Average Case Analysis of Learning *k*-CNF Concepts

Daniel S. Hirschberg
dan@ics.uci.edu

Michael J. Pazzani
pazzani@ics.uci.edu

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717 USA

***Proceedings of the Tenth International Conference on Machine Learning* (pp. 206-211). Aberdeen, Scotland: Morgan Kaufmann.**

Abstract

We present an approach to modeling the average case behavior of an algorithm for learning Conjunctive Normal Form (CNF, i.e., conjunctions of disjunctions). Our motivation is to predict the expected error of the learning algorithm as a function of the number of training examples from a known distribution. We extend the basic model to address issues that arise if the data contain attribute noise. We show how the analysis can lead to insight into the behavior of the algorithm and the factors that affect the error. We make certain independence assumptions during the derivation of the average case model and we demonstrate that the predictions of the model account for a large percentage of the variation in error when these assumptions are violated.

1 INTRODUCTION

A goal of research in machine learning is to gain an understanding of the capabilities of learning algorithms. Pazzani and Sarrett (1990) introduced a framework for average case analysis of machine learning algorithms. Here, we show how this framework can be applied to create an average case model of an algorithm for learning monotone *k*-CNF concepts.

The framework attempts to unify the formal mathematical and the experimental approaches to understanding the behavior of machine learning algorithms. In order to achieve this unification, an average case model is needed since experiments lead to findings on the average accuracy of an algorithm. In contrast, the probably approximately correct (PAC) learning model (e.g., Valiant, 1984; Haussler, 1987; Haussler, Littlestone, & Warmuth, 1990; Haussler, 1990) is a worst-case model. Pazzani and Sarrett (in press) contains more detailed discussion, motivation, and comparisons between average case and PAC models.

2 AN AVERAGE CASE MODEL OF *k*-CNF

A restricted version of conjunctive normal form, *k*-CNF, provides a more expressive language of hypotheses than the language of pure conjunctions that we analyzed previously (Pazzani & Sarrett, 1990). Hypothesis in *k*-CNF can be expressed as conjunctions of disjunctions of length at most *k*. Pure conjunctive hypotheses can be viewed as a special form of *k*-CNF (i.e., *k*=1). For simplicity, we restrict our attentions to monotone *k*-CNF (i.e., *k*-CNF in which no feature is negated).

The goal of the average case model is to predict the expected error as a function of the number of training examples. The framework requires determining:

1. The conditions under which the algorithm changes the hypothesis for a concept.
2. How often these conditions occur.
3. How changing a hypothesis affects the accuracy of a hypothesis.

The first requirement of the average case model is that the algorithm be specified so that it is possible to determine when the hypothesis changes. Unlike the PAC model, an average case model requires creating a separate model for each algorithm. We will restrict our attention to the algorithm proposed in Valiant (1984) (see Table 1). This algorithm initializes the hypothesis to the most specific *k*-CNF concept and gradually makes the hypothesis more general by deleting disjunctive terms that are not consistent with positive training examples. For example, if there are 3 features (*a*, *b* and *c*), then the initial hypothesis for a monotone 3-CNF algorithm is:

a b c (a b) (a c) (b c) (a b c).

If the first training example is *a*, ¬*b*, ¬*c* then the hypothesis will be revised to

a (a b) (a c) (a b c).

The second requirement of the average case model presupposes information about the distribution of the training examples. Therefore, unlike the PAC model, the framework we have developed is not distribution-free. Furthermore, to reduce the amount of information required by the model, we will make certain independence assumptions. In particular, we will analyze the case in which the examples are drawn from a product distribution (i.e., the values of features are chosen independently).

The third requirement of the average case model presupposes information about the test examples and information about the correct hypothesis. As in the PAC model, we will assume that the distribution of the test examples is the same as the distribution of the training examples.

Table 1. The k-cnf learning algorithm

1. `Initialize the hypothesis to the conjunction of all disjunctions of at most length k of the features that describe training examples.`
2. `If the new example is a positive example, and the hypothesis misclassifies the new example, then remove all disjunctions from the hypothesis that are false in the`

2.1 AN AVERAGE CASE MODEL FOR *2*-CNF

We will first present an analysis of an algorithm that initializes the hypothesis to the set of conjunctions of disjunctions of exactly two positive variables. We will then generalize the model to the *k*-CNF case. We will use the following notation in describing the problem:

f_j the *j*-th feature of a training example. Here, we consider only Boolean features.

p_j the probability that the *j*-th feature has a true value. To reduce the amount of information required, we will assume that all p_j are independent.

m the total number of features used to describe training examples.

