A Logic-theoretic classifier called Circle

Mehmet Dalkilic
School of Informatics
Center for Genomics and BioInformatics
Indiana University
Bloomington, IN 47405
dalkilic@indiana.edu

Arijit Sengupta
Information Systems
Kelley School of Business
Indiana University
Bloomington, IN 47405
asengupt@indiana.edu

Abstract

We present a novel classifier based upon principles of logic-theoretic Boolean function minimization. The classifier, called Circle, recursively produces a set of implicants (or rules). The implicant set contains information not only about the presence of features, but also about their absence in determining class values. Thus, Circle's implicant set is initially non-monotonic with respect to inserting new tuples that have feature values that were not in the training set. One important benefit of this non-monotonicity, however, is that Circle is capable of being robust in the presence of novel feature values. We have created a full implementation of Circle using Java as a host language and Oracle database backend. Because we are interested in data mining in bioinformatics, particularly genomic data, the database was borne out of necessity to both manage and effectively query the information.

1 Introduction

One of the hallmarks of intelligence is the ability to classify things–to find some overarching properties that do a reasonably good job at explaining or determining how things are grouped together[1, 2, 3]. Formally, we are presented with a set of feature value, class label pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x_n}, y_n)\}$. Each $\mathbf{x}_i$ denotes a list of values (features) to be used in creating a classifier. The y_i represent *a prior* knowledge of what class the respective $\mathbf{x}_i$ belongs to. This is the often used expression "supervised learning", since we are provided with the correct class. For this discussion we assume that a particular list of features maps functionally to a class label.

In our current research[4], we are inundated with categorical data. We have been struck the by the fact that a novel, useful way to begin thinking about this data was not solely statistically, but logically. One of the things that caught our attention was the work in Boolean circuit minimization. Begun some 50 years ago and discovered independently by several different people, the ability to find an equivalent, but more efficient circuits (Boolean functions) seemed to be related to classification. Consider a Boolean function $f : \{0,1\}^k \rightarrow \{0,1\}$. There are 2^k features representing, in this case, all the possible features. There are two classes, 0 (**false**), 1 (**true**); sometimes * (**don't care**) is included, but this can be interpreted as no class is specified. The task is to find an equivalent $g : \{0,1\}^k \rightarrow \{0,1\}$ that classifies the inputs much more efficiently. In its simplest form, $xy + x\overline{y} = x(y + \overline{y}) = x$, where x and y are Boolean feature values.

We develop a little notation to explore this further. Assume f is a Boolean function over k Boolean variables $B_1, B_2, \ldots, B_k$, and we discover that there is some B_i such that $f(B_1, B_2, \ldots, B_i = 0, \ldots, B_k) = f(B_1, B_2, \ldots, B_i = 1, \ldots, B_k)$, then B_k is not necessary and can be left out. Kaurnaugh or Veitch-Kaurnaugh maps (K-maps, KV-maps)[5] create a grid of cells that represent the sum-of-products (SOP), a canonical form where the function is a disjunction of conjunctions. The cells contain the classifier–the Boolean output 1 and sometimes "don't cares." The method is to circle all the 1s, while drawing circles as large as possible avoiding cells that do not explicitly contain 1. Once the circling is complete, we simply produce the sum of products, where product is the most general description of a circle. Consider, for example, the Boolean function in Fig. 1. The K-map is shown in Fig. 2.

Because graphical methods become quite clumsy beyond a few variables and they are problematic to to create procedurally, we can rely on minimizing SOPs using the Quine-McCluskey method[6]. Essentially the method involves grouping the conjuncts as small material implications called "implicants." The consequent is 1, so one need only consider the antecedent. Then iterating over this group, a new implicant is added if there are two other implicants that differ by a single Boolean value. The new implicant is identical to the other two, except a "don't care" symbol * is placed at the point where they differ. When this iterative is complete, the next phase is to judiciously pick so-called "prime implicants"–implicants that cannot be combined with other implicants–so that the original conjuncts are covered.

From this, we were inspired to create a logic-theoretic classifier called *circle* (named for the task). We then created a full implementation, freely available, and are currently in the process of adding optimizations. We have begun testing Circle on both synthetic datasets and bioinformatic data that we are currently working with.

