Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media

# Efficient Filtering on Hidden Document Streams

**Eduardo Ruiz** and **Vagelis Hristidis**
University of California-Riverside
{eruiz009,vagelis}@cs.ucr.edu

**Panagiotis G. Ipeirotis**
New York University
panos@stern.nyu.edu

**Abstract**

Many online services like Twitter and GNIP offer streaming programming interfaces that allow real-time information filtering based on keyword or other conditions. However, all these services specify strict access constraints, or charge a cost based on the usage. We refer to such streams as "hidden streams" to draw a parallel to the well-studied hidden Web, which similarly restricts access to the contents of a database through a querying interface. At the same time, the users' interest is often captured by complex classification models that, implicitly or explicitly, specify hundreds of keyword-based rules, along with the rules' accuracies.

In this paper, we study how to best utilize a constrained streaming access interface to maximize the number of retrieved relevant items, with respect to a classifier, expressed as a set of rules. We consider two problem variants. The static version assumes that the popularity of the keywords is known and constant across time. The dynamic version lifts this assumption, and can be viewed as an exploration-vs.-exploitation problem. We show that both problems are NP-hard, and propose exact and bounded approximation algorithms for various settings, including various access constraint types. We experimentally evaluate our algorithms on real Twitter data.

## 1 Introduction

Popular social sites such as Twitter, Tumblr, reddit, wordpress, Google+, YouTube, etc. have tenths of millions of new entries every day – e.g., Twitter has 400 million and Tumblr has 72 million new entries every day – making the process of locating relevant information challenging. To overcome this information overload, there are well-known solutions in the area of information filtering. An information filtering system separates relevant and irrelevant documents, to obtain the documents that a user is interested in. Generally, during a filtering process, a classifier categorizes each document and assigns a score that can be used to rank the final results. We focus on such filtering models.

Ideally, the content provider stores the user profiles and filters each document to decide if it should be returned to a user. However, most current providers of streaming data, such as Twitter, do not provide complex filtering interfaces, such as user-defined classifiers. Instead, they allow specifying a *filtering profile* that consists of sets of keywords (*filters*) that return matching documents; currently the limit is 400 such keyword filters. We refer to such streams with strict access constraints as "hidden streams" to draw a parallel to the hidden Web research (Raghavan and Garcia-Molina 2000), indicating that the stream is only available through querying. If a document matches any of the filters on the profile it is returned to the user. The interface often imposes additional rate constraints, capping the maximum amount of documents that can be retrieved through any given filter. We refer to this type of constrained filtering interface as *keyword-based hidden stream interface*.

The simple but limited keyword-based hidden stream interface poses unique challenges on how to best map a complex document classifier to a filtering profile. This is the main problem studied in this paper. In particular, the user of the keyword-based hidden stream interface must set filters that return as many relevant documents as possible, while at the same time respecting the access constraints of the interface. Two typical access constraints are (i) a constraint on the number $K$ of the keyword filters and (ii) a constraint on the rate $R$ of returned documents. For example, for Twitter, we have $K = 400$ and $R \approx 1\%$ of total tweets' rate in Twitter.

In this work, our goal is to provide classifier-based document filtering capabilities on top of a keyword-based hidden stream interface on hidden streams. The properties of relevant documents that we want to filter are described by a rule-based classifier, which consists of a set of keyword-based rules along with corresponding accuracies. For instance, a politics classifier may consist of hundreds of rules like "if keywords 'health' and 'policy' are contained then relevant with precision 0.8." Many other types of classifiers like decision trees (Quinlan 1987) and linear classifiers (Towell and Shavlik 1993; Fung, Sandilya, and Rao 2005) can be easily converted to sets of rules; hence we believe that our modeling is quite general.

Note that the key difference of our problem to the well-studied document filtering problem (Belkin and Croft 1992) is that in the latter the filtering algorithm has access to the whole stream, and decides which documents are of interest. In our case, we do not have immediate access to the documents, and we are accessing them through a "hidden document stream" using a limited querying interface, and try to decide where to allocate our capped querying resources.

The problem has the following key challenges: (i) The number of rules in the classifier may be higher than the number $K$ of filters, or may return more than $R$ documents per time period. (ii) There is overlap on the documents returned by rules (filters), e.g., rules "Obama" and "Washington" may return many common documents. Note that many interfaces like Twitter only allow conjunctive queries without negation, which makes overlap a more severe challenge. (iii) The data in streams changes over time as different keywords become more or less popular, due to various trends. Given that hidden streams only offer access to a small subset of their documents, how can we identify such changes and adapt our filters accordingly?

This paper makes the following contributions:

1. We introduce and formally define the novel problem of filtering on hidden streams.
2. We introduce the *Static Hidden Stream Filtering* problem, which assumes that the popularity of the keywords is known and constant across time (or more formally, at least in the next time window). We show that this problem is NP-Complete, and propose various bounded error approximate algorithms, for all problem variants, with respect to access constraints and rules' overlap.
3. We introduce the *Dynamic Hidden Stream Filtering* problem, which assumes keywords distributions are changing over time in the hidden stream. This problem can be viewed as an exploration-vs.-exploitation problem. We present algorithms to select the best set of filters at any time, by balancing exploitation – using effective rules according to current knowledge – and exploration – checking if the effectiveness of other rules increases with time.
4. We experimentally evaluate our algorithms on real Twitter data.

## 2 Problem Definition

**Keyword-based Hidden Stream Interface:** A *document stream* $S$ is a list of documents $d_1, d_2, \cdots, d_t$ published by the content provider. Each document has a timestamp that denotes its creation/posting time. The user has access to the document stream through a *Keyword-based Hidden Stream Interface*, which is a filtering service maintained by the content provider that allows users to access the data. The user specifies a *filter set* $\Phi$ of keyword-based *filters*, $\Phi = \{F_1, F_2, \cdots, F_n\}$. Each filter $F_i$ is a list of terms $w_1, w_2, \cdots, w_q$. The filter set can change over time; we use $\Phi_t$ to denote the filter set used at time $t$. Given a set of filters, the service filters and returns the documents that *match* any of the filters. To simplify the presentation, we assume in the rest of the paper that a filter $F_i$ matches a document if the document contains all the keywords in the filter (conjunctive semantics), which is the semantics used by popular services like Twitter. Other semantics are also possible.

**Example:** *Assume that we are interested in documents related to job offers on Twitter. Some possible filters to extract documents are:* $F_1$= {*job*}, $F_2$={*health*}, $F_3$={*legal, assistant*}.□

The service establishes a usage contract that consists of a set of *access constraints*. We study in detail two important and popular types of constraints (both used in Twitter): (i) the *filters number constraint* specifies a maximum number $K$ of filters that may be active at any time, that is, $|\Phi| \leq K$; (ii) the *rate constraint* $R$ specifies the maximum number of documents per time unit that the service may return to the user across all filters. If the filters return more than $R$ documents during a time unit, we assume the service arbitrarily decides which documents to discard. We refer to this as *interface overflow*.

