

# Investigations Concerning Applicability of Datamining Procedures to Medical Databases

Todd Iverson  
Hari Iyer  
Department of Statistics  
Colorado State University

Report Prepared for Dr. Syamala Srinivasan

December 4, 2007

## Abstract

We explore various issues that need to be addressed before and during any data-mining activity that uses large databases consisting of insurance claims information. These cover a wide range of topics including hardware needs, software needs, operating system choices, and suitability of proprietary software versus public domain software for data-mining. Additional issues include elicitation of information from medical experts that can guide in the definition of various metrics that will feed into any supervised or unsupervised learning procedures applied to the MarketScan insurance claims data.

This report focuses on two major topics we considered by way of datamining procedures. One is the development of a classification rule using support vector machines. The other is the discovery of association rules using the apriori algorithm and its extensions.

With regard to the first topic we address the question – can common kernel methods using a SVM classifier be used to predict the onset of Type II Diabetes when looking at a four year window of insurance claims? We report the results of a study in which we tested the performance of these methods on data extracted from the MarketScan Database. We summarize the results of applying popular kernels and techniques for support vector machines on data derived from this database. We also look at the learning curve for one example, as well as methods for feature selection. A discussion of the relevant tools and methodology is included.

We were able to predict the Type II Diabetes with a reasonable success rate. Further improvements might be possible using new techniques. A few possibilities are discussed at the end of the paper.

With regard to the second topic, we give a corollary to a recently published theorem on closed rules that could help substantially improve the time to find a specific type of rule. These rules have the form of a collection of diagnoses that imply a procedure or collection of procedures. In addition, we introduce a class of interesting association rules in the context of medical claims databases and illustrate their potential uses by extracting example rules from the MarketScan database.

# Chapter 1

## Introduction

### 1.1 Working with a Large SAS Database

The Marketscan database takes over 160 gigabytes of hard-drive space, and is comprised of 2 gigabyte files that contain the health insurance claims for approximately 8 million patients. There are many hurdles that need to be overcome when mining a database of this size. In the next few sections, I will summarize some of the tools that are needed along the way.

To illustrate the procedures that we use, a small example was constructed and will be used throughout this, and following sections. Here is a graph of the two groups plotted on two features in Figure 1.1. Included with this report are sample SAS tables, SAS code, and PyML code that can be used to follow the steps illustrated in this section. The example database consists of the files `OutPatClaims.sas7bdat`, `InPatClaims.sas7bdat`, and `DrugClaims.sas7bdat`.

#### 1.1.1 Hardware/Computational Issues

The sheer scope of the database operations needed to search and subset a database of this size requires superior hardware. We used was a Dell dual Xeon processor machine with 4 gigabytes of RAM and over 400 gigabytes of hard-disk space.

PyML, a software we used for training the support vector classifiers, requires a Unix or a Linux operating system. It might be possible to get PyML working on a Windows machine, but this would require some technical expertise with *Python* for Windows, as well as the correct C and C++

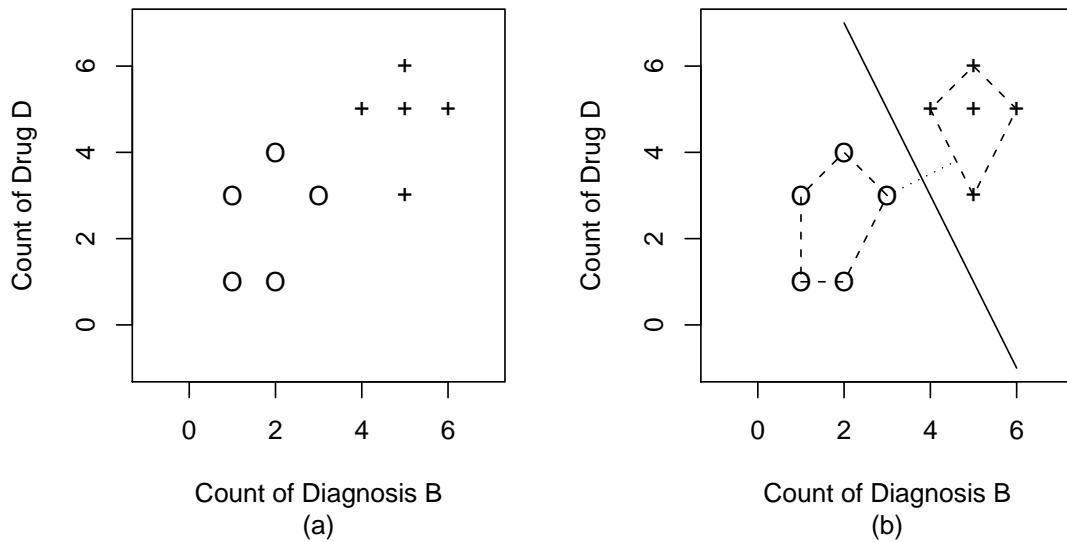


Figure 1.1: Graph of a Working Example

compilers. We did not attempt this ourselves. Instead, a second hard drive was installed and the machine was set-up to dual boot Windows XP Professional Edition and Fedora Core 4 Linux. Installing Linux can be tricky and finding a build that works with your specific hardware is more difficult than with Windows. Having said that, Linux has the ability to be fine-tuned to individual machines and individual specific needs.

### 1.1.2 SAS and SAS SQL

To work with a database of this size, it is necessary to have database management software to allow fast searching, sorting, and subsetting of the data. The database comes as SAS tables out of the box, and SAS has its own database management system built into the base SAS package. We used SAS SQL for all of our database manipulations, and it did an excellent job. SAS is able to open and manipulate files of sizes much larger than the typical 2 GB limit. At one point, SAS was required to operate on a 36 gigabyte file. Although it took a long time to complete, the fact that it could complete such an operation was noteworthy.

In fact, it is this very fact that kept us from moving away from SAS.

At one point, a complete transfer to a Linux database package, such as MySQL was contemplated. The reason that we stayed with SAS is, we are unaware of other statistics packages with the ability to work on files larger than 2 gigabytes. One thing that appears to be sorely missing is a statistical software package that has sparse data storage that will work with all of its modelling methods.

### 1.1.3 Database Management Issues

Patid	Group	Feature 1	Feature 2
1	A	1	1
2	A	2	1
3	NA	4	5
4	A	2	4
5	A	3	3
6	A	1	3
7	NA	5	3
8	NA	5	5
9	NA	5	6
10	NA	6	5

Table 1.1: Data for Working Example

Table 1.1 displays the data for a toy example which we use to illustrate the various required steps in the data mining process. First step is to identify patients that are of interest. Suppose that we wish to identify patients that have received a diagnosis A. We search the database and find that there are 5 such patients. In the actual database, this involves searching many tables across different years. The result is much replication of the code given in SASCode.sas. The example database contains 5 patients that exhibit diagnosis A.

Next we select some negative examples to contrast with our five positive examples. We will take an equal sized sample of patients that never had a diagnosis A. This involves complementing the list of positive examples with the list of all patients and taking a sample of these patients.

Now, we need to identify some variables, or features, that we believe can be used to discriminate between the patients that have, or do not have,

diagnosis A. Imagine that recent literature leads us to believe that the number of times that diagnosis B occurs is related to occurrences of diagnosis A. Also, it is believed that the number of times that drug D is prescribed is related to diagnosis A.

All the claims must be searched for claims that contain diagnosis B and/or drug D. These claims are then counted for each patient. Finally, the data is rearranged in the correct format and exported to a text file. The next section describes the requirements for this output file and gives an example. The code to do this work is given in SASCode.sas. Table 1.2 summarizes the data in this example.

Patid,	Feature,	Value
1,	0,	A
1,	1,	1
1,	2,	1
2,	0,	A
2,	1,	2
2,	2,	1
3,	0,	NA
3,	1,	4
3,	2,	5
⋮	⋮	⋮

Table 1.2: Example SAS Text Output

### 1.1.4 Outputting and Reformatting for Other Software

It soon became clear that being able to manipulate large text files is an important skill. The best method that we have found involves parsing the text with a programming language, such as Python. The tools that are needed involve reading a text file, searching and reorganizing the text, and writing the correctly formatted text to a file. As noted above, SAS is used to subset and sort the data. The data is then exported to a text file.

Our systematic approach requires that the data resides in a table with three columns. The first column contains the patient identifier. The second column contains a list of integers, each representing a unique feature (or variable). The final column contains the value of the feature for the given

patient. The data is sorted by the patient identification number first, then the feature number.

The output for our small example is shown in Table 1.2. The class variable that is being studied is given the feature number 0.

Next, the data is put into the sparse format for input into PyML. This small example is not actually a sparse example, but examples from the true database are likely to be sparse. The code in the file PythonCode.py can be used to transform the above text file. The outcome is a file, of which the first few lines given in Table 1.3. Each line represents a patient. The first value is the class label for the patient. Subsequent values are given in the form of *Feature:Value*. This format is very efficient when there are relatively few non-zero values.

A	1:1	2:1
A	1:2	2:1
NA	1:4	2:5
⋮	⋮	⋮

Table 1.3: Example Sparse Format

# Chapter 2

## Methods

The next two sections will give a brief review of the development of Support Vector Machines and Kernel Methods. The development is a bit technical and is meant for those interested in the mathematical details underlying the SVM approach. Readers not interested in the technical details can skip to the last section for a summary of the important results.

