Novelty Discovery with Heterogeneous Features
Cross-Feature Analysis for Database Intrusion Detection Systems

Ken Samuel, Peter Mork, Adriane Chapman, David Moore, Irina Vayndiner, and Erik Sax

Abstract

This paper presents experiments with a unique machine learning method called Cross-Feature Analysis, which is a novelty discovery method that can easily accommodate heterogeneous features. The domain of our work is database security, with the goal of detecting attacks that are similar to those seen in the past as well as completely novel attacks that have not yet been seen. The training data consists of database logs that have no attacks, so supervised machine learning methods cannot apply, and unsupervised machine learning methods are unsatisfactory, because we have a variety of feature types, including numerical features, categorical features, and set-valued features. However, Cross-Feature Analysis transforms our novelty discovery problem into multiple supervised machine learning problems, building one submodel for each feature by treating that feature as the *class*, Then new instances are analyzed by the submodels to determine whether they are consistent (legitimate) or anomalous (suspicious). In our experiments we discovered that, by setting a limit on the number of submodels that reject an instance, our system can distinguish legitimate instances from attacks with perfect (100%) recall of real attacks and a specificity of 99.9% on legitimate instances for one data set, and on another data set, recall = 97.2% and specificity = 99.9%.

1 Introduction

This paper presents experiments with a unique machine learning method called Cross-Feature Analysis (CFA). CFA is a novelty discovery[1] approach, which means that it learns from training data in which all instances have the same class. In addition, unlike unsupervised learning approaches, CFA can easily accommodate a variety of feature types, such as numerical features, categorical features, and set-valued features. The Query Anomaly Intrusion Detection System (QAIDS) applies CFA to database security, which is an important intrusion detection problem because mission-critical data is often stored in databases. Many people attempt to gain illegitimate access to data; USA Today reports that 450,000 SQL injection attacks[2] are launched every day. Once an attack has been detected, an algorithm can be designed to watch for similar attacks. However, adversaries are always inventing new approaches for evading these security systems. We want security mechanisms that are robust against unforeseen attacks from outsiders, as well as insiders.

[1] Novelty discovery is also known as one-class classification, outlier detection, anomaly detection, concept learning, single-class learning, single-class classification, and partially supervised classification.
[2] SQL injection attacks are the most common of several types of database attacks.

Like people who are trained to discern counterfeit currency by learning characteristics of real money, CFA develops a profile of *legitimate* database activity, anticipating that most attacks will not fit the profile well. Our training data lacks positive[3] instances, so we use CFA to transform our novelty discovery problem into multiple supervised machine learning problems, building one submodel for each feature by treating that feature as the *class*. Then, in the application phase, a new instance is run through each of the submodels, and if many of the submodels' predictions agree with the instance's feature values, the instance is labeled as negative. However, if the instance does not fit the profile defined by the submodels, that suggests it is different from the training data of negative instances, so it is labeled positive.

The remainder of the paper is organized as follows. After discussing some related work in Section 2, including CFA, we focus on our design decisions in implementing the QAIDS system in Section 3. Then Section 4 offers a description of two data sets that we used, before Section 5 presents the outstanding results of our experiments, including perfect (100%) recall on real attacks and nearly perfect specificity (99.9%) on legitimate instances.

2 Related Work

This section begins by summarizing past work on the database security task. Then it provides a very brief survey of novelty detection algorithms followed by a detailed description of CFA.

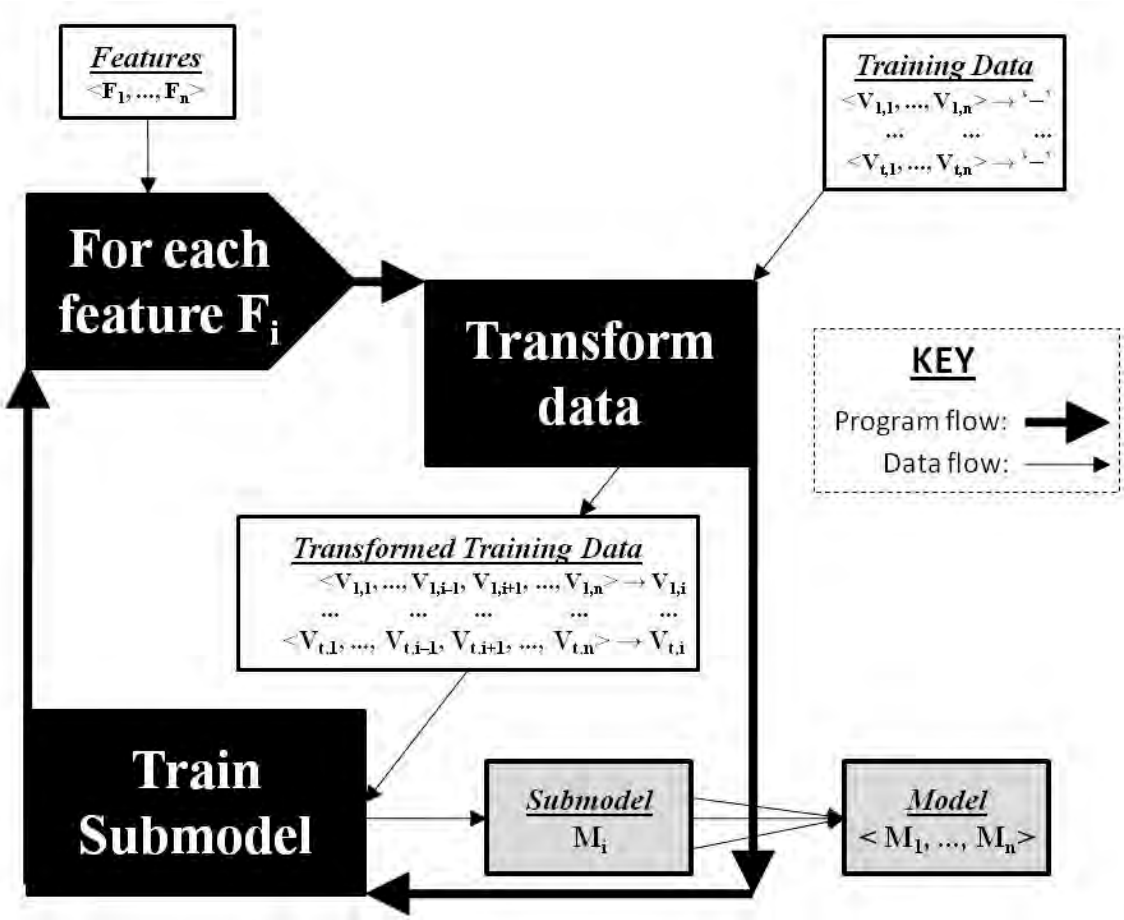
2.1 Database Security

Database security generally involves checking for attacks that are similar to those seen in the past. However, "new vulnerabilities will continue to be discovered and... our adversaries will continue to invent new attack methods" [Huang and Lee, 2003]. While we certainly want to block the known attack strategies, we are also focused on catching the attacks that are unlike any that have been seen before, which is far more difficult.

Research on anomaly detection in database systems can be loosely organized into two categories: attempts to prevent malicious modifications to the contents of the database and attempts to prevent illegitimate data exfiltration (extraction of information from a database). Hu and Panda [2003] describe an approach based on analyzing the interactions among information that is read from and written to a database during a transaction. But unlike us, they apply their application-phase analysis to transaction logs *after* the attacks have already succeeded. Fonseca and Vieira [2008] describe a similar approach, but the transactions are monitored in real time and can therefore be rolled back as soon as an anomaly is detected. However, unlike our system, neither of these approaches can detect exfiltration attacks that do not attempt to modify the database.