Table 1: Filter Usefulness for the Jobs Domain.

| Filter | Precision | Coverage | Filter | Precision | Coverage |
|---|---|---|---|---|---|
| jobs | 0.71 | 4680 | engineer | 0.93 | 945 |
| manager | 0.97 | 835 | health | 0.83 | 205 |
| legalAssist. | 0.84 | 65 | nurse | 0.94 | 570 |
| rn | 0.96 | 140 | | | |

Note that the rate constraint is necessary to make the problem interesting. Otherwise we can create very general filters and obtain all the documents in the stream. However, even in this case accessing the full stream can be very costly on network/storage resources for both the user and the provider.

Alternative cost models are possible. For instance, one may have an access budget $B$ and pay a fixed cost for each returned document or for each filter (less practical since a general filter may return too many documents). We could also include a cost overhead for each deployed filter plus a fixed cost per document. Our algorithms focus on the above ($K$,$R$) constraints, but most can be adapted for such cost models. For instance, a budget $B$ on the number of retrieved documents for a time period $t$ is equivalent to the $R$ rate constraint when $R = B/W$ (assuming the rate is constrained in time windows of length $W$ or wider).

**Filtering Classifier:** So far we have described the capabilities of the interface to the hidden stream. Now, we describe the modeling of the useful documents for the user, that is, the documents that the user wants to retrieve from the hidden stream. We define *usefulness* as a function that assigns to each document a label in $\{True, False\}$ . If the document is useful for the user the usefulness will be *True*, else *False*.

The most common way to estimate the usefulness of a document is using a document classifier (Sebastiani 2002). The most common features in document classifiers, which we also adopt here, are the terms (keywords) of the document. Further, we focus on rule-based classifiers, which are popular for document classification (Provost 1999). Another advantage of using rule-based classifiers is that several other types of classifiers can be mapped to rule-based classifiers. (Han 2005)

Formally, the rule-based classifier is a set of rules $\Psi = \{L_1, \cdots, L_m\}$, where each rule $L_i$ has the form

$$w_1 \wedge w_2 \wedge \cdots \wedge w_q \xrightarrow{p} True$$

This rule says that if a document $d$ contains all terms $w_1, w_2, \cdots, w_q$, then $d$ is useful. The ratio of useful documents is the *precision* of the filter (probability) $p(F_i)$. The number of total documents that are returned is the coverage $c(F_i)$. The number $u(F_i)$ of useful documents of filter $F_i$ is $u(F_i) = c(F_i) \cdot p(F_i)$. The set of all filters that are created

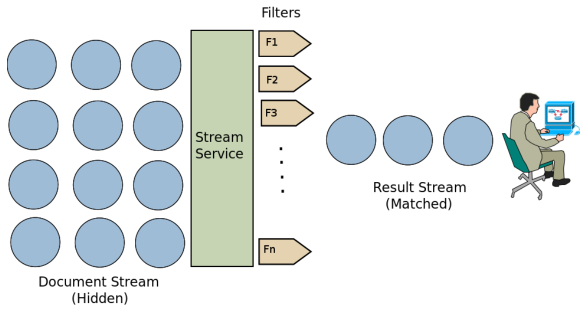



Figure 1: Filter Selection

Table 2: Useful Documents

| Document | Useful |
|---|---|
| Brownsville Jobs: Peds OT job: Soliant Health #Jobs | T |
| Army Health care Job: EMERGENCY PHYSICIAN #amedd#jobs | T |
| RN-Operating Room-Baylor Health Care - Staffing #Nursing #Jobs | T |
| #5 Health Benefits Of Latex You Should Know #article 57954 | F |
| Heel spurs causing foot pain #health #wellness #pain #aging #senior | F |
| Signed up for Community Change in Public Health #communitychange. | F |

in this way is our set of *candidate filters* $\Omega$.

**Optimization Problems** The idea is maximize the number of useful documents returned by the Hidden Stream. Clearly, not all filters in the set of candidate filters can be used in the filter set, since $|\Phi|$ may exceed $K$ or their combined documents rate may exceed $R$. Further, selecting the $K$ rules with highest precision may not be the best solution either, since these rules are generally too specific and may have very small *coverage* (number of matched documents in the document stream during a time window).

Figure 1 shows the various components of the framework. The full document stream is only visible to the service provider. This differentiates our problem from the classic information filtering problem. The service only returns documents that are matched by the filters. Table 2 shows an example of documents that match the filter $F_2$=$\{health\}$, which is specified for a classifier related to job offers. Some of the documents are useful as they are related to our domain. Others are related with generic health content but not to jobs.

**Filters overlap:** A key challenge is that filters have overlap among each other, that is, the same document may be returned by multiple filters.

Let $c(\Phi)$ and $u(\Phi)$ be the numbers of unique and unique useful documents that match any of the filters in $\Phi$, respectively. Finally, $p(\Phi) = u(\Phi)/c(\Phi)$. Knowing the n-way overlap for any set of n filters in $\Omega$ is intractable due to the exponential number of such sets and the dynamic nature of the stream.

Hence, in this paper we only consider pairwise (2-way) overlaps between filters. We define $u(F_i, F_j)$ and $c(F_i, F_j)$ as the numbers of useful and matched documents shared between filters $F_i$ and $F_j$, respectively. Hence, we assume that all n-way overlaps, $n > 2$, are zero, in the inclusion-exclusion formula, which leads to:

$$u(\Phi) \approx \sum_{F_i \in \Phi} u(F_i) - \sum_{F_i \in \Phi} \sum_{F_j \in \Phi} u(F_i, F_j) \qquad (1)$$

Clearly, keeping higher levels of overlap may improve the accuracy of filter selection algorithms, but by their nature classifier rules have little overlap, which makes this limitation less important, as shown in Section 5.

The pairwise overlaps can be represented in an **overlap graph**, shown in Figure 2, where each filter is a node and there is an edge if the filters have a non-empty overlap ($c(F_i, F_j) > 0$).

**Hidden Stream Filtering Problem:** Our objective is to maximize the number of useful documents in the filtered stream without violating the usage contract, which consists of access constraints $K$ and $R$. We refer to the pair $(K, R)$ as our *budget*. In our problem setting, *the precisions of the filters are known* from a separate process, which we consider to be well-calibrated to report accurate precision numbers. For instance, we may periodically use crowdsourcing to estimate the precision of each classifier on a training set of documents. To simplify the presentation, we view the precision of each filter as fixed, but our methods can also apply to a changing but known precision.

The first problem variant assumes that for each filter $F$ in the set of candidate filters we know its coverage $c(F)$ and $c(F)$ is fixed (or almost fixed) across time windows. In practice, we could estimate the coverages based on an initial sample of the documents. It is reasonable to assume that the coverage is fixed for topics with small or slow time variability like food-related discussions, but it is clearly unreasonable for fast changing topics like sports events or news (Dries and Ruckert 2009).