### 2.1 Kernel Support Vector Machines

Now let's review the development of support vector machines (SVM). There are two parts to this learning method. The first is a linear classification method that is adapted from the idea of an optimal separating hyperplane. The second is replacing the usual euclidean inner product with a kernel function. This has the effect of representing the data in a suitable vector space where a good linear classifier may be available. The net result is an efficient nonlinear classifier arrived at using linear techniques. Let's take a closer look.

#### 2.1.1 Linear Classification Through Optimal Separating Planes

In this section, we will follow the work from [1]. The methods of this section assume that we have a two class classification problem and that the groups are linearly separable. That is, there exists a hyperplane that completely separates the two groups. The case of classifying with more than 2 groups is

considered in 2.1.3 while groups that are not linearly separable are discussed in 2.1.2.

We will use a representation for a hyperplane (a line if the data represented in a two-dimensional euclidean space) that will be convenient for discussions contained in subsequent sections. Notice that a plane  $w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b = 0$  can be written as

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \tag{2.1}$$

where  $\langle \mathbf{w}, \mathbf{x} \rangle$  denotes the inner product of the vectors  $w = (w_1, \dots, w_n)$  and  $(x_1, \dots, x_n)$ . The side of the line that a point lies on can be determined by the following function,

$$f(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \tag{2.2}$$

as all point on the same side of the hyperplane will share the same sign. This hyperplane will be used to discriminate between the groups, and is referred to as the *decision function*.

Consider the Toy Example from the last section in Figure 1.1(a). Here we have a case where the data are linearly separable, that is the two groups can be completely separated by a line (a hyperplane in two-dimensions is a line).

As there are lots of lines that separate the two groups, consider the question: Which line is the "optimal" line? There are many ways that we could define "optimal," one being the line that has the largest margin. The *margin* is the distance between the line and it's closest point. Figure 1.1(b) on page 2 shows the line with largest margin. Geometrically, this line can be found by:

1. Drawing the Convex Hull around each group.
2. Finding the two points on each hull that are closest to each other.
3. Finding the perpendicular bisector of the line segment connecting the points in 2.

The line from 3 is the optimal separating plane, that is it has the largest margin.

In mathematical terms, this boils down to the following optimality problem:

$$\max_{\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}} \min \{ \|\mathbf{x} - \mathbf{x}_i\| \mid \mathbf{x} \in \mathbb{R}^n, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \dots, n \} \quad (2.3)$$

We will rework the optimization in the following way: we rescale  $\mathbf{w}$  and  $b$  such that the margin is unit length. Suppose that  $x_{pos}$  is the closest point on positive side and  $x_{neg}$  is the closest point on the negative side. The rescaling gives  $\langle \mathbf{w}, \mathbf{x}_{pos} \rangle + b = 1$  and  $\langle \mathbf{w}, \mathbf{x}_{neg} \rangle + b = -1$ . This implies that  $\langle \mathbf{w}, (\mathbf{x}_{pos} - \mathbf{x}_{neg}) \rangle = 2$  and thus  $\left\langle \frac{\mathbf{w}}{\|\mathbf{w}\|}, (\mathbf{x}_{pos} - \mathbf{x}_{neg}) \right\rangle = \frac{2}{\|\mathbf{w}\|}$ . This tells us that a large margin is the result of a small  $\mathbf{w}$ . Thus, the optimization problem can be written

$$\min_{\mathbf{w} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{w}\|^2 \quad (2.4)$$

$$\text{subject to } y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \text{ for all } i = 1, \dots, n \quad (2.5)$$

We solve a constrained optimization problem like this using *Lagrangian Multipliers* (see [1, page 166]). The *Lagrangian* for this problem is

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1), \quad (2.6)$$

where  $\alpha_i \geq 0$  for all  $i = 1, \dots, n$ . Notice this is each constraint multiplied by the Lagrangian multiplier and subtracted from the optimization function  $\frac{1}{2} \|\mathbf{w}\|^2$ . The Karush-Kuhn-Tucker (KKT) (see [1, page 170]) conditions tell us that a solution must occur at a saddle-point where  $L$  is maximized with respect to  $\alpha$  and minimized with respect to  $(\mathbf{w}, b)$ . Thus

$$\frac{\partial}{\partial b} L = 0 \text{ and } \frac{\partial}{\partial \mathbf{w}} L = 0 \quad (2.7)$$

which implies

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (2.8)$$

and

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i. \quad (2.9)$$

The KKT conditions also tell us that  $\alpha_i(y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1) = 0$  for all  $i = 1, \dots, n$ . Thus, either  $\alpha_i = 0$  and it doesn't matter what value  $y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b)$  takes or  $y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1 = 0$  which indicates that we are right on the edge of the margin. Points on the edge of the margin are called *Support Vectors*.

Now we develop the *dual* optimization problem by substituting 2.8 and 2.9 into 2.6 which gives us a new form for the optimization that no longer relies on  $\mathbf{w}$  and  $b$ .

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^n \alpha_i - \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (2.10)$$

$$\text{subject to } \alpha_i \geq 0 \text{ for all } i = 1, \dots, n \text{ and } \sum_{i=1}^n \alpha_i y_i = 0. \quad (2.11)$$

This is a quadratic optimization problem for which there are many known algorithms.

Now it is time to make two very important points about Support Vector Machines. First, notice that we are optimizing on  $\alpha$  and need to find  $n$  values, where  $n$  is the number of observations in the data set. Thus, the time to train a SVM only depends on the number of observations, not the dimension of  $\mathbf{x}$ . This makes SVM a great method for high-dimensional data.

Second, the only information that we need about the vectors  $\mathbf{x}_i$  are the inner products  $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ . This leads to the *Kernel Trick*. Suppose we have a function  $k$  such that  $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$  for all  $i, j = 1, \dots, n$ . Then we need not store  $\mathbf{x}_i$  but only be able to evaluate  $k$ . As we will see in the next section, this leads to a nice method for developing non-linear classification methods.

## 2.1.2 Soft Margin Classifiers

Thus far we have assumed that the data is linearly separable. We can use a so-called *Soft Margin* method to deal with linearly non-separable cases by introducing a penalty for margin errors. We rewrite (2.5) as

$$\min_{\mathbf{w} \in \mathbb{R}^n, \xi \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{n} \sum_{i=1}^n \xi_i \quad (2.12)$$

$$\text{subject to } \begin{cases} y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \text{ and} \\ \xi_i \geq 0 \text{ for all } i = 1, \dots, n \end{cases} \quad (2.13)$$

where  $C$  is a given cost parameter associated with the slack variables  $\xi$ . Applying Lagrangian multipliers and the KKT conditions leads to a similar dual problem as (2.11)

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^n \alpha_i - \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (2.14)$$

$$\text{subject to } 0 \leq \alpha_i \leq \frac{C}{n} \text{ for all } i = 1, \dots, n \text{ and } \sum_{i=1}^n \alpha_i y_i = 0. \quad (2.15)$$

Once again, solving the problem depends only on the number of observations and not the dimension of  $\mathbf{x}$ . The parameter  $C$  is a tuning parameter that can be chosen using cross validation.

### 2.1.3 Multi-Class Classification with SVMs

What do we do if the problem has more than two classes? Suppose that we have  $k$  groups from which to classify. There are a couple of common methods to deal with this situation: *One Versus the Rest* and *Pairwise Classification*.

The One Against the Rest method trains one classifier for each class. For each group  $i$ , the data is recoded as  $+1$  for items in group  $i$  and  $-1$  for items that are not. A classifier is then trained on these two groups, giving a decision function  $f_i$  similar to Equation 2.2 on page 7. A new data point is assigned a label by evaluating it on each of the  $k$  decision functions and given label  $m$  where  $f_m = \max_{i=1, \dots, k} f_i(x_{new})$ . Computationally, we are required to train  $k$  SVM classifiers, each on the full data set.

The Pairwise Classification method builds a two group SVM classifier for each pair of groups. The label assigned to a new data point is found by running all  $\frac{k(k-1)}{2}$  classifiers. Each time a pairwise classifier picks a group, that group gets a vote. The final label is the group with the highest number of votes (see footnote 14 [1, page 212] regarding ties). Computationally, we must train  $\frac{k(k-1)}{2}$  classifiers, but each pairwise data set is only a fraction of the full data set. As mentioned on [1, page 212], as each classifier is trained on a smaller number of observations; we can see an improvement over the performance of one versus the rest even though we trained many more classifiers. For very large  $k$ , this method becomes unwieldy.

## 2.1.4 Linear Classifiers and the Kernel Trick

Consider the example data shown in Figure 2.1(a) where there are again two groups of points. It is clear that there is no line that is able to successfully separate the two groups of points, that is, the data are definitely not linearly separable. Suppose that we apply the map  $\phi$  that takes  $(x_1, x_2)$  to  $(x_1^2, x_2^2)$ . Figure 2.1(b) shows the points in the new space, which are now linearly separable. The line that is shown is the result of applying a SVM classifier. The points on this line can be translated back to the original space, as shown in Figure 2.1(c). Notice that the net result is a non-linear classifier.

Recall that the only values needed from the space  $(x_1^2, x_2^2)$  are the inner products, which are given by the function  $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle = x_1^2 x_2^2$ . We call such a function a *kernel*. In practice, we never actually do the transformation from  $(x_1, x_2)$  to  $(x_1^2, x_2^2)$ . Instead we simply evaluate  $k(x_1, x_2)$ .

