An Optimal Online Algorithm For Retrieving Heavily Perturbed Statistical Databases In The Low-Dimensional Querying Model

Krzysztof Choromanski
Google Research
76, Ninth Ave
New York, NY 10011, USA
kchoro@google.com

Afshin Rostamizadeh
Google Research
76, Ninth Ave
New York, NY 10011, USA
rostami@google.com

Umar Syed
Google Research
76, Ninth Ave
New York, NY 10011, USA
usyed@google.com

ABSTRACT

We give the first $\tilde{O}(\frac{1}{\sqrt{T}})$-error online algorithm for reconstructing noisy statistical databases, where T is the number of (online) sample queries received. The algorithm is optimal up to the $poly(\log(T))$ factor in terms of the error and requires only $O(\log T)$ memory. It aims to learn a hidden database-vector $w^* \in \mathbb{R}^D$ in order to accurately answer a stream of queries regarding the hidden database, which arrive in an online fashion from some unknown distribution $\mathcal{D}$. We assume the distribution $\mathcal{D}$ is defined on the neighborhood of a low-dimensional manifold. The presented algorithm runs in $O(dD)$-time per query, where d is the dimensionality of the query-space. Contrary to the classical setting, there is no separate training set that is used by the algorithm to learn the database — the stream on which the algorithm will be evaluated must also be used to learn the database-vector. The algorithm only has access to a binary oracle $\mathcal{O}$ that answers whether a particular linear function of the database-vector plus random noise is larger than a threshold, which is specified by the algorithm. We note that we allow for a significant $O(D)$ amount of noise to be added while other works focused on the low noise $o(\sqrt{D})$-setting. For a stream of T queries our algorithm achieves an average error $\tilde{O}(\frac{1}{\sqrt{T}})$ by filtering out random noise, adapting threshold values given to the oracle based on its previous answers and, as a consequence, recovering with high precision a projection of a database-vector w^* onto the manifold defining the query-space. Our algorithm may be also applied in the adversarial machine learning context to compromise machine learning engines by heavily exploiting the vulnerabilities of the systems that output only binary signal and in the presence of significant noise.

CIKM'15, October 19–23, 2015, Melbourne, VIC, Australia.

DOI: http://dx.doi.org/10.1145/2806416.2806421.

Categories and Subject Descriptors

H.2 [**Database management**]: Security, integrity, and protection; H.3 [**Information storage and retrieval**]: Information Search and Retrieval—*Retrieval models*; I.1 [**Symbolic and algebraic manipulation**]: Analysis of algorithms

General Terms

Algorithms, Theory, Security

Keywords

statistical databases, low-dimensional querying model, bisection method, database privacy, online retrieval algorithms, adversarial machine learning

1. INTRODUCTION

1.1 Database setting

Protecting databases that contain sensitive information has become increasingly important due to its crucial practical applications, such as the disclosure of sensitive health data. Privacy preservation plays a key role in this setting since such data is often published in anonymized form so it can be used by analysts and researchers. Several mechanisms have been proposed, such as differential privacy, that allow for learning from a database while preserving privacy guarantees ([1, 2, 3, 4, 5, 6]). At the other extreme are many results showing how database privacy can be compromised by an adversary who is able to collect perturbed answers to a large number of queries regarding the database ([7, 8, 9, 10, 6]). Existing results related to breaking the privacy of a database have several key limitations. For example, most assume that each query is represented by a vector q of D independent entries taken from some fixed distribution (such as the Gaussian distribution or a specific discrete distribution), and that this structure is known to the privacy-breaking algorithm. Also, most methods learn an approximation of the unknown database-vector w^* that has L_2 error ϵD for some small constant $\epsilon > 0$. Such precision is not sufficient to obtain $o(1)$-error on the stream of T queries for $T \gg D$, as is the case in our model. Further, the focus has typically been on the offline setting, where the adversary first collects all the queries, then applies some privacy-breaking algorithm, and finally uses the reconstructed database-vector to compute good approximations of the statistics he needs. From

the machine learning point of view this means that the overall protocol for the adversary consists of two distinct phases: a training phase and a testing phase. Finally, the memory resources used by privacy-breaking algorithms are typically not analyzed, even though this is a crucial issue for the setting considered here, where the number of all the queries q coming in the stream may be huge.

The goal of this paper is to present and analyze a database privacy-breaking algorithm for a more realistic setting in which the limitations described above are lifted. The entries of the query-vector are not necessarily independent. The distribution $\mathcal{D}$ of the query-vector is not known to the adversary. The adversary is not able to first learn the database-vector before being evaluated. Our algorithm uses only $O(\log(T))$-size memory to process the entire stream of T queries and therefore is well-suited to the limited resources scenario. To make life of the adversary even more difficult, we assume that the database mechanism provides only a binary oracle $\mathcal{O}$ that answers whether the perturbed value of a dot-product between the database-vector w^* and the query-vector q is greater than a threshold that is specified by the adversary. Thus the algorithm has very limited access to the database even in the noiseless scenario. Dot-products between query-vector and a database-vector are considered in most of the settings analyzing database privacy-breaking algorithms. Considering this more challenging setting, we will show that much less than the noisy answer is needed to carry out an effective attack and compromise data privacy.

In some of the mentioned papers ([5, 6]) an effort is made to learn a good approximation of the database vector with a small number of queries that is only linear in the size of the database D. We use many more queries but our task is more challenging - we need much more accurate approximation, and get the information only about the sign of the perturbed product as opposed to the perturbed product itself. Finally, we are penalized whenever we are making a mistake. Our goal is to minimize the average error of the algorithm over a long sequence of queries so we need to learn this more accurate approximation very fast.

In this paper we present the first online algorithm that an adversary can use to reconstruct a noisy statistical database protected by a binary oracle $\mathcal{O}$ that achieves average error $\tilde{O}(\frac{1}{\sqrt{T}})$ on the stream of T queries and operates in logarithmic memory. The algorithm is optimal, i.e. we show that any other algorithm (not necessarily online and with arbitrary running time) solving this problem achieves error at least $\tilde{O}(\frac{1}{\sqrt{T}})$ in the worst case scenario. From now on we will call this algorithm a *learning algorithm*. The learning algorithm is given a set of queries taken from some unknown distribution $\mathcal{D}$ defined on a neighborhood of the low-dimensional manifold that it needs to answer in the order that they arrive (note that the entries of a fixed query do not have to be independent). The learning algorithm can use the information learned from previously collected queries but cannot wait for other queries to learn a more accurate answer. Every received query can be used only once to communicate with a database. The database mechanism calculates a perturbed answer to the query and passes the result to the binary oracle $\mathcal{O}$. The binary oracle uses the threshold provided by the adversary and passes a "Yes/No"-answer to him. The error made for a single query is defined as: $|z^t - w^* \cdot q^t|$, where q^t and z^t are the query and answer, respectively, provided by the learning algorithm in round t. As a byproduct of our methods, we recover with high precision the projection of the database-vector w^* onto the query-space. Our approximation is within $\tilde{O}(\frac{1}{\sqrt{T}})$ L_2-distance from the exact projection. By comparison, most of the previous papers focused on approximating/recovering all but at most a constant fraction ϵD of all the entries of w^* which is unacceptably inaccurate in our learning setting where $T \gg D$. The assumption that queries are taken from a low-dimensional manifold is in perfect agreement with recent development in machine learning (see: [11], [12], [13]). It leads to the conclusion that, as stated in [11]: "a lot of data which superficially lie in a very high-dimensional space $\mathbb{R}^D$ actually have low *intristic dimensionality*, in the sense of lying close to a manifold of dimension $d \ll D$". Assume that the queries are taken from a truly high-dimensional space. Then as long as the number of all queries is polynomial in D, the average distances between them are substantial. In this scenario any nontrivial noisy setting prevents the adversary from learning anything about the database since a single perturbed answer does not give much information and the probability that a close enough query will be asked in the future is negligible in D. In practice we observe however that noise can be very often filtered out and a significant number of queries can give nontrivial information about a database-vector w^*. In this paper we explain this phenomenon from the theoretical point of view. Our algorithm accurately reconstructs the part of the database that regards the lower-dimensional space used for querying. We show that this suffices to achieve average $\tilde{O}(\frac{1}{\sqrt{T}})$-error on the set of T given queries. In our model, the number of queries significantly exceeds the dimensionality of the database, and therefore we focus on optimizing our algorithm's time complexity and accuracy as a function of T. Having said that, in most of the formulas derived in the paper we will also explicitly give the dependence on other parameters of the model such as the dimensionality of the database D and the dimensionality of the query-space d. We are mainly interested in the setting: $d \ll D \ll T$. If we use the O-notation, where the dependency is not explicitly given then we treat all missing parameters as constants.