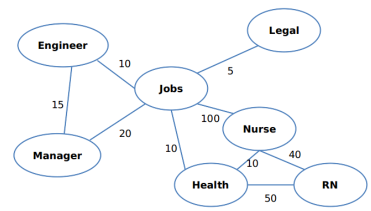



Figure 2: Overlap Graph

**Problem 1 (Static Hidden Stream Filtering). SHSF** *Given a keyword-based hidden stream interface with access restrictions* $(K, R)$*, a set* $\Omega$ *of candidate filters with known usefulness, coverage and pairwise overlap values, select the filters set* $\Phi \subseteq \Omega$ *such that the number of expected useful documents is maximized, while not violating restrictions* $(K, R)$*.* □

Given that our main goal is to maximize the number of useful documents returned by the selected filters, we want to check that the total estimated filters coverage is close to the maximum rate $R$; if it is lower then we have unused budget, and if it is higher then the service will arbitrarily drop documents, possibly from the high precision filters which is undesirable.

**Example** *Table 1 shows a set of candidate filters with their coverage and precision values. Suppose we are given constraints* $K = 2$ *and* $R = 5000$ *for the time window corresponding to the coverage numbers in Table 1. For the sake of simplicity let's assume that the filters are independent, that is, there are no documents retrieved by multiple filters.*

Table 3: Notation.

| Notation | Details |
|---|---|
| $\Omega$ | Candidate filters |
| $m$ | number of candidate filters, $m = \|\Omega\|$ |
| $F_i$ | A filter |
| $\Phi$ | Selected Filters ($\Phi \subseteq \Omega$) |
| $n$ | Number of selected filters ($n = \|\Phi\|$) |
| $K$ | Constraint on number of filters in $\Phi$ |
| $R$ | Rate constraint |
| $u(\Phi)$ | Usefulness of $\Phi$, i.e. number of useful documents |
| $c(\Phi)$ | Coverage of $\Phi$, i.e. number of matched documents |
| $p(\Phi)$ | Precision of $\Phi$. $p(\Phi) = u(\Phi)/c(\Phi)$) |
| $c(F_i, F_j)$ | Overlap coverage for $F_i, F_j$ |
| $u(F_i, F_j)$ | Overlap usefulness for $F_i, F_j$ |

*Then the best solution would be the filter set* $\Phi$=$\{F_1$=$\{$*jobs*$\}$, $F_2$=$\{$*health*$\}\}$*, which is expected to return* $0.71 \cdot 4680 + 0.83 \cdot 205 = 3493$ *useful documents.* □

As mentioned above, the assumption of the static problem variant that the coverage of filters is known and does not change with time is unreasonable is many settings. Periodically sampling the streaming service to recompute the coverage estimates would take up some of our access budget, which we generally prefer to use for the subset of selected filters $\Phi$. To address this challenging setting, we define a dynamic variant of the problem, where in addition to selecting the filters that will maximize the expected number of useful documents, we must also continuously estimate the current coverage of "promising" filters in the set of candidate filters.

This can be modeled as a exploration vs. exploitation problem (Kaelbling, Littman, and Moore 1996) where we must both use the filters that are expected to give the most useful documents given the access constraints, but we must also learn the coverages of other promising filters, which may be used in future time windows.

**Problem 2 (Dynamic Hidden Stream Filtering). DHSF** *Given a keyword-based hidden stream interface with access restrictions* $(K, R)$*, and a set* $\Omega$ *of candidate filters with known precisions, but unknown and changing coverages and pairwise overlaps, select filters set* $\Phi \subseteq \Omega$ *for the next time window such that the number of expected useful documents is maximized, while not violating restrictions* $(K, R)$. □

Table 3 summarizes the notation.

# 3 Static Hidden Stream Filtering Problem

In this section we present a suite of algorithms for the Static Hidden Stream Filtering (SHSF) problem. We discuss three variants of this problem, based on which of the K and R budget constraints are specified: SHSF-K, SHSF-R, and SHSF-KR, which have K, R or both K and R, respectively. We first analyze their complexity and then present efficient algorithms to solve them. Given that all problem variants are intractable we focus on heuristic algorithms and approximation algorithms with provable error bounds.

Table 4: Proposed Algorithms for SHSF Problem.

| Algorithm | K, R? | Error | Complexity |
|---|---|---|---|
| GREEDY-COV/PREC | R,KR | Unbounded | $O(\|\Omega\|K)$ |
| GREEDY-COV | K | $1-e$ | $O(\|\Omega\|K)$ |
| TreeFPTA | R,KR | $1-\epsilon$ | $O(K^6 logm)$ |

## 3.1 Complexity Analysis

We first show that all versions are strongly NP-Complete even if we know the usefulness/coverage and pairwise overlaps at any point in time. In other words, with complete information of the future, the selection problem is intractable.

**Theorem 1.** *The* SHSF-K, SHSF-R and SHSF-KR *problems are Strongly NP-Complete.*

Proof: We first show that *SHSF-K* is Strongly NP-Complete. We reduce to Maximum Independent Set (MIS). For each vertex in MIS we create a filter $F_i$ with profit $u_i = 1$. For each edge we create an edge with overlap 1. If the MIS graph has an independent set of size $K$ then $SHSF-K$ will find a solution of usefulness $K$ using exactly $K$ filters. For *SHSF-R*, we just set $R = K$ and use the same MIS reduction. Finally, *SHSF-KR* is harder than both *SHSF-K* and *SHSF-R* and it can use the proof of any of the two. □

The strong NP-Completeness means that there is not even a pseudo-polynomial approximate algorithm for these problems. If we assume that there are *no filter overlaps* then the $R$ (and *KR*) problem variant can be mapped to the 0-1 knapsack problem, mapping each element to a filter. As a direct consequence, we can then use the knapsack pseudo-polynomial algorithms to solve *SHSF-R* and *SHSF-KR* (Hochbaum 1996). We also propose a novel pseudo-polynomial algorithm when only the strongest filter overlaps are considered; see TreeFPTA below.

## 3.2 SHSF Algorithms

We show that the considered budget constraints directly affect the difficulty of the problem as well as the proposed solutions. A summary of the proposed algorithms is presented in Table 4. The table shows the complexity and approximation bounds achieved by the different algorithms we propose to solve all the variants of the *SHSF* problem, where we note that in the first two algorithms the error is with respect to all pairwise overlaps, whereas in TreeFPTA the error is with respect to the most important overlaps as explained below.

**GREEDY-COV:** Given the candidate filters $\Omega$ we evaluate the usefulness of each rule. Then our algorithm greedily adds the filter $F_i$ that has the maximum residual contribution considering the selected filters $\Xi$ (*GREEDY-COV*). The residual contribution is calculated as the usefulness of the filter considered $u(F_i)$, minus all the overlaps $c(F_i, F_*)$ with the filters that are already in the solution. This process continues until we violate any of the constraints K or R. *GREEDY-COV* is a $1-e$ approximation for the *SHFS-K* problem. The proof is based on the Maximum Set Coverage Greedy approximation (Hochbaum 1996).

