AN ABSTRACT OF THE THESIS OF

Bill T. Langford for the degree of Master of Science in Computer Science presented on May 13, 1993.

Title: Classification Context in a Machine Learning Approach to Predicting Protein Secondary Structure

Abstract approved: Redacted for Privacy

Thomas G. Dietterich

An important problem in molecular biology is to predict the secondary structure of proteins from their primary structure. The primary structure of a protein is the sequence of amino acid residues. The secondary structure is an abstract description of the shape of the folded protein, with regions identified as alpha helix, beta strands, and random coil. Existing methods of secondary structure prediction examine a short segment of the primary structure and predict the secondary structure class (alpha, beta, coil) of an individual residue centered in that segment. The last few years of research have failed to improve these methods beyond the level of 65% correct predictions.

This thesis investigates whether these methods can be improved by permitting them to examine externally-supplied predictions for the secondary structure of other residues in the segment. The externally-supplied predictions are called the "classification context," because they provide contextual information about the secondary structure classifications of neighboring residues. The classification context could be provided by an existing algorithm that made initial secondary structure predictions, and then these could be taken as input by a second algorithm that would attempt to improve the predictions.

A series of experiments on both real and simulated classification context were performed to measure the possible improvement that could be obtained from classification context. The results showed that the classification context provided

by current algorithms does not yield improved performance when used as input by those same algorithms. However, if the classification context is generated by randomly damaging the correct classifications, substantial performance improvements are possible. Even small amounts of randomly damaged correct context improves performance.

Classification Context in a Machine Learning Approach to Predicting Protein Secondary Structure

By

Bill T. Langford

A THESIS
submitted to
Oregon State University

in partial fulfillment of the
requirements for the degree of

Master of Science

Completed May 13, 1993
Commencement June 1994

APPROVED:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Chair of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis presented May 13, 1993

Typed by Bill Langford for Bill Langford

Acknowledgments

I want to thank a number of people who were important in the development of this thesis. I need to thank my major professor, Tom Dietterich, for the idea for the work and for all of his help and encouragement in bringing it along; my minor professor, Paul Cull, for his help and encouragement throughout my time at OSU; and Steve Giovannoni and OCATE for giving me a place to work and financial support during the last phase of this work. I also want to thank Curtis and Javier at Global Systems Modeling for their financial support when I needed it most and for the good times in Tucson.

Three good friends were especially helpful in getting the thesis written: Phil Maynard, who suffered innumerable dumb questions about molecular biology; Dietrich Wettschereck, who rescued me from daily computational disasters; and Nick Flann, who always revived my interest in computational things with his own enthusiasm and knowledge. Without their help and friendship, this thesis would not have been possible.

Finally, I want to thank the people who got me here: Marie, Sally, John, Casey, Alice, and all of my family; and the people who've kept me (mostly) sane and laughing while I'm here, Jim and Cynthia.

Table of Contents

List of Figures

List of Tables

Classification Context in a Machine Learning Approach to Predicting Protein Secondary Structure

Chapter 1

Introduction

This paper describes the results of research to see whether certain kinds of contextual information can improve the performance of machine learning techniques, in particular, when applied to predicting the secondary structure of proteins from their primary structure. We chose this problem because of its biological significance and its structural similarity to other problems where machine learning techniques using contextual information have had some success in the past.

The paper is organized as follows:

- the problem definition and some biological and motivational background for it;
- a summary of related work;

- an explanation of measures in evaluating performance of various secondary structure prediction methods;
- a description of methods;
- results of each experiment performed;
- comparison to other methods;
- general discussion of results and future work.

1.1 Biological Background

This section discusses the significance of determining the structure of a protein, some of the techniques used for structure determination, and defines some common terms such as primary, secondary, and tertiary structure.

Protein molecules are the basic structural materials of biological systems. The behavior of these molecules is governed by their three-dimensional shape. This shape, as described by the positions of all of the atoms in the protein, is known as its tertiary structure. Knowledge of this tertiary structure is important in trying to understand a molecule's role in biological processes like disease or in designing other molecules to emulate or inhibit its activity, e.g., in designing drugs.

Currently, the most common and most accurate way to determine the tertiary structure of a protein is through X-ray crystallography. Unfortunately, this is an extremely difficult process which takes anywhere from a few months to a few

years to complete for one protein, if the protein can even be crystallized. For this reason, only a few hundred proteins have had their coordinates determined this way.

Given the difficulty of trying to determine protein structure experimentally, biochemists have attempted to build computer models as an alternative. Here again results have been poor. The number of interactions between atoms in the protein and in solvents surrounding it, along with the complexity of the equations describing those interactions have made it computationally intractable to solve the structure of anything but the tiniest of proteins. A different approach comes from examining pieces of a protein's structure that are larger than individual atoms.

Each protein is distinguished by the unique chain of smaller molecules that make it up, i.e., one of twenty particular amino acids. This sequence is known as the primary structure of the protein. There can be anywhere from just a few amino acid residues[1] in the primary sequence of a protein to thousands of them. Each of these constituent amino acids has a specific one letter abbreviation, so a protein's primary sequence is analogous to a unique word written using a 20 letter alphabet.

Biochemists have recently reached the point where they can quickly determine the primary sequence of a given protein using automated processes. The

[1]When amino acids join to form a protein, a piece of each amino acid is lost in forming the bond. The remaining part of the original amino acid is then called the residue. Throughout this paper, the terms amino acid and residue will be used interchangeably.

problem is that this does not give the three-dimensional information required to understand the protein's behavior, just as knowing the letters in a word is not the same as knowing its meaning. However, in the early 1960's Anfinsen [1] showed that the sequence of amino acids which uniquely define a given protein is enough to completely determine its tertiary structure. No other information should be required. Unfortunately, in the last 30 years of research, no one has succeeded in finding a mapping from primary structure to tertiary.

One avenue that people have tried is to break proteins up into regions of structural regularity, called secondary structure. The hope is that secondary structure may be able to reduce the complexity of the problem to understanding local organizing behaviors as building blocks which can then be pieced together to suggest the full tertiary stucture. This is the problem addressed in this paper: predicting the secondary structure from the primary structure.

While there are a number of secondary structures that have been identified, the three classes in this study are called α-helices, β-strands or β-sheets[2], and coils. Helices and sheets are regions where the relative locations of adjacent amino acids repeat in a regular fashion suggested by their names and where the term coil (or random coil) is interpreted to mean anything that is not a helix or a sheet. For the purposes of this study, no more precise definition than that is required. Helix, sheet and coil are simply classifications that the program will attempt to learn

[2]In other papers, the word sheet is often used to describe this class, regardless of whether it is a strand or a sheet. We will generally use just β in order to avoid confusion.

without any real knowledge of what they are.

1.2 Problem Definition

Given this terminology, we can now state the problem examined in this study:

> For each amino acid residue in a given protein, predict which of the three secondary structure classes it belongs to, given the sequence of all residues in the protein.

Note that each prediction is only meant to apply to the given residue at the given position in the given protein. It is not trying to say that any time a leucine residue is found, it is part of a coil. A given residue can be part of a different secondary structure in different proteins and in different parts of the same protein, all depending on the composition of the rest of the protein.

1.3 Basic Approach

Many approaches have been tried for solving this problem. Some of these employ biochemical knowledge or statistical characteristics of known structures as the basis for predicting structure. However, there does not seem to be enough biochemical knowledge or statistical information available now to make good predictions on that basis, so inductive learning approaches have become more common.

The machine learning techniques are all similar in that they are repeatedly given a set of training examples consisting of:

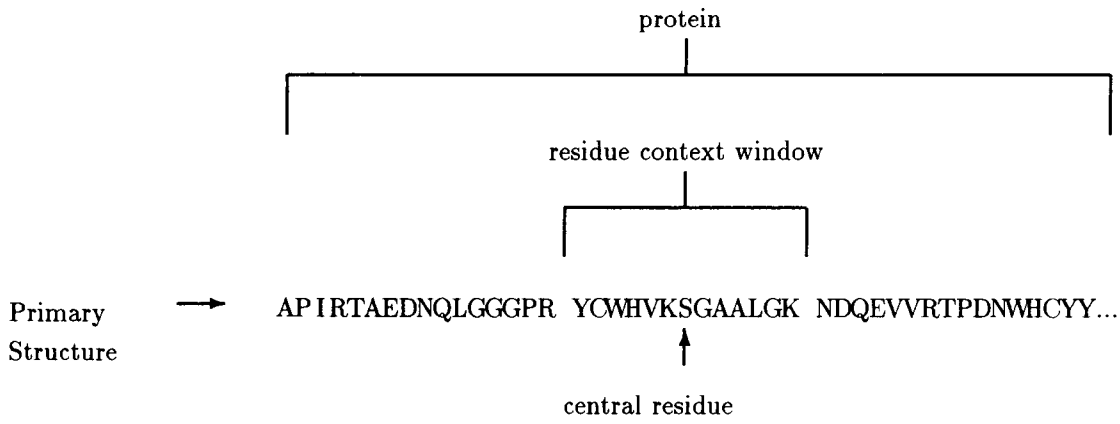


Figure 1. Residue Context Window

- the residue where the prediction is to be made
- the correct secondary structure classification for that residue
- the sequence of residues in a fixed size window around it (called the residue context), e.g., the 6 neighboring residues on either side.

For example, consider Figure 1.3:

Once the program can predict a certain percentage of classifications correctly, training is stopped and the program is given a separate test set of residues and their window of neighboring residues and it attempts to predict the classifications for these previously unseen windows.

The approach investigated in this paper involves a standard machine learning technique like that described above[3] differs in that the information in the contextual window is expanded beyond the residue context to include something

[3]The machine learning technique here is Quinlan's C4.5 decision tree program [2, 3, 4] as modified by Dietterich and Bakiri to incorporate error correcting codes [5] [6].

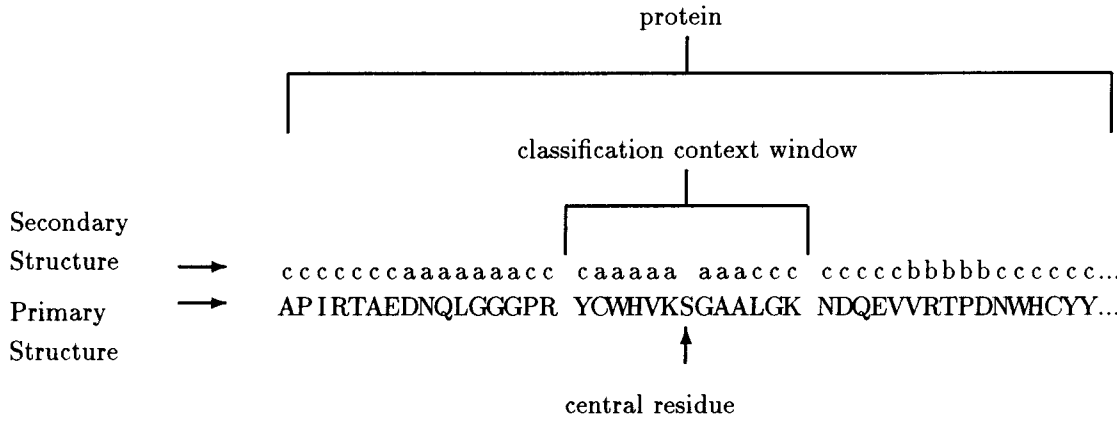


Figure 2. Classification Context Window

which we will call the classification context. This classification context gives a secondary structure classification (which may or may not be correct) for each position in the residue context.

The example shown in Figure 1.3 above would then be modified as shown in Figure 1.3 to include the classification context:

The next section will explain some reasons for investigating the addition of classification context.

1.4 Motivation

One of the first machine learning papers published on secondary structure prediction was that of Qian and Sejnowski in 1988 [7]. Sejnowski had previously done successful work on the NETtalk text-to-speech system [8] where the problem formulation was very similar to that of secondary structure prediction. Windows of letter sequences from a word were given as the input and a pronunciation for the

central letter was produced as the output classification, just as windows of residues are given for a protein and each central residue's secondary structure classification is given as the output.

This similarity suggested that they might get good results with secondary structure as well, so they mapped the text-to-speech system into the secondary structure prediction domain. Their results were the best anyone had produced at the time and still remain among the best.

After Sejnowski's work on NETtalk, Dietterich and Bakiri [5] applied error-correcting codes to the same problem and improved on Sejnowski's results. In that work they found that providing classification context in addition to letter context improved their results. Given these facts and Qian and Sejnowski's protein work, a natural question is whether the same error-correcting code and extended context improvements can be found in the protein domain as well.

Furthermore, since classification context is not required to come from any particular source, another question arises. Could a classifier be built which gets its classification context in the form of predictions from several methods to produce a joint classifier that is better than any single classifier?

So, given all of these general questions, a number of experiments were done in an attempt to answer the following specific questions:

- How does C4.5 with error-correcting codes perform on secondary structure prediction in comparison with other methods like neural networks?

- Does classification context improve classification accuracy?
- Is improvement contingent on knowledge of the bias of the provider of the context? Does that knowledge help?
- If there is improvement, are there bounds to the improvement?

Chapter 2

Related Work

Many different approaches to predicting secondary structure have been tried over the last 30 years.[4] So many different approaches have been tried, that it is necessary to restrict discussion here to a few representative groups of methods that can be compared to the work described in this paper.

The groups discussed here are:

- non-learning techniques (i.e., those based on stereochemical knowledge or statistical information)
- machine learning techniques,
- joint or combined prediction techniques.

Techniques not discussed here include those based on:

- sequence homology (as in [12])

[4]Methods suggested before 1988 have been exhaustively surveyed in [9]. Other good surveys that are less comprehensive but more recent include [10] and [11].

- those that only predict one specific class (e.g., β-Breakers as in [13]),
- those that predict all classes for only one specific protein (as in [14]).

All of the techniques discussed here are, in theory at least, designed to predict helix, sheet and coil classifications in arbitrary globular proteins without use of homologies. While the most successful way to predict the structure of a protein is usually through the use of a homologous protein whose structure is already known, that information can not be assumed to be available. Therefore, the techniques under consideration must be able to predict structure without knowledge of homologous proteins.

In the discussion that follows, little mention is made of the predictive accuracy of any of these methods because it can be very misleading without some explanation. A separate section (Measures of Results) will detail the difficulty in comparing the results of any of these methods and the Results section of this paper will show some comparisons assuming knowledge of the Measures section. In general though, in cases where homologies are removed from training and test sets, results are no better than around 65% correct. In particular, nearly all of the neural net methods are in this neighborhood.

2.1 Non-learning Techniques

According to Fasman [9] the most common of the non-learning prediction schemes are: Lim's method [15], the GOR method [16], and the Chou-Fasman method

[17, 18, 19].

Lim's method is based on stereochemical knowledge of proteins and is unusual in that it attempts to take long-range interactions into account. In particular, it considers some of the hydrophobicity and geometry constraints involved in forming a compact globular structure.

Both the GOR and the Chou-Fasman methods are based on statistical characteristics of a database of known proteins rather than on stereochemical knowledge. The GOR method computes an information theoretic measure of the relationship between residues in a window and the classification of the central residue of the window. The Chou-Fasman method computes frequencies to identify probable nuclei of sheets and helices and then extends those nuclei. Since Chou-Fasman is the most frequently referenced method, a more detailed explanation of how Chou-Fasman works is beneficial.

The Chou-Fasman method works by looking for nuclei of helix or sheet structures and then extending the nuclei out in either direction until a set of residues is found which are not amenable to the structure being extended. The easiest way to see how this works is to look at one type of structure, e.g., helices. The method for sheets is similar. Regions not classified as helix or sheet are classified as coil.

A helix nucleus is identified as a section of 6 consecutive residues containing 4 residues that have high helix former potential and no more than 1 residue with high helix breaker potential. These *former* and *breaker* potentials are defined for

each of the 20 amino acids using normalized frequencies derived from a database of 29 known structures.

The helix potential for a given residue is defined to be the ratio of the average frequency of that residue in helices over the average frequency of all residues in helices, e.g., for alanine:

$$P_\alpha(Ala) = \frac{f_\alpha(Ala)}{f_\alpha(all\ residues)}$$

where

$$f_\alpha(Ala) = \frac{\#\ Ala\ residues\ in\ helices}{\#\ Ala\ residues}$$

and

$$f_\alpha(all\ residues) = \frac{\#\ residues\ in\ helices}{\#\ residues}$$

Therefore, a P value greater than 1 indicates an above average rate of occurrence for a particular residue in a particular secondary structure. Chou-Fasman assigns 6 helix former/breaker classes to residues based on their P values [20]:

- H (strong helix former, $P_\alpha >= 1.34$),
- h (helix former, $1.12 <= P_\alpha < 1.34$),
- I (weak helix former, $1.00 <= P_\alpha < 1.12$),
- i (helix indifferent, $0.77 <= P_\alpha < 1.00$),

- b (helix breaker, $0.61 <= P_\alpha < 0.77$),
- B (strong helix breaker, $0.53 <= P_\alpha < 0.61$).

Once the nuclei are identified using the P values, each nucleus is extended out in either direction until a helix breaker section is found, i.e., a 4 residue section where the average value of the helix potential is less than 1.

One problem with the Chou-Fasman procedure is that some regions may be equally disposed to α and to β and therefore, the resulting classification is ambiguous.

2.2 Machine-learning Techniques

A number of people have applied neural networks to secondary structure prediction. Each approach has been slightly different, but all have had roughly the same performance, i.e., somewhere in the range of 60 to 65% correct.

Qian and Sejnowski [7] and Holley and Karplus [21, 22] were among the first to publish results using neural networks. Both groups tested many different network architectures, parameters, and input window sizes. While each group chose different configurations as optimal, the performance of their systems was similar. Their work on optimizing the learning parameters provided the basis for much of the work done in this paper, particularly in regard to choice of window size, input encoding, and test and training data.

Qian and Sejnowski created a data set of 106 proteins which they carefully divided into a test set and a training set with almost no homologies between the two and with the percentages of α, β and coil similar in both sets. This removal of homologies is important in being able to assess performance and it was generally not done before that time.

In testing many variations of their network's configuration, they found their optimal performance on the training data occurred using 2 cascaded networks in the following configuration:

- They found a 13 residue input window optimal.
- The first network had 21 input nodes for each residue in the 13 residue window and had 3 output nodes, one for each of the 3 classes to be predicted.
- The second network took for its input, the first network's 3 classification outputs for each of the 13 residues in the window. Its output was a single set of 3 nodes, one for each possible structure. This second network was intended to clean up the output of the first network and did improve their results for α helix and coil predictions.

In their experiments, Holley and Karplus settled on a slightly different network from that of Qian and Sejnowski.

- They found a 17 residue input window optimal instead of 13, although they found very little difference in performance between the two different window sizes.

- Instead of 3 output nodes, they had 2, one each for α and β, with coil predicted if α and β activations were below a threshold.

- They had no second cascaded network to clean up the output. They had a post-processing program which made sure that helices and sheets were at least a minimum length or else reassigned to coil.

An important observation that Holley and Karplus made was that the accuracy of their network's output was related to its certainty. The higher the level of activation for a classification, the more likely that the classification was correct. In a test where they achieved 63% overall accuracy, they found that the 34% strongest predictions were correct 78% of the time [21].

Both groups chose local binary input encodings (i.e., one bit per residue) rather than meaningful distributed codes. They each tried using input codes where each bit represented some characteristic of the given amino acid (e.g., hydrophobicity), but neither group found that it helped their performance.

One difference between the two groups was in how they determined when to stop training. Holley and Karplus stopped training when their total output error became asymptotic. Qian and Sejnowski on the other hand, stopped training when their system achieved the best performance on the test set. This may make their performance appear better than it really would be when applied to a true unknown test set.

Kneller et al. [23] duplicated and extended Qian and Sejnowski's work to include two other types of information: periodicity and structural classification.

Since α-helices and β-strands are periodic structures, they looked for measures of periodicity in the primary sequence which might be appropriate as input. They achieved small gains by adding an input for the helix hydrophobic moment and the strand hydrophobic moment, but they found no improvement when they included periodicity of oppositely charged residues as an input.

They were more successful using structural classification. Other researchers such as Deleage and Roux [24] and King [25] have suggested using structural classes like those in Levitt and Chothia [26] to break proteins into groups which may be more easily classified by an expert specific to that group.

Kneller et al. broke their data into classes of all-α, all-β, α/β (sheets and strands roughly alternating), and other. They then trained and tested separate networks for each of these classes on the smaller data sets. They achieved significant improvements for the all-α and all-β classes. While there is not yet an algorithm that can exactly predict which of the structural classifications a particular protein will fall into, these experiments did show that it is worthwhile information to have for secondary structure prediction.

Work by Nakashima et al. [27] and Muskal and Kim [28] provide this kind of structural classification information. Nakashima et al. predict the structural classification with approximately 70% accuracy. As an alternative, Muskal and Kim predict the total helix and strand content instead.

Interestingly, Muskal and Kim are able to predict total helix and strand content to within around 95% accuracy using a neural network with only 22 inputs. There is one input for the total number of each amino acid in the protein, one for the molecular weight of the protein, and one to indicate whether the protein contains a heme group. No knowledge whatsoever of the primary sequence of the protein is necessary to capture this tertiary characteristic of the molecule.

Stolorz, Lapedes and Xia [29] tested both a neural network and a Bayesian approach to secondary structure classification. Their network was different from those of previous researchers in that it combined a mutual information and mean-square error as its objective function, but its accuracy was very similar to that of previous neural net researchers. Their results using a Bayesian prediction method were only slightly less accurate than the neural net results.

2.3 Joint/Cooperative Techniques

Since none of the predictive methods alone have had much success in spite of exploiting different types of information, many researchers have tried to combine methods to build a joint predictive method, e.g., [30, 31, 24, 32, 11, 33, 34]. Several of these are discussed below.

Perhaps the simplest approach is to have several methods make a guess at the structure and then take an unweighted vote. This is the approach in [31]. They looked at 8 different methods and picked the 5 best, then combined the pre-

dictions of these 5 by majority vote. They found that the results for the combined guesser were better than the results for any of the individuals though still not much improvement.

This kind of unweighted combiner makes no use of any knowledge of what each individual method might be good or bad at. Zhang et al. [30] built a hybrid system where 3 predictors fed their guesses into a neural network which learned to combine them into a single prediction. Again, the combined guesser outperformed each of the individual predictors (a neural network, a nearest neighbor algorithm, and a statistical model). While they found that 20% of the time, all 3 predictors made the same wrong guess, they also found that 77% of the time, at least one of the predictors made the correct guess. This suggests that improving the combiner may have the possibility of substantially improving results beyond the current 65% average (although 3 constant predictors would result in at least one always being correct).