It should be also emphasized that, contrary to most previous work on reconstructing databases based on the perturbed statistics, the proposed algorithm does not use linear programming and thus gives better theoretical guarantees regarding running time than most existing methods. The algorithm uses a subroutine whose goal is to solve a linear program, however we show this program has a closed-form solution. Therefore we do not need to use any techniques such as simplex or the ellipsoid method. The algorithm is very fast: it needs only $O(dD)$-time per query. More detailed analysis of the running time of the algorithm as well as memory usage will be given later.

1.2 Connections to adversarial machine learning

One important related setting for our model as well as the algorithm we are going to present comes from adversarial machine learning. In many real world machine learning applications an adversary can use data in order to reveal information about the machine learning classifier and consequently, use this information to trick the classifier. This happens for instance in such domains as fraud, malware and spam detection, biometric recognition, auction alloca-

tion/pricing and many more. Thus, the need arises to establish secure learning in the adversarial setting.

Some introduced methods include repeated manual reconstructions of the classifier or randomizing the classifier. For example, an adversary-robust classifier in proposed in [14], where the adversarial learning problem is formally defined. In [15], a randomized approach is analyzed as a tool do defend against the adversary. The paper in fact precisely characterizes all optimal randomization schemes in the presence of both classifier manipulations and adversarial inverse engineering. On the other hand, many practical methods to violate machine learning system security have also been introduced, for example using evasion attacks. In the spam detection framework these may involve, for instance, obfuscating spam emails' content. In general, malicious samples are modified and used to exploit the vulnerabilities of the system. Other techniques take advantage of the frequent retraining phases, where data can be possibly poisoned. In that setting the goal is usually to compromise the process of learning. Most of these aforementioned scenarios fit naturally in the online framework.

The random noise in these settings setup may come from many different sources (noise introduced to protect the machine learning framework, as it is in [15] or noisy channels as is the case for various geophysical signals such as sonar signals that are subject to noise depending on the ocean geometry and weather). The role of noise in machine learning setting was extensively studied (see: [16]) and must be taken into account in many real world settings.

We can think of the database-vector w^* in our model as a linear machine learning classifier. The dot-product given to the oracle has a natural interpretation since it says on which side of the hyperplane the given query is, and consequently determines its classification. The dot-product output has been extensively used in the statistical database setting, however (contrary to many other approaches), the techniques used in our algorithm can be used for various types of outputs, not necessarily linear in the query q (giving rise to applications for nonlinear classifiers). This is due to the generalization of the bisection idea in the presence of noise, that we fully explore in this paper and which constitutes the core of the proposed algorithm, is independent of the particular form of the output. The perturbation added before this dot-product is given to the oracle may be interpreted as a noise introduced to challenge the adversary or an effect of the noisy channel, which was also the case in the database setting described before. The oracle is like the machine learning classification mechanism that outputs the classification of the given query (for instance: spam/no-spam). The answer given by the oracle in our model is binary, however, straightforward extensions of the approach proposed by us lead to models, where the oracle chooses an answer from the larger discrete set.

Finally, let us comment on the interpretation of our algorithm's error in the adversarial classification setting. Note, the goal of our algorithm is to retrieve the vector w^*, which is a stronger goal than what is usually considered in the usual adversarial classification setting, i.e. forcing false classifications. The exact value of the classifier dot-product measures how far a given query is from the boundary of the binary classification, i.e. how confident the system is that a given query should be classified in a particular way. This information is not given explicitly by the engine but is nevertheless of great importance. It is reasonable to assume that the adversary does not have unlimited resources and that issuing queries has a cost. Thus, the goal of the adversary is to minimize the average per-round error during the sequence of queries, which is equivalent to compromising the classifier (within some additive error) with as few queries as possible.

2. MODEL DESCRIPTION AND MAIN RESULT

We will now describe in detail our database access model. We assume that the database can be encoded by the database-vector $w^* \in \mathbb{R}^D$. For definiteness we will consider: $w_i^* \in [0,1]$ for $i = 1, \ldots, D$. Our method can be however used in the much more general setting, as long as w^* is taken from some fixed ball in L_∞. Each query can be represented as a vector $q = (q_1, \ldots, q_D)$, where: $0 \leq q_i \leq 1$ and $q_1^2 + \cdots + q_D^2 > 0$. Queries are taken independently at random from the unknown distribution $\mathcal{D}$ (notice that entries of a fixed query do not have to be independent). The distribution $\mathcal{D}$ is defined on some d-dimensional linear subspace $\mathcal{U} \in \mathbb{R}^d$ ($d < D$). The exact answer to the query is given as $a = \sum_{i=1}^{D} w_i^* q_i$. For the t^{th} coming query q^t the learning algorithm $\mathcal{L}$ selects the threshold value θ^t and passes q^t to the database mechanism $\mathcal{M}$ which computes $a^t = w^* \cdot q^t$. The noisy version $\tilde{a}^t$ of a^t as well as θ^t is passed by $\mathcal{M}$ and $\mathcal{L}$ to the binary oracle $\mathcal{O}$:

$$\mathcal{O}(\tilde{a}^t, \theta^t) = \begin{cases} 1 & \text{if } \tilde{a}^t > \theta^t, \\ 0 & \text{otherwise.} \end{cases}$$

The value $\mathcal{O}(\tilde{a}^t, \theta^t)$ is then given to $\mathcal{L}$. The learner records this value and can also use the information obtained from previously received queries to give an answer z^t to the query q^t. However it has only $O(\log(T))$-memory available. Further, for a fixed query the learner only has one-time access to the binary oracle $\mathcal{O}$.

The noise $\epsilon^t = \tilde{a}^t - a^t$ is generated independently at random and is of the form $D\mathcal{E}$, where $\mathcal{E}$ is some known distribution producing values from some bounded range $[-u, u]$. The boundedness assumption is not crucial. Technically speaking, as long as the random variable is not heavy-tailed (which is a standard assumption), our approach works. In fact even this condition is unnecessarily strong. This will become obvious later when we describe and analyze our method.

This setting covers standard scenarios where computing every single product in the sum of d terms for $w^* \cdot q^t$ gives an independent bounded error. We should note here that in most of the previous papers the magnitude of the noise added was of the order $o(\sqrt{D})$ (see: [7, 8, 9, 10, 6]). For instance, in [8] the authors reconstruct a database that agrees with the groundtruth one on all but $(2c\alpha)^2$ entries, where α is a noise magnitude and $c > 0$ is a constant. Thus, even though previous works do not assume that noise was added independently for every query, the average error per single product in the dot-product sum was only of the magnitude $o(\frac{1}{\sqrt{D}})$. This assumption significantly narrows the range of possible applications. This is no longer the case in our setting, where some mild and reasonable assumptions regarding independence of noise added to different queries and low-dimensionality of querying space leads to a model much more robust to noise. We will assume that ϵ^t do not have singularities, i.e. $\mathbb{P}(\epsilon^t = c) = 0$ for any fixed c.

We need a few more definitions.

