MEASURING MERCI: EXPLORING DATA MINING TECHNIQUES FOR EXAMINING SURGICAL OUTCOMES OF STROKE PATIENTS

By

Matthew Ronald McNabb

Approved:

_____	_____
Yu Cao Assistant Professor of Computer Science and Engineering (Director of Thesis)	Joseph Dumas UC Foundation Professor of Computer Science and Engineering (Committee Member)
_____	
Jack Thompson Professor of Computer Science (Committee Member)	
_____	_____
Herbert Burhenn Dean of the College of Arts and Sciences	A. Jerald Ainsworth Dean of the Graduate School

MEASURING MERCI: EXPLORING DATA MINING TECHNIQUES FOR EXAMINING SURGICAL OUTCOMES OF STROKE PATIENTS

By

Matthew Ronald McNabb

A Thesis
Submitted to the Faculty of the
University of Tennessee at Chattanooga
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2012

ABSTRACT

Mechanical Embolus Removal in Cerebral Ischemia (MERCI) has been supported by medical trials as an improved method of treating ischemic stroke past the safe window of time for administering clot-busting drugs, and was released for medical use in 2004. The importance of analyzing real-world data collected from MERCI clinical trials is key to providing insights on the effectiveness of MERCI. Most of the existing data analysis on MERCI results has thus far employed conventional statistical analysis techniques. To the best of the knowledge acquired in preliminary research, advanced data analytics and data mining techniques have not yet been systematically applied. To address the issue in this thesis, a comprehensive study on employing state of the art machine learning algorithms was conducted to generate prediction criteria for the outcome of MERCI patients. Specifically, the issue of how to choose the most significant attributes of a data set with limited instance examples was investigated. A few search algorithms to identify the significant attributes of the data set are proposed, followed by a performance analysis for each algorithm. Finally, this approach is applied to the real-world medical data provided by Southeast Regional Stroke Center at Erlanger Hospital of Chattanooga, Tennessee. Our experimental results have demonstrated that our proposed approach performs well.

DEDICATION

This work is dedicated to he who is my life and length of days, and to my lovely wife Megan, who endured many lonely nights to see the completion of this research.

ACKNOWLEDGEMENTS

A sincere thanks is due to the researchers at Erlanger Hospital for provision of data, to Dr. Yu Cao for his guidance, and to Dr. Joseph Dumas and Dr. Jack Thompson for their timely and cordial response to this thesis committee on short notice.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Mechanical Embolus Removal in Cerebral Ischemia (MERCI) is a relatively new medical procedure released by the Food and Drug Administration (FDA) in 2004, which widens the operational window for removing deadly blood clots from the brain to eight hours after the onset of acute ischemic stroke (AIS). Activase, a drug released in 1996 for dissolving the clots without mechanical operation, is highly effective but usable for only three hours after stroke onset. Although its use improves survivability by 30%, only a tiny fraction of patients (2-3%) actually qualify for dosage. This expansion of time offered by MERCI can be critical to the patient's outcome. [1]

While the usage of mechanical extraction is growing and test results are positive, it is not yet the most common of stroke-fighting tools. Receiving criticism at its initial approval in late 2004 and early 2005 due to alleged weaknesses in the FDA test requirements [2], MERCI has since been found to be "cost effective" by controlled medical experiments and practice results, when compared to other extraction methods.[3]

Erlanger Hospital of Chattanooga, Tennessee has produced a study strictly confined to stroke patients treated by MERCI, including a generous collection of detail with regards to procedure analysis, including patient diagnosis coming in the door, surgical procedure data, patient status after the procedure, and 90-day follow-ups. [4]

By introducing data mining techniques to this study, it is the hope of this project to advance surgical outcome prediction machine learning with the ultimate goal of predicting outcomes at incremental points in treatment. To this end, techniques will be used which produce training results that can be readily compared to conventional statistical analysis.

Much larger collections of data exist in state or national stroke registries, but these sources were found to be difficult to access for the purposes of this project. It is also is not guaranteed that these registries would contain specific details regarding MERCI, or easily discernable qualifications for patients who would be eligible.[5] The smaller collection of data provided by Erlanger represents a more specific study of patients in the Chattanooga area, as opposed to a general collection of stroke patients of various types over a large geographical area. Furthermore, working with a smaller data set with many attributes per instance is an issue that we have not seen addressed as frequently in our search for other data mining studies, and a new study on this subject might open another area of discourse in Information Science on solving some real problems. [6,7]

CHAPTER II

DATA

Erlanger's study consists of a total of 115 patients meeting the following criteria:

- Each patient was diagnosed with acute ischemic stroke
- Arrival for treatment was between 3 and 8 hours after stroke onset.
- Each patient was over 18 years of age.
- Each patient suffered from hypo-density in less than 1/3 middle cerebral arteries.

Because this data set is exclusively made up of MERCI patients, our study is specifically limited to gauging factors for patient health and mortality given MERCI treatment. Therefore, without a strict control group our goal becomes not that of measuring the effectiveness of MERCI itself in relation to other treatments or little to no treatment at all, but rather a search for relations among more specific factors within the MERCI process itself.

This data set had already been thoroughly analyzed with conventional statistics prior to our having received it. [4] This provides some gauge with which to measure the progress of machine learning, and to recognize how reasonable new discoveries are. Our goal is specifically to create a successful automated prediction system, which predicts the outcomes associated with subsequent input data of a similar sort and is

easily discernable by analysts both inside and out of the Information Science discipline to determine the most significant prediction sources.

Detail Size

The detail size of this sample set provides for a variety of choices from which to draw statistical trends. These details can be categorized a number of ways, but for machine learning the following categories split the data according to their logical functions in discovery:

- Static data – Information that does not involve any choice in the medical process. Examples:
    - Personal information: age, gender, etc.
    - Diagnosis information
    - Location of clot
- Non-static data – information that could be affected by decisions in treatment. Examples:
    - Procedural information
    - Device usage
    - Onset to puncture – time between stroke onset to insertion of device
    - Procedure duration
- "Negligible" or repetitive institutional data – Some details are simply listings of facilities, patient numbers, etc. It could be argued that not all of these are irrelevant, but we have discarded them for this study, either by lack of relevance, or due to repetition in values.

- Outcomes and post-procedure data – There are a number of outcome components, which reflect the condition of the patient both immediately after the procedure and after a 90-day follow up. Life vs. death is the most basic outcome to predict, but other measurements can provide a more accurate picture of a living patient's condition. Also, some outcomes might serve as useful inputs for future conditions of the patient—some machine training can be done using measures of the patients' condition immediately after the procedure to predict conditions after a 90-day follow-up with increased accuracy. To fairly measure the machine learning algorithm's success, however, we emphasize its performance with most post-procedural data removed, particularly those with a direct and obvious correlation to the status of the 90-day follow-up.

Ultimately, 40 useful instance attributes were identified for input, eliminating most attributes occurring later than immediately after the procedure. Of the attributes identified as possible outcome gauges, two were chosen for class attributes: 90-day mortality, a check for patient survivability 90 days after the surgery, and 90-day MRS (Modified Rankin Scale), a more precise measurement of the patients' health in terms disability. **Table 1** briefly describes each MRS score.

Table 1 [8]

MRS Ratings

MRS Rating	Description
0	No symptoms found.
1	Patient suffers from some symptoms, but is not significantly disabled.
2	Some slight disabilities. Patient is capable of caring for his/herself.
3	Disabilities are moderate. Patient requires assistance but is still able to walk on his/her own.
4	Patient suffers from moderate to severe disabilities, and requires assistance to walk.
5	Patient is bedridden and in need of continuous care.
6	Patient is deceased.

Sample set size

Because three patients had no result attributes recorded, they were eliminated from the data set, leaving 112 instances for training and testing. Many modern data mining efforts use massive amounts of data to sharpen weights and render detail size maxima negligible for their algorithms, so this number could be said to be small by relation, although it represents a great deal of gathering work and is over three times the size of the typical medical trials examined in preliminary research.[1] [2] [3] This set is still large enough to establish trends, but the marginal return of all 40 attributes may be limited due to insufficient instances to train them.

