Title
A Computational Pipeline to Improve Clinical Alarms Using a Parallel Computing Infrastructure

Author
Nguyen, Andrew

Publication Date
2013

Peer reviewed|Thesis/dissertation

A Computational Pipeline to Improve Clinical Alarms Using a Parallel Computing Infrastructure

by

Andrew V. Nguyen

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Biological and Medical Informatics

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, SAN FRANCISCO

[redacted]

DEDICATION AND ACKNOWLEDGEMENTS

Dedicated to my awesome parents, Loi and Elizabeth, wife, Brenna, and brother, Phap Sieu for all of their support and encouragement. I would not have been able to do this without your love and support.

I would also like to thank:

Dr. Yao Sun for guiding me these past four years. Despite your extremely busy schedule, you always made time for a weekly meeting, offering words of wisdom and making sure that I stayed on the right track.

Dr. Geoffrey Manley and Dr. Robert Gray for serving as my committee members. Despite your other obligations and hectic schedules, you made the time to discuss my research and offer helpful advice and insights.

Dr. David Avrin for mentoring me for the past 14 years. You took on a fresh high school graduate and introduced me to the field of informatics. You continued to challenge me and guide me as I was trying to find my path my first few years at UCSD. And, most importantly, you brought me back to informatics after a slight post-college detour.

Dr. Ida Sim and Dr. Russ Cucina for running the clinical informatics program and giving me the opportunity to explore an exciting field at a great institution.

Dr. Maurice Cohen and Dr. Donna Hudson for teaching BMI 202 which was the basis and catalyst for my research.

Dr. Stuart Russell for your guidance on artificial intelligence, dynamic bayesian networks, and of course, for delicious tea and cookies.

Cheryl Fong and Mary Ulman for always being there and helping me with whatever I needed.

ABSTRACT

A Computational Pipeline to Improve Clinical Alarms
Using a Parallel Computing Architecture
Andrew V. Nguyen

Physicians, nurses, and other clinical staff rely on alarms from various bedside monitors and sensors to alert when there is a change in the patient's clinical status, typically when urgent intervention may be necessary. These alarms are usually embedded directly within the sensor or monitor and lack the context of the patient's medical history and data from other sensors. A missed alarm may result in severe morbidity or mortality so alarm algorithms tend to err on the side of sensitivity at the expense of increased false positives.

Consequently, 40-90% of alarms within the clinical setting are false. These false alarms have a negative impact on patient care as clinicians become desensitized to the alarms. False alarms also directly impact patient recovery due to sleep interruption and increased anxiety.

There is an increasing amount of research dedicated to improving clinical alarms but the field lacks a standardized approach to capturing, storing, processing, and analyzing physiological sensor data. This work focuses on the informatics issues for conducting research involving the combination of various clinical data sources (e.g. EHR/EMR) with physiological sensor data (including high resolution waveforms).

The described platform can also be easily extended into the clinical environment to provide clinical decision support. There is also the additional benefit that researchers do not have to concern themselves with how to connect, retrieve, or format the data. Instead, they can focus solely on the higher level problems of designing experiments, implementing algorithms, and interpreting results.

A 3-stage computational pipeline was implemented on top of a Hadoop-based parallel computing platform to reduce the number of false alarms of ventricular fibrillation and ventricular tachycardia. Waveforms were fed through feature extraction and change (point) detection algorithms, then through supervised learning algorithms to generate a model that was able to better detect true alarm situations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

The field of medical informatics has been a cross-, inter-, and multi-disciplinary field since its inception. This intersection includes (but is definitely not limited to) scientists and researchers in medicine, physiology, clinical research, biostatistics, computer science, information science, and information technology. Each field brings unique insight and no single field can adequately address the many problems we face in medical informatics.

The problems that medical informatics seeks to solve come from those on one end of the spectrum - those who are clinically oriented and bring with them a deep domain expertise. On the other end of the spectrum, we have technologists who are bringing solutions from other domains such as finance, advertising, and the internet/web. This intersection typically results in domain experts looking for any solution to solve their particular need or problem and technologists looking to apply their technique to any problem.

In the context of this work, such an intersection has often manifested itself as a clinician looking for any analytic approach that would do a better job at segmenting patient populations. For example, cardiologists would like to do a better job at detecting critical arrhythmias, or hospitalists would like to improve the detection of septic patients. On the technology side, there are computer scientists looking to apply neural networks to many different clinical questions across different specialties.

The underlying theme of this research has been to build a better bridge to connect the domain experts with the technology. By standardizing the infrastructure or "plumbing" and some of the scientific approaches, the domain experts and technologists can focus on solving the critical problems instead of wasting time trying to build (and rebuild) the data and computational infrastructure.

The need for a scalable infrastructure becomes even more necessary as the entire healthcare industry moves to digitize *everything*. Clinicians, clinical researchers, and data scientists should not need to concern themselves with how to scale the computation or how to connect the data. They should be able to focus on the problem itself and expect that the infrastructure is there to support them.

This research is specifically targeted towards those who are attempting to apply data mining and analytics techniques to biometric, physiological, signal/sensor data. From point-in-time data such as a blood glucometer to waveforms such as EKG's, the main goal was to build and characterize a scalable system that could be applied to a wide variety of problems. In addition to the plumbing, another goal is to begin the process of defining the structure and processes of the data mining process when dealing with sensor data.

2 CLINICAL ALARMS

Clinical alarms represent one of the primary means that clinicians use to monitor the status of their patients. These alarms are critical to ensuring a workflow that allows clinicians to care for more than one patient. These alarms are based on physiological sensors such as an electrocardiogram (ECG/EKG), blood pressure, or intracranial pressure. These physiological sensors offer one of few truly objective windows into a patient's clinical status or condition.

Many studies over the past decade have hinted at patterns within physiological signal data that may be indicative of disease states or clinical conditions[11, 16, 18, 22, 27, 42, 50, 58, 59]. However, due to limitations in computational infrastructure, there have only been limited attempts at elucidating complex patterns to drive clinical decision support.

These clinical alarms, however, are the primary mechanism for detecting problems within the clinical environment. Given the staffing levels within the typical in-patient floor, the alarms play a critical role in helping allocate limited resources effectively. However, this assumes that the clinical alarms are accurate in identifying urgent or critical changes in a patient's clinical status.

The literature is filled with numerous examples where the number of false alarms far outweighs the number of true alarms[7, 13, 19, 34, 39]. While these vary between different in-patient settings, the false alarm rate varies from 40% to 90%. Especially on the higher end, false alarms have detrimental effects to both the healthcare providers as well as the patients. For clinicians, constantly shutting off an alarm is training to ignore future alarms, especially critical ones. For patients, the constant alarms create an environment of restlessness and anxiety, neither of which are conducive to healing.

The alternative, however, is to make alarm algorithms much more stringent, only letting through conditions that are guaranteed to be critical. The obvious problem is that this may cause a legitimate alarm to be masked, thus defeating the purpose of

having alarms in the first place. This highlights the core of the problem - how to balance false alarms against missed alarms.

Given that the existing alarms have been tuned to be overly sensitive, they form the first layer of data for the research described herein. The main goal is to take the alarms and craft a data mining system that is capable of extracting interesting features from the sensor data and use this information to do a better job at filtering the alarms.

The first step is to be able to extract features of individual sensor streams, many of which may not be perceptible by the clinician without a mathematical transformation. The next step would be to combine multiple features of a single sensor stream to improve alarm accuracy. At this point, however, the solution is still a univariate solution - where the alarm is based solely on the data and information coming from a single sensor stream.

Multivariate solutions become the next layer where multiple different sensor streams are connected together to provide a more comprehensive "view" of the patient's status. This is typically how the clinician operates looking at both the EKG monitor and pulse oximeter to determine if a heart rate alarm is legitimate. Another input to multivariate solutions involves looking at the patient's medical record for previous diagnoses, prescribed or administered medications, and other pertinent data that can be added to the underlying model.

By combining as much data and information as possible, a system could be created that bolts onto existing alarm algorithms to do a better job at filtering out false alarms.

3 PHYSIOLOGICAL AND BIOMETRIC SENSORS

Physiological sensors have become an integral part of the practice of medicine, playing a crucial role in giving the clinician a small window into the inner workings of the human body. These sensors range from detecting changes in electrical activity of neurons to changes in blood concentration of glucose.

When doing a survey of the literature, most studies that examine physiological data focus on lower resolution data. These include measurements that range anywhere from minutes to months. For example, a patient's heart rate or cuff-based blood pressure may be taken every 5 minutes in more critical situations or may be taken once every 6 months when the patient visits the doctor.

This research is focused on higher resolution physiological data such as waveform data from EKG's or arterial blood pressures. These data range anywhere from seconds to sub-seconds. For example, arterial pressure sensors from Philips were recorded at 125Hz as part of the MIMICII dataset[54]. EEG's, on the other hand, can be sampled up to several kHz.

3.1 PHYSIOLOGICAL SIGNAL DATA AS TIME-SERIES DATA

In order to simplify the problem of data mining physiological signal data, one useful abstraction is to treat any physiological signal as time-series data. Whether looking at daily blood glucose measurements or an EKG waveform, any physiological signal can be described as a set of key-value pairs where the key is a timestamp and the value is the measurement. This measurement can be a voltage, concentration, or any other recording by a sensor.

By treating all of these data as time-series data, a system can be constructed that is able to absorb data from nearly any type of sensor and process and analyze it accordingly.

Historically, there have been several major issues when dealing with physiological signal data, and especially the data mining of such data. Each of these poses a separate challenge, none specific to medicine. However, there are some differences in the healthcare industry that magnify these problems.

1. Synchronization of sensors
2. Gaps in the data
3. Artifact/noise in the data stream

These differences are due to the regulatory overhead of the FDA and HIPAA as well as the evolutionary path of the "sciences" of biology and medicine.

One of the first problems researchers encounter is the synchronization of sensors. As with any system, each device maintains its own clock which results in some jitter when comparing data from different sensors. One natural solution is to centralize the time-keeping in order to do a better job of synchronizing sensors. The other may be to centralize the data capture to ensure that there is a deterministic approach to recording the data.

However, due to FDA regulatory overhead, most device manufacturers have created completely isolated devices that are fully independent. The centralization of the data capture is then laid on top of the individual devices which makes it very difficult to accurately synchronize the devices.

4 HADOOP: A PARALLEL COMPUTING PLATFORM

With the continued growth and expansion of the internet, several web companies have discovered new ways to reuse old concepts in order to deal with the ever increasing amount of data being generated. The concepts of Map and Reduce have been around and integrated into functional programing languages[9, 32, 33, 63] and are familiar to those who have programmed in Lisp or similar. In 2004, Google adapted these concepts to assist in the parallelization of certain types of problems giving birth to the idea of parallel mapreduce[20]. Though patented by Google, several other companies have implemented their own parallel mapreduce platforms.

After a brief overview of the parallel mapreduce concepts, details of the specific implementation being used will be reviewed.

4.1 PARALLEL MAPREDUCE

4.1.1 *Map*

The concept of mapping is familiar to functional programming language and is a very simple concept: define a function to map one key-value pair to another key-value pair.

4.1.2 *Reduce*

The concept of reducing is also familiar to functional programming languages and simply reduces a set of key-value pairs that share the same key to a single key-value pair. Implicit within this process is some sort of operation that combines or aggregates the values. For example, the map function may emit a series of key-value pairs where the key is a particular sensor ID and the values are its measurements. The reduce function would take these values and average them such that the reducer output is the average value of the sensor.

4.1.3 *Make it parallel*

Google popularized and patented the application of the mapreduce paradigm to parallel computational problems. The concepts of mapreduce allow for easy parallelization due to their inherent characteristics. Each parallel map task is given a subset of the input and applies its mapping, emitting the intermediate key-value pairs. The intermediate key-value pairs are then sorted and shuffled to parallel reduce tasks. All key-value pairs that share the same key get sent to the same reduce task. For those problems that fit this paradigm, their parallelization becomes trivial when compared to other parallel computing solutions such as MPI[56].

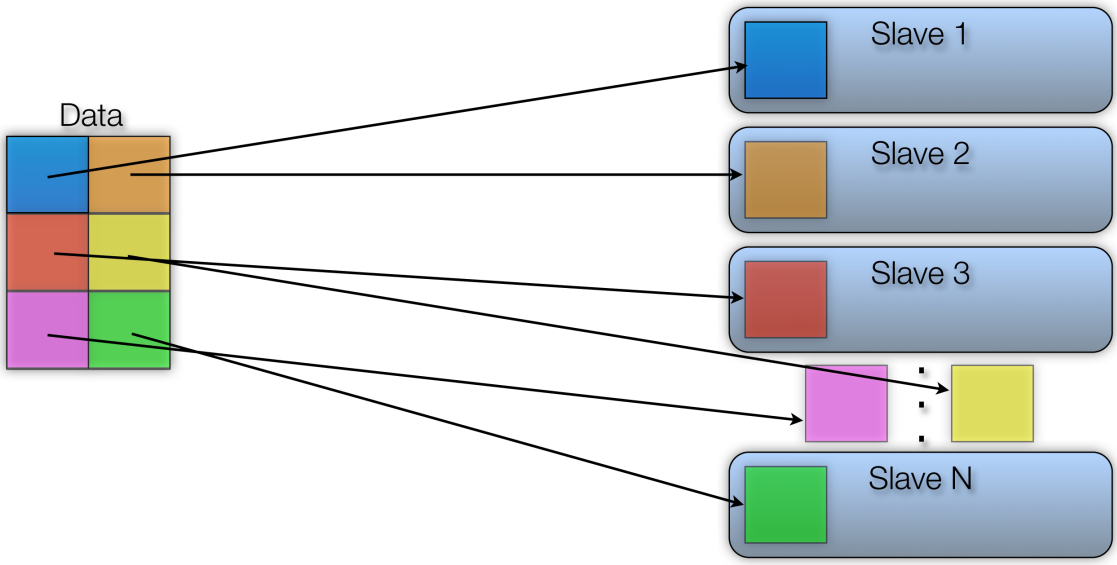

Figure 1: Parallel MapReduce Splitting

As you can see in figure 1, the data are split into multiple "splits" which are computed upon by each slave in the cluster. Where possible, the splits are assigned to nodes in the cluster that contain a local copy of the data. This concept of *data locality* is an important part of making the parallel mapreduce computationally efficient by minimizing network traffic of the data to be processed.

4.2 HADOOP

The Apache Software Foundation now manages an open-source parallel mapreduce implementation called Hadoop. It was originally developed as an internal tool at Yahoo! and later open-sourced under the Apache license. Hadoop has evolved into a computational platform or ecosystem on which many different tools are built. These include the Hadoop Distributed File System (HDFS), HBase, Pig, Hive, among many others. Due to its evolving nature, the Hadoop ecosystem contains many tools with varying degrees of overlap. In some cases, there is a nearly complete functional overlap with the difference being purely syntactic.

At a minimum, users of the Hadoop platform will utilize at least the Hadoop Core and HDFS. For this project, a distributed, column-store database that is part of the NoSQL trend, called HBase, was also integrated into the technology stack.

4.2.1 *Hadoop Core*

The Hadoop Core provides the mapreduce functionality as outlined in Google's paper and patent. It is a Java implementation of the mapreduce parallel programming paradigm and provides a framework on which applications can be built. The two primary components of the mapreduce core are the JobTracker and TaskTracker. The JobTracker acts as a master, coordinating all of the TaskTracker slaves.

