# Optimizing a Cost Matrix to Solve Rare-Class Biological Problems

**Mark J. Lawson**[1], **Lenwood S. Heath**[2], **Hai Zhao**[3], **and Liqing Zhang**[2]
[1]Center for Public Health Genomics, University of Virginia, Charlottesville, VA, USA
[2]Department of Computer Science, Virginia Tech, Blacksburg, VA, USA
[3]Department of Computer Science, Shanghai Jiaotong University, Shanghai, China

**Abstract**— *In a binary dataset, a rare-class problem occurs when one class of data (typically the class of interest) is far outweighed by the other. Such a problem is typically difficult to learn and classify and is quite common, especially among biological problems such as the identification of gene conversions. A multitude of solutions for this problem exist with varying levels of success. In this paper we present our solution, which involves using the MetaCost algorithm, a cost-sensitive "meta-classifier" that requires a cost matrix to adjust the learning of an underlying classifier. Our method finds this cost matrix for a given dataset and classification algorithm, creating a final classification model. Through a detailed description, a basic evaluation, and the application to the problem of identifying gene conversions, we show the effectiveness of this approach. Our novel approach to generating a cost matrix has proven to be quite effective in the identification of gene conversions and represents a robust way to tackle the rare-class data problem.*

**Keywords:** Rare class, cost matrix, gene conversion

## 1. Introduction

Gene conversion, an important biological process, refers to the exchange of DNA sequence information between two genes [1]. Caused through DNA strand breaks, one gene (the donor) donates part or all of its sequence to another gene (the acceptor). This can lead to two types of evolutionary processes: gene conservation and genetic diversity. By having two genes repeatedly "convert" each other for the entire sequence, they can remain identical or highly similar in sequences, despite the fact that they were duplicated a long time ago. This has been observed in genes such as ribosomal RNA genes and genes on the human X-Chromosome [2]. On the other hand, if two genes exchange only part of their sequences, it can lead to the creation of new sequences, creating the potential for genetic diversity. This has been observed in gene families where diversity is important, such as immunoglobulin genes [3] and human major histocompatiblity complex genes [4].

The identification of gene conversions is important for understanding the evolution of duplicated genes and the cause of certain genetic diseases. However, current gene conversion identification software has poor performance [2], with high false negative rates due to the fact that prediction of gene conversion is a rare-class problem.

Rare-class prediction (also referred to as "imbalanced" data prediction) is a common problem in classification [5]. In this type of problem, one class of data is far outweighed by other classes, thus making it difficult for a classification algorithm to accurately predict this class after learning. This is typically confronted in a binary class problem, in which there are two classes, often referred to as the minority and majority classes. Typically the minority class is the class of interest but the created classifier performs poorly in identifying those data members. A typical result is that the classifier classifies all data members as being majority class members, due in part to the concept of Occam's razor [6], in which the simplest hypothesis is used to create the classifier. These classification algorithms are also designed to maximize predictive accuracy, which favors the majority class.

Many approaches exist for solving the rare-class problem. These are typically one of two types: data-level approaches and algorithm-level approaches. Data-level approaches consist of two main ideas: oversampling, in which minority class members are increased through re-use, and undersampling, in which majority class members are filtered out. Both strive to attain a balanced dataset, thus allowing the classifier the ability to better differentiate between the two classes. However they both suffer from shortcomings: oversampling can easily lead to overfitting and undersampling is likely to remove relevant data objects from the training set. Recent approaches have attempted to rectify these shortcomings: SMOTE (Synthetic Minority Oversampling TEchnique) creates synthetic minority class data members based on existing ones [7] and a recent undersampling approach uses clustering to filter out irrelevant majority class data members [8].

The other methods consist of algorithm-level approaches. The most common is cost-sensitive learning in which the learning of an underlying classifier is adjusted based on predetermined misclassification costs. One of these approaches is MetaCost [9]. MetaCost takes in training data and a classification algorithm and adjusts the learning by taking into account a given cost matrix that assigns punishments for misclassifications and rewards for correct classifications. The advantage of MetaCost is that it has a "black box" approach, any classification algorithm can be used and there are no limits on types of training data. However, the cost matrix must be known in advance [10], which is usually impossible.

