AN ABSTRACT OF THE THESIS OF

Andrew Emmott for the degree of Master of Science in Computer Science
presented on June 18, 2020.

Title: A Benchmarking Study of Unsupervised Anomaly Detection Algorithms

Abstract approved: _____

Thomas Dietterich

It is common practice in the unsupervised anomaly detection literature to create experimental benchmarks by sampling from existing supervised learning datasets. We seek to improve this practice by identifying four dimensions important to real-world anomaly detection applications — point difficulty, clusteredness of anomalies, relevance of features, and relative frequency of anomalies — and then proposing how to simulate and control these factors when sampling points during benchmark creation. We apply this methodology to produce a large corpus of unsupervised anomaly detection benchmarks and then evaluate several state-of-the-art anomaly detection algorithms against this corpus. Our final analysis not only compares the performance of these algorithms across a large variety of problems, but it also assesses the impact of our identified problems dimensions on experimental outcomes.

A Benchmarking Study of Unsupervised Anomaly Detection Algorithms

by

Andrew Emmott

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 18, 2020
Commencement June 2021

Master of Science thesis of Andrew Emmott presented on June 18, 2020.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Andrew Emmott, Author

ACKNOWLEDGEMENTS

Original versions of this work listed Shubhomoy Das, Thomas Dietterrich, Alan Fern and Weng-Keen Wong as authors. The contributions of Shubhomoy Das include implementing the EGMM and LODA algorithms and advising on analysis strategies. The remaining authors worked in advisory capacities and were connected to this work via its relationship to the DARPA ADAMS project. The methodology, ideas, implementation and analysis presented here are the work of the primary author, Andrew Emmott.

This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract W911NF-11-C-0088 and in part by the Future of Life Institute (futureoflife.org) FLI-RFP-AI1 program under grant number 2015-145014. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Future of Life Institute, DARPA, the Army Research Office, or the US government.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF APPENDIX TABLES

LIST OF ALGORITHMS

Algorithm Page

Chapter 1: Introduction

Unsupervised anomaly detection is an important inference task with applications across many different domains including identifying novel attacks in computer security [27, 39, 28, 37], discovering novel astronomical phenomena [45], detecting broken environmental sensors [13], identifying machine component failures [47, 48, 1], and finding cancer cells in normal tissue [38, 18]. In all of these application domains, the data are a mixture of points that are "nominal" (i.e., reflecting normal behavior) and "anomalous". The anomalous data points are created by a process that is distinct from the process that is generating the nominal data. In computer security, for example, the anomalous points are created by adversarial attacks. In detecting cancerous cells, it is the cancer that is creating the anomalies. Based on this observation, we define an anomaly to be a data point that is generated by a process that is different from the process that is generating the nominal data. The goal of unsupervised anomaly detection algorithms is to detect these anomalous data points without labels by instead leveraging the assumption that these points are rare and meaningfully different from the nominal points.

1.1 Objective

Unsupervised anomaly detection as described above may identify a general class of problems but real applications of algorithms come from a variety of domains with their own unique concerns. Computer security faces adversarial threats where machine failure prediction does not. Particular cancers may have a well-defined behavior profile but cancer in general might not. Domain experts in a particular field may be able to select features for an application, and other times it may be preferred to use as much information as is available.

The objective of this study is to present and evaluate a framework of experimental design for testing unsupervised anomaly detection algorithms. We are motivated to do this because of the lack of care in experimental design in much of the unsupervised anomaly detection literature, particularly prior to our first publication on the topic in [15]. Further, we want to apply this methodology to make more nuanced claims about the success and a failure modes of various algorithms as well as to demonstrate the deficiencies in the simpler and more common approaches to experimental design found in the literature.

To achieve this, the methodology presented is used to create a large corpus of anomaly detection benchmarks by taking standard datasets for supervised learning and systematically varying four dimensions important to real-world anomaly detection applications: point difficulty, clusteredness of anomalies, relevance of features and relative frequency of anomalies.

We apply eight state-of-the-art anomaly detection algorithms to the corpus.

These algorithms are leading representatives of four main approaches to anomaly detection: density estimation, quantile boundary estimation, distance-based and projection methods. We summarize the result of each micro-experiment in terms of the Area under the ROC Curve (AUC) and the Average Precision (AP).

To understand the relative importance of each factor, we fit regression models at various scales to predict these metrics as a function of the benchmark factors and provide analysis of variance (ANOVA) of these models.

1.2 Background

Despite the importance of anomaly detection, the field of statistical anomaly detection lacks a standard methodology for understanding and evaluating proposed algorithms. Most published experiments evaluate their algorithms via application-specific case studies or ad hoc synthetic datasets. There are two consequences of this. First, it is very difficult to compare different algorithms to assess progress in the field. Second, it is difficult to understand the various factors or dimensions of anomaly detection problems that influence the performance of anomaly detection algorithms. This makes it difficult for experiments to guide research in algorithm development.

This situation has recently begun to change. Our 2013 workshop paper [15] introduced a systematic evaluation methodology based on repurposing existing supervised learning datasets. A notable aspect of this methodology is that we systematically vary aspects of the anomaly detection problem to gain more insight

into the impact of these aspects on algorithm performance. Two other recent papers report similar studies. Goldstein and Uchida [17] conducted a study of 19 different methods, and Campos, et al., [9] compared 12 different distance-based methods. Neither of these studies incorporated systematic variation of problem factors. An additional shortcoming of the Campos, et al. study is that many algorithm parameters, such as the number of nearest neighbors, were selected to optimize algorithm performance on test data. This favors algorithms with many adjustable parameters and does not evaluate the extent to which parameter values can be automatically selected.

Building on our previous work [15, 14], this thesis reports a careful benchmarking study of state-of-the-art anomaly detection algorithms. We develop and test a standardized evaluation methodology for statistical anomaly detection. The methodology consists of taking existing supervised learning benchmark datasets and converting them into anomaly detection benchmark datasets by manipulating four factors that affect algorithm performance:

- Point difficulty: The degree to which the anomaly points are buried within the nominal (i.e., non-anomaly) points.
- Clusteredness: The extent to which the anomaly points are clustered together.
- Irrelevant Features: The extent to which the average Euclidean distance between points is determined by features that are completely irrelevant to the problem.

- Relative frequency: The proportion of anomalies in the dataset.

Greater detail on how this work differs from our previous work can be found in Appendix A.

The remainder of this thesis is organized as follows. In Chapter 2 we first define the anomaly detection problem that we are addressing and compare it to alternative formulations in Section 2.1. In Sections 2.2 and 2.3 we review and assess existing approaches to the evaluation of anomaly detection methods. Based on this, in Section 2.4 we identify a set of requirements for experimental methodology and define the four factors to be manipulated in the experiments. In Chapters 3 and 4, we present our benchmarking methodology and provide detailed procedures for meeting our requirements and manipulating the four factors. In Chapter 5, we present the set of anomaly detection algorithms assessed in this study with parameterization details provided in Appendix E. In Chapter 6 we validate the quality of the benchmark corpus produced. In Section 6.1 we describe the statistical hypothesis tests we apply to each benchmark and provide a summary of these hypothesis tests and their results. In Section 6.2 we introduce our first linear regression models and evaluate our corpus in an algorithm-agnostic way. In Chapter 7 we present a more in-depth analysis of algorithm performance across many settings. In Chapter 8 we provide a global discussion of the findings presented in this study and provide recommendations for how these findings should impact future work in the field.

Chapter 2: Topic Review and Problem Definitions

2.1 Anomaly Detection Tasks

Three different anomaly detection problem settings can be defined. In *supervised anomaly detection*, we are given a collection of N data points $x_1, \ldots, x_N$, each a d-dimensional real-valued vector. Each data point is labeled as either "nominal" or "anomalous". The goal is to learn a classifier $f : \mathbb{R}^d \mapsto \{nominal, anomaly\}$ that can correctly label new data points drawn from the same distribution. While this is no different than binary classification in theory, two important differences are often observed. First, anomalies are typically rare, so there is a huge class imbalance in favor of the nominal points. Second, each anomaly may be anomalous for a different reason, so the anomaly points do not form a coherent class of the kind that supervised learning algorithms are designed to model.

In *clean anomaly detection*, the training data all belong to the NOMINAL class. The goal is to output a scoring function $f : \mathbb{R}^d \mapsto \mathbb{R}$ that assigns low scores to future nominal points but assigns high scores to future anomalies.

Finally, in *unsupervised anomaly detection*, the data consist of an unlabeled mixture of the "nominal" and "anomalous" points. The goal is to learn a scoring function f that assigns higher scores to all of the anomaly points and lower scores to all of the nominal points. We can distinguish two flavors of unsupervised anomaly

detection: "static" and "inductive". In the static version, the goal is to determine which of the *training set points* are anomalous, whereas in the inductive version, the goal is to apply the learned function f to detect anomalies in *future test data*.

An important question in any application of anomaly detection is whether it is safe to assume that the anomalous data points belong to a well-defined probability distribution. We call this the *Well-Defined Anomaly Distribution (WDAD)* assumption. In some applications, such as detecting repeated machine failures, the WDAD assumption is reasonable. But in other settings, such as detecting novel cyberattacks, the WDAD assumption is invalid. Some algorithmic approaches make the WDAD assumption. For example, if we apply standard supervised learning to the supervised anomaly detection task, we are assuming that future anomalies will come from the same distribution as past anomalies. Similarly, in the inductive unsupervised anomaly detection setting, one approach is to fit a mixture model to the data with the idea that one mixture component will fit the nominal points and the other component will fit the anomalous points. Applying this fitted model to new data will not succeed unless the WDAD assumption holds.

An alternative approach that does not make the WDAD assumption is to search for statistical outliers and then assert that these outliers are the anomalies. This is known as the "outliers as anomalies" approach. It makes the *Distinctive Anomaly (DA)* assumption that the anomalous points are sufficiently different from the nominal points that they can be detected as outliers. The "outliers as anomalies" approach can be applied to clean anomaly detection and to the static and inductive flavors of unsupervised anomaly detection. In this study we will focus on

this approach. We will evaluate performance in the static, unsupervised anomaly detection setting.

2.2 Metrics for Evaluating Anomaly Detection Algorithms

There are two main ways that anomaly detection algorithms are applied. In data cleaning applications, the goal is to screen out all of the anomalies from the data. Hence, we choose a threshold θ and filter out all data points x for which $f(x) > \theta$. A natural metric for evaluating such methods is the Area Under the ROC Curve (AUC), because it summarizes the behavior of the learned function f over all possible settings of θ. The AUC is also equivalent to the probability that—given one anomaly point x_a and one nominal point x_n selected uniformly at random—the function f will assign a higher score to the anomaly point than to the nominal point: $f(x_a) > f(x_n)$.

In security and fraud detection settings, the anomaly detection system typically presents the top k candidate anomaly points to an expert analyst who must then decide whether to expend resources investigating the point further. In such cases, we are interested in the quality of the ranking of anomalous points with a focus on the points ranked at the top of the list. We could employ Precision-at-k, but this requires us to choose a specific value of k, and this is generally determined by application considerations. Hence, we adopted the Average Precision (AP) metric. It summarizes the precision of the top-ranked elements for the values of k corresponding to the ranks of the true anomaly points. The Average Precision is

equal to the area under the Precision-Recall curve (if the area computation does not use trapezoidal interpolation between true positives).

Many other metrics have been studied including Normalized Discounted Cumulative Gain (NDCG), F1 (the harmonic mean of precision and recall), and Recall-at-k. Each of these requires application-specific choices, so we decided not to include them in our study.

In this study, we consider only AUC and AP.

2.3 Existing Experimental Methodology

In anomaly detection research, three kinds of data have been employed to analyze and evaluate anomaly detection algorithms. First, there are datasets drawn from specific application problems (e.g., [47, 28]). Second, there are synthetic datasets [41]. Third, there are datasets constructed by taking an existing supervised classification problem and treating one or more of the classes as the anomalies.

Application-specific datasets are very useful. They can help us understand and evaluate the algorithm refinements needed to achieve high performance in a particular application. However, often these datasets are not publicly available because of privacy or security considerations (e.g., [43]).

Synthetic datasets [29, 30, 41] permit the systematic manipulation of some properties (e.g., the relative frequency of the anomalies, the distinctiveness of the anomalies, etc). However, decades of experience in machine learning have shown that real datasets are much more complex and idiosyncratic than synthetic data,

which undermines the validity of this approach [31, 32].

Finally, the repurposing of supervised classification datasets has the desirable property that the different classes are the result of different generating processes and the data retain the idiosyncrasies of the real application (e.g., [29]). Most studies have treated the datasets "as is" without trying to manipulate properties of the data. Some researchers [24, 9, 17] downsample the anomaly class to reduce the relative frequency of the anomalies. Another interesting case is the work of Das, et al., [12], who generated anomalies by permuting features among a small subset of the data. In a few cases, supervised regression datasets have been repurposed by treating the data points with the most extreme values as anomalies [11].

We propose to combine the idea of repurposing supervised learning datasets with the idea of systematically varying properties of the data. To better motivate our methodology, we will define our requirements for a good anomaly detection experiment and the properties of the experiment we wish to manipulate.

2.4 Requirements for Anomaly Detection Experiments

As discussed above, although anomaly detection algorithms work by searching for statistical outliers, the goal is to identify points that are generated by a process that is distinct from the process generating the "nominal" points. This distinction leads to the first two requirements for benchmark datasets.

Requirement 1: Nominal data points should be drawn from a real-world

generating process.

Requirement 2: Anomalous data points should also be drawn from a real-world process, but one that is distinct from the process generating the nominal points. The anomalous points should not just be points in the tails of the "nominal" distribution. For example, Glasser and Lindauer's [16] inside threat anomaly generator takes care to define distinct processes for normal and anomalous behavior.

Requirement 3: Many benchmark datasets are needed. If we employ only a small number of datasets, we risk developing algorithms that only work on those problems. More important, presenting results on many benchmarks at a time makes for more robust and reliable reported results. While the corpus of benchmarks presented in this study may be well in excess of what is needed for a reliable experiment, Section 7.5 will illustrate the potential consequences of using too few data sources.

Requirement 4: Benchmark datasets should be characterized in terms of well-defined and meaningful problem dimensions. Applications of anomaly detection often face different challenges across domains. Experiments in the literature sometimes describe such challenges and propose strategies for addressing them, such as in Liu, Ting and Zhou [30]. It is of practical value to real-world applications that experiments acknowledge which domain-specific challenges they might be addressing.

2.5 Proposed Problem Dimensions

There is currently no established set of problem dimensions for anomaly detection. We have identified four dimensions that we believe are important, but we consider this only the first step toward a full explanatory theory of anomaly detection. We introduce the concepts here; how we measure these problem dimensions is explained in Chapter 3; and how we use these measures to generate our benchmarks is explained in Chapter 4.

2.5.1 Point difficulty

The outliers-as-anomalies assumption breaks down as the anomaly points become harder to distinguish from the nominal points. One aspect of applying anomaly detection in adversarial settings (e.g., intrusion detection or insider threat detection) is that adversaries try to blend in with the distribution of nominal points. We propose *point difficulty* as a measure of the similarity of the anomalous data points to the nominal ones. When the targets are not confined to extreme outliers, or when the extreme outliers are not anomalies, the anomalies of interest will be confused with nominal points or with uninteresting outliers.

This phenomenon has also been referred to as "swamping" [29].

2.5.2 Semantic Variation

A common aspect of many anomaly detection applications is that there can be multiple processes generating anomalies. In a cyber-security setting, there can be many different kinds of attacks and many different methods for stealing information. In cancer detection, there can be many different biological processes that result in cancerous cells. On the other hand, if there are many instances of anomalies from one generating process, they may cease to appear as statistical outliers at all; such anomalies are often described as clustered anomalies. We propose *Semantic Variation* as a measure of the degree to which the anomalies are generated by more than one underlying process, or, alternatively, the degree to which the anomalies are dissimilar from each other.

When anomaly points are tightly clustered, this creates a region of high probability density, which can defeat density estimation-based methods. This phenomenon has also been called "masking" [29].

2.5.3 Feature Relevance/Irrelevance

From the application perspective, there is a natural tendency to include any feature that could conceivably be informative, but this tendency also increases the risk of including features that are irrelevant to the task. It is well-established that irrelevant features can degrade the performance of supervised learning methods, and we now have many good algorithms for identifying and removing irrelevant features. We believe that irrelevant features are an even greater problem for anomaly

detection. From the statistical perspective, each irrelevant feature increases the dimensionality of the space, and the sample size required by (naive) density estimation methods tends to scale exponentially with the dimension. In addition, as the dimensionality of the data increases, the "surface area" of the volume containing the data also increases, which is a geometric way of saying that there are more "tails" in which the data may lie. This increases the risk that nominal points will fall in the tails of the distribution. For all these reasons, it is important to measure the effect of irrelevant features on the performance of anomaly detection algorithms.

2.5.4 Relative frequency

Relative frequency is the fraction of the incoming data points that are anomalies of interest. This is the problem dimension that is most reliably reported in the literature already and has also been called "plurality" and "contamination rate". Little is done to examine the impact it has on results, however. The behavior of anomaly detection algorithms often changes with the relative frequency. If anomalies are very rare, then methods that pretend that all training points are "nominal" and fit a model to them may do well. If anomalies are more common, then methods that attempt to fit a model of the anomalies may do well. In most experiments in the literature, the anomalies have a relative frequency between 0.01 and 0.1, but some go as high as 0.3 [24, 29]. Many security applications are estimated to have relative frequencies in the range of 10^{-5} or 10^{-6}.

Understanding the impact of relative frequency is a fundamental issue in anomaly detection: How much can the data be contaminated by anomalies before the anomalies can no longer be reliably detected?

Chapter 3: Benchmarking Methodology

With the scope, requirements and definitions given in Chapter 2 we can now describe how we generate benchmarks for this study. In summary, we employed the following steps to construct our benchmark datasets:

1. Select a set of existing supervised datasets ("parentsets") derived from real-world contexts.
2. Determine a ground truth label for each point: "nominal" or "anomaly".
3. Compute point difficulty scores to each individual point.
4. Specify problem dimension settings for each benchmark.
5. Select points from a given parentset according to specifications to construct each benchmark.

3.1 Selecting Datasets

We selected all UCI [3] datasets (as of the beginning of this study) that matched the following criteria:

- *task*: Classification (binary or multi-class) or regression. No time series.
- *instances*: At least 1000. No upper limit.

- *features*: No more than 200. No lower limit.
- *values*: Numeric only. Categorical features are ignored if present. No missing values, with one exception (see below).

To ensure objectivity, the choice of these criteria was not guided by the performance of any specific anomaly detection algorithms.

Our criteria do not cover all settings in which anomaly detection is appropriate. Instead, we focused on the common case: high-dimensional, continuous-valued, independent and identically distributed (IID) data. Future work should explore nominal and ordinal features [34] as well as more structured (non-IID) settings such as time series [19, 33] and network data (e.g., [8]).

These criteria yielded a collection of 19 datasets, which we refer to as the "parentsets", since they will produce thousands of "child" benchmark datasets. The 19 selected parentsets are the following:

- *binary classification*: MAGIC Gamma Telescope, MiniBooNE Particle Identification, Skin Segmentation, Spambase
- *multi-class classification*: Steel Plates Faults, Gas Sensor Array Drift, Image Segmentation, Landsat Satellite, Letter Recognition, Optical Recognition of Handwritten Digits, Page Blocks, Shuttle, Waveform, Yeast
- *regression*: Abalone, Communities and Crime, Concrete Compressive Strength, Wine, Year Prediction

Communities and Crime is the one exception to our rule against missing values. In this case, there were some features for which the values where missing for the majority of points. We removed these features from the dataset rather than remove the dataset from our study.

In each of these parentsets, each feature was normalized to have zero mean and unit sample variance.

3.2 Synthetic Control parentset

In our final statistical analysis, one value of each factor will be chosen as the reference or baseline value. To improve interpretability, it is useful to have a baseline value for each factor that functions as the control group for that factor. While the focus of our analysis will be on algorithm performance and the impact of our problem dimensions, we should also be able to measure the impact of using "real" datasets against a synthetic baseline. We created a synthetic control parentset for this purpose. For details of its construction, see Appendix B.