4.2.2 *Hadoop Distributed File System*

The Hadoop Distributed File System, or HDFS, provides the underlying storage mechanism for the Hadoop platform. While there are several alternatives to HDFS currently being introduced, HDFS remains the default file system. As its name implies, HDFS is a distributed file system, spread across the entire cluster. As with any file system, data stored in HDFS are split into blocks. Each block is replicated across the cluster, typically 3 nodes; however, this value is customizable. This replication provides both redundancy and efficiency during computation.

With the blocks replicated across multiple nodes, the cluster can suffer the failure of hard disks or entire slave machines without any loss of data. This becomes especially important when dealing with datasets that do not fit on a single server or storage system. The replication of blocks also enables data-local computation where the algorithms being executed are moved to the server containing the data as opposed to the other way around. While data-locality is a priority, it isn't always possible to ensure this. In these situations, the node performing the computation may need to request the data via the network. By increasing the number of nodes that contain a copy of the data to be processed, there is a greater chance at achieving data-locality.

The HDFS is designed to facilitate parallel mapreduce and, as a result, isn't very good at storing small files or providing random access to data stored within it. When designing the system, one of the requirements was to be able to plot the signal(s) of any patient at any time. As a result, HBase was chosen as the primary data storage system.

4.2.3 *HBase*

HBase is a distributed, column-store database and is part of the NoSQL variety of databases (as opposed to SQL-based relational databases). As a column-store database, HBase is designed to store sparse data efficiently unlike a relational database. The trade-off is that every value within the database is stored as a triplet - the row identifier, the column identifier, and the value (along with some other HBase-specific overhead).

4.2.4 *MongoDB*

While HBase contains all of the data from the sensors, the metadata are stored separately in MongoDB[4], a document-oriented, NoSQL database. By separating the sensor data from the metadata, the storage and management of each is optimized.

MongoDB was chosen as the underlying data store for two primary reasons: 1) Simplifying the development process, and 2) Storing disparate data in a single location.

During any sort of software development that frequently interacts with an underlying data store, there is a constant need to add, remove, and modify portions of the database schema. While there are many frameworks (e.g. Hibernate) that assist with abstracting the underlying database schema changes, they inherently add an additional layer of complexity to the system. Changes need to be tracked, organized, and managed in such a way that any modifications do not affect other developers or production systems.

MongoDB provides a document-oriented data store that is schema-less. As a result, changes can be made at the application-level and no changes to the data store are necessary. While this simplifies the development process, it also makes it possible to store two records that are very similar but still have very functional differences. When compared to a traditional, relational database, MongoDB negates the need for an overly complex table structure or the need to add a second table that looks nearly identical.

The schema-less nature of MongoDB also allows for the storage of disparate metadata in a single data store. For example, some sensors may have more parameters or descriptors than another sensor. MongoDB allows both of these records to be treated nearly identically, pushing any differences to runtime. This becomes especially useful when integrating an ontology-based management layer. Any sensor-specific metadata can be encoded within an ontology and applied when actually processing or interacting with the data.

4.2.5 *Online Processing with Hadoop*

While the dataset was previously collected and thus all data is immediately available for processing, one of the goals was to create a system and algorithms that can be applied to streaming data. By using the Hadoop ecosystem, the community of researchers and commercial providers can be leveraged. This community has been working to provide "real-time" or "online" versions of the mapreduce paradigm. Several examples include Hadoop Online[17] and Twitter Storm[46]. However, online or real-time processing will be the focus of future research.

5 A 3-STAGE COMPUTATIONAL PIPELINE

A 3-stage computational pipeline was designed to address the many problems with analyzing and data mining physiological signal data:

1. Feature extraction
2. Change detection
3. Machine learning

These concepts are not novel, nor is their combination. They are present in almost any attempt at data mining, whether or not with time-series data. However, there is nothing in the literature that examines this particular sequence for analyzing physiological signal data.

The main goal of this computational pipeline was to create a framework in which any clinical condition can be described as a set of changes of extracted features of a set of physiological signals. Due to the focus on change detection, this approach can only be used to detect and describe changes in a patient's clinical status. It cannot be used to identify a steady-state clinical condition.

To illustrate this limitation, consider the detection of an arrhythmia (such as ventricular fibrillation) in a patient:

SITUATION 1: A patient presents to the ED and is immediately connected to an EKG monitor. The nurse notices that the patient has a normal sinus rhythm. After some unknown period of time, the patient developed an arrhythmia - ventricular fibrillation.

In this example, the onset of the arrhythmia can potentially be detected and described using the this change detection approach.

SITUATION 2: A patient comes to the ED but is left in the waiting room awaiting the triage nurse. At some point, the patient experiences severe chest pain and is brought into the ED and immediately connected to an EKG monitor. The nurse notices that the patient is in ventricular fibrillation.

In this example, the arrhythmia could not be detected or described using this change detection approach.

In other words, this change detection approach is limited to detecting changes in a patient's clinical state or clinical condition and cannot identify or classify the current state or condition. Because this approach focuses on detecting changes in clinical status, it can detect both the onset of a clinical condition as well as it's resolution.

5.1 MULTIVARIATE ANALYSIS

One key benefit of the proposed approach is the ability to integrate multiple different physiological signals into the detection algorithms. While certain current alarm algorithms such as those for arrhythmia detection use multiple sensors as input, they are confined to the same physiological signals such as different leads from an electrocardiogram. The proposed approach combines multiple different physiological signals such as electrocardiograms and arterial blood pressures. This combination of multiple different physiological sensors beings to mimic the process that clinicians use to judge the veracity of an alarm.

5.2 FEATURE EXTRACTION

Feature extraction is typically used to refer to a specific form of dimensionality reduction - the transformation of data into a smaller set of features that can accurately describe the original data. However, in this context, feature extraction is a data processing algorithm that is being applied to highlight a particular characteristic of the underlying data. These algorithms can be domain-specific such as the RR-interval of an EKG or they can be generic mathematical algorithms such as the rolling mean of a series of values.

These two main classes of feature extraction algorithms - those that are specific to a particular domain or clinical focus and those that are generic, mathematical algorithms.

5.2.1 *Domain-specific algorithms*

5.2.1.1 *RR-interval*

The RR-interval is the time duration between two R-waves within an EKG waveform. This effectively represents the time between one heartbeat and the next and numerous studies in the literature have attempted to correlate RR-interval variability with various clinical conditions.

In order to implement an RR-interval extraction algorithm, a slightly modified version of the Pan-Tompkins algorithm[51] was used to isolate the QRS complexes within the source EKG waveform.

5.2.1.2 *Systolic Pressure Detection*

One of the input signals was an arterial pressure waveform. A feature extraction algorithm was implemented that extracted the systolic pressure of each beat. At their core, systolic pressure detection algorithms are "intelligent" peak detection algorithms. For this project, the implemented algorithm was a relatively simple peak detector with a tunable threshold. As a feature extraction algorithm, several different parameters were used in order to generate different feature series.

5.2.1.3 *QRS Complex Detection*

In order to extract the RR-interval, it was necessary to first extract the QRS complexes from the EKG. The Pan-Tompkins[51] algorithm was implemented within the context of the Hadoop mapreduce paradigm.

5.2.2 *General algorithms*

5.2.2.1 *Point-to-Point Delta*

The Point-to-Point Delta algorithm simply obtains the difference between two successive values within a time series. It does not account for any particular characteristics of the signal or underlying time-domain characteristics. Example usages include the calculation of the beat-to-beat variance of the systolic pressures. This algorithm supports both the magnitude and time deltas from one point to the next.

5.2.3 *Sliding Window Algorithms*

This next class of algorithms utilizes a sliding window in order to facilitate the change (point) detection process. The non-sliding-window version of these algorithms tend to produce a single value for a series of inputs. So, they are of limited value when trying to detect changes within a particular signal. As a sliding window, however, these algorithms can be used to highlight a particular characteristic and its change over time.

5.2.3.1 *Sliding-Window Central Tendency Measure*

The Central Tendency Measure calculates the chaos or variability within a system by first calculating the second-order difference, then determining the percentage of points within a certain radius of the origin. This radius is user-defined and is typically dependent on the characteristics of the dataset and/or question being studied.

Given a set of data consisting of points a_t where t=1,2,3..., the CTM can be computed as:

$$\rho(i) = \sqrt{(a_{i+2} - a_{i+1})^2 + (a_{i+1} - a_i)^2}$$

$$\delta(d_i) = \begin{cases} 1 & : \text{if } \rho(i) < r \\ 0 & : \text{otherwise} \end{cases} \quad \text{where r is the user-defined radius}$$

$$\text{CTM} = \frac{\sum_{i=1}^{t-2} \delta(di)}{t-2}$$

On its own, the CTM algorithm only provides a single piece of information from an entire input dataset - the percentage of points within a certain radius. While this has been used with some success in medicine (e.g. detection of congestive heart failure using RR-intervals), it would not work in the computational pipeline since it requires a change series in which specific change points can be detected.

As a result, the CTM algorithm was adapted to use a sliding window. Along with the sliding window, the output of the algorithm was also changed to output the radius which contains N-percentage of the points, where N is a user-defined parameter.

5.2.3.2 *Sliding-Window Power Spectral Density*

There are several different possibilities for incorporating some type of frequency domain analysis with the most common being the Fast Fourier Transform, or FFT. In order to combine some sort of temporal aspect with the frequency domain analysis, the Fourier transform was applied on top of a sliding window, as in short-time Fourier transforms.

Additionally, in order to use the Fourier transform within the change detection pipeline, the Fourier transform needed to be distilled down to produce a time series in which change points could be detected. Currently, the power spectral density is calculated and then the root-sum-square of the PSD is taken. This results in a one-dimensional time series that is loosely representative of the underlying frequency components of the signal.

One of the key trade-offs with this type of algorithm is the resolution in both the time and frequency domains. Increasing the size of the sliding window will yield higher frequency domain resolution at the sacrifice of temporal specificity. On the other hand, decreasing the size of the sliding window will improve temporal specificity at the sacrifice of frequency domain resolution.

5.2.4 *Online vs. Offline Algorithms*

In this work, there is little distinction given to online (real-time) and offline (batch processed) algorithms. For research purposes, either class of algorithms can be used successfully to process and analyze physiological signal data. However, only online algorithms would be useful in migrating this system towards (near) real-time clinical decision support. Offline algorithms, by their very nature, preclude any sort of streaming uses because they require access to the entire data stream.

Where possible, online algorithms have been chosen instead of their offline counterparts in anticipation of uses for clinical decision support.

5.3 CHANGE (POINT) DETECTION

Each of the feature extraction algorithms effectively creates a new time-series data stream. For each of these streams, change detection algorithms are applied to detect any change points. Like most algorithms, change detection algorithms perform differently depending on characteristics of the underlying data, parameter choices, thresholds, etc. For this study, only one change detection algorithm was employed - a simple difference of means algorithm.

5.3.1 *Difference of Means*

The Difference-of-Means algorithm is extremely simple - for any given point p, the mean of the N points to the left is compared to the mean of the N points to the right. If this mean is greater than a threshold τ, then a change (point) has been detected.

For the rest of the analysis presented, two values of τ were used: 0.1 and 0.9.

5.4 MACHINE LEARNING

As previously mentioned, one of the main goals is to describe changes in a patient's clinical status or condition in terms of a set of detected changes in physiological signals. In order to address the problem of synchronizing the plethora of sensors, sliding windows

were again used to aid in the machine learning phase. After the application of a sliding window to the change points, a combination of supervised and unsupervised methods were used to examine the data.

Because there was a gold standard for the clinical alarms (see section 9.4), the primary machine learning approach was supervised learning. The unsupervised learning was used only as an exploration tool.

5.4.1 *Sliding Windows*

A sliding window was used to create the learning instances to be passed to either supervised or unsupervised learning algorithms. Each of these instances represents a set of change points from the various feature extraction algorithms. The sliding window approach was chosen because it addresses several issues:

1. Mitigates issues arising from unsynchronized timestamps
2. Simplifies integration of sensor and non-sensor data
3. Enables the use of algorithms that are not temporally aware

5.4.2 *Supervised Learning*

5.4.2.1 *Neural Networks*

Neural networks provide an interesting approach to supervised learning because they are a non-linear, statistical learning approach. As a result, they are able to be adapted to a wide variety problems when looking for patterns or other relationships within complex data sources. Neural networks are implemented as a series nodes or neurons between which connections are created. The learning occurs by the manipulation of these connections and their weights.

Neural networks have been used to learn a wide variety of problems such facial recognition[26, 40, 53] or speech recognition[43, 62], or the detection of congestive heart

failure when looking at the RR interval[15]. As with any learning algorithm or model, however, neural networks run the risk of being overfitted to the data.

Additionally, neural networks are oftentimes not preferred in fields where there is a desire or need to examine the model in more detail. Typically, the end user may be interested to see how or why the model decided on a particular classification. This usually occurs when researchers or scientists would like to know how or why a model arrived at a particular conclusion. Inherent within the design and function of a neural network, it is not possible to develop any explicit understanding of how or why a particular result was obtained.

5.4.2.2 *Support Vector Machines*

Support vector machines are another class of statistical learning algorithms that are also oftentimes used when looking at non-linear learning. Unlike neural networks, SVM's offer researchers the ability to look into the model and review the statistical basis for how the model classifies the input data. SVM's have also been used as one-class classifiers which makes them an interesting option for clinical alarms.

One-class classification is useful when the training data consists only of one class of data. After being trained on the class, the model is then used to classify input data as either a "match" to the learned class or an "outlier." As the naming convention implies, they are well-suited to outlier detection problems. In the clinical context, the one-class classifier may work well if there is a dataset that is expressive in terms of healthy patients and the goal was to identify those who are *not healthy*, as opposed to *sick*.

5.4.2.3 *Random Forests*

Random forests are yet another class of statistical learning algorithms that can potentially mitigate the need for complex feature selection. This may be especially useful when dealing with a large number of features. The random forest is essentially a collection of decision trees with which the most commonly occurring result is taken as the overall

result. As such, they provide implicit feature selection because relevant features are more likely to result in the same output.

5.4.3 *Unsupervised Learning*

5.4.3.1 *k-Means Clustering*

k-Means clustering is a commonly used unsupervised learning or clustering approach. The general idea is that a particular dataset can be segmented into k, typically unknown, classes. This algorithm is an iterative algorithm and is a specific implementation of the expectation-maximization set of algorithms. The initial clusters are chosen at random (or using some other heuristic). Based on a distance function, subsequent points are assigned to a particular cluster and scored. This process is repeated until the clustering reaches a steady state or some other condition or criterion is met.

While not specific to the k-Means algorithm, this approach enables researchers to "explore" their data and see if there are any interesting patterns within the data. For this research, the main goal of implementing and using the k-Means algorithm was to demonstrate the ability to integrate an unsupervised learning algorithm.

6 RESEARCH DETAILS

6.1 MAIN GOALS

The main goal of this project is to demonstrate the application of the 3-stage computational pipeline to improve clinical alarms. Due the computational resources necessary, the pipeline was overlaid on the Hadoop parallel computing framework to create MICC, the Medical Informatics Compute Cluster.

