The Divide-and-Conquer Manifesto

Thomas G. Dietterich

Oregon State University, Corvallis, OR 97331, USA,
tgd@cs.orst.edu,
WWW home page: http://www.cs.orst.edu/~tgd

Abstract. Existing machine learning theory and algorithms have focused on learning an unknown function from training examples, where the unknown function maps from a feature vector to one of a small number of classes. Emerging applications in science and industry require learning much more complex functions that map from complex input spaces (e.g., 2-dimensional maps, time series, and strings) to complex output spaces (e.g., other 2-dimensional maps, time series, and strings). Despite the lack of theory covering such cases, many practical systems have been built that work well in particular applications. These systems all employ some form of divide-and-conquer, where the inputs and outputs are divided into smaller pieces (e.g., "windows"), classified, and then the results are merged to produce an overall solution. This paper defines the problem of divide-and-conquer learning and identifies the key research questions that need to be studied in order to develop practical, general-purpose learning algorithms for divide-and-conquer problems and an associated theory.

1 Introduction

The basic supervised learning task is to find an approximation h to an unknown function f given a collection of labeled training examples of the form $\langle x, y \rangle$, where x is a fixed-length vector of features and $y = f(x)$ is a class label or output value (e.g., drawn from a small number of discrete classes or an interval of the real line). In the theory of supervised learning, these training examples are assumed to be produced by independent draws from some underlying probability distribution.

However, when we look at current and emerging applications of machine learning, we find the situation is much more complex. The x values—instead of being fixed-length vectors—are often variable-length objects such as sequences, images, time series, or even image time series (e.g., movies, sequences of aerial photos taken over several years). The y values may be similarly complex sequences, images, or time series. Let us consider a few examples.

Example 1: Text-to-Speech. A famous demonstration of machine learning is the problem of mapping spelled English words into speech signals, as in the NETtalk system (Sejnowski & Rosenberg, 1987). Each training example is an English word (e.g., "`enough`") along with an aligned phonetic transcription (e.g., "`In^-f-`") and an aligned stress transcription (e.g, "`0>1<<<`"). This is a case in which both the x and the y values are variable-length sequences.

Example 2: Grasshopper Infestation Prediction. We have been studying the problem of predicting future infestations of grasshoppers in Eastern Oregon based on a map of the adult grasshopper population in the previous year and the daily weather during the fall, winter, and spring (Bunjongsat, 2000). In this case, each training example is a two-dimensional population map coupled with a time series of daily weather maps, and the output is another two-dimensional map.

Example 3: Fraud detection in transactions. Many applications of machine learning involve analyzing time series of transactions (e.g., telephone calls, insurance claims, TCP connection attempts) to identify changes in behavior associated with fraudulent activity (Fawcett & Provost, 1997). This can be formalized as a problem of mapping an input sequence of transactions to an output sequence of alarms.

Example 4: Finding all volcanoes on Venus (Burl, Asker, Smyth, Fayyad, Perona, Crumpler, & Aubele, 1998). Many visual applications involve scanning images to identify objects of scientific interest (volcanoes, bacteria) and estimate relevant properties (location, volume, age). In this case, the input is a two-dimensional map of pixels and the output is a two-dimensional map of detected objects (and their predicted properties).

To solve these kinds of complex problems, practitioners have applied variations on the venerable "divide and conquer" schema. Viewed abstractly, every divide-and-conquer method consists of three steps: (a) divide (divide the original problem into subproblems), (b) conquer (solve the subproblems, possibly recursively), and (c) merge (merge the subproblem solutions into a solution for the original problem).

To apply this schema in machine learning, the x and y values are decomposed into "windows" or "regions", individually classified, and then merged to provide a classification decision for the original problem. For example, in the NETtalk task, the problem of predicting the entire phoneme sequence (and stress sequence) is divided into the subproblem of predicting each individual phoneme. To predict $y(i)$, the ith phoneme (and stress) of a word, a 7-letter window of the input, from $x(i-3)$ to $x(i+3)$, is used to extract a set of input features. To map an entire word from text to phonemes, we must separately predict the phoneme and stress of each letter and then concatenate them.

Similarly, for the grasshopper task, one approach is to define a grid of cells and try to predict the grasshopper population within each cell using as input the previous year's population and weather in that cell and neighboring cells. To construct a prediction map for each year, a prediction is made within each cell and then those predictions are concatenated to get the whole map.

