SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOINFORMATICS

UNIT – I - SBIA5303 - MACHINE LEARNING FOR BIOINFORMATICS

UNIT 1 MACHINE LEARNING Machine-Learning Foundations: The Probabilistic Framework - Introduction: Bayesian modeling -The Cox Jaynes axioms Bayesian inference & induction - Model structures: graphical models & other tricks - Probabilistic Modelling & Inference: Examples -The simplest sequence models - Statistical mechanics - Machine Learning Algorithms - Introduction -Dynamic programming -Gradient descent -EM/GEM algorithms – Markov chain Monte- Carlo methods - Simulated annealing Evolutionary & genetic algorithms. Learning algorithms: miscellaneous aspects.

Machine-Learning Foundations: The Probabilistic Framework

Machine learning is by and large a direct descendant of an older discipline, statistical model fitting. Like its predecessor, the goal in machine learning is to extract useful information from a corpus of data D by building good probabilistic models. The particular twist behind machine learning, however, is to automate this process as much as possible, often by using very flexible models characterized by large numbers of parameters, and to let the machine take care of the rest. Silicon machine learning also finds much of its inspiration in the learning abilities of its biological predecessor: the brain. Hence, a particular vocabulary is required in which "learning" often replaces "fitting."

Clearly, machine learning is driven by rapid technological progress in two areas:

- Sensors and storage devices that lead to large databases and data sets
- Computing power that makes possible the use of more complex models.

As pointed out in [455], machine-learning approaches are best suited for areas where there is a large amount of data but little theory. And this is exactly the situation in computational molecular biology.

While available sequence data are rapidly increasing, our current knowledge of biology constitutes only a small fraction of what remains to be discovered. Thus, in computational biology in particular, and more generally in biology and all other information-rich sciences, one must reason in the presence of a high degree of uncertainty: many facts are missing, and some of

the facts are wrong. Computational molecular biologists are, then, constantly faced with induction and *inference* problems: building models from available data. What are the right class of models and the right level of complexity? Which details are important and which should be discarded? How can one compare different models and select the best one, in light of available knowledge and sometimes limited data? In short, how do we know if a model is a good model? These questions are all the more acute in machine-learning approaches, because complex models, with several thousand parameters and more, are routinely used and sequence data, while often abundant, are inherently "noisy."

In situations where few data are available, the models used in machine-learning approaches have sometimes been criticized on the ground that they may be capable of accommodating almost any behavior for a certain setting of their parameters, and that simpler models with fewer parameters should be preferred to avoid overfitting. Machine-learning practitioners know that many *implicit* constraints emerge from the structure of the models and, in fact, render arbitrary behavior very difficult, if not impossible, to reproduce. More important, as pointed out in [397], choosing simpler models because few data are available does not make much sense. While it is a widespread practice and occasionally a useful heuristic, it is clear that the amount of data collected and the complexity of the underlying source are two completely different things. It is not hard to imagine situations characterized by a very complex source and relatively few data. Thus we contend that even in situations where data are scarce, machine-learning approaches should not be ruled out a priori. But in all cases, it is clear that questions of inference and induction are particularly central to machine learning and to computational biology.

When reasoning in the presence of certainty, one uses deduction. This is how the most advanced theories in information-poor sciences, such as physics or mathematics, are presented in an axiomatic fashion. Deduction is not controversial. The overwhelming majority of people agree on how to perform deductions in specific way: if X implies Y, and X is true, then Y must be true. This is the essence of Boole's algebra, and at the root of all our digital computers. When reasoning in the presence of uncertainty, one uses induction and inference: if X implies Y, and Y is true, then X is *more plausible*. An amazing and still poorly known fact is that there is a simple and unique consistent set of rules for induction, model selection, and comparison. It is called Bayesian inference. The Bayesian approach has been known for some time, but only recently has it started to infiltrate different areas of science and technology systematically, with useful results [229, 372, 373]. While machine learning may appear to some as an eclectic collection of models and learning algorithms, we believe the Bayesian framework provides a strong underlying foundation that unifies the different techniques. We now review the Bayesian

framework in general. In the following chapters, we apply it to specific classes of models and problems.

The Bayesian point of view has a simple intuitive description. The Bayesian approach assigns a degree of plausibility to any proposition, hypothesis, or model. (Throughout this book, hypothesis and model are essentially synonymous; models tend to be complex hypotheses with many parameters.) More precisely, in order properly to carry out the induction process, one ought to proceed in three steps:

1. Clearly state what the hypotheses or models are, along with *all* the background information and the data.

2. Use the language of probability theory to assign prior probabilities to the hypotheses.

3. Use probability calculus for the inference process, in particular to evaluate posterior probabilities (or degrees of belief) for the hypotheses in light of the available data, and to derive *unique* answers.

Such an approach certainly seems reasonable. Note that the Bayesian approach is not directly concerned with the creative process, how to generate new hypotheses or models. It is concerned only with assessing the value of models with respect to the available knowledge and data. This assessment procedure, however, may be very helpful in generating new ideas.

But why should the Bayesian approach be so compelling? Why use the language of probability theory, as opposed to any other method? The surprising answer to this question is that it can be proved, in a strict mathematical sense, that this is the only consistent way of reasoning in the presence of uncertainty. Specifically, there is a small set of very simple commonsense axioms, the Cox Jaynes axioms, under which it can be shown that the Bayesian approach is the unique consistent approach to inference and induction. Under the Cox Jaynes axioms, degrees of plausibility satisfy all the rules of probabilities exactly. Probability calculus is, then, all the machinery that is required for inference, model selection, and model comparison.

In the next section, we give a brief axiomatic presentation of the Bayesian point of view using the Cox Jaynes axioms. For brevity, we do not present any proofs or any historical background for the Bayesian approach, nor do we discuss any controversial issues regarding the foundations of statistics. All of these can be found in various books and articles, such as [51, 63, 122, 433, 284].

The Cox Jaynes Axioms

The objects we deal with in inference are propositions about the world. For instance, a typical proposition X is "Letter A appears in position i of sequence O." A proposition is either true or false, and we denote by $\bar{X}$ the complement of a proposition X. A hypothesis H about the world is a proposition, albeit a possibly complex one composed of the conjunction of many more elementary propositions. A model M can also be viewed as a hypothesis. The difference is that models tend to be very complex hypotheses involving a large number of parameters. In discussions where parameters are important, we will consider that $M = M(w)$, where w is the vector of all parameters. A complex model M can easily be reduced to a binary proposition in the form "Model M accounts for data D with an error level ϵ" (this vague statement will be made more precise in the following discussion). But for any purpose, in what follows there is no real distinction between models and hypotheses.

Whereas propositions are either true or false, we wish to reason in the presence of uncertainty. Therefore the next step is to consider that, given a certain amount of information I, we can associate with each hypothesis a degree of plausibility or confidence (also called degree or strength of belief). Let us represent it by the symbol $\pi(X|I)$. While $\pi(X|I)$ is just a symbol for now, it is clear that in order to have a scientific discourse, one should be able to compare degrees of confidence. That is, for any two propositions X and Y, either we believe in X more than in Y, or we believe in Y more than in X, or we believe in both equally. Let us use the symbol $>$ to denote this relationship, so that we write $\pi(X|I) > \pi(Y|I)$ if and only if X is more plausible than Y. It would be very hard not to agree that in order for things to be sensible, the relationship $>$ should be transitive. That is, if X is more plausible than Y, and Y is more plausible than Z, then X must be more plausible than Z. More formally, this is the first axiom,

$$\pi(X|I) > \pi(Y|I) \quad \text{and} \quad \pi(Y|I) > \pi(Z|I) \quad \text{imply} \quad \pi(X|I) > \pi(Z|I). \quad (2.1)$$

This axiom is trivial; it has, however, an important consequence: $>$ is an ordering relationship, and therefore degrees of belief can be expressed by real numbers. That is, from now on, $\pi(X|I)$ represents a number. This of course does not mean that such a number is easy to calculate, but merely that such a number exists, and the ordering among hypotheses is reflected in the ordering of real numbers. To proceed any further and stand a chance of calculating degrees of belief we need additional axioms or rules for relating numbers representing strengths of belief.

The amazing fact is that only two additional axioms are needed to constrain the theory entirely. This axiomatic presentation is usually attributed to

Cox and Jaynes [138, 283]. To better understand these two remaining axioms, the reader may imagine a world of very simple switches, where at each instant in time a given switch can be either on or off. Thus, all the elementary hypotheses or propositions in this world, at a given time, have the simple form "switch X is on" or "switch X is off." (For sequence analysis purposes, the reader may imagine that switch X is responsible for the presence or absence of the letter X, but this is irrelevant for a general understanding.) Clearly, the more confident we are that switch X is on (X), the less confident we are that switch X is off ($\bar{X}$). Thus, for any given proposition X, there should be a relationship between $\pi(X|I)$ and $\pi(\bar{X}|I)$. Without assuming anything about this relationship, it is sensible to consider that, all else equal, the relationship should be the same for all switches and for all types of background information, that is, for all propositions X and I. Thus, in mathematical terms, the second axiom states that there exists a function F such that

$$\pi(\bar{X}|I) = F[\pi(X|I)]. \tag{2.2}$$

The third axiom is only slightly more complicated. Consider this time two switches X and Y and the corresponding four possible joint states. Then our degree of belief that X is on and Y is off, for instance, naturally depends on our degree of belief that switch X is on, and our degree of belief that switch Y is off, *knowing that X is on*. Again, it is sensible that this relationship be independent of the switch considered and the type of background information I. Thus, in mathematical terms, the third axiom states that there exists a function G such that

$$\pi(X, Y|I) = G[\pi(X|I), \pi(Y|X, I)]. \tag{2.3}$$

So far, we have not said much about the information I. I is a proposition corresponding to the conjunction of all the available pieces of information. I can represent background knowledge, such as general structural or functional information about biological macromolecules. I can also include specific experimental or other data. When it is necessary to focus on a particular corpus of data D, we can write $I = (I, D)$. In any case, I is not necessarily fixed and can be augmented with, or replaced by, any number of symbols representing propositions, as already seen in the right-hand side of (2.3). When data are acquired sequentially, for instance, we may write $I = (I, D_1, \ldots, D_n)$. In a discussion where I is well defined and fixed, it can be dropped altogether from the equations.

