Calibrating Recurrent Sliding Window Classifiers for Sequential Supervised Learning

Saket Joshi & Thomas G. Dietterich
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331

Abstract

Sequential supervised learning problems involve assigning a class label to each item in a sequence. Examples include part-of-speech tagging and text-to-speech mapping. A very general-purpose strategy for solving such problems is to construct a recurrent sliding window (RSW) classifier, which maps some window of the input sequence plus some number of previously-predicted items into a prediction for the next item in the sequence. This paper describes a general-purpose implementation of RSW classifiers and discusses the highly practical issue of how to choose the size of the input window and the number of previous predictions to incorporate. Experiments on two real-world domains show that the optimal choices vary from one learning algorithm to another. They also depend on the evaluation criterion (number of correctly-predicted items versus number of correctly-predicted whole sequences). We conclude that window sizes must be chosen by cross-validation. The results have implications for the choice of window sizes for other models including hidden Markov models and conditional random fields.

1 Introduction

The standard supervised learning problem is to learn to map from an input feature vector $\mathbf{x}$ to an output class variable y given N training examples of the form $(\mathbf{x}_i, y_i)_{i=1}^N$. Many recent learning problems can be viewed as extensions of standard supervised learning to the setting where each input object X is a *sequence* of feature vectors, $X = \langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T \rangle$, and the corresponding output object Y is a sequence of class labels $Y = \langle y_1, y_2, \ldots, y_T \rangle$. The *sequential supervised learning* (SSL) problem is to learn to map from X to Y given a set of N training examples, $(X_i, Y_i)_{i=1}^N$.

Several recent learning systems involved solving SSL problems. One example is the famous NETTalk problem of learning to pronounce English words (Sejnowski Rosenberg, 1987). Each training example consists of a sequence of letters (e.g., "enough") and a corresponding output sequence of phonemes (e.g.,

Table 1: Simple sliding windows

(X, Y)	enough	In^-f-
w_1	___enou	I
w_2	__enoug	n
w_3	_enough	^
w_4	enough_	-
w_5	nough__	f
w_6	ough___	-

Table 2: Recurrent sliding windows

(X, Y)	enough	In^-f-
w_1	___enou__	I
w_2	__enoug_I	n
w_3	_enoughIn	^
w_4	enough_n^	-
w_5	nough__^-	f
w_6	ough___-f	-

/In^-f-/). Another example is the problem of part-of-speech tagging (Màrquez, Padró, Rodríguez, 2000) in which the input is a sequence of words (e.g., "do you want fries with that?") and the output is a sequence of parts of speech (e.g., "verb pron verb noun prep pron"). A third example is the problem of information extraction from web pages in which the input sequence is a sequence of tokens from a web page and the output is a sequence of field labels (Craven, DiPasquo, Freitag, McCallum, Mitchell, Nigam, Slattery, 1998).

In the literature, two general strategies for solving SSL problems have been studied. One strategy, which we might call the "direct" approach, is to develop probabilistic models of sequential data. For example, the hidden Markov model (Rabiner, 1989) is a generative model of the joint distribution $P(X, Y)$ of the object sequence X and the label sequence Y. More recently, the conditional random field (Lafferty, McCallum, Pereira, 2001) has been proposed as a model of the conditional distribution $P(Y|X)$. The advantage of these probabilistic models is that they seek to capture the true sequential relationships that generate the data.

The other general strategy that has been explored might be called the "indirect" approach (i.e., a "hack"). In this strategy, the sequential supervised learning problem is solved indirectly by first converting it into a standard supervised learning problem, solving that problem, and then converting the results into a solution to the SSL problem. Specifically, the indirect approach is to convert the input and output sequences into a set of "windows" as shown in Table 1. Each window consists of central element $\mathbf{x}_i$ and LIC letters of left input context and RIC letters of right input context (in the figure $LIC = RIC = 3$). Contextual positions before the start of the sequence or after the end of the sequence are filled by a designated null value (in this case _).

In most SSL problems, there are regularities in the sequence of y values. In part-of-speech tagging, the grammar of natural language constrains the possible sequences of parts of speech. In text-to-speech mapping, there are patterns in the phoneme sequence. The simple sliding window method cannot capture these patterns unless they are completely manifested in the X values as well, which is rarely the case. One way to partially learn these patterns is to employ *recurrent* sliding windows, in which previous predictions (e.g., for y_{t-1}, y_{t-2}, etc.) are fed back as input features to help predict y_t. During learning, the observed labels in the training set can be used in place of these fed back values. During classification, the sequence is processed from left-to-right, and the predicted outputs $\hat{y}_{t-2}, \hat{y}_{t-1}$ are fed as input features to predict y_t. Note that the sequence could also be processed from right-to-left, but not simultaneously left-to-right and right-to-left. We denote the number of fed back y values as the left output context LOC or right output context ROC, depending on the direction of processing. Table 2 shows the training windows with $LOC = 2$ and $ROC = 0$. The advantage of the recurrent sliding window approach is that it can be combined with *any* standard supervised learning algorithm to solve

SSL problems.

