

# **Bioinformatics Algorithm Demonstrations** in Microsoft Excel

Robert M. Horton, Ph.D.

© 2004

Robert M. Horton

ALL RIGHTS RESERVED

This work describes a project submitted in partial satisfaction of the requirements for the degree of Master of Science in Computer Science at California State University, Sacramento.

An electronic version is available at <http://www.cybertory.org/exercises>

## **Abstract**

This project presents demonstrations of selected computer science algorithms important in bioinformatics, implemented in the spreadsheet program Microsoft Excel. Spreadsheets provide an interesting platform for demonstration of algorithms, since various steps of the calculations can be exposed in a manner that is easily comprehensible to users with little programming experience. The algorithms demonstrated include two approaches to approximate string matching (dynamic programming and Shift-AND numeric approximate matching), Hierarchical Clustering (used in phylogenetic studies and microarray analysis of gene expression), a Naive Bayes Classifier for simulated microarray gene expression data, and a simple Neural Network. These demonstrations are designed to serve as instructional aids in bioinformatics courses.

## **Dedication**

To my lovely wife Katherine, for her patience, forbearance, and sense of humor.

## **Acknowledgements**

I thank Professors Nick Ewing and Meiliu Lu for giving me the opportunity to co-instruct the graduate course in bioinformatics at CSUS, where the ideas for most of these demonstrations took form. I am grateful to Carl McMillin for helpful discussions, and for helping me past the initial stages of bewilderment when learning to program Visual Basic for Applications. He wrote the simple string class used in the Dynamic Programming algorithm. By abstracting operations such as left-sided concatenation, these make the algorithm code much cleaner.

Several core software components were taken from other sources. The Java graph visualization applet used to display clustering results is taken verbatim from the examples included with the Java 1.2 software development kit. The simulated microarray data used with the naïve Bayes classifier is part of the open source virtual molecular laboratory project at [www.cybertory.org](http://www.cybertory.org). The Tlearn neural network system used to experiment with topologies before attempting to use them with the spreadsheet version is obtained from <http://crl.ucsd.edu/innate/tlearn.html>.

## **Software Specifications**

The spreadsheet demonstration programs were developed in Microsoft Excel 97 on Windows XP Home Edition, service pack 2. Each program has been briefly tested on Excel XP, and seems to operate normally. All programs require that macros be enabled; on Excel XP this is the "low" security setting. Design mode must be "off" for the buttons to work.

Perl scripts for reformatting data were developed with and can be run using Perl 5.8 on Windows XP, available at no charge from [www.activestate.com](http://www.activestate.com).

The Java graph applet used to visualize clustering results was taken from the Java 1.2 software development kit. For the demonstrations in this report, it was used with the Java 1.4 runtime engine.

## Table of Contents

Abstract .....	iii
Dedication.....	iii
Acknowledgements .....	iii
Software Specifications .....	iii
Table of Contents .....	iv
List of Tables .....	v
List of Figures .....	v
Chapter 1. Introduction .....	1
1.1. Biological Background.....	1
1.2. Applications of Bioinformatics Algorithms .....	4
1.2.1. Strings .....	4
1.2.2. Clustering .....	4
1.2.3. Classification .....	5
1.3. Related work .....	5
1.3.1. Spreadsheets in bioinformatics.....	5
1.3.2. Bioinformatics tools .....	6
1.3.3. Algorithm demonstrations .....	7
Chapter 2. Algorithms .....	8
2.1. Approximate String Matching .....	8
2.1.1. Shift-AND Numeric Approximate Matching.....	8
2.1.2. Sequence Alignment and Alignment Scoring with Dynamic Programming.....	9
2.2. Hierarchical Clustering .....	11
2.2.1. Unweighted Pair Group Method with Arithmetic mean (UPGMA) .....	11
2.3. Classification .....	11
2.3.1. Artificial Neural Networks .....	12
2.3.2. Naive Bayes Classifier .....	13
Chapter 3. Demonstration Programs.....	15
3.1. Shift-AND: Shift-AND.xls.....	15
3.2. Alignment by Dynamic Programming: dynamicProgramming.xls .....	15
3.3. Hierarchical Clustering: UPGMA.xls .....	17
3.4. Artificial Neural Networks: ANN.xls .....	19
3.5. Naïve Bayes Classifier for simulated microarray gene expression data: microarrays.xls.....	22
3.5.1. Controls .....	22
3.5.2. Load Training Data .....	23
3.5.3. Enter Training Categories .....	24
3.5.4. Discretize Training Data.....	24
3.5.5. Calculate Probabilities.....	25
3.5.6. Load Unknowns .....	26
3.5.7. Discretize Unknowns.....	28
3.5.8. Classify Unknowns .....	29
3.5.9. Save Results to File .....	30
Chapter 4. Conclusion .....	31
4.1.1. General observations .....	31
Comparisons to related work.....	32
4.1.3. Future work.....	33
Appendix A. Experiments on use of Artificial Neural Networks to learn the genetic code. ....	36
A.1. Software System .....	36
A.2. Data.....	36
A.3. Testing Topologies.....	39
A.4. Implications of Experiments on Topology.....	45
Glossary.....	46
Bibliography .....	51
Index.....	54

## List of Tables

Table 1: Examples of bioinformatics tools using these and related algorithms .....	6
Table 2: Web sites of algorithm demonstrations .....	7
Table 3: Pseudocode for “naïve” pattern matching algorithm. ....	8
Table 4: URLs for Neural Network Software.....	19
Table 5: Comparison of network configuration code.....	21
Table 6: Altman's proposed core components of a bioinformatics curriculum.....	35
Table 7: Standard genetic code represented as a truth table.....	38
Table 8: Simplified truth table for standard genetic code.....	39
Table 9: Four-layer topology can learn the genetic code, but doesn't always.....	43

## List of Figures

Figure 1: Conditional probabilities and Bayes' theorem .....	13
Figure 2: Pure spreadsheet implementation of Shift-AND.....	15
Figure 3: Traversing the matrix to find alignments. ....	16
Figure 4: Distance matrix for hierarchical clustering.....	17
Figure 5: Clustering results in Newick notation. ....	18
Figure 6: Presentation of tree results by Java applet. ....	18
Figure 7: Network architecture for learning Boolean functions, including XOR (prototyped in Tlearn) .....	19
Figure 8: Spreadsheet demonstration of ANN shows layout and weights as the network learns.....	20
Figure 9: Control sheet for Naïve Bayes Classifier .....	22
Figure 10: Loaded training data sheet with categories.....	23
Figure 11: Training data converted to discrete values. ....	25
Figure 12: Spot probabilities calculated for each category.....	26
Figure 13: Measurements from test set ("unknowns") loaded into worksheet.....	27
Figure 14: Measurements from test set ("unknowns") converted to discrete values.....	28
Figure 15: Conditional probabilities for a particular sample. ....	29
Figure 16: Classification of each unknown, with probabilities. ....	30
Figure 19: The standard genetic code .....	36
Figure 20: Perl script to format genetic code for machine learning experiments .....	37
Figure 21: Simple network topology with no hidden nodes.....	40
Figure 22: Simple topology fails to learn genetic code completely .....	40
Figure 24: A topology for learning Serine .....	42
Figure 25: Four-layer topology capable of learning the genetic code, sometimes.....	44
Figure 26: The most effective topology for learning the genetic code treats Serine as a special case. ....	44



## Chapter 1. Introduction

Bioinformatics is the application of information technology and computer science to biological problems, in particular to issues involving genetic sequences. String algorithms are centrally important in bioinformatics for dealing with sequence information. Modern automated high throughput experimental procedures produce large amounts of data for which machine learning and data mining approaches hold great promise as interpretive means. After a brief discussion of general biological issues, I will describe some general problem in bioinformatics, and discuss the relevance to these problems of the algorithms I have chosen to demonstrate.

### 1.1. *Biological Background*

Genetic information flows from DNA to RNA to protein. This principle is known as the **central dogma** of biology.

In most organisms, long-term genetic information is stored in deoxyribonucleic acid (**DNA**) molecules. The information in the DNA is copied from each cell to its progeny during **replication**, controlled by enzymes called **DNA polymerases**. Portions of the DNA molecules (“**genes**”) are copied as needed into short-term “messenger” molecules of ribonucleic acid (**RNA**) in a process called **transcription** by enzymes called **RNA polymerases**. These messages are **translated** into proteins by molecular assemblies called **ribosomes**.

Various types of RNA molecules perform duties other than acting as messengers; for example, transfer RNA (tRNA) molecules are temporarily coupled to amino acids and help to translate sequences of nucleotides in nucleic acids and sequences of amino acids in proteins. Other specialized RNA molecules (ribosomal RNA, rRNA) form large portions of the ribosomes that synthesize proteins.

**Enzymes** are molecules that control particular chemical reactions. Almost all enzymes are proteins (though some include RNA molecules, and some are entirely RNA). Enzymes act by physically interacting with the molecules they affect, and their three dimensional structures are crucial to their activity. Complex biochemical processes typically involve series of chemical steps called **pathways**. Many enzymes change their activity in response to various conditions. For example, some enzymes interact with their own products, and become less active when the concentration of their product is high. Regulation of key enzymes in biochemical pathways is central to control of cellular growth, behavior, and metabolism.

**Proteins** consist of one or more chains of **amino acids**. There are twenty different amino acids commonly found in the proteins of living organisms. The **sequence** of a protein is a specification of the composition and ordering of its amino acids.

The sequence of a protein chain is its **primary structure**. Certain local folding patterns, including the “alpha helix” and the “beta pleated sheet”, are known as **secondary structure**. Both experimental evidence and computer simulations show that secondary structures form quickly [Snow 2002], and prediction of secondary structure is regarded as a step toward predicting higher-level organization. The way a polypeptide chain folds in three dimensions is its **tertiary structure**. If multiple chains (or “subunits”) interact to form a complex (“multimeric”) structure, this is called **quaternary structure**. Under the appropriate conditions, the way a protein folds and all the higher levels of structure are determined by the amino acid sequence of the protein.

Disrupting the higher-level structure of a protein (such as by cooking) is called **denaturation**; denatured proteins typically lose their biological activities. Some proteins will re-fold into their active three dimensional structures even after denaturation. These proteins provide convincing evidence that the key to higher order structure is held in the sequence of the protein itself. Most proteins, however, will not re-fold correctly after denaturation, because the conditions under which they originally folded correctly may not be present. For example, in a cellular environment, some proteins fold into their active configurations in association with "chaperone" molecules, and some have parts chopped off or are otherwise modified after folding.

Because the amino acid monomers in proteins are connected by peptide bonds, a protein chain is a **polypeptide**. Short protein chains may be called oligopeptides, or more commonly simply **peptides**. A peptide bond connects the carboxyl group of one amino acid to the alpha amino group of the next. The first amino acid in a chain thus has a free amino group, and is said to be the **amino terminus** of the chain (or **N-terminus**, because an amino group contains nitrogen). The last amino acid contains a free carboxy group and is called the **carboxy terminus** (or **C-terminus**). By convention, the sequence of a protein chain is written as a series of amino acids with the N-terminus on the left, and the C-terminus on the right. This direction, from N-terminus to C-terminus, is also the direction in which polypeptide chains are normally synthesized by ribozomes. Both single-letter and three-letter abbreviations of amino acid names are commonly used.

DNA molecules consist of long chains (polymers) made of units called **deoxyribonucleotides**. Each deoxyribonucleotide monomer comprises a molecule of the five carbon sugar 5-phospho-2-deoxyribose with a nucleotide "base" attached to carbon number 1. The base is either **adenine (A)**, **cytosine (C)**, **guanine (G)**, or **thymine (T)**. Monomers are connected by phosphodiester bonds, with the oxygen of the number 3 carbon atom of one ribose molecule connected to the phosphate at the number 5 position of the next ribose molecule. To distinguish the carbons in the ribose molecule from the carbons in the nucleotide, those in the ribose are marked with a "prime" after their number. Thus the phosphate groups connect the 3' carbon of one monomer to the 5' carbon of the next. The first nucleotide in a DNA chain has an exposed 5' phosphate group; this is called the **5' end** of the chain. The last nucleotide has a free 3' hydroxy group, and represents the **3' end** of the chain. By convention, DNA sequences are written from left to right in the 5' to 3' direction.

The long covalently linked polymers of DNA are called "strands". DNA is normally "double stranded", with the two strands being connected to one another by relatively weak and reversible hydrogen bonds. The most stable hydrogen bonding arrangement is Watson-Crick **base pairing**, in which the A nucleotides match up with T nucleotides, and C's match with G's. Two strands in which all bases are paired with their appropriate Watson-Crick partner are said to be **complementary**. Paired strands are also described as being **antiparallel**, because the 5' end of one strand pairs with the 3' end of the other; that is, the complementary sequences run in opposite directions. The sequences of the two strands are said to be **reverse complements** of one another; given the sequence of one strand, the sequence of the other strand can be deduced by replacing A with T, T with A, C with G, and G with C (complementing), then reversing the sequence to represent the opposite 5' to 3' direction.

Most of the DNA in most cells is organized into structures called **chromosomes**. In cells that have a nucleus ("eukaryotic" cells), this is where chromosomes reside. Other DNA molecules may exist in **extrachromosomal** locations. For example, eukaryotic cells contain **mitochondria**, subcellular organelles intricately involved in respiration and energy



production. Mitochondria contain their own DNA, as do the **chloroplasts** of plants, organelles involved in photosynthesis. Many **viruses** also reproduce in extrachromosomal locations. The full complement of DNA in the chromosomes of a cell is called its **genome**. The human genome contains slightly over 3 billion bases.

Because each strand contains all the sequence information of the double stranded molecule, one DNA molecule can be made into two identical molecules by separating the strands and filling in the reverse complement of each strand. This is essentially the role played by DNA polymerase during physical replication.

Genetic information is contained in the sequence of bases in the DNA. Sequences of messenger RNAs are directly related to the sequences of the DNA molecules from which they are transcribed, with two notable differences. First, the thymine monomers (T) of DNA sequences are represented by **uracil** (U) in RNA. More drastically, the sequence of the final RNA molecule may be spliced to have some regions removed. The parts that are removed are introns, and the parts that remain in the processed RNA are exons. The order of exons in the spliced product typically reflects their order in the DNA. A region of DNA encoding a particular trait is called a **gene**. Not all of an organism's genes are active in all circumstances or all cell types. Genes which are active in a given cell are said to be expressed. The human genome is believed to contain approximately 25,000 genes.

A typical gene encoding a protein has components that can be described in general terms. A **promoter** is a sequence of DNA where RNA polymerase can bind and begin transcription. A **transcription factor binding site** is a (usually short) sequence of DNA to which proteins that regulate transcription can bind. Different regulatory factors can activate or repress transcription under various circumstances; regulation of transcription is one of several levels at which expression of gene products (including enzymes that in turn regulate biochemical pathways) can be controlled. In many genes, the transcription start site is well defined. Exons and introns (as described above) can usually be mapped to specific positions along the DNA sequence.

The translation process reads three bases of RNA at a time to determine which amino acid to add to a growing protein. Each three-base unit is a **codon**. The relationship between codons and the amino acids they encode is the **genetic code**. Almost all organisms use the same genetic code, with notable exceptions including the genetic codes of various mitochondria. Since there are 64 codons (4 possible bases at each of three positions) and only 20 amino acids, some amino acids are represented by multiple synonymous codons. (Nucleotide mutations that change a codon into another synonymous codon are known as silent mutations). A DNA sequence could potentially encode different proteins depending on which base is chosen to start the first three base codon; this is the **reading frame** for translation. DNA sequences contain three potential reading frames on each strand.

Transcribed messenger RNA contains one or more ribosome binding sites where interaction with the protein translation machinery is initiated. The first codon translated is the **initiation codon**; it is almost always AUG, which encodes the amino acid methionine. Location of the initiation codon determines the actual reading frame for the protein.

Two complementary DNA strands can be separated or “melted” and reassociated or “annealed”. In living cells, the processes of separation and reassociation are controlled by enzymes, but in the laboratory, these reactions can be controlled by heating and cooling. This makes possible an experimental approach called **hybridization**, wherein one strand of DNA, usually containing some sort of detectable marker (such as a radioactive label or a fluorescent

tag) is used as a probe under conditions that favor annealing to detect the presence of its complementary strand in a sample.

Specialized organic chemistry reactions can be used to create **synthetic oligonucleotides**. These can be used as very specific probes in hybridization studies, or as **primers** to initiate template-directed synthesis of DNA at a specific location on a template. Specialized primer extension reactions are the basis of such techniques as chain termination sequencing and the **polymerase chain reaction (PCR)**.

## **1.2. Applications of Bioinformatics Algorithms**

We will consider three classes of algorithms: approximate string matching, for comparing biological sequences, clustering, for inducing relationships among sequences or samples, and classification approaches for assigning sequences or samples to categories.

### **1.2.1. Strings**

Approximate matching of a search pattern to a target (called the “text” in string algorithms) is a fundamental tool in molecular biology. The pattern is often called the “query” and the text is called a “sequence database”, but we will use “pattern” and “text” consistent with usage in computer science. When discussing the space and time complexity of algorithms, the length of the text will be called  $n$ , and the length of the pattern will be  $m$ . While exact string matching is more commonly used in computer science, it is often not useful in biology. One reason for this is that biological sequences are experimentally determined, and may include errors: a single error can render an exact match useless, where approximate matches are less susceptible to errors and other sequence differences. Another, perhaps more important, reason for the importance of approximate matching is that biological sequences change and evolve. Related genes in different organisms, or even similar genes within the same organism, most commonly have similar, but not identical sequences. Determining which sequences of known function are most similar to a new gene of unknown function is often the first step in finding out what the new gene does.

Another application for approximate string matching is predicting the results of hybridization experiments. Since strands may hybridize if they are similar to each other's reverse complements, prediction of which strands will bind to which other strands, and how stable the binding will be, requires approximate, rather than exact, string matching.

### **1.2.2. Clustering**

Clustering, or grouping items by some measure of similarity, can be achieved by a wide variety of methods, including many unsupervised learning methods. Hierarchical clustering is a general term for the grouping of items into tree-like clusters of related groups. A common approach is the pair-group method, where each item is compared to the others, and the two most similar items are joined into the group. This group is then treated as an item, and its distance to other items is calculated. This process is repeated until all items and groups have been joined into a single cluster. The key to this approach is the use of a suitable distance measurement, or (dis)similarity metric to measure how different two items or groups are from one another. Common applications of hierarchical clustering in bioinformatics are grouping of related sequences, grouping of cell or tissue samples based on their gene expression profiles, and grouping of genes based on their expression profiles in different samples. Hierarchical clustering is a type of unsupervised learning, useful for discovering categories among samples.

High-throughput experiments yielding large amounts of data (large numbers of samples, large numbers of measurements per sample, or both) are excellent candidates for automated classification. Experimental results containing significant “noise” may be difficult for humans to classify with confidence, and can be excellent applications for machine classifiers, some of which are amenable to statistical interpretation. One such type of experiment uses gene expression microarrays to simultaneously measure the expression levels of tens of thousands of mRNAs within a sample.

### 1.2.3. Classification

The problem of assigning a sample to a category based on a set of measured attributes is called **classification**. **Supervised learning** methods induce rules for classifying samples from a training set of samples with known classifications. In assessing the accuracy of such classifiers, additional samples of known classification are used as a test set. Many aspects of medical diagnosis can be described as classification problems.

## 1.3. *Related work*

Since this project demonstrates bioinformatics algorithms in Microsoft Excel, I will briefly review the use of spreadsheets in bioinformatics, and describe some existing programs intended to demonstrate the target algorithms for teaching purposes.

