Chapter 2

Boolean Functions

2.1 Representation

2.1.1 Boolean Algebra

Many important ideas about learning of functions are most easily presented using the special case of Boolean functions. There are several important subclasses of Boolean functions that are used as hypothesis classes for function learning. Therefore, we digress in this chapter to present a review of Boolean functions and their properties. (For a more thorough treatment see, for example, [Unger, 1989].)

A Boolean function, $f(x_1, x_2, \ldots, x_n)$ maps an n-tuple of (0,1) values to $\{0, 1\}$. *Boolean algebra* is a convenient notation for representing Boolean functions. Boolean algebra uses the connectives $\cdot$, $+$, and $\bar{}$. For example, the *and* function of two variables is written $x_1 \cdot x_2$. By convention, the connective, "$\cdot$" is usually suppressed, and the *and* function is written $x_1 x_2$. $x_1 x_2$ has value 1 if and only if *both* x_1 and x_2 have value 1; if either x_1 or x_2 has value 0, $x_1 x_2$ has value 0. The (inclusive) *or* function of two variables is written $x_1 + x_2$. $x_1 + x_2$ has value 1 if and only if either or both of x_1 or x_2 has value 1; if both x_1 and x_2 have value 0, $x_1 + x_2$ has value 0. The *complement* or *negation* of a variable, x, is written $\overline{x}$. $\overline{x}$ has value 1 if and only if x has value 0; if x has value 1 has value 1, $\overline{x}$ has value 0.

These definitions are compactly given by the following rules for Boolean algebra:

$1 + 1 = 1$, $1 + 0 = 1$, $0 + 0 = 0$,

$1 \cdot 1 = 1$, $1 \cdot 0 = 0$, $0 \cdot 0 = 0$, and

$\overline{1} = 0$, $\overline{0} = 1$.

Sometimes the arguments and values of Boolean functions are expressed in terms of the constants T (*True*) and F (*False*) instead of 1 and 0, respectively.

The connectives $\cdot$ and $+$ are each commutative and associative. Thus, for example, $x_1(x_2x_3) = (x_1x_2)x_3$, and both can be written simply as $x_1x_2x_3$. Similarly for $+$.

A Boolean formula consisting of a single variable, such as x_1 is called an *atom*. One consisting of either a single variable or its complement, such as $\overline{x_1}$, is called a *literal*.

The operators $\cdot$ and $+$ do not commute between themselves. Instead, we have DeMorgan's laws (which can be verified by using the above definitions):

$\overline{x_1x_2} = \overline{x_1} + \overline{x_2}$, and
$\overline{x_1 + x_2} = \overline{x_1}\ \overline{x_2}$.

2.1.2 Diagrammatic Representations

We saw in the last chapter that a Boolean function could be represented by labeling the vertices of a cube. For a function of n variables, we would need an n-dimensional *hypercube*. In Fig. 2.1 we show some 2- and 3-dimensional examples. Vertices having value 1 are labeled with a small square, and vertices having value 0 are labeled with a small circle.

Using the hypercube representations, it is easy to see how many Boolean functions of n dimensions there are. A 3-dimensional cube has $2^3 = 8$ vertices, and each may be labeled in two different ways; thus there are $2^{(2^3)} = 256$ different Boolean functions of 3 variables. In general, there are 2^{2^n} Boolean functions of n variables.

We will be using 2- and 3-dimensional cubes later to provide some intuition about the properties of certain Boolean functions. Of course, we cannot visualize hypercubes (for $n > 3$), and there are many surprising properties of higher dimensional spaces, so we must be careful in using intuitions gained in low dimensions. One diagrammatic technique for dimensions slightly higher than 3 is the *Karnaugh map*. A Karnaugh map is an array of values of a Boolean function in which the horizontal rows are indexed by the values of some of the variables and the vertical columns are indexed by the rest. The rows and columns are arranged in such a way that entries that are adjacent in the map correspond to vertices that are adjacent in the hypercube representation. We show an example of the 4-dimensional even parity function in Fig. 2.2. (An *even parity function* is