In both of these examples, the merge step was a trivial concatenation, but more sophisticated versions of both problems employ complex merge steps. For example, in our decision tree text-to-speech system (Bakiri & Dietterich, 2000), we developed a "recurrent" classifier that constrained the allowable predictions for each subproblem based on the predictions of other subproblems. Specifically, we scanned each word from back-to-front, and the results of earlier predictions were used as input features to constrain subsequent predictions. This strategy enabled us to correctly pronounce word pairs such as "photograph" and "photography", even though they differ only in the last letter.

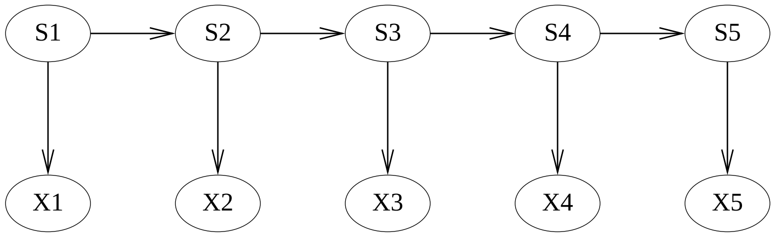

Fig. 1. Belief network representation of a hidden Markov model.

One of the most well-developed "merge methods" is based on Markov modeling (Bengio, 1999; Jelinek, 1999). Figure 1 shows a belief network representation of a hidden Markov model (HMM). Each of the hidden nodes S_i (except S_1) stores a transition probability distribution of the form $P(S_i|S_{i-1})$, and each observed node X_i stores an emission probability distribution of the form $P(X_i|S_i)$. An HMM is a stochastic finite state automaton that can be used to generate or recognize strings. To generate a string, state S_1 is chosen according to $P(S_1)$, and then the first output X_1 is chosen according to $P(X_1|S_1)$. Then the second state S_2 is generated according to $P(S_2|S_1)$ and so on. Only the X_t's are observed in the training and test data.

We can view the HMM as a divide-and-conquer method in which the base classifier is represented by $P(X_i|S_i)$ (which can be inverted by Bayes theorem to give $P(S_i|X_i)$, which assigns a class label S_i to the observed value X_i) and the merge method is represented by $P(S_i|S_{i-1})$. To merge a series of individual decisions, standard belief propagation methods can be applied to find the most likely sequence of states $S_1, S_2, \ldots, S_n$ that could have generated the observed data $X_1, X_2, \ldots, X_n$.

In speech recognition, for example, the problem is to map a speech signal into an English sentence. In this application, the hidden states of a hidden Markov model describe the temporal structure of English (i.e., what words can follow what other words, what phones can follow what other phones), and the emission probabilities can be viewed as naive Bayesian classifiers (or gaussian mixture classifiers) for deciding which phone generated each frame. One of the great virtues of the hidden Markov model is that both the base classifier and the merge step are trained jointly. This is in contrast to most other divide-and-conquer methods, where the base learning algorithm is trained independently of the merging process.

In recent years, many groups, particularly in speech recognition, have explored hybrid architectures where some other classifier (e.g., decision tree, neural network) is used in place of the emission probabilities of the HMM (Lippmann & Gold, 1987; Franzini, Lee, & Waibel, 1990; Bengio, De Mori, Flammia, & Kompe, 1992; Bourlard & Morgan, 1993). This permits a richer model of local interactions than the usual naive Bayes model, and that has led to success in such applications as online handwriting recognition (Bengio, Le Cun, & Henderson, 1994), molecular biology (Haussler, Krogh, Brown, Mian, & Sjölander, 1994; Baldi & Brunak, 1998), and part-of-speech tagging (Màrquez, 1999; Màrquez, Padró, & Rodríguez, 2000), as well as in speech recognition.

2 Research Issues in Divide-and-Conquer Learning

When applying a divide-and-conquer approach, there are six key design decisions that must be made: (a) output scale, (b) input scale, (c) alignment of outputs and inputs, (d) decomposition of the loss function, (e) base learning algorithm, and (f) merge method.

The output scale is the size of the regions or segments into which y is divided. For example, in our text-to-speech research (Bakiri & Dietterich, 2000), we chose to predict individual letters. But perhaps predicting pairs of letters would have been more effective, since some pairs of letters have highly predictable pronounciations (e.g., "st", "ck", and so on). Although we ran hundreds of experiments, we did not run this particular experiment. In our grasshopper study, we chose to predict the presence or absence of infestation in grid cells that were 10km on a side. Was this the correct size? We did not have time to test other grid sizes, so we do not know.