Clinically, the goal was to improve the detection of arrhythmias through a multivariate analysis of physiological waveform data. Specifically, this project attempted to decrease the number of false positives of ventricular fibrillation alarms by combining electrocardiograms with arterial blood pressures. The rate of false negatives were not considered because there was no gold standard with which to compare. In general, attempting to examine the rate of false negatives would require manual review of *ALL* collected data in order to generate the gold standard.

From an informatics perspective, the initial goal was to create a centralized platform with the following features:

1. Centralized data repository that is device agnostic
2. Integrated data access and visualization
3. Scalable storage and processing to accommodate large numbers of patients or high-resolution waveform data, or both
4. Clear pipeline to unify and simplify the approach to learning patterns in physiological signal data

The second goal is to create a "computable phenotype" rooted in truly objective, physiological signal data. That is, a model needs to be created that can be used to encode clinical conditions in such a way that they can be used by a computer. For example, if

you were to ask a cardiologist to describe atrial fibrillation, you will receive a qualitative description. From one patient to the next or from one cardiologist to the next, it is difficult to have a computer compare or operate on the concept of "atrial fibrillation."

However, if atrial fibrillation can be described in terms of quantifiable characteristics, it becomes easier for a computer-based algorithms to attempt to detect the onset or existence of this particular arrhythmia.

The underlying, general hypothesis of the work was:

> *The onset of any clinical condition can be described as a set of change points within a set of features extracted from sensor signals.*

6.2 MIMICII DATASET

The primary clinical problem is the detection of ventricular fibrillation through multivariate analysis of the EKG and arterial pressure signals. The dataset being used is the MIMICII dataset which was collected through a combined Harvard/MIT collaboration[54]. The data were collected at the Beth Israel Deaconess Medical Center from Philips bedside monitors within the intensive care environment.

The exact type and number of sensors varies from patient to patient due to differences in the underlying clinical condition and treatment protocol. However, most of the patient records contain at least one EKG signal and the arterial blood pressure. The MIMICII dataset has gone through several revisions and contains over 5000 patient records of waveform data and over 26000 patient records of clinical data.

For this work, a subset of the MIMICII dataset (approximately 400 patient records) is being used where every patient record has at least one EKG signal and the arterial blood pressure. Additionally, this subset also has the corresponding alarms that were generated by the Philips EKG monitors and reviewed by expert cardiologists from Harvard. Each alarm was reviewed by two different cardiologists with disagreements being arbitrated by a third. As per the IEEE standard, each valid alarm occurs within 10 seconds of the onset of the arrhythmia. The five arrhythmias are:

1. Ventricular Fibrillation
2. Ventricular Tachycardia
3. Extreme Bradycardia
4. Extreme Tachycardia
5. Asystole

The extreme bradycardia and tachycardia alarms were set using thresholds determined by the clinician and vary from patient to patient. The remaining three alarms detect ventricular fibrillation, ventricular tachycardia, and asystole, as commonly understood. The primary focus of this investigation was on ventricular fibrillation alarms as the clinical problem.

In the ventricular fibrillation subset, there are 143 patient records that have been imported into MICC. Each of these contains at least one V-Fib alarm (as determined by the Philips EKG built-in algorithms). The dataset was further distilled down to 138 patient records to include those that have lead II of the EKG. This simplifies the application of the computational pipeline to the EKG waveform data. Ideally, there would be an ontology backing the sensors and signals so that the algorithms could be applied to "any EKG lead," negating the need to specify or manage every single possible manufacturer/model combination.

The data were recorded in the MIMIC2 database at at 125 Hz despite higher sampling rates in some cases such as the EKG. Specifically for the EKG, the original signal was sampled with 12-bit precision at a very high sampling rate. The signal was then scaled and decimated which reduced the effective amplitude to 7-10 bits. Then, for every 4 consecutive samples, one was chosen and recorded using a turning-point compressor. As a result, the recorded signal was captured at 125 Hz (a period of 8 ms). However, the actual intervals between data points vary between 2 and 14 ms with an average of 8 ms. As a result, frequency domain analysis techniques may not be valid for the EKG signals.

6.3 HADOOP AND MAPREDUCE

6.3.1 *HBase Distributed Column-Store*

HBase is a column-oriented data store that is part of the Hadoop ecosystem. It was designed as a fully integrated solution for low-latency data requirements that also allowed for parallel mapreduce. Low-latency access to the data is important primarily for visualization purposes. When examining the data of a particular patient, an access time of seconds to minutes would make the overall workflow cumbersome and unusable. However, it is also important to be able to process the data using the mapreduce framework since the heart of the system is the ability to process and analyze large amounts of data.

6.3.2 *Data Storage Schema*

Because HBase is a column-oriented store, the schema in table 1 was designed to store the data. The rows are identified using the timestamp which is an 10-byte value that represents the number of picoseconds elapsed since the epoch (January 1st, 1970). The columns (TSID - Time Series ID) are a 4-byte Integer that is randomly generated and represents an instance of a signal. A signal instance is a particular type of signal (e.g. EKG Lead I, arterial blood pressure, pulmonary arterial pressure, etc.) from a particular patient. While a composite key of <patient identifier & signal type> could have been used, the creation of the TSID was chosen in order to save on storage overhead.

	<TSID>	<TSID>	...
<Timestamp>	<Value>	<Value>	...
<Timestamp>	<Value>	<Value>	...
...	...	...	...

Table 1: HBase Time Series Data Schema

The number of rows in this schema will grow with time but not the number of sensors or patients. The number of columns grow with the number of patients *AND* the number

of sensors being used. This should not be a problem because HBase is designed to store billions of rows and millions of tables[30].

HBase is being used to store only the time-series data; all of the metadata is being stored in a separate MongoDB database.

6.3.3 *Data Storage*

All of the data, whether waveforms or point-in-time data, are stored as <key, value> pairs, representing individual data points. In order to properly store individual data points from waveforms with frequencies such as 1024 Hz, the data needed to be stored with at least picosecond resolution. While this also allows for the storage of very high resolution signals, the main factor was the ability to capture periods such as 0.0009765625 seconds (976 562 500 nanoseconds or the period of 1024 Hz).

To maximize storage, picosecond timestamps were stored as a 10-byte value, 2-bytes storing the number of days from the epoch (January 1, 1970) and 8-bytes storing the number of picoseconds elapsed within the day. The limiting factor before an overflow is the 2-byte portion which limits the storage of timestamps through the year 2060.

The actual numeric data are stored as 8-byte double-precision, floating-point number (IEEE-754[1]). The performance and bandwidth costs of using a double-precision, floating-point number were not a concern though the added precision was important. However, any floating-point format would incur some loss of precision. By storing the numeric data as double-precision also leverages the bulk of algorithmic and numeric programming libraries that assume the data is already formatted as floating-point values of double-precision.

Both the keys and values described above are stored as binary data within HBase. Though this adds an additional level of (minor) complexity when trying to manually interact with the data or debug the system, it is much more efficient than storing the data as text.

6.3.4 *Metadata Storage*

Metadata in the current context consists of basic identifiers and descriptors of the sensor data such as patient ID, sensor ID, units of measure, etc. By separating out the metadata, a known issue is addressed when trying to associate data from a particular sensor to a particular patient. As patients are moved around the hospital for a procedure, they are not always reconnected to the same sensor. Because the metadata is separated, such "assignment" errors can be fixed without altering the underlying data.

Separating the metadata also makes it easier to integrate with an existing EHR or other clinical data management system. While all of the metadata have been stored inside of a MongoDB instance, these data can also be stored in the EHR while the sensor data remains within HBase.

6.4 LIMITATIONS

6.4.1 *Parallel Mapping Can Result in Data Anomalies*

As described in section 4.1.3, the data are split into multiple chunks with each being processed independently. Within the context of HBase, splits occur at the row level with each node processing a contiguous set of rows. Algorithms that require access to previous values will be "starting over" with each chunk. This can cause anomalies to be inserted into the data due to the lack of history.

To illustrate this effect, figure 2 is introduced and is a graphical representation of the calculation of the rolling mean. In figure 2, the three shaded color sections represent the split of this data into two equally sized chunks. The black and green lines represent the rolling mean of the underlying signal (not displayed) - the green is the actual rolling mean of the entire signal, and the black represents the rolling mean within each chunk.

Due to the splitting of the data into chunks with each chunk being processed independently, the rolling mean of each chunk starts out at 0 and slowly reaches the rolling mean as expected.

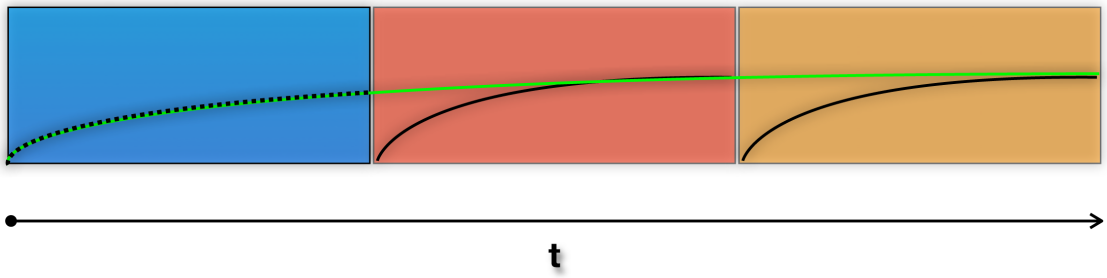

Figure 2: MR Splits - Data Anomalies

6.4.1.1 *Sliding Windows Unaffected*

The data anomalies due to the splitting does not affect those algorithms where the calculations are performed on sliding windows. During the map phase, the sliding windows are constructed with no other calculations being performed. The actual algorithm is applied during the reduce phase where the complete window is available to the algorithm.

6.5 FEATURE EXTRACTION

There are many different feature extraction algorithms that may be relevant to the detection of cardiac arrhythmias or other clinical conditions. However, only a small subset of these algorithms was chosen. The primary criteria for deciding which algorithms to use were driven by three primary goals: (1) Demonstration of the MICC pipeline and platform, (2) Adaptation of existing work, and (3) Improving the detection of cardiac arrhythmias, specifically ventricular fibrillation.

As part of the demonstration of the MICC pipeline and platform, it is also important to begin the process of categorizing different types of feature extraction algorithms. In chapter 5, the feature extraction algorithms were split into two main categories: (1) Domain-specific algorithms, and (2) General algorithms. However, even within each of these two categories, the algorithms can be further divided into smaller subgroups.

By examining the differences (and similarities) between feature extraction algorithms, concepts such as ontologies can be incorporated into the process, aiding in the automated application of algorithms to unknown problems in the future.

6.5.1 *Single-Pass Algorithms*

One of the long-term goals is to create a system and algorithms that are capable of being deployed in the clinical setting as a decision support tool. As a result, the focus is primarily on online algorithms using a single pass over the data.

Some of the algorithms that were used require some parameters such as the mean or maximum value. Using Matlab, these algorithms were re-implemented to use a rolling mean or a rolling maximum value.

6.6 CHANGE (POINT) DETECTION

Though there are many different types and approaches to change (point) detection in the literature[8, 24, 29, 36, 37, 38, 44, 45, 49, 52, 55, 60], this work focused on applying one particular change detection algorithm. The chosen change detection algorithm may not be suitable and sensitive to all changes in the extracted features. However, to simplify the overall problem, and specifically the number of possible combinations, this work focuses on the Difference of Means algorithm (5.3.1).

6.7 STATISTICAL / MACHINE LEARNING

Through the feature extraction phase of the pipeline, the overall amount of data for the learning phase is relatively very small. There was not a need to parallelize any single learning job so Weka[31] was chosen as the machine learning library. Using standard object-oriented programming techniques, Weka was integrated in such a way that it would be fairly easy to use a library such as R[61] instead.

6.7.1 *Unsupervised Learning*

The dataset used contains a gold standard which makes it an ideal candidate for supervised learning. However, in order to demonstrate the feasibility of integrating unsupervised learning and also to better understand the overall system and underlying data,

unsupervised learning techniques were applied to the dataset prior to any supervised learning techniques.

Unsupervised learning is oftentimes referred to as "clustering" and is an attempt at detecting or identifying (previously unknown) patterns within the data. Sometimes, the patterns may already be evident but further characterization is desired. Other times, it may be true, open-ended data mining where they may or may not be any relationships or patterns within the data.

For this particular dataset, the application of unsupervised learning techniques was not intended as a data mining exercise. Instead, it was intended to test and demonstrate that the 3-stage computational pipeline and underlying parallel computing infrastructure were capable of handling unsupervised learning jobs.

As a result, the only unsupervised learning algorithm tested was k-means clustering[10], specifically the version was implemented as a Weka library. When using the k-means clustering approach, the number of desired clusters must be specified as an input parameter. Two input parameters were chosen: 2 (clustering the data into "alarm" and "non-alarm" states) and 5 (clustering the data into each of the five alarms).

Another input parameter is the distance measure to use and the Euclidean distance was used. Upon completion of the clustering, the results were manually inspected by overlaying onto plots of the original waveforms.

6.7.1.1 *Extraction of Unsupervised Learning Instances*

Figure 3 is an example learning configuration file that was used as the primary input to the machine learning phase. This file is essentially a filter of all available change series data that will be used to create the learning instances input. It specifies the alarm(s) of interest, the patient identifiers (allowing for the division of the dataset into training and test sets, if necessary), and the change series of interest.

