Purifying Data by Machine Learning with Certainty Levels

(Extended Abstract)

Shlomi Dolev
Computer Science Department,
Ben Gurion University, Israel
Email: dolev@cs.bgu.ac.il

Guy Leshem
Computer Science Department,
Ben Gurion University
and Ashkelon Academic College, Israel
Email: leshemg@cs.bgu.ac.il

Reuven Yagel
Computer Science Department,
Ben Gurion University,
and Department of Software Engineering,
The Jerusalem College of Engineering, Israel
Email: yagel@cs.bgu.ac.il

***Abstract*—A fundamental paradigm used for autonomic computing, self-managing systems, and decision-making under uncertainty and faults is machine learning. Machine learning uses a data-set, or a set of data-items. A data-item is a vector of feature values and a classification. Occasionally these data sets include misleading data items that were either introduced by input device malfunctions, or were maliciously inserted to lead the machine learning to wrong conclusions. A reliable learning algorithm must be able to handle a corrupted data-set. Otherwise, an adversary (or simply a malfunctioning input device that corrupts a portion of the data-set) may lead to inaccurate classifications. Therefore, the challenge is to find effective methods to evaluate and increase the certainty level of the learning process as much as possible. This paper introduces the use of a certainty level measure to obtain better classification capability in the presence of corrupted data items. Assuming a known data distribution (e.g., a normal distribution) and/or a known upper bound on the given number of corrupted data items, our techniques define a certainty level for classifications. Another approach suggests enhancing the random forest techniques to cope with corrupted data items by augmenting the certainty level for the classification obtained in each leaf in the forest. This method is of independent interest, that of significantly improving the classification of the random forest machine learning technique in less severe settings.**

***key words* – Data corruption, PAC learning, Machine learning, Certainty level**

I. Introduction

Motivation. A fundamental paradigm used for autonomic computing, self-managing systems, and decision-making under uncertainty and faults is machine learning. Classification of machine learning algorithms that are designed to deal with Byzantine (or malicious) data are of great interest since a realistic model of learning from examples should address the issue of Byzantine data.

Acknowledgment

∗ Partially supported by ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS), Vaatat, Rita Altura Trust Chair in Computer Sciences and Lynne and William Frankel Center for Computer Sciences.

Previous work, as described below, tried to cope with this issue by developing new algorithms using a boosting algorithm (e.g., "AdaBoost", "Logitboost" etc.) or other robust and efficient learning algorithms e.g., (Servedio, 2003 [15]). These efficient learning algorithms tolerate relatively high rates of corrupted data. In this paper we try to handle the issue using a different approach, that of introducing the *certainty level* measure as a tool for coping with corrupted data items, and of combining learning results in a new and unique way. We present two new approaches to increase the certainty levels of machine learning results by calculating a certainty level that takes into account the corrupted data items in the training data-set file. The first scheme is based on identifying statistical parameters when the distribution is known (e.g., normal distribution) and using an assumed bound on the number of corrupted data items to bound the uncertainty in the classification. The second scheme uses decision trees, similar to the random forest techniques, incorporating the certainty level to the leaves. The use of the certainty level measure in the leaves yields a better collaborative classification when results from several trees are combined to a final classification. One practical example is in the scope of sensor networks, where the tiny and computational limited sensors may act in unreliable fashion transmitting wrong information over time. For example, a sensor may be deployed up-side down and constantly transmit misleading information.

Previous work. In the Probably Approximately Correct (PAC) learning framework, Valiant (Valiant, 1984) introduced the notion of PAC learning in the presence of malicious noise. This is a worst-case model of errors in which some fraction of the labeled examples given to a learning algorithm may be corrupted by an adversary who can modify both example points and labels in an arbitrary fashion. The frequency of such corrupted examples is known as the malicious noise rate. This study assumed that there is a fixed probability β $(0 < \beta < 1)$ of an error occurring independently on each request, but the error is of an arbitrary nature. In particular, the error may be chosen by an adversary with unbounded computational

resources and knowledge of the function being learned, the probability distribution and the internal state of the learning algorithm (note that in the standard PAC model the learner has access to an oracle returning some labeled instance (*x,C(x)*)for each query, where *C(x)* is some fixed concept belonging to a given target class *C* and *x* is a randomly chosen sample drawn from a fixed distribution *D* over the domain *X*. Both *C* and *D* are unknown to the learner and each randomly drawn *x* is independent of the outcomes of the other draws.