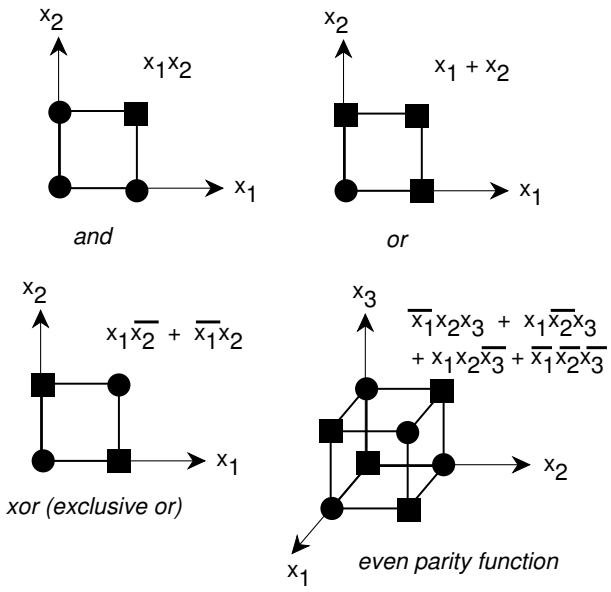

Figure 2.1: Representing Boolean Functions on Cubes

a Boolean function that has value 1 if there are an even number of its arguments that have value 1; otherwise it has value 0.) Note that all adjacent cells in the table correspond to inputs differing in only one component.

Also describe *general logic diagrams*, [Wnek, *et al.*, 1990].

2.2 Classes of Boolean Functions

2.2.1 Terms and Clauses

To use absolute bias in machine learning, we limit the class of hypotheses. In learning Boolean functions, we frequently use some of the common subclasses of those functions. Therefore, it will be important to know about these subclasses.

One basic subclass is called *terms*. A term is any function written in the form $l_1 l_2 \cdots l_k$, where the l_i are literals. Such a form is called a *conjunction* of literals. Some example terms are $x_1 x_7$ and $x_1 x_2 \overline{x_4}$. The *size* of a term is the number of literals it contains. The examples are of sizes 2 and 3, respectively. (Strictly speaking, the *class* of conjunctions of literals

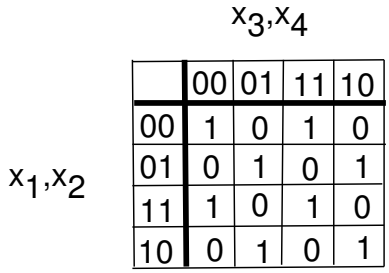

Figure 2.2: A Karnaugh Map

is called the *monomials*, and a conjunction of literals itself is called a *term*. This distinction is a fine one which we elect to blur here.)

It is easy to show that there are exactly 3^n possible terms of n variables. The number of terms of size k or less is bounded from above by $\sum_{i=0}^{k} C(2n, i) = O(n^k)$, where $C(i, j) = \frac{i!}{(i-j)!j!}$ is the binomial coefficient.

Probably I'll put in a simple term-learning algorithm here—so we can get started on learning! Also for DNF functions and decision lists—as they are defined in the next few pages.

A *clause* is any function written in the form $l_1 + l_2 + \cdots + l_k$, where the l_i are literals. Such a form is called a *disjunction* of literals. Some example clauses are $x_3 + x_5 + x_6$ and $x_1 + \overline{x_4}$. The *size* of a clause is the number of literals it contains. There are 3^n possible clauses and fewer than $\sum_{i=0}^{k} C(2n, i)$ clauses of size k or less. If f is a term, then (by De Morgan's laws) $\overline{f}$ is a clause, and vice versa. Thus, terms and clauses are duals of each other.

In psychological experiments, conjunctions of literals seem easier for humans to learn than disjunctions of literals.