B_1	B_2	B_3	B_4	f
1	1	1	0	1
1	1	0	0	1
1	0	1	1	1
1	0	1	0	1
1	0	0	0	1
0	0	0	1	1
0	0	0	0	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1

Figure 1: A Boolean function (classifier) f.

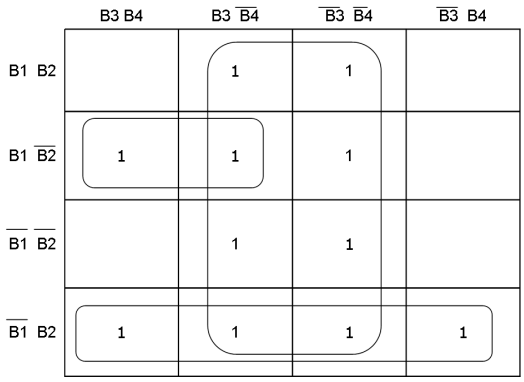

Figure 2: A K-map for the boolean function in Figure 1

2 Circle via c-maps

We will first introduce the Circle classifier graphically. As mentioned in the introduction, having been inspired by work in Boolean circuit minimization, we thought it natural and appealing to present Circle similarly. We will be using the relation **History** in Fig. 3. These values represent patient histories at a fictitious hospital. We are interested in the `Prognosis` of the patient, given his or her `Treatment`, `Symptoms`, `Age`, and `Sex`. The `PID` is nothing more than an identifier to make discussion easier. We create a grid called a classifier-map or c-map. A c-map is shown in Fig. 6(a). After creating this collection of cells, each tuple is placed somewhere on the c-map. Once this is done, the $\emptyset$ is placed in the remaining unoccupied cells. This has been done in Fig. 6(a). Our task is then to *circle* the largest collection of cells that contain the same kind of class values, incorporating any cells with $\emptyset$. A few conditions must be adhered to during the circling: (i) the circled cells must form a rectangle; (ii) they must contain the same kind of class values; (iii) the number of circled cells must be a power of two; and (iv) the circle can span opposite edges.

In Fig. 6(b) we have shown the progressive correct circling, beginning with tuple t_{23} as a finely dotted line. Once we have completed circling, the next step is to describe the circles via the essential indices. These are attribute values that index the circle using positively or negatively, but not both. In the case of the first circle we found, we see that $\overline{\texttt{X}}$ and $\overline{\texttt{male}}$ are essential. Conjoining these two negative values gives the rule $\texttt{Treatment} \neq \texttt{X} \ \wedge \ \texttt{Sex} \neq \texttt{male} \Rightarrow \texttt{sick}$. The entire set of rules found by circling is shown below:

$$\begin{aligned} \texttt{Treatment} = \texttt{X} &\Rightarrow \texttt{cured} \\ \texttt{Treatment} \neq \texttt{X} \ \wedge \ \texttt{Symptom} \neq \texttt{cough} &\Rightarrow \texttt{sick} \\ \texttt{Treatment} \neq \texttt{X} \ \wedge \ \texttt{Sex} \neq \texttt{male} &\Rightarrow \texttt{sick} \\ \texttt{Age} \neq \texttt{young} \ \wedge \texttt{Sex} = \texttt{male} &\Rightarrow \texttt{cured} \end{aligned}$$

PID	Trtmt	Symptoms	Age	Sex	Prognosis
13	X	cough	young	male	cured
24	X	fever	young	male	cured
12	X	cough	adult	female	cured
26	Y	cough	old	male	cured
23	Y	cough	adult	female	sick
52	X	sneeze	adult	male	cured
2	W	sneeze	young	female	sick
5	Z	sneeze	young	male	sick
6	Z	fever	young	male	sick

Figure 3: A relation **History** of patient data from a hospital. PID is the patient ID and serves merely as a tuple ID. The feature attributes are $\{\texttt{Treatment}, \texttt{Symptoms}, \texttt{Age}, \texttt{Sex}\}$. The classifier is `Prognosis`.