In the malicious variant of the PAC model introduced by Kearns and Li (1993), the oracle is allowed to 'flip a coin' for each query with a fixed bias η for heads. If the outcome is heads, the oracle returns some labeled instance (x,ℓ) antagonistically chosen from $X \times \{$-1,+1$\}$. If the outcome is tails, the oracle is forced to behave exactly like in the standard model returning the correctly labeled instance (*x,C(x)*) where $x \sim D$ (*x* is a drawn sample from the distribution *D*).

In both the standard and malicious PAC models the learner's goal for all inputs ε, $\Delta > 0$ is to output some hypothesis $H \in \mathcal{H}$ (where $\mathcal{H}$ is the learner's fixed hypothesis class) by querying an oracle at most *m* times for some $m = m(\varepsilon, \Delta)$ in the standard model, and for some $m = m(\varepsilon, \Delta, \eta)$ in the malicious model. For all targets $C \in \mathcal{C}$ and distributions *D*, the hypothesis *H* of the learner must satisfy $E_{x \sim D}[H(x) \neq C(x)] \leq \varepsilon$ with a probability of at least $1 - \Delta$ with respect to the oracle's randomization. We will call ε and Δ the accuracy and the confidence parameter, respectively. Kearns and Li (1993) have also shown that for many classes of Boolean functions (concept classes), it is impossible to accurately learn ε if the malicious noise rate exceeds $\frac{\varepsilon}{1+\varepsilon}$. In fact, for many interesting concept classes, such as the class of linear threshold functions, the most efficient algorithms known can only tolerate malicious noise rates significantly lower than this general upper bound.

Despite these difficulties, the importance of being able to cope with noisy data has led many researchers to study PAC learning in the presence of malicious noise (Aslam and Decatur (1998) [1], Auer (1997) [2], Auer and Cesa-Bianchi (1998) [3], Cesa-Bianchi et al. (1999) [7], Decatur (1993) [8], Mansour and Parnas (1998) [11], Servedio (2003) [15]. In Servedio (2003) [15], a PAC boosting algorithm is developed using smooth distributions. This algorithm can tolerate low malicious noise rates but requires access to a noise-tolerant weak learning algorithm of known accuracy. This weak learner, *L*, which takes as input a finite sample *S* of *m* labeled examples, has some tolerance to malicious noise; specifically, *L* is guaranteed to generate a hypothesis with non-negligible advantage provided that the frequency of noisy examples in its sample is at most 10% and that it has a high probability to learn with high accuracy in the presence of malicious noise at a rate of 1%.

Our contribution. We present a verifiable way to cope with arbitrary faults introduced by even the most sophisticated adversary, and show that the technique withstands this malicious (called Byzantine) intervention so that even in the worst case scenario the desired results of the machine learning algorithm can be achieved. The assumption is that an unknown part of a data-set is Byzantine, namely, introduced to mislead the machine learning algorithm as much as possible. Our goal is to show that we can ignore/filter the influence of the misleading portions of the malicious data-set and obtain meaningful (machine learning) results. In reality, the Byzantine portion in the data-set may be introduced by a malfunctioning device with no adversarial agenda, nevertheless, a technique proven to cope with the Byzantine data items will also cope with less severe cases. In this paper, we develop three new approaches for increasing the certainty level of the learning process, where the first two approaches identify and/or filter data items that are suspected to be Byzantine data items in the data-set (e.g., a training file). In the third approach we introduce the use of the certainty level for combining machine learning techniques (similar to the previous studies).