The input scale is the size of the input "window" that will be supplied as input to the base level classifier. In the original NETtalk system, Sejnowski and Rosenberg employed a 7-letter window. Bakiri (1991) performed an exhaustive series of experiments and found that a 15-letter window gave the best results. In our grasshopper domain, the input scale was a 30×30km square region, but other sizes may have been better.

The third decision involves how to align the output windows with the input windows. In the NETtalk domain, Sejnowski and Rosenberg manually inserted silent phonemes into the output phoneme string so that there was a direct 1:1 correspondence between input letters and output phonemes. But in many applications, the outputs and inputs are not pre-aligned. Lucassen and Mercer (1984) and Ling (1997) have both studied automatic alignment mechanisms for speech generation. Similarly, speech recognition systems typically employ forced Viterbi alignment to align the output words and phones with the input windows. Starting with a small set of aligned data, they train an initial HMM. Then this HMM is applied to unaligned data to find the most likely assignment of the given output words and phones to the input windows. This alignment is assumed to be correct, and it is then used as additional input data for training a new HMM.

The fourth decision involves how to decompose the overall loss function into a loss function that can be applied in the base case. The loss $L(\hat{y}, y)$ is the penalty incurred when the learned mapping h predicts $\hat{y} = h(x)$, but the true answer is $y = f(x)$. For example, in the grasshopper prediction task, the loss suffered when we fail to predict a grasshopper infestation is the cost of the resulting crop damage, and the loss suffered when we predict an infestation (rightly or wrongly) is the cost of spraying pesticides. This loss function decomposes perfectly into loss functions for any particular output scale, because the total loss over the entire region is the sum of the loss at each location. Such perfect decomposition means that the global loss function can be minimized by minimizing the local loss function using the base learning algorithm.

Unfortunately, in most complex learning problems, the loss function does not decompose so simply. Consider, for example, the problem of speech recognition. Here the goal is to identify the entire sentence correctly, so a loss of 1 is incurred if any word in the sentence is wrong (with a loss of 0 if no words are wrong). However, this does not decompose perfectly into a loss function for classifying each phone. In fact, as long as the maximum likelihood path through the HMM passes through the correct sequence of words, it does not matter whether every phone was correctly classified individually.

The loss function in fraud detection problems depends on the financial losses incurred by the fraudulent activity. This in turn is related to the amount of time between the start of fraudulent behavior and the time when the learned classifier raises an alarm. There is also typically a high cost to false alarms as well. This loss function is difficult to decompose into loss functions for the individual windows because only the *first* alarm in an episode matters.

The loss function for detecting volcanoes on Venus is also complex. If a volcano is detected in a slightly incorrect position, this is not a serious error. But detecting the same volcano at adjacent positions is an error (because each volcano should be detected only once), and so is the failure to detect a volcano at all. Hence, the definition of "correctly detecting a volcano" is not purely local—it depends on the results of several classification decisions in the neighborhood of the true volcano location. An additional complicating factor is that the training data (expert-labeled maps of "training regions" on Venus) is believed to contain volcanoes that were missed by the experts—inter-expert agreement is not very high.

The fifth decision involves choosing (or designing) the learning algorithm for solving the "base case" of the divide-and-conquer schema. Traditionally, standard machine learning methods have been applied here. However, many of the assumptions underlying those methods are violated in the divide-and-conquer setting: the training examples are no longer independent and identically distributed (iid) and the objective is not to maximize the percentage of correct classification decisions but instead to provide the most useful information to the merge step.

The merge method is perhaps the most important of these six decisions. This is the choice of how to merge the solutions of the individual subproblems to produce a solution to the overall problem. In the literature, many methods have been applied including simple concatenation (as in NETtalk), feeding the outputs through a second "merge" network (as in Qian and Sejnowski's (1988) protein structure prediction system), learning a recurrent classifier (as described above), and employing hidden Markov models (as described above) to find the most likely merged solution.

These six design decisions provide an agenda for machine learning research on divide-and-conquer problems. The goal of this research will be to study each of these design decisions, understand how the decisions interact, and develop methods for making them automatically.

In this paper, we will not address all six of these problems. Instead, we focus only on the input scale, the output scale, and the merge method.