*D** the set of pairs (*i*,*j*) such that f_i f_j is part of the correct hypothesis.

*D**, *m*, p_j and f_j specify the information required in order to calculate the expected error of the learning algorithm as a function of *n*, the number of positive examples. Since the algorithm ignores negative examples, we will initially consider the case in which all of the training examples are positive. A minor extension requires calculating the probability that from *T* total examples, there are exactly *n* positive examples. We will use the following notation to describe the analysis of the algorithm.

D_0 the set of pairs of features that make up the terms conjoined to form the initial hypothesis: $\{(i,j) \mid (i\ <j)\}$.

D_n the set of pairs of features representing the hypothesis after *n* positive training examples.

I_n the set of pairs from D_n that are not in *D**. $I_n = D_n – D^*$.

S the set of all positive examples.

In order to calculate the error of a 2-CNF algorithm learning *D**, it is necessary to compute the probability that a randomly drawn example is positive and the probability that a randomly drawn positive example contains various combinations of features. We introduce the following notation:

P the probability that a randomly drawn example is a member of *S* under the assumption that values of each feature are determined independently with probability p_j that feature f_j has a true value.

P_i the probability that a randomly drawn example, *X*, is a member of *S* and the *i*-th feature of *X* has a true value. Similarly, $P_{\bar{i}}$ is the probability that *X* is a member of *S* and the *i*-th feature of *X* has a false value. Furthermore, P_{ij} is the probability that *X* is a member of *S* and the *i*-th and the *j*-th features of *X* both have true values. This notation, $P_{i...}$, generalizes to any number of subscripts and combinations of negated and unnegated subscripts.

We will derive the average case model by first indicating how to calculate *P* and $P_{i...}$. Next, we will show how to use *P* and $P_{i...}$ to determine the probability that any disjunctive term remains in the hypothesis. Finally, we compute the expected error by determining the probability that a disjunctive term in the hypothesis will result in the misclassification of a randomly drawn test example.

P and $P_{i...}$ are calculated in the following manner. Let *X* be a randomly drawn positive example. Define the weighted size of a set A of positive examples to be the probability that *X* is a member of set A. We will use |A| to denote *Pr*[*X* A]. If A = B C, where denotes union of disjoint sets, then |A| = |B| + |C|. We note that if A = B C, where B and C are not necessarily disjoint, then |A| = |B| + |$\overline{\text{B}}$ ∩ C|. Define {β(i,j...,)} to be the set of examples for which the Boolean expression β(i,j...,) on the indicated features is true. For example, {i $\bar{\text{j}}$} is the set of examples for which either f_i is true or f_j is false. Then, $|\{\beta_1\ \beta_2\}| = |\{\beta_1\}\ \ \{\neg\beta_1\beta_2\}| = |\{\beta_1\}| + |\{\neg\beta_1\beta_2\}|$.

$P = Pr[X \ \ _{(i,j)\ \ D^*} (f_i\ \ f_j)]$ we need to evaluate:

$$\left|\bigcap_{(i,j)\ \ D^*} \{i\ \ \ j\}\right| = \left|\bigcap_{(i,j)\ \ D^*} \{i\}\ \ \ \{\overline{i}j\}\right|$$

This intersection can be converted symbolically to the union of a number of disjoint sets. For example, to compute $Pr[X\ \ (f_{i_1}\ f_{j_1})\ (f_{i_2}\ f_{j_2})]$, the set is initialized to $\{i_1\}\ \ \{\bar{i}_1 j_1\}$. This set is intersected with $\{i_2\}\ \ \{\bar{i}_2 j_2\}$ to form: $\{i_1 i_2\}\ \ \{\bar{i}_1 j_1 i_2\}\ \ \{i_1 \bar{i}_2 j_2\}\ \ \{\bar{i}_1 j_1 \bar{i}_2 j_2\}$. This process can be repeated for each (*i*,*j*) *D**. Although in the worst case, this will result in the union of 2^c disjoint sets (where c is the cardinality of *D**), in practice it is substantially smaller due to cancellation of terms. For example, finding the intersection of all conjunctions of disjunctions of exactly 3 of 5 features requires computing the union of 10 disjoint sets instead of 2^{10} sets.