This example illustrates the *Kernel Trick*. We imagine a mapping  $\phi$  from the data  $\mathbf{u}$  to some vector  $\mathbf{x}$  in a Hilbert Space  $\mathcal{H}$ . For technical reasons, we need the mapping to land in a Hilbert Space, which is a vector space with an inner product in which every Cauchy sequence has a limit that belongs to the space. Notice that the original data  $\mathbf{u}$  need not be a vector. This allows Kernel Methods to be applied to data that can be represented by a graph or some other non-vector format.

The kernel function  $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$  is constructed and used to train a linear classifier in the new space  $\mathcal{H}$ . The section above illustrated how the kernel trick could be applied to a SVM. There are many other linear methods that can take advantage of the kernel trick, including Linear Discriminant Analysis, Principle Component Analysis, and Ridge Regression.

Clearly, a kernel exists for any mapping to a Hilbert Space. It is defined by  $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$ . The next section answers the question: Given a function  $k$ , is there a mapping  $\phi$  from the data to some Hilbert Space  $\mathcal{H}$  such that  $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$ ? If we can answer this question, we can simply construct kernels knowing that there is a corresponding mapping and Hilbert space without ever needing to construct this mapping. This will allow us to, for instance, consider mappings into infinitely dimensional spaces.

## 2.1.5 Kernel Properties and Popular Kernels

What properties must a function  $k$  have for there to exist a mapping  $\phi$  into a Hilbert Space  $\mathcal{H}$  such that  $k(\mathbf{u}_i, \mathbf{u}_j) = \langle \phi(\mathbf{u}_i), \phi(\mathbf{u}_j) \rangle$ ? For a given set of

observations  $\mathbf{u}_1, \dots, \mathbf{u}_n$ , we construct a matrix  $K = (k(\mathbf{u}_i, \mathbf{u}_j))_{i,j}$ , which we will call the *Gram matrix*.

The necessary and sufficient conditions for  $k$  to be the kernel for some mapping  $\phi$  into some Hilbert space  $\mathcal{H}$  is: for every set of observations  $\mathbf{u}_1, \dots, \mathbf{u}_n$  the Gram matrix  $K$  is positive definite. Recall that a matrix is positive definite if it has all positive eigenvalues.

Here are some examples of some popular kernels. These kernels are all defined for data that is in vector form. The most basic kernel is simply the regular inner product and is called a linear kernel:

$$\text{Linear Kernel: } k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (2.16)$$

Another popular kernel is the polynomial kernel. It corresponds to the vector space of polynomial terms of the observations.

$$\text{Polynomial Kernel of Degree } d: k(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + a)^d \quad (2.17)$$

The value  $a$  is an additive constant that needs to be specified. We refer to such model parameters as *tuning parameters*. Typically, cross validation is used to determine tuning parameters. See 2.2.2 for further discussion.

Our last example of a common kernel is the Gaussian Kernel. It too has a tuning parameter, in this case  $\sigma$ . Notice the resemblance to the Gaussian, or Normal, distribution. This gives us some intuition into the workings of this kernel.

$$\text{Gaussian Kernel: } k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \quad (2.18)$$

The Gaussian kernel is very sensitive to normalization and the choice of  $\sigma$ . We need for values of the exponent to be generally be between -3 and 3 for this kernel to perform satisfactorily. Thus, the size of *gamma* needs to decrease as the dimension of  $\mathbf{x}$  increases.

The Gaussian kernel is an example of a kernel that represents a mapping into an infinite dimensional Hilbert space. Without the kernel trick, we would never be able to consider such a space.

We tested each of these kernels for our classification problem. All these kernels proved to be sensitive to normalization. The linear and polynomial kernels had much better training times when the feature vectors were normalized to length 1. The Gaussian kernel worked best when applied with no normalization.

There are methods for constructing new kernels by applying functions to existing kernels. Multiple kernels can also be combined in various ways. This report does not use any of these methods, and they will be the topic of further exploration.

### 2.1.6 Summary

In this section, we reviewed the development of Support Vector Machines and the Kernel Trick. A Support Vector Machines is a method of linear classification that finds the hyperplane with the largest margin that separates the classes. A Soft Margin method was introduced to deal with cases where the data are not linearly separable. One advantage of using a SVM classifier is that they lead to a quadratic optimization problem which is relatively easy to solve. A second advantage is that the time to train the SVM depends only on the number of observations (number of patients) and not the dimension of the vector  $\mathbf{x}$ . This allows SVMs to work with high dimensional data.

The Kernel Trick conceptually involved mapping the original data to a (typically) higher dimensional vector space. A good mapping leads to linear separability with a large margin in the new vector space. SVM classification is performed in this new space, leading to a simple nonlinear classification method. Because the SVM classifier only depends on the inner products in the new vector space, we define the kernel  $k$  which takes a pair of observations in the original data as inputs and outputs the inner product in the new vector space. This way, we never actually need to transform the original data into the new vector format, but only evaluate the kernel.

In practice, we rarely define the mapping from the original data to the new vector space. Instead, we use methods that guarantee that a kernel corresponds to the inner product in some vector space. A practitioner will explore existing kernels and develop new kernels for the problem at hand.

## 2.2 Applying Kernel Support Vector Machine

In this section, we will give an overview of the methods and tools that we use to apply Kernels and SVM classification to the diabetes data set discussed in Section 2.3. This includes how we selected tuning parameters, the software used to train and test the classifiers, and methods of feature selection.

## 2.2.1 Kernel Support Vector Machines and PyML

The software that was used to train and test the SVMs is a Python module called PyML developed by Dr. Asa Ben-Hur. Python is a high-level language with some unique advantages. It is very easy to read and write code in Python. Python comes with an interactive interpreter that makes writing and testing code very simple. Finally, it is easy to integrate Python with other languages such as Fortran, R, or C++.

Some of the advantages of PyML include

1. A sparse data representation that works with all the available tools,
2. Cross validation tools built into all methods, and
3. Composite methods that simplify complex tasks such as cross validating feature selection methods.

PyML has only been tested on Linux and Unix operating systems. This might make it a challenge for a Windows users to initially install the software. There are options, including virtual environments that allow linux to run as a windows application. Some of these include Cygwin, VMware, and Colinux.

## 2.2.2 A Note on Tuning Parameters

When using a Soft-Margin SVM as a classifier, we are faced with selecting the penalty parameter  $C$ . For certain kernels, such as the polynomial (2.17) and Gaussian (2.18), we face additional tuning parameters that need selection. The common approach to selecting such tuning parameters is optimizing the performance of the classifier with cross validation.

To perform  $K$ -fold cross validation, the data is divided into  $K$  (approximately) equal-sized random groupings called *folds*. For each fold, we train a classifier on the rest of the data and this classifier's performance is evaluated on the current fold. This process is completed for each fold, and statistics are computed to summarize the performance of the classifier. We will be using one such statistic, the balanced success rate, which measures the percent of correct classifications balanced to group size.

The selection of the number of folds,  $K$ , is made by balancing the variability of the statistics with the computational cost. For this application, we used 5 fold cross validation because the computational costs are high.

Investigation of repeated 5-fold cross-validation seemed to indicate that the cross-validation results are quite reliable.

One method for finding the optimal value of a tuning parameter would involve selecting a grid of values for the parameter, performing cross-validation to determine a (near) optimal value. The value would then be refined by searching in the neighborhood of this value. For large problems this method is not feasible. We have not found much work that apply to data sets of the size we are working with. We hope to develop some methods for parameter search in future research.

Due to time constraints, our exploration of the tuning parameters was limited. The results that we give in Chapter 3 are in no way optimal. One interesting discovery was the time for the training method to converge was highly dependent on the values of parameters selected. This lead an interesting dilemma. For example, for polynomial kernels, larger values of the penalty term and larger values of the additive constant always seemed to lead to better results in cross-validation. The time to train these classifiers increased as well, going from 30 minutes to train a SVM with a degree 3 polynomial kernel with small parameters ( $C = 1$  and  $a = 1$ ) to 4 to 5 days to train a degree 3 polynomial with moderate parameters ( $C = 10$  and  $a = 10$ ). As you can see, exploring all values of these parameters has practical computational limitations.

### 2.2.3 Feature Selection

The starting data set (described in Section 2.3) include all of the diagnosis, procedures, and drug prescriptions in the claim history. For example, if a patient went to their doctor for a common cold, this would be present in the data set. Thus, many of the included features probably don't affect diabetes, and act as noise. SVMs work well in high dimensions, but important features still might be washed out by all the less meaningful features.

PyML includes many methods for feature selection. The simplest method involves *eliminating sparse features*. A sparse feature is a feature that is zero for all but a few patients.

Another method is the *Filter Method*. The method calculates a score for each feature that is similar to a  $t$ -score. There are a number of scoring functions from which to choose. The user supplies a number of features to keep and the preferred scoring function, and the method removes the remaining features by the relative size of their score.

The another method is Recursive Feature Elimination (RFE). This method requires the user to supply the number of desired features  $m$  and a percent of features to eliminate at each iteration  $p$ . At each step, the method trains a SVM and eliminates  $p$  percent of the features with the smallest weight value  $w$ . The process is repeated until  $m$  features remain.

The final method is Multiplicative Update (i.e. Zero Norm Method). This method is similar to RFE but instead of stopping at a specified number of features, it attempts to converge to the optimal number of features.

The computational costs of eliminating sparse features and the filter method are small. RFE and Multiplicative update are much more costly in time to completion.

Additionally, we tested a composite method that involved first eliminating features that only occurred 5 or fewer times. Next, a filter method was applied to reduce the number of features to 4000. Finally, the RFE or Multiplicative Update was applied to further eliminate features.