Balance

In terms of 90-day mortality, the data set is well balanced, with 53 deaths and 59 survivors. The 90-day MRS benchmark, however, is unbalanced by nature. Figure 1 and **Figure 2** show the outcome distributions for 90-day mortality and 90-day MRS, respectively.

As a set of numbers, the MRS outcome distribution is heavily unbalanced to a rating of 6, and drives most classifiers to overestimation. When an understanding of its semantics is applied, however, the skewed weight towards 6 is simply a result of 6 representing death. Life, then, is divided among ratings 0-5.

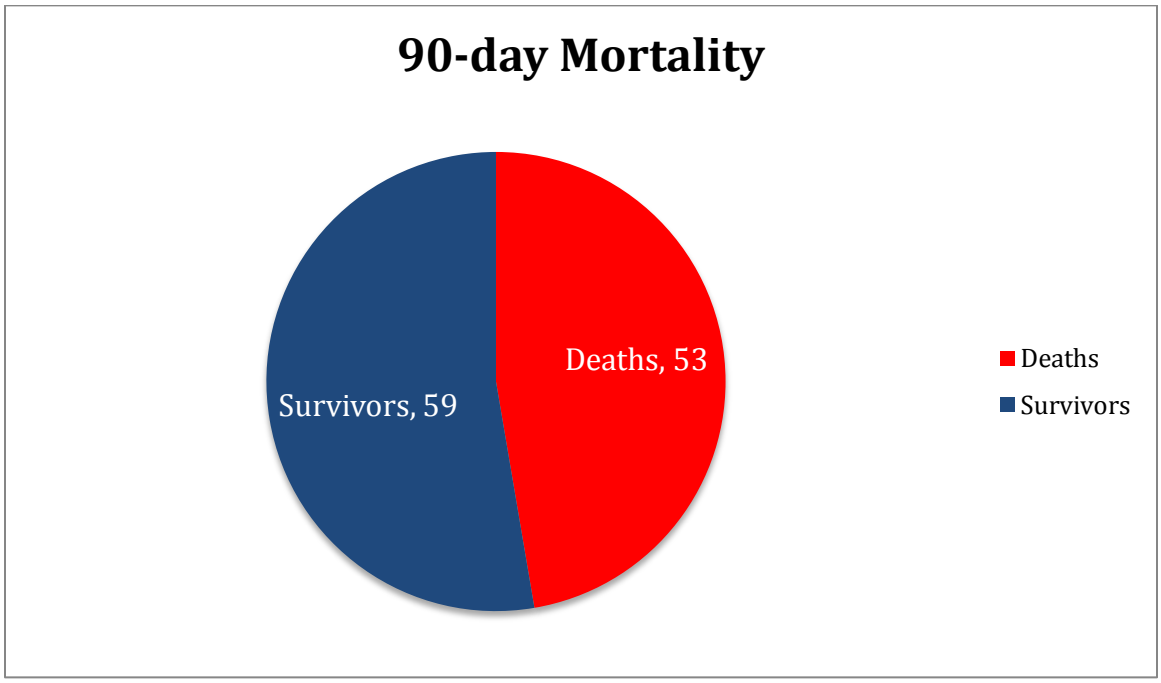

Figure 1

90-day Mortality Outcome Distribution

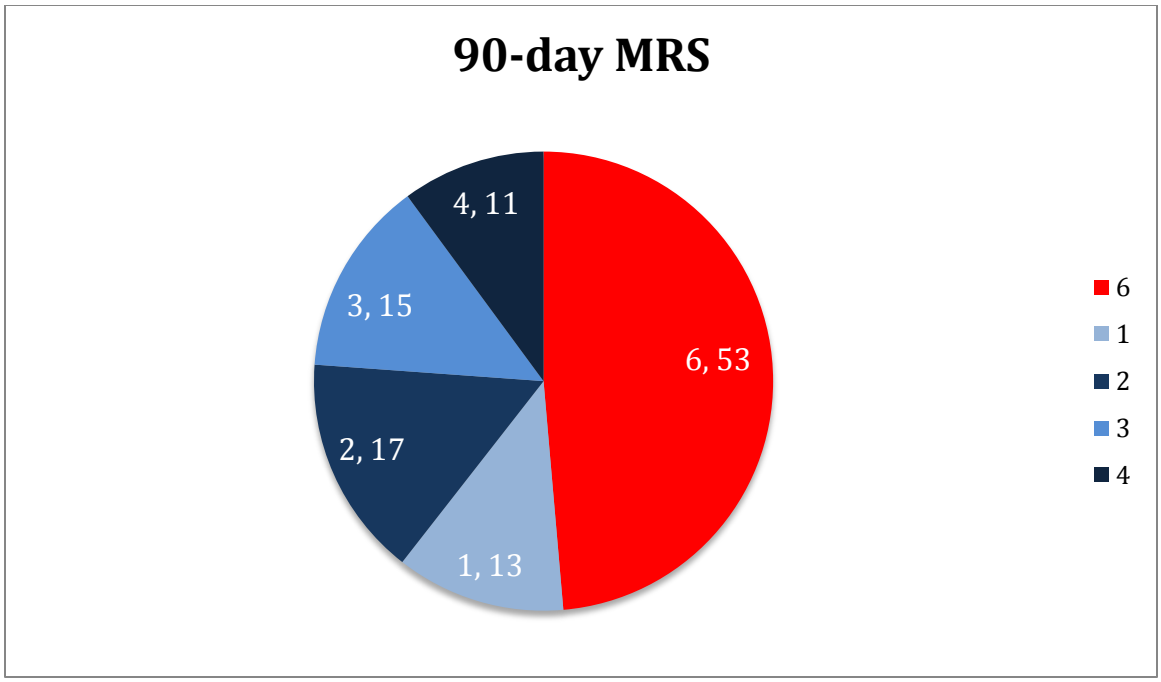

Figure 2

90-day MRS outcome distribution

It would be beneficial for data mining if it were possible to be able to divide death into similar sub-categories to represent varying levels of how close each patient's demise was to recoverability. Perhaps one patient was completely un-savable, regardless of any intervention at all, but another might have been saved with different choices made. This falls into the realm of speculation, however, and death is, in reality, a condition without levels – dead is dead, and one cannot become better or worse after dying.

Nonetheless, applying the fact that MRS = 6 refers to death, this apparently unbalanced data set rather becomes a two-layered classification problem with sub-problems of a much more balanced nature. Without rating 6, while we have no instances of 5 or 0, the distribution still has a somewhat reasonable bell-curve shape.

If the data set is first analyzed in terms of life and death, a second set of weights could be trained for MRS ratings 0-5, using only the 59 survivors. A new instance, then, would first be evaluated for mortality. With the prediction of death, its predicted MRS rating would automatically be set to 6. If predicted to live, the patient's condition would then be evaluated by the second set of weights.

CHAPTER III

PROPOSED APPROACH

Choosing Classifiers

Ultimately, a variety of machine learning algorithms could be used for an exhaustive study of this data, since automated test processes can run day and night for further analysis. For this stage in examining the data, however, we were most interested in a more limited set of criteria:

"White Box" Classification

Ideally, we would like the end results to be easily understandable by analysts outside of the Information Science discipline. Multi-layered approaches, such as neural networks, use a system of derived weights which may train well, but do not present a clear and easily traceable line of significance back to the original attributes. Single-layered weights can convey the significance of each attribute more plainly, and are thus encouraged. This exclusion of machine learning complexity, of course, may eliminate algorithms that might perform better, but similar complexities can be added with more easily traceable layering, as will be explained later in Attribute Layering.

Initial Naïve Analysis

Initially, we would like our algorithm to make no automatic assertions that any attributes are related to one another, allowing weights to be derived as independently as

possible. In the end, however, this may not be a proper constraint of the optimal solution, since we know already that major groupings of attributes exist: patient vitals, procedure data, and post-procedure data. Any predetermined relationships that seem appropriate can be inserted afterwards with Attribute Layering.

Flexibility with Missing Data

Technology that handles missing data values is a must, as the data set is small and instance loss must be kept to a minimum.

