A Thesis

entitled

Optimizing Paired Kidney Transplant by Applying Machine Learning

by

Prakash Teknarayan Jha

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Master of Science Degree in Engineering

__________________________
Dr. Devinder Kaur, Committee Chair

__________________________
Dr. Mansoor Alam, Committee Member

__________________________
Dr. Henry Ledgard, Committee Member

__________________________
Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo

May 2011
An Abstract of

Optimizing Paired Kidney Transplant by Applying Machine Learning

by

Prakash Teknarayan Jha

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Master of Science Degree in Engineering

The University of Toledo
May 2011

In this research a tree-based machine learning algorithm has been used to build a robust mechanism that optimizes paired kidney transplants in a very proficient manner. The system predicts how good or bad a specific kidney transplant match is. The system successfully classifies and predicts donor-quality. Potential donors were classified into categories based on numeric attributes generated. Based on these numeric attributes, potential donors were classified into linguistic variables such as Good, Average and Bad. To choose the data mining technique to be employed by the system, several algorithms such as J48 (tree-based machine learning algorithm), JRip (rule-based machine learning algorithm) and SMO (Sequential minimal optimization) were considered. It was found that J48 was the better choice of all the three. The donor-classification was based on donor parameters such as age, HLA A, HLA B, HLA DR, center, CMV, EBV and blood group. The system is built using JAVA, SQL and Weka (a machine learning suite). The system also provides a visual mode of communication for doctors and surgeons to consider key factors like donor quality and donor blood group before carrying out a transplant. This feature also facilitates the doctors’ decision-making, as to where should a
chain of transplants be broken so as to ensure better and desired results as well as to provide more leads to feasible transplant chains in the future. The system developed has an accuracy of 97.18% which was generated by correctly classified instances by J48 and a promising 0.9567 kappa statistic achieved by J48. This statistic is a measure to assess the decision making capability of the system built in comparison to a real physician. The predicted donor quality was then incorporated into a matching system where possible matches were visually displayed as optimal matches and other top matches. The optimal matches also showed every donor’s blood group and donor quality, which are important in making transplant decisions.
I dedicate this thesis to the many lives that will be saved by kidney transplant surgeons worldwide.
Acknowledgements

I would like to offer my sincere gratitude to my graduate advisor, Dr. Devinder Kaur, who guided me throughout my thesis with patience and confidence in me.

My special thanks to Dr. Michael Rees, Dr. Steven Selman and Dr. Gursel Serpen, who introduced me to the concept of Paired Kidney Transplant. I would also like to extend a special thanks to Dr. Matthew Franchetti, for sharing his knowledge of developing algorithms with me.

I am very grateful to my parents, Mr. Teknarayan Jha and Mrs. Suchismita Jha and my elder brother, Pallav Jha for their love, blessings and persistent encouragement.

My most heartfelt thanks to Hitee Chandra, the most special person I have added to my life. I cannot thank her enough for all her support and help. Her true belief in my capabilities and ever so encouraging words have made me push myself to the limit and accomplish my goals.

Finally, I thank God almighty for watching over me and guiding me through life.
Contents

Abstract...........................................................................................................................................iii

Acknowledgements..............................................................................................................................vi

Contents...............................................................................................................................................vii

List of Tables.........................................................................................................................................ix

List of Figures........................................................................................................................................x

1 Introduction........................................................................................................................................1

1.1 Problem..........................................................................................................................................1

1.2 Literature review............................................................................................................................2

1.3 An Algorithmic Approach to Paired-Kidney Matching...............................................................3

1.4 Detailed Explanation of Proposed Approach.................................................................................5

1.5 Organization......................................................................................................................................7

2 Data Handling....................................................................................................................................8

2.1 Data representation.........................................................................................................................8

2.2 Reconfiguring dataset....................................................................................................................8

2.3 Using the Attribute-Relation File Format.......................................................................................9

3 Machine learning – Implementation Technique.............................................................................11

3.1 Baseline comparison.......................................................................................................................11

3.2 Horizontal error analysis of SMO.................................................................................................14

3.3 Understanding Instance Misclassification in Horizontal Error Analysis....................................15

3.4 Visualizing Error in Horizontal error Analysis............................................................................16
3.5. Vertical error analysis of SMO.................................18
3.6. Understanding Instance Misclassification in Vertical Error Analysis.........18
3.7. Error analysis of SMO on development set...............................20
3.8. Comparative Analysis of Tree and Rule based Models..........................23
3.9. Performance of JRip........................................................................24
3.10 Performance of J48.................................................................25
3.11 Evaluating Significance of J48, SMO and JRip..................................27

4 Performance Enhancement: Tuning for J48 and SMO..........................29
4.1. Deciding Settings of Variants for J48...............................................30
4.2. Optimization for J48..................................................................32
4.3. Optimization evaluation for J48......................................................34
4.4. Deciding Settings of Variants for SMO..............................................35
4.5. Optimization for SMO...............................................................36
4.6. Optimization Evaluation for SMO..................................................38
4.7. Comparison between J48 A1D, SMO A2V1 and SMO Default.................40

5 System Implementation.........................................................................43

6 Conclusion and Future Work..........................................................50

7 References........................................................................................52

A. Scoring Rules.....................................................................................54
B. Terminologies..................................................................................55
C. M5PExample.java........................................................................57
List of Tables

3.1 Blood group compatibility for organ transplant.................................18
4.1 Performance of the four J48 variants over each fold of training data.........33
4.2 Performance of the four J48 variants on new test data..........................34
4.3 Performance of the three SMO variants over each fold of training data........38
4.4 Performance of the three SMO variants on new test data........................39
List of Figures

1-1 Sub-optimal matching approach.................................................................4
1-2 Optimal matching approach........................................................................4
2-1 Reconfiguring data from (a) to (b).................................................................9
3-1 Confusion Matrices for Naïve Bayes, SMO and J48....................................12
3-2 Experimenter results showing percent correct for J48, SMO and Naïve Bayes....13
3-3 Confusion matrix showing horizontal error analysis....................................15
3-4 Screen1, classifier errors visualizer for donorDB-Train.arff with SMO.........17
3-5 Screen2, classifier errors visualizer for donorDB-Train.arff with SMO.........17
3-6 Confusion matrix showing vertical error analysis........................................18
3-7 Screen3, classifier errors visualizer for donorDB-Train.arff with SMO.........20
3-8 Confusion matrix for error analysis of SMO on development set................21
3-9 Screen1, classifier errors visualizer for donorDB-Dev.arff with SMO.........21
3-10 Screen2, classifier errors visualizer for donorDB-Dev.arff with SMO.........22
3-11 Comparison of confusion matrices for JRip and J48.................................23
3-12 JRip rules generated for donorDB-Train.arff...........................................24
3-13 J48 tree generated for donorDB-Train.arff.............................................25
3-14 Subsection of J48 tree generated for donorDB-Train.arff.........................26
3-15 Classifier errors visualizer for donorDB-Train.arff with J48.....................27
3-16 Comparing significance of J48, SMO and JRip........................................27
4-1 Weka Experimenter’s results for optimization of J48.................................32
4-2 Weka’s Experimenter results for optimization of SMO...............................37
4-3 Comparing A2V1 and A1D to SMO default............................................40
4-4 Final performance of J48............................................................................42
Chapter 1