Once a CNF has been converted to this form, the probability that the CNF is true for a randomly drawn example can be found by summing the probabilities that it is a member of one of the disjoint sets described by a conjunction of features (or the inverses of features).

Table 2. Noise tolerate k-cnf learning algorithm

1. Initialize the hypothesis to the conjunction of all disjunctions of at most length k of the features that describe training examples. For each disjunction, initialize a counter to 0
2. If the new example is a positive example, and the hypothesis misclassifies the new example, then increment the counter associated with each disjunctions from the hypothesis that are false in the example.
3. When finished processing all N examples, remove those disjuncts whose counter is greater than εN.

Since we assume that individual features are independent, the probability that a conjunction of features can be found by multiplying the corresponding values of p_i. Computing P requires converting D* to this format and then summing the products of the probabilities. The computation of $P_{i...}$ is similar. The computation of $P_{i...}$ is similar. For example, $P_{a\overline{b}}$ is

$$\frac{Pr[\ X \quad \{a\overline{b}\} \ \&\ X \underset{(i,j) \quad D^*}{\quad} (f_i \quad f_j)\ ]}{Pr[X \underset{(i,j) \quad D^*}{\quad} (f_i \quad f_j)\ ]}$$

The k-CNF algorithm has only one operator to revise a hypothesis. A disjunctive term is dropped from the hypothesis when the term is false in a positive training example. This is modeled by removing pairs from I_{n-1} to form I_n. We will use the notation $r_d(n)$ to indicate the probability that the disjunction corresponding to $d = (i,j)$ remains in I_n. This occurs only if at least one of f_i and f_j has had a true value in all n positive training examples. Similarly, $r_{d_1 d_2}(n)$ indicates the probability that the disjunctions corresponding to d_1 and d_2 both remain in I_n.

Note that $r_{d_1}(n) = Pr[X \quad \{i_1 \quad j_1\}]^n$. In general:

$$r_{d_1,\ldots,d_h}(n) = Pr[X \quad \{(i_1 \quad j_1) \quad \cdots \quad (i_h \quad j_h)\ \}]^n. \qquad [1]$$

Note that misclassifying a positive example as a negative example is the only error made by the k-CNF algorithm. The hypothesis created by the k-CNF algorithm misclassifies a positive example if there is at least one pair (i,j) in I_n such that f_i and f_j are both false in the test example. We will use the notation e_n to represent the probability that an example is misclassified after n training examples. $e_n = e_n(1) - e_n(2) + e_n(3) - e_n(4)$ ... $e_n(h)$ where h is the cardinality of I_0 and $e_n(a)$ is the probability that an example is misclassified after n examples because it is contra-indicated by at least a of the pairs in I_0.

$$e_n(1) = \sum_{d = (i,j) \quad I_0} r_d(n) P_{\bar{i}\,\bar{j}}$$

$$e_n(a) = \sum_{\substack{d_1 \ldots d_a = (i_1,j_1) \ldots (i_a,j_a) \\ partitions(I_0,\,a)}} r_{d_1 \ldots d_a}(n) P_{\bar{i}_1 \bar{j}_1 \ldots \bar{i}_a \bar{j}_a} \qquad [2]$$

where *partitions(I,a)* is the set of all subsets of I with length a. $e_n(1)$ calculates the probability of getting an error from some learned hypothesis containing exactly one erroneous disjunction. $r_d(n)$ is the probability that a learned hypothesis containing the disjunction d survives n examples and $P_{\bar{i}\bar{j}}$ is the probability that the learned hypothesis containing $f_i \quad f_j$ makes an error. Thus $e_n(1)$ calculates the probability that each pair from I_0 is a pair of I_n weighted by the probability that a randomly drawn training example would be misclassified by the disjunction corresponding to that pair. For larger values of i, $e_n(i)$ corrects $e_n(i-1)$ by taking into consideration the fact that more than $i-1$ of the pairs in I_n result in a misclassification.

2.2 EXTENDING THE MODEL FOR *k*-CNF

The generalization of the model to k-CNF is fairly straightforward. First, D* is now a set of tuples of at most length k. Each tuple corresponds to a set of features corresponding to one of the terms of the correct hypothesis. D_0 (the initial hypothesis) is defined to be the set of all tuples of at most k of the features. From here, the notation of the previous section needs to be extended somewhat, but the technique for calculating P, $r(n)$, and $e(n)$ remains unchanged.