Kamra et al. [2008] tried using Naïve Bayes, k-Means Clustering, k-Centers Clustering, and Median of Absolute Deviations to build models of SQL queries by generating a cluster of feature vectors for the SQL queries submitted by users who have the same access privileges. Then, in the application phase, a user's SQL queries are tested to verify that they match the pattern of users with the same access privileges. If they seem more similar to SQL queries from users with higher

[3] In this paper, we refer to legitimate SQL statements as negative instances, and attacks as positive instances.



access privileges, that indicates an attack. Lee and Low [2002] focused on exfiltration, automatically designing regular expressions that represent legitimate SQL queries. Both of these papers present work that is limited to syntactic features of SQL queries, however attacks instigated by insiders can look similar to legitimate queries at a syntactic level. Section 3.2 discusses the features we used, which can capture more than just the syntactic information in an SQL query.

2.2 Novelty detection

We trained QAIDS on the extensive logs of daily operations that a database retains, which consist of thousands of user-submitted SQL queries. Since there are probably very few, if any, attacks in our training data, we made the assumption that all of the training instances are negative. Because of this, we could not use any supervised learning methods, because they require that all classes be represented in the training data. Some supervised machine learning methods have been modified to deal with this novelty detection problem, including Support Vector Machines [Schölkopf et al., 1999] and Naïve Bayes [Kamra et al., 2008]. Other novelty detection approaches include Minimax Probability Machines [Lanckriet et al., 2002], Median of Absolute Derivations [Kamra et al., 2008], k-Means Clustering, k-Centers Clustering (also known as k-Medoids Clustering), k-Nearest Neighbor, Neural Networks, Self-Organizing Maps, Support Vector Data Description, Principal Component Analysis (PCA) and mixtures of PCAs,

Learning Vector Quantization, the Gaussian model and Mixture of Gaussians, the Parzen Density Estimators, Auto-Encoders, and Diabolo Networks [Tax, 2001]. But unfortunately, all of these approaches require the developer to define "a measure for the distance … [from] an object to the target class (represented by a training set…)" [Tax, 2001]. In other words, these novelty detection methods require the researcher to come up with a formula that specifies the distance between a new instance and the model that represents the training data. However, our feature set has a variety of feature types, including numerical features, categorical features, and set-valued features. Developing distance metrics for heterogeneous features is very difficult, because the features must be weighted so that they are somehow consistent. For example, imagine trying to decide which of the following distances is the largest: a) the distance between 1 and 2 pounds, b) the distance between $10 and $20, c) the distance between a rose and a carnation, or d) the distance between the poker hands {ace of hearts, king of hearts} and {2 of spades, 2 of clubs}. Because this is so challenging, we decided not to use any of the distance-based methods listed above.

Table 1: Example Training Data

Ingredients	Calories	Rating
{water}	0	good
{water, sugar}	75	bad
{water, sugar}	75	good
{water, sugar, bread}	100	bad
{water, bread}	31	bad
{sugar}	75	bad
{sugar}	75	good
{sugar, bread}	96	great
{sugar, bread}	98	great
{bread}	29	good

2.3 Cross-Feature Analysis

Cross-Feature Analysis is a novelty detection technique that can learn from data with heterogeneous features but does not require that a distance metric be defined [Huang, 2006; Huang et al., 2003]. CFA transforms a novelty detection problem into a supervised machine learning problem so that Decision Trees, Support Vector Machines, Neural Networks, Transformation-Based Learning or any other supervised machine learning algorithm may be used.

2.3.1 The training phase

The CFA training algorithm, presented in Figure 1, is a loop that runs through all of the

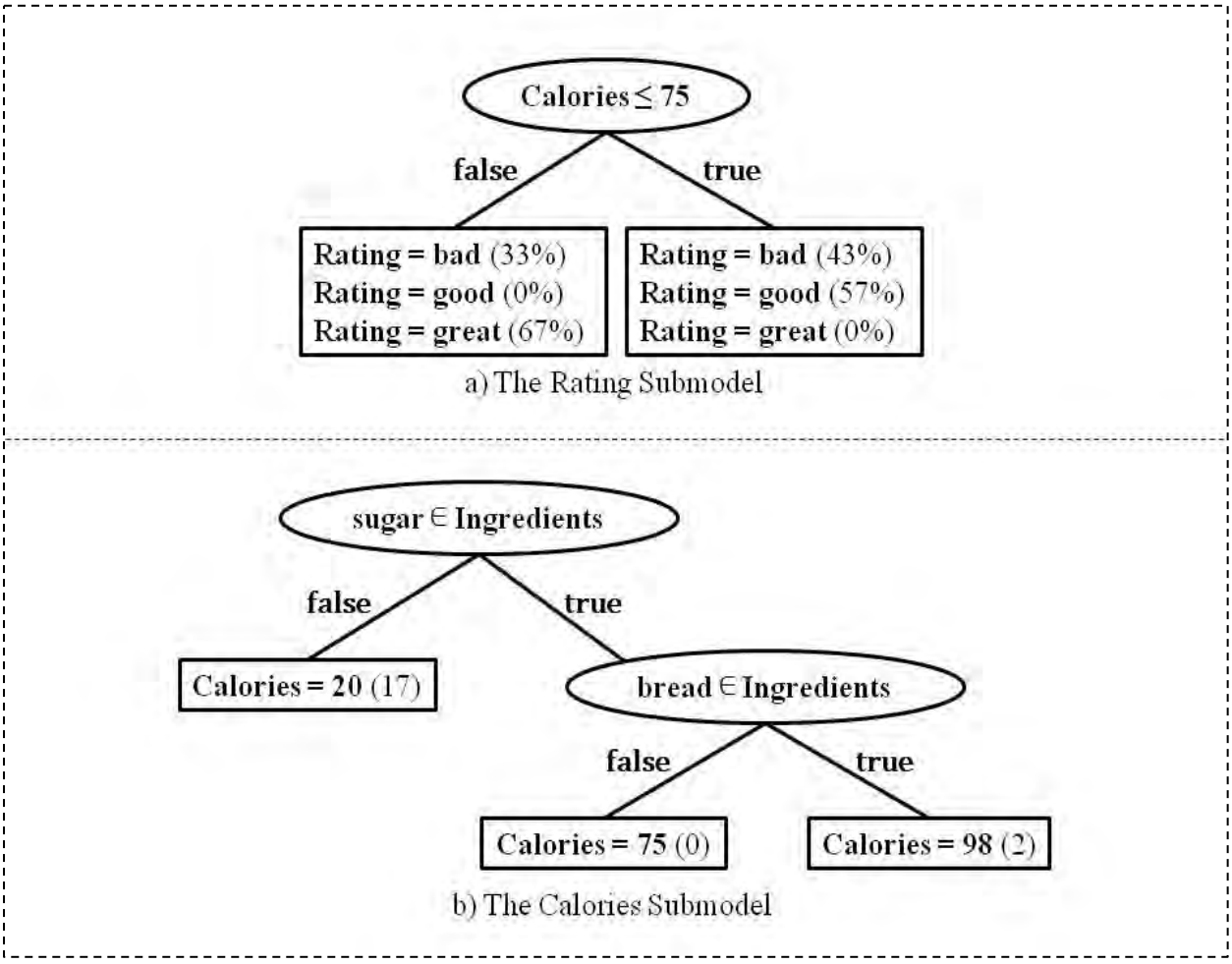


Figure 2: Examples of Submodels

features, $<F_1, ..., F_n>$. For each feature F_i, the training data is transformed so that F_i becomes the class. In other words, for each training example, $<V_1, ..., V_n> \rightarrow$ '–', the value of F_i, V_i, is removed from the feature vector to replace the class '–', effectively transforming the training instance into $<V_1, ..., V_{i-1}, V_{i+1}, ..., V_n> \rightarrow V_i$. As a result, a supervised machine learning algorithm can be trained on the transformed data to produce a submodel, M_i. The final learned model is a list of all of the submodels, $<M_1, ..., M_n>$.