Currently there is no systematic way to determine an ideal cost matrix for a given dataset and classification algorithm. We propose a greedy-based approach for determining a cost matrix. Based on the given training data and the given classification algorithm, our approach incrementally searches for a cost matrix, returning the best one it finds to the user. At worst, the returned cost matrix and classification model perform as well as an unaltered classification algorithm, but our evaluations show general improvement in rare-class datasets. In this paper, we will present a formal, detailed description of this approach and illustrate why it is effective. In addition we will show its power through the classification of a basic, example dataset and how it performs in the prediction of gene conversions.

# 2. Methods

## 2.1 MetaCost

The classification problem is to take a classification algorithm $L$ and train it on a set of training data $S$, thereby creating a model $M$. $M$ is then used to predict the classes of additional data, based on a learned hypothesis. A training set $S$ consists of a set of samples, each having a vector of attributes and an assigned label. An optimal model would sufficiently learn $S$ so that it can correctly identify every $x$ in the test data $T$. However, an optimal model is typically not possible, so we seek to create an approximation that achieves the best results.

An attempt that is focused on approximating this optimal model, especially in regards to rare-class problems, is MetaCost [9]. The basic idea of MetaCost is to take a normal, unaltered classifier and adjust the learning with a cost matrix. This is done through a series of steps. The first step is to take the training data and create multiple bootstrap samples of the data. These bootstrap samples are then used for training to create an ensemble of classifiers. The ensemble of classifiers are then combined through a majority vote to determine the probability of each data object $x$ belonging to each class label. Next, each data object in the training data is relabeled based on the evaluation of a *conditional risk* function, and a final classifier is then produced after applying the classification algorithm to the relabeled training data.

The key aspect in the MetaCost learning process is to minimize *conditional risk*,

$$R(i|x) = \sum_j P(j|x)C_{i,j}. \tag{1}$$

$R(i|x)$ defines the cost of predicting that data object $x$ belongs to class label $i$ instead of class label $j$, $P(j|x)$ is the probability that data object $x$ belongs to class label $j$, and $C_{i,j}$ is the cost for making such a classification. $C_{i,j}$ corresponds to entries in the cost matrix, essentially a variant of the confusion matrix (Table 1) where $i \in \{0, 1\}$ and $j \in \{0, 1\}$. The cost matrix allows one to punish misclassifications and reward correct classifications, for example, by negative and positive values, respectively. Clearly, the success of the evaluation of the *conditional risk* function and thereby the performance of the MetaCost prediction rests on the cost matrix. Imaginably a bad cost matrix can distort the learning and produce a bad classifier. Therefore, it is imperative to identify a high quality cost matrix.

Table 1: Confusion Matrix

| | |
|---|---|
| **TP** True Positive | **FP** False Positive |
| **FN** False Negative | **TN** True Negative |

$$C = \begin{bmatrix} C_{0,0} & C_{0,1} \\ C_{1,0} & C_{1,1} \end{bmatrix} \tag{2}$$

## 2.2 Cost Matrix Optimization

The MetaCost algorithm has input values $m$, $n$, and $p$ that are essentially tweaks or givens of the algorithm once the type of classifier is determined. To simplify the function call, we can fix some default values for them, thus, the call to the MetaCost algorithm becomes a function of $S$, $L$, and $C$ and returns a classification model $M$. Let us define an evaluation function Eval$(M, T)$ that takes as input a generated model $M$ based on a cost matrix $C$ and produces an evaluation of its performance on test set $T$. This evaluation function can be based on any of the metrics for rare-class predictions such as F-measures, ROC curves, and G-mean. Assuming that we have access to the set of all possible cost matrices $(C_i, i \in N^*)$, we can then search for the cost matrix that achieves the highest evaluation value,

$$C_{best} = \arg\max_C \ \text{Eval}(M_{C_i}, T), \tag{3}$$

and denote $C_{best}$ as the *optimal* cost matrix for the given data and classification algorithm.

So the problem is how to find the *optimal* cost matrix computationally. While an exhaustive search of all possible cost matrices can guarantee that we find the *optimal* cost matrices, it is not possible. Here we propose a greedy approach to heuristically find a matrix that produces a high evaluation value.