The three axioms above entirely determine, up to scaling, how to calculate degrees of belief. In particular, one can prove that there is always a rescaling κ of degrees of belief such that $\mathbf{P}(X|I) = \kappa(\pi(X|I))$ is in $[0, 1]$. Furthermore, $\mathbf{P}$ is unique and satisfies all the rules of probability. Specifically, if degrees of belief are restricted to the $[0, 1]$ interval, then the functions F and G must be

given by $F(x) = 1 - x$ and $G(x, y) = xy$. The corresponding proof will not be given here and can be found in [138, 284]. As a result, the second axiom can be rewritten as the sum rule of probability,

$$\mathbf{P}(X|I) + \mathbf{P}(\bar{X}|I) = 1, \tag{2.4}$$

and the third axiom as the product rule,

$$\mathbf{P}(X, Y|I) = \mathbf{P}(X|I)\mathbf{P}(Y|X, I). \tag{2.5}$$

Bayesian Inference and Induction

We can now turn to the type of inference we are most interested in: deriving a parameterized model $M = M(w)$ from a corpus of data D. For simplicity, we will drop the background information I from the following equations. From Bayes theorem we immediately have

$$\mathbf{P}(M|D) = \frac{\mathbf{P}(D|M)\mathbf{P}(M)}{\mathbf{P}(D)} = \mathbf{P}(M)\frac{\mathbf{P}(D|M)}{\mathbf{P}(D)}. \tag{2.7}$$

The prior $\mathbf{P}(M)$ represents our estimate of the probability that model M is correct before we have obtained any data. The posterior $\mathbf{P}(M|D)$ represents our updated belief in the probability that model M is correct given that we have observed the data set D. The term $\mathbf{P}(D|M)$ is referred to as the likelihood.

For data obtained sequentially, one has

$$\mathbf{P}(M|D^1, \ldots, D^t) = \mathbf{P}(M|D^1, \ldots, D^{t-1})\frac{\mathbf{P}(D^t|M, D^1, \ldots, D^{t-1})}{\mathbf{P}(D^t|D^1, \ldots, D^{t-1})}. \tag{2.8}$$

In other words, the old posterior $\mathbf{P}(M|D^1, \ldots, D^{t-1})$ plays the role of the new prior. For technical reasons, probabilities can be very small. It is often easier to work with the corresponding logarithms, so that

$$\log\mathbf{P}(M|D) = \log\mathbf{P}(D|M) + \log\mathbf{P}(M) - \log\mathbf{P}(D). \tag{2.9}$$

To apply (2.9) to any class of models, we will need to specify the prior $\mathbf{P}(M)$ and the data likelihood $\mathbf{P}(D|M)$. Once the prior and data likelihood terms are made explicit, the initial modeling effort is complete. All that is left is cranking the engine of probability theory. But before we do that, let us briefly examine some of the issues behind priors and likelihoods in general.

Model Structures: Graphical Models and Other Tricks

Clearly, the construction or selection of suitable models is dictated by the data set, as well as by the modeler's experience and ingenuity. It is, however, possible to highlight a small number of very general techniques or tricks that can be used to shape the structure of the models. Most models in the literature can be described in terms of combinations of these simple techniques. Since in machine learning the starting point of any Bayesian analysis is almost always a high-dimensional probability distribution $\mathbf{P}(M, D)$ and the related conditional and marginal distributions (the posterior $\mathbf{P}(M|D)$, the likelihood $\mathbf{P}(D|M)$, the prior $\mathbf{P}(M)$, and the evidence $\mathbf{P}(D)$); these rules can be seen as ways of decomposing, simplifying, and parameterizing such high-dimensional distributions.

Probabilistic Modeling and Inference: Examples

What are the implications of a Bayesian approach for modeling? For any type of model class, it is clear that the first step must be to make the likelihood $\mathbf{P}(D|M)$ and the prior $\mathbf{P}(M)$ explicit. In this chapter, we look at a few simple applications of the general probabilistic framework. The first is a very simple sequence model based on die tosses. All other examples in the chapter, including the basic derivation of statistical mechanics, are variations obtained either by increasing the number of dice or by varying the observed data.

The Simplest Sequence Models

The simplest, but not entirely trivial, modeling situation is that of a single coin flip. This model has a single parameter p and the data consist of a string, containing a single letter, over the alphabet $A = \{\mathsf{H}, \mathsf{T}\}$, H for head and T for tail. Since we are interested in DNA sequences, we shall move directly to a slightly more complex version with four letters, rather than two, and the possibility of observing longer strings.

Statistical Mechanics

There are at least five good reasons to understand the rudiments of statistical mechanics in connection with machine learning and computational biology. First, statistical mechanics can be viewed as one of the oldest and best examples of Bayesian reasoning [280, 281], although the presentation often given is slightly flawed in our opinion because of the confusion between MaxEnt and Bayes. Second, statistical mechanics has traditionally been concerned with deriving the statistical macroscopic properties of large ensembles of simple microscopically interacting units—the equilibrium behavior, the phase transitions, and so on. The results and techniques of statistical mechanics are useful in understanding the properties and learning evolution of a number of graphical models used in machine learning [252, 482, 50]. Statistical mechanical models have also been applied directly to biological macromolecules—for instance, in the protein-folding problem (see [151] for a review). Finally, statistical mechanics is useful for understanding several algorithms that are fundamental for machine learning, such as simulated annealing and the EM algorithms described in chapter 4.

Here we give a Bayesian derivation of statistical mechanics from first principles, and develop the basic concepts, especially those of the Boltzmann-Gibbs distribution and free energy, that will be needed in the next chapters. In the basic statistical mechanics framework, one considers a stochastic system that can be in a number of "microscopic" states: $\mathbf{S} = \{s_1, \ldots, s_{|S|}\}$, with p_s denoting the probability of being in state s for a distribution $P = (p_s)$. This can be viewed as a die model $M(w)$, with parameters $w = p_s$, although for the time being it is not necessary to assume that the tosses are independent. The key difference from the examples above is in the data. The faces of the die, the microscopic states, are not observable but act as hidden variables. Instead, we assume that there is a function $f(s)$ of the states and that the only "macroscopic" observable quantity, the data, is the expectation or average of f. So, with a slight abuse of notation, in this section we write $D = \mathbf{E}(f) = \sum_s p_s f(s)$.

Very often in statistical mechanics the states have a microscopic structure so that $s = (x_1, \ldots, x_n)$, where the x_i are local variables. For instance, the x_i can be binary spin variables, in which case $|S| = 2^n$. Likewise, f is typically the energy of the system and can be written as a quadratic function in the local variables: $f(s) = f(x_1, \ldots, x_n) = \sum_{ij} w_{ij} x_i x_j + \sum_i w_i x_i$. The interaction parameters w_{ij} can be local, as in the case of spins on a lattice, or long-range, and are related to the underlying graphical model. While such assumptions are important in modeling particular systems and developing a detailed theory, they will not be needed in the following sections. The first question we can ask is: Given the observation of the average of f, what can we say about the state distribution P?

Machine Learning Algorithms

Dynamic Programming

Dynamic programming [66] is to a very general optimization technique that can be applied any time a problem can be recursively subdivided into two similar subproblems of smaller size, such that the solution to the larger problem can be obtained by piecing together the solutions to the two subproblems. The prototypical problem to which dynamic programming can be applied is that of finding the shortest path between two nodes in a graph. Clearly the shortest path from node A to node B, going through node C, is the concatenation of the shortest path from A to C with the shortest path from C to B. This is also called the "Bellman principle." A general solution to the original problem is then constructed by recursively piecing together shorter optimal paths.

Dynamic programming and its many variations are ubiquitous in sequence analysis. The Needleman–Wunch and Smith–Waterman algorithms [401, 481, 492], as well as all other sequence-alignment algorithms such as the Viterbi decoding algorithm of electrical engineers, are examples of dynamic programming. Alignment algorithms can be visualized in terms of finding the shortest path in the appropriate graph with the appropriate metric. Aligning two sequences of length of N requires finding a shortest path in a graph with N^2 vertices. Since dynamic programming essentially requires visiting all such vertices once, it is easy to see that its time complexity scales as $O(N^2)$.

In chapters 7 and 8, dynamic programming and the Viterbi algorithm are heavily used to compute likelihoods and align sequences to HMMs during the training and exploitation phases. Accordingly, we give there a detailed derivation of the corresponding algorithms. Other variations on dynamic programming used in other chapters are sketched or left as an exercise. Because dynamic programming is very well known and is at the root of many conventional algorithms for sequence analysis, we refer the reader to the abundant literature on the topic (in particular [550] and references therein). Reinforcement-learning algorithms are also another important class of learning algorithms that can be viewed as generalizations of dynamic programming ideas [298].

Gradient Descent

Often we are interested in parameter estimation, that is, in finding the best possible model $M(w)$ that minimizes the posterior $f(w) = -\log \mathbf{P}(w|D)$, or possibly the likelihood $-\log \mathbf{P}(D|w)$. Whenever a function $f(w)$ is differentiable, one can try to find its minima by using one of the oldest optimization algorithms, gradient descent. As its name indicates, gradient descent is an iterative procedure that can be expressed vectorially as

$$w^{t+1} = w^t - \eta \frac{\partial f}{\partial w^t}, \tag{4.1}$$

where η is the step size, or learning rate, which can be fixed or adjusted during the learning process.