DEFINITION 2.1. *We say that a vector w computed by the learning algorithm ϵ-approximates database-vector w^* if $|\Pi_{\mathcal{U}}(w) - \Pi_{\mathcal{U}}(w^*)|_\infty \leq \epsilon$, where $\Pi_{\mathcal{U}}(v)$ stands for the projection of v onto d-dimensional querying space $\mathcal{U}$.*

DEFINITION 2.2. *Let $\mathcal{Q}$ be a probability distribution on the unit sphere $\mathcal{S}(0,1)$ in L_2. For a fixed vector $q \in \mathcal{S}(0,1)$ we denote by $p^{\mathcal{Q}}_{q,\theta}$ the probability that a vector x selected according to $\mathcal{Q}$ satisfies: $q \cdot x \geq \cos(\theta)$.*

DEFINITION 2.3. *Take a distribution $\mathcal{D}$ from which queries are taken. Assume that $\mathcal{D}$ is defined on the d-dimensional space $\mathcal{U}$ with orthonormal basis $\mathcal{B}$. Denote by $\mathcal{D}_n$ the normalized version of $\mathcal{D}$ and by $\mathcal{B}_n$ the normalized version of $\mathcal{B}$ (all vectors rescaled to length 1 in the L_2-norm). Then we define: $p_{\mathcal{D},\theta} = \min_{q \in \mathcal{B}_n}(p^{\mathcal{D}_n}_{q,\theta})$.*

The error ϵ_q the algorithm is making on each query q is defined as the absolute value of the difference between the exact answer to the query and the answer that is provided by the algorithm. The average error on the set of queries: $q^1, ..., q^T$ is defined as $\epsilon_{av} = \frac{1}{T}\sum_{i=1}^{T} \epsilon_{q^i}$. Let us state now main result of this paper.

THEOREM 2.1. *Let $q^1, \ldots, q^T$ be a stream of query-vectors coming in an online fashion from some d-dimensional subspace, where: $0 \leq q^t_i \leq 1$ for $i = 1, \ldots, d$ and each q^t is a nonzero vector. Then there exists an algorithm $\mathcal{A}lg$ using $O(\log(T))$-memory, acting according to the protocol defined above, and achieving average error:*

$$e_{av} = O(\frac{1}{\sqrt{T}}(rD^{\frac{7}{2}}d + \sqrt{D}\log(T)))$$

with probability $p_{succ} \geq 1 - O(\frac{\log(dDT)}{T^3} + \frac{d\log(dT)}{T^{30}})$, where $r = \frac{2}{p^2_{\mathcal{D},\phi}}$ and $\phi = 2\arcsin(\frac{1}{64\sqrt{d}})$.

We will give this algorithm, called *OnlineBisection* algorithm, in the next section. Notice that ϕ is well approximated by $\frac{1}{32\sqrt{d}}$. To see what the magnitude of r is in the worst-case scenario it suffices to analyze the setting where q is chosen uniformly at random from the query-space $\mathcal{U}$.

If this is the case then one can notice that $p_{\mathcal{D},\phi}$ is of the order $\Omega(2^{-d\log(d)})$ thus $r = O(2^{d\log(d)})$. If however there exists a basis of $\mathcal{U}$ such that most of the mass of $\mathcal{D}$ is concentrated around vectors from the basis then standard analysis leads to the $\frac{1}{poly(d)}$-lower bound on p, i.e. $poly(d)$-upper bound on r (where $poly(d)$ is a polynomial function of d).

Theorem 2.1 implies a corollary regarding the batch version of the algorithm, where test and training set are clearly separated (the proof of that corollary will be given later):

COROLLARY 2.1. *Let w_T denote the final hypothesis constructed by the OnlineBisection algorithm after consuming T queries drawn from an unknown distribution $\mathcal{D}$. Then the following inequality holds with probability at least $1 - O\Big(\frac{\log(dDT)}{T^3} + \frac{d\log(dT)}{T^{30}}\Big)$ for any future queries q drawn from $\mathcal{D}$:*

$$\mathrm{E}_{q\sim\mathcal{D}}\big[|w_T \cdot q - w^* \cdot q|\big] \leq \frac{\sqrt{D}\log(T)}{\sqrt{T}}\,.$$

In the subsequent sections we will prove Theorem 2.1 and conduct further analysis of the algorithm. Unless stated otherwise, log denotes the natural logarithm.

3. THE ALGORITHM

We will now present an algorithm (Algorithm 1) that achieves theoretical guarantees from Theorem 2.1. Our algorithm, called *OnlineBisection*, maintains a tuple of intervals $(\mathcal{I}_1, \ldots, \mathcal{I}_d)$ which encode a hypercube that contains the database-vector w^* (projected onto $\mathcal{U}$) with very high probability.

For each coming query-vector q^t the algorithm outputs an answer $w_{approx} \cdot q^t$, where w_{approx} is an arbitrarily selected vector in the current hypercube. The query-vectors received by the algorithm are used to progressively shrink the hypercube.

Algorithm 1 - *OnlineBisection*
Input: Stream $q^1, \ldots, q^T$ of T queries, database mechanism $\mathcal{M}$ and binary oracle $\mathcal{O}$.
Output: A sequence of answers $(w^1 \cdot q^1, \ldots, w^T \cdot q^t)$, returned online.
begin
  Choose an orthonormal basis $\mathcal{C} = \{e^1, \ldots, e^d\}$ of $\mathcal{U}$.
  Let $\phi = 2\arcsin(\frac{1}{64\sqrt{d}})$.
  Let $\mathcal{I}_i = [-\sqrt{D}, \sqrt{D}]$, $N^+_i = 0$ and $N^-_i = 0$ for $i = 1, \ldots, d$.
  for $t = 1, \ldots, T$ **do**
    Output $w_{approx} \cdot q^t$ for any $w_{approx} = f_1 e^1 + \cdots + f_d e^d$, where $f_i \in \mathcal{I}_i$, $i = 1, \ldots, d$.
    if $|\mathcal{I}_i| \leq \frac{\log(T)}{\sqrt{Td}}$ for $i = 1, ..., d$ **continue**.
    if $\exists i^* \in \{1, \ldots, d\}$ *such that* $\arccos(e^{i^*}, \frac{q^t}{\|q^t\|_2}) \leq \phi$ **then**
      Let $m = \max_{f_1 \in \mathcal{I}_1, ..., f_d \in \mathcal{I}_d} \sum_{i=1}^{d} f_i e^i \cdot (-q^t)$.
      Let $M = \max_{f_1 \in \mathcal{I}_1, ..., f_d \in \mathcal{I}_d} \sum_{i=1}^{d} f_i e^i \cdot q^t$.
      Let $b = \mathcal{O}(\mathcal{M}(q^t), \frac{m+M}{2})$.
      If $b > 0$ update $N^+_{i^*} \leftarrow N^+_{i^*} + 1$, otherwise update $N^-_{i^*} \leftarrow N^-_{i^*} + 1$.
    end
    Let $\Delta p = \mathbb{P}(-\frac{|\mathcal{I}_1|}{8D} \leq \mathcal{E} \leq \frac{|\mathcal{I}_1|}{8D})$, $N_i = N^+_i + N^-_i$ and $N_{crit} = \frac{30\log(T)}{\Delta p^2}$.
    if $N_i \geq N_{crit}$ *for* $i = 1, \ldots, d$ **then**
      Run *ShrinkHyperCube*$(\mathcal{I}_1, \ldots, \mathcal{I}_d, N^+_1, \ldots, N^+_d, N^-_1, \ldots, N^-_d)$.
      Update: $N^+_i \leftarrow 0, N^-_i \leftarrow 0$ for $i = 1, \ldots, d$.
    end
  end
end

As the hypercube shrinks, vector w_{approx} ϵ-approximates w^* for smaller values of ϵ. When the hypercube is large the errors made by the algorithm will be large, but on the other hand larger hypercubes are easier to shrink since they require fewer queries to ensure that hypercube continues to contain w^* (with very high probability) after shrinking. This observation plays a crucial role in establishing upper bounds on the average error made by the algorithm on the sequence of T queries.

After outputting an answer for query-vector q^t, the algorithm checks whether q^t has a large inner product with at least one vector in an orthonormal basis $\mathcal{C} = \{e^1, \ldots, e^d\}$ of $\mathcal{U}$. If so, q^t represents an observation for that basis vector; whether it is a positive or negative observation depends on

the response of the binary oracle $\mathcal{O}$. The threshold given by the algorithm to $\mathcal{O}$ is chosen by solving the linear program $\max_{y \in \mathcal{HC}} q \cdot y$ for $q = q^t$ and $q = -q^t$, where $\mathcal{HC}$ is the current hypercube. As we will see in Section 5, this linear program is simple enough that there is a closed-form expression for its optimal value. So we do not need to use the simplex method or any other linear programming tools.

