Interactive Refinement of Filtering Queries on Streaming Intelligence Data*

Yiming Ma and Dawit Yimam Seid

School of Information and Computer Science,
University of California, Irvine, USA
{maym, dseid}@ics.uci.edu

Abstract. Intelligence analysis involves routinely monitoring and correlating large amount of data streaming from multiple sources. In order to detect important patterns, the analyst normally needs to look at data gathered over a certain time window. Given the size of data and rate at which it arrives, it is usually impossible to manually process every record or case. Instead, automated filtering (classification) mechanisms are employed to identify information relevant to the analyst's task. In this paper, we present a novel system framework called FREESIA (Filter REfinement Engine for Streaming InformAtion) to effectively generate, utilize and update filtering queries on streaming data.

1 Introduction

Intelligence analysis involves routinely monitoring and correlating large amount of data streaming from multiple sources. In order to detect important patterns, the analyst normally needs to look at data gathered over a certain time window. However not all data is relevant for the analysts task; the relevant set of data needs to be selected from the streaming data. The task of monitoring involves a combination of automated filtering system to identify candidate cases and human analysis of cases and their related data. The filtering system is typically part of a data aggregation server to which transaction data are fed from numerous agencies in near real time. An analyst stores his task or goal specific filters that are matched to incoming data as it flows. Multiple filters may be needed to extract information from different sources.

Formulating the right filtering queries is an iterative and evolutionary process. Initially the analyst may draw from his domain knowledge to express a filter. But this filter needs to be refined based on how well it performs. Besides, it needs to be refined to capture the changes over time in the emphasis given to various attributes. In this paper we consider how to enable the filtering system to perform automatic query refinement based on minimal and continuous feedback gathered from the user. Below we give examples drawn from two intelligence related tasks that illustrate how such a system can be employed:

* This research was supported by the National Science Foundation under Award Numbers 0331707 and 0331690.

S. Mehrotra et al. (Eds.): ISI 2006, LNCS 3975, pp. 35–47, 2006.

Example 1 (Intelligence Report Monitoring). Massive amount of incident reports are continuously generated by law enforcement agencies which are monitored by analysts at different levels to detect trends and correlations. For instance, an analyst at federal level may want to continuously filter and analyze all relevant incident reports from local agencies that relates to multi-housing (e.g. rental apartment or condominium) and lodging (e.g. hotels or motels) facilities that have national monuments in their proximity. The analyst may also express detailed preferences on the attributes related to the suspects described in the incident report. To achieve this, the analyst draws from his domain knowledge to specify an initial (imprecise) filter to the data reporting server. The server matches the filter with incoming reports. In cases where matching reports can be large, it will be useful if the system can also rank the reports based on how strongly they match the given filter. To refine both the classification and ranking capabilities of the filter over time, the system offers the analyst a feature to provide feedback on the relevance of the reports. Based on the feedback the system automatically refines the filter.

Example 2 (Intelligence Information Dissemination). Large amount of intelligence information is gathered everyday from various sensors. For instance, US custom services use various sensing technologies (e.g., cameras, finger-print reader) to gather passenger information from airports and seaports. Different feature extraction tools are used to extract features from these data. Data matching given multi-feature criteria, watch-lists or archived data must be disseminated to analysts in different agencies for further processing. Analysts register filtering queries to the central system that gathers the data which then disseminates relevant information in a prioritized manner to analysts. Similar to the previous example, feedback from the analyst can be used to automatically adjust filtering queries stored in the system.

Technically, filtering queries can be considered as *classifiers* since their purpose is to classify each incoming data item as relevant (i.e. belong to the target class) or non-relevant. However, the following three important requirements distinguish our filtering queries from traditional classifiers:

1. Ranking and Scoring. For the purpose of filtering data instances belonging to a target class from massive volumes of streaming data, classifiers that merely make binary decisions are inadequate. The classifiers need to also score and rank records based on how strongly they match the filters. Ranking is useful for two reasons: (1) it enables the analyst to prioritize the processing of records, and (2) in cases where rigid binary partitioning of relevant and non-relevant data is undesirable, it facilitates the prioritization of records that are highly likely to be in the target class while at the same time not eliminating records. The latter issue is particularly important due to the fact that in most situations the filters are not crisp rules but rather fuzzy and approximate. This makes classifiers that score and rank data instances more appropriate than classifiers that only make binary decisions on class membership.