At the beginning of our study, we began manual experimentation with several datamining approaches, using the University of Waikato's Weka GUI [7], to find a suitable candidate for programmatic refinement. Table 2 shows a brief synopsis of the three algorithms which performed best, along with a Feed-Forward Neural Network, which might be useful in the future for performance comparisons, but does not satisfy our "white box" constraint as well as the others.

Logistic Regression proved to be the best performing approach meeting the above criteria in the experimental stage. For this reason, coding for the remainder of our proposed approach focused mainly on Logistic Regression as the training algorithm. More success in the manual experiment stage does not, however, indicate that Logistic Regression will be the best choice once all other parts of our solution are implemented, so a return to other algorithms is far from out of question in future work.

Table 2 [9]

Considered Machine Learning Algorithms

Algorithm	**Description**	**Pros**	**Cons**
Naïve Bayes	Derives rules for each attribute based on independent probabilities	Known for good performance with small data sets	Best with Boolean input.
Linear Regression	Models data with linear functions	Handles scalar values well	Not conducive for nominal attribute prediction
Logistic Regression	Models probability of Boolean output via a Logistic function $\frac{e^z}{e^z+1}$ where z is defined as a sum of weights combined with attribute values	Increased improvement as more instances are added	Overestimates weights with disproportioned attributes/instances ratio
Feed-Forward Neural Network	Layered Logistic Regression models with the output of each attribute node tied to input of intermediate nodes, predicting final outputs	Built-in automatic layering complexity	Relationship of prediction to original attributes can be clouded

Searching for Significant Attributes

Although Logistic Regression proved to be the best of the white-box methods evaluated in initial experimentation, it suffers from a tendency to overestimate weights in data sets where the number of detail variables is relatively large with respect to the number of instance samples provided.[9] This tendency is in the nature of variable

estimation, as particularly demonstrated in Gaussian Elimination solutions to algebraic systems by “free variables.” [10]

Unlike the equations of a linear system in a mathematics textbook, a data set is an abridged representative of its universe – if we do not have enough samples to pin down the significance of each attribute provided, the significance of every attribute can be skewed on the whole. [9] During initial experimentation, it was found that Logistic Regression performed better after some attributes were removed from the data set. Therefore, a means of automatically choosing a combination of attributes which best affects prediction performance is needed. Four methods of doing so were examined, with varying reliance on estimated weights:

Depth First Search (DFS)

A DFS approach was employed to exhaustively search every possible attribute combination for the most effective. The complexity of this algorithm is a sum of combinations, with 40 attributes to choose from. While this may ultimately find the best combination given enough time, the complexity of the search skyrockets as more attributes are added. [11]

$$Complexity_{DFS} = \sum_{r=1}^{n} C(n,r) = \sum_{r=1}^{40} C(40,r) = \sum_{r=1}^{40} \frac{40!}{r!\,(40-r)!} = 1.10 \times 10^{12}$$

Some technological approaches could be used to make DFS faster, such as multiprocessing or more robust and/or dedicated machines, but these innovations would be unlikely to shrink the completion of DFS to a reasonable runtime. If it were possible

to accomplish 1,000 training iterations per second, DFS for this attribute set would still require nearly 35 years to complete

Naïve Smart Search

To improve on the runtime of DFS, the simplest employed approach uses a saved-progress function related to dynamic programming. As with DFS, the number of attributes to be chosen is set up in a batch.

Given a pre-chosen attribute selection r, the algorithm begins with a simple selection of the first r attributes in the set, calculating their prediction success. For example, representing hypothetical attributes with numeric names for simplicity, with a given $r = 5$ and an initial prediction success rate of 60%:

Attributes	*Score*
12345	*60%*

Being the first evaluation made, this score is saved as the "best" evaluation. The search continues by examining the 6th attribute. An evaluation is made of r sets of attributes, with the new attribute replacing one of the attributes in the current "best" evaluation.

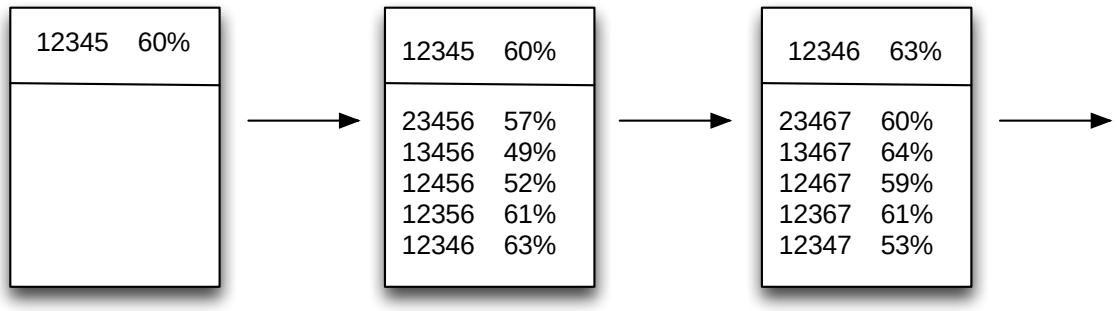

Figure 3

Naïve Smart Search (r=5)

After the accuracy of each new evaluation is calculated, all $r + 1$ (in the case of our example, 6) evaluations are compared, and the best of those becomes the new "best." This process is repeated until all attributes have been processed.

Given one combination for the first r attributes, followed by r combinations for each of the remaining $n - r$ attributes, we have $r(n - r) + 1$ complexity for each r, resulting in the following complexity for calculating all r:

$$Complexity_{Smart\ Search} = \sum_{r=1}^{n} r(n - r) + 1 = \sum_{r=1}^{40} r(40 - r) + 1 = 10{,}700$$

By eliminating attributes that perform less accurately, Smart Search runs with a computational complexity of 2.81×10^{-7} percent of that required by DFS. Apart from using no direct heuristic to further eliminate search branches, the disadvantage of using this algorithm to build a combination set, starting from attribute 1 and traversing to attribute n, is that, by design, many correlations between attributes from one side of the set to the other are eliminated from evaluation.

Weight-Based Search

The particularly crippling reality for any attribute-choosing algorithm is that each unit of complexity represents a training session, which is expensive in its own right. With 112 examples, our data set's training session will require a relatively short period of machine time, but as the dataset grows further elimination of complexity may play a key role in investigating larger data sets.

Like DFS, the Naïve Smart Search algorithm does not use any individual heuristic as a criterion for attribute elimination, but rather eliminates attribute

combinations based on the performance of the combination itself. While a drastic improvement on DFS's $O(n!)$ complexity, Naïve Smart Search still operates at $O(n^2)$ complexity for a complete search.

Naturally, the magnitude of the weight assigned to each attribute is a measure of that attribute's estimated significance to the outcome of the current combination. Thus, it is a natural and automatic heuristic for eliminating less effective attributes – this eliminates the need to evaluate each attribute in a combination as a drop candidate, reducing complexity by a factor of r, if following the pattern of Naïve Smart Search.

$$Complexity_{Weight-based\ SS} = \sum_{r=1}^{n} n - r + 1$$

$$= (n - n + 1) + (n - (n - 1) + 1) + (n - (n - 2) + 1) + \cdots + (n - 2 + 1) + (n - 1 + 1)$$
$$= (n - n + 1) + (n - n + 1 + 1) + (n - n + 2 + 1) + \cdots + (n - 2 + 1) + (n - 1 + 1)$$
$$= 1 + 2 + 3 + \cdots + (n - 1) + n$$
$$= \sum_{r=1}^{n} r$$
$$= \frac{n(n+1)}{2}$$

While this application significantly reduces the complexity of Naïve Smart Search in its steps, it still retains $O(n^2)$ complexity in its upper bound. Carrying reliance on the heuristic further, it would be possible to begin with all 40 attributes, eliminating them one at a time according to the lowest weight, until the outcome of the machine's prediction ceased to improve, coming to an estimated set of "most significant attributes" in $O(n)$ time.