3.3 Defining Nominal versus Anomalous Data Points

A central goal of our methodology is that the "nominal" and "anomalous" points should be produced by semantically distinct processes (Requirements 1 and 2). To achieve this for each of the 19 parentsets, we assign each point $x_i \in \mathbf{X}$ a label y_i to indicate whether it is a candidate nominal or candidate anomaly, informed by

the semantics of the original set. We employed the following methods to do this.

3.3.1 Binary Classification Problems

For datasets that were already binary classification problems, the data is already partitioned into two semantically-distinct groups. We chose one class as "candidate nominal" (i.e., the set from which we will select the "nominal" points) and the other as "candidate anomaly" (i.e., the set from which we will select the "anomaly" points). The class with fewer instances is chosen to be the candidate anomaly class. We do this because the final benchmarks will subsample the candidate anomalies so that the anomalies constitute a small fraction of all of the data points. The larger the majority class, the easier this is to achieve. In the case that both classes are of equal size, the class with greater variance is defined to be the candidate anomaly class; we do this because the greater variance might give us additional flexibility in selecting loosely or tightly clustered anomalies at benchmark construction time.

For parentsets that are regression or multi-class problems, our approach is to transform them into binary classification problems and then treat them as described here.

3.3.2 Regression Problems

For regression datasets, we compute the median of the regression response and partition the data into two classes by thresholding on this value. To the extent that

low versus high values of the response correspond to different generative processes, this will create a semantic distinction between the candidate nominal and candidate anomalous data points. We expect points near the median response will have high point difficulty and points near the extremes will have a low point difficulty. We expect benchmarks derived from regression problem sets might allow for flexible (and easy) control of point difficulty.

3.3.3 Multi-class Problems

For multi-class datasets, we partitioned the available classes into two sets with the goal of maximizing the difficulty of telling them apart. For parentsets with many classes, it can be impractical to try every partition of the classes in the search of the most confusing binary problem, so we employ an approximation that attempts to maximize class confusion; see Appendix C for details.

3.4 Assigning Point Difficulty Scores

Before sampling parentsets to generate benchmarks, we need additional meta-data so that we can characterize individual points better to meet benchmark specifications. In our case, the only additional meta-data we need is to assign point difficulty scores to each point, but future research may wish to insert other meta-data generating processes here.

We generate a vector of point difficulty scores $\boldsymbol{\phi}$ by training an oracle that

knows the true generating processes underlying the "nominal" and "anomalous" points. The intuition is that some measure of how far away a point is from the decision boundary will indicate how easy it is to distinguish from the opposite class.

Our oracle is trained by a constructing a training set $\{(x_i, y_i)_{i=1}^N\}$ consisting of all N points from the parentset and then applying kernel logistic regression (KLR; [21, 49, 23]) to fit a probabilistic classifier that estimates the probability $P(\hat{y}_i = \text{nominal}|x_i)$. The point difficulty of a candidate anomaly point $x_i|y_i = \text{anomaly}$ is defined as $P(\hat{y}_i = \text{nominal}|x_i)$. The intuition is that difficult anomalies will be those that are "buried" inside the nominal points so that even an oracle trained on the true labels assigns it high probability of being nominal. Similarly, the point difficulty of a candidate nominal point $x_i|y_i = \text{nominal}$ is the predicted probability $P(\hat{y}_i = \text{anomaly}|x_i)$ that it belongs to the anomaly class.

The end result is that each data point x_i is assigned a point difficulty score $\phi_i = 1 - P(\hat{y}_i = y_i|x_i)$ where ϕ_i closer to 0 means the oracle correctly classified x_i with greater confidence, ϕ_i closer to 0.5 indicates x_i is closer to the decision boundary and ϕ_i closer to 1 means the oracle incorrectly classified x_i with greater confidence.

For further details on our implementation of KLR, see Appendix D.

3.5 Specifying Problem Dimension Settings

We can now describe a fully labeled parentset as a set of data points $\mathbf{X}$ with associated ground truth label $\mathbf{y}$ and point difficulty scores $\boldsymbol{\phi}$. As is common in the literature, we will generate benchmarks by sampling data points from these parentsets. However, with the exception of relative frequency, the problem dimensions we propose to measure and manipulate are typically not addressed in other experiments. To measure the impact of this, and to evaluate the value of applying our more rigorous methodology, we want a control setting for each problem dimension that is equivalent to conducting an experiment where the problem dimension was not even considered at all. For example, elsewhere in the literature, when an experiment does not consider clusteredness, the sampling process simply did not consider the semantic variation among the candidate anomaly points; this approach would be the control group setting.

For each problem dimension, we will define the control setting, describe how the problem dimension can be measured as a property of the benchmark, and suggest how it can manipulated. Keep in mind that for each control setting, the benchmarks that result from that setting will still have a measuarable value in that problem dimension. For example, a benchmark in the point difficulty control group will not include consideration of point difficulty in the sampling process, but the resulting benchmark will still have a real-valued point difficulty score.

The specifics of how our actual sampling process works will be covered in Chapter 4.

3.5.1 Point Difficulty

We described how we assigned point difficulty scores in Section 3.4. The control setting for point difficulty is simply to disregard point difficulty scores when sampling points from the parentset. The measure of point difficulty a benchmark receives is the mean $\bar{\phi}$ of the point difficulty scores of all points selected for it. In addition to the control group, we propose four target range specifications for benchmark point difficulty (pd). A benchmark created at the pd_2 setting, for example, would be required to have a final point difficulty score in the range specified below.

- pd_0: control group; ($\bar{\phi} \in (0, 1)$)
- pd_1: $\bar{\phi} \in (0, 0.1\overline{6})$
- pd_2: $\bar{\phi} \in [0.1\overline{6}, 0.\overline{3})$
- pd_3: $\bar{\phi} \in [0.\overline{3}, 0.5)$
- pd_4: $\bar{\phi} \in [0.5, 1)$

3.5.2 Semantic Variation and Clusteredness

We define the semantic variation among anomalies with a measure of *normalized clusteredness*, which is a measure of the variance of the anomalies normalized against the variance of the nominal points. Because we are assuming the par-

entsets have independent features, we define the sample variance $\hat{\sigma}^2$ of a set of d-dimensional points $\mathbf{X}$ to be the sum of the sample variances of each feature or

$$\hat{\sigma}^2_{\mathbf{X}} = E[(\mathbf{X} - E[\mathbf{X}])\dot{(}\mathbf{X} - E[\mathbf{X}])] = \sum_{i=1}^{d} \hat{\mathrm{Var}}(\mathbf{X}_i)$$

For a benchmark with a set of nominal points $\boldsymbol{\beta}_n$ and anomaly points $\boldsymbol{\beta}_a$, we define normalized clusteredness (ν) as follows

$$\nu = \log\left(\frac{\hat{\sigma}^2_{\boldsymbol{\beta}_n}}{\hat{\sigma}^2_{\boldsymbol{\beta}_a}}\right). \tag{3.1}$$

When ν is less than 0, the anomaly points exhibit greater semantic variation than the nominal points (they are more scattered). When ν is greater than 0, the anomaly points are more tightly packed than the nominal points (on average).

The control setting for normalized clusteredness is to not consider it during the sampling process. In addition to the control group, we propose two target range specifications for benchmark normalized clusteredness (nc).

- nc_0: control; (clusteredness not considered, $\nu \in \mathbb{R}$).
- nc_1: $\nu < 0$; (scattered anomalies).
- nc_2: $\nu > 0$; (clustered anomalies).

3.5.3 Feature Irrelevance

The right way to define irrelevant features for outlier detection is unclear. To justify our desire to add irrelevant features to our benchmarks, we introduce the notions of "feature precision" and "feature recall".

In a supervised learning setting, it is assumed that all features necessary to the task are present. In this way, we would say that a supervised learning dataset has very high or even perfect "feature recall." In contrast, a supervised learning dataset might include many irrelevant features. Indeed, many supervised learning algorithms are able to in some way determine that only some of the features are required to attain optimal classification accuracy. In this way, we might describe a supervised learning dataset with only a few necessary features to have low "feature precision".

Given that our parentsets exist in a supervised setting, we can assume that any benchmark generated with them would already have high feature recall. However in real-world unsupervised anomaly detection domains, there is no such guarantee of high feature recall. To compensate for this, real-world applications could include as many features as they can, knowingly trading off high feature precision for higher feature recall. We want to add irrelevant features to some benchmarks in order to simulate this process of sacrificing feature precision in order to obtain the high feature recall that we already know is present in the data.

While the original parentsets might have varying amounts of feature precision, we don't seek to quantify this. Instead we simplify our assumptions and quantify

the amount of feature irrelevance (α) as follows. For a set of points $\mathbf{X}$ with no added features and a set $\mathbf{X}'$ which is the same set but with some number of additional features added, we measure feature irrelevance with α, which we define to be the ratio of the sums of the pairwise distances between all points in $\mathbf{X}'$ and $\mathbf{X}$.

$$\alpha_{\mathbf{X},\mathbf{X}'} = \frac{\sum\limits_{x'_i \in \mathbf{X}'} \sum\limits_{x'_j \neq x'_i \in \mathbf{X}'} ||x'_i - x'_j||_2}{\sum\limits_{x_i \in \mathbf{X}} \sum\limits_{x_j \neq x_i \in \mathbf{X}} ||x_i - x_j||_2}$$

The control setting for feature irrelevance is to simply not add any irrelevant features to the benchmark resulting in $\alpha = 1$. In addition to the control group, we propose three target specifications for benchmark feature irrelevance (fi).

- fi_0: control group; $\alpha = 1$ (no added irrelevant features).
- fi_1: $\hat{\alpha} = 1.2$
- fi_2: $\hat{\alpha} = 1.5$
- fi_3: $\hat{\alpha} = 2.0$

3.5.4 Relative Frequency

This problem dimension is very easily understood, measured and manipulated; if we desire that a benchmark has a relative frequency (ρ) of 0.01 then we simply ensure that the benchmark draws 0.99 of its points from the candidate nominals and 0.01 of its points from the candidate anomalies.

While it is a problem dimension that is commonly reported and manipulated in the literature, there does not exist a rigorous examination of the impact of this value on experimental outcomes.

Additionally, because this problem dimension is already frequently accounted for in the literature, the control setting for this problem dimension might appear counter-intuitive or even controversial. It can be easy to see that, for example, the proper control setting for point difficulty is to not consider point difficulty at all. It is less obvious that a correct control group setting for relative frequency is to simply not enforce any particular ratio of of anomalies in the benchmark data. Unlike our other control groups, this does not broadly reflect current practices in unsupervised anomaly detection literature, but it does provide a proper baseline for evaluating the impact that manipulating relative frequency can have on experimental outcomes versus not manipulating it.

In addition to the control group, we propose five target specifications for benchmark relative frequency (rf).

- rf_0: control group; (relative frequency is not considered, $\rho \in (0, 1)$).
- rf_1: $\rho = 0.001$ of the benchmark is drawn from the candidate anomalies.
- rf_2: $\rho = 0.005$ of the benchmark is drawn from the candidate anomalies.
- rf_3: $\rho = 0.01$ of the benchmark is drawn from the candidate anomalies.
- rf_4: $\rho = 0.05$ of the benchmark is drawn from the candidate anomalies.
- rf_5: $\rho = 0.1$ of the benchmark is drawn from the candidate anomalies.

Chapter 4: Benchmark Corpus Generation

Including all control settings, we have identified 20 parentsets, 5 point difficulty specifications, 3 normalized clustered specifications, 4 feature irrelevance specifications and 6 relative frequency specifications resulting in 7,200 different benchmark specifications. Because we don't want any particular specification to be subject to the whims of random selection, we further propose producing 5 benchmark replicates at each specification setting, resulting in up to 36,000 benchmarks for our corpus. However, five unique benchmarks at any given specification might not be possible and for some parentsets certain problem dimension settings might not be feasible, so the final corpus will be smaller than 36,000.

We formally define a benchmark specification as desired target ranges of $(\bar{\boldsymbol{\phi}}, \nu, \alpha, \rho)$ over labeled parentset $(\mathbf{X}, \mathbf{y}, \boldsymbol{\phi})$ with the intent of outputting a benchmark set that meets the specifications $\boldsymbol{\beta} \subset \mathbf{X}$ where $\boldsymbol{\beta} = \boldsymbol{\beta}_n \cup \boldsymbol{\beta}_a$ and $\boldsymbol{\beta}_n = \{x_i | y_i = \text{nominal}\}$ and $\boldsymbol{\beta}_a = \{x_i | y_i = \text{anomaly}\}$.

The process of achieving this can be very briefly summarized as follows: Points are sampled for the benchmark from the parentset one at a time. For each sampling, only feasible points are considered. (Points that don't break the benchmarking specifications). When the maximum budget of points is reached or when there are no more feasible points for selection, the process ends. If necessary, a number of irrelevant features are then added.

Algorithm 4.1 provides greater detail, but understanding it requires introducing a few more concepts.

4.1 Determining the Maximum Number of Points to Select

We have two practical concerns during this random procedure. The first is that there are run-time concerns among some of the anomaly detection algorithms we are going to use, and for this reason we want to institute a maximum benchmark size. For this study we chose 6,000 as this maximum size but this would generally be left as a design choice for any future experiments. In Eq. (4.1) this maximum size is indicated as parameter Ω.

We also want to ensure diversity among benchmarks created at the same specifications. For parentsets smaller than our maximum size, we would end up choosing the same nominal points nearly every time if we simply maximized the size of a benchmark. Because of this, we also want to institute a maximum number of nominal points selected to be some fraction of the total available. For this study we set this rate at 0.9 but this is indicated as parameter λ in Eq. (4.1).

So, for given hyperparameters (Ω, λ) and $(\mathbf{y}, \rho)$ from the parentset and benchmark specification, we determine the maximum number of nominals c_n and anomalies c_a to be selected as:

$$c_n = \lfloor \min(\Omega(1-\rho), \lambda(|\{y_i | y_i = \text{nominal}\}|)) \rfloor \tag{4.1}$$

$$c_a = \lceil \frac{\rho c_n}{(1-\rho)} \rceil$$

c_n and c_a act as a budget during benchmark creation that enforce relative frequency while allowing the algorithm to sample from either class until the budgets are used up. If relative frequency is not being enforced, then $c_n + c_a$ still acts as the total budget of the benchmark.

4.2 Determining Feasibility

For each individually sampled point we must recompute the set of points that are feasible to add to our benchmark set β. As the points are drawn from parentset $\mathbf{X}$ and feasibility is determined by specification parameters $(\bar{\boldsymbol{\phi}}, \nu, \alpha, \rho)$, with some abuse of notation we indicate the set of feasible points as $\mathbf{X}_{(\bar{\boldsymbol{\phi}},\nu,\alpha,\rho)} \subseteq \mathbf{X} \setminus \beta$ to mean the feasible set of points that can be added to β at a given sampling iteration. We describe here how feasibility is enforced for each problem dimension.

For relative frequency, the budgets c_n and c_a are determined as in Eq. (4.1). As long as $|\beta_n| < c_n$ then nominal points remain feasible, or more precisely, when $|\beta_n| \geq c_n$ all nominal points becomes infeasible. Similarly, when $|\beta_a| \geq c_a$ then anomaly points become infeasible. If the specification is to use the relative frequency control group, then class label does not impact feasibility.

For point difficulty we define value $\bar{\boldsymbol{\phi}}_{(\beta,x_i)}$ to be the measure of $\bar{\boldsymbol{\phi}}$ after point x_i is added to β. Any values of $\bar{\boldsymbol{\phi}}_{(\beta,x_i)}$ that are outside our specified range of $\bar{\boldsymbol{\phi}}$ render the corresponding x_i infeasible for selection. Setting the feasibility range $\bar{\boldsymbol{\phi}}$ to the entire interval $(0, 1)$ implements the point difficulty control group.

For normalized clusteredness we similarly define $\nu_{(\beta,x_i)}$ be the measure of ν after point x_i is added to β. If the specification wants anomalies to be more clustered than nominals, then values of $\nu_{(\beta,x_i)}$ below 0 render point x_i infeasible for selection. Likewise, If the specification wants anomalies to be more scattered than nominals, $\nu_{(\beta,x_i)}$ above 0 renders points x_i infeasible for selection.

Irrelevant features are added at the end of the process and do not effect feasibility.

For each point in $\mathbf{X} \setminus \beta$, the above values are computed and used to construct feasibility set $\mathbf{X}_{(\bar{\phi},\nu,\alpha,\rho)}$ from which the next point is sampled.

4.3 Determining Utility Scores for Feasible Points

It might suffice to simply sample uniformly at random from $\mathbf{X}_{(\bar{\phi},\nu,\alpha,\rho)}$. However, we want our final benchmarks to exhibit a variety of normalized clusteredness levels and we could not determine an efficient way to enforce this while maintaining consideration of the other specifications. Instead, we simply chose to have our sampling process favor more extreme values of normalized clusteredness (either minimal or maximal, depending on the specifications).

However, minimizing or maximizing clusteredness can be at odds with selecting point difficulties in so much as clustered anomalies or clustered nominals will likely have similar point difficulty scores and conversely, restraining point difficulty scores to a tight range might hinder selecting scattered points of either class. To help mitigate this relationship, we also want want to aim for a diversity of point difficulty

scores.

To do this we assign utility scores $\boldsymbol{\xi}$ to all feasible points and then define a Boltzmann (or Softmax) distribution with these scores and use this to sample from $\mathbf{X}_{(\bar{\boldsymbol{\phi}},\nu,\alpha,\rho)}$. For this we use the metrics $\bar{\boldsymbol{\phi}}_{(\beta,x_i)}$ and $\nu_{(\beta,x_i)}$ just described in Section 4.2.

For both of these metrics we assign a utility score intended to push them toward more extreme values. Keep in mind that feasibility constraints will already have been applied before scoring, so these extremes will always be in the "correct" direction and, as explained, will often be negatively correlated with each other. We select utility score ξ_{x_i} from four options depending on our benchmark specification. Respectively, if point difficulty is being considered, normalized clusteredness is being considered, both are being considered, or neither are being considered, then ξ_{x_i} is selected from:

$$\xi_{(\bar{\boldsymbol{\phi}},x_i)} = \left| \log\left(\frac{\bar{\boldsymbol{\phi}}_{(\beta,x_i)}}{\text{median}(\boldsymbol{\phi})} \right) \right| \tag{4.2}$$

$$\xi_{(\nu,x_i)} = |\nu_{(\beta,x_i)}|$$

$$\xi_{(\nu,\bar{\boldsymbol{\phi}},x_i)} = \xi_{(\nu,x_i)} + \xi_{(\bar{\boldsymbol{\phi}},x_i)}$$

$$\xi_{(\emptyset,x_i)} = 1$$

Note that in the simple case of $\xi_{x_i} = \xi_{(\emptyset,x_i)} = 1$ the resulting Boltzmann distribution will be equivalent to sampling uniformly at random. While the Boltzmann distribution is well known, for clarity we describe it here. For the set of all feasible

points $\mathbf{X}_{(\bar{\phi},\nu,\alpha,\rho)}$ and utility scores $\boldsymbol{\xi}$, point x_i is selected with probability $p_{\boldsymbol{\xi}}(x_i)$ as defined:

$$p_{\boldsymbol{\xi}}(x_i) = \frac{\exp(\xi_{x_i})}{\sum_j \exp(\xi_{x_j})}$$

4.4 Adding Irrelevant Features

To simplify the process of determining how many irrelevant features are needed, we compute an estimate of how many extra features will achieve the desired α. Note that the expected distance between two vectors whose coordinates are drawn at random (e.g., from the unit interval or from a standard normal Gaussian) grows in proportion to $\sqrt{d}$, where d is the dimensionality of the data. Hence, if a dataset already has d dimensions and we estimate d', the number of dimensions needed to increase the average pairwise distance by a factor of α, then we need $(d' - d)$ irrelevant features and define d':

$$\hat{d'} = (\alpha\sqrt{d})^2 \tag{4.3}$$

To generate a new irrelevant feature, we select a feature from the original parentset (uniformly at random), add it as a new feature, and randomly permute its values. This ensures that it provides no information about whether a point is a candidate nominal or a candidate anomaly but it also does not require us to inject purely synthetic data into our benchmarks. These features compose benchmark data $\beta_{(d'-d)}$ and β is augmented with them to produce the final benchmark.