2. Incorporating Analyst's Domain knowledge. In a great majority of intelligence applications, analyst's domain knowledge (e.g. about features of

suspects, etc.) forms a critical component. Hence, it is imperative that the system provides a mechanism to readily incorporate domain knowledge-induced filtering rules. However, while filtering rules representing domain knowledge are normally vague and imprecise, current database systems on which much of data filtering is carried out require crisp expressions. In order to express filtering rules on such systems, analysts are forced to convert their rules to very complex crisp expressions. To avoid this problem, the filtering system needs to allow direct execution of inexact filtering queries.

3. Interactive Refinement. As illustrated in the above examples, allowing the analyst to refine filters through relevance feedback (a.k.a. supervised learning) is an important requirement. This becomes necessary when the rules expressed by the analyst fail to capture the desired domain knowledge, or rules change over time. An important issue to notice here is that unlike traditional approaches where a classifier is learned and then applied in distinct phases, here the classifier needs to be incrementally refined using a feedback loop. Also notice that human domain knowledge is incorporated in two ways: first, through submission of domain knowledge in the form of initial filtering queries, and second, through feedback on the classified records.

In this paper, we propose a framework called FREESIA (Filter REfinement Engine for Streaming InformAtion) that meets the above requirements. FREESIA achieves ranking of streaming data by representing filtering queries (classifiers) as multi-parametric similarity queries which allow the analyst to express his imprecise filtering rules. Then, in the course of data analysis, the analyst can refine and update these filters through example-based training so as to achieve required accuracy and meet evolving demands. To efficiently support such dynamic adaptation of filters, FREESIA provides a set of algorithms for refining filters based on continuous relevance feedback.

2 Definitions and Preliminaries

2.1 Data Model

Filters in FREESIA assume a structured multi-dimensional data. However, originally the data can be either a set of relational tables or in any unstructured/semi-structured format. If the data is unstructured, data extraction tools[1] can be first applied to extract relevant values (e.g. names, places, time, etc.). The extracted data is then represented in the form of attribute-value pairs and fed into filtering modules.

2.2 Filtering Query Model

In this section we define a flexible query model that is powerful enough to capture human supplied filters and domain knowledge in addition to enabling incremental refinement. A filtering query or rule, henceforth simply referred to as *filter* or

[1] For example, Attensity's Extraction Engines: www.attensity.com

classifier, consists of four components: a set of similarity predicates structured in DNF form (Disjunctive Normal Form), a set of weights assigned to each similarity predicate, a ranking function and a cut-off value.

Definition 1. *A filter (classifier) is represented as a quadruple* $\langle \rho, \omega, \phi, \alpha \rangle$ *where* ρ *is a conditional expression,* ω *is a set of weights,* ϕ *is a ranking function and* α *is a cut-off value. Below we give a brief description of these four elements.*

Conditional Expression: A conditional expression, ρ, is a DNF (Disjunctive Normal Form) expression over similarity predicates. Formally, an expression $Q = C_1 \vee C_2 \vee \ldots \vee C_n$ is a DNF expression where $C_i = C_{i1} \wedge C_{i2} \ldots, C_{in}$ is a conjunction, and each C_{ij} is a similarity predicate. A *similarity predicate* is defined over the domain of a given data type (attribute type). A similarity predicate takes three inputs: (1) an attribute value from a data record, t, (2) a target value that can be a *set* of points or ranges, and (3) a similarity function, f, that computes the similarity between a data value and the target value. A similarity function is a mapping from two data attribute values, v_1 and v_2, to the range [0,1], $f : v_1 \times v_2 \rightarrow [0, 1]$. The values v_1 and v_2 can be either point values or range values. Similarity functions can be defined for data types or for specific attributes as part of the filtering system.

DNF Ranking Functions, Weights and Cut-off: A *DNF ranking function*,ϕ, is a domain-specific function used to compute the score of an incoming record by aggregating scores from individual similarity predicates according to the DNF structure of ρ and its corresponding set (template) of *weights* that indicate the importance of each similarity predicate. The template of weights, ω, corresponds to the structure of the search condition and associates a weight to each predicate in a conjunction and also to each conjunction in the overall disjunction.