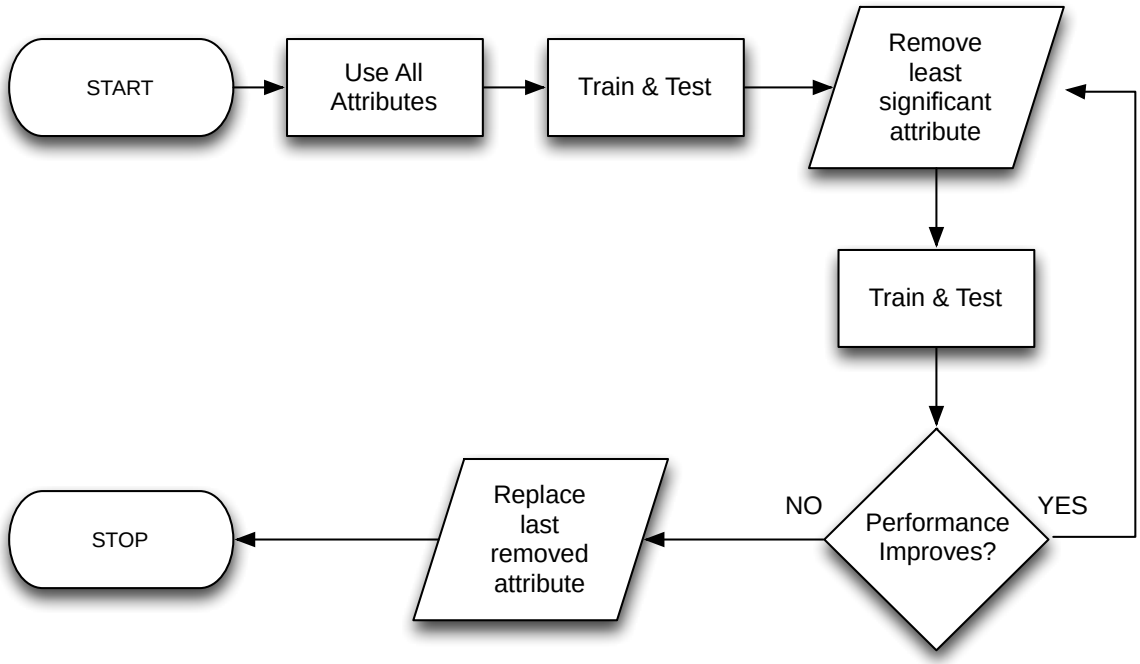

Figure 4

Weight-Based Top-Down Attribute Elimination

However, this heuristic is not exact, and becomes potentially less exact as the initial set of attributes becomes larger. [9] If Logistic Regression optimally assigns weights at the dataset's highest level of attributes, there is no need for an attribute-choosing algorithm in the first place, except to prove the fact. On the other hand, building a set from 1 to n requires more training iterations, and may miss some important hidden relationships between attributes that beginning at n may spot.

To cover both lines of thought, it would be beneficial to use the heuristic to develop an algorithm which builds a most-significant-attribute combination, starting with one attribute and traversing n, with a complexity closer to linear time.

Weight-Based Smart Search

To take full advantage of a weight-based heuristic with a 1 to n algorithm, adding and removing attributes on the fly can eliminate the summation property of Naïve Smart Search. The state machine in **Figure 5** illustrates three different phases for adding and removing attributes in a Weight-Based Smart Search. The algorithm begins by adding attributes and evaluating Logistic Regression after each add. As long as the performance of our combination improves, we continue adding.

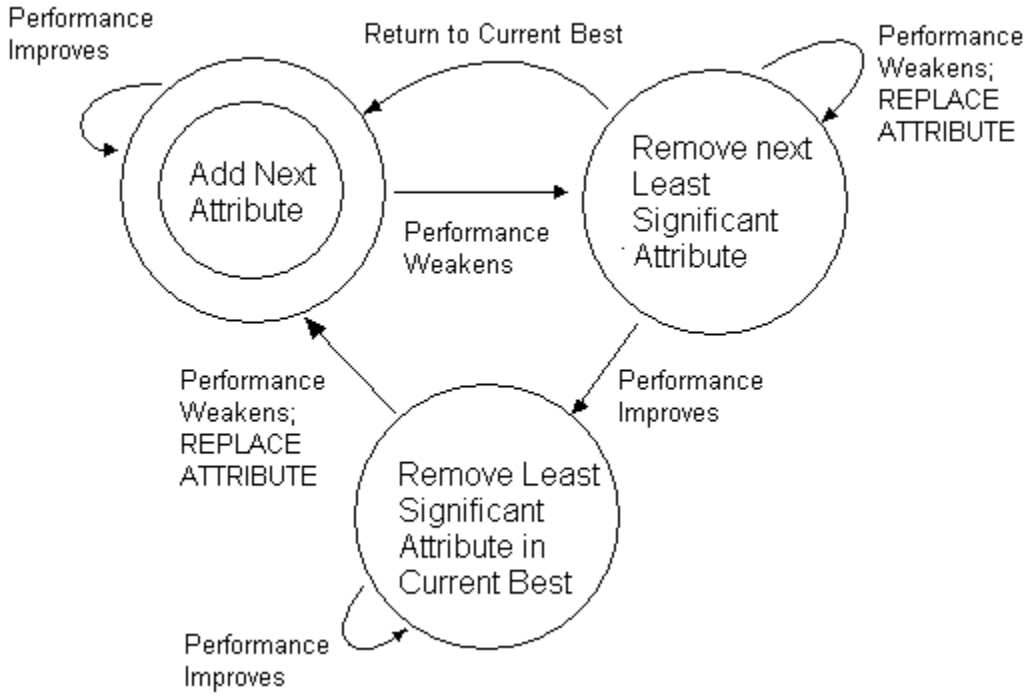

Figure 5

State Machine for Weight-Based Smart Search

When adding an additional attribute *harms* our data mining performance, we then move to a second state, which removes a single attribute at a time, starting with the attribute holding the smallest weight, that is, the “least significant” attribute. If this

removal improves our data mining performance beyond that of the current best combination, the machine moves to a 3rd state. If this removal does not improve performance, the attribute is replaced and the "next least significant" attribute is similarly removed. If this state finds itself evaluating the current best combination by one of these removals, it returns to the 1st state.

In the 3rd state, the least significant attribute in the current best combination is removed. If there is an improvement in performance, this action is repeated. If not, the attribute is replaced, and the machine returns to the 1st state. When all attributes have been evaluated, the machine finishes in the 1st state.

Unlike the previously mentioned algorithms, Weight-Based Smart Search is more stochastic in its complexity, but reasonably follows some multiple of n in the average case. In the best case – that is, the case where the best combination is that of all attributes, and a choosing algorithm is least needed – this algorithm runs with exactly n iterations. The worst case exists via a course of events exploiting an unlikely blunder in training:

1. After building a large combination of attributes in state 1 by adding half of n, an attempt to add the next attribute moves the algorithm to state 2.
2. While the new attribute is of no relation to the class attribute, Logistic Regression falsely assigns greatest significance to the new attribute.
3. State 2 traverses the entire combination with training sessions without improvement, until finally removing the new attribute and returning to state 1.
4. Repeat for the remainder of the set.

While such a course of events could lead to $(\frac{n}{2})^2$ performance, a data set would almost certainly need to be engineered to trick the training algorithm into making such poor weight estimations, and the prediction outcome for such training would be unlikely to be fruitful in the first place. Events of a *similar* nature, however, are not impossible when this algorithm is evaluating data sets, which are only filled with droves of attributes with no relation to the class attribute, where prediction outcome is poor across the board and improvements are sporadic and coincidental.

CHAPTER IV

IMPLEMENTATION

As general data mining technologies already exist, it was surmised early in the project that there was no need to reinvent the wheel with respect to implementing the chosen data mining techniques. The Weka project, from the Machine Learning Group at the University of Waikato, provides a library implementing a range of classically defined data mining algorithms, which handle missing values. [4]

After some preliminary experimentation using the Weka software itself, our attribute choosing algorithms were implemented in Java, utilizing Weka object imports. A new class, LogisticCombiner, contains the attribute searching algorithms combined with utilities for integrating this data with Weka.

Because some of these algorithms have potential to run for hours (in the case of DFS, thousands of years), with a significant usage of the heap, a tracking interface was constructed for ease of reference and for tracking algorithms as they run.

