Healthcare Data Mining Using In-database Analytics to Predict Diagnosis of Inflammatory Bowel Disease

Eric Johnson

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2012

Reading Committee:

Ankur Teredesai, Chair

Senjuti Basu Roy

Program Authorized to Offer Degree:
Computer Science and Systems, Institute of Technology

University of Washington

Abstract

Healthcare Data Mining Using In-database Analytics to
Predict Diagnosis of Inflammatory Bowel Disease

Eric Johnson

Chair of the Supervisory Committee:
Associate Professor Ankur Teredesai
Computer Science and Systems, Institute of Technology

Inflammatory Bowel Disease is a life-changing affliction with few correlative and no known causal factors for its two major forms. With the widespread use of electronic medical record systems, and therefore the availability of large, highly dimensional, and semi-structured data, the importance of finding effective and scalable data mining algorithms to handle such data has increased dramatically. With these algorithms, one can develop useful predictive and analytical tools for providers and researchers to ultimately improve the quality of patient lives.

In recent years, there has been a growing interest in the classical application of Incremental Gradient Descent techniques to convex programming problems because of their rapid convergence and tolerance to noise. And with the recent development of in-database analytics frameworks leveraging Incremental Gradient Descent algorithms and other user-defined aggregates as in the Bismarck architecture, rapid analysis of large and highly-dimensional data is facilitated.

In this thesis, we describe the first ever application of the Bismarck in-database analytics framework in a healthcare setting. We applied logistic regression using Bismarck on a four-year set of patient demographic, encounter, and hospital account data and produced predictive risk factors for a cohort of Inflammatory Bowel Disease patients. We also developed a simple, automated model builder framework that supports other cohorts of interest,

and discuss its design. We also outline our future steps to extend the algorithms to include spatial data analysis and to provide data visualization tools that assist providers and researchers in gaining insight into the correlative factors behind the disease.

The challenges of the clinical data set - large, highly dimensional, heterogenous, with statistically significant amounts of noise - highlight the advantages of the key-value structures the Bismarck architecture leverages. The predictive models produced were better than random and built on commodity hardware running an open-source, distributable, database engine. Since the Bismarck in-database analytics framework is scalable and parallelizable and facilitates straightforward extension and modification, the success of our application has shown the viability of producing predictive models for other cohorts of interest in a similar healthcare setting.

TABLE OF CONTENTS

Page

LIST OF FIGURES

GLOSSARY

DIAGNOSIS CODE: A code used to group and identify diseases, disorders, symptoms, human response patterns, and medical signs.

EMR: Electronic medical record. A computerized medical record created in an organization that delivers care.

ERD: Entity relationship diagram. A specialized graphic that illustrates the relationships between entities in a database.

FN: False negative. The number of negative labelings that are incorrect.

FP: False positive. The number of positive labelings that are incorrect.

GRADIENT DESCENT: Incremental Gradient Descent. A gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions.

HOSPITAL ACCOUNT RECORD: The medical record for a patient generated during a period of hospitalization, usually including accounts of consultants' opinions as well as nurses' observations and treatments.

IBD: Inflammatory Bowel Disease. A group of inflammatory conditions of the colon and small intestine.

IN-DATABASE ANALYTICS: A scheme for processing and applying analytics on data within the database, avoiding the movement of data that slows response time.

INPATIENT: A patient who stays at a hospital while under treatment.

OUTPATIENT: A patient who receives medical treatment without being admitted to a hospital.

PATIENT ENCOUNTER RECORD: The medical record for a patient generated during an instance of direct provider/practitioner to patient interaction, regardless of the setting, for diagnosing, evaluating or treating the patients condition.

ROC: Receiver Operating Characteristic. A standard method from signal detection theory for measuring how well a binary classifier performs.

TN: True negative. The number of negative labelings that are correct.

TP: True positive. The number of positive labelings that are correct.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to University of Washington, Tacoma, where he has had the opportunity to pursue his interests in Computer Science and to work with fellow aspiring data scientists, researchers, and professors in the field.

He would also like to extend his gratitude to MultiCare Health System for supporting him in the pursuit of his degree, and for the invaluable access and use of their enterprise data warehouse.

Furthermore, the author would like to thank the Hazy project team at the University of Wisconsin, Madison for answering questions as they arose, and to thank the open-source community – particularly the PostgreSQL, Gentoo, and emacs communities – for their continued production of quality software and eagerness and ability to help others.

Last, but not least, the author is indebted to Associate Professor Ankur Teredesai for his guidance and encouragement throughout his program.

DEDICATION

To Shanon, and to my children, Autumn and David.

Chapter 1

INTRODUCTION

Inflammatory Bowel Disease is an idiopathic, life-changing disease affecting as many as 1.4 million people in the United States alone [1]. Inflammatory Bowel Disease (IBD) encompasses at least two forms of intestinal inflammation: ulcerative colitis and Crohn's disease. The causes of the major forms of Inflammatory Bowel Disease are unknown. In the absence of identifiable causes, Crohn's disease and ulcerative colitis are defined empirically by their typical clinical, pathologic, radiologic, endoscopic, and laboratory features [1, 15]. There are no definitive preventative measures and few correlative factors. However, dietary changes may help symptoms of ulcerative colitis and reduce inflammation in Crohn's disease, smoking has been linked with the number and severity of flare-ups, and infectious agents have been linked with some occurrences [1, 15, 7].

With the increased use of electronic medical record (EMR) systems, the data available to healthcare systems, their front-line clinical staff, and researchers has grown exponentially. Healthcare data is also highly dimensional: a single patient, over a span of just a few years, may have many hundreds of diagnosis codes on their medical record. The attribute space for diagnosis codes alone of a cohort of patients may number in the tens of thousands, and if vitals, lab results, and demographic information is also considered, this attribute space increases a hundred-fold. As well, since the documentation systems and data entry workflows often cause some statistically significant noise, either a large investment in data scrubbing and data validation resources is needed up-front in order to perform analysis, or the use of algorithms capable of handling noisy data is required. Furthermore, since healthcare systems, for business and other historical reasons, have often captured data in heterogenous systems or in semi-structured formats, and this data is captured at different levels of grain, robust and flexible algorithms are necessary for rapid and effective analysis and development of predictive models.

