Uncertainty Sampling and Optimization for Interactive Database Exploration

Liping Peng[*], Enhui Huang[†], Yuqing Xing[*], Anna Liu[*], Yanlei Diao[*,†]
[*]University of Massachusetts Amherst, USA; [†] Ecole Polytechnique, France
[*]{lppeng, yanlei}@cs.umass.edu, {yuqing, anna}@math.umass.edu; [†]enhui.huang@polytechnique.edu

ABSTRACT
In the Big Data era we are faced with an increasing gap between the fast growth of data and the limited human ability to comprehend data. Consequently, there has been a growing demand of data management tools that can bridge this gap and help the user retrieve high-value content from data more effectively. In this work, we aim to build interactive data exploration as a new database service, using an approach called "explore-by-example". To build an effective system, our work is grounded in a rigorous SVM-based active learning framework. In this framework, we introduce uncertainty sampling for database exploration, propose new sampling algorithms with formal results, and a series of optimizations to improve performance. Evaluation results using real-world SDSS data and query patterns show that our system significantly outperforms state-of-the-art systems in accuracy while achieving desired efficiency for interactive exploration.

1. INTRODUCTION
Today data is being generated at an unprecedented rate, so much that 90% of the data in the world has been created in the past two years[1]. However, the human ability to comprehend data remains as limited as before. As such, the Big Data era is presenting us with an increasing gap between the growth of data and the human ability to comprehend data. Consequently, there has been a growing demand of data management tools that can bridge this gap and help the user retrieve high-value content from data more effectively.

To respond to such needs, we build a new database service for interactive exploration in a framework called "*explore-by-example*". In this service, the database system requests user feedback on strategically collected database samples through a series of "conversations" (or iterations). In each iteration, the user characterizes a database sample as relevant or irrelevant to her interest. The user feedback is incorporated into the system to build a *user interest model*. The model is then used in the next iteration to steer the user towards a new area in the data space, and further improved using the user label of a new sample from that area. Eventually, the model characterizing the relevant objects is turned into a *user interest query* that will retrieve all relevant objects from the database. This service can be used to support two types of applications:

Ad-Hoc Exploration: Consider an example that a novice scientist comes to explore a large sky survey database such as SDSS [51]. She may not be able to express her data interest precisely. Instead, she may prefer to navigate through a region of the sky, see a few samples of sky objects, provide yes or no feedback, and ask the database system to find more objects relevant to her interest. Such Ad-hoc exploration tasks are constrained by the amount of feedback that a user is willing to provide. Instead of requiring 100% accuracy of the user interest model, they often prefer a quick improvement of the accuracy with the first few dozens of samples that the user has reviewed.

Precise Exploration: In this setting, the user is engaged in a long-term conversation with the database system with explicit or implicit feedback on database objects. *Systematic reviews* are an example of a comprehensive assessment of the totality of evidence to address a question such as the effect of a treatment at a given time on mortality. Such reviews involve repeated querying of the clinical trial database and classifying the retrieved trials as relevant or not; some may take a year to complete. System-aided exploration that records user-reviewed trials continually may derive a model of the relevant trials more quickly than manual exploration by the user. Another example is *intelligent personal assistant* like Google Now and other social recommender systems (e.g., [11]). Here, the system works with the user continually by recommending social events and implicitly recording the user feedback (e.g., based on whether the user clicks on the recommended event). Over time, an explore-by-example service can learn the user interest with high accuracy.

Across both applications the performance goals of explore-by -example include: (a) *Accuracy*: the system must maximize the accuracy of the user interest model with a limited amount of user feedback, or minimize the total user feedback needed to achieve a high accuracy level. (b) *Interactive performance*: the time cost in each iteration of exploration must be kept under a few seconds as the user may be waiting online for the next sample to review.

To develop an effective system, our approach casts the learning of the user interest model in each iteration as a *classification* problem, which uses all the user labeled samples (through explicit or implicit feedback) thus far to train a classification model. This is usually quick given the small size of the labeled sample set. However, the challenge lies in the choice of a new sample, across all unlabeled objects in the database, for the user to label next – this choice affects the models developed in the subsequent iterations and how well the system achieves the above two goals. To address the challenge, we introduce **uncertainty sampling for database exploration**, which augments the current set of sampling methods (e.g., stratified sampling) for query processing in a database system. This new form of sampling must meet three requirements:

1. *Most uncertain property*: It finds a sample that is the most uncertain with respect to the current user interest model.
2. *Database efficiency*: The retrieval of the most uncertain sample from the database is within interactive performance.
3. *Convergence*: In any iteration, it can return some quantitive or qualitative description of the convergence of the model so the user can make a well-informed decision on termination.

Our work differs from prior work in several aspects. First, the state-of-the-art system on explore-by-example, Aide [16, 17], used

[1]See a recent survey at https://www-01.ibm.com/software/data/bigdata/

decision trees to build a classification model due to the natural descriptive power of the learned model. However, this approach cannot handle complex user interests characterized by non-linear patterns in the data space. Therefore, in this work we choose Support Vector Machines (SVMs) to build the classification model because they can handle both linear and non-linear patterns.

Second, active learning theory [7] has suggested choosing the example closest to the decision boundary of the SVM model in each iteration of exploration. In this work we leverage SVM active learning theory to meet the "most uncertain" requirement of sampling. However, existing implementations do not meet the efficiency requirement: they either scan the entire database [11], which is prohibitively expensive for a large database, or resort to random sampling [7], which cannot strike a balance between accuracy and efficiency. In addition, active learning work does not meet the convergence requirement: most work lacks provable results on accuracy at a given iteration of exploration [52, 46]. Recent theoretical work offers provable bounds on classification errors [10, 19, 23, 24, 25], which treat positive and negative classes equally and hence are not practical for use. It is because the true user interest over a large database often amounts to a highly selective query. To learn a classification model for the query, we must emphasize the errors related to the objects in the positive class, i.e., the answers returned by the query. Consider a user interest query with 1% selectivity. A classifier that classifies all database objects to the negative class has a low error rate of 1%, but fails to return any relevant objects to the user. For this reason, we choose *F1-score* as a proper accuracy measure for database exploration because it emphasizes the accuracy regarding the positive class. Active learning theory lacks formal convergence results for this measure.

In this paper, we develop and optimize uncertainty sampling for database exploration in the explore-by-example framework, which meets all three requirements above. Our contributions include:

1. New sampling algorithms (Section 3): We develop two uncertainty sampling algorithms that provide rigorous yet practical results on the convergence of the user interest model. (1) *Strong convergence*: The first algorithm focuses on a common class of user interest queries that have a convex shape in the data space. For such queries, we propose a novel sampling algorithm, called TSM, which augments SVM active learning theory (developed in the feature space through the kernel method) with a polytope-based partitioning function of the data space. TSM not only reduces user labeling from using active learning theory alone, but also enables us to prove a monotonically increasing lower bound of F1-score. This is the first formal result on the model accuracy based on F1-score, to the best of our knowledge, and enables the system to detect convergence based on the bound. (2) *Weak convergence*: For general query patterns, our second algorithm augments SVM active learning theory with techniques for capturing the model change rate and the trend of this measure over recent iterations. It allows the system to provide a qualitative statement of whether the model has exhibited the trend for convergence.

2. Optimizations (Section 4): We further provide a suite of optimizations for uncertainty sampling. (1) *Database efficiency*: We devise novel techniques, decision-tree based approximation and a solver method, to reduce the time cost of retrieving the most uncertain sample from the database, as well as an optimization to reduce the running time of the final query to retrieve all relevant objects. (2) *High-dimensional exploration*: For exploration in high-dimensional data space, we propose optimizations of gradient boosting regression trees (GBRT) to reduce data dimensionality and to achieve high accuracy using fewer user labeled samples.

4. System evaluation (Section 5): Evaluation using real datasets and queries from Sloan Digital Sky Survey [51]. shows the following results: (1) Our solver method for retrieving the most uncertain sample achieves both high accuracy and low running time, and outperforms the best sampling technique from active learning [7]. (2) For convex queries, our TSM uncertainty sampling algorithm can reduce the number of user labeled samples required, besides a formal bound for F1-score. This means that if the database system offers query templates for the user to choose, then for known convex templates it can choose TSM for the above benefits. (3) For high-dimensional space, our GBRT optimization improves F1-score from nearly 0 without dimensionality reduction, to high F-measure (>0.8), by adaptively choosing the number of relevant features. (4) We further compared our system to two state-of-the-art systems for explore-by-example, Aide [16, 17] and LifeJoin [11]. Our system significantly outperforms these two in accuracy when the user interest involves 4 dimensions or above, considering both ad-hoc exploration (with limited, say 100, user labeled samples) and precise exploration (with up to 500 user labeled samples), while maintaining the per-iteration time within a few seconds and the final retrieval time within a few minutes.

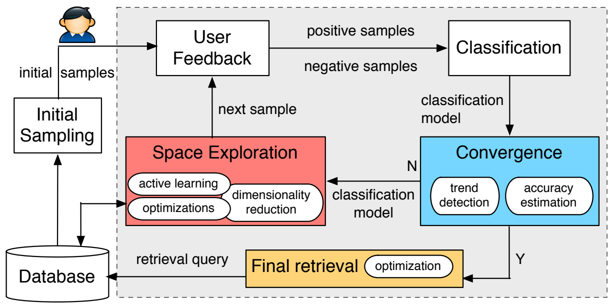


Figure 1: System architecture for explore by example.

2. BACKGROUND

In this section, we begin by reviewing our system designed for example-by-example. We then present background on the SVM classification model and basic active learning theory.

2.1 System Overview

Our data exploration system is depicted in Figure 1. The main concepts and modules are described as follows.

Data space. When a user comes to explore a database, she is presented with the database schema for browsing. Based on her best understanding of the (implicit) exploration goal, she may opt to choose a set of attributes, $\{D_i\}$, $i = 1, \ldots, d$, from a table for consideration[2]. These attributes form a superset of the relevant attributes that will be eventually discovered to characterize the true user interest, but it is the task of the system to discover those relevant attributes. Let us consider the projection of the underlying table to $\{D_i\}$, and pivot the projected table such as each D_i becomes a dimension and the projected tuples are mapped to points in this d-dimensional space – the resulting space is called a *data space* where the user exploration will take place.

Initial examples. To bootstrap data exploration, the user is asked to give an initial positive example and an initial negative example to illustrate her interest. If the user does not have such examples at hand, the system can run an initial sampling algorithm over the data space, as proposed in prior work [16, 32], to help the user find such examples. Since the initial sampling problem has been studied before, our work in this paper focuses on data exploration after such initial examples are identified.

[2]If these attributes come from different base tables, we assume that there is a materialized view that stores the related join results in a single table.

Iterative feedback, learning, and exploration. The iterative exploration process starts with a given positive sample set and a negative sample set, initially each of size one. These samples are called labeled samples. In each iteration, the labeled samples are used as the training set of a classification model that characterizes the user interest (*user interest model*). Before the model reaches convergence or a user-specified accuracy level, the model is used next to navigate the user further in the data space (*space exploration*). In particular, it is used to identify a promising data area to be considered further and to retrieve the next sample from this area to display to the user. In the next iteration, the user labels this sample as positive or negative – such feedback can be collected explicitly through a graphical interface [15], or implicitly based on whether a user clicks on the sample for reviewing or how long she examines the sample.[3] The newly labeled sample is added to an existing labeled sample set, and the above process repeats.

Convergence and final retrieval. At each iteration, our system assesses the current classification model to decide whether more exploration iterations are needed. The process is terminated when the model has exhibited the trend of convergence, or the accuracy of the model reaches a user-defined threshold. At this point, the classification model for its positive class is translated to a query which will retrieve from the database all the objects characterized as relevant. As can be seen, in our work the user interest is characterized by a classification model, which is eventually translated to a database query. Therefore, we use the terms, "*user interest*", "*classification model*", and "*query*", interchangeably.

2.2 SVM and Active Learning

Since non-linear predicates are prevalent in scientific applications and location-based searches, we seek to support user interests involving both linear and non-linear predicates. In this work, we adopt Support Vector Machines (SVM) to build the classification model. SVM is a **linear classifier** that makes a classification decision based on the value of a linear combination of the dimensions of the data space. To support non-linear patterns in the data space, it uses the *kernel method* to map the user labeled samples into a much higher dimensional space, called the *feature space*, where linear separation can be achieved. In this work we use the Gaussian kernel, and denote the decision boundary in the feature space as $\mathcal{L}$. More details on SVM are given in Appendix A.1.

Our problem of dynamically seeking the next sample to label from a large database of unlabeled objects is closely related to active learning. The active learning framework for SVM has a simple structure as shown in Algorithm 1. It starts with initial sampling and proceeds to space exploration in an iterative fashion. In both phases, users are asked to provide labels of retrieved samples (line 2 and line 7). The key focus of active learning is at line 6: to identify at each iteration, the next sample to label to quickly improve the accuracy of the current SVM model. Recent active learning theory [7] proposed to choose in each iteration the *example closest to the current decision boundary*, that is, $min_{\boldsymbol{x}} f(\phi(\boldsymbol{x}), \mathcal{L})$, where x refers to any point in the data space, $\phi(\boldsymbol{x})$ is its mapping to the feature space, and f is the distance function in the feature space.

Our work follows the same framework and uses the above theory to meet the "most uncertain" requirement of uncertainty sampling. However, our work instantiates this framework with new sampling algorithms to also meet the efficiency and convergence requirements. In particular, we propose new algorithms that address `getNextToLabel()` and `isTerminated()` together in

[3]This topic is in the purview of human-computer interaction and hence is beyond the scope of this paper, which focuses on uncertainty sampling.