The first approach fits best the case in which the Byzantine data is added to the data-set, and is based on the calculation of the statistical parameters of the data-set. The second approach considers the case where part of the data is Byzantine, and extends the use of the certainty level for those cases in which no concentrations of outliers are identified. Data-sets often have several features (or attributes) which are actually columns in the training and test files that are used for cross-checks and better prediction of the outcome in both simple and sophisticated scenarios. The third approach deals with cases in which the Byzantine data is part of the data and appear in two possible modes: where part of the data in a feature is Byzantine and/or where several features are entirely Byzantine. The third technique is based on decision trees similar to the *Random Forest* algorithm (Breiman, 1999 [5]). After the decision trees are created from the training data, each variable from the training data passes through these decision trees, and whenever the variable arrives to a tree leaf, its tree classification is compared with its class. When the classification and the class are in agreement, a *right* variable of the leaf is incremented; otherwise, the value of a *wrong* variable of this leaf is incremented. The final classification for every variable will be determined according to the right and wrong values. This enhancement of the random forest is of an independent interest conceptually and practically, improving the well known random forest technique.

Road map. The rest of the paper is organized as follows: In the next section (Section 2), we describe approaches for those cases in which Byzantine data items are added to the data-set, and the ways to identify statistical parameters when the distribution of a feature is known. In Sections 3 and 4, we present those cases in which the Byzantine adversary receives the data-set and chooses which items to add/corrupt. Section 3 describes ways to cope with Byzantine data in the case of a single feature with a classification of a given certainty level. Section 4 extends the use of the certainty level to handle several features, extending and improving the random forest techniques. The conclusion appears in Section 5. Experiment

results are omitted from this extended abstract and can be found in [9].

II. Addition of Byzantine Data

We start with the cases in which Byzantine data is added to the data-set. Our goal is to calculate the statistical parameters of the data-set, such as the distribution parameters of the uncorrupted items in the data-set, despite the addition of the Byzantine data. Consider the next examples that derive the learning algorithm to the wrong classification, where the raw data contains one feature (or attribute) of the samples (1 vector) that obeys some distribution (e.g., normal distribution), plus additional adversary data. The histogram that describes such an addition is presented on the left side of Figure 1, where the "clean" samples are inside the curve and the addition of corrupted data is outside the curve (marked in blue). The corrupted data items in these examples are defined

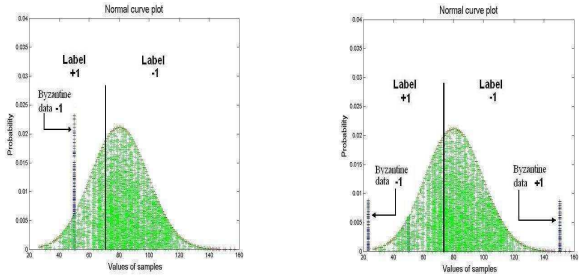

Fig. 1. Histogram of original samples with additional corrupted data outside the normal curve but in the bound of $\mu \pm 3\sigma$ (left), and outside the normal curve and outside the bound of $\mu \pm 3\sigma$ (right).

as samples that cause miscalculation of statistical parameters like μ and σ and as a result, the statistical variables are less significant. Another case of misleading data added to the data-set, a special case to the one above, is demonstrated on the right side of Figure 1. The histogram of these samples is marked in green, where the black vertical line that crosses the histogram separates samples with labels $+1$ and -1. The labels of the misleading data are inverted with relation to the labels of other data items with the same value. To achieve our goal to calculate the most accurate statistical parameters for the feature's distribution in the sample population, we describe a general method to identify and filter the histograms that may include a significant number of additional corrupted data items.