Convex programming problems provide the underpinnings of many data analytical techniques, such as logistic regression, support vector machine, and graphical models [9, 12]. Since Incremental Gradient Descent algorithms as applied to convex programming problems perform well, converge rapidly, and tolerate noise, they have seen a rise in their use recently, especially as applied to big data. In this paper we refer to Incremental Gradient Descent as Gradient Descent and Inflammatory Bowel Disease as IBD for ease of wording and legibility of the readers in the medical informatics community. The Bismarck architecture was inspired by the observation that the classical algorithm of Gradient Descent has a data-access pattern that is essentially identical to the data access pattern of any SQL aggregate function [11]. Therefore Bismarck can be used to leverage the features available in relational database management systems to efficiently perform and effectively scale analytics on large, highly-dimensional data sets. Furthermore, based on recent results from theory, since there is reason to believe that almost all combinatorial optimization problems have optimal approximations given by solving convex problems [17], the Bismarck analytical framework can facilitate the implementation of a wide range of data processing algorithms in a database.

We describe the first ever application of the Bismarck in-database analytics framework in a healthcare setting. Our target was the development of a predictive model of the likelihood of a patient being diagnosed with IBD given their demographic information and historical diagnoses. We modeled four years of patient demographic, encounter, and hospital account data in a distributed key-value store using PostgreSQL, sourced from MultiCare Health System's enterprise data warehouse and EMR system, Epic. MultiCare Health System is a health system in the south Puget Sound region near Tacoma, USA, having an ambulatory EMR extending back to 1998 and a fully integrated hospital and ambulatory EMR since 2008.

The main contributions of our work are: a predictive model for the likelihood of a diagnosis of IBD, a model building framework to facilitate flexible and rapid investigation of generated models, and insight into the viability of applying Bismarck in a healthcare setting.

In section 3.1, we describe the EMR data set, its complexity, and the challenges inherent in the data itself. We then describe the Bismarck in-database analytics architecture in

section 2.2 and its applicability to the problem space. In sections 4.1 and 4.2, we describe the data model, construction of the key-value stores, and the application of sparse logistic regression in building the predictive models. We also discuss a simple framework we designed to facilitate the generation and analysis of predictive models providing flexibility in exploring the impact of the features of a patient's medical record. Finally, in 5.1, we demonstrate our initial results that showcase the scalability of our approach while ensuring that the data mining model is able to predict IBD better than random which, in several cases, is better than the physician.

Chapter 2

BACKGROUND AND RELATED RESEARCH

Many problems from the field of data mining can be framed in terms of optimizing an objective function over a space of parameters defining a model function. Incremental, or stochastic, Gradient Descent is a method by which the optimal parameters for the model function are found incrementally via the gradient flow of the objective function. The Bismarck in-database architecture takes advantage of the fact that Gradient Descent's data access pattern is essentially identical to that of any SQL aggregate function [11].

2.1 Gradient Descent

Building a predictive model involves producing a function, or model, on the space of attributes with the desire that evaluating the model on instances of data will label the data as accurately as possible. For some data mining algorithms, an objective function is chosen that measures the distance a model is from the known data, or "fits" the data. The model is then parametrized, and minimized over these parameters relative to the objective function. More precisely, given an objective function $\phi : P \times (A \times L) \to R^+ \cup \{0\}$, where P is the space of parameters, w, for the model $m(w) : A \to L$, A is the space of attributes, and L is the space of labels, the data mining algorithms desire to minimize the total objective function $\Phi(p) = \sum_i \phi(p, a_i, l_i) = \sum_i \phi_i(p)$ over all known labeled instances (a_i, l_i). These algorithms assume that if a minimum at p over P can be found for all the labeled instances, that the resulting model $m(p)$ will "fit" or minimize the objective function over all unknown instances, as well.

By following the gradient vector field of the objective function from an arbitrary point in the space of parameters, an optimal value for the objective function, if one exists, can be found. If the function is convex, the optima is global. Incremental, or stochastic, gradient descent is an approximative technique for finding optima based on traversing this gradient

flow in discrete steps. Since it uses discrete step sizes, obviously it becomes important to iterate over various step size values. Since the purpose of the objective function is to help formulate a model which minimizes its value over the entire set of instances, calculating the value of the total objective function's gradient, $\nabla\Phi$, at each iteration when following the gradient flow would be expensive. As it turns out, it is unnecessary, too [8]. The total gradient can be approximated iteratively with the gradients, $\nabla\phi_i$, at each of the labeled instances from the training set, as in Equation 2.1.

$$w := w - c\nabla\phi_i(w) \tag{2.1}$$

Therefore, Gradient Descent uses the labeled training data instances to "condition" the parameter vector using the gradients, $\nabla\phi_i$ at each encountered instance, in an iterative and approximative manner. The value, c, represents the size of the discrete steps, is chosen by the data scientist, and often decays over iterations. Under mild assumptions, it can be guaranteed to converge to a local minimum [8], and in the case of convex programming problems, a global minimum. Furthermore, the convergence is rapid and, computationally, the method performs well and is tolerant of noise. However, the rate of convergence is affected by the order in which the training instances are iterated over. Therefore, implementations of Gradient Descent typically randomly shuffle training instances and multiple passes over the training set are made.

Since convex programming, a generalization of linear and semi-definite programming, provides the underpinnings of many data analytical techniques such as logistic regression, support vector machines, least squares, conditional random fields, and graph models, Gradient Descent is an effective method of approximating the optimal model in these cases, both quickly and accurately. Since the dimensionality of the attribute space was high, but per-patient information sparse, and we desired fast convergence and robustness with respect to noise, we chose to apply sparse logistic regression to build our models.