A DNF ranking function first uses *predicate weights* to assign aggregate scores for each conjunction, and it then uses *conjunction weights* to assign an overall score for the filter. A conjunction weight is in the range of [0, 1]. All predicate weights in a conjunction add up to 1 while all conjunction weights in a disjunction may not add up to 1. We aggregate the scores from predicates in a conjunction with a weighted L_1 metric (weighted summation). Using weighted L_1 metric as a conjunction aggregation function has been widely used in text IR query models where a query is typically expressed as a single conjunction [12, 10]. To compute an overall score of a query (disjunction), we use the MAX function over the weighted conjunction scores. MAX is one of the most popular disjunction aggregation functions [4].

2.3 Filter Refinement Model

The similarity conditions constituting a filter are refined using relevance feedback that is used as real-time training example to adapt the predicates, condition structure and corresponding weights to the information needs of the analyst. More formally, given a filter, Q, a set R of the top k records returned by Q, and relevance feedback F on these records (i.e., a triple $\langle Q, R, F \rangle$), the refinement

problem is to transform Q into Q' in such a way that, when Q' is used to filter future streaming information or is re-executed on archival information, it will return more relevant records. Section 3.2 will discuss in detail the types of feedback that are gathered by the system and how they are represented.

3 Our Approach

In this section, we present our proposed approach for applying and refining filters on streaming data. We first present an overall architecture of our system FREESIA followed by a description of how the analyst interacts with FREESIA (i.e. the feedback loop). We then propose algorithms that implement the classifier refinement and scoring/ranking model refinement components of FREESIA.

3.1 The FREESIA System Architecture

FREESIA's schematic design is depicted in Figure 1. The following four main components constitute the system.

Filter Processing Component. When a new data group is received by the system, the filters that are represented as similarity queries are executed by the Filter Processing Component in order to score and filter relevant target records. This component can be implemented in any commercial database system using common similarity query processing techniques (e.g. [5, 1]). To readily apply the similarity queries in this context, we use an SQL equivalent of the weighted DNF query defined in 2.2.

If the similarity query has been modified (refined) since its last execution, the system will also evaluate it on the *archived data store* which is used to store the *unseen* (but matching) records as well as *filtered out* records. Re-evaluating the query on the archive allows the identification of previously excluded records that match the current filter. The scored list of records that results from the filter processing component is passed to the ranking component.

Example 3. Consider the incident report analysis application from example 1. For simplicity suppose that a data instance consists of only the location coordinates, incident type, location type and number of suspects. Then one possible query that filters potential analyst is given below.

```
SELECT Location, IncidentType, LocType, NumSuspects, RankFunc(w1,s1, w2, s2, w12) AS S,
FROM IncidentReports
WHERE LocNear(location, National_Monument, s1) AND LocTypeLike(LocType, {multi-housing,
   lodging}, s2)
ORDER BY S desc
```

The label "`National_Monument`" stands for a set of national monuments stored separately. `LocNear` takes a given location and computes its distance from the nearest national monument. `LocTypeLike` implements heuristic techniques to match similarity of a location type to a classification hierarchy of places.

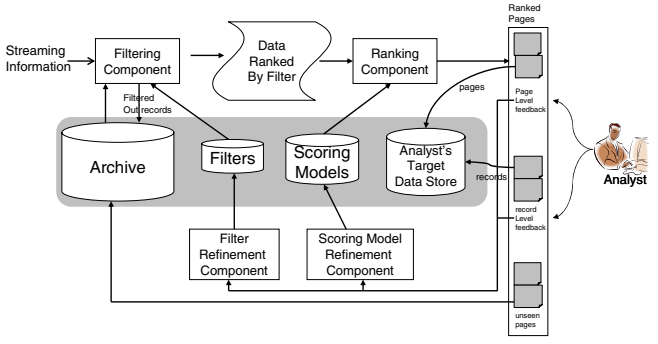

Fig. 1. FREESIA system overview

Ranking Component. This component applies scoring rules to produce a ranking of data instances. We employ two types of scoring methods. The first is the similarity scoring rule (i.e. the ranking function defined in Section 2.2) that is used by the Filter Processing Component. This represents the *long-term* filter of the analyst. In addition to this scoring rule, FREESIA also incorporates other scoring models that represent *short-term* (or special-case) rules that may not participate in the filtering process (i.e. are not evaluated as similarity queries) but are used for ranking. This, for instance, allows the analyst to temporarily force the system to rank reports that fulfill a given complex criteria at the top. Also, such rule can be specified by giving a record as a sample and asking "give me records like this". Many data mining methods can be used (e.g. [11]) to model such samples to produce scores for incoming records (more details on this will be given in Section 3.3). Given the scores from the similarity match and the scoring rules, the *Ranking Component* applies a combination method to produce the final ranking. As we mentioned in Section 2.3, the resulting ranked list of records is partitioned into pages for presentation to the analyst.