Algorithm 2 - *ShrinkHyperCube*
Input: $\mathcal{I}_1 = [x_1, y_1], \ldots, \mathcal{I}_d = [x_d, y_d]$, $N_1^+, \ldots, N_d^+$, $N_1^-, \ldots, N_d^-$.
Output: Updated hypercube $(\mathcal{I}_1, \ldots, \mathcal{I}_d)$.
begin
  Let $\alpha = \frac{3}{4}$, $\Delta p = \mathbb{P}(-\frac{|\mathcal{I}_1|}{8D} \leq \mathcal{E} \leq \frac{|\mathcal{I}_1|}{8D})$, $p_1 = \mathbb{P}(\mathcal{E} > \frac{|\mathcal{I}_1|}{8D})$ and $N_i = N_i^+ + N_i^-$.
  for $i = 1, \ldots, d$ **do**
    if $N_i^+ > N_i p_1 + \frac{N_i \Delta p}{2}$ **then**
      $\mathcal{I}_i \leftarrow [y_i - \alpha(y_i - x_i), y_i]$;
    else
      $\mathcal{I}_i \leftarrow [x_i, x_i + \alpha(y_i - x_i)]$;
    end
  end
end

The optimal values m and M of the linear programs solved by the *OnlineBisection* algorithm represent the smallest and largest possible value of the inner product of the query-vector and a vector from the current hypercube. The true value lies in the interval $[m, M]$. By choosing the average of these two values as a threshold for the oracle we are able to effectively shrink direction i^*. The intuition is that if the query-vector forms an angle $\alpha = 0$ with this direction and there is no noise added then by choosing the average we basically perform standard binary search for q. Since α is not necessarily 0 but is relatively small (and noise is added that perturbs the output), the search is not exactly binary. Instead of two disjoint subintervals of I_{i^*} we get two intervals whose union is I_{i^*} but that intersect. Still, each of them is only of a fraction of the length of I_{i^*} and that still enables us to significantly shrink each dimension whenever a sufficient number of observations have been collected for each basis vector — specifically, N_{crit} observations — by calling the *ShrinkHyperCube* subroutine (Algorithm 2).

Every shrinking of the hypercube decreases each edge by a factor α for some $0 < \alpha < 1$. A logarithmic number of shrinkings is needed to ensure that any choice of w_{approx} in the hypercube will give an error of the order $\tilde{O}(\frac{1}{\sqrt{T}})$. Notice that N_{crit} grows with T, which reflects the fact that for smaller hypercubes more observations are needed to further shrink the hypercube while preserving the property that it contains the database-vector w^* with very high probability. This is the case since if the hypercube is small we already know a good approximation of the database vector so it is harder to find even more accurate one under the same level of noise. When the hypercube is small enough (condition: $|\mathcal{I}_i| \leq \frac{\log(T)}{\sqrt{Td}}$ for $i = 1, ..., d$) there is no need to shrink it anymore since each vector taken from the hypercube is a precise enough estimate of the database vector.

Note that choosing an orthonormal basis $\mathcal{C} = \{e^1, \ldots, e^d\}$ of $\mathcal{U}$ does not require the knowledge of the distribution $\mathcal{D}$ from which queries are taken. We only assume that queries are from a low-dimensional linear subspace $\mathcal{U}$ of d dimensions. It suffices to have as $\{e^1, \ldots, e^d\}$ some orthonormal basis of that linear subspace. There are many state-of-the-art mechanisms (such as PCA) that are able to extract such a basis, and thus we will not focus on that, but instead assume that such an orthonormal system is already given. Notice that in practice those techniques should be applied before our algorithm can be run. Since such a preprocessing phase requires sampling from $\mathcal{D}$ but does not require an access to the database system, we can think about it as a preliminary period, where evaluation is not being conducted.

4. THEORETICAL ANALYSIS

In this section we prove Theorem 2.1. We start by introducing several technical lemmas. First we will prove all of them and then we will show how those lemmas can be combined to obtain our main result.

We denote: $h_T = \frac{\sqrt{T}}{\log(T)}$. Thus the stopping condition for shrinking the hypercube is of the form: $|\mathcal{I}_i| \leq \frac{1}{\sqrt{d} h_T}$ for $i = 1, ..., d$.

We start with the concentration result regarding binomial random variables.

LEMMA 4.1. *Let $Z^m = Bin(m, p_1)$, $W^m = Bin(m, p_1 + \Delta p)$ and $\mu_1 = mp_1$.*
Then the following is true:

$$\mathbb{P}(Z^m \geq \mu_1 + \frac{m\Delta p}{2}) \leq e^{-\frac{m(\Delta p)^2}{10}}, \tag{1}$$

$$\mathbb{P}(W^m < \mu_1 + \frac{m\Delta p}{2}) \leq e^{-\frac{m(\Delta p)^2}{10}}. \tag{2}$$

Proof. The proof follows from standard concentration inequalities. Let $\delta_1, \delta_2 > 0$. Note that $E(Z^m) \leq mp_1$ and $E(W^m) \geq mp_1 + m\Delta p$. Denote $\mu_2 = E(W^m)$. Note that by Chernoff's inequality we have:

$$\mathbb{P}(Z^m \geq (1 + \delta_1)\mu_1) \leq e^{-\frac{\delta_1^2}{2+\delta_1}\mu_1} \tag{3}$$

Similarly,

$$\mathbb{P}(W^m \leq (1 - \delta_2)\mu_2) \leq e^{-\frac{\delta_2^2}{2+\delta_2}\mu_2} \tag{4}$$

Take: $\delta_1 = \frac{m\Delta p}{2\mu_1} = \frac{\Delta p}{2p_1}$, $\delta_2 = \frac{m\Delta p}{2\mu_2}$. Using these values of δ_1 and δ_2, we obtain:

$$\mathbb{P}(Z^m \geq \mu_1 + \frac{m\Delta p}{2}) \leq e^{-\frac{1}{1+\frac{2}{\delta_1}}\frac{m\Delta p}{2}} \tag{5}$$

Similarly,

$$\mathbb{P}(W^m < \mu_1 + \frac{m\Delta p}{2}) \leq \mathbb{P}(W^m \leq \mu_2 - \frac{m\Delta p}{2}) \leq e^{-\frac{1}{1+\frac{2}{\delta_2}}\frac{m\Delta p}{2}} \tag{6}$$

Notice that $\delta_1, \delta_2 \geq \frac{\Delta p}{2}$ (the latter inequality holds because obviously: $\mu_2 \leq m$). Thus we get:

$$\mathbb{P}(Z^m \geq \mu_1 + \frac{m\Delta p}{2}) \leq e^{-\frac{m(\Delta p)^2}{2(4+\Delta p)}} \tag{7}$$

and

$$\mathbb{P}(W^m < \mu_1 + \frac{m\Delta p}{2}) \leq e^{-\frac{m(\Delta p)^2}{2(4+\Delta p)}} \tag{8}$$

Since $\Delta p \leq 1$, the proof is completed. ∎

DEFINITION 4.1. *Let $\mathcal{HC}$ be a d-dimensional hypercube in $\mathbb{R}^D$. We denote by $l(\mathcal{HC})$ the length of its side measured according to the L_2-norm (recall that all the sides of a hypercube have the same length).*

Next lemma is central for finding an upper bound on the average error made by the algorithm.