While the general gradient-descent principle is simple, in complex parameterized models it can give rise to different implementations, depending on how the gradient is actually computed [26]. In graphical models, this often requires the propagation of information "backwards." As we will see in the next chapters, this is the case for gradient-descent learning applied to neural networks (the backpropagation algorithm) and to hidden Markov models (the forward–backward procedure). Obviously the outcome of a gradient-descent procedure depends on the initial estimate. Furthermore, if the function being optimized has a complex landscape, gradient descent in general will terminate in a local minimum rather than a global one. Whenever feasible, therefore, it is wise to run the procedure several times, with different starting points and learning rates.

It is well known that there are situations where plain gradient descent can be slow and inefficient. To overcome such problems, a number of variations on gradient descent are possible, such as conjugate gradient descent, that use second-order information or more complex directions of descent constructed from the current gradient and the history of previous directions. Additional details and references can be found in [434]. In spite of its relative crudeness, gradient descent remains useful, easy to implement, and widely used.

EM/GEM Algorithms

Another important class of optimization algorithms is the expectation maximization (EM) and generalized expectation maximization (GEM) algorithms [147, 387]. Such algorithms have been used in many different applications and also in sequence analysis [352, 113]. In the case of HMMs, the EM algorithm is also called the Baum–Welch algorithm [54]. Since the usefulness of these algorithms goes beyond HMMs, we give here a general treatment of EM/GEM algorithms, using the concept of free energy of chapter 3, along the lines suggested in [400].

The EM algorithm is useful in models and situations with hidden variables. Typical examples of hidden variables are missing or unobservable data, mixture parameters in a mixture model, and hidden node states in graphical models (hidden units in NNs, hidden states in HMMs). If D denotes the data, we

assume that there is available a parameterized joint distribution on the hidden and observed variables $\mathbf{P}(D, H|w)$, parameterized by w. In the case of main interest to us, w denotes, as usual, the parameters of a model. Let us assume that the objective is to maximize the likelihood $\log \mathbf{P}(D|w)$. The same ideas can easily be extended to the case of MAP estimation. Since in general it is difficult to optimize $\log \mathbf{P}(D|w)$ directly, the basic idea is to try to optimize the expectation $\mathbf{E}(\log \mathbf{P}(D|w))$:

$$\mathbf{E}(\log \mathbf{P}(D|w)) = \mathbf{E}(\log \mathbf{P}(D, H|w) - \log \mathbf{P}(H|D, w)). \tag{4.2}$$

Markov-Chain Monte-Carlo Methods

Markov-chain Monte-Carlo (MCMC) methods belong to an important class of stochastic methods that are related to statistical physics and are increasingly used in Bayesian inference and machine learning [578, 202, 396, 520, 69]. Recall that one of the basic goals derived from the general Bayesian framework is to compute expectations with respect to a high-dimensional probability distribution $P(x_1, \ldots, x_n)$, where the x_i can be the values of model parameters or hidden variables, as well as observed data. The two basic ideas behind MCMC are very simple. The first idea (Monte Carlo) is to approximate such expectations by

$$\mathbf{E}(f) = \sum_{x_1, \ldots, x_n} f(x_1, \ldots, x_n) P(x_1, \ldots, x_n) \approx \frac{1}{T} \sum_{t=0}^{T} f(x_1^t, \ldots, x_n^t) \tag{4.9}$$

for large T, provided $(x_1^t, \ldots, x_n^t)$ are sampled according to their distribution $P(x_1, \ldots, x_n)$. In order to sample from P, the second basic idea is to construct a Markov chain having P as its equilibrium distribution, then simulate the chain and try to sample from its equilibrium distribution.

Before we proceed with the rudiments of Markov chains, it is worth noting a few points. The mean of the estimator on the right-hand side of (4.9) is $\mathbf{E}(f)$. If the samples are independent, its variance is $\mathbf{Var}(f)/T$. In this case, the precision of the estimate does not depend on the dimension of the space being sampled. Importance sampling and rejection sampling are two well-known Monte-Carlo algorithms for generating independent samples that will not be reviewed here. Both algorithms tend to be inefficient in high-dimensional state spaces. The samples created using Markov-chain methods are not independent. But at equilibrium they are still representative of P. The dependence of one sample on the previous one is the key to the better efficiency of MCMC methods with higher-dimensional spaces. After all, if P is differentiable or

even just continuous, the probability $P(x_1, \ldots, x_n)$ of a sample provides information about its neighborhood. This remains true even in cases where P can be computed efficiently only up to a constant normalizing factor. Finally, MCMC methods, like any other method based on a single estimator, are at best an approximation to the ideal Bayesian inference process that would rely on the calculation of $\mathbf{P}(\mathbf{E}(f)|D)$ given any sample D.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

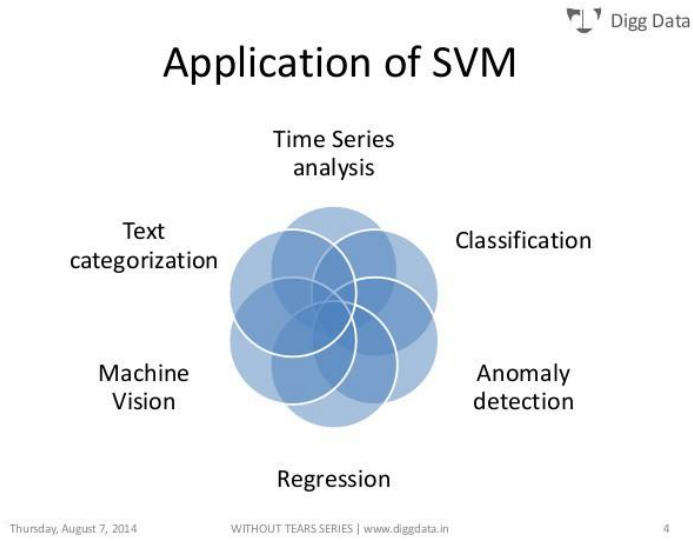
DEPARTMENT OF BIOINFORMATICS

UNIT – II - SBIA5303 - MACHINE LEARNING FOR BIOINFORMATICS

UNIT 2 SUPPORT VECTOR MACHINE (SVM) SVM: introduction – architecture – kernel - ROC – feature selection – sensitivity – specificity – accuracy – implementation svm applications in sequence analysis, structure prediction , drug design –svm light - libsvm – weka.

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes.

Support Vector Machines (SVM) is a Machine Learning Algorithm which can be used for many different tasks (Figure 1). In this article, I will explain the mathematical basis to demonstrate how this algorithm works for binary classification purposes.



The main objective in SVM is to find the optimal hyperplane to correctly classify between data points of different classes (Figure 2). The hyperplane dimensionality is equal to the number of input features minus one (eg. when working with three feature the hyperplane will be a two-dimensional plane).

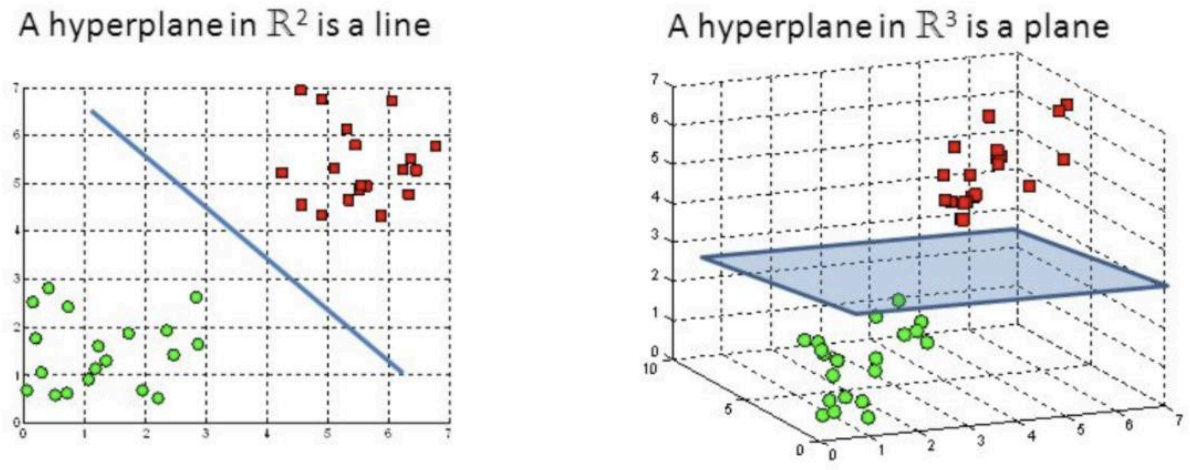


Figure 2: SVM Hyperplane [2]

Data points on one side of the hyperplane will be classified to a certain class while data points on the other side of the hyperplane will be classified to a different class (eg. green and red as in Figure 2). The distance between the hyperplane and the first point (for all the different classes) on either side of the hyperplane is a measure of sure the algorithm is about its classification decision. The bigger the distance and the more confident we can be SVM is making the right decision.

The data points closest to the hyperplane are called Support Vectors. Support Vectors determines the orientation and position of the hyperplane, in order to maximise the classifier margin (and therefore the classification score). The number of Support Vectors the SVM algorithm should use can be arbitrarily chosen depending on the applications.

Basic SVM classification can be easily implemented using the Scikit-Learn Python library in a few lines of code.

```
from sklearn import svm
trainedsvm = svm.SVC().fit(X_Train, Y_Train)
predictionsvm = trainedsvm.predict(X_Test)
print(confusion_matrix(Y_Test,predictionsvm))
print(classification_report(Y_Test,predictionsvm))
```

- **Hard Margin:** aims to find the best hyperplane without tolerating any form of misclassification.
- **Soft Margin:** we add a degree of tolerance in SVM. In this way we allow the model to voluntary misclassify a few data points if that can lead to identifying a hyperplane able to generalise better to unseen data.