Maclin and Shavlik [35] took a different approach. Rather than combining outputs of different methods, they proposed directly embedding the knowledge of one method into another. They did this by building a finite state automaton describing the Chou-Fasman method and using that as the framework for the structure of a neural network. This system, called FSKBANN, got better results than their tests using a neural net that was not initialized using the Chou-Fasman information and better than those of many other researchers. However, these results may not be equivalent to those of Qian and Sejnowski.

Their set of proteins was the same as Qian and Sejnowski's but it appears that they divided it into testing and training randomly, without regard for homology. Moreover, no mention is made of whether there is overlap between the proteins used to develop the Chou-Fasman frequencies and the test sets Maclin and Shavlik developed, so information about the test set may be inherent in the initialized network. Still, if there is overlap, it would be possible to redo this test by building Chou-Fasman numbers based solely on the training set.

Chapter 3

Standards for Comparing Results

Comparing prediction results in the secondary structure domain is not as easy as it is in some other research areas. There has never been a canonical phrasing of a secondary structure problem akin to computing's travelling salesperson problem. Consequently, different authors have tried many different data sets and performance measures. Only recently have they been more consistent in trying to carefully control the construction of data sets and to suggest biologically relevant performance measures.

This chapter will define a group of performance measures to be referred to throughout the rest of this paper. First it will survey some of the problems inherent in choosing test and training data sets, then it will discuss some of the uses of secondary structure predictions. These uses will provide a motivation for the data sets and measures described in the last section of the chapter.

3.1 Data Sets

One of the primary problems in trying to compare results is the inconsistency among the data sets chosen by various authors. To compare two methods, the two data sets must either be identical or similarly biased. Moreover, to evaluate one method alone, the data sets can not be biased in relation to that method.

There are two types of data sets to consider. First, there is the training set, which means whatever set of proteins are the basis for deriving the rules or knowledge which will be attempt to predict structure. Second, there is the test set, which is some protein or set of proteins whose structure will be predicted.

3.1.1 Comparability of Training Sets

The most important issues in comparing two training sets are their size and their composition. Of these two, the composition is the most important and must be considered in relation to the test set. This relation will be discussed below under bias.

Currently, the size of the training set does not seem to be the limiting factor in performance once it reaches a certain threshold. Qian and Sejnowski [7] tested their method using a number of different training set sizes. As they increased the size, they found a rapid increase in peformance up to about 5000 examples, but after that there was very little benefit in increasing the size of the training set.

3.1.2 Comparability of Test Sets

Since much of the work in this field has been motivated by practical biological problems, many of the studies try to predict the structure of a single protein of interest to the author rather than the structures of a standard group of test proteins. This makes it difficult to evaluate the prediction method, since it may do very well on the given test protein but poorly on others whose structures or primary sequence are significantly different.

Even authors who do try to predict the structures of a set of proteins rather than a single protein often make comparisons difficult by choosing a completely different set of proteins than those of previous authors. If the proteins chosen for the test set match the bias of the training set this can give the method artificially high scores, since it is not generalizing over truly unknown data.

3.1.3 Bias of Data Sets

In this discussion, the bias of a data set will be interpreted as an increased likelihood of making a correct prediction for certain types of proteins. For example, most machine learning schemes would be more likely to make a correct prediction if you gave them a protein whose structure was nearly identical (i.e., homologous) to one in their training set than if you gave them a randomly chosen protein from outside their training set. They are biased towards the types of proteins contained in their training set.

In most of this paper, bias is viewed as a 'bad' thing in that it can make general comparisons between methods difficult. However, it should be noted that bias may be very useful in some situations, e.g., when you know that you have a homologous protein or when you are building a joint predictor and you know that one method is good at predicting a particular type of structure and you want to build another method that is better at predicting other types of structures.

Two major sources of bias in secondary structure studies are sequence homology and relative proportion of secondary structures.

The relative proportion of each secondary structure in the training set will have an effect on the classifier's ability to predict that type of structure. For example, if the training set contains few helices, then the classifier is unlikely to predict helices' well in a general test sample containing helices. Therefore, the relative proportions in the training set should match those expected in the world where predictions will be made. Moreover, if the performance measure is to reflect the ability of the classifier to generalize, the test set should also contain relative proportions that reflect those of the real world.

Relative proportions themselves must also be examined. Some proteins are composed of several identical units which are repeated, for example, hemoglobin has 4 parts, 2 sets of 2 symmetric pieces. Using more than one identical unit from a given protein can bias the classifier toward the knowledge of that unit even though the relative proportions of each secondary structure were reasonable.

The relative proportions of structures are much less of a problem in evaluating studies of secondary structure prediction than is homology. Homology is meant here as similarity of primary sequences.[5] Many studies have derived their method from proteins that are very similar to those on which they evaluated their method. Since primary sequence similarity generally (but not always) implies secondary structure similarity, these classifiers can appear to be very successful even though they may fail on a test population that has no homologies with their training population.

In some domains such as text-to-speech conversion, overlap or similarity between the test and training sets is at least somewhat reasonable since part of the final system's job is to perform on examples it was trained on as well as ones that it has never seen before. In secondary structure prediction, performance on the training set is irrelevant since the structure of the training examples is already known.

3.2 Measuring Performance

In order to compare secondary structure predictions, some knowledge of what will be done with them is helpful. Are they only useful if they predict to a certain level of percent correct? Are there qualitative differences between predictions with

[5]In the protein literature, homology means this "similarity in primary sequence", but the more precise definition used by molecular phylogenists is one of "sharing common evolutionary origins".

the same percent correct that could justify using one method but not another? Is secondary structure even worth predicting, especially if predictions are far from perfect?

3.2.1 Uses for Secondary Structure Predictions

The primary motive for predicting secondary structure is as an aid to predicting tertiary structure. Fasman [9] discusses three typical approaches to predicting tertiary structure: sequence homology, packing of secondary structures, and ab initio energy calculations. Only the second of these approaches makes explicit use of secondary structure predictions. He discusses a typical use of these predictions in a three step procedure: predict structures, pack them into a reasonable overall fold, and refine the fold with energy calculations.

Some researchers have suggested that anything less than a perfect prediction of secondary structure will not provide a suitable starting point for tertiary energy minimizations [36]. Whether this is true or not, other researchers continue to refine their predictions of secondary structure.

The difficult part of the three step scheme mentioned above is the packing of structures into a fold, since secondary structures do not specify exact coordinates of the atoms involved. This is particularly true for random coil predictions, as their name implies.

One approach to packing structures is through the identification of super-secondary structures where specific sequences of secondary units pack in a char-

acteristic pattern, e.g., $\beta/\alpha/\beta$ or a series of α helices. Several algorithms have been proposed for recognizing these supersecondary motifs [37, 38]. Taylor and Thornton's algorithm [37] takes an initial secondary structure prediction and then matches it against a $\beta/\alpha/\beta$ template.

In this process, the exact boundaries of α and β structures are not as important as their existance and order. Therefore, a secondary structure algorithm which predicts the proper number and order of structures is more valuable than one which predicts more residues correctly, but does not get the proper set of structures. This shows where the simple total percent of correct residues can be misleading as a measure.

Fasman mentions a number of other applications of secondary structure predictions in [39]. In particular, he mentions looking at ambiguous predictions to help determine regions where a protein's structure may change due to mutation or change in solvent. This means that some measure of the certainty of a prediction may have a value beyond determining the confidence of a prediction.

Another use that he mentions is in determining structural homology between two proteins. As in the case of the supersecondary structure mentioned above, identification and order of structures is more important than total residues predicted correctly since matching of residues is not exact in structural homology.

3.2.2 Measures

The main idea to be derived from the brief survey of the uses of secondary structure is that statements about one method being several percentage points or "a statistically significant" number of percentage points better than another in total residues predicted correctly are not worth much because, in that measure, something close to perfection seems to be important. Other measures which express more qualitative information about the predictions are more meaningful.

These qualitative measures relate to whether an observed structure is predicted or not and how much overlap there is between the prediction and the observed structure. However, there is no general agreement on how these measures should be computed. Some suggestions are given in [37, 35, 30], but implementation of these suggestions is beyond the scope of this study and left for future work.

Another important facet of measurement which is beyond the scope of this study is looking at the results for smaller aggregates than an entire data set. There are many ways to break the data down which may provide better understanding of performance than by looking at the test set as a whole. Different proteins of different sizes, compositions and supersecondary structures exhibit different characteristics which are more or less difficult to learn [23, 25]. Even distance from the N-terminus vs. distance from the C-terminus has been shown to have significance in some cases [21, 9, 40].

The work in this paper applies the inferior but more common performance measures described below to the results given by classifiers on their test set. Specifics of these data sets are given in the Methods section of this paper. The measures are computed for the entire data set rather than by protein or any other subdivision of the data sets, e.g., by supersecondary structure. Again, this is left for future work.

Each of the following measures is computed:

- Percentage of residues predicted correctly [7]

$$Q3 = \frac{\mathrm{P}_\alpha + \mathrm{P}_\beta + \mathrm{P}_{\mathrm{coil}}}{N}$$

where

P_α = number of αs predicted correctly

P_β = number of βs predicted correctly

P_{coil} = number of coils predicted correctly

N = total number of residues

also

$$Q3_\alpha = \frac{P_\alpha}{N_\alpha}$$

and same for β and coil

- correlation coefficient by structure type [41]

$$C_\alpha = \frac{(P_\alpha \times N_\alpha)(U_\alpha \times O_\alpha)}{\sqrt{(N_\alpha + U_\alpha)(N_\alpha + O_\alpha)(P_\alpha + U_\alpha)(P\alpha + O_\alpha)}}$$

where

P_α = number of positive cases that were predicted correctly

N_α = number of negative cases that were rejected correctly

O_α = number of false positives (over predicted cases)

U_α = number of misses (under predicted cases)

Same thing for β and coil.

- Percent of predictions of a given structure type that were correct (which we will call the hit rate for the class):

$$Hits = \frac{P_\alpha}{P_\alpha + O_\alpha} = \frac{num\ of\ correct\ \alpha\ preds}{total\ num\ of\ \alpha\ preds}$$

Same for β and coil.

- Percent of observed structures that were correctly predicted (which we will call the true positive rate for the class):

$$TruePos = \frac{P_\alpha}{P_\alpha + U_\alpha} = \frac{num\ of\ correct\ \alpha\ preds}{total\ num\ of\ observed\ \alpha s}$$

Same idea for β and coil.

Chapter 4

Methods

We performed a number of experiments in which we varied certain aspects of the contextual information provided to the learning algorithm, for example the amount and correctness of the context. Each of the experiments had certain elements in common and those will be discussed in this section. In subsequent sections, each of the different experiments will be described and the specifics of each will be explained there.

4.1 Data Sets

The data for these experiments is the same data analyzed by Qian and Sejnowski and was obtained from them. This data was chosen because it contains a large number of examples and was carefully constructed to balance the proportions of the different types of secondary structure and to remove homologies between the test and training sets.

The data was divided 4 ways. They originally divided their data into a training set and a non-homologous test set. For these experiments, the non-homologous test set was left alone, but the full training set was split into two parts for some of the tests and left alone for others.

The division of the full training set into partial training set and cross-validation set was done by randomly selecting a small number of proteins to remove from the full training set and calling that the cross-validation set. Unlike the non-homologous test set, no attempt was made to screen for homologies between the partial training set and the cross-validation set.

The makeup of each of the sets is shown in Appendix A.

We should note that there is one question about the data sets. We do not know their makeup in terms of supersecondary structures. Later in this paper we will discuss the fact that some researchers have gotten better results when they trained specific classifiers to deal with each of several types of supersecondary structure.

Supersecondary structure describes a higher level of organization of pieces of secondary structure in a protein. For example, a protein may be made up of alternating helices and sheets or it may contain no sheets at all and simply be a sequence of linked helices.

The improved predictive ability due to knowledge of supersecondary structure implies that there may be some correlation between predictive ability on two proteins even if they are not homologous but do have the same supersecondary

structure. In that case, the makeup of training and test sets may also need to be monitored for supersecondary structure content. For the tests in this study though, this issue has been ignored.

4.2 Learning Algorithm: C4.5 with Error-correcting Codes

The learning algorithm for building the classifiers was Quinlan's C4.5 decision tree algorithm [2, 3, 4] modified with error-correcting codes [5]. We chose this system because previous work had shown that it produced results at least as good as neural network algorithms in the NETtalk domain but it requires far less training time. In most cases, a particular classifier could be trained in approximately 1 day of background CPU time on a Sun SPARCstation 2 as opposed to weeks of training time for a neural network. This short training time allowed us to do many more tests than would be possible using neural networks.

It should be emphasized here that the learning algorithm itself is treated as a black box (as much as possible) in this study. The assumption is that using a different learning algorithm in place of C4.5 would not produce significantly different relative results. In particular, given a specific learning algorithm, the relative performance induced by providing different types of classification context to that algorithm would remain the same regardless of learning algorithm. This assumption should be tested, but that is beyond the scope of this study.

4.3 Secondary Structure Classifications

For the purposes of this study, secondary structures were divided into three classifications: α, β, and coil. These are also commonly referred to as (α) helix, (β) sheet, and random coil. However, more than 3 types of secondary structure have been identified, e.g., 3_{10} helices and β turns.

Some studies work with some of these other classes. In particular, there has been a lot of work on predicting β turns [42, 43, 44]. These more detailed classifications may be important, but the Qian and Sejnowski data was only classified according to the 3 basic classes so that was the limit of this study.

4.4 Triples Classifications

Even though the final goal was to predict whether each residue was part of a helix, a sheet, or a coil, these classifications were not used directly. Instead we built a slightly more complex set of composite classes because 3 classes are not enough to allow the use of error-correcting codes.

The composite classes were defined to be triples of the normal classes, i.e., if a given residue would normally be classified as a helix and so would both of its nearest neighbors in the sequence, then the given residue's composite class would be helix/helix/helix. Since there are three basic classes (helix, sheet, and coil) and three possible positions in the triple, there can be 27 possible composite classes.

(Residues at the end of a protein arbitrarily assign the off-end neighbor as being coil.)

In practice only 19 composite classes are defined because 8 combinations are chemically meaningless, e.g., sheet/helix/sheet is not allowed since a helix must have more than one residue. Table 4.1 lists the classes defined and the ones that were disqualified since they were meaningless.

Reducing these composite "triples" classifications to the original secondary structure classifications is trivial since the program only has to look up what is the middle element of the triple. (In the rest of this paper, the secondary structure classifications will often be referred to as the "reduced" classifications or "singles" classifications to contrast them with the composite "triples".)

4.5 Training Procedure

Each protein in the training set was broken up into a set of 13 residue windows, one window for each residue in the protein. The window contained the identity of the residue to be classified, the identities of each of its 6 nearest preceding neighbors in the sequence and of its 6 nearest following neighbors in the sequence. In tests where classification context was provided, identities of the classifications of the same 6 preceding and 6 following residues were also provided as part of the window.

Table 1. Triples Classifications

Classes Used	Illegal Classes
coil-coil-coil	
coil-coil-β	
coil-coil-α	
	coil-α-coil
coil-β-β	
	coil-β-α
	coil-α-coil
	coil-α-β
coil-α-α	
β-coil-coil	
β-coil-β	
β-coil-α	
β-β-coil	
β-β-β	
β-β-α	
	β-α-coil
	β-α-β
β-α-α	
α-coil-coil	
α-coil-β	
α-coil-α	
	α-β-coil
α-β-β	
	α-β-α
α-α-coil	
α-α-β	
α-α-α	

Note that no classification was provided for the central residue whose classification was to be predicted. This was done in order to keep from making the classification problem too easy in tests where correct classification context was given and to keep all of the tests in this study as comparable as possible. In a true cooperative system where classifications were being provided from an external source, this central classification *would* probably be provided.

4.6 Input Encodings

4.6.1 Residue Encodings

The identities of the residues were presented to C4.5 using a simple one-per-class local encoding scheme. Each of the 20 residues had a code where one bit was set and all the rest were zero. There was also a special residue code for "off-end" when the input window overlapped the end of the protein.

As mentioned in the Related Work section, other researchers [7, 21] have tried using more complicated distributed encodings where each bit represents some property of the residue, e.g., whether the residue is hydrophobic or not. However, they found that the distributed encodings were not helpful.

4.6.2 Classification Encodings

Like the residues, the triples classifications were represented for input using a one-per-class scheme for each of the 19 legal triples classes.

4.7 Output Encodings

Each of the 19 legal triples classes was assigned a 50 bit error-correcting code word. These codes were generated by an annealing procedure which produces a set of codes whose average Hamming distance to any other code in the set is at least 15 bits. The set of codes for these experiments is shown in Appendix B.

50 bits was chosen as the length of the code in order to keep the training time down. Longer codes can increase the accuracy of predictions but the ability to do more tests was more important in this study.

4.8 Testing Procedure

The procedure for testing was to break the test file up into windows in the same way as was done for training and build an input file. This file was fed to a program called ecc which read a window from the input file and then used the decision trees built by C4.5 to compute a Hamming distance to the output encoding of each of the 19 legal triples classifications. All of those distances were written to an output file for analysis later.

The procedure for computing the distances to each of the classes is as follows:

- Each of the 50 bits of the code has a corresponding decision tree.
- Each of the trees was applied to the input window to produce a value indicating the likelihood for that bit of the code to be turned on, i.e., a value between 0 and 1.

- For each of the 19 classes, the difference between each bit of the classifier's guess (the likelihood of that bit being on) and the corresponding bit of the code for the given class was computed.

- The sum of the absolute values of the differences for all of the bits in each code was defined to be the distance to the given code.

For example:

- Suppose that there were only 2 classes instead of 19 and 3 bits in the code instead of 50. Also suppose that the code for class 1 is (1,1,0) and for class 2 it is (1,0,1). Since there are 3 bits in the code, there would be 3 decision trees.

- Suppose that when given a particular window, the 3 decision trees were applied and gave the likelihoods for each of the bits in the code to be on as follows: (.05, .90, .60).

- Then, the hamming distance to class 1 is defined to be:

$$HD_1 = |1 - .05| + |1 - .90| + |0 - .60|$$

$$= .95 + .10 + .40 = 1.45 \; bits$$

and for class 2:

$$HD_2 = |1 - .05| + |0 - .90| + |1 - .60|$$

$$= .95 + .90 + .40 = 2.25 \; bits$$

- So, class 1 would be chosen as the classification since its distance of 1.45 bits is closer than class 2's distance of 2.25 bits.

For each residue window, once the distance to each class was computed, the list of distances was sorted and the classification for that residue was chosen to be the code word with the smallest Hamming distance. This list of distances also provided a way to measure the certainty of the classification.

Three certainty measures were investigated: distance to the nearest code word, spread between the two closest code words, and a combination of the two measures. For example, if the closest code word was 7 bits away and the second closest was 22 bits away, then one certainty measure would have been 7, the second measure would have been $22-7 = 15$, and the final measure would have been (7,15).

4.9 Special Terms

Certain terms appear with a very specific meaning throughout the rest of the paper:

- Residue context

  The identities of the residues immediately preceding and succeeding the residue to be classified.

- Classification context

  A set of classifications for residues in a neighborhood of the residue attempting to be classified. For example, in the typical case the classification context

is the classifications for the six preceding residues and the six succeeding residues. The term classification context is intended to differentiate this kind of information from the context used in other studies, i.e., the residue context.

- Random

  Many of the experiments described below talk about some "random" values for context. These values are not truly random but are values generated through the random number routines of the gnu C++ compiler's library. Absolute randomness is not necessary for these tests since the tests only suggest general trends for the results.

- Triples classes

  One of the 19 classes defined by concatenating three secondary structure classifications (and throwing out illegal combinations).

- Reduced classes

  The basic secondary structure classifications obtained by extracting the center secondary classification from a triple. Same as a secondary structure classification.

- Secondary structure classifications

  One of: α, β or coil. Same as a reduced class.

- Data sets
  - Full training set

    The entire training set, not partitioned.
  - Cross-validation set

    The small group of proteins randomly selected from the full training set be used only for cross-validation.
  - Partial training set

    The full training set minus the cross-validation set.
  - Non-homologous test set

    Qian and Sejnowski's test set. Homologies with the full training set have been removed by Qian and Sejnowski. Not part of any training set.
- Observed class

  This is the "correct" secondary structure classification that is assigned by looking at the X-ray diffraction data.
- Predicted class

  This is the class which the classifier has assigned to a residue, as opposed to the observed class.

Chapter 5

Description of Experiments

5.1 Overview

The end of chapter 1 posed a series of specific questions that this study was designed to answer. This chapter describes the experiments which were performed in hopes of answering those questions.

The simplest of the questions posed is how does C4.5 with error-correcting codes perform in secondary structure prediction compared to other machine learning methods? A straightforward experiment was done to answer this question by feeding C4.5 the same input that was given to Qian and Sejnowski's neural network, that is, windows containing the identities of the residue to be classified plus the identities of the 6 preceding and 6 succeeding residues in the sequence.

Attempting to answer the rest of the questions required a more complicated set of experiments. The basic idea in those experiments was to test whether classification context information helps improve secondary structure prediction accuracy.

Context is interpreted here as any kind of information about something other than the residue that the system is currently trying to classify. Two types of context are of interest: residue context and classification context. Looking at a residue to be classified, residue context contains the identities of some of its neighboring residues, for example the 6 residues on either side mentioned above. This residue context is the same context that has been supplied by most other researchers in their experiments. What most other researchers did not supply was classification context.

Classification context is a set of classifications of the neighboring residues viewed as input. Possible sources of classification context might be: other classification methods, random guesses, or the classifier's own previous guess fed back to itself. Other methods would include things like Chou-Fasman, neural net predictions, or predictions based on physical measurements like circular dichroism.

The primary experiment in this study looks at a pipeline of classifiers as the context source. In this pipeline, each step is a classifier which takes its predecessor's guesses as its input (i.e., as its classification context) and attempts to improve on them.

The point of using this pipeline is to try to exploit the different levels of knowledge inherent in the problem. At one level, we can look at the tendencies of specific residues and series of residues to adopt certain secondary structures. For example, we know that the structure of a proline residue prohibits it from being in an α helix or a β sheet so it must always be classified as a coil.