Two practical issues arise in applying either the direct or indirect methods. The first issue is the size of the context. How much of the input sequence and output sequence should be considered at each point along the sequence? For the recurrent sliding window methods, this can be stated concretely as follows: how large should the left input context, right input context, and left output context be? Presumably larger contexts will lead to high variance and overfitting by the learning algorithm, whereas smaller contexts will lead to lower variance but higher bias and underfitting by the learning algorithm.

In addition, one would expect that there would be a tradeoff between input and output context. For a given amount of data, we would expect that a learning algorithm can only handle a fixed number of features. Hence, if output context is increased, then input context needs to be decreased, and vice versa.

The second issue is the criteria for measuring performance. A straightforward generalization of the 0/1 loss for standard supervised learning is to measure the number of whole sequences that are correctly predicted. However, another measure is the number of individual $\mathbf{x}_t$ items correctly classified (e.g., individual letters in NETTalk, individual words in part-of-speech tagging) (Kakade, Teh, Roweis, 2002). An interesting question is whether the optimal choice of input and output context depends on the performance criterion.

In this paper, we present an experimental study of these two issues using a general-purpose recurrent sliding window system RSW that we have constructed as part of the WEKA machine learning system developed at the University of Waikato (Witten Frank, 1999). RSW can be downloaded from our website *http://www.cs.orst.edu/~joshi/RSWdownload.html*, and it can work with any of the WEKA classifiers that return class probability distributions. Our study compares naive Bayes, decision trees, bagged trees, and adaboosted trees on two SSL problems: NETTalk and protein secondary structure prediction. The results are surprising. We find that there is no tradeoff between the amount of input context and the amount of output context. Furthermore, we show that better learning algorithms (e.g., adaboosted trees) are able to handle larger amounts of both input and output context. Finally, we show that the appropriate choice of input and output context depends on the learning algorithm and on the evaluation criterion. This suggests that there are no general-purpose methods for choosing the amount of input and output context in an algorithm-independent way.

2 Experimental Methods

For our experiments, we chose two large datasets. One is the text-to-speech dataset from the NETTalk system (Sejnowski Rosenberg, 1987). This dataset consists of 20,003 words, expressed as sequences of letters and their corresponding pronunciations in terms of phonemic sound representations. Of these, 2000 sequences (words) were picked at random without replacement. This dataset of 2000 sequences was then randomly split into a 1000-sequence training dataset and a 1000-sequence test dataset. Each class label consists of a phoneme-stress pair. For example, in words like "there", the "t" is pronounced as "T>" wherein the "T" is the phoneme label and the ">" is the stress label. There are 56 phonemic sounds produced in English speech and 6 different levels of stress. All legal combinations of phonemes and stresses give 140 class labels. The average length of English words in the data is 7 (minimum 2, maximum 17).

The second problem we studied was the Protein Secondary Structure Prediction dataset employed by Qian & Sejnowski (1988). Their task was to assign each element in a protein sequence (drawn from a 20-letter alphabet) to one of three classes (alpha helix, beta sheet, or coil). Although there are only three classes, the average length of the protein sequences is 169 (minimum 11, maximum 498). Because the sequences are so long, we never observed a case where an entire protein sequence was correctly predicted.

We computed two measures of performance: (a) the percentage of individual elements in the test set predicted correctly and (b) the percentage of whole sequences in the test set predicted correctly. For the latter, all elements in the sequence must be correctly predicted in order for the sequence to be declared correctly predicted.

For each dataset and each performance measure, we applied RSW with $LIC = RIC = 0, \ldots, 7$. This resulted in an input window size ranging from 1 to 15 by steps of 2. For each input window size, experiments were performed with the right output context (ROC) varying from 0 to 7 and then with the left output context (LOC) varying from 0 to 7. This gives a total of 120 experiments with different input and output context combinations.

Each of the 120 experiments was performed with each of four learning algorithms: Naive Bayes, J48 decision trees, Bagged J48 trees, adaboosted J48 trees (boosting by weighting). J48 is the WEKA implementation of the C4.5 algorithm (Quinlan, 1993). For decision trees, we employed pessimistic pruning with five confidence factors (0.1, 0.25, 0.5, 0.75, 1). With bagged and boosted trees, we experimented with 10 to 50 trees.

