Semantic Anomaly Detection in Online Data Sources

Orna Raz
Institute for Software Engineering International
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213 USA
orna.raz@cs.cmu.edu

Philip Koopman
Institute for Complex Engineered Systems
ECE Department
Carnegie Mellon University
Pittsburgh PA 15213 USA
koopman@cmu.edu

Mary Shaw
Institute for Software Engineering International
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213 USA
mary.shaw@cs.cmu.edu

ABSTRACT

Much of the software we use for everyday purposes incorporates elements developed and maintained by someone other than the developer. These elements include not only code and databases but also dynamic data feeds from online data sources. Although everyday software is not mission critical, it must be dependable enough for practical use. This is limited by the dependability of the incorporated elements.

It is particularly difficult to evaluate the dependability of dynamic data feeds, because they may be changed by their proprietors as they are used. Further, the specifications of these data feeds are often even sketchier than the specifications of software components.

We demonstrate a method of inferring invariants about the normal behavior of dynamic data feeds. We use these invariants as proxies for specifications to perform on-going detection of anomalies in the data feed. We show the feasibility of our approach and demonstrate its usefulness for *semantic anomaly detection*: identifying occasions when a dynamic data feed is delivering unreasonable values, even though its behavior may be superficially acceptable (i.e., it is delivering parsable results in a timely fashion).

1. Introduction

> "Reader Toby Doig nearly had a heart attack this morning when he visited Datek to check out his share portfolio and found that the Dow Jones Industrial Average (DJIA) had taken the full brunt of the stock market wobble and slumped just over 10,000 points to stand at 0.20 (down 99.999 percent, roughly)...clearly there had been a terrible computing error..." [25]

Everyday software must be sufficiently dependable for its intended purpose. Because this software is not usually mission-critical, it may be cost-effective to detect improper behavior and notify the user or take remedial actions. Detecting improper behavior depends on having a model of proper behavior. Unfortunately, specifications of everyday software are often incomplete and imprecise. The situation is exacerbated when the software incorporates third-party resources such as COTS (Commercial-Off-The-Shelf) software components, databases, or dynamic data feeds from online data sources. The latter case is especially difficult, because the proprietor of the data source may change its semantics, format, or even availability while it is being used; further, specifications for data sources, and therefore data feeds, are often even sketchier than those for software components.

We are interested in making the use of dynamic *data feeds* from online *data sources* more dependable. Examples of data sources include stock quotes, weather forecasts, airline ticket prices, and news reports. A data feed captures a particular usage of a data source: for example, stock quotes for a specific company, the weather forecast for a specific city, airline ticket prices for specific origin and destination, and news for a specific topic. Such data feeds are fallible, underspecified, and often unguaranteed — all factors that increase reuse risks. Like COTS components, data feeds present risks arising from functional deficiencies and mismatches in interfaces. In addition, data feeds remain under the control of their proprietors, often with no commitment for support, continuity, consistency, or notification of change. The resulting software systems are fragile, and their practical utility will depend on improving their dependability.

We are specifically interested in *semantic* problems with these data feeds — cases in which the data feed is responsive and delivers well-formed results, but the results are inconsistent, out of range, incorrect, or otherwise unreasonable. We focus on a particular facet of dependability: availability or readiness for usage, and change from the traditional fault model, "fail-silent" (crash failures), to "semantic".

Dependability is "the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers. The service delivered by the system is its behavior *as it is perceived* [emphasis in original] by its user(s)" [17]. Classical techniques for dependability assessment and control of potential risks must be strengthened to reflect the open and dynamic character of online data sources. Assessment should be dynamic and on-going, not just static and a priori. Control should include economic mitigation, in addition to prevention and technical mitigation (fault tolerance).

We investigate anomaly detection, which may be used as a basis for all these activities. In [24], we suggested software insurance for economic mitigation and discussed how anomaly detection may constitute a first step in that direction. Anomaly detection is also a step towards enabling dynamic assessment and technical mitigation, including enhancing dependability, preserving current quality of service, and healing. We believe anomaly detection is useful by itself in enhancing the users' perception of dependability: the more confident users are that they will be notified of problems, the more they will be able to rely on the service delivered. Fault tolerance approaches to failure detection require specifications of normal, degraded, and abnormal operations. Failure masking does not require detection, yet it requires specifications of outputs and their selection.

Unfortunately, specifications for software that incorporates dynamic data feeds are often even less satisfactory than specifications of COTS components. A software system constructed from data feeds may need to dynamically reconfigure these data feeds as they change or fail. This precludes a priori validation. Further, the sketchy and incomplete specifications of the underlying data sources may be too weak to support failure detection. For these non mission-critical data sources, the cost of providing complete specifications is likely to be much higher than the cost of a failure. Even if cost were not an issue, complete specifications would not be practical: a system that incorporates a data feed can depend on it in ways that could not be anticipated in advance, and developers of non-critical interactive systems rarely have the energy and patience to specify all possible cases.

Our approach observes the behavior of dynamic data feeds and infers invariants about their usual behavior. These invariants can augment the existing specifications to express expectations for the behavior of the data feed. They thereby become proxies for missing parts of the specifications. The augmented specifications can be used to detect anomalous behavior — that is, behavior that does not conform to the expectations. We rely on existing Statistical and Machine Learning techniques to infer these invariants.

We show it is possible to infer useful invariants about the semantics of dynamic data feeds from their behavior and that this can be done, to a large extent, automatically. Such invariants can be effectively used to discover semantic anomalies in these data feeds. By "effectively" we mean in a timely manner, within reasonable cost, with low error rate, where obvious (to a human) problems are detected, and where integration of such specifications as do exist is possible.

We introduce concepts and definitions in Section 2. Section 3 provides details about invariant inference and semantic anomaly detection. We show the feasibility of our approach via experiments which are described and discussed in Section 4. Related work is presented throughout.

2. Semantic quality

Online data sources are in particular need of improved dependability. On the one hand, timely availability greatly increases the usefulness of the data feed they provide and hence the value of this data. On the other hand, it is hard to detect anomalies in such data sources because of their incomplete specifications. Although anomalies in a data feed indicate anomalies in the data source, in our setting the system developer does not have control over the data source, so dependability enhancement can only be done by the client of the data. For the client, the data source is represented by the data feeds the client uses (Figure 1). We, therefore, concentrate on data feeds.

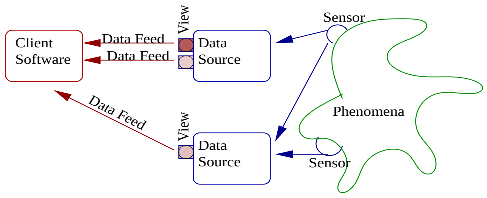

Figure 1: A client software uses a dynamic data feed that results from a view (query) on the content provided by a data source. A data source samples phenomena via sensors.

Online data sources exhibit various types of failures. In Section 2.1 we classify such failures and identify the type with which we are concerned. We need a common language to discuss anomalies and the data we use for invariant inference. Section 2.2 provides the necessary definitions.

2.1 Failure types

A *software failure* is either an outcome that violates the specifications of that software or an unexpected software behavior observed by the user. A *software fault* is the identified or hypothesized cause of the software failure [21]. We categorize failures in online data sources as either connectivity, syntax/form, or semantic. Our focus is semantic failures. We provide examples of faults that may cause failures of these different classes.