At another level, we know that secondary structures have their own characteristics regardless of which residues they contain. For example, an α helix needs at least 4 residues to make one turn and be considered a helix at all. Moreover, the continuity of a helix makes it impossible to have a string of helix residues interrupted by a single β residue ($\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha$ vs. $\alpha\alpha\alpha\alpha\beta\alpha\alpha\alpha\alpha$). A classifier that only looks at the residues themselves is unlikely to discover this higher level knowledge even though it is readily visible to a person looking at the output of the classifier.

Some researchers have explicitly encoded such rules in output cleanup programs (e.g., [21]), but there may be other characteristics of the secondary structures that an inductive learner could pick up. The classification context experiments in this study examine whether a classifier can be provided with classification context and find inherent higher level secondary structure information to improve its performance.

Once it is decided to include classification context in input, there are two important questions to answer:

- Should some of the classification context be screened from the classifier since it may include a lot of noise?
- What should the classifier be given as classification context during training?

In order to answer the first question it helps to reexamine the idea of the pipeline of classifiers. The simplest way to envision a pipeline as a context source is to

have a classifier make a series of classifications where at each step, its own last set of guesses is its next input classification context. In this simplest case, the pipe is just a loop with only one stop. However, the poor performance of all current classifiers suggests that much of the context supplied in this case would be wrong and probably not help.

A slightly different way to employ this pipeline as context is to look at a classifier's guesses and try to determine which of them are likely to be correct and only allow those in the context. In that case, the guesses which are thought unlikely to be correct are declared to be missing and the input noise is reduced. This is the scheme which this study was originally designed to test.

Regardless of whether all of the classification context is given to the classifier, creating training examples for the classifier will present a problem. The most obvious thing to try is to have the classifier preceding it in the pipeline classify all of the examples in the training set and make that the training context.

Unfortunately, since there is not much training data, that preceding classifier was probably trained on the same training data and will classify it much more accurately than it would classify previously unseen data like the test set. This is likely to cause problems for the newly trained classifier when it encounters unseen data in the test set. Consider the following example:

Assume some classifier C_1 has already been trained and a new classifier C_2 is to be trained with the intent of applying C_2 to the output of C_1 in order to improve C_1's results. Also assume that due to the small amount of data available,

C_1 and C_2 must both be trained on the same training set of proteins, P_x, and tested on the same test set of proteins, P_y. When C_1 is finished training on the proteins in P_x, it gets 80 or 90% of the classifications correct. However, when C_1 is applied to the unseen test set P_y, it only gets about 60% correct.

Now C_1's classifications of P_x are considered the classification context for training C_2. This means that C_2 sees context that is 80 or 90% correct. Once C_2's training is complete, C_2 is applied to the unseen test set P_y. Since C_2 expects classification context as part of its input, C_1's classifications of the proteins in the test set P_y are supplied as input.

The problem is that C_2 has been trained to expect the classification context to be 80 or 90% correct but the context that it gets is only 60% correct. This difference between what it gets and what it expects is likely to affect C_2's ability to classify correctly.

The primary experiment in this study attempts to remedy this problem by training C_2 on classification context that has been built to mirror the types of errors that C_1 is expected to make on unseen data. This training context is built by splitting the training data into a smaller training set and a cross-validation set. C_1 is trained on the partial training set and then tested on the cross-validation set. The confusion matrix resulting from this test is then a guide for generating errors in the classification context for training C_2. Details of this process will be discussed in the section describing these confusion matrix distribution context experiments (Section 5.3).

This idea of pipelining a series of classifiers is not restricted to receiving classification context just from other classifiers trained in exactly the same way. A pipelined classifier could be trained to receive its classification context input from any kind of classifier, for example from the other methods mentioned above (Chou-Fasman, etc.). This raises several questions:

- How important is it that the classifier know the kinds of errors and bias in the context that it is given?
- How much and what kind of noise (i.e., incorrectness) can the classifier tolerate in its input context?
- What are the bounds on the classifier's performance in relation to the noise in its classification context input?

In order to try to answer these questions in general rather than for a specific method of input, a number of experiments were done using simulated classification context generated to have specific characteristics. This simulated input was created by taking correct classification context and either damaging it with some probability or removing it with some probability. The idea was to simulate classification context that was generated by some method which achieved a specific performance level, for example, 70% correct.

This simulated input allowed us to explore what a classifier might be able to do given various levels of input correctness which are not currently achievable and thereby get some idea of the bounds of the usefulness of classification context. It

should be noted that this type of randomly damaged input is somewhat unrealistic since it is equally likely to get easy to guess and hard to guess classifications correct, which a real method would not. Still, it gives a feel for whether a classifier based on reading classification context could ever have any power.

So, given this motivation, here is a short summary of the groups of experiments that we did:

- Baseline experiments:
  - No classification context

    This first experiment was done just to see how C4.5 with error-correcting codes compared with other methods without any context and with the tests done here using classification context.
  - 100% correct classification context

    The second experiment was done as a control to verify that C4.5 was capable of learning that if the two adjacent residues were classified correctly, their classifications contained the correct classification for the central residue.
- Classification context experiments:

  Confusion matrix distribution context and simulated context
  - More probably correct context containing missing values

    These experiments looked at what happens when classification context is screened to allow only values that are known to be absolutely correct

or believed to be correct with a given level of certainty. They employ both real data fed back to a classifier trained using a confusion matrix distribution and simulated data given to a classifier trained on randomly damaged data.

- Full noisy context, no missing values

  These experiments looked at what happens when the classifier is given a value for every member of the classification context, but that value may be wrong. Again, they use the confusion matrix and simulation techniques.

Each of these experiments will be discussed in separate sections below, but here is a summary of the results.

Overall, almost every variation of the tests came up with something a little under 50% correct in predicting the triples classes when not given artificial context. However, when these results were reduced to singles classes (i.e., α or β or coil), they were generally predicting at a little under 60% which is at least in the neighborhood of the other published schemes.

The classifiers trained with artificial context like the full random correct context (i.e., randomly damaging correct context classifications at various percentages and not leaving any as missing values) provided much better performance, but only on test data that had randomly damaged context too. When the test data had more realistic context like that supplied by the output from of one the other

classifiers instead of by randomly damaging correct context, performance fell back to the 60% level again.

5.2 Classification Context Generation Methods

Before we discuss each of the experiments, we will explain the two major classification context generation methods in the experiments: confusion matrix distribution and data simulation.

Both of these methods attempt to generate training data that reflects the nature of the mistakes that a classifier might see in classification context that is piped into it from some other source of classifications. The data simulation approach also generates context for testing.

There are two sets of circumstances that can occur in training a classifier with classification context as input. In one case, you know something about how the input source behaves, i.e., what kinds of mistakes it makes. In the other case, you know little or nothing about the kind of mistakes it makes. For example, you may only know some gross statistical characteristics about the source, such as the total percentage of residues it is likely to classify correctly.

We investigate the case where something is known about the input source's mistakes by using the confusion matrix distribution to generate classification context for training. We investigate the the case of poorly characterized error behavior through simulated data as training and testing context.

5.3 Confusion Matrix Distribution

When a classifier is tested on a set of examples, we can generate a matrix of counts of correct classifications vs. classifications actually made by the classifier. This is called a confusion matrix.

5.3.1 Example of Confusion Matrix

For example, suppose that we have three classes, A, B and C, and a test set containing 1200 test cases. Now suppose that when we run the classifier on the test set, it classifies 60 of the examples that should be A's as A's, 10 that should be A's it classifies as B's, and 30 that should be A's as C's. So on class A examples, the classifier is right 60% of the time, but it splits its mistakes between class B and class C.

Similarly, for examples that should be classified as B's, suppose it classifies 40 of them as A's, 35 as B's, and 25 as C's. Finally, suppose that examples which should be C's are classified as 100 A's, 200 B's, and 700 C's. The resulting confusion matrix is shown in Table 5.3.1.

This confusion matrix gives an indication of the kinds of errors that the classifier makes. In addition to showing how often the classifier makes the correct prediction, it shows how often it happens for each particular class and it shows how often it mistakes each other class for the correct class.

This gives us much more information about the bias of the classifier than just

Table 2. Example Confusion Matrix

		Predicted Classes		
		A	B	C
	A	60	10	30
Correct	B	40	35	25
Classifications	C	100	200	700

saying that it is right 66% of the time ($.66 = (60 + 30 + 700)/1200$). For example, we can tell that we should have more confidence in the classifier's prediction of class A (60% correct) than of class B (35% correct). This is the basis for the experiments using the confusion matrix distribution for training.

5.3.2 Generating Classification Context for Training Using a Confusion Matrix

Suppose we want to train classifier C_2 to receive classification context as input from classifier C_1. The idea is to create training examples for classifier C_2 that mimic the distribution of errors made by C_1. That distribution of errors is shown in a confusion matrix for C_1.[6]

When we create training examples for C_2, we already have the set of pro-

[6]This may sound like there is just one confusion matrix for C_1, but there is no unique confusion matrix for C_1. There is a separate matrix for each data set that C_1 is tested on. The choice of data set to derive the confusion matrix from is important and will be discussed below.

teins chosen and we know each residue's correct classification. What we don't know is what to provide as the classification in the classification context so that it will reflect the kind of behavior that C_1 would exhibit if it was providing the classification context. An example with a tiny input window may help here.

Suppose that you have an input window of 3 residues and the current input window is centered on residue number 125 in the sequence of one protein. Suppose the training window looks like this:

Position	124	125	126
Residue	Q	Y	G
Correct secondary class	A	A	C
Classification Context	?	-	?

In the experiments in this study, no classification context is ever provided at the center residue of a window, so nothing is necessary at position 125 in this window. The question is, what should go in place of the question marks for positions 124 and 126?

As mentioned in the overview in Section 5.1, we could provide C_1's actual classifications of those residues in the training set, but they are unlikely to contain errors since C_1 was trained on the same proteins. Later on, when C_1 is given previously unseen proteins such as those in the test set, it will make far more

errors and C_2's training needs to reflect this noisy input.

If we assume that the example confusion matrix in Table 5.3.1 was derived from testing classifier C_1 on some data set whose error distribution we want to mimic, we can derive classification context for positions 124 and 126 from that matrix.

At position 124, the correct class is A, but according to the confusion matrix, classifier C_1 only guesses residues of class A correctly 60% of the time. 10% of the time it guesses B and 30% of the time it guesses C when it should have guessed A. The classification context generator will take this information and should give:

a 60% chance of returning class A,

a 10% chance of returning class B,

a 30% chance of returning class C,

as the classification context for a position which is correctly classified as class A.

Position 126 is treated similarly, only the context generator is basing its output on class C's distribution instead of class A's, since C is the correct classification for that position. This gives:

a 10% chance of returning class A,

a 20% chance of returning class B,

a 70% chance of returning class C,

as the classification context for a position which is correctly classified as class C.[7]

[7]One technicality that occurs with real data is when an instance of some obscure class may occur

5.3.3 Source Data Set for the Confusion Matrix

The one thing that has not been mentioned here is the choice of test data set for creating the confusion matrix. Any test set will have its own corresponding confusion matrix, since that matrix just shows what guesses the classifier made for the classes that it saw, and those are unique to each test set. The resulting confusion matrix may or may not be anything like that which C_1 will produce when it tries to classify the final non-homologous test set which is the performance measure in these experiments.

The hypothesis of these experiments is that the closer C_2's expectations match C_1's output, the more likely it is that C_2 will be able to improve its performance based on C_1's classifications. If this hypothesis is true, then the ideal confusion matrix would be the one that C_1 actually produces on the non-homologous test set, since that means C_2 would have been trained to expect exactly the error distribution that C_1 does produce on the test.

However, in a fair test, the distribution of C_1's errors on the non-homologous test data set would not be known when C_2 is built, since the test set is supposed

very, very rarely and be included in the training set but not in the test set since it is so rare. This means that there is no example of that class in the confusion matrix. In that case, the training context generator still needs to return some value for the classification context but has nothing to base it on. This was handled by simply classifying those instances as coil/coil/coil since that was by far the most common classification in general. Since these cases are very rare, it should have little impact on the experimental results.

to represent unseen data. To be more explicit, C_2's performance on the test set is supposed to be representative of how C_2 will perform when given real proteins in the future that were not part of its training. Since we have no idea what those proteins will be, we have no idea how C_1 would classify their residues. Having C_1 classify the residues in the test set is equivalent to assuming that we know these future unknown proteins.

Still, we can do an unfair test to examine our hypothesis to see how important it is to exactly mimic C_1's error distribution. We just cannot use the results of that unfair test to infer anything about how C_2 would behave on unseen proteins when it was trained on an error distribution which did not necessarily match the one that C_1 would generate on those unseen proteins. In order to look at that question, we need to have a different source for the confusion matrix.

That source is the cross-validation data set. A small subset of proteins in the training set was chosen at random and removed from the training set to produce two data sets: the partial training set and the cross-validation data set.

Using this approach, C_1 is trained on the partial training set and then tested on the cross-validation set to produce a confusion matrix. C_2 is then trained on the partial training set plus the classification context generated using C_1's cross-validation confusion matrix. The hypothesis here is that the cross-validation data set will produce an error distribution similar to that of the non-homologous test set and therefore allow C_2 to do well on that final test set.

There are a couple of possible problems here that are immediately apparent. The first is that the reduction in the size of the training set may significantly reduce the performance of each of the classifiers. The second is that since the cross-validation data set is chosen at random from the training set, it no longer has the careful balancing of classes and the careful removal of homologies that the final test set has in relation to the full training set.

This may mean that the cross-validation confusion matrix is not very representative of the non-homologous test set confusion matrix. This may occur in two ways. Since homologies have not been screened out, performance on the cross-validation test set will probably be better than on a non-homologous set. Also, if the relative proportions of the classes do not reflect the proportions in unseen proteins, the training examples may not be very helpful for predicting in unseen proteins.

These possibilities are examined via the unfair test discussed above, where C_1's true confusion matrix for the non-homologous test set is used.

5.3.4 Data Simulation

In order to look at the utility of classification context which could have come from sources whose characteristics are unknown, experiments were done where simulated classifications were generated. These simulated classifications had only gross statistical characteristics related to the percentages of classifications that were correct and the percentages that were missing. Errors were distributed by

randomly damaging correct classifications rather than dictated according to some known distribution like the confusion matrix.

These randomly damaged classification context inputs do not accurately reflect what most classifiers would produce, since most classifiers tend to find certain types of classifications easier than others. These experiments were done in spite of that inaccuracy for several reasons.

The primary reason is that it is difficult to imagine other ways to produce a classification context that intrinsically reflects a bias that is unknown. While a random assignment of errors is not perfect, it does suggest the notion of something that is not known ahead of time.

Another reason was to try to get some feel for whether there were bounds on what help classification context could provide if source classifiers were better than they are now. That is, if we could improve the performance of normal classifiers to some particular level, could we then know that a new classifier trained using that level of context correctness could get the right answers? Or, on the other hand, is it pointless to even look at classification context because no matter how well the normal classifiers do, the new classifier trained to expect their output is still unable to improve on their performance?

The final reason for ignoring the inaccuracies of using random damage was that there may be certain situations where something approaching random damage to the classification context is not a bad approximation to the error distribution. This situation might arise if a classification recognition approach is substituted for

a classification generation approach. This kind of approach is not considered in this paper but is defined and discussed under Future Work in Chapter 10.

5.3.5 Types of Simulated Data

Two types of simulated data were created: full, noisy context and correct but partial context.

The full, noisy context tests how much noise a classifier can tolerate in its classification context. The classifier is trained and tested using classification context that contains a value for every residue in the window (other than the central residue). Each of these values though, has a specified probability of being correct. For example, a classifier would be trained on context where only 10% of the classifications in the context were correct. Another would be trained on 20% correct data, and so on.

The correct but partial context tests the sensitivity of a classifier to missing classifications in the context. In these tests, not every residue in the window would be given a classification context value. Those residues which were not classified were assigned a value indicating that they were missing. Those residues which were given a classification context value were always given the correct value. Again, some specified probability dictated what percentage of the residues in the protein would be assigned context classifications. So, for example, a classifer would be trained with 30% of the classifications missing and another with 20% missing and so on.

Chapter 6

Baseline Experiments

6.1 No Classification Context

6.1.1 Hypothesis

In order to determine what effect classification context has on a classifier's performance, a measure of the classifier's performance without that context is necessary as a baseline. This experiment attempts to establish what is the baseline performance by training a classifier using the residue context without any classification context.

The results here are expected to be in the neighborhood of those achieved by Qian and Sejnowksi [7] since the problem and methods are similar to those compared in the text-to-speech domain [6]. In those experiments, the error-correcting code approach achieved better results than those of Sejnowski and Rosenberg in NETtalk.

6.1.2 Experiment

A classifier was trained using the full training set and tested on the non-homologous test data set. Since later tests would also need the partial training set and cross-validation data set, another classifier was trained on just the partial training set and then it was tested on both the cross-validation test data set and the non-homologous test data set.

All of these tests were done using a window of 13 residues and no classification context. This window size was held constant at 13 in order to make all of the tests comparable and since other researchers had found 13 to be either the optimal size [7] or that there was very little improvement through increasing the window size [21].

6.1.3 Results

Since this experiment is the baseline for all of the other tests and since it is very comparable to the work of other authors, some extra data will be reported here that will not be shown for most of the other experiments. In particular:

- The comparison of the total percent correct for the triples classes vs. the total percent correct for the reduced classes is provided in order to show the typical difference between the two.

- The results on testing the classifier against the training sets is provided for comparison with other papers who report this information. Since perfor-

mance on the training set is almost always extremely good but not representative of performance on unseen data, subsequent sections will not show results for performance on the training set.

- The confusion matrices shown in Tables 6 and 6.1.3 are provided in order to show the kinds of wrong guesses that the classifier makes.

Table 3 shows the total percent correct for both the triples classes and the reduced classes. Subsequent tables will only show results for reduced classes since that is what is of interest in these studies. The difference in percent correct between the triples and reduced classes was generally in this same 10% range.

The number in this table which is most generally compared with other studies is the Reduced Q3 value for the classifier trained on the full training set and tested on the non-homologous test set, i.e., 59.09%.

Table 4 shows the correlation coefficients for each of the reduced classes. Again, the values for the classifier trained on the full training set and tested on the non-homologous test set are the ones generally compared to other studies, i.e., $C_\alpha = 0.24$, $C_\beta = 0.18$, and $C_{coil} = 0.31$.

The values in Table 4 show that the classifier is best at predicting coil and worst at predicting β. This is the case with most other algorithms as well.

Table 5 gives some indication of the reliability of the predictions for each of the secondary structure types. The table shows that the classifier is likely to find almost all of the coil residues ($P_{TrueP} = 85.13\%$) but at the expense of

Table 3. No Classification Context - Percent Correct for Triples vs. Reduced

Trained On	Tested On	Triples Q3	Reduced Q3
Full	Non-Homologous	47.07	59.09
	Full	75.34	84.86
Partial	Partial	74.33	83.82
	Cross-Validation	50.63	62.76
	Non-Homologous	46.11	58.52

Table 4. No Classification Context - Correlation Coefficients by Secondary Structure

Trained On	Tested On	C_α	C_β	C_{coil}
Full	Non-Homo—ogous	0.24	0.18	0.31
	Full	0.79	0.73	0.72
Partial	Partial	0.79	0.69	0.70
	Cross-Validation	0.35	0.28	0.36
	Non-Homologous	0.22	0.19	0.30

Table 5. No Classification Context - Prediction Percentages by Reduced Class

Trained On	Tested On	α		β		Coil	
		TruePos	Hits	TruePos	Hit	TruePos	Hits
Full	Non-Homologous	36.51	45.06	17.78	46.67	85.13	64.27
	Full	79.83	89.04	62.95	93.93	95.27	81.44
Partial	Partial	79.34	89.01	58.26	92.50	95.35	80.32
	Cross-Validation	48.75	53.80	23.03	57.27	83.94	66.47
	Non-Homologous	34.63	43.82	18.18	46.58	84.76	63.75

overpredicting them, i.e., only 64.27% of the predicted coils are actually coils. As before, β prediction is bad, with only 17.78% of the β residues found and not even half of the predicted ones correct. Again, α's fall in between β's and coils, but much closer to β's.

The relative success in predicting coils is not surprising since over half of the training examples are coils. The lack of success for β is also not surprising since β sheets are likely to involve interactions between strands not contained in the same input window and therefore the algorithm lacks the information necessary to recognize the relationship. While α helices are more capable of standing alone than β strands, they too are often involved in packing arrangements with other structural elements outside the input window and therefore might be expected to fall somewhere between β and coil in predictive accuracy.

Table 6 shows the full confusion matrix for the classifier trained on the training set and tested on the non-homologous test data set. The most noticeable feature of the matrix is that it shows that the classifier almost never guesses transition classes, i.e., almost all guesses are $\alpha\alpha\alpha$, $\beta\beta\beta$, or ccc.

Table 6.1.3 shows the reduction of the full confusion matrix in Table 6 to singles classes, i.e., secondary structures. It shows how the classifier grossly overpredicts coil (72% predicted vs. 55% observed) and underpredicts β (8% predicted and 21% observed).

6.2 100% Correct Classification Context

6.2.1 Hypothesis

Since the triples classification for any residue contains the reduced classification of each of its adjacent residues, C4.5 should be able to derive the correct triples and reduced classifications for any specific residue given the correct triples classifications of its neighbors. This experiment should be trivial but was done as a control because subsequent experiments asssume that C4.5 is capable of learning this relationship.

6.2.2 Experiment

A classifier was trained using the full training set and tested on the non-homologous test data set. Each training and testing example consisted of the residues in the

Table 6. Full Confusion Matrix - Triples Class Counts

Observed Classes	Predicted Classes																		Total Observed
	CCC	ccb	cca	cbb	caa	bcc	bcb	bca	bbc	BBB	bba	baa	acc	acb	aca	abb	aac	aab	AAA	
CCC	1290								2	52									145	1489
ccb	117									11									16	144
cca	49								1	2									17	69
cbb	103									17									28	148
caa	58									5									10	73
bcc	118								1	10									10	139
bcb	3									1										4
bca	2																			2
bbc	96									29									20	145
BBB	239								1	85									125	450
bba	1									1									2	4
baa	3																		1	4
acc	56									3									15	74
acb																				0
aca	2																			2
abb	1																			1
aac	56									3									17	76
aab	1																			1
AAA	351	1								61									282	695
Total Predicted	2546	1	0	0	0	0	0	0	5	280	0	0	0	0	0	0	0	0	688	3520

Table 7. Reduced Confusion Matrix - Secondary Structure Percentages

		Predicted Classes			
		α	β	coil	Total Observed
Observed Classes	α	8.81	1.96	13.35	24.12
	β	4.97	3.78	12.50	21.25
	coil	5.77	2.36	46.51	54.63
Total Predicted		19.55	8.10	72.36	100

window and the correct classifications of each of the residues in the window (except the central residue, as usual).