3 Factors Affecting the Design of Divide-and-Conquer Systems

We begin with an analysis of the main factors that influence the choice of output scale, input scale, and merge method. The most important factor is the extent to which neighboring $y(i)$ values are correlated even after accounting for the information provided by the predictor x values. To make the discussion concrete, suppose that we are classifying each pixel of an image into one of two classes based on the measured red, green, and blue intensities of each pixel (the x values). Suppose the output scale is a single pixel, so $y(i)$ refers to the class of one pixel and $x(i)$ is a vector of the red, green, and blue intensities. Consider the conditional joint probability distribution $P(y(1), y(2)|x(1), x(2))$ of two adjacent pixels. Suppose that this can be perfectly factored into $P(y(1)|x(1)) \cdot P(y(2)|x(2))$. Figure 2(a) shows a belief network for this case. In this case, we can choose the output scale to be one pixel (i.e., $y(i)$), because the only way that $y(i)$ and $y(j)$ are correlated is through the correlations of $x(1)$ and $x(2)$.

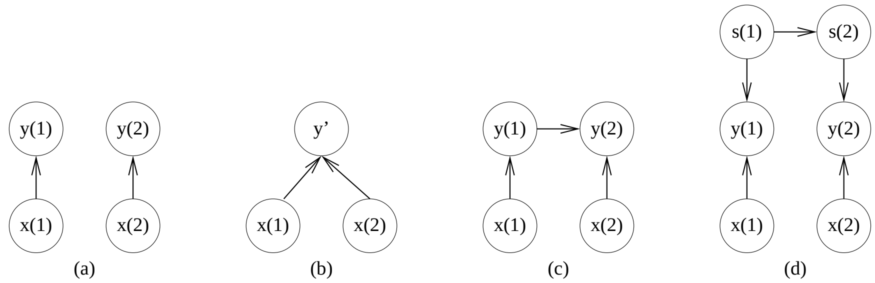

Fig. 2. Belief networks representing four architectures for divide-and-conquer systems.

However, now suppose that there is some additional correlation between $y(1)$ and $y(2)$ that cannot be accounted for by the correlation between $x(1)$ and $x(2)$.

In this case, the joint distribution $P(y(1), y(2)|x(1), x(2))$ does not factor. There are at least three ways to handle this. First, we can increase the output scale to include both $y(1)$ and $y(2)$ (and the input scale to include $x(1)$ and $x(2)$). This is equivalent to defining a new output variable y' which takes on four possible values corresponding to the four possible labels of $y(1)$ and $y(2)$ (see Figure 2(b)).

Second, we could apply the chain rule of probability and write the $P(y(1), y(2) \mid x(1), x(2))$ distribution as $P(y(1)|x(1)) \cdot P(y(2)|x(2), y(1))$ (where we have also assumed that $y(1)$ does not depend on $x(2)$.) This suggests a recurrent solution in which we first predict the value of $y(1)$ using $x(1)$, and then use this predicted value along with $x(2)$ to predict $y(2)$ (see Figure 2(c)).

The third approach is to model the relationship between $y(1)$ and $y(2)$ as a hidden Markov model (see Figure 2(d)), using hidden states $s(1)$ and $s(2)$.

This simple analysis shows that there is a tight connection between the choice of the output scale and the choice of the merge method. If we are merging the individual decisions via an HMM, we can use a smaller output scale (Figure 2(c) and (d)) than if we are merging by concatenating the independent classifications (as in Figure 2(b)), because the HMM captures the correlations between the y values that would otherwise need to be captured by a larger output scale.

The analysis also suggests that if the input scale is too small, the output scale may need to be larger or the merge step may need to be more complex. The reason is that if the input scale does not capture all of the correlations among the $x(i)$ values, then there will be "induced" correlations among the $y(i)$ values. For example, if $y(1)$ depends directly on both $x(1)$ and $x(2)$, but the base classifier ignores $x(2)$, then this will create an added dependency between $y(1)$ and $y(2)$ (because $y(2)$ depends on $x(2)$).

A second factor affecting the choice of input and output scale is the amount of noise in the $x(i)$ and $y(i)$ values. Large noise levels (for a fixed amount of input data) require high degrees of smoothing and aggregation. This is a consequence of the well-known bias-variance tradeoff. Noisy training data leads to high variance and hence, to high error rates. The variance can often be reduced by imposing a smoothing or regularizing process. In temporal and spatial data, it is natural to apply some form of temporally- or spatially-local smoothing, since we normally assume that the underlying x and y values are changing smoothly in space and time. One way of imposing local smoothing is to use a larger output scale. Consider again the example from Figure 2(b), where we introduced a new variable y' that took on four values $\{00, 01, 10, 11\}$ corresponding to the four possible pairs of labels for $y(1)$ and $y(2)$. We can impose spatial smoothing by constraining y' to only two possible values $\{00, 11\}$. In other words, the larger output scale is constraining $y(1) = y(2)$. A similar constraint can also be imposed through the merge techniques shown in Figure 2(c) and (d). These constraints can be made "soft" through Bayesian methods. For example, rather than banning the 01 and 10 values for y', we can just impose a penalty for using them by assigning them lower prior probability. In addition to building a smoothness constraint into the model, we can also impose smoothness by preprocessing the data to smooth the y values prior to running the base learning algorithm.