Suppose there is an additional tuple $t_{99} =$ $(\texttt{Z}, \texttt{cough}, \texttt{adult}, \texttt{male}, \texttt{sick})$. Using our original c-map, we can placed it and the other nine tuples along with $\emptyset$ as shown in Fig. 6(d). Conjoining this information to the original circle gives the rules:

$$\begin{aligned} \texttt{Age} \neq \texttt{young} \ \wedge \ \texttt{Sex} = \texttt{male} \ \wedge \ \texttt{Age} \neq \texttt{old} &\Rightarrow \texttt{sick} \\ \texttt{Age} = \texttt{old} \ \wedge \ \texttt{Sex} = \texttt{male} &\Rightarrow \texttt{cured} \end{aligned}$$

3 Foundations

We assume a relation $\mathbf{r}$ over a set of attributes R. From R we distinguish a single attribute C as the classifier and the remaining attributes $\{A_1, A_2, \ldots, A_n\}$ as features. Thus, $R = \{A_1, \ldots, A_n\} \cup \{C\}$. C contains the class values (classes), and the remaining attributes are the means of achieving the classification. For the initial portion of the paper, we will use the relation `History` in Fig.3. The feature attributes are $\{\texttt{Treatment}, \texttt{Symptoms}, \texttt{Age}, \texttt{Sex}\}$. The classifier is `Prognosis`. The *active domain* of an attribute is

the domain of existing values in the relation. For example, the active domain of `Treatment` in the relation **History** is $\{\mathtt{X}, \mathtt{Y}, \mathtt{W}, \mathtt{Z}\}$. To make discussion easier, we assume the attributes are ordered, and so, when we refer to the i^{th} attribute, we mean A_i. Given a tuple $t \in \mathbf{r}$, we will write $t.A_i$ to mean the value of t at the attribute A_i. In this initial exposition, we will assume that all values are categorical. It should be pointed out that selection of C is, in some sense, arbitrary, reflecting more the interest of the user than any inherent value of C. The task is to arrive at some function $f : A_1 \times A_2 \times \cdots A_n \rightarrow C$ that correctly classifies any tuple (instance) $t \in \mathbf{r}$; that is, we want to find f that makes the following implication true:
$\forall t\ t \in \mathbf{r}\ \Rightarrow f(t.A_1, t.A_2, \ldots, t.A_n) = t.C$

Definition 1 A **minterm** is a non-empty word of fixed length over the alphabet $\{0, 1, *\}$. We will assume the length to be n, the number of feature attributes. Using the relation **History** in Fig.3, for example, $110*$ is a minterm. We provide a subscript $i \geq 1$ of the word to denote the i^{th} symbol in a minterm. For example, if $\alpha = 110*$, then α_2 means 1. □

Definition 2 Two minterms α and β are **logically adjacent** when they differ by exactly one symbol at the i^{th} position and either $\alpha_i = 1$ and $\beta_i = 0$ or $\alpha_i = 0$ and $\beta_i = 1$. For example, $10 * 1$ is logically adjacent to $00 * 1$, but not adjacent to $00 * 0$. □

Definition 3 An **implicant** is a valuation of feature attributes together with the set of tuples that are determined. The valuation depends upon an arbitrary, but fixed tuple $t \in \mathbf{r}$ and a minterm α. Each of the n symbols in the minterm uniquely corresponds to a feature attribute, *e.g.* the first symbol α_1 corresponds to A_1, the second α_2 to A_2, *etc.* We achieve a valuation of feature attributes by associating a tuple $t \in \mathbf{r}$ and minterm α as a pair (t, α) in the following way.

- If $\alpha_i = 1$, then the corresponding attribute value is $t.A_i$;
- If $\alpha_i = 0$, then the corresponding value is any value from the active domain except $t.A_i$;
- If $\alpha_i = *$, then the corresponding value is any value from the active domain.

For example, $(t_{13}, \mathtt{10} * \mathtt{1})$ means

$$\mathtt{Treatment} \in \{\mathtt{X}\},\ \mathtt{Symptoms} \notin \{\mathtt{cough}\},$$
$$\mathtt{Age} \in \{\mathtt{young}, \mathtt{old}, \mathtt{adult}\},\ \mathtt{Sex} \in \{\mathtt{male}\}$$

A tuple–minterm pair (t, α) can be used to determine a tuple $s \in \mathbf{r}$, denoted $\alpha \vdash_t s$ when $s.A_i = t.A_i$ if $\alpha_i = 1$, or $s.A_i \neq t.A_i$ if $\alpha_i = 0$. For example, using the relation **History**, $\mathtt{110*} \vdash_{t_{13}} t_{12}$. An implicant is a triple $(t, \alpha, \{s \in \mathbf{r} | \alpha \vdash_t s\})$. For example, $(t_{13}, \mathtt{110*}, \{t_{12}\})$ □

An implicant is essentially material implication (see 3), $X \Rightarrow Y$. X denotes the valuation of inputs and Y the valuation of classes.

