For instance, Radlinski and Joachims (2007) propose an active exploration technique for learning rankings of documents, based on search engine logs of user behaviour. More recently, Hofmann et al. (2013) formulate the online learning to rank problem as an exploration-exploitation dilemma and address it using listwise and pairwise methods, which are known approaches for learning to rank (Liu, 2009). This research area area nourishes from the valuable insights that user interactions provide.

Online models have also been applied for news recommendations. To the best of my knowledge, the only known works on this topic come from Yahoo! Research (Agarwal et al. (2008) and Li et al. (2010)). Being one of the biggest internet companies, Yahoo! (like Google) affords to perform such research, especially since it powers the most visited portal of the web[3]. In this work, I want to investigate and make it easy to use such methods for medium-sized news websites. Li et al. (2010) frame the challenge as an exploration-exploitation dilemma (like in Hofmann et al. (2013)) and propose a contextual bandit algorithm as a solution. Before diving into the details, let me introduce a few key works from reinforcement learning.

Rooted in learning psychology (Minsky, 1954), reinforcement learning is an area of machine learning which studies how an agent should perform actions in order to maximize a numerical reward (Sutton and Barto, 1998). Typically focusing on online performance, one of the challenges of reinforcement learning is how to balance exploration and exploitation. Exploration refers to taking actions that are known to give good rewards, while exploration is needed to discover new and potentially better actions. The exploration-exploitation dilemma is that neither exploitation nor exploration can be successful by its own. This trade-off has been studied extensively through the multi-armed bandit problem.

The multi-armed bandit problem is defined as how to maximize the expected total reward over a time period, when faced repeatedly with a choice that gives a reward from a stationary, action-dependent probability distribution. Originally, it was named based on the analogy of

[3] http://www.alexa.com/topsites/countries/US

maximizing winnings when playing a number of slot machines (bandits). A very good analysis of the problem is performed by Auer et al. (2002), who present efficient policies on how to chose the next action based on past plays and observed rewards. One of the most basic policy is to play randomly a small fraction of the time (random exploration), and the rest of the time, play the current best performing action (greedy exploitation). This strategy is called $\epsilon$-greedy.

In our context, selecting an action is equivalent to picking which article to display to the user, the reward being 1 or 0 depending on if the user clicks on the article. Agarwal et al. (2009) explain the additional challenges for this setup, including the dynamic set of items, article lifetime constrains, non-stationary click-through rates, and the delay in observing user feedback, due to system performance constrains. They propose a Bayesian solution, which improves learning performance with 35% against the $\epsilon$-greedy bandit scheme, but achieves similar performance for exploitation.

Finally, Li et al. (2010) propose a new contextual bandit algorithm based on upper confidence bound, a method to balance exploration and exploitation efficiently. Context refers to information about the users and articles, such as age, gender and article category. They also introduce an offline evaluation platform based on previously recorded random traffic, and their results show a 12.5% click-lift compared to a context-free bandit algorithm.

Because of the interactive nature of the problem, offline evaluation of bandit algorithms can be *frustratingly difficult* (Li et al., 2010). Initial approaches were prohibitively complex and came with multiple caveats (Langford et al., 2008). In their work, Li et al. (2010) propose a much simpler and unbiased method to perform off-policy evaluation, with the constrain that the logging policy chooses each article uniformly at random. In follow-up work, Li et al. (2011) describe their evaluation framework in detail and provide theoretical guarantees such as unbiasedness and accuracy.

Recent work lifts the constrain of having a random logging policy, using non-random data for offline evaluation (Strehl et al., 2010). These evaluation methods, however, reject the majority of logged data, because they skip all events that do not match the evaluated policy. This is very wasteful, and can easily become a problem when data is not available in abundance. For instance, the authors use 40 million recorded events (page views) collected over just a few days to evaluate their methods. Such traffic volume is clearly infeasible to capture by a medium-sized news website. Luckily for me, I was allowed to perform online evaluation, so I didn't have to worry about this problem.

This thesis differentiates from existing work in several ways. First, it uses pure content-agnostic online models to detect engaging stories. To the best of my knowledge, only Agarwal et al. (2008) and Li et al. (2010) take a similar route. As opposed to their work, which focuses on optimizing engagement on Yahoo's main page, this thesis focuses on the feasibility of such methods when applied to medium-sized news websites. I cover both solutions for easy integration of such algorithms with existing news platforms, and the challenges that arise from working at smaller traffic scale. Having access to user demographics, I can provide for the first time insight into how click-based user clustering matches segmentation based on surface features (such as age and gender) for news story clicks. Moreover, for the first time in known literature, I describe how such methods can be plugged to a mobile news website, and investigate how the mobile setting affects their performance.

# Chapter 3

# Detecting engaging stories

The first step in my quest of optimizing user engagement on a medium-sized news website is to quickly identify the engaging potential of new stories, after their publication. For this, I apply content-agnostic reinforcement learning methods, and learn exclusively from user interactions.

## 3-1 Methods

In this section, I describe the multi-armed bandit problem and how it can be applied to detect engaging stories. I also discuss tuning, in the larger context of evaluation, which is known to be difficult for this class of problems. Finally, I give a brief overview of the other content optimization methods already in use at Hyves.

### 3-1-1 Multi-armed bandits

In a multi-armed bandit problem, the player is faced repeatedly with a choice among $K$ different options, and receives, after each choice, a numerical reward, based on the selected action. The objective is to maximize the expected total reward. In order to do so, the player has to keep a balance between exploring all possible actions to find the best ones, and choosing the best known action at each point in time.

When applied to news story selection, the available choices, called *actions* (machines or arms, in the bandit paradigm), are recent stories, from which the algorithm has to select one to show the visitor. The reward may consist of a click (if the page view leads to a click, then the reward is 1, otherwise it's 0) on the displayed article. The expected reward is the article's *click-through rate* (CTR), defined as the division of the number of clicks on an article when displayed in a certain position by the number of page views of that article in that position. Thus, the player, which in this case is the article serving policy, has to pick articles for each visitor, so that the total number of clicks is maximized.

Formally, A *K-armed bandit problem* is defined by a set of random variables $X_{a,n}$ for $1 \leq a \leq K$ and $n \geq 1$, where each $a$ is the index of a gambling machine (one of the bandit's arms), and $n$ the play index of the $a$th machine. That is, after each play of the $a$th machine, the $X_{a,1}, X_{a,2}, X_{a,3}...$ rewards (clicks) are observed, which are independent and identically distributed according to an unknown law with unknown expectation $\mu_a$ (in the news setting, the click-through rate).

The bandit problem generalizes to any scenario where different actions (originally, machines) are available, and yield different numerical outcomes (rewards), with the goal to maximize the total reward across a number of successive plays, or *trials*. In the news setting described here, these actions refer to selecting the next article to serve at each page view. Hence, the articles are the actions, and the page views are the trials. Moving forward, I will use the more general term *action* instead of *machine*, while *plays* and *trials* will be used interchangeably.

A *policy*, or *allocation strategy*, is an algorithm $A$ that chooses the next action to take (the next article to serve, or machine to play), based on past plays and obtained rewards (Auer et al., 2002).

The *regret* of a policy is defined as the expected loss of $A$, when compared to an omniscient strategy that always plays the best action. If $T_a(n)$ is the number of times $A$ played action $a$ at trial $n$, then the regret $R_A(n)$ after $n$ trials is:

$$R_A(n) = \mu^* n - \sum_{a=1}^{K} \mu_a \mathbb{E}[T_a(n)] \tag{3-1}$$

Here, $\mu^*$ is given by the action with the highest reward expectation. In our case, this would be the highest click-through rate:

$$\mu^* = \max_{1 \leq a \leq K} \mu_a \tag{3-2}$$

A well-known policy for the $K$-armed bandit problem is $\epsilon$-*greedy*. This policy sets a small value for $\epsilon$, and plays with probability $\epsilon$ a random action (exploration), and with probability $1-\epsilon$ the action with the highest current average reward (exploitation). Because of the constant exploration probability $\epsilon$, the regret increases linearly.

For the bandit problem, minimizing regret is key to improving performance. In the news publishing setting, the importance of a small regret goes beyond immediate performance (losing clicks). Serving random, potentially not engaging stories to visitors for the purpose of exploration (finding out which story performs best), can have a negative impact on the overall user experience, potentially decreasing the interest in the website.

Previous work proved that it is possible to achieve better, logarithmic growth of regret, which is clearly preferable than a linear increase. Lai and Robbins (1985) presented policies that play exponentially more often the best action, having, for any other suboptimal action $a$:

$$\mathbb{E}[T_a(n)] \geq \frac{\ln n}{D(p_a||p^*)} \tag{3-3}$$

Here, $D(p_a||p^*)$ is the Kullback-Leiber divergence between the reward density of action $a$, and the reward density of the action with highest reward, $\mu^*$ – hence, a constant. As a consequence, regret grows at least logarithmically, so these policies satisfy a regret asymptotically bounded by logarithm of $n$, which was also proved to be optimal.

The $\epsilon$-greedy policy can be improved to achieve logarithmic regret bound, by decreasing $\epsilon$ with a certain rate (for instance, $1/n$). This effectively reduces exploration, as the confidence in the expected reward becomes higher. The new policy is called $\epsilon_n$-*greedy*.

Another allocation policy that achieves logarithmic regret is *upper confidence bound* (UCB) (Auer et al., 2002). It computes a score for each bandit by summing two terms: the current average reward, and a confidence bound for the average reward. Then, it picks the action with the highest score:

$$a_n = \underset{a}{\operatorname{argmax}} \left( \mu_a + \sqrt{\frac{2 \ln n}{T_a(n)}} \right) \tag{3-4}$$

Here, $\mu_a$ is the average reward (the expectation) for action $a$, $T_a(n)$ is the number of times action $a$ has been selected, while $n$ is the total number of trials. This effectively results in a smarter way to balance exploration and exploitation, by exploring more the arms that have been played less, and thus have a bigger confidence bound.

The methods described so far are designed for a scenario where the bandits have an unknown, yet determined expectation. However, in the news setting, one of the challenges is the non-stationary click-through rate. This implies that the expectation is permanently changing, and the bandit method has to take this into account. Moreover, new articles are constantly added to the pool, while old ones get dropped.

In general, in this non-stationary setting, there isn't one action that has the highest reward expectation across all trials. Instead, at each trial, there can be a different action that yields the best reward (Li et al., 2010). By consequence, after expanding the expectations, Equation 3-1 becomes:

$$R_A(n) = \mathbb{E}\left[ \sum_{t=1}^{n} r_{t,a_t^*} \right] - \mathbb{E}\left[ \sum_{t=1}^{n} r_{t,a_t} \right] \tag{3-5}$$

Here, $a_t^*$ is the best action at trial $t$, $n$ is the current trial, and $r_{t,a_t}$ is the reward observed by playing action $a$ at trial $t$. The assumption that the confidence in the expected rewards of all actions increases as the number of plays increases doesn't hold anymore, because new actions become available during the play. This makes the $\epsilon_n$-greedy policy impractical, as it reduces exploration progressively across the entire news items pool.

The upper confidence bound (UCB) algorithm is more appropriate for exploring in a non-stationary expectation setting, because it computes a confidence bound for each arm independently. However, since articles come and go and the algorithm runs continuously, the total number of plays $n$ becomes less relevant, so the confidence is defined using a constant parameter $\alpha$, which is subject to tuning. Equation 3-4 becomes:

$$a_n = \underset{a}{\operatorname{argmax}} \left( \mu_a + \frac{\alpha}{\sqrt{T_a(n)}} \right) \tag{3-6}$$

Therefore, at each step, the action is selected based on the current expectation (click-through rate) $\mu_a$, and the confidence bound, which is dependent on the constant $\alpha$ and the number of plays for action $a$ (the number of times the article was served), $T_a(n)$.

### 3-1-2 Evaluation and tuning

To evaluate different serving schemes, I am allowed to run tests on live traffic. This is uncommon, as news publishers are reluctant to experiment on their visitors, fearing this could have a negative impact on user experience. Besides, there are logistic constrains that generally prevent easy deployment of experimental changes to a mature codebase.

Live evaluation is relatively straight forward. First, we need a serving scheme in place, which is the algorithm that selects which article to display as top story, for each page view. After deploying a serving scheme online, all page views and clicks generated by that particular scheme are recorded. Since the reward consists of clicks, the total number of clicks in a given time frame represents the evaluation measure. To compensate for variations in traffic volume, it is common to use the click-through rate instead. To this end, the average click-through rates are computed, for certain periods of time (for instance, daily), by dividing the total number of clicks, to the total number of page views.

