Doctoral Thesis

Designing Kernels for Biological Sequence Data Analysis

by

Taishin KIN

Supervised by Kenji Satou, Ph.D.

School of Knowldge Science
Japan Advanced Institute of Science and Technology

Dec. 2003

Abstract

In order to uncover the nature of life, it is very important to understand the nature of biological sequences such as DNAs, RNAs or proteins. Our research focuses on a fundamental issue of modeling and comparison of biological sequence data in general. The objectives of this research are to propose a general framework where modeling and comparison of biological sequence data can be organized and to develop efficient methods to extract features of biological sequence data.

Firstly, we proposed *the Self-Identification Learning* (SIL) for a system based on hidden Markov models to predict protein coding regions. SIL is a learning algorithm that does not require training data. By making use of its prediction results for its training data in the next iteration, it trains itself through iterative feedback loop of learning and prediction. Whereas existing genefinding systems are not useful when there are insufficient training data of a target organism because a high quality training dataset is indispensable to perform accurate predictions, SIL allows to perform genefinding in such a case.

Many of genefinding systems uses a discriminative property known as *dicodon usage measure* (DUM), one of the best measures that distinguishes protein coding regions and non-coding regions. It has been widely believed that the biological meanings of DUM can be decomposed into several biological properties while such belief is not backed by any objective examination. We found that a portion of dicodon usage parameters suffices to yield reasonable performance for prediction of coding regions. Hinted by this indication of redundancy of DUM, we devised six dicodon approximators based on some combinations of biological properties because a good dicodon approximator will lead to understanding biological meanings of dicodon and to a good basement for designing a better prediction measure. We carried out performance comparisons among DUM and the six approximators by using 17 microbial plus 6 eukaryotic genomic sequence data. However, no approximators could match performance of DUM. Thus dicodon usage cannot be interpreted with the approximators we devised. This result revokes conventional belief on the biological meanings of DUM.

Use of DUM is limited to discriminating coding and non-coding regions. Therefore we started to find a general method to extract features of sequences by thinking two essential issues of biological sequence data analysis i.e. “what is the feature of biological sequence?” and “how we can utilize such features for analysis?” We proposed a novel method to extract features of biological sequence data, *the marginalized kernel*, that is a general framework to design similarity measures among biological sequences. There are two properties dealt with this framework: feature representation and similarity quantification. We use latent variable models (e.g. hidden Markov models) for the feature representations, which allows to bind a hidden variable to a certain biological feature. The hidden variables can be estimated with regular algorithms. The highlight of our method is that we use all probable estimations which allow us to incorporate implicit feature representations. We use kernel functions for similarity quantification so that we can exploit kernel methods such as support vector machines for discriminant analysis. In order to evaluate validity of our method, we performed computational experiments to classify *gyrB* protein sequence data. The experiments show that our method successfully classified most of the proteins.

We developed a novel method: *the marginalized kernel for RNA sequence data*, which defines similarities between RNA sequences by utilizing stochastic context free grammar (SCFG). With our method, powerful multivariate analysis tools such as support vector machines, kernel PCA and etc. become available to RNA sequence data analysis. We demonstrated performance of our method with clustering experiments by using kernel PCA and supervised classification experiments by using support vector machines. The experiments show promising results.

Acknowledgment

Firstly, I would like to put my best appreciation to my family. Dr. Satou is a gracious person who should be listed here in the first place. I would like to express my profound appreciation to Dr. Konagaya for his thickest support to my student life and research activity. I also greatly appreciate Dr. Asai and Dr. Tsuda for their remarkable criticism and contribution to my research. Dr. Takahashi has been giving me coherent and practical advise to carry my research activity on. Dr. Akiyama gave me a precious opportunity to continue and expand my research activity. All of students of Genetic Knowledge System Laboratory of Japan Advanced Institute of Science and Technology has been of the best assistance in every aspect of my student life. I thank them from my heart profoundly. Let me express my special appreciation to Mr. T. Onishi and Mr. and Mrs. Unto.

To my grand father...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides some essential backgrounds of *computational biology* that is an emerging research area of biology that applies or develops computational theories or methods in order to solve biological conundrums. Though there are many research topics regarding this area, this chapter mainly focuses on biological sequences such as DNA, RNA and proteins.

1.1 Molecular Biology

1.1.1 Cell

A human is a living organism that consists of 60 billion (6.0×10^{13}) animal cells. Although the human is such a complex lifeform, one single cell is thought to be an *unit* of life because, in this world, there are the most primitive living organisms which consist of only one cell. Such the organisms are called *unicellular organism* which embrace the great kingdom of bacteria and fungi. On the other hand, we human beings, as well as plants, fishes, reptiles and insects, are called *multicellular organism*. However, the unicellular/multicellular separation is not a good criterion in terms of evolution. Because there are great deal of differences between bacteria and fungi wherease the both organisms are unicellular. The differences are presented inside of their cells. Bacteria cells always lack certain organella which are found in fungal cells. Those are nuclei and cytoskeleton. A nuclei is a spherical membrane containing many complex molecules including chromosomes that will be discribed later. A cytoskeleton is a scafold that supports and maintains physical formation of a cell. The bottom line is that there are two types of cells: one is called prokaryotic cell which lacks nuclei and cytoskeleton, and the other is called eukaryotic cell which encompasses a fungal cell and multicellular organism cells. Provided that, evolutionary sound classification of living organisms are based on the class of cell whether it is prokaryotic or eukaryotic. Thus we classify every organism into prokaryote or eukaryote (the former has further differentiation i.e. eubacteria and archaea).

No matter which class a cell belongs, each cell has one or more chromosomes inside. A chromosome is a chain of chromatins; a compound of DNA (deoxyribo-nucleic acid) and protein molecules. Nature of a chromatin is a chain of DNA winded around a reel-like protein which is called histon octamer. In 1944, DNA is found to be a chemical polymer which consists of four types of nucleic acids; adenine (A), guanine (G), cytosine (C) and thymine (T) and, much more importantly, carries genetic information.

1.1.2 Double Helix

In 1953, J.D. Watson and F.H.C. Crick found that DNA is a double-helical strand of nucleic acids. A part of DNA looks like following illustration:

$$\begin{array}{ccccccccc} 5' & \mathrm{A} & \mathrm{C} & \mathrm{C} & \mathrm{T} & \mathrm{G} & \mathrm{A} & \cdots & 3' \\ & | & | & | & | & | & | & & \\ 3' & \mathrm{T} & \mathrm{G} & \mathrm{G} & \mathrm{A} & \mathrm{C} & \mathrm{T} & \cdots & 5' \end{array}$$

$5'$ and $3'$ are labels indicate direction of DNA strand [1]. DNA double helix has two strands which are essentially same but are aligning reverse complementary order each other. The two complementary strands; one from $5'$ to $3'$ and the other is running reverse direction. The bar symbols indicate hydrogen bonds for base pairs: A–T and G–C (Watson-Click pairs) between two strands.

1.1.3 Transcription and Splicing

Certain parts of DNA are copied to RNA (ribo-nucleic acid) called messenger RNA (mRNA) through a series of complex molecular reactions called "transcription." A region transcribed to mRNA is called *gene*[2], which is the very region that encodes genetic information. However, not all the region has genetic information. After transcription, mRNAs are processed via *splicing* that is a complex process involving several proteins called spliceosomes. Spliceosomes "cut out" (splice) certain parts of mRNA and concatenate remaining parts. The cut-out parts are called *introns*, and the remaining parts are called *exons*. Only eukaryotic genes have exon–intron structure. Since introns are spliced from mRNAs then resolved and recycled as individual nucleic acids, they have not particular genetic information. Spliced mRNA is called mature mRNA while pre-mRNA is used for distinguishing mRNA before spliced.

1.1.4 Translation

After splicing, some of mRNAs are used for protein synthesis called "translation." Protein; a chain of amino acids is synthesized according to information written in mRNA. In translation, mRNA is treated as a chain of triplets:

$$\begin{array}{lccccccc} \mathrm{mRNA} & \underbrace{\mathrm{UUU}} & \underbrace{\mathrm{UAC}} & \underbrace{\mathrm{UGC}} & \underbrace{\mathrm{GGC}} & \mathrm{C} & \cdots \\ \mathrm{protein} & \mathrm{Phe} & \mathrm{Try} & \mathrm{Cys} & \mathrm{Gly} & & \cdots \end{array}$$

A triplet of nucleic acids in mRNA corresponds to a certain amino acid (see Figure. 2.1 for details). The correspondence is called *genetic code* (see Table. 2.1). Some other genes work as functional RNAs (fRNAs) without being translated.

1.2 Human Genome Project

Genome is a word stands for all of genetic information of DNA letters contained in a whole set of chromosomes. Human Genome Project (HGP) is a join effort of international researchers to

[1] These labels are originated with positions of chemical bonds in nucleic acids.

[2] Also a region that controls transcription is included.

determine every bit of human genome that consists of 3 billion letters (3.2 giga base) of AGTC. In April 2003, HGP announced completion[3] of their ten-year-long effort. The achievement of HGP is significant because human genome data is the most important and fundamental resource for scientific researches: genetics – clarifying evolutionary root of *homo sapiens*, pathology – finding mechanisms of diseases including cancers that involve a set of genes, pharmacology – drug discovery and design based on findings, molecular biology – genome comparison that unveils "what differentiates human nature from another?" and many other researches. According to HGP (press release for draft human genome from The Welcome Trust Sanger Institute), only 1.1% to 1.4% of human genome encodes protein-coding regions. There are approximately 31,000 to 34,000 protein-coding genes in human genome that is 28% of genes transcribed to mRNAs. These estimations indicates that human genome has quite generous redundancy because most part of chromosomes do not convey any evolutionary significant information. Nonetheless, since genome is very long text of AGCTs, there is no obvious boundaries to discriminate genes and non-genes. Therefore, intensive analysis of genome sequence data is required to further investigate nature of human genome. Such analysis usually requires computational power because one needs to deal with large amount of information processing.

1.3 Analysis of Biological Sequence Data

Here we use biological sequence data to refer to DNA, RNA, genome or protein sequence data. DNA/RNA sequence is a molecular chain of nucleic acids (A, G, C, T or U). Genome sequence is also a chain of AGCTs. Protein sequence is a chain of amino acids. There are 20 admissible amino acids in a chain. We add "data" to refer to digital form of these sequences. Thus genome sequence data is a set of digital files storing letters of genome sequence. There are varieties of analysis regarding biological sequence data. In this thesis, we focus on two major research area that play very important role in genome sequence data analysis. They are sequence similarity and gene finding.

1.3.1 Gene Finding

Gene finding is a computational method to find protein coding regions out of genomic sequences and it has been studied extensively for nearly a couple of decades. Methods for gene finding are roughly divided into two categories:

- Sequence similarity search
- Stochastic models based on statistical regularities in coding region; *coding measures*

Sequence similarity search is one of the earliest but the most valuable methods for gene finding. A protein coding region can be identified by finding regions show high similarity to a queried protein sequence which is available in databases such as Swiss-Prot [73], GenBank [11] and DDBJ [69]. Methods for similarity search are quite well developed [105] and realized as a variety of software tools such as Blast [2], Fasta [78], ALN [41], BLAT [57] and so on. The reason why sequence similarity search works for gene finding is originated from a mechanism of molecular evolution. The mutation is one of the major factor to cause genomic diversity among all living organisms, which is a phenomenon that cause modifications to genomes such

[3] Here, "completion" means that 99% of human genome was determined at 99.99% of accuracy.

as deletion, insertion and substitution of a nucleotide. A myriad of mutations are piled up on a genome through hundreds of millions of years across tens of thousands of generations. Naturally, mutations are very rare in coding regions because a mutation can often cause critical dysfunction of a gene which may, in turn, lead the host organism to death. Consequently, many living organisms started to have a certain mechanism to prevent and correct "errors" in coding regions which are caused by random mutations. A clear advantage of similarity search is that it allows identifying function of detected coding region as well as its position when function of the matching protein is annotated in advance.

Gene finding using stochastic models is useful to detect coding regions where no similar protein sequences are available in databases. The models are designed to integrate molecular biological attributes and signal patterns that help detecting specific feature of coding regions. Since initial breaking of genetic code by Nirenberg and Matthaei in 1961, molecular biology has been clarified many details of genomes that help designing theoretical models of coding regions. Some of the well known features of coding regions are codon usage and GC content. Both of them can be observed as compositional biases particularly in coding regions, which are consequences of the evolutionary pressure [74]. Since the early initiation of computational biology, a bunch of gene models have been developed. It was straight forward to use such attributes for defining probabilistic models to recognize coding regions. Early studies on gene finding by Shepherd [92], Fickett [33] and Staden & McLachlan [96] showed that statistical criteria based on compositional biases of amino-acids and codon usage can be used to identify coding regions. More details about gene finding can be found in summaries of gene finding problems contributed by Fickett [34]. Evaluation of several gene finding programs was offered by Burset and by Guigo [21].

1.3.2 Hidden Markov Model

Wide variety of gene finding algorithms based on machine learning approach have been developed since Stormo and et al. [98] first introduced an algorithm using neural networks for detection of translation start sites. Most of all gene finding programs make use of the high level syntax of genes resulting from general understandings of transcription, splicing, and translation [34]. Searls [90] suggested that a linguistic approach to the analysis of features in DNA sequences could be beneficial. This approach is first applied to identification of protein coding region by Dong and Searls [29], where a formal definite clause grammar of genes is described.

Hidden Markov model (HMM) [80] is one of the major stochastic language model and has been widely used for natural language processing. Application of HMM to biological sequence analysis was first introduced independently by Churchill [22], Asai et al. [4], Krogh et al. [60] and Dong & Searls [29]. HMM provides several advantages such as flexible description of signal patterns and virtually direct translation of biological attributes to HMM network. That is why HMM are widely used for biological sequence data analysis ([52, 107, 106]).

1.4 Outline of the Thesis

The objective of this research is to develop novel methods to extract features of biological sequence data in general. Following chapters of this thesis are organized as follows: Firstly, in chapter 2, we investigated a discriminative property known as *dicodon usage*, the best discriminative measure that helps distinguishing protein coding regions and non-coding regions.

However, its biological semantics is not clarified yet. Aim of this investigation is to clarify biological meanings of the dicodon usage. We devised several dicodon approximators based on some combinations of apparent biological properties in order to emulate dicodon usage. The investigation was carried out fairly comprehensively by using seventeen microbial plus several eukaryotic sequence data. Although some approximators sometimes scored as good as dicodon does, no approximators could match performance of dicodon usage. Therefore dicodon usage cannot be interpreted with the approximators devised here. Hence biological meanings of dicodon usage is yet to be known. Even though dicodon usage is still the best player in the field of practical gene finding, its use is limited to provide statistical criteria for discriminating coding and non-coding regions. Therefore we started to find a method to extract features of biological sequence data from a fundamental point of view.

Secondly, in chapter 3, we proposed a novel method to extract features of biological sequence data, that is a general framework to design similarity measures. This provides a framework to define similarity measures that take account of biological features implied by sequence data. There are two properties that should be defined in this framework: feature representation and similarity quantification. The feature representation is to define a model to represent features (e.g. secondary/tertiary structures of proteins or exon–intron boundaries of DNAs) of biological sequences. In this framework, we use latent variable model (e.g. hidden Markov models) for the feature representations, which allows us to bind a hidden variable to a certain feature. The hidden variables can be estimated in a probabilistic manner such as maximum likelihood or expectation–maximization from a biological sequence data through a latent variable model. Instead of using only one best estimation (e.g. Viterbi result of HMMs) which cannot be proven to be true, we use all probable estimations that allow us to incorporate implicit feature representations. The similarity quantification is to quantify the similarity between two biological sequences. We use kernel functions for this purpose so that we can exploit kernel methods such as support vector machines for discriminant analysis. In order to evaluate validity of our method, we performed computational experiments to classify *gyrB* protein sequence data. The experiments show that our method successfully classified most of the proteins.

In chapter 4, We developed a novel method to define similarities between RNA sequence data by utilizing stochastic context free grammar, that is capable of describing secondary structures of RNAs, into our framework. We demonstrate performance of our method with clustering experiments of human transfer RNAs (tRNA) by using kernel PCA. The experiments show promising results. We apply our method to more practical problem that is to extract features of small nucleolar RNAs (snoRNA). snoRNAs are small RNAs that play an important role in a splicing reaction of ribosomal RNA precursors. The major difficulty of feature extraction from snoRNAs is that their common secondary structures are not known well. However, the experiments show that our method successfully captured features of snoRNAs.

We summarize and put conclusion for this thesis in chapter 5.

Chapter 2

A Study on Dicodon-oriented Genefinding using Self-Identification Learning

2.1 Introduction

In this chapter we focus on one of the major open problems of computational biology, that is prediction of protein coding region in genomic sequences (commonly known as ”genefinding,” ”gene prediction” or ”gene identification”) of every species ranging from prokaryotes (mainly bacteria) and eukaryotes (such as yeast, plant, worm, and human). Because, the clarification of biological functions of genes has been one of the primary goals of computational biology. The genefinding is very important for the functional clarification of genes and complete automation of the genefinding is also demanded because a large quantity of genomic sequence data is being piled up on a host of databases and waiting to be analyzed. Such incessant increase of demands makes the genefinding more crucial.

Main stream of the genefinding has been targeting higher eukaryotic (especially human) genomic sequences which have more complicated genomic structure thus more challenging to predict than prokaryotic genome. The genefinding with higher eukaryotic genomic sequences requires precise definition of the sequence dependence of molecular biological mechanisms such as:

- the basic biochemical processes of DNA to RNA transcription
- RNA translation and splicing(i.e. exons and introns)
- knowledge about the sequence properties of known genes

Although the above mechanisms have been under intensive investigation, heaps of knowledge are still waiting to be discovered. So the gene finding has been a computational and analytical method to full fill the void of knowledge on these mechanisms. While the prediction problem is hard and challenging, it increases its importance regarding recent shift in emphasis of the Human Genome Project from reading every nucleotide of human genomic sequence to finding functional role of every genes. In order to get clues to find the functions of genes, we need to find exactly where those genes reside in a lengthy sequence of nucleotides with aids of computational methods.

This chapter emphasizes precision modeling of genomic sequences; especially discrimination of protein coding/non-coding regions based on the dicodon usage measure which has

been known as one of the most precise among protein coding measures. Although Fickett and Tung gave an objective and quantitative evidence to the superiority of the dicodon usage measure [35], there has been no sufficient investigation taken for the dicodon usage measure to clarify the biological background to explain why the dicodon works such well. This chapter aims to investigate and clarify the biological semantics of the dicodon usage measure.

2.1.1 The Dicodon Usage Measure

This chapter focuses on the statistical regularities in coding regions, where the dicodon usage measure should be discussed. Fickett and Tung evaluated every coding measures known to the public and showed that the dicodon usage measure is one of the best measures among others [35]. Every protein coding region is translated from nucleotides to amino-acids, in a triplet basis, under a rule of genetic code. The triplet is called Codon. Figure 2.1 shows how the translation occur in the molecular world. It is well known that the occurrence of codon has peculiar bias which means not every codon is used evenly in a genomic sequence and thus the codon can be used as a measure of coding region. Such unevenness is called *codon usage* and denoted as a conditional probability:

$$p(c|A(c)) = \frac{f(c)}{\sum_{c\prime \in \{c' \,|A(c')=\alpha\}} f(c')},$$

where c is for a codon, $A(c)$ is for an amino-acid corresponds to c and α stands for an amino-acid. Table 2.1 to 2.1.1 show differences between coding and non-coding region for E. coli.

Although the codon usage measure offers simple description for coding regions, it just produces lower scores (specificity and sensitivity) than other measures such as dicodon usage measure [35] (see also Table 2.1.1). The measures that perform better than the codon usage measure belong to hexamer-n measure. The hexamer-n measure (for n = 0, 1, 2) counts all hexamers (i.e. six nucleotide) offset by n from the starting base. Dicodon usage measure is identical to hexamer-0 measure. Hexamer-1 and 2 measures perform slightly worse than dicodon usage measure. Dicodon usage measure can be denoted as a conditional probability $p(c_{i+1}|c_i)$ where c_i for a codon and c_{i+1} for its next codon.

The simple calculation shows that the codon usage measure has 1,220 parameters for 61 codons and 20 amino-acids, and the dicodon usage measure has 3,721 parameters. Notice that the dicodon usage measure performs slightly better than the codon usage measure that has only one-third of the parameters. This simple fact implies that the dicodon usage measure is more redundant than the genefinding actually requires. Besides, our preliminary examination (explained later) indicated the same conclusion. Fickett stated that the dicodon usage or hexamer-n measure contains all of other known measures such as codon usage, diamino-acid, and dinucleotide bias [35] (see also Figure 2.2). According to the redundancy indicated above, it is reasonable that not all of these measures does not need to be included by the dicodon usage measure. This thesis focuses on this very point and tries to clarify which measure is the most important and which is the least important.

2.1.2 Self-identification Learning Method

Self-identification learning [6, 7] is relatively new approach that does not require training sequence while most other algorithms require the training sequence. Conventional learning scheme is trained by training sequence in order to obtain optimum set of parameters. However,

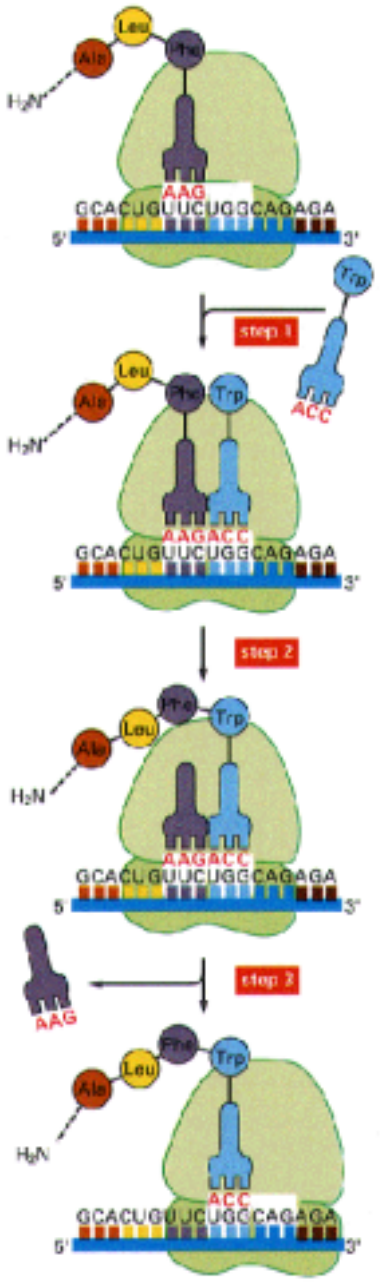


Figure 2.1: Translation process of mRNA is shown. The elongation phase of protein synthesis on a ribosome. The three-step cycle shown is repeated over and over during the synthesis of a protein. An aminoacyl-tRNA molecule binds to the A-site on the ribosome in step 1, a new peptide bond is formed in step 2, and the ribosome moves a distance of three nucleotides along the mRNA chain in step 3, ejecting an old tRNA molecule and ”resetting” the ribosome so that the next aminoacyl-tRNA molecule can bind.

Table 2.1: Codon usage (a): Differences of codon usages between coding (CD) and non-coding (NC) region in E.coli (a). Notice that codon usages in coding regions are heavily biased or apparently different from the non-coding regions.

Amino Acid	CODON	USAGE(CD)	USAGE(NC)
Ala	GCA	0.213	0.288
	GCC	0.270	0.241
	GCG	0.356	0.258
	GCT	0.161	0.213
Arg	AGA	0.039	0.177
	AGG	0.023	0.176
	CGA	0.065	0.132
	CGC	0.398	0.190
	CGG	0.098	0.168
	CGT	0.378	0.157
Asn	AAC	0.550	0.397
	AAT	0.450	0.603
Asp	GAC	0.372	0.358
	GAT	0.628	0.642
Cys	TGC	0.556	0.505
	TGT	0.444	0.495
Gln	CAA	0.347	0.518
	CAG	0.653	0.482
	GAA	0.689	0.622
	GAG	0.311	0.378
Gly	GGA	0.109	0.252
	GGC	0.403	0.284
	GGG	0.151	0.222
	GGT	0.337	0.243
His	CAC	0.429	0.406
	CAT	0.571	0.594
Ile	ATA	0.073	0.325
	ATC	0.420	0.256
	ATT	0.507	0.419
Leu	CTA	0.037	0.093
	CTC	0.104	0.116
	CTG	0.496	0.169
	CTT	0.104	0.168
	TTA	0.131	0.264
	TTG	0.128	0.191
Lys	AAA	0.765	0.680
	AAG	0.235	0.320
Met	ATG	1.000	1.000

Table 2.2: Codon usage (b): Codon Usage differences between coding and non-coding region in E.coli (b) (continued).

Amino Acid	CODON	USAGE(CD)	USAGE(NC)
Phe	TTC	0.426	0.328
	TTT	0.574	0.672
Pro	CCA	0.191	0.241
	CCC	0.124	0.224
	CCG	0.525	0.258
	CCT	0.159	0.277
Ser	AGC	0.277	0.161
	AGT	0.151	0.154
	TCA	0.124	0.239
	TCC	0.149	0.151
	TCG	0.154	0.125
	TCT	0.146	0.171
Thr	ACA	0.132	0.308
	ACC	0.434	0.219
	ACG	0.268	0.239
	ACT	0.166	0.234
Trp	TGG	1.000	1.000
Tyr	TAC	0.431	0.372
	TAT	0.569	0.628
Val	GTA	0.154	0.238
	GTC	0.216	0.190
	GTG	0.371	0.233
	GTT	0.259	0.340

Table 2.3: Performances of coding region measures are shown. Percentage accuracy (average of specificity and sensitivity) of the coding measures in predicting phase-specific coding (excerpt from [35] Table 3).

Measure	Human 54 Penrose	Human 108 Penrose	Human 162 Penrose	E.coli 54 Penrose	Human 54 Classical
Dicodon Usage (Hexamer-0)	80.7	84.3	85.4	88.7	-
Hexamer-2	79.5	82.8	84.2	87.2	-
Hexamer-1	78.6	82.0	83.3	87.1	-
Codon Usage	78.0	81.0	82.1	86.9	81.7
Diamino-acid Usage	77.2	84.9	87.7	84.2	-
Amino-Acid Usage	75.3	81.1	83.6	83.3	76.2

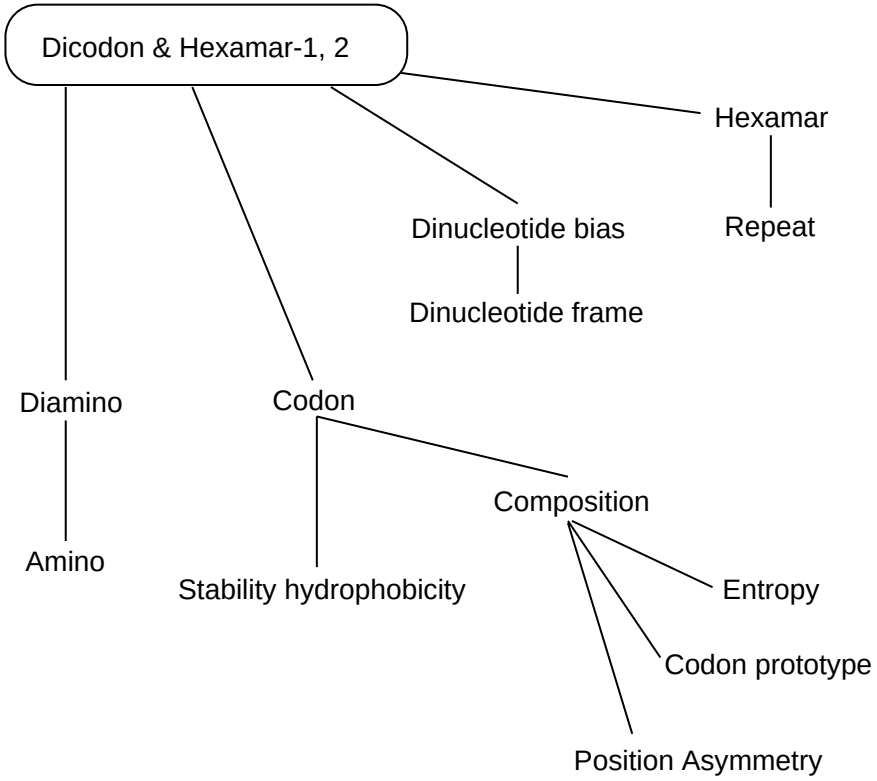


Figure 2.2: Derivability of coding measures: Each measure is derivable from any measure above it and connected to it by a line (excerpt from [34] Figure 1).

such strategy becomes totally impossible when there is no datum available for the training and, possibly in many cases, such circumstances can be arisen especially for practical gene finding where we can not expect to have correct data in advance. For example, practical genefinding often requires gene prediction against totally new species –which means there is no previously acquired similar or phylogenicaly related genomic sequence data– therefore no training data can be effective if not offered. Besides, the self-identification learning can directly reflect data specific attributes to its output although the generic learning scheme tends to treat such attributes as noise. This feature of the self-identification learning is very important especially for gene finding that is applied to new, thus previously unknown, species. Because such attributes are essential for genefinding against the new species and the generic learning scheme with training data, which obviously do not include new data, usually fails identify coding regions in such new species. The self-identification learning can obtain optimal parameters without training data in following way(see also Figure 2.3):

- It simply starts its learning with uniform learning parameters
- The first trial finds several coding regions with uniform initial parameters
- Re-calculate parameters(i.e. dicodon usages) according to the regions found
- Iterate learning with revised parameters until it reaches plateau of learning curve

Efficiency of the self-identification largely depends, by its nature, on the number of its learning parameters as well as the size of training data. When it employs a large set of parameters, it requires a large set of training data. The model is not accurate with insufficient training data.

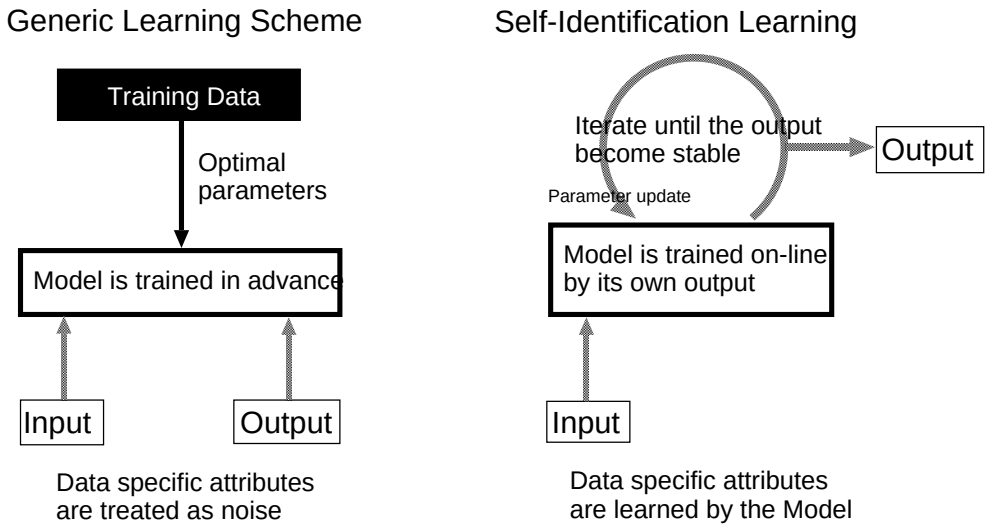


Figure 2.3: Side-by-side comparison between generic learning scheme and self-identification learning. Notice that the generic learning scheme needs training data which is based on previously acquired data although the self-identification learning does not need such data. Thus the self-identification learning reflects data specific attributes to its output while such attributes are treated as noise in the generic learning scheme.