Soft Margin SVM can be implemented in Scikit-Learn by adding a C penalty term in svm.SVC. The bigger C and the more penalty the algorithm gets when making a misclassification.

Kernel Trick

If the data we are working with is not linearly separable (therefore leading to poor linear SVM classification results), it is possible to apply a technique known as the Kernel Trick. This method is able to map our non-linear separable data into a higher dimensional space, making our data linearly separable. Using this new dimensional space SVM can then be easily implemented (Figure 3).

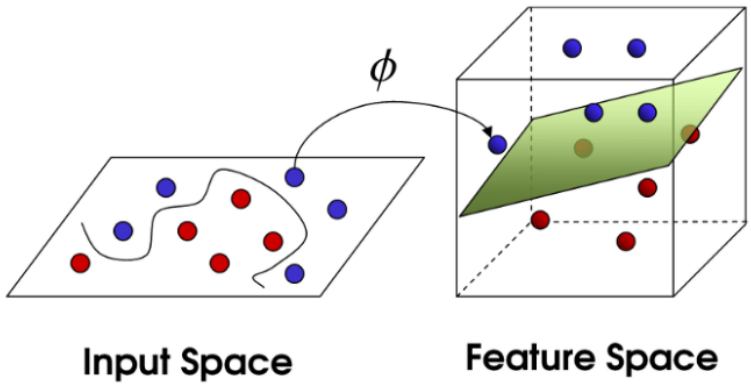


Figure 3: Kernel Trick [3]

There are many different types of Kernels which can be used to create this higher dimensional space, some examples are linear, polynomial, Sigmoid and Radial Basis Function (RBF). In Scikit-Learn a Kernel function can be specified by adding a kernel parameter in svm.SVC. An additional parameter called gamma can be included to specify the influence of the kernel on the model.

It is usually suggested to use linear kernels if the number of features is larger than the number of observations in the dataset (otherwise RBF might be a better choice).

When working with a large amount of data using RBF, speed might become a constraint to take into account.

Feature Selection

Once having fitted our linear SVM it is possible to access the classifier coefficients using .coef_ on the trained model. These weights figure the orthogonal vector coordinates orthogonal to the hyperplane. Their direction represents instead the predicted class.

Feature importance can, therefore, be determined by comparing the size of these coefficients to each other. By looking at the SVM coefficients it is, therefore, possible to identify the main features used in classification and get rid of the not important ones (which hold less variance).

Reducing the number of features in Machine Learning plays a really important role especially when working with large datasets. This can in fact: speed up training, avoid overfitting and ultimately lead to better classification results thanks to the reduced noise in the data.

In Figure 4 are shown the main features I identified using SVM on the Pima Indians Diabetes Database. In green are shown all the features corresponding to the negative coefficients and in blue the positive ones. If you want to find out more about it, all my code is freely available on my Kaggle and GitHub profiles.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOINFORMATICS

UNIT – III - SBIA5303 - MACHINE LEARNING FOR BIOINFORMATICS

UNIT 3 NEURAL NETWORKS Neural Networks: The Theory -Introduction - Universal approximation properties – Priors & likelihoods - Learning algorithms: backpropagation - Neural Networks: Applications - Sequence encoding & output interpretation- Sequence correlations & neural networks – Prediction of protein secondary structure - Prediction of signal peptides & their cleavage sites Applications for DNA & RNA nucleotide sequences - Prediction performance evaluation - Different performance measures Perceptrons and Multilayer Perceptrons -Neural Networks in Drug Design.

Neural Networks: The Theory

Artificial neural networks (NNs) [456, 252, 70] were originally developed with the goal of modeling information processing and learning in the brain. While the brain metaphor remains a useful source of inspiration, it is clear today that the artificial neurons used in most NNs are quite remote from biological neurons [85]. The development of NNs, however, has led to a number of practical applications in various fields, including computational molecular biology. NNs have become an important tool in the arsenal of machine-learning techniques that can be applied to sequence analysis and pattern recognition problems.

At the most basic level, NNs can be viewed as a broad class of parameterized graphical models consisting of networks with interconnected units evolving in time. In this book we use only pairwise connections but, if desirable, one can use more elaborate connections associated with the interaction of more than two units, leading to the "higher-order" or "sigma-pi" networks [456]. The connection from unit j to unit i usually comes with a weight denoted by w_{ij}. Thus we can represent an NN with a weight-directed graph or "architecture." For simplicity, we do not use any self-interactions, so that we can assume that $w_{ii} = 0$ for all the units.

It is customary to distinguish a number of important architectures, such as recurrent, feed-forward, and layered. A *recurrent* architecture is an architecture that contains directed loops. An architecture devoid of directed loops is said to be *feed-forward*. Recurrent architectures are more complex with richer dynamics and will be considered in chapter 9. An architecture is *layered* if the units are partitioned into classes, also called layers, and the connectivity patterns are defined between the classes. A feed-forward architecture is not necessarily layered.

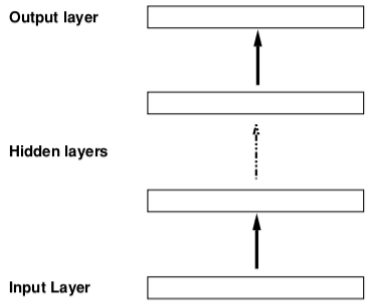


Figure 5.1: Layered Feed-Forward Architecture or Multilayer Perceptron (MLP). Layers may contain different numbers of units. Connectivity patterns between layers also may vary.

In most of this chapter and in many current applications of NNs to molecular biology, the architectures used are layered feed-forward architectures, as in figure 5.1. The units are often partitioned into *visible* units and *hidden* units. The *visible* units are those in contact with the external world, such as input and output units. Most of the time, in simple architectures the input units and the output units are grouped in layers, forming the input layer and the output layer. A layer containing only hidden units is called a hidden layer. The number of layers is often referred to as the "depth" of a network. Naturally NNs can be assembled in modular and hierarchical fashion to create complex overall architectures. The design of the visible part of an NN depends on the input representation chosen to encode the sequence data and the output that may typically represent structural or functional features.

The behavior of each unit in time can be described using either differential equations or discrete update equations (see [26] for a summary). Only the discrete formalism will be used in this book. In a layered feed-forward architecture, all the units in a layer are updated simultaneously, and layers are updated sequentially in the obvious order. Sometimes it is also advantageous to use stochastic units (see appendix C on graphical models and Bayesian networks). In the rest of this chapter, however, we focus on deterministic units. Typically a unit i receives a total input x_i from the units connected to it, and then produces an output $y_i = f_i(x_i)$, where f_i is the transfer function of the unit. In general, all the units in the same layer have the same transfer function,

and the total input is a weighted sum of incoming outputs from the previous layer, so that

$$x_i = \sum_{j \in N^-(i)} w_{ij} y_j + w_i, \tag{5.1}$$

$$y_i = f_i(x_i) = f_i\left(\sum_{j \in N^-(i)} w_{ij} y_j + w_i \right), \tag{5.2}$$

where w_i is called the bias, or threshold, of the unit. It can also be viewed as a connection with weight w_i to an additional unit, with constant activity clamped to 1. The weights w_{ij} and w_i are the parameters of the NNs. In more general NNs other parameters are possible, such as time constants, gains, and delays. In the architectures to be considered here, the total number of parameters is determined by the number of layers, the number of units per layer, and the connectivity between layers. A standard form for the connectivity between layers is the "fully connected" one, where each unit in one layer is connected to every unit in the following layer. More local connectivity patterns are obviously more economical. Note, however, that even full connectivity between layers is sparse, compared with complete connectivity among all units. In situations characterized by some kind of translation invariance, it can be useful for each unit in a given layer to perform the same operation on the activity of translated groups of units in the preceding layer. Thus a single pattern of connections can be shared across units in a given layer. In NN jargon this is called "weight sharing." It is routinely used in image-processing problems and has also been used with some success in sequence analysis situations where distinct features are separated by variable distances. The shared pattern of weights defines a filter or a convolution kernel that is used to uniformly process the incoming activity. With weight sharing, the number of free parameters associated with two layers can be small, even if the layers are very large. An example of this technique is given below in section 6.3 on secondary structure prediction.

Universal Approximation Properties

Perhaps one reassuring property of NNs is that they can approximate any reasonable function to any degree of required precision. The result is trivial[1] for Boolean functions, in the sense that any Boolean function can be built using a combination of threshold gates. This is because any Boolean function can be synthesized using NOT and AND gates, and it is easy to see that AND and NOT gates can be synthetized using threshold gates. For the general regression case, it can be shown that any reasonable real function $f(x)$ can be approximated to any degree of precision by a three-layer network with x in the input layer, a hidden layer of sigmoidal units, and one layer of linear output units,

as long as the hidden layer can be arbitrarily large. There are a number of different mathematical variations and proofs of this result (see, e.g., [264, 265]).