2.2 The Bismarck Architecture

In order to analyze large, highly-dimensional data found in healthcare, flexible, scalable, and fast-converging algorithms tolerant of noise are required. Since many common data analytics

tasks can be framed as convex programming problems, such as logistic regression, support vector machines, and graphical models, we choose to explore the use of sparse logistic regression in the Bismarck in-database analytics framework. Its architecture is inspired by the observation that Gradient Descent's data-access pattern is essentially identical that of any SQL aggregate function. The advantages yielded by this observation cannot be overstated. The Bismarck in-database analytics framework was developed by a team at the

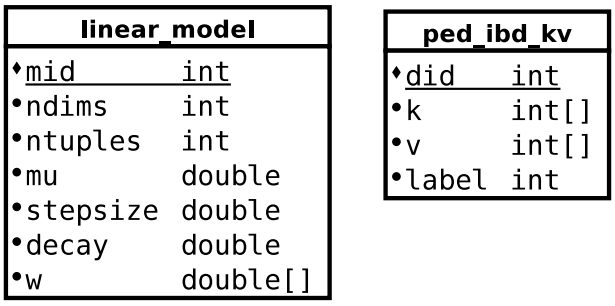

Figure 2.1: ERD for Bismarck linear model table and key-value data that will be processed

University of Wisconsin, and is available in both PostgreSQL and Greenplum [16]. It allows for the easy expansion of new analytical techniques within an in-database framework, as long as the developer creates three standard functions of a User-Defined Aggregate: Initialize, Transition, Terminate. Differences in analytical techniques mainly lie in the Transition function so few lines of C code are required to develop or expand new or existing analytical techniques. For example, logistic regression comprises 20 lines of C code and low-rank matrix factorization comprises fewer than 60 lines of C code [16]. The Bismarck procedures process data stored in tables with a particular signature and store the resulting models in a table, as in Figure 2.1.

2.3 Related Works

There are other in-database analytics frameworks that implement collections of data mining algorithms. MADlib is an open-source library and can be installed and executed within any relational database engine that supports extensible SQL. MADlib 0.4 was released in June

2012 and provides a relatively large number of widely-used statistical and analytical methods [3]. MADlib was designed so that the methods adhere to a shared-nothing requirement so that the libraries can take advantage of modern parallel database engines. Furthermore, the core functionality is written in declarative SQL statements, which facilitate data movement. Inner loops leverage SQL extensibility to make calls to math libraries in user-defined scalar and aggregate functions [13]. In [13], Hellerstein, Ré, et. al. describe in some detail the offerings of the MADlib library, as well as the inclusion of the ideas from the convex optimization abstraction central to the Bismarck architecture.

Chapter 3

PRESENTATION OF THESIS TOPIC

An in-database data analysis framework can leverage features commonly found in most relational database management systems, such as parallel processing and load distribution. Furthermore, optimizations of the query engine immediately benefit the performance of Bismarck and analytics remain "native" with the data. And with the unified architecture, development overhead is lowered allowing the data scientist to unify their algorithmic and theoretical studies, rather than studying properties of each new model.

Our topic is to assess the viability of the Bismarck in-database analytics framework in a healthcare environment. The question is whether the framework can produce better than random predictive models for IBD given a large, highly-dimensional set of data.

3.1 Complexity of the Data Set

The working data set spans the date range January 2008 through June 2012 and includes patient encounter and hospital account records together with the associated diagnosis information. It also includes patient demographic information. The set is highly dimensional, yet sparse. At the patient encounter level, there are over 22,000 possible binary attributes for diagnoses and demographic information, yet, on average, during a typical encounter a patient is assigned only a handful of diagnoses, and given the nature of healthcare data collection, discretized patient demographics elicit a few hundred attributes at most. If each patient has an average of about 13 encounters, taking the encounter diagnoses for the latest, say, ten encounters and considering each encounter as providing a distinct set of attributes, the attribute space in which these binary attributes reside quickly approaches dimensionality of a quarter million.

As a further complication, the data has some level of "noise" in that data entry workflows often cause statistically significant levels of missing or inaccurate data. Over time, electronic

medical record systems will continue to mature and healthcare systems and their workers will become more adept at capturing the patient experience and their visit attributes. Also, due to the nature of healthcare documentation standards and the human condition, patients report and exhibit signs and symptoms that may not map cleanly to the discrete codes then made available for data analysis. Therefore, data scientists must approach these problems with tools that are robust and relatively insensitive to such constraints and complications.

As well, for business and other historical reasons, healthcare systems capture data at different levels of granularity. For example, when a patient visits the hospital, billing and coding procedures and diagnoses are captured in what is known as the hospital account. When a patient visits an outpatient clinic, billing and coding procedures and diagnoses are captured at what is know as the patient encounter. Additionally, if professional services are given by specialists, data is captured at yet another, different level of granularity.

In Figure 3.1, we exhibit basic demographic and attribute statistics. In Figure 3.2,

Type	N	Distinct encounter diagnoses	Distinct account diagnoses
All patients	908K	21.8K	10.8K
IBD patients	5.5K	2.8K	4.5K

Figure 3.1: Total unique patient counts and unique count of diagnoses given

we exhibit basic patient encounter and hospital account frequencies per patient. Of note, for IBD patients the mean number of hospital accounts was roughly 2.5 times that of all patients, and the mean number of patient encounters was roughly 2.9 times that of all patients. In Figure 3.3, we exhibit diagnosis frequencies per patient encounter or hospital account. In this paper, we perform analysis at a patient encounter and hospital account level. We consider the diagnoses as binary attributes out of the entire space of diagnosis codes, and we discretize the patient demographics, appending these onto the encounter or account diagnoses attribute vector.

Type	N	μ	σ
All encounters	10.6M	13.1/pat	23.5
IBD encounters	179K	38.0/pat	47.5
All accounts	3.98M	5.72/pat	9.27
IBD accounts	67.3K	14.4/pat	17.3

Figure 3.2: Patient encounter and hospital account frequency by patient

Type	N	μ	σ
All encounters	21.8M	2.05/enc	1.75
IBD encounters	394K	2.19/enc	2.09
All accounts	9.65M	2.42/acct	2.48
IBD accounts	203K	3.01/acct	3.23

Figure 3.3: Diagnosis frequency by patient encounter or hospital account

Chapter 4

DESIGN OF EXPERIMENT

The application of the Bismarck analytical framework to the problem of creating predictive models for risk of diagnosis of IBD fell into three main buckets of work. First, we designed the supporting data model and sourced the EMR data, selected the combinations of attribute classes with which to build the models, and populated the requisite key-value stores.