2.3 DEALING WITH NOISE IN TRAINING DATA

Valiant (1985) introduces a version of the k-DNF algorithm that allows it to tolerate noise in the training data. Table 2 adapts this algorithm to learning k-CNF concepts. The k-CNF algorithm removes a disjunction from the hypothesis if the disjunction is false in at least 1 positive example. The noise tolerant k-CNF algorithm removes a disjunction if the disjunction is false in at least εN positive examples, where ε is a parameter to the algorithm and N is the size of the training set. In order to create an average case model of the noise tolerant k-CNF algorithm, the probability that a disjunction in a hypothesis is modified from the (non-noise tolerant) k-CNF algorithm to take into account that it may be false in εN training examples (Equation 3).

Once this change has been made to the definition of $r_d(n)$ in Equation 1, then Equation 2, for calculating the expected error, holds for the noise tolerant k-CNF algorithm. Modifying the definition of $r_d(n)$ is all that is needed to model the noise tolerant k-CNF learning from noise-free data. If there is noise in the training data, it is also necessary to create a model of the noise in the training data. Here, we will illustrate this process by

$$r_{d_1,\ldots,d_h}(n) = \sum_{j = n\text{ - }ceiling(\varepsilon n)}^{n} \binom{n}{j} Pr[X \quad \{(i_1 \quad j_1) \quad \cdots \quad (i_h \quad j_h)\ \}]^j Pr[X \quad \{(i_1 \quad j_1) \quad \cdots \quad (i_h \quad j_h)\ \}]^{n-j} \qquad [3]$$

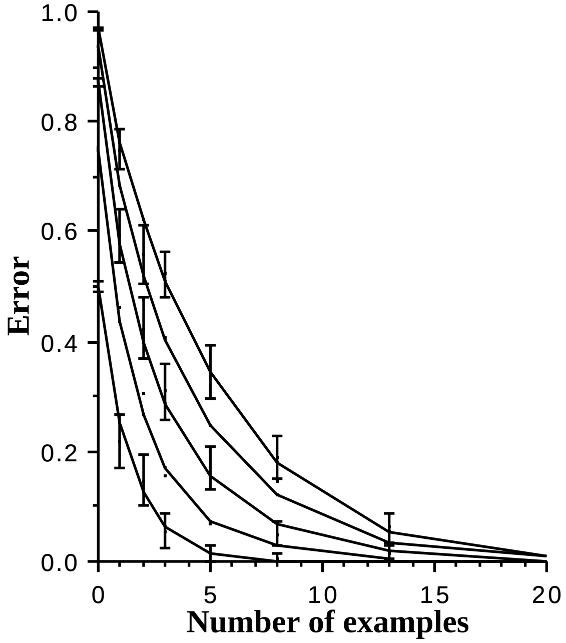


Figure 1. Expected and observed error when learning $i_1 \quad i_2$ with a 2-CNF algorithm when there are a total of 3 (lowest curve) 4, (next to lowest), 5 (middle), 6 (next to upper) and 7 features (upper curve). The curves represent predicted values and the bars are 95% confidence intervals around the mean values. To avoid clutter, confidence intervals are not shown for 4 and 6 total features.

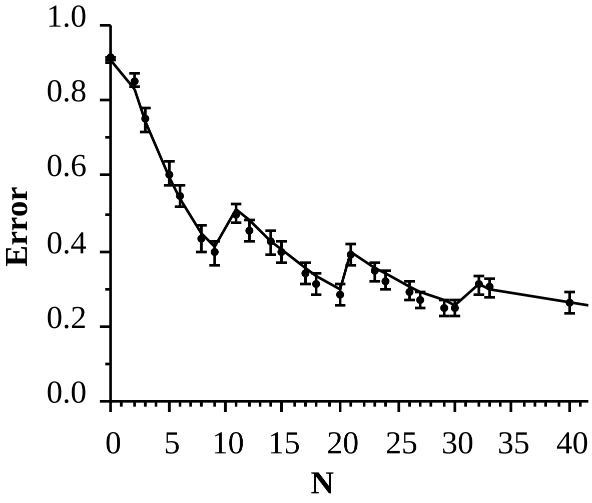


Figure 2. Expected and observed error (with a 95% confidence interval) for a noise tolerant 2-CNF algorithm.

modeling the effect of attribute noise (Quinlan, 1986). In attribute noise, there is a possibility that the value of a feature is replaced by a noisy value. In particular, we will assume that for each feature of a training or test example there is a fixed probability α, that value is replaced with a random value. The effect of this noise is that the probability $\hat{p}_i$ that a feature has a true value in an training or test example will differ from p_j. In particular, $\hat{p}_i = 0.5\alpha + (1\text{-}\alpha)p_i$. In order to calculate the probability that an example from a product distribution with attribute noise is misclassified, $\hat{p}_i$ replaces p_i in Equations 2 and 3. In Section 3.2, we illustrate how attribute noise affects the accuracy of the noise tolerant *k*-CNF algorithm.