If there is noise in the input data, then this usually requires a larger input scale, so that the base classifier can aggregate a larger number of inputs to overcome the noise. Again, we can also consider smoothing the input data prior to running the base classifier (e.g., by modelling the process by which noise is added to the data as a Markov random field (a 2-D Markov process) and then finding the maximum aposteriori probability estimate of the true data given the observed data).

A third fundamental issue influencing the choice of the merge step is the direction of causality. In standard supervised learning and in learning belief networks, there is a growing body of evidence that suggests that learning is most efficient (statistically) when the model being fit to the data matches the direction of underlying causality. In such cases, the model can usually be parameterized using a small number of parameters, and consequently, less data is needed to fit those parameters.

Let us consider the direction of causality in the three merge methods sketched above. If we treat $y(1)$ and $y(2)$ as in Figure 2(b) or (d), we are assuming that there is no particular direction of causality between them. If we employ a recurrent method, we are assuming that a label for $y(1)$ is chosen first, and then it is used to help choose a label for $y(2)$. This direction of causality is typically more appropriate for time-series data than for spatial data or biological sequence data. This suggests that the choice of merge method in a particular application should depend primarily on domain knowledge about the likely direction of causality in the problem.

4 An Experimental Study

We now describe an experimental study of the tradeoff between using a large input scale with a simple merge method and using a small input scale with the more complex HMM merge method. To generate the training and test data, we employed a hidden Markov model of the kind shown in Figure 1. In this data, each S_i is a boolean class variable that is observed in the training data and hidden (and hence, predicted) in the test data. Each X_i is a vector of 10 boolean variables $(x_{i,0}, \ldots, x_{i,9})$ generated by a simple Naive Bayes model (i.e., there is a separate probability distribution $P(x_{i,j}|S_i)$ that generates each $x_{i,j}$ depending on the value of S_i), and these are observed in both the training and test data. We will choose the transition probability distribution $P(S_{i+1}|S_i)$ and the output probability distributions $P(x_{i,j}|S_i)$ to be stationary (i.e., the same for all values of i).

Given that we have generated training data according to this HMM, we wish to compare three learning algorithms. The first algorithm is "optimal" in the sense that it learns an HMM of exactly the same structure as the true HMM that generated the data. It is trivial to directly learn the HMM, because all of its random variables are observed in the training data. To classify test examples using the learned HMM, we must apply the forward-backward algorithm to compute $P(S_i|X_1, \ldots, X_N)$ for each S_i. The forward-backward algorithm can be viewed as a combination of two separate algorithms. The forward algorithm processes the sequence from left-to-right, and for each i, it can be viewed as computing $P(S_i|X_1, \ldots, X_i)$, which is the probability of the ith class label given the sequence seen so far. The backward algorithm processes the sequence from right-to-left, and for each i, it can be viewed as computing $P(S_i|X_{i+1}, \ldots, X_N)$. At each node i, these two probability distributions can be multiplied together and appropriately normalized to obtain $P(S_i|X_1, \ldots, X_N)$. (Note: This is a non-standard description of the forward-backward algorithm. The reader is referred to (Baldi & Brunak, 1998; Jelinek, 1999) for more rigorous and detailed descriptions.)

The second algorithm is just the forward part of the forward-backward algorithm. The reason to study this method is that it is similar to the kind of "recurrent" algorithm that Bakiri and Dietterich employed in the text-to-speech task. The results of classifying X values earlier in the sequence are used as inputs to classify later values.

The third algorithm applies the standard Naive Bayes classifier to predict each S_i independently. In other words, it assumes that each pair (X_i, S_i) is generated independently from the same distribution according to the probabilities $P(S_i)$ and $P(x_{i,j}|S_i)$. We will call this third algorithm, iid-Bayes, and we will allow it to use wide input windows as follows. An input window of width 3 uses X_{i-1}, X_i, and X_{i+1} to predict the value of S_i. Since this is a Naive Bayes classifier, it does this by learning probability distributions of the form $P(x_{k,j}|S_i)$, for all j and all $k \in \{i-1, i, i+1\}$.