4.5 Generating Benchmark Datasets

Finally, we present benchmark generation algorithm Algorithm 4.1:

Algorithm 4.1 Generating Experimental Benchmarks

1: **procedure** GENERATEBENCHMARK($(\mathbf{X}, \mathbf{y}, \boldsymbol{\phi}), (\bar{\boldsymbol{\phi}}, \nu, \alpha, \rho),$)
2: **initialize** c_n, c_a ▷ As per Eq. (4.1)
3: $\beta_n \leftarrow \emptyset, \beta_a \leftarrow \emptyset$ ▷ Note: $\beta = \beta_n \cup \beta_a$
4: **while** $|\beta_n| + |\beta_a| < c_n + c_a$ **do**
5: **determine** $\bar{\boldsymbol{\phi}}_{(\beta, x_i)}$ ▷ As in Section 4.2
6: **determine** $\nu_{(\beta, x_i)}$
7: **determine** $\mathbf{X}_{(\bar{\boldsymbol{\phi}}, \nu, \alpha, \rho)}$
8: **if** $\mathbf{X}_{(\bar{\boldsymbol{\phi}}, \nu, \alpha, \rho)} = \emptyset$ **then** ▷ If we ran out of feasible points.
9: **return failure** ▷ Because ρ has not been enforced.
10: **end if**
11: **for all** $x_i \in \mathbf{X}_{(\bar{\boldsymbol{\phi}}, \nu, \alpha, \rho)}$ **do**
12: **determine** ξ_{x_i} ▷ As in Section 4.3
13: **end for**
14: **sample** i **from** Boltzmann($\boldsymbol{\xi}$)
15: **if** $y_i =$ nominal **then**
16: $\beta_n = \beta_n \cup \{x_i\}$
17: **else**
18: $\beta_a = \beta_a \cup \{x_i\}$
19: **end if**
20: **end while**
21: **generate** $\beta_{(d'-d)}$ ▷ As in Section 4.4
22: $\beta \leftarrow \left(\beta | \beta_{(d'-d)}\right)$
23: **return** β
24: **end procedure**

It should be noted that even the above is slightly simplified. Notably, at line 9 of Algorithm 4.1 we do not instantly return failure, but rather, we reverse our process, removing points from β following feasibility constraints until ρ is satisfied and then return this smaller benchmark. If this removal process also fails, then

the generation process finally quits.

4.6 Code, Datasets, and Replication

Ultimately, our process generated a corpus of 25,685 benchmarks.

The software for producing our corpus of benchmarks and the generated benchmark datasets are available at

`http://ir.library.oregonstate.edu/xmlui/handle/1957/59114`

Chapter 5: Anomaly Detection Algorithms

Hundreds of anomaly detection algorithms have been published. We have grouped them into four major approaches for the purposes of this study. We chose two of the leading representatives from each approach.

We have also chosen two trivial algorithms to include in the study, bringing the total number of algorithms to 10. We have several reasons for doing so. First, we include them simply as a baseline point of comparison. Given the size of our corpus, we expect a large variance in results and a straightforward summary of metrics might be misleading for their simplicity, but more rigorous analysis might be hard to interpret. Having simple algorithms as a point of comparison will help make sense of broad results. Related is the fact that the regression models that we will use to analyze results do well to have well-defined points of reference.

An even greater concern of ours is that some benchmarks might be trivially easy. It is often the case in unsupervised anomaly detection literature that reported results are highly accurate, such as in [35, 4, 25, 40, 2]. To make a particular example, consider the work presented in [4]; the authors share the parameterization of each algorithm on each benchmark and praise the algorithm **iNNe** for sometimes performing well with parameter $\psi = 2$. However, an understanding of the iNNe algorithm will reveal that at that particular parameter setting the algorithm is doing little more than approximating the distance of each point from the mean of

the data. While we acknowledge that this particular work is a smaller workshop publication, it should serve as a warning that benchmarks for which all algorithms perform well might be benchmarks that can be trivially solved.

It is usually taken for granted that benchmarks in the literature created by means similar to our are reasonably challenging. We seek to scrutinize this assumption - and account for the inevitability of trivially easy benchmarks appearing in our corpus - by including trivially simple algorithms. They are described below where appropriate.

Below are only short summaries of the approaches and algorithms. Full implementation and parameterization details of each algorithm are given in Appendix E.

5.1 Density-Based Approaches

These methods seek to estimate $P(X)$, the density of the nominal data points. They do this by fitting a density estimator to the given data. The anomaly score assigned to a point x is defined as the negative logarithm of the estimated density: $-\log P(x)$ or something proportional to it.

A potential drawback of these approaches is they make a strong assumption that the data has high "feature precision" as described in Section 3.5.3

5.1.1 Trivial Distance From Mean (tmd)

For clarity, this trivial algorithm simply computes the location of the arithmetic mean of the data in feature space and then assigns anomaly scores to each point in the data equal to their euclidean distance from this mean point. We include this method as a density estimation technique because this distance from the mean is a monotonic transformation of a probability density model that uses a single Gaussian radial basis distribution with the mean of the data as the basis parameter.

5.1.2 Ensemble Gaussian Mixture Model (egmm)

A standard approach to density estimation is to fit a Gaussian mixture model (GMM) using the EM algorithm. To fit a GMM, we must select the value of k, the number of mixture components, and we must initialize the EM local search procedure. To improve the robustness of the method, we fit an ensemble of GMMs across several values of k. The final anomaly score for point x is the average negative log probability density $\frac{1}{L}\sum_{\ell=1}^{L} -\log P_\ell(x)$, where L is the number of fitted GMMs and $P_\ell(x)$ is the density assigned by GMM ℓ to data point x.

5.1.3 Robust Kernel Density Estimation (rkde)

When the nominal data are contaminated with outliers, standard density estimation may fail if the density estimator fits the outliers as well as the nominal points. Kim, et al., [24] developed a kernel density estimation method that incorporates

loss functions from robust statistics to reduce the risk of fitting the outliers. RKDE fits a single probability distribution over the benchmark data and the final anomaly score for point x is the negative log probability density $-\log P(x)$ assigned by the model to data point x.

5.2 Precentile Methods

The second family of algorithms compute a decision boundary that includes fraction $1-\delta$ of the data. This is the δ quantile of the distribution. In high dimension, there are many boundaries that achieve this. These algorithms seek to find a boundary that is smooth and has a simple description. The anomaly score assigned to a point x is defined by some measure proportional to it's distance inside or outside of this decision boundary.

5.2.1 One-Class SVM (ocsvm)

The One-Class SVM algorithm (Scholkopf et al. [42]) solves an optimization problem to find a kernel-space decision boundary that separates fraction $1-\delta$ of the data from the kernel-space origin. The outlier scores produced by this algorithm are determined by the residual after each point x is projected onto the decision surface. Points outside the decision boundary have positive residuals, where interior points have negative residuals.

5.2.2 Support Vector Data Description (svdd)

The SVDD algorithm (Tax and Duin [44]) is very similar to the One-Class SVM. It seeks to find the smallest hypersphere in kernel space that encloses $1 - \delta$ of the data. As with OCSVM, the outlier scores are determined by the residual after each point is projected onto the decision surface.

5.3 Nearest Neighbor Approaches

Perhaps the oldest and simplest approach to anomaly detection is to measure the distance from a given point x to its nearest neighbor (or its k nearest neighbors). If this distance is large, then x is more likely to be an outlier. Many variants of nearest neighbor methods have been developed (see [9]). In our study, we evaluated two of the best methods.

A drawback of nearest neighbor methods is that they must compute the k nearest neighbors of each data point. A naive implementation requires $O(N^2d^2)$ time, where N is the number of data points and d is the dimensionality of the feature space. Fortunately, there are many data structures that support fast nearest neighbor search, so this cost can be greatly reduced.

A second potential drawback is that these methods all depend on a distance metric. If the data are badly scaled or contain redundant and irrelevant features, then the distance metric may be highly biased or meaningless.

5.3.1 Trivial Distance to k-th Nearest Neighbor (knn)

Our second trivial algorithm is the simplest approximation of all algorithms in this paradigm. It takes as a parameter k and, for each point in the data, assigns it an anomaly score equal to its distance from its k-th nearest neighbor.

5.3.2 Local Outlier Factor (lof)

The popular Local Outlier Factor algorithm (Breunig, et al. [7]) computes the outlier score of a point x by computing its average distance to its k nearest neighbors. It normalizes this distance by computing the average distance of each of those neighbors to *their* k nearest neighbors. Consequently, it assigns a high anomaly score to a point if it is significantly farther from its neighbors than they are from each other.

5.3.3 KNN Angle-based Outlier Detection (abod)

Angle-Based Outlier Detection, as proposed by Kriegel, et al.[26] works as follows. For each point x_i, consider all pairs of other points $(x_j, x_k) \in X, i \neq j \neq k$ and compute the angle between them as "viewed from" x_i. The sample variance of these angles determines the outlier score of x_i. If the variance is low, then x_i is likely to be an outlier, whereas if the variance is high, then x_i is likely to be in the middle of other data points. The outlier score is simply the negation of this sample variance.

5.4 Projection-Based Approaches

This family of algorithms employs low-dimensional projections of the data to assign outlier scores.

5.4.1 Lightweight Online Detector Of Anomalies (loda)

LODA (Pevny [35]) generates an ensemble of random projections from the d-dimensional data space to the real line. Each projection is constructed by choosing $\sqrt{d}$ dimensions uniformly at random (without replacement), and assigning them weights drawn from a standard normal distribution. In each of the projections, LODA computes a histogram density estimator. The anomaly score assigned to a point x is the average of the estimated negative log-likelihood of x in each of these 1-dimensional histograms.

5.4.2 Isolation Forest (iforest)

The Isolation Forest algorithm (Liu, et al. [29]) constructs an ensemble of isolation trees. Each isolation tree is constructed top-down by randomly choosing a feature j, computing the observed range $[L, U]$ of the feature values, and then selecting a splitting threshold θ uniformly at random from this interval. The data are then split according to whether $x_j <= \theta$ or $x_j > \theta$ to create two child nodes. The process continues until every data point is isolated in its own leaf. The *isolation depth* of x in an isolation tree is equal to the number of random splits required to

isolate x. The anomaly score for x is computed from the average isolation depth of x across all of the trees in the ensemble.

The intuition behind the isolation forest is that a data point is an outlier if it can be separated from the rest of the data points using only a small number of random splits. We view this as a projection method, because each splitting decision is made by projecting the data onto one of the coordinate axes.

Chapter 6: Evaluating the Benchmark Corpus

In this chapter we will evaluate our benchmark construction methodology agnostic to any comparison of algorithms. Examining all 25,685 benchmarks with 10 algorithms yields 256,850 micro-experiment results and reporting them all here is not feasible. An additional concern is that because our benchmark construction process is designed to be exhaustive and test the limits of algorithm performance it is reasonable to assume that many benchmarks are not very reasonable articulations of the unsupervised anomaly detection problem. We will use our micro-experiment results to validate the usefulness of each benchmark.

6.1 Validating Benchmark Construction With Hypothesis Testing

The process described in Chapters 3 and 4 can generate very difficult benchmark datasets. In reanalyzing our 2013 data, we discovered that these "impossible" benchmarks were biasing the conclusions. Hence, we decided to filter out datasets where our battery of algorithms do no better than random guessing.

Recall from Section 2.2 that we are considering both Area Under the Receiver Operating Characteristic Curve (AUC) and Mean Average Precision (AP). These figures of merit are both derived from the relative ranking of the entire dataset and thus the behavior of a random algorithm can be simulated by assigning anomaly

scores according to a random permutation of $\{1, \ldots, n\}$.

For each benchmark in the corpus we conducted the following hypothesis test:

H_0 : There does not exist an anomaly detection algorithm that produces a ranking better than a random one.

H_1 : There exists an anomaly detection algorithm that produces a ranking better than a random one.

Even though the benchmark corpus was intentionally designed to push beyond the boundaries of reasonable anomaly detection problems but we don't want to make assumptions about where those boundaries are. The intuition behind the hypothesis test is that, in so much as anomaly detection algorithms are correlated because they are trying to solve the same problem, if the algorithms universally fail to differentiate themselves from random behavior on a particular benchmark then we have reason to believe the benchmark does not represent a well-formed anomaly detection problem, or at the very least, comparing algorithm performance on that benchmark does not provide useful information.

To conduct our hypothesis test on a benchmark we treat AUC and AP as random variables and compute the quantiles of interest of their distributions. Details on how this can be done are provided in Appendix F.1. We compare this score to the score of the best performing algorithm on the benchmark. We accept a benchmark for further analysis if we can reject the above null hypothesis with high probability.

Table 6.1: Benchmark Acceptance Rate by Metric and p-value

$P(\mathbf{H}_0)$	**AUC**	**AP**	**Both**
$p < 0.05$	0.7501	0.7105	0.6716
$p < 0.01$	0.6653	0.5778	0.5465
$p < 0.001$	0.5862	0.4881	**0.4628**

The rate of rejecting the null hypothesis - or acceptance rate - of our construction methodology across the entire corpus is summarized in Table 6.1

The appropriate significance level for this study is debatable. Smaller p-values trade away potential evidence (by accepting fewer benchmarks) for greater confidence that the results from the benchmarks under consideration are relevant. Ultimately we choose to apply the most stringent threshold of $p < 0.001$ and we only accept benchmarks for further analysis if we can reject the null hypothesis for **both** metrics; even though the acceptance rate by these standards is rather low (46.28%) they still leave many benchmarks across all factors of interest (11,888 total).

We make note here that rejecting a benchmark for further analysis in this study does not equate to discarding it from the corpus. If a new algorithm were introduced to the corpus it may give a performance that allows for accepting more benchmarks into future analysis.

We also present here the benchmark acceptance rates across our benchmark specifications in Tables 6.2 to 6.5. Boldface values indicate an acceptance rate

Table 6.2: Benchmark Acceptance Rate by Metric and Point Difficulty Specification

pd-level	**AUC**	**AP**	**Both**
$\text{pd}_0 : \bar{\phi} \in (0, 1)$	**0.7090**	**0.5996**	**0.5723**
$\text{pd}_1 : \bar{\phi} \in (0, 0.1\overline{6})$	**0.7182**	**0.5950**	**0.5774**
$\text{pd}_2 : \bar{\phi} \in [0.1\overline{6}, 0.\overline{3})$	0.5785	0.4771	0.4523
$\text{pd}_3 : \bar{\phi} \in [0.\overline{3}, 0.5)$	0.4323	0.3511	0.3234
$\text{pd}_4 : \bar{\phi} \in [0.5, 1)$	0.2266	0.1918	0.1569

greater than the global average for that metric. While the purpose of these hypothesis tests is to ensure the validity of our analysis, the general trends in these acceptance rates across our benchmark specification levels provide some evidence that confirms some of our intuitions about the impact of these specifications. In general, as point difficulty, normalized clusteredness and feature irrelevance increase, acceptance rates go down, indicating that they all contribute to more difficult problems. The exception to this trend is that as relative frequency increases, acceptance rates go up as well. One explanation for this is that the 0.999 quantile for both random AUC and random AP fall drastically as the relative frequency of anomalies increases. We note that the control setting of rf_0 will, on average, produce benchmarks with $\rho = 0.5$ and that the acceptance rate at this level is lower than the one at rf_5. This suggests that at some value of ρ the trend reverses as the classes become balanced and the outliers-as-anomalies assumption breaks down.

For compactness of presentation we present benchmark acceptance rates by

Table 6.3: Benchmark Acceptance Rate by Metric and Normalized Clusteredness Specification

nc-level	**AUC**	**AP**	**Both**
$\text{nc}_0 : \nu \in \mathbb{R}$	0.5503	0.4047	0.3953
$\text{nc}_1 : \nu < 0$	**0.6321**	**0.5846**	**0.5379**
$\text{nc}_2 : \nu > 0$	0.5773	0.4772	0.4571

Table 6.4: Benchmark Acceptance Rate by Metric and Feature Irrelelvance Specification

fi-level	**AUC**	**AP**	**Both**
$\text{fi}_0 : \alpha = 1.0$	**0.6842**	**0.5908**	**0.5680**
$\text{fi}_1 : \hat{\alpha} = 1.2$	**0.6190**	**0.5208**	**0.4972**
$\text{fi}_2 : \hat{\alpha} = 1.5$	0.5641	0.4632	0.4356
$\text{fi}_3 : \hat{\alpha} = 2.0$	0.4778	0.3778	0.3507

Table 6.5: Benchmark Acceptance Rate by Metric and Relative Frequency Specification

rf-level	**AUC**	**AP**	**Both**
$\text{rf}_0 : \rho \in (0, 1)$	**0.6401**	**0.5671**	**0.5583**
$\text{rf}_1 : \rho = 0.001$	0.2750	0.1536	0.1008
$\text{rf}_2 : \rho = 0.005$	0.4580	0.3078	0.2680
$\text{rf}_3 : \rho = 0.01$	0.5615	0.4171	0.3773
$\text{rf}_4 : \rho = 0.05$	**0.7631**	**0.6926**	**0.6805**
$\text{rf}_5 : \rho = 0.1$	**0.7473**	**0.6997**	**0.6899**

parentset and by algorithm in Appendix F.2. Here we only make note of one finding in Table F.1 which is that the *yeast* parentset has an extremely low acceptance rate (below 3%in all cases).

6.2 Quantifying Impact of Benchmark Specifications

The remainder of this study only performs analyses on results from the 11,888 benchmarks accepted by hypothesis testing. We will incrementally introduce linear regression models for the purpose of examining factors that impact experimental outcomes. Before we can do this, we need to ensure that all variables are appropriate for such regression analysis.

6.2.1 Transformation of Metrics

Using linear models to predict metrics like AUC and AP is problematic because they are both constrained to the range $[0, 1]$. Further, AP does not have a constant expectation. For both metrics we need a function that maps $f : [0, 1] \mapsto \mathbb{R}$

For AUC this is relatively simple. Regardless of relative frequency of the benchmark, AUC has a random expectation of 0.5 and, as explained in Section 2.2, it can be interpreted as parameter p of a Bernoulli distribution predicting correct ranking of two points from opposite classes. A sensible transformation of this value is the logit transform:

$$\text{logit}(\text{AUC}) = \log\left(\frac{\text{AUC}}{1-\text{AUC}}\right) \tag{6.1}$$

For some intuition, the logit function is the inverse of the sigmoid function. Random expectation (0.5) maps to 0, scores below expectation map to negative values, and scores above expectation map to positive values.

We would like a function that exhibits the same properties for AP. Because AP does not have a constant expectation, one way to normalize AP is to compute the *lift* which is the ratio of AP to its expectation. It is commonly assumed that this expectation is equivalent to the relative frequency of the anomalies in the benchmark, but Bestgen [5] shows that, while this can be a good approximation, it is not exactly correct. More importantly it can be a bad approximation when relative frequency is low, which is the case for most of our benchmarks. We compute lift using the exact expectation. To map this ratio to all real numbers we simply take the log:

$$\log(\text{lift}) = \log\left(\frac{\text{AP}}{E[\text{AP}]}\right) \tag{6.2}$$

6.2.2 Algorithm Agnostic Regression

Our first regression models will, for each transformed metric, predict the outcome given the benchmark specifications while remaining blind to which algorithm produced the score. For this model we use our problem dimension specification levels as explained in Chapter 3 (pd,nc,fi,rf) and include the parentset that originated

the benchmark. In the notation of R, we write this as

$$\text{metric} \sim 1 + \text{parentset} + \text{pd} + \text{nc} + \text{fi} + \text{rf}. \tag{6.3}$$

Note that Eq. (6.3) is fit twice, once for logit(AUC) and once for $\log((lift))$. These two models are only using discrete variables and not the real valued measurements $\bar{\phi}, \nu, \alpha, \rho$ which we will use in later models. For now we are trying to assess the impact of our construction criterion and want to examine our construction specifications. More specifically, we want to explicitly separate our control group settings from the rest.