LEMMA 4.2. *Let $(q_1, \ldots, q_T)$ be a sequence of T queries. Let $\mathcal{HC}_0, \ldots, \mathcal{HC}_s$ be a sequence of d-dimensional hypercubes in $\mathbb{R}^D$. Assume that $l(\mathcal{HC}_{i+1}) \leq \alpha l(\mathcal{HC}_i)$ for $i = 0, \ldots, s-1$ and some $0 < \alpha < 1$. Denote $l(\mathcal{HC}_0) = L \leq D$ and assume that $s = \frac{1}{\log_2(\frac{1}{\alpha})} \log_2(L\sqrt{d}h(T))$, where $h(T)$ is some function of T. Assume that $w^* \in \mathcal{HC}_0 \bigcap \cdots \bigcap \mathcal{HC}_s$. Let $\mathcal{E}$ be a random variable defined on the interval $[-u, u]$ for some constant $u > 0$, with density ρ continuous at 0, and such that $\rho(0) > 0$. Define*

$$\phi_\epsilon(i) = \mathbb{P}(-\frac{L\alpha^i(\frac{1}{4} - \epsilon)}{D} < \mathcal{E} \leq \frac{L\alpha^i(\frac{1}{4} - \epsilon))}{D} \tag{9}$$

for some constant $0 < \epsilon \leq \frac{1}{8}$. Let $m_i = \frac{1}{\phi_\epsilon^2(i)} C \log(T)$ for some constant $C > 0$ and let $k_i = m_i r$ for some other constant $r > 0$ and $i = 0, \ldots, s$. Assume that learning algorithm uses a vector $w_{approx} \in \mathcal{HC}_0$ to answer first k_0 queries, a vector $w_{approx} \in \mathcal{HC}_1$ to answer next k_1 queries, etc. Assume also that an algorithm uses a vector $w_{approx} \in \mathcal{HC}_s$ to answer remaining $T - \sum_{i=0}^s k_i$ queries. Then the following is true about the cumulative error ϵ_{cum} made by the algorithm:

$$\epsilon_{cum} = O(L^2 D^{\frac{5}{2}} dr \log(T) h(T) + \frac{\sqrt{D}T}{h(T)}) \tag{10}$$

Proof. Note first that for any d-dimensional hypercube $\mathcal{HC} \in \mathbb{R}^D$ of side length l, two vectors: $w^1, w^2 \in \mathcal{HC}$ and a vector $q = (q_1, \ldots, q_D)$ such that: $q_i = 1$ for $i = 1, \ldots, d$ the following is true: $|w^1 \cdot q - w^2 \cdot q| \leq l\sqrt{dD}$. This comes from the fact that: $\|w^1 - w^2\|_2 \leq l\sqrt{d}$, $\|q\|_2 \leq \sqrt{D}$ and Cauchy-Schwarz inequality. Thus we see that the cumulative error ϵ_{cum}^1 made by the algorithm for the first $\sum_{i=0}^s k_i$ queries satisfies:

$$\epsilon_{cum}^1 \leq \sum_{i=0}^s k_i L\alpha^i \sqrt{dD} \leq L\sqrt{dD} r \sum_{i=1}^s m_i \alpha^i \tag{11}$$

Therefore we have:

$$\epsilon_{cum}^1 \leq CL\sqrt{dD} r \log(T) \sum_{i=0}^s \frac{\alpha^i}{\phi_\epsilon^2(i)} \tag{12}$$

We can write:

$$\epsilon_{cum}^1 \leq CL\sqrt{dD} r \log(T) \sum_{i=0}^t \frac{\alpha^i}{\phi_\epsilon^2(i)} + CL\sqrt{dD} r \log(T) \mathcal{Z},$$

where $\mathcal{Z} = \sum_{i=t+1}^s \frac{\alpha^i}{\phi_\epsilon^2(i)}$ and t is the smallest index such that $\rho(x) \geq \frac{\rho(0)}{\sqrt{2}}$ for $x \in [-\frac{\alpha^t}{8}, \frac{\alpha^t}{8}]$. Since ρ is continuous at 0, t is well-defined. Notice that t does not depend on d, D and T, but only on the random variable $\mathcal{E}$ and constant α. Observe that:

$$CL\sqrt{dD} r \log(T) \sum_{i=0}^t \frac{\alpha^i}{\phi_\epsilon^2(i)} \leq CL\sqrt{dD} r \log(T) \frac{t}{\phi_\epsilon^2(t)} \tag{13}$$

and

$$CL\sqrt{dD} r \log(T) \frac{t}{\phi_\epsilon^2(t)} \leq \frac{2CL\sqrt{d}D^{\frac{5}{2}} r \log(T) t}{\rho^2(0)\alpha^{2t}(\frac{1}{2} - 2\epsilon)^2}, \tag{14}$$

where the last inequality follows immediately from the definition of t (density ρ on the interval considered in the definition of $\phi_\epsilon(t)$ is at least $\frac{\rho(0)}{\sqrt{2}}$ thus the related probability is at least: the length of that interval times $\frac{\rho(0)}{\sqrt{2}}$, i.e.:

$$\phi_\epsilon(t) \geq \frac{\rho(0)}{\sqrt{2}} \frac{L\alpha^i(\frac{1}{2} - 2\epsilon)}{D} \tag{15}$$

Therefore the considered expression is of the order $O(L\sqrt{d}D^{\frac{5}{2}} r \log(T))$. Now let us focus on the expression: $\mathcal{R} = CL\sqrt{dD} r \log(T) \sum_{i=t+1}^s \frac{\alpha^i}{\phi_\epsilon^2(i)}$. From the definition of t we get:

$$\mathcal{R} \leq CL\sqrt{d}D^{\frac{5}{2}} r \log(T) \Pi, \tag{16}$$

where

$$\Pi = \sum_{i=0}^s \frac{2\alpha^i}{\alpha^{2i}(\frac{1}{2} - 2\epsilon)^2 \rho^2(0)} \tag{17}$$

Therefore we have:

$$\mathcal{R} \leq \frac{32CL\sqrt{d}D^{\frac{5}{2}} r \log(T)}{\rho^2(0)} \sum_{i=1}^s \alpha^{-i}. \tag{18}$$

Thus we get:

$$\mathcal{R} \leq \frac{32CL\sqrt{d}D^{\frac{5}{2}} r \log(T)}{\rho^2(0)} \frac{\alpha}{1-\alpha}((\frac{1}{\alpha})^{s+1} - 1), \tag{19}$$

so we also get:

$$\mathcal{R} \leq \frac{32CL\sqrt{d}D^{\frac{5}{2}} r \log(T)}{\rho^2(0)(1-\alpha)\alpha^s} \tag{20}$$

Using the formula on s, we get:

$$\mathcal{R} \leq \frac{32CL^2 D^{\frac{5}{2}} dr \log(T) h(T)}{\rho^2(0)(1-\alpha)} \tag{21}$$

Combining this upper bound on $\mathcal{R}$ with the upper bound on the previous expression, we obtain:

$$\epsilon_{cum}^1 = O(L^2 D^{\frac{5}{2}} dr \log(T) h(T)) \tag{22}$$

Next let us focus on the cumulative error ϵ_{cum}^2 made by the algorithm for the remaining $T - \sum_{i=0}^s k_i$ queries. By the definition of s we know that

$$l(\mathcal{HC}_s) \leq \frac{1}{\sqrt{d}h(T)} \tag{23}$$

This implies that for any $w \in \mathcal{HC}_s$ we have:

$$\|w - w^*\|_2 \leq \frac{1}{h(T)} \tag{24}$$

Thus clearly for any query coming in this phase the learning algorithm makes an error at most $\frac{\sqrt{D}}{h(T)}$ (again, by Cauchy-Schwarz inequality) and we have at most T queries in this phase. Therefore $\epsilon_{cum}^2 = O(\frac{\sqrt{D}}{h(T)}T)$. That completes the entire proof. ∎

In the following lemma we analyze cutting the hypercube according to some linear threshold.

LEMMA 4.3. *Let $w \in \mathbb{R}^D$, let $\{v^1, \ldots, v^d\}$ be a system of pairwise orthogonal vectors such that $v^i \in \mathbb{R}^D$, $\|v^i\|_2 = L$ for $i = 1, .., d$ and let $\mathcal{HC} = \{w + \sum_{i=1}^{d} f_i v^i : f_1, \ldots, f_d \in [0, 1]\}$ be a d-dimensional hypercube. Let e be a unit-length vector in L_2 that is parallel to v^1, i.e. $e = \frac{1}{\sqrt{L}} v^1$. Let z be a unit-length vector satisfying: $z \cdot e \geq \cos(\theta)$ for some $0 < \theta < \frac{\pi}{2}$. Let $0 < \beta < 1$. Define $m = \min_{y \in \mathcal{HC}} y \cdot z$ and $M = \max_{y \in \mathcal{HC}} y \cdot z$. Let $\mathcal{HC}_l = \{y \in \mathcal{HC} : z \cdot y \leq m + \beta(M - m)\}$ and $\mathcal{HC}_r = \{y \in \mathcal{HC} : z \cdot y > m + \beta(M - m)\}$. Then for $\epsilon = 8 \sin(\frac{\theta}{2})\sqrt{d}$:*

$$\max_{y \in \mathcal{HC}_l} e \cdot y - \min_{y \in \mathcal{HC}_l} e \cdot y \leq L(\beta + \epsilon) \tag{25}$$

and

$$\max_{y \in \mathcal{HC}_r} e \cdot y - \min_{y \in \mathcal{HC}_r} e \cdot y \leq L(1 - \beta + \epsilon). \tag{26}$$