On the other hand, the model is not accurate when the number of parameters is too large for the amount of training data. This problem can be generalized as a problem of complexity and accuracy of a model. Hence we have to consider trade-off between the complexity of the model and the accuracy.

In this chapter, we fed short fragments of microbial genomic sequence data to our genefinding system in order to evaluate the robustness of the self-identification learning against short training data.

2.2 Evaluation of Self-identification Learning

Two examination/analysis were performed. The first is computational genefinding using dicodon-oriented HMM with self-identification learning and the second is evaluation of dicodon usage measure. The former provides reason of the latter evaluation that is the reason to ask what make dicodon usage measure such redundant. Firstly, evaluation of self-identification learning is provided in this section.

2.2.1 Method

We used a dicodon oriented HMM genefinding system with self-identification learning [6] as a test-bed for the evaluation. Two objectives are set and they are:

- to purely evaluate gene identification accuracy for our system
- to evaluate robustness of self-identification learning against short training data

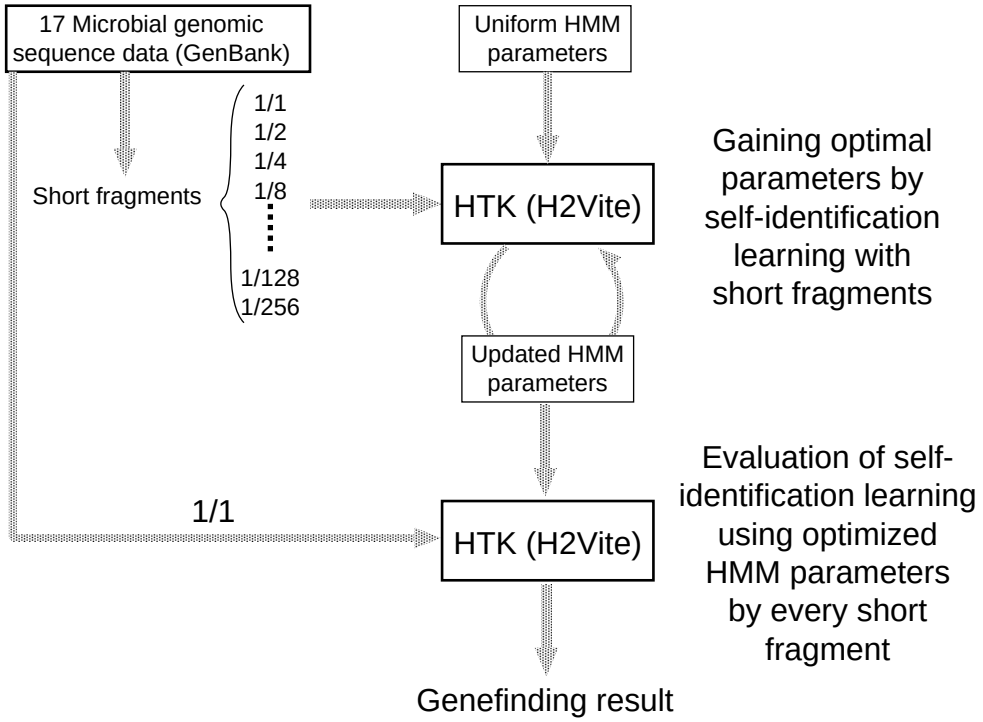


Figure 2.4: A brief overview of the genefinding examination to evaluate self-identification learning.

System Overview

We built a genefinding system that is incorporated with HTK (HMM Tool Kit) [109] which is a commercial(Entropic Inc.) software toolkit for building continuous density HMM based speech recognizers. Although the HTK is designed for dealing with continuous density distributions, the differences are minor between the continuous and discrete probability distributions. Therefore HTK offers seamless platform to the gene finding. HTK uses Baum-Welch algorithm(a.k.a. Expectation-Maximization algorithm) [9] for learning its parameters, and uses Viterbi algorithm [50] for coding region recognition. Figure 2.4 provides at-a-glance overview of our genefinding examination. Actually, by nature of our evaluation method, we used only H2Vite which is a part of the toolkit and provides coding region recognition alone with Viterbi algorithm. For the examination, we used 17 microbial complete genomic sequence data [59, 27, 38, 61, 97, 13, 36, 101, 37, 17, 45, 93, 23, 56, 52, 39, 3, 32] available from GenBank [11] (see Appendix A). We evaluated the data size dependency of the self-identification learning with short fragments of the sequence data such as 1/2, 1/4, 1/8, ..., and 1/256 of complete sequence.

Dicodon Oriented HMM

Figure 2.5 shows overview of the dicodon oriented HMM network which uses simple grammar to describe protein coding regions in a microbial genomic sequence because we need to keep the system as simple as possible in order to facilitate analysis focused on the dicodon usage measure. In microbial genome, and also in several eukaryotic no-internal-exon genome such as yeast, every protein coding region can be described, for 5' to 3' strand, as arbitrary iteration of

codons that is sandwiched by start(5') and stop(3') codons, and for 3' to 5' (complementary) strand, as arbitrary iteration of complementary codons that is sandwiched by complementary start(3') and stop(5') codons. Most of coding regions are connected by a spacer i.e. non-coding region which actually is arbitrary, but definitely shorter than coding regions, length of nucleotides. However, the genome structure is not such simple because:

- non-coding regions are occasionally not exist between coding regions
- coding regions are occasionally overlapped each other

Functions to handle these exceptions are not implemented in our system because such implementation has nothing to do with the evaluation of the dicodon usage measure and we just wanted to keep our system simple.

In figure 2.5 each rectangle corresponds to a certain structural item that forms a protein coding region structure. The non-code state corresponds to a non-coding region and is a single state and emits four output; A, C, G, and T. The start codon state corresponds to a start codon region and is a small HMM that has 11 single output states and 3 transition parameters inside when there are three possible start codons are expected [1]. The stop codon is conceptually identical to the start codon state. The Dicodon state corresponds to a coding region sandwiched by start and stop codons and is an HMM that has 185 single output states and 3,782 transition parameters inside. As the total, the HMM has 7,568 transition parameters.

Self-identification Learning

The initial parameters of the dicodon oriented HMM have uniform value. For example, non-code state has uniform distribution for every output probability i.e. 1/4 for A, C, G, and T. The self-identification learning begins the genefinding with this pre-learning condition. H2Vite outputs a file denoting the prediction where in the sequence belong to a state with likelihood calculated by Viterbi algorithm. The output from H2Vite is parsed by a Perl script and statistical data is accumulated to update HMM parameters i.e. emission parameters for non-code state and transition parameters

for start/stop codon dicodon state. During the statistical data accumulation, the system rejects apparently false answer that do not comply coding region grammar implemented in HMM network. Hence the HMM is trained by correct or possibly correct prediction results although it is never fed training data in advance. Then H2Vite try recognition again, but this time, with updated HMM parameters. The above procedures are iterated until the recognition accuracy become maximum.

In order to evaluate the robustness of the self-identification learning, we used short fragments of microbial genome sequence data such as 1/2, 1/4, 1/8, and such of whole genomic sequence data to train the HMM as described above. After the HMM is trained, the model starts to find protein coding regions from a whole genomic sequence data and we can see how the HMM can predict coding regions accurately with a short fragment of training data. Therefore we can evaluate data length dependency of self-identification learning.

[1] When there are less than three admissible start codons, the number of the HMM states are reduced to less than 11

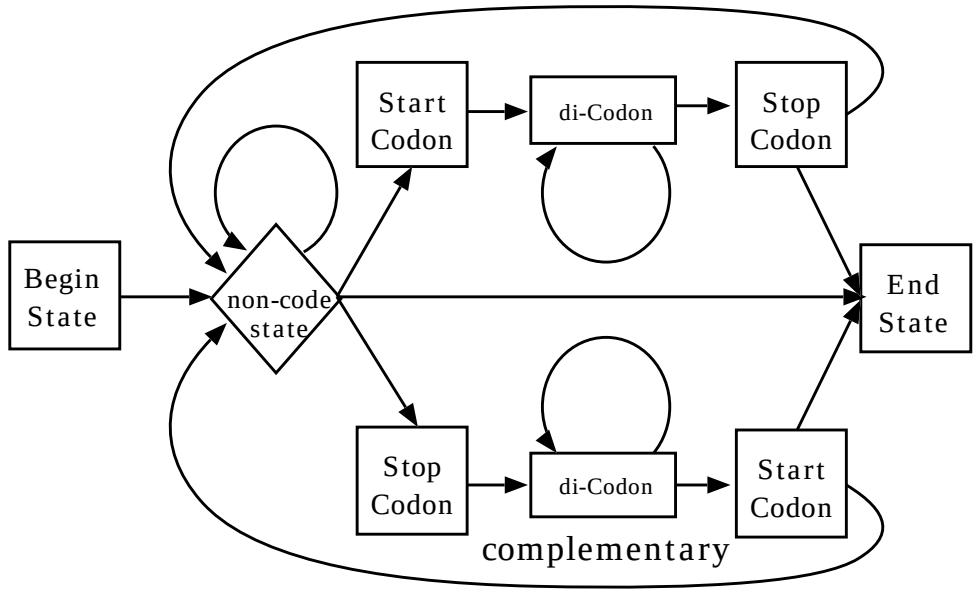


Figure 2.5: A network diagram of a dicodon-oriented HMM. ”Start Codon” state emits possible three start codons(ATG, TTG, GTG). ”Stop Codon” state emits possible three stop codons(TAA, TAG, TGA). ”di-Codon” state emits possible 61 dicodons iteratively.

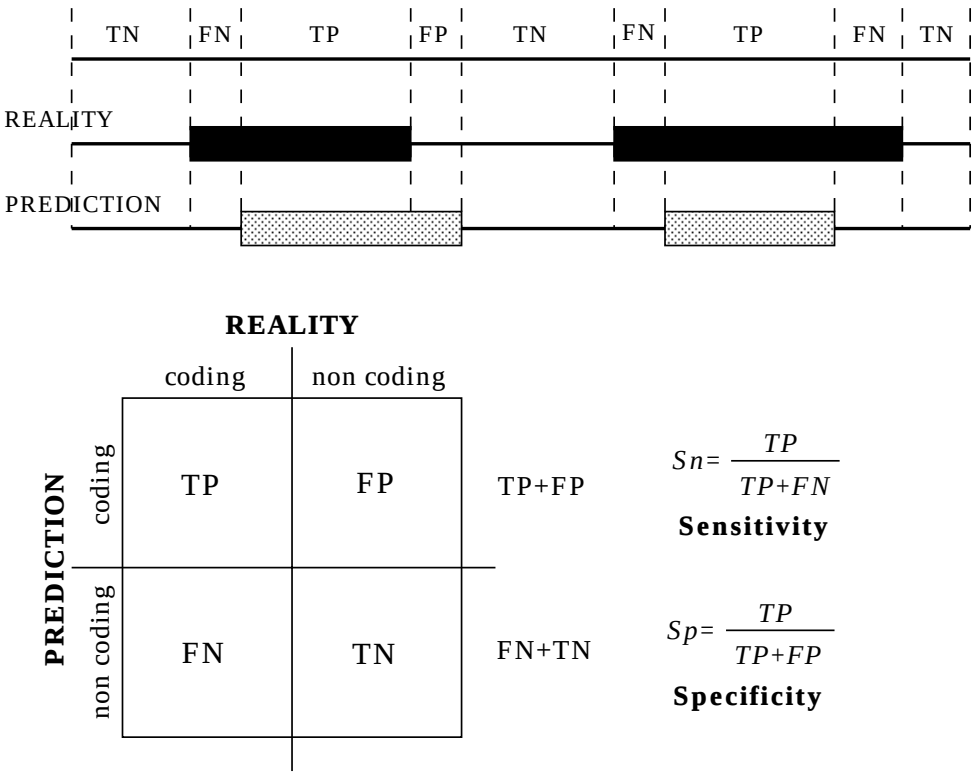


Figure 2.6: Measures for prediction accuracy at the nucleotide level (excerpt from [21] Figure 1).

2.2.2 Results and Discussion

The prediction accuracy is evaluated by counting TP(true positive): number of bases predicted as inside of coding regions correctly, TN(true negative): number of bases predicted as outside of coding regions correctly, FP(false positive): number of bases predicted as inside of coding regions incorrectly, and FN(false negative): number of bases predicted as outside of coding regions incorrectly. There are common measures to evaluate prediction accuracy at the nucleotide level [21] (see also Figure 2.6):

- Sensitivity:

$$Sn = \frac{TP}{TP + FN}$$

- Specificity:

$$Sp = \frac{TP}{TP + FP}$$

- Correlation Coefficient:

$$CC = \frac{(TP \times TN) - (FN \times FP)}{\sqrt{(TP + FN) \times (TN + FP) \times (TP + FP) \times (TN + FN)}}$$

Additionally, simple nucleotide level prediction accuracy is given by

$$R = \frac{TP + TN}{TP + TN + FP + FN}.$$

Table 2.2.3 shows the highest prediction accuracy (nucleotide level) R, sensitivity Sn, specificity Sp, and correlation coefficient CC for 17 microbial genomic sequence data. As a comparison for the score we got, the table includes scores obtained by another similar research by Audic and Claverie [7]. Please note that the objectives of this research do not include getting high prediction accuracy and our system is far simple than that of the Audic and Claveries', however our prediction accuracy exceeds their score over most species.

Figure 2.7 to 2.9 show dependency of R, Sn, Sp and CC on training data size for each microbial genomic sequence data. Sn stays constantly high level over all sequence data because almost all bacteria contains very small number of non-coding regions comparing to coding regions. Hence FN is much smaller than TP . Sp, R and CC draw similar proposition because of the same reason. There are apparent degradation of prediction accuracy for short training data size. However, the prediction accuracy stays high until the training data size is lowered to around 1/16 of whole data size.

Figure 2.10 to 2.12 show the number of learned HMM parameters versus training data size for each microbial genomic sequence data. The results shown in the figure vary widely for each sequence data because there are many differences caused by evolutionary diversity. Some of the sequence data require very small amount of HMM parameters, far below from the parameter size of codon usage measure i.e. 1,220, to identify protein coding regions. On the other hand, some of the sequence data require more than 1,220 parameters to be learned to attain good prediction accuracy. Besides the maximum number of HMM parameters often results in lower prediction accuracy i.e. over fitting. Typical proportion is shown in Archaeoglobus

Table 2.4: Recognition result for 17 microbial genomic sequence data. CC stands for correlation coefficient. R stands for nucleotide-level prediction accuracy. And R* shows another nucleotide-level prediction accuracy by Audic and Claverie [7].

Species	Sensitivity	Specificity	*CC*	*R*	*R**
Archaeoglobus fulgidus	0.97	0.97	0.65	0.94	0.92
Aquifex aeolicus	0.98	0.97	0.60	0.95	-
Borrelia burgdorferi	0.98	0.99	0.81	0.97	-
Bacillus subtilis	0.98	0.98	0.82	0.96	0.87
Chlamydia trachomatis	0.98	0.99	0.87	0.97	-
Escherichia coli	0.95	0.99	0.81	0.95	0.91
Haemophilus influenzae	0.98	0.96	0.80	0.95	0.90
Helicobacter pylori	0.98	0.97	0.75	0.95	0.93
Mycoplasma genitalium	0.98	0.92	0.54	0.91	0.96
Methanococcus jannaschii	0.98	0.98	0.85	0.97	0.89
Mycoplasma pneumoniae	0.98	0.95	0.69	0.93	0.92
Methanobacterium thermoautotrophicum	0.97	0.99	0.81	0.96	0.93
Mycobacterium tuberculosis	0.96	0.98	0.72	0.95	-
Pyrococcus horikoshii	0.97	0.94	0.61	0.92	-
Rickettsia prowazekii	0.98	0.98	0.93	0.97	-
Synechocystis PCC6803	0.96	0.99	0.82	0.96	0.91
Treponema pallidum	0.97	0.97	0.65	0.95	-

fulgidus, Borrelia burgdorferi, Chlamydia trachomatis, Escherichia coli, Haemophilus infulenzae, Methanococcus jannaschii, Mycobacterium tuberculosis, Pyrococcus horikoshii, Rickettsia prowazekii, and Treponema pallidum.

Therefore it is obvious that not all of HMM parameters are needed to identify protein coding regions with reasonable accuracy. Consequently, this evidence leads to a conclusion that the dicodon usage measure is redundant for the genefinding.

2.2.3 Conclusion for the preliminary examination

Our evaluation shows that the dicodon usage measure is redundant for the gene finding. The result implies that we can use a measure that has smaller size of parameters for genefinding in reasonable accuracy. Fickett and Tung indicated that the dicodon usage measure performs better than codon usage measure [35]. Their evaluation shows that prediction accuracy by the dicodon usage measure exceeds that by the codon usage measure but merely showing slightly better accuracy 1.3 considering the difference of parameter size among them. Although we found that the dicodon usage measure is too large in its parameter size and codon usage measure does not perform as good as the dicodon usage measure, the intermediate measure, which performs as accurate as the dicodon and has less parameter size than the dicodon, is not discovered yet. Thus our next investigation should be to find the most significant element in the dicodon usage measure which make it better than the codon usage measure so that we can discover the intermediate measure. The next section deals with the investigation.

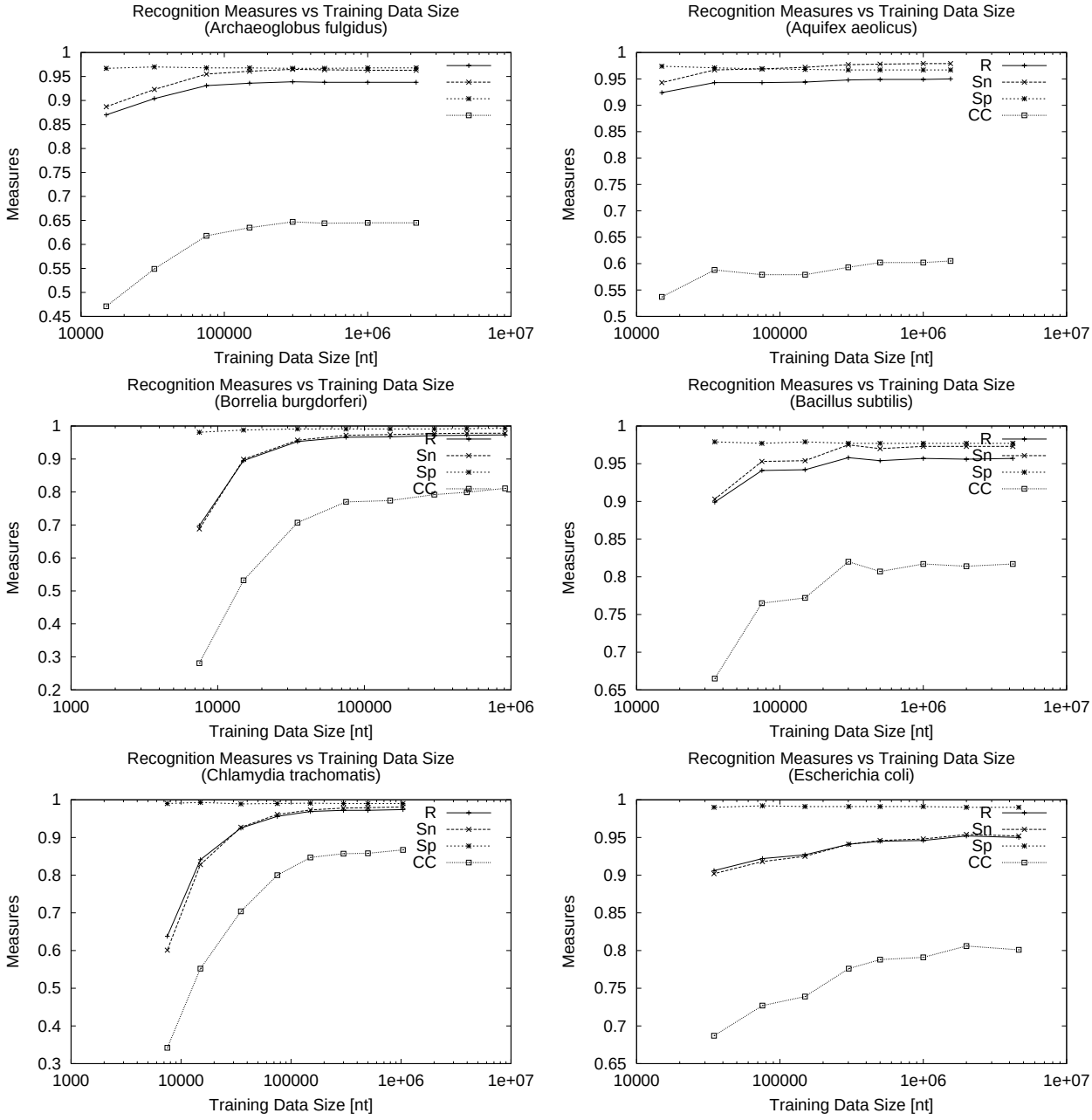


Figure 2.7: Results of genefinding (a). Measures: recognition accuracy (R), sensitivity (Sn), specificity (Sp), and correlation coefficient (CC) for 17 microbial genomic sequence data are shown. We used 1000,000, 500,000, 300,000, 150,000, 75,000, 32,500, 15,000, and 7,500 nt of fragments out of complete genomic sequences for training data of the dicodon-oriented HMM.

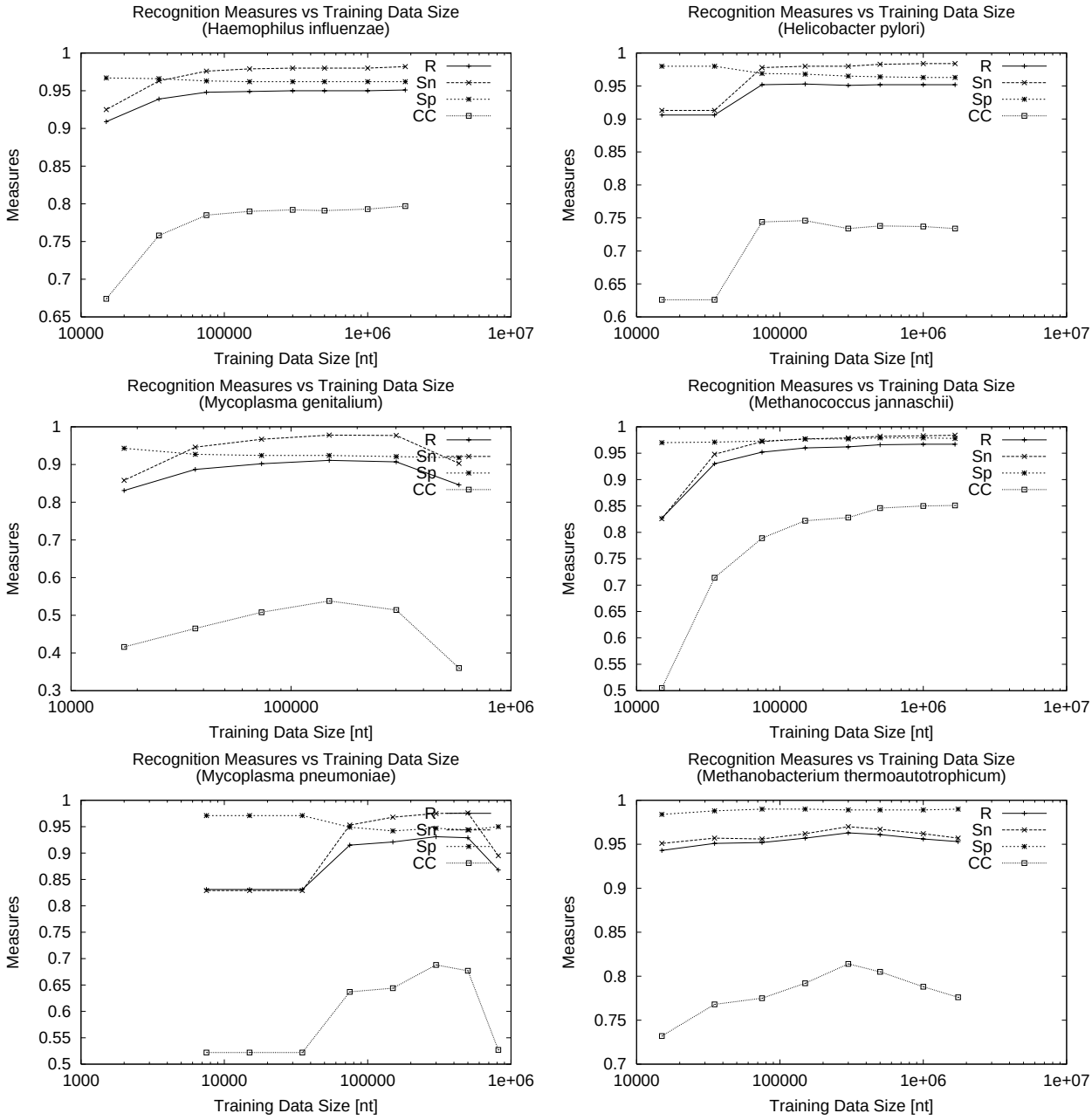


Figure 2.8: Results of genefinding (b) *continued*

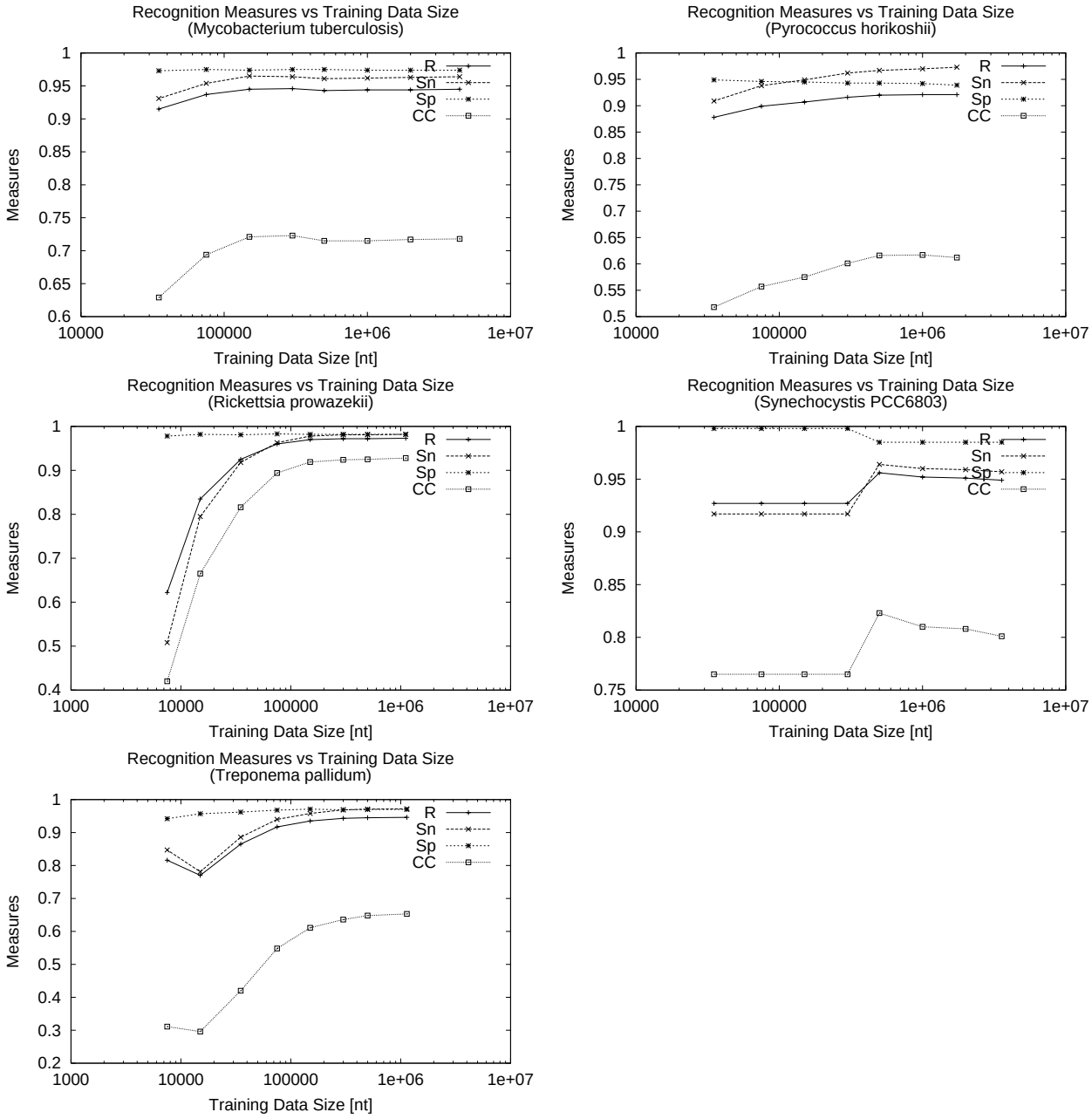


Figure 2.9: Results of genefinding (c) *continued*

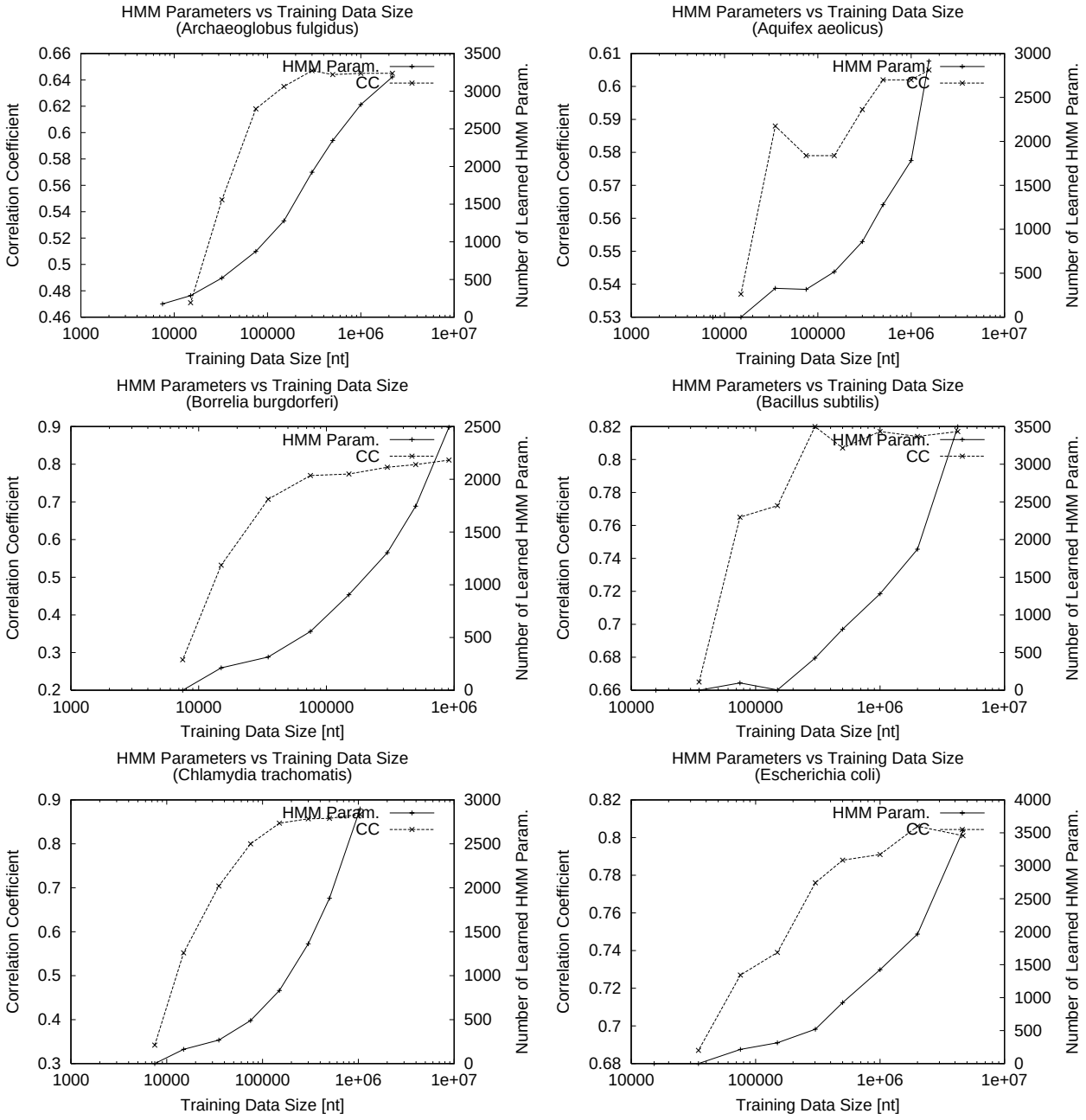


Figure 2.10: Correlation coefficient and the number of trained HMM parameters for 17 microbial genomic sequence data. (a)