Here we give a simple constructive proof of a special case, which can easily be generalized, to illustrate some of the basic ideas. For simplicity, consider a continuous function $y = f(x)$ where both x and y are one-dimensional. Assume without loss of generality that x varies in the interval $[0,1]$, and that we want to compute the value of $f(x)$ for any x within a precision ϵ. Since f is continuous over the compact interval $[0,1]$, f is uniformly continuous and there exists an integer n such that

$$|x_2 - x_1| \leq \frac{1}{n} \Longrightarrow |f(x_2) - f(x_1)| \leq \epsilon. \tag{5.8}$$

Priors and Likelihoods

We now apply the general theory of chapter 2. In particular, we show how the theory can be used to determine the choice of an objective function and of the transfer functions of the output units. In this section we shall assume that the data consist of a set of independent input-output pairs $D_i = (d_i, t_i)$. The data are noisy in the sense that for a given d_i, different outputs t_i could be observed. Noise at the level of the input d could also be modeled, but will not be considered here. The operation of the NN itself is considered to be deterministic. We have

$$\mathbf{P}((d_i, t_i)|w) = \mathbf{P}(d_i|w)\mathbf{P}(t_i|d_i, w) = \mathbf{P}(d_i)\mathbf{P}(t_i|d_i, w), \tag{5.9}$$

the last equality resulting from the fact that in general we can assume that the inputs d are independent of the parameters w. Thus, for a given architecture

parameterized by w, we have, using (2.9),

$$-\log \mathbf{P}(w|D) = -\sum_{i=1}^{K} \log \mathbf{P}(t_i|d_i, w) - \sum_{i=1}^{K} \log \mathbf{P}(d_i) - \log \mathbf{P}(w) + \log \mathbf{P}(D), \tag{5.10}$$

where we have used the fact that $\mathbf{P}((d_i, t_i)|w) = \mathbf{P}(d_i)P(t_i|d_i, w)$, and have taken into account the independence of the different data points. In the first level of Bayesian inference (MAP), we want to minimize the left-hand side. We can ignore $\mathbf{P}(D)$ as well as $\mathbf{P}(d_i)$, since these terms do not depend on w, and concentrate on the prior term and the likelihood.

In order to calculate the likelihood, we shall have to distinguish different cases, such as regression and classification, and further specify the probabilistic model. In doing so, we follow the analysis in [455]. But the basic idea is to consider that, for a given input d_i, the network produces an estimated output $y(d_i)$. The model is entirely defined when we specify how the observed data $t_i = t(d_i)$ can statistically deviate from the network output $y_i = y(d_i)$. If the output layer has many units, we need to write y_{ij} for the output of the jth unit on the ith example. For notational convenience, in what follows we will drop the index that refers to the input. Thus we derive online equations for a generic input–output pair (d, t). Offline equations can easily be derived by summing over inputs, in accordance with (5.10).

Learning Algorithms: Backpropagation

In the majority of applications to be reviewed, MAP or ML estimation of NN parameters is done by gradient descent (see [26] for a general review). The calculations required to obtain the gradient can be organized in a nice fashion that leverages the graphical structure of NN. Using the chain rule, weights are updated sequentially, from the output layer back to the input layer, by propagating an error signal backward along the NN connections (hence the name "backpropagation"). More precisely, in the online version of the algorithm, and for each training pattern, we have for any weight parameter w_{ij}

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial y_i} f_i'(x_i) y_j. \tag{5.22}$$

Thus the gradient-descent learning equation is the product of three terms,

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}}{\partial w_{ij}} = -\eta \epsilon_i y_j, \tag{5.23}$$

where η is the learning rate, y_j is the output of the unit from which the connection originates (also called the presynaptic activity), and $\epsilon_i = (\partial \mathcal{E}/\partial y_i) f_i'(x_i)$ is a postsynaptic term called the backpropagated error. The backpropagated error can be computed recursively by

$$\frac{\partial \mathcal{E}}{\partial y_i} = \sum_{k \in N^+(i)} \frac{\partial \mathcal{E}}{\partial y_k} f_k'(x_k) w_{ki}. \tag{5.24}$$

The propagation from the children of a node to the node itself is the signature of backpropagation. While backpropagation is the most widely used algorithm for MAP estimation of MLPs, EM and simulated annealing have also been used. Algorithms for learning the architecture itself can also be envisioned, but they remain inefficient on large problems.

We can now review some of the main applications of NNs to molecular biology. Other general surveys of the topics can be found in [432, 571, 572].

Neural Networks: Applications

The application of neural network algorithms to problems within the field of biological sequence analysis has a fairly long history, taking the age of the whole field into consideration. In 1982 the perceptron was applied to the prediction of ribosome binding sites based on amino acid sequence input [506]. Stormo and coworkers found that the perceptron algorithm was more successful at finding *E. coli* translational initiation sites than a previously developed set of rules [507]. A perceptron without hidden units was able to generalize and could find translational initiation sites within sequences that were not included in the training set.

This linear architecture is clearly insufficient for many sequence recognition tasks. The real boost in the application of neural network techniques first came after the backpropagation training algorithm for the multilayer perceptron was brought into common use in 1986 [456], and especially after Qian and Sejnowski published their seminal paper on prediction of protein secondary structure in 1988 [437]. This and other papers that quickly followed [78, 262] were based on an adaptation of the NetTalk multilayer perceptron architecture [480], which from its input of letters in English text predicted the associated phonemes needed for speech synthesis and for reading the text aloud. This approach could immediately be adapted to tasks within the field of sequence analysis just by changing the input alphabet into alphabets of the amino acids or nucleotides. Likewise, the encoding of the phonemes could easily be transformed into structural classes, like those commonly used for the assignment of protein secondary structure (helices, sheets, and coil), or functional categories representing binding sites, cleavage sites, or residues being posttranslationally modified.

In this chapter we review some of the early work within the application areas of nucleic acids and proteins. We go into detail with some examples of more recent work where the methodologies are advanced in terms of either the training principles applied or the network architectures, especially when networks are combined to produce more powerful prediction schemes. We do not aim to mention and describe the complete spectrum of applications. For recent reviews see, for example, [432, 61, 77, 320, 571, 572].

Sequence Encoding and Output Interpretation

One important issue, before we can proceed with NN applications to molecular biology, is the encoding of the sequence input. In any type of prediction approach, the input representation is of cardinal importance. If a very clever input representation is chosen, one that reveals exactly the essentials for a particular task, the problem may be more or less solved, or at least can be solved by simple linear methods. In an MLP the activity patterns in the last hidden layer preceding the output unit(s) should represent the transformed input information in linearly separable form. This clearly is much easier if the input representation has not been selected so as further to increase the nonlinearity of the problem.

One would think that a very "realistic" encoding of the monomers in a sequence, using a set of physical-chemical features of potential relevance, should always outperform a more abstract encoding taken from the principles and practice of information theory [137]. However, in line with the contractive nature of most prediction problems (see section 1.4), it does not always help just to add extra information because the network has to discard most of it before it reaches the output level.

During training of an MLP, the network tries to segregate the input space into decision regions using hyperplanes. The numerical representation of the monomers therefore has a large impact on the ease with which the hidden units can position the planes in the space defined by the representation that has been chosen.

In many sequence analysis problems, the input is often associated with a window of size W covering the relevant sequence segment or segments. Typically the window is positioned symmetrically so that the upstream and downstream contexts are of the same size, but in some cases asymmetric windows perform far better than symmetric ones. When the task is to predict signal peptide cleavage sites (section 6.4) or intron splice sites in pre-mRNA (section 6.5.2), asymmetric windows may outperform symmetric ones. Both these sequence types (N-terminal protein sorting signals and noncoding intronic DNA) are eventually removed, and it makes sense to have most of the features needed for their processing in the regions themselves, leaving the mature protein least constrained. Windows with holes where the sequence

appears nonconsecutively have been used especially for the prediction of promoters and the exact position of transcriptional initiation in DNA, but also for finding beta-sheet partners in proteins [268, 46] and for the prediction of distance constraints between two amino acids based on the sequence context of both residues [368, 174].

Sequence Correlations and Neural Networks

Many structural or functional aspects of sequences are not conserved in terms of sequence, not even when amino acid similarities are taken into account. It is well known that protein structures, for example, can be highly conserved despite a very low sequence similarity when assessed and quantified by the amino acid identity position by position. What makes up a protein structure, either locally or globally, is the *cooperativity* of the sequence, and not just independent contributions from individual positions in it.

This holds true not only for the protein as a whole but also locally, say for a phosphorylation site motif, which must be recognized by a given kinase. Even for linear motifs that are known to interact with the same kinase, sequence patterns can be very different [331]. When the local structures of such sequence segments are inspected (in proteins for which the structure has been determined and deposited in the Protein Data Bank), they may indeed be conserved structurally despite the high compositional diversity [74].

The neural network technique has the potential of sensing this cooperativity through its ability to correlate the different input values to each other. In fact, the cooperativity in the weights that result from training is supposed to mirror the relevant correlations between the monomers in the input, which again are correlated to the prediction task carried out by the network.

The ability of the artificial neural networks to sense correlations between individual sequence positions is very similar to the ability of the human brain when interpreting letters in natural language differently based on their language!naturalcontext. This is well known from pronunciation where, for example, the four a's in the sentence *Mary had a little lamb* correspond to three different phonemes [480]. Another illustration of this kind of ability is shown in figure 6.1. Here the identical symbol will be interpreted differently *provided* the brain receiving the information that is projected onto the retina has been trained to read the English language, that is, trained to understand the sequential pattern in English language!Englishtext.

It is precisely this ability that has made the neural networks successful in the sequence analysis area, in particular because they complement what one can obtain by weight matrices and to some degree also by hidden Markov models. The power of the neural network technique is not limited to the analysis of local correlations, as the sequence information being encoded in the input layer can come from different parts of a given sequence [368]. However, most applications have focused on local and linear sequence segments, such as those presented in the following sections.

Prediction of Protein Secondary Structure

When one inspects graphical visualizations of protein backbones on a computer screen, local folding regularities in the form of repeated structures are immediately visible. Two such types of secondary structures, which are maintained by backbone hydrogen bonds, were actually suggested by theoretical considerations before they were found in the first structures to be solved by X-ray crystallography. There is no canonical definition of classes of secondary structure, but Ramachandran plots representing pairs of dihedral angles for each amino acid residue show that certain angular regions tend to be heavily overrepresented in real proteins. One region corresponds to alpha-helices, where backbone hydrogen bonds link residues i and $i + 4$; another, to beta-sheets, where hydrogen bonds link two sequence segments in either a parallel or antiparallel fashion.

The sequence preferences and correlations involved in these structures have made secondary structure prediction one of the classic problems in computational molecular biology [362, 128, 129, 196]. Many different neural network architectures have been applied to this task, from early studies [437, 78, 262, 370, 323] to much more advanced approaches [453, 445].