Proof. Denote: $\eta = e - z$. Note that

$$\|\eta\|_2 \leq 2 \sin(\frac{\theta}{2}) \tag{27}$$

Take first $y \in \mathcal{HC}_l$. We have:

$$m \leq z \cdot y \leq m + \beta(M - m) \tag{28}$$

Thus we also get:

$$m + \eta \cdot y \leq e \cdot y \leq m + \beta(M - m) + \eta \cdot y \tag{29}$$

Define: $\tilde{m} = \min_{y \in \mathcal{HC}} y \cdot e$ and $\tilde{M} = \max_{y \in \mathcal{HC}} y \cdot e$. Notice that:

$$|\tilde{m} - m| \leq 2 \sin(\frac{\theta}{2}) L\sqrt{d} \tag{30}$$

and besides:

$$|\tilde{M} - M| \leq 2 \sin(\frac{\theta}{2}) L\sqrt{d} \tag{31}$$

This follows directly from:

$$\|y\|_2 \leq L\sqrt{d}, \tag{32}$$

$$\|\eta\|_2 \leq 2 \sin(\frac{\theta}{2}) \tag{33}$$

and Cauchy-Schwarz inequality. Thus we obtain:

$$\tilde{m} - 2 \sin(\frac{\theta}{2}) L\sqrt{d} + \eta \cdot y \leq e \cdot y \tag{34}$$

and

$$e \cdot y \leq \tilde{m} + 2 \sin(\frac{\theta}{2}) L\sqrt{d} + \beta(\tilde{M} - \tilde{m} + 4 \sin(\frac{\theta}{2}) L\sqrt{d}) + \eta \cdot y \tag{35}$$

Since, from the definition of $\tilde{M}, \tilde{m}$ and $\mathcal{HC}$ we have:

$$\tilde{M} - \tilde{m} = L, \tag{36}$$

we obtain:

$$\tilde{m} - 2 \sin(\frac{\theta}{2}) L\sqrt{d} + \eta \cdot y \tag{37}$$

and

$$e \cdot y \leq \tilde{m} + 2 \sin(\frac{\theta}{2}) L\sqrt{d} + \beta(L + 4 \sin(\frac{\theta}{2}) L\sqrt{d}) + \eta \cdot y \tag{38}$$

Therefore

$$\max_{y \in \mathcal{HC}_l} e \cdot y - \min_{y \in \mathcal{HC}_l} e \cdot y \leq L(\beta + 8 \sin(\frac{\theta}{2})\sqrt{d}) \tag{39}$$

This completes the proof of inequality 25. The proof of inequality 26 is completely analogous. ∎

We are ready to prove Theorem 2.1.

Proof. Let $L = 2\sqrt{D}$. Let us notice that the algorithm can be divided into $s + 1$ phases, where in the i^{th} phase $(i = 0, \ldots, s)$ all the intervals $\mathcal{I}_i$ are of length $L\alpha^{-i}$ and

$$s = \frac{1}{\log_2(\frac{1}{\alpha})} \log_2(L\sqrt{d} h_T) \tag{40}$$

Indeed, whenever the shrinking is conducted, the length of each side of the hypercube decreases by a factor $\frac{1}{\alpha}$ (see subroutine *ShrinkHyperCube*), the initial lengths are $2\sqrt{D}$ and the shrinking is not performed anymore if the side of each length is at most $\frac{1}{\sqrt{d} h_T}$. We will call those phases: *1st-phase*, *2nd-phase*, etc. Notice also that the value of the parameter N_{crit} is constant across a fixed phase since this number changes only when *ShrinkHyperCube* subroutine is performed. Let us denote the value of N_{crit} during the i^{th} phase of the algorithm as n_i. Notice that

$$n_i = \frac{30 \log(T)}{\Delta p_i^2}, \tag{41}$$

where Δp_i is the value of the parameter Δp of the algorithm used in the i^{th} phase. Denote by k_i the number of queries that need to be processed in the i^{th} phase for $i = 0, \ldots, s-1$. Parameter k_i is a random variable but we will show later that with high probability: $k_i \leq n_i r$ for $i = 0, \ldots, s - 1$, where:

$$r = \frac{2}{(p_{\mathcal{D},\phi})^2} \tag{42}$$

Assume now that this is the case. Denote by $\mathcal{HC}_0, \ldots, \mathcal{HC}_s$ the sequence of hypercubes constructed by the algorithm. Assume furthermore that $w^* \in \mathcal{HC}_0 \cap \cdots \cap \mathcal{HC}_s$. Again, we have not proved it yet, we will show that this happens with high probability later. However we will prove now that under these two assumptions we get the average error proposed in the statement of Theorem 2.1. Notice that under these assumptions we can use Lemma 4.2 with $L = 2\sqrt{D}$, $h(T) = h_T$, $\phi_\epsilon(i) = \Delta p_i$, $C = 30$, $m_i = n_i$. We get the following bound on the cumulative error:

$$\epsilon_{cum} = O(D^{\frac{7}{2}} dr \log(T) h(T) + \frac{\sqrt{d} T}{h(T)}). \tag{43}$$

Thus the average error is at most

$$\epsilon_{av} \leq \frac{\epsilon_{cum}}{T} \tag{44}$$

By using the expression $h(T) = \frac{\log(T)}{\sqrt{T}}$ in the above formula, we obtain the bound from the statement of Theorem 2.1.

It remains to prove that our two assumptions are correct with high probability and find a lower bound on this probability that matches the one from the statement of the theorem. We will do it now. Let us focus on the i^{th} phase of the algorithm. First we will find an upper bound on the probability that the number of queries processed in this phase is greater than k_i. Fix a vector e^j from the orthonormal basis $\mathcal{C}$. The probability that a new query q is within angle

ϕ from e^j is at least $p = p_{\mathcal{D},\phi}$, by the definition of $p_{\mathcal{D},\phi}$. Assume that u_i queries were constructed. By standard concentration inequalities, such as Azuma's inequality, we can conclude that with probability at least $1-e^{-2u_i(\frac{p}{2})^2}$ at least $\frac{u_i p}{2}$ of those queries will be within angle ϕ from e^j. If we take:

$$u_i \geq \frac{2n_i}{p}, \tag{45}$$

then we conclude that with probability at least $1-e^{-2u_i(\frac{p}{2})^2}$ at least n_i of those queries will be within angle ϕ from e^j. Denote $u_i = n_i r$, where $r > \frac{2}{p}$. We see that the considered probability is at least $1 - e^{-\frac{p^2}{2}n_i r}$. Using the expression on n_i we get that this probability is at least $1 - e^{-30r\frac{p^2}{2}\log(T)}$. Notice that when n_i queries within angle ϕ from a given vector $e^j \in \mathcal{C}$ are collected, the j^{th} dimension is ready for shrinking. Thus taking union bound over $O(\log(dT))$ phases and all d dimensions we see that if we take $k_i = rn_i$, where: $r = \frac{2}{p^2}$, then with probability at most $\frac{d\log(dT)}{T^{30}}$ some i^{th} phase of the algorithm for $i \in \{0,\ldots,s-1\}$ will require more than k_i queries. Now let us focus again on the fixed i^{th} phase of the algorithm. Assume that *ShrinkHyperCube* subroutine is being run. Fix some dimension $j \in \{1,\ldots,d\}$. We know that, with high probability, at least n_i queries q that were within angle ϕ from the vector $e^j \in \mathcal{C}$ were collected. Denote by w_j^* the j^{th} coordinate of w^*. Let $\mathcal{I}_j = [x_j, y_j]$ and assume that $w_j^* \in [x_j, y_j]$. Let us assume that the *ShrinkHyperCube* subroutine replaced $\mathcal{I}_j = [x_j, y_j]$ by $\tilde{\mathcal{I}}_j$. We want to show that with high probability segment $\tilde{\mathcal{I}}_j$ is constructed in such a way that $w_j^* \in \tilde{\mathcal{I}}_j$. Denote