Unlike all of the other experiments, tests were done using not only a window of 13 residues, but also with a window 3 residues since that window should give the classifier all the information it needs.

6.2.3 Results

As expected, C4.5 learned all of the training and testing examples correctly, regardless of whether the window size was 13 residues or only 3 residues. Therefore, subsequent tests can assume that C4.5 is capable of learning the overlap of triples when their classifications are correct.

Chapter 7

Partial Classification Context, Missing Values

7.1 Random Missing and 100% Correct Context

7.1.1 Hypothesis

Since C4.5 can learn to use full classification context that is always correct, it may also be able to learn to use context where some of the values are missing, if the context that is there is always correct.

Since all of the given context values are correct, i.e., there is no noise in the context, the classifier should do no worse than the classifier which receives no context. Moreover, it should do at least as well as the probability of deducing the correct class from the neighboring triples classes.

If you have triples classes, the learning algorithm should learn that it can often find the classification of a given residue from its neighbors just as it did in the full correct context test. In fact, if you only give the algorithm every third

classification, that should be enough for it to get every classification correct. For example, given every third triples classification as below:

... __ __ $\beta\beta$c __ __ ccc __ __ c$\alpha\alpha$ __ ...

you can deduce the following reduced classifications:

... __ _β_ _β_ _C_ _C_ _C_ _C_ _C_ _α_ _α_ ...

So, any amount of correct classification provided from 33% up, if it was evenly spaced in the string, should yield close to 100% correct classification regardless of whether the learner learned anything at all about the residue patterns themselves. However, if the locations of the missing values are randomly chosen, a higher percentage of non-missing values will be required since the triples overlap will not be perfect.

Probability of Deducing Correct Singles Classification

What is the probability that a given residue will have the correct classification available to it if there is a certain percentage of correct classifications provided at randomly chosen locations in the string? Given a residue X whose left neighbor is residue L and its right neighbor is residue R, the probability P of being able to deduce X's correct singles classification from its neighbors' triples classification is:

P(c) = 1 − (probability that classifications of both L and R are missing)

where c is the percentage of context provided

P(c) = 1 − (P(notL) × P(notR))

where P(notL) is the probability that X's left neighbor's classification has not been provided

For example, if 70% context is provided:

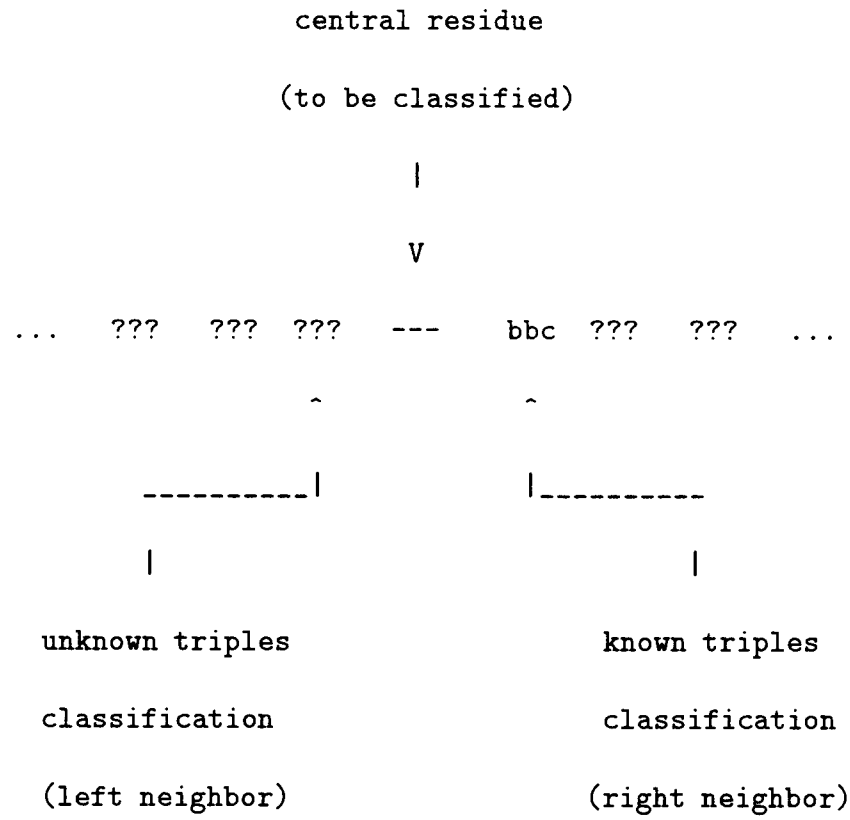
P(.7) = 1 − (.3 × .3) = 1 − .09

= .91

Probability of Deducing Correct Triples Classification

We should note here that this probability of deducing a central residue's singles classification correctly from a neighbor's triples classification is somewhat higher than the probability of getting the central residue's triples classification correct. An example should clarify this.

Assume the following set of classifications (where "???" means that the triples classification is missing for that residue):



From this you can deduce the following singles classifications:

... _?_ _?_ _?_ -B- _B_ _C_ ??? ...

which, in turn, gives 2/3 of the central residue's triples classification, i.e., "?bb". The left neighbor provides us with no information on the central residue's triples classification since it is unknown. However, if the left neighbor's left neighbor was supplied:

```
                    central residue

                          |

                          V

...   ???   aaa  ???   ---    bbc  ???   ???   ...

                  ^

                  |

      left neighbor's left neighbor
```

We can now deduce the following singles classifications:

```
...   _A_   _A_  _A_   -B-    _B_  _C_   ???   ...
```

and therefore the central residue's triples classification: "abb".

All of these positional dependancies make the true triples probability calculation much more complicated than for the singles classification. Moreover, since some of the triples are not legal there are only 19 legal triples instead of the 27 possible triple permutations, which means that there need to be class dependant calculations as well.

These calculations are not done here since they do not provide information that is particularly useful. This is especially true since the classifier can still get a singles classification correct even when it gets a triples classification wrong as long

as it guesses a triples class whose reduced class is the same as the correct reduced class. The explanation here is only provided for completeness.

7.1.2 Experiment

A set of experiments was run to see what effect various levels of missing data had on performance of the classifier. Classifiers were trained using the full training set and tested on the non-homologous test data set. Each classifier had a specific percentage of classification context missing and the rest of the context was correct. Training and testing was carried out as follows:

Each training and testing example consisted of the residues in the window and classifications of each of the residues in the window. Each classification was either a value indicating the correct classification or a value indicating that the classification was missing.

For each protein, the program that generated the data files would produce a classification string where each position had a chance of being declared missing according to the percentage missing for that classifier. For example, if the classifier was to have 70% missing data, then a random number routine was called to generate a number between 0 and 99 and if the value was less than 70, the value for that position was encoded as missing. If the value was greater than or equal to 70, then the correct classification was injected for that position. No incorrect classifications were ever provided.

Note that these experiments are simulations, not real data. The non-homologous test data set has its context assigned randomly just like the training set. In a real situation, it is very unlikely that context anywhere near absolutely correct would be available. The purpose of this test is just to separately measure the classifiers' sensitivity to missing data when no noise is present. Later experiments will describe tests where data contains noise with varying degrees of realism.

7.1.3 Results

Classifiers were trained for 4 different percentages of correct classification context: 70, 30, 20, and 10. The results are shown in Tables 8 and 9. These percentages are chosen to be mostly at the lower end since that is closer to reality and since the problem is too easy for the classifier with high percentages of correct context since there is so much overlap in the triples.

As Table 8 shows though, the percent correct on output (Q3) is still fairly good even down to the 10% context level (71.99% correct). As expected, none of the classifiers do worse than the baseline classifier in Section 6.1 which received no context. They also do better than their probability of deducing the singles class from the neighbor.

However, the correlation coefficients fall off much faster than the total percent correct. As with the no context case, Table 9 shows that the classifiers are being conservative in predicting α and β while overpredicting coil.

Table 8. Random Correct and Missing Classification Context

% Correct	P	Q3	C_α	C_β	C_{coil}
70	.91	97.78	0.98	0.96	0.96
30	.51	88.58	0.85	0.81	0.77
20	.36	80.80	0.74	0.67	0.63
10	.19	71.99	0.57	0.48	0.49

(where P is as defined above in the Hypothesis section)

These results show that the classifiers are able to exploit the knowledge in the context but still have the same weakness with regard to recognizing structures with possible long-range interactions, i.e., αs and βs.

7.2 Thresholded Confusion Matrix Distribution Context

7.2.1 Hypothesis

So far, we have considered full, absolutely correct classification context and then partial, absolutely correct context. Now we want to look at the effects of degrading this context one step further. In the experiments of the previous section, we found that even providing as little as 10% of the absolutely correct classification context significantly improved performance (from 59.09% correct to 71.99% correct). Here we will attempt to provide partial classification context that is not absolutely correct, but very likely to be correct.

Table 9. Random Correct and Missing Classification Context - Prediction Percentages by Reduced Class

% Correct	α		β		Coil	
	TruePos	Hits	TruePos	Hit	TruePos	Hits
70	97.53	98.92	94.92	98.61	99.01	96.99
30	85.75	92.04	78.07	91.25	93.92	86.45
20	71.85	88.02	61.23	85.77	92.36	77.45
10	56.07	76.40	39.57	77.49	91.63	70.06

At this point, the next logical step in the degradation of the classification context would be to perform an experiment just like the previous one with one small change. Instead of just randomly allocating missing classifications we should also randomly damage the the classifications that are not missing. For example, 70% of the classifications could be missing and the remaining 30% could be correct 90% of the time. This would allow us to measure the incremental effect of damage to the provided classifications on top of the effect of removing large parts of the classification context (as measured in the preceding experiment).

Even though this was logically the next step, the experiment was not done because of the large number of other experiments that had to be done and because it was not the primary interest of the study. Instead, we performed what would be a later step in the series. In that step, the effect of providing context that is likely to be correct but taken from a known distribution rather than a random

distribution was measured. Unfortunately, this confounds the effects of degrading classification accuracy and biasing which locations are more likely to be correct, but interest in this experiment was the original impetus for the study.

The basic idea and motivation for this experiment were discussed in Section 5.3 (Confusion Matrix Distribution). The only aspect of this experiment not discussed there is the filtering of the classifications according to some measure of certainty. Section 5.3 treated the output of the classification generator as if all classifications would be provided to the training and testing algorithms. Experiments discussed in Section 8.1(Full Noisy Classification Context) will take that approach, but in the experiment discussed here, not all classifications are supplied.

In this experiment, only classifications that we are fairly certain of (i.e., ones beyond some specified certainty threshold) are allowed to be passed into the classification context. All other classifications are declared to be missing. In this way, we hope to approach the same results as the preceding experiment where small amounts of absolutely correct context improved performance.

Filtering of classifications by certainty

There are several questions that need to be answered in order to be able to filter out classifications that are unlikely to be correct. First, we need to have some indicator of the likelihood that a classifier's guesses are correct. Second, we need to have some method of emulating that measure in building training examples.

We will estimate certainty using something that we will call the Hamming spread and we will make the confusion matrix the source for constructing training examples.

In order to see how we arrive at these two choices, it may be helpful to remember the pipeline of classifiers that we are trying to build. Figure 7.2.1 shows the final set of trained classifiers as a person would use it to try to classify some unknown protein. It represents the flow of information through the series of classifiers as each one refines its predecessor's guesses in the testing phase after all training is complete.

In this sequence, the first classifier, C_1, gets only the residue sequence as input and is identical to the no context classifier described in the baseline experiments of Section 6.1. Each subsequent classifier receives the sequence and its predecessor's thresholded classifications as input and produces a classification for each residue as its output. (Thresholding is discussed below.)

Each of these chained classifiers has been trained to expect the type of errors made by its predecessor and to expect any classifications not declared as missing to be correct with an approximate likelihood. That likelihood is determined by a threshold associated with the preceding classifier.

Each classifier in the chain may need a different certainty threshold in order to achieve a desired probability of being correct. This is because however we measure certainty, it only indirectly measures correctness, that is, you can be very certain and be very wrong.

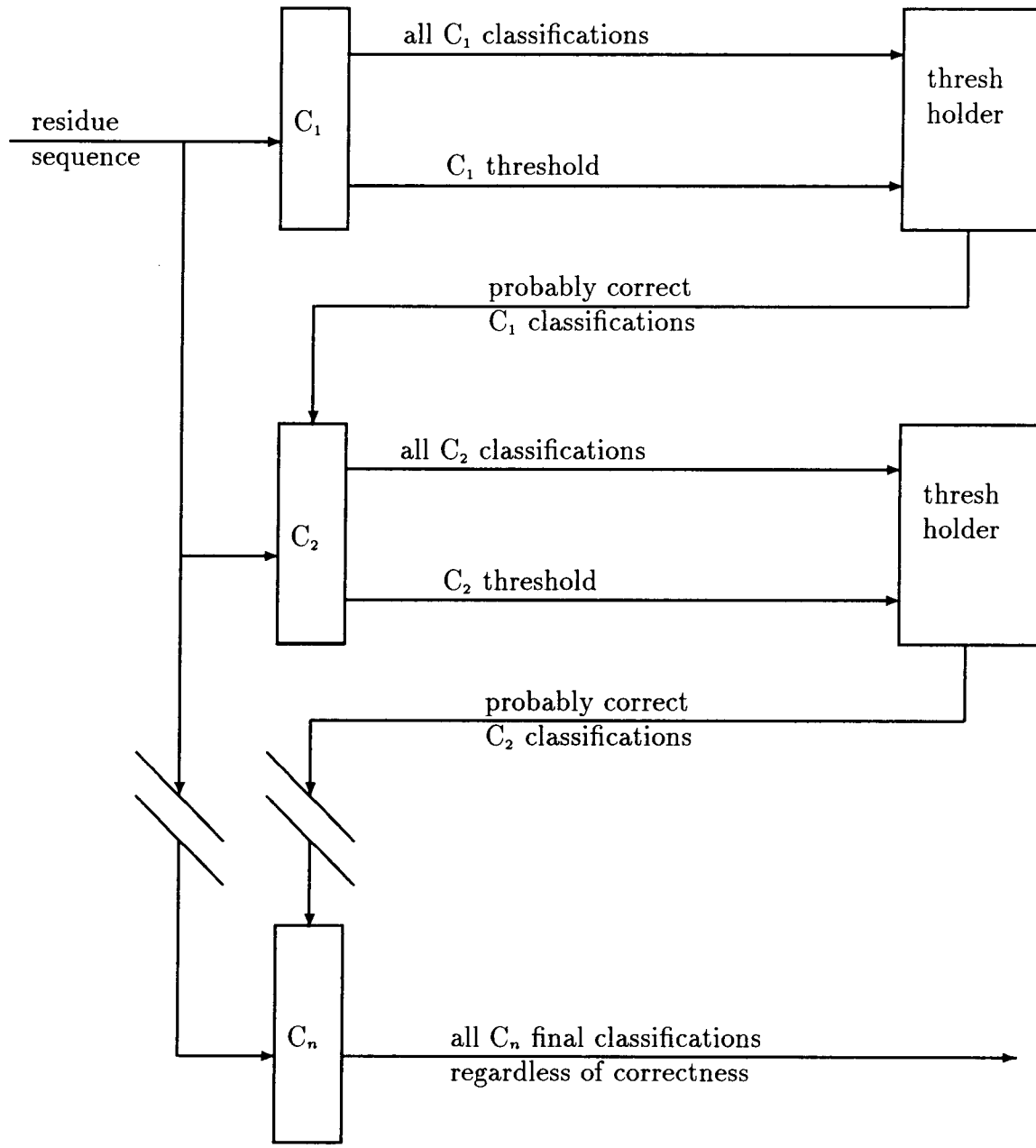


Figure 3. Classification Using Sequence of Classifiers

We chose the spread between the Hamming distances to the two nearest classes as our indication of certainty. We found that if you only classify residues where the Hamming spread is greater than some specific threshold, you are likely to get a certain percentage of the classifications correct. The larger the spread is, the more classifications that are likely to be correct. However, the relationship is only estimated. The cross-validation data provides a means of estimating what the spread threshold should be for each particular classifier.

Still, there is a question as to whether enough of the context will survive this screening to provide help to a classifier. You may be certain that 90% of the classifications that are made are correct, but only end up with 4 residues out of 1000 classified. In that case, the classification context that you are providing is equivalent to providing no classification context at all.

In summary, if the hypothesis of this experiment is true, then a series of classifiers C_1 through C_n could be trained to be used in a sequence like the one shown in Figure 7.2.1 and achieve performance improvements similar to those found in the experiments of the preceding section (Random Missing and 100

7.2.2 Experiment

Several sequences of classifiers were trained using the partial training set and tested on the cross-validation data set and the non-homologous test data set. Each series had a different certainty threshold in order to test screening varying amounts and correctness of classification context.

The primary differences among the experiments relate to the choice of data set for training, error distribution, and testing. However, there are two other experimental parameters peculiar to C4.5 with error-correcting codes which should be noted, i.e., threshold definition and missing value encoding. These are only discussed for the benefit of someone using the C4.5 system. They are not important in relation to the question of classification context.

Missing Value Encoding

In C4.5, a question mark can be entered in place of a data value to indicate a missing value. Initially in this study, a question mark was entered for each bit of the input encoding of a missing context classification. Later, this encoding was replaced by designating a particular input encoding to mean "missing", since the question mark encoding of missing values caused C4.5 to take approximately 5 times as long to train and the resulting classifiers were not as accurate.

Certainty Measures

Two different certainty measures were investigated: Hamming distance to the nearest classification and difference between the Hamming distance to the nearest and second-nearest classifications. The spread between the top two classifications was found to be more reliable than the Hamming distance to the nearest classification so that spread was chosen as the certainty measure.

Training and Testing Procedures

The detailed sequence of events and passing of data from classifier to classifier is convoluted and difficult to follow because there are so many very similar things mixing and diverging. We will attempt to explain it several different ways in hopes that the sum of all of them will make it clear what is going on.

Figure 4 gives an overview of the algorithm for training a sequence of classifiers. Figures 7.2.2 through 7.2.2 follow the overview and give a more detailed graphical depiction. A textual explanation of the whole process is given in Section 5.3.2 on Generating Classification Context for Training Using a Confusion Matrix.

Figures 7.2.2 through 7.2.2 below show a more detailed view of the process for building the first few classifiers in a sequence of classifiers. These figures contain labels that are intended to be self-explanatory but may need some definition. Some of the labels which might need more explanation are defined on page 88 under Terms.

The process in each figure consumes the output of the process in the previous figure as its input. The general process for any classifier C_i other than C_1 can be seen in the illustrations for classifier C_2; just substitute C_i for C_2 and C_{i-1} for C_1 in those figures. Classifier C_1 differs slightly from all others in that it is not given any classification context since it is the first classifier in the sequence.

Some things to note about the process shown in the figures:

Figure 4. Training Algorithm Overview

- Train a classifier.
- Test it on the non-homologous test set to measure its performance.
- Test it on the distribution source test set so that you can derive a confusion matrix. Distribution source test set is either the cross validation set or the non-homologous test set depending on what your objective. (Which to choose is discussed under Source of Context Distribution in Section 7.3.)
- Build the confusion matrix with an extra column for "missing". Any classification below the given certainty level goes in the missing column.
- Create a new training file combining the sequence of residues with classification context derived using the distribution of errors on the test file at the given certainty.
- Repeat until no improvement in test results on non-homologous data.

- There is a confusion matrix for each test file and threshold as as a pair, for example, the non-homologous test file and a Hamming spread of 7 bits. There is not just one confusion matrix for each test file. If the classifications were not being thresholded, then there would be only one confusion matrix for each test file. That matrix would be equivalent to the matrix for an infinitessimally small threshold.

- The trick in all of this is that "missing" is treated as just another output class in the confusion matrix.

- Note that Figure 7.2.2 (Testing Process for Classifier 2) is almost identical to the figure for Classifier 1 (Figure 7.2.2). The difference is that C_2 gets thresholded C_1 classifications of the non-homologous test data as its classification context while C_1 gets no classification context.

  In other words, during testing, each classifier in the pipeline classifies the test set and has its classifications thresholded before they are handed to the next classifier in the pipe. During training, each classifier builds a confusion matrix and generates simulations of thresholded classifications to be handed to the next classifier.

  As explained in Section 5.2 (Classification Context Generation Methods), this is done because the classifiers learn the training data too well and even if thresholded would pass an unrealistic set of classifications through.