Introduction

1.1 Problem

The aim of this thesis is to address an acute problem in the domain of renal transplantation. Kidney transplants are by a long way the most general transplant. Regrettably, the demand for kidneys far exceeds supply. In United States of America in 2005, 4,052 people died waiting for a life-saving kidney transplant. During this time, almost 30,000 people were added to the national waiting list, while only 9913 people left the list after receiving a deceased donor kidney [1]. The waiting list currently has over 80,000 people, and the mean waiting time ranges from 2 to 5 years, depending on blood type. Every year, a large number of patients waiting for a kidney transplant, die. As this problem gets worse, it is upon us that we devise smarter and more efficient ways of making kidney transplants happen and sustain. The need of the day is to come up with a way to facilitate quicker and successful kidney transplants. A rational answer to satisfy this would be to develop algorithms that help us speed up the rate of kidney transplants; and also ensure that these transplants give optimum or favorable outcomes.

Surveying and coming up with better pairing/matching algorithms for kidney transplant would invariably lead to the development of an enhanced decision making tool for carrying out kidney transplants.
1.2 Literature Review

Several recent papers have used simulation and market-clearing algorithms to explore the impact of a national kidney exchange [5, 3, 6, 7]. For example, using Edmond’s maximum-matching algorithm [2] shows that “a national pair wise - exchange market (using length-2 cycles only) would result in more transplants, reduced waiting time, and savings of $750 million in health care costs over 5 years. Those results are conservative in two ways. Firstly, the simulated market contained only 4,000 initial patients, with 250 patients added every 3 months. It has been reported to us that the market could be almost double this size. Secondly, the exchanges were restricted to length-2 cycles. Allowing length-3 cycles leads to additional significant gains. This has been demonstrated on kidney exchange markets with 100 patients by using CPLEX to solve an integer-program encoding of the clearing problem [7].” In another prominent study, Gentry and surgeon Segev demonstrated, “that a national matching program for kidney paired donation, also called paired kidney exchange, would ensure the best possible kidney for the greatest number of recipients who have incompatible donors”[4][7].

Existing methods can be profiled as follows:

a. In the paper by the researchers at Carnegie Mellon University [1], an alternative algorithm (Column generation algorithm) for this problem is suggested that can clear markets with over 10,000 patients (and that same number of willing donors).

b. There are disadvantages like the online aspect of the problem as well as the limited-information aspect of the problem. They are explained below.
I. The online aspect: Donors and recipients will be coming into the system over time, and it may not be the best practice to find an instant best match from the existing database; because in the future, there might come along a donor who is a suitable match for the current recipient; while the current donor could be a suitable match for a very critical recipient in the near future.

II. The limited-information aspect is that, the data provided as input is not entirely accurate: a number of donor-recipient pairs understood to be compatible turn out to be incompatible when more expensive last-minute tests are carried out. For instance, a medical procedure called Virtual Cross-match tells doctors how good a match is, but later on doing a real cross-match, the match sometimes turns out to be incompatible.

c. Therefore, it would be advantageous to execute an optimization with this in mind, such as outputting a “robust” result to be examined prior to the final match being produced, or to have a contingency plan in the event of failure.

1.3 An Algorithmic Approach to Paired-Kidney Matching

What is needed is to have an algorithm which does not finalize the first match found as the best possible match, an algorithm which considers all possible matches and then after a thorough comparison finds the best match and goes ahead with that.

A pictorial representation of these very words is shown in Fig.1-1 and Fig.1-2.
Figure 1-1: Sub-optimal matching approach

Figure 1-2: Optimal matching approach
1.4 Detailed Explanation of Proposed Approach

A donor-recipient pair comes to a hospital. After doing standard tests, they are deemed incompatible. The pair is advised to join the database maintained for kidney donors and recipients. The goal is to predict the quality of the donor as Good, Average or Bad which would eventually indicate whether the donor is an optimal match for new and existing recipients in the database.

The pair can be entered into the system’s databases and then the system can go ahead with finding the recipient of the pair an optimal match for kidney transplant.

This prediction system could be a part of a bigger system that eventually finds a best match for a recipient. Say ‘R1’ and ‘D1’ are the recipient and donor and are entered into the ‘R dB’ (Recipient Database) and ‘D dB’ (Donor Database) respectively. Similarly R2, R3,...,Rn will be entered into R dB and D2, D3,...,Dn will be entered into D dB.

Additionally, when multiple matches are generated for a recipient, the donor-quality characteristic would aid the process of finding an optimal match. In case of multiple optimal matches it would be wise not to pick ‘Good’ quality donors and instead save them for future matches. Here it is assumed that ‘Good’ quality donors could prove to be appropriate matches for recipients with rare characteristics that might come into the system in the future.

To initialize the kidney transplant process, data is obtained from the donor and recipient. However the system makes predictions based on a subset of this data. Based on these attributes, it will try to predict if a new recipient R1 added to the database is likely to generate matches. Once this kidney transplant process or chain has been initialized, the
system will demonstrate the cyclic manner in which it functions. The user (doctor) starts looking for a match for a recipient. The system will display all possible matches and after optimally matching and deciding, the user (doctor) clicks on a donor to execute a transplant with. Once this match has been approved, the system will automatically load the corresponding recipient’s information as next recipient in the chain and prompt the user (doctor) to generate a match for this recipient.

This can be summarized as follows:

R1 $\leftarrow$ D5

R5 $\leftarrow$ D90

R90 $\leftarrow$ D178

R178 $\leftarrow$ D1000

And so on…

Doctors’ judgments say that it would be ideal for the last donor of the chain to be an altruistic donor. An altruistic donor means a person who is willing to donate a kidney and does not have a recipient pair needing a kidney. If this happens, then there will be a complete chain of transplants. But due to the lack of altruistic kidney donors, doctors say that it is still alright for the last donor in the chain to be a ‘Good’ donor and have a blood group ‘O’.
1.5 Organization

The remainder of this document is organized as follows. Chapter 2 discusses data handling aspects of this research. Chapter 3 shows the implementation technique that was applied in building this system. Chapter 4 talks about the various performance enhancement tunings carried out. Chapter 5 gives the details and screenshots of the user interface designed for the developed application. Chapter 6 concludes the work done and throws light on future research related to this research.
Chapter 2

Data Handling

2.1 Data Representation

For developing this system, a donor dataset with 1000 donors as well as a recipient dataset with 1000 recipients was taken. The system was run for matching the entire donor dataset against the entire recipient dataset in order to find out the number of matches each donor generates. For every match generated, the donor got 1 match point. Assessing the match points each donor generated, donors were classified into three ‘donor quality’ categories viz. ‘Good’, ‘Average’ and ‘Bad’.