Lemma 4 Given a relation $\mathbf{r}$ and an arbitrary, but fixed tuple $t \in \mathbf{r}$, form the set

$$\mathcal{M} = \{(t, \alpha, \{s \in \mathbf{r} | \alpha \vdash_t s\})\}$$

where the minterm α is word in $\{0, 1\}^n$. Any tuple $s \in \mathbf{r}$ belongs to one and only one implicant.
Proof Assume the contrary, that s belongs to two different implicants. The implicants' minterms must disagree on at least one symbol at position i. Then $s.A_i$ will either agree with $t.A_i$ and s will belong to the minterm with $\alpha_i = 1$, or the converse; but not both.

□

Definition 5 Two implicants (t, α, s) and (t', α', s') are **consistent** if they agree on the tuple and either they determine the set of classifier values or at least one of the set s, s' is empty. Formally, $t = t'$ and either $(\{s.C | (t, \alpha, s)\} = \{s.C' | (t', \alpha', s')\}$ or $s = \emptyset$. For example, $(t_{12}, 1000, \emptyset)$ and $(t_{12}, 1001, \{t_{24}\})$ are consistent, whereas, $(t_{12}, 0100, \{t_{23}\})$ and $(t_{12}, 0101, \{t_{26}\})$ are not. □

Definition 6 A **cover** is a collection of implicants that are consistent. An *unambiguous* cover is a collection of implicants that have exactly one classifier value. □

Definition 7 A **prime implicant** is an implicant that is not logically adjacent to another implicant that has a weaker minterm. An implicant that uniquely covers one or more tuples is *essential*, otherwise it is *redundant*. □

We will now formally describe the implicant production. Given two logically adjacent and consistent implicants (t, α, s) and (t', α', s'), that differ in their respective minterms at position i, we can produce a weaker, consistent implicant
$(t, \alpha_1 \alpha_2 \cdots \alpha_{i-1} * \alpha_{i+1} \cdots \alpha_n, s \cup s')$.

Circle's objective is to determine the smallest subset of prime implicants that will cover the classes. Circle achieves this by a levelwise algorithm that iteratively produces weaker, though consistent, implicants. After the production of implicants is exhausted, the prime implicants are collected. This set of prime implicants contains both essential and redundant prime implicants. The algorithm ends by selecting prime implicants to cover the remaining tuples not covered by the essential prime implicants. In the case that a cover is ambiguous, Circle is recursively applied to those tuples belonging to the cover.

```
Treatment = X: cured
Treatment = Y
|  Age = young: null
|  Age = old: cured
|  Age = adult: sick
Treatment = Z: sick
Treatment = W: sick
```

Figure 4: Classifiers produced by the ID3 algorithm

4 Models

From our previous discuss, we have seen that Circle initially produces a collection of rules that potentially contain positive as well as negative features. This rule set is actually non-monotonic with respect to adding tuples to the relation. Non-monotonicity provides a powerful means of reasoning, in Circle's case as a classifier. While an in depth discussion is not appropriate here, we would like to point a significant problem that monotonic classifiers face. The feature values must be known *apriori*; encountering a heretofore unknown value might result in the classifier either having nothing important to say or, worse yet, simply breaking. Circle, by virtue of Lemma 4 will find a rule class that holds true.

