Thesis

A Framework for Real-time, Autonomous Anomaly Detection over Voluminous Time-Series Geospatial Data Streams

Submitted by

Walid Budgaga

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2014

Master's Committee:

Advisor: Shrideep Pallickara
Co-Advisor: Sangmi Lee Pallickara,

Asa Ben-Hur
Russ Schumacher

Abstract

A Framework for Real-time, Autonomous Anomaly Detection over Voluminous Time-Series Geospatial Data Streams

In this research work we present an approach encompassing both algorithm and system design to detect anomalies in data streams. Individual observations within these streams are multidimensional, with each dimension corresponding to a feature of interest. We consider time-series geospatial datasets generated by remote and in situ observational devices. Three aspects make this problem particularly challenging: (1) the cumulative volume and rates of data arrivals, (2) anomalies evolve over time, and (3) there are spatio-temporal correlations associated with the data. Therefore, anomaly detections must be accurate and performed in real time. Given the data volumes involved, solutions must minimize user intervention and be amenable to distributed processing to ensure scalability.

Our approach achieves accurate, high throughput classifications in real time. We rely on Expectation Maximization (EM) to build Gaussian Mixture Models (GMMs) that model the densities of the training data. Rather than one all-encompassing model, our approach involves multiple model instances, each of which is responsible for a particular geographical extent and can also adapt as data evolves. We have incorporated these algorithms into our distributed storage platform, Galileo, and profiled their suitability through empirical analysis which demonstrates high throughput (10,000 observations per-second, per-node) and low latency on real-world datasets.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

The primary focus of this work is the detection of anomalous data in multidimensional datasets. An anomaly may constitute an irregular event, inconsistent sensor readings, or other types of situations that result in data points outside of the expected norm. These points are n-dimensional tuples with each dimension representing a feature of interest. Examples of such features include temperature, pressure, humidity, etc. Features may also have linear or non-linear relationships with each other; for instance, there may be a relationship between temperature and precipitation at certain geographic locations. Ultimately, these relationships result in a classification of either *anomalous* or *normal* by our framework.

The datasets we consider are composed of streams that continually generate readings from observational devices. Some of these observational devices, such as radars and satellites, can remotely sense features of interest while other features may require in situ measurements by devices such as piezometers and barometers. The measurements are reported as observations in discrete packets that comprise a stream.

This thesis presents a framework for scalable and real time detection of anomalies in streaming data. Anomaly detection is a precursor to the discovery of impending problems or patterns of interest. Timely detection of anomalies is critical in several settings. Often such detection needs to be made in real time to be able to detect potential emergencies. Our specific problem relates to voluminous data streams and anomaly detection that accounts for evolution of the feature space over time. Also, given the rate of data arrivals and the volumes involved, human intervention is rendered infeasible.

The range of values that each feature takes may be rather large and simple checks for breaching upper and lower bounds are generally not viable. Other dimensions such as location and time may determine whether a particular feature value is considered anomalous. For example, temperatures at nighttimes are often lower than daytimes for a particular location. Also, in the case of geospatial data, what is considered normal will vary by region and anomaly classifications must account for this as well.

Given the data volumes involved, observations must be stored over a collection of resources. However, disk I/O operations should also be reduced so as to not preclude real time classification. In-memory data structures must be compact to avoid page-faults and thrashing that may occur. To cope with scaling issues, viable solutions must be able to take advantage of increases in the number of resources available to the system.

1.1. Research Challenges

There are several challenges involved with the proposed research:

(1) Streams have no preset lifetimes, and readings arrive continually. The data can thus be voluminous, making it infeasible to inspect all previous records when making a classification.

(2) Multidimensional data. Individually, feature values (i.e.values along a dimension) may be *normal*, but when collectively accounting for all the dimensions, the tuple may be *anomalous*.

(3) There are spatial and chronological dimensions associated with feature values. How features evolve is spatiotemporally correlated. What is considered *normal* for a particular geographical extent would be considered *anomalous* for another. Anomaly classifications must take these properties into account.

(4) What is considered anomalous evolves over time. A good exemplar of this is temperature readings. Over the past several years, average temperatures at various geographic locations have been trending higher overall but fluctuate from year to year.

(5) The combination of legitimate feature values is very large. Building a singular model of what constitutes anomalous data is infeasible and impractical.

(6) Anomaly detection must be done in real time despite the constant feature space evolution and data volumes involved.

1.2. Research Questions

In this research we explore the following research questions:

(1) Given that human intervention is infeasible in this work, how can we adapt autonomously to changes in the data?

(2) Since classifications will evolve over time, how can we autonomously tune the detection thresholds?

(3) How can we achieve timeliness without compromising the accuracy of the classifications?

(4) How can we account for spatio-temporal properties associated with features, and what are the implications for system design in such cases?

(5) How can we scale to assimilate a large number of machines?

1.3. Approach Summary

Our approach provides a configurable framework for anomaly detection over continuous data streams. Since anomalies evolve over time, our approach focuses on online adaptation

of our models. The framework allows for domain-specific behavior to handle changes in the data that must be treated as *normal* rather than *anomalous*, a feature that can be turned on or off at any time.

We rely on a probabilistic model to detect anomalies. Specifically, we use Expectation Maximization (EM) to build Gaussian Mixture Models (GMMs) that model the densities of the training data by using different combinations of Gaussian distributions. EM is an iterative method and aims to maximize log-likelihood by modifying GMMs parameters. Within each iteration, the log-likelihood is improved by fitting Gaussian distributions to the given data. The iterative process is stopped when no significant improvement can be achieved. EM improves a given model but does not build it from scratch, so we use K-means and K-fold cross-validation to build an initial model that can be used by EM. Also, in order to support continuous adaptation, the GMMs parameters are modified in online fashion to capture any changes in data behavior.

To deal with spatial properties in the dataset, we partition the incoming streams based on geographical extents. We then initialize an instance of our anomaly detection model for the geographic regions, each of which continually adapts based on observations recorded for the region. Depending on the data volumes involved and the number of resources available within the system, geographic boundaries and the number of model instances can be tuned.

In our approach, the determination of whether an observation is *anomalous* or not does not require specification of thresholds by the user. Observations are tagged as *normal* or *anomalous* based on their fitness to the underlying models and their locations within a d-dimensional space. Two factors determine whether an observation is *anomalous* or not: *fitness scores* and *distance measures*. The fitness score represents how well an observation fits the current model. Its calculation produces positive values for *normal* observations and

negative values for *anomalous* or suspect observations. The distance measure is calculated based on the distance between the observation and the cluster centroid to which the observation was assigned. Distance measures are normalized so that observations that are close to a cluster centroid will have distance values less than one and that those that exist outside a cluster will have values greater than one. The product of the distance and fitness scores associated with an *anomaly* will always have a value less than -1.

Our proposed framework is decentralized, scalable, and capable of achieving high throughput. We incorporated our anomaly detection implementation into the Galileo [1], [2] data storage system. Galileo provides support for multidimensional, time-series geospatial datasets, and is organized as a distributed hash table (DHT). Each storage node in Galileo manages a geographic subset of the incoming data streams, and passes the information on to the anomaly detection framework.

Each instance of the anomaly detector tunes itself autonomously based on the data distributions it has observed. The instances are executed in a thread pool, which allows us to calibrate the degree of concurrency in the system to better exploit available cores and execution pipelines. Our performance benchmarks show that each node in the system can evaluate more than 10,000 data stream packets per second.

1.4. Thesis Contributions

This thesis describes our approach for detecting anomalies in large, multi-dimensional datasets. The work presented herein includes the following contributions:

(1) Our approach scales with increases in data volumes and the number of machines available.

(2) Our approach efficiently exploits the available distributed computational capabilities without producing hot spots.

(3) Our model instances tune themselves autonomously based on observations. This allows us to account for spatio-temporal evolution of the feature space. Our anomaly detections are thus specific to particular geographical extents.

(4) Since we support model instances that tune themselves autonomously for different geographical regions, partitioning of workloads in a distributed setting is straightforward. The fact that these model instances do not interact with each other and classify observations independently also serves to ensure scalability.

(5) Since model instances tune themselves autonomously based on the data available to them, we can cope with variability in the density and availability of readings from different geospatial locations.

(6) We can fine tune the specificity of the classifications by controlling the geographical scope associated with the classification models.

Our empirical evaluations with diverse datasets and across different volumes of information confirm the suitability of our approach. Our benchmarks also demonstrate that we can perform these classifications in real time, processing 10 observations per millisecond (or 10,000 observations per second) at each node.

1.5. Thesis Organization

This thesis is organized as follows. The next Chapter provides background information on the models and methods used in this work, followed by Chapter 3 with an overview of the design of the system itself. Chapter 4 explains our online adaptive anomaly detection

algorithm, followed by Chapter 5 which includes extensive performance evaluation and experimental results. Finally, Chapter 6 reviews related work from the literature, and Chapter 7 concludes the thesis.

CHAPTER 2

Background Information

To provide some intuition on how our adaptive Gaussian Mixture Models with Expectation Maximization strategy works, we present a brief overview of the algorithms and techniques used to derive the desired functionality [3]

2.1. Multivariate Data

In many applications, multivariate data could be defined as N samples or events. Each individual sample represents measurements that form an observation vector containing d elements. The elements in vector are often called features, readings, or attributes. The N samples can be seen as N individual identically distributed observations.

In a climate dataset, for example, each sample could contain readings taken at particular time in a particular location. These readings could be surface temperature, air pressure, relative humidity, and so forth.

In some applications, multivariate analysis is needed to represent the data in a small number of parameters, which are typically the means and the variances (or covariance matrix). N samples of d readings can be represented by two vectors of size d, where one contains the means $(\mu_1, \mu_2, .., \mu_d)$ of each feature and the another contains the variances $(\sigma_1, \sigma_2, .., \sigma_d)$. In case of clustering the data into groups, mixture model could be used.

2.2. Gaussian Mixture Model

The Gaussian Mixture Model(GMM) is a parametric probability method to model a density of observations using a combination of different Gaussian component densities [4].

The assumption made in GMM is that the observations X are generated by a finite number of Gaussian distributions. Each Gaussian density $\mathcal{N}(X|\mu_k, \Sigma_k)$ is a component of the mixture and formed by mean μ_k and covariance matrix Σ_k [5].