We used 10-fold cross validation to train the weights for Logistic Regression. For each training session, prediction training was repeated 10 times (for 10 *folds*) with a randomly selected testing partition of 10 instances per fold. [9]

While the use of Weka's libraries saved a considerable amount of time for constructing data mining algorithms that have already been built, over the course of working with their Logistic Regression function it was found that it only works with

nominal (non-numeric) values. Numeric values must be converted into a list of nominals, essentially destroying the advantage of training according to magnitude. This method makes for fast, efficient training, particularly when data attributes are not numeric, but there is no recognition of marginal differences between attributes.

Age 35, for instance, is evaluated as being equally different from age 80 as is age 72 – a considerable drawback for evaluating the scalar magnitude of numeric attributes. To address this problem, a second data set was created by adding series of threshold Boolean attributes. Rather than simply listing the age as a list of nominal values, we attempted to force the machine learning algorithm to recognize the magnitude of differences between ages and other numeric attributes by adding several other attributes: "Age < 20," "Age < 30," "Age < 35," etc.

Finally, a Graphical User Interface (GUI) was constructed to provide a user-friendly means to set up new searches, and track the results. Since some of these algorithms can run for a considerable amount of time, feedback was multi-threaded to maintain a reliable status on the current best find of the algorithm at any given point in the search. Figure 6 shows a screen shot of the GUI amid a Naïve Smart Search of attributes to predict 90-day mortality.

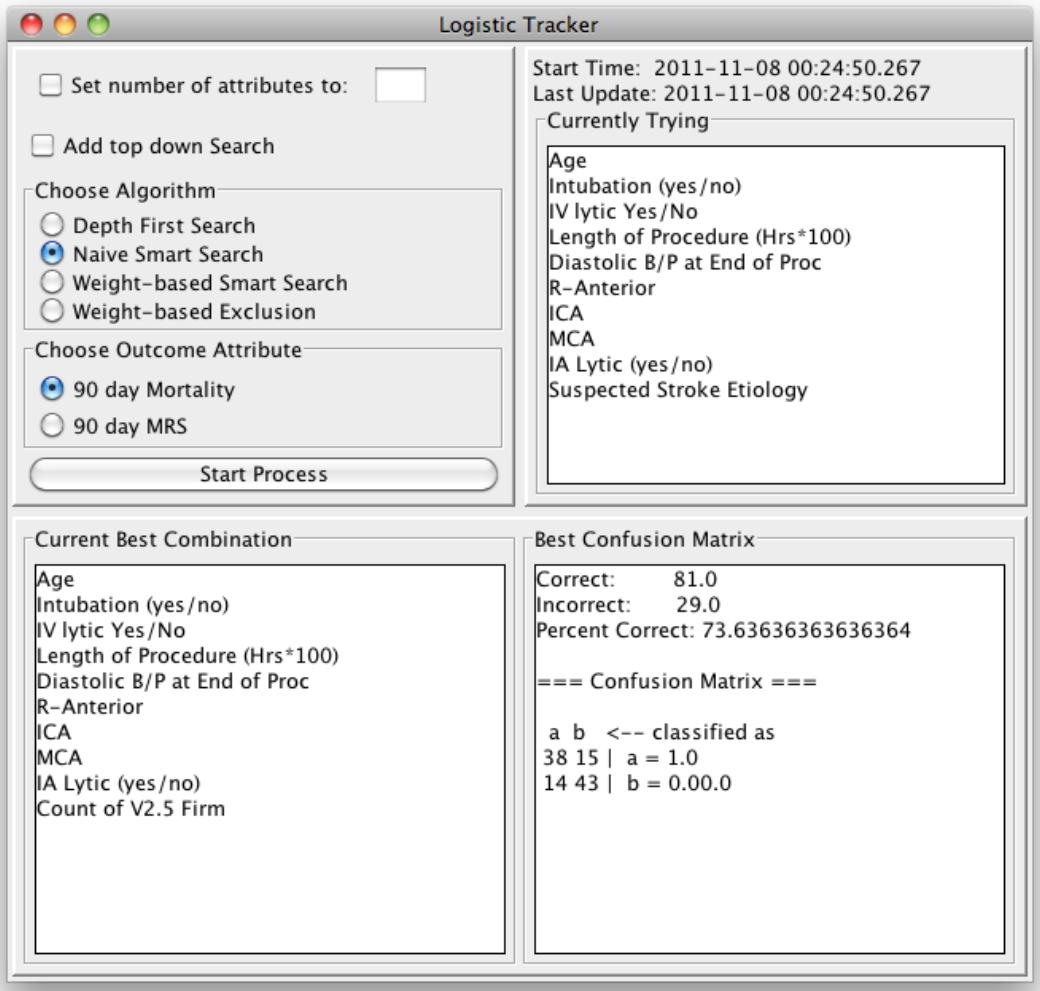

Figure 6.

Tracking GUI for Attribute Finding and Logistic Training Application

CHAPTER V

RESULTS

Due to constraints of time, a complete measurement of attribute finding with Depth First Search was impossible. Searching with a constraint of 3 attributes for single-layered Logistic Regression required 8 hours on an Intel i7 system with 1 GB dedicated to the heap, yielding a success rate of 63% for 90-day mortality training. Time prevented testing with larger sets of r.

Naïve Smart Search was able to reduce the runtime for a single attribute constraint to minutes, and completed the entire batch of constraints in 40 minutes, with a success rate averaging at roughly 74+-0.5% in single-layer Logistic Regression. False negatives vs. false positives were balanced.

Top-down weight-based attribute choosing, while finishing very quickly in a few seconds, only managed to yield a 63% success rate. As this algorithm is not novel to this project, it might be said to be a control on the other end of the spectrum to DFS.

Weight-based Smart Search, executing in less than 30 seconds, yields a success rate averaging around 70%. Table 3 shows the time requirements of each search algorithm for mortality prediction.

Table 3

Time Requirements for Mortality Prediction Searches

Algorithm	**Time, raw data**	**Time, Threshold Booleans**
DFS	Arbitrarily Large	Arbitrarily Large
Naïve Smart Search	40 minutes	Several Days
Top-Down Elimination	2-3 seconds	3 seconds
Weight-based Smart Search	30 seconds	30 seconds

Adding threshold Boolean values significantly increased prediction success rates, but also added more attributes to the data set. This did not greatly affect the run times of our Weight-based algorithms, which complete in a matter of seconds in the first place, but it considerably multiplied the run time required for Naïve Smart Search.

Figure 7 shows a graphic representation of all four algorithms, along with performance without attribute elimination. Since we were unable to run through DFS completely, we do not know how well its best attribute selection would perform. Because all of our other search algorithms are subsets of DFS, however, we know that the final pick of DFS would perform at least as well as the best values of the others, if it were given time to finish. With threshold Booleans, Naïve Smart Search ran for three days without completion – its best score prior to termination is recorded, explaining the disparity in its displayed marginal success over Weight-based Smart Search.