Algorithm 1 A Basic Framework for SVM Active Learning

```
Input: database $D$
 1: $D_{init}$ ← initialSampling($D$)
 2: $D_{labeled}$ ← getUserLabel($D_{init}$)
 3: $D_{unlabeled} \leftarrow D \setminus D_{init}$
 4: $model$ ← trainSVM($D_{labeled}$)
 5: while !isTerminated() do
 6:     $\boldsymbol{x}$ ← getNextToLabel($model, D_{unlabeled}$)
 7:     $\{\boldsymbol{x}'\}$ ← getUserLabel($\{\boldsymbol{x}\}$)
 8:     $D_{labeled} \leftarrow D_{labeled} \cup \{\boldsymbol{x}'\}$
 9:     $D_{unlabeled} \leftarrow D_{unlabeled} \setminus \{\boldsymbol{x}\}$
10:     $model$ ← trainSVM($D_{labeled}$)
11: finalRetrieval($model$, $D$)
```

Section 3, and a series of optimizations for `getNextToLabel()` and `finalRetrieval()` in Section 4.

3. NEW SAMPLING ALGORITHMS

In this section, we present two uncertainty sampling algorithms that provide rigorous yet practical results on the convergence of the user interest model.

3.1 Algorithm for Convex Queries

Our first algorithm focuses on a common class of user interest queries that have a convex shape in the data space. For this class of queries, we seek to design a sampling algorithm that is more efficient than a direct implementation of SVM active learning theory (Algorithm 1), and further enables formal provable results on F1-score as the accuracy measure of the SVM classification model in a given iteration in exploration.[4]

Formally, F1-score is evaluated on a *test set* $D_{test} = \{(\boldsymbol{x}_i, y_i)\}$, where $\boldsymbol{x}_i$ denotes a database object and y_i denotes its label according to the classification model. Then F1-score is defined as:

$$\text{F1-score} = 2 \cdot \frac{precision \cdot recall}{precision + recall},$$

where *precision* is the fraction of points returned by the model (query on D_{test}) that are positive, and *recall* is the fraction of positive points in D_{test} that are returned by the model.

However, capturing F1-score in our uncertainty sampling procedure is difficult because we do not have such a labeled test, set D_{test}, available. We cannot afford to ask the user to label more to produce one since the user labor is an important concern. The challenge here is how to provide any accuracy information related to F1-score with limited labeled points and abundant unlabeled ones.

The key idea is, at each iteration we try to use all available labeled examples, denoted as $D_{labeled}$, to building a partitioning function of the data space. This function divides the data space into the *positive region* (any point inside which is guaranteed to be positive), the *negative region* (any point inside which is guaranteed to be negative) and the *uncertain region*. We define the *evaluation set* D_{eval} as the projection of D_{test} without labels y_i's. Then for each data point in D_{eval}, depending on which region it falls into, D_{eval} can be partitioned into three sets accordingly, denoted as D^+, D^- and D^u. We can compute a metric from the number of data points in the three sets and prove that it is a lower bound of F1-score evaluated on D_{test}. As more labeled examples being provided, we have more knowledge about the uncertain region, so part of the uncertain region will convert to either the positive or the negative region in

[4]Our solution to this class also provides a foundation for extension to a more general case of a union of convex shapes.

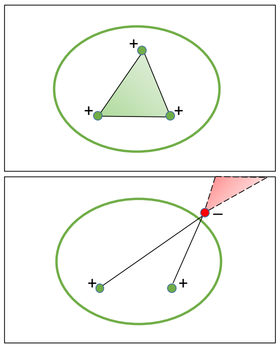


Figure 2: Illustration of a positive region (green) with three positive samples, and a negative region (red) with two positive examples and one negative example in 2D space.

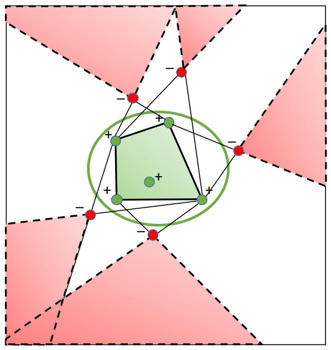


Figure 3: Illustration of positive region (green) and negative region (red) with five positive examples and five negative examples in two-dimensional space.

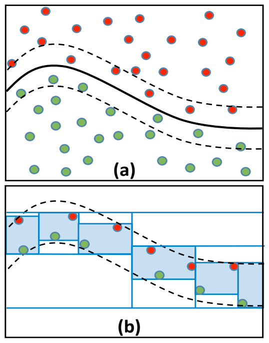


Figure 4: Decision tree based approximation of the non-linear decision boundary in the data space.

later iterations. Accordingly, some data points can be moved from D^u to either D^+ or D^-. Eventually, with enough training data, the uncertain region shrinks to the minimum, $D^u = \emptyset$, and the positive region converges to the query region.

3.1.1 Algorithm with Exact Lower Bound of F1-score

We start with a formal definition of the partitioning function, Ψ, of a data space based on the three regions mentioned above. We then introduce a metric built on this partitioning function, called the three-set metric (TSM), and an algorithm that incorporates Ψ and the TSM metric to active learning. We finally present a theorem stating that the TSM metric captures the lower bound of F1-score.

1. Partitioning Function and Metric. The following definitions and propositions depend on the assumption that the query region Q is convex, which means that any point on the line segment connecting two points $\boldsymbol{x}_1 \in Q$ and $\boldsymbol{x}_2 \in Q$ is also in Q. When Q consists of multiple disjoint convex regions which makes itself not convex, we can initiate one exploration task per region and the metric applies to each task.

Definition 3.1 (Positive Region) *Denote the examples that have been labeled as "positive" as e_i^+, $i = 1, \dots, n^+$. The convex hull of $\cup_{i=1}^{n^+} e_i^+$, i.e., the smallest convex set that contains $\cup_{i=1}^{n^+} e_i^+$, is called the positive region, denoted as R^+.*

It is known that the convex hull of a finite number of points is a convex polytope [22]. For example, the green triangle in Fig. 2 and the green pentagon in Fig. 3 are the positive regions formed by three and five positive examples, respectively, in a two-dimensional space. We can prove the following property of the positive region:

Proposition 3.1 *All points in the positive region R^+ are positive.*

All proofs in this section are deferred to Appendix B. due to space constraints.

Definition 3.2 (Negative Region) *For a negative example $\mathbf{e}_i^-$, we can define a corresponding negative region R_i^- such that the line segment connecting any point $\boldsymbol{x} \in R_i^-$ and $\mathbf{e}_i^-$ does not overlap with the positive region R^+, but the ray that starts from $\boldsymbol{x} \in R_i^-$ and passes through $\mathbf{e}_i^-$ will overlap with R^+. More formally, $R_i^- = \{\boldsymbol{x} | \overline{\boldsymbol{x}\boldsymbol{e}_i^-} \cap R^+ = \emptyset \wedge \overrightarrow{\boldsymbol{x}\boldsymbol{e}_i^-} \cap R^+ \neq \emptyset\}$. Given n^- negative examples, the negative region R^- is the union of the negative region for each negative example, i.e., $R^- = \cup_{i=1}^{n^-} R_i^-$.*

From the definition, we know that R_i^- is a convex cone generated by the conical combination of the vectors from the positive examples to the given negative example, i.e., $\overrightarrow{e_j^+ e_i^-}$ $(j = 1, \dots, n^+)$. Further constrained by the bounds of the data space, each negative region becomes a convex polytope. For example, the red triangle in Fig. 2 and five red polygons in Fig. 3 are the negative regions in a two-dimensional space. We can prove the following property of the negative region:

Proposition 3.2 *All points in the negative region R^- are negative.*

Definition 3.3 (Uncertain Region) *Denote the data space as $\mathbb{R}^d$, the uncertain region $R^u = \mathbb{R}^d - R^+ - R^-$.*

Basically R^u is the remaining region, e.g., the white area in Fig. 2 and Fig. 3. As mentioned earlier, as more examples being labeled, part of the uncertain region will be converted to either the positive region or the negative region and eventually the uncertain region will shrink to an empty set.

With the three types of regions defined, we can define the three-set metric as follows:

Definition 3.4 (Three-set Metric) *Denote $D^+ = D_{eval} \cap R^+$, $D^- = D_{eval} \cap R^-$, $D^u = D_{eval} \cap R^u$, and $|S|$ means the size of set S. At a specific iteration of exploration, the three-set metric is defined to be $\frac{|D^+|}{|D^+|+|D^u|}$.*

2. Algorithm. Next we present how to build the TSM metric in the iterative exploration process in Algorithm 2. The input is the database D, evaluating dataset D_{eval} (which could be any unlabeled dataset including the database D), and a user-defined accuracy threshold λ. First, we initialize R^+ and R^- as empty sets (line 1). Then, initial sampling is performed (line 2) and the obtained examples are labeled by users (line 3). We keep track of the labeled and unlabeled samples using $D_{labeled}$ and $D_{unlabeled}$. Based on the labeled examples, the regions can be incrementally updated (line 5-6) and hence the corresponding sub-partitions of D_{eval} are updated (line 7). The accuracy is then evaluated according to Definition 3.4 (line 8) and a model is trained based on the labeled initial samples (line 9). The process next goes into the iterative exploration until the accuracy requirement is met or the user decides to stop the process (line 10). In each iteration, an unlabeled data point with the closest distance to the decision boundary is acquired. If the data point is in R^+ or R^-, which means its label is known without involving the user, it is labeled automatically (line

Algorithm 2 TSM Sampling Algorithm for Convex Queries

```
Input: database $D$, evaluation dataset $D_{eval}$, user accuracy requirement $\lambda$.
 1: $R^+ \leftarrow \emptyset$, $R^- \leftarrow \emptyset$
    // initial sampling:
 2: $D_{init} \leftarrow$ initialSampling($D$)
 3: $D_{labeled} \leftarrow$ getUserLabel($D_{init}$)
 4: $D_{unlabeled} \leftarrow D \setminus D_{init}$
    // estimate accuracy of initial sampling:
 5: for $\boldsymbol{x} \in D_{labeled}$ do
 6:     $(R^+, R^-) \leftarrow$ updateRegion($R^+, R^-, \boldsymbol{x}$)
 7: $(D^+, D^-, D^u) \leftarrow$ updateEvalData($R^+, R^-, \emptyset, \emptyset, D_{eval}$)
 8: $accu \leftarrow$ estAccuracy($D^+, D^-, D^u$)
 9: $model \leftarrow$ train($D_{labeled}$)
    // space exploration:
10: while $accu < \lambda$ and !isTerminated() do
11:     $\boldsymbol{x} \leftarrow$ getNextToLabel($model, D_{unlabeled}$)
12:     if $\boldsymbol{x} \in R^+$ then
13:         $\boldsymbol{x}' \leftarrow (\boldsymbol{x}, 1)$
14:     else if $\boldsymbol{x} \in R^-$ then
15:         $\boldsymbol{x}' \leftarrow (\boldsymbol{x}, -1)$
16:     else
17:         $\{\boldsymbol{x}'\} \leftarrow$getUserLabel($\{\boldsymbol{x}\}$)
            // estimate current accuracy:
18:         $(R^+, R^-) \leftarrow$ updateRegion($R^+, R^-, \boldsymbol{x}$)
19:         $(D^+, D^-, D^u) \leftarrow$ updateEvalData($R^+, R^-, D^+, D^-, D^u$)
20:         $accu \leftarrow$ estAccuracy($D^+, D^-, D^u$)
21:     $D_{labeled} \leftarrow D_{labeled} \cup \{\boldsymbol{x}'\}$
22:     $D_{unlabeled} \leftarrow D_{unlabeled} \setminus \{\boldsymbol{x}\}$
23:     $model \leftarrow$ train($D_{labeled}$)
24: return $model$
```

12-15). Such examples do not add information to R^+ and R^- and hence do not change D^+, D^-, D^u and the three-set metric. Otherwise, the data point is labeled by the user (line 17) and the metric is updated (line 18-20). At the end of each iteration, the selected data point is moved from $D_{unlabeled}$ to $D_{labeled}$ (line 21-22) and a new model is trained (line 23).

There are a few procedures involved in Algorithm 2. Here we emphasize those related to the three-set metric: In `updateRegion`, the regions are updated based on the literature in computational geometry [3]. In `updateEvalData`, the partitions of D_{eval} can be incrementally updated as shown in Algorithm 3, which tests if a data example belongs to the positive or negative region based on its polytope definition. In `estAccuracy`, the metric is computed according to Definition 3.4.

3. Lower Bound of F1-score. With a good understanding of how the metric works in the exploration process, we now present our main theorem.

Theorem 3.1 *The three-set metric evaluated on D_{eval} captures a lower bound of the F1-score if evaluated on D_{test}.*

The three-set metric has several key advantages. First, it provides an *exact lower bound* of F1-score throughout the exploration process and works for any evaluation set D_{eval} Second, the metric is *monotonic* in the sense that points in the uncertain region before may be in the positive or negative region later, and the metric converges to 1 when $D^u = \emptyset$. The monotonicity means that if the metric is above the desired accuracy threshold at some iteration, it is guaranteed to be greater than the threshold in later iterations, so we can safely stop the exploration. Monotonicity also enables incremental computation: at iteration $i+1$, we only need to check the points in the uncertain region at iteration i and see if they belong to the positive or negative region of iteration $i+1$.

Algorithm 3 `updateEvalData` Incrementally update the partitions of D_{eval}

```
Input: positive and negative region $(R^+, R^-)$. positive, negative and uncertain class of the evaluation dataset $(D^+, D^-, D^u)$.
1: for $\boldsymbol{x} \in D^u$ do
2:     if $\boldsymbol{x} \in R^+$ then
3:         $D^+ \leftarrow D^+ \cup \{\boldsymbol{x}\}$, $D^u \leftarrow D^u \setminus \{\boldsymbol{x}\}$
4:     else if $\boldsymbol{x} \in R^-$ then
5:         $D^- \leftarrow D^- \cup \{\boldsymbol{x}\}$, $D^u \leftarrow D^u \setminus \{\boldsymbol{x}\}$
6: return $(D^+, D^-, D^u)$
```

Besides the lower-bound on F1-score, the TSM algorithm has several advantages over a direct implementation of active learning theory (Algorithm 1): It can automatically label any example in the positive and negative regions, including those closest to a given SVM decision boundary and hence selected by active learning theory for the user to label. As such, it can reduce the user labeling effort. In addition, since TSM works for any evaluation set D_{eval} in the data space, which can be set to the entire database or a smaller sample set (used in our approximate bound as described shortly).