Method for Identifying Suspicious Data and Reducing the Influence of Byzantine Data. This first approach is based on the assumption that we can separate "clean" data by a procedure based on the calculation of the μ and σ parameters of the uncorrupted data. According to the central limited theorem, 30 data items chosen uniformly, which we call a $batch$, can be used to define the μ and σ. Thus, the first step is to try to find at least 30 clean samples (with no Byzantine data). Note that according to the central limit theorem, the larger the set of samples, the closer the distribution is to being normal, therefore, one may choose to select more than 30 samples. We use n=30 as a cutoff point and assume that the sampling distribution is approximately normal. In the presence of Byzantine data one should try to ensure that the set of 30 samples will not include any Byzantine items. This case is similar to the case of a shipment of N objects (real data) in which m are defective (Byzantine). In probability theory and statistics, hypergeometric distribution describes the probability that in a sample of n distinctive objects drawn from the shipment, exactly k objects are defective. The probability for selecting k items that are not Byzantine is:

$$P(X=k) = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}} \quad (1)$$

Note that for clean samples k=0 and the equation will be

$$P(X=0) = \frac{\binom{N-m}{n}}{\binom{N}{n}} \quad (2)$$

In order to prevent the influence of the adversary on the estimation of μ and σ (by addition of Byzantine data), we require that the probability in equation 2 will be higher than 50% ($P > \frac{1}{2}$). Additionally, according to the Chernoff bound we will obtain a lower bound for the success probability of the majority of n independent choices of 30-sample batches (thus, by a small number of batch samplings we will obtain a good estimation for the μ and σ parameters of clean batches). The ratio between N (all samples) to m (Byzantine samples) that implies a probability to sample a clean batch that is greater than $\frac{1}{2}$ is presented in Figure 2:

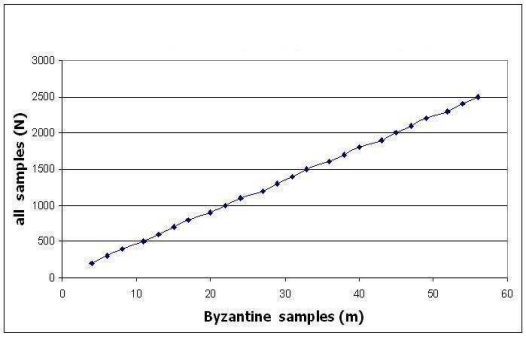

Fig. 2. Ratio between N (all samples) to m (Byzantine samples) for P $\geq \frac{1}{2}$.

As demonstrated in Figure 2 the ratio between N (all samples), and m (Byzantine samples) is about 2% (e.g., 20 Byzantine samples for every 1000 samples, so if this Byzantine ratio is found by the new method (as described below) the probability that any other column in the data-set will contain a Byzantine sample is very low (in other words, the confidence that in every other column the samples are "clean" is high)). Our goal is to sample a majority of

"clean" batches to estimate statistical parameters such as μ and σ of the non-Byzantine samples in the data-set. The estimation of these parameters will be done according to an **Algorithm 1**, presently in the full version of this paper [9].

Using Expected Value and Variance to Predict Distribution Shape. Up to this stage, we used the central limit Theorem (CLT), stating that: the average samples of observations uniformly drawn from some population with any distribution shape is approximately distributed as a normal distribution, resulting in the expected value and the variance. Based on CLT, we were able to efficiently obtain (using Chernoff bound) the expected value and the variance of the data item values. Next, for every given number of data items, and type of distribution graph, the parameters of the graph that will respect these values (expected value, variance, distribution type, and number of data items) can be found. In the sequel, we consider the case of a distribution type of graph which reflects the normal distribution. The next stage for identifying suspicious data items is based on analysis of the overflow of data items beyond the distribution curve (Figure 1). The statistical parameters which were found in the previous stage are used in the procedure described in **Algorithm 2**, presently in the full version of this paper [9].

The suspicious bins, those with a significant overflow, are marked and will not be considered for the training process of the machine learning. The data-set after the cleaning process contains values from bins (in the data histogram) without overflow (e.g., the ratio between the integral of the normal curve to the data items in the same bin is approximately 1). *Note that when the number of extra data items in the bins (which was counted during the "cleaning" process) with overflow (data items outside the integral curve) is higher than 2% of the whole data-set, we can assume that the other bins are clean. The next section deals with the remaining uncertainty*.