As an example, consider the training data in Table 1. There are three features: 1) a categorical feature called Rating with three possible values, bad, good, and great, 2) a numerical feature called Calories, and 3) a set-valued feature called Ingredients with values that are subsets of {water, sugar, bread}. For this training data the learned model might be that depicted in Figures 2 and 3. The submodel in Figure 2a is a decision tree that predicts the categorical feature, Rating, suggesting that it is related to Calories. Apparently high-calorie food tends to be great, while low-calorie food is only good. The numbers in parentheses give the percentage of the training data that reached each leaf with each value of Rating, showing that the error in this decision tree is pretty high. The submodel in Figure 2b is a regression tree where each leaf specifies a value for the Calories feature with its standard deviation given in parentheses. And, like Cohen [1996], we split a set-valued feature into multiple boolean features, one for each possible member in the feature's values. Each new feature is true if and only if the corresponding member is in the original feature's value. In the example, three submodels are learned for the Ingredients feature: one for water (Figure 3a), one for sugar (Figure 3b), and one for bread (Figure 3c).

Table 2: Experimental Results [Huang and Lee, 2003]

Attack Type	Recall	Specificity
False source route, maximum sequence, and rushing	85%	99.03%
Packet dropping	98%	99.11%
Malicious flooding	99%	99.05%
Spoofing	87%	99.02%

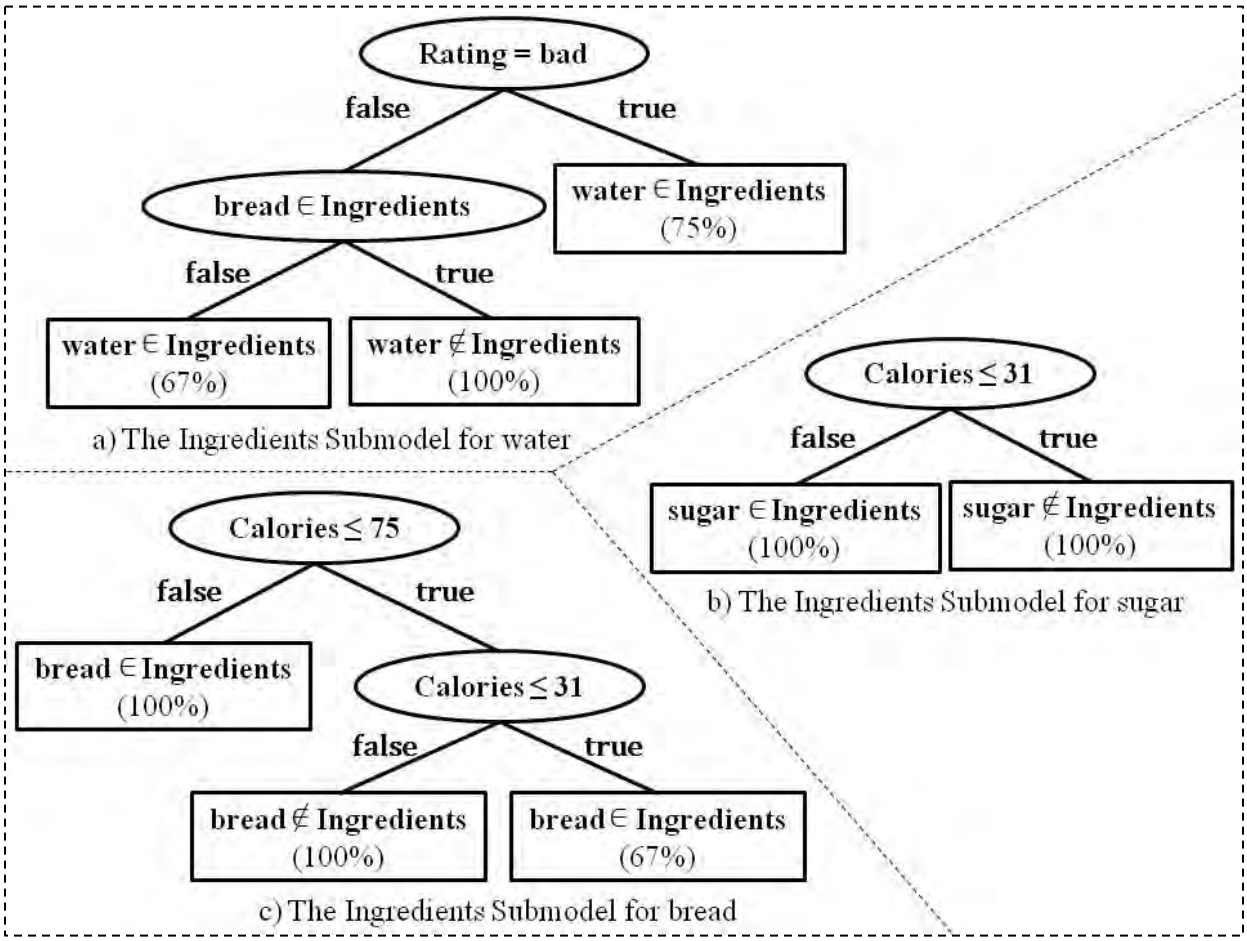


Figure 3: Examples of a Set-Valued Feature's Submodels

2.3.2 The application phase

There are two different algorithms that may be used in the application phase, Average Match Count and Average Probability [Huang et al., 2003; Huang, 2006]. This section explains how these algorithms work for categorical features, and the procedure for numerical features is presented in Section 3.3.

Average Match Count

Figure 4 shows the Average Match Count algorithm. The percentage of the submodels that accurately predict a given unlabeled instance's feature values is compared with a predetermined threshold[4] to choose the label for the instance. In other words, for each feature F_i, a given unlabeled instance, $<V_1, ..., V_n> \rightarrow ?$, is transformed by the same process as that in the training phase so that it can be processed by submodel M_i. The instance's legitimacy score L is increased by 1 point if and only if M_i correctly predicts the feature's value that the instance actually has, V_i. After doing this for all of the features, L is divided by the number of submodels, n, and that average legitimacy score is compared with a threshold θ to determine the label for the instance. An instance that receives a high average legitimacy score ($- > \theta$) is labeled '–' (the SQL query is

[4] Section 3.4 discusses how the threshold may be selected.