Second, for the data sets of interest, we labeled each record with an IBD indicator, split the set into a training set and a test set, and blinded the label in the test records. We then trained the sparse logistic regression model on the train set, and applied the model weight vector on the test set, recording the resulting factor for each record. We simplified this procedure with a script to allow for parameter modification and production of multiple models for later analysis and comparison.

We then applied an ROC analysis on the results. This was also facilitated by automation procedures. We noted some interesting results with regard to the strength of models based on some attribute classes. For example, generated models on the set comprised of the hospital account diagnoses showed greater accuracy when predicting patients who would be given a primary diagnosis of IBD as opposed to a secondary diagnosis.

4.1 Data Modeling and Sourcing of the Data

From the EMR, we sourced into the data warehouse the patient demographic data and their geocoded address histories, a diagnosis code lookup with a diagnosis code grouping structure, the patient encounters and hospital accounts with their associated diagnoses, and a cohort structure used to uniformly identify and quickly build cohorts of interest. The data model is captured in the entity relationship diagram (ERD) in Figure 4.1. We populated the diagnosis group for IBD together with the defining diagnoses, as in Figure 4.2. We also

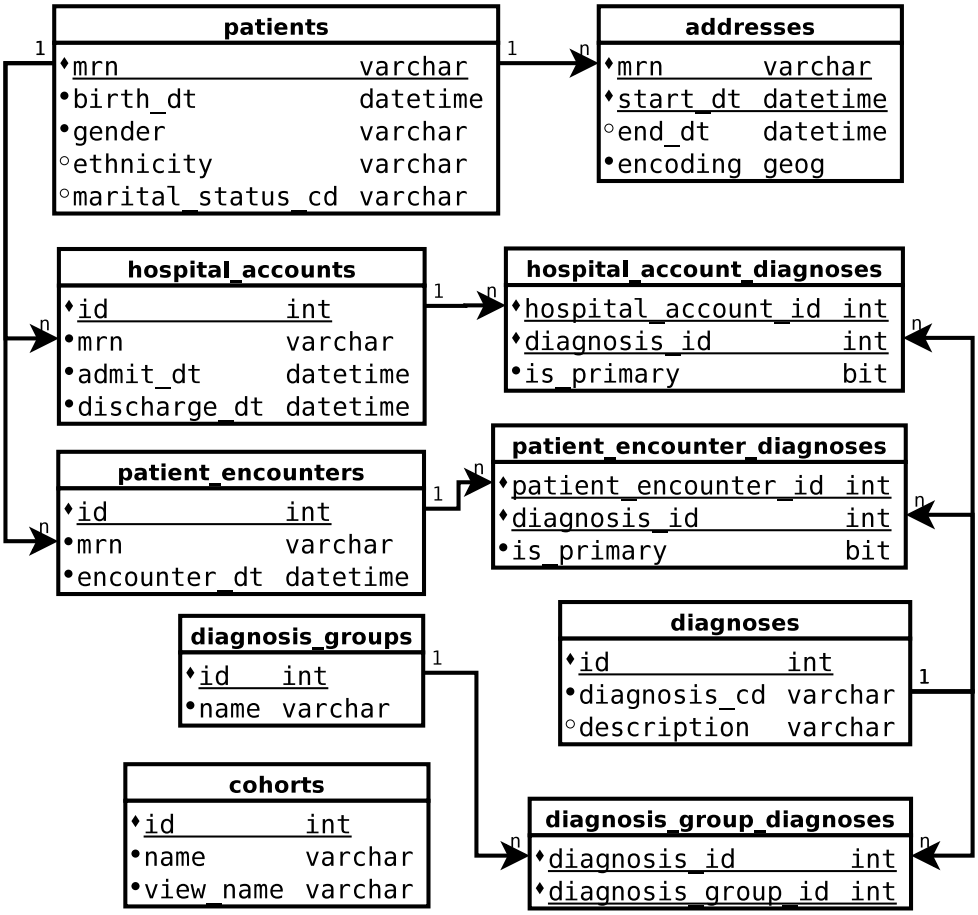

Figure 4.1: ERD for the sourced data from the EMR as well as supporting cohort builder tables

designed a target data model for the cohort attributes as a key-value store in a distributed PostgreSQL environment that adhered to the required signature the Bismarck architecture expects for processing, as depicted in Figure 2.1.

We also defined the classes of attributes of a patient's medical record that were of interest and selected several different combinations of these attribute classes to support the next phase of applying the Bismarck analytics. From the sourced data in the warehouse, we populated the key-value structures expected by the Bismarck architecture. We chose to analyze patient encounter or hospital account diagnoses together with patient demographic

```
insert into diagnosis_group (name) values ('ibd');

insert into diagnosis_group_diagnoses (diagnosis_group_id, diagnosis_id)
select dg.id, d.id
from diagnosis_group dg, diagnoses d
where dg.name = 'ibd'
and d.diagnosis_cd in ('555.0', '555.1', '555.2', '555.9',
  '556.0', '556.1', '556.2', '556.3', '556.4',
  '556.5', '556.6', '556.7', '556.8', '556.9');
```

Figure 4.2: Population of the diagnosis group and set of diagnoses

attributes. Since the number of diagnoses for each encounter or account are variable, the space of possible diagnoses relatively large, and the number of available demographic attributes inconsistent, utilizing a flexible data structure such as a key-value store handled the highly dimensional and heterogenous data found in healthcare quite well. We were able to use the same processing scripts to investigate various sets we were interested in without having to build additional data structures and retool the code.