The GMMs are frequently used as a classification method in supervised learning problems or as a clustering technique in unsupervised learning. Each Gaussian component can be used to form a cluster (group) of data in the d-dimensional space and then the appropriate component for new data samples can be determined using a Bayesian classifier. In order to place data samples into their respective clusters, the probability of membership to each Gaussian component is calculated, and the component with the highest probability will be responsible for the sample.

$$max(P(\theta_k|x))$$

Where

$P(\theta_k|x)$ is the probability of data sample x belonging to component k

x is a given data sample that consists of d features

θ_k represents the parameter(μ_k, Σ_k) of the Gaussian component k

$k = 1, 2, 3,..,K$

K is the number of Gaussian components(clusters)

The computation of $P(\theta_k|x)$ ultimately determines which component will be responsible for each sample x. Based on Bayes Rule [5] the conditional probability $P(\theta_k|x)$ can be computed from the equation:

$$P(\theta_k|x) = \frac{\pi_k P(x|\theta_k)}{P(x)} \tag{2.1}$$

Where

π_k is a mixing coefficient of the component k which can be computed by:

$$\pi_k = \frac{N_k}{N} \tag{2.2}$$

Where N_k is the sum of the observations' properties of belonging to Gaussian component k.

And satisfies the conditions:

$$0 \leq \pi_k \leq 1 \tag{2.3}$$

$$\sum_{j=1}^{K} \pi_j = 1 \tag{2.4}$$

The probability density function (PDF) of x across the Gaussian components can be computed as follows:

$$P(x) = \sum_{j=1}^{K} \pi_j P(x|\theta_j) \tag{2.5}$$

Each $P(x|\theta_j)$ expresses the probability that the data sample x is generated using the Gaussian component j, which has its own mean μ_j and covariance Σ_j.

By substituting $P(x)$ in the Formula (2.1) with its corresponding expression in Formula (2.5), we can rewrite the Formula (2.1):

$$P(\theta_k|x) = \frac{\pi_k P(x|\theta_k)}{\sum_{j=1}^{K} \pi_j P(x|\theta_j)} \tag{2.6}$$

Since the Gaussian Mixture assumes that the data is a Gaussian distribution with mean μ_k and covariance Σ_k, then the d-dimensional normal distribution ($P(x|\theta_k)$) can be computed as follows:

$$P(x|\theta_k) = \frac{1}{2\pi^{d/2}|\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)} \tag{2.7}$$

Where mean μ_k is a d-dimensional column vector and the covariance matrix Σ_k is a $d \times d$ symmetric matrix.

Once the $P(x|\theta_k)$ is computed for each Gaussian component k using equation 2.7, the results can be used to solve Formula (2.6) and find $P(\theta_k|x)$. Having calculated $P(\theta_k|x)$ for a given sample x, x's appropriate cluster is that which is equivalent to the Gaussian component with the largest value from $P(\theta_k|x)$.

2.3. Maximum Likelihood of GMM

Suppose we have N observations represented in $N \times d$ matrix $X = x_1, x_2, .., x_N$. Each of its rows x_i represents one observation, which is a vector containing d features. The observations are modeled using mixtures of Gaussians that can be defined with a density function $p(X|\theta)$ where θ represents the means μ and the covariances Σ of K Gaussian distributions. By assuming that the data vectors (samples) $x_1, x_2, .., x_N$ are independently drawn from identical distribution, then the likelihood function $\ell(\theta|X)$ which is the density of the data samples is given by:

$$\ell(\theta|X) = p(X|\theta) = \prod_{i=1}^{N} p(x_i|\theta) \tag{2.8}$$

In the Formula (2.8) $\ell(\theta|X) = p(X|\theta)$, we can state that the likelihood of K parameter values, θ, given N samples is the product of the probabilities of those observed samples given the K parameter values, θ. By applying different parameter values θ to the same samples X, different likelihood values are obtained. Each likelihood value expresses how probable X is for the used Gaussians' parameters θ [5]. By comparing the different likelihood values obtained, we can distinguish which model's parameters θ (μ , Σ) best represent the data; the best model's parameters θ produce the largest likelihood value. Therefore, maximizing the values of $\ell(\theta|X)$ leads to the best Gaussian parameters that represent the observed samples. Then, our aim is to find θ that maximizes $\ell(\theta|X)$, which we find with:

$$\underset{\theta}{\operatorname{argmax}}(\ell(\theta|X)) \tag{2.9}$$

Since the natural logarithm is monotonically increasing function, then the parameters θ that maximize $\ell(\theta|X)$ are the same values that maximize $log(\ell(\theta|X))$. In the most applications involving likelihood functions, the natural logarithm of the likelihood function, called the log-likelihood, is used in place of likelihood function because it is easier to analyze and often more convenient to work with.

In the machine learning community, it is known that the negative log of the likelihood function can be used as an error function. The value of the error function can be minimized by maximizing the log-likelihood value. In other words, choosing of parameters that maximize the log-likelihood reduces the value of the error function [5]:

$$\underset{\theta}{\operatorname{argmin}}(-log(\ell(\theta|X))) = \underset{\theta}{\operatorname{argmax}}(\ell(\theta|X)) \tag{2.10}$$

Finding the desired parameters depends on the form of the Gaussian mixture distributions. While the values of the parameters can be found easily in single Gaussian distribution by setting the derivative of $log(\ell(\theta|X))$ to zero and solving the formula for μ and Σ, it more complex for mixtures of Gaussian distributions because finding such a solution would not be possible. One powerful and widely-used solution to maximize the log-likelihood is a general iterative technique, called expectation maximization [6] [5].

2.4. Expectation Maximization (EM) for GMM

Expectation Maximization(EM) is iterative refinement technique used for estimating maximum likelihood of parameters in probabilistic models. For Gaussian Mixture models, EM is used to find the best means μ and covariance matrix Σ, representing Gaussian distributions (Gaussian components) in Mixture for a given set of multidimensional data samples [6].

The conventional EM algorithm for GMM starts with initial values of $\mu = \mu_1, \mu_2, .., \mu_k$ and $\sigma = \sigma_1, \sigma_2, .., \sigma_k$ for k Gaussian components and iteratively performs two steps, *expectation* (**E**) and *maximization* (**M**). The algorithm is complete when a difference between the last and the current likelihood value is zero or close to zero (less than user defined value).

In the expectation (**E**) step, the current estimate of the parameters (μ, σ) will be used in Formula 2.1 to compute how much each Gaussian component is responsible for each sample in the training data. This process will produce a matrix of size $N \times K$, where N is the number of the samples used to train the model and K is the number Gaussian components. The i^{th} row in this matrix will contain a vector of size K that includes the responsibilities of each Gaussian component for the i^{th} sample. This also means that each data sample in the training set will have its own weights vector containing the membership weights that

show the sample's likelihoods of belonging to each Gaussian component. The responsibility of the Gaussian component k for the data sample x can be denoted by $\gamma_k(x)$ and computed through $\gamma_k(x) = P(\theta_k|x) = \frac{\pi_k P(x|\theta_k)}{P(x)}$

In the maximization (**M**) step, the membership weights and the data samples are used to compute new estimates of likelihood parameters (μ, σ) for each Gaussian component. The new value of mean and covariance matrix for each Gaussian component k can be computed through:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_k(x_i) x_i \tag{2.11}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_k(x_i)(x_i - \mu_k)(x_i - \mu_k)^T \tag{2.12}$$

Where N_k represents the number of samples assigned to Gaussian component k and can be computed as:

$$N_k = \sum_{i=1}^{N} \gamma_k(x_i) \tag{2.13}$$

Our implementation of EM is based on the EM implementation of WEKA software [7]. Algorithm 1 shows a high level pseudo code of the standard EM.

2.5. K-Means

The K-means algorithm is a popular clustering method that assigns n data samples into k clusters. Samples are clustered based on their proximity to cluster centroids, and most implementations of the algorithm use hard assignment, where each sample belongs to only one cluster. The algorithm starts with an initial set of clusters,and their centroids are updated as new samples arrive. This also means that samples may be reassigned to different clusters over time.

Algorithm 1 The standard EM algorithm for GMM

initialize Gaussian components
repeat
/*E-Step: use (μ_k, σ_k) of each Gaussian component k to compute membership weights of each sample for all clusters*/
for (each sample i in the training set) **do**
for (each cluster k) **do**
compute $weights_i[k]$ using x_i, μ_k, and σ_k
end for
end for
/*M-Step: use the data samples and their membership weights to compute new Gaussian parameters (μ_k and σ_k) for each cluster k*/
for (each cluster k) **do**
for (each sample i in the training set) **do**
compute μ_k and σ_k using x_i and $weights_i[k]$
end for
end for
until (log-likelihood of the estimated parameters converges)

The algorithm continues until no changes occur in the clusters, with the resulting set of clusters minimizing the distance between related samples. For this purpose, the K-means algorithm attempts to minimize the value of its objective function, the minimum squared sum of Euclidean distances:

$$argmin \sum_{i=1}^{k} \sum_{j=1}^{n} (x_j - c_i)^2$$

Where:

x_j is the sample j that belongs to the cluster i and c_i is the mean of the cluster i.

Despite the simplicity of K-means algorithm, its objective function is NP-complete problem [8]. Also, specifying the correct number of clusters needed before the algorithm is applied can often prove to be problematic.

2.6. K-Fold Cross Validation

In both data mining and machine learning, cross-validation is widely used to evaluate a learned model's performance. K-fold cross-validation randomly partitions the data being used by the model into k equally-sized segments. One of the segments is chosen to be the validation set for testing the model, while the remaining segments are used as training data. The process is repeated k times, with each segment being used as the validation set once. Next, the resulting error values are averaged to provide an overall evaluation of the model's performance. One disadvantage associated with K-fold cross-validation is the model must be built k times, where k is commonly 10 for machine learning applications [9]. In this work, K-means and K-fold cross validation are used to bootstrap the EM algorithm with an initial model.

CHAPTER 3

System Overview

Our anomaly detection system was designed to be integrated with a distributed storage framework to exploit parallelism and ensure scalability. For this study, we used Galileo [1], [2], which was tasked with supplying new data to the anomaly detector as it was streamed into the system for storage. When an anomaly is found, offending records are flagged for further analysis. Figure 3.1 illustrates the integration of the detection framework and Galileo.