2.2 Reconfiguring Dataset

Fig.2-1 shows that the donor dataset was reconfigured by removing ‘gender’, ‘ethnicity’ and ‘donor_ID’ as these attributes did not play a direct role in classification of donors in this system. The ‘match points’ attribute was also removed since it has a direct mapping with the classification class i.e. ‘donor quality’.
HLA stands for human leukocyte antigen system. Epstein-Barr virus (EBV) and Cytomegalovirus (CMV) are common viruses that cause infectious mononucleosis (IM).

2.3 Using the Attribute-Relation File Format

A file with the extension arff (Attribute-Relation File Format) is nothing but an ASCII text file to facilitate explanation of a listing of instances sharing a set of characteristics. The arff header of the final donor database used is shown as follows:

```
@relation 'donorDB'
@attribute age numeric
@attribute HLA_A1 {A10,A28,A9,A19}
@attribute HLA_A2 {A10,A9,A19,A28}
```
@attribute HLA_B1 {B21,B22,B40,B12,B14,B15,B16,B17,B5}

@attribute HLA_B2 {B12,B14,B15,B16,B5,B22,B17,B21,B40}

@attribute HLA_DRB1 {N,Y}

@attribute Center {WA,CA,CO,OH,TX,NY}

@attribute CMV {+,-}

@attribute EBV {+,-}

@attribute bloodGroup {A,AB,B,O}

@attribute donorQuality {Bad,Average,Good}
Chapter 3

Machine Learning – Implementation Technique

In this chapter, key aspects like baseline comparison, error analysis and comparative analysis are discussed in detail.

3.1 Baseline Comparison

Weka is a machine learning suite which will be used for algorithm comparisons, error analyses and performance optimizations. The donor data is split into two parts: training set and development set. The first 750 instances are taken as the training set (donorDB-Train.csv) and the rest 250 instances are considered as a part of development set (donorDB-Dev.csv). The files are then converted into arff format (donorDB-Train.arff) and (donorDB-Dev.arff) to increase compatibility with Weka.

Fig.4 shows performance of donorDB-Train.arff using different algorithms with 10 fold cross validation in Weka’s Explorer. Bayes algorithm, Linear classification algorithm (SMO) and Tree based algorithm (J48) were compared to maintain diversity and perform a broad analysis.

Throughout the discussion Kappa statistic is used as the comparison and evaluation parameter. Kappa is a chance-corrected measure of agreement between the classifications and the true classes. It is calculated by taking the agreement expected by chance away
from the observed agreement and dividing by the maximum possible agreement. A value greater than 0 means that classifier is doing better than chance. Higher the kappa value, better the quality.

A confusion matrix is a means for visualizing and is used in supervised learning. Columns of the matrix correspond to instances in a predicted class, whereas rows stand for the instances in an actual class. With a confusion matrix it is simple to observe if a system is confusing two classes.

**Naive Bayes**  
Kappa value: 0.7075

**SMO**  
Kappa value: 0.8669

**J48**  
Kappa value: 0.8871

--- Confusion Matrix ---

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>26</td>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>372</td>
<td>51</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>166</td>
</tr>
</tbody>
</table>

--- Confusion Matrix ---

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>136</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>400</td>
<td>19</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>11156</td>
<td>1</td>
</tr>
</tbody>
</table>

--- Confusion Matrix ---

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>137</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>8</td>
<td>407</td>
<td>12</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>10</td>
<td>157</td>
</tr>
</tbody>
</table>

**Figure 3-1: Confusion Matrices for Naïve Bayes, SMO and J48**

By analyzing individual Kappa statistic and confusion matrix the following is observed:

1. None of the algorithms make dramatic errors like classifying a Good donor as Bad and vice versa. This conclusion can be directly mapped from individual confusion matrices generated by Weka’s Explorer.
2. Naïve Bayes shows some bias towards Average and Good donors. This is observed in the confusion matrix where 70 donors are classified as Average instead of Bad and 51 donors are classified as Good instead of Average.

3. SMO and J48 performed fairly similar on the given data. The kappa statistic is high and the difference between the individual kappas is very small. However, by only observing the individual confusion matrices it cannot be concluded whether or not this difference is significant. So Weka’s Experimenter is used to determine statistical significance.

Fig.3-2 shows the significance analyzed using Weka’s Experimenter.

Figure 3-2: Experimenter results showing percent correct for J48, SMO and Naïve Bayes

Fig.5 indicates that Naïve Bayes clearly underperforms with percent correct of 83.20 in comparison with J48 with percent correct of 93.47 and SMO with percent correct of 92.27. However, the experimenter does not show any significant difference between performance with SMO and J48. The system would then consider SMO with default Weka settings as its baseline algorithm.
Also it was seen in Weka’s Explorer that Naïve Bayes was underperforming with a low kappa value of 0.7075. Linear models like SMO, can find a good split on the data if the split exists. They will find it wherever it is.

Another reason for picking SMO as the baseline is that the system demands iterative learning on the donorDB. Iterative learning algorithms that approximate the solution of support vector machines (SVMs) have two potential advantages. First, they allow for online and active learning. Second, for large datasets, computing the exact SVM solution may be too time consuming and an efficient approximation is preferable. The powerful LaSVM (an online SVM algorithm based on a single pass through the data) iteratively approaches the exact SVM solution using sequential minimal optimization (SMO). It allows for efficient online and active learning.

However, linear functions have two problems:

1. If linear functions cannot find a good split, the results will be poor.

2. They cannot learn interactions between attributes.

To account for these shortcomings and analyze any problems that exist in the model, error analysis is performed.

### 3.2 Horizontal Error Analysis of SMO

To begin with, the confusion matrix of SMO with donorDB-Train.arff is again analyzed. As shown in Fig.3-3, the horizontal analysis says that 20 instances are being misclassified as ‘Average’ instead of ‘Bad’ (indicated in first row of confusion matrix) and 19
instances are being misclassified as ‘Good’ instead of ‘Average’ (indicated in second row of confusion matrix). Further in the horizontal analysis it is tried to understand what makes the instances different from correctly classified instances of the same class.

![Confusion Matrix](image)

Figure 3-3: Confusion matrix showing horizontal error analysis

### 3.3 Understanding Instance Misclassification in Horizontal Error Analysis

In order to analyze the reasons for wrong classification, the domain knowledge is checked with. Tissue typing is performed during the assessment of a kidney transplant. This involves sampling blood and analyzing it for markers on white blood cells which are used for matching a donor with a probable recipient. HLA-A1, HLA-A2, HLA-B1, HLA-B2 and HLA-DRB1 are usually identified antigens playing a pivotal role in tissue typing.

Epstein-Barr virus (EBV) and Cytomegalovirus (CMV) are common viruses that cause infectious mononucleosis (IM). CMV and EBV presence proves to be an essential factor in transplants as the virus strains multiply genetically in the recipient’s body after transplant. EBV/CMV infects around 90% of the world's population and can stick with a human in a latent form following primary infection. Hence, it is preferable to have a donor free from active or passive EBV and CMV strains.
In this process there is an analysis of the features that could have made the wrongly classified instances different from the instances of the class they were classified in. One such feature could be relation between attributes.