Here we summarize an analysis of variance (ANOVA) of each model and provide the $\hat{R}^2$ goodness-of-fit of each model. Table 6.6 shows the percentage of variance explained by each specification factor. The factors that influence outcomes the most (across all algorithms) are the choice of parentset and the relative frequency setting. Relative frequency explains more variance in the log(lift) model, while choice of parentset explains more variance in the logit(AUC) model.

Variance explained suggests the importance of each factor, but it does not articulate how each specification impacts the experimental outcomes. For each benchmark specification criterion we detail the linear model coefficients of each setting; these coefficients are a measure of how much the experimental outcome changes relative to the control group (which by definition will always have a coefficient of zero). Tables 6.7 to 6.10 show these coefficients and the range of a confidence interval for each of our problem dimension specifications. The minimum

Table 6.6: Metric Variance Explained By Specification Factors in Algorithm-Agnostic Models

specification	logit(AUC)	log(lift)
parentset	**37.75%**	26.78%
pd-level	4.28%	4.58%
nc-level	0.46%	2.01%
fi-level	4.34%	3.26%
rf-level	11.22%	**32.92%**
Residual σ^2	41.95%	30.45%
$\hat{R}^2$ of Model	**0.5805**	**0.6955**

and maximum coefficients for each problem dimension level are in bold (excepting the control group).

Keep in mind that the nature of the control groups at each setting are not necessarily the "lowest" setting. The random expectation of the relative frequency control group is 0.5; much higher than our tested levels. The random expectation of the normalized clusteredness control group is zero (in between our other two settings). The random expectation of the point difficulty control group cannot be easily quantified and is dependent on the parentset and the oracle used. For some intuition though, a good oracle will typically have that vast majority of point difficulty scores below 0.5, and if those scores are evenly distributed they would have an expectation of 0.25.

We can see the general trend in each problem dimension matches our intuition.

Table 6.7: Point Difficulty Level Coefficients Estimating Metrics in Algorithm-Agnostic Models

pd-level	Δlogit(AUC)	$\text{CI}_{0.999}$	Δ log(lift)	$\text{CI}_{0.999}$
$\text{pd}_0 : \bar{\phi} \in (0, 1)$	0.0000		0.0000	
$\text{pd}_1 : \bar{\phi} \in (0, 0.1\overline{6})$	**0.1214**	$\pm$0.0204	**0.0079**	$\pm$0.0150
$\text{pd}_2 : \bar{\phi} \in [0.1\overline{6}, 0.\overline{3})$	-0.3682	$\pm$0.0254	-0.2697	$\pm$0.0187
$\text{pd}_3 : \bar{\phi} \in [0.\overline{3}, 0.5)$	**-0.5261**	$\pm$0.0312	**-0.4006**	$\pm$0.0230
$\text{pd}_4 : \bar{\phi} \in [0.5, 1)$	-0.3610	$\pm$0.0464	-0.3013	$\pm$0.0342
σ^2 Explained	4.28%		4.58%	
$\hat{R}^2$ of Model	**0.5805**		**0.6955**	

With one notable exception, general algorithm performance degrades as point difficulty, normalized clusteredness, feature irrelevance and relative frequency increase. The exception to the trend, in Table 6.7, is that the highest point difficulty setting does not have as negative an impact as the second-highest. This might be explained by the fact that the highest setting uses points with score above 0.5, which means all points, nominals and anomalies alike, are on the wrong side of a decision boundary and can therefore be differentiated from each other again.

For compactness of presentation, we present coefficients related to the parentsets in Appendix F.2. We note here that all of the parentsets have a large negative impact on experimental outcomes relative to benchmarks created from the synthetic control parentset; evidence that more idiosyncratic real world data produces more challenging benchmarks.

Table 6.8: Normalized Clusteredness Level Coefficients Estimating Metrics in Algorithm-Agnostic Models

nc-level	Δlogit(AUC)	$\text{CI}_{0.999}$	Δ log(lift)	$\text{CI}_{0.999}$
$\text{nc}_0 : \nu \in \mathbb{R}$	0.0000		0.0000	
$\text{nc}_1 : \nu < 0$	**0.0803**	$\pm$0.0199	**0.1517**	$\pm$0.0147
$\text{nc}_2 : \nu > 0$	**-0.0653**	$\pm$0.0213	**-0.1197**	$\pm$0.0157
σ^2 **Explained**	0.46%		2.01%	
$\hat{R}^2$ **of Model**	**0.5805**		**0.6955**	

Table 6.9: Feature Irrelelvance Level Coefficients Estimating Metrics in Algorithm-Agnostic Models

fi-level	Δlogit(AUC)	$\text{CI}_{0.999}$	Δ log(lift)	$\text{CI}_{0.999}$
$\text{fi}_0 : \alpha = 1.0$	0.0000		0.0000	
$\text{fi}_1 : \hat{\alpha} = 1.2$	**-0.2092**	$\pm$0.0221	**-0.1119**	$\pm$0.0163
$\text{fi}_2 : \hat{\alpha} = 1.5$	-0.4188	$\pm$0.0231	-0.2172	$\pm$0.0170
$\text{fi}_3 : \hat{\alpha} = 2.0$	**-0.6122**	$\pm$0.0248	**-0.3439**	$\pm$0.0183
σ^2 **Explained**	4.34%		3.26%	
$\hat{R}^2$ **of Model**	**0.5805**		**0.6955**	

Table 6.10: Relative Frequency Level Coefficients Estimating Metrics in Algorithm-Agnostic Models

rf-level	Δlogit(AUC)	$\text{CI}_{0.999}$	Δ log(lift)	$\text{CI}_{0.999}$
$\text{rf}_0 : \rho \in (0, 1)$	0.0000		0.0000	
$\text{rf}_1 : \rho = 0.001$	**2.2254**	$\pm$0.0517	**3.2253**	$\pm$0.0381
$\text{rf}_2 : \rho = 0.005$	1.1854	$\pm$0.0331	1.7529	$\pm$0.0244
$\text{rf}_3 : \rho = 0.01$	0.8639	$\pm$0.0295	1.2800	$\pm$0.0218
$\text{rf}_4 : \rho = 0.05$	0.4159	$\pm$0.0242	0.5553	$\pm$0.0178
$\text{rf}_5 : \rho = 0.1$	**0.2908**	$\pm$0.0234	**0.3503**	$\pm$0.0173
σ^2 **Explained**	11.22%		32.92%	
$\hat{R}^2$ **of Model**	**0.5805**		**0.6955**	

Chapter 7: Analyzing Algorithm Performance

The $\hat{R}^2$ goodness-of-fit measures for each the models presented in Section 6.2 are 0.5805 and 0.6955 respectively. These are reasonably good measures, especially since the models were blind to which algorithm produced each result, but they still leave a lot of room for improvement.

7.1 Finding the Best First-Order Model

To guide further analysis, we produced four new regression models. First, for each metric, we computed a linear model as in Eq. (6.3) but included choice of algorithm as a factor.

$$\text{metric} \sim 1 + \text{algorithm} + \text{parentset} + \text{pd} + \text{nc} + \text{fi} + \text{rf} \tag{7.1}$$

We also wish to use the continuous measures of our problem dimensions to predict our metrics to see if they produce better models. As in Section 6.2 we want to make sure our variables are appropriate for linear modeling.

7.1.1 Transformation of Problem Dimension Measures

Recall that each benchmark has associated real-valued problem dimension measures $\bar{\phi}, \nu, \alpha, \rho$. We describe here our transformation for each.

- $\bar{\phi}$: Point difficulty scores are restrained to the interval $(0, 1)$ and can be interpreted as a probability of misclassification. Because of this, the **logit** function is an appropriate transformation.

- ν : Normalized Clusteredness is formulated as the log of a ratio and so does not require any transformation. It has a random expectation of 0 with positive values indicating anomalies more clustered than nominal and negative scores indicating anomalies more scattered than nominals.

- α : Feature irrelevance is measured as a ratio, so taking its logarithm is an appropriate transformation. Note that because a ratio of 1 is the lowest value allowed in our benchmark construction methods, this transformed measure will always be non-negative.

- ρ : As with point difficulty scores, relative frequency is restrained to the interval $(0, 1)$ and can be interpreted as a probability that a point selected uniformly at random is an anomaly. Because of this, the **logit** function is an appropriate transformation.

This yields our second pair of models. For each metric, we computed the linear model specified in Eq. (7.2)

Table 7.1: $\hat{R}^2$ of First-Order Models by Metric and Variable Type

Variable Type	logit(AUC)	log(lift)
Discrete Variables	0.5928	0.6955
Continuous Variables	0.6075	**0.7145**

$$\text{metric} \sim 1 + \text{algorithm} + \text{parentset} + \text{logit}(\bar{\phi}) + \nu + \log(\alpha) + \text{logit}(\rho) \qquad (7.2)$$

We compare the $\hat{R}^2$ measure for these four models in Table 7.1. The models that explain the most variance in experimental outcomes are the models fitted to our transformed continuous problem dimension measures. Tables 7.2 and 7.3 present model coefficients and ANOVA details of both of these first-order models. We do not share parentset coefficients as they are numerous and uninteresting, but we do share the amount of variance they explain as a group. The algorithm coefficients are relative to the **trivial mean distance (tmd)** algorithm as the control setting. We consider this a good reference algorithm because it accurately models the true probability density of the nominal class in our synthetic control parentset.

The algorithms with the greatest positive impact on outcomes have their coefficients in bold. Algorithms that fail to distinguish themselves from **tmd** have

their confidence intervals in bold. The coefficients for the benchmark specification measures are not directly comparable, but their polarity indicates how these measures are correlated with outcomes. The impact of each specification measure is better evaluated by comparing how much variance in outcomes it explains. The most influential problem dimension has its variance in bold.

Surprisingly, the trivial **knn** algorithm achieves the best overall logit(AUC) across the accepted benchmarks. In both metrics parentset of origin has the greatest impact on outcomes while choice of algorithm has very little.

Because both models generally confirm the same trends, a simple argument could be made to focus remaining analyses on the metric that receives the greater $\hat{R}^2$, but we observe a problematic interaction that has appeared in all models so far: A very large portion of the variance of log(lift) is explained by relative frequency. This is because the computation of *lift* and the computation of ρ both depend on knowledge of the number of anomalies and the number of nominals in the benchmark. For this reason, log(lift) and ρ are naturally correlated (negatively) and relative frequency has an oversized impact on analyzing AP-based results. This does not mean AP is a poor metric, but its sensitivity to relative frequency can make interpreting results of this study more difficult. For this reason we will report analyses of both metrics, but for compactness of presentation we will put the focus on logit(AUC) while relegating most presentation of analyses of log(lift) outcomes to Appendix G.

Table 7.2: First-Order Regression Model Predicting logit(AUC)

		Δlogit(AUC)	$\text{CI}_{0.999}$
parentset	$\sigma^2(\%)$	**37.75%**	
algorithm	$\sigma^2(\%)$	1.23%	
	tmd	0.0000	
	egmm	0.2632	±0.0360
	rkde	0.2458	±0.0360
	ocsvm	-0.0528	±0.0360
	svdd	0.0131	**±0.0360**
	knn	**0.4060**	±0.0360
	lof	0.2387	±0.0360
	abod	0.1862	±0.0360
	loda	0.1590	±0.0360
	iforest	0.3849	±0.0360
specifications	$\sigma^2(\%)$	21.77%	
pd ($\bar{\phi}$)	**8.77%**	**-0.4418**	±0.0131
nc (ν)	1.40%	-0.1145	±0.0079
fi (α)	3.82%	-0.4084	±0.0137
rf (ρ)	7.79%	-0.2484	±0.0053
Residual σ^2	39.25%		$F_{118847}^{32} \geq 5749$
$\hat{R}^2$ of Model	**0.6075**		$p \leq 0.001$

Table 7.3: First-Order Regression Model Predicting log(lift)

		Δ log(lift)	$\text{CI}_{0.999}$
parentset	$\sigma^2(\%)$	26.78%	
algorithm	$\sigma^2(\%)$	0.63%	
	tmd	0.0000	
	egmm	0.1451	$\pm$0.0265
	rkde	0.1445	$\pm$0.0265
	ocsvm	0.0968	$\pm$0.0265
	svdd	-0.0006	$\pm$**0.0265**
	knn	0.2846	$\pm$0.0265
	lof	0.1432	$\pm$0.0265
	abod	0.1477	$\pm$0.0265
	loda	0.1328	$\pm$0.0265
	iforest	**0.2929**	$\pm$0.0265
specifications	$\sigma^2(\%)$	**44.04%**	
pd ($\bar{\phi}$)	7.87%	-0.1966	$\pm$0.0097
nc (ν)	8.26%	-0.2341	$\pm$0.0058
fi (α)	2.06%	-0.2233	$\pm$0.0101
rf (ρ)	**25.85%**	**-0.3915**	$\pm$0.0039
Residual σ^2	28.55%		$F_{118847}^{32} \geq 9295$
$\hat{R}^2$ of Model	**0.7145**		$p \leq 0.001$

7.2 Second-Order Modeling

The $\hat{R}^2$ measure of 0.6075 on our best logit(AUC)-predicting model can be improved with second-order interactions. This would be especially true if there are strong interactions between our specification measures and individual algorithm performance. With some abuse of notation, Eq. (7.3) presents the formulation of our second-order model. (We do not actually include the squares of individual factors but include first-order terms and all pairwise terms.)

$$\text{metric} \sim \left(1 + \text{algorithm} + \text{parentset} + \text{logit}(\bar{\phi}) + \nu + \log(\alpha) + \text{logit}(\rho)\right)^2 \quad (7.3)$$

Table 7.4 presents the ANOVA for this second-order model, which has a much improved $\hat{R}^2$ goodness-of-fit of 0.8138. We decompose the variance explained first by grouping together all algorithm effects and interactions. Algorithm effects still only account for 6.37% of the variance in experimental outcomes. Within algorithm interactions, choice of parentset accounts for 63.68% of that variance (but only 4.06% overall).

Next we factor out all remaining parentset effects and interactions, which account for 51.79% of the variance. While first-order parentset effects are quite large (37.75%), a large amount of variance is explained by the interactions between parentset and problem specification measures (14.04%). We also show the variance explained by the interaction between parentset and point difficulty alone, because it is notably strong among interactions between parentset and problem specifica-

tions. At 7.83% this interaction alone explains more variance then all effects and interactions involving choice of algorithm and is almost as strong as point difficulty's first-order effect. One explanation for the strength of this interaction is that the assignment of point difficulty scores is dependent on a classifier run on the parentset itself. While the assignment of these scores is consistent enough that the factor has explanatory power on its own, it makes sense that the model would benefit from being able to adjust the effect of point difficulty for each individual parentset.

The remaining effects belong to first-order problem specification measures and the interactions between them. As a group they explain 23.22% of the variance in experimental outcomes, with point difficulty and relative frequency having the strongest influence. Notably, the variance explained by interactions between our specification measures is low; 6.23% of the variance is explained by this group and only 1.45% overall. We believe this is evidence that the problem specification measures are able to provide independent information about outcomes.

The analogous data pertaining to the log(lift) model is presented in Table G.1.

We call attention here to supplemental results offered in Table G.2 and Table G.3. These tables present coefficients for interactions between algorithms and choice of parentset in our second-order models. Positive interactions between algorithms and parentsets are in bold. We do not suggest that any of these coefficients are specifically interesting, but as a whole they do illustrate that the original source of data can impact algorithm performance in unpredictable ways. We examine this phenomenon in greater detail in Section 7.5.

Table 7.4: ANOVA for Second-Order Regression Model Predicting logit(AUC)

	$\sigma^2(\%)$	**group**$(\%)$	F-test
algorithm	6.37%		$p \leq 0.001$
–	1.23%	19.29%	$p \leq 0.001$
× parentset	4.06%	**63.68%**	$p \leq 0.001$
× specifications	1.08%	17.02%	$p \leq 0.001$
parentset	**51.79%**		$p \leq 0.001$
–	**37.75%**	**72.90%**	$p \leq 0.001$
× specifications	14.04%	27.10%	$p \leq 0.001$
× pd ($\bar{\phi}$)	7.83%	15.11%	$p \leq 0.001$
specifications	23.22%		$p \leq 0.001$
pd ($\bar{\phi}$)	8.77%	**37.77%**	$p \leq 0.001$
nc (ν)	1.40%	6.01%	$p \leq 0.001$
fi (α)	3.82%	16.47%	$p \leq 0.001$
rf (ρ)	7.79%	33.53%	$p \leq 0.001$
× specifications	1.45%	6.23%	$p \leq 0.001$
Residual σ^2	18.62%		$F_{118558}^{321} \geq 1614$
$\hat{R}^2$ of Model	**0.8138**		$p \leq 0.001$

We do not examine interactions between algorithms and our problem dimension measures here, because we are bringing even tighter focus on algorithm interactions in Section 7.3.

7.3 Third-Order Algorithm Models

To better evaluate the impact of our problem dimensions on individual algorithms, we examine third-order algorithm interactions with models separately predicting the behavior of each algorithm according to (Eq. (7.4)). The same abuse of notation as in Eq. (7.3) applies here.

$$\text{metric}_{(\text{algo})} \sim \left(1 + \text{parentset} + \text{logit}(\bar{\phi}) + \nu + \log(\alpha) + \text{logit}(\rho)\right)^2 \tag{7.4}$$

The performances of each algorithm are individually summarized in Tables 7.5 to 7.14. These tables present each algorithm's mean performance, ($\mu(\text{logit}(\text{AUC}))$), and the retransformed metric ($\mu'(\text{AUC})$) for some human-readable intuition about this performance; this is not the arithmetic mean AUC, but the sigmoid of $\mu(\text{logit}(\text{AUC}))$ which we feel is a more accurate accounting of the mean AUC anyway.

These tables also present the real-valued variance explained by each factor in each model as these values are comparable across algorithms. Model coefficients are not presented. The coefficients of interest would be those pertaining to our problem setting measures, but because they are *all* negative and the strength of

the interaction is better represented by variance explained, we do not show them here for compactness of presentation. Where first-order parentset effects are the strongest effect, this variance is in bold. The most influential problem dimension setting is also in bold.

The analogous data pertaining to the log(lift) models is presented in Tables G.4 to G.13.

We do not discuss the information in these tables in detail here, but leave them as an in-depth accounting of the different influences problem settings exert on different algorithms. A summary of the information in these tables (for both metrics) can be found in Table 7.15. In this table a direct comparison of algorithms can be more easily digested. The top 3 algorithms for each category are in bold. Algorithms with low unexplained variance might have more reliable performance agnostic to the problem setting. Low variance in the remaining categories might suggest a resilience to changes in the problem setting.