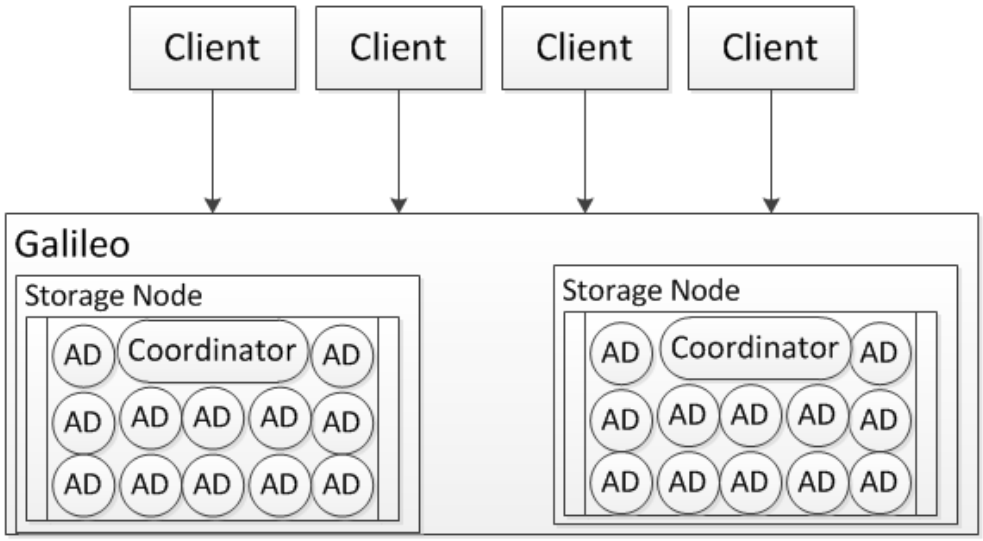

Figure 3.1. Anomaly detection framework integration with Galileo. Components include the storage nodes, detection coordinators, and anomaly detectors (AD).

3.1. Galileo

Galileo is a high-throughput distributed storage framework designed for managing multidimensional data. The systems network design is modeled as a hierarchical distributed hash table (DHT), which allows incremental assimilation of storage resources and the use of multi-tiered hash functions to enable development of novel partitioning schemes. By focusing

on spatio-temporal datasets, Galileo provides functionality that is generally not provided by standard DHTs, such as expressive query support [2], time series analysis capabilities, and polygon- or proximity-based geospatial retrieval functions [10]. Galileo is completely decentralized and composed of a network of storage nodes, which facilitate data management.

3.2. Storage Node

Each storage node in Galileo manages a single anomaly *detection coordinator* that is used to classify incoming observations in an online manner. Based on the classification output by the framework, the storage node can take a appropriate action which may vary across problem domains. The storage node does not need to wait for the result of each evaluated observation, and will assume the observations are normal until informed otherwise.

The storage node considers the detection coordinator to be a black box, but can control some of its behavior. It cooperates with the coordinator asynchronously to detect anomalies without knowledge of the particular detection algorithm being used. In addition to starting or stopping the coordinator, the storage node can ask the coordinator to turn its adaptive behavior on or off. It can also change the adaptation rate to control the speed of the adaptation process. Moreover, the storage node can modify the size of the geospatial area the coordinator will assign to each anomaly detector instance.

3.3. Geospatial Data Partitioning

Data partitioning in Galileo is done based on the observed Geohash [11] prefixes of incoming records. Figure 3.2 shows how a destination node can be specified based on Geohash computed from an observation's latitude and longitude. The Geohash algorithm is a geocoding scheme that divides the earth into a hierarchy of spatial bounding boxes referenced by

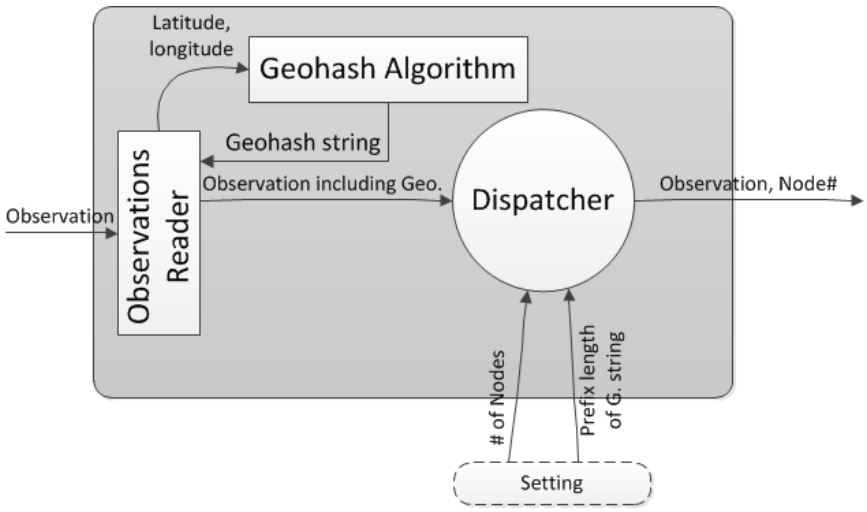

FIGURE 3.2. Data partition based on Geohash.

Base-32 strings. For instance, the latitude and longitude of coordinates of N 28.8927, W 81.9796 would map to the Geohash string *djjsqeb2*. Longer strings result in higher precision coordinates, and two Geohash strings with the same length 6 prefixes will be located in a similar geographic region. The prefix length used for partitioning is configurable, and specifies the size of the regions that will be processed by an anomaly detector instance. The default prefix length is 4, which indicates that each anomaly detector will be assigned to an area with a size of 39.1 km x 19.5 km.

This partitioning method allows us to control the number of anomaly detector instances, the amount of data processed by an instance, and the size of the regions involved. As a result, classification accuracy can be increased by lengthening the prefix string used for partitioning. This decentralized strategy also provides independence between detector instances, increasing classification throughput, scalability, and performance.

3.4. Detection Coordinator

The main job of the *detection coordinator* is creating and managing anomaly detector instances. The coordinator receives observations from the storage node and then forwards

them to the appropriate detector based on the Geohash associated with each observation, or creates a new instance if one did not already exist. Figure 3.3 shows the interactions between coordinator and anomaly detectors.

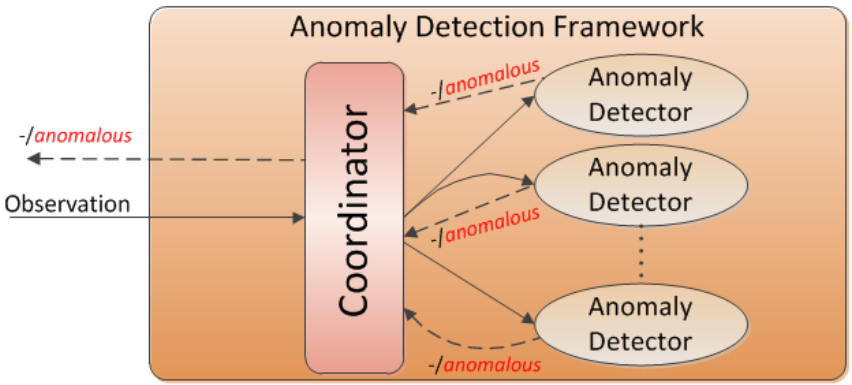

FIGURE 3.3. The coordinator at a node is responsible for orchestrating anomaly detection.

Each anomaly detector created by the coordinator implements an interface that enables it to use an independent anomaly detection algorithm, if necessary. Because of the loose coupling between the coordinator and detectors, any changes made to either component will not require corresponding changes in the other. The detector implementation uses a Gaussian Mixture Model (GMM) built using the Expectation Maximization (EM) technique to detect anomalies. More information about the implementation is provided in Chapter 4.

The coordinator can run in one of two modes: training or classification mode. In training mode, an anomaly detector instance will be created when the coordinator receives an observation whose Geohash prefix has not yet been seen. In classification mode, the coordinator will create one anomaly detector instance for each model. Once the models have been loaded and their instances have been created, incoming observations will be passed to appropriate instance based on Geohash strings.

When the coordinator starts, it creates and launches a thread pool with a configurable size. Each instance will receive a reference to the thread pool from the coordinator in order to submit anomaly detection tasks. Additionally, the coordinator may receive an administrative command from a client that will start the training process at any anomaly detector that has not yet created a model. Finally, based on instructions provided by the storage node, the coordinator can turn the adaptation feature on or off for particular detector instances at any time.

3.5. Anomalies Detector

The primary task of the anomaly detector is to accept data from a particular geospatial area, build a model for it, and then use the model to detect samples whose behaviors are outside the observed norm. A general interface is used for implementing both the anomaly detection algorithm and interacting with the coordinator, which helps ensure a loose coupling of the components in the system.

The detector supports several features that can be accessed by the coordinator. First, it has the ability to detect anomalies in an online manner, wherein observations are evaluated on the fly without keeping them all in main memory. Further, the detector can adapt to natural changes that occur over time. For example, temperature values at a particular region that were unusual thirty years ago may be considered normal in recent years. In such a case, the detector should adapt to these changes and not deem such data anomalous. However, the detector also offers the ability to turn this feature off to handle situations where adaptation is not beneficial; for example, heart rate measurements do not need to be adapted because they will not change over time. Finally, the detector can serialize and deserialize its model

to and from disk to allow migration of the model to other systems and to cope with failures or planned outages.

Each anomaly detector operates in three phases. In the *collecting phase*,the detector will store any samples received in a buffer until it reaches a specified size or receives a command from the coordinator to begin training. When a training command has been received or the required amount of training data has been collected, the detector will transition to the *training phase*, where no further inspection of the buffer will be performed. In the training phase, the model for anomaly detection is built and trained. While the model is training, any new samples will be placed in a *classification queue*. Once the model is ready, it will be serialized to disk for future use and the detector will transition to its final phase, *classification*. In the classification phase, the model is used to classify any observations that arrive in the classification queue.