In our experiments, we choose the distribution $P(S_i|S_{i-1})$ to have a symmetric form such that the class changes with probability δ and remains the same with probability $1-\delta$. When $\delta = 0.5$, this means that the individual (X_i, S_i) pairs are generated independently and identically. But when δ is small, adjacent values of S_i are highly correlated.

Our experiments consisted of 100 trials. In each trial, we applied the HMM to randomly generate a training set and a test set, each containing 10 sequences of length 50. The probability distribution $P(S_1)$ was the uniform distribution.

Figure 3 shows the results of varying δ across a range from 0.01 to 0.50 while using a window size of 1 for iid-Bayes. We see that when $\delta = 0.5$, the three algorithms give the same performance, but as δ becomes small, the methods that explicitly model the dependency between the S_i values perform much better. The forward-backward algorithm gives the best results, of course, but the forward algorithm does quite well. The lesson of this experiment is that it is a mistake to ignore the dependencies between adjacent windows!

Figure 4 shows the results of varying the window size of iid-Bayes. When δ is very small, iid-Bayes can obtain excellent performance by using a very wide window. The reason, of course, is that the wide window captures the correlations between adjacent S_i values indirectly by exploiting the resulting correlations between the X_i values. However, when δ approaches 0.5, these large windows perform poorly, because now they are overfitting the data. Furthermore, the larger the window, the greater the opportunity for overfitting, and hence, the

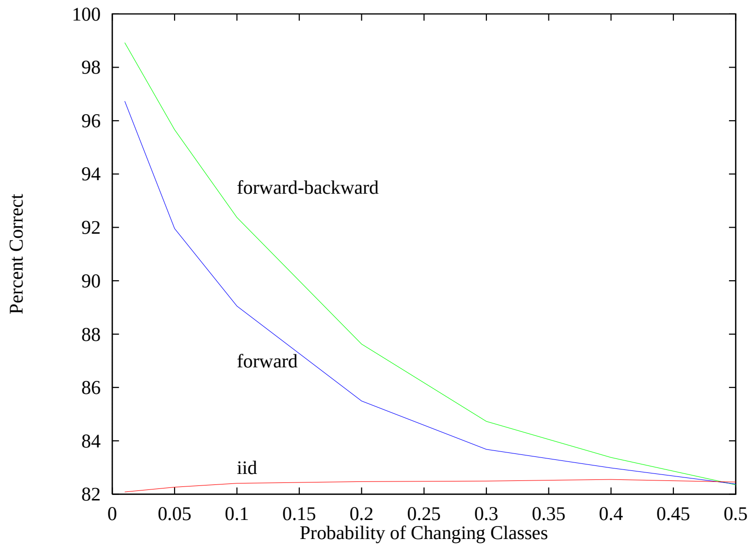

Fig. 3. A comparison of the percentage of correct predictions on the test data for the forward-backward algorithm, the forward algorithm, and the iid-Bayes(1) algorithm for different values of δ.

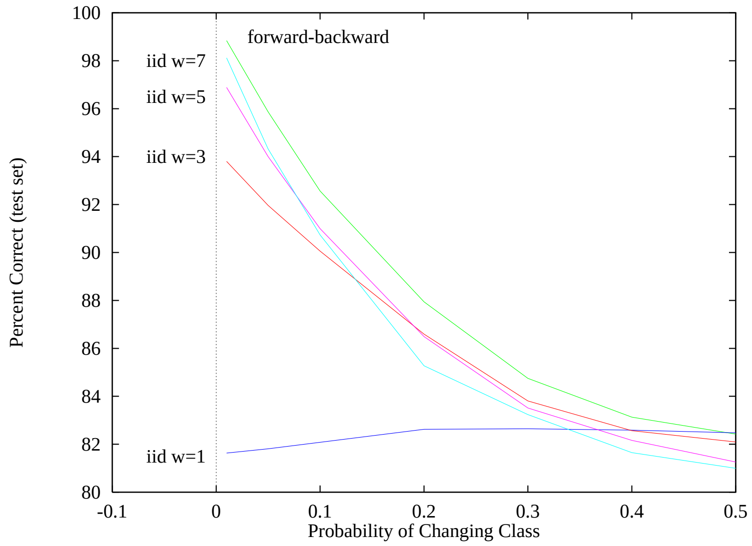

Fig. 4. Test-set performance of iid-Bayes for different input window sizes compared against the forward-backward algorithm.

worse the performance. Hence, we can see that a window size of 7 gives the best iid-Bayes performance for δ from 0 to 0.08. A window size of 5 gives the best performance for δ from 0.08 to 0.19. A window size of 3 gives the best performance for δ for 0.19 to 0.42. And for $\delta > 0.42$, a window size of 1 gives the best performance.