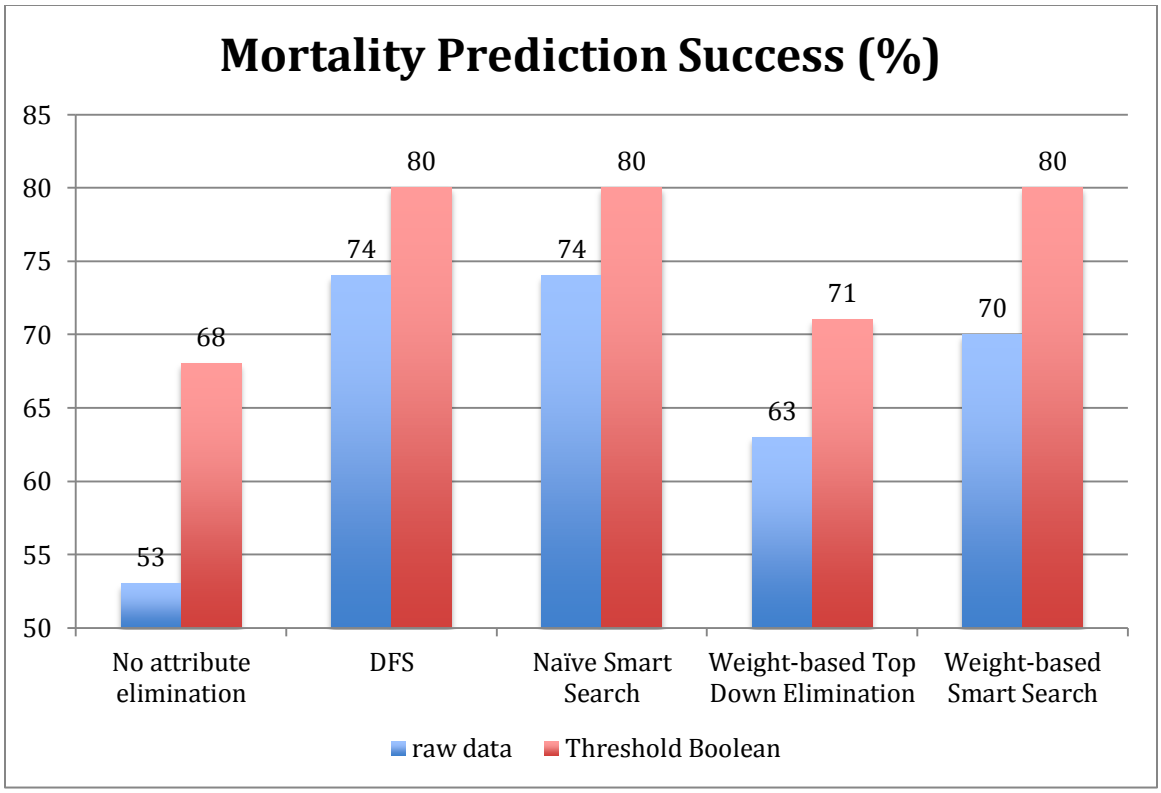

Figure 7

Mortality Prediction Success

MRS prediction, being a more complicated problem, required more time to process and was not as successful. While Naïve Smart Search ultimately finds better performing attribute sets than the weight-based algorithms, its computational disadvantage to them becomes more apparent as more complicated training sessions are required, or larger data sets examined.

Table 4

Time Requirements for MRS Prediction Searches

Algorithm	**Time, raw data**	**Time, Threshold Booleans**
DFS	Arbitrarily Large	Arbitrarily Large
Naïve Smart Search	Days	Days
Top-Down Elimination	6 minutes	3-4 minutes
Weight-based Smart Search	76 seconds	69 seconds

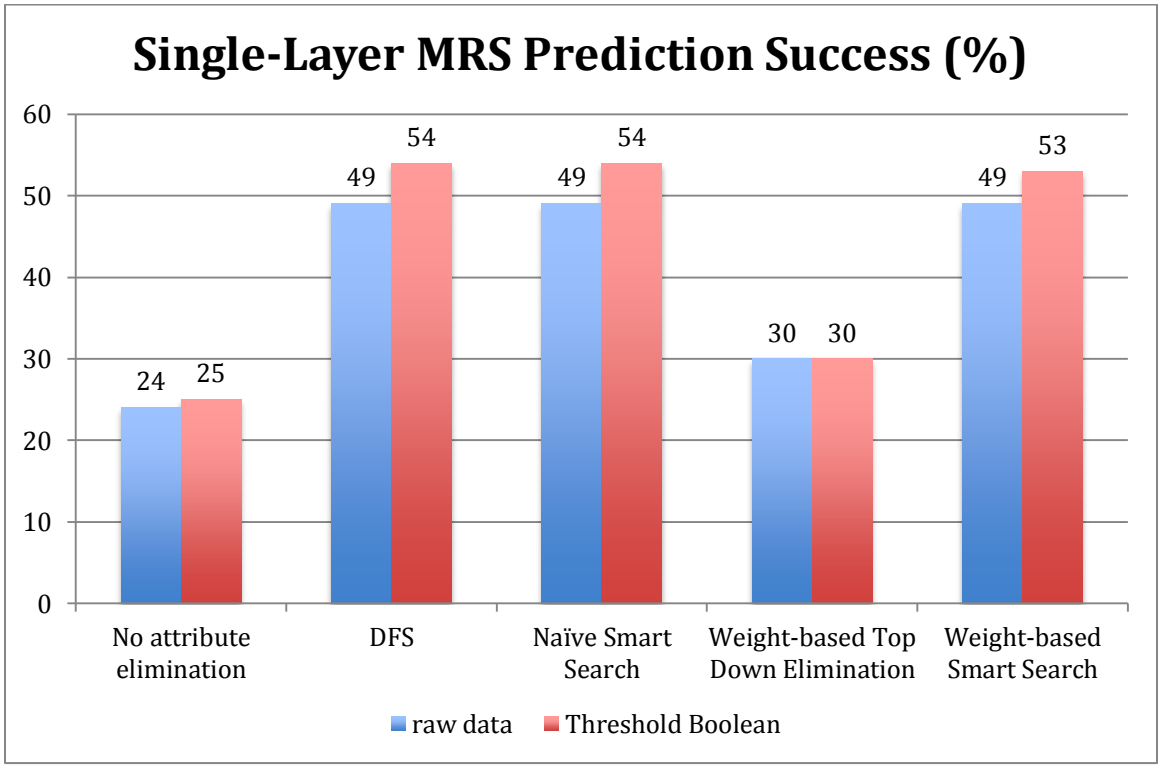

Figure 8

Single-Layer MRS Prediction Success

As seen in Table 4, an interesting shift in time requirements occurs between the results for 90-day mortality and 90-day MRS. When predicting mortality, Weight-based Top-down Elimination completed within 3 seconds for raw data and threshold Boolean sets. Weight-based Smart Search finished in 30 seconds for both, while Naïve Smart Search saw a considerable swell in run time required for the larger threshold Boolean set, due to its $O(n^2)$ complexity.

When evaluating 90-day MRS, however, Weight-based Top-down Elimination surpasses Weight-based Smart Search in time required, as training for raw data surpasses the training needed for the threshold Boolean set for both of these. The former can be explained by the extended time needed for MRS training – Weight based Top-down Elimination begins with all 40 attributes and eliminates them one at a time, while Weight-based Smart Search begins with only 1 attribute, and adds additional attributes one at a time, continually eliminating those which harm performance. The larger configurations of Weight-based Top-down search in the beginning require a great deal more time to train than the smaller configurations of Weight-based Smart Search throughout.

The latter might be explained by how Weka's implementation assigns weights. With raw data, each possible value for a nominal attribute is converted into a Boolean attribute, and then each derived Boolean is assigned 7 weights for 90-day MRS outcomes 0-6. Since the threshold Boolean set contains Boolean values which are usually chosen over the nominal attributes from which they were derived prior to any attribute searching, they are fewer in number than each nominal possibility and are only

assigned 7 weights each. This means that, while more training sessions may be required, each training session can be faster than those using the raw data.

CHAPTER VI

SUGGESTED FUTURE WORK

While attribute choosing has increased the prediction success of Logistic Regression for this data set, the following concepts are suggested as explorations for future enhancements of this project.

Solution Layering

As examined before, the distribution of results for 90-day MRS is initially unbalanced, because it is the result of taking the distribution of a balanced Boolean outcome – 90-day mortality – and splitting one of its results into 6 sub-groups, while the other remains intact under a different semantic. Since our mortality prediction has an 80% success rate, we could enhance our MRS prediction by using our mortality algorithm to predict MRS = 6. All of those patients who were predicted to survive could then be evaluated using a set of weights, which were only trained using instances where the 90-day MRS outcome was 0-5.

Figure 9 shows a graphical representation of Solution Layering for a hypothetical 90-day MRS prediction with 5 attributes. After using a choosing algorithm to optimize a mortality prediction (by eliminating attributes 3 and 5), a prediction of death becomes a prediction of MRS=6. Each patient that is predicted to survive is then re-evaluated with a 90-day MRS prediction system that is trained only with instances with values 0-5.