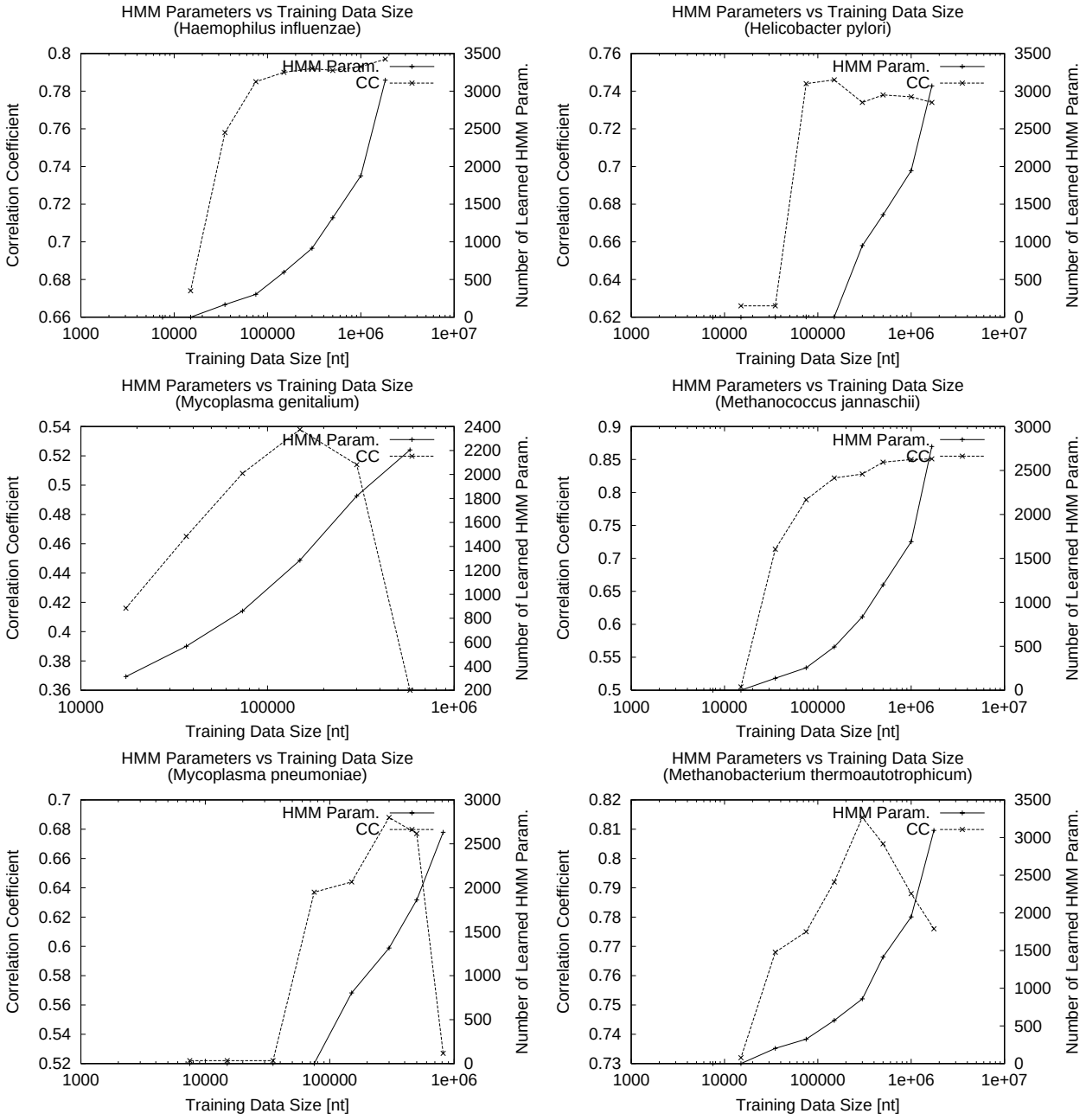


Figure 2.11: Correlation coefficient and the number of trained HMM parameters for 17 microbial genomic sequence data. (b)

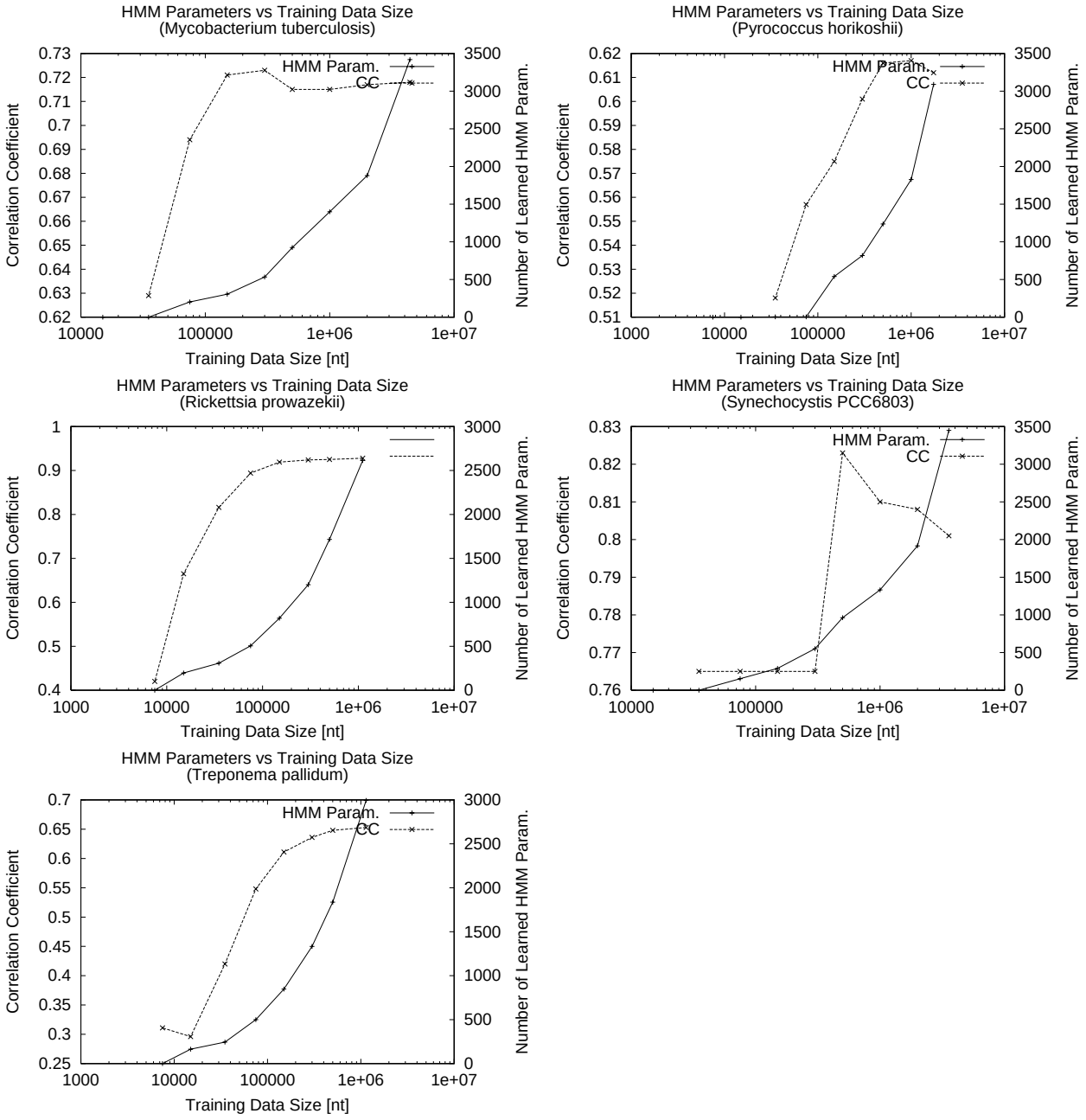


Figure 2.12: Correlation coefficient and the number of trained HMM parameters for 17 microbial genomic sequence data. (c)

2.3 Evaluation of Dicodon Usage Measure

According to our preliminary examination described above, the redundancy of the dicodon usage measure should be investigated. In this section, we prepared several different probabilistic models to emulate the dicodon model with smaller size of parameters. The size of parameters ranging from 461 to 1,024, is far bellow from the dicodon model which has 3,721. However, as our preliminary examination showed, protein coding regions in some microbial genomic sequence data require very small number of HMM parameters. Thus the models with small size of parameters should be evaluated objectively and quantitatively.

2.3.1 Models

There are 61 possible codons, possible dicodon counts up to 3,721. Hence the size of the parametric space of the dicodon model is 3,721. The size matters when we examine genefinding that uses self-identification learning. The self-identification learning with too many parameters usually fails to produce good result because it requires too large training data while they are not sufficiently available. On the other hand, accuracy of a model hardly gets high enough when the model conveys too few parameters.

Fickett and Tung [35] evaluated many protein coding measures including diaminoacid, codon usage, and dinucleotide bias. These measures never perform better than dicodon usage. However, dicodon can be represented by combinations of these well known biological attributes in certain degree. Figure 2.13 depicts each attributes contained in a nucleotide hexamer.

We presumed that the product of diamino-acid, codon usage, and G+C content emulates dicodon usage very well. Because, (i) there presumably are structural information of proteins embedded in coding regions that corresponds to the diamino-acid information. The diamino-acid information employs fairly larger amount of information ($20 \times 20 = 400$ parameters) than the information derived by a pair of dinucleotides ($16 \times 16 = 256$ parameters). (ii) codon usage determines third nucleotide which follows a couple of nucleotides determined by an amino-acid. The amino-acid information is derived from diamino-acid information. (iii) the third nucleotide might have a modification according to G+C content.

Based on the idea (i) to (iii), we defined the models B to F. Every model is a probabilistic representation of nucleotide hexamer with emphasis on the codon usage, C+G content and diamino-acid. The model B is a simple product of diamino-acid and codon usage and it does not use C+C content in order to evaluate how this model behave worse than those using C+G content information. The models C and D include correction term. In the model D, we supposed a certain bias among each nucleotide instead of seeing G-C and A-T are identical respectively. In this model, the codon usage is modified by a relation between its own third nucleotide and that of preceded codon. The model E uses two codon usage sets, which are used selectively regarding C+G content of the preceded codon. The model F uses four codon usage sets, which are used based on nucleotide-wise rather on C+G content-wise. The model G is more similar to the dicodon model than the other models. Because this model is a dicodon model without distinction of G-C and A-T at its third nucleotide position. The model conveys smaller parameter size (1,024) than that of the dicodon, but it is the largest among the other emulator models.

When these models perform well enough in comparison with dicodon model, that would help us to clarify which attribute is the most crucial to the dicodon model.

A) the dicodon model:

$61 \times 61 = 3,721$ parameters

$$p_A(c_j|c_i) \equiv p(c_j|c_i). \tag{2.1}$$

B) model of pair amino-acid and codon usage:
$20 \times 20 + 61 = 461$ parameters

$$p_B(c_j|c_i) \equiv p(A(c_j)|A(c_i))p(c_j|A(c_j)). \tag{2.2}$$

C) model of pair amino-acid and codon usage modified by C+G content:
$20 \times 20 + 61 + 2 = 463$ parameters

$$p_C(c_j|c_i) \equiv p(A(c_j)|A(c_i))\lambda_B p(c_j|A(c_j)) + (1-\lambda_B)p(f_{gc}(c_j)|f_{gc}(c_i)). \tag{2.3}$$

D) model of pair amino-acid and codon usage modified by pair C+G content:
$20 \times 20 + 61 + 4 \times 4 = 478$ parameters

$$p_D(c_j|c_i) \equiv p(A(c_j)|A(c_i))\lambda_C p(c_j|A(c_j)) + (1-\lambda_C)p(f_{atgc}(c_j)|f_{atgc}(c_i)). \tag{2.4}$$

E) model of pair amino-acid and codon usage with C+G content dependency:
$20 \times 20 + 2 \times 61 = 522$ parameters

$$p_E(c_j|c_i) \equiv p(A(c_j)|A(c_i))p(c_j|A(c_j), f_{gc}(c_i)). \tag{2.5}$$

F) model of pair amino-acid and codon usage with pair C+G content dependency:
$20 \times 20 + 4 \times 61 = 644$ parameters

$$p_F(c_j|c_i) \equiv p(A(c_j)|A(c_i))p(c_j|A(c_j), f_{atgc}(c_i)). \tag{2.6}$$

G) model of shrunk dicodon usage:
$32 \times 32 = 1024$ parameters

$$p_G(c_j|c_i) \equiv p(S(c_j)|S(c_i)). \tag{2.7}$$

$A(c)$ stands for an amino-acid which corresponds to a codon c. The function $f_{gc}(c)$ returns "GC" if the third nucleotide in a codon c is "G" or "C". Otherwise it returns "AT". Henceforth the probability $p(GC|AT)$ stands for a probability to have a codon looks like "XXG" or "XXC" right after a codon "XXA" or "XXT". Another function $f_{atgc}(c)$ returns the third nucleotide of a codon c. $p(c_j|A(c_j), f_{gc}(c_i))$ represents two codon usages. One is a codon usage observed right after a codon including "G" or "C". The another is a codon usage observed right after a codon which has "A" or "T". $p(c_j|A(c_j), f_{atgc}(c_i))$ represents four codon usages that correspond to a third nucleotide of a preceded codon c_i. It is calculated so that the square error between the dicodon model become minimal. For model G, $S(c)$ represents shrunk codon. Shrunk codon does not distinguish G-C, and A-T. For instance, $S(XXG) = S(XXC)$ and $S(XXA) = S(XXT)$.

2.3.2 Evaluation of models

In order to evaluate these six models(B to G) against the dicodon model, We used 17 microbial genomic sequences [59, 27, 38, 61, 97, 13, 36, 101, 37, 17, 45, 93, 23, 56, 52, 39, 3] and C. elegance [32] genome sequences obtained from GenBank and took following procedure:

1. So-called Jack knife strategy is applied here.

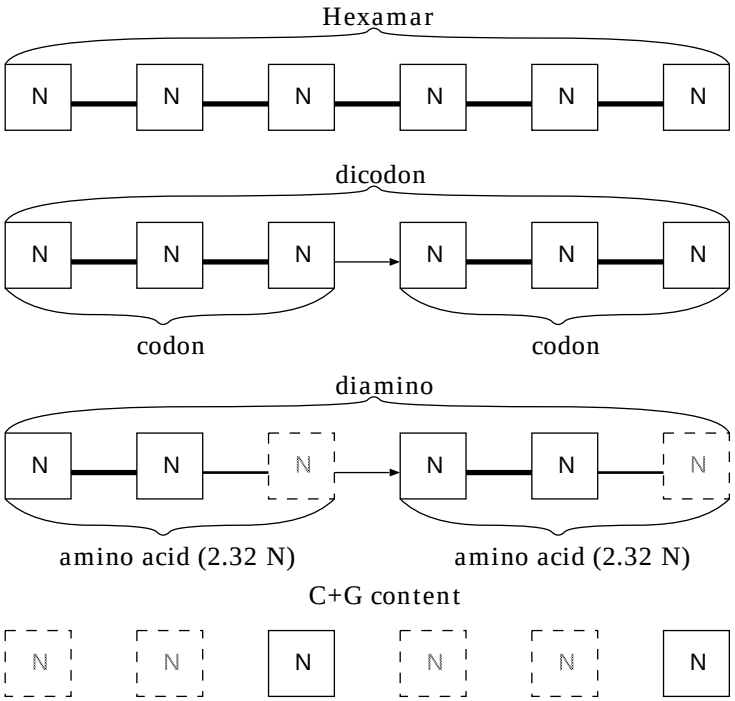


Figure 2.13: The hexamer treats six nucleotide as one unit thus is identical to 6 nucleotide window frame examination. The codon binds three nucleotide as one datum thus the dicodon stands for a pair of codons. A codon corresponds to an amino-acid but an amino-acid corresponds to one or more codons and there are only 20 possible aminoacids while there are 64 possible codons. 8 amino-acids(family box) are determined by 2 nucleotides. 12 amino-acids(2-codon set) are determined by 2.5 nucleotides. 1 aminoacid(2-codon set+1) is determined by 2.75 nucleotides. Approximately, an amino-acid is determined by 2.32 nucleotides. C+G content stands for a biased possibility to have a C or G in the third position of codon. Thus it can not be defined by single nucleotide but a certain length of window frame should be considered.

2. Several size of Learning sets and Testing sets are prepared in order to evaluate performance and robustness of each model.

3. When an examined genomic sequence has N genes, we take N/n genes out of the sequence randomly($n = 1.3, 1.7, 2, 4$).

4. The extracted genes are used for the Learning sets.

5. Rest of the genes and the non-coding regions are used as the Testing set.

6. Train six models and the dicodon model using the Learning set.

7. Accumulate coding potentials cod_x of every coding region in the Testing set based on the six models.

8. Train the dicodon model using the Testing set and accumulate a coding potential cod_o.

9. Obtain profiles of coding potentials for coding regions and non-coding regions.

10. Evaluate every models in two ways: Approximation error and Learning/Testing evaluation.

A coding potential, for model x, of a coding region $C = (c_1, c_2, \cdots, c_n)$ which consists of n codons can be computed as follows:

$$\begin{aligned} cod_x(C) &= \frac{1}{n} \log p_x(c_1, c_2, ..., c_n) \\ &= \frac{1}{n} \log p_x(c_2|c_1) \cdots p_x(c_n|c_{n-1}) \\ &= \frac{1}{n} \sum_{i=2}^{n} \log p_x(c_i|c_{i-1}). \end{aligned} \tag{2.8}$$

Approximation error

The models B to F are approximations of the dicodon model. Therefore, we can evaluate these models in terms of approximation error of each models against the dicodon model.

- We split a sequence into the learning sequence and the testing sequence.
- AT is the dicodon model that was trained with testing sequence.
- AL is the dicodon model that was trained with learning sequence.
- Other models are all trained with learning sequence.
- Compute coding potentials of coding/non-coding regions in the testing sequence for every model.
- We calculated square errors between coding potentials cod_o and coding potentials of the other model x.

$$D(x) = \sum_{\mathbf{C}} |cod_o(\mathbf{C}) - cod_x(\mathbf{C})|^2, \tag{2.9}$$

where $x = AL, B, C, ..., G$.

- This evaluation shows how these models accurately approximate the dicodon model.

Evaluation of Learning/Testing

Here we define a measure to evaluate an accuracy to distinguish coding regions and non-coding regions for each model. Then compute "distances", based on the measure, between profiles of coding/non-coding regions, and evaluate specificity/sensitivity of six models based on the distance(defined below) of each model.

We obtained profiles of coding/non-coding regions look like Figure 2.15. Two heaps of coding/non-coding regions are overlapped each other in certain degree. When we have a coding potential x for a predicted coding region, and the potential goes a midst of two heap, it has a probability to belong to a coding region and another probability for a noncoding region simultaneously. When the overlap, based on a model, is wider than that of other model, we need to do a stochastic decision for every predicted coding region whether it belongs to coding or non-coding regions more frequently than other model. This means we have to make one more guess after prediction of coding region. On the other hand, if a model has narrower overlap, most predicted coding regions are easily distinguished without guess. This can be a measure for relative accuracy of a model against other models.

Then, we defined a distance d using the measure described above (see Figure 2.14).

$$d = Sn(x_0) + Sp(x_0), \qquad Sn(x_0) = Sp(x_0) \tag{2.10}$$

$$Sn(x) = TP(x)TP(x) + FN(x), \quad Sp(x) = TP(x)TP(x) + FP(x) \tag{2.11}$$

$$TN(x) = \sum_{i=x_{min}}^{x} h_{nc}(i), \qquad FN(x) = \textstyle\sum_{i=x}^{x_{max}} h_{nc}(i) \tag{2.12}$$

$$TP(x) = \sum_{i=x}^{x_{max}} h_{cd}(i), \qquad FP(x) = \textstyle\sum_{i=x_{min}}^{x} h_{cd}(i) \tag{2.13}$$

As shown above, we take d of the equilibrium where sensitivity and specificity become equivalent.

2.3.3 Result

Table 2.4 shows maximum sensitivity+specificity of every model for 14 microbial genomic sequence data and 14 eukaryotic genomic sequence data. Mean sensitivity+specificity is shown in bottom of the table. Although the mean sensitivity+specificity shows that the dicodon and the model G (shrunk dicodon model) yield equivalent value, the details are different in each species. The dicodon scores higher than the model G in 15 species while the model G scores higher in other species. Figure 2.16 to 2.20 show sensitivity+specificity versus relative training data size of 14 microbial genomic sequence data and 14 eukaryotic genomic sequence data. The sensitivity+specificity scores tend to wobble because of the jack knife strategy. The jack knife strategy requires approximation in order to get smooth result. However we did just one time examination. Figure 2.21 shows comparisons of average square errors for coding region and non-coding region. The square errors are calculated against coding potential of dicodon model for each six models(B to G).

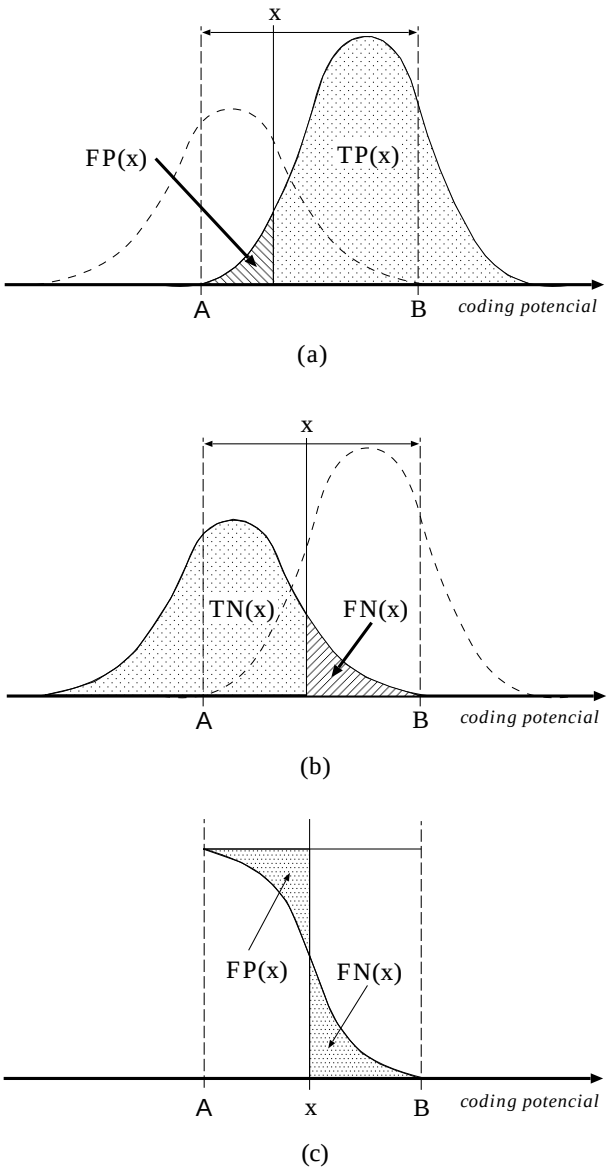


Figure 2.14: Profile of coding potentials for coding(right heap) and non-coding(left heap) regions. The two heaps have overlapped area $[A, B]$. We set a threshold coding potential x within $[A, B]$. (a) For coding potentials over x are taken to be coding regions. So cross-hatched area become false negatives. (b) For coding potentials under x are taken to be non-coding regions. The cross-hatched area become false positives. (c) We take x so that the sensitivity and specificity become equivalent. According to the definition of sensitivity and specificity, $FP(x) = FN(x)$ when $Sn(x) = Sp(x)$.

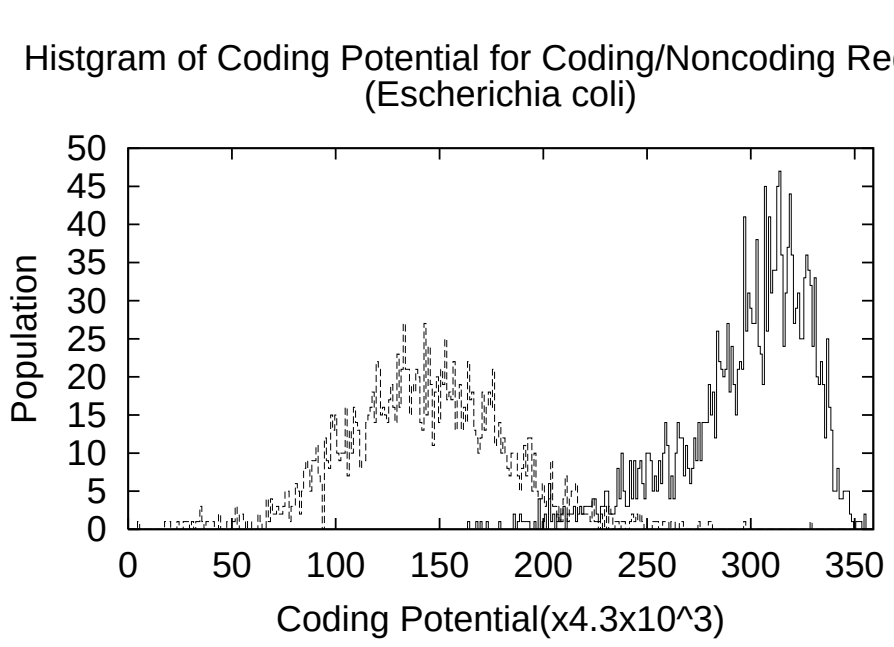


Figure 2.15: Actual histogram of coding/non-coding potential for E.coli

2.3.4 Discussion

Our evaluation shows that the dicodon model outperforms other six models(B to G) in terms of specificity and sensitivity (Table 2.4 and Figure 2.16 to 2.20). Besides, none of the emulation models(B to F) get closer than the model G in terms of approximation error (Figure 2.21).

The models B to F apparently failed emulating the dicodon model. This means that the information among a pair of codon conveys richer feature of coding regions than a mere combination of the diamino, codon usage, and C+G content, and the diamino-acid simply drops some crucial information in the coding region. Performance of the model B, which is the simplest, is constantly low among the other models. This corresponds to an evidence of the significance of C+G content. The model C performs slightly better than the B but it is not so apparent. While the model C has information of C+G content, linear interpolation of codon usage and C+G content did not work so much in this case. The model D performs better than the B and C. Although the differences of its performance between this model and the B, C are clearer than that of B and C, its performance improvement is poor. However, we should notice that nucleotide-wise bias at the third nucleotide is more significant than C+G content. The performance of the model E shows clearer improvements. This result indicates that the second codon usage depends on the C+G content of the first codon. The result of the model F is the best among the models B to F. With this result, there apparently is dependency of the second codon usage on the third nucleotide of the first codon rather on the C+G content. This indicates that a bias at the third nucleotide is not so uniform among G-C and A-T, and C+G content model is not sufficient for describing this bias. Therefore we should consider A, T, C, G individually. The model G scores the nearest performance to the dicodon model. Let us take a look at this result not from performance improvement but from performance decline. Only difference between this model and the dicodon model is that this model does not distinguish G-C and A-T at the third nucleotide. Again, this shows that the peculiar bias at third nucleotide that is indicated by the result of the model D and F.

Considering the difference between diamino and dicodon, dependency of the third nu-

cleotide of second codon on the first codon is important for describing superiority of the dicodon. Although the diamino-acid and codon usage are undoubtedly important attributes of dicodon, our result shows that C+G content is not enough for describing peculiar bias which is found at the third nucleotide.

2.4 Summary

Firstly, we showed that the redundancy of dicodon usage measure for genefinding based on the result obtained from our preliminary genefinding examination using dicodon oriented HMM with self-identification learning method, which showed that the HMM was able to predict protein coding region in microbial genomic sequence data with far less parameter size than the HMM employed. According to the fact, we performed the evaluation of the dicodon usage measure using 6 probabilistic models that emulate the dicodon usage measure with less parameter size than that in order to clarify the most significant element consists of the dicodon. However, the all emulation models, except shrunk dicodon model, failed to attain such high accuracy provided by the dicodon model. Although the shrunk dicodon model produced the result close to that of the dicodon model, it can not be identical to the dicodon based on the result of our evaluation. This fact showed that the dicodon usage measure can not be described by codon usage, pair amino acid, and C+G content. This negative result negates the widely believed common sense and, more importantly, proposed a new fact that a certain important element other than codon usage, pair amino-acid, and C+G content is still missing. This research does not deal with the missing element but indicated that the C+G content is not sufficient to emulate the dicodon model.

Table 2.5: Maximum sensitivity+specificity of every model for 14 microbial genomic sequence data and 14 eukaryotic genomic sequence data. Mean sensitivity+specificity is shown in bottom of the table.