Shown in Algorithm 1, the basic idea of the search is to start with an initial cost matrix and to increment its costs to find a cost matrix that achieves a better evaluation value. An initial cost matrix is typically a cost matrix that will create a model that is the same as the model created by an unaltered classifier. Our positive class is the minority class.

Starting with this initial cost matrix, the method creates seven new ones $(A_0, A_1, A_2, A_3, A_4, A_5, A_6)$. Each of these cost matrices represents a different combination of incrementing/decrementing the costs (correct classifications are

**Algorithm 1** Greedy-Based Search

**Input:**
$S$ is the training set
$T$ is the test set
$L$ is a classification algorithm
$n$ is the number of iterations to run the algorithm

{0,1} is the set of classes
Let Eval$(M, T)$ return an evaluation value on how Model $M$ performed on test set $T$

**Function GreedyCost**$(S, T, L, n)$

Let $I$ be the initial cost matrix where all punishments/rewards are 0
Let $C$ be the current best cost matrix, initialized to $I$
Let $M_C$ be the current best model, initialized to MetaCost$(S, L, C)$
Let $O$ be the overall best cost matrix, initialized to $I$
Let $M_O$ be the overall best model, initialized to $M_C$

**for** i = 1 to $n$ **do**
  Let $A$ be a set of cost matrices
  where
    $A_0 \leftarrow C + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$
    $A_1 \leftarrow C + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$
    $A_2 \leftarrow C + \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}$
    $A_3 \leftarrow C + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
    $A_4 \leftarrow C + \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$
    $A_5 \leftarrow C + \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}$
    $A_6 \leftarrow C + \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$
  Set $C$ and $M_C$ to $null$
  **for** j = 0 to 6 **do**
    $M$ = MetaCost$(S, L, A_j)$
    **if** Eval$(M, T) >$ Eval$(M_C, T)$ **then**
      $C$ = $A_j$ and $M_C$ = $M$
    **end if**
  **end for**

  **if** Eval$(M_C, T) >$ Eval$(M_O, T)$ **then**
    $O$ = $C$ and $M_O$ = $M_C$
  **end if**
**end for**

**return** $O$, $M_O$

decremented by one and misclassifications are incremented by one). Of note here is that the cost for correct classifications of the majority class (negative class) is not adjusted and left at zero. This is due to the fact that typically a poor classifier will order most (if not all) data members as belonging to the majority class. Therefore, there is no need to reward such behavior and our method has fewer cost matrices to test. After creating these seven new cost matrices, each one is used to create a new model through MetaCost, using the given training data $S$ and classification algorithm $L$. After these models have been created, they are evaluated on the given test set $T$ using the evaluation function Eval$(M, T)$. The model that has the highest evaluation value is kept and its cost matrix is used to initialize the next iteration of cost matrix creation.

As can be seen in the algorithm, the method keeps track of two cost matrices, a "current best" cost matrix and an "overall best" cost matrix. This was done in order to overcome one of the common problems with greedy searches, that of finding a local maximum that is lower than the global maximum. So when a potential poor local maximum is reached, it can be stored as the overall best and the method can essentially "look ahead" to see if a better cost matrix can be found. If a better one is found, the overall best cost matrix is updated. So conceivably, we can continue to generate new cost matrices while still keeping track of a good one. While this does not guarantee that a global maximum will be found, it does allow for a more comprehensive search than a typical greedy search.

The parameter $n$ is passed into the function to give a count of how many times the creation of new cost matrices occurs. A simple check of whether the overall best matrix is the same as the current best cost matrix serves as an indication of whether a maximum (local or global) has been reached. If not, the number of iterations can be increased. A possible modification to this algorithm would be to have a set number of iterations to run after a maximum has been reached.

The search for the best cost matrix can only improve upon a base classifier. At worst, the method will work as well as an unaltered classifier. This is due to the fact that a model that is built by the MetaCost algorithm with the initial cost matrix is identical to a model that was built using only the base classification algorithm. So if no better cost matrix is found, the initial cost matrix will be returned as the best.