3 Results

Figures 1,2 and 3 compile the results of the above experiments. Each graph displays a comparison of the performance of various learning algorithms across various input contexts for the best configuration of the remaining parameters. A label next to each performance curve displays the name of the learning algorithm and the output context. The output context chosen for each curve is the one where the performance of the algorithm achieves its maximum. Where there is a difference in performance between the use of left and right output context, it is specified appropriately by an L or an R.

Figure 1 shows the performance curve of the four algorithms applied to the NETTalk data. The performance criterion is the number of letters in the dataset predicted correctly. Naive Bayes works very poorly on this problem. Peak performance is obtained with no output context and an input context of 2 which is a simple 5 letter window. A clear case of overfitting is exhibited as the input context increases.

The best configuration for a single decision tree is the following: input context of 1 (i.e., 3-letter window), output context of 1, and a pruning confidence factor of 0.75. Performance starts dropping slightly for greater input contexts. Bagging achieves significantly better performance, and it attains maximum performance with an input context of 3, output context of 2 (and 50 bagged trees). Boosted decision trees do somewhat better than bagging. They achieve maximum performance with an input context of 4 and an output context of 6. So there is a pattern: as we move from a single tree (IC=1,ROC=1) to bagging (IC=2,ROC=2), to boosting (IC=4,ROC=6), the learning algorithms are able to handle larger input and larger output context.

From this figure, we draw the following conclusions. First, the best configuration