Connectivity failures are the inability to get any data or a very slow rate of service. Faults include server faults, such as "server overloaded" or "server down", and network faults, such as "router down" or "insufficient bandwidth". Solutions to individual types of faults exist, in the form of redundant servers, caching and archiving. Examples of tools providing such solutions are the Go!Zilla download manager [10], Alexa's archive of pages [1] and Google's caching [9].

Syntax/form failures are the inability to parse obtained data. Faults include data source changes, such as reformatting, service addition or removal, and client problems, such as a missing plug-in. Examples of solutions to individual types of faults are wrapper induction (a program for extracting data), wrapper re-induction (detecting problems in wrappers and adjusting to changes), and languages that facilitate information exchange among data sources (and support automatic parsing). Wrapper induction and re-induction learn the data prototype from labeled examples and require some domain specific information. Ariadne [14] and TrIAs [2] are examples of research prototypes for wrapper induction and re-induction. RAPTURE [15] is a research prototype for wrapper validation. Kangaroo [13] is a commercial tool that includes help in extracting information from HTML. Languages that facilitate information exchange require well defined semantic and syntactic standards. XML [28] is an example of a language facilitating information exchange among data sources. The Information Manifold research prototype [20] is an example of a system that provides access to a collection of closely related data sources by using a language to describe their contents and capabilities.

Semantic failures are unreasonable values of successfully parsed data. Faults include data source problems that do not cause connectivity or syntax/form failures, such as reporting data provided by an erroneous sensor, relying on erroneous data provided by a different data source or entered by a human, changing database schema, and changing the method of reporting. We could not find existing general solutions to such problems. The most relevant work seems to be wrapper re-induction and wrapper validation [14, 18, 2, 15], but these often require labeled data and domain specific information; we seek more general solutions. GritBot [11] is a commercial tool for finding anomalies in a data set to prepare it for data mining. It is domain independent, but provides only a single technique which is applicable to a single non-dynamic set of data with numeric and categorical attributes. We are interested in an approach that can handle various sources of data and various types of inconsistencies.

As demonstrated above, detection and repair techniques for individual types of faults exist for many of the connectivity and syntax/form problems, but not for semantic problems. Our work aims to provide general solutions for semantic problems, thus enhancing the semantic quality of data sources. We concentrate on detecting semantic anomalies.

We aim to detect those anomalies that will be perceived by the user as failures. However, in the absence of complete specifications, the same anomalous behavior sometimes indicates a failure and at other times does not. The example in Section 1 describes an extreme value change caused by a fault in a stock quote data source. However, stocks sometimes correctly exhibit extreme value changes. These cases cannot be distinguished without knowledge of the market, although both are anomalous.

We suggest an approach for empirically improving specifications of specific applications. We show how to infer invariants that serve as proxies for specifications and are used to detect anomalies.

The idea of inferring invariants from actual behavior is similar to dynamically discovering likely program invariants, which underlies the work of Ernst et al. [8, 7]. However, the domain, emphasis, and goal of our approach are different. We deal with online data sources rather than programs. We use off-the-shelf techniques rather than provide a new technique for inferring invariants. We aim to increase dependability rather than aid in programming tasks. The major difference between our work and intrusion detection (e.g., [16]) is the fault model. Our model is semantic, unintentional faults, whereas intrusion detection assumes malicious faults. In addition, intrusion detection research often concentrates on specific, sequence-related, techniques.

In terms of Anderson's classification of error detection checks [27], we provide primarily "reasonableness" checks: checks that use known semantic properties of the data to detect errors. However, in our setting, we cannot assume that requirements, or even the particular design of a module, are available. Rather than having known semantic properties, we *infer* potentially relevant semantic properties.

2.2 Definitions

An *anomaly* is an observed behavior of a data feed that is different from our expectation. This *expectation* describes normal behavior of the data feed. It combines whatever specifications are available ("explicit specifications") with inferred abstractions of the historical behavior of the data feed ("implicit specifications"). *Specifications* completely define how to use a component and what a user of a component can rely on. *Explicit specifications* are the specifications that have been explicitly provided for the data feed. *Implicit specifications* are likely invariants over the behavior of the data feed. We are interested in semantic failures, so the invariants we define describe consistent behavior of the data feed.

A *data feed* (Figure 1), DF, is a time-ordered sequence of observations. An *observation* at time t is a tuple of attribute values: $OBS_{i,j}(t) = \langle a_1, ..., a_k \rangle_t$ that contains information relevant to a data source DS_i and a query Q_j at a given time t. A *query*, Q, specifies a pre-defined (by the data source) set of content. For a dynamic data feed, the true values (of the observed phenomena) are not available before the data is used and may never be available.

An *attribute*, A, of a data source, has both type and origin. The *type* can be numeric, categorical or free (natural language). The *origin* can be direct (sensor data) or interpreted (the result of applying a model). A numeric attribute can be *comparable* or *non-comparable* to other attributes (excluding itself) of the same observation. In general, attribute comparability arises from the semantics of the data feed: two attributes are comparable if there is a meaningful relation between them. Because of the characteristics of the tools used in the experiments reported in Section 4, comparability is determined by the data types of the attributes (e.g., it is not meaningful to numerically compare the current value of a stock to its daily volume). For example, a stock quote data source DS_i may have the attributes current, daily high, and daily low. All these attributes are direct, numeric, and comparable. Attribute values for a particular stock may be requested by specifying the appropriate ticker symbol as the query value (e.g., $Q_j = SUNW$). As a result, an observation at time t may return $OBS_{i,j}(t) = \langle 16.2, 17, 16 \rangle$. The corresponding data feed, $DF \equiv DS_i, Q_j$, will consist of multiple, time-ordered, such observations.

A data feed can be *structured*, *semi-structured*, or *unstructured*. A structured data feed defines its own grammar (e.g., by XML). For a semi-structured data feed we can deduce a grammar (e.g., by hand-tuned scripts or information extraction methods such as wrapper induction).

An *invariant* is a function of a sequence of observations and an inference technique. An invariant can be inferred from a single data feed or multiple data feeds. The observations can be used either as *"stateless"* (singular points in time) or as *"stateful"* (time-series or unordered).

All the data feeds we deal with have a temporal aspect, and multiple data feeds may observe data sources that sample, possibly with modest delays, the same or different yet related phenomena. This raises the issue of synchronizing the results when using multiple data feeds. *Redundant* data feeds observe, for similar content, data sources that function as different sensors of the same underlying phenomena. An example is stock quotes for the same stock from multiple stock quote data sources. Attributes of redundant data feeds may have different names and may be scaled and displayed differently as well. We, therefore, require a mapping between attributes of redundant data feeds. *Correlated* data feeds observe data sources for content that is correlated in a way other than redundancy, e.g., causally correlated. An example is Pittsburgh rainfall amount and river level.

3. Anomaly detection

Fault tolerance approaches often use a state transition model with explicit transitions to model normal, broken and degraded operations. In previous work [23] we noted such models are difficult to work with when the specifications are inaccurate and suggested an alternative incremental-improvement model. Our approach to anomaly detection follows this model, overcoming the limitation of requiring precise definitions of states and transitions.