3.6. Anomalies: Group and Experts Approach

The anomaly detector instances managed by the coordinator at a storage node work independently on different geospatial data, but can also cooperate with one another to achieve a consensus in situations where classifications may be suspect. Since observations from bordering geospatial regions should have similar behavior, instances operating on nearby regions are consulted to confirm suspect classifications. If a detector instance receives an outcome indicating a suspect anomaly, it informs the coordinator, which forwards the suspect outcome to detector instances in nearby geographic regions. In this situation, the observation is considered anomalous if a majority of detectors confirm the anomaly.

Using different detection algorithms is another scenario in which a group of experts approach proves to be useful. In such a case, the coordinator passes data samples to multiple

anomaly detectors that implement different anomaly detection algorithms. The final classification for the observation is then made based on the results coming from each anomaly detection instance, where the observation is deemed anomalous only if more than the half of the instances confirm the anomaly.

3.7. Parallel Anomaly Detection: The Thread Pool

Our approach ensures available cores and execution pipelines are used efficiently by managing a thread pool to facilitate parallel detection activities. During initialization, the coordinator creates a thread pool of a configurable size, and provides each anomaly detector instance with a reference to the pool. Since the detectors maintain a classification queue, items in the queue can be used to create thread pool tasks for execution.

The thread pool will not run conflicting tasks because of the separation of processing concerns and data partitioning orchestrated by the coordinator. Each anomaly detector receives one sample at a time, and all operations pass through the coordinator. However, the actions taken by the coordinator are lightweight so as to not become a processing bottleneck and ensure the thread pool will provide a net performance gain.

3.8. Storage Framework Decoupling

To enable our anomaly detection scheme to be applied in a broad range of problem domains, we designed interfaces that were not strongly coupled to the underlying storage system. Integrating the anomaly detector with a storage framework requires only two key features: (1) event- or trigger-based functionality to submit new data points to the detector as they are stored, and (2) the capability to flag records as anomalous during post-processing.

These requirements essentially create a low-latency feedback loop that avoids expensive polling operations.

MongoDB [12] is a distributed document store that supports clustering and horizontal scalability through sharding. The system also provides range query functionality and can orchestrate MapReduce computations across computing resources. While MongoDB does not provide pre-storage processing hooks or the triggers often featured in relational databases, it does maintain an *oplog* that is updated during each storage event. Combined with a *tailable cursor* instance, which enables the latest changes to a data collection to be read, new information can be immediately passed to the anomaly detector as it is stored. When an anomaly is detected, the relevant document in MongoDB is flagged by updating its associated record. MongoDB is particularly effective in use cases that involve JSON, JavaScript Object Notation, or JSON-like documents; while integrating it with the anomaly detector, we simply converted files in our NOAA, National Oceanic and Atmospheric Administration, dataset from NetCDF to JSON and then streamed the records into the system.

CHAPTER 4

Online Adaptive GMM and EM

This section introduces the online adaptive algorithm that is used by anomaly detectors to adapt GMMs' parameters to the changes occurred in data and to classify the data streams received by Galileo.

4.1. Adaptive Algorithm

Since we are dealing with continuous data streams, our goal is to allow adaptive behavior without affecting performance. This is achieved through a configurable *adaptation speed* that can be changed at run time. The adaptation speed determines how often cluster parameters should be updated. There are two constraints to be considered during the update process: (1) only the cluster densities and their locations can be changed, and (2) each cluster should be affected based on its density and *responsibility fraction*.

To meet constraint (1), only the mixing coefficient (π_k) and the mean (μ_k) of the clusters are updated. For constraint (2), we use the responsibility fraction $\gamma_k(x)$ of each cluster for new data samples x_i to update cluster parameters (π_k,μ_k, N_K). In the adaptation process, we solve Equation 2.11 for sum_k ($\sum_{i=1}^{N} \gamma_k(x_i)x_i$) to recompute the previous sum of data samples in each cluster k using the previous N_k and mean μ_k.

$$sum_k = \mu_k * N_k$$

Include new observation x_i in the sum:

$$sum_k = sum_k + \gamma_k(x_i) * x_i$$

Compute new N_k :

$$N_k = N_k + \gamma_k(x_i)$$

Compute new mean μ_k :

$$\mu_k = \frac{sum_k}{N_k}$$

Compute the new mixing coefficient (π_k):

$$\pi_k = \frac{N_k}{N}$$

4.2. Anomaly Detection without Threshold Specification

We designed our anomaly detection framework with three primary goals in mind: (1) real time classification, (2) avoiding human intervention, and (3) domain neutrality, allowing the system to be used for different problems without extensive adjustment. While achieving real time classification is largely a function of the efficiency of our algorithms, avoiding user-defined thresholds was a key factor in achieving goals (2) and (3).

Choosing thresholds to enable efficient classification of normal or anomalous data can often be challenging [13], [14]. For instance, certain classes of problems may exhibit edge cases that cannot be easily captured with a threshold. Additionally, many applications must cope with a fluid definition of abnormality, requiring an infeasible amount of thresholds to be specified by the user given the data volumes being dealt with. Even if an optimal threshold can be found offline, it may not remain valid as the incoming data evolves. Finally, threshold specification would require domain experts to provide input for each specific application our framework was applied to.

Statistically, log-likelihood is a good measurement of how likely a model matches an observation. A higher log-likelihood value means the likelihood of a match is higher. However, the log-likelihood alone varies based on the particular features being inspected and their densities. This means that log-likelihood is a good fit for situations where it can be used as a point of comparison.

The log-likelihood of a GMM is the average of the log-likelihood values observed in the training data. Since the training data represents features with normal behavior, the log-likelihood of the GMM could be used as a reference for normal observations and is helpful when specifying a baseline value that could be used when deciding whether an observation is anomalous or not. Observations with log-likelihood values that are equal or greater to the GMM's loog-likelihood will not be tagged as anomalous.

Even if a baseline threshold has been found, comparing the log-likelihood of an observation with the threshold will not lead to accurate outcomes for all cases because the log-likelihood of the observation is affected by cluster densities and positioning in the d-dimensional space. Figure 4.1 contains a visualization of 8 clusters that were built for two-dimensional data using GMM and EM. Visually, points a and b lie within the range of normal observations, with point c representing a potential anomaly. However, consider a threshold of -2.50 for classification: $\ell(a) = 0.22$ would be considered normal, while $\ell(b) = -2.94$ and $\ell(a) = -2.94$ would be classified as anomalous. To correct the classification of point b, the threshold could be adjusted to -2.95, but this would cause observation c to also be classified as normal. This phenomenon occurs because c is near clusters with a higher density than those near b, resulting in both points having the same log-likelihood.

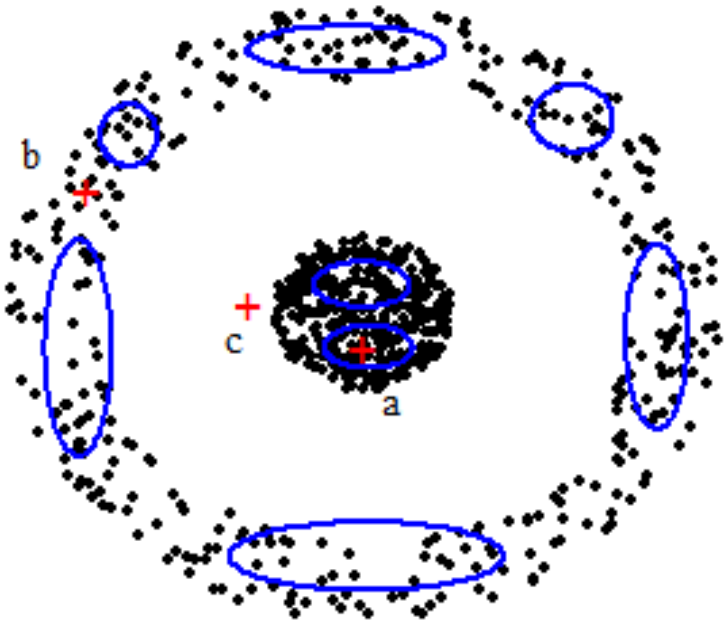

FIGURE 4.1. GMM built on synthetic data with 8 clusters. The log-likelihoods for the GMM and points a, b, and c are -1.73, 0.22, -2.94, and -2.94, respectively

To overcome this problem, we must not only compare the log-likelihood of observations, but also consider their distance from their assigned cluster centroids. The values associated with the observations should increase as their distance to the cluster centroids decrease, and vice versa. Additionally, this process should only affect suspect observations with log-likelihood values smaller than the models log-likelihood. Our proposed solution involves computing *fitness scores*, f_{score}, using the following form:

$$f_{score} = \frac{obs_{llk} - model_{llk}}{|model_{llk}|}$$

Where

obs_{llk} is the log-likelihood of an observation

$model_{llk}$ is log-likelihood of the model

f_{score} will always be positive for normal observations because its log-likelihood will be greater than the models log-likelihood ($model_{llk}$). Positive scores will not be affected by the distance between observations and their clusters. However, negative scores must be adjusted by multiplying them by a distance factor X_{dist} which increases the score value as the observation's distance to the centroid of the assigned cluster decreases. X_{dist} always has a positive value and can be computed using the following formula:

$$X_{dist} = \frac{\sqrt{\sum_{i=1}^{d}(x_i - \mu_i)^2}}{d}$$

Where

d is number the observation's features

x_i is value of feature i of the observation X

μ_i is the mean of Gaussian distribution that models the feature density in space i

X_{dist} represents the euclidean distance between an observation and its assigned cluster centroid divided by the total number of dimensions. The value of X_{dist} will decrease as long as the distance from an observation to its cluster decreases, and vice versa. The final resulting value obtained from $f_{score} \times X_{dist}$ will be less than -1 for anomalous observations and greater than -1 for those that will be classified as normal.

4.3. Evaluation Datasets

To evaluate the efficiency of the proposed anomaly detection metric, we have used six classification datasets sourced from the UCI Machine Learning Repository [15]. These particular datasets have been chosen from different application domains and have varying amounts of data samples and attributes.

Each dataset includes several different classes. For each of the datasets, we considered data from the largest class as non-anomalous and used it for training. The test data includes samples taken from a class that is randomly chosen from the remaining classes plus the same amount of samples randomly taken from the training data. Table 4.1 summaries the important information of the used UCI datasets and the following sections provide detailed descriptions of each dataset.