3 IMPLICATIONS OF THE MODEL

In this section, we show how the average case model for *k*-CNF can be used to gain an understanding of some factors that affect the error of the learning algorithm. In particular, we address the following two problems:

1. The effect on the error of adding an irrelevant feature to all training examples.
2. The effect on the error of adding attribute noise to the training examples.

3.1 IRRELEVANT FEATURES

We consider a relatively simple problem to illustrate the effect of adding irrelevant features to the training data. Consider trying to learn the concept, $i_1 \quad i_2$ with a 2-CNF learning algorithm. If there are 3 features, then the 2-CNF algorithm will converge on the equivalent (but unsimplified) hypothesis: $i_1 \quad i_2 \quad (i_1 \quad i_2) \quad (i_1 \quad i_3) \quad (i_2 \quad i_3)$. The set I_0 will contain the singleton $\{(i_3)\}$. With 0 training examples, the initial hypothesis will produce an error on $(1{-}p_3)$ of the positive examples (since an error is made when i_3 is false and i_3 is false in this proportion of training examples). After *n* training examples, the hypothesis will produce an error on $(p_3)^n(1{-}p_3)$ of the positive training examples (since the probability that i_3 is in I_0 is $(p_3)^n$). Figure 1 (lower curve) graphs the observed and expected error under these conditions (with $p_3 = 0.5$).

If there are 4 features, then the 2-CNF algorithm will converge on the equivalent (but unsimplified) hypothesis: $i_1 \quad i_2 \quad (i_1 \quad i_2) \quad (i_1 \quad i_3) \quad (i_1 \quad i_4) \quad (i_2 \quad i_3) \quad (i_2 \quad i_4)$. The set of terms that can cause errors is now $\{(i_3)(i_4)(i_3 \quad i_4)\}$. The initial hypothesis will produce an error on $(1 - p_3 p_4)$ of the positive examples (since an error is made when i_3 is false or when i_4 is false). After *n* training examples, the hypothesis will produce an error on $(p_3)^n(1{-}p_3) + (p_4)^n(1{-}p_4) - (p_3p_4)^n(1{-}p_3p_4)$ of the positive training examples. Figure 1 also graphs the observed and expected error with a total of 4, 5, 6, and 7 features (with all $p_i = 0.5$). Note that for this problem, the analysis is equivalent to learning the always true concept (i.e., $D^* = \{\}$) while ignoring the two features i_1 and i_2. This occurs because both features must appear in every positive example.

3.2 ATTRIBUTE NOISE

To illustrate how the noise-tolerant *k*-CNF algorithm performs, we ran a simulation with five features: `a`, `b`, `c`, `d`, and `e`. The probability that the features had a true value was 0.7, 0.4, 0.8, 0.5, and 0.6, respectively. The correct concept definition which would classify noise-free examples properly was (a b) (a c). The probability that a feature value was replaced with a random value (α) was 0.04. We use a 2-CNF algorithm with ε equal to 0.1 (i.e., a disjunction must be false in 10% of the positive examples to be removed from the hypothesis). Figure 3 graphs the observed and predicted probability that a positive example is misclassified by the hypothesis produced by the learning algorithm under these conditions (averaged over 100 trials).

An increase of expected error occurs when the value of *ceiling*(ε*N*) increases. In this case, this happens after every 10 training examples since ε equal to 0.1. Note that the expected error does not converge on 0. Although the algorithm can converge on the correct concept definition, the correct concept misclassifies some examples because there is noise in the test data.