In this section, we present the most relevant details for inferring invariants to be used in semantic anomaly detection. We need to understand how to match data with existing Statistical and Machine Learning techniques. Section 3.1 defines the characteristics of our data and the implications of these characteristics on the techniques we can use. In Section 3.2, we discuss the kind of invariants we can infer, the actual act of invariant inference, and present a method for anomaly detection. In our discussion, we concentrate on a particular type of data, numeric-valued data, which is also the one we use in our experiments in Section 4.

3.1 Characteristics of data and techniques

The characteristics of our data influence the kind of techniques we can use for invariant inference. We identify these characteristics and their implications for choice of techniques.

We assume the data is normal most of the time. Nonetheless, our data is noisy because it is real world data. Therefore, the techniques we use need to be able to handle noisy data or be compatible with mechanisms that can handle noisy data. Voting is an example of such a mechanism and one which we use in this paper.

Learning is classified by the Machine Learning community into supervised, unsupervised and reinforcement, as defined in [3]. In *supervised learning*, the desired output is provided for each input pattern (this is often referred to as *labeled* data). In *unsupervised learning*, no target data is used (this is often referred to as *unlabeled* data). Instead of learning an input output mapping, the goal may be to model the probability distribution of the input or to discover some structure in the data. In *reinforcement learning*, feedback related to the outputs is provided (good/bad) but not the actual desired values. Reinforcement learning is usually used for control applications and is, therefore, not relevant here.

Using supervised learning to identify anomalies requires labeling training data with "normal" (*positive example*) and "anomaly" (*negative example*). But providing labels can be a difficult task. Even if a source of labels (e.g., an oracle) exists, it may be too expensive to use, because either domain specific information or manual intervention may be required. We can overcome this problem by not providing labels and using unsupervised learning techniques or by assuming the data provides only positive examples (similar to the assumptions made by some machine learning approaches to intrusion detection [16]) and using supervised learning techniques that can handle training data that consists solely of positive examples. The latter is a severe limitation, because many supervised learning techniques will learn to identify all unseen data as positive (thus never detect an anomaly).

However, there are cases for which labeled data is available. Though the labels are often not of the form "normal"/"anomaly", they may be useful. This enables us to take advantage of supervised learning techniques. A possible source of labels is a time lagged oracle. Even though we assume the true values become known too late for validating the information before usage, they may be acceptable for training. Using true values as labels can be effective if the time lag is not too large (it should be within a change cycle of the data) and if labeling is cheap. For example, for a weather forecast, there always exists an oracle in the form of actual weather observed. We could learn the daily predominant daytime weather (sunny, rainy, etc.). The training data (past weather conditions) already contains these labels.

$\{OBS_{i,j}(t)\}$	Criterion
(1) single data feed; stateless; comparable	internal consistency
(2) single data feed; stateful unordered	reasonable range
(3) redundant data feeds	timeliness accuracy
(4) correlated data feeds	inter-consistency

Table 1: Examples of determining semantic quality criteria by observation properties; numeric attributes.

In this paper, we use unlabeled data, and, therefore, unsupervised learning techniques. The techniques we use are (1) augmented Daikon and (2) Mean. Daikon [8] is a research prototype for dynamically discovering likely program invariants. Mean is a statistical method for estimating a confidence interval for the mean of a distribution based on sample measures. Further details follow in Section 4.2.

3.2 Inferring invariants

We attempt to infer invariants, using off-the-shelf techniques. We first discuss the difficulties of directly obtaining invariants. Then we describe our approach to inferring invariants and suggest a method for using these invariants for anomaly detection.

If complete and correct explicit specifications exist, there is no need to infer invariants. However, it is not realistic to assume this is the case, especially for the everyday software with which we are concerned. Moreover, even if some form of explicit specification exists, it is rarely complete or correct. Often, the explicit specifications do not match the actual behavior of the data feed. Inferred invariants could be used as a way of validating, enhancing, or evolving the explicit specifications.

If an oracle for the expected behavior exists, we can use it to obtain invariants. But oracles tend to be time-lagged or expensive, as discussed in Section 3.1. Humans provide another kind of oracle. Users are usually good at identifying an anomaly when they see it, but can very rarely give general rules for identifying it. In addition, it is usually not reasonable to expect users to inspect an entire body of data. Although experts may be able to provide general rules, it may be very hard to find an expert and it may be prohibitively expensive to get the information needed.

Our approach to the difficulties of obtaining invariants is to *infer* a useful subset of invariants. The available observations (whose structural properties are defined in Section 2.2) determine relevant semantic quality criteria. These criteria include: internal consistency, inter-consistency, timeliness, accuracy, reasonable range, and completeness. Table 1 gives examples of determining relevant criteria for data feeds of numeric attributes. The following lists these examples by rows: (1) use $\{OBS_{i,j}(t)\}$ over data source DS_i with attributes $\langle A_1, .., A_k \rangle$, $k > 1$, $\{A\}$ numeric and comparable, queried for Q_j, treating each $OBS_{i,j}(t)$ as stateless, to infer invariants that indicate internal consistency (the Daikon experiment in Section 4 below is an example); (2) view the observations as stateful, where $k = 1$ is acceptable, and the attributes do not need to be comparable, to infer invari-

Technique			Table1	Data
name	learning	noise	Row #	labeled
Daikon	unsupervised	augment	1	no
Mean	unsupervised	ok	2	no

Table 2: Examples of matching data and techniques for rows in Table 1.

Technique	Bias	Invariant form
Daikon	set of pre-defined invariants	arithmetic exp.
Mean	concept within conf. interval	arithmetic exp.

Table 3: Examples of inductive bias and invariant functional form.

ants that indicate reasonable ranges of values (the Mean experiment in Section 4 is an example); (3) use observations from m redundant data feeds $\{OBS_{1,j}(t)\}$,..,$\{OBS_{m,j}(t)\}$, with any number of numeric attributes, comparable or not, queried for the same information Q_j, and a mapping between attributes, to infer invariants that indicate timeliness, or invariants that indicate accuracy (treating observations as either stateless or stateful); (4) if the data feeds are correlated, use the observations to infer consistency of values of attributes from different data feeds (inter-consistency).

A series/set can always be treated as a collection of stateless observations, so any criterion relevant to a data feed treated as stateless will also be relevant for the same data feed treated as stateful. Similarly, any criteria relevant to a single data feed is also relevant when this data feed is one of multiple feeds.

An inference technique is relevant to specific data and semantic quality criteria. To find techniques for invariant inference, we match properties of the data with properties of the techniques. Properties of the data include the structural properties of observations together with a semantic quality criterion (e.g., a row in Table 1), as well as how the data is labeled. Properties of a technique are the required structural properties of its input, the criterion to which the invariants it can infer are relevant (again, exemplified by a row in Table 1), as well as the learning task it is suited for and its noise handling capabilities (as described in Section 3.1). For each technique, we also specify the form of the invariants it can produce. Tables 2 and 3 provide examples of a data–technique match.

The form of invariants may be any arbitrary function. However, we do not use explicit specifications nor domain knowledge in this paper, so we do not expect to infer complex models. For example, in the experiments described in Section 4, the invariants are simple arithmetic expressions. The form of invariants is determined by the technique we use to infer invariants (Table 3 provides examples). This is a result of the "inductive bias" of the technique. *Inductive bias*, in Machine Learning terminology, is the a priori assumptions regarding the identity of the target concept. Such assumptions are necessary for generalizing beyond the observed data [22].