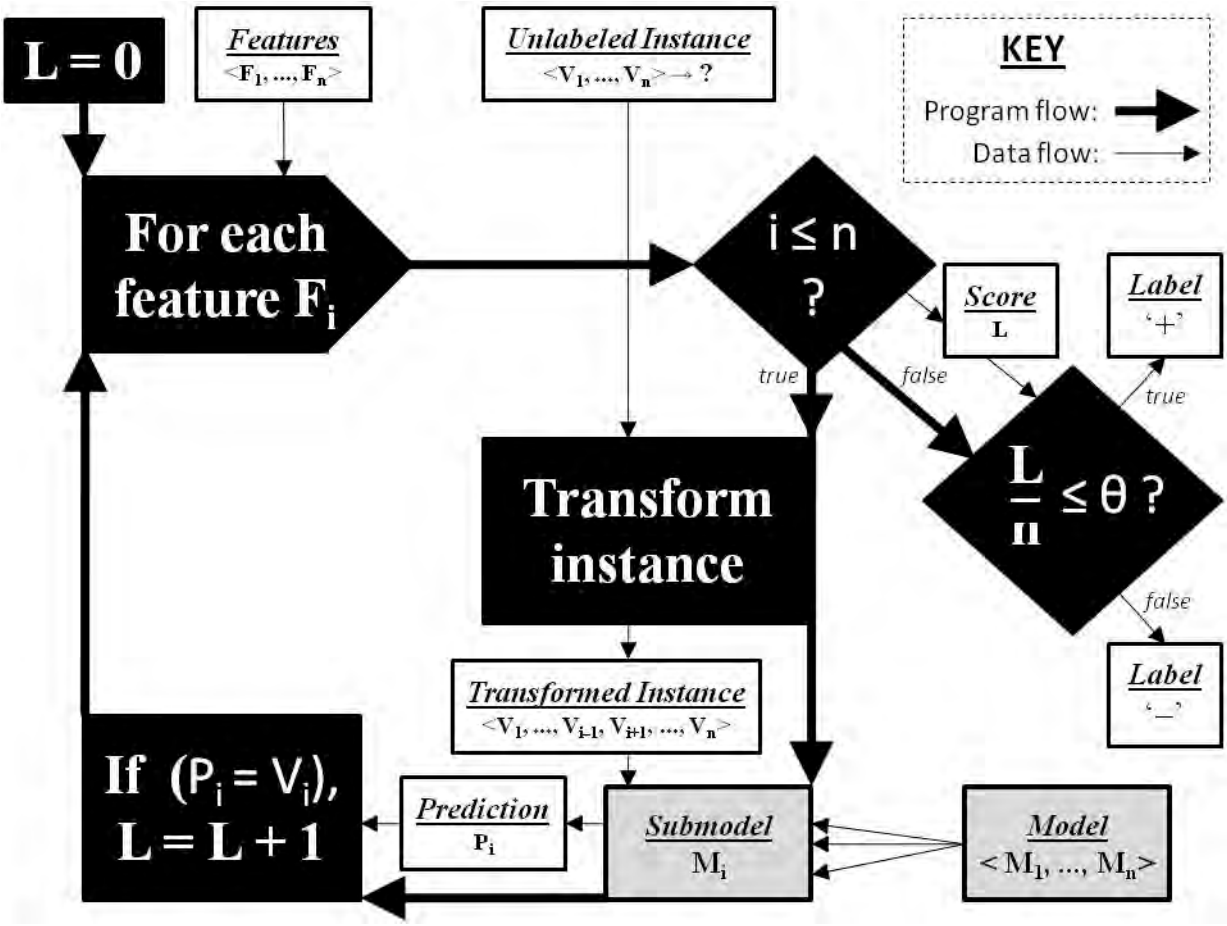


Figure 4: Average Match Count

legitimate), and an instance with a low average legitimacy score ($\frac{L}{n} \leq \theta$) is labeled '+' (the SQL query is suspicious).

For example, suppose the unlabeled instance, <good, 42, {water, sugar }>, is presented to the model in Figures 2 and 3. The Rating submodel (Figure 2a) predicts that the Rating is good, and since that is true for the unlabeled instance, a point is added to L. The Calories submodel predicts that the instance should have Calories = 75, but it actually has Calories = 42, so L is unchanged by that submodel[5]. Continuing, the three decision trees in Figure 3 predict that Ingredients has water and sugar but not bread, all of which are true. So the average legitimacy score for this instance is $\frac{L}{n} = \frac{3}{4} = 80\%$. If $\theta = 50\%$, then this instance is labeled '–'.

As another example, the instance <bad, 100, {bread}> is consistent with only one of the submodels, the Ingredients submodel for bread. So $\frac{L}{n} = \frac{1}{5} = 20\%$, and with a threshold of $\theta = 50\%$, the instance is labeled '+'.

Average Probability

If the supervised machine learning method can return distributions of class values, there is another option for the application phase procedure. The Average Probability algorithm is shown

[5] Section 3.3 explains how close the values must be.

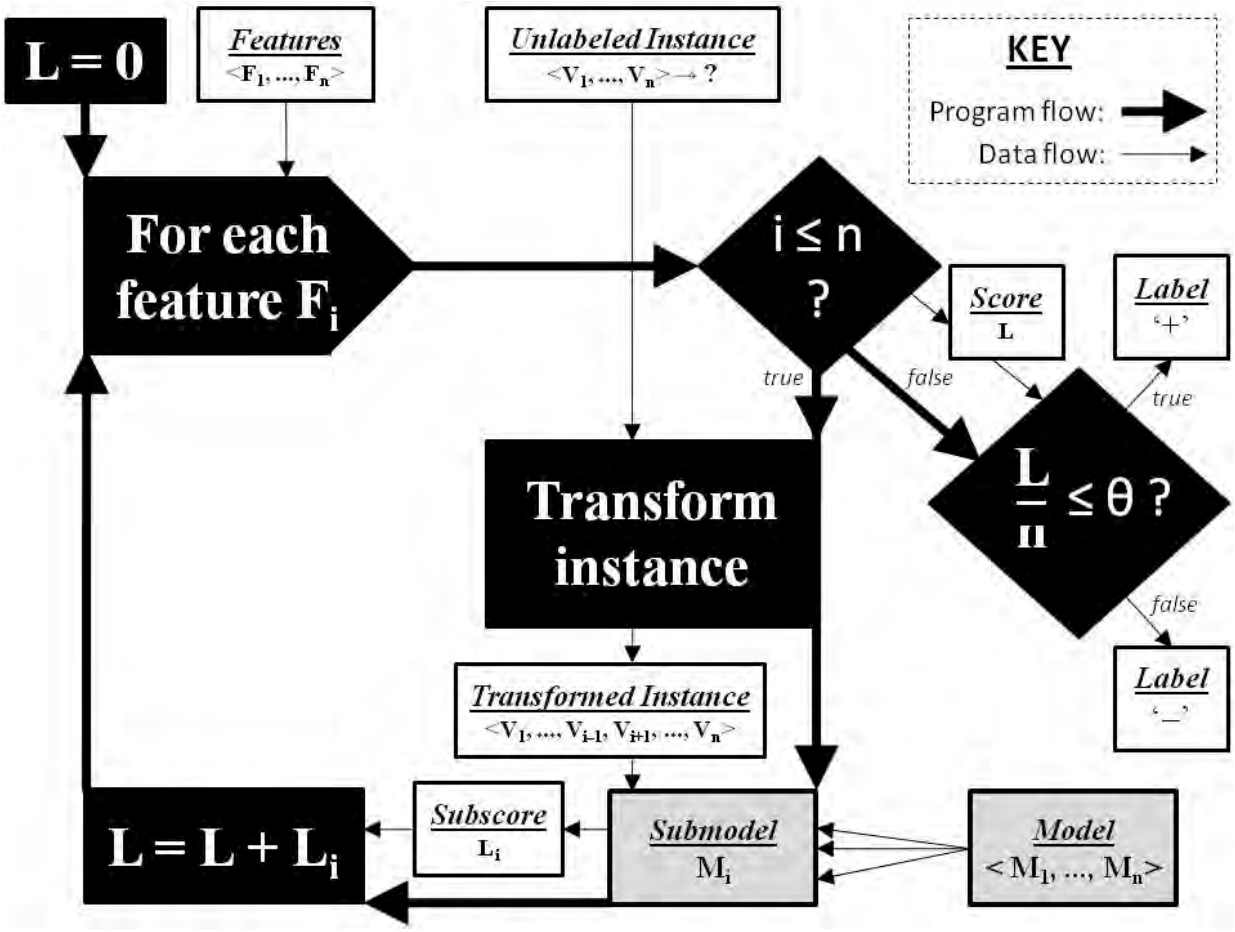


Figure 5: Average Probability

in Figure 5. It is the same as the Average Match Count algorithm (Figure 4), except that instead of awarding 1 point for each submodel that agrees with the given instance and 0 points otherwise, the legitimacy score is increased by the probability that each submodel assigns to the value that the instance has for the corresponding feature. In other words, given an unlabeled instance, $\langle V_1, ..., V_n \rangle \rightarrow ?$, for each feature F_i, submodel M_i returns a probability for each possible value of F_i, and the probability L_i that it returns for the class value V_i is added to the legitimacy score L.

Returning to the example, if the Average Probability algorithm is run on the unlabeled instance, <good, 42, {water, sugar }>, then the Rating submodel (Figure 2a) credits the instance with $L_1 = 57\%$ for its Rating value, good. The three Ingredients submodels (Figure 3) give it legitimacy subscores of $L_3 = 67\%$, $L_4 = 100\%$, and $L_5 = 100\%$ for water, sugar, and bread, respectively. And the regression submodel, Calories, (Figure 2b) returns a legitimacy subscore of $L_2 = 0\%$ using a procedure that will be presented in Section 3.3. Thus, the average legitimacy score assigned to the instance is $\frac{57\% + 0\% + 67\% + 100\% + 100\%}{5} = 65\%$. So, with a threshold of $\theta = 50\%$, this instance is assigned the label '–'. And for the other example instance, <bad, 100, {bread}>, $\frac{0\% + 0\% + 67\% + 0\% + 100\%}{5} = 32\%$, so if $\theta = 50\%$, this instance is assigned the label '+'.