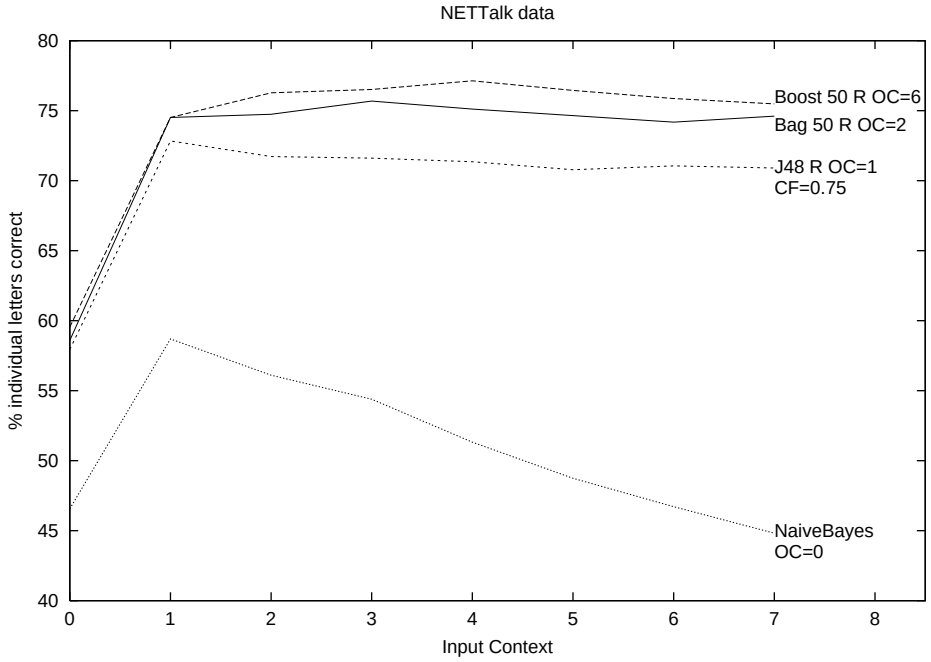

Figure 1: NETTalk: % of individual letters correct

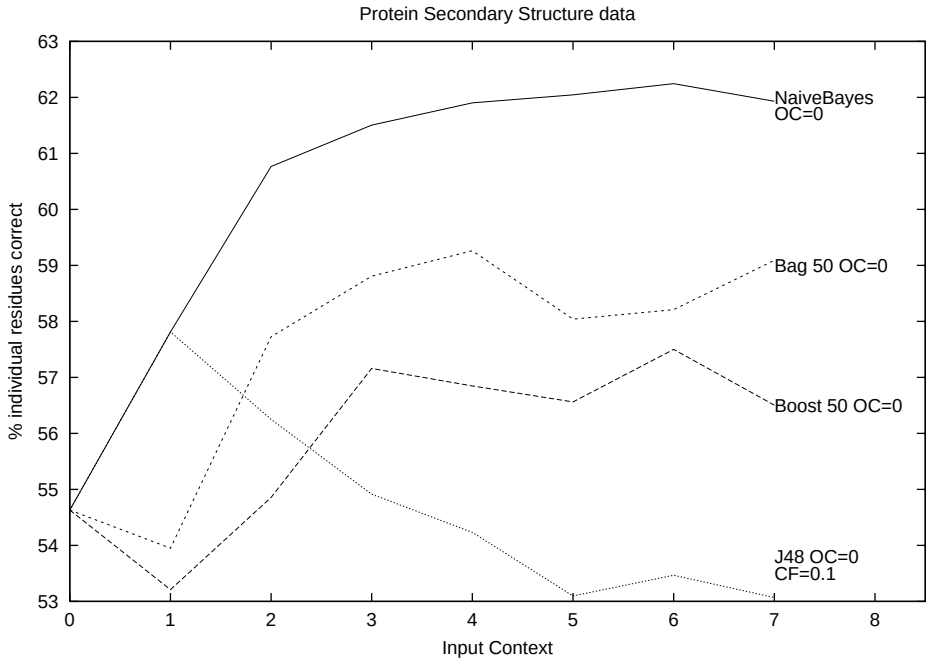

Figure 2: Protein Secondary Structure: % of individual residues (amino acids) correct

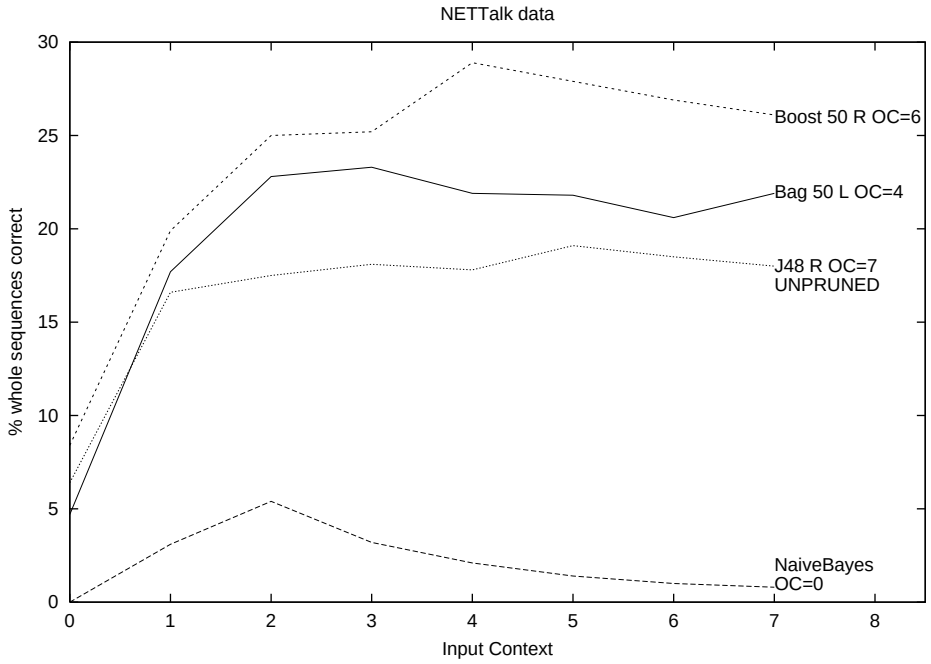

Figure 3: NETTalk: % of whole words correct

depends on the learning algorithm. Second, as expected, all of the algorithms exhibit some overfitting when the input context becomes too large. Third, all of the algorithms give very poor performance with an input context of 0 (i.e., only the one letter to be pronounced). Fourth, all algorithms obtain better results using right output context rather than left output context. This is consistent with the results of Bakiri (1991).

Figure 2 shows the performance of the four algorithms applied to the Protein dataset. In this domain, Naive Bayes gives the best performance (input context=6). The second best is bagged decision trees (IC=4). The third best is a single decision tree with strong pruning (IC=1, pruning confidence 0.1), and the worst is adaboosted decision trees (IC=6). Note that in all cases, an output context of 0 is preferred. Most state-of-the-art secondary structure prediction methods use non-recurrent sliding windows coupled with better input features and additional post-processing of the predictions (Jones, 1999).