Our approach consists of three major steps, as summarized in Figure 2: (1) initial inference of an expectation, (2) applying the expectation over unseen data to detect anomalies, and (3) updating the expectation. If additional data is available it can be used. For example, the underlying rate and character of change of the phenomenon may be useful in identifying normal changes in the data. Observa-

1. Obtain expectation
   (a) Find techniques that match the data
      i. Determine relevant criteria to infer
      ii. Match data with various candidate techniques
   (b) (Optional). Pre-processing: augment technique with noise handling capabilities; provide additional input
   (c) Apply each technique from step 1a to the data
2. Apply expectation (detect anomalies)
   (a) Apply expectation to newly observed data
   (b) (Optional). Post-processing: use attribute origin as a hint for comparison; use heuristics to reduce false alarms
   (c) If an invariant breaks, report anomaly
3. Update expectation
   (a) Update to reflect normal changes in data over time
   (b) Goto 2

Figure 2: Anomaly detection method

tions from other queries to the same data source may be added to the data in an attempt to avoid invariants that are too specific ("over-fitting", in Machine Learning terminology, e.g., invariants that fail to capture normal data that was not part of their training set or invariants that capture anomalies). Observations from redundant data feeds may be used in heuristics aimed at reducing the number of false alarms.

Expectation inference and update produce invariants that are likely to indicate an anomaly if they are broken. This is the empirical means we use to detect anomalies. In this paper we concentrate on steps 1 and 2. The need to update an expectation follows from the dynamic nature of the data. We defer this issue to future work.

4. Experiments

We demonstrate the feasibility of our approach by experiments on the single data feed cases of Table 1 and the techniques of Table 2.

We show it is possible to infer useful invariants for a single, semi-structured data feed of numeric attributes. The facets of usefulness we concentrate on are detecting semantic anomalies that would be obvious to a human and having no more than a reasonable number of false alarms. We consider a number reasonable if it is small enough to allow a human to hand-check the alarms.

We also demonstrate how detected anomalies help us understand implicit specifications of a data source.

We describe the experimental methodology in Section 4.1. In Section 4.2, we present the data and techniques we use for invariant inference. In Section 4.3, we present the experimental results, which we further discuss in Section 4.4.

4.1 Methodology

We want to improve incomplete specifications. We concentrate on the first two steps of our method for anomaly detection (see Figure 2): obtaining an expectation over a training set and applying the expectation to detect anomalies over a disjoint validation set. To get an indication of the strength of implicit specifications, we assume no explicit specifications are available. The expectation is then composed solely of inferred invariants.

We describe the experimental conditions in Section 4.1.1. We explain how we validate the experimental results of steps

1 and 2 in Sections 4.1.2 and 4.1.3, respectively. For step 2 we also explain the presentation of the results and present a voting heuristic we use in the anomaly detection process to decrease the number of false alarms.

4.1.1 Experimental conditions

We cover the first two entries of Table 1: a single data feed viewing observations as stateless and as stateful. We use techniques matched to the data feeds as indicated in Table 2: for the single stateless data feed, we use augmented Daikon, described in Section 4.2.2. For the single stateful data feed we use an empirical estimate for a confidence interval for the mean, described in Section 4.2.3.

We apply each experiment to each of three freely available, distinct, web-based, stock quote data sources, DS_0–DS_2 [4, 5, 6]. Each DS_i has multiple comparable numeric attributes, listed in Table 4.

The data feed is then $DF{=}OBS_{i,0}(t_0),...,OBS_{i,0}(t_0{+}k){=}DS_i, Q_0$, where DS_i, $i \in \{0,1,2\}$ is queried for a stock ticker symbol Q_0 (i.e., the same query is issued to all the data sources). The specific query value (stock symbol) is an arbitrary choice (with respect to invariant effectiveness) for data with similar change characteristics (beta and volume). To demonstrate this, we repeat all the experiments for two additional query values: Q_1 and Q_2.

Selecting only the comparable attributes might, in general, require manual intervention, thus limiting the automation of our approach. We test the importance of attribute selection by running two different variants for each experiment. In one, *comparable-attrs*, the data includes only the comparable numeric attributes. In the other, *all-attrs*, the data includes all numeric attributes.

To summarize, we have two types of experiments, corresponding to the technique used: Daikon and Mean. Each type of experiment has two variants: comparable-attrs and all-attrs. Each type of experiment is run over data $\{OBS_{i,j}(t)\} \mid i \in \{0,1,2\}$, $j \in \{0,1,2\}$. The data is divided into disjoint training and validation sets. In each experiment, the technique is used to infer invariants over the training set (step 1). The inferred invariants form the expectation and are applied to the validation set to detect anomalies (step 2).

4.1.2 Step 1: inferring invariants

For validation and analysis of the experimental results of step 1, a human judges whether the inferred invariants are intrinsic or incidental. *Intrinsic invariants* should always hold, due to the semantics of the attributes. *Incidental invariants* happen to hold over the training data but either should not hold in general or may change over time. Invariants inferred by Mean are likely to be incidental, as a result of treating dynamic data as stateful. Daikon infers both intrinsic and incidental invariants. For validation purposes, we compare the intrinsic invariants Daikon infers to what we believe it should infer. In addition, for each of Mean and Daikon, we compare all of the inferred invariants across experiments.

4.1.3 Step 2: applying inferred invariants

After inferring invariants from the training set these invariants are *applied* to the validation set: each invariant is evaluated for all observations of the validation set. Analysis of step 2 includes a measure of merit for the various parts constituting the result space of detection (see Figure 3).

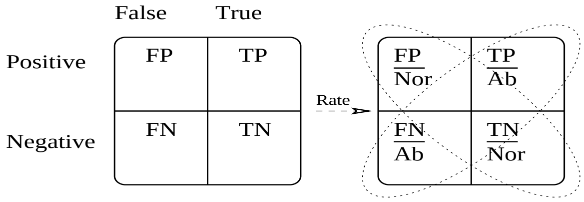

Figure 3: Result space. Right shows how we normalize the results. The ellipses mark 1-complements.

From our experiments we get:

- True Positives (TP): correctly detected anomalous data
- False Positives (FP): normal data falsely detected as anomalous
- False Negatives (FN): undetected anomalous data
- True Negatives (TN): correctly detected normal data
- Normal (Nor): Nor=TN+FP; all data that is actually anomaly-free
- Abnormal (Ab): Ab=TP+FN; all data with actual anomalies

To normalize the results to the range $[0, 1]$, we divide TP and FN by Ab, and TN and FP by Nor. We seek accurate anomaly detection techniques, those with low FP,FN and high TP,TN. Diagonal quantities are dependent — they are 1-complements. We need only display the independent measures. We chose the first row: *TP rate* and *FP rate*. These are identical to one minus the type I error rate (FN/Ab) and the type II error rate (FP/Nor) defined in [26]. When Ab=0 (entailing TP=0,FN=0) we define TP/Ab=1 and FN/Ab=0, to maintain the complement relation. Similarly, when Nor=0 we define TN/Nor=1 and FP/Nor=0. The *overall misclassification*: $\frac{FP+FN}{Nor+Ab}$ [26] (lower is better) summarizes these error rates.

Applying intrinsic invariants results in true positives only, by definition of intrinsic. To check if there are false negatives due to incomplete intrinsic invariants, we first determine all the intrinsic invariants we expect, given the attributes and the constraints on the kinds of relations Daikon infers. We then apply these invariants to all the validation sets.