**GREEDY-PREC** instead greedily selects filters with maximum residual precision. Residual precision is the precision

$p(F_i)$ of filter $F_i$ if there is no overlap among filters, whereas if there is overlap, we compute the new precision of $F_i$ considering the residual precision and coverage with respect to the selected filters $\Xi$. **GREEDY-PREC** has no error bound for any problem variant.

The intuition why the greedy algorithms have generally unbounded error is as follows. In the case of *GREEDY-COV*, the algorithm is blind to the filter effectiveness. If $R$ is a critical constraint picking low precision elements can waste the budget very quickly. On the other hand, *GREEDY-PREC* could be wasteful when $K$ is the critical constraint. Commonly, a rule with high precision can be expected to be highly specific, bringing a small number of useful documents. Selecting highly selective rules will squander the rule budget.

**TreeFPTA** Here we relax one of the complexity dimensions of the problem, namely we only consider an overlaps tree instead of an overlaps graph. We first show an exact pseudo-polynomial dynamic programming algorithm for the *SHFS-R* problem. However, this solution is infeasible, and for that we propose a polynomial-time approximation scheme (FPTAS) that is $1-\epsilon$ approximate. Finally, we extend the algorithm to handle both $K$ and $R$.

**Pseudopolynomial dynamic programming algorithm**: This algorithm considers the most important pair-wise overlaps, in a style similar to how the Chow-Liu factorization (Chow and Liu 1968) is used to model pair-wise dependencies, in order to approximate more complex joint distributions. The Chow-Liu factorization considers that each node (filter in our case) is only correlated with a single other node, its parent, where we choose as parent a highly correlated node. Then, the inference process is tractable. We use a similar idea to reduce the effect of multi-way dependencies, but in our case each node overlaps not only with its parent, but also with its children.

Given the **overlap graph** we set the weight of each edge to be the number of useful elements that are in the intersection. Then we compute a maximum spanning tree on this graph. The intuition is that we minimize the information loss keeping the most important overlaps. We assume an empty filter is the root, making the **overlap tree** connected.

**Example of overlap tree:** Consider the filters overlap graph in Figure 2. The tree in Figure 3 presents the maximum spanning tree for the same graph adding an extra empty filter to make the tree connected. Now, the overlap effects are bound to parent-child relations. For example, if we pick both "RN" and "health" we need to account for the effect of the overlap of both. On the other hand, "jobs" and "health" are assumed to have no overlap.$\square$

After the tree transformation, the interaction effects are local. This means the residual precision or coverage for any filter only is affected by its parent and children. This local effect allows to isolate the optimal solution of any sub-tree with root $\beta$, as the only element that can affect it is the parent $parent(\beta)$ of its root. For that, the dynamic programming algorithm must compute for each sub-tree two solutions: the optimal solution assuming $parent(\beta)$ is selected and not selected. Further, for each subtree we must store the optimal solution assuming a usefulness of $x \in \{0,\dots,R\}$ for this

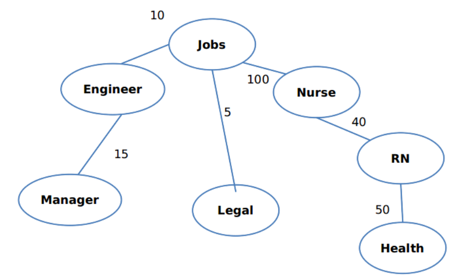



Figure 3: Overlap Tree

sub-tree. That is, for each sub-tree, we need to store up to $2 \cdot R$ solutions.

Using this property we can establish a dynamic programming algorithm that finds the optimal solution. A bottom-up process finds the optimal solution for each sub-tree until we get to the root, by considering all combinations of sharing the $R$ budget across siblings.

For every sub-tree rooted with root $\beta$, Let $cost^{p}_{\beta}(x)$ be the minimum rate we can use to achieve a usefulness of exactly $x$ given that $parent(\beta)$ is selected ($p=1$) or not ($p=0$). Let $\gamma_j$ be each one of the children of $\beta$. Let $c^{Res}(\beta)$ the residual utility of $\beta$ given that $parent(\beta)$ is selected, i.e $c(\beta)-c(\beta, parent(\beta))$. In the same way we define $u^{Res}(\beta)$ for residual usefulness.

Then, the following recurrence defines a pseudo-polynomial exact solution for *SHSF-R*.

$$cost^{p=1}_{\beta}(x) = \min\Big\{\min_{\sum j_i = x} \sum cost^{p=0}_{\gamma_j}(j_i),$$

$$\min_{\sum j_i = (x-u(\beta))} c(\beta) + \sum cost^{p=1}_{\gamma_j}(j_i)\}\Big\}$$

$$cost^{p=0}_{\beta}(x) = \min\Big\{\min_{\sum j_i = x} \sum cost^{p=1}_{\gamma_j}(j_i),$$

$$\min_{\sum j_i = (x-u^{Res}(\beta))} \{c^{Res}(\beta) + \sum cost^{p=1}_{\gamma_j}(j_i)\}\Big\}$$

**Lemma 1.** *The dynamic algorithm described above generates an exact solution for the* SHFS-R *assuming that the pair-wise overlaps are the only overlaps among filters.*

*Proof sketch:* By Induction. Consider a tree with root $\beta$ and assume the algorithm already calculated $cost^{p=1}_{\gamma_j)}(x)$ and $cost^{p=0}_{\gamma_j}(x)$ for each child of $\beta$ and $x \in 0 \dots U|\Omega|$, where $U$ is the maximum usefulness for any filter. Let assume that the $parent(\beta)$ will not be selected ($p=0$). Then, the minimum coverage with usefulness $u$ is the minimum of (1) adding $\beta$ to the solution and find the optimal way to select $x-u(\beta)$ matches, from its child's considering the overlaps $(\beta, \gamma_j)$. Hence, we combine the $cost^{p=1}_{\gamma_j}(x)$ assuming they are independent. (2) We do not add $\beta$ and we find the optimal way to obtain $u$ matches using only the child's assuming they are independent. In the case we add $p=1$ we only need to take the effect on $\beta$.

**TreeFPTA Algorithm** In above algorithm, given the optimal solution $OPT$ (that is, the maximum usefulness of a set of filters that have joint coverage up to R), we need to calculate all the possibles values for every possible value $x \in 0 \dots OPT$. This is clearly infeasible as $OPT$ is independent of the number of filters. For that, we rescale $OPT$ to make it dependent on the size of $\Omega$, and we study an approximation of the solution based on a problem transformation.