Cardiac Arrhythmia Dataset. This dataset includes multivariate data with 452 instances. Each contains 279 attributes, 206 of which are linear valued and the rest are nominal. The data contains 16 classes of beat phases, where some arrhythmias are potentially very dangerous for the patient. Class 1 refers to normal, classes 2 to 15 are different types of arrhythmia, and class 16 refers to unclassified observations. The training data was taken from Class 1, including 245 instances, and the test data was taken from Class 3, including 15 instances plus 15 instances randomly taken from the training data.

Amazon Commerce Reviews Dataset. This dataset includes multivariate data with 1500 instances, each containing 10000 attributes. The data was acquired from customer reviews on the Amazon Commerce website for authorship identification. The number of reviews collected for each author is 30. Each data instance describes the authors linguistic style, which includes the usage of digits, punctuation, words, sentence length, and word

usage frequency. The training data is 30 instances from the Wilson class and the test data was drawn from the Vision class, including 30 instances plus 30 more from the training data.

Wisconsin Diagnostic Breast Cancer (WDBC). This dataset includes multivariate data with 569 instances, each with 32 attributes. The data are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Each instance includes an ID, diagnosis (M = malignant, B = benign), and 30 input features. The training data was taken from diagnosis B, which contained 357 entries. The test data was taken from diagnosis M, which contained 212 entries plus 212 more taken from the training data.

Cardiotocography Dataset. This dataset includes multivariate data with 2126 instances, each contains 23 attributes. The data representing 2126 fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label was assigned to each of them (N = normal; S = suspect; P = pathologic). The training data was taken from class N (1655 entries) and the test data was taken from class P (176 entries plus another 176 from the training data).

Seeds Data Set. This dataset includes multivariate data with 210 instances, each contains 7 attributes. The data represents measurements of geometrical properties of kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment. The training data was taken from class 1, including 70 instances, and the test data was taken from class 2, including 70 instances plus 70 instances taken from the training data.

Statlog (Shuttle) Data Set. This dataset includes multivariate data with 58000 instances, each contains 9 attributes. There are 7 classes and approximately 80% of the data belongs to class 1 The training data was taken from class 1, including 43500 instances, and the test data was taken from class 5 data, including 2458 instances plus 2458 instances taken from the training data.

Table 4.1. UCI classification data set used to evaluate the efficiency of anomaly detection

Data set	**Instances**	**Attributes**
Amazon Commerce reviews	1500	10000
Arrhythmia	452	279
Breast Cancer Wisconsin	569	32
Cardiotocography	2126	23
Seeds	210	7
Statlog (Shuttle)	58000	9

4.4. Adaptive Algorithm Evaluation

After building the GMM with the training data from our subject datasets, we used it to detect anomalies in the test data. We used two variants of our proposed algorithm: one that was provided a best-case static threshold based on manual inspection of the log-likelihoods, and one that was allowed to adaptively adjust its threshold over time.

Figure 4.2 shows the results of this evaluation across each of the UCI datasets. Each subfigure contains an ROC [16] curve, the best-case comparison threshold used, and the resulting classification accuracies. These results illustrate that our adaptive strategy can outperform a best-case static threshold in several problem domains, and also bolsters our decision to allow adaptivity to be toggled at run time.

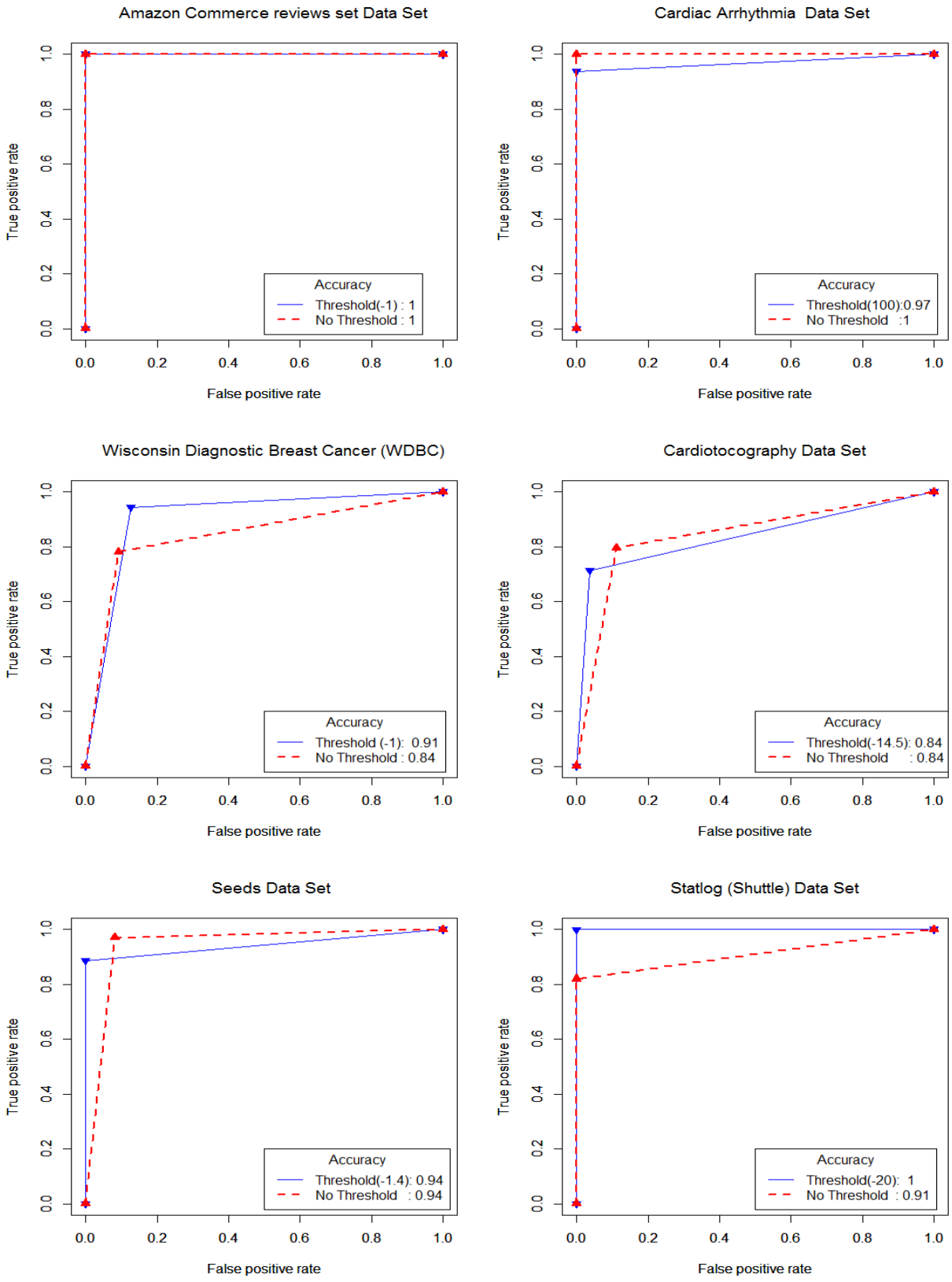

FIGURE 4.2. ROC curves and accuracies of both anomaly detection variants.

CHAPTER 5

Experimental Results

This section outlines the experiments we conducted to evaluate the efficiency of our proposed anomaly detection framework and its integration with Galileo. The experiments attempt to answer the following questions:

(1) Can our anomaly detection framework operate in real time while processing continuous data streams from Galileo?
(2) Can the framework detect anomalous observations efficiently without requiring user-defined thresholds?
(3) Is the framework able to accurately adapt its models as the dataset evolves over time?
(4) Can the framework detect anomalous observations whose features have usual values, but a combination of some features values are unusual?
(5) Can anomalies be detected based on geospatial locations?

5.1. Building the Models: Test Dataset

This study uses real-world climate data obtained from the National Oceanic and Atmospheric Administration (NOAA) North American Mesoscale Forecast System (NAM) [17]. The readings in this dataset are collected regularly from various weather and climate stations and stored in the self-describing NetCDF [18] format. Each file contains spatiotemporal information as well as a wide array of climate feature readings that include surface pressure, surface temperature, snow cover, snow depth, relative humidity, and wind speed. The particular data used in this study was collected over a ten-year period from 2004 to 2013. Each

year is comprised of roughly 1,000,000,000 observations on average. Observations from the first three years (2004 through 2006) were used as training data to build the initial models that were used in the experiments outlined in this section.

5.2. Experimental Setup

Our framework was run on a 77-node cluster with 48 HP DL160 servers (Xeon E5620 CPU, 12 GB of RAM, 15000 RPM disk) and 29 HP DL320 servers (Xeon E3-1220 V2 CPU, 8 GB of RAM, 7200 RPM disk). Ten clients running on machines outside the cluster were used to read the observations from the NOAA dataset and send them to Galileo, which operated under the OpenJDK Java Runtime, version 1.7.0 51.

Each storage node maintains an anomaly detection framework instance that receives observations from the storage node. Incoming data is partitioned spatially using the Geohash-based scheme described earlier, with a 4-character prefix used for these experiments to assign a 39.1 x 19.5 km region to each anomaly detector. In these benchmarks, the number of anomaly detector instances created by all the framework instances was 60924 in total. The training data that was used to build a model by each anomaly detector consisted of approximately 49241 observations on average.

5.3. Throughput Measurement

After building the models with our training data, we enabled adaptation mode and instructed the 77-node cluster to begin servicing storage requests. Our test clients streamed readings to the cluster sequentially to simulate the passage of time and real-world operating conditions, which would also enable us to confirm that the adaptive functionality was accounting for the gradual evolution of the dataset.

We measured the throughput at each node in the system and also recorded its cumulative throughput. We found that the 77-node cluster could process about 850,000 observations per second in parallel. Figure 5.1 illustrates the scalability of the system by reporting the cumulative throughput as additional resources are added to the cluster; the relationship between active nodes and throughput is roughly linear.