3.1.2 *Approximate Lower Bound of the Metric*

When D_{eval} is too large, we may refer to a sampling technique to reduce the time to evaluate the three-set metric. Let p and q be the true proportions of the positive and negative data in D_{eval}, i.e., $p = |D^+|/|D_{eval}|$ and $q = |D^-|/|D_{eval}|$. Then the three-set metric is $b = \dfrac{p}{1-q}$. Let $\hat{p}$ and $\hat{q}$ be the observed proportions of the positive and negative samples in a random draw of n samples from D_{eval}, and let $X_n = \dfrac{\hat{p}}{1-\hat{q}}$. Our goal is to find the smallest sample size n such that the estimation error of the exact three-set metric is less than δ with probability no less than λ. That is,

$$\Pr(|X_n - b| < \delta) \geq \lambda.$$

The following theorem will help us find the lower bound of n.

Theorem 3.2 $sup_\epsilon \| \Pr(\sqrt{n}|X_n - b| < \epsilon) - \left(2\Phi(\dfrac{\epsilon(1-q)}{\sqrt{p(1-p-q)}}) - 1\right) \|$ $= O(1/\sqrt{n})$ *for any ϵ, where Φ is the cumulative distribution function of the standard Normal distribution.*

With the above theorem, we approximate the sample size such that

$$2\Phi(\frac{\sqrt{n}\delta(1-q)}{\sqrt{p(1-p-q)}}) - 1 \geq \lambda.$$

Since $p(1-p-q)/(1-q)^2 \leq 1/4$, it is sufficient for n to satisfy $2\Phi(2\sqrt{n}\delta) - 1 \geq \lambda$ and therefore $n \geq \left(\Phi^{-1}\left(\frac{\lambda+1}{2}\right)\right)^2 / (4\delta^2)$.

3.2 Algorithm for General Queries

For general query patterns, our second algorithm augments SVM active learning theory with techniques for capturing the model change rate and the trend of convergence based on this measure

To begin the discussion, we summarize our notations for SVM as follows, while the details on SVM are given in Appendix A.1: denote $\boldsymbol{x}$ as a point in the data space, ϕ as the mapping function from the data space to the feature space, K as the kernel function where $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$; the SVM model can be uniquely identified using either $\boldsymbol{\omega}$ and b in the primal form, or $\boldsymbol{\alpha}$ and b in the dual form.

To capture the model change rate, the starting point of our algorithm design is the following observation: with the Gaussian kernel, all the points in the data space $\mathbb{R}^d$ will be mapped onto a unit hypersphere in the feature space $\mathbb{R}^f$. This is because $K_G(\boldsymbol{x}, \boldsymbol{x}) = \exp(-\gamma||\boldsymbol{x} - \boldsymbol{x}||^2) = \exp(0) = 1$. Then for the corresponding ϕ_G, we know $\phi_G(\boldsymbol{x})^T \phi_G(\boldsymbol{x}) = 1$, which means all points in the data space are mapped to a unit hypersphere in the feature space. The decision model at each iteration is a hyperplane that possibly cuts the multi-dimensional ball into one positive hyper-spherical cap and one negative hyper-spherical cap. Fig. **??** shows two decision models in the feature space in solid circles and the corresponding two $\boldsymbol{\omega}$ vectors pointing to the positive side of the model. Below, we formally define the model change rate in the feature space.

Definition 3.5 (Model Change Rate) *Denote C_{i-1} and C_i as the positive hyper-spherical caps at iteration $i-1$ and i respectively. We define the difference between C_{i-1} and C_i as $\mathcal{D}_i = (C_i \setminus C_{i-1}) \cup (C_{i-1} \setminus C_i)$ and the model change rate between iteration $i-1$ and i as $A(\mathcal{D}_i)/A(\mathcal{S}^i)$, where $\mathcal{S}^i$ is a unit sphere in i-dimensional space and A is the notation for the surface area.*

The rationale behind this definition is: $C_i \setminus C_{i-1}$ is the set of points in the feature space that are predicted as negative at iteration $i-1$ but positive at iteration i, and similarly, $C_{i-1} \setminus C_i$ is the set of points that are predicted as positive at iteration $i-1$ but negative at iteration i. Since all points in the data space are mapped to a hypersphere, we use the surface area as the metric.

Closed-form expressions for the surface area of a hyper-spherical cap and that of a hypersphere have been well-studied long ago. Recently, [30] has derived the surface area of the intersection of two hyper-spherical caps given $\boldsymbol{\omega}_{i-1}$ and b_{i-1} and $\boldsymbol{\omega}_i$, b_i. It is straightforward to see that $A(\mathcal{D}_i) = A(C_{i-1}) + A(C_i) - 2A(C_{i-1} \cap C_i)$. Therefore, in order to compute the model change rate in the feature space, the remaining problem is to obtain $\boldsymbol{\omega}$ and b.

As mentioned earlier in Section A.1, SVM is usually solved in the dual form rather than the primal form. A typical SVM solver returns α's and b as an SVM model, but Equation (8) allows us to compute $\boldsymbol{\omega}$ from α_i's and $\phi(\boldsymbol{x}_i)$'s. Although $\phi(\boldsymbol{x})$ is intensionally bypassed by the kernel trick, we can apply *Cholesky Decomposition* to the kernel matrix, which is symmetric and strictly positive-definite for Gaussian kernel, and get a unique decomposition, as formally described below.

$$
\begin{aligned}
\mathbf{K}_{i\times i} &= \begin{pmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{x}_i) \\ \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_i, \boldsymbol{x}_1) & \cdots & K(\boldsymbol{x}_i, \boldsymbol{x}_i) \end{pmatrix} \\
&= \begin{pmatrix} \phi(\boldsymbol{x}_1)^T \phi(\boldsymbol{x}_1) & \cdots & \phi(\boldsymbol{x}_1)^T \phi(\boldsymbol{x}_i) \\ \vdots & \ddots & \vdots \\ \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_1) & \cdots & \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_i) \end{pmatrix} \\
&= \begin{pmatrix} \phi(\boldsymbol{x}_1)^T \\ \vdots \\ \phi(\boldsymbol{x}_i)^T \end{pmatrix}_{i\times i} \times \big(\phi(\boldsymbol{x}_1), \cdots, \phi(\boldsymbol{x}_i)\big)_{i\times i}
\end{aligned}
$$

Note that the kernel matrix of the Gaussian kernel always has full rank for distinct examples, which means that the rank is increased by 1 every time a new example $\boldsymbol{x}_i$ is added and its projection and $\phi(\boldsymbol{x}_i)$ is independent of all previous examples' projections $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_{i-1})$ in the feature space. In other words, each example adds a new dimension to the span[5] of example's projections. When computing the surface area difference of two consecutive models $\boldsymbol{\omega}_{i-1}$ and $\boldsymbol{\omega}_i$, we add a zero to $\boldsymbol{\omega}_{i-1}$ to make it of the same dimensionality with $\boldsymbol{\omega}_i$.

Below we summarize the procedure of computing the model change rate in the feature space at iteration i in Algorithm 4.

Algorithm 4 Model Change Rate in the Feature Space

```
Input: labeled examples $x_1, \ldots, x_i$ and their coefficients $\alpha_1, \ldots, \alpha_i$[6],
       $b_i$, $\omega_{i-1}$ and $b_{i-1}$.
Output: the model change rate in the feature space at iteration $i$
 1: $\mathbf{K}_{i\times i}$ ←updateKernelMatrix($\mathbf{K}_{(i-1)\times(i-1)}$, $x_i$)
 2: $\{\phi(x_1), \ldots, \phi(x_i)\}$ ←CholeskyDecomposition($\mathbf{K}_{i\times i}$)
 3: $\omega_i \leftarrow \sum_{j=1}^{i} \alpha_j \phi(x_j)$
 4: $\omega_{i-1}$ ←append($\omega_{i-1}$, 0)
    // computations below are in $i$-dimensional space:
 5: $intersect$ ←getIntersectionArea($\omega_{i-1}$, $b_{i-1}$, $\omega_i$, $b_i$)
 6: $area_{i-1}$ ←getUnitSphericalCapArea($i$, $\omega_{i-1}$, $b_{i-1}$)
 7: $area_i$ ←getUnitSphericalCapArea($i$, $\omega_i$, $b_i$)
 8: $change \leftarrow area_{i-1} + area_i - 2 \cdot intersect$
 9: $whole$ ←getUnitSphereArea($i$)
10: return $change/whole$
```

After we obtain the model change rate, we can just check convergence using the long-established methods.

4. OPTIMIZATIONS

We further provide a suite of optimizations for uncertainty sampling, including those for reducing the time cost of retrieving the most uncertain sample from the database, for reducing the time of running the final query over the database, and for reducing the user labeled samples needed for high-dimensional exploration.

4.1 Sample Retrieval

A key performance goal in our work is to limit the cost of each iteration, including retrieving the most uncertain sample to label next, within a few seconds. Recent research [7] proposed to choose the sample closest to the current decision boundary of the SVM. However, finding the sample closest to the decision boundary from a large database is costly. Pre-computation to store the distance of each tuple to the decision boundary is not possible because the boundary changes in each iteration. As a result, existing implementations either pay the cost to scan the database, or sacrifice convergence by resorting to random sampling and among the random samples, choosing the one closest to the decision boundary.

Optimizing the retrieval of the most uncertain sample is challenging because the sample closest to the decision boundary is defined in the feature space, while sample retrieval is performed in the data space. There is no reverse mapping from the feature space to the data space. Below, we propose two optimizations for the sample retrieval problem without scanning the entire database.

Decision Tree based Approximation. The classification boundary in the data space can be of arbitrary shapes, hence hard to be interpreted but can be approximated by the union of many disjunctive hyper-rectangles. As shown in Fig. 4(a), the classification boundary is a solid black curve that cuts the data space into two regions: one filled with green positive points and one with red negative points. The black curve can be approximated by the blue boxes in Fig. 4(b). Each hyper-rectangle is a conjunction of conditions on (a subset of)

[5]It is well-known that the feature space of Gaussian kernel is infinite dimensional but the projections of finite examples span a finite dimensional subspace of the infinite dimensional space.

Algorithm 5 Decision Tree Based Approximation

```
1: Define a band that encloses the current SVM decision boundary.
2: Prepare a training dataset of synthetic grid points such that the points
   in the band are positive and otherwise are negative.
3: Feed the dataset to a decision-tree algorithm and train a decision tree.
4: Translate the leaves returned by the decision tree into a SQL query and
   send it to the backend database.
5: From all the results of the query, return the one that is closest to the
   SVM decision boundary as the sample to be used in the next iteration.
```

dimensions in the data space, just like the leaves returned by the decision tree learning algorithm. This inspires us to leverage decision trees to approximate the classification boundary in the data space.

We summarize the main steps in Algorithm 5. Let us define the decision boundary (function) in the feature space as:

$$y(\boldsymbol{x}) = \boldsymbol{\omega}^T \phi(\boldsymbol{x}) + b = 0 \tag{1}$$

where $\boldsymbol{x}$ as a point in the data space, ϕ as the mapping function from the data space to the feature space, and $\boldsymbol{\omega}$ is the weight vector.

In step 1, we define a band in the data space as $\{\boldsymbol{x}|\ y(\boldsymbol{x}) \in [-\delta, \delta]\}$. In step 2, we enumerate synthetic grid points in the data space to find those residing in the band (positive points) and outside the band (negative points). In Step 3, we feed these points to build a decision tree on the data space dimensions. We then turn the decision tree to a database query (step 4) and among its output, find the exmple closest to the SVM decision boundary (step 5).

Solver Method. The solver method expedites the most uncertain sample retrieval by first finding a point on the decision boundary through a solver of the boundary condition, $y(\boldsymbol{x}) = 0$, and then a database example locally near this point.

Given two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ such that $y(\boldsymbol{x}_1) > 0$ and $y(\boldsymbol{x}_2) < 0$, we regard $f(t) = y(t\boldsymbol{x}_1 + (1-t)\boldsymbol{x}_2)$ as a function of t on $[0, 1]$. Then we have $f(0) < 0$ and $f(1) > 0$. The Intermediate Value Theorem (IVT) [2] states that if f is continuous on a closed interval $[a, b]$, and c is any number between $f(a)$ and $f(b)$ inclusive, then there is at least one number x in the closed interval such that $f(x) = c$. According to the IVT, since the decision function as a function of t on $[0, 1]$ is continuous, there exists a t in $(0, 1)$ such that $f(t) = 0$. Solver functions such as FZERO in java can be used to find the zero point, denoted as $\boldsymbol{x}^*$. This point sits right on the current decision boundary but often does not correspond to a real data sample. To present the user a real sample to label, we form a small bounding box around $\boldsymbol{x}^*$ and scan samples in the bounding box to locate the one that is closest to the decision boundary.

Given the current decision boundary of SVM, it is convenient to find two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ such that $y(\boldsymbol{x}_1) > 0$ and $y(\boldsymbol{x}_2) < 0$. However, choices of the two points affects the convergence of the solver method. Ideally, we would like to choose a pair of point in each iteration such that the collection of pairs across iterations explore the data space in all directions. For this purpose and having in mind the positive region is only a small fraction of the data space, we choose any point $\boldsymbol{x}_1$ such that $y(\boldsymbol{x}_1) > 0$ and choose consecutively data space boundary (outmost) points as $\boldsymbol{x}_2$ such that $y(\boldsymbol{x}_2) < 0$. If an outmost point $\boldsymbol{x}_2$ has $y(\boldsymbol{x}_2) > 0$, we do not abandon that direction of search along the line connecting $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$; rather, we keep bisecting the segment between the two points, until we found a negative y value. If this fails, we locate the minimum of f on the segment using grid points at which the bounding box will be formed for the database query.