Note that if an experiment is reported as having detected 100% anomalies (a TP rate of 1), it does not mean all possible anomalies are detected, just the anomalies reflected by the invariants. The detection capabilities are limited by two major factors: the inductive bias of a technique and incomplete information. Any technique that is able to generalize beyond seen data must have an inductive bias. In the experiments, we only count in Ab anomalies that are relevant to the inductive bias of the technique used.

The other factor limiting detection capabilities is incomplete information. When applying incidental invariants it is hard, or even not possible, to decide what is a false positive or a false negative, because no oracle is available to tell us what a specific value should have been. But we are interested in the kind of problems that can be recognized and understood by a human (which we call *identifiable*), not in determining what the actual value was nor in detecting precision problems. When in doubt, we consider the report to be false. Our reported numbers are, therefore, a worst case for the specific data and technique.

To detect anomalies, we apply the expectation (inferred invariants) with and without a voting heuristic. The voting heuristic cross-checks data with another data feed. The goal of the voting heuristic is to decrease the number of false positives and to detect some false negatives.

Without the voting heuristic, the invariants obtained from the training set in step 1 are applied over the validation set.

With the voting heuristic, additional available data (a redundant data feed) is used. The voting heuristic evaluates the invariants inferred from the training set of the tested data feed over the validation set of the tested data feed and over the corresponding observations in a redundant data feed. If an invariant breaks in both cases, it assumes the data has indeed changed (i.e., "normal" change) and no anomaly is reported. This is designed to reduce the number of false positives. A false negative is indicated if an invariant only breaks over the redundant data feed.

To validate the experimental results when the voting heuristic is used, we hand-check whether the voting heuristic has removed true positives or added false positives. We cannot always determine this, because the true value is often unknown. The fault tolerance community often uses some sort of voting, assuming independence of voters, to decide what a result should be when the truth is unknown. Our voting heuristic follows this approach and can be viewed as multi-version comparison. However, it will work only for independent data feeds. To have more confidence in the results of the voting heuristic, we can use a larger number of data feeds for voting, assuming it is unlikely that the majority of the underlying data sources will have the same subset of correlated attributes.

4.2 Data and techniques

We use real-world data in our experiments, as we describe in Section 4.2.1. The techniques we use to infer invariants are the program invariant detection engine of Daikon, augmented to handle noise, described in Section 4.2.2, and estimating a confidence interval for the mean, described in Section 4.2.3.

4.2.1 Data

We chose stock quote data sources because these are semi-structured, no oracle is available for the value of stocks at any arbitrary moment in time, and stock quote data sources include a number of numeric attributes, some of which are comparable.

We downloaded HTML pages that are the result of querying DS_0–DS_2 for a ticker symbol that is one of Q_0–Q_2 (CSCO, SUNW, TXN). Stock quotes are provided by each of these data sources with minimum delays of 20 minutes. This data was collected Mon–Fri, every ten minutes between 10am and 4pm, for about six weeks. Because we are interested in semantic anomalies, we ignored pages from all data sources if any had communication problems at a specific time. We also ignored syntax/form problems by manually adjusting our parsing scripts whenever the format of the HTML page changed. Each $OBS_{i,j}(t)$ results from parsing one HTML page.

Table 4 lists the numeric attributes of the data sources. Each resulting data feed has a subset of the following attributes: current value (cur), last value (last), change in value (change), highest and lowest values in 52 weeks (52h and 52l), highest and lowest daily values (dhigh and dlow), value when daily trade began (open), stock's anticipated fluctuations relative to the market fluctuations (beta), and stock volume (vol).

DS	Numeric attributes
0	*cur*, change, *last*, *52l*, *52h*, beta
1	*cur*, change, *52l*, *52h*, *open*, *dlow*, *dhigh*, vol
2	*cur*, change, *last*, *52l*, *52h*, beta, *open*, *dlow*, *dhigh*, vol

Table 4: Numeric attributes. Comparable attributes are in italic.

We use disjoint validation and training sets with equal sizes for each of the techniques. Each validation set has observations from a period of one week (about 170 observations). Each training set has data from two and a half weeks (425 observations). We use a common approach for dealing with time-changing data: a moving window [12]. We set the window size to three and a half weeks, where the last week in the window is the validation set and the rest is the training set. At the end of each week we replace the observations of the oldest week by the observations of the current week: so for data of six weeks we have three pairs of training and validation sets for each data feed. This is a simple way to update the expectation (step 3): re-infer the invariants over recent data.

Determining the appropriate size of a training set is a difficult task. Statistical approaches exist for simple cases. Unfortunately, they often make assumptions that do not hold for our data. In addition, they can only be applied when exact information about the statistical techniques used is available. As more theoretical results become available, we should incorporate them into our framework. However, because we use off-the-shelf techniques, full implementation details are not always available[1]. We empirically find a good training set size for Daikon (Section 4.2.2), treating it as a black box, and use the same size for Mean. In future work, we plan to develop heuristics for finding a training set size.

4.2.2 Daikon

We use the program invariant detection engine of Daikon. We reformat the raw data to the input format Daikon requires. The output invariants are relations that hold over the training data. Each relation can include one to three attributes.

Daikon was originally designed for finding invariants over program executions. This affects the vocabulary it uses and its assumptions regarding its input. The first is not a problem: we map Daikon's program points and variables to our observations and attributes, respectively. Unfortunately, Daikon assumes there is no noise in the data. While justified for data structure invariants, this assumption is not justified for our data. Therefore, we augment Daikon with a noise handling capability: we use Daikon as a black box over distinct subsets of a training set. We only take invariants that are frequent (in this experiment, appear at least twice). This can be viewed as a form of voting. This way, we use Daikon to discover invariants that should *usually* (rather than always) hold.

Daikon needs data with enough instances of distinct values to justify an invariant. However, the more data (larger period of time) the more likely it is to contain an anomaly, thus falsifying a valid invariant. The period of time over which the training data should be collected depends on the change characteristics of the data. We chose stocks that have large beta, implying frequent changes. We empirically

[1] Full details are available for Daikon because it is described in technical papers and distributed in source form.

DS	Invariants
0	$cur \leq 52h$, $cur \geq 52l$
1,2	$cur \leq dhigh$, $cur \geq dlow$, $dhigh \leq 52h$, $dlow \geq 52l$, $dhigh \geq open$, $dlow \leq open$

Table 5: Intrinsic invariants inferred by Daikon

found the time constant for one stock and it applied to the other stocks as well. For our data, data from half a week is sufficient for Daikon to infer a single set of invariants. Because we need to augment Daikon with noise handling capabilities, inference of several initial sets of invariants is required for creating the final set. The cost of this augmentation is more data. Training sets of two and a half weeks (425 observations) seem to suffice.

4.2.3 *Mean*

We use a technique that provides an estimate for a symmetric confidence interval for the mean (μ) of the distribution of an attribute. This is estimated separately for each attribute and is relevant only for this attribute. The output invariants are an interval for each attribute.

The technique we use is a very simple one, based on the form of a confidence interval for the mean of a normal distribution. We would get a similar form of a confidence interval for μ by using Hoeffding's inequality with the sample variance. Hoeffding's inequality does not make any assumptions about the underlying distribution. It does, however, assume that the samples are independent. This in not true for our data: the current value depends on previous values.