III. Corruption of Existing Data, Single Feature Learning with a Certainty Level

We continue considering the case where part of the data in the feature is corrupted. Our goal in this section is to find the certainty level of every sample in the distribution in the case where the upper bound on a number of corrupted data items is known. This section is actually a continuation of the previous, as both sections deal with a single feature, where the first deals with an attempt to find overflow of samples and the second, cope with unsuccessful such attempts; either due to the fact that the distribution is not known in advance, or that no overflows are found. The histogram of these samples is colored green, where the black vertical line that crosses the histogram separates samples with labels $+1$ and -1. The labels of the Byzantine data have an inverted label with relation to the label of the non-Byzantine data items with the same value. To achieve our goal we describe a general method that bounds the influence of the Byzantine data items.

Method to Bound the Influence of the Byzantine Data Items. The new approach is based on the assumption that an upper ξ on the number of Byzantine data items that may exist in every bin in the distribution is known (e.g., maximum ξ equals 8 items). The certainty level ζ of each bin is calculated by the following equations:

$$\zeta_{-1} = \frac{L_{-1} - \xi}{N} \tag{3}$$

$$\zeta_{+1} = \frac{L_{+1} - \xi}{N} \tag{4}$$

Where L_{-1} is the number of data items that are labeled as -1, L_{+1} is the number of data items that are labeled as $+1$, and N is the number of data items in the bin.

Algorithm 3 *Finding the Certainty Level*

1) Take the original sample of size n from the population of interest (e.g., one feature from the data set),
2) Sort the n data items (samples) according to their value and create their histogram,
3) Count data items at every bin, where the size of bin is the value of natural number in the histogram $\pm$ 0.5 (e.g., for the natural number 73, the bin is between 72.5 to 73.5) and count the number of data items that are labeled as -1 and $+1$.
4) Find the certainty level ζ of each bin according to equations 3 and 4, and the assumption of the size of the maximum ξ.

Algorithm 3: Description of the method for finding the certainty level of every sample for ξ Byzantine data items in every bin in the distribution.

IV. Corruption of Existing Data, Multi-Feature Learning (with a New Decision Trees Algorithm)

Our last contribution deals with the general cases in which corrupted data are part of the data-set and can appear in two modes: (*i*) An entire feature is corrupted (Figure 3), and (*ii*) Part of the features in the data-set is corrupted and the other part is clean. Note that there are several ways to corrupt an entire feature, including: (1) inverting the classification of data items, (2) selection of random data items, and (3) producing classifications inconsistent with the classifications of other non-corrupted features. Our goal, once again, is to identify and to filter data items that are suspected to be corrupted. The first case (*i*) is demonstrated by Figure 3, where the raw data items contain one feature and one vector of labels, where part of the features are totally non-corrupted and part are suspected to be corrupted (for all samples in this column there is a wrong classification).

Method to Bound the Influence of the Corrupted Data Items. Our technique is based on the *Random Forest*; like the *Random Forest* algorithm (Breiman, 1999 [5]) we use decision trees, where each decision tree that is created depends on the value of a random vector that represents a set of random columns chosen from the training data.

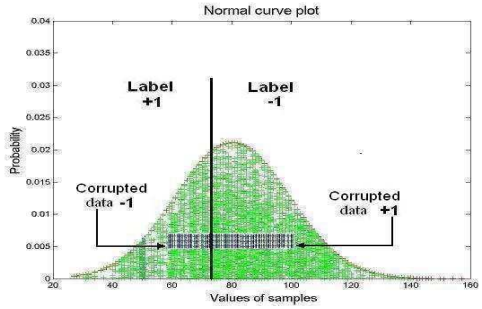

Fig. 3. Histogram of original samples with corrupted data inside the normal curve.