Each record in the tables of key-value structures is comprised of an identifier such as a patient encounter ID, hospital account ID, or patient MRN depending on the level of granularity of the analysis, an array of keys representing the attribute IDs of the record, an array of values representing the weights of the attributes, all of which we assigned a value of 1 indicating the presence of the attribute or diagnosis, and a classifier label. Figure 4.3 provides an example query that populates a key-value structures with unlabeled data, obviously removing the IBD-specific diagnosis codes from the attributes, utilizing the `array_agg` function to facilitate building the required key-value stores [6]. We then captured labeling with a subsequent update statement to the label column. We initially selected four different sets of attributes with which to perform our analysis, as shown in Figure 4.4. Following the first runs, it was clear there was a notable difference in model accuracy when analyzing the population who were subsequently coded with a primary diagnosis of IBD

```
insert into pe_ibd_dx_kv (did, k, v, label)
select pe.id, array_agg(d.id), array_agg(1), -1
from patient_encounter pe
join patient_encounter_diagnoses ped
on pe.id = ped.patient_encounter_id
join diagnoses d
on d.diagnosis_cd = ped.diagnosis_cd
left join (
  select distinct dgd.diagnosis_cd
  from diagnosis_group_diagnoses dgd
  join diagnosis_groups dg
  on dgd.diagnosis_group_id = dg.id
  and dg.name = 'ibd') g
on g.diagnosis_cd = ped.diagnosis_cd
where g.diagnosis_cd is null
group by pe.id;
```

Figure 4.3: Sample code to populate key-value structures of interest

Granularity	Attributes
Patient Encounter	Diagnoses
Patient Encounter	Diagnoses, age at encounter, gender, zip code
Hospital Account	Diagnoses
Hospital Account	Diagnoses, age at admit, gender, zip code

Figure 4.4: Level of granularity and attributes for initial sets of interest

compared with those who were subsequently coded with a secondary diagnosis of IBD. We then labeled, trained, and tested each of the four sets in two different ways - first, to predict

any diagnosis of IBD, and then to predict a primary diagnosis of IBD.

4.2 Application of Sparse Logistic Regression and Model Testing

We partitioned each key-value table into a training set and a test set, "blinding" the labels in the test set. To do the partitioning, we issued simple SQL queries, as found in Figure 4.5, where we simply utilized the ID of the data sets, as these IDs are system generated, giving us a 70/30 split of each key-value table into train and test sets. We then built various predictive

```
select *
into ped_ibd_kv_train
from ped_ibd_kv
where did % 10 < 7;
```

Figure 4.5: Sample code to split off a train set. The code for test set is nearly the same.

models using sparse logistic regression for the four sets of interest with the two different labelings. In Figure 4.6, we exhibit a query that builds the weight vector, w, for the linear model. This weight vector, along with the user-specified model id and the parameters used in its generation in the `linear_model` table. The value 27 in this example merely represents a user-specified ID for the model, each model produced receiving a unique model id, identifying it within the `linear_model` table. Without specifying values for mu, step size, and decay, the call to `sparse_logit` uses the default parameters A, B, C, respectively, and so, for example, the call in Figure 4.6 would store these values in the `linear_model` table. In our analysis, we used a range of values for the step size and decay as changes in these affect convergence of the underlying Gradient Descent.

To apply a model on the unlabeled 30% partition involves issuing simple queries. In Figure 4.7, for example, the initialization of `model_id` 27 is performed, which allocates shared memory for subsequent prediction calls, then predictions are made using the scalar function `sparse_logit_pred` on the key-value pairs for the test instance, and then the shared memory is released.

```
do $$
declare r record;
begin
for r in select max(id) m
    from (select unnest(k) id from ped_ibd_kv) q;
loop
  select sparse_logit('ped_ibd_kv_train', 27, r.m);
end loop;
end $$;
```

Figure 4.6: Sample code to generate the linear model's weight vector used for prediction

```
select sparse_logit_init(27);
select did, sparse_logit_pred(27, k, v) pred
into ped_ibd_kv_test
from ped_ibd_kv
where did % 10 >= 7;
select sparse_logit_clear(27);
```

Figure 4.7: Sample code to apply sparse logistic regression linear model for a specified trained model on test set

Chapter 5

EXPERIMENT RESULTS AND DISCUSSION

With the labeling of the test data complete, we were then able to compare these labelings with the actual classification in the medical record. A standard method for measuring the strength of a binary classifier is to perform a Receiver Operating Characteristic (ROC) analysis, as in signal detection theory. Four counts are calculated: true positives (TP) - positive labelings that are correct, false positives (FP) - positive labelings that are incorrect, true negatives (TN) - negative labelings that are correct, and false negatives (FN) - negative labelings that are incorrect. ROC curves are a graphical representation capturing the true positive rate $\frac{TP}{TP+FN}$ versus the false positive rate $\frac{FP}{FP+TN}$ and the area under the curve is the accuracy of the predictive model.

5.1 Analysis of the Results

An ROC analysis was applied to the subsequent test data sets and the evaluation of the test data stored for later reporting, comparison, and ranking. Figure 5.1 shows the signature of the function that generates ROC data for later analysis of accuracy (see Appendix figure A.1 for full code). With our desire to remain within the database engine, we investigated the use

```
create function generate_roc_data (
  model_id int,
  labeled_table_name varchar(255),
  test_table_name varchar(255),
  start_threshold double precision,
  end_threshold double precision,
  step_size double precision)
```

Figure 5.1: Signature of pl/pgsql ROC data generator to support model evaluation

of plr, the R language extension for PostgreSQL to generate the ROC data and visualizations [4]. However, at the time of design, we felt it was cleaner and created less dependencies to utilize the built-in pl/pgsql language. A sample call to generate ROC data with the function is shown in Figure 5.2.

```
select generate_roc_data(
  27, 'ped_ibd_kv', 'ped_ibd_kv_test', 0.01, 0.99, 0.04);
```

Figure 5.2: Example call to ROC data generator

In Figures 5.3 and 5.4, we show some results of our analysis. The ROC curves indicate that for conservative labeling, the model is better than random. It turned out that targeting