$$l = y_j - x_j \tag{46}$$

and

$$\delta = (\alpha - \frac{1}{2})l \tag{47}$$

Notice first that if $w_j^* \in [(1-\alpha)l, \alpha l]$ then w_j^* will be in $\tilde{\mathcal{I}}_j$ since no matter how $\tilde{\mathcal{I}}_j$ is constructed, it always contains $[(1-\alpha)l, \alpha l]$. So let us assume that this is not the case. Thus we have either

$$w_j^* \in [x_j, x_j + (1-\alpha)l] \tag{48}$$

or

$$w_j^* \in [y_j - (1-\alpha)l, y_j] \tag{49}$$

Let us assume first the former. Consider a query-vector q within angle ϕ of e^j that contributed to N_j^+. Let us denote by p_+ the probability of the following event $\mathcal{F}_q$: for q the oracle $\mathcal{O}$ gives answer: "greater than 0". Observe that the total error made by the database mechanism $\mathcal{M}$ while computing the dot-product: $w^* \cdot q$ is $D\mathcal{E}$. Now notice, that by Lemma 4.3 and the definition of $\mathcal{E}$, probability p_+ is at most $\mathbb{P}(D\mathcal{E} > \delta - \epsilon l)$, where: $\epsilon = 8\sin(\frac{\phi}{2})\sqrt{d} = \frac{1}{8}$. Thus we get:

$$p_+ \leq \mathbb{P}(\mathcal{E} > \frac{(\alpha - \frac{1}{2} - \epsilon)l}{D}) \tag{50}$$

Notice that in the i^{th} phase the hypercube under consideration has the side of length exactly α^i. Thus, since $\alpha = \frac{3}{4}$, we get:

$$p_+ \leq \mathbb{P}(\mathcal{E} > \frac{(\frac{1}{4} - \epsilon)L\alpha^i}{D}) \tag{51}$$

Let us assume now that $w_j^* \in [y_j - (1-\alpha)l, y_j]$. We proceed with the similar analysis as before. We see that the probability P_+ of an event $\mathcal{F}_q$ is at least $\mathbb{P}(D\mathcal{E} \geq -\delta + \epsilon l)$. Thus we obtain:

$$P_+ \geq \mathbb{P}(\mathcal{E} \geq -\frac{(\frac{1}{4} - \epsilon)L\alpha^i}{D}) \tag{52}$$

But now we see, by Lemma 4.1, using: $m = N_i$, $p_1 = \mathbb{P}(\mathcal{E} > \frac{(\frac{1}{4}-\epsilon)L\alpha^i}{D})$ and

$$\Delta p = \mathbb{P}(\frac{-(\frac{1}{4} - \epsilon)L\alpha^i}{D} \leq \mathcal{E} \leq \frac{(\frac{1}{4} - \epsilon)L\alpha^i}{D}) \tag{53}$$

that $N_i^+ > N_i p_1 + \frac{N_i \Delta p}{2}$ is satisfied if $w_j^* \in [x_j, x_j + (1-\alpha)(y_j - x_j)]$ with probability at most $e^{-\frac{n_i(\Delta p)^2}{10}}$. Similarly, $N_i^+ \leq N_i p_1 + \frac{N_i \Delta p}{2}$ is satisfied if if $w_j^* \in [y_j, y_j - (1-\alpha)(y_j - x_j)]$ with probability at most $e^{-\frac{n_i(\Delta p)^2}{10}}$. We can use Lemma 4.1 since (as it is easy to notice) in the i^{th} phase Δp is exactly

$$\Delta p_i = \mathbb{P}(-\frac{(\frac{1}{4} - \epsilon)L\alpha^i}{D} \leq \mathcal{E} \leq \frac{(\frac{1}{4} - \epsilon)L\alpha^i}{D}) \tag{54}$$

and p_1 is exactly

$$p_1 = \mathbb{P}(\mathcal{E} > \frac{(\frac{1}{4} - \epsilon)L\alpha^i}{D}) \tag{55}$$

We obtain the following: the probability that there exists i such that $w^* \notin \mathcal{HC}_0 \cap \cdots \cap \mathcal{HC}_i$ is at most: $O(\sum_{i=0}^{s} e^{-\frac{n_i(\Delta p_i)^2}{10}})$. Substituting in that expression the formula on n_i, and noticing that the number of all the phases of the algorithm is logarithmic in T, D and d, we get the bound $O(\frac{\log(dDT)}{T^3})$. Thus, according to our previous remarks, we conclude that with probability at least $1 - O(\frac{\log(dDT)}{T^3} + \frac{d\log(dT)}{T^{30}})$ *OnlineBisection* algorithm makes an average error at most:

$$\epsilon_{av} = O(\frac{1}{T}(D^3 d^{\frac{3}{2}} r \log(T) h_T + \frac{\sqrt{d}T}{h_T})) \tag{56}$$

As mentioned before, we complete the proof by using the formula:

$$h_T = \frac{\sqrt{T}}{\log(T)} \tag{57}$$

∎

5. ANALYSIS OF THE RUNNING TIME OF THE ALGORITHM AND MEMORY USAGE

We start with the analysis of the running time of *OnlineBisection*. First we will show that the linear program used by the algorithm to determine the threshold in each round has a closed form solution.

LEMMA 5.1. *For any query-vector q, $\mathcal{I}_1 = [x_1, y_1], \ldots, \mathcal{I}_d = [x_d, y_d]$ and orthonormal basis $\mathcal{C} = \{e^1, \ldots, e^d\}$ the value*

$$\max_{f_1 \in \mathcal{I}_1, \ldots, f_d \in \mathcal{I}_d} \sum_{i=1}^{d} f_i e^i \cdot q$$

is given by

$$opt = \sum_{j \in \mathcal{J}_+} y_j e^j \cdot q + \sum_{j \in \mathcal{J}_-} x_j e^j \cdot q$$

where $\mathcal{J}_+ = \{i \in \{1,\ldots,d\} : e^i \cdot q \geq 0\}$ *and* $\mathcal{J}_- = \{i \in \{1,\ldots,d\} : e^i \cdot q < 0\}$.

Proof. Take some point: $c_1 e^1 + \cdots + c_d e^d$, where: $x_i \leq c_i \leq y_i$ for $i = 1,\ldots,d$. For $j \in \mathcal{J}_+$ the following is true: $c_j e^j \cdot q \leq y_j e^j \cdot q$, since: $c_j \leq y_j$ and $e^j \cdot q \geq 0$. Similarly, for $j \in \mathcal{J}_-$ we have: $c_j e^j \cdot q \leq x_j e^j \cdot q$, again by the definition of $\mathcal{J}_-$. Combining these inequalities we get that for every point v in the hypercube $\mathcal{HC}$ induced by $\mathcal{I}_1,\ldots,\mathcal{I}_d$ and $\mathcal{C}$ the following is true: $v \cdot q \leq opt$. Besides clearly there exists $v^* \in \mathcal{HC}$ such that: $v^* \cdot q = opt$. ∎

Now let us fix a query q. It is easy to notice that q is being processed by the algorithm in $O(dD)$ time. Indeed, a single query requires updating $O(d)$ variables of the form N_i^+, N_i^- and computing the closed-form solution given in Lemma 5.1 in $O(dD)$ time. Computing dot product of the query with the given approximation of the database vector clearly takes $O(D)$ time. Thus *OnlineBisection* runs in the $O(dD)$-time per query. Notice that *OnlineBisection* algorithm does not store any nontrivial data structures, only segments: $\mathcal{I}_1,\ldots,\mathcal{I}_d$, counts: N_i^+, N_i^- for $i = 1,\ldots,d$ and a constant number of other variables. The counts can be represented by $O(\log(T))$-digit numbers thus we conclude that *OnlineBisection* runs in the $O(\log(T))$-memory.

6. OPTIMALITY OF THE ALGORITHM

In this section we prove a negative result showing that up to the $poly(\log(T))$ factor no algorithm (offline or online) can beat *OnlineBisection* in respect to the achieved error. We prove that this is the case even if the oracle is turned off and the perturbed answer is given directly to the adversary.