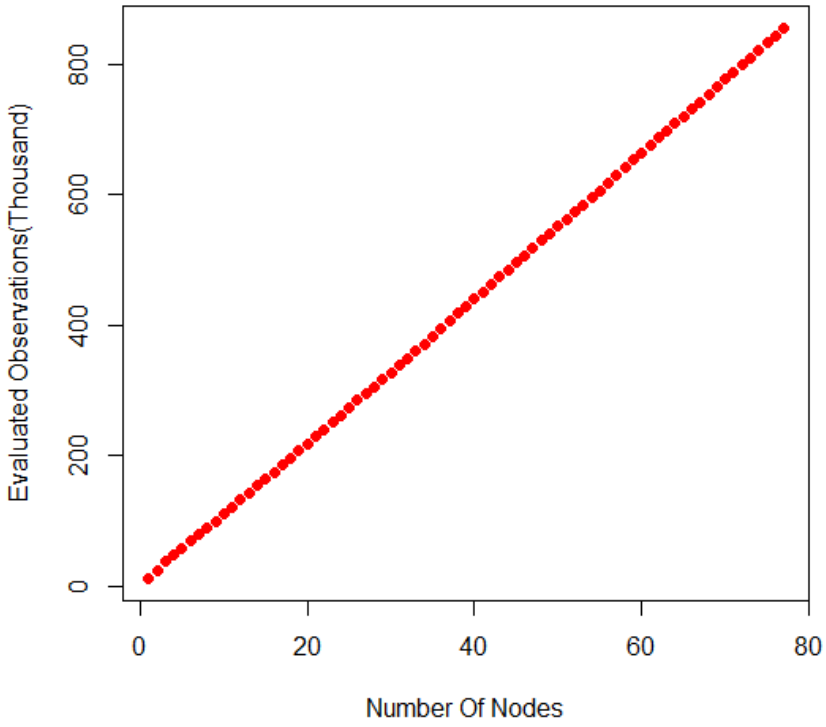

FIGURE 5.1. Number of machines vs number of classified observations per second.

5.4. NON-ADAPTIVE ANOMALY DETECTION

In this experiment, we extracted a random observation from the training data that was classified as normal. The particular observation in question was recorded in Washington DC at 6:00 pm on April 22, 2005. We also used the detector instances for the Geohash prefix

of *dqcj* covering Washington DC, and disabled the adaptation feature for this test. The framework was then used to evaluate the observation 100 times, where each evaluation step incremented the temperature of the observation. The initial temperature of the sample was set to 240 K, and the final modified temperature was 340 K.

As seen in Figure 5.2, all observations whose temperature values fall in the interval [266, 318] have been considered normal because their classification outcomes are larger than the fixed anomalous threshold (-1). Since the model has been built on training data from 2004, 2005, and 2006, then we can conclude that the normal temperature values for that time were from 266 Kelvin to 318 Kelvin. This experiment shows that without threshold specification, the framework is able to detect anomalous observations if they have values higher or lower than the norm.

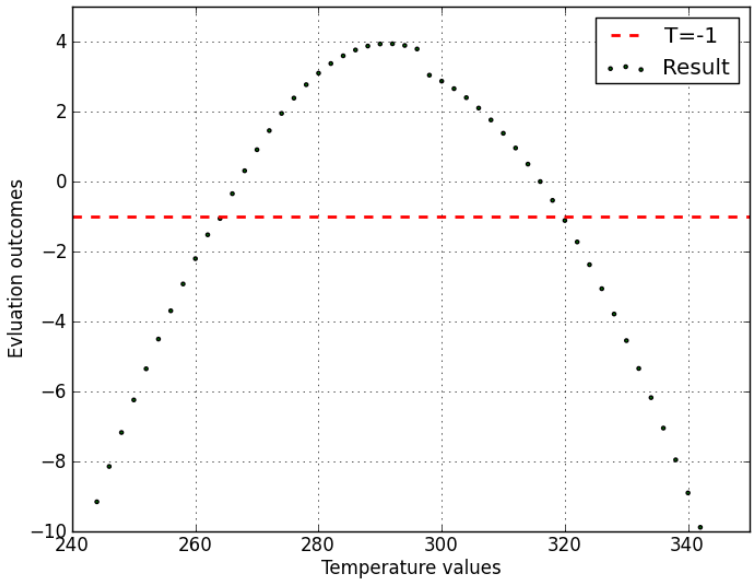

FIGURE 5.2. Outcomes obtained with adaptation disabled and gradually-increasing temperature values being classified from 240 to 340 K. All observations below the red line (outcome = -1) are considered anomalous.

5.5. Adaptive Anomaly Detection

In this experiment, we test how the framework copes with changes occurring in observed behavior when the adaptation feature is turned on. We have used the same observation that was used in the previous experiment, but in this experiment we created 300,000 copies of the observation. In each copy, the temperature feature was set to a value generated randomly between 330 and 360 Kelvin, which would constitute anomalous temperatures based on the results obtained in the previous experiment. The adaptation speed was set to 100, which means that adaptation will be performed once every 100 classifications. The framework was then used to classify the 300,000 copies of the observation, followed by another 300,000 classifications with adaptation turned off.

The outcomes of two cases can be seen in Figure 5.3. In the figure, the x-axis represents the evaluation steps and the y-axis represents the outcomes. As we can see, when the adaptation feature was disabled, the observation outcomes shown in blue were not changed over the 300,000 evaluation steps and were always below the anomalous threshold (-1), which means these observations were always considered anomalous. However, when the adaptation feature was enabled, the same observations were evaluated as anomalous early on in the test but began being considered normal over time. Consequently, we can conclude that our adaptation feature allows continuously-anomalous observations to eventually become normal over time, whereas the nonadaptive model would not be able to handle such changes.

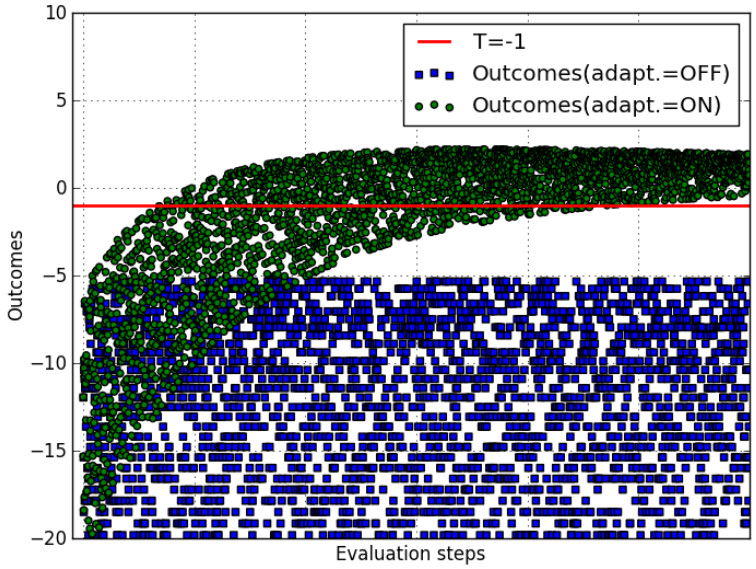

FIGURE 5.3. Outcomes obtained by running the anomaly detection approach twice, with and without the adaptation feature.

We also considered the experiments done in the previous section (see Figure 5.2) when testing our adaptive models. With adaptation enabled and the temperature incremented from 240 to 380 Kelvin this time, we can observe that the model can now cope with the evolving data and reflects the changes in its classifications seen in Figure 5.4. We can also observe that in the previous experiment the normal observations have temperature values in the range from 266 Kelvin to 318 Kelvin, while in this experiment the normal observations have temperature values in the range of 268 to 366 Kelvin.

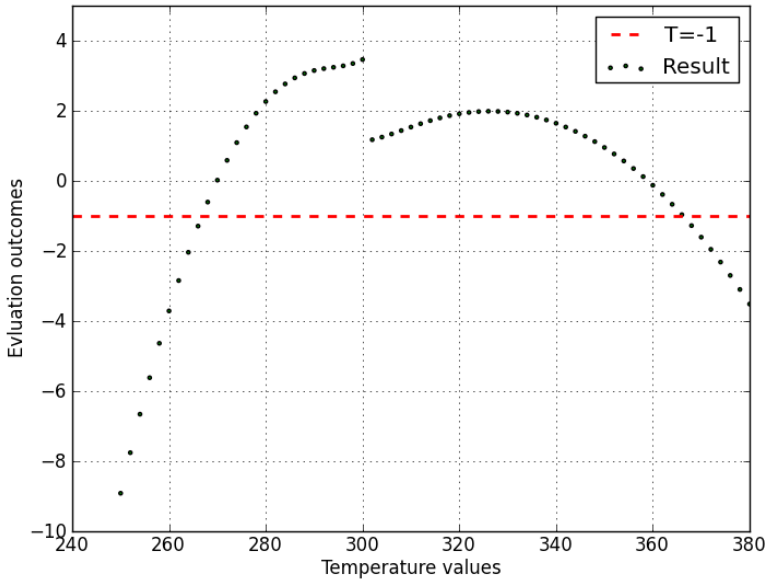

FIGURE 5.4. Outcomes obtained with adaptation enabled and gradually-increasing temperature values being classified from 240 to 380 K. In this case, the model adapts to accept a wider range of values as normal. All observations whose temperature values are below the red line (outcome = -1) are considered anomalous.

5.6. FEATURE CORRELATION

Each individual feature could have an unusual value that might justify classifying its related observation as anomalous. These kind of anomalies have been well-tested in previous experiments with our framework. In this experiment, we aim to test for anomalous observations that have valid feature values, but collectively as a tuple should be considered anomalous. This means that the combination of values must also be considered by the system, rather than only inspecting features individually; for instance, high temperatures may be common in a particular region during the daytime, but at night one might expect temperatures to drop.

In this experiment, we investigated correlations between different features to determine if the framework was able to detect anomalous observations that resulted from combinations of usual feature values. We have chosen two normal observations from the Washington DC region for this test, with some of the feature values displayed in Table 5.1. The last column in table contains the evaluation outcomes that indicate both observations are normal. This means that each individual feature has normal values and also the entire tuples are considered normal as well.

TABLE 5.1. Some features of the chosen observations

Time	**Temp.**	**Wind**	**Humidity**	**Surface Pressure**	**Outcome**
6-2-04, 6:00 pm	310 K	9	45%	101099	-0.22
6-7-04, 6:00 am	275 K	3	90%	101000	1.07

Based on the two test observations, we have created new observations including different combinations of values of features that may be correlated.For each new created new observation, the month is June, the year 2004, the day is randomly selected, and the time is drawn from [12:00 am, 6:00 am, 12:00 pm, 6:00 pm].