### 1.3.1. Spreadsheets in bioinformatics.

Spreadsheets have been used for a variety of sequence analysis applications. Because the twenty amino acids have such a wide variety of chemical characteristics, proteins have many interesting properties that are determined by their amino acid composition. For example, the molecular weight of a protein is computed by summing the weights of the amino acids (minus the water molecules lost in the formation of peptide bonds). The isoelectric point, which is the pH at which the number of positive and negative charges on a protein are equal, is similarly determined by considering the contributions of the constituent amino acids. These parameters of proteins are extremely important in laboratory investigations; when designing a purification strategy to isolate a particular protein from a mixture, knowing such parameters is invaluable. Calculating many such parameters is basically an exercise in accounting, and is easily accomplished with a spreadsheet, without resorting to sophisticated algorithms [Han 1998]. Many fundamental DNA sequence analysis procedures, such as translating to protein sequences and determining codon usage, are also straightforward to implement using spreadsheet functions [McEwan 1998].

By considering a short “sliding window” of a few positions at a time, many accounting-style calculations can be used to make graphs showing how a particular average characteristic varies along the length of a sequence. Plotting hydropobicity along a protein sequence, for example, may reveal which portions of the molecule are likely to be associated with cell membranes. Spreadsheets have also been used for making “dot plots” to visualize the areas of similarity between protein or DNA sequences [Shaw 1997]. Both the sliding window style plots and the matrix-like results of a dot plot are easily displayed in a modern spreadsheet like Excel.

Procedural scripting languages significantly enhance the capabilities of spreadsheets. Scripts have been written to display, analyze, and compare multiple aligned sequences [Delamarche 2000], to calculate melting temperatures of oligonucleotides using nearest-neighbor

thermodynamics [Schütz 1999], and to assist in the design of molecular beacon probes for sensitive real-time detection of specific DNA sequences [Monroe 2003].

Biologists commonly use spreadsheets to design experimental protocols [Stowe 1996] and to organize and analyze experimental data, including data from microarrays [Schageman 2002] and real-time PCR experiments [Schageman 2002]. Indeed, the fact that many biologists are familiar with Excel was a strong motivation to use that platform for algorithm demonstrations.

### 1.3.2. Bioinformatics tools

The next table shows freely available bioinformatics tools that employ the algorithms demonstrated in this project.

**Table 1: Examples of bioinformatics tools using these and related algorithms**

ClustalW	<a href="http://www.ebi.ac.uk/clustalw/">http://www.ebi.ac.uk/clustalw/</a>
agrep	<a href="http://www.tgries.de/agrep/">http://www.tgries.de/agrep/</a>
Cluster/Treeview	<a href="http://rana.lbl.gov/EisenSoftware.htm">http://rana.lbl.gov/EisenSoftware.htm</a>
PHYLIP	<a href="http://evolution.genetics.washington.edu/phylip.html">http://evolution.genetics.washington.edu/phylip.html</a>
Bioconductor	<a href="http://www.bioconductor.org">http://www.bioconductor.org</a>
The Comprehensive R Archive Network	<a href="http://cran.r-project.org/">http://cran.r-project.org/</a>
(Only freely available software is included in this table.)	

ClustalW [Higgins 1994] is a multiple-sequence alignment tool, available in both stand-alone and web-based versions. Multiple sequence alignment is a more complex optimization problem than the two-sequence alignments considered in the demonstration algorithm, but dynamic programming is still used. Clustal uses clustering followed by alignment (hence the name). It avoids the complexity of true multiple sequence alignment by first clustering the genes by edit distance, aligning the most closely related genes (two at a time), and adding the next most related sequence repeatedly until the whole set of sequences is included in the alignment.

Approximate matching by the shift-AND approach (really shift-OR) is implemented by the inventors of the algorithm in the program "agrep" [Wu]. It is named for "approximate grep", since it can be used like the classic Unix grep utility, but allows approximate matching in addition to a subset of regular expressions.

Hierarchical clustering of microarray data is implemented in the program "Cluster", and described in [Eisen 1998]. Various methods for constructing phylogenetic trees, including distance methods similar to UPGMA as well as more sophisticated approaches more suitable to various evolutionary applications, are available in the PHYLIP package [Felsenstein 1989, 2004].

The Bioconductor toolset [Gentleman 2004] is built in the open source "R" statistical language. Modules are available for a wide variety of classification approaches, including Bayesian classifiers.

Neural networks can be used to diagnose cancer subtypes from gene expression data [O'Neil 2003] (that group used a commercial neural network package). The R package "nnet" supports single hidden layer feed-forward neural networks, and may be suitable for classifying cancer samples.

### 1.3.3. Algorithm demonstrations

Programs demonstrating algorithms, especially animations, are commonly used teaching aids in computer science. The next table lists useful web sites that help to index these resources, as well as some demonstrations closely related to my own.

**Table 2: Web sites of algorithm demonstrations**

<i>Catalogs</i>	
The Complete Collection of Algorithm Animations	<a href="http://www.cs.hope.edu/~algaanim/ccaa/">www.cs.hope.edu/~algaanim/ccaa/</a>
Dictionary of Algorithms and Data Structures (DADS), National Institute of Standards and Technology	<a href="http://www.nist.gov/dads/">www.nist.gov/dads/</a>
Exact string matching algorithms: Thierry Lecroq site	<a href="http://www-igm.univ-mlv.fr/~lecroq/string/index.html">www-igm.univ-mlv.fr/~lecroq/string/index.html</a>
Sequence comparison algorithms: Thierry Lecroq site	<a href="http://www-igm.univ-mlv.fr/~lecroq/seqcomp/">www-igm.univ-mlv.fr/~lecroq/seqcomp/</a>
<i>Demonstrations related to those in this project</i>	
Shift Or algorithm	Java animation at <a href="http://www-igm.univ-mlv.fr/~lecroq/string/node6.html">http://www-igm.univ-mlv.fr/~lecroq/string/node6.html</a>
Sequence alignment by dynamic programming	Java animation at <a href="http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node4.html">http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node4.html</a>

Note that I could not find an existing animation for hierarchical clustering. Because of the simplicity and broad applicability of this approach, I consider it very unlikely that no such animation has ever been done. One thing that complicates the search for such demonstrations is that the approach is known by many names, from the general "hierarchical clustering" to the specific UPGMA and "average linkage clustering".

I was also unable to find an animation or interactive demonstration of a naïve Bayes classifier. Again, this is not strong evidence that none exist, but it may reflect the fact that it is hard to make a catchy animation of a Bayes classifier.

## Chapter 2. Algorithms

As might be expected, most of the algorithms that prove useful in bioinformatics are related to familiar problems in computer science. Clustering and classification use well-characterized machine learning approaches. Computer science students venturing into bioinformatics primarily need to understand and suitably frame the problems in order to apply these approaches. The particular problem of approximate string matching, so crucial to biological sequence analysis, is perhaps given less prominence in typical computer science curricula, which often emphasize exact matching approaches (e.g. Boyer-Moore).

### 2.1. Approximate String Matching

Biological sequences can be represented as strings, but the variability implicit to the evolution of living things renders ordinary exact matching approaches of little use. Approximate matching algorithms that can tolerate insertions, deletions, and substitutions are extremely important for biological sequence comparison.

#### 2.1.1. Shift-AND Numeric Approximate Matching

The Shift-AND method uses a bit manipulation approach to accelerate the process of approximate matching. The approach can be explained by comparing it to the naïve exact matching method, in which the pattern is compared character by character at each position along the text. This simple approach is inefficient (its time complexity is  $O(n*m)$ ) because it contains two nested loops, where the inner loop is executed at each position along the text. Shift-AND uses bit-wise operations in entire registers to perform the inner loop operations on multiple positions in parallel. For patterns that can be contained within the length of a register, it has a time complexity proportionate to the length of the text being searched ( $O(n)$ ). Shift-AND actually uses a set of four registers (called the “U” registers [Gusfield 1997] ) to contain the pattern, with one bit in each register to represent each base of the pattern. Thus, a machine with 32 bit registers can easily represent 32 base patterns for use in shift-AND. Longer registers, such as the 128 bit registers of the PowerPC AltiVec vector engine, can represent proportionately longer patterns. The algorithm can also be extended to use multiple registers to represent longer patterns, but (depending on the architecture), this would likely be at a cost of increased time complexity.

**Table 3: Pseudocode for “naïve” pattern matching algorithm.**

```
character[] pattern, text;
integer i,j;
for (i=0;i<length text;i++){
    for (j=0; j< length pattern; j++){
        if (text[i+j] != pattern[j]) next i;
    }
    record_match(i);
}
```

Note: the command “next i” exits the current j loop without completing the call to record\_match.

Of course, shift-AND is more sophisticated than a simple short-circuit of the inner loop in the naïve exact matching approach, in that it can be extended to allow for approximate matches

[Wu 1991]. The full algorithm allows for a given number of substitutions, insertions, or deletions in the pattern. This is achieved by maintaining a set of matrixes, which allow different numbers of errors. A bit is set if the current characters match and the prefixes of each string were within the error limits; if the prefixes had not reached the error limits, the bit is set even if there is an error at the current position. Note that in practice the whole matrix does not need to be maintained, but only the final two columns.

### 2.1.2. Sequence Alignment and Alignment Scoring with Dynamic Programming

The most general and complete approach to approximate matching is to perform sequence alignment between the pattern and the text. This allows each approximate matching position to be assigned a score based on how well it matches. The most commonly used alignment score is the **edit distance**, measured by the number of insertions, deletions, and substitutions it would take to transform one sequence into the other [Gusfield 1997]. Alignment score also serves as a widely used similarity metric to compare related sequences to one another.

The rules for scoring of alignments between DNA sequences are generally simple: some number of points is given for each match, (negative) penalty points are given for mismatches, and penalty points are given for each gap inserted. A different number of points may be given for extending a gap than for initiating a new gap (this is called an affine gap penalty). The user may in general set how many points are given for each match, mismatch, or gap, and different scoring values may be useful in different circumstances. For example, higher gap penalties can be used to favor alignments with fewer gaps.

The same algorithm can be extended to the more complex task of aligning amino acid sequences through the use of a scoring table. Matches and mismatches are not generally treated as quite so black and white for amino acids. Two amino acids may be similar in size, chemical behavior, electrical charge, etc., or may be known to be commonly interchanged within similar proteins. A scoring table allows for “partial credit” when aligning two amino acids that are similar but not identical. Commonly used scoring tables are **PAM** (Percent Acceptable Mutations) and **BLOSUM** (Blocks Substitution Matrix), which use different approaches to represent the frequency with which each amino acid is replaced by each other amino acid in similar positions among similar proteins.

In bioinformatics, the “gold-standard” alignment algorithm is attributed to Smith and Waterman [Smith 1981] for local alignments, or to Needleman-Wunsch [Needleman 1970] for global alignments (but note that [Gusfield 1997] points out that the original Needleman-Wunch algorithm runs in cubic rather than quadratic time). The widely used (quadratic) solutions to both problems can be described as minor variations of a dynamic programming approach [Setubal 1997]. This is done in two phases, first to find the best scores and their positions, and second to determine the alignments themselves. In the first phase, an  $(m+1) * (n+1)$  table is constructed, where each column represents a position in the text, and each row represents a base in the pattern.

The process of filling in the table is as follows. There are three possible ways to arrive at a value for each cell. We will compute the scores for each of these three possibilities, and put the highest value in the table. The first possibility is that the base of the text represented in the column will be paired with the base of the pattern represented by the row. If the bases match, the score will be the score of the diagonal cell (the cell in the previous row and previous column) plus the score for pairing the base in this row of the pattern with the base in this column of the text. For DNA, this is either the match score or the mismatch score, depending on whether the pattern and text match. The second possibility is that a gap is

inserted in the pattern in this position (this is equivalent to a relative deletion in the text). The total score so far with a gap in the pattern is the score of the previous cell on the same row, plus the gap score. The third possibility is a gap in the text; this is computed by adding the gap score to the value in the previous cell of the same column.

Once the table is filled in, the values in each cell represent the maximum alignment score that can be obtained up to the position in the text and pattern represented by the row and column of that cell. The largest number in the entire table represents the highest score of any local alignment between any substring of the pattern and any substring of the text. The score in the last row of the last column represents the highest possible score of a global alignment containing the full strings of both the text and the pattern.

The second phase is to produce the alignments that give the scores in the table. This is done by tracing paths up and back through the table. Each path represents an alignment. Only three possible transitions are allowed from each cell; going to the diagonal cell in the previous row and column represents a pairing between the base in the pattern and the base in the text. Going up to the previous cell in the same column represents inserting a gap in the text, and going to the previous cell in the same row represents a gap in the pattern.

Where we choose to start and stop the paths depends on what kind of alignment we are trying to achieve. To force the alignment to include the end of the pattern, we must start in the bottom row. To force it to include the end of the text, we must start in the last column. To include the beginning of the pattern, we must follow the path to the first row, and to include the beginning of the text, we must follow it to the first column. Thus, a complete global alignment between text and pattern is represented by a path from the cell in the last row of the last column to the cell in the first row of the first column. Shorter paths represent partial (or local) alignments.

Every path through the table (following the rule that we can only go up, left, or to the upper left diagonal from one cell to the next) represents an alignment between the pattern and text, but we are only interested in the “best”, high-scoring alignments. These are found by following the transitions that can account for the value in the cell (the “best score so far” value that we calculated in the first phase of the algorithm). That is, if the score in a cell could have been achieved by adding the gap score to the score in the previous cell in the same column, then we can transition to that cell. Similarly, if the score could have been achieved by adding the gap score to the value in the neighboring cell to the left, we can transition to that cell, and if it could have been achieved by adding the match score (or mismatch score, if the bases don't match) to the score in the upper left diagonal neighbor, then we can transition to that cell.

Clearly, there may be multiple alternative alignments that achieve the same score. We can find them all by recursively tracing the paths until all possible transitions from each cell are traversed.

In the spreadsheet demonstration, the first phase of filling in the table is done using spreadsheet formulas. The second phase, of recursively tracing the paths through the table to find the high-scoring alignments, is done using a recursive script function. The user selects a cell in the table (the cells with high scores are more interesting, but the alignment-finding algorithm will work starting with any cell), and repeatedly clicks the button until all paths are found.



## **2.2. Hierarchical Clustering**

Many clustering approaches are useful for attempting to induce relationships within large data sets. Those that group elements into a hierarchy, or tree-like structure, are of particular interest in biology because they can describe evolutionary relationships.

### **2.2.1. Unweighted Pair Group Method with Arithmetic mean (UPGMA)**

This is a "distance-based" method that works on a matrix of distances (or similarities) between pairs of objects. It can be applied to essentially any situation where distances are additive. In bioinformatics, it is used in such widely divergent applications as construction of evolutionary trees and analysis of microarray gene expression data. It can be used for constructing phylogenetic trees based on edit distances between sequences, though it only achieves correct phylogenies if all branches evolve at equal rates [Durbin 1998]. It was also one of the early methods used to visualize overall patterns of gene expression in genome-scale microarray experiments both by finding groups of genes with similar expression profiles [Eisen 1998], and for grouping cancer cells [Alizadeh 2000].

In each case, the first step is to construct a distance matrix, where every item in the set to be clustered is represented on a row and a column of the matrix, and the values in the matrix represent the distance between the row item and the column item. The distance from an item to itself is typically zero, so the diagonal positions in the matrix are populated by zeros. We assume that the distance from A to B is the same as the distance from B to A, so the values in the matrix are symmetrical about the diagonal. Thus the matrix can be specified by filling in just the lower (or upper) diagonal half of the matrix. For sequence comparisons, edit distance can serve as a suitable distance metric for filling in the matrix. For microarray gene expression data, the Pearson correlation coefficient is used to measure similarity between vectors of expression values for either a given gene in a set of samples [Eisen 1998], or for a set of genes in a given sample [Alizadeh 2000].

The algorithm proceeds by identifying the smallest distance (or greatest similarity) in the matrix, grouping those two items, and building a new matrix where the two grouped item are treated as a new item, whose distance to the other items is determined by averaging the distances of its constituents. This process is repeated until the matrix has a single cell, and all items are in a single group. The result is a rooted tree, or hierarchy.

## **2.3. Classification**

Classification is the process of assigning a sample to a category based on the values of its attributes. One example is classifying an RNA sample based on the expression levels of its genes as determined in a microarray hybridization experiment. You might, for instance, want to know whether the sample came from cancer cells or normal cells, or which virus a patient is infected with. Another type of classification problem is predicting things about sequences. For example, one can use classification approaches to attempt to determine which parts of a protein sequence will assume which secondary structure. A sequence can be conceptually regarded as a number of classifiable attribute vectors by using a "sliding window" a few amino acids in length; the amino acid at each subposition in the window is the value of the subposition attribute of the window at a given position of the window along the protein.

Supervised learning methods use a training set of pre-classified examples to induce rules or patterns by which further samples can be classified. Classic classification approaches based

on supervised learning include decision trees, neural networks, and Bayesian classifiers. Here we will examine the latter two approaches.

### 2.3.1. Artificial Neural Networks

An **artificial neural network** is a set of interconnected **artificial neurons**. Each neuron has a set of inputs, and computes its output based on applying weights to its inputs. Training the network amounts to setting the correct weight values in all the cells.

Some functions can be represented using a single artificial neuron. A simple type of artificial neuron is the perceptron, in which each input is multiplied by a weight, and the weighted sum of the inputs is compared to a threshold. If the weighted sum exceeds the threshold, the perceptron produces a "1"; otherwise, it produces a "0". Training a single perceptron is a simple matter of computing the error between its output and the target from the training data, and adjusting each input weight a small amount to minimize this error. Since the perceptron output is thresholded, it will often learn functions exactly in a reasonably small number of steps.

An individual neuron can only represent a linearly separable function, such as AND, OR, or NOT. It cannot represent "exclusive or" (XOR), for example. However, a multi-layer network can represent a linearly inseparable function like XOR, since this function can be expressed by combining AND, OR, and NOT operations, for example,  $A \text{ XOR } B = (A \text{ AND NOT } B) \text{ OR } (\text{NOT } A \text{ AND } B)$ . The exclusive or function is something of a classic toy problem for neural networks.

Training a multi-layer network is more complex than training a single perceptron. The typical approach to this task is called "backpropagation". Errors for the output layer are calculated based on how closely the output matches the target. These errors are propagated back to nodes in the hidden layers by dividing the error in proportion to the input weights of the nodes receiving inputs from the hidden nodes, but also taking into account the steepness of the hidden node's output with regard to changes in its inputs. Errors are backpropagated recursively all the way to the input layer. To determine the steepness, backpropagation requires that the output of each node be differentiable, making the square step function of the simple perceptron is not suitable. A differentiable output function that maps values into a finite range (say, from -1 to 1) is called a "**squashing function**". A typical squashing function is

$$f(y) = 1 / (1 + \exp(-y))$$

where  $y$  is the weighted sum of inputs [Mitchell 1997]. The derivative of this function is

$$df(y)/dy = f(y) * (1 - f(y))$$

Training can be done either by determining the errors for the whole training set, or on an example-by-example basis; this latter approach is called **stochastic gradient descent**, and is the method used in my demonstration. The artificial neural network is an architecture that allows a great degree of parallelism. At the dawn of the computing age, it was apparent that massive parallelization was one of the striking differences between how computers operated and how the brain must work [von Neumann 1958]. Though many "neural networks" are really computer simulations of networks, ANNs can be implemented in hardware, which is one way to realize the advantages of this potential parallelization.

### 2.3.2. Naive Bayes Classifier

**Bayes' theorem** states a relationship between conditional probabilities. The conditional probability of A given B is written  $P(A|B)$ ; it represents the probability that A is true if B is true. If characteristics A and B are independent, then  $P(A|B) = P(A)$ , but if A depends on B then the conditional probability  $P(A|B)$  will be different from the unconditional probability  $P(A)$ .