Two serving schemes are used to produce baselines: the first serves the last ten published articles randomly, and the second one always serves the latest published article. Evaluation is also performed against existing content optimization methods deployed at Hyves, described in Section (3-1-3).

Recall the parameter $\alpha$ from Equation 3-6, which needs to be optimized. While I am allowed to run tests on a live setting, tuning the parameter online would be infeasible, because it would simply take too much time to run sequential tests, of at least several days each, with a lot of different values for $\alpha$, to figure out the optimal value.

For optimizing $\alpha$, I resort to an offline evaluation framework introduced by (Li et al., 2010). The goal is to evaluate a serving algorithm, hereby referred to as policy $\pi$. To do this based on a stream of logged *events* - each consisting of an action (page view of an article as top story) and an observed reward (click) - an evaluator steps through the logged events and only keeps the ones for which the selected article matches $\pi$, otherwise dropping the unmatched event. The evaluator is therefore a simulation algorithm, which filters out logged events that don't match the evaluated policy and, by doing this, mimics how the policy would interact with the real world. The evaluator is described in Algorithm 3-1.

**Algorithm 3-1** *Policy_Evaluator*$(T, \pi, S)$

**Input:** $T > 0$, policy $\pi$, stream of events

$h_0 \leftarrow \phi$ {Initialize history}
$R_0 \leftarrow 0$ {Initialize total payoff}
**for** $t = 1 \rightarrow T$ **do**
  **repeat**
    $a, r_a \leftarrow get\_next\_event()$
  **until** $\pi(h_{t-1}) = a$
  $h_t \leftarrow concatenate(h_{t-1}, (a, r_a))$
  $R_t \leftarrow R_{t-1} + r_a$
**end for**
**return** $R_T/T$

Here, $T$ is the total number of trials, $\pi$ is the policy, $S$ is the sequence of logged events, $h_t$ is the sequence of matched events at trial $t$ and $get\_next\_event()$ retrieves the next logged action $a$, together with its observed reward $r_a$. The authors prove that, if the stream of actions $S$ are collected using a random policy, then any unknown click distribution $D$ can be evaluated correctly using Algorithm 3-1:

$$\Pr_{Policy\_Evaluator(T,\pi,S)}(h_T) = \Pr_{\pi,D}(h_T) \tag{3-7}$$

Offline evaluation presents itself with a different problem: it wastes a lot of data, by discarding all logged events that don't match the evaluated policy. Regardless of that policy, it is easy to see that each event is retained with probability $1/K$, where $K$ is the number of available actions. In the news case, the number of actions is the number of articles that the algorithm chooses from (recent articles), which is around 20. That means that only 1/20 of all logged user interactions are retained. In our case, there is clearly not enough data to estimate click-through rates for the articles, after discarding so many events. In the case of (Li et al., 2010), they log millions of events during a few days interval, while I was able to log no more than a few tens of thousands.

Because of the data wastefulness of offline evaluation on logged events, I resort to another technique to tune $\alpha$: I generate artificial user interactions, based on a simple model. After picking 10 random articles and computing their click-through rates based on their online performance, I generate a long list of artificial user interactions (page views, and sequential clicks). This model is a crude simplification of the live setting, as it has stationary expectations and a fixed number of actions (articles). However, I hypothesise that it is good enough to estimate $\alpha$, and I will compare the results with the ones of (Li et al., 2010) to validate this hypothesis, in Section 3-3-1.

### 3-1-3 Existing methods

Hyves is already using two content optimization methods for picking the top stories: a content-agnostic classification-based approach, and a content-based approach. At the time of my work, both methods were running in parallel, boosting similar performance. The

classification-based method was developed first and tested against a manual approach, where editors were selecting the top story, showing an improvement of 12% in click-through rate. Describing these methods in detail is beyond the scope of this work, so I provide only a high-level overview for each of them.

The classification-based approach uses user segments, based on features such as age and gender. Then, using all interactions with articles, a naive Bayes classifier determines a score for each topic/segment pair. Each visitor is presented with the most appealing topic for its segment. Interactions are predominantly clicks, but also include comments and votes given on articles. Each (user, topic) interaction pair is considered once, regardless of its type.

The content-based approach performs user modelling based on textual features. For each new article, a set of features is extracted, including the raw content, Wikipedia concepts (Meij et al., 2012) and related categories extracted from DBpedia[1]. The user modelling is performed again using all user interactions. An individual model is built for each user, containing the most frequent features she interacts with, extracted from articles based on the click history. Also, models are built for each user segment, to be used as a fall-off model for users that don't have enough interactions. The top story is selected by querying a Lucene[2] search index for most related recent stories, based on the user model.

Both these methods suffer from several limitations, mainly because they are based an all user article interactions, regardless of their context. For starters, they don't take into account the position bias. Articles which are highlighted as top stories naturally trigger more engagement from all user segments. This could lead to self-fulfilling prophecies.

A limitation that makes the evaluation against these methods troublesome is that they suffer from contagion effects, when ran in parallel with other methods. For instance, the classification-based approach scores the articles for each user segment based on all user interactions, including the ones resulting from other serving schemes.

The bandit methods I developed are designed to learn independently of other serving schemes that may run in parallel, so the data I report is unbiased. Although I will report the performance of the other two methods as well, keep in mind they may influence each other, and can be influenced by the bandits serving scheme as well.

It is important to note that these existing methods use significantly more information than the methods I'm testing, as input for learning the best stories to highlight. That's because they make use of all user interactions, including clicks on articles across the entire website. In contrast, the bandit serving schemes base their learning only on the clicks performed on the top story position, performed by users that fall within a designated test group.

Finally, for these existing methods, the editors also have the option to override the algorithm and *pin* an article to the top story position. This provides additional advantage for these methods, as the editors have insight on current events of public interest, and a bias which is impossible to control.

[1] http://dbpedia.org/About
[2] http://lucene.apache.org/core/

## 3-2 Experimental setup

In this section, I present the news website used to perform the experiments. I then describe how the data collection is performed, including the testing architecture used to isolate the experimental setup from the actual product.

### 3-2-1 Hyves Nu&Straks

At the time of this writing, Nu&Straks (English: "now and later") is the news section of the Hyves website. Founded in 2004, Hyves competes with Facebook as the social network of choice for many Dutch people. In May 2010, Hyves had around 10 million accounts, which corresponds to two thirds of the Dutch population[3].

Figure 3-1 shows how a portion of the Nu&Straks main page looks like for a regular visitor. The top story position is highlighted with a red rectangle. The serving schemes control which article is shown in this position. Logged in users are presented with a very similar interface, but the articles may differ. My experiments are performed on a subset of logged in users, which I name *the bandit cohort*. In terms of magnitude, this segment generates tens of thousands of page views and thousands of top story clicks each day.


**Figure 3-1:** Hyves Nu&Straks main page for regular visitors (top story is highlighted).

[3] http://en.wikipedia.org/wiki/Hyves

The news stories on Nu&Straks cover a wide range of aspects. It is important to note that the articles are grouped in so-called *topics*, such as "Halloween", or "Alles over Justin Bieber" (English: "All about Justin Bieber"). The presentation logic and interaction logging are performed at topic level. Based on convention, for any topic, the latest article associated with that topic is presented to the user. Therefore, these experiments are also performed at topic granularity. Topics are narrow, with many of them containing a single article. However, in certain cases, like follow-ups on the same story, multiple articles are grouped together within the same topic. Human editors publish about 30 articles each day, and create a new topic whenever they find suitable.

### 3-2-2 Architecture

In order to test various serving schemes, I need a flexible, yet robust architecture that allows easy and safe deployment of new logic to the live setting. Working directly with the Hyves codebase would be impractical, as I would have to adhere to the development and deployment procedures in use, such as having weekly deployments. Instead, I implement an external service, and perform minimal modifications to the Hyves codebase. The website is programmed to perform an API request to the external bandit service to retrieve the top story identifier. If the reply doesn't come in time, a fall-off scheme is used to pick the top story. The architecture is sketched in Figure 3-2. I intentionally leave out most of the details regarding the Hyves architecture, as it is not relevant for this work.

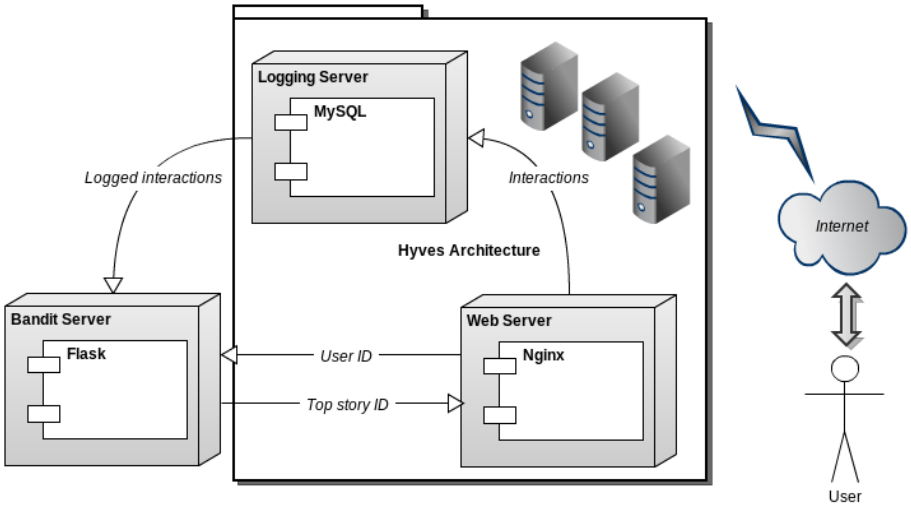


**Figure 3-2:** Simplified architecture of the experimental setup, outlining how the Hyves infrastructure interacts with the bandit server.

The website already has in place two logging schemes. However, the first one doesn't store page views (only interactions such as clicks, comments), and doesn't store the article position at click-time - there's no way to determine if the click was performed while the article was at the top story position. The second logging method uses an external reporting service, and retrieving data in real time is not possible. Therefore, an additional logging scheme is

implemented on the main website. All main page views and top story clicks (collectively named *interactions*), together with additional information such as a user identifier, are stored in a database, for the entire bandit cohort. The bandit service has real time access to this data, and uses the information to decide which topic to serve.

### 3-2-3 Data collection

Using the logged page views and clicks, the bandit service reconstructs *events*, in the sense of Algorithm 3-1. Recall these events, or plays, consist of (action, reward) pairs. In this case, the action is the selected story (therefore, all main page views are events), and the reward depends on the existence of a subsequent click. To this end, all clicks are mapped to the latest previous page view generated by the same user. This produces a list of events (page views), having either reward 1 (if a click followed) or reward 0 (if no click followed).

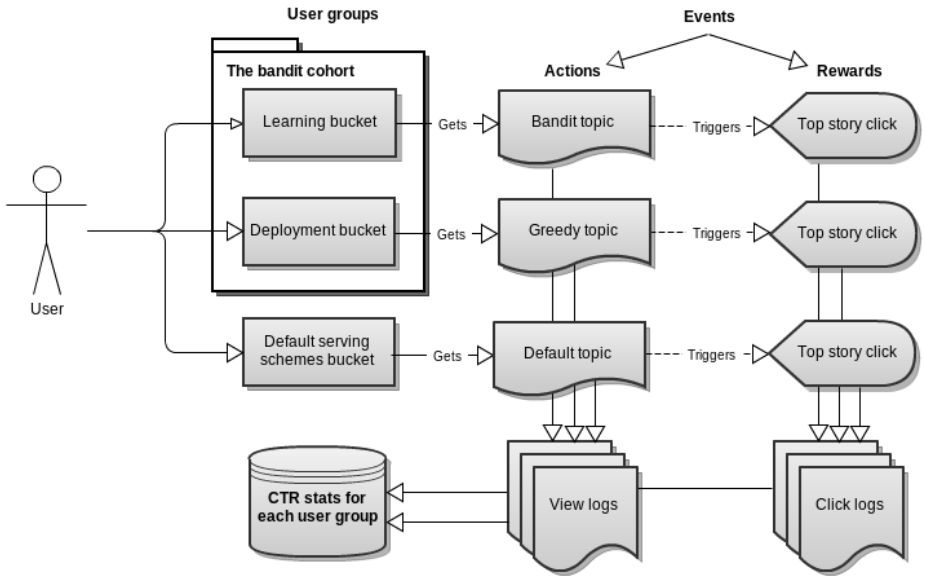


**Figure 3-3:** User flow diagram, including segmentation and data collection: a visitor sees a top story based on its assigned bucket, an event that may receive a reward if the user then clicks on that story.