Filter Refinement Component. As discussed before, it is often necessary to interactively refine the analyst's initial filtering queries. This is achieved in FREESIA by collecting relevance feedback on the outputs of a filter. Upon seeing the ranked list of records, the analyst can submit feedback on the relevance (or otherwise) of the records - i.e. whether the records belong to the target class or not. Based on this feedback, the Filter Refinement Component refines the similarity queries. Section 3.3 will give details on the refinement process.

Scoring Model Refinement Component. In addition to its use for filter refinement, the feedback from the analyst is also utilized to refine the additional scoring rules. Section 3.3 will give details of this refinement process.

3.2 Gathering and Representing Feedback

Various types of feedback are gathered in FREESIA. One type of feedback is what we call *record-level feedback* where the analyst provides feedback on

particular records. However, in cases where the analyst receives large amount of matching reports, FREESIA also provides feature to provide group feedback. For this we exploit the fact that the system presents the results in pages (groups) of a certain size which enables *page-level feedback*. Often, when an analyst looks at a page, he can tell whether most of the records in the page are relevant in his initial scanning of the results. If some level of error in a page is tolerable, the analyst may want to accept all the records in a page for further processing. On the contrary, in cases where the analyst determines that a page contains only some relevant records, she may want to give record-level feedback.

For feedback gathering purposes, we distinguish three types of pages:

- *Highly relevant pages:* almost all the records in these pages are relevant. In other words, the analyst will use *all* the record in these pages for further actions despite the fact that there could be a few records in these page which are not relevant.
- *Relevant pages:* only some of the records in these pages are relevant. For these pages, the analyst provides feedback on each record.
- *Unseen pages:* these are the pages returned by the filter but are not viewed by the analyst. We assume that these are deemed to be non-relevant.

Despite the availability of page-level feedback, providing record-level feedback may still be a time consuming operation in some cases. To deal with this, FREESIA provides a parameter to specify the number of pages the analyst wants to give record-level feedback on. The remaining pages are considered unseen pages.

3.3 Filter Refinement

Feedback Preprocessing. The refinement strategies used by the Query Refinement and the Scoring Model Refinement components require two sets of data: contents of records on which the analyst gave relevance feedback (for e.g. to modify target values in predicates), and the feedback itself. We initially capture these two types of information in the following two tables:

(1) A *Result Table* contains the ranked list of records returned by the filter as well as the score assigned to each by the system.
(2) A *Feedback Table* contains the relevance feedback given by the analyst on records that are a subset of those in the Result Table. Particularly, this table contains record-level feedback given on *Relevant Pages*. Table 1 shows a sample feedback table from the intelligence report monitoring example.

Since data attributes can have complex and non-ordinal attributes, performing query refinement directly on the *result* and *feedback* tables is difficult as this will require specialized refinement method for each attribute type. To circumvent this problem, we transform the diverse data types and similarity predicates defined on them into a homogeneous similarity space on which a single refinement method can operate. We refer to the resulting table as *Scores Table*. It

Table 1. Example feedback table (I=Irrel,R=Rel)

ID	Location	Incident Type	LocType	#Suspect	FB
4	Irvine	photographing	retail center	2	I
1	Irvine	Bomb threat	hotel	0	R
7	LA	Request for building blueprints	apartment	1	R
10	LA	Unauthorized access	hotel	2	R
60	LA	Arson	Inn	5	I
2	San Diego	suspicious package delivery	office	1	I
3	San Diego	Larceny	Fed. Building	5	I

contains the following five columns that store statistical information useful for the refinement process:

(1) Entry Identifier:– This identifier is a triple $\langle AttributeID, ValueID, ConjunctionID\rangle$. The first two entries show the attribute-value pair. The Conjunction ID, which comes from the filter, identifies the conjunction that is satisfied by the attribute-value pair. Since we use a DNF representation, a conjunction contains one or more predicates.
(2) Counts of relevant records having the value in this entry.
(3) Count of non-relevant records having the value in this entry.
(4) Proximity to other values (of same attribute) of relevant records.
(5) Proximity to other values (of same attribute) of non-relevant records.