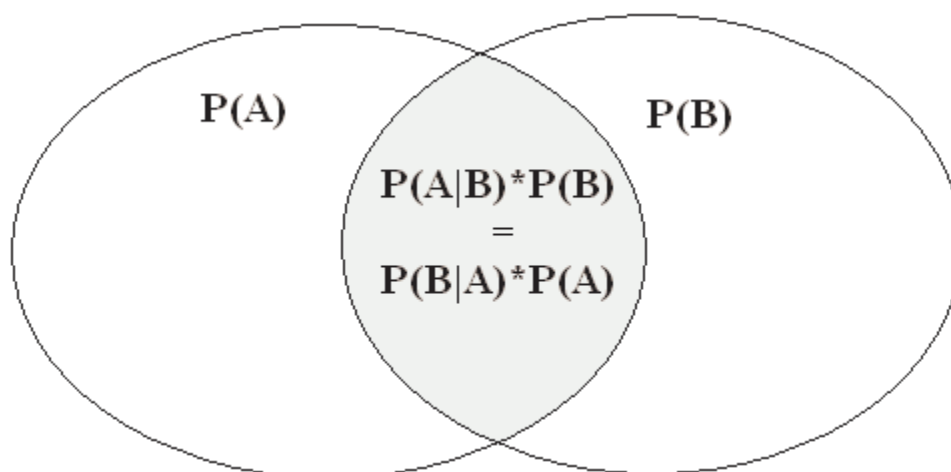


Figure 1: Conditional probabilities and Bayes' theorem

If the probabilities of events A and B are given by ovals in a Venn diagram, then the conditional probability of A given B is the intersection of the two ovals, shown in gray. Since  $P(A|B)$  is a probability relative to B, the absolute area of the gray region is  $P(A|B) * P(B)$ . But we could just as well describe the area of the gray region as  $P(B|A) * P(A)$ , as both descriptions apply to the same area. Thus,

$$P(A|B) * P(B) = P(B|A) * P(A)$$

Algebraic rearrangement of this equality gives Bayes Theorem:

$$P(A|B) = P(B|A) * P(A) / P(B)$$

In machine learning applications, we usually refer to probabilities for "hypotheses" and "data", rather than generic events A and B, and Bayes' theorem is stated as follows [Mitchell 1997]:

$$P(h|D) = P(D|h) * P(h) / P(D)$$

What we want to determine is the probability of a "hypothesis" h, given a set of experimentally observed data D. For a classification problem, each hypothesis is that the sample to be classified belongs to a given category. Bayes' theorem lets us compute this probability for each category; we will classify the sample into the category with the highest probability.

$P(h|D)$  is called the **posterior probability** of hypothesis h because it reflects the probability of a classification after we have observed the data D. The hypothesis for which this probability is highest is called the **maximum a posteriori hypothesis**, or **MAP hypothesis**.

$P(D)$  is the probability of observing a particular set of measurements for any sample, regardless of the category to which the sample belongs. Note that this probability is independent of h. Since we are looking for the hypothesis h that has the highest probability

relative to the other hypotheses, and  $P(D)$  is constant across all hypotheses, we can drop this term and just look for the hypothesis  $h$  for which  $P(D|h) * P(h)$  is the greatest.

$P(h)$ , the unconditional probability of hypothesis  $h$ , is also called the **prior probability** of that hypothesis. This is the probability one would assign to the hypothesis before doing any observations to gather the data,  $D$ . The prior probability might reflect prior knowledge about a system; for example, we might know that certain types of cancer are common, while others are quite rare. Such frequencies could be used to set prior probabilities in a cancer sample classifier. In the absence of prior knowledge, it is common to assume that all hypotheses are equally likely [Mitchell 1997].

The key feature of a Bayesian classifier is that we can calculate the conditional probability  $P(D|h)$  from a **training set** of observations made on samples that have already been classified. For each category (or classification hypothesis) in the training set, we must calculate the probability of a particular set of observations,  $D$ . In practice, we need to make some simplifying assumptions to be able to calculate this value. A particularly significant simplification is to assume that all the individual attributes that make up an observed set of data  $D$  are independent of one another. Using this assumption creates a "**Naïve Bayes Classifier**"; it is called naïve because it blissfully ignores the possibility that attributes might correlate with one another<sup>1</sup>.

In practice, naïve Bayes classifiers have been shown to be very effective even for cases where the assumption of independence of attributes is known to be inaccurate, such as classifying text documents based on the words they use while disregarding the relationships between the words [Graham2002]. As with other machine learning approaches, naïve Bayes classifiers can be validated by determining how well they work on test sets. Though validation can show that a classifier works well even if the assumption of independence is wrong, the absolute values of the "probabilities" determined by the classifier may not be meaningful.

---

<sup>1</sup> For more information, see Wikipedia article on "Naive Bayes Classifier",  
[http://en.wikipedia.org/wiki/Naive\\_Bayes](http://en.wikipedia.org/wiki/Naive_Bayes)

## Chapter 3. Demonstration Programs

Demonstrations of the selected bioinformatics algorithms are implemented as five separate Excel files. Each is described below, along with instructions on their use.

### 3.1. Shift-AND: Shift-AND.xls

The algorithm is demonstrated in two ways. First, spreadsheet functions alone are used to implement the basic, exact-match algorithm, followed by the more complex extensions to this basic algorithm which allow approximate matches. These allow the user to closely inspect the formula for each bit position to determine how it is calculated.

[illegible]

**Figure 2: Pure spreadsheet implementation of Shift-AND**

Finally, an “animated” version of exact matching shows how the text is scanned, the match register (“d”) is shifted by one base, filling in with a 1, the appropriate U register is chosen for each position in the text, and the U-register value is bitwise ANDed with the match register to compute the new match register value. A match is recorded whenever the match register position matching the last base of the pattern is set to 1. This animated version has all values calculated by a script. In both the animated and pure spreadsheet versions, users may enter their own pattern and text values in the designated areas to experiment with the algorithm.

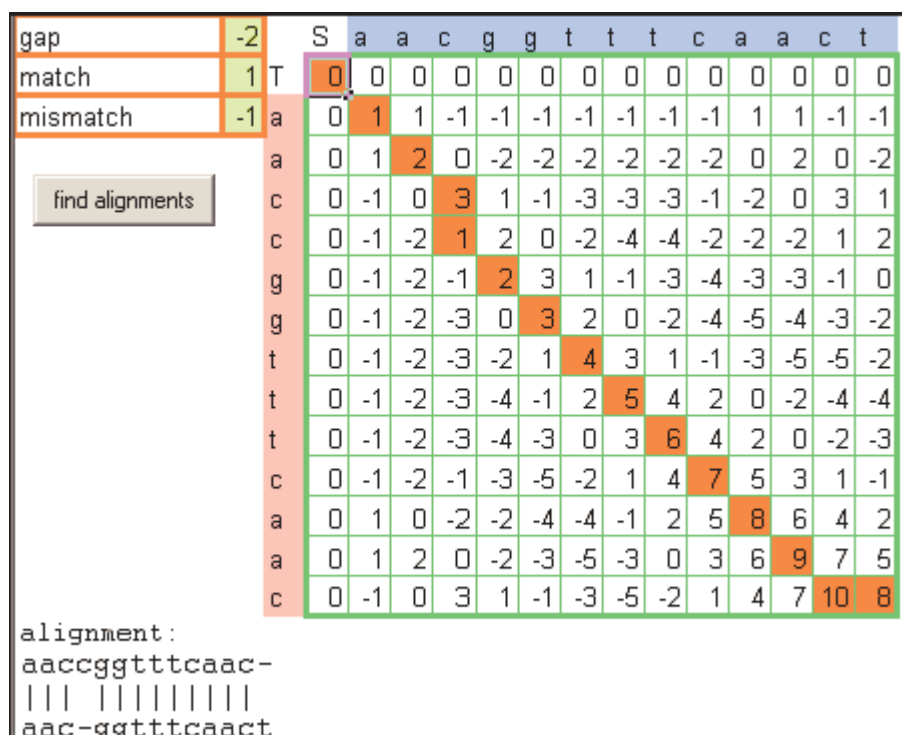
### 3.2. Alignment by Dynamic Programming: *dynamicProgramming.xls*

The dynamic programming approach to sequence alignment is one of the fundamental algorithms in bioinformatics. This approach is guaranteed to find the alignment or alignments between two sequences that has the maximal score according to a set of scoring rules. The characters of sequences to be compared (called "S" and "T") correspond to the columns and rows, respectively, of a matrix. Each path through the matrix represents an alignment between the "S" and "T" sequences. The diagonal path, for example, represents an alignment where each base in "S" is paired with one in "T". A step in the path leading up from one cell to its neighbor above represents a relative gap in "S", whereas a step to the left represents a gap in "T".

Calculating the values in this matrix is the major step in the dynamic programming algorithm. Each cell in the matrix represents a position in S and a position in T, and the value of the cell must represent the maximum possible alignment score of S and T up to those positions. We begin by filling in the first row and column, which do not depend on the

sequences. For a "local alignment", where we are only concerned with the high-scoring region of the alignment between the sequences, the first row and first column are filled with zeros. (If we wanted to penalize an alignment for unmatched bases at the ends, a "global alignment", the first row and columns would be filled with -2, -4, -6, etc., to reflect the gap penalty.)

**Figure 3: Traversing the matrix to find alignments.**



For the remaining cells, we begin in the upper left corner, and consider three possibilities: First, we can get to that cell on the diagonal, in which case the score will equal the score of the upper left neighbor plus the match score (if the bases in S and T match at the positions. Second, we could come down from the neighbor above, achieving a score of the above neighbor plus the (negative) gap penalty. Third, we can come across from the left, to get a score of the left neighbor plus the gap penalty. The score entered into this cell is the highest of these three possibilities. This is done in the spreadsheet using the following formula:

$$\text{cell "F4" value} = \text{MAX}((\text{F3}+\text{gap}),(\text{E4}+\text{gap}),\text{IF}(\$C4=\text{F\$1},\text{E3}+\text{match},\text{E3}+\text{mismatch}))$$

Note that column C contains sequence T and row 1 contains sequence S. The "gap", "match", and "mismatch" scores are held in named ranges of the spreadsheet. These determine the score for any alignment, and changing them will result in the matrix being automatically updated.

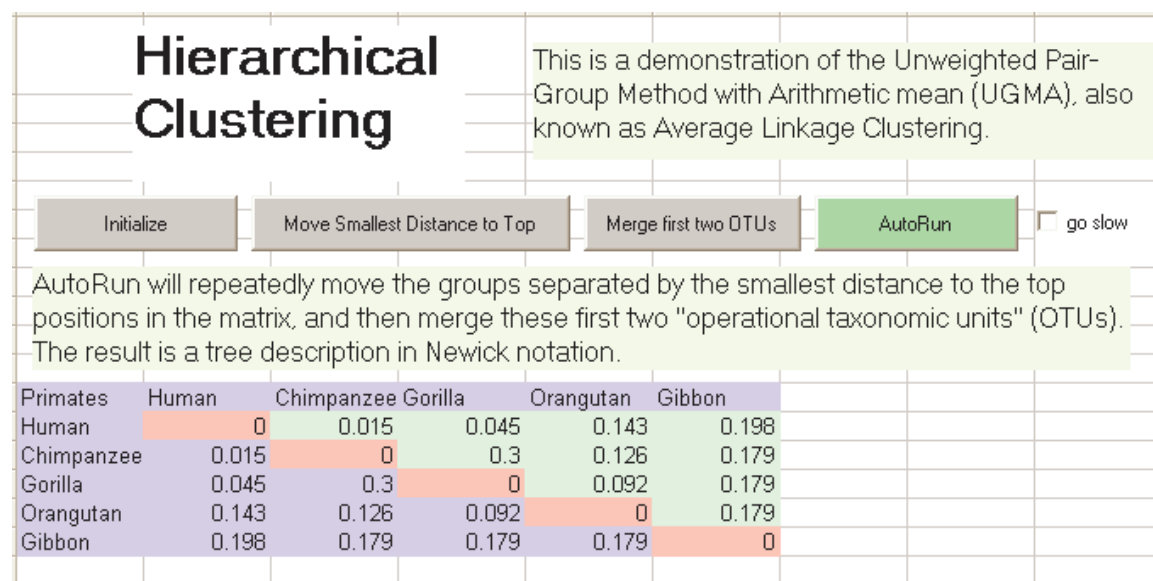
To generate alignments, the user picks a cell in the matrix to mark the end point. Any cell in the matrix can be chosen as a starting point, but some cells are more meaningful than others. For example, the cell in the lower right corner will give alignments that go all the way to the end of both sequences, while the largest number in the bottom row gives the highest-scoring local alignments that extend all the way to the end of the "S" sequence. Click the "find alignments" button to begin tracing paths from your chosen cell back to the first row or column. Paths follow any allowable transitions from cell to cell. Note that the alignment is

constructed in the area below the matrix as each path is highlighted. The algorithm recursively traces all paths that end at the chosen cell, since more than one alignment can end on the same bases and give the same score.

For some applications, the desired result is just the maximum possible score, and the alignments themselves may not need to be determined.

### 3.3. Hierarchical Clustering: UPGMA.xls

The UPGMA algorithm works on a distance matrix representing some distance metric between the items in the set. The demonstration includes sample data showing evolutionary distances between mitochondrial sequences from primate species [Weir 1996]. Since this matrix will be modified by the algorithm, we keep the original on a second worksheet ("Data"), so we can re-initialize easily and start over. Because we are assuming that distances are symmetrical, that is, the distance from A to B is equal to the distance from B to A, and that the distances on the diagonal are all zero, the data are only filled in for the lower triangular portion of the matrix below the diagonal. The zeros on the diagonal and the values in the upper triangular region are filled in by a script when the matrix is copied from the data worksheet during initialization.



**Figure 4: Distance matrix for hierarchical clustering.**

When the initialize button is clicked, the program will copy whatever area of the data worksheet has been selected; this makes it easy to keep other data sets on that worksheet. If no region has been selected, it defaults to using the Primate sample data.

The button marked "Move Smallest Distance to Top" causes the pair of items separated by the smallest distance to be moved to the first and second positions in the matrix. The button marked "Merge first two OTUs" will combine items one and two into a single unit, and shrink the matrix by one. The term "OTU" stands for "operational taxonomic unit" [Weir 1996]; this phrase is really only applicable for using the algorithm for generating evolutionary trees. In fact, the same approach can be used for hierarchical clustering in a wide variety of applications. AutoRun repeatedly moves the closest-spaced pair to the top and merges them, until the matrix is reduced to a single cell. When OTUs are merged, the designation for the new group is written using the "Newick notation" for trees, which

includes a pair of OTUs in parentheses, separated by a comma, annotated with the distance of each OTU to the root of that node. This structure is recursive in that each OTU in the pair can itself be a multi-level description. After AutoRun completes, the description of the one remaining node actually describes the entire tree.

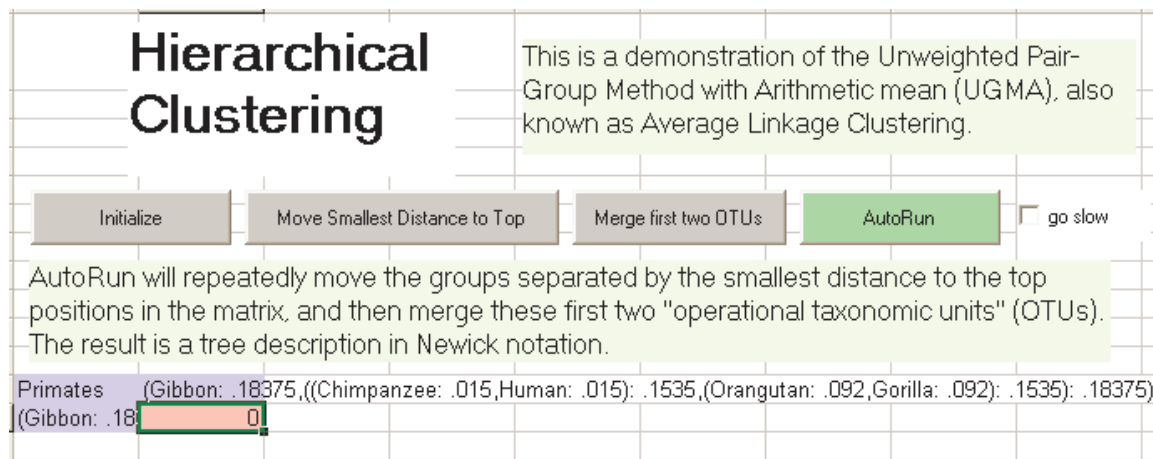


Figure 5: Clustering results in Newick notation.

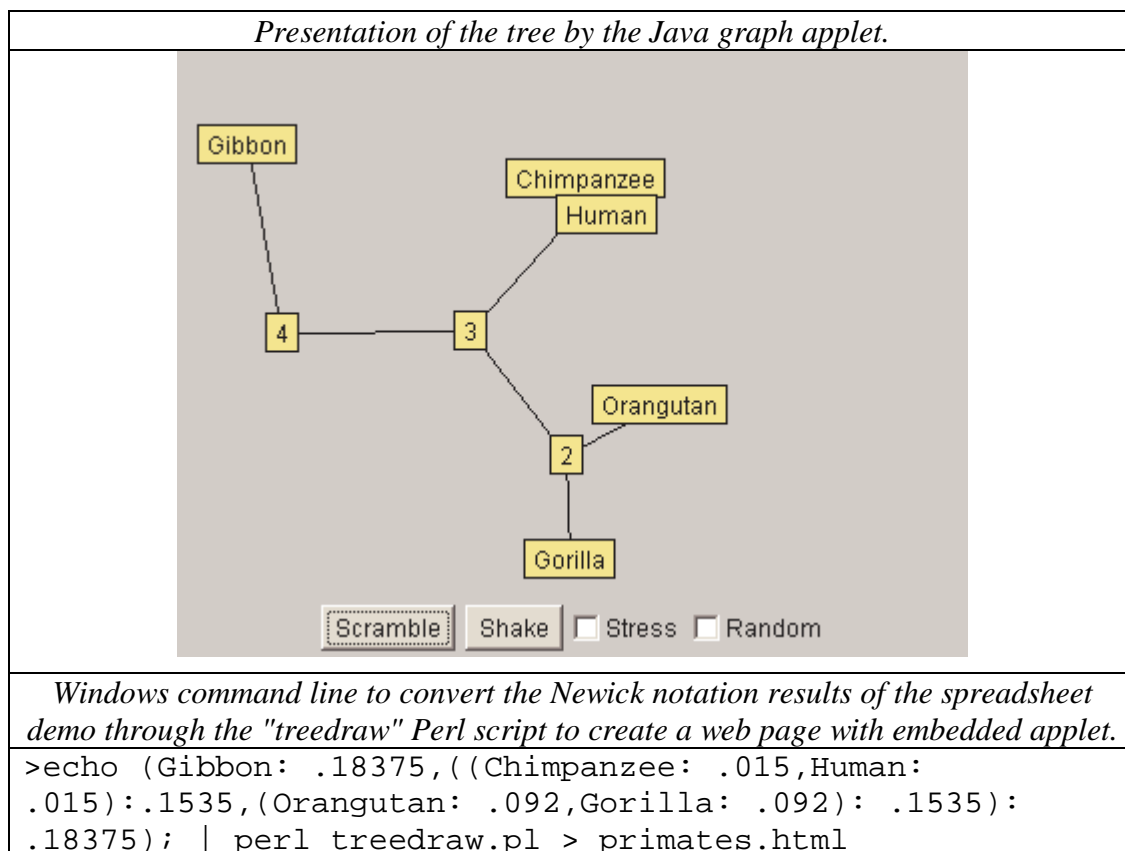


Figure 6: Presentation of tree results by Java applet.

I have written a Perl script to reformat trees from Newick notation so that they may be displayed using the Java Graph applet, which originally came with the Java Software



Development kit as a demonstration developed by Sun. The script takes the Newick notation description from standard input and sends an HTML page including the appropriate applet parameters to standard output. (Note that you must add a semicolon at the end of the Newick format!)