For all experiments, the bandit cohort is equally divided in two buckets: the learning bucket and the deployment bucket. The user flow diagram is presented in Figure 3-3 and describes the interaction of a user with the top story position on the main page of Nu&Straks. Depending on the bucket the user falls into, she is served a different topic as top story. For the deployment bucket, the best topic is always displayed, which is recomputed every 10 minutes based on the logged events. It is called *the greedy topic*, and is computed simply as the topic with the highest click-though rate in the past 12 hours. For the $\epsilon$-greedy algorithm, the learning bucket is used for random exploration. In the case of the upper confidence bound, the UCB algorithm runs in the learning bucket.

Based on the reconstructed events, the bandit schemes can compute expectations (click-through rates) in real time, and decide which topic to serve next. These expectations are computed based on a certain time frame, which is set to the last 12 hours.

Note that, for the purpose of these experiments, I use equal-sized learning and deployment buckets. Given the available traffic volume, it is necessary to have a large-enough learning bucket to estimate click-through rates. To get a better understanding, I perform a simple analysis. Topics for the top story position are always picked from the last 10 published articles, and there are about 30 new articles published each day. Let's say we need at least 10 clicks per article to estimate it's click-through rate. Given a random exploration policy, this means that the system needs at least 300 top-story clicks per day, only for exploration. Considering a click-through rate of 0.1, this means that at least 3000 page views per day have to be dedicated to exploring article performance. This is a bare minimum, and more is usually needed to track click-through rate decay, and compensate for publishing patterns (articles are published in bursts, not being equally distributed throughout the day).

If the method where to be deployed in an environment of higher traffic, the deployment bucket could be set to be (considerably) larger than the learning bucket. This would lead to better overall results, as the exploration, which drags the click-through rate down, happens only in the learning bucket.

## 3-3 Results and analysis

In this section, I describe how tuning is performed based on artificial data, and what baselines are used to evaluate the algorithms under consideration. Then, I evaluate two bandit serving schemes, which are deployed on the Nu&Straks section of Hyves, to detect the most engaging news stories. I report on daily click-through rates and relative total click-through rate improvement, and discuss the results and their implications.

### 3-3-1 Tuning

To tune the $\alpha$ parameter from Equation 3-6, I use the method described in Section (3-1-2). I aim to have the same order of magnitude for the two terms of the summation - the click-through rate $\mu_a$, and the confidence bound $\frac{\alpha}{\sqrt{T_a(n)}}$ - so they can both play a role in the selection process. Internal traffic metrics recorded before these experiments express a range for the click-through rate $\mu_a$ and number of views per article $T_a(n)$. Based on these metrics, I chose the [0, 1.6] interval for $\alpha$ as sufficiently large to cover variations, and split it in 30 parts. For each point, 30 runs of the policy evaluator are performed on the synthetic interactions, which are shuffled before each run. At each step, the average click-through rate is computed, and a random policy is run in parallel on the same data, to get a better insight into UCB's relative performance. The results are presented in Figure 3-4.

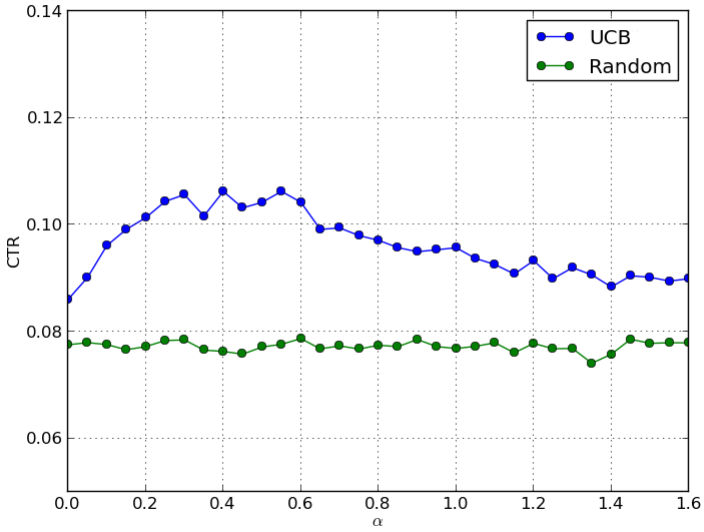


**Figure 3-4:** Tuning parameter $\alpha$ of the upper confidence bound equation.

While the click-through rate is computed over 30 runs, we can still notice some sharp variations. This may be due to very close values for the topic click-through rates. Nevertheless, we can observe a maximum at 0.4. Therefore, I will use this value for $\alpha$. While (Li et al., 2010) tune the $\alpha$ parameter using live data, their graphs have a similar shape and the same maximum, which confirms the hypothesis that the simplifications of this artificial setup provide a good-enough environment for the tuning task.

Next, I plot how the $\epsilon$-greedy and UCB algorithms perform against a random policy, and an *omniscient* policy (that knows beforehand which is the topic with the highest click-through rate and always serves that topic). Recall this setup uses a synthetic dataset with static click-through rates and a constant article pool, so this plot in Figure 3-5 serves only to get an initial impression on policy performance.

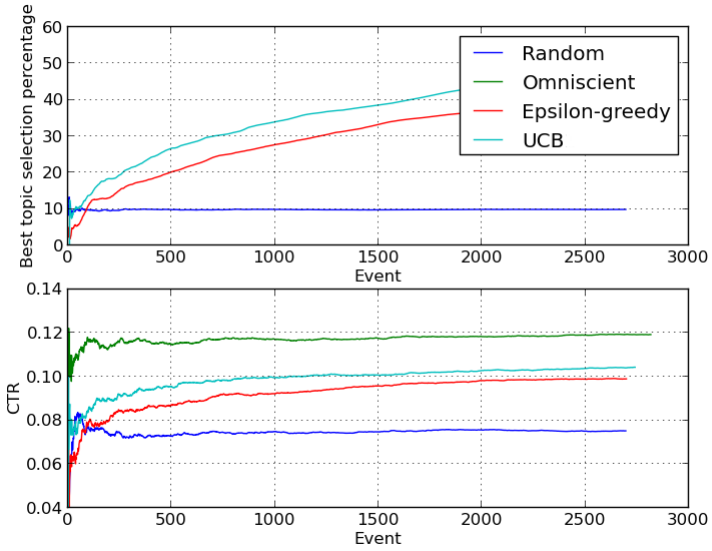


**Figure 3-5:** Policy performance comparison performed on artificial data.

As expected, UCB performs best (after *omniscient*), followed by $\epsilon$-greedy. Somewhat surprising is that the best topic is picket at most close to 50% of the time, when one would expect that an algorithm should eventually learn which is the topic with the highest click-through rate, and pick it much more often than the others. The explanation for this is the very close values between click-through rates of high-performing stories. In this case, the top two stories have click-through rates of 0.114 and 0.099, so there's a difference of 0.015. As it results from Equation 3-6, regardless of the number of interactions, UCB will continue to serve the second best-performing story almost as often as best performing one. This is particularly useful for adapting to changes in click-through rates, as they decay over time.

### 3-3-2 Baseline

A baseline is used as a starting point for comparisons in order to perform evaluation. The new serving schemes run in parallel with the existing methods used by Hyves. However, due to the issues discussed in Section (3-1-3), these existing methods do not provide reliable baselines. Nevertheless, I will report their performance as well. However, to get a better, independent baseline, I run two simple serving algorithms: a random serving scheme, and a recency-based scheme.

First, a random algorithm picks the top story from the last 10 published articles. This serving scheme is used for a user segment for 7 days. The overall daily click-through rate is plotted in Figure 3-6. The average click-through rate is 0.066.

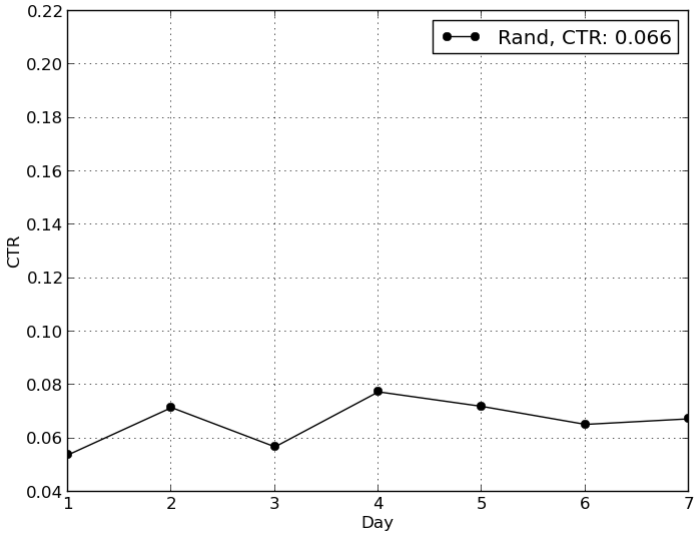


**Figure 3-6:** Click-through rate of a random serving scheme.

Next, I test the recency-based algorithm. For this, I run in parallel a random serving scheme and a recency-based serving scheme, which always serves as top story the last published article. The results are plotted in Figure 3-7.

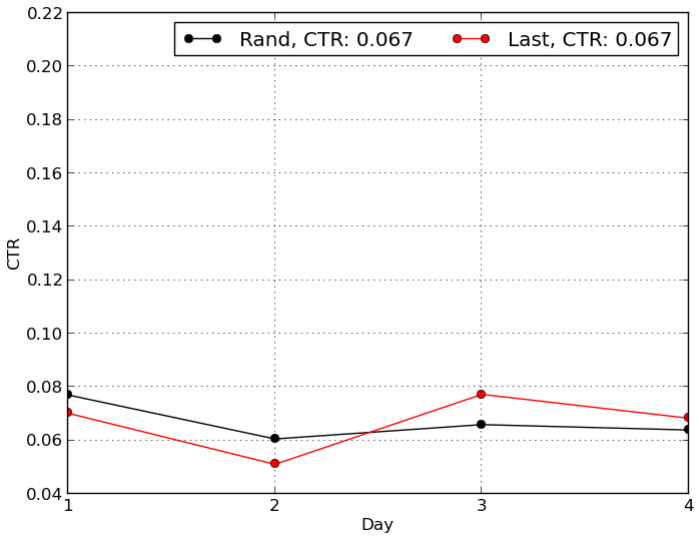


**Figure 3-7:** Click-through rate comparison of random vs. *most recent* serving schemes.

We can see that both algorithms produce the same average click-through rate: 0.067. This implies the expected click-through rate on a top story position is 0.067, when no intelligent serving scheme is used, and only the website layout and the overall article quality (subjective to the representative user demographics) is taken into account. Ideally, I would be able

to run these baseline serving schemes in parallel when testing each new method, but this is not possible due to data sparsity. Since the click-through rates are invariant to traffic fluctuations, I pick this value (0.067) as a static click-through rate baseline for all further experiments, which will be labelled as *Baseline* in all plots.

### 3-3-3 $\epsilon$-greedy and UCB

It is time to put the two serving schemes which are suitable for the news setting to the test. I start with running the $\epsilon$-greedy algorithm for 5 days. The bandits cohort is split in two buckets: the learning bucket and the deployment bucket, as described in Section (3-2-3). The click-through rates are plotted in Figure 3-8.

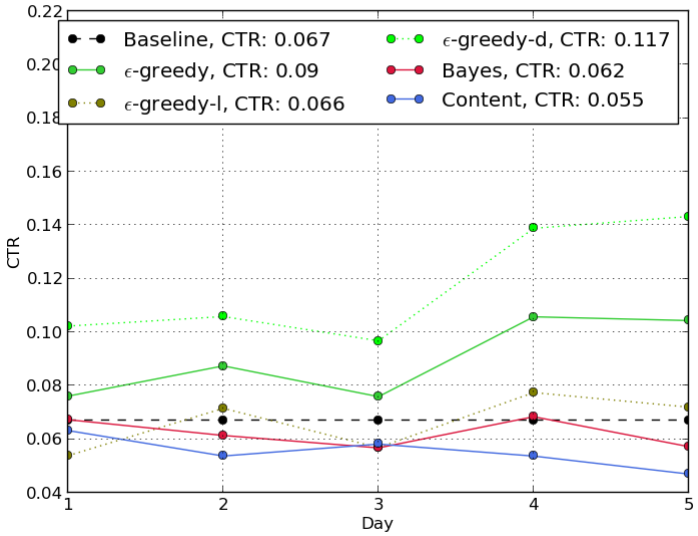


**Figure 3-8:** Click-through rate performance of the $\epsilon$-greedy algorithm, compared with the static baseline and the existing Bayes and Content methods.