The lesson of this experiment is that the proper choice of input scale depends on the strength of correlation between adjacent S_i values, even when that correlation is a first-order Markov process. Another lesson is that there is an overfitting cost to using wide windows when they are inappropriate.

We performed a third experiment to see what happens when the temporal dependency model in the HMM is incorrect. We took each training example and re-ordered the individual (X_i, S_i) pairs to have the following order: (X_1, S_1), (X_{26}, S_{26}), (X_2, S_2), (X_{27}, S_{27}), $\ldots$, (X_{25}, S_{25}), (X_{50}, S_{50}). However, the HMM learning algorithm still applied the (now incorrect) HMM from Figure 1 to fit the data.

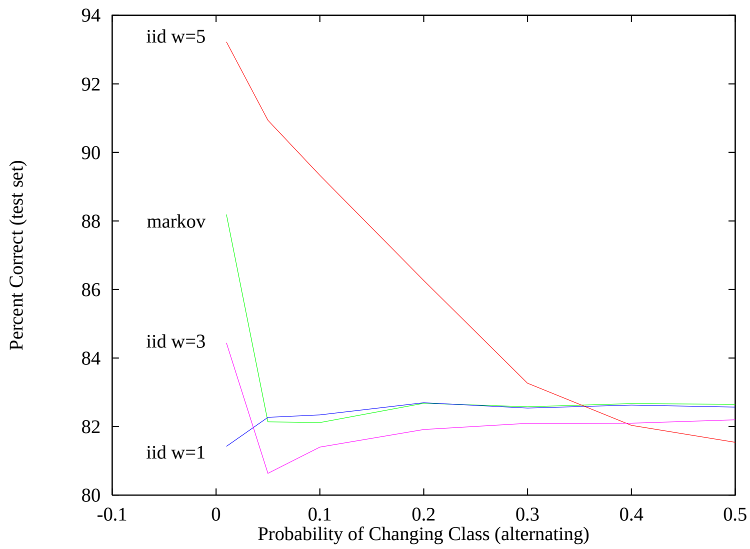

Fig. 5. Comparison of HMM and iid-Bayes on shuffled data, where the HMM model does not correctly capture the sequential dependencies in the data.

Figure 5 compares the performance of this incorrect Markov model with the iid-Bayes model for various settings of δ. We see that now iid-Bayes with a window size of 5 is able to do much better than the HMM, because a window size of 5 is large enough to capture the dependencies between S_i and S_{i-2} and S_{i+2}, whereas the first-order HMM cannot capture these dependencies. Notice that the first-order HMM gives essentially the same performance as iid-Bayes with a window size of 1 with the exception of very small values for δ. At these very small values for δ, there is a non-trivial correlation between S_i and S_{i+25}, so even a first-order HMM can capture some useful information. It is interesting that iid-Bayes with a window size of 3 also captures some of this information, but because of overfitting, it performs uniformly worse than the HMM.

This simple experimental study shows that if you have a correct model of the temporal dependencies in sequential data, then the HMM (forward-backward) approach to divide-and-conquer problems is the best method to apply. Sliding window methods that rely on a wide input window and a trivial merge step perform almost as well, but the window size must be adjusted depending on the strength of the temporal correlations. Finally, if you have an incorrect model of the temporal correlations, then the HMM method is much less robust, and the sliding window iid-Bayes approach gives superior results.

5 Concluding Remarks

Emerging applications of machine learning require algorithms that can learn mappings from complex input spaces to complex output spaces. A natural approach to solving such problems is to employ some form of divide-and-conquer. However, there are many difficult decisions that must be made in designing a divide-and-conquer learning system: (a) the input scale, (b) the output scale, (c) alignment of inputs and outputs, (d) decomposition of the loss function, (e) the base learning algorithm, and (f) the merge method. These design decisions interact in complex ways.

We presented a simple theoretical analysis which suggests that the input and output scales interact with the choice of merge method. Our experimental study verified this for the simple case in which the data was generated by an HMM. If we applied an HMM classifier, the input scale and output scale could both be 1. But if applied a Naive Bayes classifier and merged by simple concatenation, then we needed much larger input scales.