For every distinct value v_i in the scores table, we compute its weighted proximity to other relevant values of the same attribute *in the scores table* using the following formula:

$$v_i.RelevantCount + \sum_{j=1}^{k-1}(v_j.RelevantCount * sim(v_i, v_j))$$

where $v_i.RelevantCount$ is the count of relevant records (second column in the scores table), k is the total number of distinct values of the attribute corresponding to v_i that also have the same $conjID$, and $sim(v_i, v_j)$ is the similarity between v_i and v_j as computed by the similarity function corresponding to the attribute. In the same fashion, we compute proximity to non-relevant values using the above formula with $v_i.nonRelevantCount$ and $v_j.nonRelevantCount$ values. The intuition behind the proximity values is to bolster the exact counts of every distinct attribute-value pair in the scores table with the counts of other values of the same attribute weighted by their similarity to the attribute value at hand. This in essence allows us to capture the query region which the user is giving an example of. Table 2 shows an example scores table with one conjunction ($C1$) of one predicate on the attribute *location*.

Table 2. Example scores table

OBJ ID	$Rel\ Count$	$Irrel\ Count$	$AggRel\ Count$	$AggIrrel\ Count$
$< Location, Irvine, C1 >$	1	1	1+2*0.8 =2.6	$1 + 1 * 0.8 +2 * 0.2 = 2.2$
$< Location, LA, C1 >$	2	1	2+1*0.8 =2.8	1+1*0.8 +2*0.2=2.2
$< Location, SD, C1 >$	0	2	0+1*0.2 +2*0.2=0.6	2+1*0.2 +1*0.2=2.4

Refinement Algorithms. In principle, the feedback given by the analyst can potentially result in one of three types of filter refinement. A filter that is too specific can be made more general (called *filter expansion*), a filter that is too general can be made more specific (called *filter contraction*) and finally the target values of predicates can be shifted to a new value (called *filter movement*).

The refinement algorithm in figure 2 performs all three kinds of refinements and adjusts weights. The algorithm starts by pruning insignificant predicates (entries) from scores table using the $pruneInsigEntires$ function. Due to limited space, we skip the detailed discussion of this function. In short, it uses statistical method to measure the performance of each candidate predicate, and deletes the useless ones. The output of this function is a subset of the scores table, ST_{pruned}, whose entries are used as *candidates* to refine the filter.

Using the pruned scores table, ST_{pruned}, the algorithm next tries to determine whether the filter should be updated (line 3 to line 13). For each predicate in each conjunction, the algorithm first extracts the relevant entries, ST_{Pj}. This is performed by matching the $conjunctionID$ and attribute name in the filter with ST_{pruned} entries. This matching may result in many candidate predicates. Hence, we need a mechanism to select those that represent the right values to which we should move the filter. We do this selection by first clustering the candidate predicates and then choosing the cluster centroid as a representation of the new target value of P_j. For this, we use hierarchical agglomerative clustering (HAC) [2] method where the distance measure is computed based on the similarity between predicates in ST_{Pj}.

Once we get a set of candidate target points using HAC clustering, the algorithm tests whether each candidate is actually a new target value (line 7). The $isNewPoint$ function determines the closeness of each candidate to each of the existing filter predicate. If a candidate is not near any of the existing target points, we add it as a new target value of the current predicate (line 8).

Next, the algorithm updates the weights of the predicates and conjunctions in the query. The predicate weight is computed as the average confidence level

ComputeNewQuery()
Input: Filter (Q), Scores_table(ST)
$NumCase, NumRelCase, HACThresh$
Output: NewFilter
1. ST_{pruned} = pruneInsigEntries (NumCase, NumRelCase)
2. **Foreach** Conjunction (C_i) **in** Query
3. **Foreach** Predicate (P_j) **in** C_i
4. ST_{Pj} = filterScoreTable (ST_{pruned}, $P_j.attribute$, C_i)
5. $Clusters_{Pj}$ = computeHACCluster (ST_{Pj}, $HACThresh$)
6. **Foreach** Cluster (Cl_k) **in** $Clusters_{Pj}$
7. **if** P_j.isNewPoint($Cl_k.centroid$, P_j)
8. P_j.addQueryPoint ($Cl_k.centroid$)
9. **endif**
10. **endFor**
11. $P_j.weight$ = averageConf($P_j.queryPoints$)
12. **endFor**
13. $C_i.weight = \frac{\sum_{j=1}^{|C_i|} P_j.weight}{|C_i|}$
14. NewFilter.addConjunction(C_i)
15. NewFilter.normalizeWeight()
16. **endFor**