The donor characteristics include attributes that are related to each other such as;

1. HLA-A1 and HLA-A2

HLA A has two parts A1 and A2. If HLA-A1 and HLA-A2 are same, then the donor does not offer any diversity, hence might not be a good match for potential recipients.

2. HLA-B1 and HLA-B2

Similarly HLA B has two parts B1 and B2. If HLA-B1 and HLA-B2 are same, then the donor does not offer any diversity, hence might not be a good match for potential recipients.

3. CMV and EBV

If both CMV and EBV are positive then it is highly unlikely for the donor to be a good match for a recipient, except the case where the recipient himself is CMV and EBV positive.

### 3.4 Visualizing Error in Horizontal Error Analysis

In Fig.3-4, screen shot of Weka’s classifier errors visualizer, it is observed that many instances that had same HLA-A1 and HLA-A2 or same HLA-B1 and HLA-B2 or both were usually ‘Average’ or ‘Bad’ donors but were misclassified by SMO as ‘Good’. It is
possible that the mismatch is due to the fact that SMO did not consider the relationship between HLAs.

Figure 3-4: Screen 1, classifier errors visualizer for donorDB-Train.arff with SMO classifier

In Fig.3-5 it is tried to analyze if the relationship between CMV and EBV are captured by SMO. As mentioned earlier, a donor is very unlikely to be good if both CMV and EBV are positive, but as observed in Fig.8, SMO was unable to capture this relation between CMV and EBV.

Figure 3-5: Screen 2, classifier errors visualizer for donorDB-Train.arff with SMO classifier
3.5 Vertical Error Analysis of SMO

The confusion matrix of SMO with donorDB-Train.arff is further analyzed. As shown in Fig.3-6, the vertical analysis says that 20 instances are being misclassified as ‘Average’ instead of ‘Bad’ (indicated in second column of confusion matrix) and 19 instances are being misclassified as ‘Good’ instead of ‘Average’ (indicated in third column of confusion matrix).

3.6 Understanding Instance Misclassification in Vertical Error Analysis

In this section, it is tried to understand what makes the instances similar to the instances of the class they were misclassified into.

Table 3.1: Blood group compatibility for organ transplant

<table>
<thead>
<tr>
<th>BLOOD TYPE COMPATIBILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Donor can be type: If the blood type of the recipient is:</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>AB</td>
</tr>
</tbody>
</table>
From Table 3.1 it can be derived that if the donor has blood group ‘O’ then the donor is very likely to be a good donor.

Since SMO is a good linear classifier, it will classify across blood groups easily. With each function, what is learnt is like a single split in a decision tree. SMO generates a function such as;

\[ F(x) = C_0 + C_1X_1 + C_2X_2 + C_3X_3 \]

where,

\( X_1\text{--}X_n \) are attributes

\( C_0\text{--}C_n \) are coefficients / weights

Therefore SMO assigns very high coefficient to attribute Blood Group = ‘O’ while classifying between ‘Good’ and ‘Average’ donors.

This could be one reason that certain donors are always classified as ‘Good’ ignoring the equally important relationships with CMV, EBV and HLA’s. Fig.3-7 confirms this fact.
Therefore, SMO might possibly not be the optimal algorithm for the system to be built around. To make this conclusion more robust, error analysis is performed on the development set.

3.7 Error Analysis of SMO on Development Set

This section describes the error analysis of the donorDB-Dev.arff dataset trained using SMO. The confusion matrix in Fig.3-8 is first observed to find that similar errors are occurring where the algorithm confuses between ‘Bad’ and ‘Average’ instances and also ‘Good’ and ‘Average’ instances.
While visualizing classifier errors in Fig.3-9 and Fig.3-10, it is observed that error patterns similar to the ones existing in the training set are being repeated.
To sum it up from figures 3-9 and 3-10, the SMO algorithm ignores important relations between CMV, EBV and HLAs and pays major focus on blood group, which leads to misclassification of many instances.
3.8 Comparative Analysis of Tree and Rule based Models

The error analysis done in section 3.7 concluded that a model which is more efficient in considering the relationships between attributes is needed. It is known that Tree and Rule based models are very effective in capturing attribute relationships.

Tree based learning is based on ‘divide and conquer’. Decision is based on what will have the biggest overall effect on “purity” at leaf nodes. Rule based learning is based on ‘separate-and-conquer’. It considers only one class at a time (usually starting with smallest). It analyzes what separates this class from the default class.

Fig.3-11 shows comparison between one rule based and one tree based algorithm.

**JRip** (rule based)  **J48** (tree based)

Kappa value: 0.8283  Kappa value: 0.8871

--- Confusion Matrix ---  --- Confusion Matrix ---

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>Classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>30</td>
<td>0</td>
<td>a = Bad</td>
</tr>
<tr>
<td>19</td>
<td>390</td>
<td>18</td>
<td>b = Average</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>159</td>
<td>c = Good</td>
</tr>
</tbody>
</table>

Figure 3-11: Comparison of confusion matrices for JRip and J48
3.9 Performance of JRip

JRip performed poorer compared to SMO and J48 with respect to Kappa Statistic which is considered as the metric for comparison in the system. This could be because Rule learning is more prone to over-fitting.

Fig.3-12 displays the rules generated by JRip on the given data. One can observe the following:

1. JRip does a good job considering the relations between EBV and CMV while classifying ‘Bad’ donors which does lead to correct classifications.
2. JRip uses default rules for ‘Average’ donor quality which leads to many misclassifications.
3. JRip does not consider any other attribute except blood group while classifying ‘Good’ donors.
4. JRip does not consider the relationships between HLA attributes.

Figure 3-12: JRip rules generated for donorDB-Train.arff
3.10 Performance of J48

Fig. 3-13 is the tree structure constructed by Weka’s tree visualizer. This tree structure is studied in order to analyze the performance of J48.

Figure 3-13: J48 tree generated for donorDB-Train.arff
It is noticed that the tree in Fig.3-13 does not consider the relationship between HLA A and HLA B parameters. Hence it does not add much significant value to the classification that was being provided by the baseline SMO.

Fig.3-14 shows a subsection of Fig.16 created from the branch ‘O’ at the node ‘blood group’. This is the rightmost branch of the tree shown in Fig.16. This subsection of the J48 tree indicates that J48 does consider the EBV and CMV attributes in relation to each other while classifying the results. However, EBV and CMV still have low impact while classifying the donor with blood group ‘O’ as a ‘Bad’ donor.

Figure 3-14: Subsection of J48 tree generated for donorDB-Train.arff

Fig.3-15 shows that J48’s predictions created errors similar to SMO’s predictions. Although donor quality was ‘Average’, J48 classified it as ‘Good’.
3.11 Evaluating Significance of J48, SMO and JRip

In Fig.3-16, Weka’s Experimenter’s results are analyzed to compare significance of J48, SMO and JRip.