Species	**dicodon**	**B**	**C**	**D**	**E**	**F**	**G**
Archaeoglobus fulgidus	1.976	1.960	1.962	1.960	1.961	1.965	1.976
Aquifex aeolicus	1.924	1.910	1.908	1.908	1.913	1.918	1.932
Borrelia burgdorferi	1.954	1.887	1.887	1.890	1.882	1.911	1.950
Bacillus subtilis	1.950	1.923	1.923	1.924	1.923	1.932	1.949
Chlamydia trachomatis	1.962	1.887	1.891	1.887	1.898	1.902	1.962
Escherichia coli	1.959	1.940	1.941	1.941	1.941	1.944	1.959
Haemophilus influenzae	1.951	1.926	1.924	1.926	1.920	1.932	1.946
Mycoplasma genitalium	1.881	1.821	1.785	1.813	1.833	1.840	1.873
Methanococcus jannaschii	1.973	1.920	1.921	1.918	1.924	1.940	1.970
Mycoplasma pneumoniae	1.856	1.823	1.833	1.823	1.831	1.835	1.856
Methanobacterium	1.956	1.946	1.946	1.947	1.950	1.950	1.954
Rickettsia prowazekii	1.975	1.920	1.923	1.920	1.931	1.930	1.970
Synechocystis sp.	1.946	1.922	1.922	1.923	1.923	1.932	1.950
Treponeuma pallidum	1.900	1.880	1.877	1.877	1.872	1.899	1.918
C. elegance(Chr I)	1.931	1.758	1.759	1.759	1.799	1.805	1.921
C. elegance(Chr II)	1.927	1.753	1.752	1.760	1.795	1.798	1.923
C. elegance(Chr III)	1.930	1.769	1.774	1.782	1.804	1.816	1.919
C. elegance(Chr IV)	1.946	1.816	1.819	1.822	1.846	1.859	1.938
C. elegance(Chr V)	1.929	1.739	1.743	1.744	1.778	1.798	1.918
Saccharomyces cerevisiae(Chr II)	1.755	1.615	1.644	1.639	1.620	1.681	1.780
Saccharomyces cerevisiae(Chr III)	1.674	1.545	1.523	1.545	1.536	1.549	1.709
Saccharomyces cerevisiae(Chr IV)	1.904	1.828	1.815	1.827	1.834	1.847	1.907
Saccharomyces cerevisiae(Chr VI)	1.724	1.597	1.605	1.580	1.642	1.591	1.742
Saccharomyces cerevisiae(Chr VIII)	1.847	1.672	1.678	1.714	1.700	1.754	1.832
Saccharomyces cerevisiae(Chr X)	1.835	1.727	1.725	1.727	1.748	1.762	1.843
Saccharomyces cerevisiae(Chr XI)	1.861	1.689	1.704	1.717	1.730	1.753	1.849
Saccharomyces cerevisiae(Chr XIII)	1.888	1.781	1.786	1.794	1.801	1.812	1.887
MEAN	1.896	1.808	1.809	1.812	1.822	1.834	1.896

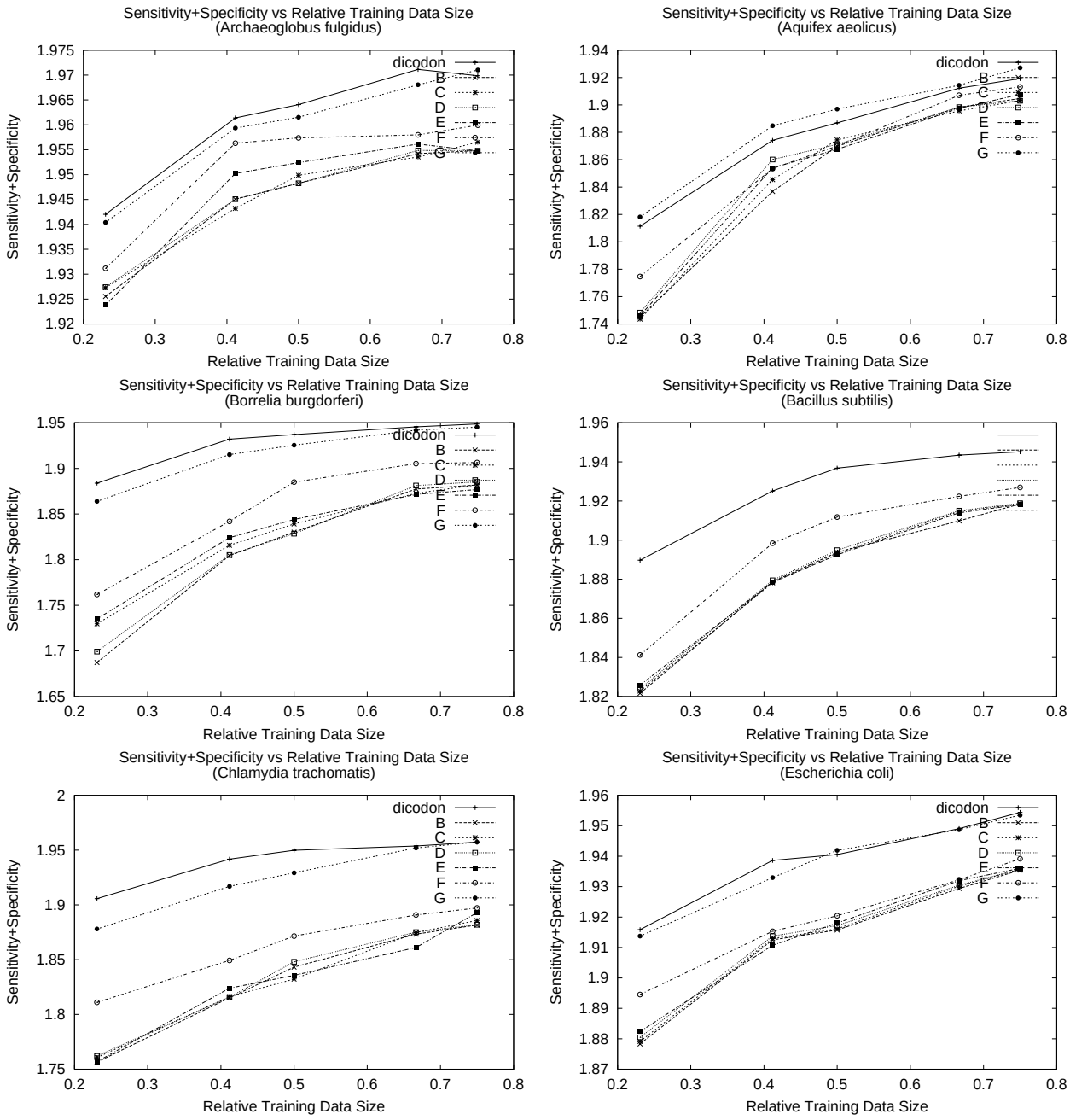


Figure 2.16: Sensitivity+Specificity versus relative training data size for 14 microbial genomic sequence data and 14 eukaryotic genomic sequence data (a)

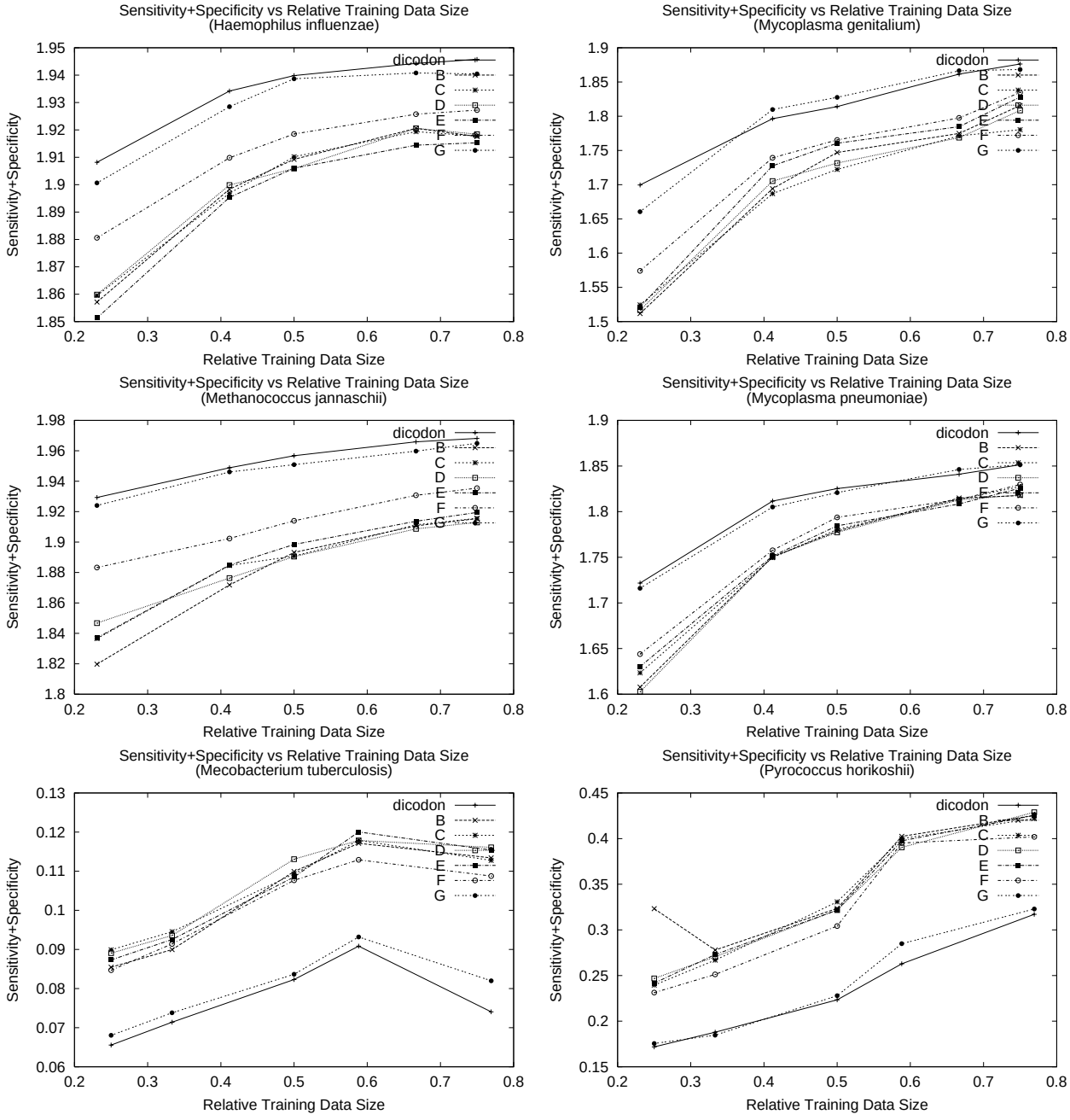


Figure 2.17: Sensitivity+specificity versus relative training data size (b) continued

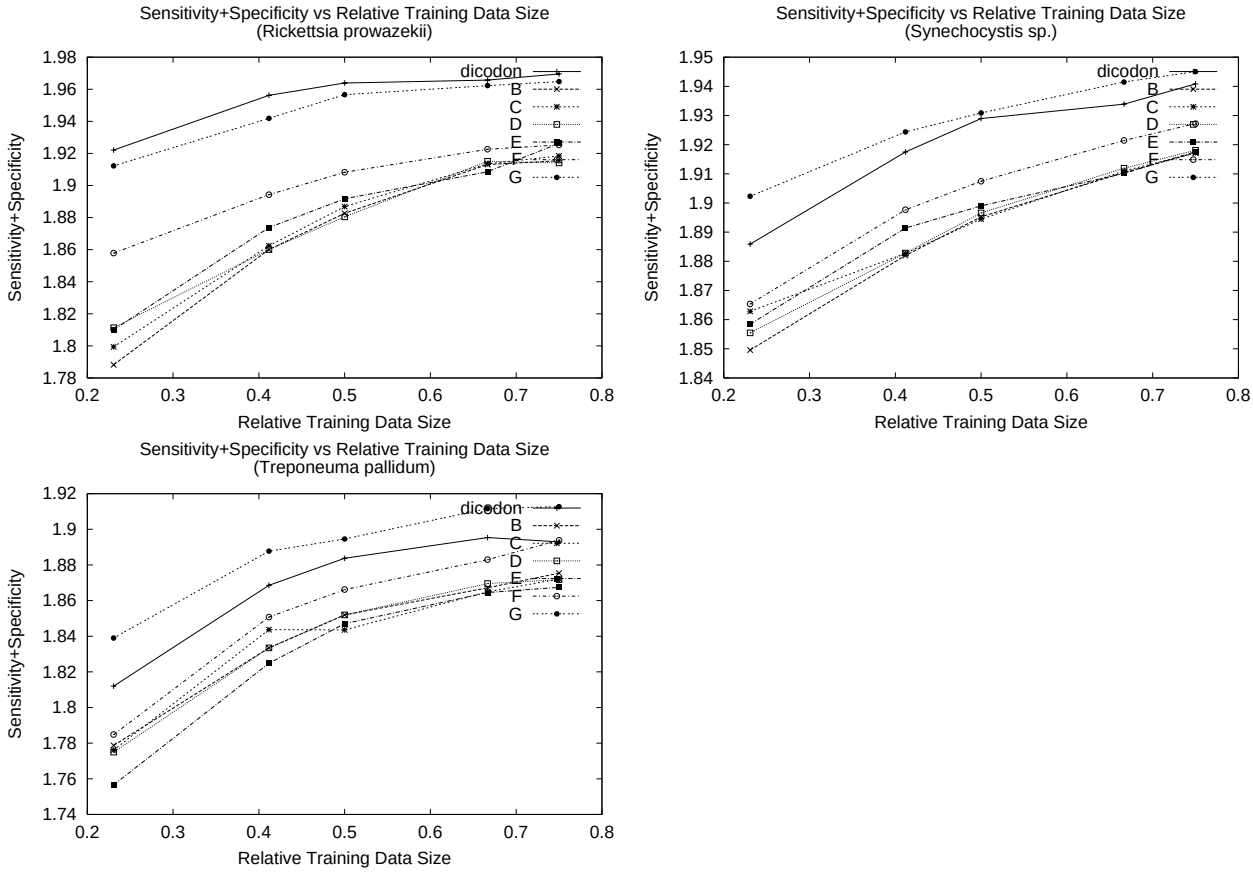


Figure 2.18: Sensitivity+Specificity versus relative training data size (c) continued

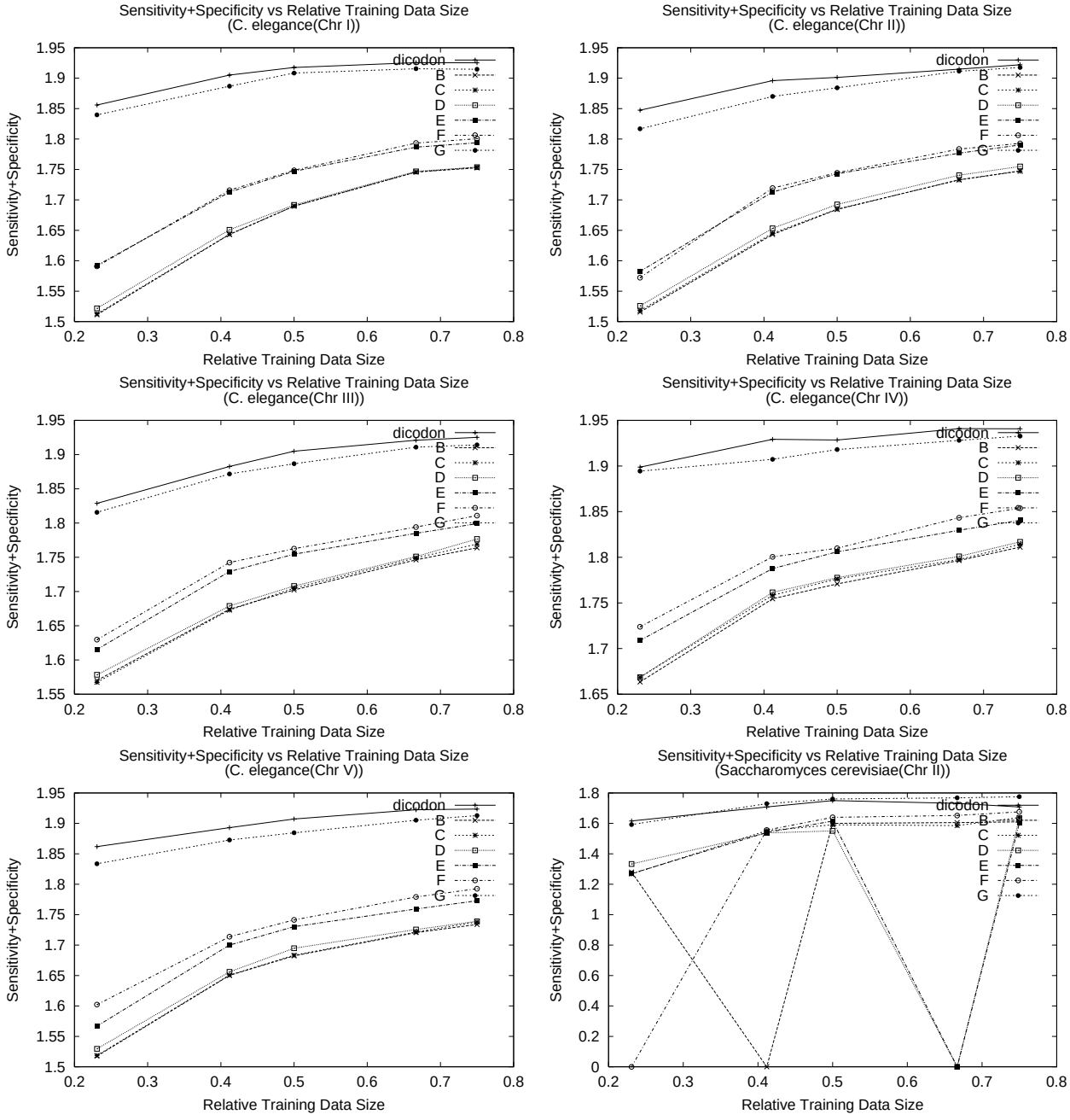


Figure 2.19: Sensitivity+Specificity versus relative training data size (d) continued

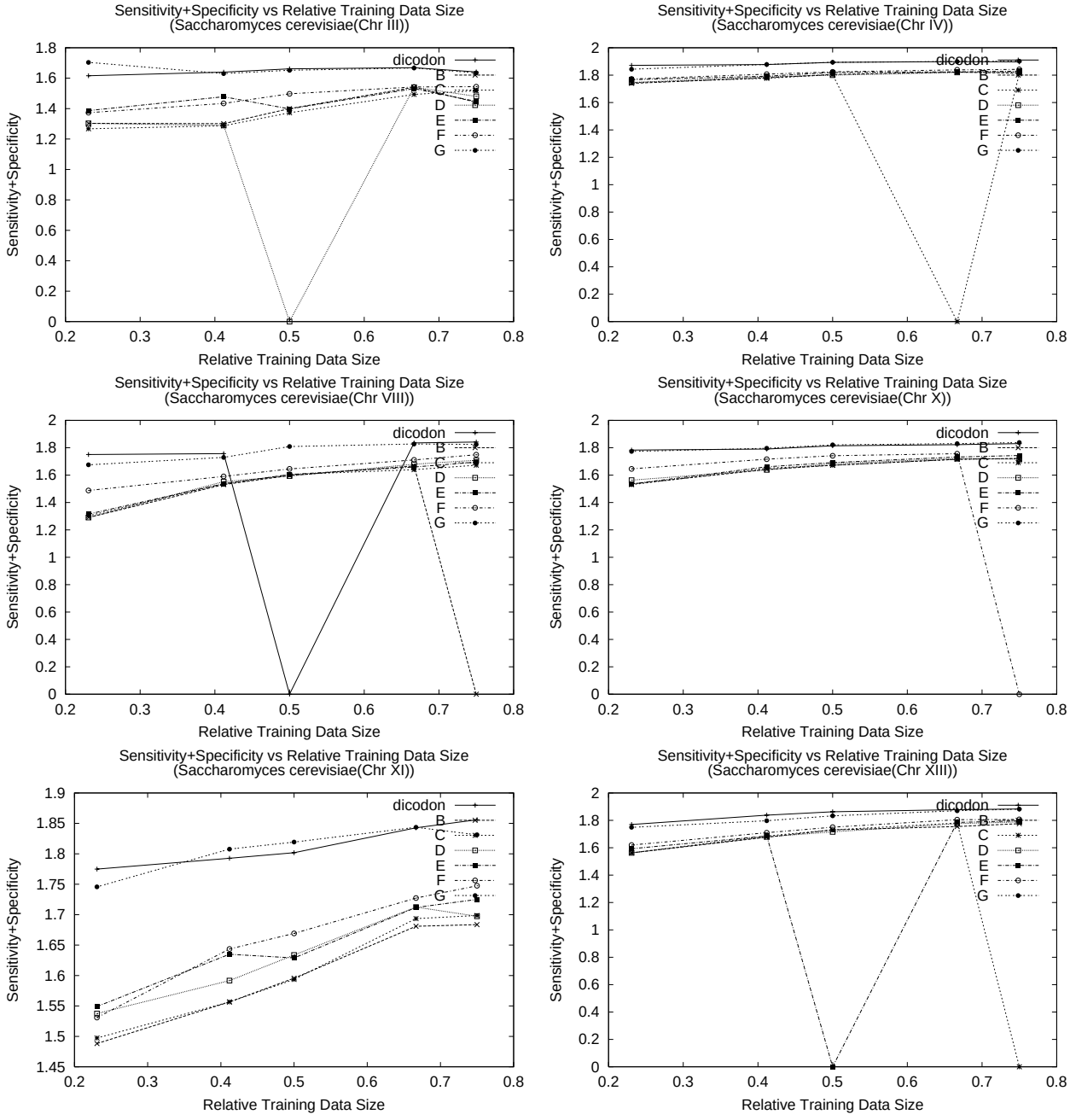


Figure 2.20: Sensitivity+Specificity versus relative training data size (e) continued

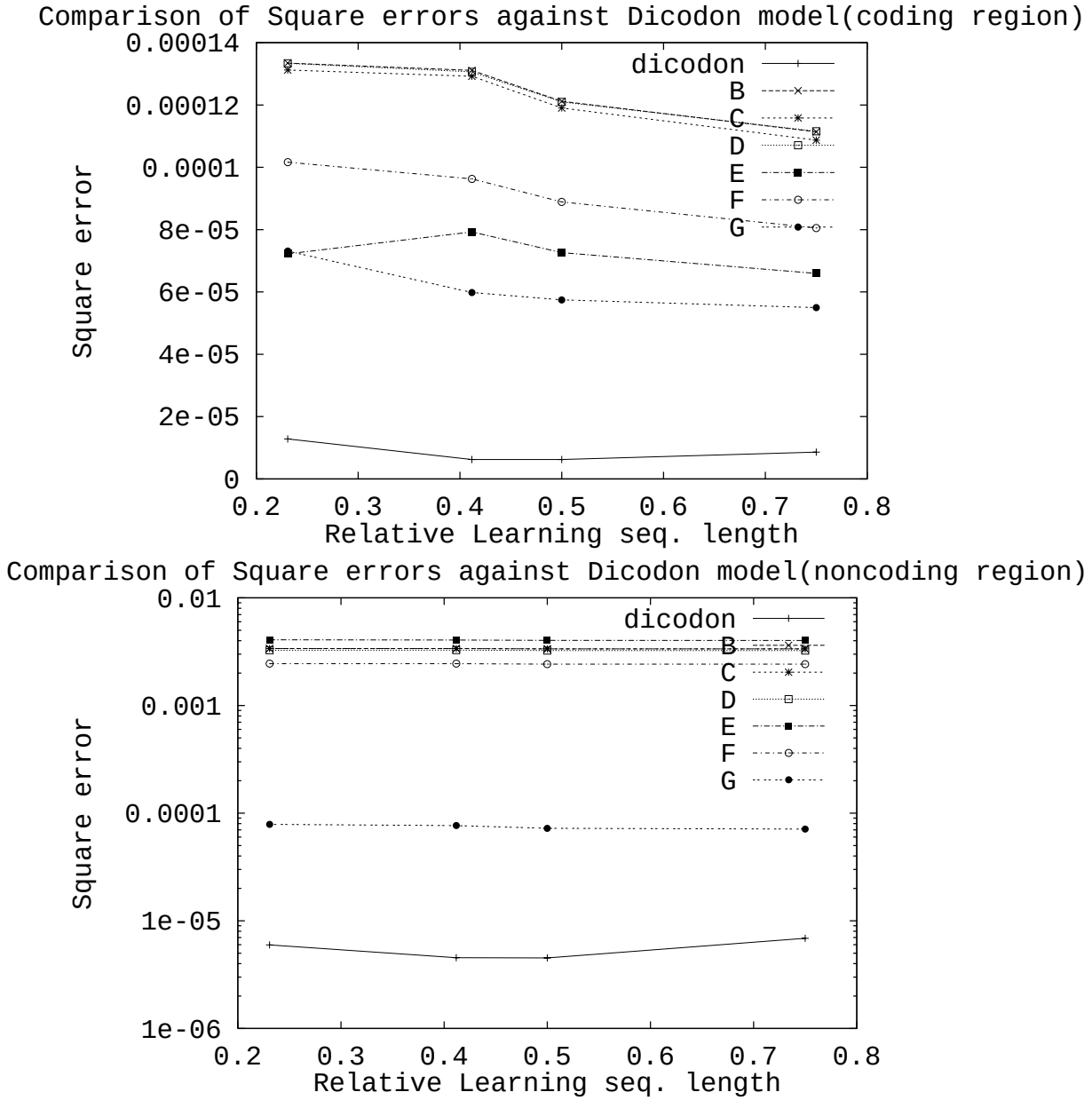


Figure 2.21: Figures show square errors of the coding potentials of testing sequences (above: coding regions, bellow:non-coding regions) for each models against the coding potential of dicodon model that was trained with testing sequences. The square errors are average values over 13 microbial and 2 eukaryotic genomes.

Chapter 3

Kernel Design for Biological Sequence Data

3.1 Introduction

Since a biological sequence is usually represented as a text that consists of a finite set of characters that represent nucleotides or amino acids, designing models based on formal languages have been constantly proposed since the early era of bioinformatics [1] [44, 24, 90, 29, 91, 25, 5, 48, 64, 46]. Therefore, we find the first applications of hidden Markov models (HMMs) [80] to biological sequences [100, 4, 16] in fairly early stage of bioinformatics. HMM is one of probabilistic formal language models that represents a biological sequence as a time-series data set generated from a certain stochastic process that behaves as a Markov process[2].

Several examples of biological sequence models by using HMMs are bacterial protein coding regions [60], human genes [18], secondary structure of proteins [4], profiles of protein functional families [95], and conserved motif sequences [42]. One of the major reasons why HMMs are used for such a wide range of biological sequence data is that ease of building models with HMMs. Representing biological sequences with HMMs is identical to building a grammar of the sequences. For example, let us think of defining a simple grammar for bacterial protein coding regions (CDS). According to some basic biological knowledge, CDS begins with a start codon, then repeats of intermediate codons follow and ends with a stop codon. This knowledge can be almost intuitively converted to an HMM network that emits start codon symbols at first state next to initial state, iteratively emits intermediate codons at second state, finally emits stop codon at the third state. Such an aspect of HMMs is very useful for biological sequence data analysis. However, there are some difficulty to use HMMs. In most cases of biological sequence data analysis, HMMs are used for pattern recognition problems such as gene finding discussed in previous chapter. HMMs themselves do not deal with label information which is crucial to pattern recognition problems. Instead of yielding a label whether a query matches to a trained pattern or not, HMMs provide likelihood, a probability that represents how a query

[1] Aside from bioinformatics, another major stream of researches that involve DNA and formal language theory is DNA computing [1] which aims utilizing the nature of DNAs as computational resources.

[2] A random process whose future probabilities are determined by its most recent values. More precisely, a stochastic process $x(t)$ is called Markov if for every n and $t_1 < t_2 \cdots < t_n$, we have $P(x(t_n) \leq x_n | x(t_n - 1), \cdots, x(t1)) = P(x(t_n) \leq x_n | x(t_{n-1}))$. To model biological sequences by using HMMs is based on hypothesis that every occurrence of a nucleotide or an amino acid in a sequence depends on its preceding nucleotides or amino acids.

is similar to a trained pattern. Therefore, HMM users need to devise appropriate discriminator to classify queries into positives or negatives based on their likelihoods. However, there is no method to deliver good discriminator in general as far as we know. Performance of the discriminator basically depends on quality of HMMs, in other words, quality of biological sequence models. Unfortunately, building good sequence models is usually very difficult task because even though newly discovered facts of molecular biology are rapidly piled up in daily basis, there are still lots of things are kept unexplained for biological sequences. In our point of view, these two problems should be addressed.

Jaakkola and et al. coined a novel method introducing a kernel method for biological sequence data analysis [47]. Their method presents a way to deliver Fisher kernel from protein sequence data by utilizing HMMs, then a support vector machine (SVM) [26] is used for classifying queries. They presented a method to introduce SVMs as a back-end of HMMs. This addresses the problem that use of HMMs requires auxiliary discriminators. Especially, SVMs are known to be a very good method for pattern recognitions such as text categorizations [49, 30], image pattern recognitions [75, 79, 76, 83, 12], hand-written character recognitions [67, 10, 14] and so on. The Fisher kernel has been successfully applied to many tasks e.g. protein classification [47, 53], microarray data classification [15, 70], promoter region detection [77], and translation initiation site recognition [110]. Unfortunately, their method is tightly bound to HMMs due to the Fisher kernels which derivation requires fairly complex mathematical operations. Thus their method has difficulty for designing kernels for varieties of HMMs or even other latent variable models. We propose a novel framework for kernel design that allows separating kernel design from latent variable models. Using the framework, we show that Fisher kernel is a special case of marginalized kernels, which gives another viewpoint to the Fisher kernel theory. Although our approach can be applied to any object, we particularly derive several marginalized kernels useful for biological sequences (e.g. DNA and proteins). The effectiveness of marginalized kernels is illustrated in the task of classifying bacterial gyrase subunit B (*gyrB*) amino acid sequences.

In this chapter, Sec. 3.2 gives brief overview of kernels in general. Sec. 3.3 presents conceptual illustration of our kernel design framework as well as mathematical descriptions including connection to Fisher kernels. Computational experiments demonstrating performances of marginalized kernels are presented in Sec. 3.4, in comparison with the Fisher kernel, by using *gyrB* amino acid sequence data. Summary is given in Sec. 3.5.

3.2 Kernel

Here we give a fundamental idea of a kernel in general. Readers aware of this topic may skip this section. Readers need more details are encouraged to consult one of the appropriate textbooks such as [26, 88].