One final note is in regards to the use of training and test data. While it is ideal if they are different, it is not necessary and training data can be used for both the creation of the model and evaluation of the cost matrix. Having a separate set of test data gives the learning process more breadth as using only one set of training data does bias the classification model towards this training data. So for evaluation purposes of the final generated classification model, one must have an additional set of test data to use that was not part of the learning process and cost matrix search.

# 3. Experiment

## 3.1 Gene Conversion Data and Classification Programs

Because actual gene conversion data is difficult to obtain, we created simulated gene conversion data similar to an approach developed by Marais [11]. Essentially we simulated the creation of a gene family from a root sequence (through mutation along a simulated phylogenetic tree) and inserted a gene conversion event between two of the genes. This way we could create recent gene conversion events (by having mutations take place mostly before the gene conversion

event) and more ancient gene conversion events (by having more mutations occur after the event). We then created two sets of data: SET1 which consisted of multiple recent gene conversions and SET2 which consisted of multiple ancient gene conversions. Each of these datasets consisted of multiple gene families (consisting of six genes each) and one (or no) gene conversion event. For each of these datasets, we created a large set of training data, a set of evaluation test data to be used in the greedy-based search, and a set of final test data to evaluate the final generated classifiers. The results shown in the next section are of how the classifiers performed on this final test data, data that was not used in the learning process.

For our experiments, we used two gene conversion prediction programs, GENECONV [12] and Partimatrix [13]. GENECONV is a program designed for the identification of gene conversions that gives a prediction of what sequence fragments have the highest, unique similarity between two sequences, ranking these predictions by $p$-value. Partimatrix uses bipartitions to determine if DNA sequences show evidence of anomalous phylogenetic history, giving support and conflict scores for each prediction.

For classification, we represented each pair of genes within a gene family through a feature vector. In this representation, we can see that gene conversion is a rare-class data problem. A set of six genes represents 15 gene pair combinations and at most one of these gene pairs will have a gene conversion event. Each of these feature vectors consists of the following attributes: average GC content, overall sequence similarity, GENECONV prediction global and pairwise $p$-values, and Partimatrix conflict and support scores.

Classification was done through the greedy-based search for a cost matrix that we detailed in the methods section. We used the following classification algorithms as the underlying classifiers: NaiveBayes (as implemented by John and Langley [14]), J4.8 (an implementation of the C4.5 decision tree learner [15]), PART (a combination of rule-based learning and C4.5 [16]), and JRip (a rule-based learner based on RIPPER [17]). All classification algorithms were implemented in weka [18], a collection of machine learning algorithms.

## 3.2 Results

In Table 2 we can see the classification results. For our purposes the positive class is when a gene pair has a gene conversion and the negative class is when it does not. For the learning of each set, we created separate training and test data and then evaluated the final model on a second set of unique test data.

In SET1, one can see that GENECONV performs quite well. "GENECONV Strict" has a high accuracy, even higher than the "Just Say No approach". However, through our method we are able to increase the amount of true positives, increase the accuracy, and most importantly, increase the F-measure. The best performers are JRip and PART, which is not surprising as they are rule-based classifiers and rule-based classifiers are known to perform well on rare-class data [19]. Both have a higher F-measure than "GENECONV Strict", a higher accuracy, and both identify more true positives. J4.8 does well too and identifies more true positives than PART, but more false positives as well. Of all the cost matrix classifiers, NaiveBayes identifies the most true positives, but is hindered by the number of false positives it identifies.

In SET2, one can see that ancient gene conversions are far more difficult to accurately detect, as the mutations after the conversion makes some difficult to differentiate. GENECONV performs quite poorly, both in Strict and LP. Partimatrix identifies more gene conversions and G-or-P has the best F-measure of these basic classifiers. This set also shows the shortcoming of using accuracy as a metric as the "Just Say No" approach would appear to be the best classifier. Among the cost matrix classifiers, the NaiveBayes classifier performs quite poorly. It has an F-measure lower than G-or-P, so it shows no improvement over a basic classifier (it does not identify more gene conversions