Figure 3-16: Comparing significance of J48, SMO and JRip
Fig.3-16 shows the kappa statistic of the three algorithms. The kappa for J48 is 0.89, kappa for SMO is 0.87 and kappa for JRip is 0.83. JRip has a significantly low kappa value. Therefore, JRip is not a suitable candidate.

Since J48 and SMO did not show any significant difference in their Kappa values, the system’s model should be built by fine tuning and comparing J48 and SMO results.

Hence going forward, J48 and SMO are fine-tuned and compared.
Chapter 4

Performance Enhancement: Tuning of J48 and SMO

In this section, characteristics of J48 and SMO such as minimum number of instances per leaf, confidence factor, complexity parameter, kappa value and confusion matrix are being analyzed for enhancing performance.

Algorithm: J48

Base Settings: Performance on entire dataset with default settings

Minimum number of instances per leaf = 2
Confidence factor = 0.25

Performance: Kappa value: 0.8871

Confusion Matrix:

```
  a  b  c <-- classified as
137 19  0 | a = Bad
  8 407 12 | b = Average
  0 10 137 | c = Good
```
Algorithm: SMO

Base Settings: Performance on entire dataset with default settings

Complexity Parameter = 1.0

Performance: Kappa value = 0.8669

Confusion Matrix:

--- Confusion Matrix ---

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>136</td>
<td>20</td>
<td>0</td>
<td>a = Bad</td>
</tr>
<tr>
<td>6</td>
<td>400</td>
<td>19</td>
<td>b = Average</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>156</td>
<td>c = Good</td>
</tr>
</tbody>
</table>

4.1 Deciding Settings of Variants for J48

In order to enhance the performance of J48 the confidence factor (c) and minimum number of instances per leaf (m) should be varied.

The confidence factor is used for pruning (smaller values incur more pruning).

A pessimistic guess of the actual classification error is taken based on the training data. If a given node is labeled ‘positive’ and contains 1 positive and 1 negative (incorrectly classified) instance, its error is 50%. The actual error, however, for classifying future instances at this node could be anywhere between 25% and 75% with a confidence factor of 0.25.

The default value of confidence factor for J48 is 0.25. Studies have shown that performance of the classifier on the testing set increased as the confidence factor
increased up to about 0.5 after which the classifier exhibited effects of over-training. These effects are seen by a decrease in classification accuracy with a confidence factor above 0.5.

Thus, two values of confidence factor, c=0.25 and c=0.5, are analyzed to determine which provides better performance.

The goal of tree pruning by minimum number of instances per leaf parameter is to remove the instances whose duplicates/similars occur as frequently or less frequently in coming test sets. But the tree is desired to be more complete while considering other attributes that it did not consider before; therefore the minimum number of instances per leaf parameter can be altered from 2 to a lower value.

Thus, two values of minimum number of instances per leaf, m=1 and m=2, are analyzed to determine which provides better performance.

Thus there are the following variations of the algorithm:

1. Default (A1D) : J48 with c = 0.25 and m = 2

2. Variation1 (A1V1) : J48 with c = 0.25 and m = 1

3. Variation2 (A1V2) : J48 with c = 0.50 and m = 2

4. Variation3 (A1V3) : J48 with c = 0.50 and m = 1
4.2 Optimization for J48

To perform optimization, the data is first divided into five train-test pairs using **Unstratified Remove Folds Filter**. Unstratified implies unsupervised. This particular filter is used because the train-test pairs should be divided randomly i.e. without finding any logical relation between data.

To estimate tuned performance, on each fold, all of the algorithm versions are tested over training data to find the optimal one for that fold.

In Fig.4-1, the rows show kappa statistics for the four variants over five instances of training the donorDB. Since Fig.4-1 showed many cases which were statistically insignificant, other factors such as ‘percent correct’ were also considered to make the
final decision about the optimal setting. In some cases all factors were inconclusive; so the simplest form (which is the default setting) A1D was considered.

Table 4.1 illustrates system performance over training data as generated by Weka’s Experimenter.

Table 4.1: Performance of the four J48 variants over each fold of training data

<table>
<thead>
<tr>
<th>Fold</th>
<th>Train Set Performance For C = 0.25 M = 2 (A1D) (default)</th>
<th>Train Set Performance For C = 0.25 M = 1 (A1V1)</th>
<th>Train Set Performance For C = 0.50 M = 2 (A1V2)</th>
<th>Train Set Performance For C = 0.50 M = 1 (A1V3)</th>
<th>Optimal Setting</th>
<th>Test Set Performance (From Table.3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.67</td>
<td><strong>0.68</strong></td>
<td>0.63</td>
<td>0.66</td>
<td>A1V1</td>
<td>0.8075</td>
</tr>
<tr>
<td>2</td>
<td><strong>0.87</strong></td>
<td>0.86</td>
<td>0.86</td>
<td>0.86</td>
<td>A1D</td>
<td><strong>0.8794</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>0.91</strong></td>
<td>0.89</td>
<td>0.90</td>
<td>0.89</td>
<td>A1D</td>
<td><strong>0.8087</strong></td>
</tr>
<tr>
<td>4</td>
<td><strong>0.85</strong></td>
<td>0.85</td>
<td>0.85</td>
<td>0.85</td>
<td>A1D</td>
<td><strong>0.8553</strong></td>
</tr>
<tr>
<td>5</td>
<td><strong>0.87</strong></td>
<td>0.86</td>
<td>0.87</td>
<td>0.85</td>
<td>A1D</td>
<td><strong>0.9330</strong></td>
</tr>
</tbody>
</table>
4.3 Optimization Evaluation for J48

To evaluate performance over testing data, each algorithm variant is trained using training data and the performance is tested on new test data. Table 4.2 illustrates performance of the four variants of J48 on the new test data.

Table 4.2: Performance of the four J48 variants on new test data

<table>
<thead>
<tr>
<th>Fold</th>
<th>Test Set Performance For C = 0.25 M = 2 (A1D)</th>
<th>Test Set Performance For C = 0.25 M = 1 (A1V1)</th>
<th>Test Set Performance For C = 0.50 M = 2 (A1V2)</th>
<th>Test Set Performance For C = 0.50 M = 1 (A1V3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8300</td>
<td>0.8075</td>
<td>0.8300</td>
<td>0.8670</td>
</tr>
<tr>
<td>2</td>
<td><strong>0.8794</strong></td>
<td>0.9034</td>
<td>0.8794</td>
<td>0.8670</td>
</tr>
<tr>
<td>3</td>
<td><strong>0.8087</strong></td>
<td>0.8077</td>
<td>0.8066</td>
<td>0.8187</td>
</tr>
<tr>
<td>4</td>
<td><strong>0.8553</strong></td>
<td>0.8528</td>
<td>0.8314</td>
<td>0.8199</td>
</tr>
<tr>
<td>5</td>
<td><strong>0.9330</strong></td>
<td>0.9330</td>
<td>0.9330</td>
<td>0.9330</td>
</tr>
<tr>
<td>Avg.</td>
<td><strong>0.8613</strong></td>
<td>0.8609</td>
<td>0.8561</td>
<td>0.8611</td>
</tr>
</tbody>
</table>
It is observed that the algorithm gives optimal performance over the Testing Data in two folds (highlighted in green) and gives sub-optimal performance for three folds (highlighted in red).