7.4 High-Contrast Setting Models

For a more explicitly quantified examination of how algorithm performance changes in different settings, we analyze experimental outcomes with four mixed effect models. This time each model brings focus on one axis of benchmark specification by constraining analysis to benchmarks in the most extreme settings of that problem dimension. The problem dimension under examination is treated as a discrete factor again as in our earliest models, and only two extreme levels of those factors

Table 7.5: ANOVA For Model Predicting **tmd** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
tmd	**0.7877**	**1.5468**	**0.6873**
	σ^2(%)	σ^2(real)	F-test
parentset	**45.09%**	**0.6974**	$p \leq 0.001$
pd ($\bar{\phi}$)	**15.38%**	**0.2380**	$p \leq 0.001$
–	5.99%	0.0927	$p \leq 0.001$
× parentset	9.39%	0.1453	$p \leq 0.001$
nc (ν)	4.47%	0.0691	$p \leq 0.001$
–	1.44%	0.0222	$p \leq 0.001$
× parentset	3.03%	0.0469	$p \leq 0.001$
fi (α)	11.27%	0.1744	$p \leq 0.001$
–	2.87%	0.0443	$p \leq 0.001$
× parentset	8.41%	0.1301	$p \leq 0.001$
rf (ρ)	7.13%	0.1103	$p \leq 0.001$
–	4.72%	0.0730	$p \leq 0.001$
× parentset	2.41%	0.0373	$p \leq 0.001$
spec × spec	0.96%	0.0149	$p \leq 0.001$
Residual σ^2	15.69%	0.2427	$F_{11782}^{105} \geq 602$
$\hat{R}^2$ of Model	**0.8431**	**1.3041**	$p \leq 0.001$

Table 7.6: ANOVA For Model Predicting **egmm** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
egmm	**1.0509**	**1.8023**	**0.7409**
	σ^2(%)	σ^2(real)	F-test
parentset	**42.15%**	**0.7597**	$p \leq 0.001$
pd ($\bar{\phi}$)	**17.94%**	**0.3234**	$p \leq 0.001$
–	10.62%	0.1913	$p \leq 0.001$
× parentset	7.32%	0.1320	$p \leq 0.001$
nc (ν)	3.68%	0.0663	$p \leq 0.001$
–	2.20%	0.0397	$p \leq 0.001$
× parentset	1.48%	0.0266	$p \leq 0.001$
fi (α)	10.36%	0.1867	$p \leq 0.001$
–	7.47%	0.1346	$p \leq 0.001$
× parentset	2.89%	0.0520	$p \leq 0.001$
rf (ρ)	9.09%	0.1638	$p \leq 0.001$
–	7.27%	0.1310	$p \leq 0.001$
× parentset	1.82%	0.0328	$p \leq 0.001$
spec × spec	1.59%	0.0287	$p \leq 0.001$
Residual σ^2	15.19%	0.2738	$F_{11782}^{105} \geq 626$
$\hat{R}^2$ **of Model**	**0.8481**	**1.5286**	$p \leq 0.001$

Table 7.7: ANOVA For Model Predicting **rkde** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
rkde	**1.0335**	**1.7034**	**0.7376**
	σ^2(%)	σ^2(real)	F-test
parentset	**37.62%**	**0.6409**	$p \leq 0.001$
pd ($\bar{\phi}$)	**16.94%**	**0.2886**	$p \leq 0.001$
–	7.72%	0.1315	$p \leq 0.001$
× parentset	9.22%	0.1571	$p \leq 0.001$
nc (ν)	2.26%	0.0384	$p \leq 0.001$
–	1.05%	0.0179	$p \leq 0.001$
× parentset	1.20%	0.0205	$p \leq 0.001$
fi (α)	14.04%	0.2391	$p \leq 0.001$
–	7.91%	0.1348	$p \leq 0.001$
× parentset	6.12%	0.1043	$p \leq 0.001$
rf (ρ)	9.82%	0.1672	$p \leq 0.001$
–	8.03%	0.1368	$p \leq 0.001$
× parentset	1.79%	0.0304	$p \leq 0.001$
spec × spec	1.72%	0.0293	$p \leq 0.001$
Residual σ^2	17.60%	0.2998	$F_{11782}^{105} \geq 525$
$\hat{R}^2$ **of Model**	**0.8240**	**1.4036**	$p \leq 0.001$

Table 7.8: ANOVA For Model Predicting **ocsvm** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
ocsvm	**0.7349**	**2.4944**	**0.6759**
	σ^2(%)	σ^2(real)	F-test
parentset	**53.62%**	**1.3374**	$p \leq 0.001$
pd ($\bar{\phi}$)	**14.56%**	**0.3633**	$p \leq 0.001$
–	7.05%	0.1760	$p \leq 0.001$
× parentset	7.51%	0.1873	$p \leq 0.001$
nc (ν)	5.94%	0.1482	$p \leq 0.001$
–	1.95%	0.0486	$p \leq 0.001$
× parentset	3.99%	0.0996	$p \leq 0.001$
fi (α)	2.29%	0.0570	$p \leq 0.001$
–	0.04%	0.0009	$p \leq 0.001$
× parentset	2.25%	0.0561	$p \leq 0.001$
rf (ρ)	10.67%	0.2662	$p \leq 0.001$
–	7.72%	0.1926	$p \leq 0.001$
× parentset	2.95%	0.0735	$p \leq 0.001$
spec × spec	0.68%	0.0169	$p \leq 0.001$
Residual σ^2	12.25%	0.3055	$F_{11782}^{105} \geq 803$
$\hat{R}^2$ **of Model**	**0.8775**	**2.1889**	$p \leq 0.001$

Table 7.9: ANOVA For Model Predicting **svdd** (logit(AUC))

	$\mu(\text{logit(AUC)})$	$\sigma^2(\text{real})$	$\mu'(\text{AUC})$
svdd	**0.8008**	**1.5296**	**0.6901**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	**44.31%**	**0.6778**	$p \leq 0.001$
pd ($\bar{\phi}$)	**15.75%**	**0.2408**	$p \leq 0.001$
–	6.37%	0.0974	$p \leq 0.001$
× parentset	9.38%	0.1434	$p \leq 0.001$
nc (ν)	4.28%	0.0655	$p \leq 0.001$
–	1.30%	0.0198	$p \leq 0.001$
× parentset	2.98%	0.0456	$p \leq 0.001$
fi (α)	11.45%	0.1751	$p \leq 0.001$
–	3.19%	0.0488	$p \leq 0.001$
× parentset	8.26%	0.1263	$p \leq 0.001$
rf (ρ)	7.26%	0.1110	$p \leq 0.001$
–	4.87%	0.0744	$p \leq 0.001$
× parentset	2.39%	0.0366	$p \leq 0.001$
spec × spec	1.01%	0.0155	$p \leq 0.001$
Residual σ^2	15.95%	0.2440	$F_{11782}^{105} \geq 591$
$\hat{R}^2$ **of Model**	**0.8405**	**1.2856**	$p \leq 0.001$

Table 7.10: ANOVA For Model Predicting **knn** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
knn	**1.1937**	**1.8626**	**0.7674**
	σ^2(%)	σ^2(real)	F-test
parentset	**38.59%**	**0.7188**	$p \leq 0.001$
pd ($\bar{\phi}$)	**19.75%**	**0.3679**	$p \leq 0.001$
–	10.51%	0.1957	$p \leq 0.001$
× parentset	9.25%	0.1722	$p \leq 0.001$
nc (ν)	2.66%	0.0495	$p \leq 0.001$
–	1.21%	0.0226	$p \leq 0.001$
× parentset	1.45%	0.0270	$p \leq 0.001$
fi (α)	11.51%	0.2144	$p \leq 0.001$
–	6.71%	0.1250	$p \leq 0.001$
× parentset	4.80%	0.0894	$p \leq 0.001$
rf (ρ)	12.35%	0.2300	$p \leq 0.001$
–	10.15%	0.1891	$p \leq 0.001$
× parentset	2.19%	0.0408	$p \leq 0.001$
spec × spec	1.30%	0.0243	$p \leq 0.001$
Residual σ^2	13.84%	0.2577	$F_{11782}^{105} \geq 698$
$\hat{R}^2$ **of Model**	**0.8616**	**1.6049**	$p \leq 0.001$

Table 7.11: ANOVA For Model Predicting **lof** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
lof	**1.0264**	**1.9117**	**0.7362**
	σ^2(%)	σ^2(real)	F-test
parentset	**42.51%**	**0.8126**	$p \leq 0.001$
pd ($\bar{\phi}$)	**17.24%**	**0.3296**	$p \leq 0.001$
–	10.79%	0.2062	$p \leq 0.001$
× parentset	6.45%	0.1234	$p \leq 0.001$
nc (ν)	2.50%	0.0478	$p \leq 0.001$
–	1.31%	0.0250	$p \leq 0.001$
× parentset	1.19%	0.0228	$p \leq 0.001$
fi (α)	9.85%	0.1882	$p \leq 0.001$
–	3.84%	0.0735	$p \leq 0.001$
× parentset	6.00%	0.1148	$p \leq 0.001$
rf (ρ)	12.12%	0.2317	$p \leq 0.001$
–	9.69%	0.1852	$p \leq 0.001$
× parentset	2.43%	0.0465	$p \leq 0.001$
spec × spec	1.23%	0.0235	$p \leq 0.001$
Residual σ^2	14.56%	0.2783	$F_{11782}^{105} \geq 658$
$\hat{R}^2$ **of Model**	**0.8544**	**1.6334**	$p \leq 0.001$

Table 7.12: ANOVA For Model Predicting **abod** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
abod	**0.9739**	**1.6466**	**0.7259**
	σ^2(%)	σ^2(real)	F-test
parentset	**32.83%**	**0.5406**	$p \leq 0.001$
pd ($\bar{\phi}$)	**16.63%**	**0.2739**	$p \leq 0.001$
–	10.82%	0.1781	$p \leq 0.001$
× parentset	5.82%	0.0958	$p \leq 0.001$
nc (ν)	5.17%	0.0852	$p \leq 0.001$
–	0.99%	0.0163	$p \leq 0.001$
× parentset	4.18%	0.0689	$p \leq 0.001$
fi (α)	14.34%	0.2361	$p \leq 0.001$
–	8.63%	0.1421	$p \leq 0.001$
× parentset	5.71%	0.0940	$p \leq 0.001$
rf (ρ)	10.15%	0.1672	$p \leq 0.001$
–	7.31%	0.1204	$p \leq 0.001$
× parentset	2.84%	0.0468	$p \leq 0.001$
spec × spec	1.49%	0.0246	$p \leq 0.001$
Residual σ^2	19.37%	0.3190	$F_{11782}^{105} \geq 466$
$\hat{R}^2$ of Model	**0.8063**	**1.3276**	$p \leq 0.001$

Table 7.13: ANOVA For Model Predicting **loda** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
loda	**0.9467**	**1.3784**	**0.7205**
	σ^2(%)	σ^2(real)	F-test
parentset	**38.77%**	**0.5345**	$p \leq 0.001$
pd ($\bar{\phi}$)	**19.62%**	**0.2705**	$p \leq 0.001$
–	8.99%	0.1239	$p \leq 0.001$
× parentset	10.63%	0.1466	$p \leq 0.001$
nc (ν)	3.63%	0.0501	$p \leq 0.001$
–	1.51%	0.0209	$p \leq 0.001$
× parentset	2.12%	0.0292	$p \leq 0.001$
fi (α)	8.19%	0.1129	$p \leq 0.001$
–	3.82%	0.0526	$p \leq 0.001$
× parentset	4.37%	0.0603	$p \leq 0.001$
rf (ρ)	10.25%	0.1412	$p \leq 0.001$
–	8.16%	0.1125	$p \leq 0.001$
× parentset	2.08%	0.0287	$p \leq 0.001$
spec × spec	1.26%	0.0174	$p = 0.0011$
Residual σ^2	18.27%	0.2518	$F_{11782}^{105} \geq 501$
$\hat{R}^2$ **of Model**	**0.8173**	**1.1266**	$p \leq 0.001$

Table 7.14: ANOVA For Model Predicting **iforest** (logit(AUC))

	μ(logit(AUC))	σ^2(real)	μ'(AUC)
iforest	**1.1726**	**2.0048**	**0.7636**
	σ^2(%)	σ^2(real)	F-test
parentset	**42.36%**	**0.8491**	$p \leq 0.001$
pd ($\bar{\phi}$)	**20.98%**	**0.4205**	$p \leq 0.001$
–	11.40%	0.2285	$p \leq 0.001$
× parentset	9.58%	0.1921	$p \leq 0.001$
nc (ν)	3.37%	0.0676	$p \leq 0.001$
–	1.37%	0.0274	$p \leq 0.001$
× parentset	2.00%	0.0402	$p \leq 0.001$
fi (α)	5.16%	0.1034	$p \leq 0.001$
–	2.09%	0.0420	$p \leq 0.001$
× parentset	3.06%	0.0614	$p \leq 0.001$
rf (ρ)	14.39%	0.2885	$p \leq 0.001$
–	12.11%	0.2428	$p \leq 0.001$
× parentset	2.28%	0.0457	$p \leq 0.001$
spec × spec	0.97%	0.0195	$p \leq 0.001$
Residual σ^2	12.78%	0.2562	$F_{11782}^{105} \geq 766$
$\hat{R}^2$ of Model	**0.8722**	**1.7486**	$p \leq 0.001$

Table 7.15: Summary of Variance Explained by Third-Order Models Predicting Both Metrics

algorithm	μ(logit(AUC))	Res. σ^2	$\sigma^2_{(\text{pset})}$	σ^2(pd)	σ^2(nc)	σ^2(fi)	σ^2(rf)
tmd	0.7877	**0.2427**	0.6974	**0.2380**	0.0691	0.1744	**0.1103**
egmm	**1.0509**	0.2738	0.7597	0.3234	0.0663	0.1867	0.1638
rkde	1.0335	0.2998	**0.6409**	0.2886	**0.0384**	0.2391	0.1672
ocsvm	0.7349	0.3055	1.3374	0.3633	0.1482	**0.0570**	0.2662
svdd	0.8008	**0.2440**	0.6778	**0.2408**	0.0655	0.1751	**0.1110**
knn	**1.1937**	0.2577	0.7188	0.3679	**0.0495**	0.2144	0.2300
lof	1.0264	0.2783	0.8126	0.3296	**0.0478**	0.1882	0.2317
abod	0.9739	0.3190	**0.5406**	0.2739	0.0852	0.2361	0.1672
loda	0.9467	**0.2518**	**0.5345**	**0.2705**	0.0501	**0.1129**	**0.1412**
iforest	**1.1726**	0.2562	0.8491	0.4205	0.0676	**0.1034**	0.2885
algorithm	μ(log(lift))	Res. σ^2	$\sigma^2_{(\text{pset})}$	σ^2(pd)	σ^2(nc)	σ^2(fi)	σ^2(rf)
tmd	0.7569	0.1357	0.4386	**0.1316**	0.1988	0.0375	**0.3122**
egmm	0.9020	0.1370	0.3900	0.1813	**0.1154**	0.0801	0.3972
rkde	0.9014	0.1626	**0.3612**	**0.1456**	**0.1088**	0.0617	0.3884
ocsvm	0.8537	**0.1144**	0.5675	0.1845	0.2085	**0.0159**	0.5371
svdd	0.7563	0.1353	0.4335	**0.1334**	0.1726	0.0379	**0.3028**
knn	**1.0415**	**0.1160**	**0.3395**	0.1733	0.1203	0.0627	0.5048
lof	0.9001	0.1404	0.3762	0.1708	0.1315	0.0532	0.4408
abod	**0.9046**	0.1861	**0.2912**	0.1537	**0.1009**	0.1176	0.3964
loda	0.8897	0.1454	0.3741	0.1641	0.1637	**0.0227**	**0.3612**
iforest	**1.0498**	**0.0956**	0.4760	0.2066	0.1398	**0.0132**	0.5549

are under consideration. Also, the interaction effect between this problem dimension and the choice of algorithm is the only representation either factor has in the model, and they are together treated as separate random effect. The remainder of the model includes first and second-order interactions between all other factors. For clarity the models for each problem dimension are specified here:

Contrast $\mathbf{pd}_1, \mathbf{pd}_3$: We use the pd_3 level instead of pd_4, because the results in Table 6.7 suggest that pd_3 is actually our most difficult setting. We contrast results across algorithms in benchmarks constructed at the pd_1 and pd_3 levels with the model:

$$\text{metric} \sim (1 + \text{parentset} + \nu + \log(\alpha) + \text{logit}(\rho))^2 + (1|\text{algorithm} \times \text{pd}_{(1,3)}) \quad (7.5)$$

Contrast $\mathbf{nc}_s, \mathbf{nc}_c$: For normalized clusteredness we only had two construction specifications with a lot of variation within each of them, so we define two new levels here to enforce greater contrast:

nc_s: benchmarks with a measure of $\nu < -\log(2)$ (highly scattered anomalies).

nc_c: benchmarks with a measure of $\nu > \log(2)$ (highly clustered anomalies).

We contrast results across algorithms in benchmarks constructed at the nc_s and nc_c levels with the model:

$$\text{metric} \sim \left(1 + \text{parentset} + \text{logit}(\bar{\phi}) + \log(\alpha) + \text{logit}(\rho)\right)^2 + (1|\text{algorithm} \times \text{nc}_{(s,c)}) \quad (7.6)$$

Contrast $\mathbf{fi}_0, \mathbf{fi}_3$: Our control setting of adding no noise features makes sense here so we contrast results across algorithms in benchmarks constructed at the fi_0 and fi_3 levels with the model:

$$\text{metric} \sim \left(1 + \text{parentset} + \text{logit}(\bar{\phi}) + \nu + \text{logit}(\rho)\right)^2 + (1|\text{algorithm} \times \text{fi}_{(0,3)}) \quad (7.7)$$

Contrast $\mathbf{rf}_1, \mathbf{rf}_5$: Ignoring our control setting, we contrast results across algorithms in benchmarks constructed at the rf_1 and rf_5 levels with the model:

$$\text{metric} \sim \left(1 + \text{parentset} + \text{logit}(\bar{\phi}) + \nu + \log(\alpha)\right)^2 + (1|\text{algorithm} \times \text{rf}_{(1,5)}) \quad (7.8)$$

We use mixed effects models to ensure that our coefficients of interest remained linearly independent. Tables 7.16 to 7.19 present these results. Confidence intervals across algorithms are computed relative to the **tmd** algorithm, while the problem dimension confidence intervals are computed around the observed change in performance. Confidence intervals that fail to differentiate themselves from their reference group are in bold.

For each setting, the best performing algorithm is in bold. The minimum and maximum statistically significant differences between settings are also in bold (that is, the algorithms with the best change and worst change).

The analogous data pertaining to the log(lift) models is presented in Tables G.14 to G.17.

These tables further illustrate how a comparison of algorithms can be greatly impacted by benchmark construction. Some highlights to observe are that **isola-**

Table 7.16: Algorithm Performance When Contrasting pd_1 and pd_3 (logit(AUC))

		logit(AUC)			
algo	$\mathrm{CI}^{\mathrm{algo}}_{0.999}$	pd_1	pd_3	$\mathrm{pd}_3 - \mathrm{pd}_1$	$\mathrm{CI}^{\mathrm{pd}}_{0.999}$
tmd		1.9666	1.2748	-0.6917	±0.0809
egmm	±0.0548	2.2667	1.4550	-0.8117	±0.0809
rkde	±0.0548	2.2364	1.5570	**-0.6794**	±0.0809
ocsvm	**±0.0548**	1.9464	1.1820	-0.7643	±0.0809
svdd	**±0.0548**	1.9836	1.2766	-0.7071	±0.0809
knn	±0.0548	2.4418	**1.6048**	-0.8370	±0.0809
lof	±0.0548	2.2595	1.4063	-0.8532	±0.0809
abod	±0.0548	2.1737	1.4557	-0.7181	±0.0809
loda	±0.0548	2.1640	1.3798	-0.7842	±0.0809
iforest	±0.0548	**2.4706**	1.4463	**-1.0242**	±0.0809

Table 7.17: Algorithm Performance When Contrasting nc_s and nc_c (logit(AUC))

		logit(AUC)			
algo	$\mathrm{CI}^{\mathrm{algo}}_{0.999}$	nc_s	nc_c	$\mathrm{nc}_c - \mathrm{nc}_s$	$\mathrm{CI}^{\mathrm{nc}}_{0.999}$
tmd		0.4564	0.1441	-0.3123	±0.0840
egmm	±0.0656	0.7900	0.6648	-0.1252	±0.0840
rkde	±0.0656	0.5626	0.7500	0.1874	±0.0840
ocsvm	±0.0656	0.7642	-0.1119	**-0.8761**	±0.0840
svdd	**±0.0656**	0.4576	0.2171	-0.2405	±0.0840
knn	±0.0656	0.7956	0.9416	0.1460	±0.0840
lof	±0.0656	0.8379	0.6114	-0.2266	±0.0840
abod	**±0.0656**	0.4721	**0.9743**	**0.5021**	±0.0840
loda	**±0.0656**	0.4487	0.5177	0.0690	**±0.0840**
iforest	±0.0656	**0.9456**	0.5727	-0.3728	±0.0840

Table 7.18: Algorithm Performance When Contrasting fi_0 and fi_3 (logit(AUC))

		logit(AUC)			
algo	$\text{CI}^{\text{algo}}_{0.999}$	fi_0	fi_3	$\text{fi}_3 - \text{fi}_0$	$\text{CI}^{\text{fi}}_{0.999}$
tmd		-0.3254	-0.8350	-0.5096	±0.0715
egmm	±0.0621	0.1209	-0.8955	-1.0164	±0.0715
rkde	±0.0621	0.1696	-0.8020	-0.9716	±0.0715
ocsvm	±0.0621	-0.5661	-0.6781	**-0.1120**	±0.0715
svdd	**±0.0621**	-0.2998	-0.8377	-0.5379	±0.0715
knn	±0.0621	**0.2911**	-0.6896	-0.9806	±0.0715
lof	±0.0621	-0.0669	-0.7495	-0.6826	±0.0715
abod	±0.0621	0.0645	-0.9741	**-1.0386**	±0.0715
loda	±0.0621	-0.1236	-0.7656	-0.6420	±0.0715
iforest	±0.0621	0.0858	**-0.5218**	-0.6076	±0.0715

Table 7.19: Algorithm Performance When Contrasting rf_1 and rf_5 (logit(AUC))

		logit(AUC)			
algo	$\text{CI}^{\text{algo}}_{0.999}$	rf_1	rf_5	$\text{rf}_5 - \text{rf}_1$	$\text{CI}^{\text{rf}}_{0.999}$
tmd		0.5788	-0.9232	-1.5020	±0.1247
egmm	±0.1601	1.3833	-0.7610	-2.1443	±0.1247
rkde	±0.1601	0.9976	-0.7500	-1.7476	±0.1247
ocsvm	±0.1601	1.6172	-1.0797	**-2.6969**	±0.1247
svdd	**±0.1601**	0.5838	-0.9095	**-1.4933**	±0.1247
knn	±0.1601	1.4466	**-0.6519**	-2.0986	±0.1247
lof	±0.1601	1.3139	-0.8557	-2.1696	±0.1247
abod	±0.1601	0.7845	-0.8258	-1.6104	±0.1247
loda	±0.1601	0.7717	-0.7897	-1.5614	±0.1247
iforest	±0.1601	**1.7584**	-0.6561	-2.4145	±0.1247

tion forest is the top performer on pd_1 benchmarks but it is the most negatively impacted by the change to pd_3, while **rkde** is impacted the least. **Isolation forest** is again the top performer on benchmarks with scattered anomalies, but the performance of **abod** actually improves with the shift to highly clustered anomalies and is the top performer in nc_c, an effect that was not discovered in the third-order regression models. **Knn** is, surprisingly, the top performer in many settings, including fi_0. **Isolation forest** is the top performer in fi_3 (many irrelevant features), something that runs counter to the claims of the algorithm's authors, but that is highly desirable.