Here, *$\epsilon$-greedy* refers to the overall algorithm performance, *$\epsilon$-greedy-l* tracks the performance obtained in the learning bucket, while *$\epsilon$-greedy-d* tracks the deployment bucket. As previously described, for this method, the learning bucket runs a random exploration algorithm. Therefore, as expected, the average click-through rate here matches the static baseline.

The $\epsilon$-greedy serving scheme clearly outperforms the static baseline, providing a 34% increase in click-through rate. Recall that, for more traffic, we could use a smaller learning bucket and a bigger deployment bucket (decreasing $\epsilon$), pushing the total click-through rate even higher. In that line of thinking, notice that, in the deployment bucket, the click-through rate increase reaches 74%.

For reasons beyond my control, for this time interval, the two serving schemes used by Hyves perform worse than the static baseline, making a comparison against them useless.

Next, I test the upper confidence bound (UCB) algorithm. For another five-days interval, the algorithm is run in the learning bucket, while the deployment bucket is used to always

serve the best current topic. The results are plotted in Figure 3-9.

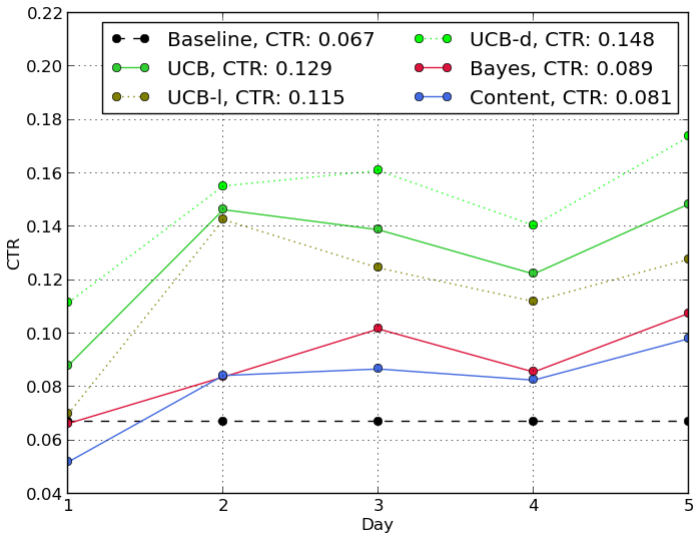


**Figure 3-9:** Click-through rate performance of the UCB algorithm,compared with the static baseline and the existing Bayes and Content methods.

Applying the UCB algorithm leads to a big improvement. First, the average click-through rate (0.129) provides a 92% lift over the static baseline, and a 45% increase over the best-performing Hyves serving scheme (segmentation using naive Bayes). As expected, the improvements are more modest in the learning bucket (72% against Rand, 29% against Bayes) than in the deployment bucket (120% against Rand, 66% against Bayes). However, the difference between the learning and deployment buckets is smaller than for the $\epsilon$-greedy algorithm, which proves the efficiency of using a smarter exploration method.

It is important to note that the values from the deployment bucket are also higher than those of the deployment bucket of the $\epsilon$-greedy algorithm. This implies that the smarter learning of UCB also leads to better, quicker detection of the best performing stories. This was to be expected. For $\epsilon$-greedy, a new story has to wait until a random serving scheme picks it for a sufficient number of times to get a click-through rate estimate. For UCB, a new story gets instant preference for exploration, since the second term in Equation 3-6, the confidence bound $\frac{\alpha}{\sqrt{T_a(n)}}$, is much higher than for older stories, which have a larger value of the served page views $T_a(n)$. This naturally leads to a quick detection of engaging stories.

# Chapter 4

# Clustering visitors and news personalization

So far I looked at detecting engaging topics for all users. However, there may be groups of users who share interests, and using this information can help further improve the performance of the bandit methods. A natural next step is to cluster users based on their click histories. I investigate which methods work best, and how similar they are with two different approaches based on surface features, such as age and gender. I then analyse how this insight can be used for news personalization, in conjunction with multi-armed bandit methods.

## 4-1 Methods

In this section, I describe how to cluster users based on their click histories, how to evaluate different clustering results, and how a segmented upper confidence bound algorithm can be used for personalized news recommendations.

### 4-1-1 Clustering

For clustering, each user $u$ is represented in the $N$-dimensional feature space $F_u$. Each feature represents a topic, and in each of the $N$ dimensions there is a binary value determining if the user interacted (clicked) with the topic or not. The goal is to group together users based on shared features.

Agglomerative hierarchical clustering is a method that starts with each point in its own cluster and builds a hierarchy by repetitively merging pairs of clusters. Three parts play a role in this process: a distance metric, the method used to decide which clusters to merge (called a linkage criterion), and a cutoff threshold which determines when to stop merging.

As a distance metric, the Jaccard coefficient is used. The Jaccard coefficient, defined in Equation 4-1, computes how similar two sets are, which is exactly what we are interested

in when comparing click histories. The clustering methods I use compute the coefficient explicitly, based on the feature representations $F_i$ and $F_j$ of users $u_i$ and $u_j$:

$$J(u_i, u_j) = \frac{|F_i \cup F_j|}{|F_i \cap F_j|} \tag{4-1}$$

I test several clustering methods: *single*, *complete*, *average* and *weighted*. These methods describe different ways to compute the distance between two clusters. The *single* method uses the distance between the closest points (nearest point algorithm), while the *complete* method uses the distance between the farthest points (farthest point algorithm). *Average* and *weighted* both use all pairs of distances between points in each cluster. *Weighted*, also known as WPGMA (Weighted Pair Group Method with Arithmetic Mean), computes the distance between clusters as a simple average of all pairs. *Average* uses UPGMA (Unweighed Pair Group Method with Arithmetic Mean), having the averages weighted by the number of elements in each cluster at each step:

$$D(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a, b) \tag{4-2}$$

Here, $d(a, b)$ refers to the distance between two points, as determined using the preferred metric. In this case, the metric used is the Jaccard coefficient.

For each data set and method, I plot dendrograms, which are tree diagrams used to visualise the arrangement of clusters. Using the dendrograms, you can see how well a clustering method works, for a specific data set. If most points appear to be outliers, and the clustering is performed by adding points one after the other in one big cluster, than the clustering has little value. However, if we observe balanced clusters growing bigger and bigger, than those clusters are potentially useful.

The result of hierarchical clustering is a linkage matrix that defines the sequence of merging clusters, up until one cluster remains. From this structure, we can form flat clusters - assign points (users) to a cluster - based on a threshold. In this case, the criterion used is the distance between points: forming flat clusters so that the original observations in each flat cluster have no greater a certain cophenetic distance. The *cophenetic distance* of two objects is a measure of how similar those two objects have to be in order to be grouped into the same cluster. The threshold is between 0 and 1 and represents the cutoff place.

### 4-1-2 Evaluation

Clustering evaluation is used to assess the quality of a clustering method. In external evaluation, the clusters are evaluated against data that was not used for performing the clustering task. In our case, we don't have a gold standard for validation. Instead, recall we want to investigate to which extent clicking behaviour can be stereotyped with gender and age segmentation. Therefore, I evaluate two different clustering methods based on surface features (age and gender) against clustering performed based on click histories.

Recall I have access to surface user features, such as age and gender. To this end, I perform two feature-based clusterings: one based on the median age, and, after gaining insight into

how click histories tend to group together, another one based on age and gender. I then test which of these two clusterings is closer to the click-based clustering.

Evaluation is performed using the Rand Index, a measure of the similarity between two data clusterings. This measure is computed based on counting, for all pairs of points, how many are in the same cluster, in both clusterings:

$$RI = \frac{TP + TN}{TP + FP + FN + TN} \tag{4-3}$$

Here, $TP$ represents the true positives, $TN$ the true negatives, $FP$ the false positives and $FN$ the false negatives.

The age-based clustering is performed using the users' median age: 14. The median age is chosen for practical reasons. Due to data sparsity, we can barely run tests for two equal-sized clusters. Any segmentation that contains less than half the number of users doesn't generate enough clicks to learn a dedicated policy online. The clustering based on age and gender splits the users in two groups: the first contains females up to 28 years old, and the second segment contains everybody else.

There are alternatives to the Rand Index. For instance, the Adjusted Rand Index is the corrected-for-chance version of the Rand Index. However, to get a clear picture on how a random clustering would perform, I prefer to explicitly compute the Rand-index of the click-history against a random clustering, where users are assigned to one of two clusters based on their numerical identifiers (if they are even or odd).

### 4-1-3 Segmented UCB

To optimize engagement on a news website, detecting which stories generate more interactions in general is a good first step. The next step is news personalization - as different users may find different stories to be more engaging than others. How to achieve this efficiently is still an open problem.

I capitalize on the insight gained from analyzing the users' click histories and attempt to use it to perform news personalization. For this, I adapt the $K$-armed bandit algorithms to the user's context, which is a set of user features. As described by (Li et al., 2010), a *contextual-bandit algorithm* $A$ proceeds in discrete trials $t = 1, 2, 3...$, such that, at each step:

1. The algorithm observes user $u_t$, a set of actions (articles) $A_t$, and the feature vector $x_{t,a}$ called the *context*, which summarises information about the user $u_t$, and possibly the action $a$.
2. Based on the rewards from previous trials, it chooses an arm $a_t \in A_t$ and receives a reward of $r_{t,a_t}$.
3. $A$ improves its strategy with the new observation $(x_{t,a_t}, a_t, r_{t,a_t})$.

Apart from adding the user context to the decision logic, a contextual-bandit algorithm has the same behaviour as the $K$-armed bandit algorithms, with the same $n$-trial regret as defined in Equation 3-5.

The upper confidence bound algorithm was shown to have a logarithmic bound on regret and is suitable for the news article setting, as described in Section (3-1-1). For the contextual bandit approach, I plan to run this algorithm in parallel for different user groups, or segments. The resulting approach is called a *segmented upper confidence bound* algorithm (segmented UCB). Segmented UCB runs a copy of UCB in each user segment. If the segments truly represent users with different interests, this leads to different policies, and different engaging stories detected for each group. However, this also divides the available learning data to the number of segments used, thus making the learning process potentially slower or less accurate. These limitations are discussed in more detail in Section (4-2-3).

The first experiment based on segmented UCB uses two balanced clusters, where users are grouped based on their median age (14). A copy of the UCB algorithm is ran for each group. After gaining more insight into how users behave based on how their click histories are clustered, another segmentation is performed, based on both age and gender. Because of the constrain of having two equally sized clusters, the first group will contain females up to 28 years old, and the second segment will contain everybody else.

## 4-2 Experimental setup

In this section, I give an overview of the user demographics of the Nu&Straks section of Hyves, I describe how the user clustering is performed, and present the limitations associated with the segmented upper confidence bound approach.

### 4-2-1 User demographics

Nu&Straks news are geared toward a younger audience. In Figure 4-1, you can see the user demographics for this website section, as extracted from a random sample of recent visitors. Females dominate males at a ratio of roughly 2:1 overall, and, as the figure shows, the ratio is visibly higher in the 10 to 15 years old age group. The average age is 22, while the median age is 14.