4 VIOLATING ASSUMPTIONS

In the previous section, the intent of the simulations was to obtain confirmation of the mathematical result and to visualize the equations. A common measure used to describe how well a model fits the data is the coefficient of determination (r^2). Since we generated data according to the assumptions used to derive the model, it is not surprising that r^2 is greater than 0.99 for the data graphed in Figures 1 and 2. In this section, we evaluate the sensitivity of the model to violations of the independence assumption. We run a series of experiments in which the the value of one feature is dependent on the value of another feature. In particular, we experiment with learning when there are 6 features, a, b, c, d, e, and f and the correct concept is (a b) (c d). We consider the following dependencies:

I. Dependencies between features of the same disjunction (e.g., a and b).
II. Dependencies between features of a different disjunction (e.g., a and c).
III. Dependencies between a features in the hypothesis and an irrelevant feature (e.g., a and e).
IV. Dependencies between irrelevant features (e.g., e and f).

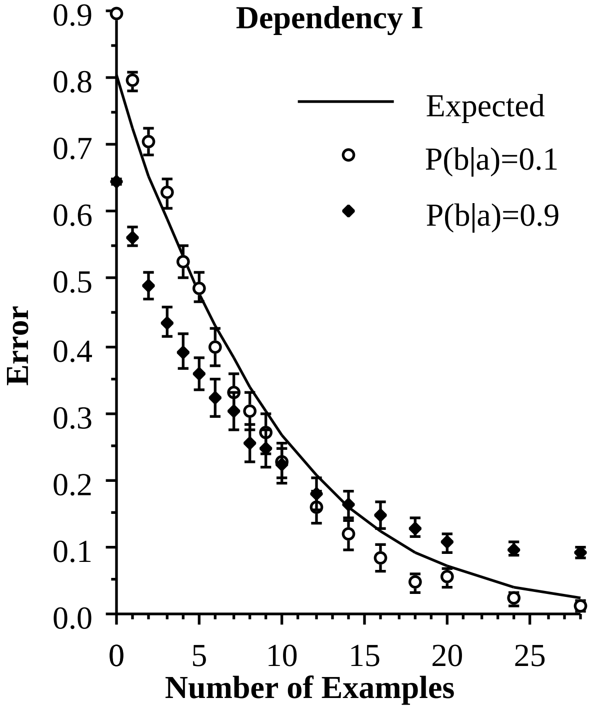
In each case, there is a 0.5 probability that each feature has a true value. However, we systematically vary the dependence between features. For example, if P(a) is 0.5, P(b|a) is 0.9 and P(b|¬a) is 0.1, then P(b) will be 0.5. Since P(b) is 0.5, the independence assumption of the average case model is not violated only when P(b|a) is 0.5. For values of P(b|a) greater than 0.5, a and b are correlated and for values less than 0.5, these features are inversely correlated. We ran a series of simulations in which we varied the value of $P(f_1|f_2)$ from 0.1 to 0.9 by 0.1 increments and measured the accuracy of the *k*-CNF algorithm at various points between 0 and 30 training examples. Figure 3, Parts I and IV, graph the accuracy predicted by the average case model and the observed accuracy for these experiments. The points with 95% confidence intervals indicate the mean error (averaged over 150 trails) measured on 200 randomly drawn positive examples. The line represents the error predicted by the average case model. Although the data violate the assumptions of the average case model, the predicted accuracy still accounts for a large percentage of the variation in observed error. In particular, for dependency class I, r^2 is 0.971 when P(b|a) =0.1 and 0.838 when P(b|a) =0.9. For class II, r^2 is 0.957 when P(c|a) =0.1 and 0.941 when P(c|a) =0.9. For class III, r^2 is 0.986 when P(e|a) =0.1 and 0.966 when P(e|a) =0.9. For class IV, r^2 is 0.983 when P(f|e) =0.1 and 0.959 when P(f|e) =0.9. In all cases, when $P(f_1|f_2)$ is 0.5 r^2 is greater than 0.99.

Iit is possible to characterize the systematic deviations from the predicted error as a function of the class of the dependency between features and the direction of the correlation. These conclusions are also supported by the data for $P(f_1|f_2)$ in the range 0.2 to 0.8 that are not graphed in this paper:

- Class IV dependencies (correlations between irrelevant features): When the correlation is positive (i.e., P(f|e) > 0.5), the model overestimates the actual error. The extreme case of this positive correlation (P(f|e) = 1.0) is equivalent to having one fewer irrelevant feature. On the other hand, when e and f are inversely correlated, it is more difficult for the *k*-CNF algorithm to remove terms involving these variables from the hypothesis, so the model underestimates the error.
- Class III dependencies (correlations between a relevant and an irrelevant feature): When the correlation is positive, the model overestimates the error. Once again, the extreme case of this positive correlation (P(e|a) = 1.0) is equivalent to incorrectly telling the model that there is one additional irrelevant feature.
- Class II dependencies (correlations between features of different disjuncts): When the correlation is positive, the model overestimates the error. This occurs because when the algorithm drops an irrelevant term (e.g., (a f)) involving one feature, it is also likely to drop the irrelevant term involving its dependent feature (e.g., (c f)).
- Class I dependencies (correlations between features of the same disjunct): There is a complex interaction in that for positive correlations, the model initially overestimates the error and after several training examples it underestimates the error. The behavior is reversed for negative correlations.