Because the unsupervised learning experiments that were executed did not require parallelization on the Hadoop platform, this file was used as input to a regular Groovy-

```
--Alarm--
V-FIB/TACH
--Patients--
a41993
a42062
a40570
a42157
a41225
a42010
a41900
--ChangeSet Group--
60d0683694f74d4c7b # [ABP / SPD 20.0] P2PDelta (Time) / DoM(w=125,d=0.01)
60d0683694f84d4c7b # [ABP / SPD 30.0] P2PDelta (Time) / DoM(w=125,d=0.01)
60d0683694f94d4c7b # [ABP / SPD 35.0] P2PDelta (Time) / DoM(w=125,d=0.01)
60d0683694fa4d4c7b # [ABP / SPD 40.0] P2PDelta (Time) / DoM(w=125,d=0.01)
--ChangeSet Group--
5cd0683694e44d4c7b # [ABP / SPD 20.0] P2PDelta (Value) / DoM(w=125,d=0.01)
5cd0683694e34d4c7b # [ABP / SPD 30.0] P2PDelta (Value) / DoM(w=125,d=0.01)
5cd0683694e24d4c7b # [ABP / SPD 35.0] P2PDelta (Value) / DoM(w=125,d=0.01)
5bd0683694e14d4c7b # [ABP / SPD 40.0] P2PDelta (Value) / DoM(w=125,d=0.01)
--ChangeSet Group--
63d0683694084e4c7b # [ABP / SPD 20.0 / P2P Time] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
63d0683694074e4c7b # [ABP / SPD 30.0 / P2P Time] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
63d0683694094e4c7b # [ABP / SPD 35.0 / P2P Time] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
63d06836940a4e4c7b # [ABP / SPD 40.0 / P2P Time] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
--ChangeSet Group--
62d0683694064e4c7b # [ABP] SlidingWindowPSD (w=1024,s=125,p=0.008) / DoM(w=125,d=0.01)
60d0683694fb4d4c7b # [ABP] SlidingWindowPSD (w=1024,s=250,p=0.008) / DoM(w=125,d=0.01)
62d0683694054e4c7b # [ABP] SlidingWindowPSD (w=1024,s=500,p=0.008) / DoM(w=125,d=0.01)
64d06836940b4e4c7b # [ABP] SlidingWindowPSD (w=1024,s=62,p=0.008) / DoM(w=125,d=0.01)
--ChangeSet Group--
60d0683694f54d4c7b # [II/RR Interval 0.65] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
60d0683694f64d4c7b # [II/RR Interval 0.6] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
5fd0683694f34d4c7b # [II/RR Interval 0.75] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
5fd0683694f44d4c7b # [II/RR Interval 0.7] SWCTM (w=128,s=64), gap 4000 / DoM(w=125,d=0.01)
--ChangeSet Group--
5fd0683694f04d4c7b # [II/RWave 0.6/1.001] P2PDelta (Time) / DoM(w=125,d=0.01)
5ed0683694ef4d4c7b # [II/RWave 0.65/1.001] P2PDelta (Time) / DoM(w=125,d=0.01)
5ed0683694ee4d4c7b # [II/RWave 0.7/1.001] P2PDelta (Time) / DoM(w=125,d=0.01)
5ed0683694ed4d4c7b # [II/RWave 0.75/1.001] P2PDelta (Time) / DoM(w=125,d=0.01)
--ChangeSet Group--
5bd0683694e04d4c7b # [II/RWave 0.6/1.001] P2PDelta (Value) / DoM(w=125,d=0.01)
5bd0683694df4d4c7b # [II/RWave 0.65/1.001] P2PDelta (Value) / DoM(w=125,d=0.01)
5bd0683694de4d4c7b # [II/RWave 0.7/1.001] P2PDelta (Value) / DoM(w=125,d=0.01)
```

Figure 3: Example machine learning input configuration file

based script[3] that extracted the desired data into Weka's standard file format (ARFF). The ARFF file was then loaded into Weka directly via the Weka graphical user interface, after which the unsupervised learning job was executed within the Weka environment.

The data were extracted into sliding windows (see section 5.4.1) which were encoded as learning instances. Because integrating multiple different sensors/algorithms is a core feature of the system, only those sliding windows with change points from two or more different change series were used as learning instances.

6.7.2 *Supervised Learning*

6.7.2.1 *Parallelization of Supervised Learning*

Though each individual supervised learning job did not require parallelization, there was a need to run many jobs due to changes in parameters of the features extraction algorithms, or of the learning instance extraction algorithms (section 6.7.2.2). In order to address this particular issue, the supervised learning phase was implemented within the Hadoop framework such that multiple learning jobs could be run in parallel. While Hadoop is intended as a parallel mapreduce framework for data processing, in this particular situation, it was being used primarily as a job coordination system where multiple jobs were being run concurrently, where each job was nearly identical except for particular parameters.

Given the number of parameters available to be tuned, there was a combinatoric explosion of the number of jobs to be run. Despite some pruning of the number of combinations, there were still upwards of hundreds of thousands of jobs to run. The Hadoop cluster became an ideal target for parallelizing this portion of the pipeline.

In order to streamline the process, the extraction of the learning instances was integrated directly into both the training and testing of the supervised learning model. The map phase took a configuration file as input and emitted learning instances extracted from the underlying data store. The reduce phase grouped all learning instances that corresponded to a particular learning job and then invoked the Weka library to do the actual learning.

While there are many ways to adapt the parallel mapreduce paradigm to this particular problem, the outlined approach operates under the assumption that the amount of data (i.e. the number of learning instances) is relatively low and that the training of the model does not require any parallelization. Should the amount of training data increase to a point where individual reduce tasks are unable to handle the computational

load, projects such as Apache Mahout[28] attempt to make parallelization of learning algorithms simpler to implement.

6.7.2.2 *Extraction of Supervised Learning Instances*

The same input file format (figure 3) as described in section 6.7.1.1 was also used as the input for the extraction of learning instances for supervised learning jobs. However, because these experiments resulted in many supervised learning jobs, parallel mapreduce was used to run multiple experiments in parallel.

The input file is parsed and combinations of the change series are emitted as map tasks. Each map task receives the alarm(s) of interest, patient identifiers, and a subset of the change series of interest. The map tasks then perform the actual job of extracting the relevant data from the data store and then group these data into sliding windows. Each sliding window along with an identifying key is emitted and sent to the reduce phase where the actual learning job is executed.

7 OVERVIEW OF RESULTS & DISCUSSION

Generally, the results of the described research can be broken down into two primary categories, those related to informatics and those related to engineering. Given the nature of the problem and infrastructure necessary to carry out the research, there were several non-trivial engineering hurdles to overcome. However, the bulk of the work was dedicated to solving problems in the field of informatics, regardless of the specific system or tools used.

7.1 ENGINEERING

7.1.1 *Data Storage*

Some design decisions were not dependent or related to Hadoop and, instead, were based on the high-level architecture such as the abstraction of all sensor data to time-series data. This also includes the separation of the data from the metadata, making everything easier to manage.

7.1.2 *Hadoop*

Hadoop served as the computational foundation, delivering both a scalable storage solution as well as a scalable computational engine. Many design decisions were made in order to adapt the approach specifically to Hadoop (including HBase). These were primarily to distributed storage of the data and optimizing the data-locality during parallel task execution.

7.2 INFORMATICS

7.2.1 *Data Management and Processing*

One of the large benefits of this approach is the separation of the sensor data from their associated metadata. This provides an extremely valuable flexibility in constructing the

high-level data management layer while providing an immediate ability to begin loading and working with the sensor data.

7.2.2 *Unsupervised Learning*

While unsupervised learning was not an objective of this research, we wanted to demonstrate the ability to integrate unsupervised learning algorithms. However, in doing so, we learned several valuable lessons regarding the high-level workflow and tooling necessary when attempting to apply any sort of unsupervised learning to clinical data.

7.2.3 *Supervised Learning*

Since the data contained gold-standard annotations, the primary approach was supervised learning of the data. It became very evident that sensor-based data is inherently imbalanced. There were also some issues regarding population-based learning versus learning by individual patients.

8 RESULTS & DISCUSSION: ENGINEERING

8.1 DATA STORAGE

8.1.1 *Timestamp Resolution*

As described in section 3.1, the major abstraction is to treat all sensor data as time-series data. Whether these are point-in-time readings (e.g. lab tests or glucometer measurements) or continuous readings (e.g. EKG), all data are stored as a measurement at a particular time. Originally, the data were stored using an 8-byte integer timestamp representing the number of microseconds elapsed from the epoch. This allowed for the storage of data up to 1 MHz in frequency resolution.

While this worked fine for point-in-time data and waveforms recorded at 125 Hz, it was unable to properly capture data from a NeuroSky EEG, which records data at 256 Hz. An easy "fix" was to store timestamps as an 8-byte double-precision, floating-point number. While this increased the precision and allowed for the capture of most of the data, some data was being lost due to the rounding inherent to floating-point numbers.

Given the specific dataset being used (MIMIC2[54]) and in anticipation of future data requirements, the decision was made to support timestamps up to picosecond resolution. This captures nearly any resolution that appeared during a quick search of device specifications. The main concern was the need to balance the supported resolution against the storage overhead to be incurred.

8.1.2 *Storage Overhead*

Because all of the data were being stored as key-value pairs, the size of each individual data point was a major concern. This becomes critical when considering high-resolution waveform signals that can have up to billions of data points per second. Using a 10-byte timestamp for a sensor generating data at 1024 Hz would result in a storage requirements

of roughly 864 MiB per day. Increasing each timestamp by only two bytes would increase the storage requirement to over 1 GiB per day. Extrapolating this over thousands of patients would result in an overall effect in the terabytes per day of data.

Once capturing and storing high-resolution continuous or waveform data is considered, the data storage requirements can quickly explode. This is the primary reason Hadoop was chosen as the computational foundation. Hadoop allows the entire system to scale quickly and easily, without the need to rebuild anything. As the amount of data grows, the only necessary change is the addition of new servers to support either the storage or computational load.

8.2 HADOOP AND HBASE

Given its success in both the web and enterprise markets, Hadoop became a natural choice as the foundation on which to build. There would be a natural explosion of data if waveform data were to be captured, especially when looking at very high-resolution data sources such as clinical EEG's which are capturing data into the GHz (billions of data points per second). With Hadoop, a system could be designed and implemented that does not require any modifications or changes regardless of the size of the data (and resulting cluster).

For example, for much of the development, various algorithms were implemented and tested on a single-node cluster, a laptop. This "cluster" only contained 10 patients worth of data. There was also a 6-node cluster that contained several hundred patients worth of data. Whether using one cluster or the other, none of the source code needed to be changed or altered in any way.

8.2.1 *Hadoop Data-Locality and Job Splitting*

As described in section 4.1.3, one of the primary benefits to Hadoop is data-local processing. That is, of all the nodes in a cluster, the processing of any chunk of data is primarily handled by a node that has a local copy of the data. This is a large differentiator from

other parallel computing approaches such as MPI because it minimizes the amount of network traffic related to synchronizing parallel tasks.

8.2.1.1 *Default Splitting Mechanism*

Originally, the default splitting mechanism provided by Hadoop and HBase was used. There was some consideration that this may cause a certain amount of artifact during the feature extraction and change-detection phases of the pipeline. However, the default splitting algorithm optimized job-splitting and the overall time it took to apply a particular algorithm to the data with minimal implementation complexity.

However, as expected, the default mechanism resulted in artifact during feature extraction that was significant enough to not continue with the rest of the computational pipeline. While working on an implementation of the Pan-Tompkins QRS detection algorithm[51] , the results were being visually inspected. During this process, some unexpected behavior was observed that resulted in erroneous detection of QRS complexes. After some significant testing and debugging of the QRS detection algorithm implementation, it was determined that the only explanation was artifact due to the Hadoop job splits.

While there was not a rigorous investigation or testing of the relationship between the artifact and default splitting mechanism, it was easily confirmed that the QRS detection algorithm worked as expected once the splitting mechanism was modified.

8.2.1.2 *Splits Aligned with Natural Gaps in the Data*

Experience working with physiological sensor data has highlighted that there are numerous problems with gaps and other anomalies within the data. While the MIMIC2 data is somewhat cleaner than what would be expected in a production clinical environment, the data still contained obvious gaps within the data. This was exhibited by completely missing data and also data that was impossible (e.g. negative pressure).

Within this section, "missing data" refers to data that is completely missing, where no values were recorded or stored.

During a review of the sensor data, few patients had sensor data that was completely contiguous for the entirety of the record. In other words, for nearly all patients, data that appeared to be "contiguous" still had missing data points. In order to better split the data to avoid injecting any artifact, the obvious choice was to align the splits with gaps in the data.

With the default splitting scheme, any given feature extraction job generated roughly 300-700 splits. By naively aligning the splits with gaps in the data, the number of splits increased to over 50 000 splits in many cases. Such a large increase in the number of splits actually created two major problems:

1. Too much overhead for the Hadoop job

   By default, Hadoop limits the information available to a particular job in order to minimize the job's metadata, which gets sent around to every node in the cluster. The "easy" fix to this problem is to increase the limit accordingly.

2. Splits were only 1 data point apart

   The bigger issue, however, was that most of these splits were separated by a gap of only a single data point. As a result, many splits were being created with too few data points or were potentially unnecessary.

By default, Hadoop limits the overall size of a parallel job with respect to the number of splits. Typically, a large number of splits should be avoided due to the overhead in starting up each parallel task. While the limit could have been increased, this would have been a sub-optimal solution since it addresses a particular symptom without addressing the underlying problem.

By addressing the larger problem of having gaps of only one or two data points, the problem of too many splits would also be addressed implicitly. With the addition of the

parameter *gap tolerance*, the size of the gap in data that would result in the creation of a new split could be controlled. Obviously, depending on the specified gap tolerance, the number of splits was dramatically reduced.

Gap tolerance is an algorithm-specific parameter and tuned accordingly. Either through a theoretical, analytical, or empirical process, the system was designed such that the gap tolerance was tunable based on the algorithm and not particular patients. In other words, the gap tolerance parameter is part of a particular algorithm's metadata, not a particular sensor or patient.

In conclusion, by adding the *gap tolerance* parameter, the problem of splitting the data was addressed. There was also the added benefit of improving the overall performance of the feature extraction algorithms.

8.2.2 *Duplication of Data for Processing and Viewing*

When reading any documentation related to Hadoop and its overall architecture, there is usually a theme of "duplication." Basically, the suggested, general approach is to duplicate the data as necessary in order to satisfy any other requirements. These other requirements typically include, but are not limited to, access latency or processing/computational performance. In other words, storage requirements tend to fall much lower on the list of priorities. If storage is an issue, the solution is to just add additional hard disks or servers.

The recommended approach was to set up two parallel clusters, each built around:

1. HBase for low-latency access
2. HDFS for batch processing

HBase was originally designed for low-latency access to data store within Hadoop and is obviously well-suited for this purpose. However, most batch processing is performed with data stored directly in HDFS, the Hadoop Distributed File System. By setting up two parallel clusters, it would be possible to provide low-latency access while optimizing

batch processing. Using a single cluster for both functions would decrease visualization-related performance if the cluster were running a concurrent batch-processing job.

Given budgetary restrictions, however, it was not possible to set up and maintain two clusters at the same time. Also, for the duration of this project, it was unlikely to have any visualization-related activities compete for overall computing resources. However, the system still needed to be designed as a general architecture that would allow both low-latency visualization and efficient batch processing, even if suboptimal.

The next section discusses many of the observations and lessons-learned when attempting to use HBase as both a source and sink of batch-processing data. Despite some difficulties, it was possible to construct a system that provided expected performance because there would be a maximum of two users at any point. Without the need for true concurrent users, the main focus was to design a "schema" that was conducive to analytics that could efficiently read/write data from HBase.

8.2.3 *HBase "Schema"*

As outlined in section 6.3.2, the schema consisted of storing data by timestamp and signal identifier (either the identifier of the sensor or algorithm). This is a very similar approach to how an open-source project, OpenTSDB[5] stores server-related diagnostic data for StumbleUpon[6]. While this addresses the storage and access of data for visualization, it was only part of the overall design when also combining analytics.

As with any standard approach, two tables were setup with the exact same structure within HBase. One was used as the source from which data was read and the other as the sink to which data was written. For example, consider the following series of feature extraction algorithms to be applied to the data:

Figure 4: An example flow of two feature extraction algorithms applied to a raw EKG signal. Red represents data stored in the "red" table and blue represents data stored in the "blue" table

Figure 4 above outlines the consecutive application of two feature extraction algorithms to an EKG waveform. In the first step, the source table is the "red" table and the target is the "blue" table. In the second step where the RR interval is extracted, the source table is the "blue" table and the target is the "red" table. Consequently, the "red" table contains both the raw EKG waveform (at 125 data points per second) and the RR interval data (at roughly 1-3 data points per second). The two types of data are stored with resolutions differing by two orders of magnitude.