Huang claimed that, in comparison to Average Match Count, Average Probability "can improve [recall] since a sub-model should be preferred where the labeled feature has stronger

Table 3: Experimental Results [Cabrera et al., 2008]

Attack Type	AUC
AODV denial of service	0.4263
AODV black hole	0.4280
OLSR denial of service	0.3772
OLSR black hole	0.3572

Table 4: Experimental Results [Liu et al., 2007]

Attack Type	Recall	Specificity
Flooding	100%	99.71%
Black hole	83.33%	99.71%
Sleep deprivation	85.33%	99.71%
Packet dropping	72.00%	99.71%

confidence to appear in normal data." [Huang, 2006] In other words, the Average Probability algorithm effectively gives greater weight to those submodels that report the highest probabilities that an instance is negative. And experimental results have shown that Average Probability is superior to Average Match Count [Huang et al., 2003]. However, in Section 5.2, we will show that we observed the opposite result in our experiments.

2.3.3 Experimental results from prior work

Throughout this paper, we report experimental results with the evaluation metrics[6] recall (R) and specificity (S). The recall metric is solely based on the system's performance on positive instances, while specificity only takes the negative instances into account. Recall is defined to be —, where TP is the number of true positives (positive instances that the system labeled correctly), and P is the total number of positive instances. And specificity is defined to be — , where FP is the number of false positives (negative instances that the system incorrectly labeled as positive), and N is the total number of negative instances[7].

In our survey of the literature, we found only three research groups that have worked with CFA, all of them working in the field of mobile ad hoc networks (MANET) security. Some of their experimental results are given below.

[Huang, 2006; Huang et al., 2003; Huang and Lee, 2003]

Huang et al. experimentally compared three different supervised machine learning algorithms within a CFA system, finding that C4.5 [Quinlan, 1993] was best, followed by RIPPER [Cohen, 1995] and then Naïve Bayes [Mitchell, 1997]. With C4.5, using the Average Probability method, they obtained the extremely good results shown in Table 2, with recall and specificity as high as R = 99% and S = 99%. They "believed that strong feature correlation exists in normal behavior, and that such correlation can be used to detect deviations caused by abnormal (or intrusive) activities." We believe that this is also true for our database security task.

[Cabrera et al., 2008]

Cabrera et al. used C4.5 as the supervised machine learning algorithm because of the great experimental results reported by Huang et al. (and we chose C4.5 for the same reason). Using the Average Probability method, they generated ROC (Receiver Operating Characteristic) curves by

[6] Although precision is a standard evaluation metric, it is highly dependent on the ratio of negative instances to positive instances. So, since we have far more negative instances than positive instances, the precision scores are misleading.

[7] The false alarm rate is (1 – S).

varying the threshold. Their experimental results on different attack types[8], some of which are shown in Table 3, were very good, with the area under the ROC curve (AUC) higher than 0.42 in some cases. (The optimal value is 0.45.)

Table 5: Some of the Features

Name	Type
ResultSize	numerical
CostOfExplainPlan	numerical
TouchSystemCatalog	boolean (categorical)
SelectionOperator	set-valued
AccessMethodAppliedFor<tableName>	categorical
Timestamp	date (numerical)
NumberOfConjunctions	numerical

[Liu et al., 2007; Liu et al., 2005]

Liu at al. also used C4.5. With the Average Probability method, they obtained the results shown in Table 4. Also, they claimed that CFA "... is suitable for long-term always-on profiling and can preserve the precious node energy."

3 The Query-Anomaly Intrusion Detection System

This section describes the design decisions we made when implementing QAIDS.

3.1 Supervised Machine Learning Algorithms

We used two different supervised machine learning methods, depending on the type of the feature that is treated as the class. Prior work showed that CFA performed better with Decision Trees in comparison with other supervised machine learning algorithms [Huang et al., 2003], so, for non-numerical (categorical, boolean, and set-valued) classes, we chose Decision Trees. For numerical classes (including dates and times), we chose Regression Trees. Specifically, QAIDS uses the Weka [Hall et al., 2009] implementations of the J48 Decision Trees algorithm [Frank, 1999a] (which is based on C4.5 [Quinlan, 1993]) and the REPTree Regression Trees algorithm [Frank, 1999b]. Both of these supervised machine learning algorithms can easily handle all of our various types of features, unlike the novelty detection methods listed in Section 2.2. The parameter settings we chose included requiring at least 5 instances at each leaf (when possible), applying reduced-error pruning with 10 folds, and permitting the decision trees to have binary splits.

3.2 Features

We selected our features through an analysis of known attacks and discussions with database security experts. Table 5 lists some of the features that we used for our TRS data set (which is described in Section 4). We included a variety of different types of features: numerical features

[8] AODV (ad-hoc on demand distance vector routing) and OLSR (optimized link-state routing) are two routing protocols.

(including a date-valued feature), categorical features (including boolean features), and set-valued features. ResultSize is an integer that estimates the quantity of data that should satisfy the SQL query's requirements, and CostOfExplainPlan is an integer that estimates the relative amount of time required to respond to the SQL query. TouchSystemCatalog is a categorical (more specifically, boolean) feature that specifies whether the database system catalog is being accessed (which should never happen). And the value of SelectionOperator is a set containing the names of the columns that the SQL query is requesting. (The AccessMethodAppliedFor<tableName> feature is actually a template that expands into multiple categorical features, one for each table in the database.)

3.3 Numerical Features

The two application-phase algorithms (Section 2.3.2) were designed for categorical features. Huang [2006] proposed two ways that numerical features might be handled by CFA:

1. Numerical features can be transformed into categorical features by creating a finite number of buckets, each of which represents a different range of values, which is what Cabrera et al. [2008] and Liu et al. [2007] did. We found this approach undesirable, because bucketing loses information about the ordering of values, both between and within buckets. Also, it is not clear how to determine the number of buckets and their ranges.
2. Multiple linear regression can handle numeric features by defining the difference between the true value, V, and the predicted value, P, to be $-$. However, we wanted a function with a range of [0,1] rather than [0,).

We took a different approach. Since the REPTree algorithm maintains a record of the variance of the training data at each leaf, we decided to make use of this information. For the Average Match Count method, QAIDS increases an instance's legitimacy score L by 1 point if it is within 2 standard deviations[9] of the mean. And for Average Probability, we used a modified[10] Chebyshev formula [Knuth, 1997], ——, where σ is the standard deviation at the leaf that was reached when traversing the regression tree.

3.4 Threshold Selection

The application phase algorithms in Section 2.3.2 require a threshold that can be used to separate positive instances from negative instances. Determining the value of the threshold manually is difficult, so we wanted to do it automatically. In prior work, the threshold was automatically set to the highest value for which the specificity on a held-out tuning set was at least 99% [Huang and Lee, 2003].

3.5 Unseen Values

In the application phase, it is possible to encounter an instance with an unseen value, which is a value of a categorical feature or a member of a set-valued feature's value that was not found in

[9] The standard deviation is the square root of the variance.

[10] Chebyshev's original formula is ——. Our modification insures that its value is between 0 and 1.

the training data. Certainly unseen values are suspicious, but with a limited quantity of training data, it is possible for legitimate instances to have unseen values. So, instead of automatically labeling an instance with any unseen values as positive, we took a more conservative approach, assigning each instance a legitimacy subscore of 0 for each submodel in which the class has an unseen value. In the case of an unseen value in a set-valued feature's value, an extra 0 is averaged into the final legitimacy score, as if there was another submodel to predict the unseen value. Also, any submodel whose decision tree cannot be traversed to a leaf, because it reaches a branching node that has no option for the unseen value, returns a legitimacy subscore of 0.

For example, using the Average Match Count algorithm (Section 2.3.2) with the model in Figures 2 and 3, the instance, <excellent, 101, {sugar, bread}> , in which the Rating, excellent, is an unseen value, receives the average legitimacy score $\frac{\quad}{\quad} = 80\%$, and so, if the threshold $\theta = 50\%$, then this instance is labeled '–', despite its unseen value. (Note that a binary branch like the one at the root of Figure 3a can still be traversed, even though the feature it is testing, Rating, has an unseen value. This is because its question, "Rating = bad?" can still be answered false, despite the unseen value.)