Let $X_1, ..., X_n$ denote a random sample from a normal distribution. Then $(\bar{X} - \frac{c\sigma'}{\sqrt{n}}, \bar{X} + \frac{c\sigma'}{\sqrt{n}})$ is a confidence interval for μ, with a $(1-\alpha)$ confidence coefficient, where μ is the true mean, $\bar{X} = \frac{1}{n}\Sigma_{i=1}^{n} X_i$ is the sample mean, ${\sigma'}^2 = \frac{1}{n-1}\Sigma_{i=1}^{n}(X_i - \bar{X})^2$ is the sample variance, and c is an arbitrary constant.

If our data were normal or the samples independent, we could fix α and find the appropriate c, n. Because this is not the case, we choose both n and c to be large, and make no claims about the confidence coefficient. We empirically chose $c = 50$. As for Daikon, $n = 425$ observations.

4.3 Results

We find that the inferred invariants (step 1) are useful in detecting anomalies in the tested data feed. Furthermore, anomalous behavior of a data feed, detected by applying the inferred invariants to unseen data (step 2), suggests feasible implicit specifications of the data source. We explain these results further: in Section 4.3.1 we look at the inferred invariants; in Section 4.3.2 we look at the detected anomalies.

4.3.1 *Step 1: inferring invariants*

The invariants in comparable-attrs are a proper subset of the invariants in all-attrs for both Daikon and Mean, because the attributes in comparable-attrs are a proper subset of the attributes in all-attrs. All invariants over two attributes (Daikon) in comparable-attrs are intrinsic, because comparable-attrs includes only attributes that are meaningful to compare.

Daikon: intrinsic invariants are similar both within a single query value Q_i (ticker symbol) and between different queries. Table 5 shows these invariants.

Daikon's output helped us to identify intrinsic invariants.

		Q_0			Q_1			Q_2		
		DS_0	DS_1	DS_2	DS_0	DS_1	DS_2	DS_0	DS_1	DS_2
Ab										
D	atr	0	34	52	4	1	44	8	0	0
	obs	0	14	45	4	1	44	3	0	0
M	atr	0	20	32	4	0	0	8	0	0
	obs	0	14	16	4	0	0	3	0	0
Nor										
D	atr	2040	3026	3519	2008	3017	3477	2032	3060	3570
	obs	510	496	465	499	502	459	502	510	510
M	atr	2040	3040	3538	2008	3018	3521	2032	3060	3570
	obs	510	496	494	499	503	503	502	510	510

Table 6: Total number of abnormal (Ab) and normal (Nor) data in the validation sets, found by either technique. Counted by attributes (atr) and by distinct observations (obs), in the comparable-attrs variant for Daikon (D) and for Mean (M)

We had some relations in mind, yet after examining the inferred invariants, we realized additional relations should hold. For example, in advance of these experiments, we did not think of invariants related to the attribute open.

In the majority of the experiments (26 out of 27 for each of comparable-attrs, all-attrs), all intrinsic invariants were inferred. Often, an inferred invariant does not include equality (e.g., $<$ rather than $\leq$), because the training examples do not include equality. In one experiment (DS_2,Q_2) two invariants were missing ($dhigh \leq 52h$, $dhigh \geq open$). This is due to the training data containing anomalies related to the involved attributes in at least four of the subsets used in the noise handling augmentation.

Incidental invariants over one attribute differ slightly within a single query, indicating normal changes in data, and significantly between different queries, as expected for different stocks. In one experiment (for each of comparable-attrs and all-attrs; DS_2,Q_0) Daikon learned an anomalous value ($52l = 8$). This is balanced by intrinsic invariants that detect these observations as anomalous. In all-attrs, incidental invariants over two non-comparable attributes exist, due to the different units used for these attributes. Examples include: $cur < vol$, $dlow > change$.

Mean: all Mean invariants are incidental. These invariants always involve specific values of an attribute, for example: $42.31 \leq cur \leq 63.65$. Although such values may be inherent to an attribute, resulting in an intrinsic invariant, this was not the case for our data. We examine $\bar{X}$ and σ'. These vary significantly between different attributes. For a specific data source, query, and attribute, the values are rather similar. For specific query and attribute across different data sources, the values are usually similar. Cases that are not similar may suggest a difference in implicit specifications, as discussed in Section 4.4.4 below.

4.3.2 *Step 2: applying inferred invariants*

We apply the invariants inferred in step 1 over the validation set. An anomaly is detected if an invariant does not hold over the validation set. The results are summarized in Table 6 and in Figure 4.

A single observation can trigger multiple warnings, because it contains multiple attributes. An observation is anomalous if it contains at least one anomalous attribute. The numbers we report in Table 6 are, for each comparable-attrs variant of an experiment (Daikon or Mean applied to

$\{OBS_{i,j}(t)\}$ for all DS_i,Q_j combinations), the total abnormal data and the total normal data, as reported by the two techniques, counted by attributes and by distinct observations.

The number of observations is the same for both comparable-attrs and all-attrsvariants. The number of attributes is larger in all-attrs: 33% more for DS_0, 25% more for DS_1, and 30% more for DS_2. The majority of identifiable anomalies involve invariants solely over comparable attributes. Only in one case is there an identifiable anomaly over a non-comparable attribute (*beta*), in three different observations (for Q_2, DS_0). Some anomalies involve multiple attributes of the same observation (Table 6, Q_0, DS_1 and DS_2; Q_2, DS_0: the number of abnormal attributes is larger than the number of abnormal observations). It seems that often an anomalous attribute indicates the raw data used by the data source was somehow corrupted and therefore other attributes are anomalous as well. In addition, some attributes are calculated based on others (e.g., $52l$ is based on *dlow*) and an anomaly might propagate.

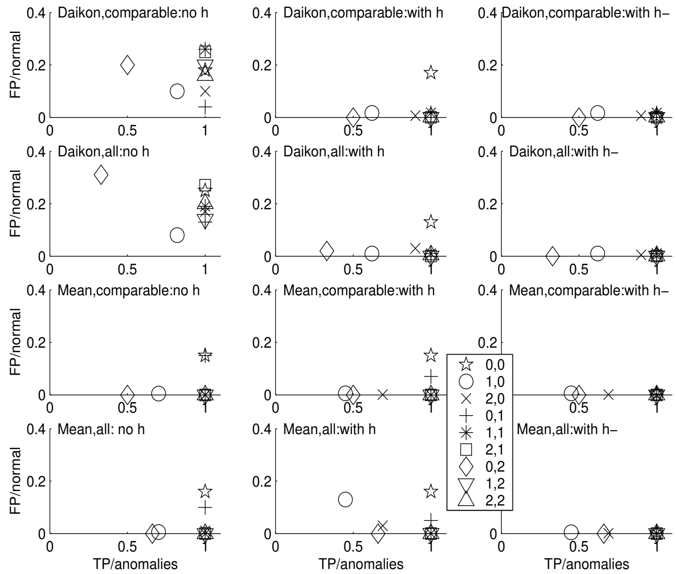

Figure 4: TP (True Positives) ratio vs. FP (False Positives) ratio. Best possible result: $(TP/anomalies, FP/normal) = (1, 0)$. Each data feed is indicated by a single marker denoted in the legend by i,j, corresponding to DS_i,Q_j. Rows 1–2 depict results for Daikon, rows 3–4 for Mean. For each of Daikon/Mean, first row is the comparable-attrs variant, second row is all-attrs. First column is results without heuristic (h); second with h; third with h, after manually removing attributes that seem to be computed differently by the different data sources (h-).