Figure 3 shows the four algorithms applied to the NETTalk data again, but this time the evaluation criterion is the number of whole words in the test set predicted correctly. This graph is somewhat similar to the one in Figure 1 except that the peak performances now have different window sizes. Bagging does better with a lower input context and a larger output context. Boosting uses the same output context of 6 but performs better with an input context of 4 instead of 3. A big surprise is that J48 gives its best performance with an output context of 7 (and no pruning). Even Naive Bayes changes its best configuration from input context 1 to input context 2. The major conclusion to be drawn from this figure is that the optimal choice of input and output context depends on both the learning algorithm and the evaluation criterion.

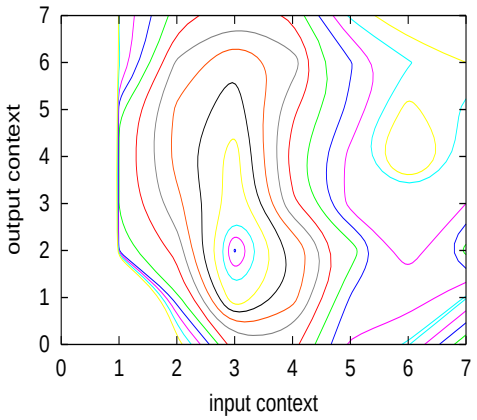

Figure 4: Contour plot showing Bagging performance as a function of input and output context.

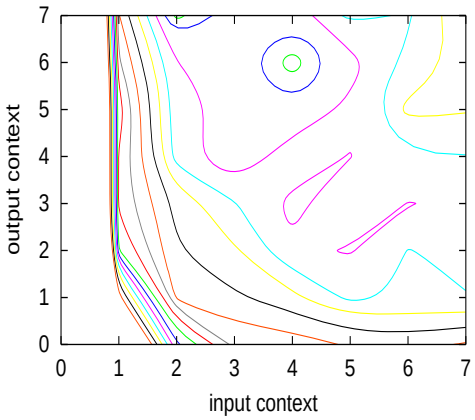

Figure 5: Contour plot showing boosting performance as a function of input and output context.

Figures 4 and 5 show contour plots of the performance of bagging and boosting for various input and right output contexts. For bagging, the maximum at input context 3, output context 2 is clearly visible. However, there is no evidence for a tradeoff between input and output context. For example, decreasing input context does not allow us to increase output context or vice versa. This is even more clear in Figure 5, where we see the exact opposite of the expected tradeoff. There is a large plateau of good performance when the input and output contexts are sufficiently large. Shrinking either input or output context hurts performance.

4 Conclusions

All methods for solving sequential supervised learning problems must confront the question of how much context to employ. For HMMs and Conditional Random Fields, the question of output context becomes the question of the order of the Markov process (first order, second order, etc.). HMMs cannot handle large input contexts (at least not without violating the assumptions of the generative model), but CRFs can. Hence, CRFs also must face the choice of the size of the input context. The choice of input and output context is analogous to classical feature selection. One common approach to feature selection is to fit some simple model to the data and/or compute some figure of merit for the informativeness of each attribute (Kira Rendell, 1992; Koller Sahami, 1996). The experiments reported here show that this will not work for SSL problems, because the correct choice of input and output contexts depends on the learning algorithm. Hence, the input and output contexts chosen by some simple method (e.g., Naive Bayes) are not good choices for boosted decision trees. A consequence of this is that the input and output contexts need to be chosen by cross-validation or holdout methods.

The experiments have also shown the the optimal input and output context depends on the performance criterion. Window sizes that maximize the correct classifications of individual examples are not the ones that maximize the correct classifications of whole sequences and vice versa.

A third conclusion is that there is no trade-off between input and output context. Algorithms that can accept larger input contexts typically can also accept larger output contexts and vice versa. Hence, a simple bias-variance analysis does not

explain our experimental results.

Acknowledgements

The authors gratefully acknowledge the support of the National Science Foundation under grants IIS-0083292 and ITR-0001197.

References

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.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., & Slattery, S. (1998). Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pp. 509–516 Menlo Park, CA. AAAI Press/MIT Press.

Jones, D. T. (1999). Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology, 120*, 97–120.

Kakade, S., Teh, Y. W., & Roweis, S. (2002). An alternate objective function for Markovian fields. In *Proceedings of the Nineteenth International Conference on Machine Learning* San Francisco, CA. Morgan Kaufmann.

Kira, K., & Rendell, L. A. (1992). A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 249–256 San Francisco, CA. Morgan Kauffman.

Koller, D., & Sahami, M. (1996). Toward optimal feature selection. In *International Conference on Machine Learning*, pp. 284–292.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pp. 282–289 San Francisco, CA. Morgan Kaufmann.

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.

Quinlan, J. R. (1993). *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*(2), 257–286.

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

Witten, I. H., & Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, San Francisco, CA.