### 3.4. Artificial Neural Networks: ANN.xls

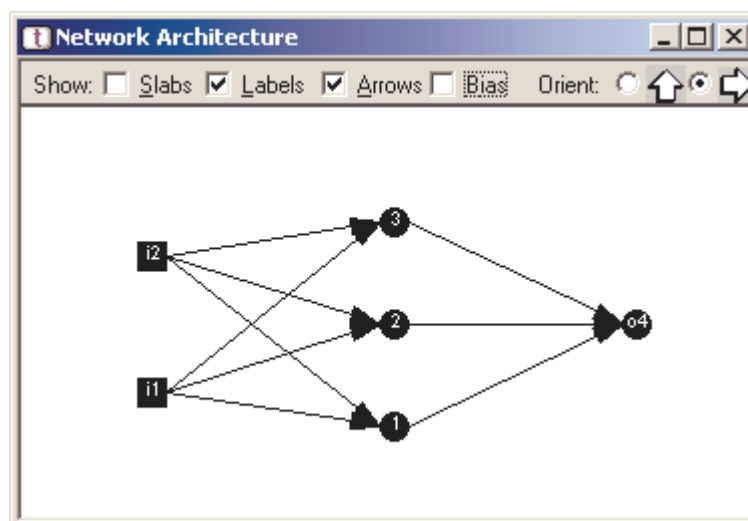
A web search revealed several implementations of neural networks in Microsoft Excel. Many are native code add-ons to add machine learning capabilities that can be used from within a spreadsheet; these are generally commercial products, and are not shown in the table. Others perform the calculations using the spreadsheet itself. I found none that take my approach of laying out the nodes graphically, and letting the user observe the updating of the weights.

Neural networks were prototyped in the program Tlearn (see table for URL). This made it possible to verify that a given artificial neural network topology was capable of learning the problem before that topology was implemented in the Excel demo.

**Table 4: URLs for Neural Network Software**

<i>A neural network implemented in Excel</i>	
Neural Network Models in Excel	<a href="http://www.geocities.com/adotsaha/NNinExcel.html">http://www.geocities.com/adotsaha/NNinExcel.html</a>
<i>Tlearn, the neural network learning package I used for prototyping</i>	
Tlearn	<a href="http://crl.ucsd.edu/innate/tlearn.html">http://crl.ucsd.edu/innate/tlearn.html</a>

Because the ability of a neural network to learn a given function is highly dependent on network topology, I first set out to characterize an architecture that can learn the XOR function reliably. Experiments on topology were done in the program Tlearn, which is described in greater detail in the appendix. The most reliable topology for learning XOR that I was able to find has three hidden nodes, each of which receives both inputs and connects to the single output node. It is theoretically possible for an ANN containing only two hidden nodes to represent XOR, and such architecture did indeed learn the function occasionally in my experiments. However, the topology with three hidden nodes was much less likely to get stuck in local minima, and was my choice for the Excel demonstration.



**Figure 7: Network architecture for learning Boolean functions, including XOR (prototyped in Tlearn)**

Training data for a simple 2-input, 1-output Boolean function is stored in tabular form on the first sheet of the workbook. The user can change the function it represents by modifying the values in the output column of the table. The second worksheet represents a single artificial neuron. A button marked "Next Training Case" loads a row from the function table; the inputs from the table are loaded into the inputs of the neuron, and the output value from the table is loaded into the target value for the neuron. The "Update Weights" button adjusts the weights of the neuron based on how closely the output matches the target. The "Auto Learn" button repeatedly loads the next training case and updates the weights. By clicking "Next Training Case" after the neuron has been trained, you can see how well the output matches the target.

The third worksheet contains a multi-layer neural network. Its inputs and target are also loaded from the same function table. This network is capable of learning the XOR function, while the single neuron is not. Note that the network is not guaranteed to always learn the given function, because it can be trapped at a local minimum.

All of the calculations for both forward data flow and backpropagation in this demonstration are done in spreadsheet formulas, so users can study where the numbers come from at each step. The only parts done by script are loading the input data and target values from the function table, and copying the "new weights" into the current weight cells.

The strengths of this demonstration are that it simultaneously shows the relationships between the nodes and the adjustments of the weights during training. It also exposes all of the calculations, both for feeding results forward and propagating errors backwards, as spreadsheet functions. The user can click on any cell to see its function, and thus learn how its value is derived. None of the basic calculations are done behind the scenes in scripts.

This is not meant to be a general purpose ANN toolset; native tools like Tlearn are orders of magnitude faster, and are much better suited for student use for purposes such as exploring topologies, or trying to create networks to learn more complex functions. I have therefore not added common features such as graphing mean square error during learning.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Learning Rate																	
2	0.5																	
3																		
4	Inputs	1.0000	-2.1992	-0.0012	-2.2004	-2.1992	-0.0998	1.0000	-3.5406	-0.0018	-3.5424	-3.5406	-0.0612	0.0612	0			
5		0.0000	5.5859	0.0000	5.5859	0.0000	-0.0023	0.0998	7.3959	-0.0002	7.3957	0.7383	-0.0035					
6		0.0000	5.6712	0.0000	5.6712	0.0000		0.0040	-7.8431	0.0000	-7.8431	-0.0317						
7								0.5173	0.1990	-0.0009	0.1981	0.1030						
8																		
9		1.0000	-5.5057	0.0001	-5.5056	-5.5057	0.0040											
10		0.0000	3.6221	0.0000	3.6221	0.0000	0.0001											
11		0.0000	3.6318	0.0000	3.6318	0.0000												
12																		
13																		
14		1.0000	0.0694	-0.0001	0.0693	0.0694	0.5173											
15		0.0000	2.0190	0.0000	2.0190	0.0000	-0.0002											
16		0.0000	2.1690	0.0000	2.1690	0.0000												

**Figure 8: Spreadsheet demonstration of ANN shows layout and weights as the network learns.**

The most complex part of the code for this demonstration is not even used by the student; it is the VBA code for laying out the network in the first place. Once the network is laid out, including interconnections between nodes, essentially all calculations are done by the spreadsheet itself. The network layout process used VBA objects to represent a Network and the various Nodes. An object model is built to keep track of the connections between the nodes, the inputs, and the outputs. Once all nodes are added, the object model is used to determine the addresses of the inputs, outputs, weights, and training-related cells for each

node. These addresses are hard-coded into formulas on the spreadsheet, color-coding is applied to emphasize the locations of nodes, inputs and outputs, and named ranges are added to the spreadsheet for convenient access by the run-time scripts that load the training cases.

**Table 5: Comparison of network configuration code**

<i>Tlearn configuration file</i>
<pre> NODES: nodes = 4 inputs = 2 outputs = 1 output nodes are 4 CONNECTIONS: groups = 0 1-4 from 0 1-3 from i1-i2 4 from 1-3 SPECIAL: weight_limit = 1.00 </pre>
<i>Visual basic code to set up network</i>
<pre> Sub makeBooleanNetwork()   Dim net As Network   Set net = New Network   ' Network.init(worksheet, inputCount, outputCount)   Call net.init(Worksheets("boolean_network"), 2, 1)   ' Network.addNode(nodeNumber, inputArray, layer)   Call net.addNode(1, Array("i0", "i1", "i2"), 1)   Call net.addNode(2, Array("i0", "i1", "i2"), 1)   Call net.addNode(3, Array("i0", "i1", "i2"), 1)   Call net.addNode(4, Array("i0", "1", "2", "3"), 2)   ' Network.mapOutput(outputNumes, sourceId)   Call net.mapOutput(1, "4")   Call net.finishLayout End Sub </pre>

The VBA object model is not used when the model is run, only to set up the spreadsheet. This approach makes setting up an alternative topology fairly straightforward for users familiar with Visual Basic for Applications. The configuration code must be edited by hand, and the subroutine containing it is then run from the VBA debugging environment; and example is given in the table.

The most complex part of the code that is actually used during the demonstration run time is that for the "Auto learn" subroutine. The fundamental operation is simply to load each of the training examples, then update the weights in the network. a single pass through the entire set of training examples is called a "sweep", and training usually requires hundreds to thousands of sweeps. The code is made more complex by user interaction requirements, such as periodically showing status and asking if the user wants to continue training. This subroutine also tests to see whether the sum of squared errors has crossed a given threshold, and exits if so.

### 3.5. Naïve Bayes Classifier for simulated microarray gene expression data: *microarrays.xls*.

The Naïve Bayes Classifier uses a supervised machine learning approach to assign samples to categories. It uses Bayes' theorem to determine the probability of each possible classification given a set of observed measurements for a sample, based on the probabilities it has learned from its training sets of each measurement given the known classification.

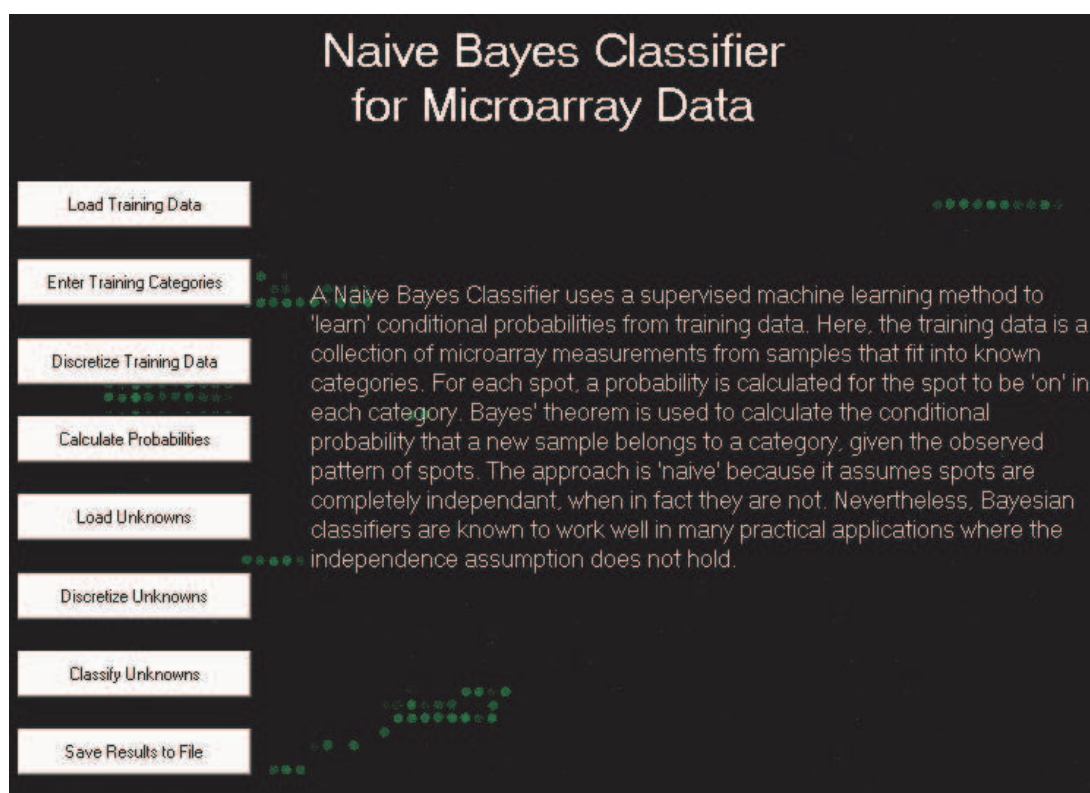


Figure 9: Control sheet for Naïve Bayes Classifier

My demonstration uses simulated microarray data for a “virochip” [Wang 2002, Wang 2003]. This microarray contains approximately 12000 spots, each containing a small sequence of a conserved region from a known virus. By hybridizing a sample of an unknown virus to this array of segments from known viruses, we hope to be able to identify which known virus the sample most closely resembles. The hybridization simulations were done as part of the virtual lab project at [www.cybertory.org](http://www.cybertory.org), and the data are available at that site.

#### 3.5.1. Controls

The first worksheet in the microarray classifier, "controls", contains the buttons that initiate the various steps of the process. Note that the background image shows a microarray; this is a simulated image made with the Cybertory microarray image generator ([www/cybertory.org](http://www.cybertory.org)). Each spot represents a reporter for a known virus, and its intensity reflects how strongly a simulated virus sample is predicted to bind to that reporter. There are 12000 spots on these arrays, and the measurements from each sample are reduced to a vector of 12000 floating point numbers, reflecting the intensities of the green channel of each spot normalized against the red channel.

As you point the mouse to each button, a brief paragraph will be presented describing the corresponding step. Running the classifier is a matter of conducting the steps in order. Various calculations are performed on the different worksheets, accessible by the tabs on the bottom of the screen labeled "controls", "training data", "discretized training data", "probabilities", "unknowns", "discretized unknowns", and "classified unknowns". The spreadsheet has already been loaded with data and run, so each of these worksheets should contain valid entries when the spreadsheet is first opened. To become familiar with the program, users may want to examine each of these worksheets before loading and processing their own data. We will describe the process by discussing each step in turn.

### 3.5.2. Load Training Data

All of the data for this classification exercise is from simulated microarray experiments on various respiratory viruses. Each sample is represented by 12000 intensity values, one for each spot on the microarray. Measurements were simulated using two types of experimental conditions; one set of measurements were taken using 40 degrees as the hybridization temperature, and the other using 50 degrees. The higher temperature produces cleaner results, with lower background signal.

	A	B	C	D	E	F	G	H	I	J	K
1	trainingData	corona	corona	corona	corona	corona	corona	fluA	fluA	fluA	fluA
2	SPOT	Human_co	Human_co	Human_co	Human_co	SARS_cor	SARS_cor	flu_A_Calif	flu_A_Calif	flu_A_Leni	flu_A_Leni
3	1	0.013	0.183	0.043	0.031	0.005	0.08	0.074	0.058	0.3	0.024
4	2	1.473	0.288	0.805	0.178	0.705	0.222	0.463	0.791	0.38	0.768
5	3	0.407	0.221	0.248	0.262	0.215	0.32	0.224	0.303	0.267	0.138
6	4	0.032	0.179	0.262	0.032	0.151	0.085	0.07	0.092	0.062	0.058
7	5	0.17	0.05	0.951	0.547	0.45	0.275	0.369	0.127	0.544	0.126
11981	11979	1.679	0.852	0.594	0.82	0.547	1.029	1.701	0.659	0.514	0.982
11982	11980	0.149	0.116	0.244	0.11	0.108	0.526	0.073	0.183	0.097	0.514
11983	11981	0.01	0.134	0.124	0.032	0.055	0.043	0.013	0.123	0.047	0.098
11984	11982	0.39	1.121	0.756	0.656	0.365	0.695	1.028	0.235	0.891	0.421
11985	11983	0.1	0.172	0.201	0.049	0.081	0.319	0.028	0.044	0.065	0.058
11986	11984	0.507	1.027	1.33	0.753	0.73	0.985	0.999	1.285	1.73	0.444
11987	11985	1.659	0.441	1.205	1.409	1.508	0.731	0.82	1.456	1.106	1.724
11988	11986	0.029	0.024	0.027	0.04	0.05	0.017	0.431	0.022	0.024	0.022
11989	11987	0.247	0.434	1.21	0.274	0.237	0.212	0.08	0.233	0.191	0.514
11990	11988	0.124	0.057	0.656	0.81	0.024	0.154	0.607	0.74	0.26	0.367
11991	11989	0.93	0.751	0.274	0.527	0.702	0.365	0.284	0.516	0.322	0.304
11992	11990	0.1	0.084	0.134	0.379	0.141	0.046	0.12	0.036	0.098	0.102
11993	11991	0.365	0.125	0.051	0.02	0.124	0.085	0.155	0.095	0.292	0.112
11994	11992	0.025	0.033	0.064	0.023	0.181	0.031	0.086	0.024	0.062	0.029
11995	11993	0.998	0.11	0.76	0.346	0.347	1.172	0.623	0.548	0.596	0.407
11996	11994	0.404	0.857	0.03	0.031	0.139	0.227	0.392	0.133	0.879	0.207
11997	11995	0.888	0.52	0.74	1.232	0.602	0.561	1.051	0.552	1.051	0.513
11998	11996	0.33	0.029	0.046	0.454	0.26	0.007	0.142	0.027	0.355	0.041
11999	11997	0.505	0.165	0.721	0.497	0.163	0.793	0.163	1.019	0.785	0.058
12000	11998	0.254	0.255	0.292	0.328	0.055	0.182	0.135	0.25	0.113	0.318
12001	11999	0.334	0.105	0.415	0.1	0.41	0.407	0.416	0.202	0.559	0.268
12002	12000	0.075	0.614	0.179	0.935	0.264	0.552	0.754	0.193	0.243	0.474
12003	AVERAGE	0.439046	0.440739	0.433108	0.433091	0.440141	0.439287	0.434646	0.437088	0.436068	0.442072
12004	STDEV	0.410734	0.412982	0.402025	0.397865	0.403966	0.404361	0.403318	0.404512	0.402454	0.409205
12005											

Figure 10: Loaded training data sheet with categories.

The collection of simulated experiments from has been divided into a "training set" and a "test set", each representing samples from several kinds of viruses. These sets have been saved in the four tab-delimited text files in the "data" directory: "testData40.exp", "testData50.exp", "trainingData40.exp", and "trainingData50.exp". When you click "Load Training Data", you will be asked to choose one of these files to load. They are all in the

same format, and you can load test data instead of training data if you desire, though the training set is larger, and may give better statistical power to the classifications.

### 3.5.3. Enter Training Categories

The training set contains representative viruses from various taxonomic groups of viruses. There are a variety of ways in which they could potentially be categorized. For example, all parainfluenza viruses (PIV) could be put into a single group, or they could be put into the more detailed taxonomic groups PIV1, PIV2, and PIV3. Similarly, all influenza viruses could be considered as a single category, or they could be divided into the more specific subtypes A, B, and C. To specify how the training samples should be categorized, a dialog box asks the user to enter a group name for each sample. These names are copied into the top row of the column containing the measurements for each sample in the "training data" worksheet. After a category name has been entered for each sample, the columns are sorted by category, to bring all samples of the same category next to one another, and each category is assigned a color. This makes it easy to visually inspect the training data to ensure that the intended categories were entered correctly. Spelling errors in category name will create new, unintended categories; these should be particularly obvious when the columns are sorted and the categories color-coded. Any errors should be corrected by typing the correct category name into the cell containing the erroneous name. After inspecting the "training data" worksheet, click the "controls" tab to choose the next step. Clicking "Enter Training Categories" again will re-sort and re-color the categories.

### 3.5.4. Discretize Training Data

The next step is to turn the continuous intensity values into discrete 1 (a positive spot) and 0 (a negative spot) values. Deciding whether a spot is on or off is a matter of deciding whether it is significantly brighter than the average spot in the same sample. A prompt requests that you enter a number of standard deviations to be used for a cutoff. The mean and standard deviation are calculated for each sample column, and any spot more than the given number of standard deviations above the mean is considered positive, while the others are negative. The sum below each column indicates the number of positive spots.

	A	B	C	D	E	F	G	H	I	J	K	
1	cutoff	corona	corona	corona	corona	corona	corona	fluA	fluA	fluA	fluA	fluB
2	3	Human_cc	Human_cc	Human_cc	Human_cc	SARS_co	SARS_co	flu_A_Cali	flu_A_Cali	flu_A_Len	flu_A_Len	flu_
3	1	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	0	0
5	3	0	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0	0	0	0
7	5	0	0	0	0	0	0	0	0	0	0	0
11980	11978	0	0	0	0	0	0	0	1	0	0	0
11981	11979	1	0	0	0	0	0	1	0	0	0	0
11982	11980	0	0	0	0	0	0	0	0	0	0	0
11983	11981	0	0	0	0	0	0	0	0	0	0	0
11984	11982	0	0	0	0	0	0	0	0	0	0	0
11985	11983	0	0	0	0	0	0	0	0	0	0	0
11986	11984	0	0	0	0	0	0	0	0	1	0	0
11987	11985	0	0	0	0	0	0	0	0	0	1	0
11988	11986	0	0	0	0	0	0	0	0	0	0	0
11989	11987	0	0	0	0	0	0	0	0	0	0	0
11990	11988	0	0	0	0	0	0	0	0	0	0	0
11991	11989	0	0	0	0	0	0	0	0	0	0	0
11992	11990	0	0	0	0	0	0	0	0	0	0	0
11993	11991	0	0	0	0	0	0	0	0	0	0	0
11994	11992	0	0	0	0	0	0	0	0	0	0	0
11995	11993	0	0	0	0	0	0	0	0	0	0	0
11996	11994	0	0	0	0	0	0	0	0	0	0	0
11997	11995	0	0	0	0	0	0	0	0	0	0	0
11998	11996	0	0	0	0	0	0	0	0	0	0	0
11999	11997	0	0	0	0	0	0	0	0	0	0	0
12000	11998	0	0	0	0	0	0	0	0	0	0	0
12001	11999	0	0	0	0	0	0	0	0	0	0	0
12002	12000	0	0	0	0	0	0	0	0	0	0	0
12003	SUM	141	128	142	128	142	151	148	139	133	141	

Figure 11: Training data converted to discrete values.