4.2 Final Result Retrieval

Once the exploration terminates upon convergence or by the user request, we obtain an SVM model as represented in Eq. 1. The ultimate goal is to find all tuples in the database D with positive predictions by this model, i.e., $\boldsymbol{x}$ such that $\boldsymbol{x} \in D$ and $y(\boldsymbol{x}) > 0$. To expedite the retrieval of the final results, we propose to build R-tree as the index over the database, and perform a top-down search in a depth-first fashion.

Branch and Bound. The unique aspect of the R-tree search is a solver-based branch and bound approach. Each R-tree node offers a hyper-rectangle as a minimum bounding box of all the data points reachable from this node. With an explicit decision function, $y(\boldsymbol{x})$ in Eq. 1, we can compute an upper bound of $y(\boldsymbol{x})$ without visiting the descendent nodes. Instead, we can obtain it by solving the following constraint optimization problem.

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & y(\boldsymbol{x}) \\ \text{s.t.} \quad & a_j \le \boldsymbol{x}^{(j)} \le b_j,\ j = 1, \dots, d. \end{aligned} \tag{2}$$

where $[a_j, b_j]$ is the range of the tree node on the j-th dimension and $\boldsymbol{x}^{(j)}$ is the value of $\boldsymbol{x}$ on the j-th dimension. If the upper bound is already smaller than 0, we can prune the entire subtree rooted at this node. Since the positive results tend to be clustered and mark only a small portion of the database in practice, with such index structure, we may gain a significant improvement over scanning the entire database and running the model on each tuple.

4.3 High-Dimensional Exploration

As the dimensionality of the data space increases, the convergence rate of the SVM model degrades fast. This is first due to the fact that data exploration is fundamentally constrained by the available labeled samples. Furthermore, higher-dimensionality means higher volume of the space and the limited set of labeled samples become even more sparse in higher-dimensional space, making it very difficult for SVM to determine the linear hyperplane. In the discussion below, we focus on the case that the true user interest lies in a low-dimensional space, but the user initially selects a larger set of attributes to begin data exploration. It is likely to occur when users do not feel confident in pruning attributes initially, but some of these attributes will be eventually recognized as irrelevant based on the user feedback on an increasing number of samples.

We begin by examining a range of popular dimension reduction techniques including: 1) Principled Component Analysis (PCA) as a query-agnostic database compression method; 2) Random forests (RF) as an online feature selection method during a user data exploration session; 3) Gradient boosting regression trees (GBRT) as another online feature selection method. These method are described in Appendix C.2 and evaluated in Section 5. Based on our results, GBRT works better than RF and PCA for reducing the number of features or dimensions needed to train the SVM classification model. Therefore, our following discussion focuses on GBRT.

We first outline how Gradient boosting regression trees (GBRT) is used for online feature selection in our work[7]. When a user is interacting with the system, in each iteration we first feed all existing labeled samples to the GBRT learner, and ask the learner to return the top-k features that are deemed most important in the learning process (to be formally defined shortly). Next we build an SVM classification model using only the top-k features and select the next sample to be labeled as the object closest to the current SVM decision boundary. Then we repeat these two steps in the next iteration until the SVM model converges. Although GBRT works better than others for feature selection, we observe two limitations:

Unbalanced training data. GBRT is very sensitive to unbalanced classes, that is, when the training data is dominated by the

[7]Since *feature selection* is the standard term in the literature, we use "features" and "attributes" interchangeably in this context.

Table 1: Query templates (selectivity is reported on 1% dataset with 1,918,287 tuples)

Attributes	Query template	Selectivity
($rowc$,$colc$)	**Q1.1:** $rowc > 662.5$ and $rowc < 702.5$ and $colc > 991.5$ and $colc < 1053.5$	0.1%
	Q1.2: $rowc > 617.5$ and $rowc < 747.5$ and $colc > 925$ and $colc < 1120$	0.97%
	Q1.3: $rowc > 480$ and $rowc < 885$ and $colc > 719$ and $colc < 1326$	9.43%
	Q2.1: $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 29^2$	0.1%
	Q2.2: $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 90^2$	0.98%
	Q2.3: $(rowc - 682.5)^2 + (colc - 1022.5)^2 < 280^2$	9.43%
(ra,dec)	**Q3.1:** $ra > 190$ and $ra < 200$ and $dec > 53$ and $dec < 57$	0.1%
	Q3.2: $ra > 180$ and $ra < 210$ and $dec > 50$ and $dec < 60$	0.95%
	Q3.3: $ra > 150$ and $ra < 240$ and $dec > 40$ and $dec < 70$	10.24%
($rowc$,$colc$) + (ra,dec)	4-dimensional queries as a combination of the above ($rowc$,$colc$) and (ra,dec) queries, e.g., **Q1.2 + Q2.1**	varied
($rowc$,$colc$) + 34 attributes	extend ($rowc$,$colc$) queries with up to 34 irrelevant attributes, including $rowv$, $colv$, ra, dec, u, g, r, i, z , . . ., to test dimensionality reduction	varied

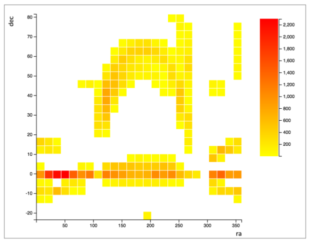


Figure 5: (ra, dec) distribution

samples belonging to the negative class. This is a common case in data exploration because the true user interest is often a highly selective query, retrieving a small fraction of the database. Therefore, the most uncertain sample retrieved is likely to be labeled as negative by the user. In fact, it may take many iterations for us to see the next positive sample. As a result, the training set is severely skewed with most samples being negative. In this case, it will take many labeled samples (iterations) for GBRT to see enough positive samples and recognize the truly relevant attributes as top-k features.

While class imbalance is an inherent property in data exploration, we draw upon two insights in this work to mitigate the imbalance problem. First, when the true user interest has a convex shape, our Three-Set-Metric (TSM) learning algorithm already maintains a list of positive samples that are derived from the polytope-based partitioning function of the data space and known to be correct. We can retrieve positive samples from this set, merge them with the user-labeled samples, and feed the extended, balanced training set to GBRT to improve its performance. Second, before TSM accumulates a sizable positive sample set, we can also boost GBRT using synthetic positive samples: if the user interest has a convex shape, as long as there are two positive samples, we can use linear interpolation to draw points from the line that connects these two samples, and treat these points as synthetic positive samples to make the training data balanced.

How many feature to select? Another issue is that we do not know how many features to select, or the exact value of top-k. The user does not offer this information a priori. Choosing the right value of k is nontrivial because GBRT may not have high confidence for selecting the correct features in many iterations from the start; in some iterations it ranks the correct features above others, but in the next iteration it may change its ranking.

To formalize the notion of confidence, we first review an important concept in the GBRT learning algorithm. As described in [42] GBRT is a gradient boosting machine with decision trees as base-learners. The additive model of GBRT is a linear combination of the decision tree base-learners. Features do not contribute equally to the construction of these base leaners. Usually, only a small fraction of the features contribute significantly. Individual decision trees intrinsically perform feature selection by choosing good features for splitting nodes. The more frequently a feature is used for splitting nodes of a tree, the more important the feature is. For each decision tree, the importance score of a feature is computed as a sum of impurity decreases for all the nodes where the feature is used. Across all the trees, the importance score of a feature X_j for predicting Y is defined by adding up the weighted impurity decreases $p(t)\triangle(t)$ for all nodes t where X_j is used, and averaged over all trees h_m, $m = 1, \ldots, M$, in the forest [8]:

$$\text{Imp}(X_j) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t \in h_m} \mathbf{1}(j_t = j) \cdot p(t)\triangle(t) \qquad (3)$$

where $p(t)$ is the proportion $\frac{N_t}{N}$ of samples reaching node t and j_t denotes the feature used for splitting node t [33]. Summing over all features, we have the **sum of important scores** (SIS):

$$\sum_j \text{Imp}(X_j) = \frac{1}{M} \sum_{m=1}^{M} \left(\sum_j \sum_{t \in h_m} \mathbf{1}(j_t = j) \cdot p(t)\triangle(t) \right) \quad (4)$$

In sklearn implementation, the sum of feature importances within a given tree (the inner two sums in Eq. 4) is normalized to 1 if the tree has more than one node. Otherwise, it is zero for root-only trees, which make random predictions and are not base learners. Therefore, SIS is the same as the proportion of non-root trees.

Given feature importance scores, we propose two strategies to characterize "sufficient confidence" of GBRT for feature selection.

Sum of Importance Scores (SIS): The intuition is that all decision trees must be weak learners and hence find some features useful in distinguishing the positive class from the negative class. Based on this intuition, our first strategy is that as SIS increases to 1, all decision trees become weak leaners and we believe that the confidence of GBRT has reached a sufficient level.

Entropy of Importance Scores (EIS): The intuition here is that we prefer to have a lower entropy of the importance scores across all the features. That is, the distribution of importance scores departs from a uniform distribution and becomes highly concentrated. Based on this, our second strategy is that as EIS drops significantly below the expected entropy of uniform importance scores. In other words, the importance scores of some features start to stand out, and we believe that the confidence of GBRT has been sufficient.

Adaptive Feature Filtering and Selection: We then devise adaptive strategies to decide top-k features to return, depending on whether the current iteration has reached the point of gaining sufficient confidence, based on any of the two above strategies. Before reaching this point, we perform conservative feature filtering to accommodate the uncertainty of GBRT, which selects top-k features that cover 50% of the SIS or whose feature scores sum up to at least 0.5 (two variants). After the point, we perform aggressive feature selection by choosing top-k features that have the largest gap from the bottom features in the ranked list of feature importance scores.

5. EXPERIMENTAL EVALUATION

We have implemented all of our proposed techniques in a Java-based prototype for data exploration, which connects to a PostgreSQL database. In this section, we evaluate our techniques in terms of accuracy (using the F1-score), convergence rate (the number of user labeled samples needed to reach an accuracy level), and efficiency (the execution time in each iteration and in final query retrieval). We also compare our system to two state-of-the-art systems on explore-by-example, Aide [16, 17] and LifeJoin [11], as well as specific sampling methods from active learning [7].

Datasets: We evaluate our techniques using the "PhotoObjAll" table, which contains 510 attributes, from the Sloan Digital Sky Survey (SDSS) with data release 8[8]. For experiment purposes, we generated tables by random sampling the base table with different sampling ratios from 0.03% to 10%. Since data exploration usually operates on a dataset that fits in memory, we used the 1% sampled dataset, which is the largest that fits in memory, as our default data exploration space. It contains 1.9 million tuples and has the size of 9991MB after being loaded to PostgreSQL. Additional indexes are built on the relevant attributes.

User Interest Queries: We extracted a set of queries from the SDSS query release 8 to represent true user interests[9]. They allow us to run *simulations* of user exploration sessions using the same approach as [16, 17]: We precompute the answer set of each query as the proxy of the user. We then run a data exploration session as described in Section 2; during each iteration, when our uncertainty algorithm presents a new sample to be labeled, the simulator consults the proxy to decide to give a positive or negative label.

When choosing queries in our experiments, we consider the following factors : (1) *pattern*: query predicates can be linear or nonlinear, (2) varied query *selectivities*, and (3) varied query *dimensionalities*. The queries are summarized in Table 1. Regarding the attribute choices, ($rowc$, $colc$) represent the row and column center positions, and have roughly evenly distributed data; (ra, dec) are the right-ascension and declination in the spherical coordinate system, and present represent skewed data (see Figure 5). We also combine queries on ($rowc$, $colc$) and (ra, dec) to obtain 4-dimensional queries with varied selectivities, We further add a varied number of irrelevant attributes that the dimensionality of the exploration space goes up to 36.

Servers: Our experiments were run on five identical servers, each with 12-cores, Intel(R) Xeon(R) CPU E5-2400 0 @2.4GHz, 64GB memory, JVM 1.7.0 on CentOS 6.6.

5.1 Sample Retrieval Methods

We first use the general active learning framework (without making convex assumptions) and evaluate our sample retrieval methods, the decision tree and solver methods (§4.1), for finding the sample closest to the SVM decision boundary in each iteration. For comparison, we also include the best sampling method reported for active learning [7]. This method retrieves a fixed number L of random samples at each iteration and among them chooses the one closest to the decision boundary, denoted as x^*. L is chosen under the condition that x^* is among the top $p\%$ closest instances in the original dataset with probability q. We varied L from 500 to 50,000, whose corresponding $p\%$ and q values are (1%, 0.993) and (0.01%, 0.993). We call these methods "random-top-500" to "random-top-50k". We made consistent observations that for queries with selectivity 1% and 10%, all techniques have marginal difference in terms of accuracy. Therefore, we show the results for queries with 0.1%, considered as harder queries, in Fig. 6.

Expt 1 ($rowc$, $colc$): First consider Fig. 6(b) and 6(c), both for Q2.1 with a nonlinear pattern in Table 1. We can see that random-top-500 is much less accurate compared to random-top-50000, especially in early iterations. This is because for highly-selective queries, it is hard to hit a good and informative sample among a set of only 500 samples. However, random-top-500 only takes 0.25 second per iteration while random-top-50k takes around 25 seconds due to much more samples retrieved from the database per iteration. The accuracy of the decision tree method and the solver method lies in between, closer to random-top-500k. The time per iteration is around 2 seconds for the solver method and does not increase with iterations; the average time of the decision tree method is 2.86 seconds over the first 200 iterations but increases fast after some point. **Overall, our solver method finds the best trade-off between accuracy and response time**. The accuracy trends of various methods for Q1.1 with the linear pattern are very similar, as shown in Fig. 6(a). We omit the time plot for the same reason.

Expt 2 (ra, dec): Since the decision-tree-based method is always observed inferior to the solver method, we omit it in the following experiments. Fig. 6(d) shows the accuracy result of Q3.1, which is on a skewed dataset. The solver method still approximates well random-top-50k, especially in the early iterations, while the response time stays slow, very similar to Fig. 6(c).