The assignment of the secondary structure categories to the experimentally determined 3D structure is nontrivial, and has in most of the work been performed by the widely used DSSP program [297]. DSSP works by analysis of the repetitive pattern of potential hydrogen bonds from the 3D coordinates of the backbone atoms. An alternative to this assignment scheme is the program STRIDE, which uses both hydrogen bond energy and backbone dihedral angles rather than hydrogen bonds alone [192]. Yet another is the program DEFINE, whose principal procedure uses difference distance matrices for evaluating the match of interatomic distances in the protein to those from idealized secondary structures [442].

None of these programs can be said to be perfect. The ability to assign what visually appears as a helix or a sheet, in a situation where the coordinate data have limited precision, is not a trivial algorithmic task. Another factor contributing to the difficulty is that quantum chemistry does not deliver a nice analytical expression for the strength of a hydrogen bond. In the prediction context it would be ideal not to focus solely on the visual, or topological, aspects of the assignment problem, but also to try to produce a more predictable assignment scheme. A reduced assignment scheme, which would leave out some of the helices and sheets and thereby make it possible to obtain close to perfect prediction, could be very useful, for example in tertiary structure prediction, which often uses a predicted secondary structure as starting point.

Prediction of Signal Peptides and Their Cleavage Sites

Signal peptides control the entry of virtually all proteins to the secretory pathway in both eukaryotes and prokaryotes [542, 207, 440]. They comprise the N–terminal part of the amino acid chain, and are cleaved off while the protein is translocated through the membrane.

Strong interest in automated identification of signal peptides and prediction of their cleavage sites has been evoked not only by the huge amount of unprocessed data available but also by the commercial need for more effective vehicles for production of proteins in recombinant systems. The mechanism for targeting a protein to the secretory pathway is believed to be similar in all organisms and for many different kinds of proteins [296]. But the identification problem is to some extent organism-specific, and NN-based prediction methods have therefore been most successful when Gram-positive and Gram-negative bacteria, and eukaryotes have been treated separately [404, 131]. Signal peptides from different proteins do not share a strict consensus sequence—in fact, the sequence similarity between them is rather low. However, they do share a common structure with a central stretch of 7–15 hydrophobic amino acids (the hydrophobic core), an often positively charged region in the N-terminal of the preprotein, and three to seven polar, but mostly uncharged, amino acids just before the cleavage site.

This (and many other sequence analysis problems involving "sites") can be tackled from two independent angles: either by prediction of the site itself or by classifying the amino acids in the two types of regions into two different categories. Here this would mean classifying all amino acids in the sequence as cleavage sites or noncleavage sites; since most signal peptides are below 40 amino acids in length, it makes sense to include only the first 60–80 amino acids in the analysis. Alternatively, the amino acids could be classified as belonging to the signal sequence or the mature protein. In the approach described below, the two strategies have been combined and found to contribute complementary information. While the prediction of functional sites often is fairly local and therefore works best using small windows, larger windows are often needed to obtain good prediction of regional functional assignment.

Applications for DNA and RNA Nucleotide Sequences

Prediction Performance Evaluation

Over the years different means for evaluating the accuracy of a particular prediction algorithm have been developed [31]. Some prediction methods are optimized so as to produce very few false positives, others to produce very few false negatives, and so on. Normally it is of prime interest to ensure, for any type of prediction algorithm, that the method will be able to perform well on novel data that have not been used in the process of constructing the algorithm. That is, the method should be able to generalize to new examples from the same data domain.

It is often relevant to measure accuracy of prediction at different levels. In signal peptide prediction, for example, accuracy may be measured by counting how many *sequences* are correctly classified as signal peptides or non-secretory proteins, instead of counting how many residues are correctly predicted to belong to a signal peptide. Similarly, protein secondary structure may be evaluated at the mean per-chain level, or at the per-amino acid level.

At higher levels, however, the measures tend to be more complicated and problem-specific. In the signal peptide example, it is also relevant to ask how many signal peptide sequences have the position of the cleavage site correctly predicted. In gene finding, a predicted exon can have have both ends correct, or only overlap to some extent. Burset and Guigo [110] have defined four simple measures of gene-finding accuracy at the exon level—sensitivity, specificity, "missing exons", and "wrong exons"—counting only predictions that are completely correct or completely wrong. For secondary structure prediction, this approach would be too crude, since the borders of structure elements (helices and sheets) are not precisely defined. Instead, the segment overlap measure (SOV) can be applied [454, 580]. This is a set of segment-based heuristic evaluation measures in which a correctly predicted segment position can give maximal score even though the prediction is not identical to the assigned segment. The score punishes broken predictions strongly, such as two predicted helices where only one is observed compared to one too small unbroken helix. In this manner the uncertainty of the assignment's exact borders is reflected in the evaluation measure. As this example illustrates, a high-level accuracy measure can become rather *ad hoc* when the precise nature of the prediction problem is taken into consideration.

For the sake of generality, we will therefore focus our attention on single residue/nucleotide assessment measures. For the secondary structure problem, consider an amino acid sequence of length N. The structural data **D** available for the comparison is the secondary structure assignments $\mathbf{D} = d_1, \ldots, d_N$. For simplicity, we will first consider the dichotomy problem of two alternative classes, for instance α-helix versus non-α-helix. In this case, the d_is are in general equal to 0 or 1. We can also consider the case where d_i has a value between 0 and 1, for example representing the surface exposure of amino acids, or the probability or degree of confidence, reflecting the uncertainty of our knowledge of the correct assignment at the corresponding position. The analysis for the multiple-class case, corresponding for example to three states, α-helices, β-sheets, and coil, is very similar. We now assume that our prediction algorithm or model, outputs a prediction of the form $\mathbf{M} = m_1, \ldots, m_N$. In general, m_i is a probability between 0 and 1 reflecting our degree of confidence in the prediction. Discrete 0/1 outputs, obtained for instance by thresholding or "winner-take-all" approaches, are also possible and fall within the theory considered here. The fundamental and general question we address is: How do we assess the accuracy of **M**, or how do we compare **M** to **D**?

A variety of approaches have been suggested in different contexts and at different times and this may have created some confusion. The issue of prediction accuracy is strongly related to the frequency of occurrence of each class. In protein secondary structure prediction the non-helix class covers roughly 70% of the cases in natural proteins, while only 30% belong to the helix class. Thus a constant prediction of "non-helix" is bound to be correct 70% of the time, although it is highly non-informative and useless.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOINFORMATICS

UNIT – IV - SBIA5303 - MACHINE LEARNING FOR BIOINFORMATICS

UNIT 4 HIDDEN MARKOV MODELS Hidden Markov Models: The Theory - Introduction -Prior information & initialization -Likelihood & basic algorithms - Learning algorithms - Applications of HMMs: general aspects -Protein applications - DNA & RNA applications - Advantages & limitations of HMMs – tools.

Hidden Markov Models: The Theory

In the 1990s, only roughly a third of the newly predicted protein sequences show convincing similarity to other known sequences [80, 224, 155], using pairwise comparisons [11, 418]. This situation is even more unfortunate in the case of new, incomplete sequences or fragments. Large databases of fragments are becoming available as a result of various genome, cDNA, and other sequencing projects, especially those producing ESTs (expressed sequences tags) [200]. At the beginning of 1997, approximately half of GenBank consisted of fragment data. Such data cover a substantial fraction, if not all, of the expressed human genome. It is of course of great interest to recognize and classify such fragments, and recover any additional useful information.

A promising approach to improve the sensitivity and speed of current database-searching techniques has been to use consensus models built from multiple alignments of protein families [23, 52, 250, 334, 41, 38]. Unlike conventional pairwise comparisons, a consensus model can in principle exploit additional information, such as the position and identity of residues that are more or less conserved throughout the family, as well as variable insertion and deletion probabilities. All descriptions of sequence consensus, such as profiles [226], flexible patterns [52], and blocks [250], can be seen as special cases of the hidden Markov model (HMM) approach.

HMMs form another useful class of probabilistic graphical models used, over the past few decades, to model a variety of time series, especially in speech recognition [359, 439] but also in a number of other areas, such as ion channel recordings [48] and optical character recognition [357]. HMMs have also earlier been applied to problems in computational biology, including the modeling of coding/noncoding regions in DNA [130], of protein binding sites in DNA [352], and of protein superfamilies [553] (see also [351]). Only since the mid-1990s [334, 41], though, have HMMs been applied systematically to model, align, and analyze entire protein families and DNA regions, in combination with machine-learning techniques.

HMMs are closely related to, or special cases of, neural networks, stochastic grammars, and Bayesian networks. In this chapter we introduce HMMs directly and show how they can be viewed as a generalization of the multiple dice model of chapter 3. We develop the theory of HMMs—in particular the main propagation and machine-learning algorithms—along the lines of chapter 4. The algorithms are used in the following sections where we outline how to apply HMMs to biological sequences. Specific applications are treated in chapter 8, while relationships to other model classes are left for later chapters.

Prior Information and Initialization

There are a number of ways in which prior information can be incorporated in the design of an HMM and its parameters. In the following sections we will give examples of different architectures. Once the architecture is selected, one can further restrain the freedom of the parameters in some of its portions, if the corresponding information is available in advance. Examples of such situations could include highly conserved motifs and hydrophobic regions. Linking the parameters of different portions is also possible, as in the weight-sharing procedure of NNs. Because of the multinomial models associated with HMM emissions and transitions, the natural probabilistic priors on HMM parameters are Dirichlet distributions (see chapter 2).