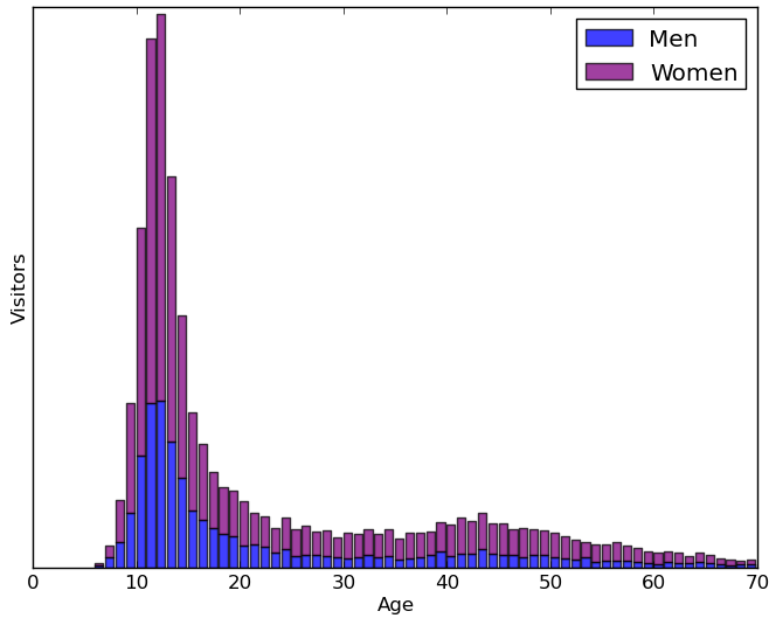


**Figure 4-1:** Age and gender distribution of the Nu&Straks users.

This skewed distribution makes it hard to construct useful user segments, since most users already fall within a small age group (preteen and teenage girls). As we will see in Section (4-2-3), we are limited to build at most two user groups, which have to be balanced. Therefore, one group will naturally contain a lot of young girls, and the other will contain the long tail of other users, which vary a lot in age, and so presumably their interests vary too.

### 4-2-2 Active users and click history overview

First, I select a set of user candidates. The goal is to pick users that are active, from the bandits cohort. For this, all users that fall within the bandits cohort, and that have at least one click on the top story during a certain interval are considered. I extract a sets of candidates from a 4 day period (between October 12 and October 16, 2012), resulting in *userset-4-days*.

For the click histories, I initially consider clicks on top stories within a one week period, between October 17 and October 24, 2012. However, there are too few clicks per user (on the top story) in this set. Based on the bandits cohort, the long-tailed click distribution over this one week period is plotted in Figure 4-2. Even active users don't click more than a few times per week on the top position. Therefore, I have to make use of a different click log.

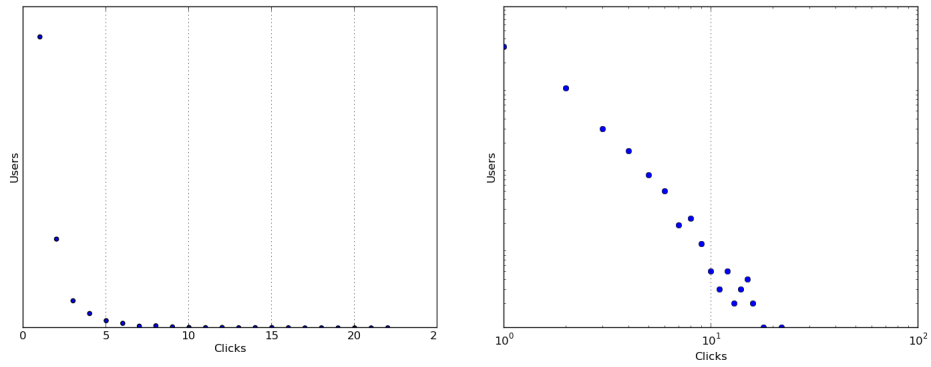


**Figure 4-2:** The distribution of the number of clicks per user on the top story position, over a one week period, as scatterplot (left) and in log-log scale (right).

For Nu&Straks, Hyves stores all interactions performed by all users on topics, regardless on the topic's position at click time. An interaction may consist of a click, a comment or a follow action on a topic. They are stored as tuples, containing the user id, the topic id, the interaction type and the date/time. Interactions with the same topic are merged and considered once, regardless of the type of interaction, or the number of interactions a user had with it (e.g. repeated clicks on the same topic).

Using this comprehensive interaction log, I consider all clicks performed in the previous 3 months before October 19, 2012, and prune users that clicked less than 3 times in this interval. To build the final data set, based on *userset-4-days*, an additional condition is set: I consider only users with at least 7 interactions in the past 3 months.

The *userset-4-days* set contains 2,673 distinct users (that clicked at least once in the 4 days interval). For the click history, there are 2,051 (out of 2,673) users that pass the 7 clicks minimal history threshold. These users generated a total of 126,703 interactions during the three months interval, bringing the average to 61.77 interactions per user, which is roughly 20 interactions per active user, per month.

Next, features are selected for clustering the active users. A feature consists of a topic identifier. The numerical identifiers of all topics that have at least two interactions are considered as feature candidates, totalling 1,433 features.

### 4-2-3 Limitations

A segmented upper confidence bound algorithm uses predefined user groups, or segments, and learns different policies for each group. This implies keeping track of rewards on a per-group basis. If the groups indeed reflect different user preferences, then the resulting policies will differ, and the overall performance can be improved. However, note that such an approach also needs proportionally more data for learning. For instance, if we split the users in two equal groups, the segmented upper confidence bound algorithm needs double the amount of page views and clicks to achieve the same confidence levels as it's non-contextual counterpart. This turns out to be a major drawback.

Based on the available volumes of traffic, I can at most split the users in two balanced clusters, and run a learning algorithm for each of these two user segments. The algorithms will have half the amount of data, in terms of page views and clicks, available for learning. Anything less than that would simply be too little to make use of.

Ideally, we would like to split the users in segments based on the groups that result after performing click-based clustering, and run a learning algorithm for each such group. Unfortunately, due to the data sparsity described above, this is not feasible. Therefore, I have to make use of the insight gained from click-based clustering, to make a decision on what user segments (based on surface features) would work better, and then test the validity of this decision.

## 4-3 Results and analysis

The goal is to investigate ways to achieve personalization for top story selection, while increasing performance. First, I run a segmented UCB algorithm based on median age, and test how it affects the overall click-through rate. Next, the focus shifts to user clustering based on click histories, and how these clusters evaluate against segmentation based on surface features. Finally, a segmented UCB algorithm is tested on a new user segmentation, which is based on the insight gained from user clustering.

### 4-3-1 Segmented UCB based on median age

Before diving into user clustering, I ran a first experiment where users are grouped based on their median age (14), and a copy of the UCB algorithm selects what story to show within each group. The results are presented in Figure 4-3.

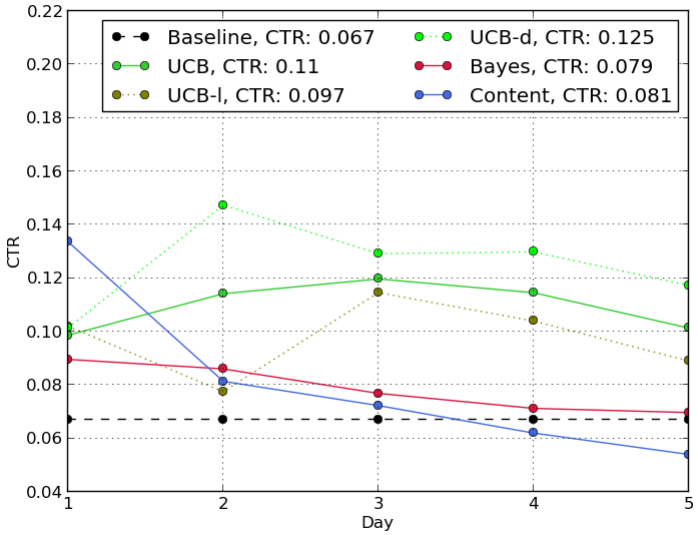


**Figure 4-3:** Segmented UCB based on age

This segmented UCB approach achieves an overall click-through rate of 0.11, which represents an 64% improvement over the baseline and a 35% improvement over the best performing method ran in the same period (*Content*). However, recall that running an unsegmented UCB algorithm achieved an expected click-through rate of 0.129 (giving a 92% lift over the static baseline). This means that this segmentation approach fails to improve over the unsegmented variant, and actually performs worse. I discuss possible reasons for this later on in Section 4-3-3, after performing this experiment again using a different user segmentation.

### 4-3-2 Clustering

I first plot dendrograms for the tested linkage methods, as it can be seen in Figure 4-4. For readability, the dendrograms are plotted only for distances between 0.5 and 1 (the y axis), starting with the 20 biggest clusters, in terms of number of elements. On the X axis are the actual clusters, where the number in brackets refers to the number of clusters associated to that line (lines with no number represent a single cluster).

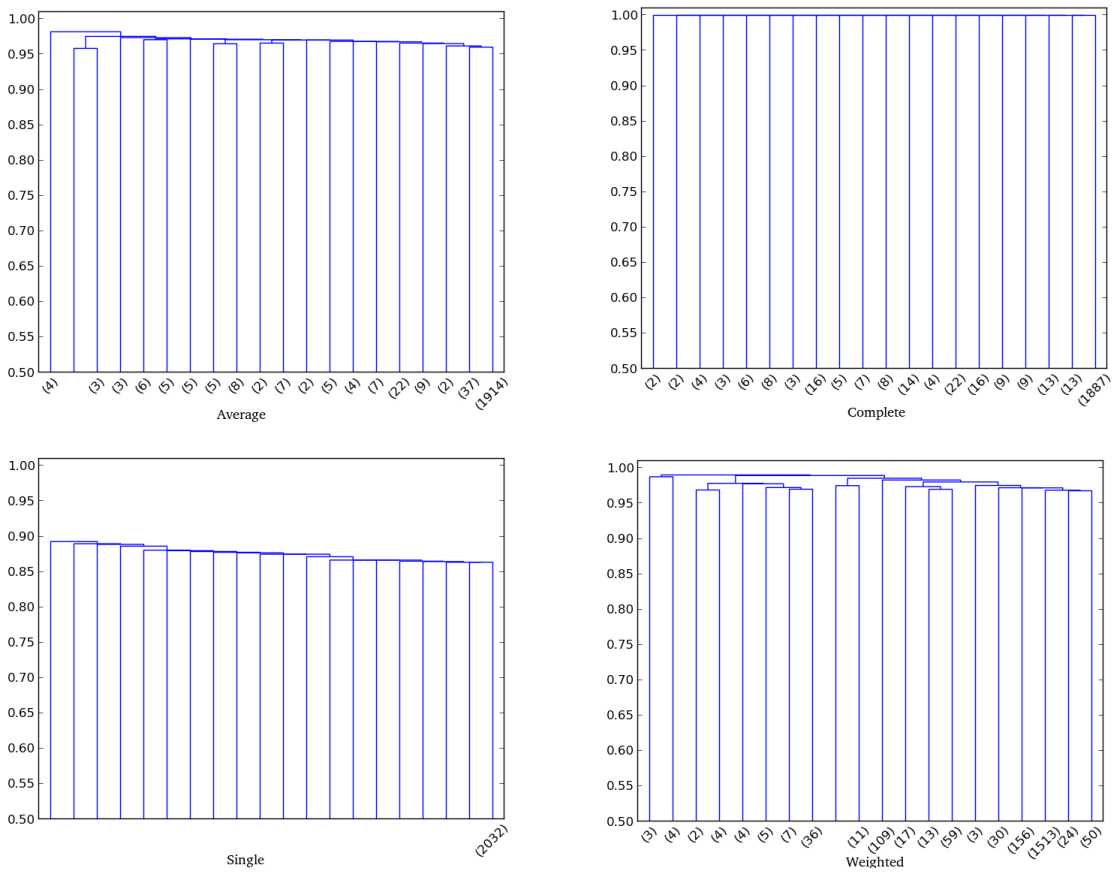


**Figure 4-4:** Dendrograms for different linkage methods.

As expected, the *average* and *weighted* linkage methods perform best, with a clear, reasonably balanced hierarchy emerging in the dendrograms. The other two methods, *complete* and

*single* linkage, which group clusters based on the distance between only two points (nearest or farthest), perform poorly. For *single* linkage, the typical pattern of a single mega-cluster emerges, while for *complete*, the method fails to cluster the biggest clusters together.

Next, I focus on the most promising linkage methods: *average* and *weighted*. For these, I build flat clusters using various cutoff thresholds. The higher the cutoff threshold, the bigger the clusters and the smaller their number. However, as clusters get bigger, the similarity between their members also decreases.

To get a better understanding on how different cutoff thresholds influence the clusters formation, I plot all users from the four biggest clusters. Each cluster is color-coded. Recall that I have access to user surface features. Therefore, I plot each point in an age-gender graph. This way, we can investigate to which extent the biggest clusters map to gender and age segments. To get a better insight, I compute the average age ($Age$) and gender ratios ($F/M$) for each of the four biggest clusters, and display the number of elements in each of these clusters ($N$). For the *average* method, the plots are presented in Figure 4-5.