Because the false positives resulting from training for 90-day mortality prediction are consistently balanced, its 80% success rate can be combined with the 53.7% success rate found for the 0-5 90-day MRS training algorithm to project a likely success rate for 2-layered 90-day MRS prediction:

$Success_{Mortality}$ x (Death (%) + Survivors (%) x $Success_{MRS=0-5}$)

= 80% x (48% + 52% x 53.7%)

= 60.7%

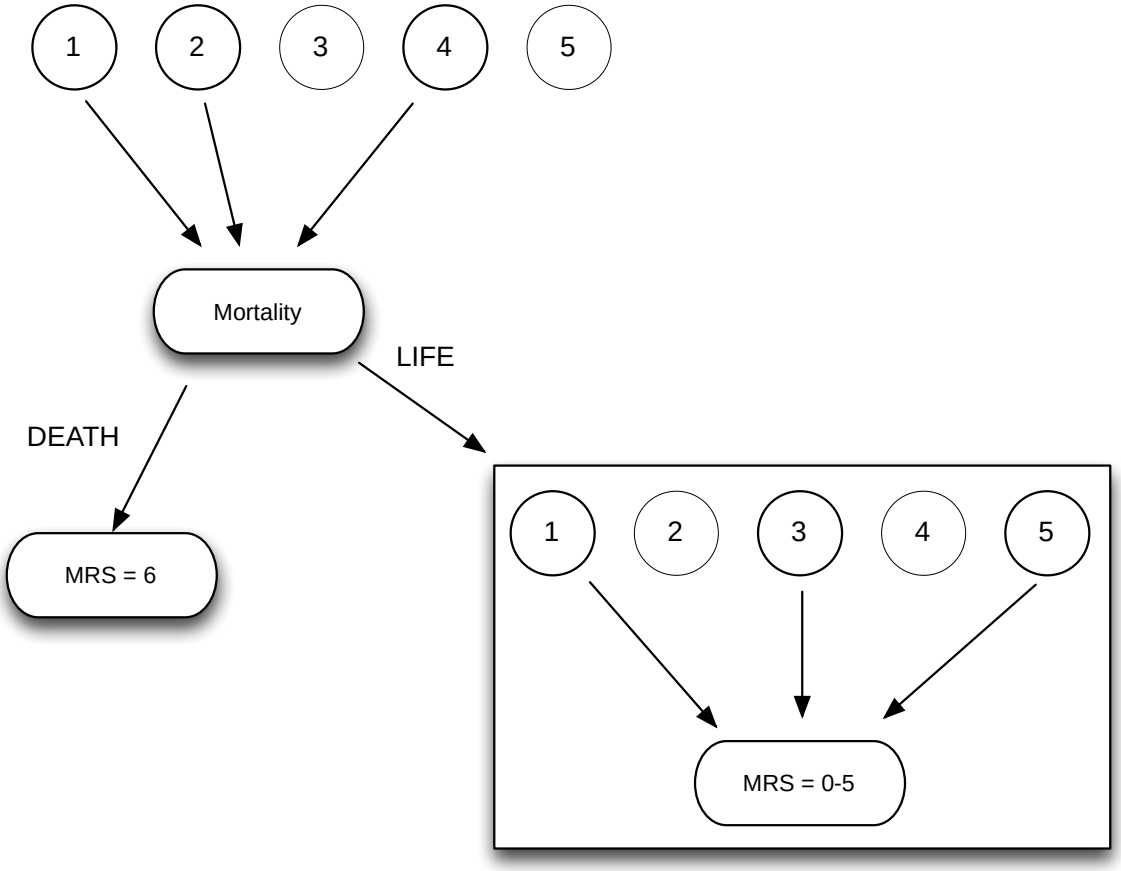

Figure 9

Solution Layering for 90-day MRS

Attribute Significance Layering

While eliminating less-significant attributes can improve the performance of Logistic Regression, the unfortunate fact remains that many of those eliminated attributes may still have a significant influence on each respective outcome – they are eliminated, simply because we do not have enough samples to pinpoint their significance, and their presence therefore skews more significant attributes. If we could somehow include them in our estimations for an improvement in prediction performance, we could both put their significance to use, and better prepare for predictions where the most significant attributes are not present.

Addressing the latter issue can be done simply by removing more significant attributes from the dataset and running our attribute finding algorithms again, finding successively significant attributes and setting up prediction criteria with them. This can make for a robust classification system, which makes the best prediction for a new instance based on the most significant data provided, without violating our white-box constraint.

To address the former, a layering of Logistic Regression training might be used, at the cost of adding further repetitions of search time. Once the most significant attributes are found, we can use their weights to combine them into a single attribute of the same class type as the class attribute, with its value set at the predicted outcome. Replacing the most significant attributes in the original data set with the new attribute could then create a new data set. Those attributes having been reduced to one single attribute, a new search could then be conducted to find the most significant attributes of

the new dataset, to see if there is any improvement in our prediction. When a point is reached where no new improvements can be made, the search is over.

Figure 10 displays a graphic representation of Significance Layering in a hypothetical 5-attribute group, where attributes 3 and 5 are ruled out by our Attribute Search in the first layer, while attributes 1, 2, and 4 are combined in initial trained predictor P_1. Attributes 3 and 5 are then evaluated in a second predictor P_2, along with predictor P_1 as an additional attribute.

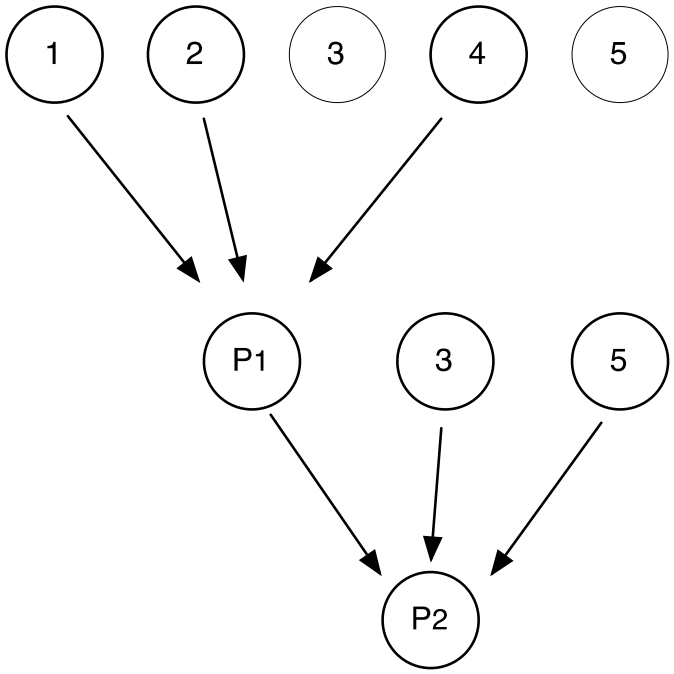

Figure 10

Attribute Significance Layering

It is important to note that, unlike the layers in a Feed-Forward Neural Network, this second layer of weights does not cloud our understanding of the first layer. P_1 is the result of using the algorithms already implemented in this project. P_2 is the result of using P_1's predictions to form a new set of data with the remaining attributes, and then

running the same algorithms. We still have the same level of information obtained by previous efforts through P_1, but by adding a second layer we are now potentially adding more details to our understanding of how significant the other attributes are. Layering in this fashion strengthens our White-Box standard.

Logical Group Layering

Attribute Layering also provides us with a means of associating attributes into logical groups. Rather than simply converging attributes according to their significance via our attribute finding algorithms, we might group attributes into logical categories. Consider the directional problem between symptoms, treatments, and disease. Symptoms and treatments are both indicators that a patient has a particular disease, and could be used by machine learning algorithms to predict or classify a patient's health – a person who has been treated for cancer, for example, is more likely to suffer from the effects of cancer than any randomly selected person who has not been treated. A machine learning algorithm would rightly use cancer treatment to predict whether a patient will suffer from the effects of cancer.

Using treatments to predict the effects of a disease, however, is actually a reliance of the algorithm upon the patient's doctor for diagnosis. It would be more useful if the algorithm could predict a patient's outcome with symptoms only, and then re-evaluate patients with treatment attributes, to see if treatment attributes might affect the patient's projected likelihood of encountering symptoms.