Let us think of a discriminant analysis that separates $\boldsymbol{x}$ from $\boldsymbol{o}$ ($\boldsymbol{x}, \boldsymbol{o} \in X$) as shown in figure 3.1 (*left*). If we can define a non-linear function $\Phi : X \to F$ which maps the data points from X to F where $F = \{\Phi(\boldsymbol{z}) : \boldsymbol{z} \in X\}$ and we can use a linear discriminator (figure 3.1 *right*). F is called a *feature space* and $\Phi(\boldsymbol{z})$ is called a *feature vector*. The linear discriminator $f(\boldsymbol{z})$ can

be written as follows in respect to $\Phi(\boldsymbol{z})$:

$$\begin{aligned} f(\boldsymbol{z}) &= \sum_{i=1}^{N} w_i \phi_i(\boldsymbol{z}) + b \\ &= \langle \boldsymbol{w} \cdot \Phi(\boldsymbol{z}) \rangle + b \\ & f(\boldsymbol{x}) \geq 0, \quad f(\boldsymbol{o}) < 0, \end{aligned} \tag{3.1}$$

where $\boldsymbol{w}$ is a parameter obtained from training data $\boldsymbol{x}$ (positive training data) and $\boldsymbol{o}$ (negative training data). $\boldsymbol{w}$ can be represented as a linear combination of training data as follows (y_i is a class label of each training data):

$$\begin{aligned} \boldsymbol{w}(\boldsymbol{t}) &= \sum_{i=1}^{N} \alpha_i y_i \Phi(\boldsymbol{t}_i) \\ & \boldsymbol{t} = (\boldsymbol{t}_i, y_i), \quad \boldsymbol{t}_i \in \boldsymbol{x}, \boldsymbol{o} \end{aligned} \tag{3.2}$$

where α_i is a weight for each training points. In regard to Eqn. 3.1 and 3.2, $f(\boldsymbol{z})$ can be rewritten as a sum of dot products between the training data $\boldsymbol{t}$ and test data $\boldsymbol{z}$:

$$f(\boldsymbol{z}) = \sum_{i=1}^{N} \alpha_i y_i \langle \Phi(\boldsymbol{t}_i) \cdot \Phi(\boldsymbol{z}) \rangle + b. \tag{3.3}$$

In general, finding Φ is not an easy task. However, if we know the actual value of the feature vector $\Phi(\boldsymbol{z})$ or the dot product $\langle \Phi(\boldsymbol{t}_i) \cdot \Phi(\boldsymbol{z}) \rangle$ without knowing Φ, we can bypass several complications of non-linear discriminant analysis [3]. Such a method involving the direct computation is called *kernel* [26]. In this case, kernel is a dot product between two feature vectors $\langle \Phi(\boldsymbol{t}_i) \cdot \Phi(\boldsymbol{z}) \rangle$. In more general form, a kernel for all $\boldsymbol{z}, \boldsymbol{z}' \in X$ is defined as [4]:

$$K(\boldsymbol{z}, \boldsymbol{z}') = \langle \Phi(\boldsymbol{z}) \cdot \Phi(\boldsymbol{z}') \rangle. \tag{3.4}$$

Since a kernel is a generic similarity measure in Hilbert space, a kernel must satisfy following conditions [26]:
symmetry,

$$K(\boldsymbol{z}, \boldsymbol{z}') = K(\boldsymbol{z}', \boldsymbol{z})$$

inequality,

$$K(\boldsymbol{z}, \boldsymbol{z}') \leq K(\boldsymbol{z}, \boldsymbol{z}) K(\boldsymbol{z}', \boldsymbol{z}')$$

and positive semi-definiteness (Mercer's theorem [68]):
Let Z be a finite input space with $K(\boldsymbol{z}, \boldsymbol{z}')$ a symmetric function on Z. Then $K(\boldsymbol{z}, \boldsymbol{z}')$ is a kernel function if and only if the matrix

$$\mathbf{K} = (K(\boldsymbol{z}_i, \boldsymbol{z}_j))_{i,j=1}^{n}$$

is positive semi-definite (has non-negative eigenvalues). We call such a matrix *the kernel matrix*.

[3] This bypass technique is called kernel trick

[4] Although a kernel presented here is a linear kernel, there are several other kernel functions such as polynomial, radial basis function, and sigmoid kernels. All these functions are various definitions of a dot product in Hilbert space. One can choose a kernel function which best reflects desired aspects of target objects.

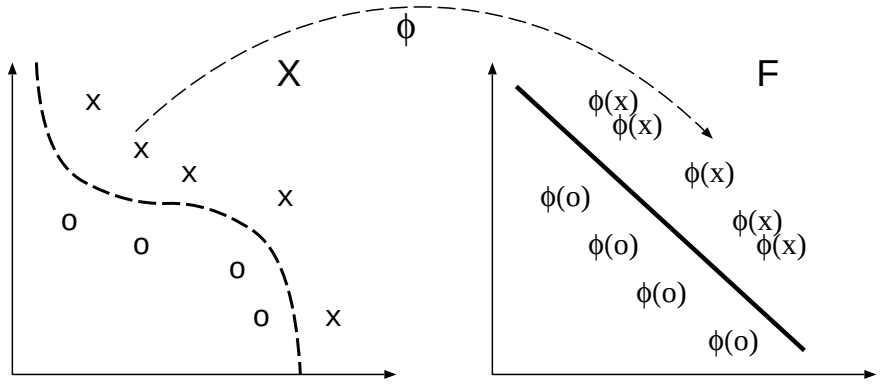


Figure 3.1: A projection into *feature space* F where "features" of data points become clear thus easily discriminated (borrowed from [26] pp. 28). $\Phi(\boldsymbol{x})$ and $\Phi(\boldsymbol{o})$ is called *feature vectors* of original data points $\boldsymbol{x}$ and $\boldsymbol{o}$.

3.3 Kernel Design for Protein/DNA Sequences

A biological sequence is a chain of molecules such as nucleotides or amino acids. In this section, the "biological sequence" is refereed to a protein or a DNA sequence. The discourse begins with designing the most simple kernel called *count kernel*. Then design of kernels based on stochastic models, specifically HMMs, which we call *the marginalized count kernels* are presented.

3.3.1 Designing Count Kernels

For example, here is a nucleotide sequence,

$$\mathrm{CGATTGCC}$$

. We count the occurrence of each nucleotide in the example sequence based on an assumption that counting each composition in a sequence reflects a basic feature of the sequence [5] Counting compositions results in a vector

$$\begin{array}{l}\mathrm{A}\times 1\\ \mathrm{C}\times 3\\ \mathrm{G}\times 2\\ \mathrm{T}\times 2\end{array}\rightarrow\begin{pmatrix}1\\3\\2\\2\end{pmatrix}$$

A dot product between two of such vectors is called a *count kernel* [6]

For a formal representation, let $x=(x_1,x_2,\cdots,x_n)$ represents a biological sequence which length is n, where $x_i\in\{1,\cdots,m\}$ denotes i-th compositional symbol. m is 4 for nucleic acid sequence or 20 for amino acid sequence. n varies per sequence. k is one of the compositional

[5] Counting compositions in biological sequences is a fundamental analysis in molecular biology. In the age of early molecular biology, many analysis were done on macro-scopic compositional aspects of biological sequences such as [99].

[6] Actual computation of count kernel involves normalizing counting vectors by multiplying normalization coefficient that is the inverse of sequence length to each compositional count.

symbols e.g. $k \in \{A, C, G, T\}$ for nucleotide sequences. Then a feature vector $\Phi(x)$ and its kernel is defined as:

$$\begin{aligned}
\Phi(x) &= (c_1(x), c_2(x), \cdots, c_m(x)) \\
c_k(x) &= \frac{1}{n} \sum_{i=1}^{n} \delta\left[x_i = k\right] \\
\delta(c) &= \begin{cases} 1 & (c = true) \\ 0 & (c = false), \end{cases}
\end{aligned}$$

where $1/n$ is a normalization factor in regards to the length of the sequence. A kernel computed with a couple of the 1st order count feature vectors is *the 1st order count kernel* which is represented as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{x}') \rangle = \sum_{k=1}^{m} c_k(\boldsymbol{x}) c_k(\boldsymbol{x}') \tag{3.5}$$

It is fairly straightforward to devise a more complex feature vector by counting the occurrence of two consecutive nucleotides in a sequence. Such a model is called a bi-gram model. It has been known that we can exploit much more crucial information of a biological sequence by using the bi-gram model [4] [58]. Counting the two consecutive nucleotides in the example sequence reads:

$$\begin{array}{l}
\mathrm{AA} \times 0 \\
\mathrm{AC} \times 0 \\
\mathrm{AG} \times 0 \\
\mathrm{AT} \times 1 \\
\mathrm{CA} \times 0 \\
\mathrm{CC} \times 1 \\
\mathrm{CG} \times 1 \\
\mathrm{CT} \times 0 \\
\mathrm{GA} \times 1 \\
\mathrm{GC} \times 1 \\
\mathrm{GG} \times 0 \\
\mathrm{TA} \times 0 \\
\mathrm{TC} \times 0 \\
\mathrm{TG} \times 1 \\
\mathrm{TT} \times 1
\end{array}
\rightarrow
\begin{pmatrix}
0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1
\end{pmatrix}$$

Thus the feature vector is 16 (4×4) dimensional. Such feature vector is defined as:

$$\begin{aligned}
\Phi(x) &= (c_{11}(x), c_{12}(x), \cdots, c_{mm}(x)) \\
c_{k\ell}(x) &= \frac{1}{n-1} \sum_{i=1}^{n-1} \delta\left[x_i = k, x_{i+1} = \ell\right],
\end{aligned} \tag{3.6}$$

and *the 2nd order count kernel* is defined as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{k=1}^{m} \sum_{\ell=1}^{m} c_{k\ell}(\boldsymbol{x}) c_{k\ell}(\boldsymbol{x}'). \tag{3.7}$$

```
                1      1  2         3
          1.3.....0......7..0.........O.........
Sequence: MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSP...
2nd.Str.:   HHHHHHHHH  EEEEEE TTS EEEETTEEEESSS...
```

Figure 3.2: A part of protein sequence and its secondary structure (a snapshot from a secondary structure assignment software DSSP [51]). Numbers presented over the sequence indicate positions of amino acids. For the label codes, H=alpha helix, E=extended strand participating in beta ladder, T=hydrogen bonded turn, S=bend and white space=no particular structure.

3.3.2 Count Kernels for Labeled Biological Sequences

Fig. 3.2 shows a part of a protein sequence along with the labels indicating its secondary structures. The labels indicate that "I" (isoleucine) at 3rd position and "I" at 17th position have different preferences because they are attached to different labels "H" and "E." We call such labels attached to a sequence *the context information*. The count kernel does not exploit much information out of a sequence because it monotonically treats every compositions without being sensitive to which *context* the symbol belongs. However, it is very important to take account of such context information especially for biological sequence data. For instance, significance of observing a certain nucleotide in a protein coding region differs from observing the same nucleotide in a non-coding region. In such a case, the context means coding/non-coding. We can exploit the context information of the sequence data when it is available. Let us represent such context information as a series of labels:

$$\boldsymbol{h} = (h_1, \cdots, h_n),\ h_i \in \{1, \cdots, m\},$$

where each label h_i is bound to i-th symbol x_i in a sequence $\boldsymbol{x}$. We use $\boldsymbol{z}$ for representing a combined sequence of the symbols and labels i.e.

$$\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{h}).$$

Then we define *the 1st order joint feature vector* as:

$$\begin{aligned} \Phi(\boldsymbol{z}) &= \left(c_1^1(\boldsymbol{z}), c_1^2(\boldsymbol{z}), \cdots, c_m^h(\boldsymbol{z})\right) \\ c_\ell^k(\boldsymbol{z}) &= \frac{1}{n}\sum_{i=1}^{n} \delta\left[x_i = k, h_i = \ell\right], \end{aligned} \tag{3.8}$$

and *the 1st order joint kernel* is defined as:

$$K(\boldsymbol{z}, \boldsymbol{z}') = \sum_{k=1}^{m}\sum_{\ell=1}^{h} c_\ell^k(\boldsymbol{z}) c_\ell^k(\boldsymbol{z}'). \tag{3.9}$$

We can take the bi-gram approach also for this. *The 2nd order joint feature vector* exploits two consecutive symbols and labels:

$$c_{uv}^{ab}(\boldsymbol{z}) = \frac{1}{n-1}\sum_{i=1}^{n-1} \delta\left[x_i = a, h_i = u, x_{i+1} = b, h_{i+1} = v\right], \tag{3.10}$$

where each of a and b represents a symbol and each of u and v is a label. *The 2nd order joint kernel* is defined as follows:

$$K(\boldsymbol{z}, \boldsymbol{z}') = \sum_{a=1}^{m}\sum_{b=1}^{m}\sum_{u=1}^{h}\sum_{v=1}^{h} c_{uv}^{ab}(\boldsymbol{z})c_{uv}^{ab}(\boldsymbol{z}'). \tag{3.11}$$

3.3.3 Count Kernels for Biological Sequences without Labels

Let us think of a case we have to deal with sequence data without knowing the context information but we try to estimate the contexts by using some prediction tools. Actually, there are varieties of prediction tools for biological sequences are available. We can estimate the coding/non-coding regions of a genomic sequence with several *ab initio* gene finding programs such as GenScan [18], Glimmer [28], GeneMark [66] etc. For protein secondary structure prediction, several tools are available: PREDATOR [40], PSSP [85], NPS [43] etc. By using the predicted context information, we are able to compute the joint kernels (3.9 and 3.11) as if the context information was given in advance.

In the case of using stochastic models where we can define the likelihood function $p(\boldsymbol{x}|\boldsymbol{h})$, the optimal labels $\hat{\boldsymbol{h}}$ such that

$$\hat{\boldsymbol{h}} = \text{argmax}_{\boldsymbol{h}}\, p(\boldsymbol{x}|\boldsymbol{h})$$

can be estimated with maximum likelihood estimation. In terms of HMM, $\hat{\boldsymbol{h}}$ can be computed with Viterbi algorithm [50].

However, since we do not know the correct $\boldsymbol{h}$, it is impossible to verify if $\hat{\boldsymbol{h}}$ is truly optimal. Besides, it is usually unlikely that a very reliable stochastic model is available to do estimations. Hence, instead of using only the most probable estimation, we take account of *all* the probable estimations over a sequence. It is inevitable to include noise with this approach, but inclusion of the correct estimation and all of its proximities is expected to reduce the risk of noise. We show how this can be done by using stochastic models.

Firstly, we replace $\delta\left[h_i = \cdots\right]$ used in Eqn. 3.8 and 3.10 with probability functions, which are defined on a stochastic model θ, respectively correspond to them. For Eqn. 3.8, we replace

$$\delta\left[h_i = \ell\right] = 0,\ 1$$

with

$$0 \leq p(h_i = \ell|\boldsymbol{x}, \theta) \leq 1, \tag{3.12}$$

which is a probability to observe a label ℓ at i-th position of the given sequence $\boldsymbol{x}$. Hence *the 1st order marginalized count feature vector* can be defined as a probabilistic rewrite of Eqn. 3.8 as:

$$\begin{aligned} v_\ell^k(\boldsymbol{x}) &= \frac{1}{n}\sum_{i=1}^{n} p(h_i = \ell|\boldsymbol{x})\delta\left[x_i = k\right] \\ &= \frac{1}{n}\sum_{i=1, x_i=k}^{n} p(h_i = \ell|\boldsymbol{x}). \end{aligned} \tag{3.13}$$

And *the 1st order marginalized count kernel* (MCK) is defined as follows:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{k=1}^{m}\sum_{\ell=1}^{h} v_\ell^k(\boldsymbol{x})v_\ell^k(\boldsymbol{x}'). \tag{3.14}$$

For Eqn. 3.10,

$$\delta\left[h_i = u, h_{i+1} = v\right] = 0,\ 1$$

is replaced with

$$0 \leq p(h_i = u, h_{i+1} = v|\boldsymbol{x}, \theta) \leq 1, \tag{3.15}$$

which is a joint probability to observe both a label u at i-th position and v at $(i+1)$-th position in $\boldsymbol{x}$. Then, as a probabilistic rewrite of Eqn. 3.10, we define *the 2nd order marginalized count feature vector* as:

$$\begin{aligned} v_{uv}^{ab}(\boldsymbol{x}) &= \frac{1}{n-1}\sum_{i=1}^{n-1} p(h_i = u, h_{i+1} = v|\boldsymbol{x})\delta\left[x_i = a, x_{i+1} = b\right] \\ &= \frac{1}{n-1}\sum_{i=1, x_i=a, x_{i+1}=b}^{n-1} p(h_i = u, h_{i+1} = v|\boldsymbol{x}). \end{aligned} \tag{3.16}$$

And *the 2nd order MCK* is defined as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{a=1}^{m}\sum_{b=1}^{m}\sum_{u=1}^{h}\sum_{v=1}^{h} v_{uv}^{ab}(\boldsymbol{x}) v_{uv}^{ab}(\boldsymbol{x}'). \tag{3.17}$$

3.3.4 Computing MCKs with a Hidden Markov Model

Readers are supposed to have knowledge on HMMs in order to understand the following discourse. Please consult with an appropriate material such as [80] for general description of HMMs.

Two stochastic parameters are required to compute MCKs. One is the probability represented as Eqn. 3.12 and the other is Eqn. 3.15. In terms of HMM, the labels as we refereed with $\boldsymbol{h}$ correspond to a series of *hidden states* of HMMs. Therefore we use a term "state" in place of "label." Here we think of a case where each symbol x_i in a sequence $\boldsymbol{x}$ corresponds to a hidden state h_i. Hence,

$$\boldsymbol{h} = (h_1, h_2, \cdots, h_n),$$

where n is the length of sequence $\boldsymbol{x}$. When the probability distribution $p(\boldsymbol{x}|\theta)$ is represented as an HMM (θ), the posterior probability [7] $p(h_i|\boldsymbol{x}, \theta)$ is computed easily by the forward-backward algorithm [31]. An HMM is described as

$$p(\boldsymbol{x}|\theta) = \sum_{\boldsymbol{h}} p(\boldsymbol{x}, \boldsymbol{h}|\theta) = \sum_{\boldsymbol{h}} q_{h_1} e_{h_1 x_1} [\prod_{i=2}^{n} a_{h_{i-1} h_i} e_{h_i x_i}] d_{h_n}, \tag{3.18}$$

where the HMM parameters $\theta = \{a, e, q, d\}$ are transition probabilities, emission probabilities, initial state distribution and terminal state distribution, respectively. Here the parameters are supposed to be optimized in advance such that

$$\boldsymbol{\theta} = \underset{\theta}{\text{argmax}}\, p(\boldsymbol{X}|\theta), \tag{3.19}$$

[7] By the nature of HMMs, we do not observe which hidden state is corresponding to x_i, but we can estimate its probability after observing $\boldsymbol{x}$. Therefore, the probability is called *posterior* probability.

where $\boldsymbol{X}$ is the training data set. The forward and backward algorithms provide the forward and backward probabilities respectively,

$$\begin{aligned}\alpha_i(\ell) &= p(x_1, \ldots, x_i, h_i = \ell), \\ \beta_i(\ell) &= p(x_{i+1}, \ldots, x_n | h_i = \ell).\end{aligned}$$

Then the posterior probability is described as

$$p(h_i = \ell | \boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})} \alpha_i(\ell)\beta_i(\ell), \tag{3.20}$$

where $p(\boldsymbol{x}|\boldsymbol{\theta})$ is a likelihood of a sequence $\boldsymbol{x}$ in respect to an HMM $\boldsymbol{\theta}$. The probability $p(h_i = u, h_{i+1} = v|\boldsymbol{x})$ is denoted as:

$$p(h_i = u, h_{i+1} = v | \boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})} \alpha_i(u) a_{uv} e_{ux_{i+1}} \beta_{i+1}(v). \tag{3.21}$$

Now one can compute MCKs over HMMs by applying Eqn. 3.20 and 3.21 to Eqn. 3.13 and 3.16 respectively. It is worth noting that Eqn. 3.20 and 3.21 are probabilities regularly computed in HMMs during training phase by using Baum-Welch algorithm [8]. Therefore, no special computation is required to prepare parameters to compute the feature vectors.

3.3.5 Connections to Fisher Kernels

We derive the Fisher kernel (FK) [47] from HMMs and discuss its connection to MCKs. The joint distribution of HMM is described as (Eqn. 3.18)

$$p(\boldsymbol{x}, \boldsymbol{h}|\boldsymbol{\theta}) = q_{h_1} e_{h_1 x_1} [\prod_{i=2}^{n} a_{h_{i-1} h_i} e_{h_i x_i}] d_{h_n},$$

where $\boldsymbol{\theta}$ is optimized in advance such that Eqn. 3.19. As in the literature [47], we take the derivatives with respect to emission probabilities e only:

$$\frac{\partial}{\partial e_{\ell k}} \log p(\boldsymbol{x}, \boldsymbol{h}|\boldsymbol{\theta}) = \frac{n v_\ell^k(\boldsymbol{x})}{e_{\ell k}} - n \sum_{k=1}^{m} v_\ell^k(\boldsymbol{x}), \tag{3.22}$$

where $v_\ell^k(\boldsymbol{x})$ is a marginalized count defined in Eqn. 3.13. Note that the second term of Eqn. 3.22 comes from the constraint of emission probabilities $\sum_{k=1}^{m} e_{\ell k} = 1$. If we do not use the Fisher information matrix as in [47], the joint kernel is described as

$$\nabla_{\boldsymbol{e}} \log p(\boldsymbol{x}, \boldsymbol{h}|\boldsymbol{\theta})^\top \nabla_{\boldsymbol{e}} \log p(\boldsymbol{x}', \boldsymbol{h}'|\boldsymbol{\theta}). \tag{3.23}$$

This is rewritten as

$$\sum_{k=1}^{m} \sum_{\ell=1}^{h} \frac{n n'}{e_{\ell k}^2} (v_\ell^k(\boldsymbol{x}) - \bar{v}_\ell^k(\boldsymbol{x}))(v_\ell^k(\boldsymbol{x}') - \bar{v}_\ell^k(\boldsymbol{x}')), \tag{3.24}$$

where

$$\bar{v}_\ell^k(\boldsymbol{x}) = e_{\ell k} \sum_{k'=1}^{m} v_\ell^{k'}(\boldsymbol{x}).$$

This has a similar form to the marginalized count kernel (3.14), however the count is centralized and the dot product is taken with respect to the weight $\frac{nn'}{e_{\ell k}^2}$. The weight is dependent on the length n, so a proper normalization is needed for the Fisher kernel. Since $e_{\ell k}$ represent the emission probability that symbol k is produced from state ℓ, the weight becomes large when the symbol k is rarely produced from state ℓ. It makes sense, because the occurrence of a rare symbol is a strong clue of high similarity. However this weight is still arguable, because a huge weight can appear when $e_{\ell k}$ is very small. In addition, FK has potential advantages when applied to biological sequences where the background noise has to be concerned in general.

3.4 Computational Experiments

In this section, we evaluate the performance of marginalized kernels in classification experiments using bacterial *gyrB* amino acid sequences. *gyrB* [54] - gyrase subunit B - is a DNA topoisomerase (type II) which plays essential roles in fundamental mechanisms of living organisms such as DNA replication, transcription, recombination and repair etc. One more important feature of *gyrB* is its capability of being an evolutionary and taxonomic marker alternating popular 16S rRNA. Our dataset consists of 84 *gyrB* amino acid sequences from five genera in *Actinobacteria* which are *Corynebacterium*, *Mycobacterium*, *Gordonia*, *Nocardia* and *Rhodococcus*, respectively [55]. For brevity these genera will be called genus 1 to 5, respectively. The number of sequences in each genus is listed as 9, 32, 15, 14 and 14. The sequences are, by their nature, quite similar in terms of sequence similarity. Pairwise identity for each sequence is at least 62% and 99% at most. For computing distance matrix based on the sequence similarity, one can use the BLAST scores [2]. However, since such scores cannot directly be converted into positive semi-definite kernels, kernel methods cannot be applied to them in principle.

In order to investigate how well the kernels reflect underlying genera, we performed two kinds of experiments – clustering and supervised classification. The following kernels are compared:

- CK1: 1st order count kernel (Eqn. 3.5)
- CK2: 2nd order count kernel (Eqn. 3.7)
- FK: Fisher kernel (Eqn. 3.24)
- MCK1: 1st order marginalized count kernel (Eqn. 3.14)
- MCK2: 2nd order marginalized count kernel (Eqn. 3.17)

As the first experiment, K-Means clustering is performed in feature spaces corresponding to kernels (see [71] for details). The number of clusters are determined as five (i.e. the true number). In FK and MCKs, we used complete-connection HMMs with 3,5 and 7 states. Note that FK is normalized by the sequence lengths. In training HMMs, all 84 sequences are used. One can also train HMMs in a class-wise manner [102]. However, we did not do so because the number of sequences is not large enough. For evaluating clusters, we used the adjusted Rand index (ARI) [108]. The advantage of this index is that you can compare two partitions whose number of clusters are different. The ARI becomes 1 if the partitions are completely correct. Also, the expectation of the ARI is 0 when partitions are randomly determined.

The kernel matrices by FK and MCKs are shown in Fig. 3.4. Additionally, the ideal kernel is shown for reference, where $K(\boldsymbol{x}, \boldsymbol{x}')$ is 1 for any two sequences in the same genus, and -1

Table 3.1: Mean error rates (%) of supervised classification between two bacterial genera ([·] shows the standard deviation). The best result in each task is written in bold face.

Genera	CK1	CK2	FK	MCK1	MCK2
3-4	24.5 [9.67]	9.10 [7.87]	10.4 [9.15]	12.8 [9.85]	**8.48** [7.76]
3-5	12.7 [8.93]	6.43 [7.76]	10.9 [10.1]	10.4 [8.17]	**5.71** [7.72]
4-5	25.6 [13.0]	13.5 [15.5]	23.1 [14.3]	20.0 [14.6]	**11.6** [14.6]

otherwise. Here, the number of HMM states is three in all cases. For fair visualization, each kernel matrix is normalized in the same manner: First, the kernel matrix is "centralized" as

$$K_c := K - 1_n K - K 1_n + 1_n K 1_n,$$

where 1_n is the $n \times n$ matrices whose elements are all $1/n$. Here n denotes the number of sequences, i.e. $n = 84$ in this experiment. Then, K_c is normalized by the Frobenious norm as $K_c / \|K_c\|_F$. As seen in the figure, MCK2 is the best to recover the underlying structure. This result is quantitatively shown by ARI in Fig. 3.4, where CK1 and CK2 correspond to the MCKs with only one HMM state. Notably FK was worse than MCK1, which shows that the joint kernel of the FK (3.24) is not appropriate for this task.

In order to see how genera are separated by introducing the second order information and hidden variables, we performed the following supervised classification experiments as well. First, we pick up two genera out of three genera (3,4,5). Genera 1 and 2 were not used because they can be separated easily by all kernels. The sequences of two genera are randomly divided into 25% training and 75% testing samples. Kernels are compared due to the test error by the kernel Fisher discriminant analysis (KFDA) [84], which compares favorably with the SVM in many benchmarks. Note that the regularization parameter ϵ of KFDA [84] is determined such that the test error is minimized [8]. The test errors of five kernels are shown in Table 3.4. The second order kernels (i.e. CK2 and MCK2) were significantly better than the first order kernels. This result coincides with the common understanding that higher order information of protein sequences is essential for classification and structure prediction (e.g. [4]). Comparing CK2 and MCK2, MCK2 always performed better, which indicates that incorporating hidden variables (i.e. context information) is meaningful at least in this task.

3.5 Summary

In this chapter, we proposed a novel way to design kernels for biological sequences that is *the marginalized kernels*, which utilizes parameters of stochastic models in order to extract context information of the target biological sequences. Started with a very simple kernel called *count kernel*, step-by-step derivation of *the marginalized count kernel* (MCK) is shown including the case when we use HMMs for stochastic models. In addition, the Fisher kernel (FK) was described as a special case of marginalized kernels, This gives a new viewpoint to the theory of FK. We showed that MCKs performed well in protein classification experiments, where MCKs yielded better performances than the FK.

[8] For regularization parameter ϵ, 10 equally spaced points on the log scale are taken from $[10^{-4}, 10^{2}]$. Among these candidates, the optimal one is chosen.

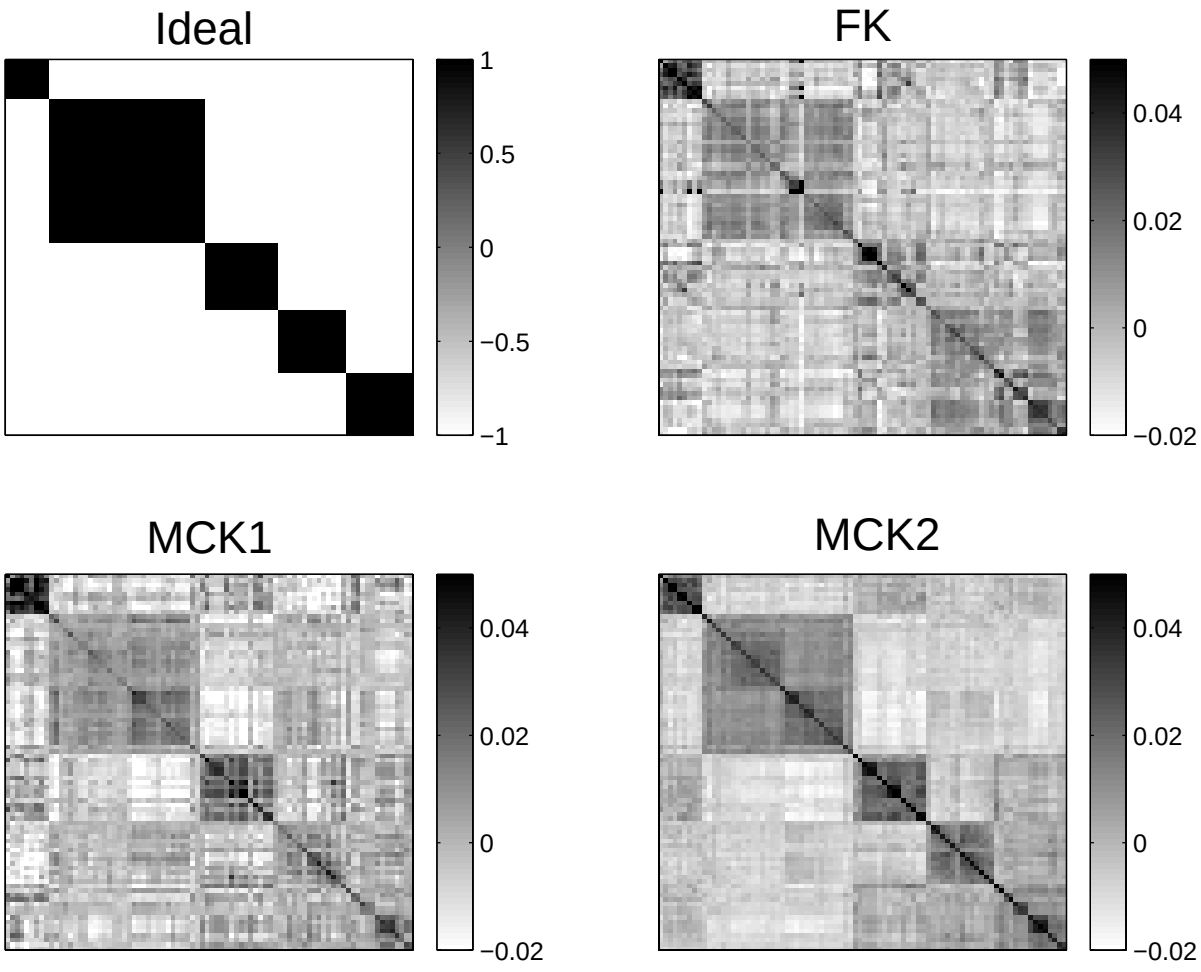


Figure 3.3: (Upper left) Ideal kernel matrix to illustrate the true clusters. (Upper right) Kernel matrix of the Fisher kernel. (Lower left) Kernel matrix of the first-order marginalized count kernel. (Lower right) Kernel matrix of the second-order marginalized count kernel.