This highlighted an interesting problem with respect to how HBase stores and reads data given the particular usage pattern. When applying the feature extraction algorithms to the data, the data is read using linear scans over a subset of the data. However, as a distributed key-value store, HBase reads in all possible row keys and then checks for the existence of a particular column. In order to better illustrate this, consider the following table:

	Sig A	Sig B	...
<Timestamp>	<Value>	<Value>	...
<Timestamp>	<Value>		...
<Timestamp>	<Value>		...
<Timestamp>	<Value>		...
<Timestamp>	<Value>	<Value>	...
<Timestamp>	<Value>		...
<Timestamp>	<Value>		...
<Timestamp>	<Value>		...
<Timestamp>	<Value>	<Value>	...
...	...	...	...

Table 2: Differing resolutions of data stored in the same table

Given the example in table 2, when reading in the data corresponding to column "Sig B," three additional row keys are scanned for every data point of interest. As a result, scanning over all of the data available for "Sig B" is about 3x slower due to the data

residing in "Sig A." In the case where the RR interval was being stored in the same table as the raw EKG waveform, there was a slowdown of approximately 100x.

The solution to this problem was to create tables based on the expected resolution of the data. The two guidelines governing the tables to be used are:

1. Source and target tables must be different
2. The target table must only store data of comparable resolution

In practice, this resulted in three tables. One to store the raw source data (125 Hz), and two to store the data at roughly 1Hz.

Once HBase's architecture was understood, it was easy to tune the software and table schema to provide for an efficient and scalable platform to store and process clinical signal data.

8.2.4 *Hadoop for Non-Technical Users*

One of the major goals of the project was to create a platform that handled most, if not all, of the plumbing necessary to interact with clinical signal data. Hadoop turned out to be a very good framework on which to build because it allows the abstraction of nearly all of the plumbing. Users looking to implement any sort of feature extraction algorithm needed to only worry about creating functions that received a sequence of key-value pairs and writing key-value pairs.

Given the structure of the framework, it was also very easy to integrate unit-testing into the development process so that the algorithms could be tested on a known set of data prior to being applied to the entire dataset. This proved to be invaluable when debugging more complex algorithms such as QRS detection.

9 RESULTS & DISCUSSION: INFORMATICS

9.1 DATA MANAGEMENT AND PROCESSING

One of the core informatics problems to be addressed was the overall management of sensor data with respect to the rest of a patient's clinical record and some of the problems when trying to collect data from the clinical environment. These problems are:

1. Integrating waveform data with existing electronic medical records
2. Linking sensor data to the proper patient (record)
3. Semantically link the sensor data

9.1.1 *Separation of Data from Metadata*

The core feature to address the above issues is the decoupling of the metadata from the data. At a minimum, only three elements of metadata are necessary to retrieve sensor date:

1. Time Series ID (see section 6.3.2)
2. Start timestamp
3. End timestamp

By separating the metadata from the data, it was possible to construct an extremely simple design regarding the storage of sensor data. A randomly generated 4-byte integer value was chosen to uniquely identify any given "signal" (what is referred to as the Time Series ID, see section 6.3.2) which included both raw data as well as outputs of feature extraction and change detection. During the development process, the storage of and interaction with the metadata was changed several times. However, the sensor data and its associated schema remained unaffected. The only situation that necessitated a change

in the sensor data schema was the migration from microsecond to picosecond resolution of the timestamps.

9.1.2 *Integration with Existing Electronic Medical Records*

While there has been an increasing rate of adoption of electronic medical records[21], these records were designed to capture and manage clinical data generated by clinicians. They have been adapted to manage some sensor data such as lab values or other point-in-time data (e.g. glucometer, non-invasive blood pressures, basic vitals signs). However, they are ill-equipped to store continuous sensor data, especially waveform data.

However, with this approach, there are only three metadata elements that need to be tracked within the traditional EMR. While it would be easy enough to store a unique identifier and start/stop timestamps, it may be easier to deploy a web service that acts as a bridge between the EMR and MICC. This service would operate very similarly to an URL-shortening service such as bit.ly[2] by generating a unique, random identifier that links to the complete metadata record (which, in this case is the TSID, and start/stop timestamps).

9.1.3 *Integration with Ontological Approaches*

Another benefit of this approach is the ability to integrate with some of the latest developments in data management and semantic computing. One approach that is specific to biomedicine is the Health Ontology Mapper (HOM)[64]. This approach uses the Object Web Language (OWL)[47] file format as a vehicle for defining, storing, and transmitting "rules" that govern the semantic linking of disparate biomedical data. While it was primarily created to harmonize standard terminologies such as SNOMED, LOINC, or RxNORM[12], it was adapted and slightly modified to create a methodology for easily managing sensor data.

9.1.3.1 *Health Ontology Mapping - HOM*

Because the details of the HOM approach have not been published in their entirety, a quick overview is provided:

The HOM approach is a mechanism where traditional ontological concepts and relationships can be augmented with data processing rules. These rules are codified directly within the ontology though attributes referred to as *HOM tags*. In the current HOM approach, these tags can codify rules such as formatting (e.g. adding the period to an ICD-9 code which is usually stored without the period), or matching text against another terminology (e.g. matching semi-structured text data against standard terminologies such as LOINC).

9.1.3.2 *Modifications to HOM*

In this application of the HOM approach, an additional *HOM tag* was added that is specific to the management of sensor data, *HOM_Conversion*.

To provide a more concrete example, assume that there is a base "ontology" that is essentially a simple taxonomy:

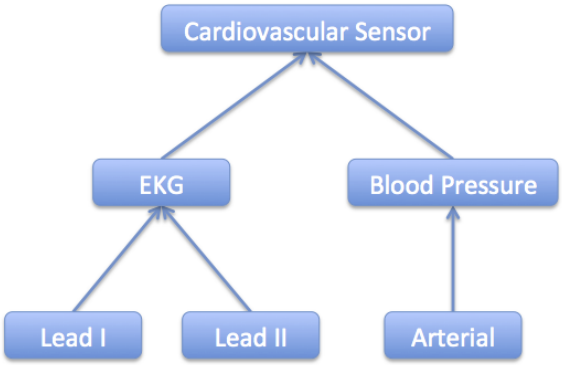

Figure 5: Example sensor taxonomy

To make the example more complete, assume that the above concepts have an "is_a" relationship to parent concepts (denoted by the arrows) and have attributes specifying

the expected units of "mV" for the EKG and "mmHg" for the blood pressures. Then, for any particular sensor, the HOM approach defines at least two additional attributes:

1. HOM_Parent - a unique PURL that resolves to a particular ontological concept

   In this specific example, the HOM_Parent tag will be a PURL that points to a concept in the above ontology (figure 5). This allows the concept of a specific EKG sensor, such as the Philips EKG monitor used at the Beth Israel Deaconess Medical Center, to inherit all of the ontological information from the parent ontology. In this case, the elements being inherited are the "is_a" relationship to a parent as well as the expected units.

2. HOM_Conversion - contains the conversion from stored units to the units specified by the sensor ontology

   Since there is no common standard to the units of measure that a device may store raw data, this tag provides a mechanism for converting the data such that it is stored in the same units as the parent ontology concept. For example, the device may transmit and store raw data as Analog-to-Digital (ADC) counts with a particular gain and offset in order to convert the data to the physical units of mV or mmHg. This tag can also be used to encode any necessary normalization.

A very minimal demonstration of the above methodology was implemented in order to demonstrate the feasibility of such an approach for managing and normalizing sensor data. However, this approach was not incorporated into the overall pipeline. However, incorporating such an approach into the pipeline would allow algorithms to refer to signals purely based on high-level concepts. For example, instead of referring to specific signals, a "Beat-to-Beat Time Delta" algorithm could be implemented to use any "Beat-to-Beat" signal as a source. This would allow the algorithm to use the EKG, pulse oximeter, or arterial blood pressure to extract the beat-to-beat variation.

Regardless of the methods used, this ultimately only affects the metadata. The sensor data can be collected and stored even without a clearly defined data management system in place.

9.2 NORMALIZATION OF LEARNING INSTANCES

Each learning instance consists of:

1. Alarm or Non-Alarm, indicating if it was a true positive or false positive
2. Set of detected change points within the specified window of time

The change points are originally extracted from the data as a real number, a positive or negative value indicating the direction and magnitude of the detected change. As a result, some normalization is necessary prior to feeding the data into the learning algorithms, whether supervised or unsupervised.

9.2.1 *Magnitude of Changes*

The first pass at data normalization was to take the magnitudes of the change points, take their absolute value, and then normalize them to the interval [0,1]. While the normalization to the [0,1] interval is fairly trivial, the first issue encountered was that there may be more than one detected change point within a single change series. Because the learning instances are extracted using sliding windows, it is possible that a single learning instance (i.e., a single sliding window) may be larger than the sliding window size used during the change detection phase. In these situations, there may be more than one change point for any given change series.

As a result, normalizing the magnitude of changes requires more than just mapping the data to the [0,1] interval. Options include taking some sort of aggregate of the magnitude of all change points within the window such as the mean, median, max, min, etc. Given that these change points should be relatively similar, the mean of the magnitudes were extracted and then normalized to the [0,1] interval.

The main problem encountered when trying to use the magnitude of changes over the entire set of data (i.e., over all patients) was that the magnitudes varied wildly between patients. In other words, when applying the same feature extraction / change detection chain of algorithms to the same data across multiple patients, the range is so large that changes from a specific patient may effectively be zero. For example, for patient A, the highest, average magnitude was 3 orders of magnitude higher than patient B. As a result, after normalization, patient B's data essentially had a value of 0.

9.2.2 *Number of Changes*

Given the wide range of magnitudes due to various combinations of sensors, algorithms, and patients, an alternative to normalizing the magnitudes is to count the number of change points detected. This was a very trivial implementation and provided much better performance than using the actual magnitudes.

9.2.3 *Binary Change - Yes or No*

Since the number of change points had already been extracted, it was another trivial normalization to extract a binary *yes* or *no*, depending on whether or not any change points were detected within the particular window. A *yes* basically indicates that at least one change was detected within the window. From a clinical execution standpoint, this would be implemented but an algorithm that would trigger an alarm based on the first detected change. While the duration or elapsed time from a detected change to the alarm generated by the monitor was not measured, this would be an interesting data point to see if the binary change approach may provide satisfactory sensitivity and specificity while also decreasing the delay in detecting the alarm state.

9.3 UNSUPERVISED LEARNING

Unsupervised learning methods, such as clustering, are an interesting tool when attempting to mine data for new patterns. These methods allow researchers and scientists

to explore their data for new relationships or patterns that can be linked to existing theory or used to create new theories. In other words, unsupervised learning methods facilitate the discovery process. These discoveries can then be compared against existing knowledge or used as the basis from which to generate new knowledge.

Given that there was a gold-standard against which to compare, unsupervised learning was initially used as an initial test of the underlying data management mechanisms. The Weka[31] machine learning toolkit was used given prior experience and ease of integration. Specifically, the provided k-means clustering algorithm was used to try and find two or five clusters within the data. The k-means algorithm was run only on a subset of the total available patients.

Once the learning instances had been extracted from the data, they were fed into the k-means clustering algorithm. The learning itself was relatively very fast and took very little time to cluster. However, once the clustering was finished, it was quickly evident that the clustering itself was the easy part of the problem.

The difficulty comes from the need to "overlay" the clustering results over the real data. At a minimum, there was a need to be able to take any given learning instance and view the actual sensor data that corresponded to that particular learning instance. While this was not a difficult feature to add, it immediately highlighted a workflow issue. If this were a true unsupervised learning exercise to see how the data would cluster as part of a discovery effort, there was no way to reasonably inspect all of the underlying sensor data.

Any sort of unsupervised learning requires a solid workflow to actually review and sift through the results of the learning process. This is especially important when attempting to use unsupervised learning as one of the first methods for interacting with a brand new, previously unanalyzed dataset.

Learning instances had been defined to be any window that contained one or more detected changes in any of the source feature series. This results in an overwhelming number of learning instances, most of which contain a single change. Intuitively, the

majority of these learning instances would probably constitute a false positive; that is, a change in a feature series that does not correspond to any actual changes in a patient's clinical status.

The next step was to try and reduce the number of learning instances, either through a filter or by resampling the learning instances. One of the primary goals was to lay the foundation for a multivariate approach to improving clinical alarms. As a result, it was decided that any learning instances that consisted of only one change point would be filtered out. These, by definition, were instances that were univariate.

While it may be safe to assume that this sort of filtering is acceptable, it does expose a potential issue with making such decisions - what if there was a single feature extraction algorithm that performed extremely well at detecting a particular (unknown) clinical condition? Regardless, this filter was applied to the data and was successful in decreasing the number of learning instances but not to a level where the results were any easier to sift through.

After several iterations of attempting to narrow down the number of learning instances, the focus of this particular series of experiments changed from the unsupervised learning itself to highlighting potential workflow problems and technological solutions. The MIMIC2 dataset already contained a gold standard (see section 9.4) to which the machine learning results could be compared and the overall goal was to improve clinical alarms, not to engage in the discovery process.

Because the primary gating factor is the lack of domain experts to manually review and annotate the clustering results, the crowd-sourcing model was briefly examined to see if it could be applied to this particular problem. The dominant issue is getting qualified people to spend time reviewing the results. When looking at EKG's and blood pressures, an obvious, ideal candidate is someone trained in cardiology. At the very least, a reviewer should have some understanding of cardiovascular physiology.

9.3.1 *Tooling and Workflow*

Based on experience, any sort of technical solution must have at least three main functions:

1. "One-click" overlaying of unsupervised learning results over actual sensor data
2. Easy annotation of clustering results in a computable format
3. Enables collaboration with other reviewers

As described earlier in this section, the most important feature is the ability to overlay the learning results over the actual data in order to manually interpret or verify the learning output. Once the reviewer can examine the data, s/he will need a method for annotating the results. This may include a simple "yes" or "no" if the assigned cluster makes sense or not. However, it will most likely include richer data such as the observations of the reviewer. Ideally, these annotations would be recorded in such a way that they may be fed back into a future unsupervised learning exercise or experiment. Lastly, it is unrealistic to expect that a single person will review thousands to hundreds of thousands of data points stemming from a single unsupervised learning experiment. As a result, it is important to have a system that supports collaborative review of the results.

9.4 SUPERVISED LEARNING

The MIMIC2 dataset contained a subset of data specifically geared toward arrhythmias, and specifically associated alarms. These data had been recorded with alarms generated by the bedside monitors and manually reviewed by expert cardiologists. Each alarm was reviewed by at least two cardiologists with adjudication by a third. As a result, the bulk of the focus was on supervised learning of the alarms in an effort to improve the detection of ventricular fibrillation or ventricular tachycardia.

9.4.1 *False Negative Alarms*

The cardiologists that reviewed the data only looked at EKG and other sensor data from those segments surrounding recorded alarm events. In other words, they started with alarms generated by the bedside monitors and determined if these alarms were true positives or false positives. As a result, there is no way to track the number of false negatives without a separate, manual review of all available data.

9.4.2 *Feature Selection*

From the perspective of the 3-stage pipeline, the inputs to the learning phase were called *change series*. However, for the purpose of machine learning, these change series are usually referred to as *features*. Feature selection is the process of distilling the input data, removing redundant or spurious features that may have a negative effect on the overall classification process. In the context of the pipeline, feature selection is the process of deciding which change series to include in the learning phase.