Now the averages across each variation are computed to choose the version of the algorithm that best optimizes the performance. Table 4.2 shows that out of all the four variants of J48, variant A1D has best kappa of 0.8613 shown in column 1. Therefore, going forward, it is chosen for comparison with SMO algorithm.

**Algorithm**: J48 A1D

**Settings**: Minimum number of instances per leaf = 2

Confidence factor = 0.25

### 4.4 Deciding Settings of Variants for SMO

Performance of SMO is analyzed by tuning the **complexity parameter (C parameter)**. Complexity parameter represents the strictness of classification. Setting a small C parameter increases the strictness of the model leading to crisp classification and making it bigger may lead to under fitting due to broader classification. The complexity parameter is used by SMO to build the 'hyperplane' between any two target classes. It is known that a support vector machine makes a hyperplane or set of hyperplanes in a high or infinite dimensional space, which is useful for classification. Instinctively, a good quality separation is achieved by the hyperplane that has the most distance to the nearest
training data points of any class, since larger the margin, the lower the generalization error of the classifier. So C controls how soft the class margins are.

Typical values for C are exponents of 10, for example 0.01, 0.1, 1.0, 10.0, 100.0. The following variations for C parameter are considered to determine if optimal performance can be achieved C = 1.0, C = 10.0 and C = 100.0

Thus there are the following variations of the algorithm:

1. Default (A2D) : SMO with C = 1.0
2. Variation1 (A2V1) : SMO with C = 10.0
3. Variation2 (A2V2) : SMO with C = 100.0

4.5 Optimization for SMO

To perform optimization, five train pairs and five test pairs that were obtained previously using Unstratified Remove Folds Filter are used.

To estimate tuned performance, on each fold, all versions of algorithm are tested over training data to find optimal one for that fold.
In Fig.4-2, the rows show kappa statistics for the three variants over five instances of training the donorDB. Since Fig.4-2 showed many cases which were statistically insignificant, other factors such as ‘percent correct’ were also considered to make the final decision about the optimal setting. In some cases all factors were inconclusive; so the simplest form that is the default setting A2D was considered.

Table 4.3 shows that A2D (default SMO) gives best performance on training data.
Table 4.3 Performance of the three SMO variants over each fold of training data

<table>
<thead>
<tr>
<th>Fold</th>
<th>Train Set Performance For C = 1.0 (A2D)</th>
<th>Train Set Performance For C = 10.0 (A2V1)</th>
<th>Train Set Performance For C = 100.0 (A2V2)</th>
<th>Optimal Setting</th>
<th>Test Set Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.56</td>
<td>0.44</td>
<td>0.44</td>
<td>A2D</td>
<td>0.8671</td>
</tr>
<tr>
<td>2</td>
<td>0.84</td>
<td>0.83</td>
<td>0.84</td>
<td>A2D</td>
<td>0.9274</td>
</tr>
<tr>
<td>3</td>
<td>0.90</td>
<td>0.88</td>
<td>0.84</td>
<td>A2D</td>
<td>0.7750</td>
</tr>
<tr>
<td>4</td>
<td>0.86</td>
<td>0.85</td>
<td>0.83</td>
<td>A2D</td>
<td>0.9287</td>
</tr>
<tr>
<td>5</td>
<td>0.85</td>
<td>0.84</td>
<td>0.80</td>
<td>A2D</td>
<td>0.9221</td>
</tr>
</tbody>
</table>

4.6 Optimization Evaluation for SMO

To **Evaluate Performance over testing data** each algorithm variant is trained using training data and the performance is tested on new test data.

Table 4.4 illustrates performance of the three variants of SMO on the new test data.
Table 4.4: Performance of the three SMO variants on new test data

<table>
<thead>
<tr>
<th>Fold</th>
<th>Test Set Performance For C = 1.0 (A2D)</th>
<th>Test Set Performance For C = 10.0 (A2V1)</th>
<th>Test Set Performance For C = 100.0 (A2V2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8671</td>
<td>0.9880</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>0.9274</td>
<td>0.9158</td>
<td>0.8255</td>
</tr>
<tr>
<td>3</td>
<td>0.7750</td>
<td>0.7966</td>
<td>0.8054</td>
</tr>
<tr>
<td>4</td>
<td>0.9287</td>
<td>0.9408</td>
<td>0.8963</td>
</tr>
<tr>
<td>5</td>
<td>0.9221</td>
<td>0.8853</td>
<td>0.8514</td>
</tr>
<tr>
<td>Avg.</td>
<td>0.8841</td>
<td><strong>0.9053</strong></td>
<td>0.8757</td>
</tr>
</tbody>
</table>

Table 4.4 shows that A2D did not give optimal performance on test data with a kappa of 0.8841. A2V1 gave the best performance with a kappa of 0.9053.

It is observed in Table 4.4 that the algorithm gives optimal performance over the Testing Data in two folds (highlighted in green) and gives sub-optimal performance for three folds (highlighted in red).
The averages across each variation are now computed to choose the version of the algorithm that best optimizes the performance. Table 4.4 shows that out of all the three variants of SMO, variant A2V1 has best kappa of 0.9053 shown in column 1. Therefore, going forward, it is chosen for comparison with J48 A1D.

**Algorithm:** SMO A2V1

**Settings:** Complexity Parameter = 10.0

### 4.7 Comparison between J48 A1D, SMO A2V1 and SMO Default

Finally, the performances of the two optimal versions of algorithms are compared to the baseline (SMO default). In other words, the best variant of J48 and the best variant of SMO are being compared to the existing baseline i.e. the default SMO.

Figure 4-3 indicates that Weka’s Experimenter results do not show any major significant improvement over SMO (default). But it is still observed that J48 (A1D) has the best kappa. Therefore, it is the best choice.

![Figure 4-3: Comparing A2V1 and A1D to SMO default](image)
Even though Weka’s Experimenter’s results do not show any significant improvement, the results of A1D (J48 default) were slightly better than that of the baseline (SMO default). Also in the error analysis, it was observed that J48 also takes into account factors such as CMV and EBV while performing classification. Presence of CMV and EBV in a donor brings down the donor-quality. This key factor was accounted for by J48 in its classification. Such factors were not considered by SMO or other algorithms. Thus the final model will be constructed using Decision Tree based learning algorithm – J48 with optimized settings: Minimum number of instances per leaf as 2 and Confidence factor as 0.25.

As a final proof of evaluation, the algorithm will be trained over the training set and tested over the development set.

Note: The development set was never used throughout the optimization process. It was only used once in error analysis to verify the error analysis done for baseline performance. Thus this evaluation might give pessimistic results.