Table 2: Simulation Results

**SET1**

| Classifier | TP | FP | Accuracy | F-measure |
|---|---|---|---|---|
| *Perfect* | 139 | 0 | 1 | 1 |
| *Just Say No* | 0 | 0 | 0.937 | UNDEF |
| *GENECONV Strict* | 102 | 4 | 0.975 | 0.840 |
| *GENECONV LP* | 123 | 57 | 0.955 | 0.776 |
| *Partimatrix* | 9 | 137 | 0.833 | 0.064 |
| *G-or-P* | 128 | 191 | 0.874 | 0.561 |
| *NaiveBayes* | 122 | 58 | 0.954 | 0.770 |
| *PART* | 107 | 5 | 0.978 | 0.859 |
| *J4.8* | 109 | 11 | 0.975 8 | 0.848 |
| *JRip* | 111 | 9 | 0.978 | 0.864 |

**SET2**

| Classifier | TP | FP | Accuracy | F-measure |
|---|---|---|---|---|
| *Perfect* | 150 | 0 | 1 | 1 |
| *Just Say No* | 0 | 0 | 0.933 | UNDEF |
| *GENECONV Strict* | 1 | 8 | 0.930 | 0.014 |
| *GENECONV LP* | 5 | 68 | 0.905 | 0.045 |
| *Partimatrix* | 15 | 135 | 0.880 | 0.100 |
| *G-or-P* | 19 | 197 | 0.854 | 0.104 |
| *NaiveBayes* | 8 | 75 | 0.904 | 0.069 |
| *PART* | 35 | 214 | 0.854 | 0.175 |
| *J4.8* | 23 | 160 | 0.872 | 0.138 |
| *JRip* | 40 | 265 | 0.833 | 0.176 |

This table represents the performance of the various classification methods on datasets SET1 and SET2. The upper half represents the basic classifiers that do not use the greedy-based approach. *Perfect* represents a theoretical optimal classifier and is included for comparison. *Just Say No* represents a classifier that classifies all data elements as majority class. *GENECONV Strict* uses only global $p$-values for predictions, whereas *GENECONV LP* uses local pairwise $p$-values (with 0.05 being used as the threshold for positive classification). *Partimatrix* represents a prediction based on the lowest conflict score between a gene pair within a gene family. *G-or-P* is a basic unification of *GENECONV LP* and *Partimatrix* predictions. The lower half represents the classification algorithms predictions after using the greedy-based search for a cost matrix.

correctly either). But the rule-based classifiers again perform quite well, with both identifying more gene conversions and having higher F-measures than any of the basic classifiers. In fact, aside from NaiveBayes, all classifiers exhibit both a higher recall and a higher precision than the basic classifiers, showing a definite improvement.

Table 3: Final Generated Cost Matrices

| | SET1 | | SET2 | |
|---|---|---|---|---|
| NaiveBayes | -3 | 2 | -2 | 2 |
| | 2 | 0 | 1 | 0 |
| PART | -4 | 3 | -3 | 1 |
| | 5 | 0 | 19 | 0 |
| J4.8 | -2 | 3 | 0 | 1 |
| | 4 | 0 | 4 | 0 |
| JRip | -4 | 1 | 0 | 1 |
| | 6 | 0 | 7 | 0 |

In Table 3, we can see the cost matrices that were determined for each classifier by the greedy-based approach and subsequently used to make gene conversion predictions. From this table it is quite clear that a cost matrix is highly dependent on both the classifier and the data being used. No classifier has the same cost matrix across both datasets and no dataset has a cost matrix that is best for more than one classifier. In fact, all cost matrices that were determined by our approach are unique. All final classification models were generated after 25 iterations of the greedy-based search method.