If the "incorrect" features are selected and used as part of the learning phase, the resultant model may be overfitted to the training data. In this situation, the model has been so specifically tuned to the training data that it may perform extremely well on the training data, but it will be unable to correctly classify any unseen data (to an expected level of accuracy or performance). The feature selection process may also provide insight or other visibility into the theory underlying the model.

As a result, feature selection is an important step of the learning phase of the pipeline. However, this ultimately depends on the specific learning algorithm being used. For example, random forests can implicitly handle feature selection based on the parameters used. Because random forests were the primary choice of learning algorithm, there was not as much focus on the feature selection process because it is implicitly embedded within the random forest.

9.4.3 *Data Imbalance*

Once the learning instances were extracted for the purpose of supervised learning, it was observed that the data were imbalanced when looking at the number of true positives to false positives. Of the 409 learning instances across all patients, 280 represented non-alarms (false positives) and 129 represented alarms (true positives). There was over a 2x difference in false positives to true positives. From the perspective of the supervised learning algorithms, there were 129 positives and 280 negatives to be classified.

Supervised learning algorithms attempt to classify the learning instances such that overall accuracy is maximized. When dealing with clinical alarms, the most "expensive" error is the false negative, the failure to detect a life critical alarm. However, false positives also have an associated cost in the form of alarm fatigue and their effects on overall patient recovery. As a result, not all errors can be treated the same and weighed equally. Improving overall accuracy may result in a suboptimal classifier for clinical purposes.

Two common approaches for solving the data imbalance problem are *S*ynthetic *M*inority *O*versampling *TE*chnique (SMOTE) and Cost-Sensitive Classifiers. SMOTE, as the name implies, oversamples the minority class in an attempt to "balance" the two datasets. As such, SMOTE is used to pre-process the data, prior to training the model. Cost-Sensitive Classifiers (CSC), on the other hand, wrap other learning algorithms and assign a "cost" to each class of results (true positives, true negatives, false positives, and false negatives). This allows the tuning of the results towards or away from a particular class of results.

The SMOTE technique did not perform as well as the Cost-Sensitive classifier in the experiments, both techniques using random forests as the learning algorithm. The problem with any oversampling approach is that the oversampling draws from a much smaller pool of samples (the reason why oversampling is needed in the first place). However, in doing so, the "balanced minority" samples may be much more homogeneous than "majority."

For example, within the data, there was a 2x difference between the non-alarms and alarms. As a result, the SMOTE algorithm was parameterized to oversample the alarms by a factor of 2. This would balance the two datasets such that there were 280 non-alarm instances and 258 alarm instances within the input data. However, doubling the number of alarm instances in the data risks skewing the model.

The Cost-Sensitive Classifier, on the other hand, yielded better performance while also being more intuitive. Given the clinical context, the cost of a false negative (missed alarm) is much more expensive than the cost of a false positive (false alarm). By assigning increased costs to false negatives and false positives, the Cost-Sensitive Classifier is able to "tune" the results and indirectly address the imbalance issue.

By increasing the cost of false negatives relative to false positives (and true positive/negative), it was possible to bias the model towards classifying any given input as a positive. Naturally, this increased the number of false positives. The key in tuning the relative cost was to track the increase in true positives versus false positives. Several experiments were conducted where the cost of a false negative was steadily increased while keeping all other costs constant. Figure 6 is a plot of the sensitivity and specificity as a function of the relative cost of a false negative. As evident, there is a crossing point where change in sensitivity and specificity no longer result in an overall improvement in the C statistic.

In Figure 7, a pair of sensitivity and specificity curves from figure 6 were plotted, along with their rates of change. This shows a rough relationship between how quickly the sensitivity increases as compared to how quickly the specificity decreases. Aside from overall performance, the relative difference between the two rates of change could also be used to decide the cost that yields the preferred random forest model.

Generally speaking, the Cost-Sensitive Classifier can be parameterized using "real-world" costs such as the (fictional) fact that a missed alarm costs a hospital 2x more than a false alarm. On the other hand, CSC's can be parameterized based on an empirical approach and looking at the C statistic, accuracy, or other measure of performance.

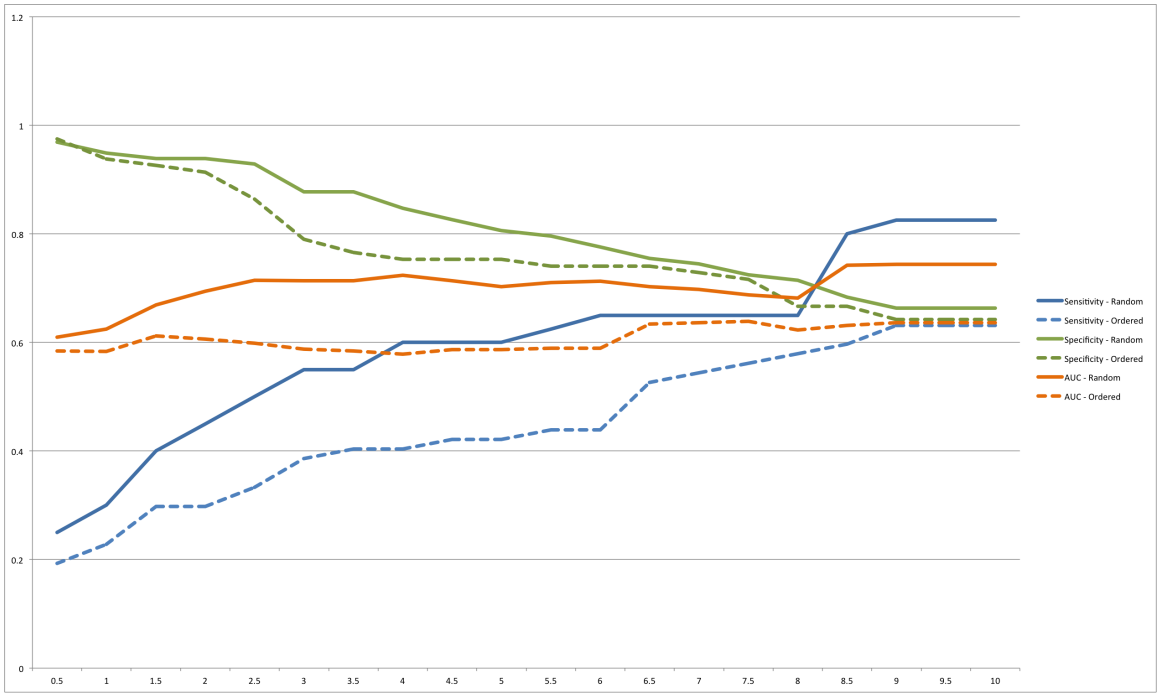

Figure 6: Performance as a function of increasing FN cost

9.4.4 *Population-level vs. Patient-specific Learning*

9.4.4.1 *Population-level Learning*

All of the results discussed to this point have revolved around population-based models. That is, all of the alarm data were split into two sets, one for training and the other reserved for testing. However, there are two methods for splitting the data:

1. Randomly sort all patients and then assign patients (and their alarm and non-alarm instances) to either a training or test set

2. Randomly sort all alarms and non-alarms instances and then assign instance to either a training or test set

In some of the figures and tables, the former are referred to as *ordered* and the latter as *random*, indicating that the alarms were either ordered/grouped by patient or were randomly sorted.

From a clinical perspective, grouping patients (and their respective alarm and non-alarm data) into either a training or test set is similar to other clinical research. This

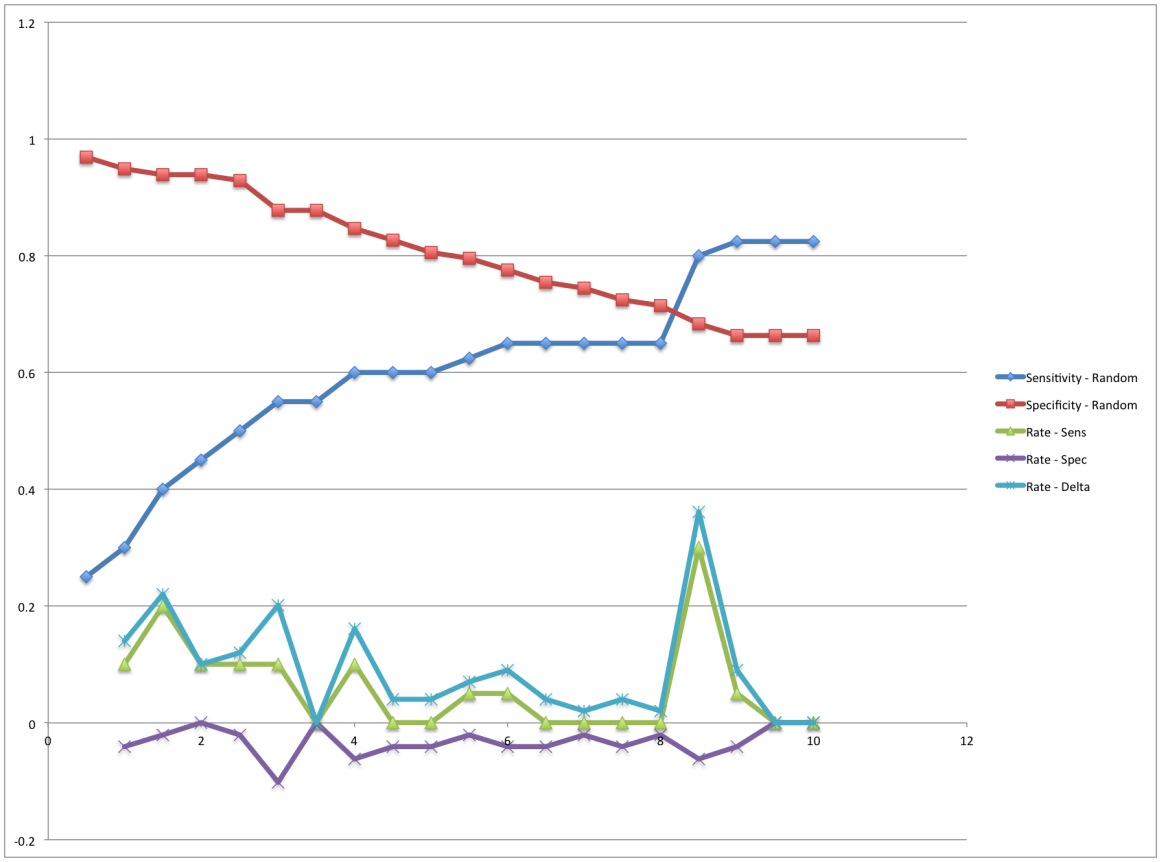

Figure 7: Relative rate of change of sensitivity and specificity as a function of increasing FN cost

allows the building of a model on one population of patients and then test or applied to a different (e.g. future) population of patients. Given this parallel, the initial experiments focused on dividing the data at the patient level. However, splitting in this manner yielded generally poorer performance (see table 3 for a representative result).

	Sensitivity	Specificity	AUC
Randomly split	0.822	0.755	0.849
Split by patient	0.544	0.707	0.563

Table 3: Representative example of the effect on performance of splitting the training (66%) and test (33%) learning instances by patient or randomly (with a random forest)

The alternative was to take all of the alarm and non-alarm data, regardless of the patient, randomly sort them and then divide them into a training and test set. While this

approach is very similar to the patient-level splitting discussed earlier, the key difference is that the model being trained gets trained and then tested on the "same" patients.

For example, if the dataset consisted of 3 patients (PtA, PtB, and PtC), each with 6 (non-)alarm instances, the first approach would divide the data such that the training set consists of all data from PtA and PtB (a total of 12 learning instances). The test set would contain only data from PtC (6 learning instances). Obviously, this assumes a 66%/33% split between training and test sets.

Using the second approach, the 18 learning instances from all three patients would be randomly sorted together. Again, with a 66%/33% split, the training set would contain 12 learning instances and the test set would contain 6 learning instances. However, the training data would contain a random number of instances from PtA, PtB, *and* PtC. This is also true of the test data.

With the latter approach, any characteristics specific to one patient or another have the opportunity to influence the overall model. With this alternative splitting scheme, the same experiments as with the patient-level splitting were repeated. The random forest models exhibited better performance (sensitivity, specificity, and C statistic) compared to the patient-level approach (see table 4). In every case, the AUC is better when randomly splitting the learning instances.

The distribution of the (non-)alarm learning instances within the data offers a potential explanation for such behavior. Figure 8 contains a histogram of the number of learning instances (alarms and non-alarms) for patients in the dataset. Of the 143 patients in the dataset, 73 have only a single non-alarm. In other words, 51% of the patients have only a single data point to contribute to the model and it is a false alarm. Only 41 of the patients had 1 or more true alarms and only 15 had 3 or more true alarms. So, when training the data on a random subset of patients, almost half of the patients (statistically speaking) contain only a single data point, a false alarm.

Cost of FN	AUC - Split per Pt	AUC - Randomly Split
0.5	0.584	0.610
1.0	0.583	0.624
1.5	0.612	0.669
2.0	0.606	0.694
2.5	0.599	0.714
3.0	0.588	0.714
3.5	0.584	0.714
4.0	0.578	0.723
4.5	0.587	0.713
5.0	0.587	0.703
5.5	0.590	0.710
6.0	0.590	0.713
6.5	0.634	0.703
7.0	0.636	0.697
7.5	0.639	0.687
8.0	0.623	0.682
8.5	0.632	0.742
9.0	0.637	0.744
9.5	0.637	0.744
10.0	0.637	0.744

Table 4: Comparison of splitting training and test sets randomly or by patient using a random forest with cost-sensitive classification in the context of increasing the cost of false negatives

9.4.4.2 *Patient-specific Learning*

Given this distribution of the data, the next step was to take those patients with at least 3 true alarms and see if it was possible to construct a patient-specific model. In this case, a model was trained to data from a single patient and then tested on data from the same patient. To mimic real-world behavior, the learning instances were sorted temporally and then 66% were selected for the training set. This would be equivalent to training on data from the patient over a period of time and then applying the model to future data from the same patient.