Fig. 4-4 shows the results of J48 A1D. The results are very positive. The confusion matrix showed that only 2 instances were misclassified as Average instead of Bad and only 5 instances were misclassified as Good instead of Average. It gave a Kappa of 0.9567. It classified instances with an accuracy of 97.18%.
Figure 4-4: Final performance of J48.
Chapter 5

System Implementation

This chapter illustrates the complete system developed in Java, based on J48 algorithm. The system is programmed using Java Swings and Core Java and establishes connectivity to SQL database using MySQL server. The system also incorporates Weka’s libraries to classify and predict the output.

This system aids the matching between kidney donors and recipients by visually displaying donor blood group and predicted donor-quality.

The goal was to predict the quality of the donor of a pair which would eventually indicate whether the donor could be an optimal kidney match either in the existing donor database or in the future. This was accomplished through machine learning.

Additionally, a system capable of handling donor and recipient pairs and optimizing the matching process by introducing the element of donor quality prediction was implemented.

The system starts the cycle with a new recipient and donor pair being entered into the system’s database. Say ‘R1’ and ‘D1’ are the recipient and donor and are entered into the ‘R dB’ and ‘D dB’ respectively.
Now the system predicts matches for the newly entered recipient. When multiple matches are generated for a recipient, the quality characteristic aids the process of finding an optimal match. In case of multiple optimal matches it would be wise not to pick ‘Good’ quality donors and instead save them for future matches.

Here the procedural steps of the system are being illustrated with an example showing initial data gathering from donor and recipient, followed by an optimal match being searched for recipient R1002. An optimal match is found with donor D374. Automatically, R374 is selected for matching with an optimal donor. This shows the cyclic manner in which the system works.

Following screen shots give a step by step visualization of the entire system:

**STEP 1:** Enter new Donor’s details.

**STEP 2:** Click ‘New’ to fill Recipient’s details.
STEP 3: Enter new Recipient’s details.

STEP 4: Click on ‘Save Data’ to save Recipients data.
STEP 5: Recipient ID is generated and auto-populated on donor screen.

STEP 6: Click on ‘Save Data’ to save donor’s data along with paired recipient ID.
STEP 7: Click on ‘Generate Match’ to generate matches for recipient.

STEP 8: System generates matches for R1002 based on criteria in Appendix A. System displays donor’s blood group and also generates ‘Predicted donor Quality’ based on predictions done using J48 algorithm*.

* Uses J48 default settings and 10 fold cross validation to create model. Code provided in Appendix C.
STEP 9: One can now analyze the matched donors and select a donor (Double click) which is Average or Bad in case of multiple matches.

Reasoning: A good donor could possibly be a good match for another recipient as well.

STEP 10: System loads the data of the selected donor along with his/her paired recipient ID.
STEP 11: One can now continue the cycle and find suitable matches for R374 by clicking on ‘Generate Match’.

STEP 12: System generates matches along with the donor quality results. In this way the cyclic loop continues until no matches are found for a particular recipient or the search ends at an altruistic donor *

*An altruistic donor is a donor without a paired recipient.
Chapter 6

Conclusion and Future Work

A prediction system for optimizing paired kidney transplants has been built. Various algorithms were compared on performance metrics such as kappa statistics and correctly classified instances. Decision tree based machine learning algorithm J48 was the clear winner and hence was chosen for implementation in the system. Weka’s libraries were integrated with the Java based kidney transplant system, to train the model and predict outcomes based on real-time data. The system performed with an accuracy of 97.18%. These predictions would definitely facilitate paired kidney transplants and optimize outcomes.

Possible future works on this project are given as follows:

- To develop an interface for visualizing depth in transplant chains. A good design project could arise out of this aspect. Visualizing these chains would further add to the efficacy in communicating results or interpreted data to doctors and patients alike.

- To extend the number of parameters to the classification rationale of the system. Also there could be additions to the values an attribute can have. For example, this system considers values A9, A10, A19 and A28 as acceptable values for the
attribute HLA A. There could be more variants of HLA A that the system could accept.

- To use this classification system along with other classifying parameters to predict what type of donors generally fall out of transplant chains. This would help doctors in making decisions about individual transplant chains generated.
- J48’s performance can be visualized as a tree as depicted in Fig.16. A fuzzy decision tree could be made out of this and could open up yet another vista for further research in Fuzzy logic.
References


## Appendix A

**Table 1. Scoring Rules**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Condition</th>
<th>Points</th>
<th>Transplant 1</th>
<th>Transplant 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Age difference between donor and recipient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>Less than 10 years</td>
<td>3</td>
<td>Age difference between Donor 2 and Recipient 1</td>
<td>Age difference between Donor 1 and Recipient 2</td>
</tr>
<tr>
<td>difference</td>
<td>10 – 20 years</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21 – 30 years</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>More than 30 years</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Age difference between donors</td>
<td></td>
<td></td>
<td>Both pairs get the same number of points</td>
</tr>
<tr>
<td>Age</td>
<td>Less than 10 years</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>difference</td>
<td>10 – 20 years</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>More than 20 years</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Age of recipient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>0 – 5 years</td>
<td>8</td>
<td>Age of Recipient 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 – 10 years</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 – 18 years</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>More than 18 years</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>HLA mismatches between donor and recipient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 – 1 mismatches</td>
<td>6</td>
<td>HLA mismatches between Donor 2 and Recipient 1</td>
<td>HLA mismatches between Donor 1 and Recipient 2</td>
<td></td>
</tr>
<tr>
<td>2 – 3 mismatches</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 – 5 mismatches</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 mismatches</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Highest serologic PRA of recipient</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highest</td>
<td>More than 75%</td>
<td>2</td>
<td>Highest serologic PRA of Recipient 1</td>
<td>Highest serologic PRA of Recipient 2</td>
</tr>
<tr>
<td>serologic PRA</td>
<td>50% – 75%</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Less than 50%</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Distance between donor’s and recipient’s centers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distance</td>
<td>Same center</td>
<td>5</td>
<td>Distance between Donor 2 center and Recipient 1 center</td>
<td>Distance between Donor 1 center and Recipient 2 center</td>
</tr>
<tr>
<td></td>
<td>75 or fewer miles</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75 – 300 miles</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>More than 300 miles</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Waiting time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Waiting</td>
<td>N is the number of pairs; R is the current pair’s rank within the list of registration dates in order from shortest to longest wait</td>
<td>1 point for each year since registration, plus R/N</td>
<td>Registration date of Pair 1</td>
<td>Registration date of Pair 2</td>
</tr>
<tr>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>CMV bonus</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Donor and</td>
<td>Donor and recipient are both CMV negative</td>
<td>2</td>
<td>Donor 2 and Recipient 1 are both CMV negative</td>
<td>Donor 1 and Recipient 2 are both CMV negative</td>
</tr>
<tr>
<td>recipient</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>are both</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>EBV bonus</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Donor and</td>
<td>Donor and recipient are both EBV negative</td>
<td>2</td>
<td>Donor 2 and Recipient 1 are both EBV negative</td>
<td>Donor 1 and Recipient 2 are both EBV negative</td>
</tr>
<tr>
<td>recipient</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>are both</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>AB-to-BA bonus</td>
<td></td>
<td></td>
<td>Both pairs get the same number of points</td>
</tr>
<tr>
<td>A donor</td>
<td>A donor and recipient have blood types A and B respectively, while the other donor/recipient have blood types B and A, respectively</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reference: Michael A. Rees, M.D., Ph.D., Professor, University of Toledo Health Science Campus, Departments of Urology and Pathology.
## Appendix B