Fig. 2. ComputeNewFilter Algorithm

of the query points in the updated predicate. The confidence value of a predicate is computed based on its proximity values stored in the scores table as: $\frac{ProximityToRelevant}{ProximityToRelevant+ProximityToIrrelevant}$. The weight for a conjunction is computed as the *average* of the weights of its constituent predicates.

Refining The Scoring Model. The primary task of FREESIA's ranking component is to assign an accurate ranking score to each record. This component uses the maximum of the scores from the Filtering Component and the score from the short-term scoring model to be a record's final ranking score. When the analyst provides samples to form the scoring rules, many general incremental learning methods can be applied. In FREESIA, we use a pool based active learning method [11] which is suited to streaming data and is able to capture sample based *short-term* user model. It is a three-step procedure:

- Train a Naive Bayes classifier – *short-term* model – using sampled feedback records.
- Apply the *short-term* model to score the records returned by the Filter Component.
- Merge the scores from *long-term* model (i.e., filter score) and from *short-term* model.

4 Experiments

In this section, we present the results of the experiments we conducted to evaluate the effectiveness of our refinement method.

Table 3. Dataset Descriptions and Parameters

Dataset	**# Cases**	**# Cls.** attrs	**# cont** attrs	**# disc**	**Page** size	**Data** Group size
adult	32,561	2	6	8	40	1,000
covertype	10,000	7	10	40	40	1,000
hypo	3,163	2	7	18	20	316
waveform21	5,000	2	21	0	20	500

We used four real-life datasets from the UCI machine learning repository [8]. The datasets are a good ensemble to some intelligence data. They are reasonable in size and have predefined target (classes); they also cover some portions of US census data (adult), environmental data (covertype), disease data (hypo) and scientific analysis data (waveform21). Table 3 shows the characteristics of the datasets. There are two types of attributes in the datasets (viz. continuous and discrete). We manually generate 20 initial and target query pairs for each dataset.

Our evaluation process closely follows the FREESIA architecture(Section 3.1). Table 3 shows two of the parameters we used for each dataset, namely *page size* and *data group size*. Page size specifies the number of records in each page. Data group size shows the number of records streaming into the system at each iteration. In addition, we set two more parameters, namely *precision threshold* and *record-level feedback page threshold*. Precision threshold shows that if the precision in a page is higher than this number, the page will be treated as a highly relevant page. We use 80% for all data sets. For all data sets, record-level feedback is gathered for 2 page.

The initial query is executed in the first iteration. The system then presents the data in pages. If a page is a *highly relevant* page, the system continues fetching the next page, and no feedback will be given to the system. This is to simulate the page-level feedback. If a page is not a highly relevant page, then in reality record-level feedback will be given on this page. Since we are using a pre-labeled dataset, we simulate this feedback process by assigning the respective true label of each record in the page. If the number of record-level feedback pages is beyond the page limit of record-level feedback specified above, the system will dump the remaining pages to the data archive (i.e. they are unseen pages).

Tested Strategies. Four approaches were compared in our experiments:

(1) *Baseline method (Q).* This uses only the initial query.
(2) *Query and Scoring Model Refinement (QM+).* This refines the scoring model, but the similarity query is not refined. This, in effect, simply makes the query more specific.
(3) *Query Refinement (Q+).* This refines the similarity query only. This performs all three types of refinement.
(4) *Query Refinement and Scoring Model Refinement (Q+M+).* This refines both the query and the scoring models. This also refines all three types of refinement but is capable of producing much more focused queries.