Initialization

The transition parameters are typically initialized uniformly or at random. In the standard architecture, uniform initialization without a prior that favors transitions toward the main states is not, in general, a good idea. Since all transitions have the same costs, emissions from main states and insert states also have roughly the same cost. As a result, insert states may end up being

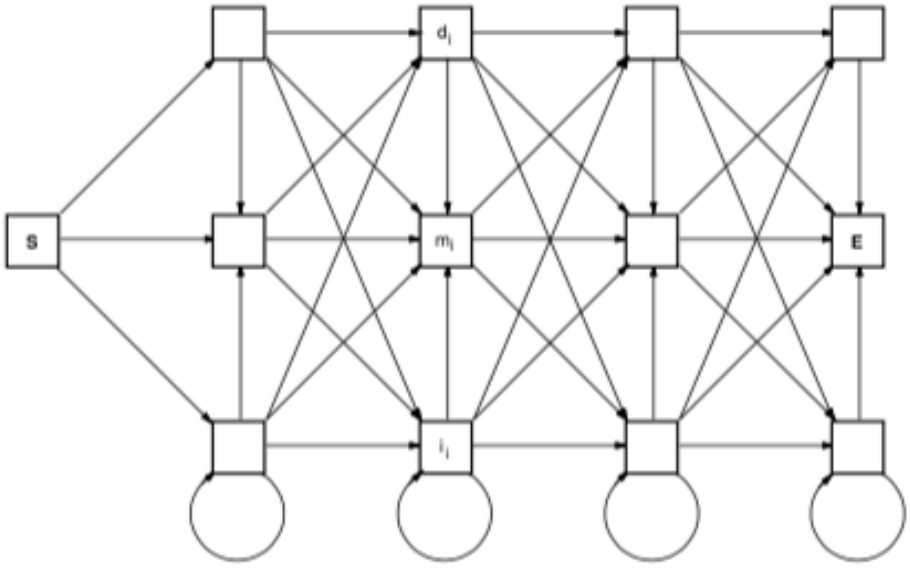


Figure 7.3: Variation on the Standard HMM Architecture. S is the start state, E is the end state, and d_i, m_i, and i_i denote delete, main, and insert states, respectively.

used very frequently, obviously not a very desirable solution. In [41], this problem was circumvented by introducing a slightly different architecture (figure 7.3), where main states have a lower fan-out (3) than insert or delete states (4), and therefore are less costly around the point where transitions out of each state are uniformly distributed. In a similar way, emissions can be initialized uniformly, at random, or sometimes even with the average composition. Any initialization that significantly deviates from uniform can introduce undesirable biases if Viterbi learning (see 7.4.3) is used.

Hidden Markov Models: Applications

Protein Applications

In the case of proteins, HMMs have been successfully applied to many families, such as globins, immunoglobulins, kinases, and G-protein-coupled receptors (see, e.g., [334, 41, 38]). HMMs have also been used to model secondary structure elements, such as alpha-helices, as well as secondary structure consensus patterns of protein superfamilies [187]. In fact, by the end of 1997, HMM data bases of protein families (Pfam) [497] and protein family secondary structures (FORESST) [187] became available. Multiple alignments derived from such HMMs have been reported and discussed in the literature. Large multiple alignments are typically too bulky to be reproduced here. But in most cases, HMM alignments are found to be very good, within the limits of variability found in multiple alignments produced by human experts resulting from diverse degrees of emphasis on structural or phylogenetic information. In the rest of this first half of the chapter, we concentrate on the application of HMMs to a specific protein family, the G-protein-coupled receptors (GCRs or GPCRs), along the lines of [38, 42]. Additional details can be found in these references.

Advantages and Limitations of HMMs

Advantages of HMMs

The numerous advantages of HMMs in computational molecular biology should be obvious by now. HMMs come with a solid statistical foundation and with efficient learning algorithms. They allow a consistent treatment of insertions and deletion penalties, in the form of locally learnable probabilities. Learning can take place directly from raw sequence data. Unlike conventional supervised NNs, HMMs can accommodate inputs of variable length and they do not require a teacher. They are the most flexible generalization of sequence profiles. They can be used efficiently in a number of tasks ranging from multiple alignments, to data mining and classification, to structural analysis

and pattern discovery. HMMs are also easy to combine into libraries and in modular and hierarchical ways.

Limitations of HMMs

In spite of their success, HMMs can suffer in particular from two weaknesses. First, they often have a large number of unstructured parameters. In the case of protein models, the architecture of figure 7.2 has a total of approximately $49N$ parameters ($40N$ emission parameters and $9N$ transition parameters). For a typical protein family, N is on the order of a few hundred, resulting immediately in models with over 10,000 free parameters. This can be a problem when only a few sequences are available in a family, not an uncommon situation in early stages of genome projects. It should be noted, however, that a typical sequence provides on the order of 2N constraints, and 25 sequences or so provide a number of examples in the same range as the number of HMM parameters.

Second, first-order HMMs are limited by their first-order Markov property: they cannot express dependencies between hidden states. Proteins fold into complex 3D shapes determining their function. Subtle long-range correlations in their polypeptide chains may exist that are not accessible to a single HMM. For instance, assume that whenever X is found at position i, it is generally followed by Y at position j, and whenever X′ is found at position i, it tends to be followed by Y′ at j. A single HMM typically has two fixed emission vectors associated with the i and j positions. Therefore, it cannot capture such correlations. Only a small fraction of distributions over the space of possible sequences can be represented by a reasonably constrained HMM.[1] It must be noted, however, that HMMs can easily capture long-range correlations that are expressed in a constant way across a family of sequences, even when such correlations are the result of 3D interactions. This is the case, for example, for two linearly distant regions in a protein family that must share the same hydropathy as a result of 3D closeness. The same hydropathy pattern will be present in all the members of the family and is likely to be reflected in the corresponding HMM emission parameters after training.

Chapters 9 to 11 can be viewed as attempts to go beyond HMMs by combining them with NNs to form hybrid models (chapter 9), by modeling the evolutionary process (chapter 10), and by enlarging the set of HMM production rules (chapter 11).

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOINFORMATICS

UNIT – V- SBIA5303 - MACHINE LEARNING FOR BIOINFORMATICS

UNIT 5 PROBABILISTIC GRAPHICAL MODELS Probabilistic Graphical Models in Bioinformatics - Markov models & DNA symmetries - Markov models & gene finders Hybrid models & neural network parameterization of graphical models -The single-model case – Bi-directional recurrent neural networks for protein secondary structure prediction

Probabilistic Graphical Models in Bioinformatics

The Zoo of Graphical Models in Bioinformatics

High-dimensional probability distributions are one of the fist obstacles one encounters when applying the Bayesian framework to typical real-life problems. This is because the data is high-dimensional, and so are the models we use, often with many thousand parameters and up. High-dimensionality comes also with other so called hidden variables. In general, the resulting global distribution $\mathbf{P}(D, M, H)$ is mathematically intractable and this is where the theory of graphical models comes into play. Using the fact that to a large extent the bulk of the dependencies in the real world are usually local, the high-dimensional distribution is approximated by a product of distributions over smaller clusters of variables defined over smaller spaces and which are tractable [348, 292]. In standard Markovian models, for instance, phenomena at time $t + 1$ may be linked to the past only through what happens in the present at time t. As a result, the global probability distribution $\mathbf{P}(X_1, \ldots, X_N)$ can be factored as a product of local probability distributions of the form $\mathbf{P}(X_{t+1}|X_t)$.

To be more specific, let us concentrate on a particular class of graphical models, namely Bayesian networks [416] (a more formal treatment of graphical models is given in appendix C). A Bayesian network consists of a directed acyclic graph with N nodes. To each node i is associated a random variable X_i. The parameters of the model are the local conditional probabilities, or characteristics, of each random variable given the random variables associated with the parent nodes $\mathbf{P}(X_i|X_j : j \in N^-(i))$, where $N^-(i)$ denotes all the parents of vertex i. The"Markovian" independence assumptions of a Bayesian network

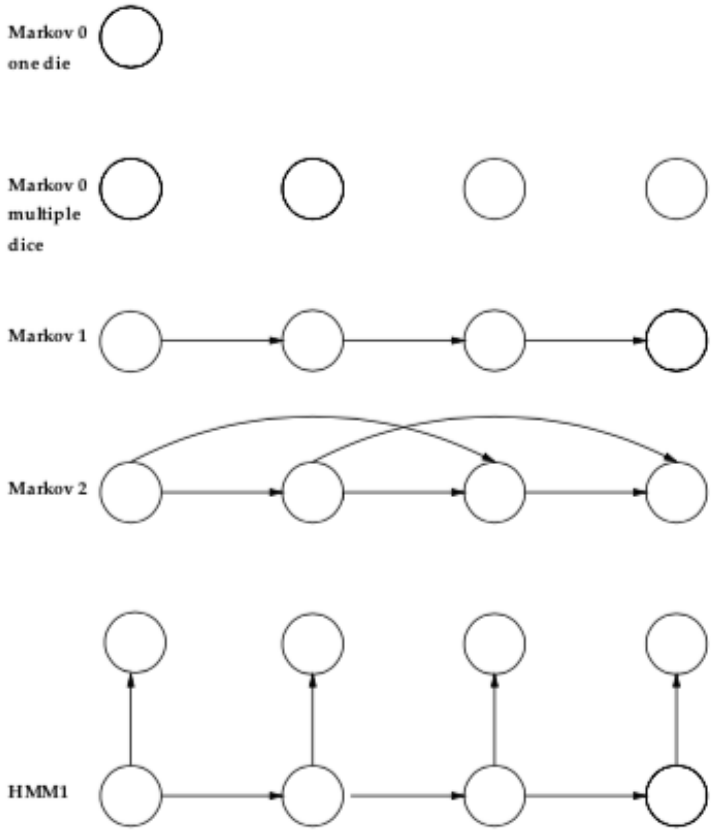


Figure 9.1: Bayesian Network Representation of Markov Models of Increasing Complexity. Markov models of order 0 correspond to a single die or a collection of independent dice. Markov models of order 1 correspond to the standard notion of first order Markov chain. In Markov models of order 2, the present depends on the two previous time steps. All HMMs of order 1 have the same Bayesian network representation given here.