We adapt the knapsack polynomial-time approximation scheme (FPTAS) (Hochbaum 1996) to our problem. The algorithm scales the profit (usefulness) of each filter to a new $\tilde{u}(F_i)$ and then solves the scaled problem using the pseudo-polynomial algorithm presented above. The modification guarantees that the algorithm finishes in a polynomial number of steps on $\Omega$. Let $U$ be the maximum usefulness for any filter and let $factor = \frac{\epsilon U}{|\Omega|}$, where $\epsilon$ is the approximation ratio and can be modified. We modify each usefulness as $\tilde{u}(F_i) = \lceil \frac{u(F_i)}{factor} \rceil$ and the residuals $\tilde{ru}(F_i) = \lceil \frac{ru(F_i)}{factor} \rceil$ . We call this algorithm *TreeFPTA*

**Lemma 2.** TreeFPTA *is a* $1 - \epsilon$ *approximation assuming knowledge of the pair-wise overlaps*

The proof of the approximation bound is similar to the one used for the knapsack problem (Hochbaum 1996). This solution modifies the cost by scaling the contributions for each object. As filters have dependencies, the transformation to knapsack is not straightforward. To overcome the effect of residual utilities, we create extra variables that are activated when two neighbor filters are selected. These new variables are affected in the same way as regular contributions when scaled.

A simple modification can be used to obtain an algorith for the *SHSF-KR* variant. We add an extra dimension that considers the number of filters used in the selection. $cost_{\beta}^{p=1}(x, k)$ is the minimum coverage to obtain a usefulness $x$ with exactly $k$ filters. The pseudo-polynomial approximation depends on $K$, instead of $|\Phi|$.

## 4 Dynamic Hidden Stream Filtering Problem

**Solution Challenges and Overview** We focus on the scenario where the precision of filters is known at any time, but we must maintain the coverage values. The precision can be updated by periodically crowdsourcing subsets of the stream for classification, or by applying the same rules (and corresponding filters) that we generate for external data sources like news articles during the same time window. Dynamically updating the precision of stream classification rules has been studied in previous work, which we could apply in our problem as well (Nishida, Hoshide, and Fujimura 2012).

We propose an exploration vs. exploitation approach, inspired by Multi-Armed Bandit solutions (Anantharam, Varaiya, and Walrand 1987) (see Related Work section). The idea is, given our budget constraints $K$ and $R$, to exploit the filters that we believe that will give us a high profit (many useful documents) given the current knowledge of all filters' coverages, but also explore some promising filters in $\Omega - \Phi$ to update their coverage estimates and check if their benefit may surpass that of the selected ones in $\Phi$. Exploration also allows us to adapt to non-stationary domains.

Given a sample of new documents in the current time window, and previous coverage values, we compute estimates of the filter coverages $\hat{c_i}(t)$, and of the filter usefulness values $\hat{u_t}(F_i)$ at time $t$ for filter $F_i$:

$$\hat{u_t}(F_i) = p(F_i) \times \hat{c_t}(F_i) \quad (2)$$

We show how $\hat{c_t}(F_i)$ is estimated using $\hat{c_{t-1}}(F_i)$ later.

In each time window we need to take $K$ decisions, that correspond to the filters that will be selected in that step. After the window is finished we use the new information to update the estimates, and we start the process again.

Some of the challenges of selecting the best filters in DHSF problem, which make the problem significantly different from the well-studied Multi-Armed Bandit problem, are: (i) Weighted Cost: The cost of a filter changes over time and we only know an estimate of it. In the simple multi-armed bandit problem, the cost of playing in any machine is the same, that is, filters are not weighted. (ii) Overlap: Due to filters overlap, the cost of a filter is not constant but depends on the other filters that are scheduled in the same window. Further, the profit (usefulness) of the solution is not simply the sum of the profit of each filter.(iii) Non-Stationary: Some domains are highly dynamic and precision/coverage have a high variance.

Table 5: Proposed Algorithms for DHSF Problem Variants.

| Problem | Algorithm | Description |
|---|---|---|
| DHSF-R | DNOV-R | Select by Precision |
| DHSF-K | DNOV-K | Select by Estimated Usefulness |
| DHSF-KR | DNOV-KR | Hybrid |

### 4.1 DHSF Algorithms

In this section we present algorithms for the DHSF problem, which are summarized in Table 5. To simplify the presentation, we first present the algorithms assuming no overlap among filters; as we see the problem is hard even under this assumption. At the end of the section we discuss how we maintain estimates of overlap among filters and how these estimates are incorporated into the presented algorithms.

**DHSF-R Problem:** In *DHSF-R* the rate is the critical (and only) constraint. We order the filters by decreasing precision and add to $\Phi$ until we fill the $R$ rate. As precision is static in our setting, this algorithm is not completely dynamic, and work very similar to the static version, *GREEDY-PREC*.

The main difference is how to maintain some estimate of the coverage, as we need this value to decide if a filter is added in the solution The modified algorithm is called *DNOV-R*.

To maintain an estimate of the coverage (Equation 2), our scheduling approach will leverage well-known Reinforcement Learning Strategies (Kaelbling, Littman, and Moore 1996). These strategies have strong theoretic guarantees on simple scenarios like the Simple Multi-Armed Bandit problem. However, they have also been shown to be successful on problems that break the model assumptions (independence of the decisions, stationary model, complex payoff).

To maintain estimates of coverage, a simple model is to use the average of the coverage we obtain each time the filter is scheduled (selected in $\Phi$). This estimate is the MLE for the stationary model. Nevertheless, as we consider non-stationary distributions, we expect that an exponential weighted model that gives a higher importance to new observations will perform better. This model has been successfully used for adaptive training models (Kaelbling, Littman, and Moore 1996). In particular, we update the estimated coverage in each step using the following equation:

$$\hat{c_{t+1}}(F_i) \leftarrow \hat{c}_t(F_i) + \alpha \cdot (c_{t+1}(F_i) - \hat{c}_t(F_i))$$

where $\hat{c}_t(F_i)$ is the estimated coverage for the filter $F_i$ at moment $t$, and $c_t(F_i)$ is the coverage for the particular iteration. $\alpha$ controls the weights (decay) of the previous estimates.

**DHSF-K Problem:** Here, we want to add filters that have the highest usefulness. To maintain an estimate of the usefulness we use equation 2. The coverage estimate is calculated in the same way as before. However compared with the previous version we do not select by a fixed value but schedule the selection depending on the current usefulness estimates.

To balance the exploration and exploitation, we use a variation of the soft-max action selection presented in (Kaelbling, Littman, and Moore 1996). The idea is to select the filters using a probability distribution $\mathbb{P}$; that weighs each filter depending on the expected profit. Ranking using the soft-max allows us to focus the exploration of the most promising filters. The probability of selecting the filter $F_i$ is given by the following distribution:

$$P_t(F_i) = \frac{e^{\hat{u_t}(F_i)/\tau}}{\sum_{F_j \in \Omega} e^{\hat{u_t}(F_j)/\tau}}$$

The parameter $\tau$ is commonly connected to the concept of temperature. If the temperature is very high, the system tends to be random, that is, it performs more exploration. If the temperature cools down the system becomes more greedy and tries to select the most relevant filter, that is, exploitation dominates. In the experiments we start from a smaller $\tau$ and slowly colds (logarithmically) until it reaches an effective value, as discussed in the Experimental section.

The soft-max rule provides a framework to take one decision. However our scheduler must take several ($K$) decisions in each window. We use a greedy loop that selects a single filter using the soft-max probabilities, and then updates the rest of the probabilities given the selected filter.