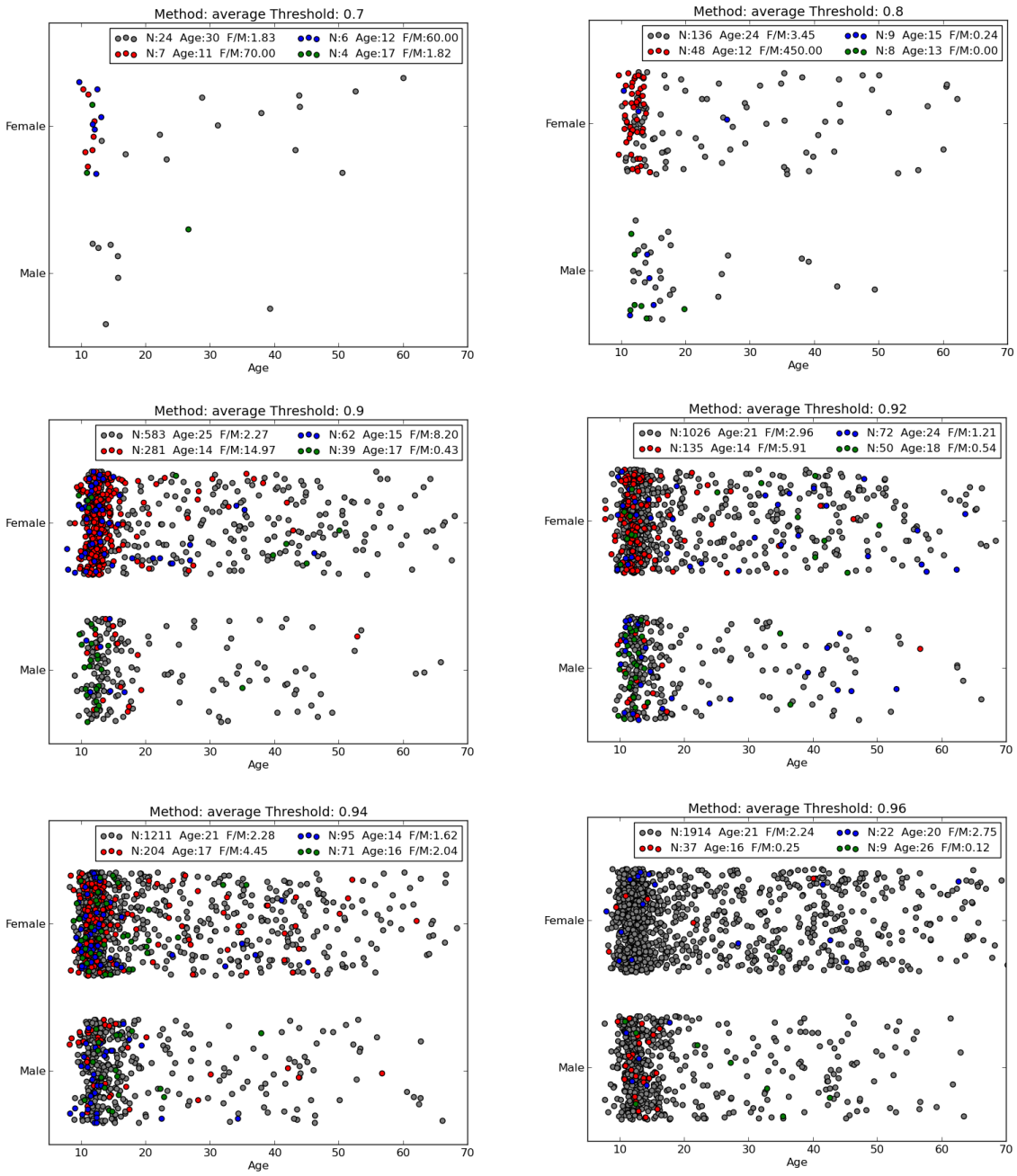


**Figure 4-5:** Distribution of users over age and gender in the four biggest clusters, computed using average linkage, at different cutoff thresholds.

Here, we can see how the top clusters get bigger as the cutoff threshold increases. Eventually, one mega-cluster eats away the other clusters (see plot for the threshold 0.96). For values 0.9 and 0.92, there's a background cluster that holds users representative of the overall website demographics (average ages 25 and 21, gender ratios 2.27 and 2.96), and then there is a distinct cluster of young females (average age 14, gender ratio 14.97 and 5.91). This

suggests that these young females have indeed distinct interests in news articles, that differ somewhat from the background demographics. However, note that the website already has a skewed audience of young females, as is was shown in Section 4-2-1.

Next, I present the clustering plots for the *weighted* linkage method (Figure 4-6).

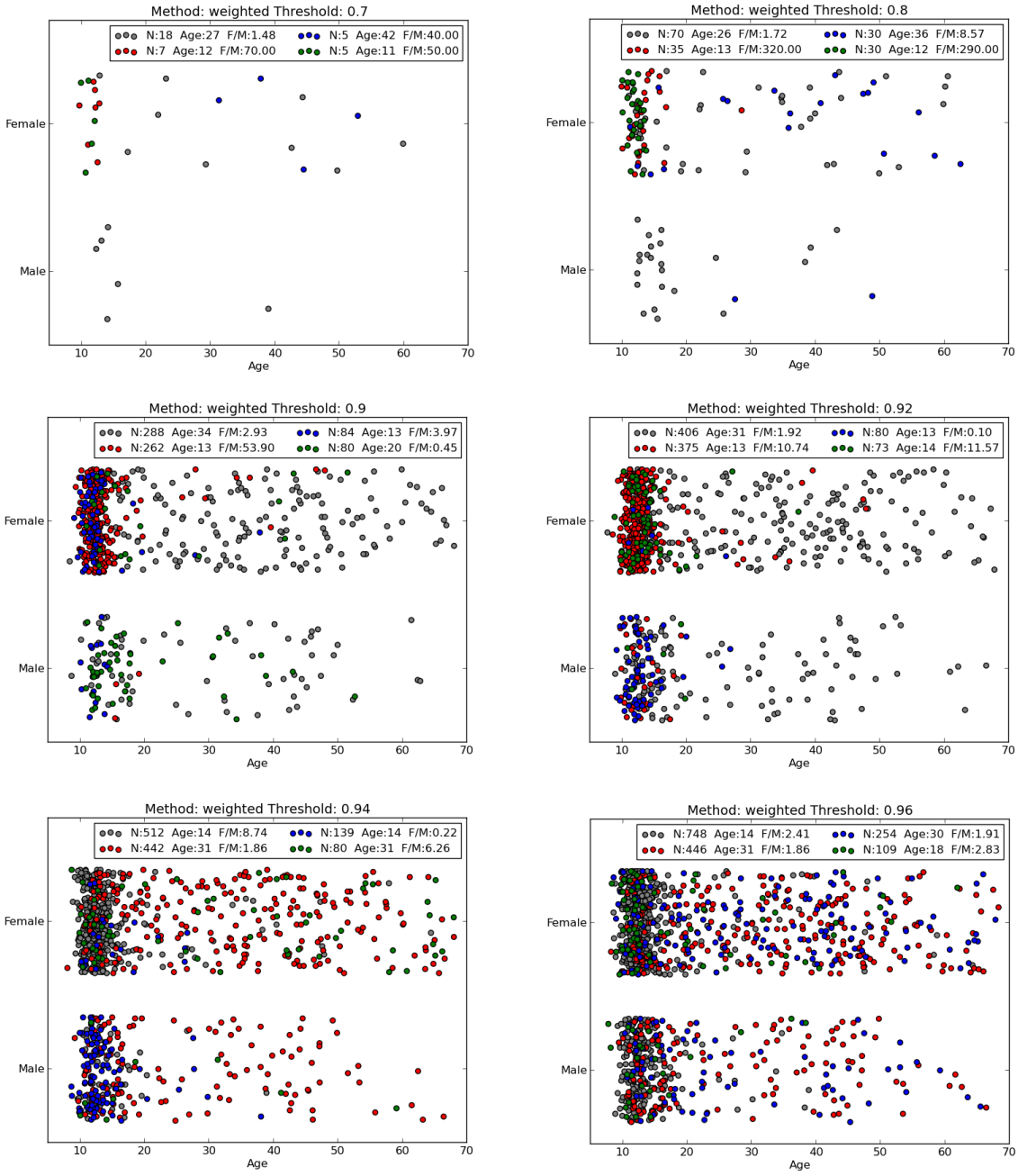


**Figure 4-6:** Distribution of users over age and gender in the four biggest clusters, computed using weighted linkage, at different cutoff thresholds.

**Table 4-1:** Clustering evaluation results, showing the Rand Index values at different cutoff thresholds (T), which result in different number of clusters (N) and of users in the top four biggest clusters (U).

| | | | Random vs. click-based | | Age-based vs. click-based | |
|---|---|---|---|---|---|---|
| T | N | U | All clusters | Top 4 | All clusters | Top 4 |
| *Average* | | | | | | |
| 0.7 | 1938 | 41 | 0.484 | 0.490 | 0.464 | 0.484 |
| 0.8 | 1298 | 201 | 0.484 | 0.499 | 0.484 | **0.539** |
| 0.9 | 235 | 965 | 0.487 | 0.497 | 0.474 | 0.507 |
| 0.92 | 128 | 1283 | 0.492 | 0.499 | 0.482 | 0.501 |
| 0.94 | 53 | 1581 | 0.497 | 0.501 | 0.487 | 0.493 |
| 0.96 | 19 | 1982 | 0.511 | 0.512 | 0.526 | 0.529 |
| *Weighted* | | | | | | |
| 0.7 | 1938 | 35 | 0.484 | 0.497 | 0.464 | **0.598** |
| 0.8 | 1298 | 165 | 0.484 | 0.496 | 0.464 | **0.552** |
| 0.9 | 270 | 714 | 0.486 | 0.498 | 0.482 | **0.610** |
| 0.92 | 156 | 934 | 0.487 | 0.498 | 0.493 | **0.590** |
| 0.94 | 73 | 1173 | 0.490 | 0.501 | 0.506 | **0.583** |
| 0.96 | 29 | 1557 | 0.494 | 0.500 | 0.541 | **0.603** |

There is a similar trend here as for the *average* method. In the begging, the cutoff threshold is too small to form reasonable sized clusters. As the threshold increases, the clusters become bigger, and more balanced than for the *average* linkage. For weights 0.9 and higher, we can clearly see that the top two clusters represent a background cluster, and a young females cluster. The higher contrast is achieved at threshold 0.92, where where the young females cluster has a gender ratio of 10:1, and an average age of 13. Moreover, here there is also a distinctive young boys cluster (albeit smaller), with a gender ratio of 1:10 and an average age of 13.

These plots suggest that, even for such skewed background demographics, there is a distinct group of young females that generate a distinct clicking patterns than the background group. This is encouraging, as it gives weight to the hypothesis that gender and age play a significant role in click patterns.

Next, I set to investigate which of these clusterings are most similar to an age-based clustering. For this, I perform clustering evaluation using the Rand Index. I use two different clustering approaches, one by grouping users at random in two clusters, and the other by grouping users based on their median age, in two clusters. Then, I evaluate the results of these two methods against the click-based clustering result obtained at the previous step. These results are displayed in Table 4-1.

Here, $T$ refers to the threshold, $N$ to the number of clusters, and $U$ to the number of

users in the top four clusters. The number express Rand Index values.

When evaluation is performed for all users, the age-based clustering doesn't improve over the random baseline, for none of the linkage method/cutoff combinations. This may come as a surprise, but all click-based clustering methods produce a much higher number of clusters than 2 (the smallest is 19). Therefore, users are spread across many clusters.

If we limit ourselves to evaluating only the users that fall within the top four biggest clusters (for the click history based approach), results do get better. In accordance with the conclusions drawn from Figure 4-6, the *weighted* linkage method with higher cutoff thresholds (0.92, 0.94, 0.96) performs best.

The somewhat unexpected low correlation with age based segmentation has multiple causes. First, if we look closely at the plots in Figure 4-5 and Figure 4-6, we see that the background cluster contains a comparable number of elements in the same segment as the young females cluster. Therefore, there is a lot of overlap between clusters in terms of age and gender distribution. Secondly, as previously mentioned, these methods produce many clusters, and only merge the last 20 or so at very high thresholds, suggesting a high level of data sparsity (many users sit in their own cluster until late in the process due to a distinctive click history).