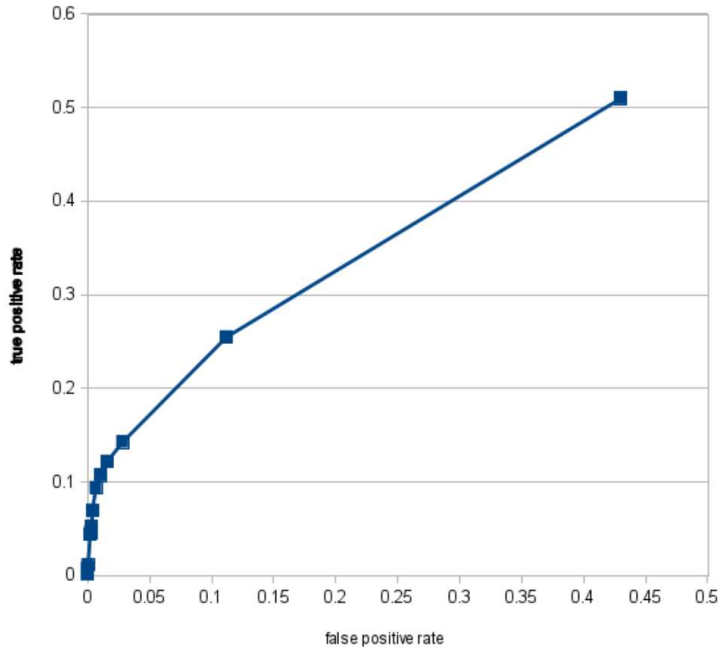

Figure 5.3: ROC curve for predictive model of IBD based on patient encounter diagnoses only using sparse_logit default parameters

a model that would predict a primary diagnosis of IBD was more effective than trying to produce a model that would predict any diagnosis of IBD. Figure 5.4 depicts the ROC curve of a model predicting a primary diagnosis of IBD trained on hospital account records, while Figure 5.3, depicts the ROC curve of a less accurate model predicting any diagnosis of IBD trained on patient encounter records. levelbased on hospital account level data. Both ROC curves depicted are based on applying `sparse_logit` with its default parameters.

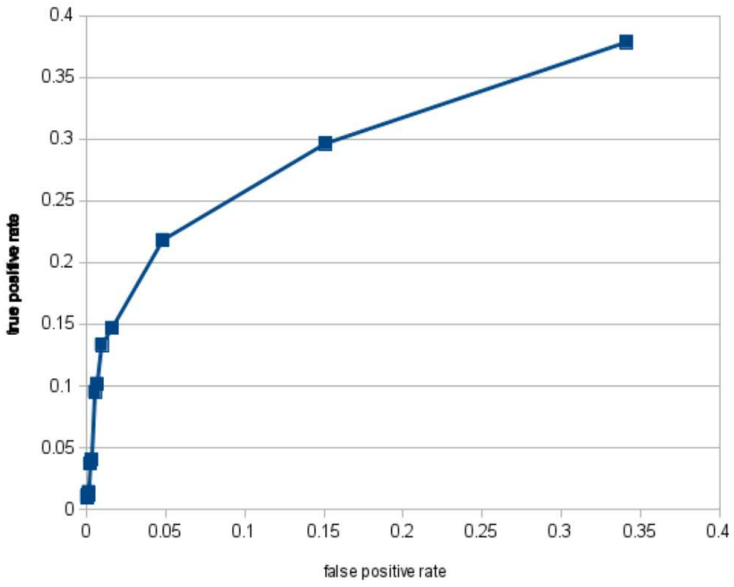

Figure 5.4: ROC curve for predictive model of IBD as a primary diagnosis based on hospital account diagnoses only using sparse_logit default parameters

5.2 Discussion

Naturally arising from our work was a model builder framework with the Bismarck architecture at its core. The scripts and queries used in pivoting the data into key-value structures, labeling the data, training and testing the models, and evaluating the test results became the components of the framework. This framework is depicted in Figure 5.5. For data visualizations, we made preliminary investigations into the use of plv8js, Google's V8 engine