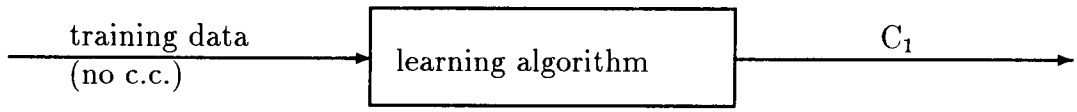


Figure 5. Training Process for Classifier 1

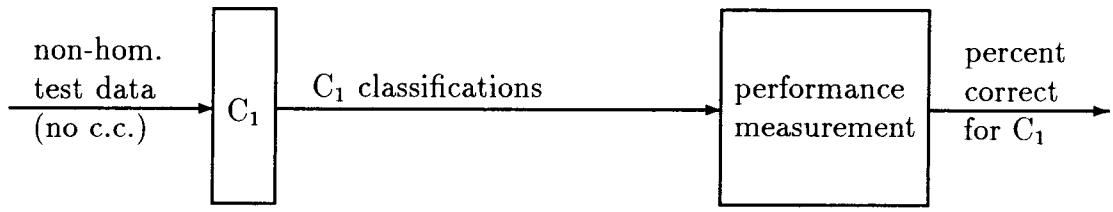


Figure 6. Testing Process for Classifier 1

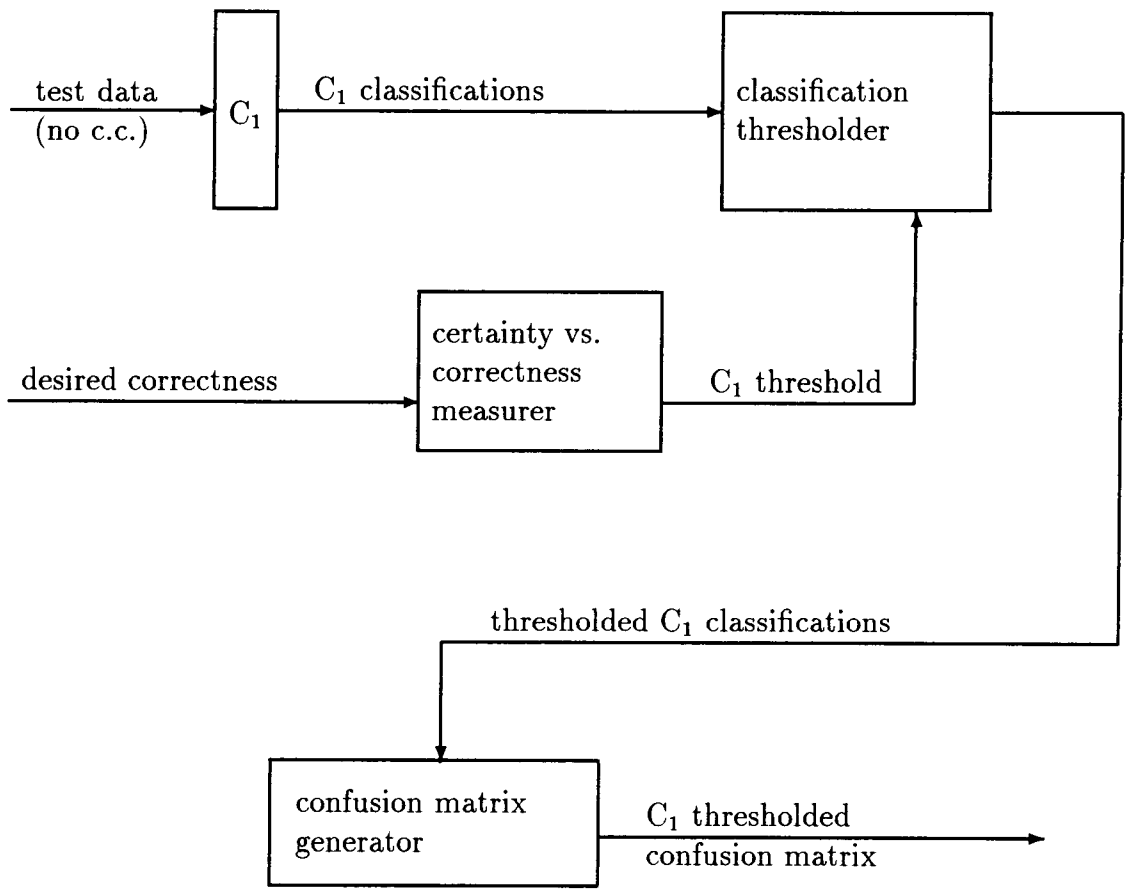


Figure 7. Confusion Matrix Creation Process for Classifier 2

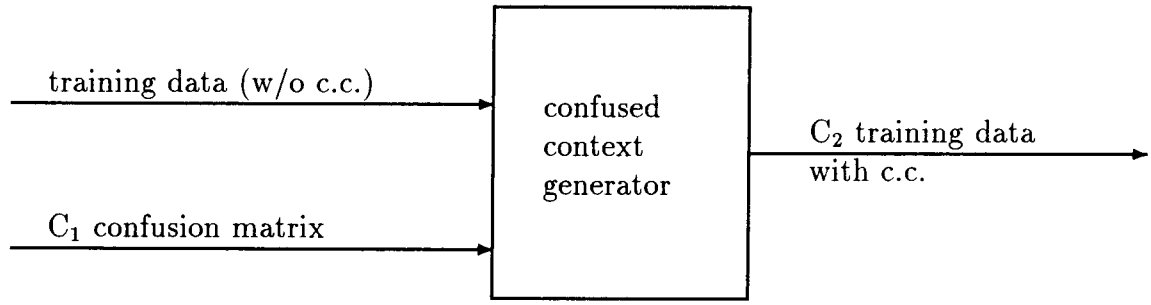


Figure 8. Training File Creation Process for Classifier 2

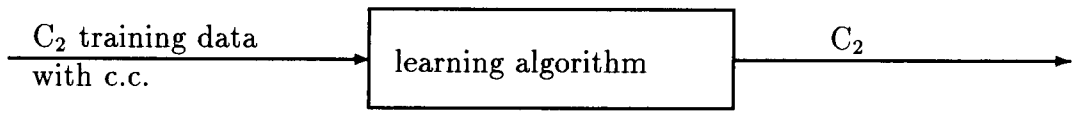


Figure 9. Training Process for Classifier 2

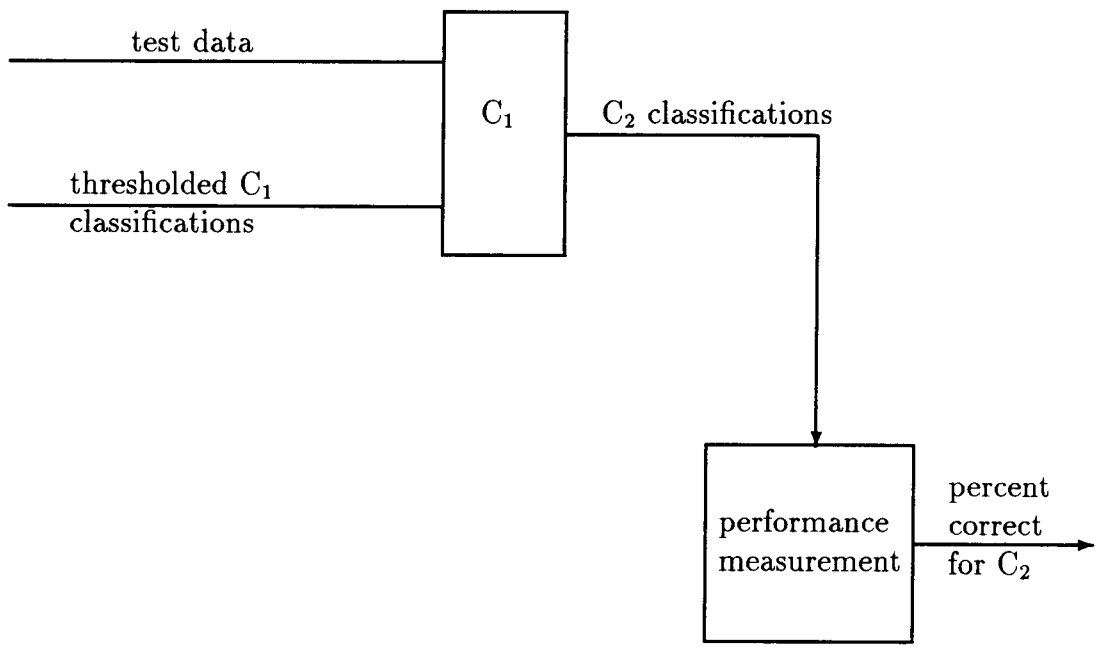


Figure 10. Testing Process for Classifier 2

Terms:

- c.c.

  Abbreviation for "classification context".

- learning algorithm

  The inductive learning program which builds the classifiers, i.e., C4.5 (not to be confused with C_1 .. C_n which are the classifiers that C4.5 builds; the name C4.5 is just an unfortunate coincidence here).

- test data

  Means either the non-homologous test file or the cross-validation test file. The Results section (7.3) discusses which one is used at any given time and why under the heading Source of Context Distribution.

- desired correctness

  The minimum percent of guesses not declared "missing" which should be correct. This is a target, but can not be guaranteed. Note that while this value need not be the same for all classifiers in the sequence, for consistency it is held at the same value for each of the classifiers in a given sequence in these experiments. For example, all classifiers in one sequence will attempt to classify 70% correctly while in another sequence they will all try for 90% correct.

- certainty vs. correctness measurer

  The program which looks at the classifications that were made and computes what Hamming distance or Hamming distance spread can be selected as a threshold in order to attain the desired level of correctness on the test data.

Given this procedure for training classifiers using classification context with missing values, several experiments were done.

- The first experiment was a baseline test again. It tested direct, thresholded feedback of the classifications in training without the use of a confusion matrix distribution.

- The second experiment was the primary experiment of interest for this study. It tested the use of both a confusion matrix distribution and a certainty threshold.

- A final systematic block of experiments was done to look at the relationship between a classifier's expectations about the certainty of its input and the certainty of the input that it actually receives.

7.3 Results

None of the tests showed good results. If they had, then a series of classifiers would have been trained for those instances, but since the results were not good, only one or two classifiers were trained in each proposed sequence.

Table 10. Direct, Thresholded Feedback (Threshold = 90% Certainty)

Test	Q3	C_α	C_β	C_{coil}
partial training set[8]	57.70	0.17	0.17	0.24
full training set	57.90	0.15	0.20	0.24

Each group of experiments is described separately now.

Baseline: Thresholded direct feedback

Table 10 shows the results of the baseline test. In this experiment, the results of applying the C_1 classifier to the training set are thresholded at a certainty level so that approximately 90% of the classifications not declared as missing are correct. These classifications are then provided as the classification context in training the C_2 classifier. The confusion matrix distribution is not used at all.

Similarly, the C_1 classifier's output on the non-homologous test set is thresholded and supplied as the classification context for testing the C_2 classifier on that test set.

As expected, the results here are not good. The correlation coefficients are even worse than those for the classifier trained with no context at all (Section 6.1 No Classification Context).

However, the results show little impact due to reduction in the size of the training set. This agrees with Qian and Sejnowski's results which showed that once

they reached about 5000 residues in their training set, performance did not improve significantly with the addition of more training examples. The implication for this study is that if there are significant performance problems in experiments where we have used the partial training set instead of the full set, the problems should probably not be attributed to the reduction in the number of training examples.

Thresholded confusion matrix distribution feedback

In this experiment, the results of applying the C_1 classifier to the cross-validation test set are thresholded at a certainty level so that approximately 90% of the classifications not declared as missing are correct. A confusion matrix is created from these classifications and the C_2 classifier is trained with classification context derived from that confusion matrix.

Similarly, the C_1 classifier's output on the non-homologous test set is thresholded and passed as the classification context for testing the C_2 classifier on that test set. The results of this test are called Pass 1 in Table 11.

The process is repeated for Pass 2 in the table by using C_2's classifications to provide classification context for C_3 in the same way that C_1 classified for C_2.

The seed question for this whole study was whether you could use a cross-validation confusion matrix thresholded to approximately a 90% certainty level in a pipeline and get good results. Table 11 shows the results of doing that, and the answer is no. The results are even worse than for the no context classifier (Section 6.1).

Table 11. Thresholded Cross-Validation Test Set Confusion Matrix Distribution (Threshold = 90% Certainty (spread: 15 bits))

Test	Q3	C_α	C_β	C_{coil}
Pass 1	58.49	0.23	0.17	0.28
Pass 2	58.52	0.22	0.19	0.30

Table 12. Hamming Spread vs. % Context and % Correct in Classification Context

	Hamming Spread (in bits)							
	1	3	5	7	9	11	13	50
% Provided	89.	69.	52.	38.	27.	16.	9.	0.
% Correct	61.2	65.2	70.3	75.5	81.5	83.9	86.7	100

Two possible reasons for the bad results are that high certainty gives a high rejection rate and that the cross-validation error distribution may not match the non-homologous test set's error distribution. These two issues are discussed below.

Rejection rates

It may be that very little classification context is provided if certainty is required to be as high as 90%. Table 12 shows the rejection rates for various Hamming spreads when the C1 classifier is applied to the non-homologous test set. (Figure 7.3 shows the same data in graphical form.) In order to be correct on 90% of the classifications, less than 9% of the residues can be classified.

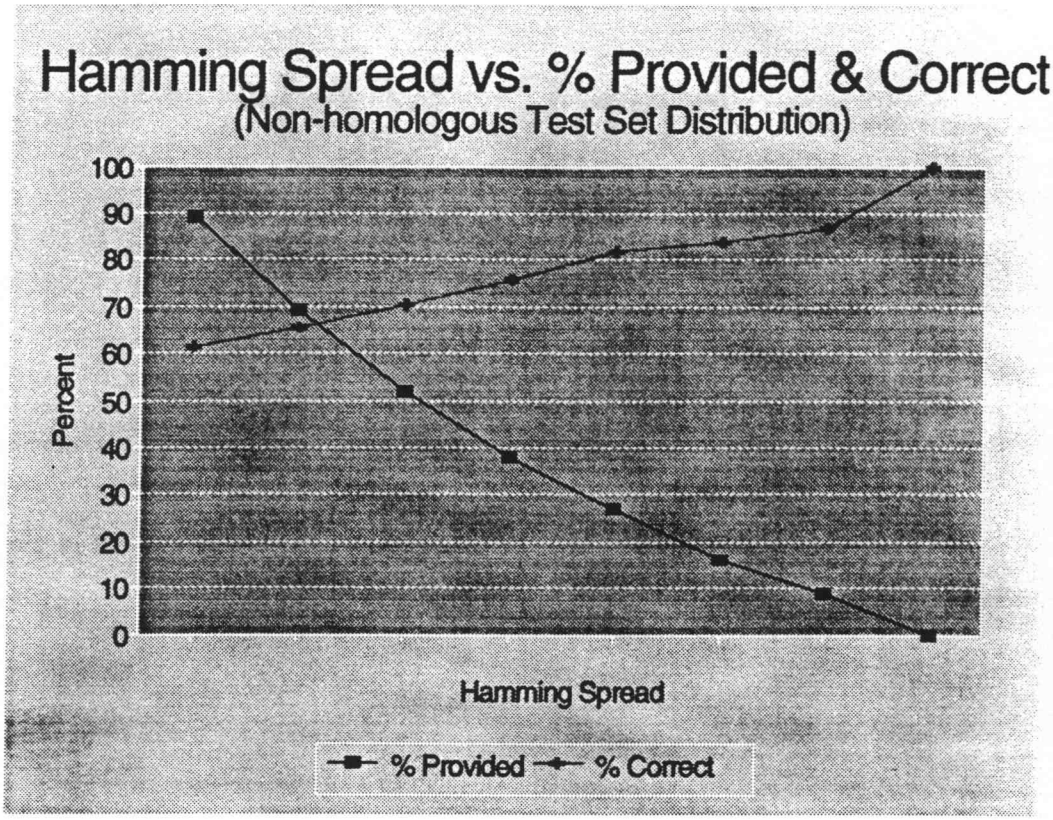


Figure 11. Rejection Rate Curves

Table 13. Thresholded Confusion Matrix Distribution - 90% vs. 70% Certainty

Test	Q3	C_α	C_β	C_{coil}
90% certainty	58.49	0.23	0.17	0.28
70% certainty (spread: 5 bits)	59.32	0.22	0.19	0.29

In order to allow more classification context to be provided, the test was repeated using a 70% certainty instead of 90%. The results shown in Table 13 are only a slight improvement even though the amount of classification context provided has increased from less than 9% up to around 52%.

Source of Context Distribution

The other possible source of error discussed above was the match between the expected classification context error distribution and the distribution of errors actually received. Another experiment was done using the non-homologous test set as the source of the confusion matrix instead of using the cross-validation set. As is discussed in Section 5.3.3, this is not a fair test, but it should be the best possible confusion matrix to derive the expected error distribution. However, the results again show no improvement.

Expected certainty vs. provided certainty

Given all of these permutations of bad results, an additional systematic grid of tests was done using the non-homologous test data set's distribution to look at

Table 14. Thresholded Test Set Confusion Matrix Distribution - Non-Homologous vs. Cross-Validation (90% Certainty, Spread: 15 bits)

Test	Q3	C_α	C_β	C_{coil}
Cross-Validation Set	58.49	0.23	0.17	0.28
Non-Homologous Set	58.84	0.23	0.18	0.31

the system's behavior over a wide range of certainties and expectations about the makeup of the input.

Since one of the things that this experiment is trying to determine is the relationship between the expectation of the classifier and the makeup of the input, classifiers were trained using various levels of certainty in their context. They were then given input with various levels of certainty which did not necessarily match what they were trained on.

The Hamming spreads for measuring certainty were varied from 1 to 13 bits, stepping by 2. The approximate percentages of correctness and context provided for each certainty threshold can be seen in Table 12 mentioned above (Hamming Spread vs. % Context).

One additional Hamming spread was provided outside the 1 to 13 bit range. That spread was 50 bits. This spread was provided as a control to see whether the performance using the representation of no classification context as "all missing values" was different from that using no classification context at all. Since the

codes were 50 bits long, a 50 bit spread was an easy way to create a window where all classification context values were missing. The result using no context was 58.52% correct and the result using all missing was 58% correct so it looks like if there is any effect, it is very small.

Note that these experiments are done using the non-homologous test data set's confusion matrix distribution rather than that of the cross-validation test set. They should set an upper bound on how well the training scheme works if performance changes are due to how well the error distribution of the cross-validation data set matches that of the non-homologous test data set.

The results of the tests are shown in Tables 15- 17 and their corresponding Figures 7.3 - 7.3.

These experiments show that regardless of the distribution used to build the training context and regardless of the certainty of the input, the classifiers are unable to improve on the original classifier trained with no classification context.

Table 15. Total Percent Correct (Q3) for Distribution Context

Given	Trained On Spread (in bits)							
Spread	1	3	5	7	9	11	13	50
1	**58.5**	58.2	57.9	57.1	56.4	56.9	56.6	38.1
3	58.8	**59.0**	58.2	58.4	57.1	58.	57.4	41.5
5	57.8	57.7	58.9	59.2	59.3	**59.8**	58.4	43.2
7	54.7	55.7	57.3	58.3	58.1	**59.3**	58.7	46.
9	51.3	52.6	54.9	56.1	57.3	58.4	**59.6**	50.1
11	46.4	46.8	49.5	50.9	52.3	55.9	**58.0**	53.9
13	43.6	43.4	46.1	47.1	48.6	52.9	**56.6**	56.3
50	42	40.5	42.7	43.6	43.6	49.8	54.8	**58.0**

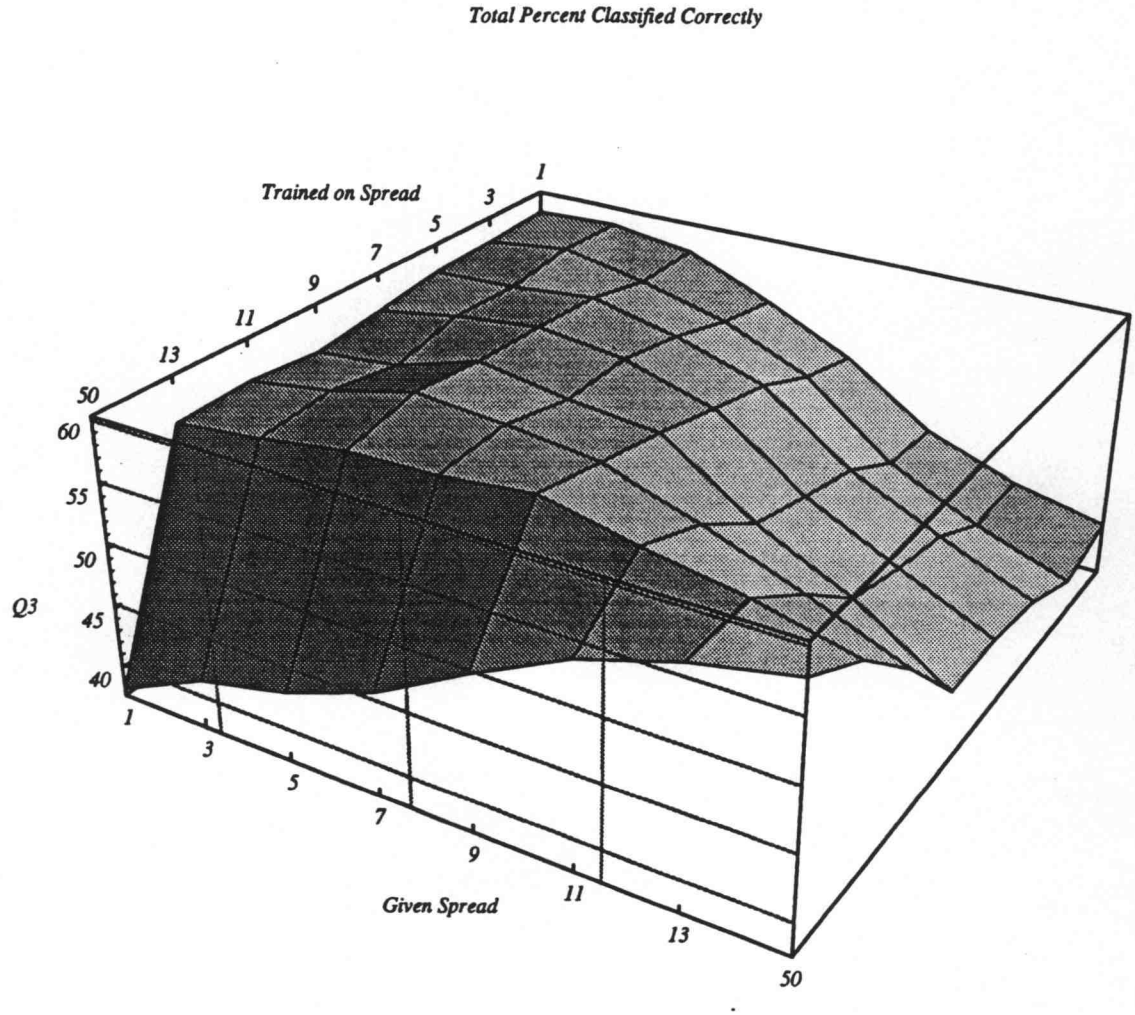


Figure 12. Total Percent Correct (Q3)