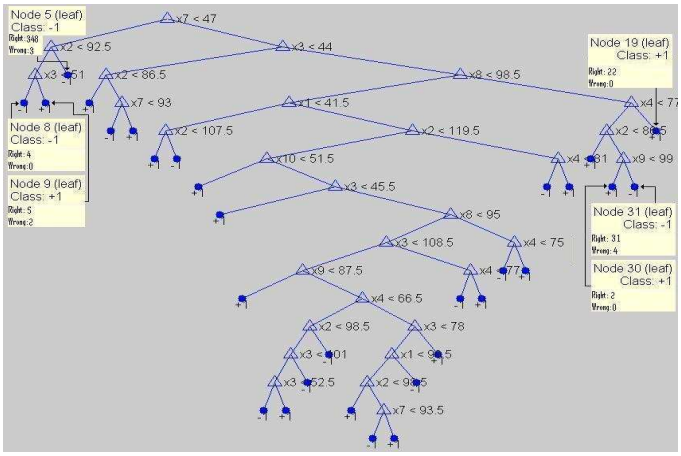

Fig. 4. Example of a decision tree for predicting the response for the instances in every leaf with right or wrong classification.

Large numbers of trees are generated to create a Random Forest. After this forest is created, each instance from the training data set passes through these decision trees. Whenever a data set instances arrives to a tree leaf, its tree classification is compared with its class ($+1$ or -1); when the classification and the class agree the *right* instance of the leaf is incremented; otherwise the value of the *wrong* instance of this leaf is incremented, e.g., 351 instances were classified by Node 5 (leaf): 348 with the right classification and 3 with the wrong classification (Figure 4).

Certainty Adjustment Due to Byzantine Data Bound. The certainty level ζ of each leaf can be calculated based on the assumption that the upper bound on the number of corrupted data items ξ at every leaf in the tree is known. These calculations are arrived at using equations 3 and 4, where, L_{-1} is the number of variables (in the leaf) that are labeled as -1, L_{+1} is the number of instances (in the leaf) that are labeled as $+1$, and *N* is the total number of variables that were classified by the leaf.

In the second step, each instance from the test data set passes through these decision trees to get its classification. Each new tested instance will get a classification result and a confidence level, where the confidence level is in the terms of the (training) right and wrong numbers associated with the leaf in the tree. The final classification is a function of the vector of tuples $\langle classification; right; wrong; \rangle$ with reference to a certainty level rather than a function of the vector of $\langle classification \rangle$ which is used in the original *Random Forest* technique. In this study we show one possibility for using the vector of $\langle classification; right; wrong; \rangle$, though other functions can be used as well to improve the final classification.

Algorithm 4 *Identify and Filter Byzantine Data*

1) First, select the number of trees to be generated, e.g. K,
2) **For** k=1 to K **do**
3) A vector θ_k is generated, where θ_k represents the data samples selected for creating the tree (e.g., random columns chosen from training data sets - these columns are usually selected iteratively from the set of columns, with replacement between iterations),
4) Construct tree $T(\theta_k$,y) by using the decision tree algorithm,
5) **End for**
6) Each instance from the training data passes through these decision trees, and for every leaf the number of instances that are classified correctly (right) and incorrectly (wrong) are counted, then the percentages of right and wrong classifications are calculated,
7) Each instance from the test data set passes through these decision trees and receives a classification,
8) Each new instance will receive a result $\langle classification; right; wrong; \rangle$ from trees in the forest, right and wrong percentages from all the trees are summarized (e.g., sample 10 is classified by Tree No. 1 at Node 5 as $+1$ with 90% (or 0.9) correctness and 10% (or 0.1) incorrectness, by Tree No. 2 at Node 12 as $+1$ with 94% (or 0.94) correctness and 6% (or 0.06) incorrectness, where the total correctness of $+1$ for this sample from both trees is 92% (or 0.92) and 8% (or 0.08) for -1). The final classification for each instance will be determined according to the difference between the total correctness (right classifications) for $+1$ to the total incorrectness (wrong classifications) for $+1$ that are summarized from all trees*.