The evaluation outcomes of the first 11 days of June can be seen in Figure 5.5. The Figure shows that some observations are classified as anomalous. Some of the anomalous observations with their outcomes are shown in Table 5.2. To find out which features combinations cause that observations in table 5.2 are classified as anomalous, we plotted in Figure 5.6 average values of some observations' features from training data for the first 10 days of June.

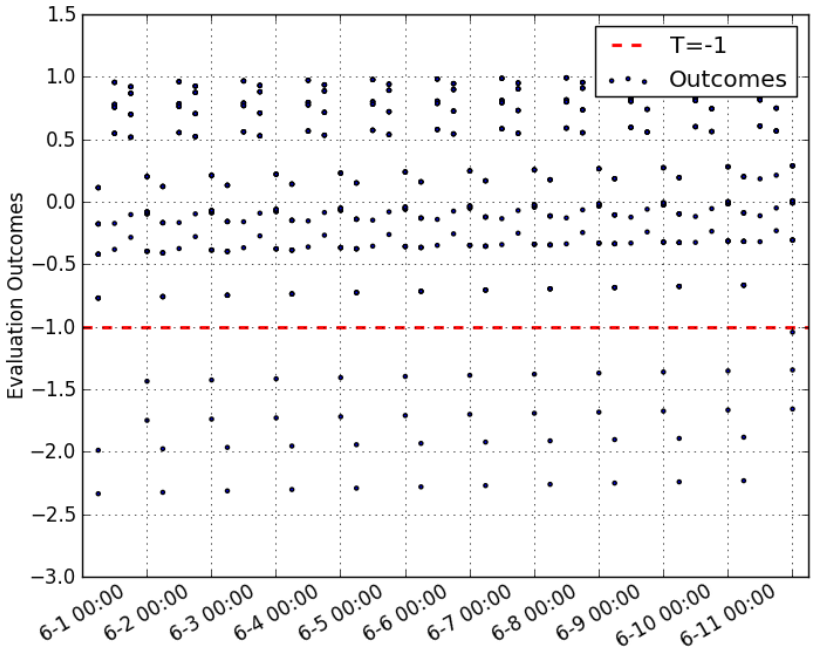

FIGURE 5.5. Outcomes of observations created from different feature combinations of normal observations.

As we can see in Figure 5.6, the observations in training data have quite low temperature and high humidity at 12:00 am, but the first and second observations have high temperature and low humidity at that time. This clears why they are classified as anomalous. Also, we can say that the reason why the last observation is anomalous might be that it has high humidity and temperature although the training data shows that both features are inversely proportional.

TABLE 5.2. The correlated features of observations created as combinations from two chosen normal observations

Time	**Temp.**	**Wind**	**Humidity**	**Surface Pressure**	**Outcome**
6-1-04, 12:00 am	310 K	3	45%	101000	-2.34
6-2-04, 12:00 am	310 K	9	45%	101000	-1.98
6-11-04, 6:00 pm	310 K	9	90%	101000	-1.04

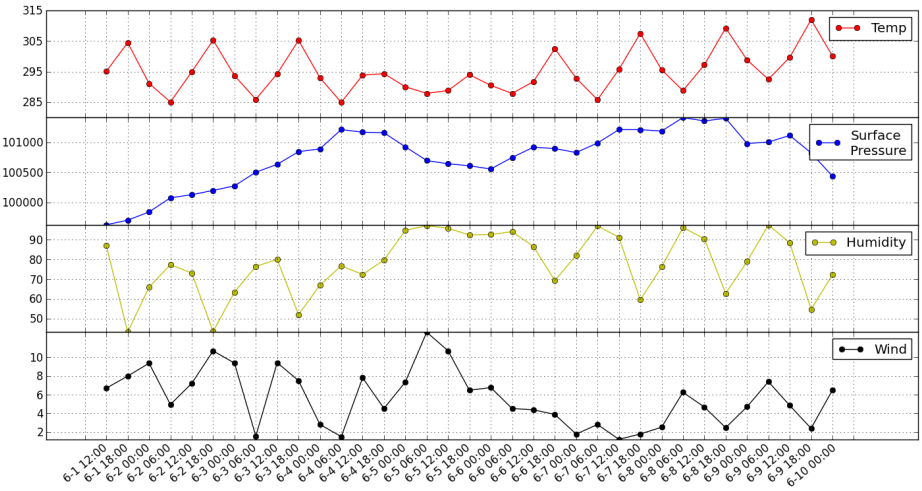

FIGURE 5.6. Average values of some observation's features at different times for the first 10 days in June.

5.7. Impact of Geospatial Scope on Model Accuracy

This experiment demonstrates the impact of the size of the geographic regions being managed by anomaly detector instances. We trained two models, one built with observations taken from Florida in the United States, and another built from data points belonging to Hudson Bay in Canada. Figure 5.7 demonstrates the geographic differences between the two locations. If geographic boundaries do not have an impact on classifications, then both models should be able to accurately detect anomalous data.

Random observations were sampled from both regions and used to test the models. In Table 5.3, observations in first three rows are taken from Hudson bay in Canada and have Geohash prefixes of $f4du$. The last three rows are taken from Florida and have Geohash prefixes of $djjs$. The evaluation outcomes of the four cases are shown in the last two columns. The fourth column shows the outcomes from the model that was built with training data from Florida, while the fifth column represents results obtained from the model that was built

using data from Hudson bay. It is clear that the observations from Florida were flagged as anomalous when they were classified with the model assigned to Hudson Bay, but classified as normal when they were evaluated by their own model. This proves that each model captures fine-grained details within its own spatial region, and reinforces our decision to use multiple detection instances.

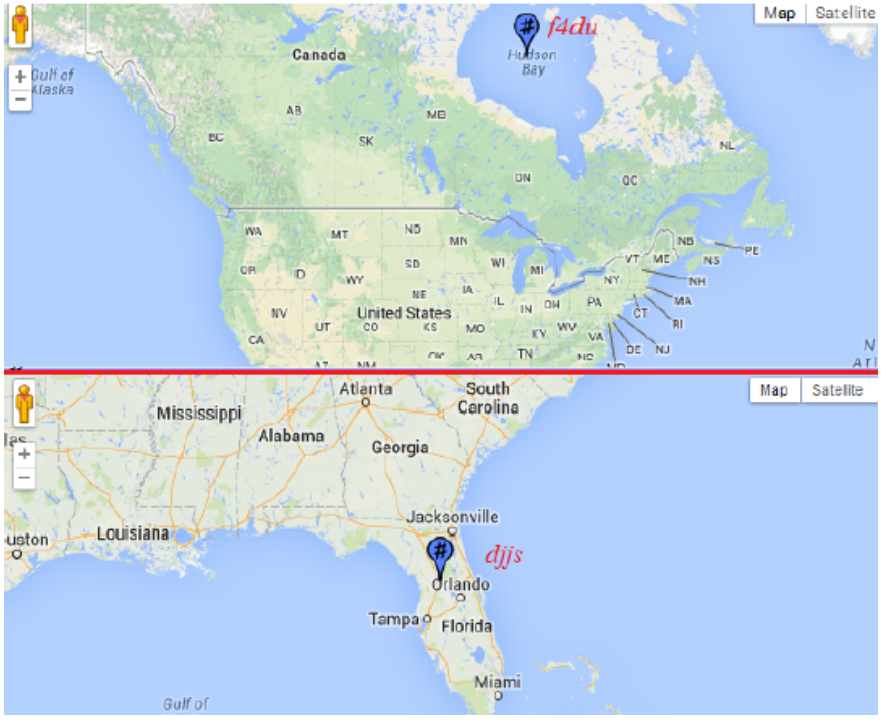

FIGURE 5.7. Two GMM models built with observations from two different areas (Florida in the USA and Hudson Bay in Canada).

TABLE 5.3. Observations taken in January 2013 for both Hudson Bay (Geohash: $f4du$) and Florida (Geohash: $djjs$).

Geohash	**Temp.**	Pressure	**Outcome at** $djjs$	**Outcome at** $f4du$
f4du	241	100711	-1.1E22	-0.5
f4du	244	101658	-1.1E22	-0.5
f4du	244	101659	-1.1E22	-0.4
djjs	306	101707	0.017	-538577
djjs	306	101692	-0.089	-708725
djjs	298	101461	0.007	-538253

CHAPTER 6

Related Work

A class of problems referred to as conditional anomaly detection requires dividing observation attributes into environmental (context) and indicator attributes [19] [20] [21] [22]. Solutions to this problem attempt to detect anomalies within specific contexts where a feature value (within a multidimensional observation) could be normal in one context and anomalous in another. An anomalous observation in this case is one that has an unusual indicator value at a specific environmental value. Correlated attributes have to be specified by a user or detected by performing additional processing. This correlation specification or processing step is unnecessary in our case because we model data using GMMs which are able to capture the correlations between attributes or dimensions within the data item [23] [24] [5].

Approaches have also performed anomaly detection by comparing the log-likelihood value of an observation with a sorted list containing log-likelihood values of the training data [19]. An observation is tagged as anomalous if its log-likelihood is less than the threshold specified by the user. Unlike our approach, this approach requires human intervention in the detection of anomalies including specification of the context and indicator variables. Some approaches tend not to be generic and require specification of a threshold. For example, Catterson et al. [21] use an approach similar to the one described by Song et al. [19] to monitor aging power transformers, where both environmental and indicator values are known in advance.

Approaches attempt to circumvent user invention by automatically discovering context [20]. However, this approach requires polynomial time to return the outcomes as a

list containing anomalous scores that need interpretation. The time-complexity of the algorithm precludes its use in real time anomaly detection. Our approach is amenable for use in real time settings with the model updates being performed in an online fashion.