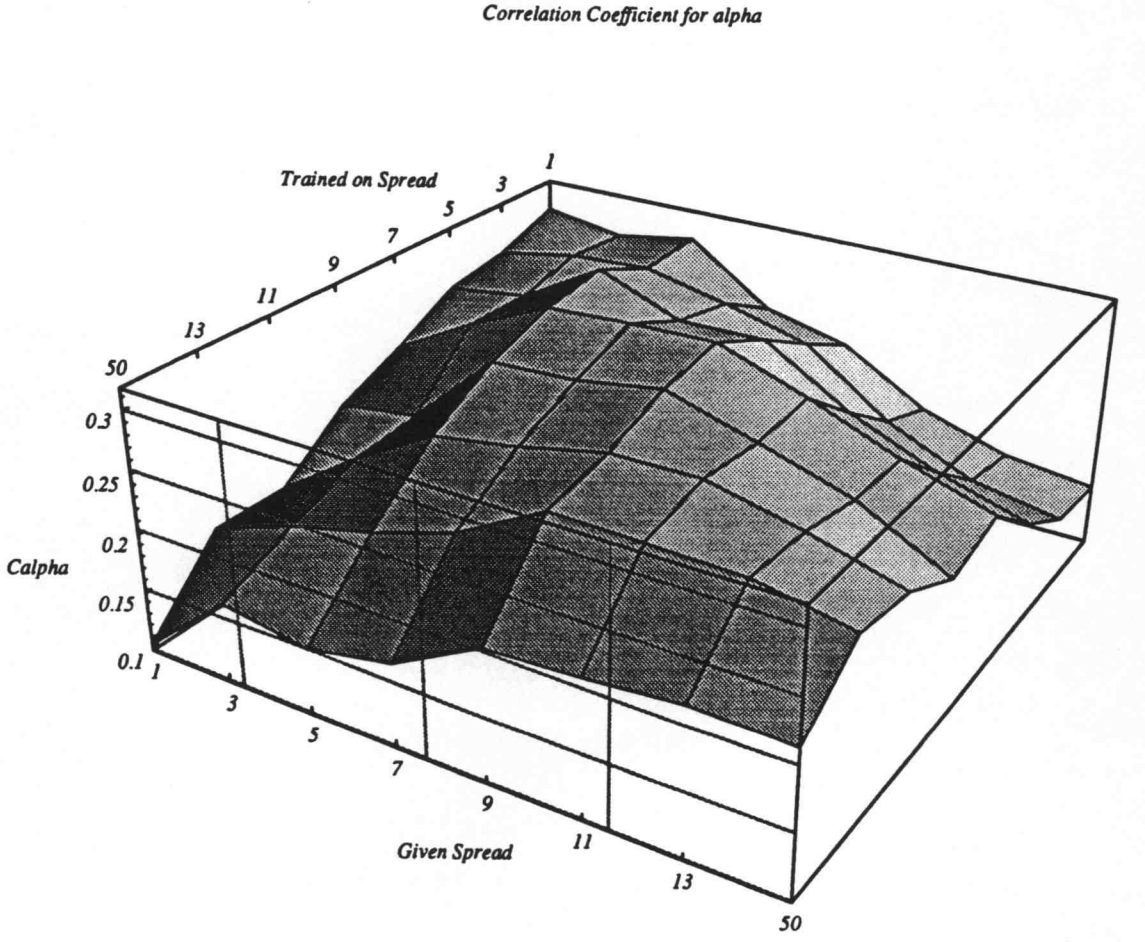


Figure 13. Correlation Coefficient for α

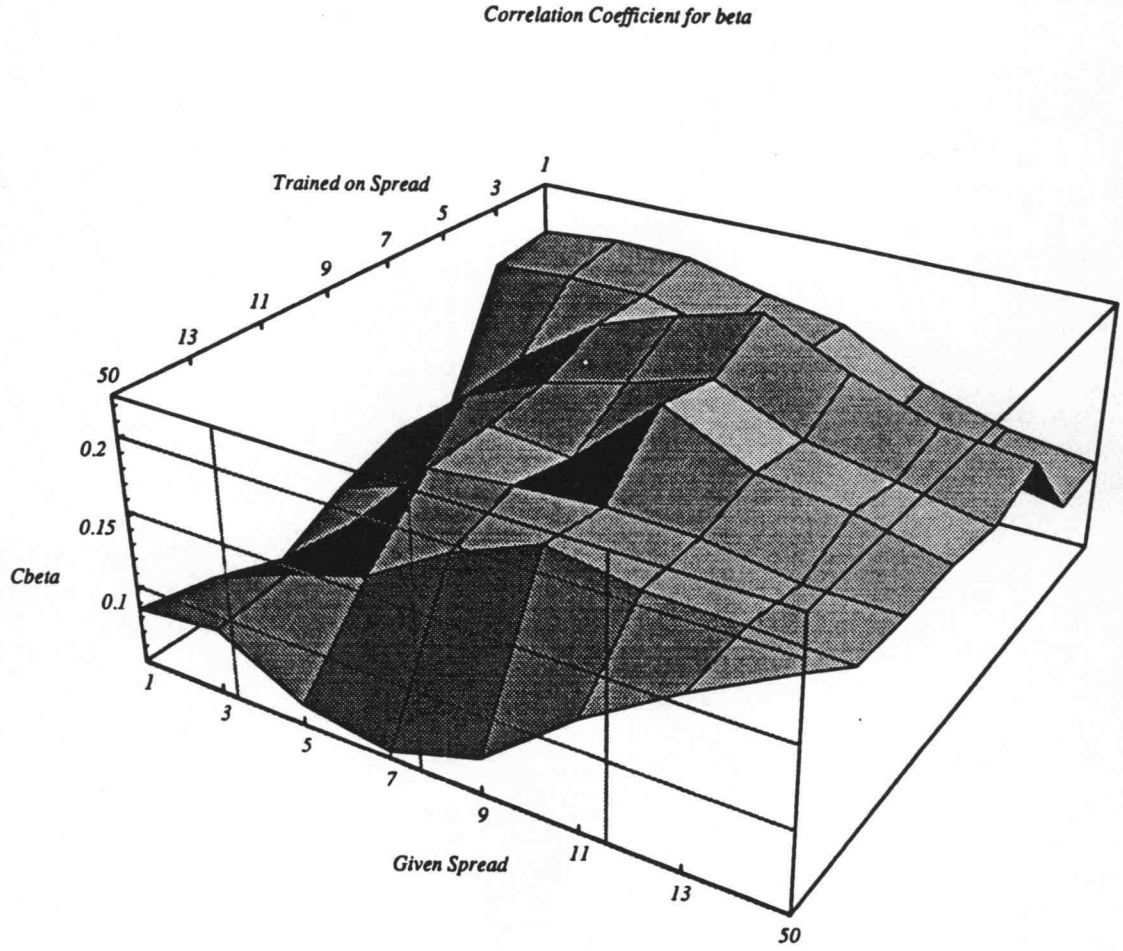


Figure 14. Correlation Coefficient for β

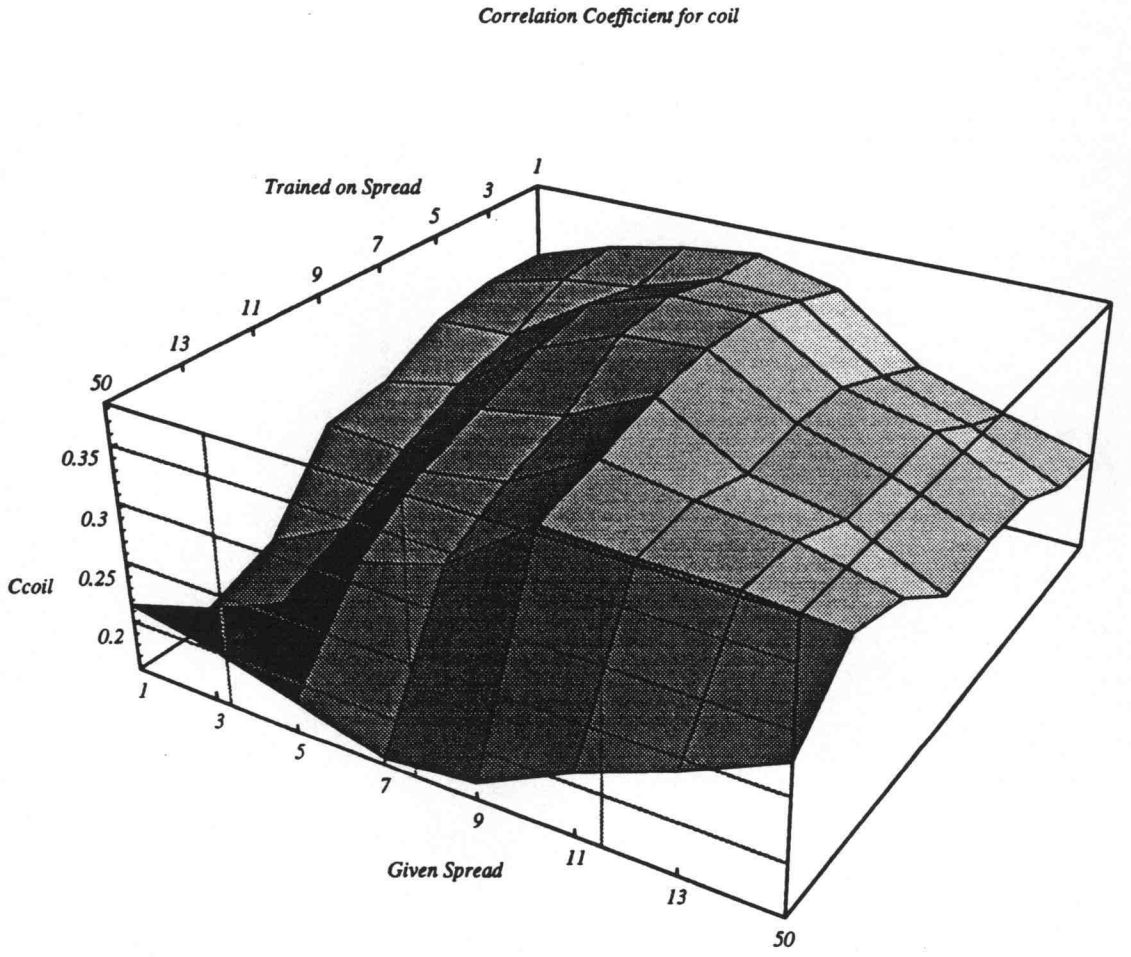


Figure 15. Correlation Coefficient for Coil

Table 16. α Correlation Coefficient for Distribution Context

Given	Trained On Spread (in bits)							
Spread	1	3	5	7	9	11	13	50
1	**0.29**	0.27	0.26	0.23	0.21	0.17	0.17	0.1
3	**0.28**	**0.28**	0.27	0.27	0.22	0.22	0.19	0.17
5	0.29	0.29	**0.31**	**0.31**	0.29	0.26	0.21	0.15
7	0.24	0.27	0.28	**0.31**	0.29	0.27	0.23	0.17
9	0.22	0.25	0.29	**0.31**	0.30	0.28	0.27	0.21
11	0.18	0.20	0.24	**0.29**	0.28	**0.29**	0.27	0.21
13	0.16	0.17	0.21	0.26	0.26	**0.27**	**0.27**	0.22
50	0.15	0.16	0.19	0.23	0.22	0.25	**0.26**	0.22

Table 17. Coil Correlation Coefficient for Distribution Context

Given	Trained On Spread (in bits)							
Spread	1	3	5	7	9	11	13	50
1	**0.32**	**0.32**	0.31	0.28	0.27	0.19	0.17	0.22
3	**0.34**	**0.34**	0.31	0.29	0.27	0.23	0.2	0.2
5	**0.36**	0.34	**0.36**	0.35	0.32	0.3	0.26	0.19
7	0.36	0.37	**0.38**	0.37	0.34	0.32	0.28	0.16
9	0.35	0.36	**0.38**	**0.38**	0.37	0.36	0.34	0.17
11	0.29	0.31	**0.34**	**0.34**	0.33	**0.34**	**0.34**	0.21
13	0.27	0.29	0.32	0.31	0.31	0.33	**0.34**	0.24
50	0.25	0.25	0.28	0.28	0.28	0.31	**0.33**	0.28

Chapter 8

Full Noisy Classification Context

8.1 Sensitivity to Noise in the Classification Context

8.1.1 Hypothesis

The fact that just a little absolutely correct context provides large improvement in classification while no context provides mediocre results makes it natural to ask what level of noise can be tolerated in the classification context and still give good results. Moreover, noisy context of unknown quality is more like the data that a cooperative classifier system would face if allowed to take context from other sources.

8.1.2 Experiment

In order to investigate the role of noisy classification context, several experiments were done.

- The first experiment was another baseline test. It tested direct feedback of the classifications in training without the use of a confusion matrix distribution.

- The second experiment tested the use of a confusion matrix distribution with no certainty threshold.

- Finally, a systematic block of experiments similar to those discussed above was done to look at the relationship between a classifier's expectations about the correctness of its input and the actual correctness of the input.

8.1.3 Results

Each of group of experiments is described separately now.

Baseline: Direct feedback

Table 18 shows the results of the baseline test. In this experiment, the results of applying the C_1 classifier to the training set are passed directly through as the classification context for training the C_2 classifier. The confusion matrix distribution is not used at all. (This should produce a classification context that is much more accurate than the clasifier will ever see on the test set, because the classifier has already learned the training data very well.)

After C_2 is trained, the C_1 classifier's output on the non-homologous test set is passed straight through as the classification context for testing the C_2 classifier

Table 18. Direct Feedback, No Thresholding

Test	Q3	C_α	C_β	C_{coil}
Pass 1	57.81	0.26	0.16	0.32
Pass 2	57.78	0.28	0.17	0.32
Pass 3	57.13	0.28	0.16	0.30

on that test set. The results of testing C_2 on the non-homologous test set are shown as Pass 1 in the table. (C_1 acts as Pass 0.)

The process is repeated for Pass 2 in the table by using C_2's classifications to provide classification context for C_3 in the same way that C_1 classified for C_2. Each subsequent pass is done in an analogous way.

Again, the results are almost identical to every other test. In particular, sequential passes degrade the results rather than improve them. The poor performance is not surprising since the training data's classification context is much more accurate than the test data's classification context.

Confusion matrix distribution feedback

In this experiment, a confusion matrix is created from C_1's classifications of the cross-validation test set and the C_2 classifier is trained with classification context derived from that confusion matrix.

Like the direct feedback experiment, the C_1 classifier's output on the non-

Table 19. Cross-validation Test Set Confusion Matrix Distribution (No Thresholding)

Test	Q3	C_α	C_β	C_{coil}
Pass 1	57.81	0.26	0.16	0.32
Pass 2	56.16	0.27	0.15	0.29
Pass 3	56.34	0.28	0.17	0.30
Pass 4	56.45	0.29	0.16	0.30

homologous test set is passed straight through as the classification context for testing the C_2 classifier on that test set. The results of this test are called Pass 1 in Table 19.

The process is repeated for Pass 2 in the table by using C_2's classifications to provide the confusion matrix classification context for C_3 in the same way that C1 classified for C_2. Each subsequent pass is done in an analogous way.

Random Noisy Context

This experiment tried injecting various levels of noise into the classification context of the full training data set by assigning a context classification to every residue but randomly corrupting it to some percentage level, e.g., 40% of the context was correct and 60% was assigned some random choice of anything but the correct class. (Note that the incorrect class was a triple class, not a singles class. It could still imply the correct singles class. Another full set of tests could be done

where the choice of incorrect class was restricted to classes which did not imply the correct secondary structure class.)

Since a real system would not necessarily know much about the characteristics of the context that is being supplied to it, the sensitivity of each of the classifiers was measured by testing it on the non-homologous test data set with classification context corrupted with various levels of noise. For example, one classifier was trained using context that was 80% correct and then tested on data sets that were 5% correct, 20% correct, and so on.

Since each context supplier in a cooperative system would have its own bias, it would be unlikely to produce truly random corruption of classifications like the tests just described. Therefore, each classifier was also tested using context taken from the classifications produced by the classifier built for Section 6.1, i.e., the classifier trained on the full data set using no context.

Table 20 shows results for the classifier trained on 60% correct context as one example. Since the number of tests is large and the resulting tables are daunting, the detailed results of training and testing the set of classifiers at various levels of correctness are shown in Appendix C. Figures 8.1.3 through 8.1.3, show graphs of the total percent correct and correlation coefficients for each of the tests in the tables.[9]

[9]The plotting package which produced the graphs required interpolation of extra points between some of the tests in order to produce the grid so there are more points on the plotted surface than there are tests in the tables. Interpolated points were calculated by just averaging their

Table 20. Full Noisy Context (Trained on 60% correct context)

Percent Correct	Percent Classified	α			β			Coil		
in Context	Correctly (Q3)	C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	59.3182	0.28	34.39	52.05	0.1	7.086	43.44	0.28	90.64	61.44
5	39.9148	0.	22.38	23.66	0.01	27.41	22.26	0.03	52.52	56.24
10	47.4432	0.11	32.51	32.02	0.13	34.09	30.43	0.17	59.23	62.58
20	62.0739	0.41	55.71	54.87	0.31	46.39	44.89	0.38	70.98	72.41
30	70.4261	0.57	69.38	66.55	0.44	54.95	56.69	0.5	76.91	77.43
40	79.9716	0.69	78.21	74.86	0.63	68.98	73.09	0.67	85.02	84.85
50	85.0568	0.79	85.63	83.18	0.71	71.39	82.79	0.74	90.12	86.61
60	91.3068	0.89	92.23	90.42	0.82	81.28	90.61	0.85	94.8	91.93
70	94.0625	0.93	95.29	93.74	0.88	87.57	93.97	0.89	96.05	94.23
80	97.1023	0.96	97.17	96.94	0.95	94.52	97.52	0.95	98.08	97.02

All of the plots show a smooth degradation from the point of high context correctness in both testing and training to each of the corners where either testing or training context correctness was low. The point marked "Full" represents the results of using the output of the classifier trained on the full training set with no context. The test with 5% correct context represents approximately random context since there are 19 triples classes, and it provides the worst performance.

As expected, these results show that predictive ability in all categories (Q3 and correlation coefficients) increases with context correctness. They also show that the classifiers are fairly robust in their ability to handle input context whose accuracy does not match that of their training data.

However, it looks like performance is mostly a function of how correct the

---

immediate neighbors.

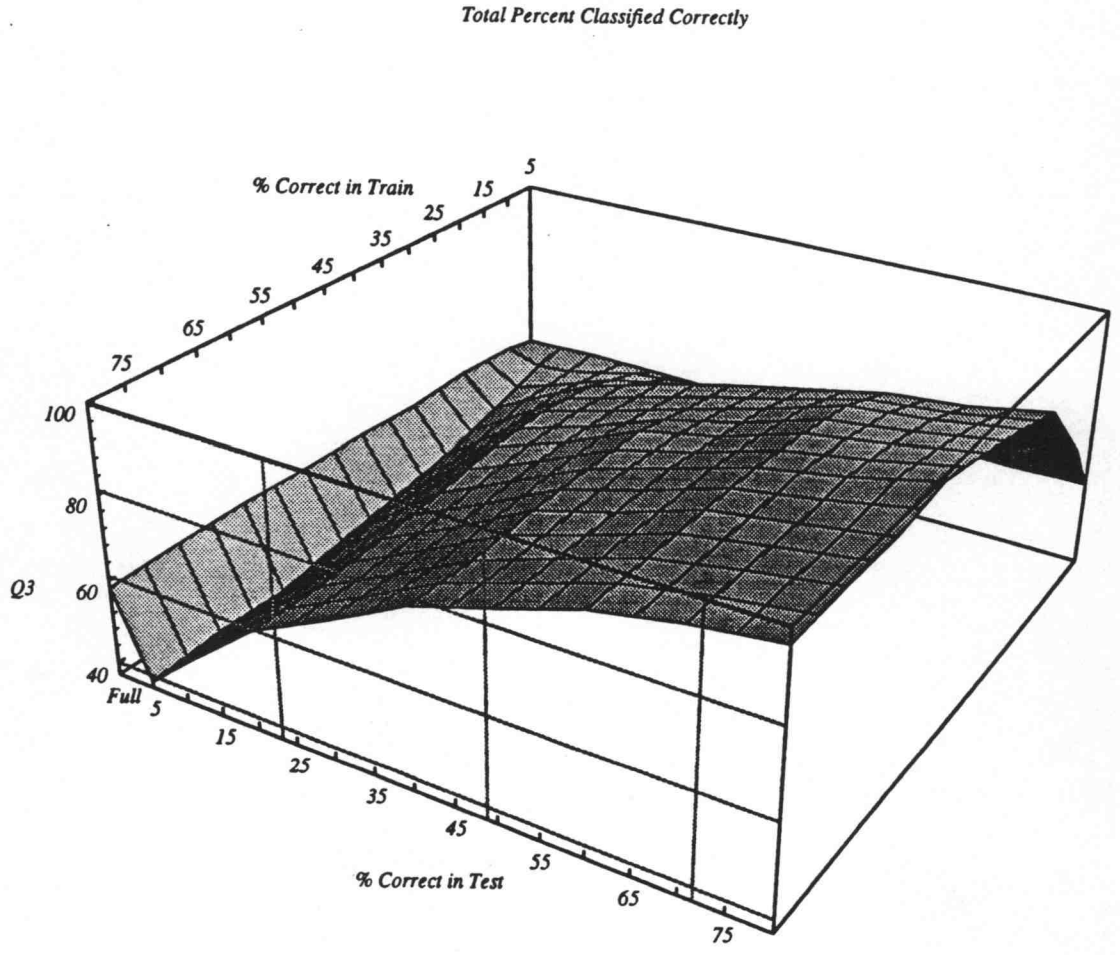


Figure 16. Total Percent Correct (Q3)