### 3.5.5. Calculate Probabilities

Among the samples within each category, the program counts how many of each spot are positive and how many are negative, and computes an observed probability of the spot being on for each category. It is not quite as simple as dividing the number of positive spots by the number of samples in the category, however, because we must take care not to let probabilities go to zero.



	A	B	C	D	E	F	G	H	I	J	K
1		Overall Prc	corona	fluA	fluB	fluC	piv1	piv2	piv3	rhino	rsv
2											
3	1	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
4	2	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
5	3	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
6	4	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
7	5	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
8	6	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
9	7	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
10	8	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11	9	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
12	10	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
13	11	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
14	12	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
15	13	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11990	11988	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11991	11989	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11992	11990	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11993	11991	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11994	11992	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11995	11993	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11996	11994	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11997	11995	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11998	11996	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
11999	11997	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
12000	11998	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
12001	11999	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
12002	12000	3.23E-05	0.000143	0.0002	0.0002	0.0002	0.000333	0.000333	0.000333	0.0002	0.000333
12003											
12004											
12005											
12006											

**Figure 12: Spot probabilities calculated for each category.**

This is because we will be calculating the probability for the whole set of spot intensities by multiplying the individual spot intensities together (the naïve assumption of statistical independence). If any spot's probability is zero, the product of all the probabilities will be zero. The traditional approach adds a number of "virtual samples" to the actual observations. Each virtual sample is fractionally positive; this fraction represents the apriori probability, which is just our guess as to how many spots will be turned on. The program prompts for a number of virtual samples and for an apriori probability. The probability of a spot being "on" is then given by:

$$P(\text{spot on in category}) = \frac{(\text{number of positive spots} + (\text{number of virtual samples} * \text{apriori probability}))}{(\text{number of samples in category} + \text{number of virtual samples})}$$

Note that as long as you have some virtual samples, and a nonzero apriori probability, the spot probability will never be quite zero.

In addition to calculating the spot probabilities for each category, this button sets up the area where conditional probabilities will be calculated.

### 3.5.6. Load Unknowns

The unknowns are the test set. They can be loaded from the tab-delimited files in the data directory, just as the training set was loaded. You will not need to enter categories for the unknowns, though, because that is what the classifier will do. The test set values will be loaded into the "unknowns" worksheet. Note that each sample in the test set has a name, which shows the virus it represents. For an actual unknown, we would not have that information. The classifier does not look at the name, though, just the values. After



classification, we can check the names against the determined category to see if the classification is correct.

	A	B	C	D	E	F	G	H	I	J	K
1	testData40.exp										
2	SPOT	Human_R	Human_R	Human_co	Human_co	Human_pa	Human_pa	Human_rhi	Human_rhi	SARS_cor	SARS_co
3	1	0.101	0.073	0.052	0.08	0.242	0.013	0.113	0.096	0.043	0.061
4	2	0.512	0.423	0.612	0.416	0.165	1.264	0.293	0.683	1.069	0.408
5	3	1.251	0.124	0.977	0.149	0.322	0.098	0.675	0.027	0.065	0.319
6	4	0.475	0.394	0.059	0.244	0.121	0.063	0.041	0.101	0.651	0.102
7	5	0.131	0.616	0.14	0.085	0.553	0.237	0.308	0.071	0.526	0.991
11982	11980	0.561	0.048	0.266	0.116	0.036	0.279	0.572	0.047	0.374	0.076
11983	11981	0.01	0.115	0.029	0.057	0.016	0.12	0.017	0.087	0.103	0.031
11984	11982	0.062	0.555	0.384	0.653	0.183	0.256	0.275	0.403	1.001	0.298
11985	11983	0.067	0.272	0.2	0.079	0.019	0.185	0.039	0.105	0.04	0.044
11986	11984	0.235	1.331	1.213	0.699	0.206	1.783	0.409	1.519	1.274	1.44
11987	11985	1.704	0.203	1.4	1.172	0.951	0.637	1.151	0.766	0.364	1.811
11988	11986	0.016	0.034	0.188	0.003	0.034	0.07	0.02	0.064	0.02	0.032
11989	11987	0.222	0.64	0.146	0.056	0.526	0.193	0.382	0.413	0.236	0.551
11990	11988	0.204	0.266	0.14	0.109	0.334	0.162	0.08	0.108	0.347	0.157
11991	11989	0.279	0.216	0.458	0.342	0.968	1.132	0.243	0.124	1.03	1.219
11992	11990	0.845	0.235	0.111	0.026	0.12	0.354	0.379	0.259	0.216	0.274
11993	11991	0.217	0.276	0.188	0.111	0.009	0.042	0.046	0.107	0.108	0.316
11994	11992	0.06	0.107	0.047	0.012	0.112	0.139	0.007	0.057	0.257	0.033
11995	11993	0.259	0.77	1.233	0.136	0.126	1.389	1.025	0.337	0.239	1.607
11996	11994	0.117	0.206	0.263	0.484	0.201	0.186	0.089	0.108	0.111	0.16
11997	11995	0.941	1.031	0.937	0.716	0.404	0.248	0.706	0.94	0.713	0.878
11998	11996	0.099	0.278	0.089	0.06	0.052	0.187	0.645	0.052	0.136	0.157
11999	11997	0.892	0.191	0.205	1.093	0.333	1.031	0.065	0.514	0.752	0.179
12000	11998	0.219	0.145	0.215	0.179	0.634	0.148	0.049	0.037	0.158	0.089
12001	11999	0.193	0.099	0.246	0.11	0.119	0.027	0.078	0.255	0.12	0.114
12002	12000	0.168	1.078	0.34	0.159	0.268	0.467	0.144	0.401	0.514	0.245
12003	AVERAGE	0.440405	0.44262	0.446602	0.441752	0.433815	0.437559	0.455954	0.452294	0.435697	0.433125
12004	STDEV	0.417391	0.416198	0.422657	0.419097	0.403359	0.40604	0.447909	0.445971	0.404822	0.402302
12005											
12006											

Figure 13: Measurements from test set ("unknowns") loaded into worksheet.

### 3.5.7. Discretize Unknowns

The test set is converted to discrete values just as the training set was. You will be asked to enter a standard deviation cutoff as before. The results will appear in the "discrete unknowns" worksheet.

	A	B	C	D	E	F	G	H	I	J	K
1	cutoff										
2	3	Human_R	Human_R	Human_co	Human_co	Human_pa	Human_pa	Human_rhi	Human_rhi	SARS_cor	SARS_co
3	1	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	0
5	3	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0	0	0
7	5	0	0	0	0	0	0	0	0	0	0
11980	11978	0	0	0	0	0	0	0	0	0	0
11981	11979	0	0	0	0	0	0	0	0	0	0
11982	11980	0	0	0	0	0	0	0	0	0	0
11983	11981	0	0	0	0	0	0	0	0	0	0
11984	11982	0	0	0	0	0	0	0	0	0	0
11985	11983	0	0	0	0	0	0	0	0	0	0
11986	11984	0	0	0	0	0	1	0	0	0	0
11987	11985	1	0	0	0	0	0	0	0	0	0
11988	11986	0	0	0	0	0	0	0	0	0	0
11989	11987	0	0	0	0	0	0	0	0	0	0
11990	11988	0	0	0	0	0	0	0	0	0	0
11991	11989	0	0	0	0	0	0	0	0	0	0
11992	11990	0	0	0	0	0	0	0	0	0	0
11993	11991	0	0	0	0	0	0	0	0	0	0
11994	11992	0	0	0	0	0	0	0	0	0	0
11995	11993	0	0	0	0	0	0	0	0	0	0
11996	11994	0	0	0	0	0	0	0	0	0	0
11997	11995	0	0	0	0	0	0	0	0	0	0
11998	11996	0	0	0	0	0	0	0	0	0	0
11999	11997	0	0	0	0	0	0	0	0	0	0
12000	11998	0	0	0	0	0	0	0	0	0	0
12001	11999	0	0	0	0	0	0	0	0	0	0
12002	12000	0	0	0	0	0	0	0	0	0	0
12003	SUM	149	142	123	152	133	145	147	158	135	147

**Figure 14: Measurements from test set ("unknowns") converted to discrete values.**

### 3.5.8. Classify Unknowns

One by one, the unknown samples are loaded into the blue "samples" column on the probabilities worksheet. Then the conditional probabilities for each spot in each category are calculated.

	K	L	M	N	O	P	Q	R	S	T	U
1	rsv		sample	corona	fluA	fluB	fluC	piv1	piv2	piv3	rhino
2			flu_C_Miya	5.8779E-306	0	0	1	0	0	0	0
3	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
4	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
5	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
6	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
7	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
8	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
9	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
10	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
12	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
13	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
14	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
15	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11990	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11991	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11992	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11993	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11994	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11995	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11996	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11997	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11998	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
11999	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
12000	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
12001	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
12002	0.000333		0	-0.000142867	-0.0002	-0.0002	-0.0002	-0.00033	-0.00033	-0.00033	-0
12003				-968.2021321	-1051.96	-978.117	-265.382	-1124.12	-1067.4	-1118.12	-13
12004				-702.8198459	-786.574	-712.735	0	-858.736	-802.022	-852.742	-10
12005				5.8779E-306	0	0	1	0	0	0	0

Figure 15: Conditional probabilities for a particular sample.

Unfortunately, for performance reasons, the spreadsheet formulas are converted to plain numbers right after the calculation is done, so you can't see the formula. Therefore I will explain it here. Say the sample column is "M" (the actual column will depend on how many categories are used). Then the conditional probabilities for the first category will be in column "N". The first spot is on row 3, since rows 1 and 2 are reserved for the category name and overall probability, respectively.

The observed probabilities for the first category are in column "C". Finally, to deal with the very small probability numbers that would result from multiplying a huge number of small values together, we will convert everything to logarithms. The formula for cell N3 is "LN(IF(\$M3,C3,1-C3))". In other words, if the spot is positive in the sample, then we take the observed probability for the spot being positive in this category. Otherwise, we use one minus this probability, which is the probability that the spot will be negative. Finally, we take the logarithm of the probability, to avoid underflow problems with extremely small multiplication products being rounded to zero.

The overall probability of the sample being in the category is reflected in the product of all the spot probabilities. This product will need to be normalized. Assuming that the sample

does indeed belong to one (or more) of the categories, we will force the normalized probabilities to add up to one. This takes a few steps to avoid underflow. Since we are working with logarithms, we add them instead of multiplying; the sums of the logs of the spot probabilities are given in the first row beneath the pink area. These numbers may be fairly large negative exponents, and we may still have a rounding to zero problem if we take e to these powers. So first we find the maximum value in the row (that is, the smallest negative number), and subtract this from all the exponents. This is like multiplying all of the probabilities by a constant. One of the subtracted exponents will be zero. Now we can raise e to the power of these subtracted exponents and be confident that, even if some of the results round to zero, not all of them will, and the sum will be nonzero. This sum is the divisor for normalizing the category probabilities so that they add to one.

	A	B	C	D	E	F	G	H	I	J	K
1		corona	fluA	fluB	fluC	piv1	piv2	piv3	rhino	rsv	
2	Human_Rs	5.4E-161	2.6E-175	1.1E-175	3.3E-167	3.4E-200	1.1E-159	1.3E-195	1.6E-288	1	rsv
3	Human_Rs	7.8E-151	1.3E-171	7.3E-184	1E-159	2.1E-170	1.8E-158	4.6E-188	4.6E-284	1	rsv
4	Human_co	1	1.31E-84	1.22E-87	2.69E-68	4.65E-87	2.6E-78	2.98E-83	6.8E-190	5.62E-95	corona
5	Human_co	1	2.6E-99	2.7E-108	5.28E-85	3.9E-118	1.5E-112	8.6E-105	1E-218	9.1E-111	corona
6	Human_pa	3.28E-21	2.56E-52	1.63E-36	1.63E-61	1.1E-75	1	2.77E-74	2.3E-176	2.25E-64	piv2
7	Human_pa	1.3E-39	3.53E-49	1.45E-50	4.39E-49	2.25E-62	1	1.31E-82	1.6E-196	6.57E-97	piv2
8	Human_rhi	1.1E-145	1.1E-134	5.9E-137	5.7E-113	5.61E-28	2.6E-118	1E-144	1	8.4E-135	rhino
9	Human_rhi	8.8E-163	3E-138	1.6E-162	1.1E-137	2.37E-25	9.5E-138	6.5E-139	1	1E-141	rhino
10	SARS_cor	1	7.3E-112	6.1E-106	4.39E-85	2.5E-122	2.8E-100	4E-119	5E-208	6.57E-96	corona
11	SARS_cor	1	2.48E-64	4.56E-76	2.28E-56	2.92E-93	3.8E-106	3.7E-102	3.4E-217	2.9E-118	corona
12	flu_A_Wisi	2.85E-33	1	4.85E-60	6E-40	1.37E-82	4.28E-37	5.27E-75	2.6E-184	4.56E-67	fluA
13	flu_A_Wisi	1.9E-49	1	2.4E-54	3.97E-38	4.25E-83	2.63E-81	3.02E-86	7.8E-196	2.6E-109	fluA
14	flu_B_Shig	2.2E-117	1.53E-88	1	2E-110	2.9E-142	7.9E-147	1.5E-155	1.3E-226	6.2E-133	fluB
15	flu_B_Shig	1.04E-97	1.07E-66	1	7.56E-86	2.9E-121	3.6E-106	7.6E-122	7.2E-208	1.8E-121	fluB
16	flu_C_Miyz	0	0	0	1	0	0	0	0	0	fluC
17	flu_C_Miyz	5.9E-306	0	0	1	0	0	0	0	0	fluC

**Figure 16: Classification of each unknown, with probabilities.**

After the conditional category probabilities are calculated for each sample in the unknowns, they are copied into a table on the "Classified Unknowns" worksheet. The highest probability determines the category the sample is assigned to, and this category is written in the last column of the table.

### 3.5.9. Save Results to File

The table of probabilities and classifications is copied from the "Classified Unknowns" worksheet and appended to a tab-delimited text file called "BAYESFILE.txt" in the program directory. This makes it somewhat more convenient to compare results from classifications using different parameters, for example.

## Chapter 4. Conclusion

Certain algorithms can be implemented quite conveniently in the spreadsheet computing paradigm. Microsoft Excel's powerful scripting functions provide a general-purpose, procedural language extension to the capabilities of the spreadsheet, and make it possible to implement certain algorithmic steps that would otherwise be beyond the reach of the basic spreadsheet.

### 4.1.1. General observations

Spreadsheet calculations have the advantages that they are explicit and exposed; each calculated value is determined by a formula associated with a cell, and these formulas can be examined in the formula toolbar.

Two of the algorithms, sequence alignment by dynamic programming and hierarchical clustering by pair grouping, were particularly well suited for a scripted spreadsheet. In each case, the major operations are done on two-dimensional arrays of numbers. Either the numbers themselves or their presentation are manipulated by scripts during algorithm execution, providing an animated view of the process. Students found these demonstrations engaging and informative.

Other implementations were perhaps less successful. The shift-AND algorithm, for instance, can be implemented entirely within the spreadsheet, with no scripting required. While this does make it possible to see how the calculations are preformed for each bit value by examining the formulas of various cells, students were generally underwhelmed with the first version of this demonstration, since nothing moves. The "animated" version is flashier, but it ends up hiding some of the formulas. I compromised by including both versions. Because the animation capabilities of Exel are limited, I did not attempt animating the more complex approximate matching version of this algorithm, which was only done in a non-scripted version. This will obviously require students to study the formulas of the cells.

The artificial neural network demonstration is interesting because it exposes so much of the inner working of the neurons. All calculations for forward flow and the major steps of backpropagation are computed with spreadsheet formulas. The only key step done by the script is to copy new weights over old weights. While I believe this demonstration may provide students with some insight into the algorithms involved, is not well suited as a general tool for experimenting with neural networks. Though much of the configuration is done with scripts, abstracting setup to a higher level than putting formulas in cells, it is still moderately difficult to reconfigure a new topology. The functions that handle network layout can only be run from the VBA debugging environment, which requires some user expertise. Perhaps more importantly, the spreadsheet is too slow to be well suited for complex training tasks. This demonstration may be most useful as an adjunct to conventional tool sets such as Tlearn.

The Naïve Bayes classifier for microarray data is the most ambitious of these demonstrations. The major challenge in this case stems from my decision to use realistically sized data sets, rather than some toy problem. Though the experimental results are simulated, they use the full complement of reporters from the virus identification microarrays of the DeRisi lab [Wang 2002]. Data sets of this size pose special problems in spreadsheets. First, calculations may be slow. This is particularly true of secondary calculations, such as taking the sum of a column of numbers, where the values in the column are themselves calculated by formulas. Second, performing multiple steps in different worksheets with a data set this large pushes

the intrinsic limits on the number of calculation dependencies allowed within a workbook. I avoided both the dependency limits and the secondary calculation problems by "fixing" cell values after each step. Thus, even though spreadsheet formulas were used for most of the calculations, many of these formulas are replaced by the hard-coded values of their results before the user has an opportunity to examine them. Nevertheless, breaking the problem into steps and exposing the intermediate results probably has a great deal of pedagogical value. In addition, the ability to handle realistically sized data sets may make this program useful for various data analysis exercises for which the results are of more interest than the algorithmic details.

Students of bioinformatics usually have a background in either biology or computer science. These demonstrations generally emphasize algorithms over applications, and were developed with computer science students in mind. Nevertheless, they may prove useful for biology students as well, because many of the calculations are done with spreadsheet formulas, and do not require extensive background in computer programming to be understood. Students of biology should be, or should become, proficient at the use of spreadsheets (but see [Zeeberg 2004] for cautionary tales, such as Excel auto-converting gene names to dates!) Even for those familiar with the algorithms, implementation in the spreadsheet paradigm may provide a fresh perspective, such as by exposing opportunities for parallelization.

#### 4.1.2. Comparisons to related work

The Java program from Professor Lecroq's site that animates sequence alignment by dynamic programming is shown in the following figure. It is not surprising that it is very similar to my demonstration, since both follow the approach commonly used to describe this algorithm on paper (see, for example [Setubal 1997]). My version has two advantages. First, because the calculations of the matrix values are done by spreadsheet formulas, the user can inspect the spreadsheet to learn how the values are computed. This may be more obvious to some users that pseudocode or verbal explanations. Secondly, my animation recursively traces paths through the matrix, and creates the corresponding alignment as the path is traversed. Dr. Lecroq's animation constructs a single graph of allowable paths and shows the resulting alignments all at once. I think my simultaneous display makes the connection between path and alignment more clear. The Java version has the perhaps considerable advantage that it runs right on a web page.