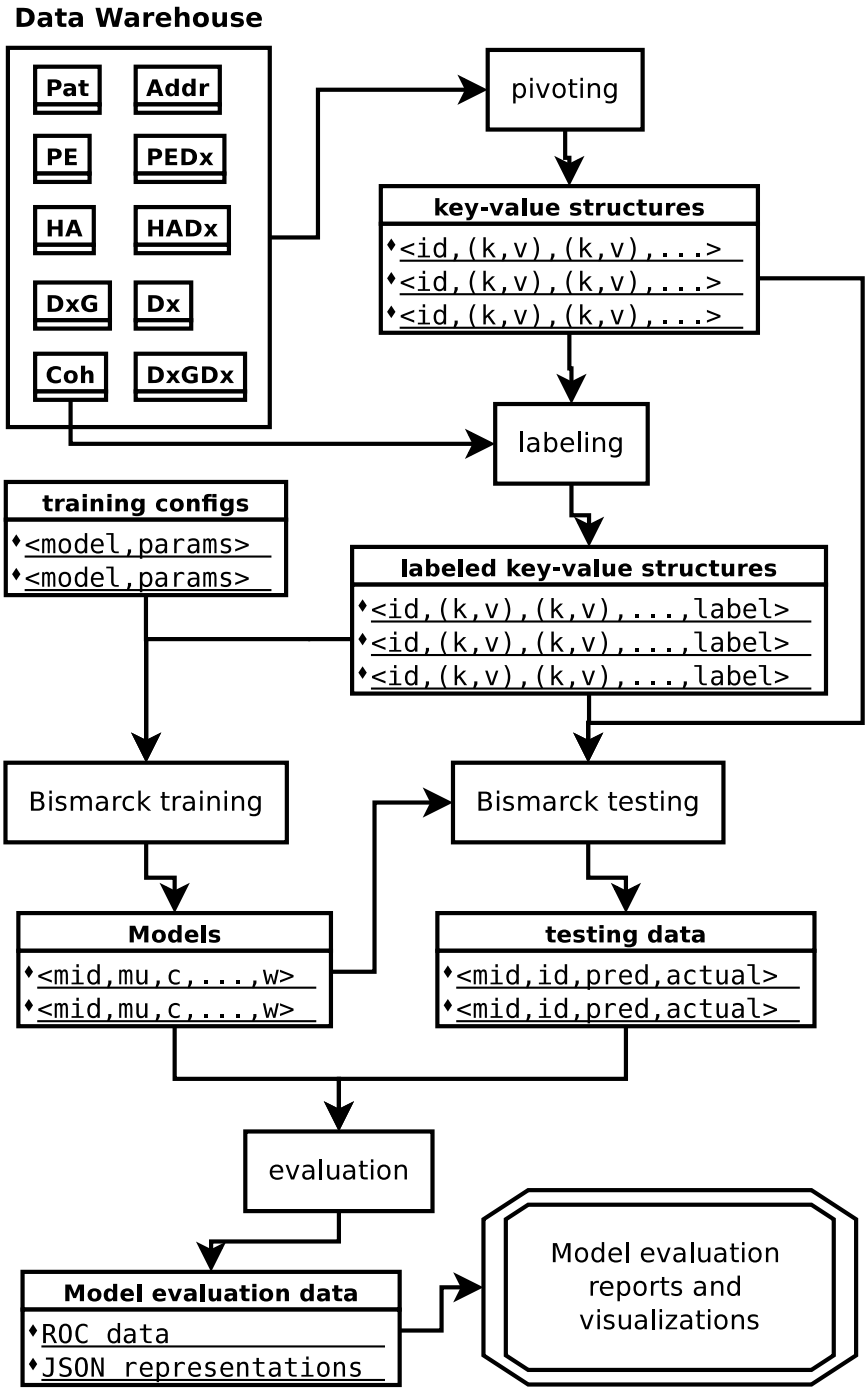

Figure 5.5: The model builder framework

javascript procedural language extension for PostgreSQL [5] together with d3.js, a javascript library for manipulating documents based on data, emphasizing web standards [2].

From our conversations with clinicians, providing a raw number alone representing risk will not be as impactful as offering an interactive tool to perform investigations of the data informed by the predictive model. Also, although we've produced a predictive model that's better than random, it is the method that provides the most benefit. We predict such data mining can be performed for other cohorts of interest quite easily.

Chapter 6
FUTURE RESEARCH

Our work highlights the flexibility and applicability of the Bismarck in-database architecture in a healthcare environment, as well as the feasibility of building a simple framework for automatically building and evaluating predictive models, driven by queries of data sourced out of the electronic medical record, informed by domain expertise. As described below, we see three future areas of research: increased scope of the attribute space, development of interactive data visualizations, and domain expansion of the Bismarck algorithm.

6.1 Model Builder Framework

The model builder framework needs some additional development and simplification. Furthermore, there are opportunities to run many more models and tests within the current data scope.

- *Model builder framework.* The scripts and components involved in building the models and collecting the data should be simplified to provide robustness and ease of debugging. We'd like to pursue development to facilitate data scientist work to proceed more smoothly.

- *Expanded runs of model types and parameters.* Even with the currently scoped data, there are ample opportunities to build and test many models, including using models other than sparse logistic regression, to, say, SVM. As well, we should expand the range of parameters passed to the Bismarck architecture in Figure 4.6 to build the model.

6.2 Increased Scope of the Attribute Space

There are several natural elements that can be brought into the attribute space:

- *Ethnic data.* Since there is a known higher incidence of IBD among people of Jewish descent, we'd like to include ethnic features [14, 18].

- *Genetic information.* There are some genetic markers that have a correlation with the occurence of IBD that we may be able to utilize. Investigations into where these are captured in the EMR or other ancillary system would need to be made [10, 19].

- *Geospatial data.* In our data set, there are some preliminary indications of higher incidence of IBD by a patient's address. This may simply be due to clustering of, say, a Jewish population, or there may be some underlying environmental factors. We'd like to expand our attribute space to include geospatial data.

- *Medication history.* We have a rich set of medication history within the EMR. Are there any correlations with medications and the occurence of IBD?

With the flexibility of the Bismarck architecture, and our model builder framework, increasing the scope of the attribute space is only limited by the availability of the data and the available domain expertise to source the data from the EMR and build the appropriate queries.

6.3 Development of Interactive Data Visualizations

Care providers and researchers have shared interest in being able to "interactively work with the data", both at the point of care, and during cohort analysis.

- *Patient scenario builder.* Provide an interface with "slider bars", "dials", and "selection boxes" to capture age, gender, ethnicity, medications, historical diagnoses, comorbidity, family history, genetic markers, personal habits, dietary intake, and geographic location to visualize how the predicted risk factor changes, as exhibited in [20].

- *Cohort and trend charts.* Provide visualizations to display predicted risk factor trended over time for selected cohorts.

- *Model ranking visualizations.* Provide the data scientist with a visual catalog of model assessments for generated models. Utilize the d3.js Javascript library.

6.4 Domain Expansion and Bismarck Algorithm Extension

We see expanding the domain of application of the Bismarck architecture and its ideas in two ways.

- *Geospatial data types.* To support our increased scope into geospatial data, we want to investigate extending Bismarck algorithms to efficiently process spatial data types.

- *Streaming environments.* The ideas behind the Bismarck architecture may be capable of being applied in a streaming environment. We'd like to investigate this avenue, as offering care providers and others the ability to examine risk factors in real time would be very useful.

Chapter 7

CONCLUSION

Inflammatory Bowel Disease is a life-changing disease and impacts the lives of millions of people around the world. With a lack of causal factors, care providers must use empirical evidence alone to define it and, therefore, electronic medical records capture much of the information used in diagnosis of IBD. Since EMR systems are typically large and hold highly dimensional, heterogenous, and noisy data, it was important that the data mining techniques we applied were compatible with these requirements.

From our work, we discovered the flexibility and applicability of the Bismarck in-database analytics framework in a healthcare setting. We developed predictive models of increasing accuracy against heterogenous, noisy data, and quickly built up a framework to generate predictive models driven by queries against data sourced from an EMR informed by domain expertise. We exhibited some interesting findings for predicting diagnosis of IBD, and we utilized structures that could easily be applied to various cohorts of interest. There are several avenues of future research that may prove useful to further knowledge in this area and, more importantly, improve the lives of patients.