5 CONCLUSION

We have presented an approach to modeling the average case behavior of an algorithm for learning *k*-CNF. The model predicts the expected error of the algorithm as a function of the number of training examples. We evaluated how well the average-case model predicts the observed error when the independence assumption of the product distribution is violated. We have shown the analysis can lead to insight into factors that affect the



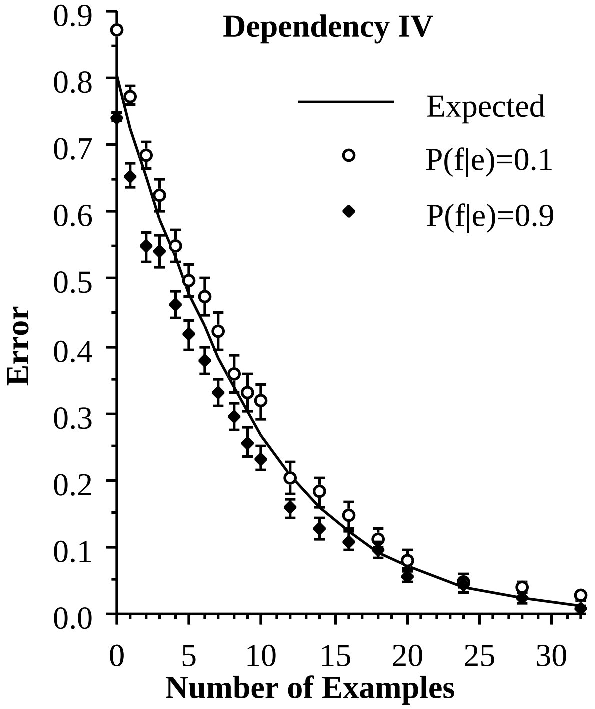


Figure 3. The observed accuracy when the assumptions of the average case model are violated by introducing dependencies between feature values.

error of the learning algorithm. A longer technical report is available from the authors providing addition details and implications of the model.

The average case model requires much more information about the training examples than the PAC learning model. The information required by the model is exactly the information required to generate artificial data to test learning algorithms. One future research direction would be to relax some of these assumptions. For example, rather than requiring the correct concept definition, it might be possible to perform a similar analysis for a given probability distribution of possible concepts.

Acknowledgements
We would like to thank Dennis Kibler, Kamal Ali, Tim Cain and Jim Wogulis for helpful comments on this work and Caroline Ehrlich for reviewing an earlier draft of this manuscript. This research is supported by a National Science Foundation Grant IRI-8908260.

References

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association of Computing Machinery, 36,* 929-965.

Haussler, D. (1987). Bias, version spaces and Valiant's learning framework. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 324-335). Irvine, CA: Morgan Kaufmann..

Haussler, D. (1990). Probably approximately correct learning. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 1101-1108). Boston: AAAI Press.

Haussler, D., Littlestone, N. & Warmuth, M. (1990). Predicting {0,1}-functions on randomly drawn points. *Technical Report* USCS-CRL-90-54, University of California, Santa Cruz.

Pazzani, M., & Sarrett, W. (1990). Average case analysis of conjunctive learning algorithms. *Proceedings of the Seventh International Workshop on Machine Learning*, Austin, TX: Morgan Kaufmann.

Pazzani, M., & Sarrett, W. (in press) A framework for average case analysis of conjunctive learning algorithms. *Machine Learning.*

Quinlan, J.R. (1986). *The effect of noise on concept learning.* In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), Machine Learning: An artificial intelligence approach (Vol 2). San Mateo, CA: Morgan Kaufmann.

Valiant, L. (1984). A theory of the learnable. *Communications of the Association of Computing Machinery, 27*, 1134-1142.

Valiant, L. (1985). Learning disjunctions of conjunctions. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp 560-566). Los Angeles, CA: Morgan Kaufmann.