Several approaches perform anomaly detection based on the nearest neighborhood approach [25] [26] [27] [28]. The underlying hypothesis in these approaches is that normal observations exist in dense neighborhoods, while anomalous observations tend to be quite distant from their closest neighbors [29]. Breunig et.al. [28] compute a local outlier factor for each instance in the dataset. The local outlier factor value can be used as an anomaly score dependent on the local density of the observations neighborhood. Zhang et.al. [25] provide an approach that computes a local distance-based outlier factor that shows the degree of deviation of an observation from its neighborhood and then returns n observations with the highest outlier scores that are then tagged as anomalous. A refinement proposed in [26] tries to alleviate the computation overheads by reducing the number of observations involved for computing distance-based outlier scores. Others have relied on ranking each observation based on the sum of the distances from the k-nearest neighbors [27]. Despite the accuracy of results, all distance-based solutions are very compute-intensive making then unsuitable for high-throughput, online anomaly detection.

Clustering is an unsupervised learning technique used to group data items into clusters based on similarities [30] that are measured in terms of distances to a clusters centroid. This has motivated several efforts in cluster-based anomaly detection. K-means [31] [32] [33] [34] is a widely used clustering approach to detect anomalies. The hypothesis in clustering-based approaches is that anomalies are observations that either belong to small clusters or are quite distant from any other clusters. These approaches start with an initial number of clusters and then perform an iterative refinement step that tries to find the optimal

clusters that minimize the squared sum of Euclidean distances between observations and the centroid of their cluster. Despite the simplicity of k-means, finding clusters that minimize the aforementioned objective function is an NP-complete problem [8]. Also, specifying the optimal number of clusters and their initial centroids is very difficult.

Clustering-based statistical techniques can be used to detect anomalous observations [29]. These employ Gaussian mixture models with expectation maximization to cluster data into groups based on densities. Some approaches build GMM on noisy training data with the assumption that the number of items tagged as normal will outnumber those tagged as anomalous [29]. A test sample is anomalous if adding it to the distribution causes the distribution's log-likelihood to change by a predefined threshold. The process that requires computing the distribution's log-likelihood with and without the test sample is unneeded in this case because the process is equivalent to comparing the log-likelihood of the test sample with a threshold. Some approaches [35] compare a threshold with the likelihood that anomalies are found, while others [36] compute anomaly scores for an observation based on GMM. These approaches require human intervention the former approach requires threshold specification, while in the latter approach the anomaly scores need to be interpreted by the user.

There are solutions [37] [38] [14] that attempt to adapt the underlying model based on observed data. Hasan and Gan [37] provide an approach similar to our own which modifies the GMM parameters (μ_k,Σ_k,N_k) in an online manner to adapt the GMM components. In this solution where GMM is used for classification, the values of variances stored in Σ_k will always increase, which leads to increasing the sizes of clusters that will cover the entire dataset, both the new and old. Increasing the size of clusters produces overlapping clusters. However, both increasing the clusters sizes or overlapping between clusters produce

inaccurate classification results. To address this, a refinement of the approach tries to detect cluster overlaps retrain the model which in turn affects the online classification process.

Approaches also rely on user-feedback to achieve adaptation [14]. Here, the threshold that is used to classify whether an observation is anomalous or not changes over time. Based on feedback obtained from the end user the system adjusts the threshold to reduce false classifications. However, this approach is not suitable for online adaptation of models and requires extensive human intervention.

CHAPTER 7

Conclusion

In this thesis we presented our approach encompassing algorithms and system design for scalable detection of anomalies in multidimensional data streams. We used Expectation Maximization (EM) to build Gaussian Mixture Models (GMM) that model the densities of the training data by using different combinations of Gaussian distributions. The feasibility of this approach for anomaly detection in multi-dimensional datasets was verified with well-known datasets. The ROC curves in our empirical evaluations over these datasets demonstrate the suitability of our approach.

Given the data volumes, along with the dimensionality and the rates at which data arrive, it is infeasible to employ human-intervention in the anomaly classification process. Our approach does not require human intervention for either the establishment or adjustment of anomaly detection thresholds.

Since anomalies evolve over time, it is important for the underlying model to account for this evolution. Our model continually adapts itself based on the data observed by the model i.e., both the adaptation and the classification are performed concurrently. Our empirical benchmarks demonstrate both the efficiency (per-packet classifications) and accuracy of these adaptations.

To deal with observations that are spatio-temporally correlated, our model instances are responsible for a particular geographical extent and account for timestamps as one of the dimensions associated with each data point. Each model instance tunes itself independently based on the data it observes. This allows each model instance to account for spatio-temporal correlations accurately. Rather than have a singular model that attempts

to preserve such correlations, model instances that span smaller geographical extents reduce both the complexity of the detection and improve the accuracy. This approach works well in situations where there is variability in the density of data streams available for particular geographical extents: the geographical extents can be adaptively refined with corresponding addition of model instances. Ultimately, our approach allows finetuning of the specificity of the classifications by controlling the geographical scope associated with the classification models.

Having model instances associated with particular geographical extents results in a scalable design i.e. we can scale with increases in data volumes and the number of machine available. Having multiple model instances is amenable to dispersion, with model instances executing on multiple machines, which allows for concurrent, distributed classifications as data streams corresponding to observations from multiple locations arrive. At a particular node, our approach involves having multiple model instances managed using a thread pool, which allows for concurrent classifications of data streams. Our empirical results validate the scalability and throughput of our approach at both the individual nodes and in a distributed cluster as well.

Bibliography

[1] M. Malensek, S. Lee Pallickara, and S. Pallickara, "Exploiting geospatial and chronological characteristics in data streams to enable efficient storage and retrievals," *Future Gener. Comput. Syst.*, vol. 29, pp. 1049–1061, June 2013.

[2] M. Malensek, S. L. Pallickara, and S. Pallickara, "Expressive query support for multidimensional data in distributed hash tables," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, (Washington, DC, USA), pp. 31–38, IEEE Computer Society, 2012.

[3] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2nd ed., 2010.

[4] D. A. Reynolds, "Gaussian mixture models," in *Encyclopedia of Biometrics*, pp. 659–663, 2009.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[6] A. P. Dempster, N. M. Laird, D. B. Rubin, *et al.*, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, Nov. 2009.

[8] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of euclidean sum-of-squares clustering," Hingham, MA, USA, May 2009.

[9] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Roy. Stat. Soc.*, vol. 36, pp. 111–147, 1974.

[10] M. Malensek, S. Pallickara, and S. Pallickara, "Polygon-based query evaluation over geospatial data using distributed hash tables," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, UCC '13, (Washington, DC, USA), pp. 219–226, IEEE Computer Society, 2013.

[11] Wikipedia, "Geohash— Wikipedia, the free encyclopedia," 2013. [Online; accessed 15-July-2013].

[12] MongoDB Developers, "MongoDB," 2014. [Online; accessed 29-January-2014].

[13] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[14] M. Raginsky, R. M. Willett, C. Horn, J. Silva, and R. F. Marcia, "Sequential anomaly detection in the presence of noise and limited feedback," *Information Theory, IEEE Transactions on*, vol. 58, no. 8, pp. 5544–5562, 2012.

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

[16] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, (New York, NY, USA), pp. 233–240, ACM, 2006.

[17] National Oceanic and Atmospheric Administration, "The north american mesoscale forecast system," 2014.

[18] R. Rew and G. Davis, "Netcdf: an interface for scientific data access," *Computer Graphics and Applications, IEEE*, vol. 10, no. 4, pp. 76–82, 1990.

[19] X. Song, M. Wu, C. Jermaine, and S. Ranka, "Conditional anomaly detection," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 5, pp. 631–645, 2007.

[20] X. Wang and I. Davidson, "Discovering contexts and contextual outliers using random walks in graphs," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pp. 1034–1039, IEEE, 2009.

[21] V. M. Catterson, S. D. McArthur, and G. Moss, "Online conditional anomaly detection in multivariate data for transformer monitoring," *Power Delivery, IEEE Transactions on*, vol. 25, no. 4, pp. 2556–2564, 2010.

[22] S. D. McArthur, C. D. Booth, J. McDonald, and I. T. McFadyen, "An agent-based anomaly detection architecture for condition monitoring," *Power Systems, IEEE Transactions on*, vol. 20, no. 4, pp. 1675–1682, 2005.

[23] K. Subramanian, "Task space behavior learning for humanoid robots using gaussian mixture models.," in *AAAI*, 2010.

[24] S. Calinon and A. Billard, "A probabilistic programming by demonstration framework handling constraints in joint space and task space," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 367–372, IEEE, 2008.

[25] K. Zhang, M. Hutter, and H. Jin, "A new local distance-based outlier detection approach for scattered real-world data," in *Advances in Knowledge Discovery and Data Mining*, pp. 813–822, Springer, 2009.

[26] E. M. Knox and R. T. Ng, "Algorithms for mining distance based outliers in large datasets," in *Proceedings of the International Conference on Very Large Data Bases*, pp. 392–403, Citeseer, 1998.

[27] F. Angiulli, S. Basta, and C. Pizzuti, "Distance-based detection and prediction of outliers," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 18, no. 2, pp. 145–160, 2006.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM Sigmod Record*, vol. 29, pp. 93–104, ACM, 2000.

[29] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[30] O. R. Zaïane, "Introduction to data mining," 1999.

[31] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[32] C. Elkan, "Using the triangle inequality to accelerate k-means," in *ICML*, vol. 3, pp. 147–153, 2003.

[33] G. Hamerly and C. Elkan, "Learning the k in k-means," in *NIPS*, vol. 3, pp. 281–288, 2003.

[34] K. Alsabti, S. Ranka, and V. Singh, "An efficient k-means clustering algorithm," 1997.

[35] I. Davidson, "Anomaly detection, explanation and visualization," tech. rep., SGI, Tech. Rep, 2007.

[36] X. Yang, L. J. Latecki, and D. Pokrajac, "Outlier detection with globally optimal exemplar-based gmm.," in *SDM*, pp. 145–154, SIAM, 2009.

[37] B. A. S. Hasan and J. Q. Gan, "Sequential em for unsupervised adaptive gaussian mixture model based classifier," in *Machine Learning and Data Mining in Pattern Recognition*, pp. 96–106, Springer, 2009.

[38] M. Song and H. Wang, "Highly efficient incremental estimation of gaussian mixture models for online data stream clustering," in *Defense and Security*, pp. 174–183, International Society for Optics and Photonics, 2005.