Expt 3 ($rowc$, $colc$, ra, dec): We next combine the $rowc$-$colc$ query of 1% selectivity with a ra-dec query of 1% selectivity. As Fig. 6(e) and Fig. 6(f) show, the solver method approximates random-top-50k for accuracy and random-top-500 for response time, hence achieving a good tradeoff between them.

5.2 Uncertainty Sampling for Convex Queries

We next consider the convex properties of queries. We compare our uncertainty sampling algorithm, Algorithm 2, to default active learning work, Algorithm 1, for any given example retrieval method. These algorithms are labeled as "TSM on" and "TSM off" in Fig. 7. The x-axis is the iteration (conversation) number; at each iteration, the user provides a label for one example. The system is automatically terminated after 500 examples labeled either by the user or TSM.

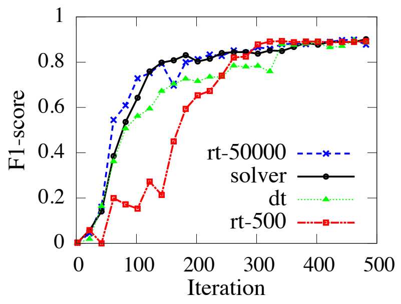
Expt 4: Accuracy for Q2.1 with different example retrieval methods are shown in Fig. 7(a)-7(c). At any iteration, TSM has a better accuracy than the baseline active learning algorithm; the F1-score of the latter does not even reach 0.99 with 500 iterations while TSM only requires around 200-300 iterations to obtain at least 0.99 F1-score. **Overall, with TSM turned on, the user is expected to label much fewer examples while achieving higher accuracy.** The reasons are: (1) TSM keeps track of regions (i.e., the positive and negative regions) where labels are certain based on the existing labeled examples, and only requires the user to label an example if it falls in the uncertain region. As exploration proceeds, TSM requires less user labeling because the uncertain region shrinks over time. (2) SVM itself can make wrong predictions for points in the positive and negative regions, while TSM will not (see Proposition 3.1 and 3.2), which is guaranteed to bring a better accuracy. Accuracy for a four-dimensional query combining Q2.2 and Q3.2 is shown in Fig. 7(d). Compared with Q2.1, the increase in the dimensionality makes it harder to form effective certain regions for TSM, so TSM does not help to label any example retrieved by the solver method in early iterations and the F1-score is almost identical. But eventually the user only needs to label around 400 examples, which is a 20% reduction on the user effort.

Expt 5: We next study the effectiveness of the lower bound given by TSM. We compute both the exact lower bound and the approximate lower bound based on 16588[10] samples. The results on Q2.1 is shown in Fig. 7(e). It can be seen that the blue and red lines are very close and both are indeed lower bounds for the black line. The There is a gap between the truth and the lower bound but eventually the lower bound converge to the true F1-score: when the true

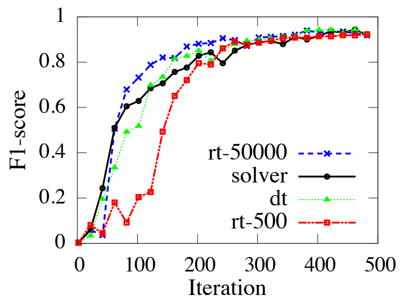
[8]http://www.sdss3.org/dr8/

[9]http://skyserver.sdss.org/dr8/en/help/docs/realquery.asp

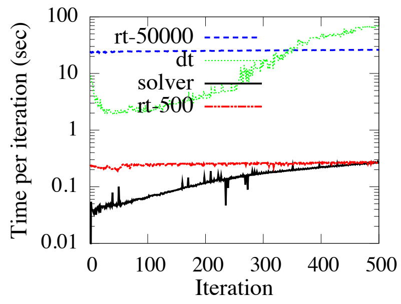
[10]Derived from Theorem 3.2 when $\lambda = 0.99$ and $\delta = 0.01$.



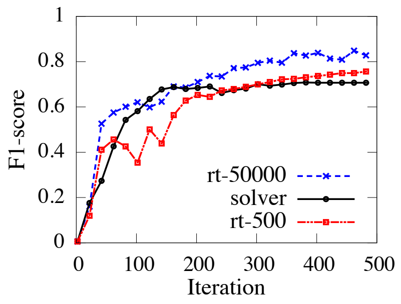
(a) Q1.1 Accuracy



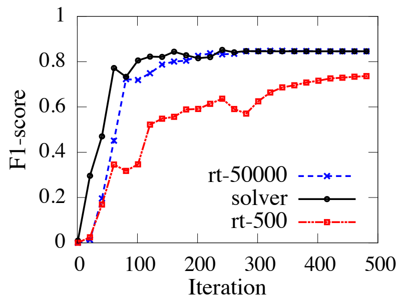
(b) Q2.1 Accuracy



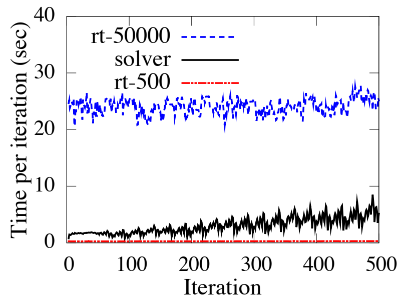
(c) Q2.1 Speed



(d) Q3.1 Accuracy



(e) Q2.2+Q3.2 Accuracy



(f) Q2.2+Q3.2 Speed

Figure 6: The accuracy and response time of various example acquisition methods on various workloads.

F1-score is 0.99, the lower bound is around 0.96. The same observation is made in Q1.1.

Expt 6: The response time per iteration for Q2.1 with TSM turned on and off is shown in Fig. 7(f). With TSM on, we do see overheads. The overhead increases with iteration because as exploration proceeds, TSM will have better knowledge about the query so it can help the user to label more samples and feed the labeled samples to the learning module. We can also see that with the approximation the running time is much reduced without sacrificing the accuracy according to Fig. 7(e).

5.3 Dimensionality Reduction

In this section, we evaluate our techniques for dimensionality reduction. To do so, we fix the true user interest pattern as shown in Table 1, but add irrelevant attributes from the SDSS schema, up to 36 dimensions in total. Below we first report results using Query 2.2 ($rowc$-$colc$, nonlinear, 1% selectivity) with 2 to 34 added irrelevant attributes.

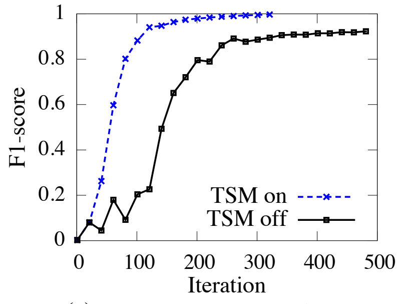
Expt 7: We begin by studying the effect of dimensionality on convergence of SVM-based active learning. As Figure 8(a) shows, the convergence of SVM active learning degrades fast with increased dimensionality. The F-measures remain almost flat at the bottom from 16 dimensions and higher.

Expt 8: We next compare 3 dimensionality reduction methods: PCA as a database compression method, and Random Forests (RF) and Gradient Boosting Regression Trees (GBRT) for online feature selection. To understand the best achievable performance, we fed the true number of relevant attributes to the online feature selection algorithms, and used a full scan of the dataset to find the point closest to the decision boundary in each iteration. Given the slow performance of some algorithms, we used a smaller 66k-tuple dataset. As Figure 9 shows, Our results shown that F-scores of PCA and RF for 36-dimensions stay close to 0, while GBRT can achieve over 95% with 500 samples (if the number of relevant attributes is fed to it).

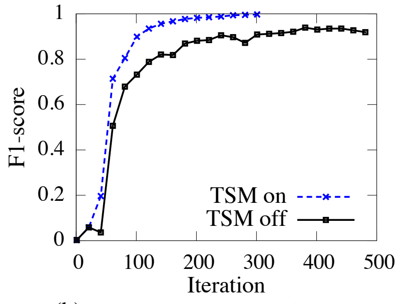
Expt 9: We next use GBRT as an online feature selection technique, which is activated for feature selection from the beginning of exploration. As mentioned, GBRT suffers from the imbalance class problem. We next evaluate our optimization in § 4.3 that feeds GBRT with additional positive samples from our internal TSM model to overcome its class imbalance problem. Our results show that this optimization indeed improves the F-score of GBRT.

In this experiment, we evaluate the effectiveness of our optimization in Section 4.3 that feeds GBRT with additional positive samples from our internal model of the data space to overcome class imbalance. Again we manually set the value of top-k using the number of relevant features, which marks the best achievable performance of GBRT. Figure 8(b) shows F-scores for no feature selection (nofs), GBRT with unbalanced training data (ufs), GBRT with balanced training data (bfs). As can be seen, balanced training data allows GBRT to rise earlier from the initial region of having 0 F-measure. Figure 8(c) shows the scenario that the data exploration starts from two initial positive samples. Then for a convex pattern, the optimization of taking synthetic points from the linear interpolation of the two initial positive samples works from the beginning of data exploration. Therefore, the F-score of GBRT with balanced training data departs from the 0 F-measure region much earlier. This indicates that the database system can encourage the user to provide a few additional initial examples, which can lead to remarkable performance gains. We further show the importance scores of 36 features across iterations. Figures 8(d) and 8(e) correspond to the scores of unbalanced and balanced training sets, respectively. The red and blue curves represent the scores of the relevant attributes, $rowc$ and $colc$. As can be seen, the scores of these attributes become separate from others after 350 iterations using unbalanced training data, while they become separate within 200 iterations using balanced training data.

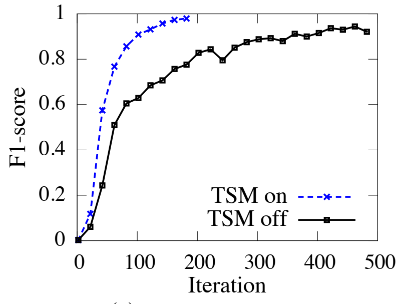
Expt 10: We next evaluate our adaptive strategy to choose top-k features from 36 given features. First, we compare the two strategies for deciding when GBRT has gained sufficient confidence of its ranking of feature importance scores: (1) Sum of Importance Scores (SIS) = 1, and (2) Entropy of Importance Scores (EIS) drops significantly below the expected entropy of uniform importance



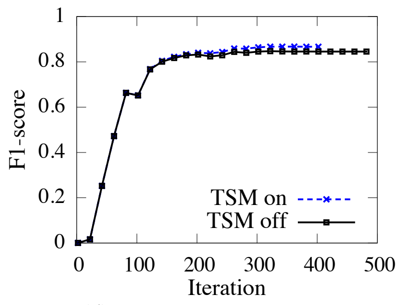
(a) Q2.1 with random-top-500



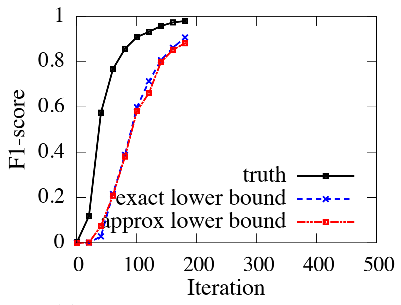
(b) Q2.1 with random-top-50000



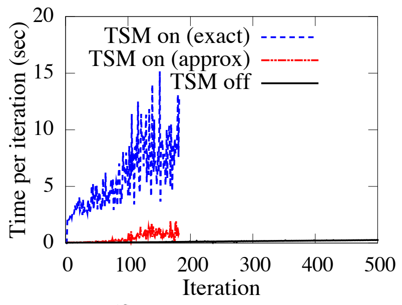
(c) Q2.1 with solver



(d) Q2.2+Q3.2 with solver



(e) Q2.1 with solver and TSM on



(f) Q2.1 with solver

Figure 7: The effectiveness of optimizations on convex queries.

scores. Figure 8(f) shows how these two measures change over iterations of data exploration. We see that after SIS reaches 1, it is quite stable and GBRT soon starts to distinguish the true features from the irrelevant ones. In contrast, EIS does not exhibit a clear trend of reducing entropy and stabilizing at a low level. It is inherently more sensitive to the change of ranking of features. Therefore, we use the first iteration of SIS = 1 as the point that GBRT has gained sufficient confidence. Before this point, we perform conservative feature filtering, but after this point, perform more aggressive feature selection, as described in Section 4.3.