## 3.3 Additional Analysis

In order to further analyze the improvement our greedy search method has over an "unaltered classifier," i.e. a classifier whose learning has not been altered by a cost matrix, we generated 10 samples for each gene conversion dataset and compared the performance of each classification algorithm. These samples were created by taking each dataset, SET1 and SET2, and splitting up the data as 1/2 training, 1/4 evaluation test data (for the evaluation in the greedy algorithm), and 1/4 for the final test data, which was not involved in the learning process. For the unaltered classification the evaluation test data was added back to training data, so 3/4 of the data was used for training and 1/4 for testing. Since we are dealing with datasets in which the amount of positive examples is few and the ratio between positive and negative examples is important, the creation of these samples was not entirely random. First the dataset was ordered into positive and negative examples. Then 1/2 of the positive samples were put into the training data, 1/4 in the evaluation test data, and 1/4 into the final test data. The same is then done with the negative data. This ensures that each set contains positive data members and that the ratio is conserved. After running the simulations, we used a Wilcoxon signed rank test [20] to determine the significance of improvement.

In Tables 4 and 5, we see the resulting F-Measures, summarized as average, maximum, and minimum. In the SET1 simulation results, we can see improvement for each classification algorithm except NaiveBayes. The others see improvement in their average, maximum, and minimum F-Measures (however JRip has a lower maximum). Unfortunately, only PART shows significant improvement by using the greedy method, generating a $p$-value of 0.024.

In the SET2 simulation (Table 5), we see F-Measure improvements for all classification algorithms. However, the improvement for JRip is misleading. In its unaltered form, the generated classifiers made no positive predictions, hence the F-Measure of 0. The classifiers generated with the greedy method made ONLY positive predictions, generating an F-Measure of 0.125 each time. Clearly, this is not a "better" classifier. The other three classification algorithms showed significant improvement, each generating a $p$-value of 0.0027.

Table 4: SET1 Simulation

| | F-Measure | | |
|---|---|---|---|
| **Classifier** | **Average** | **Max** | **Min** |
| *Unaltered* | | | |
| NaiveBayes | 0.770 | 0.796 | 0.744 |
| JRip | 0.874 | 0.904 | 0.846 |
| PART | 0.858 | 0.880 | 0.823 |
| J4.8 | 0.861 | 0.883 | 0.839 |
| *Greedy* | | | |
| NaiveBayes | 0.766 | 0.793 | 0.738 |
| JRip | 0.876 | 0.892 | 0.857 |
| PART | 0.875 | 0.885 | 0.862 |
| J4.8 | 0.873 | 0.900 | 0.848 |

Table 5: SET2 Simulations

| | F-Measure | | |
|---|---|---|---|
| **Classifier** | **Average** | **Max** | **Min** |
| *Unaltered* | | | |
| NaiveBayes | 0.105 | 0.119 | 0.089 |
| JRip | 0.000 | 0.000 | 0.000 |
| PART | 0.014 | 0.031 | 0.000 |
| J4.8 | 0.000 | 0.000 | 0.000 |
| *Greedy* | | | |
| NaiveBayes | 0.135 | 0.174 | 0.125 |
| JRip | 0.125 | 0.125 | 0.125 |
| PART | 0.176 | 0.185 | 0.161 |
| J4.8 | 0.151 | 0.169 | 0.140 |

## 3.4 Real-World Data

In order to see how the generated classification models perform on real world data, we used the "Transcription Elongation Factor A" gene family on the X-Chromosome that has been shown to exhibit gene conversions [2]. The three gene family members are located in a large syntenic region that is conserved between primates and rodents, indicating that these genes were generated/duplicated before

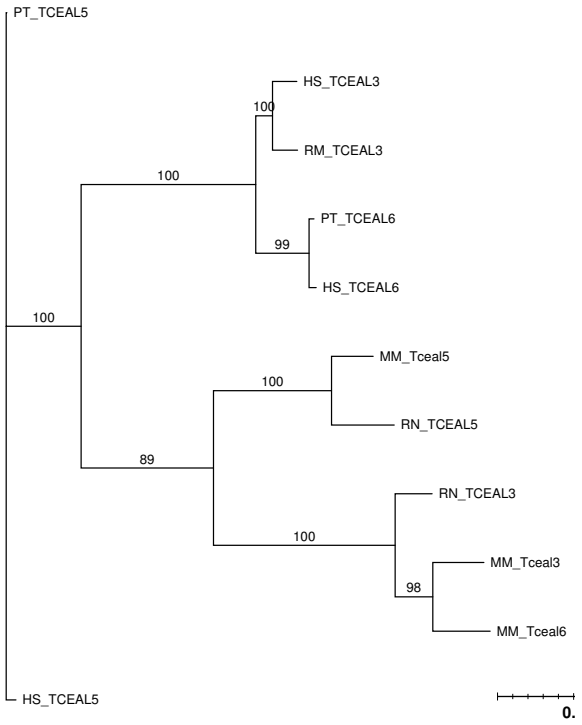


Fig. 1: Transcription Elongation Factor A phylogenetic tree
HS = Homo sapien, PT = Pan troglodytes, RM = Rhesus Monkey (Macaca mulatta)
MM = Mus musculus, RN = Rattus norvegicus