Figure 4 shows that Daikon and Mean detect most of the anomalies in the tested data feeds (most of the points are at the lower right corner), with TP/Ab near 1 and FP/Nor less than 0.3. The overall misclassification is always under 0.3. With the voting heuristic it is always under 0.15 and usually under 0.02, and with the voting heuristic after manually removing attributes that seem to be computed differently by different data sources (h-; Section 4.4.4) it is always under 0.02. Although the worst cases we encountered have a lot of data points, most of these data points were misclassified due to only one or two invariants — so it was easy to deal with manually. In addition, the voting heuristic is very effective in removing false positives, provided attributes in different data sources are independent.

4.4 Discussion

In Section 4.4.1 we compare the all-attrs and comparable-attrs variants and find that they produce similar results. In Section 4.4.2 we compare the two techniques we use and find it is helpful to use both. In Section 4.4.3 we demonstrate how detected anomalies can help us find feasible implicit specifications. Section 4.4.4 discusses the voting heuristic, which we find to be very effective in reducing the number of false positives, but not the number of false negatives. Remaining challenges, including a discussion of how automatic we can make our approach, are presented in Section 4.4.5.

Whenever we indicate DS_i,Q_j the experiment can be found in Figure 4 by using the indices i, j to locate the appropriate marker symbol.

4.4.1 comparable-attrs vs. all-attrs

The same anomalies are detected in both comparable-attrs and all-attrs variants. Using all numeric attributes sometimes produces more false alarms than using comparable attributes only. This is to be expected, because the total number of attributes is larger. Still, the results are good, as summarized at the end of Section 4.3.2. This is due to comparable-attrs invariants being a proper subset of all-attrs invariants and to the additional all-attrs invariants rarely triggering anomalies. For Daikon, non-comparability of attributes helps to explain the latter: the units are calibrated differently. Comparability of attributes is not meaningful for Mean, because it uses only a single attribute per invariant. These results are encouraging, as they indicate attribute selection is not a major issue for Daikon and for Mean. Due to the similarity of comparable-attrs and all-attrs, the following analysis is relevant to both.

4.4.2 Daikon vs. Mean

We found Mean useful for inferring invariants over single attributes and Daikon useful for inferring invariants over multiple attributes. Daikon invariants for single attributes contain exact values, which may result in a large false positives rate (Figure 4). However, this is easily solved by ignoring these few invariants. There are some anomalies that Mean does not and cannot be expected to detect. These are anomalies that stem from a relation that should hold between attributes, where the values for each attribute are reasonable (not very different from other values of the attribute). In our experiments these anomalies are discovered by the Daikon intrinsic invariants: $open \leq dhigh$, $open \geq dlow$ (data feeds: Q_0,DS_2; Q_1,DS_1 and DS_2), and incidental invariants for $52h$ and $52l$ that accompany the intrinsic invariants: $dhigh \leq 52h$ and $dlow \geq 52l$ (Q_0,DS_1). However, Mean is able to identify anomalous attributes that Daikon cannot. In the case of DS_0,Q_2 (0,2 in Figure 4), a Daikon incidental invariant involving $52l$ detects 4 anomalous observations. The same observations are detected by Mean invariants involving *beta* and *last*. The Mean invariant involving $52l$ does not detect anomalies. Only by using both techniques do we notice that these observations include multiple anomalous attributes.

In addition, the combination of Daikon and Mean invariants makes it easier to decide what anomalies are true.

Whereas Daikon intrinsic invariants are easy to understand, Mean can explain some anomalies detected by applying the Daikon incidental invariants, which otherwise seem accidental. For example, for DS_0,Q_1 Daikon $52h = 129.31$ does not hold in 4 observations and for DS_0,Q_2 Daikon $52l = 35.00$ does not hold in 4 observations. This might be due to normal changes. But Mean invariants involving $52h$ or $52l$ break as well. Manual inspection of the data shows that these values are inconsistent: other values before, between, and after these observations have not changed. In general, when both Daikon and Mean invariants break over a observation, our confidence regarding the validity of the warning increases.

4.4.3 *Exposing implicit specifications*

Even though the data sources underlying the data feeds we test provide a similar service, and may even use the same raw data, the anomalies detected in the experiments help us expose some differences in the behavior of the data sources, beyond the stated 20 minute delay. Such differences are a manifestation of incomplete specifications and entail feasible implicit specifications. Daikon and Mean invariants involving $dhigh$, $52h$, $dlow$, $52l$, detect anomalies in DS_1,Q_0 and in DS_2,Q_0, that help us expose such implicit specifications. DS_2 immediately updates the $52h$ or $52l$ value whenever it is exceeded by the $dhigh$ or $dlow$ value; DS_1 does not. The behavior of DS_2 causes inconsistent values of $52l$ or $52h$ whenever the $dlow$ or $dhigh$ values are anomalous, which is detected by Mean invariants and by Daikon incidental invariants. The behavior of DS_1 causes $dhigh \leq 52h$ or $dlow \geq 52l$ to break whenever $dlow$ or $dhigh$ are anomalous.

4.4.4 *Voting heuristic*

The voting heuristic is usually very effective in eliminating false positives for both Daikon and Mean. Moreover, it sometimes exposes additional implicit specifications. The heuristic is not effective in eliminating false negatives. However, the number of false negatives is small and all the cases in which anomalous attributes are missed include multiple anomalous attributes for a single observation. At least one of these attributes is always detected.

The voting heuristic sometimes either adds false positives (Figure 4, Mean, all-attrs, DS_1,Q_0 and DS_2,Q_0), as a result of problems at the redundant data feed, or does not reduce the number of false positives significantly (DS_0,Q_0), as a result of the data feeds being dependent. Using more data feeds for voting should alleviate these problems, as we hope only a minority of data sources are dependent with respect to the same attributes. Dependent data feeds can also cause the voting heuristic to eliminate true positives, as consistently happens for DS_1,Q_0 and DS_2,Q_0, where both data feeds have the same anomalous values for $dhigh$, $dlow$, hinting the underlying data sources may be using the same raw data feed. However, the detection is still effective because the processing of this raw data (e.g., $52l$, $52h$) seems to be independent (as exposed in the description of the implicit specifications in Section 4.4.3), and the voting heuristic works well with independent voters.

If the voting heuristic is ineffective in reducing false positives even when a larger number of data feeds is used in voting, true differences among the data sources may exist (implicit specifications). We found attributes that seemed to be calculated and updated differently in the different data sources by looking at these cases (DS_0,Q_0 for attributes $52l$ and $beta$; Mean, DS_1,Q_0 and DS_2,Q_0 for vol). For example, by looking at $52l$ in DS_0 and in DS_2 we discover additional implicit specifications of DS_2: we already know $52l$, $52h$ change immediately with $dlow$, $dhigh$. In addition, we realize that whenever the values of $dhigh$ or $dlow$ do not exceed those of $52h$ or $52l$, $52h$ and $52l$ are updated infrequently: less frequently than once in a few weeks. DS_0 seems to update $52h$, $52l$ about every week. For DS_1 and DS_2 we notice differences in the calculation of vol, which seem to exist only for Q_0 (the values of vol are consistently and significantly different). This is an example of implicit specifications that apply only to a specific stock. Demanding that such specifications be made explicit is not reasonable.