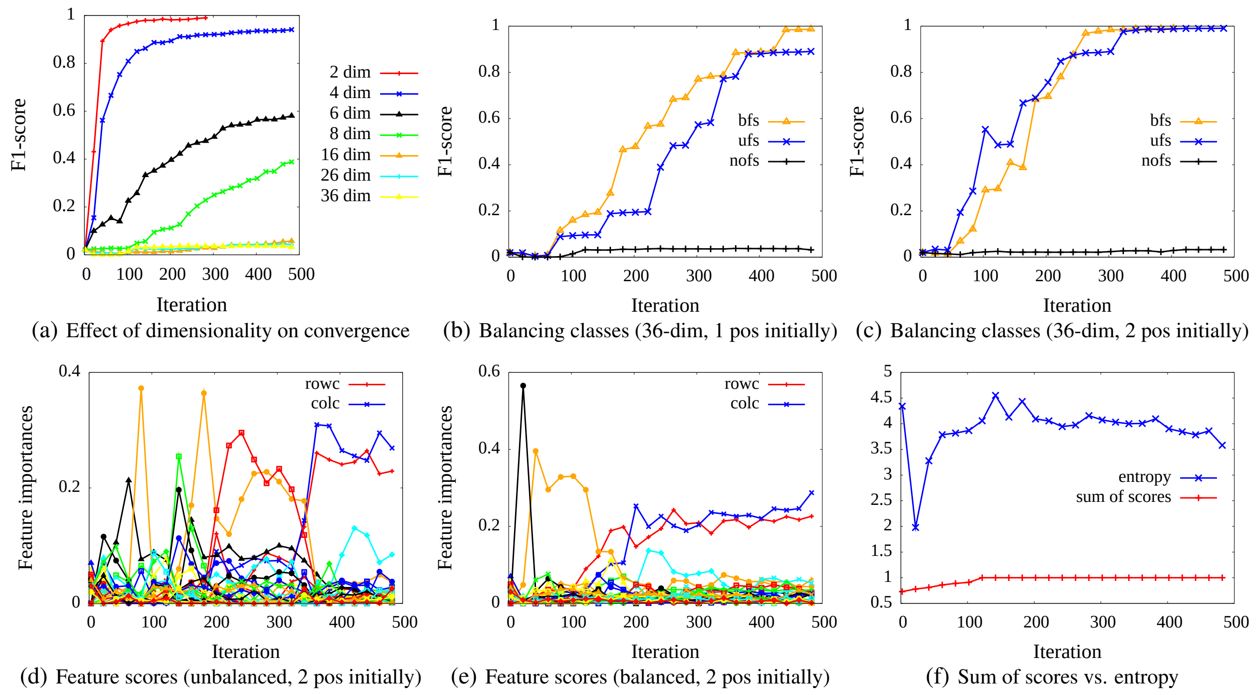
Figure 10(a) compares our adaptive strategy in two versions: during conservative filtering, we select top-k features that cover 50% of the sum of scores (SIS), named 50%-adafs, or whose feature scores sum up to at least 0.5, named 0.5-adafs. As reference, we compare to (1) a simple variant (36dim-adafs) that does not filter features until GBRT becomes confident (when sum of scores reaches 1); (2) the baseline with no feature selection (no-fs); (3) a hypothetical upper bound where we manually set the true top-k (manually-2fs). As can be seen, our adaptive schemes work better than the simple variant, and approximate the upper bound very well, with even better performance in some iterations. In contrast, no feature selection stays at close to 0 accuracy. Figure 10(b) shows the feature importance scores of 0.5-adafs, where the true features start to separate from the rest as early as 100 iterations. In terms of time per iteration, our adaptive top-k increases from no feature selection by a small factor within 1.6. Hence, adaptive feature selection at the cost of a little extra time improves the convergence of no feature selection dramatically.

Finally, Figure 10(c) shows the convergence of adaptive top-k against manually setting k=2 and no feature selection for Q2.1. This query has a lower selectivity, 0.1%, and hence convergence is more difficult. However, our adaptive top-k increases F-measure significantly, and even outperforms manually setting k=2. The reason for the latter is that when GBRT cannot distill right features yet, forcing it to choose only 2 features causes wrong features to be selected, which affects active learning significantly.

5.4 Compare to Alternative Systems

We finally compare our system to two alternative systems for explore-by-example. (1) **Aide** [16, 17] uses decision trees as the classification model. If a query pattern is non-linear, it uses a disjunction of conjunctive linear predicates (or a collection of hyper-rectangles in the data space) to approximate the pattern. We obtained the source code from the authors. (2) **LifeJoin** [11] reports a method, named "hybrid", as its best performing method. At each iteration, this method uses all the labeled samples to train a collection of weak-learners (error-free regarding the training data), extracts basic predicates from these learners, and train a **linear** SVM over these predicates. Then the linear SVM is used to seek the next sample for labeling, which is the one closest to the SVM boundary. The final retrieval method collects the support vectors of the final SVM, uses it as a training set to build a decision tree, and converts the positive class of the decision tree to a query to retrieve all the objects. We reimplemented the LifeJoin with two modifications: we used Random Forest (RF) with overfitted decision trees to build the weak learns as RF is a better known approach than a program synthesizer for this purpose, and we used our sample retrieval method to find the one closest to the SVM boundary, avoiding scanning the entire dataset. We tried to make parameters consistent with those recommended in the paper, including the number of weak learners used (10) and the number of basic features (on the order of hundreds). We run all experiments up to 500 user-labeled samples.

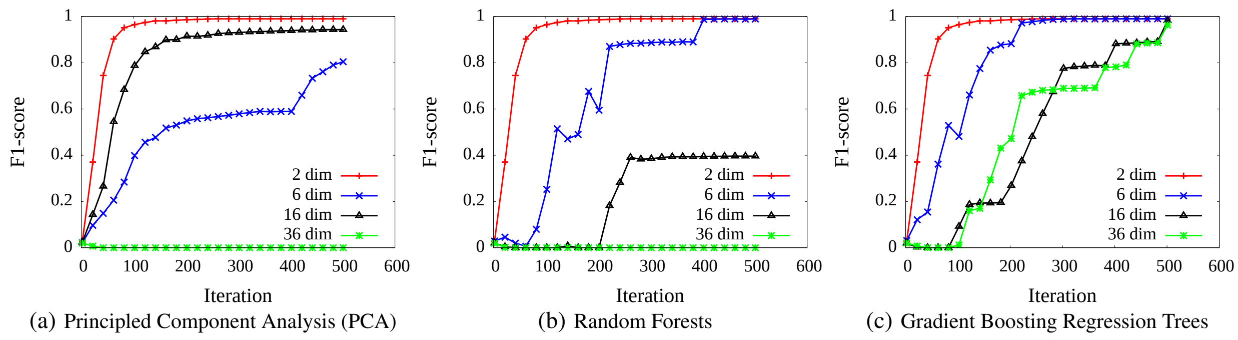
Fig. 11 shows the results for four workloads. The main observations are: (1) For the 2D linear query (Q1.1), three systems are similar in accuracy. For the 2D nonlinear query (Q2.1), our system and Aide are similar, while LifeJoin is significantly worse in both accuracy and per-iteration time. (2) For the 4D query that combines linear and nonlinear predicates (Q2.2+Q3.2), the accuracy of Aide and LifeJoin drops to below 10%, while our system achieves 85% with the per-iteration time within a few seconds. (3) Aide and



(a) Effect of dimensionality on convergence (b) Balancing classes (36-dim, 1 pos initially) (c) Balancing classes (36-dim, 2 pos initially)

(d) Feature scores (unbalanced, 2 pos initially) (e) Feature scores (balanced, 2 pos initially) (f) Sum of scores vs. entropy

Figure 8: F1-score and feature importances of GBRT-based feature selection (Q2.2 with added irrelevant features).



(a) Principled Component Analysis (PCA) (b) Random Forests (c) Gradient Boosting Regression Trees

Figure 9: Compare Dimensionality Reduction and Feature Selection techniques for learning Query 2.2 using a SDSS 66k dataset.

Table 2: Compare to Aide and LifeJoin for the final retrieval.

Query	Metrics	LifeJoin	AIDE	Ours
Q1.1	F-score (%)	6.1 (95.4)	95.8	88.0
	retrieval time (s)	0.683	0.013	79.7
Q2.1	F-score (%)	36.9 (61.4)	86.5	93.9
	retrieval time (s)	0.338	0.018	104.3
Q2.2+Q3.2	F-score (%)	0.02 (0.007)	4.6	84.9
	retrieval time (s)	2.575	0.088	207.2
Q2.1+34 attr	F-score (%)	1.2 (1.2)	34.0	93.9
	retrieval time (s)	0.338	4.0	104.3

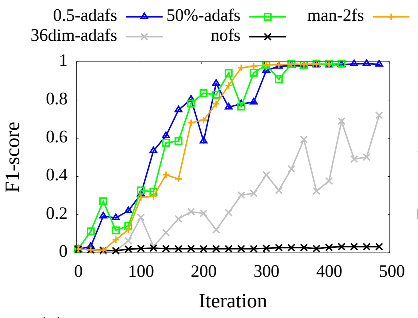
LifeJoin cannot handle higher-dimensional space, and have accuracy close to 0 for the 36D workload. (4) For ad-hoc exploration, where the user offers to label a limited number (e.g., 100) samples, we can achieve above 60% accuracy for 2D and 4D queries, but lower accuracy for workloads with up to 36 dimensions.

Finally, Table 2 shows the final result retrieval after 500 iterations and compares the three systems in both accuracy and running time. Again, LifeJoin suffers from low accuracy, using either its decision-tree based final retrieval method or running the SVM model over the database (in parentheses). Aide loses accuracy for workloads beyond 2D. Our system maintains high accuracy while having a modest final retrieval time of a few minutes.

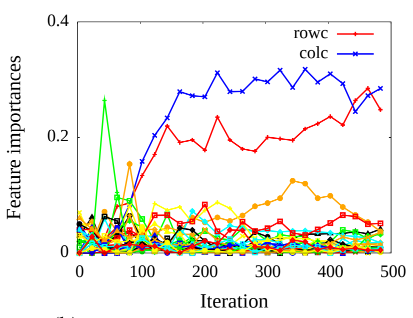
6. RELATED WORK

Data Exploration. *Faceted search* iteratively recommends query attributes for drilling down into structured databases, but the user is often asked to provide attribute values until the desired tuple(s) are returned [44, 45, 29] or provides an"interestingness" measure and its threshold [14]. Semantic windows [28] are pre-defined multidimensional shape-based and content-based predicates that a user can explore. Its utility is restricted to the case that such patterns exactly suit the user interest. To speed up interactive exploration, adaptive tree indexes are built [58] on parts of the data that the user has actually queried, rather than on all the data upfront. The work [39] specifically focuses on iterative "linear algebra programs" and proposes techniques based on matrix factorization to make incremental view maintenance substantially cheaper than re-evaluation. Most recent work has proposed a model to interpret the variability of likely queries in a workload [18], and dynamic prefetching of data tiles for interactive visualization [4].

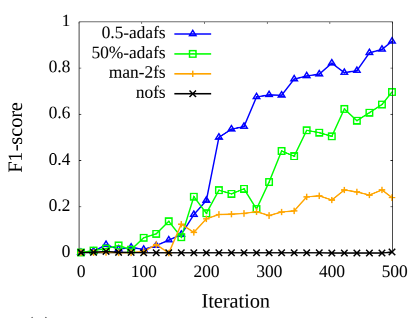
Query by Example is a specific framework for data exploration. Earlier work on QBE focused on a visualization front-end that aims to minimize the user effort to learn the SQL syntax [26, 35, 41, 57]. Recent work [38] proposes exemplar queries which treat a *query* as



(a) Accuracy of adaptive top-k features

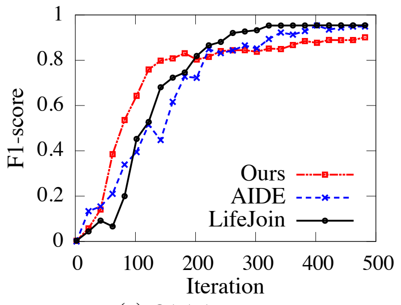


(b) Scores in adaptive top-k features

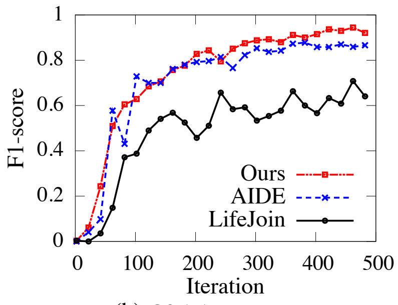


(c) Feature selection from 36-dim for Q2.1

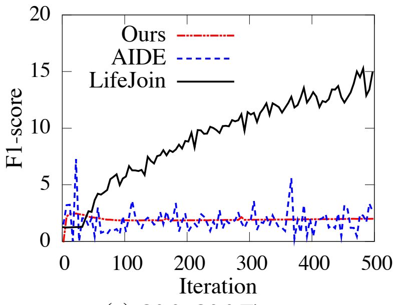
Figure 10: Optimization of GBRT feature selection and convergence rate in number of iterations



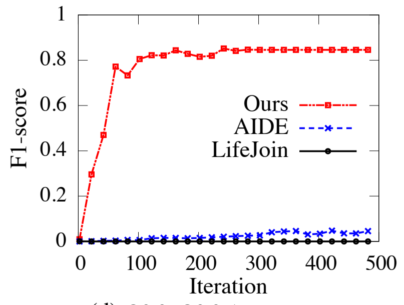
(a) Q1.1 Accuracy



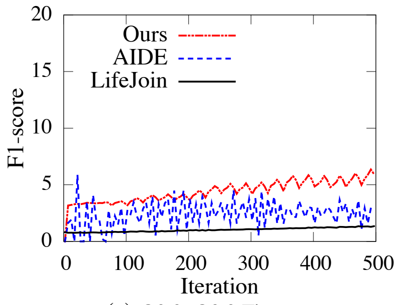
(b) Q2.1 Accuracy



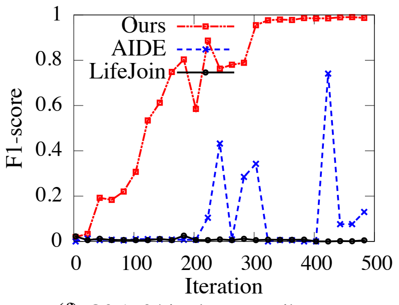
(c) Q2.2+Q3.2 Time



(d) Q2.2+Q3.2 Accuracy



(e) Q2.2+Q3.2 Time



(f) Q2.1+34 irrelevant attributes

Figure 11: Compare our system to Aide and LifeJoin in accuracy and per-iteration time for four query workloads.

a sample from the desired result set and retrieve other tuples based on similarity metrics, but for graph data only. The work [48] considers data warehouses with complex schemas and aims to learn the minimal project-join queries from a few example tuples efficiently. It does not consider selection with complex predicates, which is the main focus of our work. The work [27] helps users construct join queries for exploring relational databases, and [31] does so by asking the user to determine whether a given output table is the result of her intended query on a given input database. These works are relevant yet orthogonal to our active learning based approach.

Query formulation has been surveyed in [12]. The closest to our work is LifeJoin [11], which we described and compared in our performance study. Query By Output (QBO) [53] takes the output of some query on a database, and aims to construct an alternative query such that running these two queries on the database are instance-equivalent. Das Sarma et al. [13] studied the complexity of finding a query that describes the relationships between a database and an existing view. Dataplay [1] provides a graphical interface for users to directly construct and manipulate query trees, assuming that users already have knowledge about quantified queries. In [55], the user is asked to provide both the input and output relations, with a small number of sample tuples in each, in order to synthesize a SQL query consistent with the sample tuples. In summary, our work takes an active-learning approach with new sampling algorithms and theory on convergence.