Researchers in speech recognition have had the most experience with learning complex mappings, and their HMM-based techniques appear very promising for explicitly representing temporal constraints. However, our study also showed that if the assumptions of the model (e.g., of first-order Markov interactions) is wrong, then HMM-based methods will perform very poorly, while large input windows are more robust. This is consistent with work combining neural networks (and wide input windows) with HMMs to overcome some of the modeling shortcomings of HMMs. It will be interesting to see how well other learning algorithms, such as tree- and rule-learning methods, can be combined with HMM-based merge procedures.

I hope this paper will encourage machine learning researchers to mount a systematic attack on the problems of divide-and-conquer learning. We are in the midst of a machine learning revolution, as the learning algorithms developed over the last 20 years are becoming widely applied in industry and science. However, many of the new applications of machine learning are complex, and require divide-and-conquer methods. Rather than continue the current trend of constructing ad hoc divide-and-conquer systems, we need to study these complex problems and develop learning algorithms specifically tailored to them. One can imagine a divide-and-conquer toolkit in which it would be easy to (a) describe the temporal and spatial structure of complex input and output data, (b) represent the global loss function of the application, and (c) automatically construct and train a divide-and-conquer architecture. As machine learning moves beyond simple classification and regression problems, complex divide-and-conquer methods are one of the most important new directions to pursue.

Acknowledgements

I wish to thank Dragos Margineantu for many conversations which helped refine the ideas presented in this paper. The author gratefully acknowledges the support of NSF grant 9626584-IRI.

Bibliography

Bakiri, G. (1991). Converting English text to speech: A machine learning approach. Tech. rep. 91-30-2, Department of Computer Science, Oregon State University, Corvallis, OR.

Bakiri, G., & Dietterich, T. G. (2000). Achieving high-accuracy text-to-speech with machine learning. In Damper, R. I. (Ed.), *Data Mining Techniques in Speech Synthesis*. Chapman and Hall, New York, NY.

Baldi, P., & Brunak, S. (1998). *Bioinformatics, the Machine Learning Approach*. MIT Press.

Bengio, Y. (1999). Markovian models for sequential data. *Neural Computing Surveys*, *2*, 129–162.

Bengio, Y., De Mori, R., Flammia, G., & Kompe, R. (1992). Global optimization of a neural-network hidden Markov model hybrid. *IEEE Transactions on Neural Networks*, *3*(2), 252–258.

Bengio, Y., Le Cun, Y., & Henderson, D. (1994). Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden Markov models. In Cowan, J. D., Tesauro, G., & Alspector, J. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 6, pp. 937–944. Morgan Kaufmann, San Francisco.

Bourlard, H., & Morgan, N. (1993). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer.

Bunjongsat, W. (2000). Grasshopper infestation prediction: An application of data mining to ecological modeling. Tech. rep., Department of Computer Science, Oregon State University. MS Project Report.

Burl, M. C., Asker, L., Smyth, P., Fayyad, U., Perona, P., Crumpler, L., & Aubele, J. (1998). Learning to recognize volcanoes on Venus. *Machine Learning*, *30*(2/3), 165–194.

Fawcett, T., & Provost, F. (1997). Adaptive fraud detection. *Knowledge Discovery and Data Mining*, *1*, 291–316.

Franzini, M., Lee, K., & Waibel, A. (1990). Connectionist Viterbi training: a new hybrid method for continuous speech recognition. In *International Conference on Acoustics, Speech, and Signal Processing*, pp. 425–428.

Haussler, D., Krogh, A., Brown, M., Mian, S., & Sjölander, K. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, *235*, 1501–1531.

Jelinek, F. (1999). *Statistical methods for speech recognition*. MIT Press.

Ling, C. X., & Wang, H. (1997). Alignment algorithms for learning to read aloud. In *Proceedings the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 874–879.

Lippmann, R. P., & Gold, B. (1987). Neural classifiers useful for speech recognition. In *IEEE Proceedings of the First International Conference on Neural Networks*, Vol. IV, pp. 417–422.

Lucassen, J. M., & Mercer, R. L. (1984). An information theoretic approach to the automatic determination of phonemic base forms. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-84*, pp. 42.5.1–42.5.4.

Màrquez, L. (1999). *Part-of-speech Tagging: A Machine Learning Approach Based on Decision Trees*. Ph.D. thesis, Department de Llenguatges i Sistemes Informàtics, Universitat Politecnica de Catalunya.

Màrquez, L., Padró, L., & Rodríguez, H. (2000). A machine learning approach to POS tagging. *Machine Learning*, *39*(1), 59–91.

Qian, N., & Sejnowski, T. J. (1988). Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, *202*, 865–884.

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, *1*, 145–168.