Figure 11 shows a graphical example of this process. First, symptoms S_1, S_2, and S_3 are examined by an attribute finding algorithm and predictor P_s is trained using selected attributes S_1 and S_2. The predictions of P_s are added as an attribute to a new

set with treatments T_1, T_2, and T_3. The attribute finding algorithm eliminates T_2, and a new predictor P_{st} is trained with P_s, T_1, and T_3.

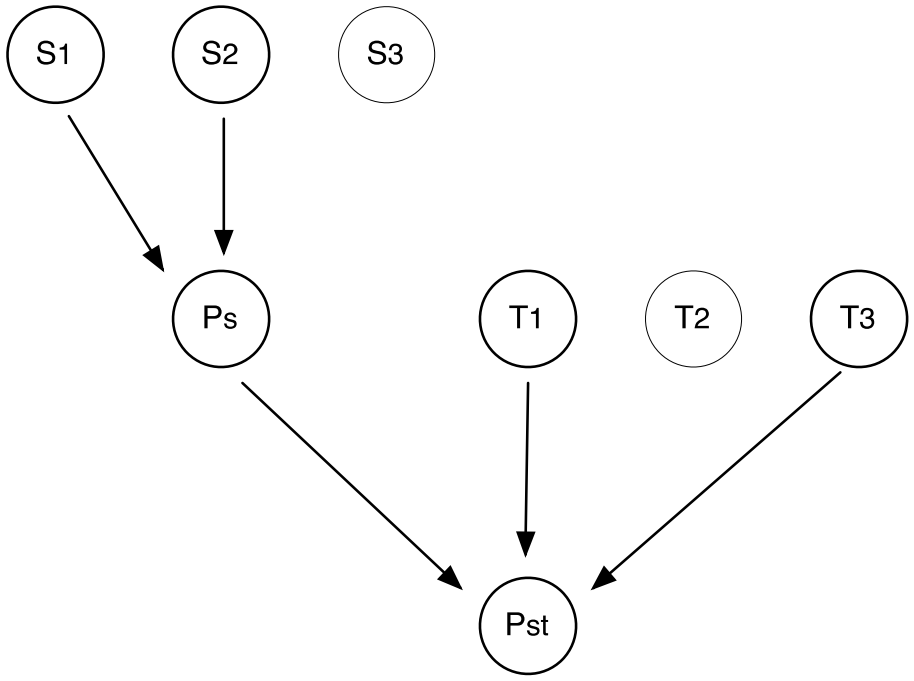

Figure 11

Logical Group Layering

Other Suggested Enhancements

In addition to layering, an examination of attribute finding with other classical machine-learning algorithms might have a worthwhile cost-benefit ratio. More benefits might be found with the same practices applied to other algorithms.

An implementation of a patient outcome prediction tool, using the findings of machine learning, could add a useful real-world application to this list of assets. The use of post-procedure data could also increase its success rate when available. While using death on the operating table to predict 90-day mortality would be a completely

useless (although unvaryingly successful) implementation of machine learning, use of post-procedure MRS evaluations of 0-5 could significantly improve prediction criteria for 90-day mortality and 90-day MRS. A patient outcome prediction tool, of course, would need to be robust enough to work with users who do not have post-procedure data as well.

CHAPTER VII

CONCLUSIONS

The attribute finding algorithms employed in this project successfully increased the prediction performance of Logistic Regression significantly in both chosen outcomes. The ultimate success results produced by these algorithms leave considerable room for improvement, particularly for MRS prediction.

While this project is dedicated to predicting the surgical outcomes of stroke patients, the fruits of this research are more generally applicable to Information Science as a whole. These solutions are not particular to medical data mining – machine learning for any dataset with the same concerns for data Instances vs. data attributes can benefit from attribute choosing algorithms.

REFERENCES CITED

[1] Ellen Barker. modernmedicine.com. [Online]. http://www.modernmedicine.com/modernmedicine/article/articleDetail.jsp?id=310563

[2] Kyra J., Thomas G. Brott Becker. (2005) Approval of the MERCI Clot Retriever: A Critical View.

[3] T.G., B.W. Baxter, T.A. Feintuch, N.A. Desbiens Devlin. (2007) The Merci Retrieval System for Acute Stroke: The Southeast Region Stroke Center Experience.

[4] TG, MM Rymer, RF Budzik, TG Devlin, HL Lutsep, WS Smith Jovin, "INTERVENTIONAL ACUTE STROKE THERAPY WITH THE MERCI RETRIEVER EMBOLECTOMY DEVICE," Erlanger Hospital, Chattanooga, TN, 2010.

[5] Tennessee Stroke Registry, "Stroke in Tennessee," TN Advisory Counsel for Heart Disease and Stroke Prevention, 2011.

[6] Zeno Gantner, Lars Schmidt-Thieme Thai-Nghe, "A New Evaluation Measure for Learning from Imbalanced Data," in *The 2011 International Joint Converence on Neural Networks*.

[7] Charles X., Chenghui Li Ling. Data Mining for Direct Marketing: Problems and Solutions.

[8] Cowen and Company. Rankinscale.org. [Online]. http://www.rankinscale.org/docs/Section_2-Extract_SG_Cowen-Therapeutic_Outlook-02102007.pdf

[9] Ian H., Eibe Frank Witten, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. San Francisco, CA, USA: Elsevier, Inc., 2005.

[10] Jim Hefferon, *Linear Algebra*. Colchester, VM, USA: Saint Michael's College.

[11] Jerome H. Klotz, *A Computational Approach to Statistics*. Madison, WN, USA: University of Wisconsin at Madison.

[12] The University of Waikato. [Online]. http://www.cs.waikato.ac.nz/ml/weka/

APPENDIX A

DATA ATTRIBUTES

Data Attribute	Brief Explanation
Patient Data	
Gender	Male/female
Age	18+
Baseline MRS	Patient's known MRS condition prior to stroke
Baseline NIHSS	Known NIHSS condition prior to stroke – another measure of circulatory health, like MRS
TICI – Pre	Thrombolysis in Cerebral Infarction – drug-induced clot busting success
TICI – Post	
Suspected Stroke Etiology	Diagnosis
R-Anterior	Clot location(s)
L-Anterior	
Posterior	
Internal Carotid Artery	
Middle Cerebral Artery	
Procedure data	
Intubation	Yes/No
Number of Retriever Passes	
Total MCs used	
IV Lytic Yes/No	Intravenous thrombolytic therapy (activase)
IA Lytic Yes/No	Intra-arterial thrombolytic therapy
IA Vasodilator yes/no	Arterial muscle relaxer
Interventionalist Last Name	Name of surgeon
Type of Proc Steps	
Number of Steps	
L4 devices used	Quantity of each device used in procedure.
L5 devices used	
L6 devices used	
Kmini device used	
Count of V2.5 Firm	
Count of V2.5 Soft	
Count of V3.0 Firm	
Count of V3.0 Soft	
Count of V2.0 Firm	
Count of V2.0 Soft	
Onset to arterial puncture	Time between onset of stroke to entrance of device
Length of procedure	

Data Attribute	Brief Explanation
Patient Data during or immediately post-procedure	
Recan (TICI 2a, 2b, 3)	Recanalization
Recan (TICI 2b, 3)	
Systolic Blood Pressure	
Diastolic Blood Pressure	
Length of stay	
Outcomes (used for class attributes)	
90-day mortality	Patient is living or dead at 90-day follow-up
90-day MRS	MRS after 90-day follow-up

VITA

Matthew McNabb was born in Fort Oglethorpe, GA to Terry and Mary McNabb, the younger of two children. After graduating Marion County High School in Jasper, TN, he attended the University of Tennessee at Chattanooga as a student of Music, and later double-majored in Computer Science. After graduation, he worked for a semester under a graduate assistantship with the University, jointly developing the “Music Therapy Gateway in Communication” project. Later, he was hired by Blue Cross Blue Shield of Tennessee as a software developer while continuing a Master of Science degree in Computer Science at UTC. Still employed by BCBST, he is awaiting graduation in August of 2012.