Active Learning. Tong and Koller [52] provide a theoretical motivation on selecting which examples to request next using the notion of a version space. However, the convergence speed is unknown. Related to our work is a lower bound on the probability of misclassification error on the unlabeled training set based on a large deviation theory [10]. However, the calculation of the lower bound relies on user labeling of an additional sampled subset from the unlabeled pool, which is not required in our work. A recent set of papers [19, 23, 24, 25] offered probabilistic bounds for the classification error and sample complexity. Our work differs from these in that 1) we focus on F1-score, which suits selective user interest queries (imbalanced classes in classification), 2) our lower bound is deterministic. Note that different performance metrics (F measure versus classification error) lead to different relative performances of the active learning methods. Since the user interest exploration is naturally an imbalanced problem, i.e., the true user interest query selects way less than 50% database objects, F measure is a more suitable measure because it emphasizes the accuracy regarding the positive class (i.e., objects in the query answer set). Recent work [43] focuses on preference learning by pairwise comparison on structured entities. It uses linear SVM and the way to select the next example is similar to [52].

There are also a set of stopping criteria proposed for active learning. Schohn and Cohn [46] developed a heuristic stopping rule that labeling stops when the examples in the margin of the SVM have all been labeled. It does not give any indication of classification error at convergence. Four simple stopping criteria based on confidence estimation over the unlabeled data pool and the label consistency between consecutive training rounds of active learning have been presented [56]. Fu and Yang use as the stopping criterion a measure on whether SVM's separating hyperplane lies in a low density region [21]. Vlachos defines the confidence of the SVM classifier as the sum of the decision margins for the instances of a test set and stops the active learning process when the confidence reaches its peak [54]. Recent work in NLP [6] compares successive model predictions on a set of examples that do not need to be labeled and choose a proper set size as well as a cut-off value on a measure of "agreement" . Another study [40] focuses on the committee-based active learning and proposes to stop active learning when the Selection Agreement (decision on the most informative example selected for the next iteration) is no less than the Validation Set Agreement (decision on the validation of the current classifier on an unannotated dataset). The above techniques are based on various heuristics, while our work provides stronger results, namely, provable lower bounds on the F1 accuracy measure, and uses the lower bound as the stopping criteria for active learning.

7. CONCLUSION AND FUTURE WORK

We presented the design of a new database service for data exploration by example. We devised new uncertainty sampling algorithms with formal results, and a series of optimizations to improve performance. Our main results are: (1) Our solver method for sample retrieval achieves a good balance between accuracy and running time. (2) For convex patterns, our TSM algorithm offers a lower-bound for F-score, and reduces the user labeling effort. (3) For high-dimensional space, our GBRT optimization improves F-score from nearly 0 without feature selection, to high F-measures (above 0.8). (4) Our system significantly outperforms Aide and LifeJoin, two alternative systems, in accuracy while achieving desired efficiency for interactive exploration.

In future work, we will address more complex user interest patterns such as a union of convex shapes, and explore unlabeled data to improve accuracy. We will also explore database optimizations using materialized views and multi-query optimization.

8. REFERENCES

[1] A. Abouzied, J. Hellerstein, and A. Silberschatz. Playful Query Specification with DataPlay. In *Proc. VLDB Endow.*, 5(12):1938–1941, 2012.

[2] H. Anton. *Calculus, with analytic geometry*. Wiley, 1984.

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, Dec. 1996.

[4] L. Battle, R. Chang, et al. Dynamic prefetching of data tiles for interactive visualization. In *SIGMOD*, 1363–1375, 2016. ACM.

[5] R. N. Bhattacharya and J. .K. Ghosh On the Validity of the Formal Edgeworth Expansion In *Ann. Statist.*, 6(2):434–451, 1978.

[6] M. Bloodgood and K. Vijay-Shanker. A method for stopping active learning based on stabilizing predictions and the need for user-adjustable stopping. In *CoNLL*, 39–47, 2009.

[7] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *J. Mach. Learn. Res.*, 6:1579–1619, Dec. 2005.

[8] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[9] L. Breiman. Manual on setting up, using, and understanding random forests v3. 1. *Statistics Department, University of California Berkeley*, 2002.

[10] C. Campbell, N. Cristianini, and A. J. Smola. Query learning with large margin classifiers. In *ICML*, 111–118, 2000.

[11] A. Cheung, A. Solar-Lezama, and S. Madden. Using Program Synthesis for Social Recommendations. In *CIKM*, 1732–1736, 2012.

[12] A. Cheung and A. Solar-Lezama. Computer-Assisted Query Formulation. *Found. Trends Program. Lang.*, 3(1):1–94, 2016.

[13] A.S. Das, A. Parameswaran, et al. Synthesizing View Definitions from Data. *ICDT*, 89–103, 2010.

[14] D. Dash, J. Rao, N. Megiddo, et al. Dynamic faceted search for discovery-driven analysis. In *CIKM*, 2008.

[15] Y. Diao, K. Dimitriadou, Z. Li, et al. AIDE: an automatic user navigation system for interactive data exploration. *PVLDB*, 8(12):1964–1967, 2015.

[16] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *SIGMOD*, pages 517–528, 2014.

[17] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. AIDE: an active learning-based approach for interactive data exploration. *TKDE*, 2016. Accepted for publication.

[18] R. Ebenstein, N. Kamat, et al. Fluxquery: An execution framework for highly interactive query workloads. *SIGMOD*, 1333–1345, 2016.

[19] R. El-Yaniv and Y. Wiener. Active learning via perfect selective classification. *J. Mach. Learn. Res.*, 13(1):255–279, Feb. 2012.

[20] S. Ertekin, J. Huang, L. Bottou, and L. Giles. Learning on the border: active learning in imbalanced data classification. In *ACM Conference on information and knowledge management*, 127–136, 2007.

[21] C. Fu and Y. Yang. Low density separation as a stopping criterion for active learning svm. *Intelligent Data Analysis*, 19(4):727–741, 2015.

[22] B. Grünbaum. Convex polytopes. In *Convex Polytopes*. Springer-Verlag New York, 2 edition, 2003.

[23] S. Hanneke. Rates of convergence in active learning. *Ann. Statist.*, 39(1):333–361, 02 2011.

[24] S. Hanneke. Theory of disagreement-based active learning. *Found. Trends Mach. Learn.*, 7(2-3):131–309, June 2014.

[25] S. Hanneke. Refined error bounds for several learning algorithms. *J. Mach. Learn. Res.*, 17(1):4667–4721, Jan. 2016.

[26] B. E. Jacobs and C. A. Walczak. A Generalized Query-by-Example Data Manipulation Language Based on Database Logic. *IEEE Transactions on Software Engineering*, 9(1):40–57, 1983.

[27] M. Kahng, S. B. Navathe, et al. Interactive browsing and navigation in relational databases. *Proc. VLDB Endow.*, 9(12):1017–1028, 2016.

[28] A. Kalinin, U. Cetintemel, and S. Zdonik. Interactive data exploration using semantic windows. In *SIGMOD*, 505–516, 2014.

[29] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed Interactive Cube Exploration. In *ICDE*, 2014.

[30] Y. Lee and W. C. Kim. Concise formulas for the surface area of the intersection of two hyperspherical caps. Technical report, KAIST technical report IE-TR-2014-01, 2014.

[31] H. Li, C.-Y. Chan, and D. Maier. Query from examples: An iterative, data-driven approach to query construction. *Proc. VLDB Endow.*, 8(13):2158–2169, Sept. 2015.

[32] W. Liu, Y. Diao, and A. Liu. An analysis of query-agnostic sampling for interactive data exploration. Technical report, UM-CS-2016-003, University of Massachusetts Amherst, 2016.

[33] G. Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

[34] Large synoptic survey telescope: the widest, fastest, deepest eye of the new digital age. `http://http:/www.lsst.org/`.

[35] D. McLeod. The translation and compatibility of SEQUEL and Query by Example. In *International Conference on Software Engineering* (ICSE),1976.

[36] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446, 1909.

[37] D. Mottin, M. Lissandrini, Y. Velegrakis, et al. Exemplar queries: Give me an example of what you need. *PVLDB*, 7(5):365–376, 2014.

[38] D. Mottin, M. Lissandrini, Y. Velegrakis, et al. Exemplar queries: Give me an example of what you need. *Proc. VLDB Endow.*, 7(5):365–376, Jan. 2014.

[39] M. Nikolic, M. ElSeidy, and C. Koch. Linview: Incremental view

maintenance for complex analytical queries. In *SIGMOD*, 253–264, 2014.

[40] F. Olsson and K. Tomanek. An intrinsic stopping criterion for committee-based active learning. In *CoNLL*, 138–146, 2009.

[41] G. Özsoyoglu and H. Wang. Example-Based Graphical Database Query Languages. *Computer*, 26(5):25–38, 1993.

[42] L. Peng, E. Huang, et al. Uncertainty Sampling and Optimization for Interactive Database Exploration. UMass TR, 2017, `http://www.cs.umass.edu/~yanlei/explore2017.pdf`

[43] L. Qian, J. Gao, and H. V. Jagadish. Learning user preferences by adaptive pairwise comparison. *Proc. VLDB Endow.*, 8(11):1322–1333, July 2015.

[44] S. B. Roy, H. Wang, G. Das, et al. Minimum-effort driven dynamic faceted search in structured databases. In *CIKM*, 2008.

[45] S. B. Roy, H. Wang, U. Nambiar, et al. Dynacet: Building dynamic faceted search systems over databases. In *ICDE*, 2009.

[46] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *ICML*, 839–846, 2000.

[47] J. Shawe-Taylor and N. Cristianini. Section 2.2.3. kernel-defined nonlinear feature mappings. In *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[48] Y. Shen, K. Chakrabarti, S. Chaudhuri, et al. Discovering queries based on example tuples. In *SIGMOD*, 493–504, 2014.

[49] J. Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[50] I. Steinwart, D. Hush, and C. Scovel. An explicit description of the reproducing kernel hilbert spaces of gaussian rbf kernels. Technical report, IEEE Trans. Inform. Theory, 2005.

[51] A. S. Szalay, P. Z. Kunszt, A. Thakar, et al. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. In *SIGMOD*, 451–462, 2000.

[52] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, Mar. 2002.

[53] Q.T. Tran, C.Y. Chan, and S. Parthasarathy. Query by Output. In *SIGMOD*, 535–548, 2009.

[54] A. Vlachos. A stopping criterion for active learning. *Comput. Speech Lang.*, 22(3):295–312, July 2008.

[55] S. Zhang and Y. Sun. Automatically synthesizing SQL queries from input-output examples. In *ASE*, 224-234, 2013.

[56] J. Zhu, H. Wang, E. Hovy, and M. Ma. Confidence-based stopping criteria for active learning for data annotation. *ACM Trans. Speech Lang. Process.*, 6(3):3:1–3:24, Apr. 2010.

[57] M. M. Zloof. Query-by-example: operations on hierarchical data bases. In *AFIPS*, 845–853, 1976.

[58] K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 1555–1566, 2014.

APPENDIX

A. ADDITIONAL BACKGROUND

We provide additional background in this section. To illustrate a user interest pattern, Figure 12(a) shows a circle pattern in the Sloan Digital Sky Survey (SDSS) database. The Aide system for explore by example [16, 17] uses decision trees to build classification models. To approximate the circle pattern, Aide may need to use dozens of range predicates connected by logical *and* and *or* operators, and require the user to label many samples to achieve high accuracy. We also illustrate the distribution of the SVM classification model for the same query in Figure 12(b).

A.1 Primer on Support Vector Machines

For classification, the learning algorithm for SVM takes a set of training examples in the *data space*, each labeled using one of the two output classes, and builds a binary classifier that assigns labels for the new test examples in the same space. SVM is a **linear classifier** in the sense that it makes a classification decision based on the value of a linear combination of an example's characteristics. Interestingly, the algorithm works not only when the training examples from different classes are linearly separable in the data space, but also when they are not. In the latter case, the examples will be mapped into a much higher-dimensional space called the *feature space*, where linear separation can be achieved. Among the many hyperplanes that linearly separate the examples from different classes either in the data space or in the feature space, the one with the largest distance to the nearest (mapped) examples is returned by the algorithm as the *decision boundary*. In general, the larger the margin the lower the generalization error of the classifier. Therefore, SVMs are also called **large margin classifiers**.

Formally, the decision boundary in the feature space can be described as

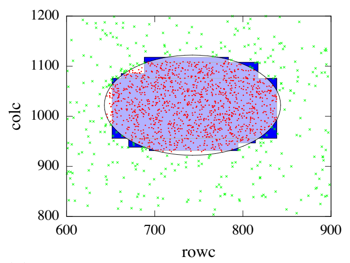
$$y(\boldsymbol{x}) = \boldsymbol{\omega}^T \phi(\boldsymbol{x}) + b = 0, \tag{5}$$

where $\boldsymbol{x}$ denotes a point in the data space and $\phi(\boldsymbol{x})$ is its mapped value in the feature space. We can select two hyperplanes parallel to the decision boundary such that there are no examples between them, and then try to maximize their distance. Without loss of generality, we select $\boldsymbol{\omega}^T \phi(\boldsymbol{x}) + b = 1$ and $\boldsymbol{\omega}^T \phi(\boldsymbol{x}) + b = -1$, the distance between which is $\frac{2}{||\boldsymbol{\omega}||}$, and we can state the constraints that $\boldsymbol{\omega}^T \phi(\boldsymbol{x}) + b \geq 1$ for all positive examples and $\boldsymbol{\omega}^T \phi(\boldsymbol{x}) + b \leq -1$ for all negative examples. Maximizing the distance $\frac{2}{||\boldsymbol{\omega}||}$ is equivalent to minimizing $||\boldsymbol{\omega}||$, or $\frac{1}{2}||\boldsymbol{\omega}||^2$ for mathematical convenience without changing the solution for $\boldsymbol{\omega}$ and b. Putting it all together, we have the following optimization problem:

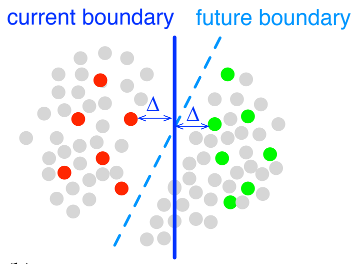
$$\begin{aligned} & \underset{\boldsymbol{\omega}, b}{\text{minimize}} && \frac{1}{2}||\boldsymbol{\omega}||^2 \\ & \text{subject to} && \bar{y}_i(\boldsymbol{\omega}^T \phi(\boldsymbol{x}_i) + b) \geq 1 \quad i = 1, \ldots, n. \end{aligned} \tag{6}$$

where $\bar{y}_i$ is the true label of a training point $\boldsymbol{x}_i$. The above is an optimization problem with a convex quadratic objective and only linear constraints. It can be solved using quadratic programming (QP) and the solution gives us the optimal margin classifier.