2.2.2 DNF Functions

A Boolean function is said to be in *disjunctive normal form (DNF)* if it can be written as a *disjunction* of terms. Some examples in DNF are: $f = x_1x_2 + x_2x_3x_4$ and $f = x_1\overline{x_3} + \overline{x_2}\ \overline{x_3} + x_1x_2\overline{x_3}$. A DNF expression is called a k-term DNF expression if it is a disjunction of k terms; it is in the class k-DNF if the size of its largest term is k. The examples above are 2-term and 3-term expressions, respectively. Both expressions are in the class 3-DNF.

Each term in a DNF expression for a function is called an *implicant* because it "implies" the function (if the term has value 1, so does the

function). In general, a term, t, is an implicant of a function, f, if f has value 1 whenever t does. A term, t, is a *prime implicant* of f if the term, t', formed by taking any literal out of an implicant t is no longer an implicant of f. (The implicant cannot be "divided" by any term and remain an implicant.)

Thus, both $x_2\overline{x_3}$ and $\overline{x_1}\ \overline{x_3}$ are prime implicants of $f = x_2\overline{x_3} + \overline{x_1}\ \overline{x_3} + x_2x_1\overline{x_3}$, but $x_2x_1\overline{x_3}$ is not.

The relationship between implicants and prime implicants can be geometrically illustrated using the cube representation for Boolean functions. Consider, for example, the function $f = x_2\overline{x_3} + \overline{x_1}\ \overline{x_3} + x_2x_1\overline{x_3}$. We illustrate it in Fig. 2.3. Note that each of the three planes in the figure "cuts off" a group of vertices having value 1, but none cuts off any vertices having value 0. These planes are pictorial devices used to isolate certain lower dimensional *subfaces* of the cube. Two of them isolate one-dimensional *edges*, and the third isolates a zero-dimensional *vertex*. Each group of vertices on a subface corresponds to one of the implicants of the function, f, and thus each implicant corresponds to a subface of some dimension. A k-dimensional subface corresponds to an $(n-k)$-size implicant term. The function is written as the disjunction of the implicants—corresponding to the union of all the vertices cut off by all of the planes. Geometrically, an implicant is prime if and only if its corresponding subface is the largest dimensional subface that includes all of its vertices and no other vertices having value 0. Note that the term $x_2x_1\overline{x_3}$ is not a prime implicant of f. (In this case, we don't even have to include this term in the function because the vertex cut off by the plane corresponding to $x_2x_1\overline{x_3}$ is already cut off by the plane corresponding to $x_2\overline{x_3}$.) The other two implicants are prime because their corresponding subfaces cannot be expanded without including vertices having value 0.

Note that all Boolean functions can be represented in DNF—trivially by disjunctions of terms of size n where each term corresponds to one of the vertices whose value is 1. Whereas there are 2^{2^n} functions of n dimensions in DNF (since any Boolean function can be written in DNF), there are just $2^{O(n^k)}$ functions in k-DNF.

All Boolean functions can also be represented in DNF in which each term is a prime implicant, but that representation is not unique, as shown in Fig. 2.4.

If we can express a function in DNF form, we can use the *consensus* method to find an expression for the function in which each term is a prime implicant. The consensus method relies on two results:

We may replace this section with one describing the Quine-McCluskey method instead.

- Consensus:

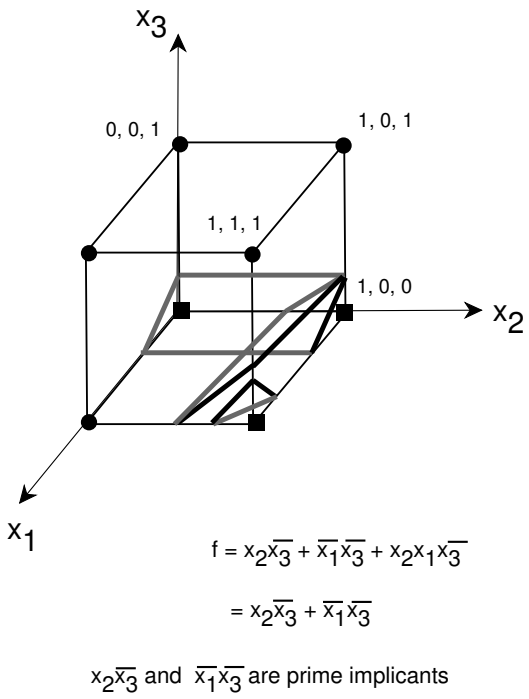

Figure 2.3: A Function and its Implicants

$$x_i \cdot f_1 + \overline{x_i} \cdot f_2 = x_i \cdot f_1 + \overline{x_i} \cdot f_2 + f_1 \cdot f_2$$

where f_1 and f_2 are terms such that no literal appearing in f_1 appears complemented in f_2. $f_1 \cdot f_2$ is called the *consensus* of $x_i \cdot f_1$ and $\overline{x_i} \cdot f_2$. Readers familiar with the *resolution* rule of inference will note that consensus is the dual of resolution.

Examples: x_1 is the consensus of $x_1 x_2$ and $x_1 \overline{x_2}$. The terms $\overline{x_1} x_2$ and $x_1 \overline{x_2}$ have no consensus since each term has more than one literal appearing complemented in the other.

- Subsumption:

$$x_i \cdot f_1 + f_1 = f_1$$

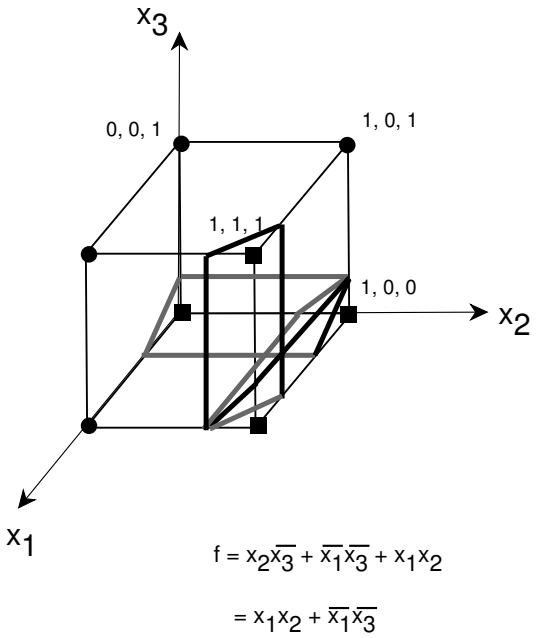

Figure 2.4: Non-Uniqueness of Representation by Prime Implicants

where f_1 is a term. We say that f_1 *subsumes* $x_i \cdot f_1$.

Example: $\overline{x_1}\ \overline{x_4}x_5$ subsumes $\overline{x_1}\ \overline{x_4}\ \overline{x_2}x_5$

The consensus method for finding a set of prime implicants for a function, f, iterates the following operations on the terms of a DNF expression for f until no more such operations can be applied:

1. initialize the process with the set, $\mathcal{T}$, of terms in the DNF expression of f,
2. compute the consensus of a pair of terms in $\mathcal{T}$ and add the result to $\mathcal{T}$,
3. eliminate any terms in $\mathcal{T}$ that are subsumed by other terms in $\mathcal{T}$.

When this process halts, the terms remaining in $\mathcal{T}$ are all prime implicants of f.

Example: Let $f = \overline{x_1}x_2 + \overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}x_5$. We show a derivation of a set of prime implicants in the *consensus tree* of Fig. 2.5. The circled numbers adjoining the terms indicate the order in which the consensus and subsumption operations were performed. Shaded boxes surrounding a term indicate that it was subsumed. The final form of the function in which all terms are prime implicants is: $f = \overline{x_1}x_2 + \overline{x_1}x_3 + \overline{x_1}\,\overline{x_4}x_5$. Its terms are all of the non-subsumed terms in the consensus tree.