THEOREM 6.1. *In the considered database model (but with online mode possibly turned off) any algorithm achieves error at least* $\tilde{O}(\frac{1}{\sqrt{T}})$ *in the worst case scenario.*

Proof. The proof is a direct consequence of the tightness of Azuma's inequality. We will prove that the result holds even for $D = 2$. Our queries q will be of the form $(x, 1)$. Consider a database vector $w = (a, b)$ for some $b > 0$. The perturbed answer that the oracle receives is of the form $ax + b + \epsilon_q$, where ϵ_q is the error added to the exact answer for the query q. Obviously, it suffices to show our result if the oracle is turned off and the adversary receives $ax + b + \epsilon_x$ instead of the binary signal.

Let $\{q_i = (x_i, 1) : i = 1, ..., T\}$ be the set of all the queries. Intuitively, we want to show that for T queries there always exists a vector $w_{near} = (a, b_{near})$, where: $b_{near} - b = \Omega(\frac{1}{\sqrt{T}\log(T)})$ such that with probability almost one the adversary will not be able to distinguish between w and w_{near} based on the outcome for these T queries.

Let us analyze the following expression: $E = \frac{\sum_{i=1}^{T} \epsilon_{q_i}}{T}$. Notice that from Azuma's inequality we know that $\mathbb{P}(E > \frac{f(T)}{\sqrt{T}}) = o(1)$ for any increasing positive function $f(T)$. We heavily used this fact before. However, from the tightness of Azuma's inequality (see: [17]) we also know that there exists a symmetric bounded distribution $\mathcal{Z}$ such that if every ϵ_q is taken from $\mathcal{Z}$ then $\mathbb{P}(E > \frac{1}{\sqrt{T}f(T)}) = 1 - o(1)$, where $f(T) = \log(T)$.

Let us assume that this is the case, i.e. $E > \frac{1}{\sqrt{T}\log(T)}$. But then one can easily check that there exists w_{near} (and related distribution determining the amount of error being added to each answer) that gives the same perturbed answers as w. Thus, given this set of queries the adversary will incur an error of at least $\theta(\frac{1}{\sqrt{T}\log(T)})$ from the exact answer given by at least one of the two database mechanisms determined by vectors: w and w_{near} for each asked query (since the distance between w and w_{near} is of order $\theta(\frac{1}{\sqrt{T}\log(T)})$). Thus, by the Pigeonhole Principle, for at least one of the two mechanisms the adversary will achieve average error at least $\frac{\frac{T}{2}\cdot\theta(\frac{1}{\sqrt{T}\log(T)})}{T} = \theta(\frac{1}{\sqrt{T}\log(T)})$.

That completes the proof since both database mechanisms are legitimate mechanisms that can communicate with the adversary. ∎

7. ONLINE-TO-BATCH CONVERSION

Throughout the paper we have considered the challenging online scenario, where the algorithm both learns and is evaluated on a single set of streaming queries. However, we note that the *OnlineBisection* algorithm also works well in the batch setting, i.e. when there is a separate train and test phase. We prove here Corollary 2.1, that for clarity we state once more:

COROLLARY 7.1. *Let* w_T *denote the final hypothesis constructed by the OnlineBisection algorithm after consuming* T *queries drawn from an unknown distribution* $\mathcal{D}$. *Then the following inequality holds with probability at least* $1 - O\Big(\frac{\log(dDT)}{T^3} + \frac{d\log(dT)}{T^{30}}\Big)$ *for any future queries* q *drawn from* $\mathcal{D}$*:*

$$\mathrm{E}_{q\sim\mathcal{D}}\big[|w_T \cdot q - w^* \cdot q|\big] \leq \frac{\sqrt{D}\log(T)}{\sqrt{T}}\,.$$

PROOF. This simply follows from the fact that, as argued in the proof of Theorem 2.1, $w^* \in \mathcal{HC}_s$ with at least the probability indicated in the statement of this corollary. Furthermore, by definition of the algorithm, we have $w_T \in \mathcal{HC}_s$ and the length of the side of the hypercube $\mathcal{HC}_s \leq \log(T)/\sqrt{T}$. Thus, with at least the probability indicated, $|w_T \cdot q - w^* \cdot q| \leq \|w_t - w^*\|_2 \|q\|_2 \leq \frac{\log(T)}{T}\sqrt{D}$. □

8. CONCLUSIONS AND FUTURE WORK

We presented in this paper the first $\tilde{O}(\frac{1}{\sqrt{T}})$-error algorithm for database reconstruction. It is adapted to the highly challenging, yet very realistic setting, where the answers given by the database are heavily perturbed by a random noise and besides there exists strong privacy mechanism (binary oracle $\mathcal{O}$) that aims to protect the database against an adversary attempting to compromise it. We show that even if the learning algorithm receives only binary answers on the database side and needs to learn database-vector w^* with high precision at the same time it is being evaluated, it can still achieve very small average error. We assume that the query-space is low-dimensional but this fact is needed only to guarantee that the term $r = \frac{2}{p^2_{\mathcal{D},\phi}}$ from the bound on the error is not exponential in D. The low-dimensionality assumption is indispensable here if one wants to achieve average error of the order $o(1)$ and considers nontrivial models with random noise. It is also worth to mention that if no noise is added, the low-dimensionality query-space is not required and a simple modification of our algorithm enables to get rid of the $\frac{2}{p^2_{\mathcal{D},\phi}}$-term. Our algorithm operates in the

very limited (logarithmic) memory and is very fast ($O(d)$ processing time per query). By not using linear programming we obtain better theoretical bounds regarding running time of the algorithm than previously considered methods. *OnlineBisection* algorithm adapts next threshold values sent to the binary oracle $\mathcal{O}$ to its previous answers in order to obtain good approximation of the projection of a database-vector w^* onto the low-dimensional query-space $\mathcal{U}$.

It would be interesting to know whether the assumption about the low-dimensional querying model is indeed necessary or may be at least relaxed. Another area that can be explored is the application of the presented method to machine learning settings other than adversarial machine learning. It seems that presented technique provides a general mechanism for efficient online information retrieval. Finally, the authors plan also to extend presented algorithm to work in the setting, where no independence assumption for the mechanism of noise addition is required.

9. REFERENCES

[1] Cynthia Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.

[2] Cynthia Dwork. Differential privacy in new settings. In Moses Charikar, editor, *SODA*, pages 174–183. SIAM, 2010.

[3] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In Leonard J. Schulman, editor, *STOC*, pages 715–724. ACM, 2010.

[4] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In Andrew Chi-Chih Yao, editor, *ICS*, pages 66–80. Tsinghua University Press, 2010.

[5] Kobbi Nissim, Rann Smorodinsky, and Moshe Tennenholtz. Approximately optimal mechanism design via differential privacy. *CoRR*, abs/1004.2888, 2010.

[6] Krzysztof Choromanski and Tal Malkin. The power of the dinur-nissim algorithm: breaking privacy of statistical and graph databases. In *PODS*, pages 65–76, 2012.

[7] Sergey Yekhanin. Private information retrieval. *Commun. ACM*, 53(4):68–73, 2010.

[8] Cynthia Dwork, Frank McSherry, and Kunal Talwar. The price of privacy and the limits of LP decoding. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 85–94. ACM, 2007.

[9] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210. ACM, 2003.

[10] Cynthia Dwork and Sergey Yekhanin. New efficient attacks on statistical disclosure control mechanisms. In *CRYPTO*, pages 469–480, 2008.

[11] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *STOC*, pages 537–546, 2008.

[12] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[13] Richard G. Baraniuk, Volkan Cevher, and Michael B. Wakin. Low-dimensional models for dimensionality reduction and signal recovery: A geometric perspective. pages 959–971, 2010.

[14] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 99–108, New York, NY, USA, 2004. ACM.

[15] Yevgeniy Vorobeychik and Bo Li. Optimal randomized classification in adversarial settings. In *AAMAS*, pages 485–492, 2014.

[16] Kevin G. Jamieson, Maya R. Gupta, Eric Swanson, and Hyrum S. Anderson. Training a support vector machine to classify signals in a real environment given clean training data. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2010, 14-19 March 2010, Sheraton Dallas Hotel, Dallas, Texas, USA*, pages 2214–2217, 2010.

[17] S. Ross. Stochastic processes. *Wiley*, 1996.