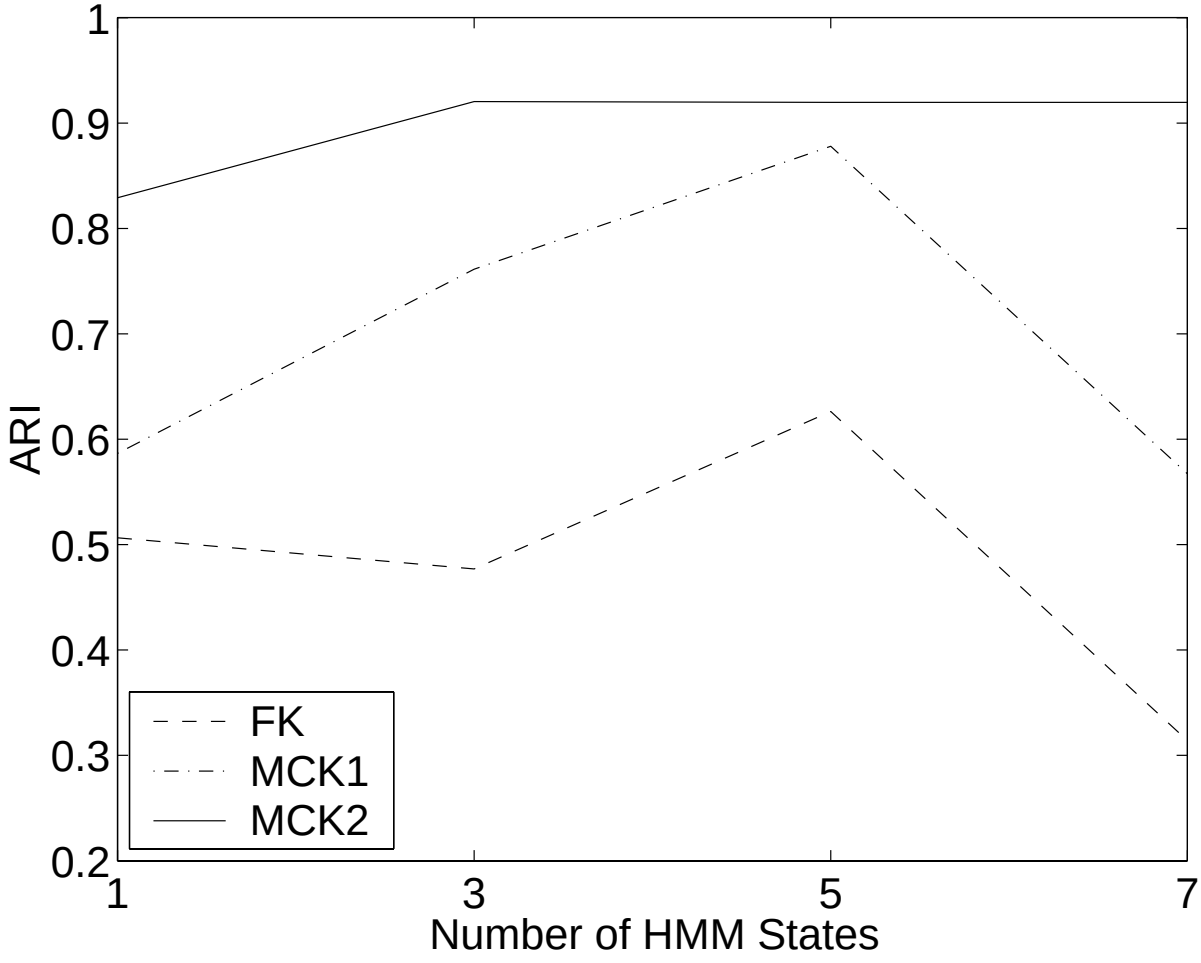


Figure 3.4: Evaluation of kernels in clustering in terms of the adjusted Rand index (ARI). The x-axis corresponds to the number of states in HMM, from which the kernels are derived. The Fisher kernel (FK), the marginalized count kernels of first-order (MCK1) and second-order (MCK2) are compared. Note that the count kernels of first-order (CK1) and second-order (CK2) correspond to MCK1 and MCK2 at one HMM state, respectively.

Chapter 4

Marginalized Kernels for RNA Sequence Data Analysis

4.1 Introduction

Due to the recent success of the support vector machines (SVMs) in many practical problems, emerging kernel methods are appealing in bioinformatics. For instance, SVMs are well known for their performance superiority in many real-world problems such as text-categorization, pattern recognition and, especially now, biological sequence data analysis. Examples for biological sequence data analysis are protein family clustering [53], promoter classification[77], protein homology detection[47], translation initiation site prediction[110] and splice site recognition[94].

However, to our knowledge, kernel methods have not been applied to RNA sequence data analysis. The primary reason is that it is hard to define a kernel function to reflect the RNA secondary structure. For defining the similarity (or kernel) between two RNAs, simple sequence comparison (e.g BLAST) is never enough because the sequences can form stems, hairpins, bulges etc (see 4.2 for details of RNA secondary structure). In addition, string kernels [104] including the spectrum kernel [63] do not define RNA similarities because they do not take into account the secondary structure.

We propose a new method for designing a kernel for RNAs. First we will consider a case in which the secondary structures of two RNAs are known and represented using context-free grammar (CFG) where a state (or a non-terminal) is associated with one or two base(s) in an RNA sequence. A feature vector is constructed by counting the base-state combinations and the kernel is defined as the dot product between two vectors. We call this the "count kernel for RNAs". This concept can be generalized to take into account the consecutive two base-state combinations into. We call this kernel the "2nd order count kernel for RNAs". However, it is often the case that the RNA secondary structure is not known, but estimated with some probabilistic model such as stochastic context-free grammar (SCFG). In such cases, we use the expectation of the count kernel with respect to the secondary structure. We call this the "marginalized count kernel for RNAs". Similarly, a second-order version can be obtained. Note that these kernels are generalized versions of the kernels proposed for HMMs by Tsuda *et al* [103]. We performed computational experiments using human tRNA sequence data, which are a visualization of sequence similarities using kernel PCA [89] and a supervised classification using SVMs. For the latter, we compared the performance of the classifications with MCKs and with SCFG likelihood.

Rest of this chapter is organized as follows. Firstly, brief description of RNA secondary

structure is given in Sec. 4.2. Then in Sec. 4.3, we deal with a formal grammar that represents secondary structure. Some essential parts of SCFG is described in Sec. 4.4. Sec. 4.5 presents details of kernel PCA that is one of the powerful feature extraction methods. Sec. 4.6 describes details of designing our novel kernel for RNA sequences. Sec. 4.7 presents several performance tests for our kernels. We summarize this chapter in Sec. 4.8.

4.2 RNA Secondary Structure

RNA secondary structure is composed primarily of double-stranded RNA regions formed by folding the single-stranded molecule back on itself. To produce such double-stranded regions, a run of bases downstream in the RNA sequence must be complementary to another upstream run so that Watson-Crick base pairing between the complementary nucleotides G/C and A/U (analogous to the G/C and A/T base pairs in DNA) can occur. In addition, however, G/U wobble pairs may be produced in these double-stranded regions. As in DNA, the G/C base pairs contribute the greatest energetic stability to the molecule, with A/U base pairs contributing less stability than G/C, and G/U wobble base pairs contributing the least. From the RNA structures that have been solved, these base pairs and a number of additional ones (see [20, 19]) have been identified. RNA structure predictions comprise base-paired and non-base-paired regions in various types of loop and junction arrangements, as shown in Fig. 4.2.

4.3 Grammar of RNA

Fig. 4.2 (left) shows a simple RNA sequence which has a hairpin-like structure (middle). The example secondary structure can be represented with a set of state transitions associated with symbols:

$$S \rightarrow R_1, R_1 \rightarrow P_1 a, P_1 \rightarrow g P_2 c, P_2 \rightarrow g P_3 c, P_3 \rightarrow g L_1 c, L_1 \rightarrow c L_2, L_2 \rightarrow a L_3, L_3 \rightarrow u E,$$

where P represents the state corresponds to a base pair. R and L correspond to the states which emit single base to right side and left side, respectively. S and E are special states not associated to any symbols but represent start and end of transition, respectively. The set of generative rules is interpreted as: R_1 emits "a" to right and makes transition to P_1, then P_1 emits "g" to left and "c" to right simultaneously then move to P_2, and so on. *The CFG matrix* shown in Fig. 4.2(right) gives another way to represent state-symbol associations as well as state transitions, where transitions start at upper right corner of the 10×10 matrix (size is determined by the target sequence length), R comes first cell at $(1, 10)$ which corresponds to "a" in 10th position of the sequence, then P comes in a cell at $(1, 9)$ which is associated to the 1st "g" and 9th "c." Another P fills $(2, 8)$ which corresponds to the 2nd "g" and 8th "c." One more P is in $(3, 7)$ and corresponds to 3rd "g" and 7th "c". Then three Ls fill in $(4, 6)$, $(5, 6)$, $(6, 6)$ and correspond to 4th "c", 5th "a", 6th "u" respectively.

We generalize the above generative rules by defining six types of states as follows (W represents any of the states except S):

- S: start $S \rightarrow W$
  a special state which means the beginning of the secondary structure.
- B: bifurcation $B \rightarrow WW$
  makes transition to two states.

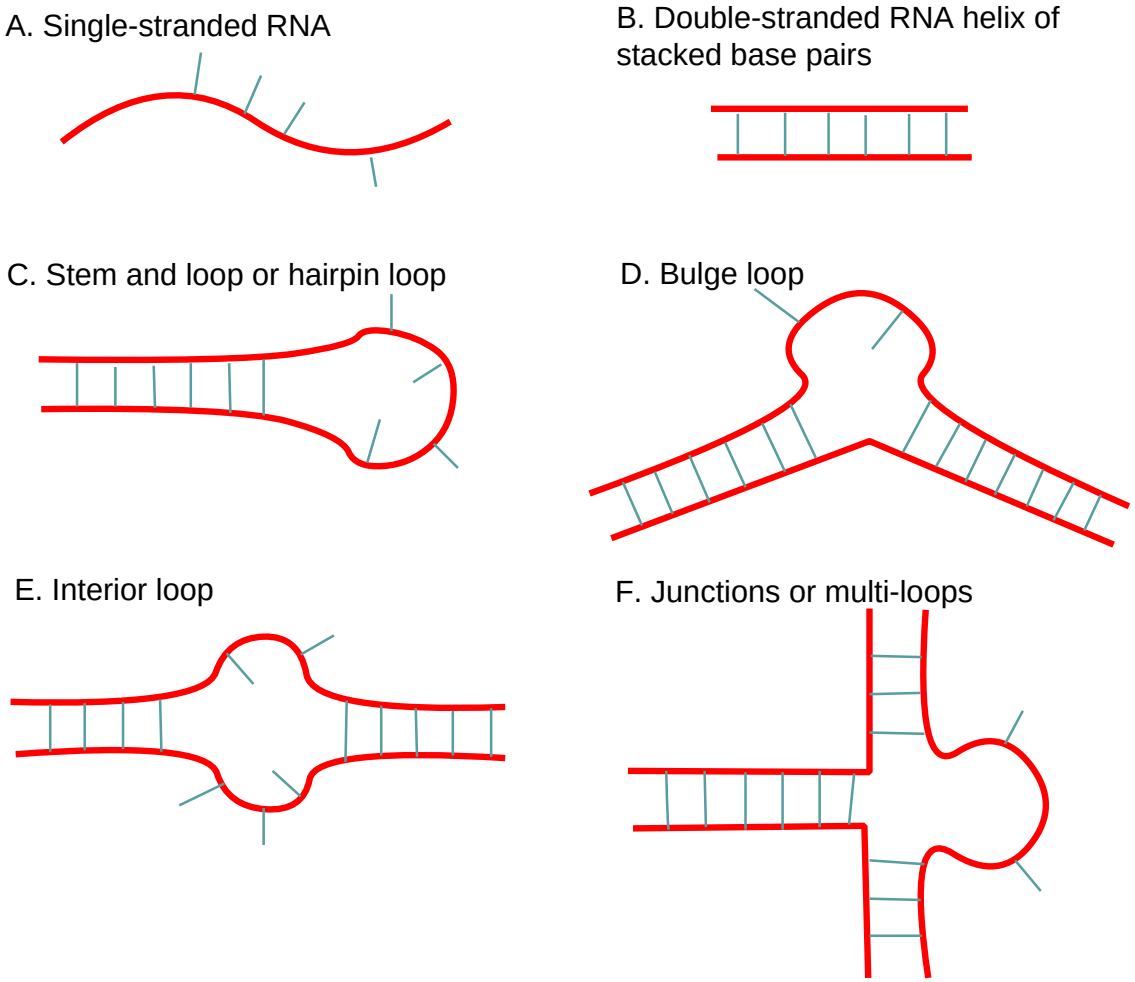


Figure 4.1: Single-stranded RNA molecules fold back on themselves and produce double-stranded helices where complementary sequences are present. A particular base may either not be paired, as in A, or paired with another bases as in B. The double-stranded regions will most likely form where a series of bases in the sequence can pair with a complementary set elsewhere in the sequence. The stacking energy of the base pairs provides increased energetic stability. Combinations of double-stranded and single-stranded regions produce the types of structures shown in C-F, with the single-stranded regions destabilizing neighboring double-stranded regions. The loop of the stem and loop in C must generally be at least four bases long to avoid steric hindrance with base-pairing in the stem part of the structure. The stem and loop reverses the chemical direction of the RNA molecule. Interior loops, as in D, form when the bases in a double-stranded region cannot form base pairs, and many be symmetric with a different number of base pairs on each side of the loop, as shown in E, or symmetric with same number on each side. Junctions, as in F, may include two or more double-stranded regions converging to form a closed structure. The RNA backbone is red, and both unpaired and paired bases are blue. The types of loop structures can be represented mathematically, thereby aiding in the prediction of secondary structure [87, 111].

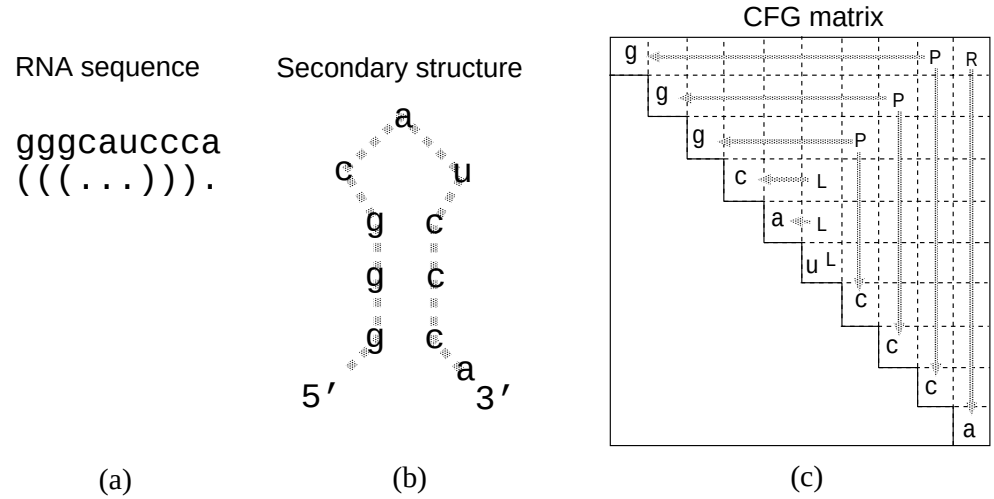


Figure 4.2: An example of an RNA sequence and a representation of the secondary structure using *the CFG matrix*. (a) a tiny RNA sequence and its secondary structure where base pairs are represented with parentheses (b) a hairpin-like structure of the RNA (c) the secondary structure represented in *the CFG matrix*, where RNA sequence is aligned in the diagonal cells, and states are assigned to each cell so that each state-symbol association becomes obvious (please see text for details). The shaded arrows show one or two symbols correspond to a certain state.

- P: pair-wise $P \rightarrow aWb$
  emits two symbols a to left and b to right. $ab \in \{AU, UA, CG, GC\}$
- L: left $L \rightarrow aW$
  emits single symbol a to left. $a \in \{A, C, G, U\}$
- R: right $R \rightarrow Wa$
  emits single symbol a to right.
- E: end $E \rightarrow \epsilon$
  a special state which means the end of the secondary structure.

We do not consider transitions $S \rightarrow E$ and $B \rightarrow EE$ because these transitions does not make sense. Generalized grammar for RNA secondary structure is defined with a following set of generative rules:

$$\begin{aligned} S &\rightarrow W \\ B &\rightarrow WW \\ P &\rightarrow aWb \\ L &\rightarrow aW \\ R &\rightarrow Wa \\ W &= B|P|L|R|E \end{aligned} \tag{4.1}$$

Table 4.1: Values of $\triangle^V_{\ell|r}$ which represent if the state V emits a symbol to left or right. "1" indicates the occurrence of emission while "0" indicates no emission.

v	P	L	R
$\triangle^v_L$	1	1	0
$\triangle^v_R$	1	0	1

4.4 SCFG

SCFG Θ is defined with several sets of parameters denoted as $\Theta = \{\boldsymbol{S}, \Sigma, t, e\}$: $\boldsymbol{S}$ is a set of states, Σ is a set of symbols, t is a set of transition probabilities and e is a set of emission probabilities. We denote a transition probability from state V to Y as $t_V(Y)$ ($V, Y \in \boldsymbol{S}$). Some states emit two symbols respectively to left-hand side and right-hand side simultaneously which we call *the pair-wise emission*. We use P to represent one of such states. P corresponds to a base pair in an RNA sequence. Some other states emit one symbol to left-hand side. Such states are represented as L. R is used to represent states that emit a symbol to right-hand side. So emission probability at state P is written as $e_P(a, b)$ ($a, b \in \Sigma$), where a for left-hand and b for right-hand side. Otherwise $e_V(a)$ for $V \in \{L, R\}$. L and R correspond to bases that are not involved in base pairs. SCFG allows to define transition and emission probabilities to the generative rules Eqn. 4.1. There are several standard algorithms for SCFG. Inside-outside algorithm [62] provides two important parameters i.e. inside and outside parameters. Inside parameter, we denote it as $\alpha_W(i, j)$ represents a probability that all probable successive transitions from a state W occur within a sub-sequence $[i, j]$, $(i \leq j)$. Outside parameter, we denote it as $\beta_W(i, j)$ represents a probability that all probable preceding transitions to a state W occurred outside of a sub-sequence $[i, j]$. By using these parameters, we can compute an important probability that a state V is associated to i th and j th bases of a sequence $\boldsymbol{x}$:

$$p(W(i, j) = V|\boldsymbol{x}, \Theta) = \frac{1}{p(\boldsymbol{x}|\Theta)}\alpha_V(i, j)\beta_V(i, j). \tag{4.2}$$

Please remember the CFG matrix in which state-symbol associations are deterministically represented. The concept of using SCFG is to represent the state-symbol association stochasticly by using the probability $p(W(i, j) = V|\boldsymbol{x}, \Theta)$. Another important probability is a probability that a state V is associated to i th and j th bases in a sequence $\boldsymbol{x}$ and the consecutive state W is associated to $i + \triangle^V_L$ the and $j - \triangle^V_R$ the bases:

$$p(W(i, j) = V, W(i + \triangle^V_L, j - \triangle^V_R) = Y|\boldsymbol{x}, \Theta) = \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})}\beta_V(i, j)t_V(Y)\alpha_Y(i + \triangle^V_L, j - \triangle^V_R), \tag{4.3}$$

where $\triangle^V = 0,\ 1$ depends on type of V (see Table. 4.4).

SCFG parameters are trained with an expectation-maximization learning method using the standard inside-outside algorithm. Maximizing the likelihood of each sequence leads to maximizing the expectation to use the state P because the pair-wise emission is more economic than are the single emission states such as L and R to annotate the sequence. Therefore, the expectation-maximization learning corresponds to the Nussinov [72] algorithm which estimates RNA secondary structure in a way to maximize the occurrence of base-pairs. However, the same learning algorithm behaves like the Zuker algorithm [112] with appropriate parameter settings in regard to the stacking energies (see Fig. 4.2).

Major drawback of SCFG comes from its computational complexity. Its memory complexity is $O(L^2M)$ where L is length of the target RNA sequence and M is number of states. Besides, its time complexity is $O(L^3M^3)$.

4.5 Kernel PCA

Principal component analysis (PCA) is one of the most popular techniques for feature extraction. When using PCA for feature extraction, it amounts to a *linear* projection of the data onto the principal subspace. The standard formulation of PCA is the eigendecomposition of the covariance matrix of the data. Here we illustrate that PCA can also be carried out on the kernel matrix as shown in [89].

Let $\{\boldsymbol{x}^n\}$ be a data set with N examples of dimension d which is mapped into feature space $\{F|\Phi(\boldsymbol{x}^n)\}$ by non-linear projection $\Phi : \Re^d \rightarrow F$. We suppose the mapped data to be centered: $\sum_n \Phi(\boldsymbol{x}^n) = \mathbf{0}$. [1] The matrix $\boldsymbol{X} = [\Phi(\boldsymbol{x}^1), \Phi(\boldsymbol{x}^2), \cdots, \Phi(\boldsymbol{x}^n)]$ represents the data in a compact way. Standard PCA is based on finding the eigenvalues and orthonormal eigenvectors of the (sample) covariance matrix in the feature space where the covariance matrix can be written as:

$$\boldsymbol{C} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top. \tag{4.4}$$

We are interested in the dot product matrix of size $N \times N$:

$$\boldsymbol{K} = \frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}, \tag{4.5}$$

where $\boldsymbol{K}$ is the kernel matrix since

$$K_{ij} = \frac{1}{N}\langle\Phi(\boldsymbol{x}^i)\cdot\Phi(\boldsymbol{x}^j)\rangle = \frac{1}{N}K(\boldsymbol{x}^i, \boldsymbol{x}^j),$$

where $K : \Re^d \times \Re^d \rightarrow \Re$ is the kernel function. Kernel PCA is based on the fact that there is a one-to-one correspondence between the non-zero eigenvectors $\boldsymbol{v}^k$ of $\boldsymbol{C}$ and the non-zero eigenvectors $\boldsymbol{u}^k$ of $\boldsymbol{K}$ and that they have the same eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_p$ (of course, $p \leq \min(d, N)$):

$$\boldsymbol{v}^k = \boldsymbol{X}\boldsymbol{u}^k/\sqrt{\lambda_k N} \tag{4.6}$$

$$\boldsymbol{u}^k = \boldsymbol{X}^\top\boldsymbol{v}^k/\sqrt{\lambda_k N}, \tag{4.7}$$

where the scaling by $\sqrt{\lambda_k N}$ normalizes the eigenvectors. Thus, the principal eigenvectors of the covariance matrix of the mapped data lie in the span of the Φ-images of the training data.
Proof :
Let $\boldsymbol{v}$ be an eigenvector of the covariance matrix $\boldsymbol{C}$ in F with eigenvalue λ: $\frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{v} = \lambda\boldsymbol{v}$. Then:

$$\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{v}) = \boldsymbol{X}^\top(\frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{v}) = \lambda\boldsymbol{X}^\top\boldsymbol{v},$$

so λ is also an eigenvalue of the kernel matrix $\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}$ with corresponding eigenvector $\boldsymbol{X}^\top\boldsymbol{v}$ provided $\boldsymbol{X}^\top\boldsymbol{v} \neq 0_N$ which follows from:

$$\lambda \neq 0 \;\Rightarrow\; \lambda\boldsymbol{v} \neq 0_d \;\Leftrightarrow\; \boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{v} \neq 0_d \;\Leftarrow\; \boldsymbol{X}^\top\boldsymbol{v} \neq 0_N.$$

[1] In general, this is not the case. However, all calculations can be reformulated to include centered data without having to calculate explicitly the mean in F [89].

So we only have to take the non-zero eigenvectors into account. By symmetry (in $\boldsymbol{X}$ and $\boldsymbol{X}^\top$), we can also conclude that each non-zero eigenvector of the kernel matrix $\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{u} = \lambda\boldsymbol{u}$ corresponds to a non-zero eigenvector $\boldsymbol{X}\boldsymbol{u}$ of the covariance matrix with eigenvalue λ. The one-to-one correspondence as stated in Eqn. 4.6 and 4.7 follows after a straightforward normalization of the eigenvectors. Given normalized eigenvectors $\boldsymbol{u}$ for the kernel matrix, one can normalize the eigenvectors for the covariance matrix:

$$\mathrm{norm}(\boldsymbol{v}) = \boldsymbol{v}^\top\boldsymbol{v} = \boldsymbol{u}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{u} = \lambda N\boldsymbol{u}^\top\boldsymbol{u} = \lambda,$$

and the other way round. □

A direct consequence of this one-to-one correspondence is that one can perform kernel PCA feature extraction entirely in terms of kernel functions. It requires determining the orthonormal eigenvectors $\boldsymbol{u}^k$ of $\boldsymbol{K}$ and its eigenvalues λ_k, and projecting a point $\Phi(\boldsymbol{x})$ onto the principal eigenvectors $\boldsymbol{v}^k$ in feature space as defined by Eqn. 4.7 (leaving out the normalization):

$$\Phi(\boldsymbol{x})\cdot\boldsymbol{v}^k = \left[\sum_{i=1}^{N} u_i^k\{\Phi(\boldsymbol{x})\cdot\Phi(\boldsymbol{x}^i)\}\right] = \left[\sum_{i=1}^{N}\boldsymbol{u}_i^k K(\boldsymbol{x},\boldsymbol{x}^i)\right]. \tag{4.8}$$

Kernel PCA corresponds exactly to linear PCA in the high-dimensional feature space F and, therefore, has all the properties of PCA in F. Because of the non-linearity of the map Φ, the features are extracted in data space in a non-linear way: the contour lines of constant projections onto a principal eigenvector are non-linear in data space [89].

4.6 Kernel Design for RNA Sequences

4.6.1 Count Kernels for RNAs

We design the count kernel for RNAs which secondary structures are known. Computation of count kernels for RNAs follows quite similar way as shown in the previous section. We use the symbol-emitting three states i.e. P, L and R as "labels." With an RNA sequence shown in figure 4.2, the occurrences of the base-state combinations count; $(a,R)\times 1, (gc,P)\times 3, (c,L)\times 1, (a,L)\times 1, (u,L)\times 1$, while the other combinations count zero. These numbers build up a 12 ($=4\times 3$) dimensional vector (for the dimension, 4 bases for each of L and R, and 4 canonical base pairs for P). W refers to CFG matrix and $W(i,j)$ represents a label at j-th column of i-th row. The 1st order count feature vectors are defined as:

$$c_P^{ab}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\delta\left[W(i,j)=P, x_i=a, x_j=b\right] \tag{4.9}$$

$$c_L^{a}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i}^{n}\delta\left[W(i,j)=L, x_i=a\right] \tag{4.10}$$

$$c_R^{a}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i}^{n}\delta\left[W(i,j)=R, x_j=a\right] \tag{4.11}$$

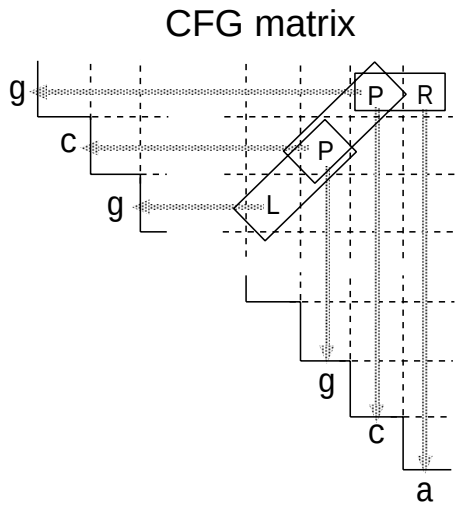


Figure 4.3: The counting scheme of the 2nd order count feature vector from a CFG matrix. An exemplar sate-path is shown as R, P, P and L. Notice that each state is associated to one or two bases with shaded arrows. The rectangles surrounding every pair of states indicate how consecutive two states are counted. It reads $[(R,a),(P,gc)]$, $[(P,gc),(P,cg)]$, $[(P,gc),(L,g)]$, and so on. There are 144 possible combinations to be counted. Thus 2nd order count feature vector has 144 dimensions.