When focusing on the log lift metric, some of these comparisons change. **Ocsvm** is the top performer in rf_1 (very few anomalies), while losing that honor to **isolation forest** in the logit(AUC) metric.

7.5 Impact of Parentset Choices

Given the high impact that a benchmark's parentset of origin can have on experimental outcomes, we want to see if selecting only a few particular parentsets can alter straightforward algorithm comparisons. We believe this is an important test, because studies in the literature rarely use as many data sources as we do here. We reproduce our first-order models as in Eq. (7.2), but this time we select only a few parentsets of origin and compare this against results from a different selection of parentsets. We are concerned here only with observing how a straightforward comparison of algorithms can be changed drastically when the original data source

Table 7.20: Algorithm Coefficients When Benchmarks Constrained to Parentsets **abalone particle wine** and **yearp**

algo	$CI_{0.999}$	Δlogit(AUC)	Δ log(lift)	$CI_{0.999}$
tmd		0.0000	0.0000	
egmm	±0.0646	-0.1182	-0.2682	±0.0501
rkde	±0.0646	**0.1792**	0.0263	±**0.0501**
ocsvm	±0.0646	-0.0741	-0.0339	±**0.0501**
svdd	±**0.0646**	0.0083	-0.0209	±**0.0501**
knn	±0.0646	0.1129	-0.0296	±**0.0501**
lof	±0.0646	-0.1028	-0.2039	±0.0501
abod	±0.0646	-0.1078	-0.2753	±0.0501
loda	±0.0646	0.1024	**0.0677**	±0.0501
iforest	±0.0646	0.0705	-0.0536	±0.0501

changes. Tables 7.20 and 7.21 show the algorithm coefficients predicting both of our metrics.

We note that in our study the overall best performing algorithms have been **isolation forest** and the trivial **knn** algorithms. Table 7.20 shows results from an analysis that only involves benchmarks constructed from parentsets *abalone, particle, wine* and *yearp*. Here, in the logit(AUC) metric, **rkde** is the best overall algorithm, and in the log(lift) metric, **loda** is the best overall algorithm.

Table 7.21 shows results from an analysis that only involves benchmarks constructed from parentsets *spambase, landsat, gas* and *comm.and.crime*. Here, in the logit(AUC) metric, **isolation forest** is the best overall algorithm, and in the log(lift) metric, **ocsvm** is the best overall algorithm.

Table 7.21: Algorithm Coefficients When Benchmarks Constrained to Parentsets **spambase landsat gas** and **comm.and.crime**

algo	$CI_{0.999}$	Δlogit(AUC)	Δ log(lift)	$CI_{0.999}$
tmd		0.0000	0.0000	
egmm	±0.0471	0.1897	0.1130	±0.0467
rkde	±0.0471	0.1475	0.0869	±0.0467
ocsvm	±0.0471	0.1464	**0.3791**	±0.0467
svdd	±**0.0471**	0.0454	0.0136	±**0.0467**
knn	±0.0471	0.3424	0.2640	±0.0467
lof	±0.0471	0.3086	0.2722	±0.0467
abod	±0.0471	0.2490	0.1364	±0.0467
loda	±0.0471	0.1852	0.1129	±0.0467
iforest	±0.0471	**0.3914**	0.3744	±0.0467

Unlike our problem dimension settings, the impact of data source (and choice of metric) on experimental outcomes is not as easily understood and yet it is also the most dramatic.

Chapter 8: Conclusions and Recommendations

Given the evidence provided, we feel confident making several conclusions and recommendations. We acknowledge that many conclusions are only one possible explanation.

8.1 Conclusions From Hypothesis Tests

Section 6.1 reveals that it is not uncommon for benchmarks created by methods common in the literature to be of a quality such that many algorithms cannot distinguish themselves from a random ranking in a statistical hypothesis test. We recommend that this concern be addressed more often in future experiments.

The only benchmark construction specification with an unnacceptably low acceptance rate was point difficulty level pd_4, which built benchmarks only from points that an oracle misclassified. This is intuitively a bad choice anyway and was only included in this study for thoroughness. We feel confident recommending against constructing benchmarks at this setting.

Given the demonstrably high impact of choice of parentset on outcomes, we recommend consulting Table F.3 when considering any of the data sources used in this study. Specifically we recommend against using parentsets *letter.rec, yeast* and *yearp* because of their particularly low acceptance rate.

We do not recommend using an uncontrolled relative frequency for intuitively obvious reasons, but we acknowledge that Table 6.5 does not provide sufficient evidence to support this. The confidence intervals used in our hypothesis tests are themselves functions of the relative frequency, so the acceptance rates in this context are not as informative. However, the fact that the acceptance rates at the uncontrolled level were higher than average does provide some evidence that anomaly detection algorithms in general are capable of picking up some amount of signal (albeit weak) as long as there is some imbalance in the classes. This reinforces results published by Liu, et al. [29] where results were positive even when the reported relative frequency was high.

Conversely, benchmarks with low relative frequency had a low acceptance rate, but Tables 6.10 and 7.19 suggest that the remainder of those benchmarks were also relatively easy compared to other benchmarks. This can be explained by what should be intuitively obvious: that benchmarks with very few anomalies are going to see much higher variance in their results, meaning more outstanding successes and more abject failures.

8.2 Conclusions and Recommendations About Experiment Design

The average AUC scores for the highest performing algorithms across all accepted benchmarks are between 0.7 and 0.8, yet AUC scores reported in the literature are typically above 0.9. Given the very good overall performance of the trivial **knn** method in this study and the inability of some algorithms to statistically

distinguish themselves from the trivial **tmd** algorithm, we are concerned that the overwhelmingly positive results reported in literature indicate benchmarks that would have been easily solved by a trivial algorithm anyway. We recommend the inclusion of more trivial baseline algorithms in future experiments to provide perspective on the triviality or non-triviality of the benchmarks used.

Based on Table F.3, we recommend against using synthetic datasets in general or at least recommend that the relative ease of solving synthetic problems be seriously acknowledged when drawing conclusions.

Tables 7.2 and 7.4 and others suggest that the biggest factor impacting experimental results is the selection of parentset or data source. This impact is explicitly demonstrated in Tables 7.20 and 7.21 where evaluating algorithms by a simple comparison of their mean performance can reach drastically different conclusions depending on data source and metric used. We conclude that experiments that do not well-justify their selection of datasets or their metric reported are simply not reporting reliable information.

We took focus away from AP-based metrics, but we do not conclude that they are poor metrics. Rather we point out that there are strong natural correlations between relative frequency and accepting benchmarks for analysis and that there are further correlations between relative frequency and the computation of log(lift) itself. The outsized impact of relative frequency in our AP-based models suggests to us that these results are simply hard to interpret in a useful way, not that they are inaccurate. Our AUC-based models still suggest the impact of relative frequency is significant and so the sensitivity of AP and lift to relative frequency

may be a desired quality of the metric depending on use-case.

Tables 7.2, 7.4 and 7.16 to 7.19 and others suggest that our defined problem dimensions all have an impact on experimental results. Unlike choice of parentset, these factors offer dimensions of outcome manipulation that can be more specific and more easily justified. For example, noting that **abod** does well when anomalies are highly clustered or that **isolation forest** is more resilient to excessive irrelevant features is a more reliable and reproducible finding than simply comparing algorithms across an unspecified problem setting with an unjustified selection of data sources.

Based on this we are able to recommend using our methodology (or something appropriately similar) for controlling and measuring these problem dimensions. We encourage further work that focuses on specific contexts that can be defined by these problem dimensions, especially if it maps these contexts to real-world applications.

8.3 Algorithm Recommendations

Because **isolation forest** performed very well on average and also in several specific contexts and because it has very good runtime properties, we recommend it for general use. However, we also recommend that context should impact your choice of algorithm. Based on Table 7.18 we observe that if you are confident in your feature space, probability density estimates such as **egmm** and **rkde** outperform **isolation forest**, but in large feature spaces of unknown quality they do not.

Based on Table 7.17 we recommend **abod** or **lof** for highly clustered anomalies.

Even with the surprising effectiveness of trivial **knn**, we also observe that the runtime properties of **isolation forest** and **loda** scale well to large datasets while the other algorithms do not. Similarly, density estimating methods such as **rkde** and **egmm** do not scale well to a large number of features (even when they are all relevant).

In general **svdd** and **ocsvm** performed poorly compared to other algorithms. We made our best effort to parameterize them well. Appendix E details our parameterization and what other works they are based on. However, we cannot conclude that these are poor algorithms, but rather that they are difficult to parameterize correctly and we were unable to get them to perform competitively with the other algorithms in this study. However, difficulty of use would be a reason not to recommend an algorithm. We also point out that **ocsvm** was the top performing algorithm in one category in Table G.17 and so we still recommend further work be done to understand and improve these algorithms.

8.4 Final Recommendations

Among the other algorithms we point out that the difference in performance among them is not very large, so while we do observe that **isolation forest** does very well overall, we emphasize that our battery of anomaly detection algorithms are more or less solving benchmarks with the same efficacy.

Experimental design and understanding the impact of different real world con-

texts seem to be of more importance given the evidence in this study. Incremental improvements in non-standardized environments demonstrate that a particular algorithm is effective, but they do not demonstrate any particular breakthroughs in the field. It is our opinion that on average most algorithms are roughly measuring the same quantity and producing the same results. Worse, the performances of **tmd** and **knn** in this study suggest that often algorithms are only marginally better than a trivial solution.

Well-mastered benchmarks do not allow any room for improvement, and a lack of standard benchmarks do not enable the recognition of true breakthroughs in the field. We recommend our own corpus of benchmarks or a corpus produced by similar methods as the *beginning* of a standardized test bed. Our corpus and the software that produced it can be found at:

`http://ir.library.oregonstate.edu/xmlui/handle/1957/59114`

We welcome future contributions in this area as well as criticisms and refinements of our existing corpus.

Bibliography

[1] Ahmad Alzghoul and Magnus Lofstrand. Increasing availability of industrial systems through data stream mining. *Computers & Industrial Engineering*, 60(2):195 – 205, 2011.

[2] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, ODD '13, pages 8–15, New York, NY, USA, 2013. ACM.

[3] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[4] Tharindu R Bandaragoda, Kai Ming Ting, David Albrecht, Fei Tony Liu, and Jason R Wells. Efficient anomaly detection by isolation using nearest neighbour ensemble. In *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, pages 698–705. IEEE, 2014.

[5] Yves Bestgen. Exact expected average precision of the random baseline for system evaluation. *The Prague Bulletin of Mathematical Linguistics*, 103(1):131–138, 2015.

[6] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] M. Breunig, H-P. Kriegel, R. T. Raymond T. Ng, and J. Sander. LOF: identifying density-based local outliers. *ACM SIGMOD Record*, pages 93–104, 2000.

[8] Robert A. Bridges, John Collins, Eric Ferragut, Jason Laska, and Blair D. Sullivan. Multi-level anomaly detection on streaming graph data. Technical Report 1410.4355v1, arXiv, 2014.

[9] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J. G. B. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, 2016.

[10] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[11] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[12] Kaustav Das, Jeff Schneider, and Daniel B Neill. Anomaly pattern detection in categorical datasets. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–176. ACM, 2008.

[13] Ethan Dereszynski and Thomas G Dietterich. Spatiotemporal models for anomaly detection in dynamic environmental monitoring campaigns. *ACM Transactions on Sensor Networks*, 8(1):3:1–3:26, 2011.

[14] Andrew Emmott, Shubhomoy Das, Thomas G. Dietterich, Alan Fern, and Weng-Keen Wong. Systematic construction of anomaly detection benchmarks from real data. *CoRR*, abs/1503.01158, 2015.

[15] Andrew F Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. Systematic construction of anomaly detection benchmarks from real data. In *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, pages 16–21. ACM, 2013.

[16] Joshua Glasser and Brian Lindauer. Bridging the gap: A pragmatic approach to generating insider threat data. In *2013 IEEE Security and Privacy Workshops*, pages 98–104. IEEE Press, 2013.

[17] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, 11(4):1–31, 04 2016.

[18] Julie Greensmith, Jamie Twycross, and Uwe Aickelin. Dendritic cells for anomaly detection. In *IEEE Congress on Evolutionary Computation*, pages 664–671. IEEE, 2006.

[19] Huaming Huang, Kishan Mehrotra, and Chilukuri K Mohan. An online anomalous time series detection algorithm for univariate data streams. In *Recent Trends in Applied Artificial Intelligence*, pages 151–160. Springer, 2013.

[20] Tommi Jaakkola, Mark Diekhans, and David Haussler. Using the fisher kernel method to detect remote protein homologies. In *In Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158. AAAI Press, 1999.

[21] Tommi Jaakkola and David Haussler. Probabilistic kernel regression models. In *Proceedings of the 1999 Conference on AI and Statistics*, volume 126, pages 00–04. San Mateo, CA, 1999.

[22] Deovrat Kakde, Arin Chaudhuri, Seunghyun Kong, Maria Jahja, Hansi Jiang, and Jorge Silva. Peak criterion for choosing gaussian kernel bandwidth in support vector data description. *arXiv preprint arXiv:1602.05257*, 2016.

[23] S.S. Keerthi, K.B. Duan, S.K. Shevade, and A.N. Poo. A fast dual algorithm for kernel logistic regression. *Machine Learning*, 61(1-3):151–165, 2005.

[24] Joo Seuk Kim and C. Scott. Robust kernel density estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3381–3384, 2008.

[25] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: Local outlier probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1649–1652, New York, NY, USA, 2009. ACM.

[26] Hans-Peter Kriegel, Arthur Zimek, et al. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452. ACM, 2008.

[27] Terran Lane and Carla E Brodley. Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43–49, 1997.

[28] Ar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *In Proceedings of SIAM Conference on Data Mining*, 2003.

[29] F T Liu, K M Ting, and Z-H Zhou. Isolation forest. In *Proceedings of the IEEE International Conference on Data Mining*, pages 413–422, 2008.

[30] F T Liu, K M Ting, and Z-H Zhou. On detecting clustered anomalies using SCiForest. In *Machine Learning and Knowledge Discovery in Databases*, pages 274–290, 2010.

[31] Matthew V Mahoney and Philip K Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

[32] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM transactions on Information and system Security*, 3(4):262–294, 2000.

[33] Qipei Mei and Mustafa Gul. An improved methodology for anomaly detection based on time series modeling. In *Topics in Dynamics of Civil Structures, Volume 4*, pages 277–281. Springer, 2013.

[34] Matthew Eric Otey, Amol Ghoting, and Srinivasan Parthasarathy. Fast distributed outlier detection in mixed-attribute data sets. *Data Mining and Knowledge Discovery*, 12(2-3):203–228, 2006.

[35] Tomáš Pevnỳ. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

[36] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[37] D. Pokrajac, A. Lazarevic, and L.J. Latecki. Incremental local outlier detection for data streams. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 504–515, 2007.

[38] Kemal Polat, Seral Sahan, Halife Kodaz, and Salih Gunes. A new classification method for breast cancer diagnosis: Feature selection artificial immune recognition system (fs-airs). In Lipo Wang, Ke Chen, and YewSoon Ong, editors, *Advances in Natural Computation*, volume 3611 of *Lecture Notes in Computer Science*, pages 830–838. Springer Berlin Heidelberg, 2005.

[39] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*. Citeseer, 2001.

[40] Sutharshan Rajasegarar, Christopher Leckie, James C Bezdek, and Marimuthu Palaniswami. Centered hyperspherical and hyperellipsoidal one-class support vector machines for anomaly detection in sensor networks. *Information Forensics and Security, IEEE Transactions on*, 5(3):518–533, 2010.

[41] David M. Rocke and David L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91(435):1047–1061, 1996.

[42] B. Scholkopf, J. C. Platt, J. Shawe-taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution, 1999.

[43] Ted E. Senator, Henry G. Goldberg, Alex Memory, William T. Young, Brad Rees, Robert Pierce, Daniel Huang, Matthew Reardon, David A. Bader, Edmond Chow, Irfan Essa, Joshua Jones, Vinay Bettadapura, Duen Horng Chau, Oded Green, Oguz Kaya, Anita Zakrzewska, Erica Briscoe, Rudolph IV L. Mappus, Robert McColl, Lora Weiss, Thomas G. Dietterich, Alan Fern, Weng-Keen Wong, Shubhomoy Das, Andrew Emmott, Jed Irvine, Jay-Yoon Lee, Danai Koutra, Christos Faloutsos, Daniel Corkill, Lisa Friedland, Amanda Gentzel, and David Jensen. Detecting insider threats in a real corporate database of computer usage activity. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 1393–1401, New York, NY, USA, 2013. Association for Computing Machinery.

[44] Tax and Duin. Support vector data description. *Machine Learning*, 54:45–66, 2004.

[45] K. L. Wagstaff, N. L. Lanza, D. R. Thompson, T. G. Dietterich, and M. S. Gilmore. Guiding scientific discovery with explanations using DEMUD. In *Proceedings of the Association for the Advancement of Artificial Intelligence AAAI 2013 Conference*, 2013.

[46] Yingchao Xiao, Huangang Wang, Lin Zhang, and Wenli Xu. Two methods of selecting gaussian kernel parameters for one-class svm and their application to fault detection. *Knowledge-Based Systems*, 59:75–84, 2014.

[47] Feng Xue, Weizhong Yan, Nicholas Roddy, and Anil Varma. Operational data based anomaly detection for locomotive diagnostics. In *International Conference on Machine Learning*, pages 236–241, 2006.

[48] Bin Zhang, C. Sconyers, C. Byington, R. Patrick, M. Orchard, and G. Vachtsevanos. Anomaly detection: A robust approach to detection of unanticipated faults. In *Prognostics and Health Management, 2008. PHM 2008. International Conference on*, pages 1–8, 2008.

[49] Ji Zhu and Trevor Hastie. Kernel logistic regression and the import vector machine. In *Journal of Computational and Graphical Statistics*, pages 1081–1088. MIT Press, 2001.