The next figure shows the "shift-OR" animation

from the Lecroq site. This demonstration is somewhat underwhelming, because it just

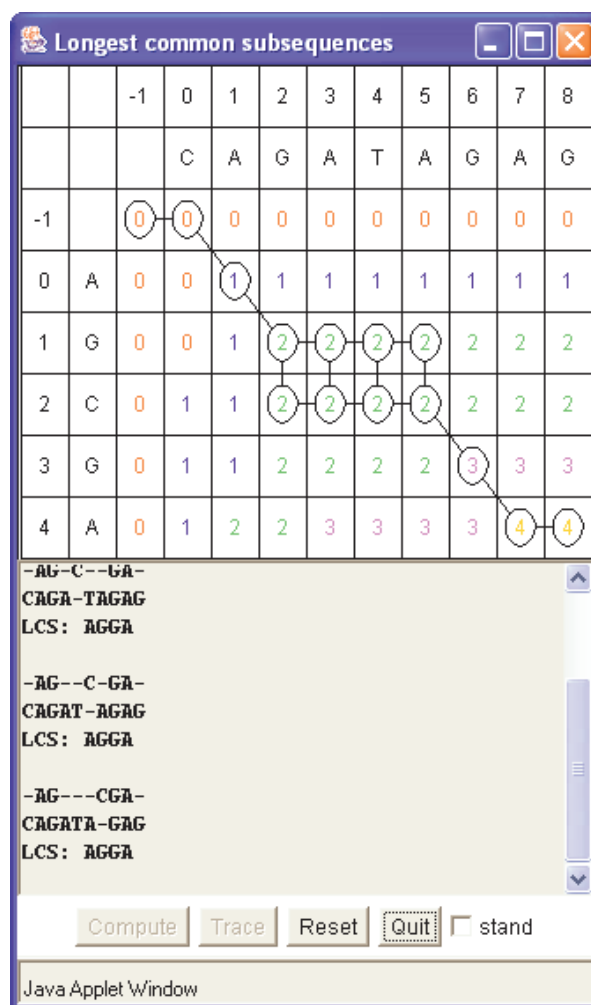
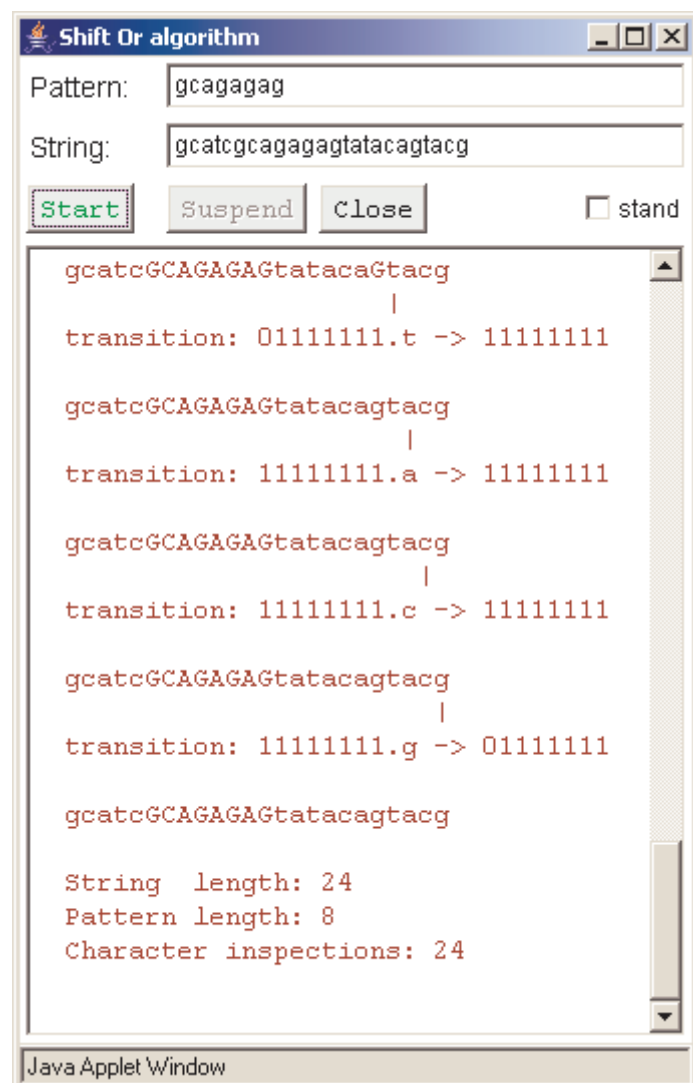


Figure 17: Dynamic programming demonstration applet

prints out a series of bit-strings, which the unfortunate user is expected to interpret as they scroll by. In this case, I much prefer my own demonstration, which presents the bit vectors in a table, so the change from one to the next is more obvious.



**Figure 18: Shift-OR applet**

Because I was unable to locate a pedagogical demonstration of UPGMA, I will compare it briefly to the FITCH distance tree program in the PHYLIP package. The PHYLIP programs work from the command line with textual menus, and can be baffling to biologists. The inputs must be properly formatted in a text file, and the output is sent to a text file. Advantages of the PHYLIP programs include their extensive documentation, many algorithmic variations, and years of use and scrutiny by scientists. They share common data formats, so outputs of many of the programs can be used as inputs to others. For example, the package has programs to draw lovely publication-quality tree diagrams in Postscript. Of course, the PHYLIP programs are not intended to demonstrate how the algorithm works, which is the main point of my program.

Similarly, the Bayes classifier might be compared to tools designed for actually classifying samples, perhaps Bioconductor R modules. The differences are that my program slowly takes the user through each painful step.

For neural networks, the obvious comparison might be between my demonstration and Tlearn. Clearly, Tlearn is better suited for experimenting with topologies and solving moderately complex Boolean functions. My demonstration mostly provides a different view; the user can examine the formulas for all the major calculations, and see all the weights update in real time.

#### 4.1.3. Future work

I will consider two aspects of this project that might be worth pursuing in the future; improvement of the existing demonstrations, and adding more spreadsheet-based bioinformatics algorithm demonstrations to the collection.

Probably what the existing demonstrations need most is to be used in teaching. This will afford an opportunity to find out what students find to be unclear or confusing. Error-

handling, interfaces (including messages to the user), and documentation all need to be tested in a teaching environment.

Several needed improvements are already obvious. The microarray classifier is slow and cumbersome; it is also unforgiving if steps are not carried out in the correct order. Students will need to be familiar with Excel, so that the automated switching between worksheets is not overly confusing. One simple "improvement" might be to merely use smaller data sets; that would greatly speed the calculations! Depending on the teaching objective, smaller data sets may be perfectly suitable. Another change that might improve the pedagogical value might be to leave the formulas in the cells at the end of each computational step. Currently, these formulas are converted to fixed numerical values at the end of each step. If they were left as formulas until the beginning of the next step, the formulas would be available for students to examine. Alternatively, with a sufficiently small data set, it would not be necessary to fix the formula values at all.

The hierarchical clustering demonstration constructs a tree description in the Newick notation used by evolutionary biologists. This description is converted to a web page containing the appropriate parameters for the Java graph applet by an external Perl script. Producing the applet web page would be simpler if the HTML were generated directly by the Excel script; the user could then click a button to create the web page.

Clearly, many features could be added to the ANN demonstration. For example, the scripts that handle network layout are currently available from the Visual Basic editor, and are only useable by VBA programmers. Exposing these layout abilities to users might make the demonstration more flexible. However, it may not be reasonable to try to make this into a "complete" ANN teaching and experimentation environment. The wiser approach might be to use this demonstration for what it is good for: it provides a view of the ANN as it learns a function so the user can see the weights update, and it exposes the calculations in spreadsheet formulas. It is probably not reasonable to expect this demonstration to learn complex functions, for example, or to do the other things that are already done well by programs like Tlearn..

The algorithms covered by the present demonstrations cover a good portion of the "key technologies" list assembled by Altman in 1998 (see table). Since then, the rise of high-throughput experimental methods, such as microarrays, would increase the importance of various machine learning approaches, but the list is still reasonable.

Probably the most important type of algorithm on the list not covered in this set of demonstrations are those using "stochastic context free grammars" [Sakakibara 1994], including Hidden Markov Models (HMMs). Since an HMM can be represented as a graph somewhat similar to finite automaton, it might be possible to lay the model out on the spreadsheet in such a way as to make the connection obvious between the model and the sets of sequences being analyzed. Bounded search algorithms might also be represented as graphs laid out on a spreadsheet, with animated traversal. Some experimentation would be required to see if such demonstrations might be worth pursuing; clearly spreadsheets are more suitable for demonstrating some algorithms than others.

The other notable item on Dr. Altman's list that is missing from the current demonstration set is a genetic algorithm. This should be of particular interest to a biological audience in that, as with artificial neural networks, the approach is biologically inspired. A demonstration program could serve as both a simulation of an evolving system and a machine learning approach. A spreadsheet implementation may provide a unique twist. Genetic algorithms



generally operate on a population of potential solutions, each represented by a vector of attribute values. One might display the members of the population in rows of a spreadsheet, with the attributes in columns. a fitness measure might be calculated by spreadsheet functions, selection might use the built-in sort functions, and recombination and mutation could be animated.

**Table 6: Altman's proposed core components of a bioinformatics curriculum**

<p style="text-align: center;"><b><i>Fundamental concepts</i></b></p> <ul style="list-style-type: none"> <li>• Pairwise sequence alignment (dynamic programming, heuristic methods, similarity matrices)</li> <li>• Multiple sequence alignment</li> <li>• Hidden Markov Models (construction, use in alignment, prediction)</li> <li>• Phylogenetic Trees</li> <li>• Fragment and map assembly and combinatorial approaches to sequencing</li> <li>• RNA Secondary structure prediction</li> <li>• Sequence feature extraction/annotation</li> <li>• Protein homology modeling</li> <li>• Protein threading</li> <li>• Protein molecular dynamics</li> <li>• Protein ab initio structure prediction</li> <li>• Integration of molecular biology databases</li> <li>• Support of laboratory biology (sequencing, structure determination, DNA arrays, etc.)</li> <li>• Design and implementation</li> </ul> <p style="text-align: center;"><b><i>Key technologies commonly used in bioinformatics</i></b></p> <ul style="list-style-type: none"> <li>• Optimization (Expectation Maximization, Monte Carlo, Simulated Annealing, gradient-based methods)</li> <li>• Dynamic programming</li> <li>• Bounded search algorithms</li> <li>• Cluster analysis</li> <li>• Classification</li> <li>• Neural Networks</li> <li>• Genetic Algorithms</li> <li>• Bayesian Inference</li> <li>• Stochastic Context Free Grammars</li> </ul> <p style="text-align: right;"><i>from [Altman 1998]</i></p>
--

## Appendix A. Experiments on use of Artificial Neural Networks to learn the genetic code.

This appendix documents preliminary experiments attempting to identify a neural network configuration to reliably learn the genetic code. The results indicate that network configuration is important to being able to learn this function using backpropagation, as some configurations clearly work better than others. The genetic code may be an interesting example system for adaptive neural network configuration algorithms.

### A.1. Software System

The neural network program Tlearn (<http://crl.ucsd.edu/innate/tlearn.html>) was used for these experiments. This is a freely available, open-source program available for Windows in binary form. This system uses three text files to specify a problem to be solved by the neural network. The "data" file contains the inputs in tab-delimited format. The "teach" file contains the expected output signals, again in tab delimited format. The "cf" (configuration) file describes the connections between nodes.

### A.2. Data

The genetic code specifies how triplets of nucleic acid sequence ("codons") are translated into the amino acids of proteins. The standard genetic code used by most living organisms (<http://molbio.info.nih.gov/molbio/gcode.html>) is given in a simple tabular format in Figure A1. The gray columns are line numbers The codons are listed in alphabetical order, with the first codon ('AAA') on line 1 and the last codon ('TTT') on line 64.

1	A	A	A	K	17	C	A	A	Q	33	G	A	A	E	49	T	A	A	.
2	A	A	C	N	18	C	A	C	H	34	G	A	C	D	50	T	A	C	Y
3	A	A	G	K	19	C	A	G	Q	35	G	A	G	E	51	T	A	G	.
4	A	A	T	N	20	C	A	T	H	36	G	A	T	D	52	T	A	T	Y
5	A	C	A	T	21	C	C	A	P	37	G	C	A	A	53	T	C	A	S
6	A	C	C	T	22	C	C	C	P	38	G	C	C	A	54	T	C	C	S
7	A	C	G	T	23	C	C	G	P	39	G	C	G	A	55	T	C	G	S
8	A	C	T	T	24	C	C	T	P	40	G	C	T	A	56	T	C	T	S
9	A	G	A	R	25	C	G	A	R	41	G	G	A	G	57	T	G	A	.
10	A	G	C	S	26	C	G	C	R	42	G	G	C	G	58	T	G	C	C
11	A	G	G	R	27	C	G	G	R	43	G	G	G	G	59	T	G	G	W
12	A	G	T	S	28	C	G	T	R	44	G	G	T	G	60	T	G	T	C
13	A	T	A	I	29	C	T	A	L	45	G	T	A	U	61	T	T	A	L
14	A	T	C	I	30	C	T	C	L	46	G	T	C	U	62	T	T	C	F
15	A	T	G	M	31	C	T	G	L	47	G	T	G	U	63	T	T	G	L
16	A	T	T	I	32	C	T	T	L	48	G	T	T	U	64	T	T	T	F

Figure 19: The standard genetic code

The standard genetic code was formatted into a truth table using the Perl script shown in Figure A2. It uses a two-bit binary number to represent each of the four bases (A=00, C=01, G=10, T=11), so that a codon can be represented in 6 bits. The 64 possible codons must be translated into a 21 character alphabet, representing the 20 amino acids plus "termination" (represented by a period). A termination codon signals the end of a protein reading frame. Characters in the protein alphabet are represented in the output of the script as a 21-place bit vector, containing a single '1' to indicate one of the 21 characters of a translated sequence.

The script generates a truth table with 6 columns of input and 21 columns of output.

```

C:\bob\CSUS\Masters\ANN\codon_data.pl
1  #!perl -w
2  # codon_data.pl: converts the genetic code to binary format for
3  #               use in machine learning experiments.
4  use strict;
5
6  while (<DATA>){
7      chomp;
8      my ($b1,$b2,$b3,$aa) = split(/\t/);
9      my $binStr = '';
10     foreach my $base ($b1,$b2,$b3){
11         $binStr .= base2binary($base);
12     }
13     $binStr .= letter2binary($aa,"ACDEFGHIKLMNPQRSTUWV.");
14     print join("\t",split('',$binStr)),"\n";
15 }
16
17 sub letter2binary {
18     my ($letter,$alphabet)=@_;
19     my $result = '0' x length $alphabet;
20     substr $result,index($alphabet, $letter),1,'1';
21     return $result;
22 }
23
24 sub base2binary{          # base letter ACGT to 2-bits
25     my ($base) = @_;
26     my %binary = (
27         A => '00',
28         C => '01',
29         G => '10',
30         T => '11'
31     );
32     return $binary{$base};
33 }

```

Figure 20: Perl script to format genetic code for machine learning experiments

**Table 7: Standard genetic code represented as a truth table**

[illegible]

This binary truth table can be simplified using standard logic minimization techniques. The program UC Berkeley Espresso (obtained from [www.dei.isep.ipp.pt/~acc/bfunc/](http://www.dei.isep.ipp.pt/~acc/bfunc/)) reduced the table to 26 rows. I then rearranged the rows so the positive bits in the outputs are ordered sequentially, and used the alphabet to mark the output columns. Though the rows are no longer in order by codon, this table shows more clearly how the bitmaps represent the 21 characters of the protein alphabet.

**Table 8: Simplified truth table for standard genetic code**

	ACDEFGHIKLMNPQRSTVWY .
1001--	1000000000000000000000
1110-1	0100000000000000000000
1000-1	0010000000000000000000
1000-0	0001000000000000000000
1111-1	0000100000000000000000
1010--	0000010000000000000000
0100-1	0000001000000000000000
00110-	0000000100000000000000
0011-1	0000000100000000000000
0000-0	0000000010000000000000
-111-0	0000000001000000000000
0111--	0000000001000000000000
001110	0000000000100000000000
0000-1	0000000000010000000000
0101--	0000000000001000000000
0100-0	0000000000000100000000
0-10-0	0000000000000001000000
0110--	0000000000000000100000
1101--	0000000000000000010000
0010-1	00000000000000000010000
0001--	000000000000000000010000
1011--	000000000000000000001000
111010	000000000000000000000100
1100-1	000000000000000000000010
11-000	0000000000000000000000001
1100-0	0000000000000000000000001

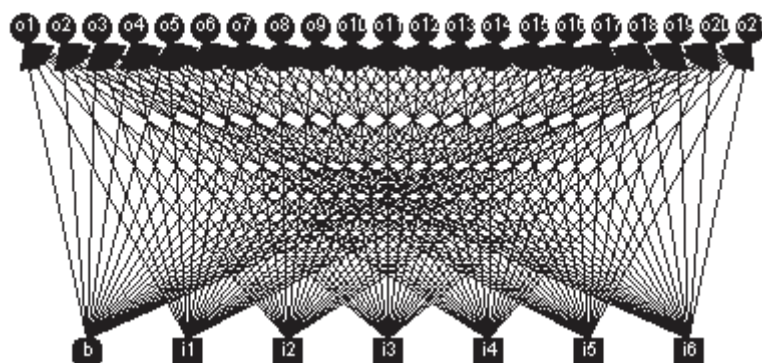
The minimized table shows some patterns among the codons. Most notable is the characteristic "wobble", where the third position in the codon has less significance for amino acid selection than the first two positions. For example, the amino acid Alanine (A) is encoded by 1001--, which represents four codons sharing GC in the first two positions, GC[ACGT]. Cystine (C) is represented by 1110-1, or the two codons TGC and TGT. In particular, note that Serine (S) is encoded by the conjunction of 1101-- and 0010-1, which are the six codons TCA, TCC, TCG, TCT, AGC, and AGT.

### A.3. Testing Topologies

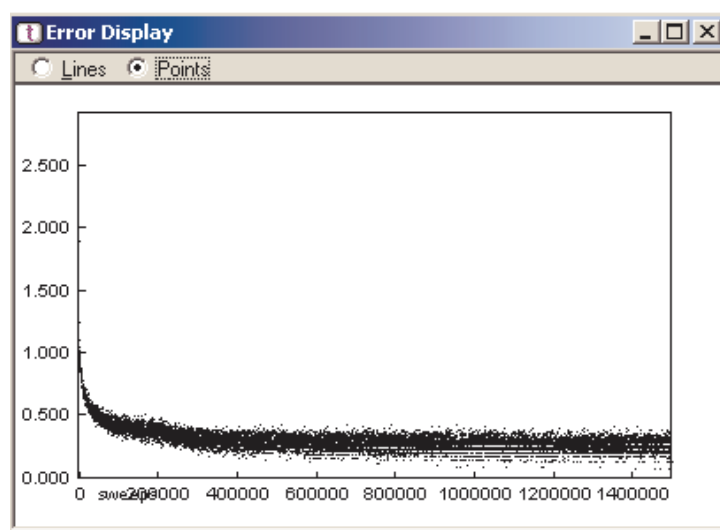
I chose to use the full (un-minimized) table for training neural networks. I will first show the results of the simplest topology: 6 input nodes and 21 output nodes, with no hidden nodes, and each of the 26 output nodes receiving input from all 6 input nodes. This topology is

diagrammed below. The 'b' node at the beginning of the bottom (input) row provides a constant bias input.

**Figure 21: Simple network topology with no hidden nodes**



The results of a typical training run are shown in the next figure, with the abscissa giving the number of "sweeps" through the training data, and the ordinate giving the error from comparing network output to the expected values from the training data. Note that the error never approaches zero, but bounces around in the area of about 0.1 to 0.4. This figure uses individual points to represent every 1000<sup>th</sup> sweep, and reveals that some error levels seem to recur frequently, reminiscent of bifurcation diagrams. Training was repeated many times, using a different random number seed to set the initial network weights. In no case did a network with this topology successfully learn the complete genetic code table.