* This is one option for using the *right* and *wrong* counters to determine the classification.

Algorithm 4: Description of the method for identifying and filtering Byzantine data for multi-feature data-sets.

We tune down the certainty in each leaf using a given bound on the corrupted/Byzantine data items. The contribution of this part includes a conceptual improvement of the well known random forest technique; by re-examining all data

items in the data set. The re-examination counts the number of right and wrong classifications in each leaf of the tree.

V. Conclusion and Future Work

In this work we present the development (the details of the experiment results appear in ([9]) of three methods for dealing with corrupted data in different cases: The first method considers Byzantine data items that were added to a given non-corrupted data set. Batches of uniformly selected data items and Chernoff bound are used to reveal the distribution parameters of the original data set. The adversary, knowing our machine learning procedure, can choose, in the most malicious way on, up to the 2%. malicious data; Note, that there is no requirement for the additional noise to come from distribution different than the data items distribution. We prove that the use of uniformly chosen batches and the use of Chernoff bound reveals the parameters of the non-Byzantine data items. We propose to use certainty level that takes into account the bounded number of Byzantine data items that may influence the classification. The third method is designed for the case of several features, some of which are partly or entirely corrupted. We present an enhanced random forest technique based on certainty level at the leaves. The enhanced random forest copes well with corrupted data. We implemented a system and show that ours performs significantly better than the original random forest both with *and without* corrupted data sets; we are certain that it will be used in practice.

In the scope of distributed systems, such as sensor networks, the methods can withstand malicious data received from a small portion of the sensors, and still achieve meaningful and useful machine learning results.

References

[1] Aslam, J., Decatur, S.: Specification and simulation of statistical query algorithms for efficiency and noise tolerance. *J. Comput. Syst. Sci.* 56, 191–2087 (1998)

[2] Auer, P.: Learning nested differences in the presence of malicious noise. *Theoretical Computer Science* 185(1), 159–175 (1997)

[3] Auer, P., Cesa-Bianchi, N.: On-line learning with malicious noise and the closure algorithm, *Ann. Math. and Artif. Intel.* 23, 83–99 (1998)

[4] Berikov, V., Litvinenko, A.: Methods for statistical data analysis with decision tree, Novosibirsk Sobolev Institute of Mathematics, (2003)

[5] Breiman, L.: Random forests, Statistics department, Technical report, University of California, Berkeley (1999)

[6] Breiman, L., Friedman, J. H., Olshen, R.A., Stone, C.J.: *Classification and Regression Trees*, hapman & Hall, Boca Raton (1993)

[7] Cesa-Bianchi, N., Dichterman, E., Fischer, P., Shamir, E., Simon, U. H.: ample-efficient strategies for learning in the presence of noise, . *ACM* 46(5), 684–719 (1999)

[8] Decatur, S.: Statistical queries and faulty PAC oracles, *Proc. Sixth Work. on Comp. Learning Theory*, 262–268 (1993)

[9] Dolev, S., Leshem, G., Yagel, R.: Purifying Data by Machine Learning with Certainty Levels, *Technical Report August 2009, Dept. of Computer Science, Ben-Gurion University of the Negev* (TR-09-06)

[10] Kearns, M., Li, M.: Learning in the presence of malicious errors,*SIAM J. Comput.* 22(4), 807–837 (1993)

[11] Mansour, Y., Parnas, M.: Learning conjunctions with noise under product distributions, *Inf. Proc. Let.* 68(4), 189-196 (1998)

[12] Mitchell, T. M.: *Machine Learning*, McGraw-Hill (1997)

[13] Quinlan, J. R.: *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers (1993)

[14] Quinlan, J. R.: Induction of Decision Trees, *Machine Learning* (1986)

[15] Servedio, A. R.: Smooth boosting and learning with malicious noise, *Journal of Machine Learning Research* (4), 633–648 (2003)

[16] Valiant, G. L.: A theory of the learnable, *Communications of the ACM* 27(11), 1134–1142 (1984)