Lemma 8 Suppose Circle produces a rule set $\mathcal{R} = \{X_1 \Rightarrow Y_1, \ldots, X_k \Rightarrow Y_k\}$ over some relation $\mathbf{r}$ with features
$\{A_1, A_2, \ldots, A_n\}$ and classifier C. A new instance s is presented that has at least one new feature value not present in the active domain of at least one attribute. Then only one class of the rules will hold true.
PROOF The rules were created using a sequence of tuples $t_1, \ldots, t_\ell$. The sequence of minterms that uniquely determine s is found by comparing the individual attribute values of t_i to s. For example, the first minterm for tuple t_1 is found by iterating over the n different attributes, concatenating 1 if $t.A_i = s.A_i$ and 0 otherwise. By Lemma 4, s can belong in only one implicant. Because of logically adjacency and consistency, s will remain in a single class as weaker implicants are produced. □

We can move toward a monotonic rule set by treating the negative features as relative complement. We call this rule set "mid-monotonic" for lack of a better term. For the original rule set above we have

R'_1 $\quad$ $\texttt{Treatment} = \texttt{X} \Rightarrow \texttt{cured}$

R'_2 $\quad$ $\texttt{Treatment} \in \{\texttt{Y}, \texttt{W}, \texttt{Z}\} \wedge \texttt{Symptom} \in \{\texttt{fever}, \texttt{sneeze}\} \Rightarrow \texttt{sick}$

R'_3 $\quad$ $\texttt{Treatment} \in \{\texttt{Y}, \texttt{W}, \texttt{Z}\} \wedge \texttt{Sex} = \texttt{f} \Rightarrow \texttt{sick}$

R'_4 $\quad$ $\texttt{Age} \in \{\texttt{old}, \texttt{adult}\} \wedge \texttt{Sex} = \texttt{male} \Rightarrow \texttt{cured}$

What is interesting to observe is that though these rules capture all the instances, they are over-specified

```
 1. Treatment=X ==> Prognosis=cured
 2. Treatment=X Sex=m ==> Prognosis=cured
 3. Treatment=Z ==> Age=young Sex=m Prognosis=sick
 4. Treatment=Z Age=young ==> Sex=m Prognosis=sick
 5. Treatment=Z Sex=m ==> Age=young Prognosis=sick
 6. Treatment=Z Prognosis=sick ==> Age=young Sex=m
 7. Sex=m Prognosis=sick ==> Treatment=Z Age=young
 8. Treatment=Z Age=young Sex=m ==> Prognosis=sick
 9. Treatment=Z Age=young Prognosis=sick ==> Sex=m
10. Treatment=Z Sex=m Prognosis=sick ==> Age=young
```

Figure 5: Classifiers produced by the Apriori algorithm

and actually try to speak to instances that are not present. To complete our monotonic rule set, we simply verify which of the rules are supported by the instances. Continuing with our example,

R''_1 $\quad$ $\texttt{Treatment} = \texttt{X} \Rightarrow \texttt{cured}$

R''_2 $\quad$ $\texttt{Treatment} = \texttt{Z} \wedge \texttt{Symptom} \in \{\texttt{fever}, \texttt{sneeze}\} \Rightarrow \texttt{sick}$

R''_3 $\quad$ $\texttt{Treatment} = \{\texttt{W}, \texttt{Y}\} \wedge \texttt{Sex} = \texttt{f} \Rightarrow \texttt{sick}$

R''_4 $\quad$ $\texttt{Age} \in \{\texttt{old}, \texttt{adult}\} \wedge \texttt{Sex} = \texttt{male} \Rightarrow \texttt{cured}$

Although not appropriate to discuss fully here, we believe each of these different kinds of rule sets have something potentially interesting and important to say about the instances. The last rule set is, of course, monotonic with the addition of new instances, providing that ambiguity is not created. We thought it interesting to juxtapose Circle's sets of classifiers to those generated by ID3[7] a decision tree based on information gain (entropy) and association rules[8]. Observe there are some differences, not the least of which is presentation. Certainly how the classifier is to be used would determine which, if any, are deemed best. In our opinion, using several classifiers is probably the best approach to most problems.

5 Circle Algorithm and Implementation

The design of Circle is based on a core algorithm that we call CoreCircle, which takes a relation R as the parameter, and returns a set of rules that cover R. As before, R has the structure $R < id, A_1, A_2, \ldots, A_k, A_c >$, where each row has an identifying attribute, k attributes on which clustering is to be created, and the classifier attribute A_c. Figure 7 shows the Core Circle algorithm. For the algorithm, and implicant is of the form $(t, b_1 b_2 \ldots b_k, \mathbf{r}')$, where $t \in \mathbf{r}$, $b_1 b_2 \ldots b_k$ is a minterm where $b_i \in \{0, 1, *\}$, and $\mathbf{r}' \subseteq \mathbf{r}$. We assume for this algorithm that A and A_c are fixed and ordered.