Certain terminology is explained below to help the reader effectively comprehend domain-specific jargon.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barter Exchange</td>
<td><strong>Barter</strong> is a type of trade in which goods or services are directly exchanged for other goods and/or services, without the use of money. It can be bilateral or multilateral, and usually exists parallel to monetary systems in most developed countries, though to a very limited extent. Barter usually replaces money as the method of exchange.</td>
</tr>
<tr>
<td>Matching</td>
<td>A <strong>perfect matching</strong> is a matching which matches all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching. Every perfect matching is maximum and hence maximal. In some literature, the term <strong>complete matching</strong> is used.</td>
</tr>
<tr>
<td>Column Generation</td>
<td>Column generation algorithm is used when dealing with an exponentially large number of columns. It is effectively a cost reduction algorithm.</td>
</tr>
<tr>
<td>Cycle length</td>
<td>This shows the depth achieved in transplant chains, as to how many transplants were achieved in a match cycle generated by the system.</td>
</tr>
<tr>
<td>Kidney Transplant</td>
<td><strong>Kidney transplantation</strong> or <strong>renal transplantation</strong> is the organ transplant of a kidney in a patient with end-stage renal disease.</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Social Networks</td>
<td>A <strong>social network</strong> is a social structure made of nodes (which are generally individuals or organizations) that are tied by one or more specific types of interdependency, such as values, visions, ideas, financial exchange, friendship, kinship, dislike, conflict or trade. The resulting graph-based structures are often very complex.</td>
</tr>
<tr>
<td>Exchange Markets</td>
<td>An <strong>exchange</strong> market is a highly organized market where (especially) tradable securities, commodities, foreign exchange, futures, and options contracts are sold and bought.</td>
</tr>
<tr>
<td>Recommendation Algorithms</td>
<td>Recommendation algorithms apply statistical and knowledge discovery techniques to problems.</td>
</tr>
<tr>
<td>Set Packing</td>
<td><strong>Set packing</strong> is an NP-complete problem in combinatorics, and was one of Karp's 21 NP-complete problems.</td>
</tr>
<tr>
<td>Edge Partitioning</td>
<td>It consists of dividing a graph into pieces, such that the pieces are of about the same size and there are few connections between the pieces.</td>
</tr>
</tbody>
</table>
Appendix C

/**
 *
 * Class Name : J48Weka.java
 *
 * Modification Log -
 *
 * Date                Author               Description
 *
 * ===================================================================
 * 18-Nov-10          Prakash Jha         Created
 *
 * References: M5PEexample.java
 *
 * Copyright (C) 2009 University of Waikato, Hamilton, New Zealand
 *
 **/
package kidneyTransplant;

import weka.classifiers.Classifier;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.M5P;
import weka.core.Attribute;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.SerializationHelper;
import weka.experiment.InstanceQuery;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

public class J48Weka {

}
public final static String FILENAME = "D:\FINAL\Program\Model.arff";

Instances data;

BufferedReader reader;

public static BufferedReader getReader(String fileName)
{
    BufferedReader r = null;
    try
    {
        r = new BufferedReader(new FileReader(fileName));
    }
    catch (Exception e)
    {
        // handle exception
    }
    return r;
}
public void train() throws Exception {
    
    System.out.println("Training...");

    try {
        
        reader = getReader("D:\\FINAL\\Program\\donorDB.arff");

        data = new Instances(reader);

        int classIndex = data.numAttributes() - 1;

        data.setClassIndex(classIndex);

        J48 cl = new J48();

        //further options...

        cl.buildClassifier(data);

    } catch (IOException e) {
        e.printStackTrace();
    }

    // save model + header
Vector v = new Vector();

v.add(cl);

v.add(new Instances(data, 0));

SerializationHelper.write(FILENAME, v);

System.out.println("Training finished!");
}

catch (Exception e)
{
    System.out.println("Exception (sorry!):
" + e);
}

public void predict() throws Exception
{
    System.out.println("Predicting...");

    reader = getReader("D:\FINAL\Program\Test1.arff");
data = new Instances(reader);

int classIndex = data.numAttributes() - 1;
data.setClassIndex(classIndex);

// load data from database that needs predicting

// read model and header

Vector v = (Vector) SerializationHelper.read(FILENAME);
Classifier cl = (Classifier) v.get(0);
Instances header = (Instances) v.get(1);

// output predictions

System.out.println("actual -> predicted");
for (int i = 0; i < data.numInstances(); i++)
{
    Instance curr = data.instance(i);
// create an instance for the classifier that fits the training data

// Instances object returned here might differ slightly from the one
// used during training the classifier, e.g., different order of
// nominal values, different number of attributes.

Instance inst = new Instance(header.numAttributes());

inst.setDataset(header);

for (int n = 0; n < header.numAttributes(); n++) {
    Attribute att = data.attribute(header.attribute(n).name());

    // original attribute is also present in the current dataset
    if (att != null) {
        if (att.isNominal()) {
            // is this label also in the original data?

            // Note:
            // "numValues() > 0" is only used to avoid problems with nominal
// attributes that have 0 labels, which can easily happen with

// data loaded from a database

if ((header.attribute(n).numValues() > 0) && (att.numValues() > 0)) {
    String label = curr.stringValue(att);
    int index = header.attribute(n).indexOfValue(label);
    if (index != -1)
        inst.setValue(n, index);
}

else if (att.isNumeric()) {
    inst.setValue(n, curr.value(att));
}

else {
    throw new IllegalStateException("Unhandled attribute type!");
}

// predict class

double pred = cl.classifyInstance(inst);
String strActual = "";
String strPredicted = "";

if(inst.classValue() == 0.0)
{
    strActual = "Bad";
}
else if(inst.classValue() == 1.0)
{
    strActual = "Average";
}
else if(inst.classValue() == 2.0)
{
    strActual = "Good";
}

if(pred == 0.0)
{
}
strPredicted = "Bad";
}
else if(pred == 1.0)
{
    strPredicted = "Average";
}
else if(pred == 2.0)
{
    strPredicted = "Good";
}

//System.out.println(inst.classValue() + " - > " + pred);

System.out.println(strActual + " - > " + strPredicted);

}

System.out.println("Predicting finished!");

}

public static void main(String[] args) throws Exception {

J48Weka j = new J48Weka();

j.train();

j.predict();

}