APPENDICES

Appendix A: Comparison to Our Previous Work

While the general goals and conclusions of our previously published work are similar to this study there are many key differences in benchmark construction and evaluation of results.

In [15] we identified the problem dimensions of **relative frequency**,**point difficulty** and **clusteredness** and created a corpus of benchmarks varying these properties across several values, using the same 19 parentsets as in this study.

In [14] we added the problem dimension of **feature irrelevance**.

These studies used a different battery of algorithms than the current study; we believe this study uses a more representative set of algorithms for evaluation and also includes trivial algorithms. This study also adds a control group setting for each problem dimension, including the addition of the synthetic parentset. While the previous studies may have made a convincing argument for the impact of the problem dimensions, this study explicitly demonstrates that manipulating these problem dimensions has a statistically significant impact in most cases.

In previous studies, evaluation was done with a linear regression model, but here we construct multiple models and present more in depth ANOVA.

In this study the corpus of benchmarks itself is also evaluated more rigorously. We perform a statistical hypothesis test on each algorithm's output to each benchmark. Benchmarks for which even the best performing algorithm fails to

differentiate itself from a random ranking with high probability are discarded as unsuitable.

Appendix B: The Synthetic parentset

The synthetic parentset was generated by producing 10,000 candidate nominals and 10,000 candidate anomalies from two different multivariate distributions with the intention of being able to manipulate all problem dimensions with ease. The candidate nominals are drawn from a multivariate gaussian with a covariance matrix of I; that is, each feature is drawn from the standard normal distribution independently of the others. The anomalies are drawn uniformly from the hypercube defined by the range $(-4, 4)$ in each dimension. Both distributions have ten dimensions; that is, each point exists in R^{10}.

Appendix C: Choosing a Confusing Partition of Classes

Our heuristic procedure begins by training a Random Forest as in [6] to solve the multi-class classification problem. Then we calculate the amount of confusion between each pair of classes. For each data point x_i, the Random Forest computes an estimate of $P(\hat{y}_i = k|x_i)$, the predicted probability that x_i belongs to class k. We construct a confusion matrix C in which cell $C_{j,k}$ contains the sum of $P(\hat{y}_i = k|x_i)$ for all x_i whose true class $y_i = j$. We then define a graph in which each node is a class and each edge (between two classes j and k) has a weight equal to $C[j,k] + C[k,j]$. This is the (un-normalized) probability that a data point in class j will be confused with a data point in class k or vice versa. We then compute the maximum weight spanning tree of this (complete) graph to identify a graph of "most-confusable" relationships between pairs of classes. We then two-color this tree so that no adjacent nodes have the same color. The two colors define the two classes of points.

This approximately maximizes the confusion between the candidate nominal and candidate anomaly data points and also tends to make both classes diverse, which increases semantic variation in both sets.

Appendix D: Our Implementation of KLR

We implemented the kernel logistic regression algorithm described by Keerthi, et al., [23] to assign point difficulty scores. Given a set of data points $\{(x_i, y_i)_{i=1}^n\}$ where each y_i is a binary class label, general logistic regression can be summarized as fitting a model to estimate the probability $P(y_i = 1|x_i)$ by minimizing the negative log-likelihood of the data given this model. The implementation suggested by Keerthi, et al. in [23] identifies the dual of this optimization problem as a convex optimization problem and solves the dual with methods similar to the SMO algorithm presented by Platt in [36].

As is typical, a radial basis function RBF kernel was employed in our implementation. Two important hyperparameters for this algorithm are kernel bandwidth σ and regularization penalty scalar C. σ was chosen with the Jaakkola heuristic as described in [24] and attributed to [20] which works as follows: for all points in the data set, determine the distance to their nearest neighbor, then select the median of these distances.

The regularization penalty scalar C was chosen via 5-fold cross-validation from the 11 values in $\{10^{2i}\}_{i=-5}^{5}$.

For the RBF kernel the number of kernel parameters to optimize grows linearly with the size of the data set, which makes individual optimization steps scale quadratically with this value. Because many parentsets contain over 10,000 data

points several others contain over 100,000, this made straightforward kernel logistic regression impractical. On parentsets larger than 5,000 we employed a further approximation of our own invention.

The justification for our approximation comes from an intuitive understanding of the radial basis kernel. The kernel space defined by such a kernel has features inversely proportional to the euclidean distance from each point in the original feature space. In this way, the probability distribution of each class is defined by a few "import vectors" [49] whose location defines regions of high probability.

In practice, despite the existence of a parameter for each data point, only a fraction are needed, and selected import vectors can be well-approximated by other points near them in feature space. Because of this, we simplify computation costs by randomly (but preserving original class balance) sampling 5,000 data points from each parentset larger than 5,000 data points. 10 samples of 5,000 are taken from each parentset and KLR optimization is performed on each of these samples. The sample model with the minimal negative log-likelihood given the entire parentset is selected as our final model.

To validate our approach and verify that our implementation was performing reasonably well, we present the final classification accuracy of our implementation on each parentset in Table D.1. The original task of each parentset is noted as well. parentsets larger than 5,000 requiring our sampling approximation are noted in bold.

Table D.1: Classification Accuracy of KLR Oracle

parentset	**Accuracy**
Binary Classification	
synthetic	0.9879
magic.gamma	0.8712
particle	0.8707
skin	0.9850
spambase	0.9948
Multiclass Classification	
fault	0.9279
gas	0.9698
imgseg	1.0000
landsat	0.9565
letter.rec	0.9619
opt.digits	0.9979
pageb	0.9832
shuttle	0.9284
wave	0.9694
yeast	0.9710
Regression	
abalone	0.8118
comm.and.crime	0.9714
concrete	0.9816
wine	0.9267
yearp	0.5510

Appendix E: Parameterizing the Battery of Algorithms

We made a good faith effort to get every algorithm to perform well on our entire corpus but individual algorithms might see their performance improve with different formulations or better parameterizations. In an effort toward transparency we report on literature source, implementation source, and parameterization details for each algorithm.

Some algorithms have parameters that might make sense to scale proportionately with the relative frequency or some other measurable property of the benchmark. Because this is an unsupervised problem we did not allow parameters to be tuned according to any such information, but in real applications it is possible that a domain expert might be able to estimate this information and make a more informed parameter choice for an algorithm in practice.

E.1 TMD

This algorithm has no parameters, is easily understood, and was implemented in R. It is worth noting, however, that in Appendix B we detail our synthetic control parentset and we should point out that the true density function defined by the distribution of the nominal points in the synthetic parentset is a monotonic transformation of the output of this algorithm.

E.2 EGMM

EGMM constructs an ensemble of Gaussian mixture models for density estimation. The procedure was of our own design and implemented in C++.

To reduce the computational cost of fitting, and to improve the numerical stability of the process, we first transformed each benchmark via principle component analysis. We selected principle components (in descending eigenvalue order) to retain 95% of the variance.

To generate the members of the ensemble, we varied the number of clusters k by trying all values in $\{1, 2, 3, 4, 5, 6\}$. For each value of k, we generated 15 GMMs by training on 15 bootstrap replicates of the data and by randomly initializing each replicate. We then computed the average out-of-bag log likelihood for each value of k and discarded k values whose average log likelihood was less than 0.85 times the average log likelihood of the best value of k. The purpose of this was to discard GMMs that do not fit the data very well. Finally, an anomaly score is computed for each point x by computing the average "surprise", which is the average negative log probability density $\frac{1}{L}\sum_{\ell=1}^{L} -\log P_\ell(x)$, where L is the number of fitted GMMs and $P_\ell(x)$ is the density assigned by GMM ℓ to data point x. We found in preliminary experiments that this worked better than using the mean probability density $\frac{1}{L}\sum_{\ell=1}^{L} P_\ell(x)$.

E.3 RKDE

We followed the approach described by Kim and Scott in [24]. The authors provided their own Matlab implementation but we rewrote an implementation in R for ease of interfacing with our study.

We employed a Gaussian radial basis kernel, with kernel bandwidth selected as suggested by the authors and described in Appendix D. We optimized over a Hampel loss function with the additional parameters set as suggested by the authors.

E.4 OCSVM

We followed several approaches but the algorithm is present by Scholkopf et al. in [42].

We employed the `libsvm` implementation of Chang and Lin [10] available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. For each benchmark, we employed a Gaussian radial basis function kernel. Selection of kernel bandwidth was done using the DFN method proposed in [46], while the quantile parameter ν was set to 0.1. We found this to be the best overall performing value of ν across a large range of choices.

E.5 SVDD

The original formulation is presented by Tax and Duin in [44]. The problem is presented as a linear program and solving its dual is suggested as the solution. The optimization is performed in kernel space. We used our own R implementation and solve the dual with the simplex method.

We employed a radial basis function kernel with the kernel bandwidth selected using the method of Kakde, et al. [22]. To generate the needed statistics, we tested 100 different kernel bandwidths.

As with OCSVM, we tried a large range of quantiles to estimate. Surprisingly, a quantile boundary of 0.5 was the best performing quantile for this algorithm.

E.6 KNN

We implemented this trivial algorithm in R.

Like the others in its family, this algorithm requires a choice of k as a parameter. As we did with all other algorithms in the study we made a good faith attempt to get the best performance out of this algorithm as we could. Running this algorithm with several values of k on our corpus, we chose $k = 5$ as the best performing value of k on average.

To be clear, our implementation is as minimal as possible. We do not average the distances to the k-th nearest neighbors or any similar normalization. The anomaly score is derived from exactly one measure of distance: the distance to the k-th nearest neighbor.

E.7 LOF

We followed the algorithm as proposed by Breunig, et al. in [7]. We employed the `R` package `Rlof` available at `http://cran.open-source-solution.org/web/packages/Rlof/`.

We chose k to be 3% of the dataset. This was the largest value for which LOF would complete in a practical amount of time on all benchmarks.

E.8 ABOD

We followed the algorithm proposed by Kriegel, et al. in [26]. The running time of the canonical algorithm is very high: $O(n^3d^2)$ where d is the number of dimensions.

For our R implementation we employed a simplified version of the algorithm as suggested by the authors: We only compute angles with respect to the k nearest neighbors of x_i, where k is set to $0.005 \times n$, where n is the number of points in the benchmark dataset.

E.9 LODA

We used the algorithm as described in Pevny [35]. We provided our own R implementation. We followed the authors suggestion of using approximately $\sqrt{d}$ features for each random projection, where d is the number of dimensions in the benchmark set. The total ensemble is built from $3 \times d$ random projections.

E.10 IFOR

We used the algorithm as presented by Liu, et al. in [29]. The authors provided their own R implementation. Each isolation tree is grown on a subsample of the full dataset. The authors suggest a sample size of 256, but the choice feels motivated by their run-time analysis which claims a linear run-time for the algorithm as long as this parameter is constant.

For our study we found a subsample size of 2048 (or the entire dataset when a benchmark is smaller than this) to work best instead of the suggestion by the authors. It is likely that for even larger sets this parameter should be even larger.

Appendix F: Corpus Evaluation Supplement

F.1 Treating AUC and AP as Random Variables

The AUC or AP of a random ranking of a benchmark can be seen as a discrete parametric distribution with parameters $|\beta_n|$ (number of nominals) and $|\beta_a|$ (number of anomalies). The distribution is discrete because there are "only" $(|\beta_n|+|\beta_a|)!$ possible rankings, meaning there are a finite number of possible AUC or AP scores for a given set of parameters.

In both cases it is possible to enumerate these scores and compute how much probability mass each score carries, and thus quantiles of these distributions can be computed. However for larger values of $|\beta_n|+|\beta_a|$ this becomes computationally inefficient.

Instead we computed the quantiles of interest empirically. For each set of parameters present in our corpus, we produced 1 million random ranking samples, computed the AUC and AP of each, and from this estimated the quantiles of interest.

F.2 Supplemental Results

For compactness of presentation we have additional results tables related to Chapter 6 here. Table F.1 shows the benchmark acceptance rate by parentset. Rates

higher than the global average for the metric are in bold. There is one interesting result in this table, which is that the acceptance rate of benchmarks from the *yeast* parentset are extremely low. We do not eliminate the accepted benchmarks from the final analysis, but future research might conclude that this parentset is unsuitable for the task.

Table F.2 shows null hypothesis rejection rates by algorithm *on benchmarks already accepted in that metric*. This is not a rigorous examination of the algorithms and so is not present in the main body of the test, but the numbers here do show trends similar to the ones observed in our regression analyses. The most successful algorithm in each metric is in bold; surprisingly knn, one of our trivial algorithms, proves best at differentiating itself from random ranking.

Table F.3 shows the model coefficients and 0.999 confidence interval ranges for the algorithm-agnostic model.

Table F.1: Benchmark Acceptance Rate by Metric and parentset

parentset	AUC	AP	Both
synthetic	**0.7583**	**0.7455**	**0.7273**
magic.gamma	**0.6700**	**0.6656**	**0.6117**
particle	**0.7517**	**0.6333**	**0.5822**
skin	**0.8920**	**0.6313**	**0.6313**
spambase	**0.6389**	**0.5133**	**0.4911**
fault	**0.6007**	0.3947	0.3888
gas	**0.6378**	0.4167	0.4100
imgseg	**0.7236**	**0.7444**	**0.6653**
landsat	0.4720	0.4142	0.3835
letter.rec	0.3884	0.2807	0.2720
opt.digits	0.4258	0.2691	0.2544
pageb	**0.9543**	**0.8021**	**0.8011**
shuttle	**0.8844**	**0.7167**	**0.7156**
wave	0.5352	0.4157	0.3981
yeast	0.0256	0.0200	0.0078
abalone	**0.6168**	**0.5559**	**0.5362**
comm.and.crime	0.5121	**0.5076**	0.4515
concrete	0.4921	0.4184	0.3799
wine	0.5157	0.3669	0.3496
yearp	0.2967	0.2644	0.2328

Table F.2: Algotihm H_0 Rejection Rate on Accepted Benchmarks

algorithm	**AUC**	**AP**	**Both**
tmd	0.5346	0.5134	0.4972
egmm	0.6503	0.5847	0.5828
rkde	0.6616	0.6085	0.6063
ocsvm	0.4862	0.5348	0.4681
svdd	0.5490	0.5104	0.5028
knn	**0.7916**	**0.7296**	**0.7204**
lof	0.6796	0.6000	0.5767
abod	0.6998	0.6589	0.6482
loda	0.6505	0.6238	0.6063
iforest	0.7643	0.7141	0.7042

Table F.3: Parentset Coefficients Estimating Metrics in Algorithm-Agnostic Models

parentset	Δlogit(AUC)	$CI_{0.999}$	Δ log(lift)	$CI_{0.999}$
synthetic	0.0000		0.0000	
magic.gamma	-2.4216	±0.0405	-0.9605	±0.0299
particle	-2.5212	±0.0415	-1.3572	±0.0306
skin	-2.4766	±0.0422	-1.7894	±0.0311
spambase	-2.8602	±0.0528	-1.7090	±0.0389
fault	-2.2678	±0.0489	-1.3085	±0.0360
gas	-3.1271	±0.0562	-1.9265	±0.0414
imgseg	-2.6583	±0.0513	-1.6481	±0.0378
landsat	-3.0020	±0.0467	-1.7771	±0.0344
letter.rec	**-3.1686**	±0.0561	**-1.9893**	±0.0413
opt.digits	-2.9680	±0.0628	-1.9376	±0.0463
pageb	**-1.8034**	±0.0444	**-0.7998**	±0.0327
shuttle	-2.5167	±0.0465	-1.4467	±0.0343
wave	-2.9106	±0.0532	-1.7491	±0.0392
yeast	-1.8674	±0.3443	-1.1358	±0.2537
abalone	-1.9147	±0.0425	-0.9039	±0.0313
comm.and.crime	-2.5092	±0.0478	-1.2560	±0.0352
concrete	-2.0348	±0.0531	-1.0750	±0.0391
wine	-2.8175	±0.0537	-1.6420	±0.0396
yearp	-2.7097	±0.0550	-1.4489	±0.0405
σ^2 **Explained**	37.75%		26.78%	
$\hat{R}^2$ **of Model**	**0.5805**		**0.6955**	

Appendix G: Supplemental Tables for Chapter 7

Table G.1: ANOVA for Second-Order Regression Model Predicting log(lift)

	$\sigma^2(\%)$	**group**$(\%)$	F-test
algorithm	4.86%		$p \leq 0.001$
–	0.63%	13.01%	$p \leq 0.001$
× parentset	3.10%	**63.85%**	$p \leq 0.001$
× specifications	1.13%	23.14%	$p \leq 0.001$
parentset	35.26%		$p \leq 0.001$
–	**26.78%**	**75.95%**	$p \leq 0.001$
× specifications	8.48%	24.05%	$p \leq 0.001$
× pd ($\bar{\phi}$)	4.05%	11.49%	$p \leq 0.001$
specifications	**46.43%**		$p \leq 0.001$
pd ($\bar{\phi}$)	7.87%	16.96%	$p \leq 0.001$
nc (ν)	8.26%	17.79%	$p \leq 0.001$
fi (α)	2.06%	4.43%	$p \leq 0.001$
rf (ρ)	25.85%	**55.67%**	$p \leq 0.001$
× specifications	2.39%	5.15%	$p \leq 0.001$
Residual σ^2	13.45%		$F_{118558}^{321} \geq 2377$
$\hat{R}^2$ of Model	**0.8655**		$p \leq 0.001$

Table G.2: Second-Order Algorithm-parentset Coefficients Predicting logit(AUC)

	tmd	egmm	rkde	ocsvm	svdd	knn	lof	abod	loda	iforest
synthetic	0.00	-0.01	**0.03**	-0.06	-0.02	-0.06	-0.13	-0.47	-0.43	-0.32
magic.gamma	-0.11	**0.01**	**0.06**	-0.56	**0.02**	**0.12**	-0.03	**0.64**	**0.37**	**0.12**
particle	**0.22**	-0.27	**0.03**	-0.60	**0.03**	-0.30	-0.38	**0.13**	**0.34**	-0.45
skin	-0.86	**0.61**	**0.62**	-0.81	**0.02**	**0.79**	**0.11**	**1.45**	**0.79**	**0.24**
spambase	**0.72**	-0.47	-0.25	-0.26	-0.00	-0.44	-0.66	**0.14**	**0.71**	-0.09
fault	**0.85**	**0.26**	**0.29**	-0.40	**0.01**	**0.46**	-0.26	**0.59**	**0.55**	**0.25**
gas	-1.12	-0.22	**0.00**	-0.26	**0.22**	-0.16	-0.24	**0.56**	**0.08**	-0.62
imgseg	**2.18**	-1.09	-0.20	-1.00	-0.05	-0.89	-1.00	-0.73	**0.38**	-0.61
landsat	-0.28	**0.11**	**0.06**	-0.59	**0.00**	**0.16**	**0.23**	**0.65**	**0.48**	**0.08**
letter.rec	**0.28**	**0.05**	**0.00**	-0.54	-0.01	**0.37**	-0.36	**0.95**	**0.46**	-0.35
opt.digits	**1.51**	-0.44	-0.29	-0.70	-0.04	-0.25	-0.27	**0.13**	**0.17**	-0.33
pageb	**1.40**	-0.60	-0.40	-0.63	-0.00	-0.70	-0.42	-0.61	**0.04**	-0.29
shuttle	-0.37	**0.30**	**0.25**	-2.60	**0.00**	**0.21**	-0.00	**0.71**	**0.68**	**0.12**
wave	-0.07	-0.31	-0.01	-0.83	**0.01**	-0.29	-0.33	-0.04	**0.25**	-0.29
yeast	**0.67**	-1.37	-0.29	-1.16	-0.03	-0.59	-0.66	-0.90	**0.80**	-0.39
abalone	-1.18	-0.33	**0.30**	-0.66	**0.02**	**0.20**	-0.41	**0.34**	**0.53**	**0.14**
comm.and.crime	-0.62	-0.45	-0.04	-0.56	-0.00	-0.15	-0.29	-0.02	**0.31**	**0.03**
concrete	**0.10**	**0.42**	**0.38**	-0.53	**0.03**	**0.33**	**0.19**	**0.41**	**0.61**	**0.48**
wine	**2.87**	-0.20	-0.01	-0.59	**0.01**	-0.21	-0.32	**0.32**	**0.34**	-0.18
yearp	**0.15**	-0.09	-0.01	-0.29	**0.05**	-0.03	-0.04	**0.35**	**0.36**	**0.04**