## 2.3 Data Set Construction

The data was extracted from the MarketScan Database using the years 1999 to 2003. We first decided on a subset of patients that had at least one diabetes diagnosis. First, it was decided that of the patients that had been diagnosed with type II diabetes, we would only include patients that:

1. had at least four years of claims history prior to their first diabetes diagnosis,
2. had complete drug claims history for the four years,
3. and were at least 23 years of age at the end of the four year history.

There were approximately 16,000 patients that fit this description. An equal number of patients that had no diagnosis of type II diabetes were selected at random. These patients were required to have at least a four year claims window and be at least 23 years of age at the end of this window.

NOTE: From this point forward, "Positive Patients" will denote those patients with a diabetes diagnosis, and "Negative Patients" will refer to those that did not have a diabetes diagnosis.

Next, the claims for these patients were collected. For the positive patients, all claims within four years of the first diabetes diagnosis were collected. For negative patients, the most recent four years worth of claims were collected.

All of the popular kernels require a data set in the form of vectors. The demographic information for the patients was already in vector form. There are a number of ways to get the cost and frequency of claims, as well as the diagnosis, procedure, and drug code information into this form. The next two sections describe two methods of "vectorizing" the claims data.

### **2.3.1 The Bag of Words Approach**

We consider each of the present diagnosis, procedure, or drug prescription to be it's own category. This method disregards the time of the claim completely, and simply totals the number of times that each category (diagnosis, procedure, etc.) occurs for each patient. For drug prescriptions, the drugs were grouped by their therapeutic class. The total cost and number of claims for the patient is also recorded. This data set contained 16,756 features (or variables).

### **2.3.2 One Year Stratified Bag of Words**

Next, in order to include some information about the timing of the claim, the claims were stratified into four strata. Each stratum contained one year's worth of claims. Each diagnosis, procedure, or drug therapeutic class was split into 4 categories, one for each of the yearly stratum. The totals for these categories was collected within each stratum. The total cost and number of claims were also recorded for each stratum. This data set contained 46,138 features.

### **2.3.3 Alternate Cost Variables**

We constructed an added set of features in an attempt to capture changes in cost in a patients history. We calculated a cumulative cost step function by totaling each patients cumulative cost each fortnight. Conceptually, the idea was to fit various polynomials to interpolate the cost function. It was hoped that various cumulative cost patterns such as a linear increase, quadratic increase, or oscillations would then be detected.

Specifically, we constructed orthonormal polynomials of increasing degree using the Gram-Schmidt process. Then the coefficients in this basis is given by the inner product of the patients' cost vector with the orthonormal polynomial vector. These features were added to the above data set and tested.

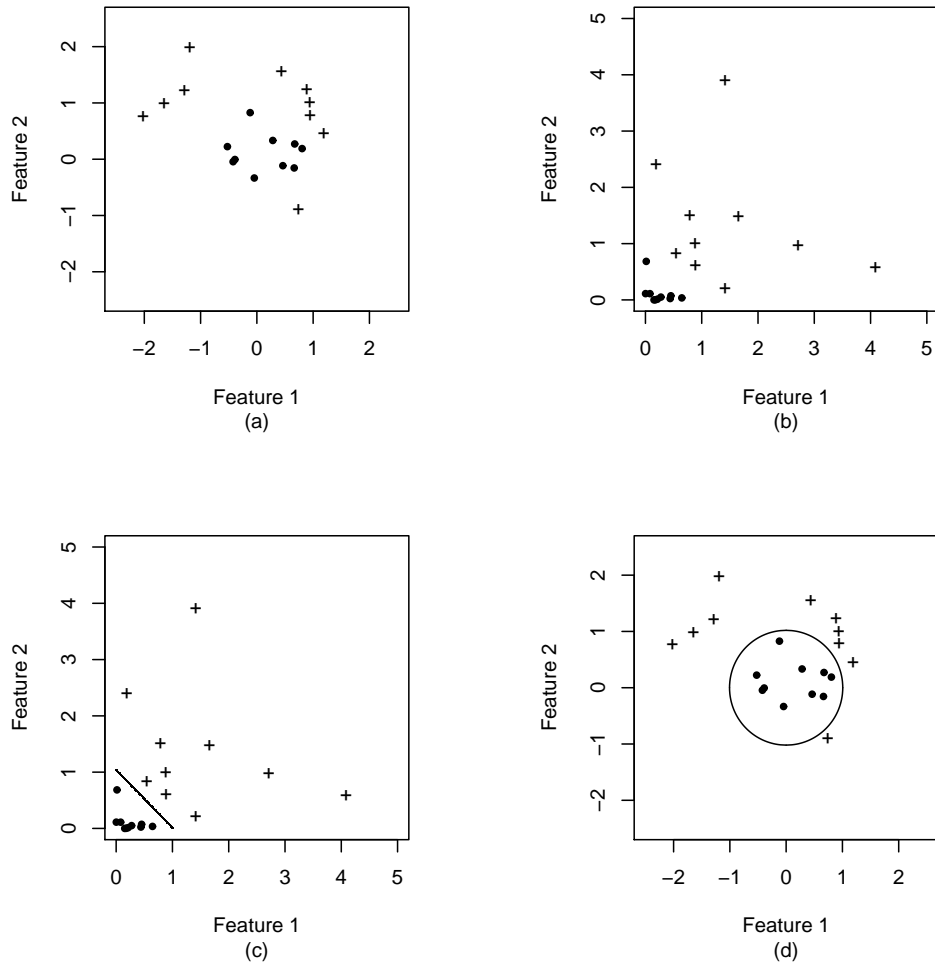


Figure 2.1: Non-Linear Example. (a) Two Groups ● and +. (b) Plot of Square of Each Feature. (c) Optimal Separating Plane in the Square-Square Scale. (d) Optimal Separating Plane Translated In Original Scale.

# Chapter 3

## Results

In this chapter, we will discuss the results we obtained using some of the popular kernels. We also look at the learning curve for a specific example, to get an idea of the performance of the classifier and kernel at various sample sizes. Finally, the results for various methods for feature selection are given.

### 3.1 The Bag of Words Approach

Recall that this vectorization of the data simply counts the number of times each diagnosis, procedure, or drug prescription occurs for a patient. There is no indication of when the claim occurred. This data set also includes any demographic information that we have for the patient. Finally, the total cost and number of claims for each patient is included.

Table 3.1 illustrates the best kernels that we found for each of the four kernels. In each case, it appeared that increasing the value of each tuning parameter would improve the performance. The only exception to this rule is the parameter sigma for the Gaussian Kernel. This parameter was the trickiest to tune and small changes in  $\sigma$  had a much bigger impact on the results than with other kernels. Note that smaller values were required to best the performance of the other kernels.

It must be noted that increasing the value of these parameters also has a profound effect on the time for the training method to converge. For this reason, we did not fully explore the parameter space. The best cross validation results for this data set were obtained with a Gaussian kernel.

Kernel	Tuning Parameters	Balanced Success Rate
Linear	$C = 100$	0.614
Polynomial Degree 2	$C = 1, a = 100$	0.648
Polynomial Degree 3	$C = 1, a = 10$	0.661
Gaussian	$C = 10, \sigma^2 = 0.001$	0.695

Table 3.1: Results for Various Kernels on the Bag of Words Data Set

Sample Size	Balanced Success Rate
1000	0.556
5000	0.563
10000	0.569
15000	0.568
20000	0.572
25000	0.573
32400	0.577

Table 3.2: The Learning Curve for a Linear Kernel and the Bag of Words Data Set

## 3.2 The Learning Curve and Sampling

It was of interest to look at the learning curve for one example. The learning curve is the plot of the function that takes sample size as it's input and outputs the average balanced success rate for all sample of that size.

We needed to run many replications on various sized samples, so a kernel was selected that had a fast training time. We then ran 5 trials at each sample size, each trial consisting of a 5 fold cross validation. The results are illustrated in Table 3.2.

We see that, while more observations do improve the learning rate, the difference between 20,000 patients and all 32,400 patients is not that great. As the training time is on the order of the number of observations cubed, we could use this fact in subsequent runs to save time.

### 3.3 One Year Stratified Bag of Words

Recall that this approach stratified the claims by year and then used the Bag of Words approach within the year. This meant that the same diagnosis code on different years was considered a different feature. This led to 46,138 features. Included were the yearly totals for cost and number of claims.

In every case, the cross validation results of this data set improved on the results of the last. The table below highlights the best results for various kernels. It must be noted that the time to train a SVM was greater for this data set. For this reason, exploration of the tuning parameters was not as extensive as in the Bag of Words approach. It is our (untested) contention is that further exploration would always lead to better results from this data set.

Kernel	Parameter	Balanced Success Rate	
Linear	$C = 100$	0.704	0.611
Polynomial Degree 2	$C = 1 \ a = 100$	0.712	0.648
Polynomial Degree 3	$c = 1 \ a = 1$	0.687	0.633

Table 3.3: Comparison of One Year Data Set to Bag of Words

Table 3.3 compares the results of the One Year Stratified Bag of Words data set to the Bag of Words data set for various kernels. In almost every case, the results with the One Year data set were about 5% better based on balanced success rate. The exception to this was the gaussian kernel. As the choice of parameter  $\sigma$  is dependent on the dimension, it is not appropriate to compare the data sets on the same values. We were unable to find a specific  $\sigma$  that could compare to the results in Table 3.1, but we feel that further exploration would produce a similar trend.

To illustrate the increased time for training, an attempt to train a polynomial kernel of degree three with  $C = 10$  and  $a = 10$  took nearly 5 days to complete one fold of a cross validation. In future work, we will explore the parameter space more fully using paired comparisons on random samples of observations.