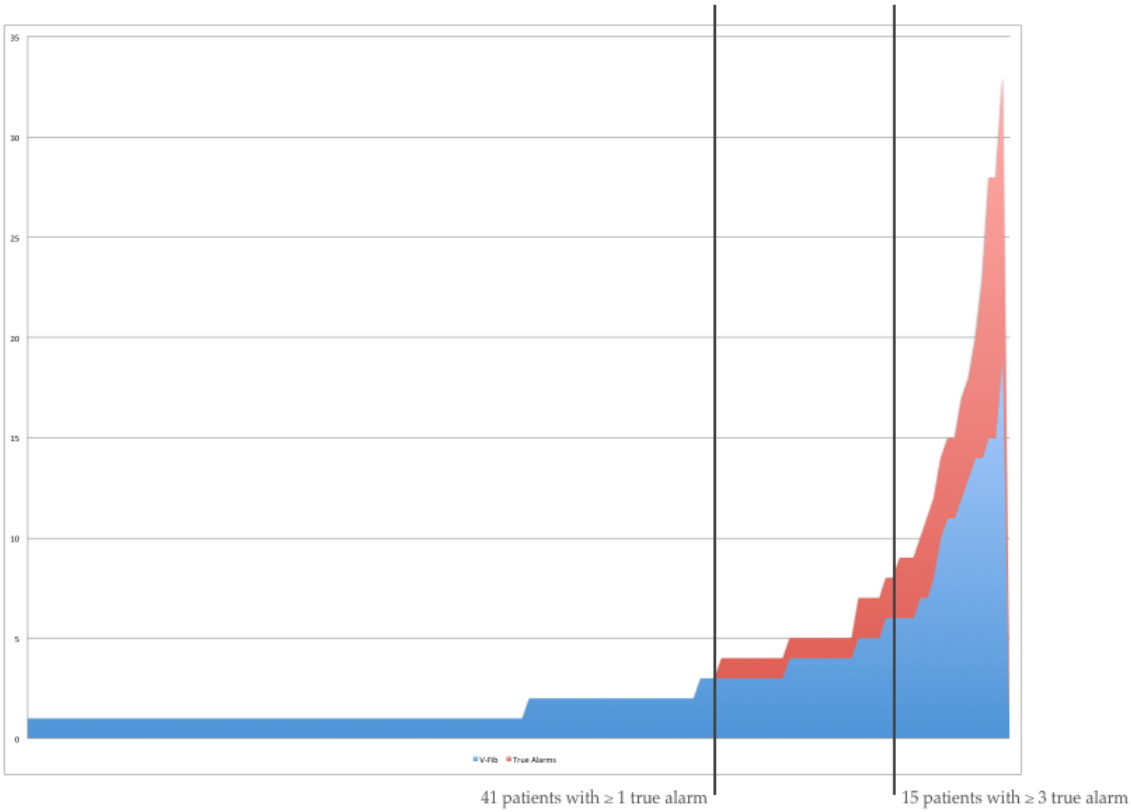

Figure 8: Histogram of the true positive (red) and false positive (blue) V-Fib alarms

The main issue with this approach was that few patients had both false alarms and true alarms within the data. Patients either had all false alarms or all true alarms. After running patient-specific models, the data were aggregated which yielded 23 true alarms and 23 missed alarms with 1 false alarm over 15 patients. This is a sensitivity of 0.5 and a specificity of 0.75, generally very poor performance. However, if the model were constructed to always classify the data as an alarm, regardless of the input data, the result would be 46 true alarms, 0 missed alarms, and 1 false alarm. This yields a sensitivity of 1 and a specificity of 0. This highlights the fact that the dataset is not large nor diverse enough to make this a truly meaningful exercise.

9.4.5 *Neural Networks, Support Vector Machines, Random Forests*

When looking through the literature, several supervised learning techniques are more popular than others. These include neural networks, support vector machines, and ran-

dom forests. This, however, is not to say that there are not other common approaches such as Bayesian networks and other statistical models. However, given their success in the literature, these three were chosen.

9.4.5.1 *Neural Networks*

Neural networks are especially interesting when applied to biomedical problems given their biological roots. However, their very nature also makes them difficult to apply in clinical practice because the models they generate are inherently vague or unclear. The power behind neural networks lies in the various weights between nodes. This makes them especially useful in problems of high dimensionality and where many other techniques breakdown.

Clinical scientists and researchers, however, must be able to examine the resultant model (of any supervised learning process) and understand its behavior and performance before using the model to treat patients. Though neural networks may provide superior empirical performance in many cases, there is no model that can be easily inspected and understood. Consequently, neural networks are oftentimes referred to as "black boxes" where data comes in on one side and a result pops out on the other side; however, the how or why behind the classification remains largely unknown.

In any case, a neural network was implemented, mostly as an exercise to show that this pipeline can integrate nearly any type of supervised learning algorithm. However, it was interesting to see how a neural network may perform, recognizing that it was a conscious decision not to invest heavily in this particular approach.

9.4.5.2 *Support Vector Machines*

Support vector machines (SVM) have become a popular learning approach [14, 25, 41, 48] because they are capable of handling multi-dimensional learning problems and are statistically based. This allows researchers and scientists to "peek" inside the model to gain additional insight or see why a particular model behaved a certain way.

Generally, SVM's did not perform as well as random forests. Specifically, the performance of the top two sets of features is shown in table 5.

	Sensitivity	Specificity	AUC
Feature Set A	0.522	0.942	0.732
Feature Set B	0.567	0.895	0.731

Table 5: Top two performing sets of features using an SVM (with minimal SVM parameter tuning)

As with neural networks and most other supervised learning approaches, feature selection can be an important factor in the overall performance of the SVM. This was not a specific area of focus so there was no effort to rigorously tune SVM parameters or features. Instead, SVM's were another data point in demonstrating that the pipeline could integrate any type of learning approach.

9.4.5.3 *Random Forests*

Random forests (RF) are another popular learning approach, including biomedical informatics [23, 35, 57]. One of the interesting benefits of the RF approach is that there is built-in feature selection, depending on the RF parameters. Because the classification of an RF algorithm is the most frequent result from hundreds or thousands of individual trees, the algorithm effectively performs a feature selection. Several experiments were conducted that involved numerous parameter options but, generally, a very low number (2-5) of features per tree and many trees (hundreds to thousands) were used (see table 4).

Given the implicit feature selection and overall preliminary results of random forests, RF's were chosen as the basis for subsequent experiments to identify and further understand other characteristics of the 3-stage pipeline.

9.4.5.4 *Effects of Feature Extraction Thresholds*

Most of the feature extraction algorithms have parameters that can be tuned and are specific to the particular algorithm. When tuning various parameters, the end effect can be grouped into several categories, those that affect sensitivity, specificity, or both (or neither). Ideally, the parameters increase both sensitivity and specificity. However, this is often not the case. With clinical alarms, the tendency is to increase sensitivity at the cost of specificity since the cost of a false negative (undetected clinical condition) is typically more expensive than the cost of a false positive (false alarm).

One argument regarding false positives is that they should "fall away" when other, multivariate data are incorporated into the learning process. While it is unfeasible to test this argument as a generality, several experiments were conducted where a particular parameter was varied and the resulting C statistic was compared. This gave provided a sense of the general performance of a particular algorithm as its parameters were modified.

Generally, as thresholds were lowered, the sensitivity increased and the specificity decreased. However, this was not always the case. For example, in one set of experiments, the arterial pressure and EKG waveforms were used as the input signals. While any algorithms with varying parameters can be used, the ones of particular interest are the Systolic Pressure Detector (SPD) and Point-to-Point Time Delta (P2PDelta-Time) when applied to the arterial pressure waveform. For the SPD, the thresholds of 20.0, 30.0, 35.0, and 40.0 were used. Lower threshold would result in an increased sensitivity for the detection of "systolic peaks." When sorting all of the machine learning runs (using random forests) by sensitivity, the top 100 combinations of features all have the SPD with a threshold of 30.0 or 35.0 and not the expected lower threshold of 20.0. Specifically, the particular algorithms and thresholds were:

1. Sliding Window CTM of P2P Time Delta of SPD with threshold 30.0
2. P2P Time Delta of SPD with threshold 35.0

3. P2P Amplitude Delta of SPD with threshold 35.0

Intuitively, it was expected that the lower threshold would result in more change points thus resulting in more positives (whether true or false), thus increasing the sensitivity. However, the data show that this is not always the case (see table 6). Given that the learning is based on multiple variables, the threshold of any one input variable tunes the overall model. By decreasing the threshold of one of the variables, the model may "break" in the sense that it is unable to learn from the training set. This results in overall poorer performance, including a decrease in sensitivity and/or specificity, despite the intuition that the sensitivity would increase.

Increasing R-wave detection threshold	Specificity
0.6	0.710
0.65	0.710
0.7	0.702
0.75	0.710
0.8	0.710
0.85	0.710
0.9	0.723

Table 6: Example of the lack of correlation between feature extraction thresholds and overall sensitivity (0.822 for all thresholds) and specificity

9.4.5.5 *Overall Comparison*

The overall "best" performing model was a random forest wrapped with a cost-sensitive classifier. The confusion matrix of the test set is detailed in table 7. The sensitivity was 0.857 and specificity was 0.742 with a C statistic of 0.799. The cost of a false negative was 8.5 and the cost of a false positive was 2.5, relative to a cost of 1.0 for both true positives and true negatives. In this specific set of experiments, increasing the cost of a false negative did not increase the sensitivity but decreased the specificity.

	Predicted	
	Positive	Negative
Positive	42	7
Negative	23	66

Table 7: Overall best performing run

10 FUTURE WORK

This work has laid the foundation for much more research of physiological sensors and their role in healthcare. This research can be broken into two main categories, clinical and technological research.

10.1 CLINICAL WORK

This research focused only on cardiovascular sensor data (EKG's and arterial blood pressures) as the primary source of data. The dataset used also contains a significant amount of clinical data that could be used to augment the sensor data. However, one of the major problems with incorporating more and more data is the problem of context. Taking all available data and feeding it to any data mining process may cause two problems: 1) computationally overwhelming the data mining process, or 2) finding a "signal" within the data when there isn't any.

The former problem becomes less of an issue as we learn how to scale the technology through graphics processor units (GPU's) and other forms of parallel computing. The latter remains a problem and will only increase as the healthcare industry does a better job of capturing and storing increasing amounts of data.

However, by incorporating knowledge about the system (in this case, knowledge of the human physiology and the practice of medicine), researchers may be able to include only the relevant data. This, however, means that some data is being included and others discarded. The underlying model is then influenced by the contextual data and those interpreting it.

The key problem for informatics is to create a framework that assists researchers in systematically including (or excluding) contextual clinical data. It will be especially important to learn how to deal with potentially "circular" information. For example, if we are attempting to detect arrhythmias, do we include a prior diagnosis of that

arrhythmia? What if the diagnosis was incorrectly coded into the patient's record due to a previous false alarm?

The issue of context also affects open-ended data mining, another interesting area of future work. Since my computational pipeline can also be used for unsupervised learning or discovery, it would be very interesting to feed increasing amounts of data to the system for true data mining. The "discoveries" can be reconciled with existing knowledge in order to validate their viability. Viable "discoveries" can then be used to seed other clinical research projects whether through other machine and statistical learning approaches or through more conventional clinical research.

10.2 TECHNOLOGICAL WORK

10.2.1 *Genetic algorithms*

Many of the experiments conducted involved manually changing various parameters in a semi-educated fashion to try and improve the overall performance of the system. In some sense, this was effectively a "search" for the best combination of parameters and features - best as defined by the combination of sensitivity, specificity, and C statistic. Consequently, this is the perfect setup for the application of search algorithms such as genetic algorithms.

The fitness function would be defined as some combination of sensitivity, specificity, and C statistic, most likely dependent on the current state of the art. Then, the evolutionary nature of genetic algorithms can be used to create random offspring of varying parameters and combinations of features. This would largely automate the general approach that I took with my work. Instead of focusing on the manual tuning of parameters and/or feature selection, the researcher can then focus on the curation of results and seeing they "made sense."

10.2.2 *Real-time processing*

While the algorithms were constructed and coded as either online or offline algorithms, the underlying infrastructure is based on Hadoop. While it is an extremely powerful and flexible platform, Hadoop is a batch processing system. There are ways of utilizing Hadoop so that it behaves as a real-time or streaming system. The next step would be to integrate one of these approaches such as Hadoop Online[17] to extend the system to provide real-time, clinical decision support.

Another alternative to the Hadoop ecosystem is the Twitter Storm[46] framework. It is oftentimes referred to as a "real-time Hadoop" but was designed from the bottom up to be a real-time, stream processing system. This makes it well-suited to real-time decision support applications. Since it passes data between "bolts" in a mechanism similar to Hadoop, many of the algorithms implemented for Hadoop would take minimal changes to make them work with Storm.

In either case, the key is modifying the system to "stream" the data to the parallel processing framework and then connecting this information to a dashboard. The overarching clinical research question is if such a system helps improve the clinical workflow and patient outcomes.

BIBLIOGRAPHY

[1] IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008. doi: 10.1109/IEEESTD.2008.4610935.

[2] bit.ly: URL shortening service, 2013. URL http://www.bit.ly.

[3] Groovy: A dynamic language for the java platform, 2013. URL http://groovy.codehaus.org/.

[4] mongoDB: document database, 2013. URL http://www.mongodb.org.

[5] OpenTSDB: distributed, scalable, time series database, 2013. URL http://opentsdb.net/.

[6] StumbleUpon, 2013. URL http://www.stumbleupon.com.

[7] Anton Aboukhalil, Larry Nielsen, Mohammed Saeed, Roger G Mark, and Gari D Clifford. Reducing false alarm rates for critical arrhythmias using the arterial blood pressure waveform. *Journal of Biomedical Informatics*, 41(3):442–451, June 2008. ISSN 1532-0480. doi: 10.1016/j.jbi.2008.03.003. URL http://www.ncbi.nlm.nih.gov/pubmed/18440873. PMID: 18440873.

[8] Ryan Prescott Adams and David J. C. MacKay. Bayesian online changepoint detection. *arXiv:0710.3742*, October 2007. URL http://arxiv.org/abs/0710.3742.

[9] Joe Armstrong, Robert Virding, Claes Wikstrom, and Mike Williams. *Concurrent Programming in ERLANG*. Prentice Hall, 1996.

[10] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, page 1027–1035, Philadelphia, PA, USA, 2007. Society for Indus-

trial and Applied Mathematics. ISBN 978-0-898716-24-5. URL http://dl.acm.org.ucsf.idm.oclc.org/citation.cfm?id=1283383.1283494.

[11] Riccardo Bellazzi and Blaz Zupan. Predictive data mining in clinical medicine: Current issues and guidelines. *International Journal of Medical Informatics*, 77(2):81–97, February 2008. ISSN 1386-5056. doi: 10.1016/j.ijmedinf.2006.11.006. URL http://www.sciencedirect.com/science/article/pii/S1386505606002747.

[12] Olivier Bodenreider. The unified medical language system (UMLS): integrating biomedical terminology. *Nucleic Acids Research*, 32(suppl 1):D267–D270, January 2004. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkh061. URL http://nar.oxfordjournals.org/content/32/suppl_1/D267. PMID: 14681409.

[13] Marie-Christine Chambrin. Alarms in the intensive care unit: how can the number of false alarms be reduced? *Critical Care*, 5(4):1–5, August 2001. ISSN 1364-8535. doi: 10.1186/cc1021. URL http://link.springer.com/article/10.1186/cc1021.

[14] Xiujuan Chen, Yong Li, Robert Harrison, and Yan-Qing Zhang. Genetic fuzzy classification fusion of multiple SVMs for biomedical data. *Journal of Intelligent and Fuzzy Systems*, 18(6):527–541, January 2007. URL http://iospress.metapress.com/content/L3421K663QN136M1.

[15] M. E. Cohen, D. L. Hudson, and P. C. Deedwania. Applying continuous chaotic modeling to cardiac signal analysis. *IEEE Engineering in Medicine and Biology Magazine*, 15(5):97–102, 1996.

[16] Christopher R. Cole, Eugene H. Blackstone, Fredric J. Pashkow, Claire E. Snader, and Michael S. Lauer. Heart-rate recovery immediately after exercise as a predictor of mortality. *New England Journal of Medicine*, 341(18):1351–1357, 1999. ISSN 0028-4793. doi: 10.1056/NEJM199910283411804. URL http://www.nejm.org/doi/full/10.1056/NEJM199910283411804. PMID: 10536127.

[17] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. MapReduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, page 21–21, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855711.1855732.