**DHSF-KR Problem:** When we have both constraints, the problem becomes how to decide which of the two is the critical constraint in each step, so that we make a selection accordingly. For instance, if we have used up half of our $K$ filters, but only 10% of the rate $R$, then $K$ becomes more critical, and the selection of the next filter should take this into account.

The idea is to combine the two previous approaches, selecting one of them for each greedy selection of a filter. In particular we select by precision (as in *GREEDY-PREC*) if the ratio between the remainder rate and $R$ is less than the remainder number of filters and $K$, otherwise we use the usefulness (as in *DNOV-K*). Algorithm 1 shows this hybrid algorithm, which we call *DNOV-KR*. *generateDistributionPrecision()* assigns to each filter a probability proportional to its precision.

**Algorithm 1** DNOV-KR

```
procedure DNOV-KR(F, K, R, τ)
    G ← ∅
    i ← 0; R' ← R
    while i < K ∧ |G| < R do
        if (K − i)/K < R'/R then
            ℙ ← generateDistributionUsefulness()
        else
            ℙ ← generateDistributionPrecision()
        end if
        p ← random(0, 1)
        j ← select_softmax(Ω, ℙ, p, τ)
        if |F_i ∪ G| < R then
            R' = R' − c(F_j)
            G = G ∪ F_i
            Ω = Ω − F_j
        end if
        i ← i + 1
    end while
end procedure
```

**Account for the overlap among filters**

To handle the overlap among filters, we maintain estimates of the overlap values. The two key challenges are how to obtain these estimates and how to use them.

For the first challenge, let $c_t(F_i, F_j)$ be is the number of documents matching both filters $F_i$ and $F_j$, during the time window starting at time $t$. Then we want to obtain an estimate $\hat{c}_{t+1}(F_i, F_j)$ for the next window. This estimated is value whenever $F_i$ or $F_j$ is selected (is in $\Phi$). If $F_i \in \Phi$, then we count how many documents returned satisfy each of the other filters $F_j$, even if $F_j \notin \Phi$. Then we estimate the overlap as:

$$\hat{c_{t+1}}(F_i, F_j) \leftarrow \hat{c}_t(F_i, F_j) + \alpha(c_{t+1}(F_i, F_j) - \hat{c}_t(F_i, F_j))$$

To obtain the number of documents that are useful in the intersection we are faced with a problem. As we do not know the precision on the intersection we can assume the worst case and use the maximum of the two filters we are considering. In particular:

$$\hat{u_t}(F_i, F_j) = max(p(F_i), p(F_j)) \cdot \hat{c}_t(F_i, F_j) \quad (3)$$

Given now the pairwise estimates we can use these values to update the probabilities in each step of the algorithm, considering the filters that are already selected. For example, if we have three filters $F_1, F_2, F_3$ and we have already selected $F_1$, then *DNOV-K* would calculate $\mathbb{P}$ using usefulness values $u(F_2) - u(F_{12})$ and $u(F_3) - u(F_{13})$.

## 5 Experimental Evaluation

In the following section we describe the data acquisition process and how we create the set of candidate filters $\Omega$ using rule-based classification methods. Then we present our results for the Static Hidden Stream Filtering problem and its variants. Finally, we present the results for the Dynamic Hidden Stream Filtering problem.

### 5.1 Data Acquisition and Rule Generation

We obtained the tweets from the Twitter public Streaming API, without specifying any conditions, which returns about one percent of the total traffic, between January 1st and January 31st of 2013. We removed the tweets that do not contain any hashtag (we build classifiers based on hashtags) or

Table 6: Dataset and Rules Descriptive Statistics

| | |
|---|---|
| #Avg. Tweets per Day | 4.2M |
| #Avg. Tweets per Day (Hashtag) | 800K |
| #Tweets per Day (Hashtag+English+ Norm.) | 303K |
| #Tweets Full DataSet (Hashtag+English+Norm.) | 9.4M |
| #Rules Per Hashtag (Min,Max,Avg) | 3/96/18.1 |
| #Conflicts (5%) | 14.1 |
| Precision Per Rule (Min,Max,Avg) | 0.05/1.0/0.45 |
| Coverage Per Rule (Min,Max,Avg) | 5/37803/307.6 |

are not in English (using http://code.google.com/p/language-detection/). Tweets are normalized to remove user names, URLs, the retweet symbol (RT), and the pound (#) character on hashtags. Finally, we remove tweets that have less than 10 characters. Table 6 shows the statistics of the dataset.

We divide time in windows of 4 hours, that is, there are 6 windows per day, and $6 \cdot 31 = 186$ in total. We assign a tweet to a window based on its timestamp. The first 48 hours (12 windows) are used as training set to obtain the candidate set of filters $\Omega$. For the static problem variants, the training set is also used to obtain the precisions and the initial coverages of the filters, whereas for the dynamic problem variants it is only used to obtain the precisions of the rules. The rest windows are used for evaluation.

Hashtags are used to build classifiers to evaluate our methods. That is, each hashtag is viewed as a topic, and we build a classifier that uses the rest keywords (excluding the hashtag itself) in the tweets that contain the hashtag to predict if the hashtag is present or not. Hashtags have been used to build Twitter classifiers before (Nishida, Hoshide, and Fujimura 2012).

We obtain the top 100 most frequent hashtags in the first 48 hours, ignoring those that do not represent a topic, specifically the ones that contain any of the following patterns: FF, FOLLOW, F4F , JFB, TFB, RT, 0DAY, TFW. These hashtags refer to meta-content in the network instead of a particular topic. They are also highly abused by spammers.

For each hashtag, we generate classification rules using a modified version of WEKA's 3.7.9 Ripper implementation ti make compatible with the IREP algorithm (Cohen 1995). The modifications are as follows: (i) Rules are constrained to the existence of the terms (keywords), that is, we disallow negated features. This is consistent with the Twitter Streaming API, which does not allow negation conditions. (ii) The minimum precision as stop condition is ignored. In the default version a rule is accepted if the precision is at least half of the covered examples. The reason we ignored this parameter is to get more rules in $\Omega$, given that our goal is to achieve maximum recall. (iii) We increase the Description Length parameter used as stopping criterion is increased, augmenting the number of rules generated by IREP. (iv) We do not run the post-optimization section of the algorithm.

We use 10 cross-validation for training-pruning. As the dataset is highly unbalanced we use all the tweets that are tagged with a hashtag and only 1% of the ones that are not when we create the training set. The positive/negative ratio is then around 1/10 on the training set. Table 6 shows the best, lowest and average precision/coverage and number of filters for our 100 hashtags. To measure the amount of overlap we show the number of pairs with high overlap, i.e the number of documents covered by both filters divided by the number of documents covered by any filter is at least 0.05). As we can see the number of pairs of rules that have conflicts is small.

## 5.2 Static Hidden Stream Filtering (SHSF) Problem

Our experiments simulate an environment where the budget is divided for multiple use cases. This means we simulate multiple users and requirement sharing the bigger budget. Our strategies allows to give to divide $K$ and $R$ in a fair way for this settings.