As another example, for the instance <bad, 54, {water, eggplant}>, an imaginary submodel for the unseen member of Ingredients, eggplant, produces a legitimacy score of 0, so the average legitimacy score is $\frac{\quad}{\quad} = 50\%$.

4 Description of the Data

We experimentally analyzed QAIDS on data generated by two different web services: the Medical Discharge Data Access System (MDDAS) and our company's Time Reporting System (TRS). Information about these data sets is provided in Sections 4.1 and 4.2.

We divided each data set into ten random subsets of equal size. Then, for each of the experiments presented in Section 5, we averaged the results from ten runs, each of which used one of the subsets for training, a different subset for tuning, and a third subset for testing. Each subset was used exactly once in each of the three roles. This procedure is similar to 10-fold cross validation.[11]

4.1 The MDDAS Data Set

MDDAS is a web service that provides access to a database of several million anonymized patient discharge records, which contain each patient's discharge status, demographic information, and diagnosis and procedure codes. We submitted requests through a web service interface to generate 202,300 SQL queries for this data.

MDDAS has 135 categorical features, 77 numerical features, and 1 set-valued feature with 154 different members in its values, for a grand total of 135 + 77 + 154 = 366 submodels, of which 289 were generated by J48 and 77 of which were generated by REPTree.

[11] Normally, in 10-fold cross validation, 8 of the sets are used for training in each iteration. However, due to memory limitations of our computer, we were unable to run the training algorithm with that much data, so we used only one set for training.

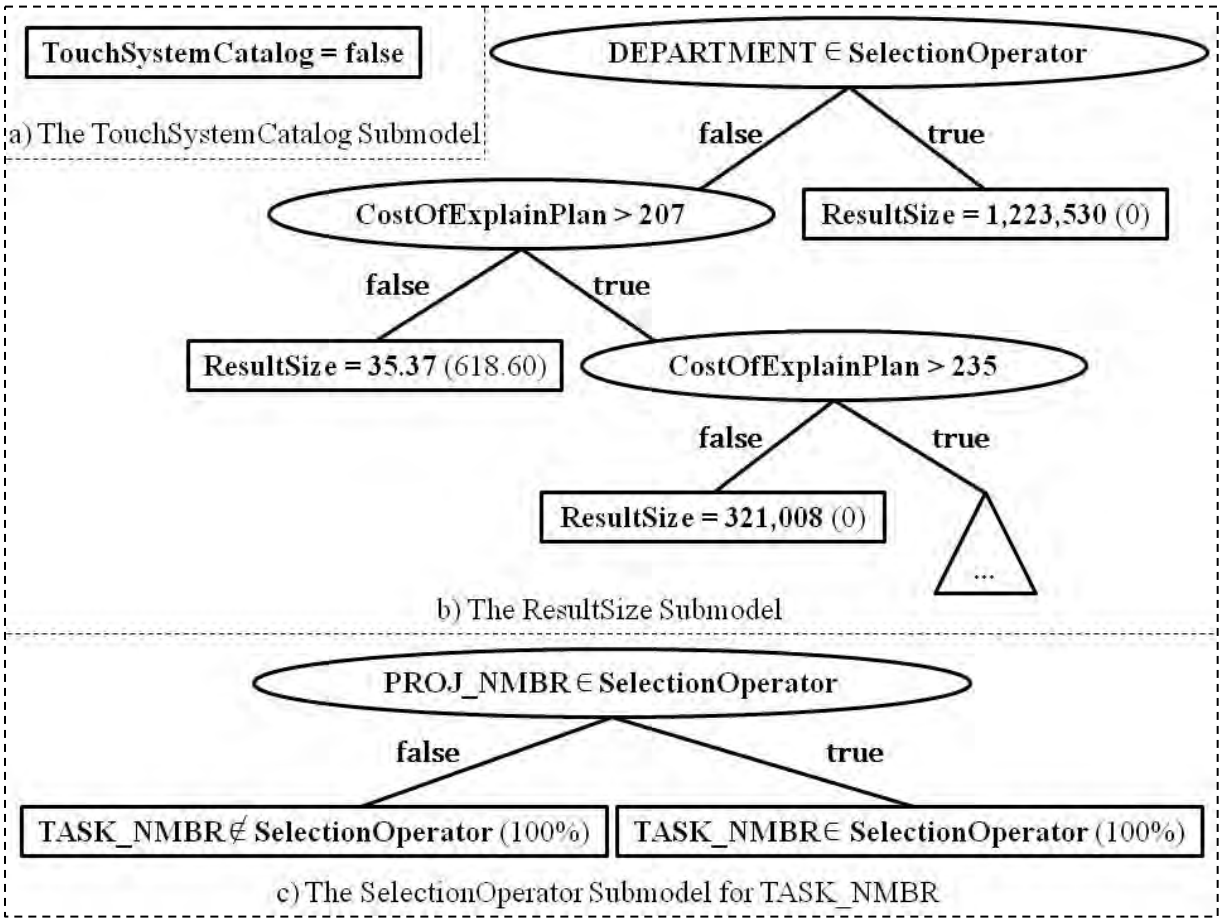


Figure 6: Some of the Submodels

4.2 The TRS Data Set

TRS is used by employees of our company to report how much time they spend working on different projects. The TRS database processes more than 100,000 SQL queries each day. We used procedures stored in the database along with the SQL queries built into the web interface to create 122,450 negative instances.

The TRS database has 104 tables in the database, so we used 117 features. (See Section 3.2.) Three of the features are set-valued, with 37, 191, and 69 unique members in their training data values, so they were expanded into 37 + 191 + 69 = 297 binary features, resulting in a grand total of 117 + (297 – 3) = 411 features. Thus, the learned model had a grand total of 411 submodels[12], 403 of which were generated by J48 and 8 of which were generated by REPTree.

4.3 Positive Instances

[12] Actually, 2 of the 10 models the system built had fewer submodels, because some members of the set-valued features' values were not found in their training data.