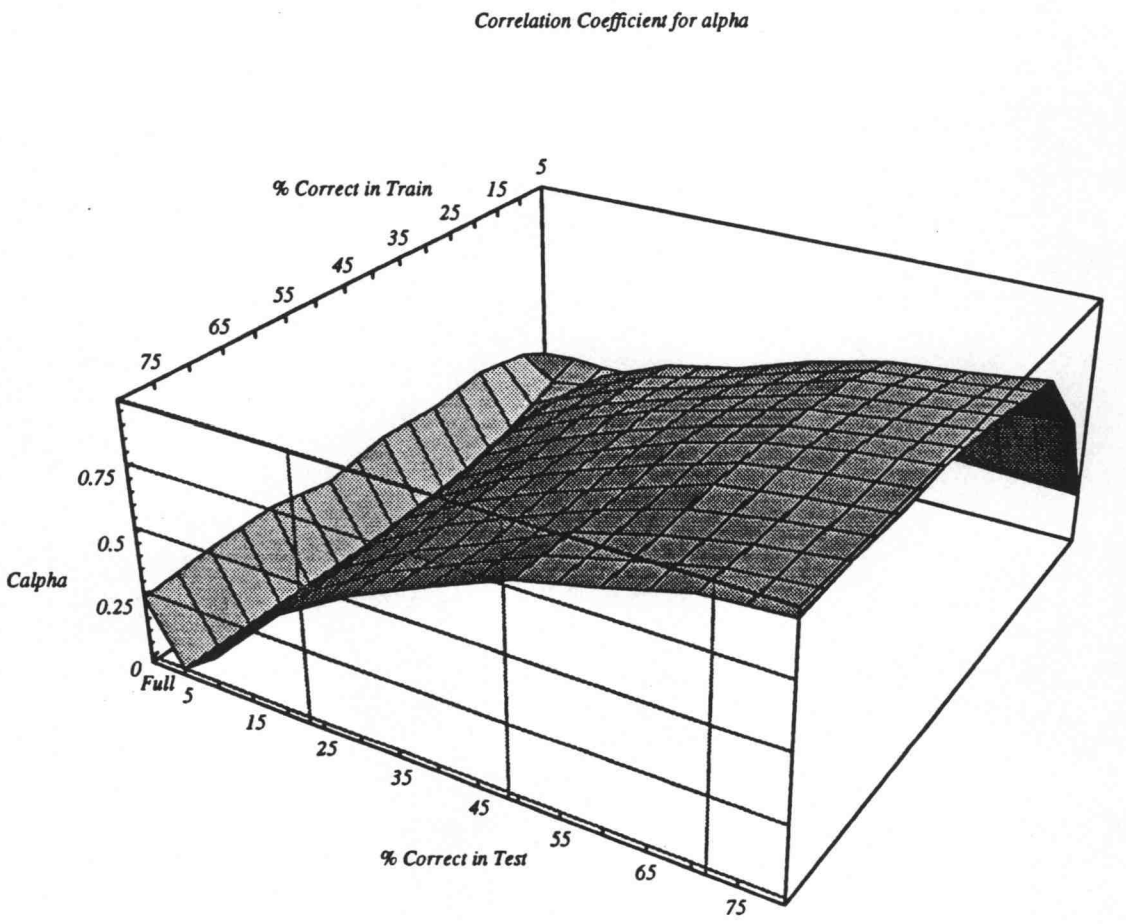


Figure 17. Correlation Coefficient for α

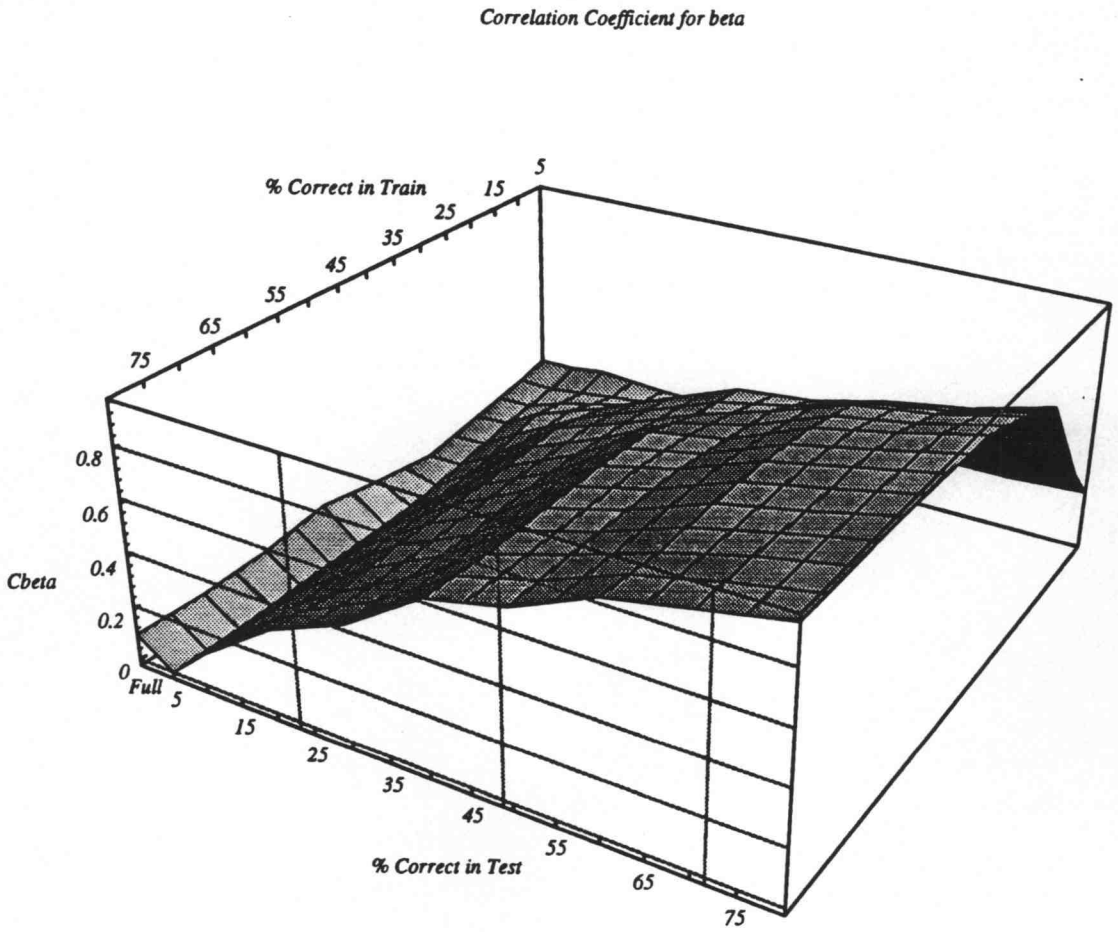


Figure 18. Correlation Coefficient for β

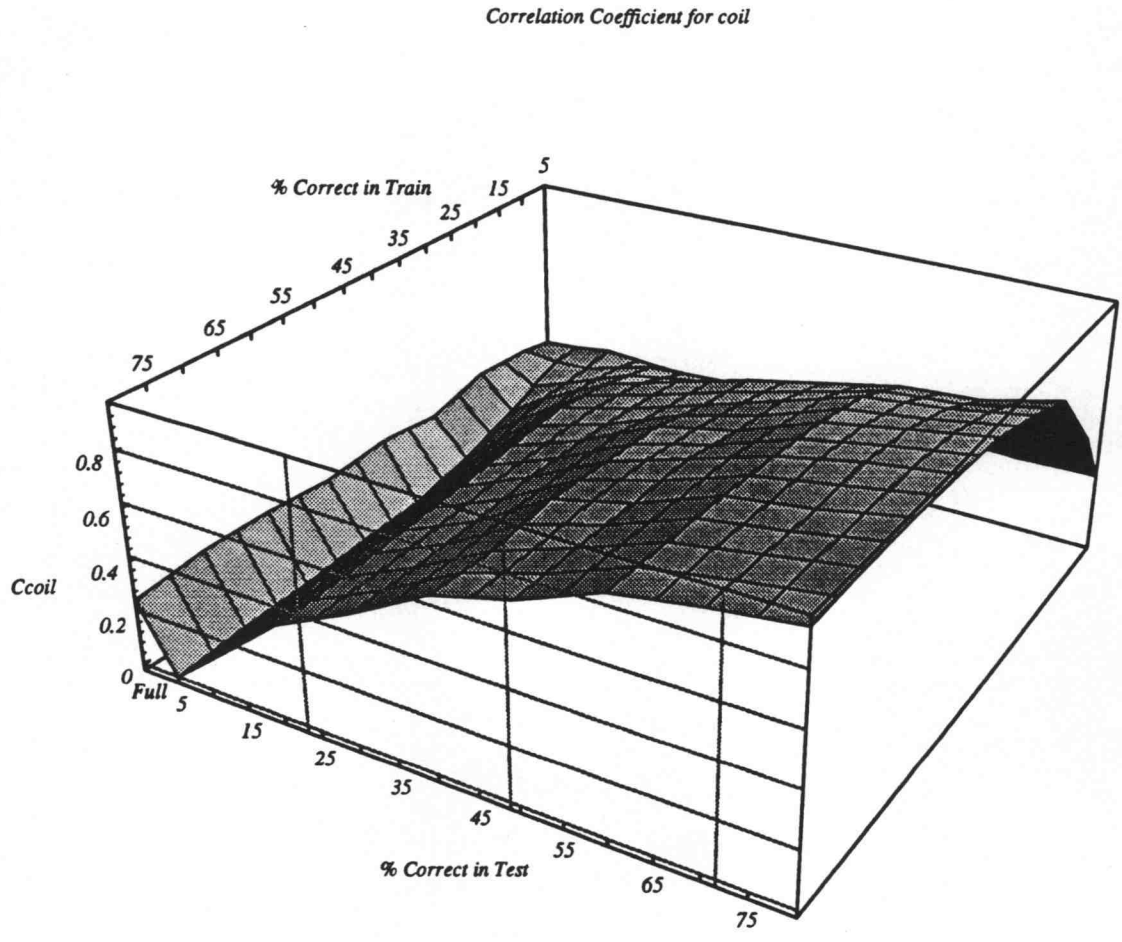


Figure 19. Correlation Coefficient for Coil

input context of the test data is rather than what the classifier expects. For any given percentage correct in the input context, the performance is nearly flat across all of the classifiers until you reach the classifiers trained on 20% or less correct context. Less than 20% (random) correct context appears to be more confusing to the classifier than no context at all.

Still, as the tests using the Full input context show, randomly correct context is much more powerful than the biased correct context at the same percentage. An example will help explain this.

Call the classifier trained to expect 60% correct context "C_{60}". When C_{60} is given the output of the "Full" classifier as its classification context, it gets about 60% of the output classifications correct. The Full context has about 50% of its triples classifications correct so C_{60} has improved its input by about 10%. Now consider the 50% random correct context.

Like the Full context, roughly 50% of its classifications are correct. The only difference between the "Full" and "50% random" contexts is in *which* residues are right. As far as C_{60} is concerned though, they are much different. When we give C_{60} the 50% random correct context as input it gets 85% correct instead of just 60%. This shows that having the same percentage correct is less important than how those correct classifications are distributed.

Chapter 9

Discussion

Nearly all of these experiments have had discouraging results and generally confirm the limitations found by other researchers in this field, but they have shown that there may be some power available in classification context. This section will discuss each of the questions posed at the start of this study (Chapter 1).

9.1 How Does C4.5 with Error-correcting Codes Perform in Comparison with Other Methods Like Neural Networks?

C4.5 with no classification context performed slightly worse than the neural network classifiers of other authors in both the Q3 and correlation measures. The correlation coefficients show that it did poorly in the same situations as other types of classifiers, i.e., it did badly on β strands and only slightly better on α helices.

Table 21. No Classification Context (C4.5 with ecc's vs. Qian and Sejnowski)

Method	Q3	C_α	C_β	C_{coil}
C4.5 w/ ecc	59.09	0.24	0.18	0.31
Q&S w/ 1 net	62.7	0.35	0.29	0.38
Q&S w/ 2 nets	64.3	0.41	0.31	0.41

Comparing most of the results in this paper with those in other studies is not meaningful since most of the experiments here are attempts to find the bounds of performance using artificially constructed context. Moreover, only Qian and Sejnowski's paper [7] employs the same data sets and, as discussed previously, that is important in attempting to make valid comparisons. (Zhang gives a table comparing a number of different studies in [30] which can be examined to compare several different methods without regard to data sets.)

Table 21 shows the difference in performance between Qian and Sejnowski's results and those obtained here with no classification context. Qian and Sejnowski's single network results are the ones most like those derived here.

While all of their numbers are better than those in this paper, they chose parameterizations for their network based on which ones gave them the best results rather than doing it blind, so the improvement is not measured fairly. However, their improvement due to cleaning up with a second net was something that the classification context classifiers attempted unsuccessfully to do here.

9.2 Reasons for Poor Performance of All Secondary Structure Predictors

The main reason for the poor performance is probably the one discussed in every study, i.e., the fact that much of the structure of a protein is probably determined by tertiary effects and the local sequence does not provide tertiary information. Studies like those of Wilson et al. [45] have shown that identical residue sequences in different proteins can have different secondary structures.

Another biological reason for poor performance may not be encountered very often but has not been considered here at all. Some proteins like insulin have a precursor which folds up and then pieces at the ends are cleaved off leaving the protein we are trying to classify. It may be that the missing pieces would help in determining the classification of the remaining core.

A more artificial reason for poor performance is not often discussed, i.e., the definition of correct classifications. Most if not all of the studies use Kabsch and Sanders' DSSP program [46] to assign "correct" secondary structure classifications to residues. However, several authors have shown that there can be differences between assignments made by the program and by authors [31, 28]. In fact, Muskal and Kim show an average difference of 4 or 5% and in two cases, 24 to 26%. If the structures are not classified correctly, then the program may try to learn a relationship that is not biologically true, however, this not likely to be a big source of error.

A similar source of error is the variable resolution of the x-ray data on which the classifications are based. Not all protein structures are known to the same resolution and a residue may be classified differently depending on the resolution. Moreover, the secondary structures assigned by DSSP assume that the atomic coordinates in the database are correct, which has not been verified here.

9.3 Does Classification Context Improve Classification Accuracy?

The experiments show that certain kinds of classification context can greatly improve classification accuracy, but that type of context may be difficult to obtain. All context that was provided by a classifier rather than through random context generation failed to improve the results. However, the random context generation experiments showed that if context sources that are less correlated with the classifier's own knowledge can be found, they may provide the basis for improving performance substantially.

9.4 Is Improvement Contingent on Knowledge of the Bias of the Provider of the Context? Does that Knowledge Help?

Knowledge of the bias of the context provider was not sufficient to improve the performance of classifiers. In the case of the random context, the most important factor was the correctness of the context provided, rather than having been trained on a similar level of correctness.

In the case of the context provided through a confusion matrix distribution, nothing was able to improve performance, even using the non-homologous distribution. This may be because the confusion matrix distribution as it stands does not carry enough detail to do what it is trying to do.

For example, proline will always be a coil and the classifier will probably never make the mistake of classifying a proline as something other than a coil. However, the current confusion matrix would allow that because it just sees it as a coil position which will be confused some percentage of the time. Adding another dimension to the confusion matrix so that it showed each residue within each class could carry a lot more information. Still, this does not provide any tertiary information and probably would not provide a substantial improvement in performance.

9.5 If there is Improvement, Is It Bounded?

Since there was no improvement in any case except the random correct context, this question does not apply. In the case of the random context, the improvement depended only on the level of correctness in the context.

Chapter 10

Future Work

While this work has not produced the desired results, it has suggested a number of directions for future work.

Simple minor enhancements

One easy thing to do would be to have longer error-correcting codes than the current 50 bits. Fifty bits was chosen because it made training faster, but more than fifty bits would probably improve performance somewhat.

Another small enhancement would be to write a post-processor like those of other authors to clean up runs where an anomalous classification was inserted. For example, if a β was inserted in the middle of six αs, it could be replaced by an α.

Other black boxes

Since C4.5 was treated as a black box in these experiments, the experiments could be repeated using other learning algorithms (like neural networks) to see if they got similar results.

Get classification context from other classifiers

Many other classifiers exist and it would be interesting to see what effect it would have to build context from their classifications. Two good examples would be Chou-Fasman and Qian and Sejnowski.

Knowledge of supersecondary structure

Perhaps the most promising enhancement would be to train classifiers specific to each of a set of domain or supersecondary structure types. Kneller et al. [23] discusses large improvements in classification performance if the type of the protein is known ahead of time.

As mentioned in the Related Work section, another author [28] has demonstrated a method for providing this information based on the relative proportions of the 20 residue types in the protein, its molecular weight, and whether it contains a heme group. This information could also be easily provided directly to the type of classifier already built in the experiments in this study. It would only require appending 22 more input values to each input window.

Experimental data might provide similar qualitative tertiary information. For example, circular dichroism measurements show approximate proportions of α and β in a protein and could be used in place of or in conjunction wih the 22 input values mentioned above.

Recognizers vs. generators

A completely different approach is suggested by the work in this thesis. Some of the classifiers built here seem to be capable of recognizing a good solution

even if they are not capable of generating one. This was shown in that the classifiers that were fed random correct context were able to guess a correct solution with high reliability and high certainty.

The classifiers that are generating solutions now may be just choosing the most probable solution at each residue in a protein. Any lower probability solution will be ignored, even if its probability is only slightly lower or even if constraints elsewhere in the protein favor it.

The thing that needs to be examined is whether a good evaluation function could be found for use in a global optimizer. For example, we could look at how well the certainty correlates with the accuracy given many different types of context.

If the certainty (in the form of Hamming spreads) was reliable, it could be the basis for an objective function in something like an annealer or a genetic algorithm. The fact that these global optimizers are slow is more tolerable here than in some environments because determining the secondary structure experimentally can take so long.

X-ray phase reconstruction

One final area where it looks like there might be some interesting possibilities is in the area of X-ray diffraction phase reconstruction. Protein tertiary structure determination is generally done using X-ray diffraction. This is a difficult process and far more structures have been sequenced than have had their structure determined through X-ray diffraction.

The X-ray process produces information in the form of Fourier transforms whose amplitude is correct but whose phase is only partially known. It may be possible to learn to predict the phase information from a combination of the correct amplitude information, the partial phase information, the sequence, and the partial predictions of secondary structure. If the phase information could be reconstructed, the time required to determine a structure would be greatly reduced.

Chapter 11

Conclusions

While we have been unable to improve the accuracy of predictions of secondary structure, we have been able to show that knowledge of context classifications is capable of significantly enhancing the accuracy if it has a more random bias than the direct feedback of the current results.

We have also shown that the machine learning techniques we used may be able to recognize good solutions even if they can not generate them. This suggests that there may be a way around the 65% corrrect barrier that everyone seems to encounter if we try to learn evaluation rather than generation of answers and apply the results to global optimization schemes like simulated annealing or genetic algorithms.

It may also be that even though the classifiers in this study are not that good on the non-homologous test data, they may still be worthwhile. They may still perform well on homologous proteins and they do have a fairly reliable indicator

of certainty. It may be important to users to have some partial prediction of the structure of a molecule as a place from which to start their work.

BIBLIOGRAPHY

[1] Christian B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(4096):223–230, 20 July 1973.

[2] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[3] J.R. Quinlan. Decision trees as probabilistic classifiers. In Pat Langley, editor, *Proceedings of the Fourth International Workshop on Machine Learning June 22-25, 1987 University of California, Irvine*, pages 31–37, 1987.

[4] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[5] Thomas G. Dietterich and Gulum Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. Technical Report 91-30-2, Oregon State University Computer Science Dept., 1991.

[6] Ghulum Bakiri. *Converting English Text to Speech: A Machine Learning Approach.* PhD thesis, Oregon State University, 1991.

[7] Ning Qian and Terrence J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884, 1988.

[8] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[9] Gerald D. Fasman. The development of the prediction of protein structure. In Gerald D. Fasman, editor, *Prediction of Protein Structure and the Principles of Protein Conformation*, pages 193–316. Plenum Press, 1989.

[10] G.M. Maggiora, B. Mao, K.C. Chou, and S.L. Narasimhan. Theoretical and empirical approaches to protein-structure prediction and analysis. In *Meth-*

ods of Biochemical Analysis, Volume 35: Protein Structure Determination, volume 35, pages 1–86. 1991.

[11] J. Garnier. Protein structure prediction. *Biochimica et Biophysica Acta*, 72:513–524, 1990.

[12] Thomas Niermann and Kasper Kirschner. Use of homologous sequences to improve protein secondary structure prediction. *Methods in Enzymology*, 202:45–59, 1991.

[13] Nathalie Colloc'h and Fred E. Cohen. β-breakers: An aperiodic secondary structure. *Journal of Molecular Biology*, 221:603–613, 1991.

[14] Henrik Bohr et al. Protein secondary structure and homology by neural networks: The alpha-helices in rhodopsin. *FEBS Letters*, 241(1,2):223–228, 1988.

[15] V.I. Lim. Algorithms for prediction of α-helical and β-structural regions in globular proteins. *Journal of Molecular Biology*, 88:873–894, 1974.

[16] J. Garnier, D.J. Osguthorpe, and B. Robson. Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins. *Journal of Molecular Biology*, 120:97–120, 1978.

[17] P.Y. Chou and G.D. Fasman. Conformational parameters for amino acids in helical, beta-sheet and random coil, regions calculated from proteins. *Biochemistry*, 13:211–222, 1974.

[18] P.Y. Chou and G.D. Fasman. Prediction of protein conformation. *Biochemistry*, 13:222–245, 1974.

[19] P.Y. Chou and G.D. Fasman. Prediction of the secondary structure of proteins from their amino acid sequence. *Advances in Enzymology*, 47:45–148, 1978.

[20] Jr. Peter Prevelige and Gerald D. Fasman. Chou-fasman prediction of the secondary structure of proteins: Chou-fasman-prevelige algorithm. In Gerald D. Fasman, editor, *Prediction of Protein Structure and the Principles of Protein Conformation*, pages 391–416. Plenum Press, 1989.

[21] L. Howard Holley and Martin Karplus. Protein secondary sructure prediction with a neural network. In *Proc. Natl. Acad. Sci. USA*, volume 86, pages 152–156, 1989.

[22] L. Howard Holley and Martin Karplus. Neural networks for protein structure prediction. *Methods in Enzymology*, 202:204–224, 1991.

[23] D.G. Kneller, F.E. Cohen, and R. Langridge. Improvements in protein secondary structure prediction by an enhanced neural network. *Journal of Molecular Biology*, 214:171–182, 1990.

[24] Gilbert Deléage and Bernard Roux. Use of class prediction to improve protein secondary structure prediction: Joint prediction with methods based on sequence homology. In Gerald D. Fasman, editor, *Prediction of Protein Structure and the Principles of Protein Conformation*, pages 587–598. Plenum Press, 1989.

[25] Ross D. King and Michael J.E. Sternberg. Machine learning approach for the prediction of protein secondary structure. *Journal of Molecular Biology*, 216:441–457, 1990.

[26] M. Levitt and C. Chothia. Structural patterns in globular proteins. *Nature*, 261:552–558, 1976.

[27] H. Nakashima, K. Nishikawa, and T. Ooi. The folding type of a protein is relevant to the amino acid composition. *J. Biochem. (Tokyo)*, 99:153–162, 1986.

[28] Steven M. Muskal and Sun-Hou Kim. Predicting protein secondary structure content: A tandem neural network approach. *Journal of Molecular Biology*, 225:713–727, 1992.