[18] Mithilesh Kumar Das, Chandan Saha, Hicham El Masry, Jonathan Peng, Gopi Dandamudi, Jo Mahenthiran, Paul McHenry, and Douglas P. Zipes. Fragmented QRS on a 12-lead ECG: a predictor of mortality and cardiac events in patients with coronary artery disease. *Heart Rhythm*, 4(11):1385–1392, November 2007. ISSN 1547-5271. doi: 10.1016/j.hrthm.2007.06.024. URL http://www.sciencedirect.com/science/article/pii/S1547527107007989.

[19] Yadin David, J. Tobey Clark, J. Ott, T. Bauld, B. Patail, I. Gieras, M. Shepherd, S. Miodownik, J. Heyman, O. Keil, A. Lipschultz, B. Hyndman, W. Hyman, J. Keller, M. Baretich, W. Morse, and D. Dickey. Improving patient safety through clinical alarms management. In *11th Mediterranean Conference on Medical and Biomedical Engineering and Computing 2007*, pages 1051–1054. 2007. URL http://dx.doi.org/10.1007/978-3-540-73044-6_271.

[20] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. San Francisco, CA, December 2004. URL http://labs.google.com/papers/mapreduce.html.

[21] Catherine M. DesRoches, Dustin Charles, Michael F. Furukawa, Maulik S. Joshi, Peter Kralovec, Farzad Mostashari, Chantal Worzala, and Ashish K. Jha. Adoption of electronic health records grows rapidly, but fewer than half of US hospitals had at least a basic system in 2012. *Health Affairs*, 32(8):1478–1485, August 2013. ISSN 0278-2715, 1544-5208. doi: 10.1377/hlthaff.2013.0308. URL http://content.healthaffairs.org/content/32/8/1478. PMID: 23840052.

[22] Polychronis E. Dilaveris and John E. Gialafos. P-wave dispersion: A novel predictor of paroxysmal atrial fibrillation. *Annals of Noninvasive Electrocardiology*, 6(2):159–165, 2001. ISSN 1542-474X. doi: 10.1111/j.1542-474X.2001.tb00101.x. URL http://onlinelibrary.wiley.com.ucsf.idm.oclc.org/doi/10.1111/j.1542-474X.2001.tb00101.x/abstract.

[23] Ramón Díaz-Uriarte and Sara Alvarez de Andrés. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(1):3, January 2006. ISSN 1471-2105. doi: 10.1186/1471-2105-7-3. URL http://www.biomedcentral.com/1471-2105/7/3/abstract. PMID: 16398926.

[24] David R. Easterling and Thomas C. Peterson. A new method for detecting undocumented discontinuities in climatological time series. *International Journal of Climatology*, 15(4):369–377, 1995. ISSN 1097-0088. doi: 10.1002/joc.3370150403. URL http://onlinelibrary.wiley.com.ucsf.idm.oclc.org/doi/10.1002/joc.3370150403/abstract.

[25] I. El-Naqa, Yongyi Yang, M.N. Wernick, N.P. Galatsanos, and R.M. Nishikawa. A support vector machine approach for detection of microcalcifications. *IEEE Transactions on Medical Imaging*, 21(12):1552–1563, 2002. ISSN 0278-0062. doi: 10.1109/TMI.2002.806569.

[26] Meng Joo Er, Shiqian Wu, J. Lu, and Hock Lye Toh. Face recognition with radial basis function (RBF) neural networks. *IEEE Transactions on Neural Networks*, 13(3):697–710, 2002. ISSN 1045-9227. doi: 10.1109/TNN.2002.1000134.

[27] Emile Ferrari, Alain Imbert, Thierry Chevalier, Alain Mihoubi, Philippe Morand, and Marcel Baudouy. The ecg in pulmonary embolism : Predictive value of negative t waves in precordial leads—80 case reports. *CHEST Journal*, 111(3):537–543, March 1997. ISSN 0012-3692. doi: 10.1378/chest.111.3.537. URL http://dx.doi.org/10.1378/chest.111.3.537.

[28] The Apache Foundation. Apache mahout: Scalable machine learning and data mining, 2013. URL http://mahout.apache.org.

[29] Colin Gallagher, Robert Lund, and Michael Robbins. Changepoint detection in climate time series with long-term trends. *Journal of Climate*, 26(14):4994–5006, July 2013. ISSN 0894-8755, 1520-0442. doi: 10.1175/JCLI-D-12-00704.1. URL http://journals.ametsoc.org.ucsf.idm.oclc.org/doi/abs/10.1175/JCLI-D-12-00704.1.

[30] Lars George. *HBase: The Definitive Guide*. O'Reilly Media, Inc., September 2011. ISBN 9781449396107.

[31] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL http://doi.acm.org.ucsf.idm.oclc.org/10.1145/1656274.1656278.

[32] Rich Hickey. The clojure programming language. In *Proceedings of the 2008 symposium on Dynamic languages*, DLS '08, page 1:1–1:1, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-270-2. doi: 10.1145/1408681.1408682. URL http://doi.acm.org/10.1145/1408681.1408682.

[33] Paul Hudak, Simon Peyton Jones, Philip Wadler, Brian Boutel, Jon Fairbairn, Joseph Fasel, María M. Guzmán, Kevin Hammond, John Hughes, Thomas Johnsson, Dick Kieburtz, Rishiyur Nikhil, Will Partain, and John Peterson. Report on the programming language haskell: a non-strict, purely functional language version 1.2. *SIGPLAN Not.*, 27(5):1–164, May 1992. ISSN 0362-1340. doi: 10.1145/130697.130699. URL http://doi.acm.org/10.1145/130697.130699.

[34] Michael Imhoff and Silvia Kuhls. Alarm algorithms in critical care monitoring. *Anesthesia and Analgesia*, 102(5):1525–1537, May 2006. ISSN 1526-7598. doi: 10.1213/

01.ane.0000204385.01983.61. URL http://www.ncbi.nlm.nih.gov/pubmed/16632837. PMID: 16632837.

[35] Rui Jiang, Wanwan Tang, Xuebing Wu, and Wenhui Fu. A random forest approach to the detection of epistatic interactions in case-control studies. *BMC Bioinformatics*, 10(Suppl 1):S65, January 2009. ISSN 1471-2105. doi: 10.1186/1471-2105-10-S1-S65. URL http://www.biomedcentral.com/1471-2105/10/S1/S65/abstract. PMID: 19208169.

[36] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *ICDM 2001, Proceedings IEEE International Conference on Data Mining, 2001*, pages 289–296, 2001. doi: 10.1109/ICDM.2001.989531.

[37] Tze Leung Lai. Sequential changepoint detection in quality control and dynamical systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(4):613–658, January 1995. ISSN 0035-9246. doi: 10.2307/2345934. URL http://www.jstor.org/stable/2345934. ArticleType: research-article / Full publication date: 1995 / Copyright © 1995 Royal Statistical Society.

[38] M. Lavielle and G. Teyssière. Detection of multiple change-points in multivariate time series. *Lithuanian Mathematical Journal*, 46(3):287–306, July 2006. ISSN 0363-1672, 1573-8825. doi: 10.1007/s10986-006-0028-9. URL http://link.springer.com.ucsf.idm.oclc.org/article/10.1007/s10986-006-0028-9.

[39] S T Lawless. Crying wolf: false alarms in a pediatric intensive care unit. *Critical care medicine*, 22(6):981–985, June 1994. ISSN 0090-3493. PMID: 8205831.

[40] S. Lawrence, C.L. Giles, Ah Chung Tsoi, and A.D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997. ISSN 1045-9227. doi: 10.1109/72.554195.

[41] Ki-Joong Lee, Young-Sook Hwang, Seonho Kim, and Hae-Chang Rim. Biomedical named entity recognition using two-phase model based on SVMs. *Journal of Biomedical Informatics*, 37(6):436–447, December 2004. ISSN 1532-0464. doi: 10.1016/j.jbi.2004.08.012. URL http://www.sciencedirect.com/science/article/pii/S1532046404000863.

[42] Erica L Liebelt, Paul D Francis, and Alan D Woolf. ECG lead aVR versus QRS interval in predicting seizures and arrhythmias in acute tricyclic antidepressant toxicity. *Annals of Emergency Medicine*, 26(2):195–201, August 1995. ISSN 0196-0644. doi: 10.1016/S0196-0644(95)70151-6. URL http://www.sciencedirect.com/science/article/pii/S0196064495701516.

[43] Richard P. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1(1):1–38, March 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.1.1. URL http://dx.doi.org/10.1162/neco.1989.1.1.1.

[44] Song Liu, Makoto Yamada, Nigel Collier, and Masashi Sugiyama. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, 43:72–83, July 2013. ISSN 0893-6080. doi: 10.1016/j.neunet.2013.01.012. URL http://www.sciencedirect.com/science/article/pii/S0893608013000270.

[45] Robert Lund, Xiaolan L. Wang, Qi Qi Lu, Jaxk Reeves, Colin Gallagher, and Yang Feng. Changepoint detection in periodic and autocorrelated time series. *Journal of Climate*, 20(20):5178–5190, October 2007. ISSN 0894-8755, 1520-0442. doi: 10.1175/JCLI4291.1. URL http://journals.ametsoc.org.ucsf.idm.oclc.org/doi/abs/10.1175/JCLI4291.1.

[46] Nathan Marz. Storm: Distributed and fault-tolerant realtime computation, 2013. URL http://www.storm-project.net.

[47] Deborah L. McGuinness and Frank Van Harmelen. OWL web ontology language overview. *W3C recommendation*, 10(2004-03):10, 2004. URL http://cies.hhu.edu.cn/pweb/~zhuoming/teachings/MOD/N4/Readings/5.3-B1.pdf.

[48] Tomohiro Mitsumori, Masaki Murata, Yasushi Fukuda, Kouichi Doi, and Hirohumi Doi. Extracting protein-protein interaction information from biomedical text with SVM. *IEICE TRANSACTIONS on Information and Systems*, E89-D(8):2464–2466, August 2006. ISSN 1745-1361, 0916-8532. URL http://search.ieice.org/bin/summary.php?id=e89-d_8_2464&category=D&year=2006&lang=E&abst=.

[49] Valentina Moskvina and Anatoly Zhigljavsky. An algorithm based on singular spectrum analysis for change-point detection. *Communications in Statistics - Simulation and Computation*, 32(2):319–352, 2003. ISSN 0361-0918. doi: 10.1081/SAC-120017494. URL http://www.tandfonline.com/doi/abs/10.1081/SAC-120017494.

[50] Timo H Mäkikallio, Søren Høiber, Lars Køber, Christian Torp-Pedersen, Chung-Kang Peng, Ary L Goldberger, and Heikki V Huikuri. Fractal analysis of heart rate dynamics as a predictor of mortality in patients with depressed left ventricular function after acute myocardial infarction. *The American Journal of Cardiology*, 83(6): 836–839, March 1999. ISSN 0002-9149. doi: 10.1016/S0002-9149(98)01076-5. URL http://www.sciencedirect.com/science/article/pii/S0002914998010765.

[51] Jiapu Pan and Willis J. Tompkins. A real-time QRS detection algorithm. *IEEE Transactions on Biomedical Engineering*, BME-32(3):230–236, 1985. ISSN 0018-9294. doi: 10.1109/TBME.1985.325532.

[52] Bonnie K. Ray and Ruey S. Tsay. Bayesian methods for change-point detection in long-range dependent processes. *Journal of Time Series Analysis*, 23(6):687–705, 2002. ISSN 1467-9892. doi: 10.1111/1467-9892.00286. URL http://onlinelibrary.wiley.com.ucsf.idm.oclc.org/doi/10.1111/1467-9892.00286/abstract.

[53] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998. ISSN 0162-8828. doi: 10.1109/34.655647.

[54] M. Saeed, C. Lieu, Raber, and R. G Mark. MIMIC II: a massive temporal ICU patient database to support research in intelligent patient monitoring. *Computers in Cardiology*, 29:641–644, September 2002. URL http://www.cinc.org/Proceedings/2002/pdf/641.pdf.

[55] O. Seidou and T. B. M. J. Ouarda. Recursion-based multiple changepoint detection in multiple linear regression and application to river streamflows. *Water Resources Research*, 43(7):n/a–n/a, 2007. ISSN 1944-7973. doi: 10.1029/2006WR005021. URL http://onlinelibrary.wiley.com.ucsf.idm.oclc.org/doi/10.1029/2006WR005021/abstract.

[56] Marc Snir, Steve W. Otto, David W. Walker, Jack Dongarra, and Steven Huss-Lederman. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA, 1995. ISBN 0262691841.

[57] Alexander Statnikov, Lily Wang, and Constantin F. Aliferis. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. *BMC Bioinformatics*, 9(1):319, July 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-319. URL http://www.biomedcentral.com/1471-2105/9/319/abstract. PMID: 18647401.

[58] J. S. Steinberg, S. Zelenkofske, S. C. Wong, M. Gelernt, R. Sciacca, and E. Menchavez. Value of the p-wave signal-averaged ECG for predicting atrial fibrillation after cardiac surgery. *Circulation*, 88(6):2618–2622, December 1993. ISSN 0009-7322, 1524-4539. doi: 10.1161/01.CIR.88.6.2618. URL http://circ.ahajournals.org/content/88/6/2618. PMID: 8252672.

[59] Hans-Ulrich Strohmenger, Karl H. Lindner, and Charles G. Brown. Analysis of the ventricular fibrillation ecg signal amplitude and frequency parameters as predictors of countershock success in humans. *CHEST Journal*, 111(3):584–589, March 1997. ISSN 0012-3692. doi: 10.1378/chest.111.3.584. URL http://dx.doi.org/10.1378/chest.111.3.584.

[60] J. Takeuchi and K. Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE Transactions on Knowledge and Data Engineering*, 18(4):482–492, 2006. ISSN 1041-4347. doi: 10.1109/TKDE.2006.1599387.

[61] R Core Team. R: A language and environment for statistical computing, 2013. URL http://www.R-project.org.

[62] Alex Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1(1):39–46, March 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.1.39. URL http://dx.doi.org/10.1162/neco.1989.1.1.39.

[63] P. H. Winston and B. K. Horn. LISP. second edition. January 1986. URL http://www.osti.gov/scitech/biblio/7203980.

[64] Rob Wynden, Mark G. Weiner, Ida Sim, Davera Gabriel, Marco Casale, Simona Carini, Shannon Hastings, David Ervin, Samson Tu, John H. Gennari, Nick Anderson, Ketty Mobed, Prakash Lakshminarayanan, Maggie Massary, and Russ J. Cucina. Ontology mapping and data discovery for the translational investigator. *AMIA Summits on Translational Science Proceedings*, 2010:66–70, March 2010. ISSN 2153-4063. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3041530/. PMID: 21347152 PMCID: PMC3041530.

PUBLISHING AGREEMENT

It is the policy of the University to encourage the distribution of all theses, dissertations, and manuscripts. Copies of all UCSF theses, dissertations, and manuscripts will be routed to the library via the Graduate Division. The library will make all theses, dissertations, and manuscripts accessible to the public and will preserve these to the best of their abilities, in perpetuity.

I hereby grant permission to the Graduate Division of the University of California, San Francisco to release copies of my thesis, dissertation, or manuscript to the Campus Library to provide access and preservation, in whole or in part, in perpetuity.

2013-09-10

Andrew V. Nguyen