### 3.4 Results using Feature Selection and Popular Kernels

It was felt that the inclusion of all 46,138 features in the One Year data set, many of which are probably unrelated to type II diabetes, could be masking important features. For this reason, we attempted a number of methods of Feature Selection described in 2.2.3. For the most part, the results were similar, or slightly better than the results on the full feature set. We have yet to find a feature selection method that significantly improved the results over the full feature set.

First, let's look at the result of eliminating sparse features. For this trial we used a polynomial kernel of degree 2 with  $C = 1$  and an additive constant of 10. Recall that we eliminate any procedure that was non-zero less than the prescribed number of times. Table 3.4 gives the comparison of the results for various minimum thresholds. You can see that the elimination of the sparse features had little impact on the results. This method takes no time at all to perform. For these reason, our composite method starts by eliminating sparse features with a conservative threshold.

Minimum Threshold	Number of Features Remaining	Balanced Success Rate
0	32400	0.690
32	11454	0.691
160	2696	0.691
320	1499	0.691

Table 3.4: Elimination of Sparse Features

Next, let's look at the results of the filter method. Table 3.5 gives the results after reducing the number of features to various values. For this trial we used a polynomial kernel of degree 2 with  $C = 1$  and an additive constant of 10. You can see that the results for keeping no less that 1000 features is similar to the results using the whole data set. Keeping less than 1000 features does lower the performance, but not by too much. The time to complete this method is minimal. We included this method as the second step in our composite method.

Another approach to reducing the number of features was to condense the

Number of Features Remaining	Balanced Success Rate
100	0.673
500	0.678
1000	0.691
5000	0.690
10000	0.690
15000	0.691
20000	0.691
32400	0.690

Table 3.5: Filter Method for Feature Selection

Number of Digits	Balanced Success Rate
2	0.682
3	0.683
5	0.690

Table 3.6: Combining Features Based on ICD 9 Codes

diagnosis by their ICD 9 codes. The diagnosis and procedures in the database use the common ICD 9 code. We condensed the diagnosis and procedures by pooling codes that shared the first two or three values. Table 3.6 compares the results of collapsing the codes to two or three digits to the full feature set. Again, a polynomial kernel of degree 2 with  $C = 1$  and  $a = 10$  is shown. Results were similar for other kernel/parameter combinations. This method resulted in similar, but slightly lower success rates when combining the diagnosis codes. In future work, we will try to combine the diagnosis codes through a novel clustering method.

We attempted to use the RFE method (see 2.2.3) to eliminate features, but the training time was prohibitive. It took days to finish one 5-fold cross validation. To help speed this training time, we applied a composite method for feature elimination that first eliminated sparse features. Then a filter method was used to further reduce the number of features. Finally, the RFE method was applied on the remaining features.

Table 3.7 gives the results of the first few tests of this method. The first

Final Number of Features	% Eliminated Each Iteration	Balanced Success Rate
32400	0%	0.690
2000	50%	0.690
500	50%	0.690

Table 3.7: Composite Feature Selection Method

line of the table gives the results on the full features set. Specifically, features that had less than 5 non-zero observations were eliminated, then the filter method reduced the number of features to 4000. Then various parameters were given to the RFE method.

# Chapter 4

## Discussion

Overall, kernel methods with a SVM classifier performed respectably well. We were able to predict the onset, or lack of onset, of type II diabetes a little over seven times out of ten. On the first data set, the best kernel was a Gaussian kernel. We found that Gaussian kernels were sensitive to normalization and changes in the tuning parameter  $\sigma$ , making it difficult to find a set of parameters giving results on par with polynomial kernels. A polynomial kernel of degree 3 performed better than a polynomial of degree 2. Perhaps a polynomial kernel of higher degree would work even better.

The learning curve suggested that perhaps we can use smaller sample sizes to test various kernels. In the presence of so many features, many perhaps unrelated to the questions at hand, we felt that feature selection would improve the results. Various methods for feature selection performed just as well as the full feature set. This seems to validate the assumption that many of the features are not necessary. Unfortunately, we were unable to improve the results significantly. It is possible that more exploration would lead to better results, but preliminary results suggest otherwise. Unless a person is interested in a minimal set of important features, common methods might not be worth the extra time and complexity.

There are many areas for further exploration. We noted an improvement when going from the bag of words approach to the one year stratified bag of words approach. A finer stratification needs to be tested to see if more information about relative time is helpful. The next step would be to compare the current results to a six month stratification. This new data set would have nearly twice the features as the one year stratification, close to 90,000. We might find that feature selection becomes important for this data set

and other methods of feature selection need to be explored. These include combining features through clustering and designing an experiment using highly fractionalized methods.

Another area of future exploration is developing new kernels. The hope is that a kernel for this problem could take into account more of the time series information present in the database. This new kernel might not require that the data be in vector form.

In conclusion, our study has identified the essential hardware, software, and methodology components that are needed to successfully develop a classification scheme for important health outcomes based on insurance claims data. It was found that both SAS and PyML are important ingredients in this process. PyML is freely available whereas SAS is proprietary software package. Our experience also suggests that the datamining activities are better carried out on a Unix or Linux system. The bottleneck here is the availability of SAS for these systems. Our study had to rely on the windows version of SAS as the licensing cost for SAS on unix or linux is prohibitive. As for methods for classification, SVM methodology definitely is promising. We plan to compare SVM based methods with other classification approaches. Results of such a study will be available sometime in the near future.

# Chapter 5

## Association Rules

Classification rules based on support vector machines are an example of application of supervised learning methods. We have *training data* for which features (predictors) as well as group membership are available for a sample of patients and our task is to train a support vector machine using the sample data. The generalizability of resulting rule(s) can be evaluated by cross-validatory methods.

In contrast to earlier chapter, the current chapter is concerned with unsupervised learning using the Marketscan data. Here we are required to extract *interesting* and *useful* patterns from the database. A standard approach for unsupervised learning involves the mining of association rules. We first review a commonly used algorithm, called the *apriori* algorithm, for extraction of rules. In this connection, we also review the concept of closed rules and discuss some recent results from the literature. We then prove a corollary to these results which is particularly useful when considering mining for rules according to which a set of diagnoses imply a set of procedures. Additionally, we introduce a class of *interesting rules* that take into account costs associated with various procedures prescribed based on available diagnoses for a patient. We provide examples of such interesting rules by mining the Marketscan database.

### 5.1 Association Rules and Medical Databases

In this chapter, we will summarize our results on using association rules on a medical database. Section 1 through 4 will give an overview of association

rules using frequent and closed-frequent sets respectively. Section 5 will offer a new way to look at families of association rules that, we believe, will lead to more interesting collection of rules involving diagnoses and procedures.

### 5.1.1 Association Rules: The Apriori Approach

Association Rules were first introduced in relation to the Market Basket Analysis problem. Let  $\mathcal{I}$  denote the collection of all items sold at a supermarket. Suppose the supermarket has stored records of items purchased by customers. A typical way to store this is in a "horizontal" list format where row  $i$  in the list represents a transaction which has an identification number  $y$  and the set of all identification numbers is denoted by  $\mathcal{T}$ . The transactions themselves are subsets of  $\mathcal{I}$ . Thus, transaction  $T_y$  (transaction with identification tag  $y$ ) is a subset of  $\mathcal{I}$  with elements  $x_{yi} \in \mathcal{I}$  for  $i = 1, \dots, n_y$ . The items  $x_{yi}$  are the items purchased during transaction  $y$  and  $n_y$  is the total number of items purchased in transaction  $y$ .

Our goal is to find rules of the form  $A \xrightarrow{q} B$ . The rule  $A \xrightarrow{q} B$  is defined if  $A, B$  are disjoint subsets of  $\mathcal{I}$  and there exists  $C \subseteq \mathcal{T}$  such that  $A \cup B \subseteq T_y$  for some  $y \in \mathcal{T}$ . That is, the elements of  $A$  and  $B$  are common to at least one transaction  $T_y$ .

Association rules typically have two attributes, the *support* and *confidence*. The support is defined as the number of transactions that contain the items in  $A \cup B$ . The confidence  $q$  is the probability that a transaction will contain  $B$  given that it contains  $A$ . It is computed by dividing the support of  $A \cup B$  by the support of  $A$ .

Let's look at the total number of possible rules. For any subset of  $\mathcal{I}$  with  $k$  elements, there are  $2^k - 2$  possible rules. This comes from selecting all possible left-hand-sides of the rule except the null set and the full set. Suppose that there are  $I$  unique elements in  $\mathcal{I}$ . The total possible number of rules will be

$$\sum_{k=2}^I \binom{I}{k} (2^k - 2)$$

For only 20 unique items, this is already just shy of 3.5 billion rules. This is far too many rules to mine from even a moderate database. We must apply some constraints to limit the size of the class of rules we wish to mine or discover. Apriori algorithm and other similar algorithms force the support

of a rule to be above a preset *minsup* (which stands for *minimum support*) and the confidence to be above *minconf* (*minimum confidence*). Itemsets with a support greater than *minsup* are called *frequent* itemsets.

The Apriori algorithm was the first major breakthrough in mining Association rules. The search is divided into two parts. First, it uses a breadth-first approach to search the space of itemsets for frequent itemsets. Next association rules are created for these frequent itemsets. The first portion dominates the computation time, and there have been many algorithms suggested to speed up this phase.