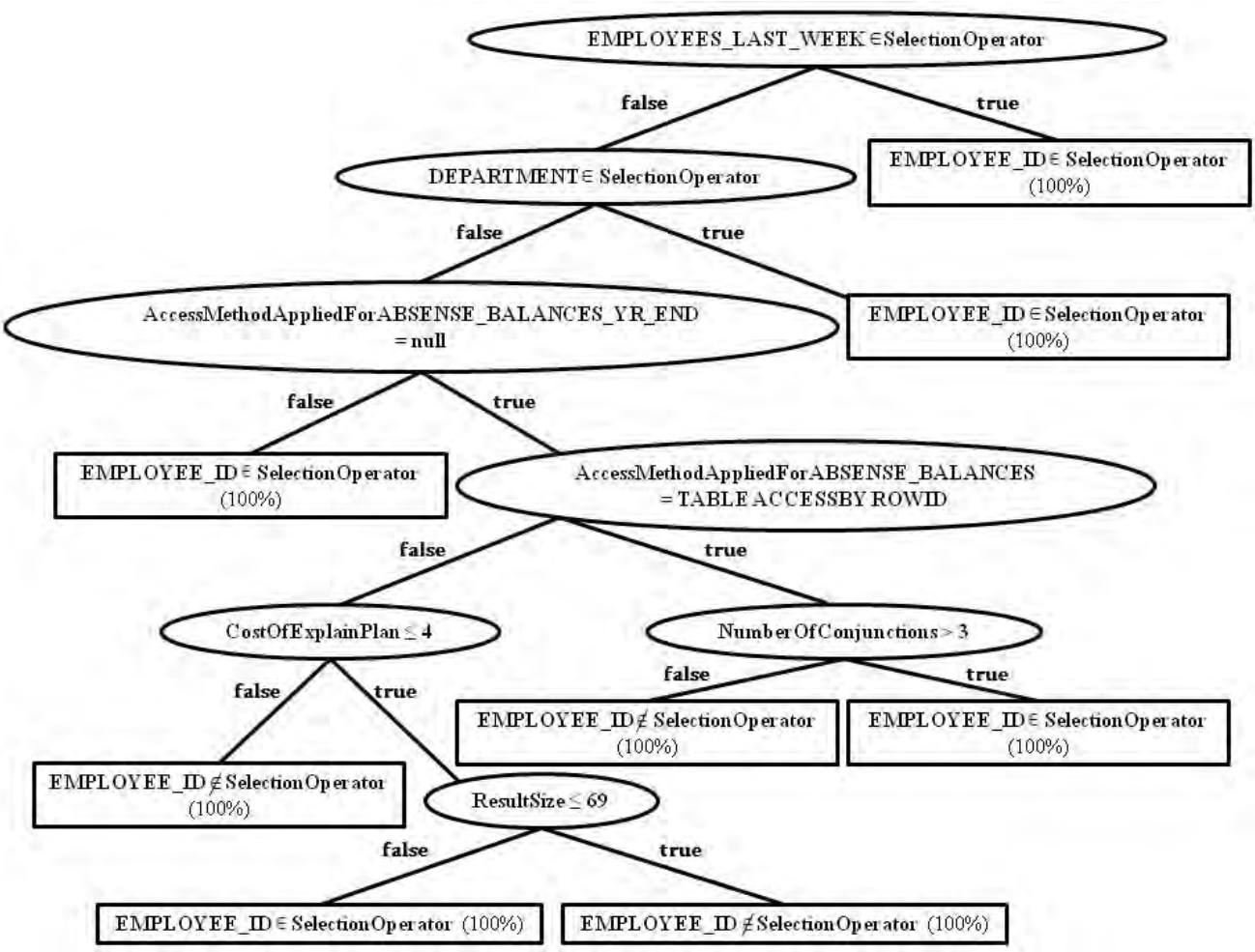


Figure 7: The SelectionOperator Submodel for EMPLOYEE_ID

While the database logs provided us with plenty of negative instances, it is much harder to find positive instances; our set of positive instances consists of only 36 attacks for MDDAS and 33 attacks for TRS[13].

5 Experimental Results

Figures 6 and 7 show some of the submodels that QAIDS learned from the TRS data. The submodel in Figure 6a is the decision tree that predicts the value of the boolean feature, TouchSystemCatalog, a one-node tree that correctly predicts that the system catalog is never accessed. A regression tree that predicts ResultSize, a numerical feature, is shown in Figure 6b[14]. (As in Figure 2b, the numbers in parentheses give the standard deviation at each leaf.) And the

[13] People familiar with database security created these attacks in two different ways. Some of them were generated by standard attack methods, like hunting for passwords, and others were created by submitting SQL queries to find prohibited business information.

[14] For brevity, part of this regression tree is not shown, being represented by a triangle instead.

Table 6: Application Phase Algorithms

	MDDAS Data		TRS Data	
	Recall	**Specificity**	**Recall**	**Specificity**
Average Match Count	**97.222%**	**99.853%**	100.000%	99.335%
Average Probability	95.000%	99.774%	100.000%	99.521%

SelectionOperator feature, being a set-valued feature, is split into one tree for each of its values, as explained in Section 2.3.1. Figures 6c and 7 show two of these submodels, which predict whether the SQL query asks for the values of TASK_NMBR and EMPLOYEE_ID, respectively.

The following sections present some of our experiments. To determine whether the differences between different recall or specificity values are statistically significant, we used the two-tail t-test assuming equal variances. When we report that a difference is significant, we mean that $p < 0.05$ (i.e. there is less than a 5% chance that the difference can be completely attributed to random variance. In Tables 6, 7, 8, and 9, values that are significantly different are separated by horizontal lines, with the highest value(s) printed in **bold**.

In the experiments that we report in this section, we used the Average Match Count application-phase algorithm (Section 2.3.2) and the approaches described in Sections 3.1, 3.3, 3.4, and 3.5, except where specified otherwise.

5.1 Application Phase Algorithms

In Section 2.3.2, we discussed two different ways to compute a legitimacy score for an unlabeled instance, Average Match Count and Average Probability. The performance of the two different application-phase algorithms is given in Table 6. Although no conclusions could be drawn from the results on the TRS data, the MDDAS data revealed that Average Match Count is significantly better than Average Probability for both recall and specificity. This is the opposite conclusion than that previously reported [Huang et al., 2003], suggesting that it may be somehow dependent on the features or the task. In any event, both algorithms work very well.

5.2 The Threshold

The two application phase algorithms (Section 2.3.2) both require a threshold, θ, to separate positive and negative instances. In Section 3.4 we noted that Huang and Lee [2003] set the threshold to the highest value for which $S \geq 99\%$ on a tuning set, because they decided that a specificity of 99% was acceptable, and they wanted the recall to be as high as possible given this constraint. We wanted to see if we could get a perfect specificity without a significant degradation in recall, so we tried setting θ to the highest value for which $S = 100\%$ on the tuning set[15].

The results in Table 7 show the tradeoff between the two cases for both data sets. When we only required that $S \geq 99\%$ on the t uning data, it maintained that requirement on the test data, and S was significantly higher when we required that $S = 100\%$ on the training data. However, in the latter case the recall dropped significantly to 71% and 69%.

[15] See Section 2.3.3 for definitions of recall and specificity.

Table 7: Varying the Threshold

	MDDAS Data		TRS Data	
	Recall	**Specificity**	**Recall**	**Specificity**
S ≥ 99%	**97.222%**	99.853%	**100.000%**	99.335%
S = 100%	69.444%	**99.990%**	70.606%	**99.989%**

5.3 Set-Valued Features

We transformed set-valued features into categorical features by replacing them with boolean features, one for each value that can occur in the features' sets [Cohen, 1996]. As a consequence of this, the learned model has multiple submodels for each set-valued feature. So, since we do not assign weights to the submodels, a set-valued feature has more influence on the average legitimacy score than any categorical or numerical feature. This might not be desirable.

One way to eliminate this imbalance is to, for each set-valued feature, average the legitimacy subscores from its submodels before averaging them with the rest of the subscores. That would change the average legitimacy score of the first example in Section 2.3.2, <good, 42, {water, sugar }>, from ——— = 80% to ——— = 67%, and the average legitimacy score of the second example, <bad, 100, {bread}>, would change from ——— = 20% to ——— = 11%.

However, our experimental results, presented in Table 8, showed that this modification significantly decreased the recall of the TRS data. This suggests that each member of a set-valued feature's values is as important as any categorical or numerical feature.

5.4 Unseen Values

Section 3.5 discussed the problem of unseen values and the way we address it by penalizing an instance with any unseen values. But an unseen value would seem to be a strong indicator of an attack, so we thought it might be better to automatically label an instance that has any unseen values as positive. However, our experimental results did not reveal any significant differences between the two approaches.

5.5 Counting Rejections

We investigated setting a limit on the number of submodels that may reject a given instance, where a submodel rejects an instance if one of three things happens: 1) The instance has an unseen value that causes the submodel to return a legitimacy subscore of 0 (Section 3.5), 2) for Decision Trees, the submodel returns a value that differs from the value of the corresponding

Table 8: Averaging Set-Valued Features

Average Set-Valued Features?	MDDAS Data		TRS Data	
	Recall	**Specificity**	**Recall**	**Specificity**
no	97.222%	99.853%	**100.000%**	99.335%
yes	96.667%	99.861%	98.182%	99.321%

Table 9: Counting Rejections

Maximum Number of Rejections	MDDAS Data		TRS Data	
	Recall	Specificity	Recall	Specificity
0	**100.000%**	94.641%	**100.000%**	96.940%
1	97.222%	99.853%	**100.000%**	98.943%
threshold	97.222%	99.853%	**100.000%**	99.335%
2	93.611%	99.914%	**100.000%**	99.630%
3	91.667%	99.950%	**100.000%**	**99.767%**
4	83.889%	99.972%	**100.000%**	**99.802%**
5	68.056%	99.985%	**100.000%**	**99.826%**
6	58.889%	99.993%	**100.000%**	**99.849%**
7	55.556%	99.995%	**100.000%**	**99.884%**
8	55.278%	**100.000%**	98.182%	**99.898%**
9	53.889%	**100.000%**	95.152%	**99.907%**

feature in the instance, or, 3) for Regression Trees, the value of the corresponding feature in the instance is not within 2 standard deviations of the value that the submodel returns.