the split of primates and rodents. Thus, the phylogenetic tree (Figure 1) provides strong evidence for gene conversions within biological orders (primates and rodents). Interestingly, gene conversion seems to occur independently in both primates and rodents after their split but before the further splits within primates and within rodents.

We used the classification models from SET1 that were generated with the PART, J4.8, and JRip classification algorithms (NaiveBayes was left out due to its poor performance). Both PART and J4.8 made the same 3 predicted gene conversions: MM_Tceal3 and MM_Tceal5, MM_Tceal6 and MM_Tceal5, and RN_TCEAL3 and RN_TCEAL5. JRip made the same predictions, with the addition of a gene conversion between HS_TCEAL5 and PT_TCEAL6 that is a false positive due to the fact that gene conversion only occurs between genes from the same species. These predictions are consistent with the phylogenetic evidence.

Using the threshold of a $p$-value of 0.05 as sufficient evidence that two genes have undergone gene conversion GENECONV Strict gives evidence for 13 gene conversions and GENECONV LP for 44. While some gene conversions do correspond with what is seen in the graph, others do not, for instance gene conversions between primates and rodents. Partimatrix does not provide guidelines for a threshold to be used for predicting gene conversions. However those with the highest support scores also involve conversions between primates and rodents.

Unlike the simulated cases where we can use the F-measure to compare the performance of gene conversion prediction programs with our ensemble method, it is difficult to perform this analysis on real data because we do not know the exact numbers of true or false positives and negatives. The challenge of the difficulty in performance evaluation on real world data can be addressed in future work by manual compilation of a carefully monitored set of genes for which exact numbers of true or false positives and negatives can be accurately inferred.

# 4. Discussion

Due to the complexity and uniqueness of datasets, as well as the differing performance of classification algorithms, the best performance can be achieved with a cost-sensitive classification method when a best cost matrix is found for both the given data and the given classification algorithm. Theoretical research on the rare-class problem has shown that aspects of data that are difficult to quantify (such as the "complexity of concept") play a role in classification [10] and our own results have shown that a cost matrix that achieves good performance is dependent on both the given training data and the given classification algorithm. Thus a cost matrix must be found taking these two entities into consideration.

A greedy search is efficient but not optimal. While it cannot be proven that the eventual "overall best" cost matrix is one that achieves optimal classification results, we have shown that it will improve upon an unaltered classifier. At worst, the resulting classification model will perform as well as a classifier that was generated without MetaCost. This is more than can be said about other methods for dealing with rare-class data that can cause overfitting and/or eliminate relevant data and achieve even poorer results.

One thing we were able to recreate with this method, was the "black box" approach that MetaCost used. Of importance was the fact that the details are hidden from the end-user, with inputs being passed in and a final model being returned, with little user interaction. Our approach requires only the same inputs with the simple addition of a value being given for the number of iterations. At the end of these iterations, the best model and cost matrix found will be returned to the user. In addition, our method only requires a "meta-classifier" that takes in a cost matrix and adjusts the learning of a classification algorithm according to it. While MetaCost is a great method for accomplishing this, it can easily be replaced with a method that might be better suited for a specific problem domain.