What we gain from this cluster analysis is the insight that a large subgroup from the young female audience has distinctive interests patterns. Next, I will use this insight to test if a segmented UCB approach performs better using a two segments approach, where the first segment consists of young females.

### 4-3-3 Segmented UCB based on clusters insight

The cluster analysis suggests the most distinctive user segment comprise of young females. Given this insight, and keeping in mind the limitations described in Section (4-2-3), a segmented UCB algorithm is run for a new set of two balanced user segments: the first group contains females up to 28 years old, and the second segment contains everybody else. The results are plotted in Figure 4-7.

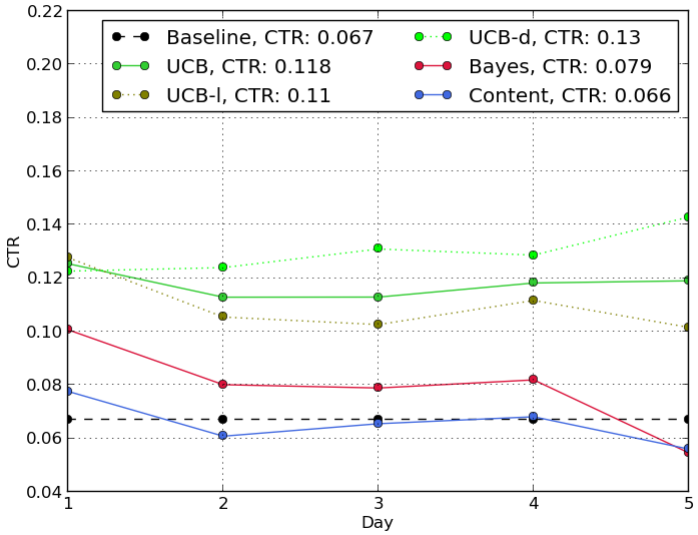


**Figure 4-7:** Segmented UCB based on age and gender

For these new segments, UCB achieves an average click-through rate of 0.118, improving with 76% over the static baseline, and with 49% over the best alternative method running in the same interval (*Bayes*). This represents a slightly better performance than the age-based segmentation, but it still underperforms when comparing with the unsegmented UCB.

The fact that this new segmentation still fails to improve over the unsegmented UCB doesn't come as a surprise. As it results from the cluster analysis, there is a lot of age and gender overlap between the click-based clusters. The Rand Index clustering evaluation measure shows that, when considering all clusters, the age-based clustering doesn't improve over the random baseline, when comparing with the click-based clustering. The data is also sparse, with many users having short click histories, which many times don't match well with others, and the website has a pre-existing skewed audience of young females, as described in Section 4-2-1.

Another reason why segmented UCB doesn't improve results is the balanced clusters limitation. Because I can construct at most two balanced clusters, the first group has to contain females up to 28 years old, while the cluster analysis outlines a much younger female group with distinctive preferences. If I were to make a group of females of up to 16 years old, the clusters would become unbalanced, and there wouldn't be enough data to learn in the smaller segment.

The reasoning above only explains the risk of not getting a performance boost when doing segmented UCB. To explain why segmentation actually hurts performance, recall the segmented UCB algorithm runs a copy of UCB for each user segment. This means the amount of data needed to achieve the same learning rate multiplies with the number of segments, assuming they are equal. Even when having just two segments, cutting the learning rate in half is a big burden for the overall learning performance.

Besides taking more time to learn if a new article has top story potential, not having enough clicks can also lead the algorithm to fail detecting good stories entirely. As the average

number of available page views per article decreases, especially during publishing hours, some articles may end up getting very limited exposure, that would not trigger a sufficient number of clicks to detect their potential. This explains why performance decreases not only in the learning bucket, but also in the deployment bucket.

# Chapter 5

# Detecting engaging stories in a mobile setting

In this chapter, I apply the user engagement optimization methods described in Chapter 3 to a mobile setting. The main goal is to investigate if their performance holds in this different setting, as the information structure and screen size can significantly affect the navigation behaviour and perceptions of mobile internet users (Chae and Kim, 2004). While doing this, I also build a platform than allows easy integration of the upper confidence bound serving algorithm on virtually any web application, including news websites, through an application programming interface (API).

## 5-1 Methods

The serving scheme used is the upper confidence bound algorithm described in Section 3-1-1. Recall that UCB computes a score for each available article by summing the current click-through rate estimate with the value of a confidence bound, based on Equation 3-6, and always serves the article with the highest score. Parameter $\alpha$ is used to control the ratio of exploration versus exploitation: the greater $\alpha$ is, the higher is the weight of the confidence bound when computing the article score, and so articles are explored more while their number of views is still low.

I put an emphasis on building a plug-and-play platform that can be easily be connected to any web application. For this, I identify the main algorithmic components, and implement them as part of an API. While the architectural details are presented in Section 5-2, I describe here the logical structure of the algorithm.

There are two main components needed to implement the UCB logic, which are implemented as separate API entry points. The first one, described in Algorithm 5-1, decides which article to serve (in this case, to display it in the top story position), based on the available articles, specified by their numerical identifiers. The second component is straight-forward,

and only has to increment the number of clicks stored for an article, when a user visits it coming from the top story position.

**Algorithm 5-1** *Get_Headline*$(ids, alpha)$

**Input:** array of article identifiers *ids*, parameter *alpha*
  *play_aid* $\leftarrow 0$ {Initialize the article identifier to be served}
  *max_ucb* $\leftarrow 0$ {Initialize the maximum upper confidence bound value}
  **for** *aid* $\in$ *ids* **do**
    *a_views* $\leftarrow$ *get_views*(*aid*)
    *a_clicks* $\leftarrow$ *get_clicks*(*aid*)
    **if** *a_views* $> 0$ **then**
      *ctr* $\leftarrow$ *a_clicks*/*a_views*
      *ucb* $\leftarrow$ *ctr* $+$ *alpha*/$\sqrt{a\_views}$
      **if** *ucb* $>$ *max_ucb* **then**
        *play_aid* $\leftarrow$ *aid*
        *max_ucb* $\leftarrow$ *ucb*
      **end if**
    **else**
      *play_aid* $\leftarrow$ *aid* {If the article doesn't have any views, display it at least once}
      break
    **end if**
  **end for**
  *increment_views*(*play_aid*, 1)
  **return** *play_aid*

Based on the numerical identifiers of the available articles, the algorithm proceeds to iterate through these identifiers, and computes for each article its current click-through rate and upper confidence bound score. Unless there is an article that has no views at all (in which case, that article is served to initialize its score), the article with the highest UCB value is returned. Also, the number of views associated with the returned article is incremented.

## 5-2 Experimental setup

In this section, I first describe the mobile website used to perform the experiments. Then, I present the testing platform, including the architecture of the web service used to integrate the upper confidence bound serving scheme with the mobile website.

### 5-2-1 Telegraaf Privé

De Telegraaf ("The Telegraph") is the largest Dutch daily morning newspaper, and its website, telegraaf.nl, is one of the top news websites in the country.[1] A prominent section of the website features content supplied by the gossip-magazine Privé ("Private").

[1] http://en.wikipedia.org/wiki/De_Telegraaf

The telegraaf.nl website has a mobile version designed for smartphones with high-resolution touchscreens. This version of the website has the same structure as the main website, and, as such, it also contains the Privé section. I run experiments within this section of the mobile version of telegraaf.nl, by reordering articles and measuring their performance in the top story position, in term of click-through rate. The layout of this section is presented in Figure 5-1.

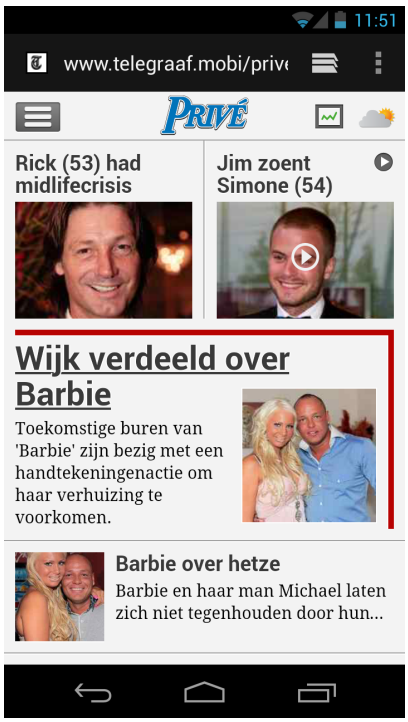


**Figure 5-1:** The Privé section of the telegraaf.nl mobile website. The top story has the title "Wijk verdeeld over Barbie".

Here, the top story position is highlighted with thick red lines, and features the biggest font size for the title. The goal is to maximize the click-through rate on this position. To achieve this, I devise a method to control which story is placed here, from the pool of available articles.

An important aspect is that the editorial team has access to powerful tools that track the performance of articles in real time, across all positions. This allows them to make well informed decisions on what articles to highlight. They can learn which articles perform well from other positions, and only then place them as top stories. This gives a significant advantage to the editorial team over our method, which uses the same position for both learning how all articles perform, and exploiting the ones that perform best.

### 5-2-2 Architecture

The architecture of the serving algorithm is designed as a web service, called here *the bandits API*. This application is hosted independently of the tested news website, in a cloud-

based *platform as a service* (PaaS) named AppFog.[2] The API has two main entry points: one used to retrieve the article to serve based on a sequence of available article identifiers (implements Algorithm 5-1), and the second one is used to increment the number of clicks on an article.

Plugging the bandits API to the Telegraaf Privé mobile website has to be done with minimal tampering of the website's codebase. I use a client-based solution, which doesn't interfere with the back-end logic of the application. Once the main page is loaded, a Javascript function exploits the Document Object Model (DOM) structure of the page to extract a list of all available article identifiers, and performs an AJAX call to the Bandits API. The API runs Algorithm 5-1 and replies with the identifier of the article to place as top story. Finally, using DOM manipulation, a switch is performed between the article in the top story position, and the one specified by the Bandits API. This process should happen fast enough so the user doesn't notice the switch - if it takes too long, a timeout event is triggered and logged, which prevents the article switch. The action flow for testing the serving scheme, together with the architecture of the service, are presented in Figure 5-2.

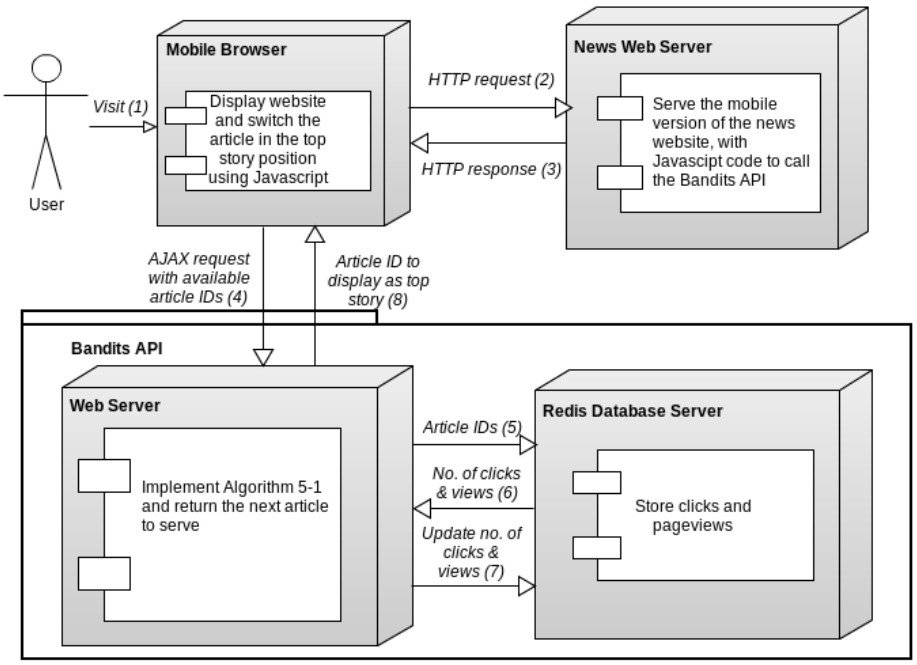


**Figure 5-2:** The architecture of the bandits API, together with the information flow between the involved components, when a user visits the main page; the numbers in brackets reveal the order of interactions.