We ran experiments varying the maximum number of submodels, M, that may reject an instance. If more than M submodels reject a given instance, then that instance is labeled '+'. Otherwise, it is labeled '–'. Notice that there is no need for a threshold value. The submodels can be viewed as monitors that each independently flag suspicious SQL queries, and if enough flags are raised, then the SQL query is declared to be suspicious.

Our results are presented in Table 9. The baseline approach, in which a threshold is used, is included in the table for comparison. The results show that the independent monitors significantly improve specificity if M is high enough ($M \geq 8$ is best for MDDAS and $M \geq 3$ for TRS). Also, in the extreme case in which M = 0 (a single rejection is sufficient to label an instance '+'), the recall was perfect on both data sets. In addition, although there is a recall-specificity tradeoff for MDDAS, it is possible to optimize both of them simultaneously on TRS with M = 3, 4, 5, 6, or 7.

6 Discussion

QAIDS uses CFA to learn from data with heterogeneous features and no positive training instances. It uses a supervised machine learning algorithm to predict each feature's value from the values of the other features. By basing an instance's label on the number of submodels that reject each instance, QAIDS can distinguish between positive and negative instances with R = 97.2% and S = 99.9% on MDDAS and R = 100% and S = 99.9% on TRS.

As we move forward, we are expanding this work into two different areas. 1) We intend to evaluate QAIDS on attacks created by real hackers. In order to accomplish this, we have organized a red team of people who are attempting to exfiltrate specific prohibited data from a database. This exercise will test QAIDS on a range of varying, real world attacks. 2) We hope to extend our system to handle modifications to a database. The current implementation of QAIDS is only meant to protect against data exfiltration in static databases. However, if the database is

changing, the problem is more difficult. To see why, consider the feature ResultSize, an estimate of the total number of results to be returned. Since the size of the database might change if we allow data modification, the legitimate value of ResultSize might differ from that in the training data. We would need a way to account for changes in the values of legitimate feature vectors over time.

There are other things that we would like to do. We need a way to set the value of M, the maximum number of submodels that may reject an instance (Section 5.5). Its optimal value(s) is surely dependent on the number of features, among other things. Also, all of the experimental results from prior work in Section 2.3.3 used the Average Probability application-phase method. In light of our experiment in which Average Match Count was significantly better than Average Probability (Section 5.1), it might be worth trying Average Match Count in the other systems.

In our literature review, CFA was the only novelty detection method we were able to find that can handle heterogeneous features. So CFA fills a research gap, and we recommend considering it for any problem in which only one class of training instances are available and there are a variety of feature types.

CFA was a highly successful solution for our database security problem. Prior work also had success with CFA [Huang et al., 2003; Huang, 2006; Huang and Lee, 2003; Cabrera et al., 2008; Liu et al., 2007]. However, we believe that these three research groups, who all work in the field of MANET IDS, are the only ones to report using CFA before us. It seems clear that this outstanding machine learning method is worthy of greater exposure, particularly in the machine learning community.

7 Bibliography

Boley, D. Borst, V. Gini, M. (1999) “An Unsupervised Clustering Tool for Unstructured Data.” *Proceedings of the Machine Learning for Information Filtering Workshop at the International Joint Conference on Artificial Intelligence*. Stockholm.

Cabrera, João B.D. Gutiérrez, Carlos. Mehra, Raman K. (2008) “Ensemble Methods for Anomaly Detection and Distributed Intrusion Detection in Mobile Ad-Hoc Networks.” *Information Fusion*. Vol. 9(1). 96-119.

Cohen, William W. (1995) “Fast Effective Rule Induction.” *Machine Learning: The 12th International Conference*. Lake Taho, CA. Morgan Kaufmann.

Cohen, William W. (1996) “Learning Trees and Rules with Set-Valued Features.” *Proceedings of AAAI/IAAI*. Vol. 1. 709-716.

Fonseca, J. Vieira, M. Madeira, H. (2008) “Online Detection of Malicious Data Access using DBMS Auditing.” *SAC*. Fortaleza, Ceara, Brazil. 1013–1020.

Frank, Eibe. (1999a) “`J48.java`.” Revision 1.9. *The WEKA Data Mining Software*. The University of Waikato. Hamilton, New Zealand.

Frank, Eibe. (1999b) “`RepTREE.java`.” Revision 5535. *The WEKA Data Mining Software*. The University of Waikato. Hamilton, New Zealand.

Hall, Mark. Frank, Eibe. Holmes, Geoffrey. Pfahringer, Bernhard. Reutemann, Peter. Witten, Ian H. (2009) “The WEKA Data Mining Software: An Update.” *SIGKDD Explorations*. Vol. 11(1). http://www.cs.waikato.ac.nz/ml/weka/

Hartigan, J.A. (1975) *Clustering Algorithms*. Wiley.

Hu, Y. Panda, B. (2003) "Identification of Malicious Transactions in Database Systems." *IDEAS*. Hong Kong, China. 329–335.

Huang, Yi-an. (2006) *Intrusion Detection and Response Systems for Mobile Ad Hoc Networks*. Georgia Institute of Technology. Ph.D. thesis.

Huang, Yi-an. Fan, Wei. Lee, Wenke. Yu, Philip S. (2003) "Cross-Feature Analysis for Detecting Ad-Hoc Routing Anomalies." *Proceedings of the 23rd International Conference on Distributed Computing Systems*. Providence, RI.

Huang, Yi-an. Lee, Wenke. (2003) "A Cooperative Intrusion Detection System for Ad Hoc Networks." *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*. Editors: Setia, Sanjeev. Swarup, Vipin. ACM. 135-147.

Kamra, Ashish. Terzi, Evimaria. Bertino, Elisa. (2008) "Detecting Anomalous Access Patterns in Relational Databases." *The VLDB Journal*. Vol. 17. 1063–1077.

Knuth, Donald. (1997) *The Art of Computer Programming*. 3rd edition. Vol. 1.

Kohonen, T. (1982) "Self-Organized Formation of Topologically Correct Feature Maps." *Biological Cybernetics.* Vol. 43. 59-69.

Lanckriet, Gert R.G. El Ghaoui, Laurent. Jordan, Michael I. (2002) "Robust Novelty detection with Single-Class MPM." *Advances in Neural Information Processing Systems*. 15.

Lee, S. Y. Low, W. L. Wong, P. Y. (2002) "Learning Fingerprints for a Database Intrusion Detection System." *ESORICS*. Zurich, Switzerland. 264–280.

Liu, Yu. Li, Yang. Man, Hong. (2005) "MAC Layer Anomaly Detection in Ad Hoc Networks." *Proceedings of Sixth IEEE SMC Information Assurance Workshop*. 402-409.

Liu, Yu. Li, Yang. Man, Hong. Jiang, Wei. (2007) "A Hybrid Data Mining Anomaly Detection Technique in Ad Hoc Networks." *Proceedings of IJWMC*. Vol. 2(1). 37-46.

Lloyd, S. P. (1957) "Least Square Quantization in PCM." *Bell Telephone Laboratories Paper*.

Mitchell, Tom M. (1997) *Machine Learning*. WCB/McGraw-Hill. Boston, Massachusetts.

Quinlan, Ross. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers. San Mateo, California.

Schölkopf, B. Platt, J.C. Shawe-Taylor, J. Smola, A.J. Williamson, R.C. (1999) "Estimating the Support of a High-Dimensional Distribution." *Microsoft Research*. MSR-TR-99-87.

Tax, D.M.J. (2001) *One-Class Classification*. Delft University of Technology. Ph.D. thesis. ISBN: 90-75691-05-x.