The core Circle algorithm iteratively builds the implicant list by starting with the initial set of impli-

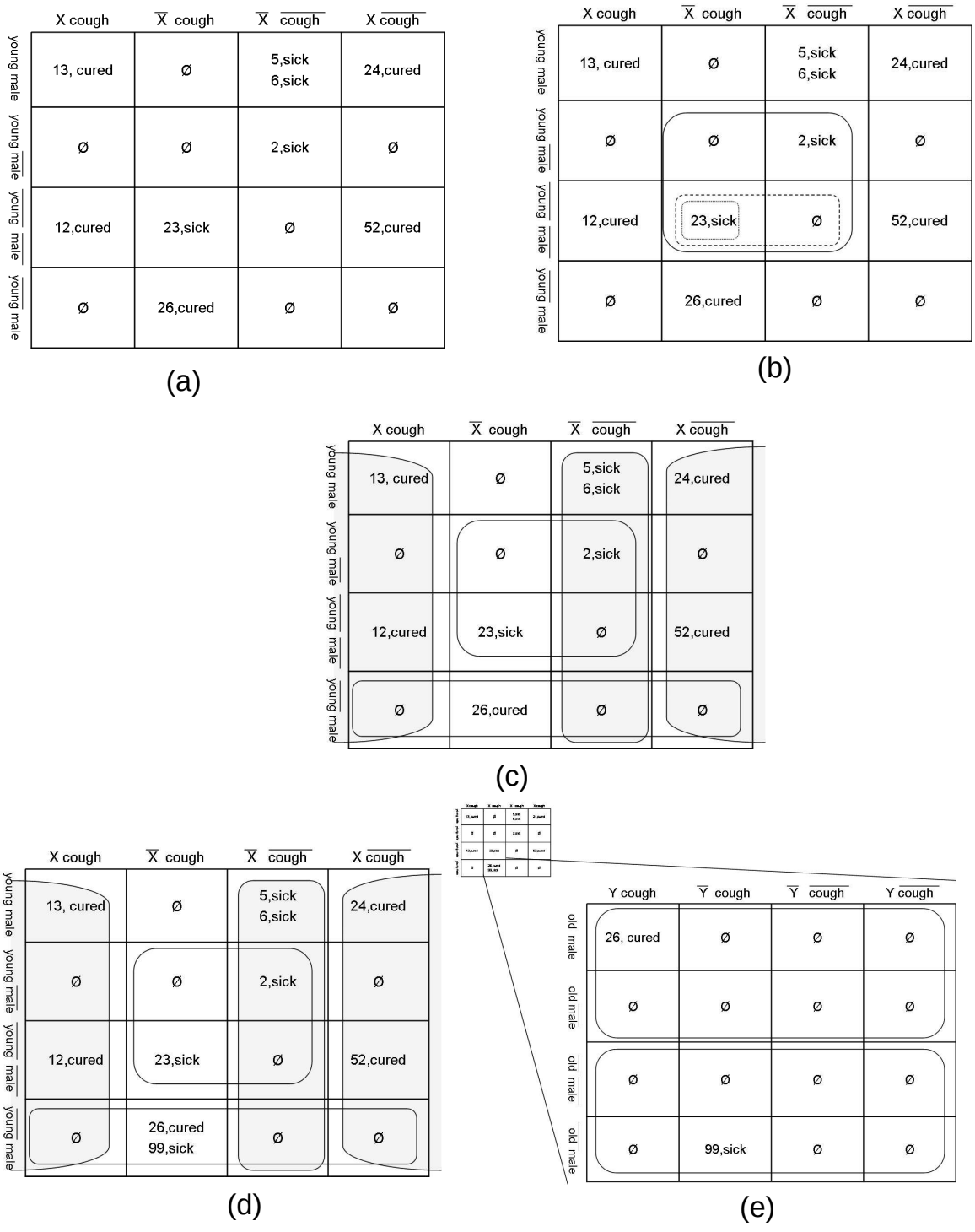

Figure 6: Illustration of the Circle algorithm

```
CoreCircle(r)
  if size(r) < 2 or size(classes(r)) < 2
    MakeFullRule(r);
  else
    pick t in r;
    I[0] = MakeInitialImplicants(t, r);
    i = 0;
    while(I[i] != emptyset and i < k)
      I[i+1] = MakeNextImplicants(I[i]);
      i++;
    end while
    PI = MakePrimeImplicants(I);
    P = MakeEssentialImplicants(PI, r);
    for each (t, b1b2...bk, r') in P
      if size(r') < 2 or size(classes(r')) < 2
         MakeFullRule(t, b1b2...bk, r')
      else
         MakePartialRule((t, b1b2...bk, r') v
                              CoreCircle(r'))
      end if
    end for
  end if
end Circle
```

Figure 7: The Core Circle algorithm

cants, and combining them to create subsequent sets, until either no more implicants are generated, or the full depth of implicants have been generated. In this algorithm, the *MakeInitialImplicants* procedure generates the implicants from every row of r using the row t which is picked. The t row can be picked at random, although the actual implementation simply uses the first row in the relation. The *MakeNextImplicants* procedure is a simple loop which uses two iterators to go through a set, and for every pair of implicants found with the same set of classifiers and minterms differing only in one place, a new implicant is generated, with the particular bit of the minterm (where the difference was obtained) as '*' and marking the original two implicants as used. The *MakePrimeImplicants* procedure goes through this collection of sets I, and collects all the unused implicants.

The *MakeEssentialImplicants* algorithm selects a minimal set of essential implicants from the Prime implicants generated. A greedy algorithm is then used to generate the essential prime implicants. For each of the essential prime implicants created, a full rule can be created for every implicant for which the set of classifiers is 1. For all the implicants with a set of classifiers more than 1, CoreCircle is recursively called with the tuples covered by such implicants.