Suspected anomalies help us understand the origin of each attribute (direct vs. interpreted), which gives us hints as to reasons for differences in values (implicit specifications). For direct attributes, the reasons are usually different update policies. For interpreted attributes, the reasons may also include different calculation methods. Such suspected anomalies may indicate true problems at one of the data feeds. However, because we cannot determine this, we indicate these cases as false positives. We should have a way to flag these as suspected different implicit specifications.

The graphs involving $h-$ in Figure 4 display the results of applying the voting heuristic after manually eliminating attributes that seem to be computed differently by different data sources. This is part of checking our former assumption regarding the heuristic exposing implicit specifications. We check whether a large number of false alarms involves attributes which we have identified as being computed differently. We see that this is indeed the case. Manually eliminating such attributes always improves the results (reduces false positives). However, to verify that this is indeed due to different computation methods, we would need additional information about the data sources.

4.4.5 *Remaining challenges*

The major challenges which we still need to address are how automatic we can make our approach and demonstrating the generality of our approach.

We aim to make our approach for anomaly detection as automatic as possible. We want user intervention to be inversely proportional to its expected frequency. Automation challenges are related to the data and techniques. These challenges include attribute selection, determining training set size, and handling normal changes in the data. We do not consider attribute extraction an obstacle to automation, because Information Integration research [19] deals with providing solutions for automatic extraction. Attribute selection can be a difficult task and may require user intervention. This is a subject of current research in Machine Learning and Data Mining and we plan to take advantage of this research. Finding a good training set size is a challenge, as we discussed in Section 4.2.1. The need to update the expectation arises from the dynamic nature of the data, because normal changes should not be identified as anomalies. Handling concept drift is an area of current research in Machine Learning [12]. Techniques adjusted for concept drift or traditional techniques that are re-applied over a moving window or with partially forgetting older data may be used to update the expectation. Many techniques have parameters which are data dependent, but no mechanisms for adjusting the parameters. These parameters need to be stated and explained as part of the information supplied for matching data and techniques in our approach, because it might not always be possible to provide heuristics for setting these parameters.

We believe our approach should generalize over different types of data feeds, including those with categorical and free attributes[2], over different criteria, over different candidate techniques, and when some specifications do exit. We hope our approach can provide a unifying high level view for various types of data feeds.

We also believe that using multiple, independent data feeds can improve the effectiveness of our approach. In this paper we presented an example using a single data feed. We plan to support using data from a single as well as multiple data feeds.

5. Conclusions

We have presented our approach for inferring invariants to be used as proxies for missing specifications in semantic anomaly detection. The work presented is the first step in providing a method to assess and improve the dependability of dynamic data feeds.

The experimental results demonstrate it is possible to infer useful invariants over a single data feed of numeric attributes and that this can be done, to a large extent, automatically (in the context of stock market tickers). In particular, Daikon, which was originally designed for detecting program invariants from program executions, and which we augmented to handle noise, is effective in inferring useful invariants that indicate internal consistency, when viewing observations as singular points in time. Estimating a confidence interval for the mean is effective for inferring useful invariants that indicate reasonable values, when viewing observations as stateful.

The inferred invariants were effective in discovering semantic anomalies in a data feed. Moreover, we were able to deduce implicit specifications of the underlying data source. The number of false positives and false negatives was reasonable for a human to handle. The voting heuristic described in Section 4.1.3 worked well in reducing the number of false positives. Nonetheless, more data feeds are needed for voting (we assume values for each attribute are independent among the majority of such data feeds). Attribute selection was not a major issue for the data and techniques we used. Moreover, except for the selection of a training set size and an additional parameter (c for Mean), invariant inference and anomaly detection were fully automated.

6. Acknowledgments

We greatly thank Michael Ernst, Aaron Greenhouse, Rosie Jones, Gal Kaminka, Beth Latronico, Roy Maxion, and Dan Pelleg for helpful discussions and suggestions.

This research was supported by the National Science Foundation under Grant CCR-0086003 and by the Software Industry Center at Carnegie Mellon University.

References

[1] Alexa browser enhancement. www.alexa.com. Accessed April 2001.

[2] M. Bauer and D. Dengler. Trias: Trainable information assistants for cooperative problem solving. In *3rd Int'l Conference on Autonomous Agents*, 1999.

[3] C. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1996.

[4] Stock quotes data source. finance.northernlight.com. Accessed September–November 2000.

[5] Stock quotes data source. qs2.cnnfn.com. Accessed September–November 2000.

[6] Stock quotes data source. quote.pathfinder.com. Accessed September–November 2000.

[7] M. Ernst. *Dynamically Discovering Likely Program Invariants.* PhD thesis, University of Washington, Department of Computer Science and Engineering, 2000.

[8] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. In *IEEE Transactions of Software Engineering*, pages 99–123, 2000.

[9] Google search engine. www.google.com. Accessed April 2001.

[10] Go!Zilla download manager. www.gozilla.com. Accessed April 2001.

[11] GritBot. http://www.rulequest.com/gritbot-info.html. Accessed January 2002.

[12] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *7th Int'l Conference on Knowledge Discovery and Data Mining*, 2001.

[13] Kangaroo. www.kangaroonet.com. Accessed August 2001.

[14] C. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: A machine learning approach. In *Data Engineering Bulletin*, 1999.

[15] N. Kushmerick. Regression testing for wrapper maintenance. In *AAAI-99*, pages 74–79, Orlando, 1999.

[16] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *ACM Transactions on Information and System Security*, volume 2, pages 295–331, August 1999.

[17] J. Laprie. *Dependability: Basic Concepts and Terminology.* Springer-Verlag, Vienna, 1991.

[18] K. Lerman and S. Minton. Learning the common structure of data. In *17th National Conference on Artificial Intelligence*, 2000.

[19] A. Levy, C. Knoblock, S. Minton, and W. Cohen. Trends and controversies: Information integration. In *IEEE Intelligent Systems*, volume 13, pages 12–24, 1998.

[20] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *22nd Int'l Conference on Very Large Data Bases*, September 1996.

[21] M. R. Lyu. *Handbook of Software Reliability Engineering.* IEEE Computer Society Press and McGraw-Hill, 1996.

[22] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[23] O. Raz and M. Shaw. An approach to preserving sufficient correctness in open resource coalitions. In *10th Int'l Workshop on Software Specification and Design*, 2000.

[24] O. Raz and M. Shaw. Software risk management and insurance. Position paper. In *3rd Workshop on Economics-Driven Software Engineering Research*, 2001.

[25] Dow Jones average collapses to 0.20. TheRegister, March 19 2001. www.theregister.co.uk/content/28/17700.html.

[26] P. Runeson, M. Ohlsson, and C. Wohlin. A classification scheme for studies on fault-prone components. In *Product focused software process improvement*, pages 341–355, 2001.

[27] W. Torres-Pomales. Software fault tolerance: A tutorial. Technical Report NASA/TM-2000-210616, NASA Langley Research Center, Hampton, Virginia, 2000.

[28] XML 1.0, W3C recommendation. w3c.org, http://www.w3.org/TR/2000/REC-xml-20001006. Accessed Nov. 2001.

[2] However, free text raises additional challenges which are beyond the scope of our work