Table G.3: Second-Order Algorithm-parentset Coefficients Predicting log(lift)

	tmd	egmm	rkde	ocsvm	svdd	knn	lof	abod	loda	iforest
synthetic	0.00	-0.01	**0.02**	-0.24	**0.01**	-0.11	-0.19	-0.27	-0.29	-0.39
magic.gamma	-0.39	**0.18**	**0.08**	**0.04**	**0.02**	**0.21**	**0.16**	**0.52**	**0.27**	**0.32**
particle	**0.29**	-0.34	-0.10	-0.22	-0.02	-0.19	-0.25	-0.05	**0.19**	-0.45
skin	-0.28	**0.19**	**0.31**	-0.36	-0.02	**0.47**	**0.12**	**0.92**	**0.36**	**0.08**
spambase	**1.98**	-0.40	-0.15	**0.34**	-0.07	-0.12	-0.23	**0.12**	**0.18**	**0.15**
fault	**0.39**	**0.15**	**0.19**	**0.04**	-0.01	**0.32**	-0.04	**0.37**	**0.28**	**0.30**
gas	-0.30	-0.18	-0.04	**0.06**	**0.07**	**0.01**	**0.02**	**0.33**	-0.08	-0.30
imgseg	-1.08	-0.99	-0.36	-0.05	-0.09	-0.61	-0.60	-0.80	**0.16**	**0.33**
landsat	-0.19	**0.03**	-0.16	**0.21**	-0.02	**0.08**	**0.38**	**0.27**	**0.27**	**0.32**
letter.rec	**0.33**	**0.43**	**0.00**	-0.05	-0.02	**0.68**	**0.17**	**1.16**	**0.25**	-0.05
opt.digits	**1.26**	-0.43	-0.25	-0.13	-0.09	-0.04	-0.06	**0.07**	-0.12	**0.11**
pageb	**0.57**	-0.19	**0.02**	**0.02**	**0.03**	-0.03	-0.05	**0.13**	**0.08**	**0.14**
shuttle	**0.05**	**0.17**	**0.25**	-0.90	-0.02	**0.39**	**0.24**	**0.66**	**0.40**	**0.19**
wave	-0.09	**0.05**	-0.05	-0.34	-0.02	**0.13**	**0.15**	**0.40**	**0.05**	-0.09
yeast	**1.45**	-0.96	-0.30	-0.26	-0.05	-0.17	-0.35	-0.66	**0.64**	**0.25**
abalone	-0.51	-0.62	-0.08	-0.25	-0.00	-0.28	-0.54	-0.35	**0.20**	-0.00
comm.and.crime	-0.34	-0.40	-0.04	**0.15**	-0.03	-0.01	-0.07	-0.25	**0.11**	**0.17**
concrete	-0.05	**0.25**	**0.12**	-0.04	**0.00**	**0.12**	**0.09**	**0.06**	**0.26**	**0.34**
wine	**2.02**	-0.15	-0.06	-0.14	-0.01	-0.08	-0.09	**0.18**	**0.14**	**0.00**
yearp	-0.13	-0.13	**0.02**	-0.02	-0.06	**0.03**	**0.07**	**0.24**	**0.22**	**0.12**

Table G.4: ANOVA For Model Predicting **tmd** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
tmd	**0.7569**	**1.2827**	**2.1317**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	**34.19%**	**0.4386**	$p \leq 0.001$
pd ($\bar{\phi}$)	10.26%	0.1316	$p \leq 0.001$
–	5.44%	0.0698	$p \leq 0.001$
× parentset	4.82%	0.0618	$p \leq 0.001$
nc (ν)	15.50%	0.1988	$p \leq 0.001$
–	12.04%	0.1545	$p \leq 0.001$
× parentset	3.45%	0.0443	$p \leq 0.001$
fi (α)	2.92%	0.0375	$p \leq 0.001$
–	1.21%	0.0155	$p \leq 0.001$
× parentset	1.71%	0.0220	$p \leq 0.001$
rf (ρ)	**24.34%**	**0.3122**	$p \leq 0.001$
–	18.34%	0.2353	$p \leq 0.001$
× parentset	6.00%	0.0769	$p \leq 0.001$
spec × spec	2.21%	0.0284	$p \leq 0.001$
Residual σ^2	10.58%	0.1357	$F_{11782}^{105} \geq 948$
$\hat{R}^2$ **of Model**	**0.8942**	**1.1470**	$p \leq 0.001$

Table G.5: ANOVA For Model Predicting **egmm** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
egmm	**0.9020**	**1.3277**	**2.4646**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	29.38%	0.3900	$p \leq 0.001$
pd ($\bar{\phi}$)	13.65%	0.1813	$p \leq 0.001$
–	9.52%	0.1264	$p \leq 0.001$
× parentset	4.13%	0.0548	$p \leq 0.001$
nc (ν)	8.69%	0.1154	$p \leq 0.001$
–	7.34%	0.0975	$p \leq 0.001$
× parentset	1.35%	0.0179	$p \leq 0.001$
fi (α)	6.03%	0.0801	$p \leq 0.001$
–	4.78%	0.0634	$p \leq 0.001$
× parentset	1.25%	0.0167	$p \leq 0.001$
rf (ρ)	**29.91%**	**0.3972**	$p \leq 0.001$
–	24.39%	0.3238	$p \leq 0.001$
× parentset	5.52%	0.0733	$p \leq 0.001$
spec × spec	2.01%	0.0267	$p \leq 0.001$
Residual σ^2	10.32%	0.1370	$F_{11782}^{105} \geq 975$
$\hat{R}^2$ of Model	**0.8968**	**1.1907**	$p \leq 0.001$

Table G.6: ANOVA For Model Predicting **rkde** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
rkde	**0.9014**	**1.2473**	**2.4632**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	28.96%	0.3612	$p \leq 0.001$
pd ($\bar{\phi}$)	11.67%	0.1456	$p \leq 0.001$
–	7.24%	0.0903	$p \leq 0.001$
× parentset	4.43%	0.0552	$p \leq 0.001$
nc (ν)	8.72%	0.1088	$p \leq 0.001$
–	5.72%	0.0713	$p \leq 0.001$
× parentset	3.00%	0.0375	$p \leq 0.001$
fi (α)	4.95%	0.0617	$p \leq 0.001$
–	3.86%	0.0481	$p \leq 0.001$
× parentset	1.09%	0.0136	$p \leq 0.001$
rf (ρ)	**31.14%**	**0.3884**	$p \leq 0.001$
–	26.93%	0.3358	$p \leq 0.001$
× parentset	4.21%	0.0525	$p \leq 0.001$
spec × spec	1.53%	0.0191	$p \leq 0.001$
Residual σ^2	13.03%	0.1626	$F_{11782}^{105} \geq 748$
$\hat{R}^2$ of Model	**0.8697**	**1.0847**	$p \leq 0.001$

Table G.7: ANOVA For Model Predicting **ocsvm** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
ocsvm	**0.8537**	**1.6480**	**2.3484**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	**34.43%**	**0.5675**	$p \leq 0.001$
pd ($\bar{\phi}$)	11.20%	0.1845	$p \leq 0.001$
–	6.84%	0.1128	$p \leq 0.001$
× parentset	4.35%	0.0718	$p \leq 0.001$
nc (ν)	12.65%	0.2085	$p \leq 0.001$
–	10.21%	0.1683	$p \leq 0.001$
× parentset	2.44%	0.0402	$p \leq 0.001$
fi (α)	0.96%	0.0159	$p \leq 0.001$
–	0.14%	0.0024	$p \leq 0.001$
× parentset	0.82%	0.0135	$p \leq 0.001$
rf (ρ)	**32.59%**	**0.5371**	$p \leq 0.001$
–	26.67%	0.4395	$p \leq 0.001$
× parentset	5.92%	0.0976	$p \leq 0.001$
spec × spec	1.23%	0.0202	$p \leq 0.001$
Residual σ^2	6.94%	0.1144	$F_{11782}^{105} \geq 1504$
$\hat{R}^2$ of Model	**0.9306**	**1.5336**	$p \leq 0.001$

Table G.8: ANOVA For Model Predicting **svdd** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
svdd	**0.7563**	**1.2416**	**2.1303**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	**34.92%**	**0.4335**	$p \leq 0.001$
pd ($\bar{\phi}$)	10.75%	0.1334	$p \leq 0.001$
–	5.87%	0.0729	$p \leq 0.001$
× parentset	4.88%	0.0605	$p \leq 0.001$
nc (ν)	13.90%	0.1726	$p \leq 0.001$
–	10.87%	0.1350	$p \leq 0.001$
× parentset	3.03%	0.0376	$p \leq 0.001$
fi (α)	3.05%	0.0379	$p \leq 0.001$
–	1.41%	0.0176	$p \leq 0.001$
× parentset	1.64%	0.0203	$p \leq 0.001$
rf (ρ)	**24.39%**	**0.3028**	$p \leq 0.001$
–	18.09%	0.2246	$p \leq 0.001$
× parentset	6.29%	0.0781	$p \leq 0.001$
spec × spec	2.10%	0.0261	$p \leq 0.001$
Residual σ^2	10.90%	0.1353	$F_{11782}^{105} \geq 917$
$\hat{R}^2$ of Model	**0.8910**	**1.1063**	$p \leq 0.001$

Table G.9: ANOVA For Model Predicting **knn** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
knn	**1.0415**	**1.3339**	**2.8334**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	25.45%	0.3395	$p \leq 0.001$
pd ($\bar{\phi}$)	12.99%	0.1733	$p \leq 0.001$
–	8.81%	0.1175	$p \leq 0.001$
× parentset	4.18%	0.0558	$p \leq 0.001$
nc (ν)	9.02%	0.1203	$p \leq 0.001$
–	6.72%	0.0896	$p \leq 0.001$
× parentset	2.30%	0.0307	$p \leq 0.001$
fi (α)	4.70%	0.0627	$p \leq 0.001$
–	3.90%	0.0521	$p \leq 0.001$
× parentset	0.80%	0.0106	$p \leq 0.001$
rf (ρ)	**37.84%**	**0.5048**	$p \leq 0.001$
–	35.04%	0.4673	$p \leq 0.001$
× parentset	2.81%	0.0374	$p \leq 0.001$
spec × spec	1.29%	0.0172	$p \leq 0.001$
Residual σ^2	8.70%	0.1160	$F_{11782}^{105} \geq 1178$
$\hat{R}^2$ **of Model**	**0.9130**	**1.2179**	$p \leq 0.001$

Table G.10: ANOVA For Model Predicting **lof** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
lof	**0.9001**	**1.3384**	**2.4600**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	28.11%	0.3762	$p \leq 0.001$
pd ($\bar{\phi}$)	12.76%	0.1708	$p \leq 0.001$
–	9.23%	0.1235	$p \leq 0.001$
× parentset	3.54%	0.0474	$p \leq 0.001$
nc (ν)	9.82%	0.1315	$p \leq 0.001$
–	8.57%	0.1147	$p \leq 0.001$
× parentset	1.26%	0.0168	$p \leq 0.001$
fi (α)	3.97%	0.0532	$p \leq 0.001$
–	2.15%	0.0287	$p \leq 0.001$
× parentset	1.83%	0.0245	$p \leq 0.001$
rf (ρ)	**32.94%**	**0.4408**	$p \leq 0.001$
–	29.78%	0.3986	$p \leq 0.001$
× parentset	3.16%	0.0423	$p \leq 0.001$
spec × spec	1.91%	0.0255	$p \leq 0.001$
Residual σ^2	10.49%	0.1404	$F_{11782}^{105} \geq 957$
$\hat{R}^2$ of Model	**0.8951**	**1.1980**	$p \leq 0.001$

Table G.11: ANOVA For Model Predicting **abod** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
abod	**0.9046**	**1.2764**	**2.4709**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	22.82%	0.2912	$p \leq 0.001$
pd ($\bar{\phi}$)	12.04%	0.1537	$p \leq 0.001$
–	9.41%	0.1201	$p \leq 0.001$
× parentset	2.63%	0.0336	$p \leq 0.001$
nc (ν)	7.90%	0.1009	$p \leq 0.001$
–	4.57%	0.0583	$p \leq 0.001$
× parentset	3.34%	0.0426	$p \leq 0.001$
fi (α)	9.22%	0.1176	$p \leq 0.001$
–	6.92%	0.0884	$p \leq 0.001$
× parentset	2.29%	0.0293	$p \leq 0.001$
rf (ρ)	**31.06%**	**0.3964**	$p \leq 0.001$
–	26.12%	0.3334	$p \leq 0.001$
× parentset	4.93%	0.0630	$p \leq 0.001$
spec × spec	2.39%	0.0305	$p \leq 0.001$
Residual σ^2	14.58%	0.1861	$F_{11782}^{105} \geq 657$
$\hat{R}^2$ of Model	**0.8542**	**1.0903**	$p \leq 0.001$

Table G.12: ANOVA For Model Predicting **loda** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
loda	**0.8897**	**1.2547**	**2.4343**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	**29.81%**	**0.3741**	$p \leq 0.001$
pd ($\bar{\phi}$)	13.08%	0.1641	$p \leq 0.001$
–	8.04%	0.1009	$p \leq 0.001$
× parentset	5.03%	0.0632	$p \leq 0.001$
nc (ν)	13.04%	0.1637	$p \leq 0.001$
–	10.72%	0.1345	$p \leq 0.001$
× parentset	2.32%	0.0291	$p \leq 0.001$
fi (α)	1.81%	0.0227	$p \leq 0.001$
–	1.18%	0.0148	$p \leq 0.001$
× parentset	0.63%	0.0079	$p \leq 0.001$
rf (ρ)	**28.78%**	**0.3612**	$p \leq 0.001$
–	24.30%	0.3048	$p \leq 0.001$
× parentset	4.49%	0.0563	$p \leq 0.001$
spec × spec	1.88%	0.0236	$p \leq 0.001$
Residual σ^2	11.59%	0.1454	$F_{11782}^{105} \geq 855$
$\hat{R}^2$ of Model	**0.8841**	**1.1093**	$p \leq 0.001$

Table G.13: ANOVA For Model Predicting **iforest** (log(lift))

	$\mu(\log(\text{lift}))$	$\sigma^2(\text{real})$	$\mu'(\text{lift})$
iforest	**1.0498**	**1.5073**	**2.8571**
	$\sigma^2(\%)$	$\sigma^2(\text{real})$	F-test
parentset	31.58%	0.4760	$p \leq 0.001$
pd ($\bar{\phi}$)	13.71%	0.2066	$p \leq 0.001$
–	9.63%	0.1451	$p \leq 0.001$
× parentset	4.08%	0.0615	$p \leq 0.001$
nc (ν)	9.27%	0.1398	$p \leq 0.001$
–	8.05%	0.1213	$p \leq 0.001$
× parentset	1.23%	0.0185	$p \leq 0.001$
fi (α)	0.87%	0.0132	$p \leq 0.001$
–	0.38%	0.0057	$p \leq 0.001$
× parentset	0.50%	0.0075	$p \leq 0.001$
rf (ρ)	**36.81%**	**0.5549**	$p \leq 0.001$
–	32.65%	0.4921	$p \leq 0.001$
× parentset	4.17%	0.0628	$p \leq 0.001$
spec × spec	1.41%	0.0213	$p \leq 0.001$
Residual σ^2	6.34%	0.0956	$F_{11782}^{105} \geq 1656$
$\hat{R}^2$ of Model	**0.9366**	**1.4117**	$p \leq 0.001$

Table G.14: Algorithm Performance When Contrasting pd_1 and pd_3 (log(lift))

		log(lift)			
algo	$\mathrm{CI}^{\mathrm{algo}}_{0.999}$	pd_1	pd_3	$\mathrm{pd}_3 - \mathrm{pd}_1$	$\mathrm{CI}^{\mathrm{pd}}_{0.999}$
tmd		0.1710	-0.0988	**-0.2699**	±0.0574
egmm	±0.0389	0.3861	-0.0635	-0.4496	±0.0574
rkde	±0.0389	0.3495	0.0666	-0.2830	±0.0574
ocsvm	±0.0389	0.3175	-0.1087	-0.4262	±0.0574
svdd	±**0.0389**	0.1810	-0.1047	-0.2857	±0.0574
knn	±0.0389	0.5171	**0.1232**	-0.3939	±0.0574
lof	±0.0389	0.3781	-0.0914	-0.4695	±0.0574
abod	±0.0389	0.3912	-0.0003	-0.3915	±0.0574
loda	±0.0389	0.3634	-0.0389	-0.4023	±0.0574
iforest	±0.0389	**0.5889**	-0.0195	**-0.6084**	±0.0574

Table G.15: Algorithm Performance When Contrasting nc_s and nc_c (log(lift))

		log(lift)			
algo	$\mathrm{CI}^{\mathrm{algo}}_{0.999}$	nc_s	nc_c	$\mathrm{nc}_c - \mathrm{nc}_s$	$\mathrm{CI}^{\mathrm{nc}}_{0.999}$
tmd		0.1936	-0.6536	-0.8472	±0.0717
egmm	±0.0560	0.2645	-0.3349	-0.5993	±0.0717
rkde	±**0.0560**	0.1894	-0.1521	-0.3414	±0.0717
ocsvm	±0.0560	0.4409	-0.6861	**-1.1269**	±0.0717
svdd	±**0.0560**	0.1613	-0.6210	-0.7823	±0.0717
knn	±0.0560	0.3694	0.0009	-0.3685	±0.0717
lof	±0.0560	0.3259	-0.3166	-0.6425	±0.0717
abod	±0.0560	0.1337	**0.0482**	**-0.0854**	±0.0717
loda	±0.0560	0.2577	-0.3749	-0.6327	±0.0717
iforest	±0.0560	**0.4762**	-0.4065	-0.8827	±0.0717

Table G.16: Algorithm Performance When Contrasting fi_0 and fi_3 (log(lift))

		log(lift)			
algo	$\text{CI}^{\text{algo}}_{0.999}$	fi_0	fi_3	$\text{fi}_3 - \text{fi}_0$	$\text{CI}^{\text{fi}}_{0.999}$
tmd		-0.1261	-0.3972	-0.2711	±0.0476
egmm	±0.0414	0.2199	-0.5136	-0.7335	±0.0476
rkde	±0.0414	0.1907	-0.3898	-0.5805	±0.0476
ocsvm	±0.0414	-0.0640	-0.2069	**-0.1429**	±0.0476
svdd	**±0.0414**	-0.1155	-0.4163	-0.3007	±0.0476
knn	±0.0414	**0.3585**	-0.3135	-0.6720	±0.0476
lof	±0.0414	0.0913	-0.3630	-0.4543	±0.0476
abod	±0.0414	0.2917	-0.5645	**-0.8562**	±0.0476
loda	±0.0414	0.0240	-0.2966	-0.3205	±0.0476
iforest	±0.0414	0.1588	**-0.0835**	-0.2423	±0.0476

Table G.17: Algorithm Performance When Contrasting rf_1 and rf_5 (log(lift))

		log(lift)			
algo	$\text{CI}^{\text{algo}}_{0.999}$	rf_1	rf_5	$\text{rf}_5 - \text{rf}_1$	$\text{CI}^{\text{rf}}_{0.999}$
tmd		2.3438	-0.0458	-2.3896	±0.0876
egmm	±0.1125	2.7192	0.0101	-2.7091	±0.0876
rkde	**±0.1125**	2.4141	0.0442	-2.3699	±0.0876
ocsvm	±0.1125	**3.0606**	-0.0638	**-3.1245**	±0.0876
svdd	±0.1125	2.2258	-0.0405	**-2.2664**	±0.0876
knn	±0.1125	2.9788	0.1051	-2.8737	±0.0876
lof	±0.1125	2.7266	-0.0450	-2.7716	±0.0876
abod	±0.1125	2.4630	0.0028	-2.4602	±0.0876
loda	**±0.1125**	2.4499	0.0462	-2.4037	±0.0876
iforest	±0.1125	2.9921	**0.1243**	-2.8678	±0.0876