Markov Models and DNA Symmetries

In a piece of double-helical DNA, the number of As is equal to the number of Ts, and the number of Cs is equal to the number of Gs. What appears today as a trivial property in fact was essential in guiding Watson and Crick towards the discovery of the double-helix model in the early 1950s. This property is also known as Chargaff's first parity rule [119]. Chargaff's second parity rule, however, is less known and states that the same thing is approximately true for a piece of *single-stranded* DNA of reasonable size. This rule, first stated in the 1960s [303, 120], has received some recognition in the recent years [430, 185, 231].

The validity of Chargaff's second parity rule can be studied across different organisms, across different kinds of DNA such as coding versus non-coding, and across different length scales. For simplicity, here we look only at genomic DNA in yeast. If we measure the DNA composition of the W and C strands of each chromosome of yeast we find that this composition is remarkably stable and follows Chargaff's second parity rule with approximately 30% for A and T, and 20% for C and G (table 9.1). Notably, the same symmetry is observed in yeast mitochondrial DNA but with a different composition. Likewise, single-stranded genomic DNA in other organisms has a different but still symmetric average composition.

To study the symmetries of double-stranded DNA we count how often each nucleotide occurs on each strand over a given length. These frequencies correspond to a probabilistic Markov model of order 1. It is then natural also to look at Markov models of higher orders (order N) by looking at the statistics of the corresponding N-mers. In particular, we can ask whether Chargaff's second parity rule holds for orders beyond the first, for instance for dinucleotides, equivalent to second-order Markov models.

A DNA Markov model of order N has 4^N parameters associated with the transition probabilities $P(X_N|X_1,\ldots,X_{N-1})$, also denoted $P(X_1,\ldots,X_{N-1} \to X_N)$, for all possible $X_1,\ldots,X_N$ in the alphabet together with a starting distribution of the form $\pi(X_1,\ldots,X_{N-1})$. Because the number of parameters grows exponentially, only models up to a certain order can be determined from a finite data set. A DNA Markov model of order 5, for instance, has 1,024 parameters and a DNA Markov model of order 10 has slightly over one million parameters. Conversely, the higher the order, the larger the data set needed to properly fit the model.

Because of the complementarity between the strands, a Markov model of order N of one strand immediately defines a Markov model of order N on the reverse complement. We say that a Markov model of order N is *symmetric* if it is identical to the Markov model of order N of the reverse complement. Thus a Markov model is symmetric if and only if $P(X_1 \ldots X_N) = P(\overline{X_N} \ldots \overline{X_1})$.

in figure 8.12. The whole model was trained using all types of sequences (known signal peptides and known anchor sequences, as well as cytoplasmic and nuclear sequences). The most likely path through the combined model yields a prediction of which of the three classes the protein belongs to.

In terms of predictive performance in relation to discrimination between signal peptide sequences and nonsignal peptide sequences, the combination of C-score and S-score neural networks (see section 6.4.1) had a discrimination level comparable to that of the HMM. For eukaryotes the networks were slightly better, while for Gram-negative bacteria the HMM was slightly better [406]. For discrimination between cleaved signal peptides and uncleaved signal anchors, the HMM had a correlation coefficient of 0.74, corresponding to a sensitivity of 71% and a specificity of 81%—while the S-score from the neural network could be used to obtain a performance on this task not exceeding 0.4 for the correlation coefficient. The HMM is much better at recognizing signal anchor and therefore at detecting this type of membrane-associated protein.

However, these results should not be taken as a claim that the neural network method is unable to solve the signal anchor problem, since the signal anchors were not included as training data in the neural network model, as was the case for the HMM [406].

A similar approach in the form of a structured HMM has been used to model and predict transmembrane protein toplogy in the TMHMM method [335]. TMHMM can discriminate between soluble and membrane proteins with both specificity and sensitivity better than 99%, although the accuracy drops when signal peptides are present. Due to the high degree of accuracy the method is excellent for scanning entire genomes for detection of integral membrane proteins [335].

Markov Models and Gene Finders

One the most important applications of Markov and graphical models to sequence analysis has been the construction of various gene finders and gene parsers such as GeneMark and GeneMark.hmm [81, 82, 367], GLIMMER [461], GRAIL [529], GenScan [107] and now GenomeScan, and Genie [441]. Our goal here is not to give an exhaustive list of all gene finders, nor to describe each one of them in detail, nor to compare their respective merits and drawbacks,

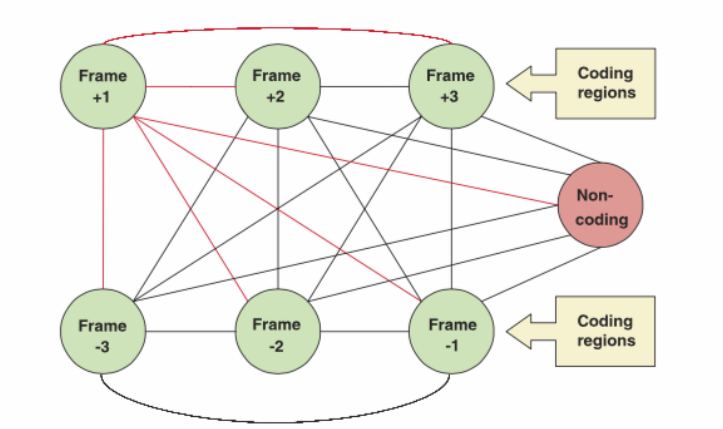


Figure 9.5: Graphical Representation of GeneMark for Prokaryotic Genomes. For prokaryotic genomes, typical high-level modules include modules for coding region and non-coding regions.

but to provide a synthetic overview showing how they can be constructed and understood in terms of probabilistic graphical models.

Integrated gene finders and gene parsers typically have a modular architecture and often share the same basic strategies. They comprise two basic kinds of elementary modules aimed at detecting boundary elements or variable length regions. Examples of boundary modules associated with localized signals include splice sites, start and stop codons, various transcription factor and other protein binding sites (such as the TATA-box), transcription start points, branch points, terminators of transcription, polyadenylation sites, ribosomal binding sites, topoisomerase I cleavage sites, and topoisomerase II binding sites. Region modules instead are usually associated with exons, introns, and intergenic regions. Exon models in turn are often subdivided into initial, internal, and terminal exons due to the well-known statistical dif-

Hybrid Models and Neural Network Parameterization of Graphical Models

The General Framework

In order to overcome the limitations of HMMs, we shall look here at the possibility of combining HMMs and NNs to form hybrid models that contain the expressive power of artificial NNs with the sequential time series aspect of HMMs. In this section we largely follow the derivation in [40]. There are a

number of ways in which HMMs and NNs can be combined. Hybrid architectures have been used in both speech and cursive handwriting recognition [84, 126]. In many of these applications, NNs are used as front-end processors to extract features, such as strokes, characters, and phonemes. HMMs are then used in higher processing stages for word and language modeling[1]. The HMM and NN components are often trained separately, although there are some exceptions [57]. In a different type of hybrid architecture, described in [126], the NN component is used to classify the pattern of likelihoods produced by several HMMs. Here, in contrast, we will cover hybrid architectures [40] where the HMM and NN components are inseparable. In these architectures, the NN component is used to reparameterize and modulate the HMM component. Both components are trained using unified algorithms in which the HMM dynamic programming and the NN backpropagation blend together. But before we proceed with the architectural details, it is useful to view the hybrid approach from the general probabilistic standpoint of chapter 2 and of graphical models.

The Single-Model Case

Basic Idea

In a general HMM, an emission or transition vector θ is a function of the state i only: $\theta = f(i)$. The first basic idea is to have a NN on top of the HMM for the computation of the HMM parameters, that is, for the computation of the function f. NNs are universal approximators, and therefore can represent any f. More important perhaps, NN representations of the parameters make possible the flexible introduction of many possible constraints. For simplicity, we discuss emission parameters only in a protein context, but the approach extends immediately to transition parameters as well, and to all other alphabets.

In the reparameterization of (7.33), we can consider that each of the HMM emission parameters is calculated by a small NN, with one input set to 1 (bias), no hidden layers, and 20 softmax output units (figure 9.9A). The connections between the input and the outputs are the w_{iX} parameters. This can be generalized immediately by having arbitrarily complex NNs for the computation of the HMM parameters. The NNs associated with different states can also be linked with one or several common hidden layers, the overall architecture being dictated by the problem at hand (figure 9.9B). In the case of a discrete alphabet, however, such as for proteins, the emission of each state is a multinomial distribution, and therefore the output of the corresponding network should consist of $|A|$ normalized exponential units.

Bidirectional Recurrent Neural Networks for Protein Secondary Structure Prediction

Protein secondary structure prediction (see also section 6.3) can be formulated as the problem of learning a synchronous sequential translation from strings in the amino acid alphabet to strings written in the alphabet of structural categories. Because biological sequences are spatial rather than temporal, we have seen that BIOHMMs are an interesting new class of graphical models for this problem. In particular, they offer a sensible alternative to methods based on a fixed-width input window. The expressive power of these models enables them to capture distant information in the form of contextual knowledge stored into hidden state variables. In this way, they can potentially overcome the main disadvantage of feedforward networks, namely the linear growth of the number of parameters with the window size. Intuitively, these models are parsimonious because of the implicit weight sharing resulting from their stationarity; i.e., parameters do not vary over time.

We have used BIOHMMs directly to predict protein secondary structure with some success [36]. As graphical models, however, BIOHMMs contain undirected loops and therefore require a computationally intensive evidence-propagation algorithm (the junction tree algorithm [287]), rather than the simpler Pearl's algorithm for loopless graphs such as HMMs (see also appendix C). Thus to speed up the algorithm, we can use the technique of the previous section and use neural networks, both feedforward and recurrent, to reparameterize the graphical model.