Our future work will focus on improving the search for a best cost matrix. Simulated annealing [21] and genetic

algorithms [22] will be experimented with to see if they achieve better performance in terms of classification. While these methods can achieve better results than greedy search, they do require more time as they generate many more possible solutions. Therefore, we will investigate whether there is a trade-off between performance gain and increased searching time when compared to the greedy-based solution. In addition, we will also look into any improvements to the MetaCost algorithm that may increase performance (for instance, using boosting instead of bagging to determine probabilities as suggested in [23]). Finally, although our current analysis shows that MetaCost with the greedy search of a cost-matrix made some improvement in predicting gene conversion over GENECONV and Partimatrix, it is based on simulated data and rather limited real data. Future work will involve the curation and application of more real data on gene conversion to train and test models in order to further improve the performance of the prediction programs.

# 5. Acknowledgments

We thank Naren Ramakrishnan for helpful suggestions. The work was supported by NSF grant IIS-0710945 to L.Z.

# References

[1] J.-M. Chen, D. N. Cooper, N. Chuzhanova, C. Ferec, and G. P. Patrinos, "Gene conversion: Mechanisms, evolution and human disease," vol. 8, pp. 762–775, 2007, nature Reviews Genetics.

[2] M. J. Lawson and L. Zhang, "Sexy gene conversions: Locating gene conversions on the X-chromosome," *Nucl. Acids Res.*, vol. 37, no. 14, pp. 4570–4579, 2009. [Online]. Available: http://nar.oxfordjournals.org/cgi/content/abstract/37/14/4570

[3] N. Maizels, "Immunoglobulin gene diversification," *Annual Review of Genetics*, vol. 39, pp. 23–46, 2005.

[4] N. Takahata and Y. Satta, "Selection convergence, and intragenic recombination in HLA diversity," *Genetica*, vol. 103, pp. 157–169, 1998.

[5] N. V. Chawla, N. Japkowicz, and A. Kolcz, "Editorial: Special issue on learning from imbalanced data sets," *SIGKDD Explorations*, vol. 6, no. 1, pp. 1–6, 2004.

[6] P. Murphy and M. Pazzani, "Exploring the decision forest: An empirical investigation of Occam's razor in decision tree induction," *Journal of Artificial Intelligence Research*, pp. 171–187, 1994.

[7] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[8] K. Yoon and S. Kwek, "A data reduction approach for resolving the imbalanced data issue in functional genomics," *Neural Computing & Applications*, vol. 16, pp. 295–306, 2007.

[9] P. Domingos, "MetaCost: A general method for making classifiers cost-sensitive," *Advances in Neural Networks, International Journal of Pattern Recognition and Artificial Intelligence*, pp. 155–164, 1999.

[10] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–450, 2002.

[11] G. Marais, "Biased gene conversion: Implications for genome and sex evolution." *Trends Genet*, vol. 19, no. 6, pp. 330–8, 2003.

[12] S. Sawyer, "Statistical tests for detecting gene conversion," *Molecular Biology and Evolution*, vol. 6, no. 5, pp. 526–538, 1989.

[13] I. B. Jakobsen, S. R. Wilson, and S. Easteal, "The partition matrix: Exploring variable phylogenetic signals along nucleotide sequence alignments," *Molecular Biology and Evolution*, vol. 14, no. 5, pp. 474–484, 1997.

[14] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Eleventh Conference on Uncertainty in Artificial Intelligence*. San Mateo: Morgan Kaufmann, 1995, pp. 338–345.

[15] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[16] E. Frank and I. H. Witten, "Generating accurate rules sets without global optimization," *Fifteenth International Conference on Machine Learning*, pp. 144–151, 1998.

[17] W. W. Cohen, "Fast effective rule induction," *Machine Learning: Proceedings of the Twelfth International Conference (ML95)*, 1995.

[18] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Elsevier, 2005.

[19] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction To Data Mining*. Addison-Wesley, 2006.

[20] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, pp. 80–83, 1945.

[21] B. Suman and P. Kumar, "A survey of simulated annealing as a tool for single and multiobjective optimization," *Journal of the Operational Research Society*, vol. 57, no. 10, pp. 1143–1160, 2006.

[22] C. R. Reeves and J. E. Rowe, *Genetic Algorithms — Principles and Perspectives*. Kluwer Academic Publishers, 2003.

[23] K. M. Ting, "An Empirical Study of MetaCost using Boosting Algorithms," *In: Proceedings of the Eleventh European Conference on Machine Learning*, pp. 413–425, 2000.