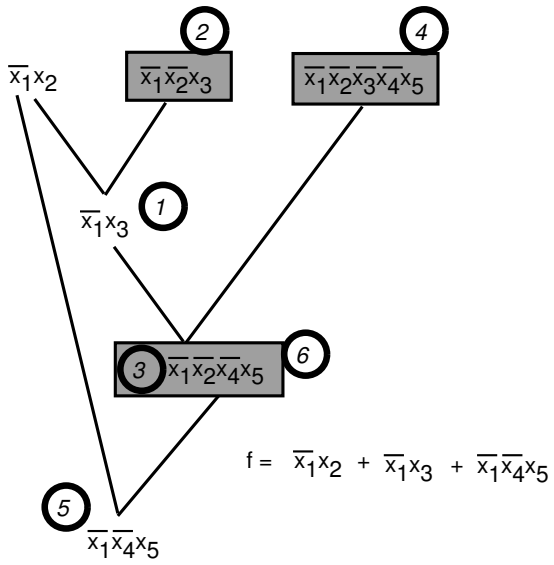

Figure 2.5: A Consensus Tree

2.2.3 CNF Functions

Disjunctive normal form has a dual: *conjunctive normal form (CNF)*. A Boolean function is said to be in CNF if it can be written as a *conjunction* of clauses. An example in CNF is: $f = (x_1 + x_2)(x_2 + x_3 + x_4)$. A CNF expression is called a k-clause CNF expression if it is a conjunction of k clauses; it is in the class k-CNF if the size of its largest clause is k. The example is a 2-clause expression in 3-CNF. If f is written in DNF, an

application of De Morgan's law renders $\overline{f}$ in CNF, and vice versa. Because CNF and DNF are duals, there are also $2^{O(n^k)}$ functions in k-CNF.

2.2.4 Decision Lists

Rivest has proposed a class of Boolean functions called *decision lists* [Rivest, 1987]. A decision list is written as an ordered list of pairs:

(t_q, v_q)

(t_{q-1}, v_{q-1})

...

(t_i, v_i)

...

(t_2, v_2)

(T, v_1)

where the v_i are either 0 or 1, the t_i are terms in $(x_1, \ldots, x_n)$, and T is a term whose value is 1 (regardless of the values of the x_i). The value of a decision list is the value of v_i for the first t_i in the list that has value 1. (At least one t_i will have value 1, because the last one does; v_1 can be regarded as a *default* value of the decision list.) The decision list is of *size* k, if the size of the largest term in it is k. The class of decision lists of size k or less is called k-DL.

An example decision list is:

$f =$

$(\overline{x_1}x_2, 1)$

$(\overline{x_1}\ \overline{x_2}x_3, 0)$

$\overline{x_2}x_3, 1)$

$(1, 0)$

f has value 0 for $x_1 = 0$, $x_2 = 0$, and $x_3 = 1$. It has value 1 for $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$. This function is in 3-DL.

It has been shown that the class k-DL is a strict superset of the union of k-DNF and k-CNF. There are $2^{O[n^k k \log(n)]}$ functions in k-DL [Rivest, 1987].

Interesting generalizations of decision lists use other Boolean functions in place of the terms, t_i. For example we might use linearly separable functions in place of the t_i (see below and [Marchand & Golea, 1993]).

2.2.5 Symmetric and Voting Functions

A Boolean function is called *symmetric* if it is invariant under permutations of the input variables. For example, any function that is dependent only on the number of input variables whose values are 1 is a symmetric function. The *parity* functions, which have value 1 depending on whether or not the number of input variables with value 1 is even or odd is a symmetric function. (The *exclusive or* function, illustrated in Fig. 2.1, is an odd-parity function of two dimensions. The *or* and *and* functions of two dimensions are also symmetric.)

An important subclass of the symmetric functions is the class of *voting functions* (also called m-of-n functions). A k-voting function has value 1 if and only if k or more of its n inputs has value 1. If $k = 1$, a voting function is the same as an n-sized clause; if $k = n$, a voting function is the same as an n-sized term; if $k = (n + 1)/2$ for n odd or $k = 1 + n/2$ for n even, we have the *majority* function.