As mentioned above, we use the training set to obtain our initial estimates of precision and coverage. The ultimate evaluation metric is the number of useful documents (tweets that contain the hashtag) in each window, given the selected filters. We use different values of $K$ and $R$. The value of $R$ is expressed as a percentage of document returned if we use all the filters in $\Omega$. We use $R \in \{50, 25, 12.5, 6.25\}$. For $K$ we use values $\{2, 4, 8, 16, 32\}$. The value $\epsilon$ is set to be a 0.8 for the approximation algorithms. We compare both our approximation algorithm TreeFPTA and the greedy *GREEDY-COV*. As a baseline we use the rules as returned by the original algorithm. We call this algorithm RULEORDER.

Note that a user may share his/her budget across several topics or an application may share the budget between multiple users. Hence, even though Twitter's $K = 400$ may sound large, it is too small if it must accommodate many classifiers, where each classifier may only use as few as $K = 5$ or less.

Note that *GREEDY-COV* rate is very close to $R$, because on the one hand it underestimates the rate by assuming that the overlapping posts between a new filter and the already selected filters are distinct to each other, and on the other hand overestimates the rate by assuming all n-way ($n > 2$) overlaps are zero. However, this is not the case for TreeFPTA, because it underestimates the rate in two ways: by assuming n-way ($n > 2$) overlaps are zero and ignoring some 2-way overlaps. For that, we also show the performance of *TreeFPTA* for input $R' = 1.1 \cdot R$, which leads to actual rate $R$.

Table 7 shows the number of the returned useful (containing the hashtag) documents in the test set, averaged over the time windows in the test set considering all the $K$ and $R$ combinations. We see that two of the proposed strategies clearly improve the baseline. *GREEDY-PREC*'s meticulous selection favors rules with higher precision rather than usefulness. The TreeFPTA approximation is at least 7-9% superior than *GREEDY-COV*. Nevertheless, we note that *GREEDY-COV* is competitive while having lower complexity and an easier implementation.

Table 7 also compares the average number of returned documents (regardless of usefulness) over all windows in the test set, for all the strategies considering all $K, R$ combinations. We see that *GREEDY-PREC* has the smallest number of returned documents, that is, it uses the smallest ratio of the available rate budget $R$. On the other hand, the

Table 7: SHSF-KR averaged for multiple rates and rates.

| Algorithm | Useful | Covered | Precision |
|---|---|---|---|
| RULEORDER | 2265 (+ 0%) | 15837 (+ 0%) | 0.14 |
| GREEDY-PREC | 1191 (-48%) | 2288 (-86%) | **0.52** |
| GREEDY-COV | 2475 (+ 9%) | 24067 (+52%) | 0.10 |
| TreeFPTA | **2695 (+18%)** | 26203 (+65%) | 0.10 |

Table 8: DHSF-K/R averaged for multiple $R$ and $K$.

| Algorithm | Usef. | Cov. | Prec. | Usef. | Cov. | Prec. |
|---|---|---|---|---|---|---|
| | Only-R | | | Only-K | | |
| DNOV-K | 0.80 | 0.95 | 0.06 | 0.70 | 0.68 | 0.13 |
| DNOV-R | 0.86 | 0.89 | 0.07 | 0.51 | 0.21 | 0.30 |
| ORACLE | 1.00 | 1.0 | 0.07 | 1.00 | 1.0 | 0.12 |
| RANDOM | 0.73 | 0.94 | 0.05 | 0.37 | 0.33 | 0.13 |

*GREEDY-COV* and *TreeFPTA* algorithms have similar number of returned documents.

### 5.3 Dynamic Hidden Stream Filtering (DHSF) Problem

This section study the behavior of the strategies to solve the DHSF problem. Again, we use the first 48 hours to obtain the initial rules. We use the precision in the training set as our oracle precision, that is, we assume the precision is fixed. On the other hand, we assume no initial knowledge of the coverage of each filter; we consider an initial filter coverage of $R/K$ for each filter. Parameter $\alpha$ is set to 0.95 and the temperature is updated as $\tau(t) = 1.0/log(t) + 1.0$ where $t$ is the current time window, until it reaches $\tau = 2/7$, and then remains constant. We found that these parameters perform reasonably in our experimental setting.

We use the strategies *DNOV-R*, *DNOV-K* and *DNOV-KR*. We also add two baselines: the first one is a version of the *GREEDY-COV* with perfect knowledge, i.e., we know the actual coverage of all the filters in each step (clearly this is unrealistic). The idea is to use this baseline as an oracle to compare against. The second baseline selects a random set of rules and uses it if it does not breach any of the constraints. The reason is to see if our algorithms are performing better than a random selection process. We call this algorithms *ORACLE* and *RANDOM*.

Table 8 shows the results for the *DHSF-R* problem averaged over different rates $R$ (3%, 6%, 12% and 25%). All the strategies are pretty close to the fully informed (*ORACLE*) strategy. As the number of rules we can add is unbounded, we can fit as many rules as we can until we get the bound $R$. As expected, selecting by precision is the best strategy when we are only bounded by rate. The reasoning is that this strategy will fill the budget with small rules at the beginning, reducing later regret. In particular for lower rates (3%, 6%) *DNOV-R* is 15%-20% better than *DNOV-K*.

Table 8 shows the results for the *DHFS-K* problem average for different values of $K$ (4,8,16). In this case we expect that selecting by usefulness we can use the rule budget efficiently. As we can see the *DNOV-K* is the best algorithm out-performing the selection by precision. The *DNOV-R* is at least 25% worst that our best strategy, but still improve the random strategy.

Table 9: DHSF-KR averaged for multiple $R$ and $K$.

| Algorithm | Useful | Covered | Precision | Overflow |
|---|---|---|---|---|
| DNOV-K | 0.60 | 0.59 | 0.13 | 11 (6%) |
| DNOV-R | 0.23 | 0.48 | 0.26 | 10 (6%) |
| DNOV-KR | 0.60 | 0.58 | 0.13 | 5 (2%) |
| ORACLE | 1.00 | 1.00 | 0.13 | 22 (12%) |
| RANDOM | 0.39 | 0.33 | 0.11 | 8 (5%) |

Figure 4 shows the results for *DHSF-KR* problem. As the number of rules plays an important role we see that strategies that consider the usefulness are better than *DNOV-R*. Combining the precision is positive, as the hybrid seems to improve the basic *DHSF-K* algorithm.

In general, our algorithms outperform the random baseline. However, selecting by precision seems to be a poor strategy when we consider both $K$ and $R$ constraints. The usefulness directed strategy is better across all the combinations. The hybrid algorithm, which combines both paradigms, is more robust and effective in general, improving the usefulness strategy.