**Figure 22: Simple topology fails to learn genetic code completely**

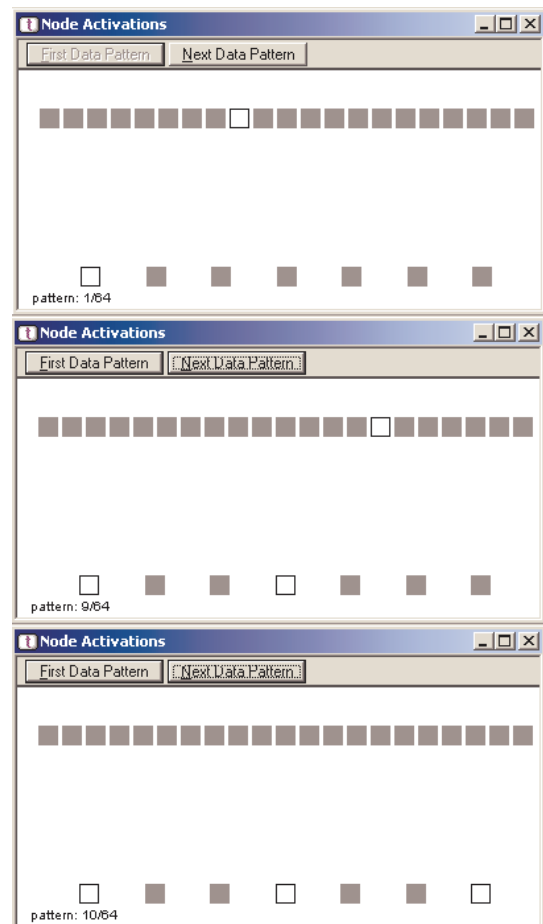
The Tlearn program has a Node Activation window that shows the output of a trained network with given inputs. The 21 output nodes are shown in order from right to left at the top of the Node Activation window, and the seven inputs (the bias, plus the six input bits) are shown at the bottom. Note that the bias input is always on.

Examples of node activation diagrams show that the simple network learns to identify many amino acids, but not all of them. In the top node activation window, the first codon in the training set, AAA, is represented by 000000. This causes the 9<sup>th</sup> output to light up; the 9<sup>th</sup> amino acid is K (Lysine), which is indeed encoded by AAA. The next illustrated example shows that an input of 001000 (representing the 9<sup>th</sup> codon, AGA) produces output number 15 (R, for Arginine), which is also correct. In our third example, however, the network fails to produce any high-valued output (input 001001 = AGC should be identified as S, Serine, the 16<sup>th</sup> letter in our amino acid alphabet. This network produced a single, correct output for each set of input values except codons 10 and 12 (not shown). Both of these codons should encode S. In ten independent attempts at training this network, eight gave a similar result, in which codons 10 and 12 gave no output. One run gave no output for codons 10, or 12, or for 62, and 64. Codons 62 and 64 represent the 5<sup>th</sup> amino acid, F (Phenylalanine). One of these ten trials did produce a network that correctly identified codons 10 and 12 as S, but failed to identify codons 53, 54, 55, and 56, which also represent S. Since a network's topology can affect its ability to be trained with certain datasets by backpropagation, I set about to find a better topology.

Adding a single layer of hidden nodes between the inputs and the outputs (to make a 3-layer network) did not lead to networks able to learn the genetic code any better than the simple, single layer topology. I tried various numbers of hidden nodes, all connected to all inputs and all outputs. One of the 3-layer networks contained 26 hidden nodes. I predicted that this network could learn the code, because there are 26 terms in the simplified truth table, each of which can be represented by a simple Boolean equation). But even this never completely learned the code in many long attempts.

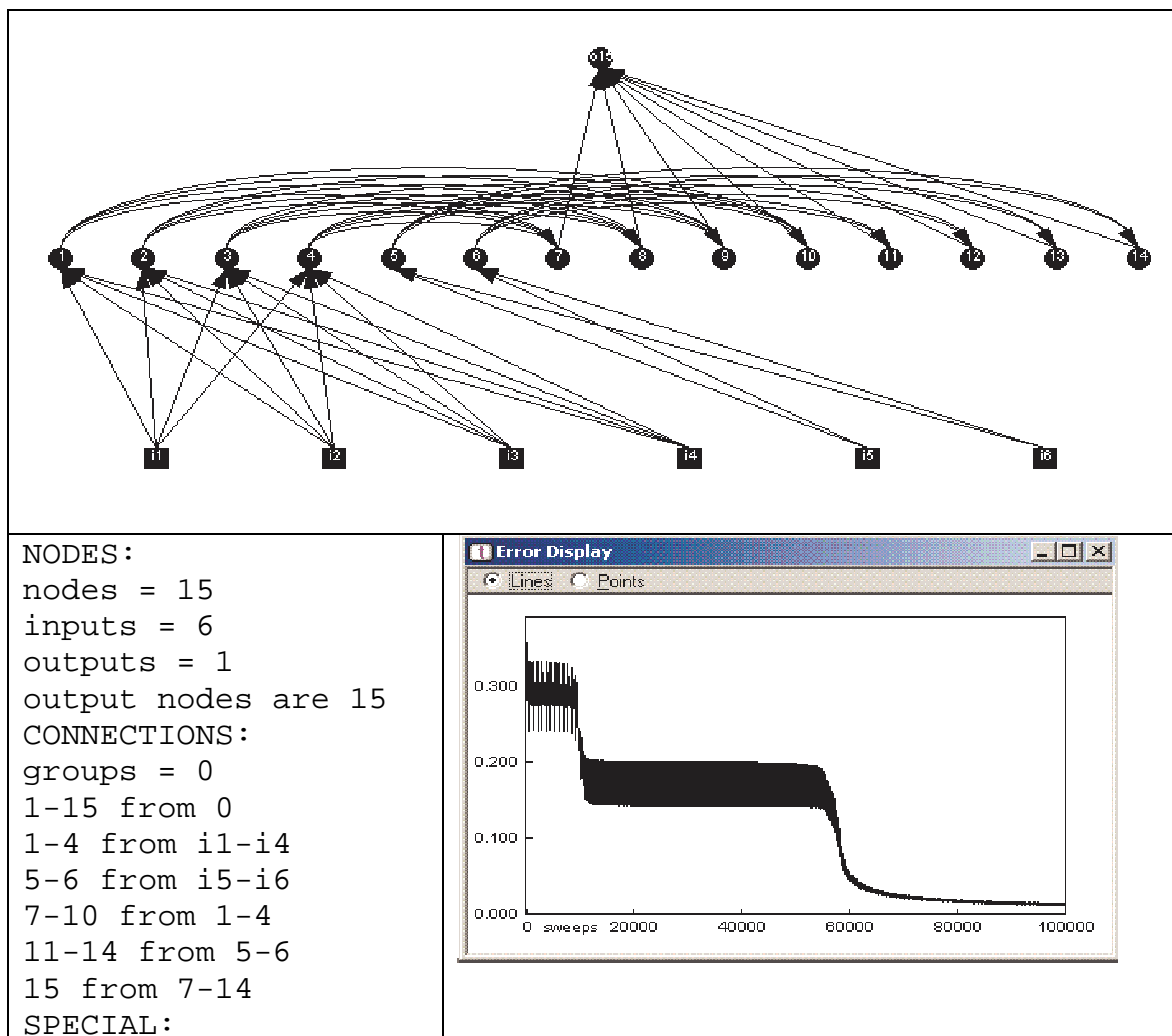
In most cases, the 3-layer networks got "stuck" at a level where all codons were recognized except codons 10 and 12 (for Serine). Thus, they suffered the same problem as the simple 2-layer network. Since most topologies had trouble with Serine, I decided to try to learn its code separately.

As mentioned above, the code for Serine is complicated because this amino acid is represented by 6 codons. Four of them start with TC (the third base doesn't matter), while two start with AG (codons 10 and 12 are AGC and AGT.) I reasoned that the network was getting confused trying to learn two classes of codons (TCN and AG[CT]) for the same output, so I experimented with a topologies in which some hidden nodes connect to the first



**Figure 23: Testing network output with specific inputs**

four inputs (for the first two bases), and some hidden nodes connect to only the last two inputs (for the base in the "wobble" position).



**Figure 24: A topology for learning Serine**

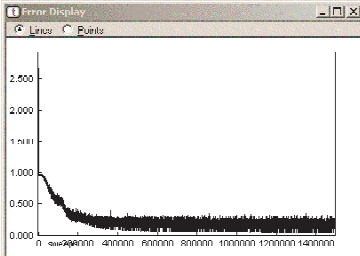
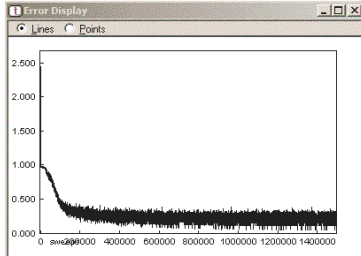
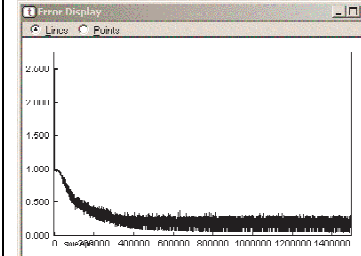
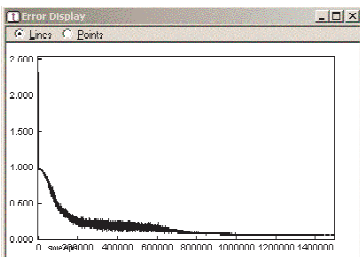
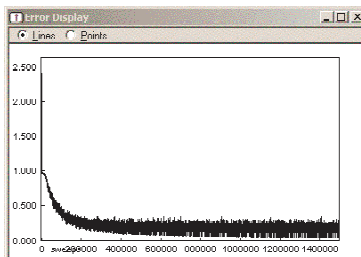
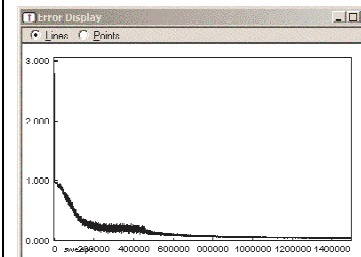
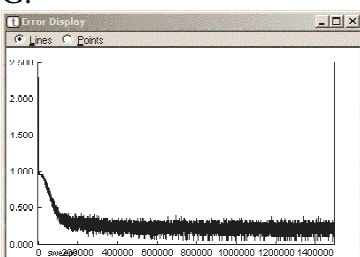
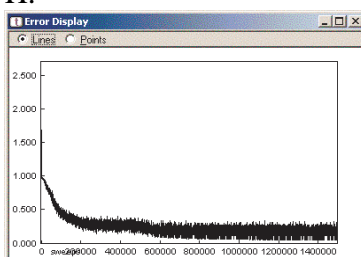
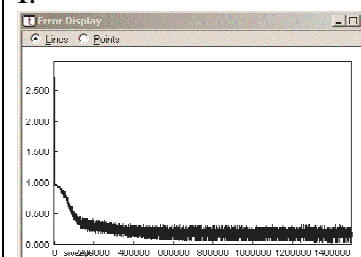
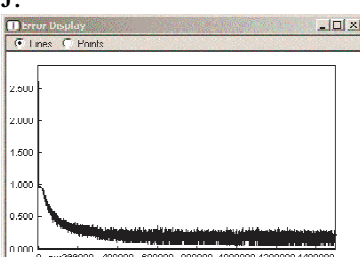
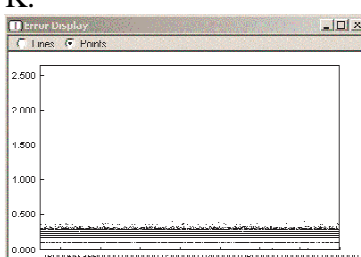
The most reliable Serine-learning network I was able to identify has two hidden layers (4 layers total). I will describe the nodes using Tlearn's conventions; input nodes are called i1 through i6; the first layer of hidden nodes are 1 through 6, the second hidden layer has seven nodes numbered 7 through 14, and node 15 is the output. The first four hidden nodes take input from all of the first four inputs together, and give output to all of the first four nodes in the next layer. Nodes 5 and 6 in the first hidden layer both take input from both inputs 5 and 6, and send output to the set of nodes 11-14 on the second hidden layer. All the nodes in the second hidden layer send results to the output node. The Tlearn Network Architecture diagram and configuration file are shown below.

This topology was very successful in learning the Serine values from the truth table. In 40 trials (each with a maximum of 100000 sweeps, learning rate = 1, momentum = 0.2), it learned the Serine output correctly each time.

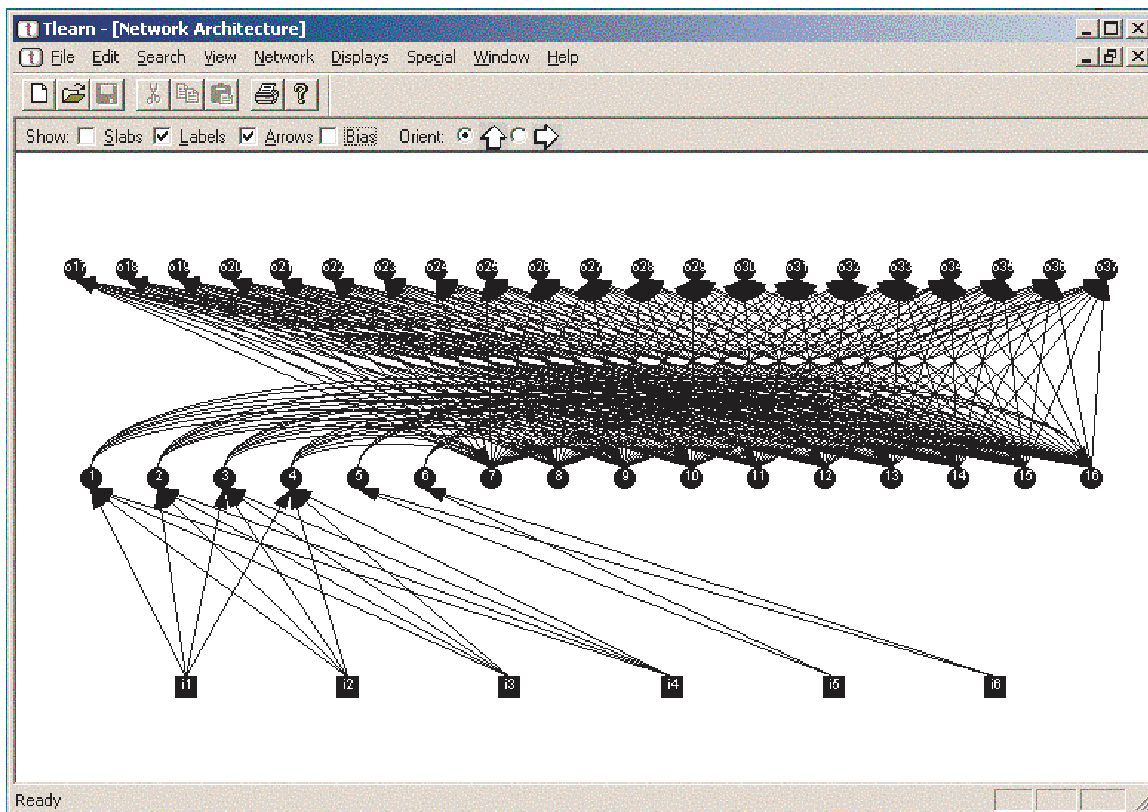
In theory, a single hidden layer should be sufficient to represent any function (I think this is why Tlearn diagrams the hidden nodes all on a single level). However, I was never able to devise a topology with a single hidden layer that could learn the Serine values.



Table 9: Four-layer topology can learn the genetic code, but doesn't always

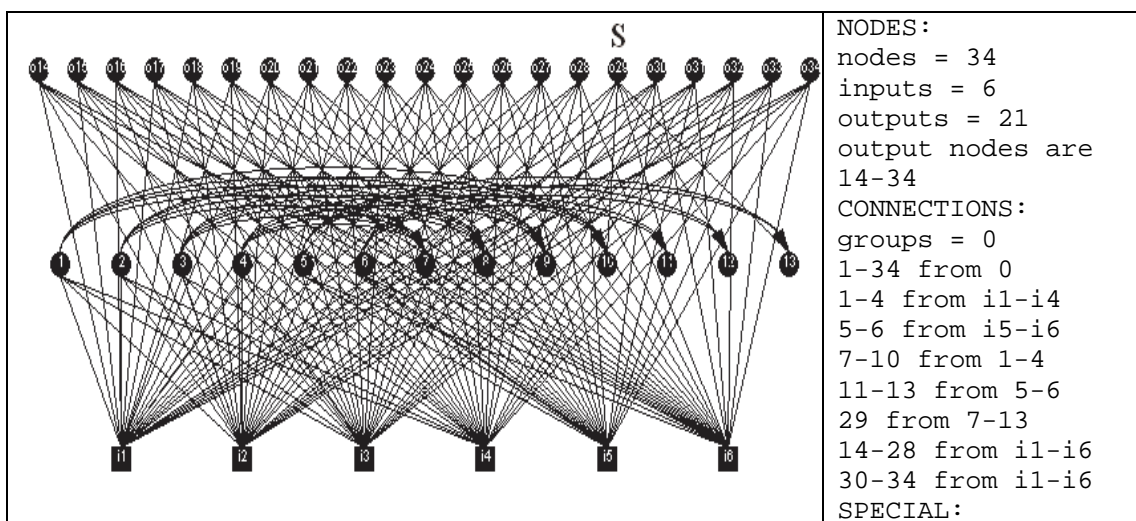
<p>A.</p> 	<p>B.</p> 	<p>C.</p> 
<p>D.</p> 	<p>E.</p> 	<p>F.</p> 
<p>G.</p> 	<p>H.</p> 	<p>I.</p> 
<p>J.</p>  <p>Network: training Options learning rate: 0.1 momentum: 0 Train in random sequence.</p>	<p>K.</p>  <p>Some of the stuck runs go on and on...</p>	<p>L. Tlearn configuration file.</p> <pre> NODES: nodes = 37 inputs = 6 outputs = 21 output nodes are 17-37 CONNECTIONS: groups = 0 1-16 from 0 1-4 from i1-i4 5-6 from i5-i6 7-12 from 1-4 13-16 from 5-6 17-37 from 7-16 SPECIAL: </pre>

Testing this topology on other amino acids showed that it could learn them also, so it is not restricted to learning Serine. I then added more outputs to test whether similar topologies could learn the entire genetic code at once. This is very similar to the four layer network described above for learning Serine, except it has a few more nodes added to each of the two groups in the second hidden layer, and the second hidden layer connects to all of the 21 outputs. Of ten trials using this topology, the network learned the genetic code in two (D and F in the table).



**Figure 25: Four-layer topology capable of learning the genetic code, sometimes.**

In a last-ditch attempt at finding a topology that can learn the genetic code reliably, I reasoned that if a single layer can learn all amino acids but Serine, and if the four-layer topology described above can learn Serine alone, then I should be able to construct a hybrid topology that treats Serine as a special case. This is shown below, along with its Tlearn configuration.



**Figure 26: The most effective topology for learning the genetic code treats Serine as a special case.**

The topology treating Serine as a special case was very effective for learning the genetic code. In 20 of 20 trials, it succeeded in learning the entire code in < 200000 sweeps (learning rate = 1, momentum=0.5).

#### ***A.4. Implications of Experiments on Topology***

The genetic code is a toy problem, in that it can be completely described in a look-up table, so neural network pattern finding does not reveal anything new. It is a suitable example problem as an application of artificial neural networks in bioinformatics only in that it may help students become familiar with the concept that the genetic code is in fact a lookup table (the term is commonly misused to mean "genome", more or less, in the popular press.)

These experiments demonstrate that choice of a suitable neural network topology is extremely important in learning certain Boolean functions, such as the genetic code. One topology (treating serine as a special case) was found which is quite reliable for learning the genetic code. This topology is not very satisfying in a general sense, however, because it is highly specialized for this problem. A general machine learning solution, either for finding appropriate topologies, or for better training of standard topologies, would be more interesting. This function could serve as a good example system for experimenting with approaches such as using genetic algorithms for neural network topology identification; this is called "neuroevolution"<sup>2</sup>.