To see how the Apriori algorithm works, notice that any subset of a frequent itemset must also be frequent. Taking advantage of this fact, we start with small itemsets and eliminate infrequent sets. No subset containing these eliminated sets need be searched, thus we limit our search space.

Table 5.1: Example Transactional Database

tid	items
1	a, c
2	a, b
3	a, b, c
4	a, b
5	a, b, c
6	a, c
7	c
8	c, d

Consider the database in Table 5.1. There are 8 transaction and four items. Suppose that we set *minsup* = 4. We start with the individual items *a, b, c, d* and scan the database to count their frequency. The only item that is not frequent (happens less than 4 times) is *d*, which we eliminate. This leaves *a, b, c*. We now form all pairs of frequent items, namely *ab, ac, bc* and scan the database to count their frequency. The itemset *bc* is not frequent and is removed leaving *ab, ac*. Again, all combinations of frequent itemsets from the last set are generated and checked. In this case the procedure leads to *abc* which is infrequent as *bc* is infrequent. There is nothing left to check and the algorithm terminates.

Many variations on the Apriori algorithm exist. These variations focus on clever ways of storing and checking the frequency of the itemsets. These include growing trees and using hash tables.

There are a few problems with Apriori. First, we may miss low support items that are of interest. For example, as stated on [2, page 444], customers that purchase vodka may also purchase caviar. As these purchases happen infrequently, they will most likely be missed by an Apriori-like algorithm.

Furthermore, high confidence does not translate to a dependence relationship. For example, suppose that  $A$  and  $B$  are independent and frequent, and  $B$  occurs in 80% of the transactions. The confidence of the rule  $A \xrightarrow{0.8} B$ ,  $P(B|A) = P(B)$ , is high. This may be thought to indicate a predictive association between  $A$  and  $B$ , which is incorrect. Other metrics have been suggested to solve this problem. Each of these metrics attempt to measure the departure from independence in one way or another.

In the next section, we will look at using Apriori-like algorithms to generate association rules in medical databases.

### 5.1.2 Apriori-like Algorithms and Medical Databases

Now suppose that the database has a row for each insurance claim. Let  $\mathcal{D}$  denote the set of possible diagnoses and  $\mathcal{P}$  the set of possible procedures that could be prescribed. For each claim, say claim number  $y$  in the database, we have a record of the patient's diagnoses  $d_{y1}, \dots, d_{yn_y} \in \mathcal{D}$  and the procedures  $p_1, \dots, p_{ym_y} \in \mathcal{P}$  that were performed. Here  $n_y$  is the total number of diagnoses associated with claim  $y$  and  $m_y$  is the total number of procedures associated with claim  $y$ . The first use of association rules on this database might be to find rules of the form  $A \xrightarrow{p} B$  involving just the diagnoses and rules involving just the procedures. This would identify diagnoses and procedures, respectively, that occur frequently together.

Now consider rules of the form  $D \xrightarrow{p} P$  where  $D \subseteq \mathcal{D}$  and  $P \subseteq \mathcal{P}$ . These are natural rules to consider, as they represent a frequent and high probability decision that patients with diagnoses  $D$  would receive procedure  $P$ . For example, suppose we have the rule  $\{d_1\} \xrightarrow{.7} p_1$ , and this rule is frequent. What have we learned? There is a high chance that a patient with diagnosis  $d_1$  will be prescribed procedure  $p_1$ .

We believe there is more to the story than one rule can tell. If we instead knew that  $\{d_1, d_2\} \xrightarrow{.3} p_2$  and  $\{d_1 \cap d_2^C\} \xrightarrow{.7} p_1$ , we are in a much better

position. We now know that the common procedure for  $d_1$  is  $p_1$  except in the presence of  $d_2$ , which leads to procedure  $p_2$ . See section 5.1.5 for more discussion on this important point.

Zoddi et al [3] also considered rule extraction from medical databases, but their formulation of the problem leads to very different results than ours would. They reformulated the database so that a row represented a patient and contained all the diagnoses and procedures for that patient over the course of a year. Then the Apriori algorithm was used to find association rules of the form  $D \xrightarrow{R} P$ . Notice that this approach is less than satisfactory. The rules generated point to an association with diagnoses and procedures but the procedures may not directly correspond to the diagnoses since the data are pooled over an entire year. Our formulation overcomes this deficiency in the sense that rules are mined from individual claims. The procedures and diagnoses are directly related, that is the prescribed procedures are a direct result of the diagnoses and a doctor's decision. Thus the *doctor* provides the link between diagnoses and procedures. As mentioned earlier, in the formulation given in [3], the diagnoses and procedures may not have come from the same claim. They may not be the result of a doctor's decision, but they are related through the *patient*.

### 5.1.3 Mining Rules Using Closed Frequent Itemsets

In this section, we review an appealing way, first introduced by Zaki and Hsiao [4], using which one can shrink the search space and eliminate redundant rules. Suppose that the items  $a, b, c \in \mathcal{I}$  always occur together and  $D \subseteq \mathcal{I}$  not containing  $a, b, c$ . The following rules would form a redundant set of rules in the sense that they are guaranteed to have the same support and confidence.

$$\begin{aligned} \{a\} &\xrightarrow{q} D \\ \{a, b\} &\xrightarrow{q} D \\ \{a, c\} &\xrightarrow{q} D \\ \{b, c\} &\xrightarrow{q} D \\ \{a, b, c\} &\xrightarrow{q} D \end{aligned}$$

In this example, the set  $\{a, b, c\}$  is a *closed* set. We will now give some results from Formal Concept Analysis that will provide a definition for a

closed set. We follow closely the notation and development given in [4]. The development starts with the definition of a partially ordered set and a lattice.

A *partial order* on a set  $S$  is a binary relation  $\leq$  with the following properties:

Reflexive :  $x \leq x, \forall x \in S$ .

Anti-Symmetric:  $x \leq y$  and  $y \leq x, \Rightarrow x = y, \forall x, y \in S$ .

Transitive :  $x \leq y$  and  $y \leq z, \Rightarrow x \leq z, \forall x, y, z \in S$ .

Let  $(S; \leq)$  be a partially ordered set and  $A \subseteq S$ . We define the following:

Upper Bound of A:  $u \in S$  such that  $a \leq u$  for all  $a \in A$ .

Lower Bound of A:  $l \in S$  such that  $l \leq a$  for all  $a \in A$ .

Meet of A: The greatest lower bound, denoted  $\bigwedge A$ .

Join of A: The least upper bound, denoted  $\bigvee A$ .

Figure 5.1: The Complete Lattice of Itemsets

A partially ordered set  $(L, \leq)$  is a *lattice*, if for any two elements  $x$  and  $y$  in  $L$ , the join  $x \vee y$  and meet  $x \wedge y$  always exist.  $L$  is a complete lattice if  $\bigwedge A$  and  $\bigvee A$  exist for all  $A \in L$ . The partially ordered set  $(\mathcal{P}(S), \subseteq)$  is a *complete lattice*. Here  $\mathcal{P}$  denotes the set of subsets of  $P$  (power set). The complete lattice for the itemsets in the database in Table 5.1 is shown in Figure 5.1. Sets that are higher in the lattice are "larger" than the sets below them according to the ordering defined by  $\subseteq$ . The meet for two items can be found by following lines down to a common vertex. The join can be found by following lines up to a common vertex. The sets  $AB$  and  $BCD$  meet at  $B$  and join at  $ABCD$ , their intersections and unions respectively.

Figure 5.2: The Meet Semi-Lattice of Frequent Itemsets

If we are only guaranteed that joins will exist, then  $L$  is called a *join semi-lattice*. If only meets are guaranteed to exist, then  $L$  is called a *meet semi-lattice*. Figure 5.2 shows all frequent itemsets, which form a meet semi-lattice. Notice that for any two items, there is a common vertex below the pair, but not necessarily a vertex above. For example,  $B$  and  $C$  meet at  $\emptyset$  but their join,  $BC$  is not frequent.

Now let's consider the connection between the complete lattice of itemsets  $(\mathcal{I}, \subseteq)$  and the complete lattice of tidsets (transaction identification number sets)  $(\mathcal{T}, \subseteq)$ . Let  $X \subseteq \mathcal{I}$  be an itemset and  $Y \subseteq \mathcal{T}$  be a tidset.

First, we define functions that map from one lattice to the other.

$$\begin{aligned} t : \mathcal{P}(\mathcal{I}) &\rightarrow \mathcal{P}(\mathcal{T}); & t(X) &= \{y \in \mathcal{T} \mid \forall x \in X; x\delta y\} \\ i : \mathcal{P}(\mathcal{T}) &\rightarrow \mathcal{P}(\mathcal{I}); & i(Y) &= \{x \in \mathcal{I} \mid \forall y \in Y; x\delta y\} \end{aligned}$$

The notation  $x\delta y$  means that  $x$  and  $y$  are “related”, that is, item  $x$  occurs in transaction  $y$  ( $x \in T_y$ ). For example, in the database in Table 5.1, we can write  $a\delta 1$  as item  $a$  occurs in transaction 1.

The function  $i$  applied to  $Y$  gives all the items that occur together in every transaction in the tidset  $Y$  and the function  $t$  gives all the transactions that contain every item in the itemset  $X$ . It is obvious that  $t(X) = \bigcap_{x \in X} t(x)$ .

As an illustration, consider the database in Table 5.1. One can see that  $t(ab) = 2345$  and  $i(12) = a$ .