[29] Paul Stolorz, Alan Lapedes, and Yuan Xia. Predicting protein secondary structure using neural net and statistical methods. *Journal of Molecular Biology*, 225:363–377, 1992.

[30] Xiru Zhang, Jill P. Mesirov, and David L. Waltz. Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, 225:1049–1063, 1992.

[31] Ken Nishikawa and Tamotsu Noguchi. Predicting protein secondary structure based on amino acid sequence. *Methods in Enzymology*, 202:31–44, 1991.

[32] J. Garnier, J.M. Levin, J.F. Gibrat, and V. Biou. Secondary structure prediction and protein design. 57:11–24, 1990.

[33] S. Thornton et al. Prediction of protein secondary structures using a combined method based on the recognition, lim, and garnier-osguthorpe-robson algorithms. 232:321–336, 1991.

[34] Vellarkad N. Viswanadhan, Benjamin Denckla, and John N. Weinstein. New joint prediction algorithm (q_7-jasep) improves the prediction of protein secondary structure. *Biochemistry*, 30:11164–11172, 1991.

[35] Richard Maclin and Jude W. Shavlik. Refining algorithms with knowledge-based neural networks: Improving the chou-fasman algorithm for protein folding. Technical Report Machine Learning Research Group Working Group Paper 91-2, University of Wisconsin Computer Sciences Dept., 1991.

[36] A.W. Burgess and H.A. Scheraga. Assessment of some problems associated with the prediction of the three-dimensional structure of a protein from its amino acid sequence. In *Proc. Natl. Acad. Sci. USA*, volume 72, pages 1221–1225, 1975.

[37] William R. Taylor and Janet M. Thornton. Recognition of super-secondary structure in proteins. *Journal of Molecular Biology*, 173:487–514, 1984.

[38] F. Kaden, I. Koch, and J. Selbig. Knowledge-based prediction of protein structures. *Journal of Theoretical Biology*, 147:85–100, 1990.

[39] Gerald D. Fasman. A critique of the utility of the prediction of protein secondary structure. *Proc. Int. Symp. Biomol. Struct. Interactions, Suppl. J. Biosci.*, 8(1 & 2):15–23, August 1985.

[40] J. Garnier and B. Robson. The gor method for predicting secondary structures in proteins. In Gerald D. Fasman, editor, *Prediction of Protein Structure and the Principles of Protein Conformation*, pages 417–466. Plenum Press, 1989.

[41] B.W. Mathews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta*, 405:442–451, 1975.

[42] Scott R. Presnell, Bruce I. Cohen, and Fred E. Cohen. A segment-based approach to protein secondary structure prediction. *Biochemistry*, 31:983–993, 1992.

[43] F.E. Cohen, R.M. Abarbanel, I.D. Kuntz, and R.J. Fletterick. Turn prediction in proteins using a pattern-matching approach. *Biochemistry*, 25:266–275, 1986.

[44] Steffen B. Petersen et al. Training neural networks to analyse biological sequences. *Trends in Biotechnology*, 8:304–308, 1990.

[45] Ian A. Wilson et al. Identical short peptide sequences in unrelated proteins can have different conformations: A testing ground for theories of immune recognition. In *Proc. Natl. Acad. Sci. USA*, volume 82, pages 5255–5259, 1985.

[46] Wolfgang Kabsch and Christian Sander. *Biopolymers*, 22:2577–2637, 1983.

Appendices

Appendix A

Proteins Used in Each Data Set

This appendix lists the proteins used in each of the data sets.

Table 22. Non-Homologous Test Set Composition

Brookhaven Database Code	Number of Residues	Residue Proportions % alpha	% beta	% coil	Number of Subunits	Subunit Numbers Used	Protein Name
1ABP	306	34.64	5.88	59.48	1	All	l-Arabinose-binding protein
1ACX	108	0.00	43.52	56.48	1	All	Actinoxanthin
1HMQ	113	64.60	0.00	35.40	4	1	haemerythrin (met)
1IGE	322	4.97	37.58	57.45	2	1	Fe fragment (model)
1NXB	62	0.00	41.94	58.06	1	All	Neurotoxin b
1PPD	212	23.11	16.98	59.91	1	All	2-hydroxyethylthiopapain d
1PYP	281	12.81	9.96	77.22	1	All	Inorganic pyrophosphatase
2ACT	218	25.69	18.35	55.96	1	All	Actinidin (sulphhydryl proteinase)
2ALP	198	4.04	52.53	43.43	1	All	alpha-Lytic protease
2CDV	107	25.23	9.35	65.42	1	All	Cytochrome c3T
3GRS	461	27.11	18.66	54.23	1	All	Glutathione reductase
2LHB	149	67.11	0.00	32.89	1	All	Haemoglobin V (cyano. met)
2SBT	276	21.38	13.77	64.86	2	All	Subtilisin novo
3GPD	334	25.45	20.96	53.59	2	1	Glyceraldehyde-3-P-dehydrogenase
6API	375	29.07	33.07	37.87	2	All	Modified alpha-l-antitrypsin
TOTALS	3522	24.11	21.24	54.66			

Table 23. Full Training Set Composition

Brookhaven Database Code	Number of Residues	Residue Proportions % *alpha*	% *beta*	% coil	Number of Subunits	Subunit Numbers Used	Protein Name
1apr	324	3.40	12.04	84.57	1	All	Acid protease
1aza	129	10.08	33.33	56.59	2	1	Azurin
1azu	125	11.20	27.20	61.60	1	All	Azurin
1bp2	123	43.90	6.50	49.59	1	All	Phospholipase A2
1cac	256	7.03	26.56	66.41	1	All	Carbonic anhydrase form c
1cc5	83	46.99	0.00	53.01	1	All	Cytochrome c5 (oxidized)
1ccr	111	39.64	0.00	60.36	1	All	Cytochrome c (rice)
1cpv	108	48.15	5.56	46.30	1	All	Calcium-binding parvalbumin b
1crn	46	41.30	8.70	50.00	1	All	Crambin
1ctx	71	5.63	22.54	71.83	1	All	alpha-Cobratoxin
1cy3	118	13.56	0.00	86.44	1	All	Cytochrome c3
1cyc	103	33.98	0.00	66.02	1	All	Ferrocytochrome c
1ecd	136	71.32	0.00	28.68	1	All	Haemoglobin (deoxy)
1est	240	5.42	34.17	60.42	1	All	Tosyl-elastase
1fc2	252	14.29	36.11	49.60	2	All	Immunoglobulin FC-Frag B complex
1fdh	288	66.67	0.00	33.33	2	All	Haemoglobin (deoxy, human fetal)
1fdx	54	9.26	7.41	83.33	1	All	Ferredoxin
1fxl	147	29.25	21.77	48.98	1	All	Flavodoxin
1gcn	29	48.28	0.00	51.72	1	All	Glucagon (pH 6-pH 7 form)
1gcr	174	2.87	44.25	52.87	1	All	gamma?-Crystallin
1gfl	70	28.57	0.00	71.43	1	All	Insulin-like growth factor
1gf2	67	29.85	5.97	64.18	1	All	Insulin-like growth factor
1gpl	185	21.08	15.68	63.24	4	1,2	Glutathione peroxidase
1hds	287	52.96	0.00	47.04	4	1,2	Haemoglobin (sickle cell)
1hip	85	11.76	10.59	77.65	1	All	High potential iron protein
1ig2	456	3.29	40.79	55.92	2	All	Immunoglobunlin G1
1ins	52	42.31	5.77	51.92	4	1,2	Insulin
1ldx	329	34.65	13.68	51.67	1	All	Lactate dehydrogenase
1lzl	130	30.00	7.69	62.31	1	All	Lysozyme
1lzm	164	50.61	8.54	40.85	1	All	Lysozyme
1lzt	129	32.56	6.20	61.24	1	All	Lysozyme, triclinic crystal form
1mbd	153	73.86	0.00	26.14	1	All	Myoglobin (deoxy, pH8.4)
1mbs	153	72.55	0.00	27.45	1	All	Myoglobin (met)
1mlt	26	84.62	0.00	15.38	2	1	Melittin
1p2p	124	36.29	4.84	58.87	1	All	Phospholipase A2
1pfc	111	3.60	30.63	65.77	1	All	Fragment of IgC
1ppt	36	50.00	0.00	50.00	1	All	Avian pancreatic polypeptide
1rei	56	0.00	0.00	91.07	2	1	Immunoglobulin B-J fragment V
1rhd	293	27.65	10.92	61.43	1	All	Rhodanese

Table 24. Full Training Set Composition (Part II)

Brookhaven Database Code	Number of Residues	Residue Proportions % *alpha*	% *beta*	% coil	Number of Subunits	Subunit Numbers Used	Protein Name
1rn3	124	17.74	34.68	47.58	1	All	Ribonuclease A
1sn3	65	12.31	18.46	69.23	1	All	Scorpion neurotoxin (variant 3)
1tim	247	42.91	17.00	40.08	2	1	Triose phosphate isomerase
1tgs	282	8.87	34.04	57.09	2	All	Trypsinogen complex
2adk	194	55.67	11.34	32.99	1	All	Adenylate kinase
2ape	308	2.92	33.12	63.96	1	All	Acid proteinase, endothiapepsin
2app	323	9.29	45.51	45.20	1	All	Acid proteinase, penicillopepsin
2b5c	85	24.71	24.71	50.59	1	All	Cytochrome b5 (oxidized)
2cab	256	6.64	30.08	63.28	1	All	Carbonic anhydrase form b
2ccy	127	70.87	0.00	29.13	2	1	Cytochrome c (prime)
2cyp	293	45.73	5.46	48.81	1	All	Cytochrome c peroxidase
2dhb	288	59.72	0.00	40.28	2	All	Haemoglobin (horse, deoxy)
2fdl	106	0.00	0.00	100.00	1	All	Ferredoxin
2gch	239	5.86	32.64	61.51	3	All	gamma?-Chymotrypsin a
2gn5	87	0.00	4.60	95.40	1	All	Gene 5/DNA binding protein
2icb	75	62.67	0.00	37.33	1	All	Calcium-binding protein
2kai	291	5.84	29.55	64.60	3	All	K Allikrein a
2lhl	153	69.93	0.00	30.07	1	All	Leghaemoglobin (acetate, met)
2mcp	443	1.81	47.63	50.56	2	All	Ig Fab mcpc603/phosphocholine
2mdh	650	32.77	16.92	50.31	2	All	Cytoplasmic malate dehydrogenase
2mt2	61	0.00	0.00	100.00	1	All	Cd, Zn metallothionein
2pab	114	7.02	51.75	41.23	2	1	Prealbumin (human plasma)
2rhe	114	0.00	42.98	57.02	1	All	Immunoglobulin B-J fragment V-MN
2sga	181	6.63	54.14	39.23	1	All	Proteinase A
2sns	141	18.44	19.86	61.70	1	All	Staphylococcal nuclease complex
2sod	151	0.00	38.41	61.59	4	1	Cu, Zn superoxide dismutase
2ssi	107	15.89	24.30	59.81	1	All	Streptomyces subtilisin inhibito
2stv	184	9.78	44.57	45.65	1	All	Satellite tobacco necrosis virus
2taa	478	20.71	14.44	64.85	1	All	Taka-amylase a
2tbv	493	1.62	33.27	65.11	6	1,2	Tomato bushy stunt virus,5
3c2c	112	39.29	0.00	60.71	1	All	Cytochrome c2 (reduced)
3cna	237	0.00	40.51	59.49	1	All	Concanavalin A
3fxc	98	7.14	15.31	77.55	1	All	Ferredoxin
3hhb	288	68.06	0.00	31.94	2	All	Haemoglobin (deoxy)
3pcy	99	4.04	35.35	60.61	1	All	Plastocyanin (Hg2+ substituted)
3pgk	415	34.46	11.08	54.46	1	All	Phosphoglycerate kinase complex
3pgm	230	30.00	6.52	63.48	1	All	Phosphoglycerate mutase
3rp2	224	5.36	37.05	57.59	2	1	Rat mast cell protease
3sgb	236	9.32	45.34	45.34	2	All	Proteinase B
3tln	316	37.34	16.46	46.20	1	All	Thermolysin
451c	82	46.34	0.00	53.66	1	All	Cytochrome c551 (reduced)

Table 25. Full Training Set Composition (Part III)

Brookhaven Database Code	Number of Residues	Residue Proportions % *alpha*	% *beta*	% coil	Number of Subunits	Subunit Numbers Used	Protein Name
4cts	437	51.03	4.12	44.85	2	1	Citrate synthase complex
4dfr	159	20.75	30.82	48.43	2	1	Dihydrofolate reductase
4fxn	138	34.06	21.01	44.93	1	All	Flavodoxin (semiquinone form)
4sbv	422	13.27	33.65	53.08	3	1,3	Southern bean mosaic virus coat pro
5atc	464	28.88	13.36	57.76	4	1,2	Aspartate carbamoyltransferase
5cpa	307	35.18	16.29	48.53	1	All	Carboxypeptidase
5ldh	333	37.24	9.31	53.45	1	All	Lactate dehydrogenase complex
5pti	58	13.79	24.14	62.07	1	All	Trypsin inhibitor
5rxn	54	0.00	14.81	85.19	1	All	Rubredoxin (oxidized)
6adh	374	15.51	19.25	65.24	2	1	Alcohol dehydrogenase complex
8cat	498	27.51	15.46	57.03	2	1	Catalase
TOTAL	18064	25.42	19.85	54.71			

Appendix B

Error Correcting Code

This appendix shows the error correcting code used to represent each of the 19 triples classes.

Table 26. Error Correcting Code

101010010110101101010110100101111111001001100011111

011100010100010011100011000001100111111011100000000

101011111101011010000111111011011011100111011000011

100110011010110011111111011101100100001100110101010

100001100111001111101000100001101001011101010110101

100010001111101011101100100011001001001001011011001

101110001100101010100100000010000101111000000001110

110111011011111000001011000001000001111111111011110

001001111010011001011100001011110101110000100111110

000100011101001101000000111100000111111001110110111

011100001000110111000111011000110100000111000110111

101001100100111111111011011100001010110100110001110

111001010011011001100010110010100110000100001110100

001010000000000101101001101010011010101011000101111

011001011010000100101011111111010100010000011001111

010000000111000011001110001101110010110111001011110

000111011110100001000010110110000011000110111001011

111101100111000101010011011011010001111000001110000

001000100011111100010101100000000111110010111001111

Appendix C

Random Noisy Context Tables

This appendix shows the data used to construct the plots for random noisy context in Section 8.1.3 (Figures 8.1.3 through 8.1.3.) It gives tables for classifiers trained on 5, 20, 30, 40, 50, 70, and 80% correct context. (The table for 60Table 20.)

Table 27. Full Noisy Context (Trained on 5% correct context)

Percent Correct	Percent Classified	α			β			Coil		
in Context	Correctly (Q3)	C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	57.3864	0.21	34.98	42.25	0.18	19.52	43.98	0.27	82.01	63.46
5	57.5852	0.22	34.98	43.48	0.18	21.93	42.49	0.28	81.44	63.89
20	57.6136	0.22	35.45	42.45	0.18	20.32	43.93	0.28	81.9	63.89
30	57.2159	0.20	34.51	41.56	0.18	20.45	42.62	0.28	81.54	63.84
40	57.4716	0.20	34.16	40.85	0.19	22.46	43.75	0.30	81.38	64.51
50	57.1023	0.19	33.22	40.58	0.16	19.92	40.93	0.29	82.11	64.16
60	57.4148	0.20	34.63	41.29	0.19	20.99	44.60	0.28	81.64	63.93
70	56.8182	0.19	33.57	40.43	0.19	21.26	43.68	0.27	80.92	63.48
80	57.4432	0.18	32.27	39.77	0.21	21.52	47.63	0.28	82.53	63.66

Table 28. Full Noisy Context (Trained on 20% correct context)

Percent Correct	Percent Classified	α			β			Coil		
in Context	Correctly (Q3)	C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	59.6307	0.31	46.88	48.18	0.11	6.15	47.42	0.31	86.06	63.73
5	52.0455	0.09	24.73	32.46	0.12	20.99	33.62	0.18	76.18	60.89
20	65.4545	0.44	53.36	59.84	0.30	29.14	56.77	0.41	84.92	68.64
30	70.8239	0.56	64.55	67.99	0.42	39.44	67.20	0.49	85.80	72.53
40	75.2841	0.62	71.14	70.81	0.53	45.45	78.70	0.57	88.72	76.33
50	78.9773	0.72	78.68	78.96	0.56	45.72	84.65	0.63	92.04	77.97
60	82.8125	0.79	83.75	84.34	0.63	55.88	86.01	0.69	92.88	81.52
70	84.517	0.82	87.16	85.75	0.68	58.96	90.55	0.71	93.29	82.67
80	87.983	0.86	88.57	89.84	0.76	68.45	94.46	0.78	95.32	85.61

Table 29. Full Noisy Context (Trained on 30% correct context)

Percent Correct	Percent Classified	α			β			Coil		
in Context	Correctly (Q3)	C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	58.6364	0.28	35.92	50.33	0.07	3.209	42.86	0.25	90.22	60.71
5	47.642	0.04	21.20	28.08	0.07	23.40	27.69	0.11	68.75	58.83
20	66.9886	0.47	57.01	61.73	0.35	38.90	55.53	0.44	82.32	71.56
30	72.017	0.59	68.79	69.03	0.45	45.32	65.57	0.51	83.83	74.73
40	77.3864	0.67	73.62	75.67	0.56	52.67	76.50	0.60	88.66	78.25
50	82.3011	0.76	80.80	82.75	0.65	58.16	85.80	0.69	92.36	81.32
60	87.9261	0.85	88.57	88.37	0.75	69.79	91.42	0.78	94.7	86.80
70	89.6591	0.88	90.58	90.90	0.79	73.80	93.56	0.81	95.42	88.05
80	92.5284	0.91	91.28	95.33	0.86	80.61	97.26	0.86	97.71	90.03

Table 30. Full Noisy Context (Trained on 40% correct context)

Percent Correct	Percent Classified	α			β			Coil		
in Context	Correctly (Q3)	C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	59.1761	0.29	42.64	48.14	0.1	4.412	52.38	0.28	87.78	62.40
5	45.4261	0.02	19.79	26.25	0.03	19.39	23.54	0.06	66.87	56.80
20	66.2784	0.47	58.54	60.98	0.33	37.83	54.01	0.42	80.76	71.21
30	72.4432	0.60	69.85	68.87	0.46	46.52	65.66	0.52	83.67	75.58
40	80.3693	0.71	78.92	77.73	0.63	60.16	80.65	0.65	88.87	81.38
50	84.0909	0.79	84.69	83.80	0.68	61.90	87.86	0.71	92.46	83.28
60	89.9148	0.87	91.52	88.80	0.80	75.13	92.59	0.82	94.96	89.60
70	92.6989	0.91	93.88	92.03	0.86	82.75	95.67	0.87	96.05	92.03
80	95.9375	0.95	96.58	96.13	0.93	89.71	98.53	0.92	98.08	94.96

Table 31. Full Noisy Context (Trained on 50% correct context)

Percent Correct in Context	Percent Classified Correctly (Q3)	α			β			Coil		
		C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	58.8636	0.24	28.62	50.31	0.14	6.952	54.74	0.26	92.41	60.4
5	43.0114		20.38	24.75	0.02	23.40	22.94	0.05	60.63	56.66
20	63.7784	0.42	54.89	57.25	0.32	42.38	48.10	0.40	76.03	71.42
30	71.9886	0.58	67.61	69.16	0.46	52.41	61.15	0.52	81.54	76.53
40	80.2557	0.70	75.97	78.18	0.64	66.71	75.61	0.66	87.42	82.6
50	85.6534	0.81	85.51	85.61	0.71	69.12	84.89	0.75	92.15	85.89
60	91.1648	0.89	91.52	91.3	0.82	81.42	90.49	0.84	94.8	91.33
70	94.4602	0.94	95.52	95.19	0.88	86.9	94.75	0.90	96.93	94.05
80	96.875	0.97	97.29	97.64	0.94	92.78	97.2	0.94	98.28	96.43

Table 32. Full Noisy Context (Trained on 70% correct context)

Percent Correct in Context	Percent Classified Correctly (Q3)	α			β			Coil		
		C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	58.9489	0.27	37.93	48.28	0.09	5.214	43.82	0.28	89.13	62.01
5	40.0284	0.0	19.08	23.01		27.27	21.61	0.02	54.24	55.72
20	60.5682	0.38	51.83	54.05	0.29	46.52	42.65	0.36	69.89	71.11
30	69.4034	0.54	64.90	65.36	0.44	58.16	54.65	0.49	75.77	77.46
40	78.5227	0.66	74.20	73.94	0.62	68.98	70.40	0.64	84.14	83.62
50	84.8864	0.79	84.22	83.82	0.71	72.06	81.17	0.74	90.17	86.57
60	90.9091	0.87	89.99	90.20	0.81	82.35	88.13	0.85	94.64	992.2
70	94.4318	0.93	96.00	93.68	0.89	89.17	93.55	0.90	95.79	95.1
80	97.3295	0.97	98.00	97.31	0.95	84.65	96.85	0.95	98.08	97.52

Table 33. Full Noisy Context (Trained on 80% correct context)

Percent Correct	Percent Classified	α			β			Coil		
in Context	Correctly (Q3)	C_{alpha}	TruePos	Hits	C_{beta}	TruePos	Hits	C_{coil}	TruePos	Hits
Full	58.267	0.24	27.56	50.98	0.10	6.818	41.8	0.25	91.84	60.09
5	38.7216	0.00	19.67	22.15		26.07	21.17		52.05	54.25
20	60.5966	0.37	50.18	52.79	0.32	48.4	44.97	0.35	69.94	70.49
30	69.5455	0.54	64.19	65.98	0.44	57.62	54.97	0.49	76.55	77.07
40	79.0057	0.67	73.03	76.17	0.63	71.66	70.16	0.65	84.50	83.68
50	84.7443	0.79	83.51	85.01	0.70	72.33	80.51	0.73	90.12	86.05
60	90.8239	0.86	90.11	89.26	0.83	83.16	89.37	0.85	94.12	92.02
70	94.4034	0.94	95.41	94.96	0.89	99.57	92.67	0.90	95.84	94.80
80	96.9318	0.96	96.70	96.82	0.95	94.25	97.38	0.94	98.08	96.82