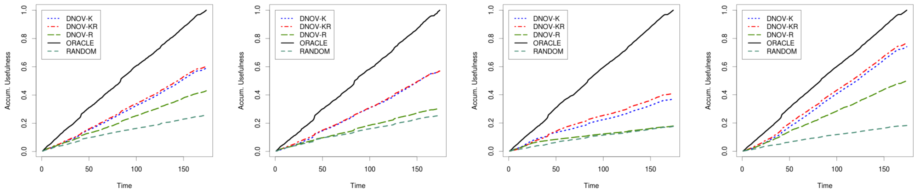
Table 9 summarizes the performance of all the techniques averaged across different $K$ (4,8,16) and $R$ (3%,6%,12%,25%,50%) values. All the numbers are normalized against the ORACLE strategy. As we can see the proposed strategies are between 40-70% better than the baseline (*RANDOM*). If we optimize for usefulness then *DNOV-K* and *DNOV-KR* are the best strategies. If we optimize for precision or we want to reduce the number of overflows (times the selection goes over the rate $R$) then *DNOV-R* is the best strategy.

## 6 Related Work

**Information Filtering** The motivation of our work is similar to the classical Personalized Information Filtering Problem (Hanani, Shapira, and Shoval 2001; Belkin and Croft 1992; Foltz and Dumais 1992). However, in our case the stream is hidden, and we have strict access constraints. Other filtering works related to our application are: incremental explicit feedback (Allan 1996), implicit feedback (Morita and Shinoda 1994) and social filtering (Morita and Shinoda 1994). The last aspect is interesting as users would share filters or profiles instead of documents.

**Exploring Hidden Web Databases** The Hidden Web (Bergman 2001) is the part of the Web that is accessible through forms, instead of hyperlinks. Many approaches on how to crawl, estimate, categorize (Gravano, Ipeirotis, and Sahami 2003), or obtain relevant documents (Hristidis, Hu, and Ipeirotis 2011; Ntoulas, Pzerfos, and Cho 2005) from these sources have been studied. Our problem can be viewed as a streaming version of this problem, since the hidden stream can only be accessed through a restricted query form, which are for instance the calls to the Twitter Streaming API.

**Multi-Armed Bandits and Adaptive Learning** A related problem to our problem is the multi-player multi-armed bandit (Anantharam, Varaiya, and Walrand 1987) problem. In




(a) K=8, R=6% (b) K=8, R=12% (c) K=8, R=25% (d) K=8, R=50%

Figure 4: Accumulated Useful Documents by the DHFS-KR strategies.

this game a player faces $n$ machines/bandit where each machine is characterized by some fixed expectation of winning $\theta$. In each turn, the player can play $K$ levers from the $n$ machines. The paper present a strategy with bounded guarantees on the expected winnings. However, the process does not consider a budget similar to our rate $R$. Another problem that can be also related is the budgeted multi-armed bandit (Tran-Thanh et al. 2010). In this case each pulling of a lever has a cost $c_i$ and the player has a budget $B$ for all his plays. A difference with our approach is we have a budget per play, instead of unified budget.

## 7 Conclusions

We presented the Hidden Stream Filtering problem, where a service provider allows access to a stream of documents, through keyword-based continuous queries (filters). The user must also honor an access contract that includes restrictions on how the service can be used. We focused on two common constraints: maximum number of filters and maximum rate of documents returned in a unit of time. We showed that selecting the best filters to maximize the number of useful documents returned is NP-Complete, so we proposed heuristics and approximation algorithms with provable error bounds.

## Acknowledgements.

This Project was partially supported bythe National Science Foundation (NSF) grant IIS-1216007, the Defense Threat Reduction Agency (DTRA) grant HDTRA1-13-C-0078, and a Google Focused Award.

## References

Allan, J. 1996. Incremental relevance feedback for information filtering. In *Proc. of ACM SIGIR*, 270–278. ACM.

Anantharam, V.; Varaiya, P.; and Walrand, J. 1987. Asymptotically efficient allocation rules for the multiarmed bandit problem with multiple plays-part i: I.i.d. rewards. *Automatic Control, IEEE Transactions on* 32(11):968–976.

Belkin, N. J., and Croft, W. B. 1992. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM* 35(12):29–38.

Bergman, M. K. 2001. White paper: the deep web: surfacing hidden value. *journal of electronic publishing* 7(1).

Chow, C., and Liu, C. 1968. Approximating discrete probability distributions with dependence trees. *Information Theory, IEEE Transactions on* 14(3):462–467.

Cohen, W. W. 1995. Fast effective rule induction. In *Proc. of ICML*, 115–123. Morgan Kaufmann.

Dries, A., and Ruckert, U. 2009. Adaptive concept drift detection. *Stat. Anal. Data Min.* 311–327.

Foltz, P. W., and Dumais, S. T. 1992. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM* 35(12):51–60.

Fung, G.; Sandilya, S.; and Rao, R. B. 2005. Rule extraction from linear support vector machines. In *Proc. ACM SIGKDD*, 32–40.

Gravano, L.; Ipeirotis, P. G.; and Sahami, M. 2003. Qprober: A system for automatic classification of hidden-web databases. *ACM Transactions on Information Systems (TOIS)* 21(1):1–41.

Han, J. 2005. *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hanani, U.; Shapira, B.; and Shoval, P. 2001. Information filtering: Overview of issues, research and systems. *User Modeling and User-Adapted Interaction* 11(3):203–259.

Hochbaum, D. S. 1996. *Approximation algorithms for NP-hard problems*. PWS Publishing Co.

Hristidis, V.; Hu, Y.; and Ipeirotis, P. 2011. Relevance-based retrieval on hidden-web text databases without ranking support. *Knowledge and Data Engineering, IEEE Transactions on* 23(10):1555–1568.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *arXiv preprint cs/9605103*.

Morita, M., and Shinoda, Y. 1994. Information filtering based on user behavior analysis and best match text retrieval. In *Proc. of ACM SIGIR*, 272–281.

Nishida, K.; Hoshide, T.; and Fujimura, K. 2012. Improving tweet stream classification by detecting changes in word probability. In *Proc. of ACM SIGIR*, SIGIR '12, 971–980.

Ntoulas, A.; Pzerfos, P.; and Cho, J. 2005. Downloading textual hidden web content through keyword queries. In *Proc. of ACM/IEEE JCDL'05.*, 100–109. IEEE.

Provost, J. 1999. Naive Bayes vs. rule-learning in classification of email. (AI-TR-99-284, University of Texas at Austin.

Quinlan, J. R. 1987. Generating production rules from decision trees. In *Proc. AAAI 1987*, volume 30107, 304–307.

Raghavan, S., and Garcia-Molina, H. 2000. Crawling the hidden web. Technical Report 2000-36, Stanford InfoLab.

Sebastiani, F. 2002. Machine learning in automated text categorization. *ACM computing surveys (CSUR)* 34(1):1–47.

Towell, G. G., and Shavlik, J. W. 1993. Extracting refined rules from knowledge-based neural networks. *Machine learning* 13(1):71–101.

Tran-Thanh, L.; Chapman, A.; Munoz De Cote Flores Luna, J. E.; Rogers, A.; and Jennings, N. R. 2010. Epsilon–first policies for budget–limited multi-armed bandits. *Proc. of AAAI*.