Now we wish to show that these functions form a *Galois connection*. A Galois connection must satisfy the following conditions:

- (i)  $X_1 \subseteq X_2 \Rightarrow t(X_1) \supseteq t(X_2)$ .
- (ii)  $Y_1 \subseteq Y_2 \Rightarrow i(Y_1) \supseteq i(Y_2)$ .
- (iii)  $X \subseteq i(t(X))$  and  $Y \subseteq t(i(Y))$ .

The following proposition is easily proved.

**Proposition 1** *The functions  $t : \mathcal{P}(\mathcal{I}) \rightarrow \mathcal{P}(\mathcal{T})$  and  $i : \mathcal{P}(\mathcal{T}) \rightarrow \mathcal{P}(\mathcal{I})$  form a Galois connection between  $(\mathcal{I}, \subseteq)$  and  $(\mathcal{T}, \subseteq)$ .*

The utility of a Galois connection is that it gives us an easy way to form two closure operators. A function  $c : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$  is a *closure operator* on  $S$  if, for all  $X, Y \subseteq S$ ,  $c$  satisfies the following properties:

- (i) Extension:  $X \subseteq c(X)$ .
- (ii) Monotonicity: if  $X \subseteq Y$ , then  $c(X) \subseteq c(Y)$ .
- (iii) Idempotency:  $c(c(X)) = c(X)$ .

We call a subset  $X$  of  $S$  *closed* if  $c(X) = X$ .

**Proposition 2** *Let  $X \subseteq \mathcal{I}$  and  $Y \subseteq \mathcal{T}$ . Let  $c_{it}(X) = i(t(X))$  and  $c_{ti}(Y) = t(i(Y))$ . Then  $c_{it} : \mathcal{P}(\mathcal{I}) \rightarrow \mathcal{P}(\mathcal{I})$  and  $c_{ti} : \mathcal{P}(\mathcal{T}) \rightarrow \mathcal{P}(\mathcal{T})$  are both closure operators on itemsets and tidsets respectively.*

We define a closed itemset as an itemset  $X$  such that  $X = c_{it}(X)$ . For any closed itemset  $X$ , there exists a closed tidset given by  $Y$ , with the property that  $Y = t(X)$  and  $X = i(Y)$ . Such a pair  $(X, Y)$  is called a *concept*. It turns out that the concepts form a lattice with meets and joins given in the Zaki and Hsiao [4]. This leads us to the two main results:

**Theorem 1** *For any itemset  $X$ , its support is equal to the support of its closure, i.e.,  $|t(X)| = |t(c_{it}(X))|$ .*

**Theorem 2** *The rule  $X_1 \xrightarrow{p} X_2$  is equivalent to the rule  $c_{it}(X_1) \xrightarrow{q} c_{it}(X_2)$  in the sense the rules share the same support and  $q = p$ .*

### 5.1.4 Closed Frequent Itemsets and Medical Databases

The big advantage of searching for closed itemsets is that one is searching over a smaller lattice, and hence can potentially get a much faster rule-mining algorithm. Furthermore, the list of association rules is smaller and does not contain redundant rules. We will see that there is even more benefit to looking for association rules between closed sets in a medical database.

Suppose that our itemset  $\mathcal{I}$  is partitioned into diagnoses  $\mathcal{D}$  and procedures  $\mathcal{P}$ . We wish to find rules of the form  $D \xrightarrow{p} P$ , where  $D \subseteq \mathcal{D}$  and  $P \subseteq \mathcal{P}$ . When only looking for this type of rule, we can further reduce the search space in such a problem.

We will define two new functions that map from the transaction sets to each partition.

$$\begin{aligned} i_{\mathcal{D}} : \mathcal{P}(\mathcal{I}) &\rightarrow \mathcal{P}(\mathcal{D}); & i_{\mathcal{D}}(Y) &= \{x \in \mathcal{D} \mid \forall y \in Y; x\delta y\} \\ i_{\mathcal{P}} : \mathcal{P}(\mathcal{I}) &\rightarrow \mathcal{P}(\mathcal{P}); & i_{\mathcal{P}}(Y) &= \{x \in \mathcal{P} \mid \forall y \in Y; x\delta y\} \end{aligned}$$

If we ignore the items in  $\mathcal{P}$ , the results of the previous subsection imply that  $i_{\mathcal{D}}$  and  $t$  form a Galois connection between the lattice  $(\mathcal{P}(\mathcal{D}), \subseteq)$  and  $(\mathcal{P}(\mathcal{I}), \subseteq)$ . Therefore, the functions  $c_{i_{\mathcal{D}}t}(X) = i_{\mathcal{D}}(t(X))$  and  $c_{ti_{\mathcal{D}}}(Y) = t(i_{\mathcal{D}}(Y))$  are closure operators. Similarly, the functions  $c_{i_{\mathcal{P}}t}(X) = i_{\mathcal{P}}(t(X))$  and  $c_{ti_{\mathcal{P}}}(Y) = t(i_{\mathcal{P}}(Y))$  are closure operators. Therefore, methods described in Zaki and Hsiao [4] can be used to prove the following Corollary to Theorem 2.

**Corollary** *The rule  $X_1 \xrightarrow{p} X_2$  is equivalent to the rule  $c_{i_{\mathcal{D}}t}(X_1) \xrightarrow{q} c_{i_{\mathcal{P}}t}(X_2)$  in the sense that the rules share the same support and  $q = p$ .*

This has the possibility of reducing the search by a significant margin. It has been stated that the time to search of itemsets is proportional to  $2^M$  where  $M$  is the length of the longest frequent itemset. If the length of the longest frequent diagnoses and procedure is much smaller than the length of the longest combined frequent itemset, we should see a major boost to the speed of the algorithm. In the MarketScan database the length of the longest diagnoses and procedure has size 15 for combined in-patient claims. On the other hand, the longest itemset has size 30.

### 5.1.5 Association Rules and Medical Databases: A New Objective Related to Cost

#### Introduction

Let's reconsider this problem. If we are interested in finding connections between diagnoses and procedures then we should be interested in rules of the form  $D \xrightarrow{R} P$ , where  $D \subseteq \mathcal{D}$  and  $P \subseteq \mathcal{P}$ . A few points need to be made. Use of rules where  $D \subseteq \mathcal{D}$  should, for the most part, give us a general idea of which diagnoses are related to a procedure or set of procedures. They will not give us specific information on the decisions that doctors make on a patient by patient basis.

On the other hand, suppose that  $D$  was the entire set of diagnoses for a group of patients, but we observed different resulting procedures. Some natural questions arise: Why did the same diagnoses result in different procedures? Did any of these procedures produce better results than others? Are there two procedures that produced similar results, but one costs less? This could be the starting place for some very interesting and fruitful investigation.

The last question in the above paragraph leads to another question. The cost of health-care is of major concern. Association rules in their usual form do not allow for constraints involving individual, or in particular, total cost of either side of the rule. In a medical database, we have the cost information available. It would be useful to produce a way of restricting rules based on their individual or total cost as well as their frequency.

#### Problem Formulation - Interesting Rules

We suggest the following reformulation of the definition of interesting rules:

**Problem** We wish to find a family of rules  $D \xrightarrow{q_1} P_1, D \xrightarrow{q_2} P_2, \dots, D \xrightarrow{q_{n_D}} P_{n_D}$ , called a *diagnoses family of rules* where:

1. For each  $i$ ,  $D$  and  $P_i$  are the entire set of diagnoses and procedures for a group of patients.
2. We redefine the *support* of the rule  $D \xrightarrow{q_i} P_i$  as the number of patients that have  $D$  and  $P_i$  as their entire set of diagnoses and procedures.
3. We redefine the *confidence*,  $q_i$ , of the rule  $D \xrightarrow{q_i} P_i$  as the fraction of patients with diagnoses  $D$  that received procedure  $P_i$ .
4. We will compute the average cost for patients with diagnoses  $D$  and procedures  $P_i$ .
5. The total cost of diagnoses  $D$  is the sum of the costs for all patients with diagnoses  $D$ .

There are a number of ways that we can explore interesting diagnosis families. First, if there is a particular procedure of interest; we can find the diagnosis families that involve it by taking a “round-trip”. First, we find all the diagnoses that lead to the procedure of interest. Then, we construct the diagnosis family for each unique diagnosis. This is done by collecting all the procedures that resulted from the diagnosis.

An alternate way to find interesting diagnosis families is to define an “interesting” criteria. For example, a diagnoses family of rules could be deemed interesting if

1. One of the rules in the family has a high total cost. In our exploratory work, the top 1000 total cost diagnoses are considered to have high total cost.
2. There is a high amount of variability in the average cost of rules in the family. To find families with high variability, the top 1000 total cost families are sorted by their average cost variability.

As a result, we will find collections of diagnoses and procedures that greatly contribute to high health care costs, but for one reason or another lead to a different procedures with different cost structures. It should be very interesting to study the reasons for the variability in cost of procedures.

## Implementation

Algorithmically, this problem is not as difficult as the original formulation of association rule mining. We are not searching the space of subsets, but instead we are only looking at sets that occur in totality. This a much smaller set of items!

Even on a database the size of the Market Scan database, the search for individual rules can be done with SAS SQL in a relatively short amount of time.

## First Example Family

In this example, we considered the 2003 Market Scan Database information on inpatient claims. The database consolidates all the claims for one visit to the hospital onto one line in a table. This line includes up to 15 diagnoses and 15 procedures, as well as some demographic information.