$\frac{1}{n}$ is a normalization factor in regard to length of the sequence. The 1st order count kernel is defined as:

$$K(\boldsymbol{z},\boldsymbol{z}') = \sum_{V\in\{P,L,R\}} C_V(\boldsymbol{z},\boldsymbol{z}'), \tag{4.12}$$

$$C_V(\boldsymbol{z},\boldsymbol{z}') = \begin{cases} V=P: & \sum_{ab\in\boldsymbol{\Omega}} c_P^{ab}(\boldsymbol{z})c_P^{ab}(\boldsymbol{z}') \\ & \boldsymbol{\Omega}=\{AU,UA,CG,GC,GU,UG\} \\ V=L,R: & \sum_{a\in\{A,C,G,U\}} c_V^{a}(\boldsymbol{z})c_V^{a}(\boldsymbol{z}') \end{cases}$$

We design the 2nd order count kernel by taking account of two consecutive base-state pairs. Fig. 4.6.1 shows the counting scheme using CFG matrix. Counting two consecutive base-state pairs for an example RNA sequence looks like this: $[(R,a),(P,gc)]\times 1, [(P,gc),(P,gc)]\times 2, [(P,gc),(L,c)]\times 1, [(L,c),(L,a)]\times 1, [(L,a),(L,u)]\times 1$. Then these numbers build up a 144 ($3^2\times 4^2$) dimensional vector accounting three states and four kinds of symbols. We denote these vectors as follows:

$$c_{PP}^{abcd}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^P}\sum_{j=i+1+\triangle_r^P}^{n} d_{PP}^{abcd}(i,j) \tag{4.13}$$

$$d_{PP}^{abcd}(i,j) \equiv \delta\Big[W(i,j)=P, W(i+\triangle_\ell^P, j-\triangle_r^P)=P, x_i=a, x_j=b, x_{i+\triangle_\ell^P}=c, x_{j-\triangle_r^P}=d\Big]$$

$$c_{PL}^{abc}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^P}\sum_{j=i+1+\triangle_r^P}^{n} d_{PL}^{abc}(i,j) \tag{4.14}$$

$$d_{PL}^{abc}(i,j) \equiv \delta\Big[W(i,j)=P, W(i+\triangle_\ell^P, j-\triangle_r^P)=P, x_i=a, x_j=b, x_{i+\triangle_\ell^P}=c\Big]$$

$$c_{PR}^{abd}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^P}\sum_{j=i+1+\triangle_r^P}^{n} d_{PR}^{abd}(i,j) \tag{4.15}$$

$$d_{PR}^{abd}(i,j) \equiv \delta\Big[W(i,j)=P, W(i+\triangle_\ell^P, j-\triangle_r^P)=P, x_i=a, x_j=b, x_{j-\triangle_r^P}=d\Big]$$

$$c^{acd}_{LP}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle^L_\ell}\sum_{j=i+1+\triangle^L_r}^{n} d^{acd}_{LP}(i,j) \tag{4.16}$$

$$d^{acd}_{LP}(i,j) \equiv \delta\left[W(i,j)=L, W(i+\triangle^L_\ell, j-\triangle^L_r)=P, x_i=a, x_{i+\triangle^L_\ell}=c, x_{j-\triangle^L_r}=d\right]$$

$$c^{ac}_{LL}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle^L_\ell}\sum_{j=i+1+\triangle^L_r}^{n} d^{ac}_{LL}(i,j) \tag{4.17}$$

$$d^{ac}_{LL}(i,j) \equiv \delta\left[W(i,j)=L, W(i+\triangle^L_\ell, j-\triangle^L_r)=L, x_i=a, x_{i+\triangle^L_\ell}=c\right]$$

$$c^{ad}_{LR}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle^L_\ell}\sum_{j=i+1+\triangle^L_r}^{n} d^{ad}_{LR}(i,j) \tag{4.18}$$

$$d^{ad}_{LR}(i,j) \equiv \delta\left[W(i,j)=L, W(i+\triangle^L_\ell, j-\triangle^L_r)=R, x_i=a, x_{j-\triangle^L_r}=d\right]$$

$$c^{bcd}_{RP}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle^R_\ell}\sum_{j=i+1+\triangle^R_r}^{n} d^{bcd}_{RP}(i,j) \tag{4.19}$$

$$d^{bcd}_{RP}(i,j) \equiv \delta\left[W(i,j)=R, W(i+\triangle^R_\ell, j-\triangle^R_r)=P, x_j=b, x_{i+\triangle^R_\ell}=c, x_{j-\triangle^R_r}=d\right]$$

$$c^{bc}_{RL}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle^R_\ell}\sum_{j=i+1+\triangle^R_r}^{n} d^{bc}_{RL}(i,j) \tag{4.20}$$

$$d^{bc}_{RL}(i,j) \equiv \delta\left[W(i,j)=R, W(i+\triangle^R_\ell, j-\triangle^R_r)=L, x_j=b, x_{i+\triangle^R_\ell}=c\right]$$

$$c^{bd}_{RR}(\boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle^R_\ell}\sum_{j=i+1+\triangle^R_r}^{n} d^{bd}_{RR}(i,j) \tag{4.21}$$

$$d^{bd}_{RR}(i,j) \equiv \delta\left[W(i,j)=R, W(i+\triangle^R_\ell, j-\triangle^R_r)=R, x_j=b, x_{j-\triangle^R_r}=d\right],$$

where $\triangle^V_\ell$ and $\triangle^V_r$ are binary flags indicating whether V emits a symbol to left ($\triangle^V_\ell$) and right ($\triangle^V_r$) See table 4.4 for actual values. We define the 2nd order count kernels as:

$$K(\boldsymbol{z},\boldsymbol{z}') = \sum_{VY\in\boldsymbol{\Psi}} C_{VY}(\boldsymbol{z},\boldsymbol{z}'), \tag{4.22}$$

$$\boldsymbol{\Psi} = \{PP, PL, PR, LP, LL, LR, RP, RL, RR\}$$

$$C_{VY}(\boldsymbol{z},\boldsymbol{z}') = \begin{cases} VY=PP: & \sum_{abcd\in\boldsymbol{\Omega}\times\boldsymbol{\Omega}} c^{abcd}_{PP}(\boldsymbol{z})c^{abcd}_{PP}(\boldsymbol{z}') \\ VY=PL: & \sum_{abc\in\boldsymbol{\Omega}\times\boldsymbol{\Sigma}} c^{abc}_{PL}(\boldsymbol{z})c^{abc}_{PL}(\boldsymbol{z}') \\ VY=PR: & \sum_{abd\in\boldsymbol{\Omega}\times\boldsymbol{\Omega}} c^{abd}_{PR}(\boldsymbol{z})c^{abd}_{PR}(\boldsymbol{z}') \\ VY=LP: & \sum_{acd\in\boldsymbol{\Sigma}\times\boldsymbol{\Omega}} c^{acd}_{LP}(\boldsymbol{z})c^{acd}_{LP}(\boldsymbol{z}') \\ VY=LL: & \sum_{ac\in\boldsymbol{\Sigma}\times\boldsymbol{\Sigma}} c^{ac}_{LL}(\boldsymbol{z})c^{ac}_{LL}(\boldsymbol{z}') \\ VY=LR: & \sum_{ad\in\boldsymbol{\Sigma}\times\boldsymbol{\Omega}} c^{ad}_{LR}(\boldsymbol{z})c^{ad}_{LR}(\boldsymbol{z}') \\ VY=RP: & \sum_{bcd\in\boldsymbol{\Sigma}\times\boldsymbol{\Omega}} c^{bcd}_{RP}(\boldsymbol{z})c^{bcd}_{RP}(\boldsymbol{z}') \\ VY=RL: & \sum_{bc\in\boldsymbol{\Sigma}\times\boldsymbol{\Sigma}} c^{bc}_{RL}(\boldsymbol{z})c^{bc}_{RL}(\boldsymbol{z}') \\ VY=RR: & \sum_{bd\in\boldsymbol{\Sigma}\times\boldsymbol{\Sigma}} c^{bd}_{RR}(\boldsymbol{z})c^{bd}_{RR}(\boldsymbol{z}'). \end{cases} \tag{4.23}$$

4.6.2 Marginalized Count Kernels for RNAs

We design the marginalized count kernels for RNAs which secondary structures are unknown. In this case, each cell of the CFG matrix contains a probability $p(W(i,j) = V|\boldsymbol{x})$ instead of states. Since we do not know the secondary structure, $\delta[W(i,j) = V]$ used for Eqn. 4.9 through 4.11 need to be replaced with the probability. The probability is computed as follows using the inside (α) and outside (β) parameters of SCFG:

$$p(W(i,j) = V|\boldsymbol{x}) = \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})}\alpha_V(i,j)\beta_V(i,j),$$

which is one of commonly computed parameters for SCFG learning with inside-outside algorithm. Using the probability, we define *the 1st order marginalized count feature vectors* for RNAs as follows:

$$\begin{aligned}
g_P^{ab}(\boldsymbol{x}) &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i}^{n} p(W(i,j) = P|\boldsymbol{x})\delta\left[x_i = a, x_j = b\right] \\
g_L^{a}(\boldsymbol{x}) &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i}^{n} p(W(i,j) = L|\boldsymbol{x})\delta\left[x_i = a\right] \\
g_R^{a}(\boldsymbol{x}) &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i}^{n} p(W(i,j) = R|\boldsymbol{x})\delta\left[x_j = a\right]
\end{aligned}$$

Hence *the 1st order marginalized count kernel* is defined as:

$$K(\boldsymbol{x},\boldsymbol{x}') = \sum_{V\in\{P,L,R\}} G_V(\boldsymbol{x},\boldsymbol{x}'), \tag{4.24}$$

$$G_V(\boldsymbol{x},\boldsymbol{x}') = \begin{cases} V = P: & \sum_{ab\in\boldsymbol{\Omega}} g_P^{ab}(\boldsymbol{x})g_P^{ab}(\boldsymbol{x}') \\ & \boldsymbol{\Omega} = \{AU, UA, CG, GC, GU, UG\} \\ V = L, R: & \sum_{a\in\{A,C,G,U\}} g_V^{a}(\boldsymbol{x})g_V^{a}(\boldsymbol{x}') \end{cases} \tag{4.25}$$

For the 2nd order count kernel, we need to represent $\delta[W(i,j) = V, W(i+\triangle_\ell^V, j-\triangle_r^V) = Y]$ used for Eqn. 4.13 through 4.21 in a probabilistic manner i.e. $p(W(i,j) = V, W(i+\triangle_\ell^V, j-\triangle_r^V) = Y|\boldsymbol{x})$. Let us refer to the probability as $\xi_{VY}(i,j)$. The probability is computed using the common SCFG parameters as follows:

$$\xi_{VY}(i,j) = \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})}\beta_V(i,j)t_V(Y)\alpha_Y(i+\triangle_L^V, j-\triangle_R^V). \tag{4.26}$$

Then we define *the 2nd order marginalized count kernel* as follows:

$$g_{PP}^{abcd}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^P}\sum_{j=i+1+\triangle_r^P}^{n} d_{PP}^{abcd}(i,j) \tag{4.27}$$

$$d_{PP}^{abcd}(i,j) \equiv \xi_{PP}(i,j)\delta\left[x_i = a, x_j = b, x_{i+\triangle_\ell^P} = c, x_{j-\triangle_r^P} = d\right]$$

$$g_{PL}^{abc}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^P}\sum_{j=i+1+\triangle_r^P}^{n} d_{PL}^{abc}(i,j) \tag{4.28}$$

$$d_{PL}^{abc}(i,j) \equiv \xi_{PL}(i,j)\delta\left[x_i = a, x_j = b, x_{i+\triangle_\ell^P} = c\right]$$

$$g_{PR}^{abd}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^P}\sum_{j=i+1+\triangle_r^P}^{n} d_{PR}^{abd}(i,j) \tag{4.29}$$
$$d_{PR}^{abd}(i,j) \equiv \xi_{PR}(i,j)\delta\left[x_i = a, x_j = b, x_{j-\triangle_r^P} = d\right]$$

$$g_{LP}^{acd}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^L}\sum_{j=i+1+\triangle_r^L}^{n} d_{LP}^{acd}(i,j) \tag{4.30}$$
$$d_{LP}^{acd}(i,j) \equiv \xi_{LP}(i,j)\delta\left[x_i = a, x_{i+\triangle_\ell^L} = c, x_{j-\triangle_r^L} = d\right]$$

$$g_{LL}^{ac}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^L}\sum_{j=i+1+\triangle_r^L}^{n} d_{LL}^{ac}(i,j) \tag{4.31}$$
$$d_{LL}^{ac}(i,j) \equiv \xi_{LL}\delta\left[x_i = a, x_{i+\triangle_\ell^L} = c\right]$$

$$g_{LR}^{ad}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^L}\sum_{j=i+1+\triangle_r^L}^{n} d_{LR}^{ad}(i,j) \tag{4.32}$$
$$d_{LR}^{ad}(i,j) \equiv \xi_{LR}(i,j)\delta\left[x_i = a, x_{j-\triangle_r^L} = d\right]$$

$$g_{RP}^{bcd}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^R}\sum_{j=i+1+\triangle_r^R}^{n} d_{RP}^{bcd}(i,j) \tag{4.33}$$
$$d_{RP}^{bcd}(i,j) \equiv \xi_{RP}(i,j)\delta\left[x_j = b, x_{i+\triangle_\ell^R} = c, x_{j-\triangle_r^R} = d\right]$$

$$g_{RL}^{bc}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^R}\sum_{j=i+1+\triangle_r^R}^{n} d_{RL}^{bc}(i,j) \tag{4.34}$$
$$d_{RL}^{bc}(i,j) \equiv \xi_{RL}(i,j)\delta\left[x_j = b, x_{i+\triangle_\ell^R} = c\right]$$

$$g_{RR}^{bd}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n-1-\triangle_\ell^R}\sum_{j=i+1+\triangle_r^R}^{n} d_{RR}^{bd}(i,j) \tag{4.35}$$
$$d_{RR}^{bd}(i,j) \equiv \xi_{RR}(i,j)\delta\left[x_j = b, x_{j-\triangle_r^R} = d\right].$$

Then *the 2nd order marginalized count kernel* is defined as follows:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{VY \in \boldsymbol{\Psi}} G_{VY}(\boldsymbol{x}, \boldsymbol{x}'), \tag{4.36}$$

$$\boldsymbol{\Psi} = \{PP, PL, PR, LP, LL, LR, RP, RL, RR\}$$

$$G_{VY}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} VY = PP: & \sum_{abcd \in \boldsymbol{\Omega} \times \boldsymbol{\Omega}} g_{PP}^{abcd}(\boldsymbol{x}) g_{PP}^{abcd}(\boldsymbol{x}') \\ VY = PL: & \sum_{abc \in \boldsymbol{\Omega} \times \boldsymbol{\Sigma}} g_{PL}^{abc}(\boldsymbol{x}) g_{PL}^{abc}(\boldsymbol{x}') \\ VY = PR: & \sum_{abd \in \boldsymbol{\Omega} \times \boldsymbol{\Omega}} g_{PR}^{abd}(\boldsymbol{x}) g_{PR}^{abd}(\boldsymbol{x}') \\ VY = LP: & \sum_{acd \in \boldsymbol{\Sigma} \times \boldsymbol{\Omega}} g_{LP}^{acd}(\boldsymbol{x}) g_{LP}^{acd}(\boldsymbol{x}') \\ VY = LL: & \sum_{ac \in \boldsymbol{\Sigma} \times \boldsymbol{\Sigma}} g_{LL}^{ac}(\boldsymbol{x}) g_{LL}^{ac}(\boldsymbol{x}') \\ VY = LR: & \sum_{ad \in \boldsymbol{\Sigma} \times \boldsymbol{\Omega}} g_{LR}^{ad}(\boldsymbol{x}) g_{LR}^{ad}(\boldsymbol{x}') \\ VY = RP: & \sum_{bcd \in \boldsymbol{\Sigma} \times \boldsymbol{\Omega}} g_{RP}^{bcd}(\boldsymbol{x}) g_{RP}^{bcd}(\boldsymbol{x}') \\ VY = RL: & \sum_{bc \in \boldsymbol{\Sigma} \times \boldsymbol{\Sigma}} g_{RL}^{bc}(\boldsymbol{x}) g_{RL}^{bc}(\boldsymbol{x}') \\ VY = RR: & \sum_{bd \in \boldsymbol{\Sigma} \times \boldsymbol{\Sigma}} g_{RR}^{bd}(\boldsymbol{x}) g_{RR}^{bd}(\boldsymbol{x}'). \end{cases} \tag{4.37}$$

4.7 Computational Experiments

4.7.1 Clustering Human tRNA Sequence Data

We performed kernel PCA using *the marginalized count kernels* in order to visualize similarities among human tRNA sequences.

tRNAs We use 3 class 74 Human tRNAs: 25 Asn-GTT, 26 Ala-AGC, and 24 Cys-GCA, retrieved from GtRDB [65] (`http://rna.wustl.edu/GtRDB/`). These amino-acid/codon combinations are chosen because they are the top three of the most popular combinations in Human tRNAs. The sequence data are processed not to include any identical sequences. Thus these 75 sequences are all unique.

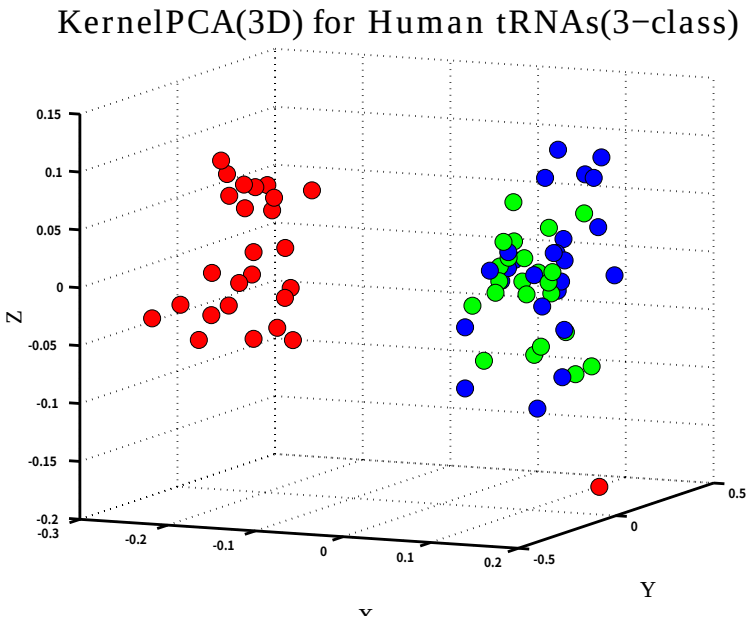
Method We use SCFG for RNAs presented in the previous section. SCFG is trained using the 75 tRNA sequence data with uniform initial parameters. Based on the trained SCFG, we compute 1st and 2nd order MCKs. Using the MCKs, kernel PCA is performed.

Result Fig. 4.4 shows results of kernel PCA based on 1st order MCK (upper) and 2nd order MCK (bottom). It is obvious that, for 1st order MCK, the black boxes (Asn-GTT) and white boxes (Ala-AGC) are separated effectively while plus symbols (Cys-GCA) are entirely submerged in the white boxes. The bottom figure shows very preferable result which almost all points are clearly and correctly clustered into three aggregations.

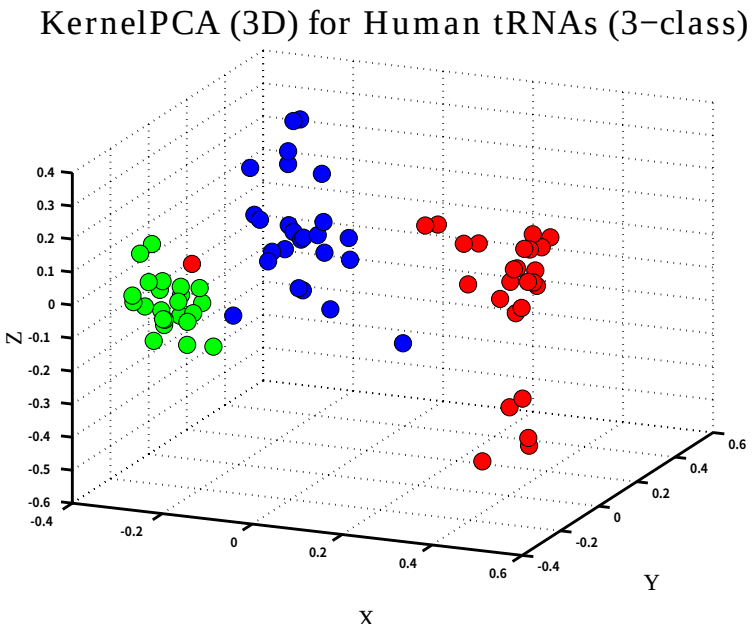
4.7.2 Clustering snoRNA Sequence Data

We performed kernel PCA for yeast snoRNAs using MCK in order to examine if our snoRNA grammar works for feature extraction of snoRNAs.

snoRNAs We use 2 class 66 Yeast snoRNAs: 20 H/ACAs, 46 BOX-C/Ds (see Fig. 4.5) retrieved from Yeast snoRNA Database [86].



(a)



(b)

Figure 4.4: Results of kernel PCA for 74 Human tRNA sequence data. The upper figure shows a result based on 1st-order MCK. The bottom represents 2nd-order MCK result where three clusters are almost clearly separated.

Method We use fairly generic grammar for snoRNAs shown as a graph in Fig. 4.6 based on secondary structures illustrated in Fig. 4.5, which is devised for this purpose. By using this grammar, SCFG is trained using the 66 snoRNA sequence data with uniform initial parameters. Based on the trained SCFG, we compute 2nd order MCKs. Then kernel PCA is performed.

Result Fig. 4.7 shows results of kernel PCA, where two classes are mostly separated.

4.7.3 Supervised classification

Method In supervised classification based on SCFG likelihood, an SCFG is trained with the training sequences per sequence class. In the classification of a test sequence, the likelihoods of all SCFGs are computed and the sequence is assigned to the class with the largest likelihood.

On the other hand, when SVMs are used for the classification, the MCKs (1st and 2nd order) are first computed as in the kernel PCA case (i.e. SCFG is trained using 74 sequences). Then SVMs are trained due to the "one-against-others" scheme: each SVM is trained with the training sequences of a class as positive samples and those of the other classes as negative samples. In classification of a test sequence, the outputs of all SVMs are computed and the sequence is assigned to the class with the largest output.

For the cross-validation, we divided 74 sequences into training/testing data as follows: 10 sequences per class, thus 30 sequences, are randomly chosen for testing; the rest are used for training. In both the likelihood and SVM experiments, the training/testing stage is iterated 250 times.

Results In Fig. 4.8, 4.9 and 4.10, the averaged ROC curves for each of the three classes are shown. Note that ROC curves are plots of the fraction of false-positives (FFP) $\frac{fp}{tn+fp}$ versus the fraction of true-positives (FTP) $\frac{tp}{tp+fn}$. The 2nd order MCK yielded the best results. Although the curves tend to fluctuate due to small number of test data, overall performance differences are obvious. The 2nd order MCK exibited stable performance which constantly scores over 0.8 FTP at zero false positives (FFP=0). The performance of the 1st order MCK is not as strong. This result corresponds with the observations we made on the kernel PCA. In fact, the 1st order MCK tends to perform worse than the raw likelihood classification at the point of zero false positives; and in the case of Cys-GCA against others, it performs worse than the raw likelihood classification.

4.8 Summary

We propose a new method for designing a kernel for RNAs. First we will consider a case in which the secondary structure of two RNAs is known and represented using context-free grammar (CFG) where a state (or a non-terminal) is associated with one or two base(s) in an RNA sequence. A feature vector is constructed by counting the base-state combinations and the kernel is defined as the dot product between two vectors. We call this the "count kernel for RNAs". This concept can be generalized to take into account the consecutive two base-state combinations into. We call this kernel the "2nd order count kernel for RNAs". However, it is often the case that the RNA secondary structure is not known, but estimated with some probabilistic model such as stochastic context free grammar (SCFG). In such cases, we use the expectation of the count kernel with respect to the secondary structure. We call this the

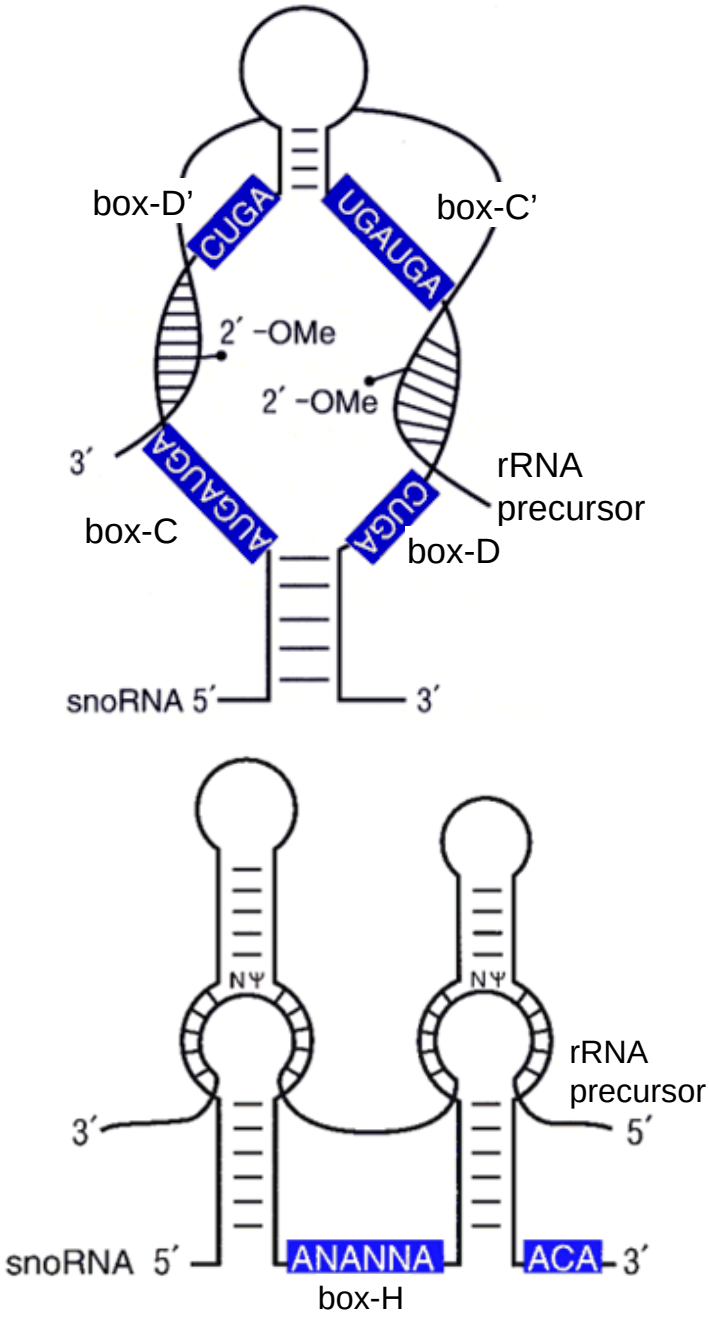


Figure 4.5: Illustrations of secondary structures of snoRNAs: BOX-C/D (top) and H/ACA (bottom).

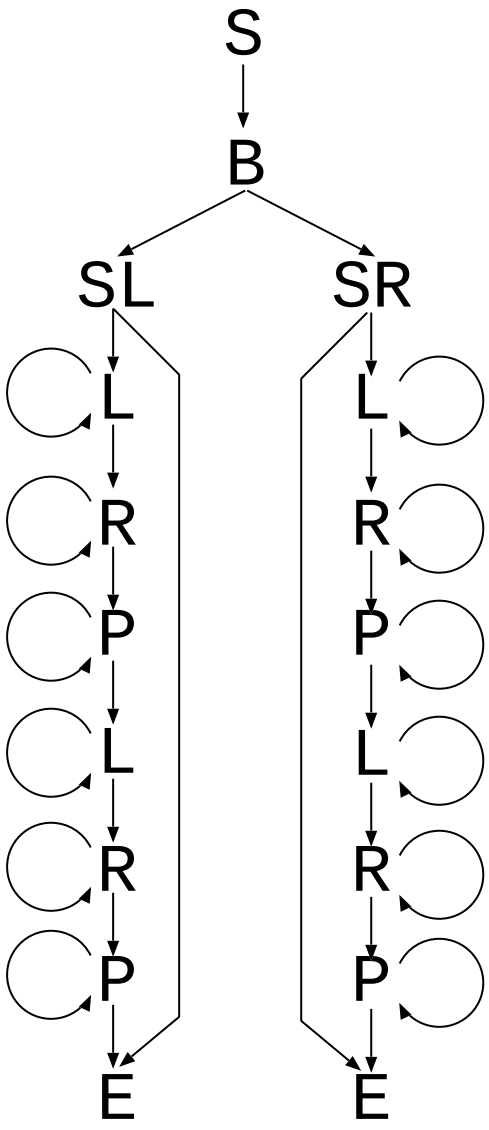


Figure 4.6: A graphical representation of a grammar for snoRNAs, which consists of two hairpin-like structures and intends to capture both of BOX-C/D (single hairpin) and H/ACA (dual hairpin) types by allowing immediate termination (a transition to end state) from bifurcation top (SL/SR).

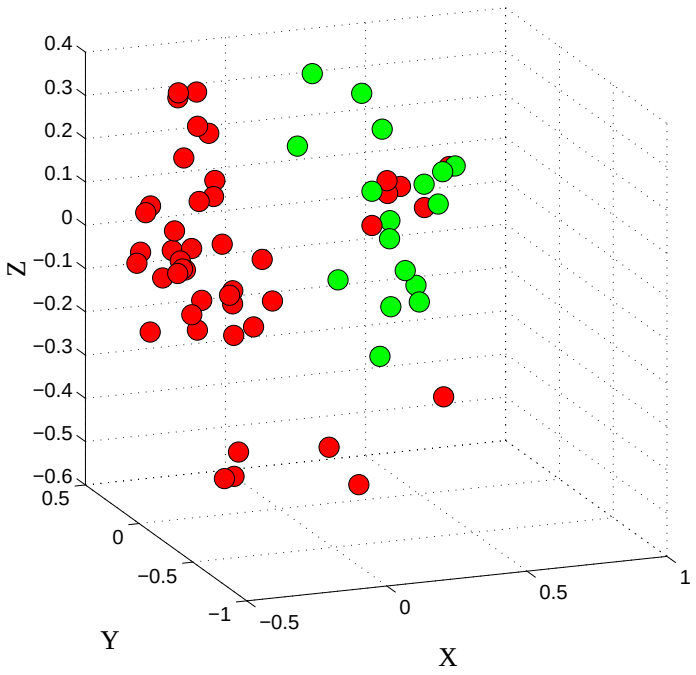


Figure 4.7: Results of kernel PCA for Yeast snoRNAs of two classes: box-C/D (red) and box-H/ACA (green). Two classes are mostly separated.

"marginalized count kernel for RNAs". Similarly, a second-order version can be obtained. Note that these kernels are generalized versions of the kernels proposed for HMMs by Tsuda *et al* [103]. We performed computational experiments using human tRNA sequence data, which are a visualization of sequence similarities using kernel PCA and a supervised classification using SVMs. For the latter, we compared the performance of the classifications with and without MCKs.

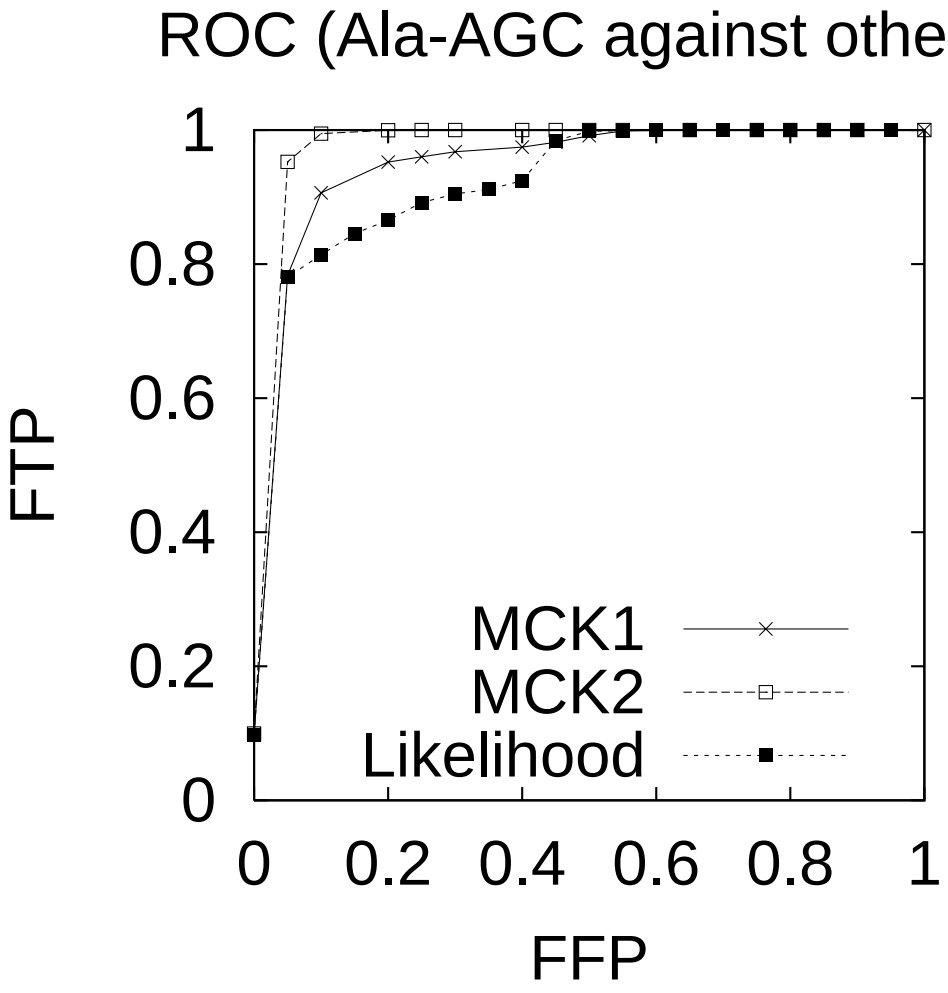


Figure 4.8: ROC curves from the supervised classifications are shown. Each curves shows the result of “one class against others” for Ala-AGC tRNAs.