To ensure linear separation of training examples, sometimes feature spaces may have an exponential or even infinite number of dimensions, which would make it seem impossible to provide efficient computation [47]. The main theory of SVMs states that one can solve the *dual*, which is derived by introducing Lagrange multipliers α_i, in lieu of the *primal* problem. The dual optimization



(a) A circle pattern, training points (red:$+$, green:$-$), and approximation by decision trees (blue region).



(b) Unlabeled data in feature space (red points:$+$, green points:$-$, grey points: unlabeled).

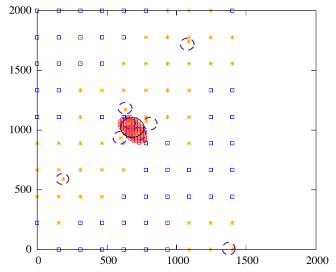

(c) Distribution of samples close to the decision boundary in the data space.

Figure 12: SVM for learning a circle query in SDSS, $(rowc-742.76)^2+(colc-1022.18)^2<100^2$.

problem is a maximization problem with parameters being the α_i's:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{maximize}} \quad & \sum_{i=1}^{n}\alpha_i\bar{y}_i-\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) \\ \text{subject to} \quad & \sum_{i=1}^{n}\alpha_i=0 \end{aligned} \tag{7}$$

It can be derived that the α_i's are zero except for the *support vectors*, defined to be the training examples that are on the margin. We do not show the derivation of the dual problem and its solution here due to space constraints, but one important intermediate formula that was derived previously and will be used for our later derivation is:

$$\boldsymbol{\omega}=\sum_{i=1}^{n}\alpha_i\phi(\boldsymbol{x}_i)=\sum_{\phi(\boldsymbol{x}_i)\in\boldsymbol{S}}\alpha_i\phi(\boldsymbol{x}_i) \tag{8}$$

where $\boldsymbol{S}$ refers to the set of support vectors. Plugging Eq. (8) to Eq (9), the decision boundary can be rewritten as

$$y(\boldsymbol{x})=\sum_{\phi(\boldsymbol{x}_i)\in\boldsymbol{S}}\alpha_i\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x})+b. \tag{9}$$

Now the decision boundary requires merely the inner product between $\phi(\boldsymbol{x})$ and each support vector $\phi(\boldsymbol{x}_i)$.

Another benefit of the dual problem relates to the so-called Kernel trick. Given a mapping ϕ, a **kernel** is defined to be $K(\boldsymbol{x}_i,\boldsymbol{x}_j)=\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$. Any proposed kernel function must be validated by Mercer's theorem [36]. Most notably, $K(\boldsymbol{x}_i,\boldsymbol{x}_j)$ can be inexpensive to calculate because it applies to the variables in the data space, even when $\phi(\boldsymbol{x})$ may be very expensive to calculate due to a high dimensionality. In this case, SVMs can be learned without ever having to explicitly find or represent vectors $\phi(\boldsymbol{x})$.

Commonly used kernels include the linear kernel, polynomial kernel, and Gaussian kernel (a.k.a., radial basis function kernel). Without prior knowledge of what the user interest may be, the Gaussian kernel is considered more flexible than the linear or polynomial kernels, hence used in this work. The Gaussian kernel is defined as: $K_G(\boldsymbol{x}_i,\boldsymbol{x}_j)=\exp(-\gamma||\boldsymbol{x}_i-\boldsymbol{x}_j||^2)$. The feature space of the Gaussian kernel is known to be of an infinite number of dimensions [50].

B. PROOFS FOR CONVERGENCE

Proof of Proposition 3.1:

PROOF. Since (1) $e_i^+\in Q$ for $i=1,\dots,n^+$, (2) R^+ is the smallest convex set that contains $\cup_{i=1}^{n^+}e_i^+$, and (3) Q is convex, we can derive that $R^+\subseteq Q$, which means all points in R^+ are guaranteed to be positive. □

Proof of Proposition 3.2:

PROOF. Let us first prove that all points in each R_i^- are negative. Suppose that some point $\boldsymbol{x}_0\in R_i^-$ is positive. According to the definition of R_i^-, $\overrightarrow{\boldsymbol{x}_0\boldsymbol{e}_i^-}\cap R^+\neq\emptyset$, which means we can find a point $\boldsymbol{x}_1$ such that $\boldsymbol{x}_1\in R^+$ and $\boldsymbol{x}_1\in\overrightarrow{\boldsymbol{x}_0\boldsymbol{e}_i^-}$. Then $\boldsymbol{e}_i^-$ is on the line segment connecting two positive points $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$. This contradicts the convex query assumption. Hence the supposition is false and all points in R_i^- are negative. Since R^- is just a union of all R_i^-'s, all points in R^- are negative as well. □

Proof of Theorem 3.1:

PROOF. At any iteration i, D_{eval} can be partitioned into D^+, D^- and D^u. Recall that D_{eval} is a projection of D_{test} without the labels. We know for certain that the labels for all points in D^+ (or D^-) are positive (or negative) in D_{test} according to Proposition 3.1, 3.2 and the definition of D^+ and D^- in Definition 3.4; only the labels for points in D^u are uncertain.

Let us assume that $p\%$ points in D^u are predicted as positive by the SVM model trained at Line 23 of Algorithm 2. Denote the set of points as D^{u+}. Then $|D^{u+}|=p\%\cdot|D^u|$ and we can write the precision and recall of the trained SVM model as

$$\begin{aligned} precision &= \frac{|D^+|+|D^{u+}\cap Q|}{|D^+|+|D^{u+}|}=\frac{|D^+|+|D^{u+}\cap Q|}{|D^+|+p\%\cdot|D^u|} \\ &\geq \frac{|D^+|}{|D^+|+|D^u|} \\ recall &= \frac{|D^+|+|D^{u+}\cap Q|}{|D^+|+|D^u\cap Q|}\geq\frac{|D^+|}{|D^+|+|D^u|} \end{aligned}$$

F1-score is the harmonic mean of precision and recall. So F1-score is lower-bounded by $|D^+|/(|D^+|+|D^u|)$. □

Proof of Theorem 3.2:

PROOF. Since $(n\hat{p},n\hat{q})^T$ follows Multinomial$(n,p,q,1-p-q)$, the vector converges to the bivariate Normal distribution when n increases according to the Central Limit Theorem. Specifically,

$$\sqrt{n}\left(\begin{pmatrix}\hat{p}\\\hat{q}\end{pmatrix}-\begin{pmatrix}p\\q\end{pmatrix}\right)\xrightarrow{D}N\left(\begin{pmatrix}0\\0\end{pmatrix},\Sigma\right)$$

where $\Sigma=\begin{pmatrix}p & -pq\\-pq & q\end{pmatrix}$.

Define $v=(p,q)^T$, $\hat{v}=(\hat{p},\hat{q})^T$, and $g(v)=\dfrac{p}{1-q}$. Then $b=g(v)$ and $X=g(\hat{v})$. According to the Delta method [5]

$$sup_\epsilon||\Pr(\sqrt{n}|X_n-b|<\epsilon)-\int_{-\epsilon}^{\epsilon}\phi_{\sigma^2}(t)\mathrm{d}t||=O(1/\sqrt{n})$$

where ϕ is the density function of the Normal distribution with mean zero and variance $\sigma^2 = (\partial g(v)/\partial v)^T \Sigma (\partial g(v)/\partial v) = p(1-p-q)/(1-q)^2$. Therefore, the theorem is proved. $\square$

C. MORE ON OPTIMIZATIONS

We present additional techniques for expediting convergence in this section.

C.1 TSM-based Optimization

Toward the goal of faster convergence, one useful technique falls naturally from our Three-Set Metric (TSM) Algorithm (Algorithm 2) for convex user interest patterns. The algorithm internally maintains a positive sample set and a negative set based on its partitioning function of the data space. Compared to a direction application of active learning theory (Algorithm 1), which retrieves the sample closest to the current SVM decision boundary, our TSM avoids asking the user to label the retrieved sample if it is known to belong to its positive or negative sample set. From the user's perspective, the convergence has expedited because the same accuracy can be achieved by labeling fewer samples. The performance benefits of this optimization can be significant as we show in the evaluation.

C.2 Dimensionality Reduction Methods

In our work, we examine a range of popular feature selection techniques in our active learning framework for data exploration. These techniques are provided by the scikit-learn library[11]:

Principled Component Analysis (PCA) defines a set of orthogonal directions that capture the maximum variance of a dataset, with an implicit hope that the variance along a small number of principal components provides a reasonable characterization of the entire dataset [49]. In our work, PCA is used as a query-agnostic dimensionality reduction technique. That is, we use it to compress a database table $D(A_1, \ldots, A_d)$ into a new table $D'(B_1, \ldots, B_k)$ with fewer columns, $k < d$. Each database object has two presentations using $(A_1, \ldots, A_d)$ and $(B_1, \ldots, B_k)$, respectively. In each iteration of data exploration, we display a new sample to the user using the $D(A_1, \ldots, A_d)$ representation, but train an SVM with all existing labeled samples on $D'(B_1, \ldots, B_k)$.

Random forests (RF) are an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the mean prediction of the individual trees.

Gradient boosting regression trees (GBRT) are a gradient boosting machine with decision trees as base-learners. The basic idea is to construct a series of decision trees in a forward stagewise manner, where each new decision tree is constructed to be maximally correlated with the negative gradient of the loss function – the error of the whole ensemble learnt so far. The additive model of GBRT is a linear combination of the base learners.

In our work, RF and GBRT are both used for online feature selection[12]. That is, when a user is interacting with the system for data exploration, in each iteration we first feed all existing labeled samples to the RF or GBRT learner, and ask the learner to return the top-k features that are deemed most important in the learning process (to be formally defined shortly). Next we build an SVM classification model using only the top-k features and select the next sample to be labeled as the object closest to the current decision boundary of the SVM model. Then we repeat these two steps in the next iteration until the SVM classification model converges.

As our experimental results show, GBRT works better than RF and PCA for reducing the number of features or dimensions needed to train the SVM classification model. Therefore, our work focuses on the additional optimizations of feature selection based on GBRT.

[11]http://scikit-learn.org/stable/

[12]Since *feature selection* is the standard term in the literature, we use "features" and "attributes" interchangeably in this context.

D. ADDITIONAL EVALUATION RESULTS

D.1 Experimental Setup

Datasets: We evaluate our techniques using the "PhotoObjAll" table, which contains 510 attributes, from the Sloan Digital Sky Survey (SDSS) with data release 8[13]. The table contains the full photo metric catalog quantities for SDSS imaging, one entry per detection. We downloaded around 192 million tuples. For experiment purposes, we generated tables by random sampling the base table with different sampling ratios, 0.03%, 1%, 10%. After loading to PostgreSQL, the sizes were 300MB, 9991MB, and 98GB, respectively. B+ tree indexes on the key attribute $objid$ were pre-built to facilitate example (i.e., tuple) retrieval. Since data exploration usually operates on a sampled dataset that fits in memory, we used the 1% dataset, which is the largest that fits in memory, as our default data exploration space.

User Interest Queries: We extracted a set of queries from the SDSS query release 8 to represent true user interests[14], as shown in Table 1. These user interest queries allow us to run simulation of user exploration sessions: we precompute the answer set of each query, then run a data exploration session as described in the previous sections; during each iteration, when the active learning algorithm presents a new sample to be labeled, we consult the query answer set to decide whether to give a positive or negative label.

When choosing queries in our experiments, we consider the following factors : (1) pattern: queries can be linear or non-linear, (2) varied query selectivities, and (3) varied query dimensionalities. The queries are summarized in Table 1: On ($rowc$, $colc$), i.e., the row and column center positions, data are roughly evenly distributed, and we have two groups of queries, one for the linear pattern (Q1) and one for the non-linear (Q2). On (ra, dec), i.e., the right-ascension and declination in the spherical coordinate system, represent workloads with skewed data (see Figure 5). Within each group, we consider three selectivities: 0.1%, 1% and 10%, which we believe covers the general settings in real applications. We combine queries on ($rowc$, $colc$) and (ra, dec) to obtain 4-dimensional queries with varied selectivities, e.g., combining Q2.2 on ($rowc$, $colc$) and Q3.3 on (ra, dec) will result in a query on ($rowc$, $colc$, ra, dec) with selectivity 0.1%. We also consider workloads with varied number of irrelevant attributes added such that the dimensionality of the exploration space is greater than that of the query space and goes up to 36.

We conduct 10 runs for each query and in each run, one positive sample and one negative sample that are randomly selected will be fed to the system as initial samples.

Servers: Our experiments were run on five identical servers, each with 12-cores, Intel(R) Xeon(R) CPU E5-2400 0 @2.4GHz, 64GB memory, JVM 1.7.0 on CentOS 6.6.

[13]http://www.sdss3.org/dr8/

[14]http://skyserver.sdss.org/dr8/en/help/docs/realquery.asp