We selected the top 1000 diagnoses in terms of maximum total cost and sorted these by the variability in the average cost of the family of procedures resulting from each diagnosis. The example that we show here is ranked 38th in variability of average cost of procedures. The two diagnoses are *Intermediate Coronary Syndrome* and *Chest Pain*. There were 351 different procedures that resulted from this diagnosis.

A good question is: Can the variability be explained by the demographic information that we have. Let's attempt to answer this question. First we will look at the average cost split by each demographic category. The standard error for each average is also included. This can be used when determining if two costs are "different". The total number of cases and the standard deviation have been provided. We can use the standard deviation to look for specific classes where there is a highly variable cost. All the numbers have been rounded to three significant digits.

The family included two cases that had extreme relative costs. These two cases have a cost that is an order of magnitude larger than all other cases. It is unclear why these two cases cost so much more, but it seems unlikely that it is a data entry error. The averages calculated after excluding these two patients is given in parenthesis.

Some obvious things to note: Table 5.2 indicates that this diagnosis is more expensive for men than women. Excluding the two extremes, Table 5.3 shows that the cost tends to rise as the patient age increases. Not surprisingly,

we see in Table 5.4 that the cost also increases with the length of stay. Once the two extremes are excluded, none of the other tables show much of a difference in categories.

It seems strange that out of 359 patients with this diagnoses, there were 351 different sets of procedures performed. It might be the case that a number of procedure sets are only superficially different, but this would require more specific topic knowledge than is currently available. This is one place where medical experts can enhance the data-mining exercise by providing suitable metrics for similarity between diagnoses and/or procedures.

Table 5.2: Mean Cost by Gender for First Example Family

Gender	Mean Cost	SE	Number	Standard Deviation
Male	15700(13500)	1750	237	27000
Female	9810	687	122	7590

Table 5.3: Mean Cost by Age Group for First Example Family

Age Group	Mean Cost	SE	Number	Standard Deviation
18-34	7240	.	1	.
35-44	19600(9380)	10300	24	50400
45-54	13800(11800)	2190	142	26000
55-64	13000	859	191	11900
65 and older	5270	.	1	.

### Second Example Family

The next example family was picked with the desire to find a particular type of family. The desire was to find a parsimonious example where the diagnosis had a large number of occurrences, but a relatively small number of procedures. As a side note, it was a little shocking how difficult it was to find this type of example. Typically, there were nearly as many different procedures as there were patients for a given set of diagnoses.

Table 5.4: Mean Cost by Length of Stay for First Example Family

Days	Mean Cost	SE	Number	Standard Deviation
1	9440	552	160	6980
2	15800(13400)	2630	118	28600
3	17800(12900)	5070	49	35500
4	18200	3510	27	18200
5+	37400	9160	5	22000

Table 5.5: Mean Cost by Industry for First Example Family

Industry	Mean Cost	SE	Number	Standard Deviation
Missing	13000	1870	53	13600
Manufact., Durable Goods	11700	7390	154	9180
Manufact., Nondurable Goods	14200	2810	25	14000
Transport., Commun., Util.	11600	2930	31	16300
Retail Trade	19200(12000)	5190	73	44300
Finance, Insurance, Real Estate	12600	2700	15	10500
Services	14600	3800	8	10800

This family ranked 212th in total variability among the top 1000 maximum total cost families. There are 7 cases of the diagnosis family, which involves *Digestive System Complications (ICD-9-CM code 278.01)*, *Respiratory Distress (ICD-9-CM code 786.09)*, and *Localized Adiposity (ICD-9-CM 997.4)*. It is interesting that these cases all involved the same doctor and patients with very few differences. One of the patients was charged a total of just over \$110,000, where as all the other patients were charged close to \$14,000. All of the procedures occurred between 02/03/2003 and 04/24/2003.

First thing to note, is that there appear to be a large number of mistaken entries in the total cost. These are usually indicated by a total cost that is an order of magnitude above the typical cost, but having a net cost that is of the same magnitude as the typical cost. This difference is attributed to a data entry error in which an extra decimal is added to the total cost. This is not the case in this example. Both the net and total cost are one order of

Table 5.6: Mean Cost by Employment Status for First Example Family

Employment Status	Mean Cost	SE	Number	Standard Deviation
Active Full Time	15200(12900)	1770	233	27000
Active Part Time or Seasonal	12700	10100	2	14300
Early Retiree	12000	1130	84	10400
Medicare Eligible Retiree	9010	2200	12	7600
Retiree (status unknown)	8130	2350	8	6650
COBRA Continuee	10400	2910	9	8720
Long Term Disability	10400	3170	3	5500
Surviving Spouse/Depend	3080	2120	3	3680
Other/Unknown	9570	3100	5	6920

magnitude larger for the high-cost patient than for the typical patient.

Let's look at similarities and differences between the patient with the high cost and the typical patients.

- All of the low-cost patients were treated in hospitals with the same zip-code (perhaps the same hospital). The high-cost patient received the procedure in a hospital with a different, but very similar, zip-code.
- All of the patients had similar zip-codes and thus from nearly the same place.
- All of the patients were discharged with the same status, *Discharged to home self-care*.
- The length of stay for the high cost patient was 2 days, which was 1 day shorter than the other patients.
- Five of the patients had a comprehensive insurance plan and the other two, including the high-cost patient, had a PPO plan.
- The age of the high-cost patient was 46. The ages of the rest of the patients were 31, 47, 53, 54, 54, 58.

Table 5.7: Mean Cost by Employee Classification for First Example Family

Employee Class.	Mean Cost	SE	Number	Standard Deviation
Salary Non-union	13900	1480	44	9810
Salary Union	12800	2790	7	7390
Salary Other	19100	7150	3	12400
Hourly Non-union	16900(12500)	3310	119	36100
Hourly Union	10600	860	110	8900
Hourly Other	14700	7500	6	18400
Non-union	14400	2250	41	14400
Union	40400	.	1	.
Unknown	9640	1320	31	7360

Table 5.8: Mean Cost by Region for First Example Family

Region	Mean Cost	SE	Number	Standard Deviation
Northeast	8900	1280	28	6770
North Central	12700	851	137	9960
South	15500(12400)	2370	168	30700
West	12600	3640	25	18200
Unknown	9720	.	1	.

- The high-cost patient and all but one of the low-cost patients were female and the spouse of the primary beneficiary. The other patient was male and the primary beneficiary.
- The primary beneficiaries for all of the low-cost patients had employee classifications of *Hourly Union*, where as the primary beneficiary for the high-cost patient was classified as *Non-Union*.
- The primary beneficiary for the high-cost patient had an industry classification of *Transportation, Communications, Utilities*. The primary beneficiary for the rest of the patients was classified as *Manufacturing, Durable Goods*.

It would be very interesting to explore this family more. It seems unlikely

Table 5.9: Mean Cost by Plan Type for First Example Family

Plan Type	Mean Cost	SE	Number	Standard Deviation
Comprehensive	10600	807	102	8150
EPO	11300	809	2	1150
HMO	14500	5440	22	25500
POS	12300	1010	56	7590
PPO	16400(13200)	2460	160	31100
POS with capitation	10500	2430	17	10000

Table 5.10: Mean Cost by Discharge Status for First Example Family

Discharge Status	Mean Cost	SE	Number	Standard Deviation
Missing	14800	2500	28	13200
Discharged to home self-care	14800(13000)	1460	285	24700
Transfer to short-term hospital	5710	620	37	3770
Transfer to ICF	9530	.	1	.
Transfer to other facility	8330	2240	4	4480
Discharged home under care	13700	.	1	.
Left against medical advice	1930	.	1	.
Not Yet discharged/Transferred	3250	1890	2	2670

that the difference in cost could be explained by the differences noted above. Perhaps the cost difference is a clerical error. It would definitely be worth the time of the employer or health insurance provider to find and investigate such a discrepancy further.

### 5.1.6 Further Research

There appears to be a potential to mine further knowledge from the diagnoses and procedure codes. One possible approach would be to have an expert assign a subjective "distance" between two procedures. If such a set of distances was available, a large number of learning methods would be

applicable.

# Chapter 6

## Summary

In this report we have documented results of our explorations regarding the applicability of datamining procedures to the marketscan database. It was shown that support vector classifiers can be successfully used in early identification of patients likely to be diagnosed with type-II diabetes at a later point in time. We also reported on the hardware and software needs for successfully implementing such datamining methodologies.

We also examined rule extraction methods. Notably, we introduced a class of interesting rules which extract patient/claim combinations that have associated with them high cost procedures whereas similar patient/claim combinations have lower cost procedures associated with them. The implementation of this rule extraction process is algorithmically much faster than standard apriori procedures. We also proved a corollary to some recently published results on closed rules which should help speed up algorithms for finding special classes of interesting rules.

Finally, it has been noted in various places that additional information from medical experts can be fruitfully used to construct similarity metrics among diagnoses and among procedures that can substantially enhance any datamining effort on medical databases.

# Bibliography

- [1] Schölkopf, B., Smola, A., *Learning with Kernels - Support Vector Machines, Regularization, Optimization and Beyond*, first edition, The MIT Press, London, 2002. ISBN 0-262-19475-9
- [2] Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning*, first ed., Springer-Verlag, New York, 2001, ISBN 0-387-95284-5
- [3] Doddi, S., Marathe, A., Ravi, S., and Torney, D., Discovery of association rules in medical data. *Med. Inform. Internet. Med.*, 26:25–33, 2001.
- [4] Zaki, M.J. and Hsiao, C., Charm: an efficient algorithm for closed association rule mining. *Tech. Report.*, RPI, 1999. <http://citeseer.ist.psu.edu/zaki99charm.html>