Circle is implemented in Java, configurable to use any JDBC-compliant database in the backend (or ODBC compliant databases using Sun's JDBC-ODBC bridge). The system was tested with Microsoft Access, Microsoft SQL server, and Oracle. The current implementation is configurable through a configuration file which could be specified from the command line, or set up using a simple web interface[1]. Development of a fully configurable web-based application is part of the future enhancements.

6 Summary and Conclusion

We have begun work on creating a classifier whose inspiration is drawn from Boolean function minimization. The classifier called Circle possesses a number of attractive properties, *e.g.*, non-monotonicity. On the initial synthetic datasets we examined, Circle performed quite well. We are also encouraged by preliminary success on the bioinformatics data that drove all of this in the first place. A number of serious limitations in terms of complexity still need to be addressed. We have opted for an easy first-solution, probabilistically employing Circle. There are, no doubt, other techniques that can be adopted, by smartly guiding Circle's classification through the use of information theory. There is a lot work to do on the rule sets themselves, perhaps mining them or adding support and confidence as is done with association rules. We are also interested in applying Circle to streams, where work has begun in developing classifiers and clustering. Circle can likely be changed slightly to cluster. One can imagine agglomeration based on logical adjacency. We would like to thank Sun Kim for helpful comments.

References

[1] I. H. Witten and E. Frank, *Data Mining - Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[2] T. M. Mitchell, *Machine Learning*. McGraw Hill, 1997.

[3] S.L.Salzberg, "On comparing classifiers: Pitfalls to avoid and a recommended approach," *Data Mining and Knowledge Discovery*, vol. 1, pp. 317–328, 1997.

[4] Z. Wang, M. Dalkilic, and S. Kim, "Guiding motif discovery by iterative pattern refinement," in *ACM SAC BioInformatics Track*, 2004.

[5] M. Karnaugh, "The map method for synthesis of combinatorial logic circuits," *Trans. AIEE. pt I*, vol. 72, pp. 593–599, November 1953.

[6] E. L. McCluskey Jr., "Minimization of boolean functions," *Bell System Technical Journal*, vol. 35, pp. 1417–1444, April 1959.

[7] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[8] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB-94)*, pp. 487–499, 2000.

[1] A binary version of the implementation is freely available http://www.kelley.iu.edu/sengupt/circle.