Here, the distinction between the website architecture and the bandits API is made clear. On one hand, there are separate components for the mobile browser running on the user's phone and the server architecture responsible for hosting the news website (here, presented as just one component: the news web server). Then, there is the bandits API, consisting of a web server and a database server, which uses the Redis in-memory key-value store. The

[2] https://www.appfog.com/

communication is performed on the client-side, using Javascript code which runs in the mobile browser and performs asynchronous requests to the bandits API.

Only the interaction with the main page is presented in Figure 5-2, for retrieving the identifier of the top story. The call to the second API entry point happens over the same architecture, and in a similar way: when an article is visited as a result of a click on the top story position, the browser makes a request to the bandits API specifying the identifier of the clicked article, which leads to incrementing to stored number of clicks for that day and for one of the two user groups, as described in the following section.

### 5-2-3 Data collection

The data collection is performed after splitting the visitors in two groups, by assigning them a random number, and storing it as a cookie on their computer. If this number is even, then the top story is switched based on the UCB serving algorithm. If the number is odd, the original top story, as it is set by the editorial team, is left unchanged. In both cases, a parameter is added to the top story position link url, so that the article visit can be identified as coming from this position.

Page views and clicks are counted for each user group for each day, so I can compute daily click-through rates, and compare the results. The experiment is carried out during a 7 day period, between January 24 and January 30, 2013. To produce a static baseline, I use a similar approach to the one described in Section 3-3-2: I run a random serving scheme for two days (January 31 and February 1, 2013), and compute the average click-through rate for this period.

## 5-3 Results and analysis

The random serving scheme used to select the top stories for two consecutive days generates click-through rates of 0.073 and 0.069 respectively. I consider the average of 0.071 as a static baseline. It is interesting to note that this value is slightly higher, but close to the static baseline used in the Nu&Straks experiments (0.067). This can be interpreted as a first indication that the mobile setup doesn't have a substantial impact on the click-through rates of the top story position. The small increase may be caused by the lack of alternative links on the smaller mobile layout, but this would require further research to attest.

The click-through rates achieved by the two methods are presented in Figure 5-3. Despite the editors having access to state of the art tools for monitoring article performance, and having the advantage of being able to use other position to learn which articles perform best before placing them as top stories, the upper confidence bound algorithm still outperforms the manual approach by 20%. It is important to note that the algorithm does so while using the same position for both exploiting high-performing articles, but also exploring the potential of all new articles, as they become available. The increase over the static baseline is a whopping 108%.

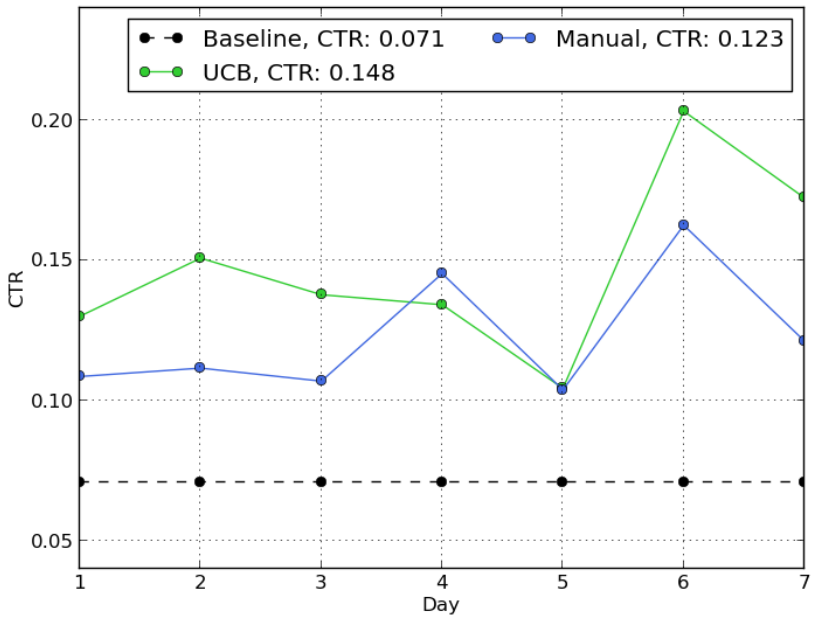


**Figure 5-3:** Click-through rates on the top story position, when controlled by the editors (Manual) and the upper confidence bound algorithm (UCB).

While the bandit method clearly outperforms the manual approach in 5 out of the 7 tested days, in one of the other two remaining days the performance is reversed, and in the other one they produce similar click-through rates. Before providing possible explanations for this behaviour, I test if the overall lift in click-through rate is indeed statistically significant.

I use an unpaired t-test to compare the average click-through rates of the two methods. For this, based on the experimental results, I compute the values in Table 5-1.

**Table 5-1:** Measures used for computing statistical significance.

| Group | Sample size | Sample mean | Sample standard deviation |
|---|---|---|---|
| Manual (1) | 7 | 0.123 | 0.020 |
| UCB (2) | 7 | 0.148 | 0.029 |

Since the variances have very close values, an unpaired t-test is indeed applicable.[3] The sample sizes for the two groups are noted $n_1$ and $n_2$, the sample means $\bar{x}_1$ and $\bar{x}_2$, and the samples standard deviations $s_1$ and $s_2$. Then, in order to compute the T-statistic (Equation 5-1), I need to compute the pooled standard deviation $s_p$ (Equation 5-3) and the standard error $SE(\bar{x}_1 - \bar{x}_2)$ (Equation 5-2).

$$T = \frac{\bar{x}_1 - \bar{x}_2}{SE(\bar{x}_1 - \bar{x}_2)} \tag{5-1}$$

[3] http://mlsc.lboro.ac.uk/resources/statistics/Unpairedttest.pdf

$$SE(\bar{x}_1 - \bar{x}_2) = s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}} \tag{5-2}$$

$$s_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}} \tag{5-3}$$

After performing the calculations, the value of the T-statistic is 1.879. Under the null-hypothesis, the statistic follows a t-distribution with 12 ($n_1 + n_2 - 2$) degrees of freedom. Using tables of the t-distribution I determine that the p-value $p < 0.05$. Therefore, the probability of the improvement in click-through rate happening by chance is less than 5%, so there is strong evidence that the UCB algorithm performs better than the editorial team in generating a higher click-through rate on the top story position.

Several reasons could explain why, during day 4, the manual approach achieved a higher click-through rate. First, if the editorial team constantly monitors the performance of articles and always places as top story only the best performing ones (based on other positions), than this approach is likely to beat the upper confidence bound algorithm in its current incarnation, because the algorithm also has to explore all new articles in the same position. It could also be the case that a high-performing article is missed by the algorithm. For instance, if the article doesn't perform well initially, its score remains low, and it may not be sufficiently explored later on, to detect a possible change in performance. Therefore, if for whatever reason, a poorly performing article becomes highly attractive some time after its publication, the algorithm may not detect the change.

# Chapter 6

# Conclusions and future work

In this work, I investigated the potential of content-agnostic, online reinforcement learning methods to increase user engagement on medium-sized news websites. The upper confidence bound algorithm achieved best results for this task, boosting a 92% lift in click-through rate against a *most recent article* serving scheme, and a 45% increase over the best performing existing method used on the tested website. These high lifts in click-through rate validate the potential of such methods to significantly benefit not only top-tier portals, such as Yahoo!, but also medium-sized news websites.

Generating more clicks on the top story position is the primary, easily measurable effect of using such methods. But second order effects, such as improving user satisfaction and return rates, can prove even more beneficial, by increasing the number of total visits, and consequently producing even more clicks on the entire website. More visits and more clicks usually also means more revenue, depending on the company's business model. These secondary effects are harder to measure and can be an interesting avenue for future work: there are many factors that play a role in user satisfaction and return rates, and it can be very hard to isolate them.

Testing different methods for clustering users based on their click histories allowed me to observe a segment of young females with distinctive topic preferences. However, the demographics of the tested website were already heavily skewed towards a young, female population, making the potential gains of a segmented learning approach inapplicable, especially under limited traffic conditions. This comes as an interesting lesson, highlighting the costs and limitations of news personalization. However, testing other hybrid approaches besides this segmented learning algorithm, or experimenting on a website with a more general audience, could eventually yield improvements over the unsegmented approach, making it another potentially interesting research route.

The upper confidence bound algorithm proves to be medium-invariant, as it significantly outperforms an informed manual approach when tested on a mobile news website. The 20% lift in click-through rate, when comparing to the manual approach, while being high, is lower than in the case of the desktop website tested previously. On the other hand, when compared

to the static baseline of a random serving scheme, the lift is higher in the mobile setting. Several reasons may contribute to this difference, such as different layouts (on the mobile website, there are fewer distractions towards other links), and the fact that the editorial team of the mobile website has access to advanced performance monitoring tools, thus making better informed decisions on article placements.

Besides investigating second order effects of increasing the top story performance or testing new hybrid approaches, this work could inspire a wide range of other research directions. One is to apply similar methods to completely different products which share similar traits, such as online advertising platforms. Increasing click-through rates there by quickly detecting and displaying the best performing advertisements would have an immediate impact on revenues, and presumably would have a positive impact on user experience also.

This work focused on optimizing the performance of one story position, which is used for both exploring new articles and exploiting the good performing ones. Another direction would be to investigate how these methods generalize to more positions, if one has access to control the content of several placements on a page. For instance, I can imagine a scenario where a less prominent position is used for learning, while the top story position is only used to exploit the best articles. Such a scenario could be more acceptable from an editorial point of view, but, to maintain optimality, the serving algorithms would probably need to be adjusted.

In a world where online user interaction data is becoming more easily available, I truly believe there is still a huge untapped potential in using and learning from this information to improve existing products, or even design and build entirely new ones. My work strives to contribute to this endeavour, and I hope it will encourage others to apply content-agnostic, online reinforcement learning methods for solving engagement optimization problems in a variety of settings, as well as foster more research in this area.

# Bibliography


Agarwal, D., Chen, B.-C., and Elango, P. (2009). Explore/exploit schemes for web content optimization. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 1–10, Washington, DC, USA. IEEE Computer Society.

Agarwal, D., Chen, B. C., Elango, P., Motgi, N., Park, S. T., Ramakrishnan, R., Roy, S., and Zachariah, J. (2008). Online Models for Content Optimization. In Koller, D., Schuurmans, D., Bengio, Y., Bottou, L., Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *NIPS*, pages 17–24. MIT Press.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

Carreira, R., Crato, J. M., Gonçalves, D., and Jorge, J. A. (2004). Evaluating adaptive user profiles for news classification. In *Proceedings of the 9th international conference on Intelligent user interfaces*, IUI '04, pages 206–212, New York, NY, USA. ACM.

Chae, M. and Kim, J. (2004). Do size and structure matter to mobile users? An empirical study of the effects of screen size, information structure, and task complexity on user activities with standard web phones. *Behaviour & Information Technology*, 23(3):165–181.

Das, A. S., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA. ACM.

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., and Sampath, D. (2010). The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 293–296, New York, NY, USA. ACM.

Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. (1999). Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Hofmann, K., Whiteson, S., and de Rijke, M. (2013). Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval Journal*.

Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*, 22(1):89–115.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA. ACM.

Koren, Y. (2010). Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22.

Langford, J., Strehl, A., and Wortman, J. (2008). Exploration scavenging. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 528–535, New York, NY, USA. ACM.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 661–670, New York, NY, USA. ACM.

Li, L., Chu, W., Langford, J., and Wang, X. (2011). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 297–306, New York, NY, USA. ACM.

Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

Liu, J., Dolan, P., and Pedersen, E. R. (2010). Personalized news recommendation based on click behavior. In *Proceedings of the 15th international conference on Intelligent user interfaces*, IUI '10, pages 31–40, New York, NY, USA. ACM.

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

Meij, E., Weerkamp, W., and de Rijke, M. (2012). Adding semantics to microblog posts. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 563–572, New York, NY, USA. ACM.

Minsky, M. (1954). *Theory of neural-analog reinforcement systems and its application to the brain-model problem*. Princeton University.

Radlinski, F. and Joachims, T. (2007). Active exploration for learning rankings from click-through data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 570–579, New York, NY, USA. ACM.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA. ACM.

Strehl, A. L., Langford, J., Li, L., and Kakade, S. (2010). Learning from logged implicit exploration data. In *NIPS*, pages 2217–2225.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Vinh, N. X., Epps, J., and Bailey, J. (2009). Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1073–1080, New York, NY, USA. ACM.