Because of the complexity and specialized nature of the only topology I found that could learn the genetic code reliably, I chose not to attempt this problem in the Excel demonstration. The spreadsheet would be rather complicated, and conducting the required number of training steps with so many nodes seemed impractical. Instead, the demonstration uses the classic XOR function..

---

<sup>2</sup> See also the Wikipedia article on "Neuroevolution", <http://en.wikipedia.org/wiki/Neuroevolution>

## Glossary

**3' end:** the end of a DNA sequence where the free 3' hydroxy group is found.

**5' end:** the end of a DNA chain where the free 5' phosphate group is normally located. DNA sequences are normally written with the 5' end on the left.

**A:** the abbreviation for adenine in a nucleic acid sequence.

**adenine:** a purine base found in DNA and RNA. Pairs with T in normal Watson-Crick pairing.

**amino acid:** one of the basic building blocks of proteins. There are twenty "normal" amino acids commonly found in proteins. This means that the sequences of most proteins can be described using a twenty-character alphabet.

**amino terminus:** the beginning of a protein chain, where the free amino group is located. Proteins are synthesized in the amino to carboxy terminus direction by ribozymes. The amino terminus of a protein is encoded by sequences toward the 5' end of the gene for that protein. By convention, protein sequences are written with the amino terminus on the left.

**antiparallel:** strands running alongside one another but having opposite orientations.

**artificial neural network:** a set of artificial neurons connected with a certain topology.

**base pairing:** the hybridization of complementary nucleotides. A normally pairs with T, and C pairs with G.

**Bayes' theorem:**  $P(A|B) = P(B|A) * P(A) / P(B)$

**BLOSUM:** "Blocks Substitution Matrix", one of the classic types of scoring tables for amino acid comparisons.

**C:** see cytosine.

**carboxy terminus:** the end of a protein chain that has a free carboxy group. See amino terminus.

**central dogma:** the concept that genetic information flows from long-term storage in DNA, to short term storage in mRNA, to proteins capable of carrying out biological function. There are known exceptions to this pattern, such as the reverse transcriptases of certain viruses (retroviruses) that copy information from RNA to DNA. Also, some organisms, notably viruses, keep long term genetic information in RNA molecules rather than DNA.

**chloroplast:** the subcellular organelle of plants wherein photosynthesis occurs.

**chromosome:** organized structures containing the major portion of an organism's DNA. Eukaryotic chromosomes are located in the nucleus.

**classification:** the process of assigning instances to categories.

**codon:** a triplet of three consecutive nucleotide bases encoding a particular amino acid.

**complementary:** having have G match with C, and A match with T.

**C-terminus:** see carboxy terminus.

**cytosine:** a pyrimidine base found in DNA and RNA. Pairs with G in normal Watson-Crick pairing.

**denaturation:** any process that causes the normal three dimensional structure of a protein to be disrupted. Denaturation can have drastic effects on the properties of proteins. Cooking egg whites, for example, causes the albumin proteins to denature, and to lose their solubility in water, and denatured enzymes lose their catalytic activities.

**deoxyribonucleotide:** any of the monomers from which DNA polymers are constructed. Deoxyribonucleotides consist of a nucleotide base (A, C, G, or T) connected to a deoxyribose sugar molecule, which has a phosphate group that can form a phosphodiester bond with the previous monomer in the chain.

**DNA polymerase:** any enzyme that produces DNA could be called a DNA polymerase. There are many types of DNA-producing enzymes. For example, an enzyme involved in DNA replication might be called a DNA-directed DNA polymerase. A reverse transcriptase could be called an RNA directed DNA polymerase. There are also enzymes that can produce DNA molecules without a template.

**DNA:** deoxyribonucleic acid. In most organisms, this is the genetic material.

**edit distance:** a similarity metric for comparing two sequences which is scored by the number of edits (insertions, deletions, and substitutions) required to convert one sequence into the other. Each editing operation may have its own score, which may be contained in a scoring table.

**enzyme:** a biological molecule capable of acting as a catalyst for a particular biochemical reaction. Most enzymes are proteins, but some include RNA, and some are purely RNA ("ribozymes").

**eukaryotic:** a type of organism having nucleate cells. Organisms without cell nuclei, such as bacteria, are called prokaryotic.

**extrachromosomal:** being located outside of a chromosome. Mitochondrial genes are said to be extrachromosomal, for instance.

**G:** see guanine.

**gene:** a region of genetic material that carries the information for a specific trait. This is a broad definition, since "trait" could mean many things. Simple examples of usually encode a particular protein, including promoter regions, exons, etc. However, some genes encode RNA molecules that are not translated to proteins (such as tRNAs), some act as regulatory regions, and some may have no known biological effect, as with many of the "genes" for some markers used to distinguish individuals in forensic investigations, for example.

**genetic code:** a mapping of codons to amino acids. A standard genetic code is used by most organisms.

**genome:** the content of all of an organism's chromosomes.

**guanine:** a purine base found in DNA and RNA. Normally pairs with C in double-stranded nucleic acids.

**Hierarchical Clustering:** grouping of instances into a tree-like structure showing how closely they are related to one another. Clustering is commonly done using unsupervised machine learning methods.

**hybridization:** annealing of complementary strands, especially strands with different origins (such as a target and a probe).

**initiation codon:** the first triplet in a reading frame.

**MAP hypothesis:** see "maximum a posteriori hypothesis"

**maximum a posteriori hypothesis:** the hypothesis with the highest posterior probability.

**mitochondria:** a subcellular organelle found in most nucleated cells. Mitochondria are centrally important in the process of respiration.

**naïve Bayes classifier:** a classifier based on Bayes' theorem in which the simplifying assumption is made that the probabilities of all attribute values are independent of one another.

**N-terminus:** see amino terminus.

**Operational Taxonomic Unit:** (OTU) a node in an evolutionary tree.

**PAM:** "Percent Acceptable Mutations", one of the classic types of scoring tables for amino acid comparisons.

**pathway:** a series of biochemical reactions leading from one or more starting materials to metabolic products. The steps in a pathway are usually catalyzed by enzymes.

**PCR:** see "polymerase chain reaction".

**peptide:** a short protein chain. Peptides typically do not have enzymatic activity, since they are too small to form the molecular machines we call enzymes. Many peptides have biological activities as neurotransmitters or immunological targets. The name refers to the type of amino-ester bonds used to join amino acids together.

**polymerase chain reaction:** (PCR) a process by which targeted regions of DNA molecules can be reproduced under experimental conditions. It is a recursive application or primer extension.

**polypeptide:** a chain of amino acids joined by peptide bonds (a protein chain).

**posterior probability:** the probability of a hypothesis, given the observed data.

**primary structure:** the sequence of a protein chain. This can be thought of as a one-dimensional description of the protein.

**primer:** a single-stranded nucleic acid molecule (usually a synthetic oligonucleotide) that hybridizes to a template molecule and serves to initiate template-directed synthesis by DNA polymerase.

**prior probability:** the assumed probability of a hypothesis before observations are made.

**promoter:** a region of a gene directing RNA polymerase activity at a particular transcription start site.

**protein:** a polymer of amino acids linked by peptide bonds. Many proteins have biological functions that depend on their three-dimensional structures. A principle determinant of how one or more protein chains of amino acids folds into a three dimensional structure is the sequence of amino acids in the chains. The process of protein folding is an important step in converting one-dimensional sequence information (as stored in DNA) into biological activities.

**quaternary structure:** the association of multiple protein chains into a single three-dimensional structure.

**reading frame:** The modulus or register in which a nucleic acid sequence is or could be translated into protein. A strand of DNA has three possible reading frames, and its complementary strand has three more.

**replication:** the process of reproducing molecules or simple organisms (viruses are said to replicate, for example).

**reverse complement:** the sequence of the strand that would pair exactly with a given strand. Matching strands must be both complementary and antiparallel.

**ribosome:** a macromolecular assembly responsible for translation of RNA sequences to proteins.

**RNA polymerase:** an enzyme that catalyzes the production of polymers of ribonucleic acid.

**RNA:** ribonucleic acid. Similar in structure to DNA, RNA is typically less stable. RNA performs many vital functions in living things. Some types of RNA molecules (mRNA) act as messengers in the process of protein production. Others (tRNA) couple to activated amino acids to help match them to their codons. Ribosomes contain large amounts of structural rRNA. Other RNA molecules are intimately involved in other enzymatic functions, such as splicing.

**secondary structure:** local three-dimensional structure within a polymer chain. Common secondary structures in proteins include the alpha helix, the beta pleated sheet, and "turns" or "coils". Prediction of secondary structure from a protein sequence is easier than predicting its complete three-dimensional structure, and is widely considered to be a first step in structure prediction.

**sequence:** the order of monomers within a polymer, or of characters within a string. Genetic information is contained in the sequence of bases in DNA.

**squashing function:** a differentiable function that maps the output of an artificial neuron into a finite range.

**stochastic gradient descent:** an approach to training a neural network wherein errors are back-propagated for each individual example, rather than for the entire training set at once.

**supervised learning:** the process by which a program induces rules for classifying instances, based on pre-classified examples given in a training set.

**synthetic oligonucleotide:** a short strand of DNA created using techniques of organic chemistry.

**T:** see thymine.

**tertiary structure:** the three dimensional arrangement of a single protein chain.

**thymine:** a pyrimidine base found in DNA. (The thymine of DNA sequences are transcribed into uracils in RNA). Normally pairs with A.

**training set:** a group of pre-classified examples that can be used with machine learning approaches.

**transcription factor:** a protein that interacts with a gene sequence to control RNA polymerase activity at that gene.

***transcription***: the process of copying information from DNA to RNA. This is done by DNA-dependant RNA polymerase.

***translation***: the process of creating a protein molecule based on the sequence information contained in a molecule of messenger RNA.

***uracil*** : the pyrimidine base that takes the place of thymine in RNA.

***virus***: an acellular infectious particle that replicates within cells of a host organism.



## Bibliography

- Alizadeh, A.A., Eisen, M.B., Davis, R.E., Ma, C., Lossos, I.S., Rosenwald, A., Boldrick, J.C., Sabet, H., Tran, T., Yu, X., Powell, J.I., Yang, L., Marti, G.E., Moore, T., Hudson, J., Lu, L., Lewis, D.B., Tibshirani, R., Sherlock, G., Chan, W.C., Greiner, T.C., Weisenburger, D.D., Armitage, J.O., Warnke, R., Staudt, L.M. (2000). Distinct Types of Diffuse Large B-Cell Lymphoma Identified by Gene Expression Profiling. *Nature* 403, 503-11. [http://rana.lbl.gov/papers/Alizadeh\\_Nature\\_2000.pdf](http://rana.lbl.gov/papers/Alizadeh_Nature_2000.pdf)
- Altman, R. (1998) A curriculum for bioinformatics: the time is ripe (editorial). *Bioinformatics* 14(7):549-550. [http://www-smi.stanford.edu/pubs/SMI\\_Reports/SMI-98-0744.pdf](http://www-smi.stanford.edu/pubs/SMI_Reports/SMI-98-0744.pdf)
- Delamarche, C. (2000) Color and Graphic Display (CGD): Programs for Multiple Sequence Alignment Analysis in Spreadsheet Software. *BioTechniques* 29:100-107.
- Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998) *Biological Sequence Analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.
- Eisen, M.B., Spellman, P.T., Brown, P.O. and Botstein, D. (1998). Cluster Analysis and Display of Genome-Wide Expression Patterns. *Proc Natl Acad Sci U S A* 95, 14863-8.
- Felsenstein, J. (1989) PHYLIP - Phylogeny Inference Package (Version 3.2). *Cladistics* 5: 164-166.
- Felsenstein, J. (2004) PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.
- Gentleman, R.C., Carey, V.J., Bates, D.M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S., Irizarry, R., Leisch, F., Li, C., Maechler, M., Rossini, A.J., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J.Y., and Zhang, J. (2004) Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol.* ;5(10):R80. <http://genomebiology.com/2004/5/10/R80>
- Graham, P. (2002) A Plan for Spam. Retrieved December 4<sup>th</sup>, 2004 from <http://www.paulgraham.com/spam.html>
- Gusfield, D. (1997) *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.
- Han, B., and Tashjian, A. H. (1998) User-Friendly and Versatile Software for Analysis of Protein Hydrophobicity. *BioTechniques* 25:256-263.
- Higgins D., Thompson J., Gibson T. Thompson J.D., Higgins D.G., Gibson T.J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22:4673-4680. <http://www.pubmedcentral.nih.gov/picrender.fcgi?tool=EBI&pubmedid=7984417&action=stream&blobtype=pdf>
- McEwan, N. R., and Gatherer, D. (1998) Adaptation of Standard Spreadsheet Software for the Analysis of DNA Sequences. *BioTechniques* 24:131-138.
- Mitchell, T.M. (1997) *Machine Learning*. WCB/McGraw-Hill.

- Monroe, W.T., and Haselton, F.R. (2003) Molecular Beacon Sequence Design Algorithm. *BioTechniques* 34:68-73.
- Needleman, S.B. and Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48:443-453.
- O'Neill, M.C., and Song, L. (2003) Neural network analysis of lymphoma microarray data: prognosis and diagnosis near-perfect. *BMC Bioinformatics* 4:13.  
<http://www.biomedcentral.com/1471-2105/4/13>
- Sakakibara, Y., Brown, M., Hughey, R., Mian, I.S., Sjolander, K., Underwood, R.C., Haussler, D. (1994) Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*. 22(23):5112-20.  
<http://www.pubmedcentral.nih.gov/articlerender.fcgi?tool=pubmed&pubmedid=7800507>
- Schageman, J.J., Basit, M., Gallardo, T.D., Garner, H.R., and Shohet, R.V. (2002) MarC-V: a spreadsheet-based tool for analysis, normalization, and visualization of single cDNA microarray experiments. *BioTechniques* 32:338-344
- Schütz, E., and von Ahsen, N. (1999) Spreadsheet Software for Thermodynamic Melting Point Prediction of Oligonucleotide Hybridization with and without Mismatches. *BioTechniques* 27:1218-1224.
- Setubal, J. and Meidanis, J. (1997) *Introduction to Computational Molecular Biology*. PWS Publishing Company, Boston.
- Shaw, G. (1997) "Spreadsheets in Molecular Biology". Chapter 7 of "Spreadsheets in Science and Engineering", pages 203-228, Editor Gordon Filby, Springer-Verlag, Heidelberg, Germany.
- Smith, T.F. and Waterman, M.S. (1981) Identification of common molecular subsequences. *Journal of Molecular Biology* 147:195-197.
- Snow C.D., Nguyen, H., Pande, V.S., and Gruebele, M. (2002) Absolute comparison of simulated and experimental protein-folding dynamics. *Nature* 420(6911):102-10.  
<http://vsp27.stanford.edu/papers/SnowNature2002.pdf>
- Stowe, R. P., and Pierson, D. L. (1996) Spreadsheet Macro for Setting Up PCR Assay Tubes. *BioTechniques* 20:1088-1089.
- Tobler, J.B., Molla, M.N., Nuwaysir, E.F., Green, R.D., Shavlik, J.W. (2002) Evaluating machine learning approaches for aiding probe selection for gene-expression arrays. *Bioinformatics*. 2002;18 Suppl 1:S164-71.  
[http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=pubmed&dopt=Abstract&list\\_uids=12169544](http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=pubmed&dopt=Abstract&list_uids=12169544)
- von Neumann, J. (1958) *The Computer and the Brain*, Yale University Press, New Haven.
- Wang, D., Coscoy, L., Zylberberg, M., Avila, P.C., Boushey, H.A., Ganem, D., and DeRisi, J.L. (2002) Microarray-based detection and genotyping of viral pathogens. *Proceedings of the National Academy of Sciences USA*, 99(24):15687-92.  
<http://derisilab.ucsf.edu/publications/pdfs/VirusChip.pdf>
- Wang, D., Urisman, A., Liu, Y.-T., Springer, M., Ksiazek, T.G., Erdman, D.D., Mardis, E.R., Hickenbotham, M., Magrini, V., Eldred, J., Latreille, J.P., Wilson, R.K., Ganem, D.,

- and DeRisi, J.L. (2003) Viral discovery and sequence recovery using DNA microarrays. *PLoS Biology*, 1(2):E2 [http://www.plosbiology.org/archive/1545-7885/1/2/pdf/10.1371\\_journal.pbio.0000002-L.pdf](http://www.plosbiology.org/archive/1545-7885/1/2/pdf/10.1371_journal.pbio.0000002-L.pdf)
- Weir, B.S. (1996) *Genetic Data Analysis II: Methods for discrete population genetic data*, Sinauer Associates, Sunderland MA
- Wu, S., and Manber, U. (1991) Fast text searching with errors. University of Arizona Technical Report TR 91-11. <ftp.cs.arizona.edu/agrep/agrep.ps>.1
- Zeeberg, B.R., Riss, J., Kane, D.W., Bussey, K.J., Uchio, E., Linehan, W.M., Barrett, J.C., and Weinstein, J.N. (2004) Mistaken Identifiers: Gene name errors can be introduced inadvertently when using Excel in bioinformatics. *BioMed BMC Bioinformatics* 5:80 [www.biomedcentral.com/1471-2105/5/80](http://www.biomedcentral.com/1471-2105/5/80)

## Index

adenine, 2, 46  
amino acid, 1, 2, 3, 9, 11, 36, 39, 41, 43, 44, 46  
amino terminus, 2, 46  
antiparallel, 2, 46  
artificial neural network, 12, 19, 45, 46  
base pairing, 2, 46  
Bayes' theorem, 13, 22, 46  
BLOSUM, 9, 46  
carboxy terminus, 2, 46  
central dogma, 1, 46  
chloroplast, 46  
chromosome, 46  
classification, 5, 11, 13, 14, 22, 23, 27, 46  
clustering, 4, 11, 17, 47  
codon, 3, 36, 39, 41, 46, 48  
complementary, 2, 3, 46  
C-terminus, 2, 46  
cytosine, 2, 46, 47  
denaturation, 2, 47  
deoxyribonucleotide, 2, 47  
DNA, 1, 2, 3, 4, 9, 47, 49  
DNA polymerase, 1, 3, 47  
edit distance, 9, 11, 47  
enzyme, 47  
eukaryotic, 2, 47  
extrachromosomal, 2, 47  
gene, 3, 4, 5, 11, 22, 47  
genetic code, 3, 19, 36, 37, 38, 40, 41, 43, 44, 45, 47  
genome, 3, 45, 47  
guanine, 2, 47  
hybridization, 3, 4, 11, 22, 23, 48  
initiation codon, 3, 48  
machine learning, 5, 11, 12, 14, 22, 23, 24, 26, 28, 41, 49  
MAP hypothesis, 13, 48  
maximum a posteriori hypothesis, 13, 48  
mitochondria, 2, 3, 48  
naïve Bayes classifier, 14, 48  
N-terminus, 2, 48  
Operational Taxonomic Unit, 17, 48  
PAM, 9, 48  
pathway, 48  
PCR, 4, 48  
peptide, 2, 48  
polymerase chain reaction, 4, 48  
polypeptide, 1, 2, 48  
posterior probability, 13, 48  
primary structure, 1, 48  
primer, 4, 48  
prior probability, 14, 48  
promoter, 3, 48  
protein, 1, 2, 3, 11, 36, 39, 48  
quaternary structure, 1, 49  
reading frame, 3, 36, 48, 49  
replication, 1, 3, 49  
reverse complement, 2, 3, 4, 49  
ribosome, 49  
RNA, 1, 3, 11, 49  
RNA polymerase, 1, 3, 49  
secondary structure, 1, 11, 49  
sequence, 1, 2, 3, 4, 9, 11, 15, 16, 22, 36, 43, 49  
squashing function, 12, 49  
stochastic gradient descent, 12, 49  
synthetic oligonucleotide, 4, 49  
tertiary structure, 1, 49  
thymine, 2, 3, 49  
transcription, 1, 3, 50  
transcription factor, 3, 49  
translation, 3, 50  
uracil, 3, 50  
virus, 11, 22, 26, 50