BIBLIOGRAPHY

[1] Crohn's and colitis foundation of america. *http://www.ccfa.org/what-are-crohns-and-colitis/.*

[2] d3.js: Data driven documents: a javascript library for manipulating documents driven by data. *http://d3js.org.*

[3] Madlib website. *http://madlib.net/.*

[4] plr: the r procedural language add-on for postgresql. *http://www.joeconway.com/plr/.*

[5] plv8js: the v8 engine javascript procedural language add-on for postgresql. *http://code.google.com/p/plv8js.*

[6] Postgresql 9.1 documentation. *http://www.postgresql.org/docs/9.1/static/index.html.*

[7] The foundation for clinical research in inflammatory bowel disease - advances in ibd. *http://myibd.org/PatientEducation/AdvancesinIBD/*, 2004.

[8] Léon Bottou. Online learning and stochastic approximations, 1998.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[10] Richard H. Duerr, Kent D. Taylor, Steven R. Brant, John D. Rioux, Mark S. Silverberg, Mark J. Daly, A. Hillary Steinhart, Clara Abraham, Miguel Regueiro, Anne Griffiths, Themistocles Dassopoulos, Alain Bitton, Huiying Yang, Stephan Targan, Lisa Wu Datta, Emily O. Kistner, L. Philip Schumm, Annette T. Lee, Peter K. Gregersen, M. Michael Barmada, Jerome I. Rotter, Dan L. Nicolae, and Judy H. Cho. A genome-wide association study identifies il23r as an inflammatory bowel disease gene. *Science*, 314:1461–3, 2006.

[11] Xixuan Feng, Arun Kumar, Ben Recht, and Christopher Ré. Towards a unified architecture for in-rdbms analytics. *CoRR*, abs/1203.2574, 2012.

[12] T.J. Hastie, R.J. Tibshirani, and J. Friedman. *The elements of statistical learning. Data mining, inference and prediction.* Springer-Verlag, New York, 2001.

[13] Joe Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. The madlib analytics library or mad skills, the sql. In *PVLDB*, 2012.

[14] HT Lynch, RE Brand, and GY Locker. Inflammatory bowel disease in ashkenazi jews: implications for familial colorectal cancer. *Fam Cancer*, 3:229–32, 2004.

[15] Laurence Scott Bailen M.D. Inflammatory bowel disease i. *http://ocw.tufts.edu/Content/48/lecturenotes/571273*.

[16] University of Wisconsin at Madison. Hazy project website - bismarck. *http://hazy.cs.wisc.edu/hazy/victor/bismarck/*.

[17] Prasad Raghavendra. Optimal algorithms and inapproximability results for every csp. In *In Proc. 40 th ACM STOC*, pages 245–254, 2008.

[18] MP Roth, GM Petersen, C McElree, E Feldman, and JI Rotter. Geographic origins of jewish patients with inflammatory bowel disease. *Gastroenterology*, 97:900–4, 1989.

[19] J Satsangi, MS Silverberg, S Vermeire, and J-F Colombel. The montreal classification of inflammatory bowel disease: controversies, consensus, and implications. *GUT*, 55:749–53, 2006.

[20] CA Siegel, LS Siegel, JS Hyams, S Kugathasan, J Markowitz, JR Rosh, N Leleiko, DR Mack, W Crandall, J Evans, DJ Keljo, AR Otley, M Oliva-Hemker, S Farrior, CR Langton, IT Wrobel, G Wahbeh, JA Quiros, G Silber, RJ Bahar, BE Sands, and MC Dubinsky. Real-time tool to display the predicted disease course and treatment response for children with crohn's disease. *Inflamm Bowel Dis*, 17:30–8, 2011.

[21] Rao S.J. Regression modeling strategies: With applications to linear models, logistic regression, and survival analysis. *Journal of the American Statistical Association*, 98:257–258, 2003.

Appendix A

APPENDIX

The query in Figure A.1 generates ROC data for use in analysis of predictive models, labeling each instance as a true positive, true negative, false positive, or false negative.

```
create function generate_roc_data (
  model_id int, labeled_table_name varchar(255), test_table_name varchar(255),
  start_threshold double precision, end_threshold double precision,
  step_size double precision)
declare
  threshold double precision;
begin
  select start_threshold into threshold;
  while threshold <= end_threshold loop
    insert into roc_analysis (
      mid, labeled_table, test_table, threshold, fn, fp, tn, tp)
    select model_id, labeled_table_name, test_table_name, threshold,
        fn, fp, tn, tp
    from crosstab (
      'select ' || cast(model_id as varchar(255)) || ', ' ||
      'case when q.pred = -1 and q.label =  1 then ''fn'' ' ||
      '     when q.pred =  1 and q.label = -1 then ''fp'' ' ||
      '     when q.pred = -1 and q.label = -1 then ''tn'' ' ||
      '     when q.pred =  1 and q.label =  1 then ''tp'' ' ||
      'end roc_indicator, ct ' ||
      'from ( ' ||
      '  select count(1) ct, i.label,  ' ||
      '  case when t.pred <= ' || cast(threshold as varchar(255)) ||
      '  then -1 else 1 end pred ' ||
      '  from ' || labeled_table_name || ' i ' ||
      '  join ' || test_table_name || ' t on i.did = t.did ' ||
      '  group by i.label, ' ||
      '  case when t.pred <= ' || cast(threshold as varchar(255)) ||
      '  then -1 else 1 end) q ' ||
      '  order by roc_indicator',
      'select ''fn'' f union select ''fp'' union select ''tn'' ' ||
      'union select ''tp'' order by f'
    ) as (mid int, "fn" int, "fp" int, "tn" int, "tp" int);
    select threshold + step_size into threshold;
  end loop;
end;
$$ language plpgsql volatile;
```

Figure A.1: Full code for pl/pgsql ROC data generator to support model evaluation

VITA

Eric Johnson is a Data Architect at MultiCare Health System in Tacoma, USA.

edj371@uw.edu

eric.johnson@multicare.org

tokenmathguy@gmail.com