2.2.6 Linearly Separable Functions

The linearly separable functions are those that can be expressed as follows:

$$f = \text{thresh}(\sum_{i=1}^{n} w_i x_i, \theta)$$

where w_i, $i = 1, \ldots, n$, are real-valued numbers called *weights*, θ is a real-valued number called the *threshold*, and $\text{thresh}(\sigma, \theta)$ is 1 if $\sigma \geq \theta$ and 0 otherwise. (Note that the concept of linearly separable functions can be extended to non-Boolean inputs.) The k-voting functions are all members of the class of linearly separable functions in which the weights all have unit value and the threshold depends on k. Thus, terms and clauses are special cases of linearly separable functions.

A convenient way to write linearly separable functions uses vector notation:

$$f = \text{thresh}(\mathbf{X} \cdot \mathbf{W}, \theta)$$

where $\mathbf{X} = (x_1, \ldots, x_n)$ is an n-dimensional vector of input variables, $\mathbf{W} = (w_1, \ldots, w_n)$ is an n-dimensional vector of weight values, and $\mathbf{X} \cdot \mathbf{W}$ is the *dot* (or *inner*) product of the two vectors. Input vectors for which f has value 1 lie in a half-space on one side of (and on) a hyperplane whose orientation is normal to $\mathbf{W}$ and whose position (with respect to the origin)

is determined by θ. We saw an example of such a separating plane in Fig. 1.6. With this idea in mind, it is easy to see that two of the functions in Fig. 2.1 are linearly separable, while two are not. Also note that the terms in Figs. 2.3 and 2.4 are linearly separable functions as evidenced by the separating planes shown.

There is no closed-form expression for the number of linearly separable functions of n dimensions, but the following table gives the numbers for n up to 6.

n	Boolean Functions	Linearly Separable Functions
1	4	4
2	16	14
3	256	104
4	65,536	1,882
5	$\approx 4.3 \times 10^9$	94,572
6	$\approx 1.8 \times 10^{19}$	15,028,134

[Muroga, 1971] has shown that (for $n > 1$) there are no more than 2^{n^2} linearly separable functions of n dimensions. (See also [Winder, 1961, Winder, 1962].)

2.3 Summary

The diagram in Fig. 2.6 shows some of the set inclusions of the classes of Boolean functions that we have considered. We will be confronting these classes again in later chapters.

The sizes of the various classes are given in the following table (adapted from [Dietterich, 1990, page 262]):

Class	Size of Class
terms	3^n
clauses	3^n
k-term DNF	$2^{O(kn)}$
k-clause CNF	$2^{O(kn)}$
k-DNF	$2^{O(n^k)}$
k-CNF	$2^{O(n^k)}$
k-DL	$2^{O[n^k k \log(n)]}$
lin sep	$2^{O(n^2)}$
DNF	2^{2^n}

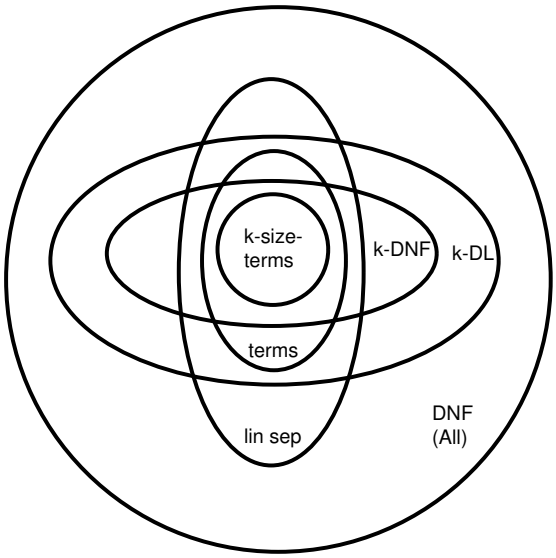

Figure 2.6: Classes of Boolean Functions

2.4 Bibliographical and Historical Remarks

To be added.