4.1 Results

Figures 3 to 6 show the precision and recall measures across different refinement iterations. In the first two datasets (*adult* and *hypo*), we show results where the desired refinement of the initial queries is achieved in the first few iterations (around two or three iterations). Moreover, the system was able to maintain the high precision and recall measures across the subsequent iterations. As can be clearly seen in these two figures, the two algorithms that perform similarity

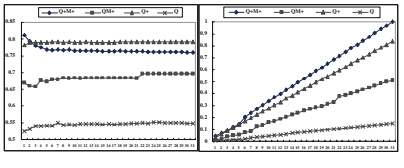

Fig. 3. Adult: Prec-Rec over 31 iters

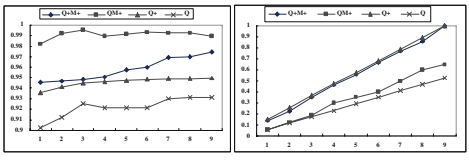

Fig. 4. Hypo: Prec-Rec over 9 iters

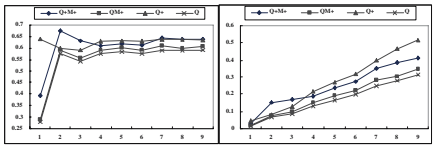

Fig. 5. Wave21: Prec-Rec over 9 iters

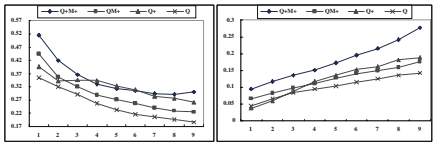

Fig. 6. Covertype: Prec-Rec over 9 iters

query refinement (i.e. $Q+$ and $Q+M+$) have much better performance compared to the other two which do not perform query refinement. For the dataset ($waveform21$), to achieve the desired refinements, more refinement iterations are required compared to the above two datasets (see the recall graph). Here as well $Q+M+$ and $QM+$ achieved the best precision and recall. The last dataset (*covertype*) shows cases where the initial query is very different from the desired target. As shown in the graph, precision declines as more iterations are needed (i.e. relatively more non-relevant records are retrieved). Still, $Q+M+$ performs better than the rest. The above results clearly show the effectiveness of FREESIA's refinement algorithms.

5 Related Work

The filtering process studied in this paper is related to target data selection techniques proposed in data mining on static data warehouses. However, unlike data mining on data warehouses (where a relevant subset of the database is filtered out for data mining tasks by carrying out as much refinement on the filters as required), in streaming data filtering has to be done continuously to allow data mining to occur as soon as the data arrives. There has been some research to address the problem of target subset selection from static data using classifiers [7, 9]. This body of research, however, only dealt with the problem of automatic classifier generation and the data considered were static. Recently, [3, 6] have considered the problem of data mining on streaming data. These works considered dynamic construction and maintenance of general models in a precise data environment. Whereas, our work deals with user predefined imprecise selection filters, and exploits the user knowledge to improve the accuracy of the filtering process.

6 Conclusions

In this paper, we have proposed a novel filtering framework called FREESIA, which enables analysts to apply the classifiers directly on database systems (in the form of similarity queries) to filter data instances that belong to a desired target class on a continuous basis. We believe our system can be used in many intelligence related tasks.

Acknowledgments. We would like to thank Prof. Sharad Mehrotra for giving us valuable feedback on this work.

References

1. S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *Proc. of the Twenty-fifth International Conference on Very Large Databases (VLDB99)*, 1999.
2. W. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. 1(1):7–24, 1984.

3. P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
4. R. Fagin. Combining Fuzzy Information from Multiple Systems. *Proc. of the 15th ACM Symp. on PODS*, 1996.
5. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS'2001, Santa Barnara, California*, pages 83 – 99, May 2001.
6. D. Lambert and J. C. Pinheiro. Mining a stream of transactions for customer patterns. In *Knowledge Discovery and Data Mining*, pages 305–310, 2001.
7. C. Ling and C. Li. Data mining for direct marketing: problems and solutions. In *Proceedings of ACM SIGKDD (KDD-98)*, pages 73–79, 1998.
8. C. J. Merz and P. Murphy. UCI Repository of Machine Learning Databases. http://www.cs.uci.edu/ mlearn/MLRepository.html, 1996.
9. G. Piatetsky-Shapiro and B. Masand. Estimating campaign benefits and modeling lift. In *Proceedings of ACM SIGKDD (KDD-99)*, pages 185–193, 1999.
10. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice Hall, 1971.
11. N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of ICML'01*, pages 441–448, 2001.
12. R. B. Yates and R. Neto. Modern information retrieval. In *ACM Press Series Addison Wesley*, 1999.