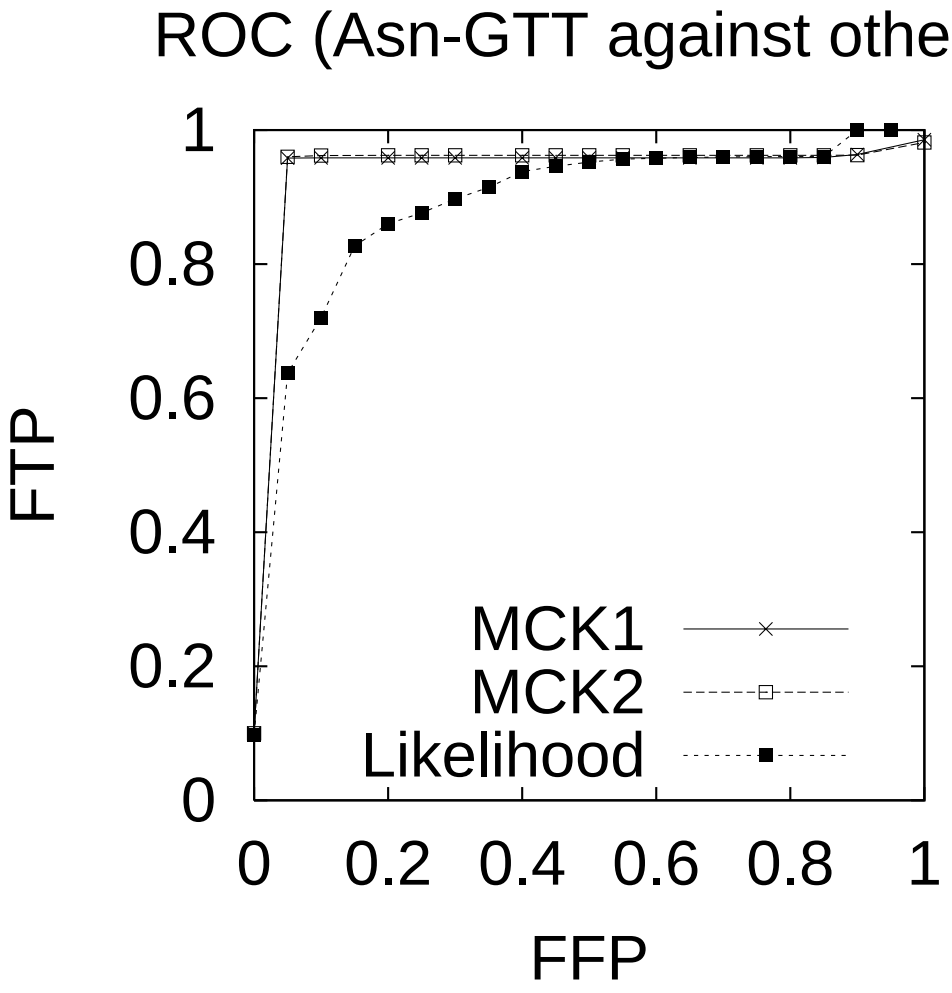


Figure 4.9: ROC curves from the supervised classifications are shown. Each curves shows the result of “one class against others” for Asn-GTT tRNAs.

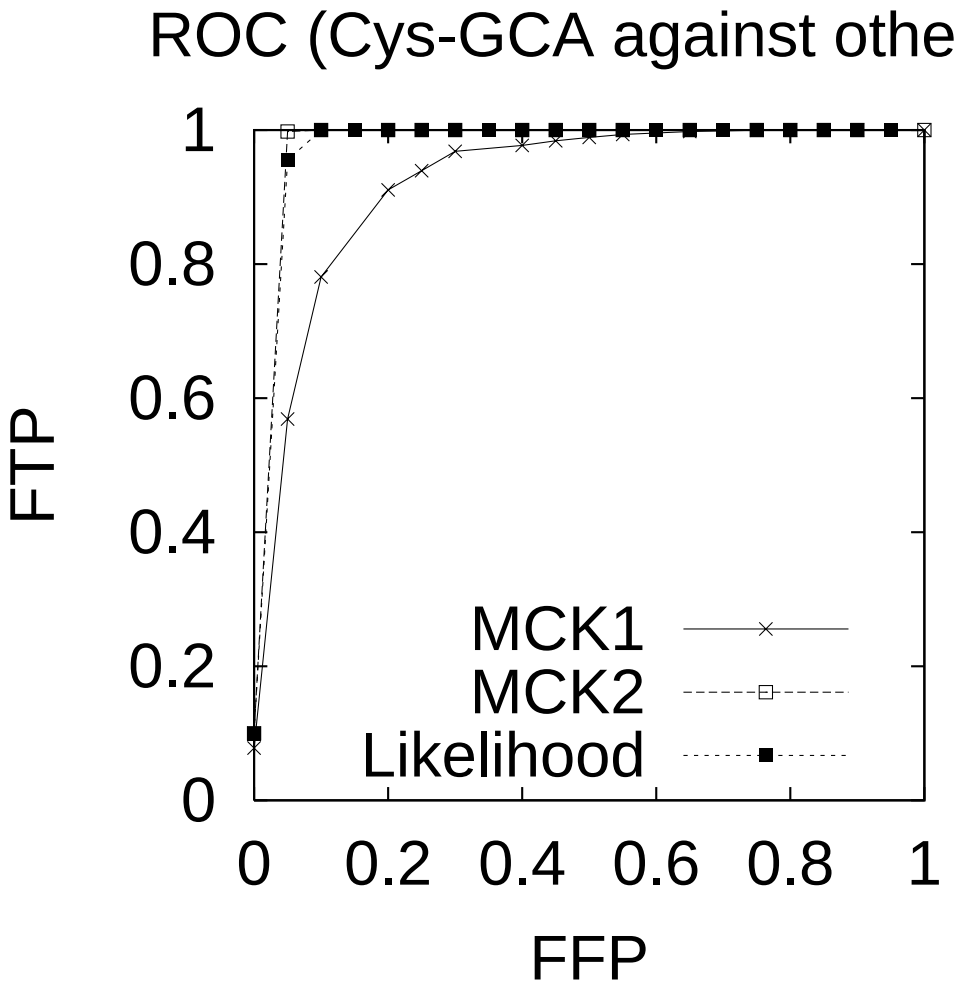


Figure 4.10: ROC curves from the supervised classifications are shown. Each curves shows the result of “one class against others” Cys-GCA tRNAs.

Chapter 5

Conclusion

Let us summarize all the topics discussed in this thesis. The objective of this research is to develop novel methods to extract features of biological sequence data in general. We began our research with investigating a discriminative property known as *dicodon usage*, the best discriminator to distinguish protein coding regions and non-coding regions. Aim of the investigation is to clarify relation between dicodon usage and other biological properties by comparing performances of discrimination among dicodon usage discriminator and several dicodon approximators devised according to some biological properties. The investigation was carried out fairly comprehensively by using seventeen microbial and several eukaryotic genomic sequence data. Although some approximators sometimes scored as good as dicodon usage does, no approximators could replace dicodon based discriminator. Therefore the dicodon usage cannot be interpreted with the approximators we devised. This experimental results indicate that the dicodon usage has some particular aspects yet to be known in terms of biology. We decided to broaden our view angle because although dicodon usage is still the best player in the field of practical gene finding, its use is limited to provide statistical criteria for discriminating protein coding and non-coding regions. Therefore we started to find a method to extract features of biological sequence data from a fundamental point of view.

We proposed a novel method to extract features of biological sequence data that is a general framework to design similarity measures. This framework allows to define two intrinsic aspects, i.e. feature representation and similarity measure definition, to quantify a similarity between two biological sequences in terms of their biological features such as secondary/tertiary structures of proteins or exon–intron boundaries of DNAs. Feature representation is provided by using latent variable models such as hidden Markov models (HMMs), and similarity measure is defined by using a kernel which is a generic framework for similarity measures. In order to evaluate validity of our method, we performed computational experiments to classify *gyrB* protein sequence data. The experiments show that our method can be a powerful approach for practical biological sequence data analysis. This framework provides a method for deriving varieties of kernels by utilizing latent variable models. And we showed that the framework embrace Fisher kernels as its subset. Thus it provides a mathematical basement for Fisher kernels and other derivatives such as *marginalized count kernels* (MCKs). One of the major advantages of using kernel is that it allows exploiting kernel methods such as support vector machines, kernel principal component analysis, kernel k-means plus varieties of kernel methods for biological sequence analysis. Since kernel methods are showing better performances in a variety of practical pattern recognition problems than traditional methods such as neural networks and HMMs (Actual examples are too many to be listed. Please consult with appropriate references

such as [88] or website at `http://kernel-machines.org/`), utilizing kernel methods for biological sequence data analysis makes significant sense.

We developed a novel method to define similarities between RNA sequence data by utilizing stochastic context free grammar (SCFG) that is capable of describing secondary structures of RNAs. We demonstrated performance of our method by doing clustering experiments of human transfer RNAs (tRNA) with kernel PCA. The experiments show promising results. We apply our method to more practical problem that is to extract features of small nucleolar RNAs (snoRNA). snoRNAs are small RNAs that play an important role in a splicing reaction of ribosomal RNA precursors. The major difficulty of feature extraction from snoRNAs is that their common secondary structures are not known well. However, the experiments show that our method successfully captured features of snoRNAs. This study showed that effectiveness of our kernel design framework by showing an actual realization of kernels utilizing SCFG, another latent variable model. Additionally, it is worth noting that the 2nd order MCK possibly exploits the stacking energy which is indispensable for the secondary structure. Although a detailed modeling of RNAs requires massive implementation of SCFG states [81], the 2nd order MCK facilitates feature extraction considering the stacking energy even with a simple SCFG that does not have the energy information. This is because, in terms of the marginalized kernel, the design of a kernel and a model are separated [103]. Besides, the introduction of MCKs sheds a new light on RNA sequence data analysis. Without the kernel, the performance of a stochastic model such as SCFG relies on a simple discriminator called likelihood, which is essentially a weighted average of all parameters. In fact, the limitation of the likelihood is the major issue when it comes to a practical analysis against large scale data, for instance, the bold challenge issued by Rivas and Eddy [82]. Therefore, allowing further exploitation of the parameters using MCK makes sense.

The major contributions of this thesis in the field of computational biology are proposition of *marginalized kernels* and derivation of several practical kernels, which introduced novel and extensible methods for biological sequence analysis.

Bibliography

[1] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 11, 1994.

[2] S. Altschul, W. Gish, E. Myers, and D. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[3] S. Andersson and et al. The genome sequence of rickettsia prowazekii and the origin of mitochondria. *Nature*, 396(6707):133–40, 1998.

[4] K. Asai, S. Hayamizu, and K. Handa. Prediction of protein secondary structure by the hidden markov model. *CABIOS (currently Bioinformatics)*, 9(2):141–146, 1993.

[5] K. Asai, K. Itou, Y. Ueno, and T. Yada. Recognition of human genes by stochastic parsing. In *Pacific Symposium on Biocomputing*, pages 228–239, 1998.

[6] K. Asai, Y. Ueno, K. Itou, and T. Yada. Automatic gene recognition without using training data. In *Genome Informatics*, volume 8, pages 15–24, 1997.

[7] S. Audic and J. Claverie. Self-identification of protein-coding regions in microbial genomes. *Proc Natl Acad Sci USA*, 95(17):10026–31, 1998.

[8] L. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[9] L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41(1):164–71, 1970.

[10] Y. Bengio, Y. LeCun, and D. Henderson. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks and hidden markov models. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 937–944, 1994.

[11] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, and D. Wheeler. Genbank. *Nucleic Acids Res*, 27(1):12–7, 1999.

[12] V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3D models. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 251–256, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[13] F. Blattner and et al. The complete genome sequence of escherichia coli k-12. *Science*, 277(5331):1453–74, 1997.

[14] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th International Conference on Pattern Recognition and Neural Networks, Jerusalem*, pages 77–87. IEEE Computer Society Press, 1994.

[15] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, M. Ares, and D. Haussler. Support vector machine classification of microarray gene expression data. Technical report, University of California, Santa Cruz, 1999.

[16] M. Brown, R. Hughey, A. Krogh, I. Mian, K. Sjolander, and H. Haussler. Using dirichlet mixture priors to derive hidden markov models for protein families. In *Proc Int Conf Intell Syst Mol Biol*, volume 1, pages 47–55, 1993.

[17] C. Bult and et al. Complete genome sequence of the methanogenic archaeon, methanococcus jannaschii. *Science*, 273(5278):1058–73, 1996.

[18] C. Burge. *Identification of Genes in Human Genomic DNA (Doctoral Thesis)*. Stanford University, March 1997.

[19] M. Burkhard, D. Turner, and I. T. Jr. *Appendix 2: Schematic diagrams of secondary and tertiary structure elements*. Cold Spring Harbor Laboratory Press, 1999.

[20] M. Burkhard, D. Turner, and I. T. Jr. *The interactions that shape RNA secondary structure*. Cold Spring Harbor Laboratory Press, 1999.

[21] M. Burset and R. Guigo. Evaluation of gene structure prediction programs. *Genomics*, 34:353–67, 1996.

[22] G. Churchill. Stochastic models for heterogeneous dna sequences. *Bull. Math. Biol.*, 51:79–94, 1989.

[23] S. Cole and et al. Deciphering the biology of mycobacterium tuberculosis from the complete genome sequence. *Nature*, 393(6685):537–44, 1998.

[24] J. Collad-Vides. A syntactic representation of units of genetic information–a syntax of units of genetic information. *J Theor Biol*, 148(3):401–29, Feb. 1991.

[25] J. Collado-Vides. Towards a unified grammatical model of sigma 70 and sigma 54 bacterial promoters. *Biochimie*, 78(5):351–63, 1996.

[26] N. Cristianini and J. Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[27] G. Deckert and et al. The complete genome of the hyperthermophilic bacterium aquifex aeolicus. *Nature*, 392(6674):353–8, 1998.

[28] L. Delcher, D. Harmon, S. Kasif, O. White, and S. Salzberg. Improved microbial gene identification with glimmer. *Nucleic Acids Research*, 27(23):4636–4641, 1999.

[29] S. Dong and D. Searls. Gene structure prediction by linguistic methods. *Genomics*, 23:540–551, 1994.

[30] S. Dumais. Using SVMs for text categorization. *IEEE Intelligent Systems*, 13(4), 1998. In: M.A. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt: Trends and Controversies — Support Vector Machines.

[31] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[32] T. C. elegans Sequencing Consortium. Genome sequence of the nematode c. elegans: a platform for investigating biology. *Science*, 282(5396):2012–8, 1998.

[33] J. Fickett. Recognition of protein coding regions in dna sequences. *Nucleic Acid Research*, 10:5503–5518, 1982.

[34] J. Fickett. The gene identification problem: An overview for developers. *Computers Chem.*, 20(1):103–118, 1996.

[35] J. Fickett and C. Tung. Assessment of protein coding measures. *Neucleic Acid Research*, 20(24):6441–50, 1992.

[36] R. Fleischmann and et al. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269(5223):496–512, 1995.

[37] C. Fraser and et al. The minimal gene complement of mycoplasma genitalium. *Science*, 270(5235):397–403, 1995.

[38] C. Fraser and et al. Genomic sequence of a lyme disease spirochaete, borrelia burgdorferi. *Nature*, 390(6660):580–6, 1997.

[39] C. Fraser and et al. Complete genome sequence of treponema pallidum, the syphilis spirochete. *Science*, 281(5375):375–88, 1998.

[40] D. Frishman and P. Argos. Seventy-five percent accuracy in protein secondary structure prediction. *Proteins*, 27:329–335, 1997.

[41] O. Gotoh. Homology-based gene structure prediction: simplified matching algorithm using a translated colon (tron) and improved accuracy by allowing for long gaps. *Bioinformatics*, 16(3):190–202, 2000.

[42] W. Grundy, T. Bailey, C. Elkan, and M. Baker. Meta-meme: Motif-based hidden markov models of protein families. *Comput. Appl. Biosci.*, 13:387–406, 1997.

[43] Y. Guermeur, C. Geourjon, P. Gallinari, and G. Deleage. Improved performance in protein secondary structure prediction by inhomogeneous score combination. *Bioinformatics*, 15:413–421, 1999.

[44] T. Head. Formal language theory and dna: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.

[45] R. Himmelreich and et al. Complete sequence analysis of the genome of the bacterium mycoplasma pneumoniae. *Nucleic Acids Res*, 24(22):4420–49, 1996.

[46] S. Hussini, L. Kari, and S. Konstantinidis. Coding properties of DNA languages. In *DNA Computing, 7th international Workshop on DNA-Based Computers, DNA 2001, Tampa, U.S.A., 10-13 June 2001*, pages 107–118, 2001.

[47] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.

[48] S. Ji. The linguistics of dna: words, sentences, grammar, phonetics, and semantics. *Ann N Y Acad Sci*, 18(870):411–7, May 1999.

[49] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*, pages 137–142. Springer, 1998.

[50] G. F. Jr. The viterbi algorithm. *Proc. of the IEEE*, 61(3):268–78, 1973.

[51] W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.

[52] T. Kaneko and et al. Sequence analysis of the genome of the unicellular cyanobacterium synechocystis sp. strain pcc6803. ii. sequence determination of the entire genome and assignment of potential proteincoding regions. *DNA Res*, 3(3):109–36, 1996.

[53] R. Karchin, K. Karplus, and D. Haussler. Classifying G-protein coupled receptors with support vector machines. *Bioinformatics*, 18:147–159, 2002.

[54] H. Kasai, A. Bairoch, K. Watanabe, K. Isono, S. Harayama, E. Gasteiger, and S. Yamammoto. Construction of the *gyrb* database for the identification and classification of bacteria. In *Genome Informatics 1998*, pages 13–21. Universal Academic Press, 1998.

[55] H. Kasai, T. Ezaki, and S. Harayama. Differentiation of phylogenetically related slowly growing mycrobacteria by their gyrB sequences. *J. Clin. Microbiol.*, 38:301–308, 2000.

[56] Y. Kawarabayasi and et al. Complete sequence and gene organization of the genome of a hyper-thermophilic archaebacterium, pyrococcus horikoshii ot3. *DNA Res.*, 5(2):55–76, 1998.

[57] W. Kent. Blat - the blast-like alignment tool. *Genome Res.*, 12:656–664, 2002.

[58] C. Kim, K. Asai, and A. Konagaya. A generic criterion for gene recognitions in genomic sequences. In *Genome Informatics*, volume 10, pages 13–22, 1999.

[59] H. Klenk and et al. The complete genome sequence of the hyperthermophilic, sulphate-reducing archaeon archaeoglobus fulgidus. *Nature*, 390(6658):364–70, 1997.

[60] A. Krogh, I. Mian, and D. Haussler. A hidden markov model that finds genes in e. coli dna. *Nucleic Acids Res*, 22(22):4768–78, 1994.

[61] F. Kunst and et al. The complete genome sequence of the gram-positive bacterium bacillus subtilis. *Nature*, 390(6657):249–56, 1997.

[62] K. Lari and S. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[63] C. Leslie, E. Eskin, and W. Noble. The spectrum kernel: a string kernel for svm protein classification. In *Proceedings fo the Pacific Symposium on Biocomputing 2002*, pages 564–575, 2002.

[64] S. Leung, C. Mellish, and D. Robertson. Basic gene grammars and dna-chartparser for language processing of escherichia coli promoter dna sequences. *Bioinformatics*, 3:226–36, Mar. 2001.

[65] T. Lowe and S. Eddy. trnascan-se: A program for improved detection of transfer rna genes in genomic sequence. *Nucleic Acids Research*, 25:955–964, 1997.

[66] A. Lukashin and M. Borodovsky. Genemark.hmm: new solutions for gene finding. *Nucleic Acids Research*, 26:1107–1115, 1998.

[67] N. Matic, I. Guyon, J. Denker, and V. Vapnik. Writer adaptation for on-line handwritten character recognition. In *Second International Conference on Pattern Recognition and Document Analysis*, pages 187–191. IEEE Computer Society Press, 1993.

[68] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, A 209:415–46, 1909.

[69] S. Miyazaki, H. Sugawara, T. Gojobori, and Y. Tateno. Dna data bank of japan (ddbj) in xml. *Nucleic Acids Research*, 30(1):13–16, 2003.

[70] S. Mukherjee, P. Tamayo, J. Mesirov, D. Slonim, A. Verri, and T. Poggio. Support vector machine classification of microarray data. Technical report, CBCL, AI Memo 1676, 1999.

[71] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Trans. Neural Networks*, 12(2):181–201, 2001.

[72] R. Nussinov, G. Pieczenk, J. Griggs, and D. Kleitman. Algorithms for loop matchings. *SIAM journal of Applied Mathematics*, 35:68–82, 1978.

[73] C. O'Donovan, M. Martin, A. Gattiker, E. Gasteiger, A. Bairoch, and R. Apweiler. High-quality protein knowledge resource: Swiss-prot and trembl. *Briefings in Bioinformatics*, 3(3):275–284, 2002.

[74] S. Osawa. *Evolution of the Genetic Code*. Oxford University Press, 1995.

[75] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *In Proceedings of CVPR'97*, 1997.

[76] C. Papageorgiou, T. Evgeniou, and T. Poggio. A trainable pedestrian detection system. In *IEEE Conference on Intelligent Vehicles*, pages 241–246, 1998.

[77] P. Pavlidis, T. Furey, M. Liberto, D. Haussler, and W. Grundy. Promoter region-based classification of genes. In *Proc. PSB 2001*, pages 151–163, 2001.

[78] W. Pearson. Rapid and sensitive sequence comparison with fastp and fasta. *Methods in Enzymology*, 183:63–98, 1990.

[79] M. Pontil and A. Verri. Support vector machines for 3-d object recognition. *IEEE Trans. PAMI*, 20:637–646, 1998.

[80] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages 4–16, 1986.

[81] E. Rivas and S. Eddy. A dynamic programming algorithm for rna structure prediction including pseudoknots. *Journal of Molecular Biology*, 283:1168–1171, 1999.

[82] E. Rivas and S. Eddy. Secondary structure alone is generally not statistically significant for the detection of noncoding rnas. *Bioinformatics*, 16:573–585, 2000.

[83] D. Roobaert and M. V. Hulle. View-based 3d object recognition with support vector machines. In *IEEE Neural Networks for Signal Processing Workshop*, 1999.

[84] V. Roth and V. Steinhage. Nonlinear discriminant analysis using kernel functions. In S. Solla, T. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 568–574. MIT Press, 2000.

[85] A. Salamov and V. Solovyev. Protein secondary structure prediction using local alignmments. *Journal of Molecular Biology*, 268:31–36, 1997.

[86] D. Samarsky and M. Fournier. A comprehensive database for the small nucleolar rnas from saccharomyces cerevisiae. *Nucleic Acids Res.*, 27:161–164, 1999.

[87] D. Sankoff, J. Kruskal, S. Mainville, and R. Cedergren. *Fast algorithms to determine RNA secondary structures containing multiple loops*. Addison-Wesley, 1983.

[88] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2002.

[89] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[90] D. Searls. The linguistics of dna. *American Scientist*, 80:579–591, 1992.

[91] D. Searls. String variable grammar: A logic grammar formalism for the biological language of DNA. *Journal of Logic Programming*, 24(1 2):73–102, 1995.

[92] J. Shepherd. Method to determine the reading frame of a protein from the purine/pyrimidine genome sequence statistics, identification, and applications to genome project. *Proc. Natl. Acad. Sci. USA*, 78:1596–1600, 1981.

[93] D. Smith and et al. Complete genome sequence of methanobacterium thermoautotrophicum deltah: functional analysis and comparative genomics. *J. Bacteriol*, 179(22):7135–55, 1997.

[94] S. Sonnenburg, G. Rätsch, A. Jagota, and K. Müller. New methods for splice site recognition. In *Proc. of the International Conference on Artificial Neural Networks*, pages 329–336, 2002.

[95] E. Sonnhammer, S. Eddy, and R. Durbin. Pfam: A comprehensive database of protein domain families based on seed alignments. *Proteins*, 28:405–420, 1997.

[96] R. Staden and A. McLachlan. Codon preference and its use in identifying protein regions in long dna sequences. *Nucleic Acid Research*, 12:505–519, 1984.

[97] R. Stephens and et al. Genome sequence of an obligate intracellular pathogen of humans: Chlamydia trachomatis. *Science*, 282(5389):754–9, 1998.

[98] G. Stormo, T. Schneider, L. Gold, and A. Ehrenfeucht. Use of the 'perceptron' algorithm to distinguish translational initiation sites in e.coli. *Nucleic Acids Research*, 10:2997–3011, 1982.

[99] N. Sueoka. A statistical analysis of deoxyribonucleic acid distribution in density gradient centrifugation. *Proceedings of the National Academy of Sciences*, 45(10):1480–1490, 1959.

[100] H. Tanaka, M. Ishikawa, K. Asai, and A. Konagaya. Hidden markov models and iterative aligners: study of their equivalence and possibilities. In *Proc Int Conf Intell Syst Mol Biol*, volume 1, pages 395–401, 1993.

[101] J. Tomb and et al. The complete genome sequence of the gastric pathogen helicobacter pylori. *Nature*, 388(6642):539–47, 1997.

[102] K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.-R. Müller. A new discriminative kernel from probabilistic models. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, 2002. to appear.

[103] K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18:268S–275S, 2002.

[104] J. Vert. Support vector machine prediction of signal peptide cleavage site using a new class of kernels for strings. In *Proceedings fo the Pacific Symposium on Biocomputing 2002*, pages 649–660, 2002.

[105] M. Waterman. *Introduction to Computational Biology: Maps, sequences and genomes*. Chapman & Hall/CRC, 1995.

[106] T. Yada and M. Hirosawa. Gene recognition in cyanobacterium genomic sequence data using the hidden markov model. *DNA Research*, 3(6):355–61, 1996.

[107] T. Yada, M. Ishikawa, H. Tanaka, and K. Asai. Signal pattern extraction from dna sequences using hidden markov model and genetic algorithm. *IPSJ Trans.*, 37(6):1117–29, 1996.

[108] K. Yeung and W. Ruzzo. Principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9):763–774, 2001.

[109] S. Young and et al. *HTK Book (for HTK version 2.2)*. Entropic Inc., 1999. `ftp://ftp.entropic.com/pub/htk/HTKBook a4.ps.gz`.

[110] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16:799–807, 2000.

[111] M. Zuker and D. Sankoff. Rna secondary structures and their prediction. *Bull. Math. Biol.*, 46:591–621, 1984.

[112] M. Zuker and P. Stiegler. Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9:133–148, 1981.

Publications

[1] T. Kin, A. Konagaya and K. Asai: “A Generic Criterion for Gene Recognition in Genomic Sequences,” Proc. of Genome Informatics Workshop 1999, pp.13-22.

[2] T. Kin, K. Tsuda and K .Asai: “Marginalized Kernels for RNA Sequence Data Analysis,” Proc. of Genome Informatics Workshop 2002, pp.112-122. *GIW2002 Best Paper Award Winner*

[3] K. Tsuda, T. Kin and K .Asai: “Marginalized Kernels for Biological Sequences,” Bioinformatics, 18(Suppl. 1):S268–S275, 2002.

[4] T. Kin, T. Tsuda and K. Asai: “Computation and an application of marginalized kernels for biological sequence data,” CBRC Technical Report, AIST02-J00001-1, Oct. 1. 2002.

Appendix A

Microbial Genomes

Table. A shows a list of complete genomes used for this research.

Table A.1: 17 microbial and 6 eukaryotic genomic sequence data.

Species	Acc. No	Length (nt)
Archaeoglobus fulgidus	AE000782	2,178,400
Aquifex aeolicus	AE000657	1,551,335
Borrelia burgdorferi	AE000783	910,724
Bacillus subtilis	AL009126	4,214,814
Chlamydia trachomatis	AE001273	1,042,519
Escherichia coli	U00096	4,639,221
Haemophilus influenzae	L42023	1,830,138
Helicobacter pylori	AE000511	1,667,867
Mycoplasma genitalium	L43967	580,074
Methanococcus jannaschii	L77117	1,664,970
Mycoplasma pneumoniae	U00089	816,394
Methanobacterium thermoautotrophicum	AE000666	1,751,377
Mycobacterium tuberculosis	AL123456	4,411,529
Pyrococcus horikoshii	Pyro h	1,738,505
Rickettsia prowazekii	AJ235269	1,111,523
Synechocystis PCC6803	AB001339	3,57,3470
Treponema pallidum	AE000520	1,138,011
Caenorhabditis elegans chromosome I	chr I	16,183,833
Caenorhabditis elegans chromosome II	chr II	17,004,925
Caenorhabditis elegans chromosome III	chr III	12,114,540
Caenorhabditis elegans chromosome I	chr IV	15,887,371
Caenorhabditis elegans chromosome V	chr V	21,280,512
Caenorhabditis elegans chromosome X	chr X	17,624,844

Appendix B

Softwares

All computations introduced here are realized as a couple of computer programs. One of them is a standard HMM program with capabilities of computing 1st/2nd order MCKs, and the other is a SCFG program specific to RNA secondary structures. These softwares will be available at `http://www.cbrc.jp/~taishin/`.

B.1 HMM software

The HMM program is based on UMDHMM version 1.0.2: a straightforward implementation of [80] by Kanungo (`http://www.cfar.umd.edu/~kanungo/`). We added our functionalities and extended the program in some extent in order to compute marginalized count kernels. The program is a single binary executable: `khmm` which is capable of:

- Reading FASTA sequence file
- Model evaluation (forward-backward algorithm)
- Expectation-maximization learning (Baum-Welch algorithm)
- Kernel computations: 1st/2nd-order marginalized count kernel and Fisher kernel

Usage `khmm` accepts several command line options:

```
khmm [OPTIONS] -h <HMM file> <sequence file>
```

`<HMM file>` gives HMM parameters which defines a transition matrix and emission matrices for all states.
`<sequence file>` provides one or more sequence data in FASTA file format.

Options

- `-n` skips learning
- `-1 <file>` computes and outputs 1st-order marginalized count kernel matrix to `<file>`.
- `-2 <file>` computes and outputs 2nd-order marginalized count kernel matrix to `<file>`.
- `-f <file>` computes and outputs Fisher kernel matrix to `<file>`.

- `-s DNA|RNA|AMINOACID` specifies symbol set. Omitting this option means that a custom symbol set is given. Hence `-c <charsetfile>` becomes mandatory.
- `-c <charsetfile>` uses `<charsetfile>` as a custom symbol set. `-s` cancels this option.
- `-o <file>` outputs trained HMM parameters to `<file>`.
- `-h <HMM file>` reads HMM parameters from `<HMM file>`.
- `-t <int>` iterates learning up to `<int>` times.
- `-d <double>` iterates learning while it keeps more than `<double>` times improvement.

B.2 SCFG software

The SCFG program `scfg` was built from scratch but we referred to algorithms for generic SCFG and Covariance Model [31]. `sokos` is capable of:

- Reading FASTA sequence file
- Model evaluation (inside-outside algorithm)
- Expectation-maximization learning
- RNA secondary structure prediction (CYK algorithm)
- Kernel computations: 1st/2nd-order marginalized count kernel

Usage

```
sokos [OPTIONS] <sequence file> <scfg file>
```

`<sequence file>` provides one or more sequence data in FASTA file format.
`<scfg file>` defines SCFG parameters i.e. a transition matrix and emission matrices.

Options

- `-c` skips learning.
- `-o <file>` outputs trained SCFG parameters to `<file>`.
- `-p <file>` outputs likelihood of each sequence to `<file>`.
- `-t` predicts secondary structure.
- `-d <float>` repeats learning until it keeps more than `<float>` times improvement.
- `-T <int>` repeat learning up to `<int>` times.
- `-f <file>` compute and output 1st-order feature vectors of each sequence to `<file>`.

- `k <file>` compute and output 1st-order marginalized kernel matrix to `<file>`.
- `F <file>` compute and output 2nd-order feature vectors of each sequence to `<file>`.
- `K <file>` compute and output 2nd-order marginalized kernel matrix to `<file>`.
- `m <int>` use `<int>`threads (default 1).