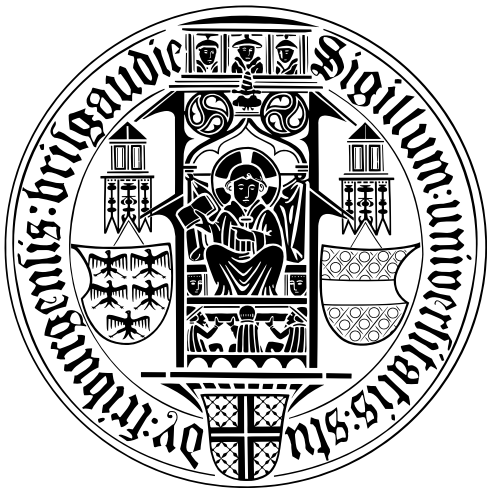
RNA-Protein Interaction Prediction with Graph Kernels

Master Thesis

Li Zhang



Albert-Ludwigs-Universität Freiburg

Technische Fakultät

Institut für Informatik

October 2011

Acknowledgements

First of all, i will express my appreciation to Mr. Dr. Fabrizio Costa, from whom i got my thesis topic. Thanks for giving me so many helpful tutorials during the whole process of my thesis, especially for his patient explanations to my "stupid" questions.

Secondly, i will thank Mr. Daniel Maticzka, who worked as the second tutor of my thesis and gave me many advices, which helps me a lot.

In addition, i will express my gratitude to Mr. Prof. Backofen and all the faculties in the bioinformatics team, thanks for their support on my work.

At last, i will thank my parents, thanks for their continuous support and encouragement behind me.

DECLARATION

I hereby declare, that I am the only author of my thesis and that no other sources or learning aids have been used. Furthermore, I declare that I have acknowledged the work of others by giving detailed references of the corresponding work.

place, date	Signature

Abstract

RNA-binding proteins (RBPs) are a class of proteins which are found in many living organisms, they regulate post-transcriptional gene expression and play important roles. But up to now, the binding preferences for most RBPs are not well characterized. It is believed that the RBPs recognize the RNA targets in a sequence-specific manner, but the structural context of the binding region also influences the binding. So in order to make clear the mechanism of the binding between RBPs and their target RNAs, it is important for us to develop some methods for the prediction of RBPs' binding preferences. Different to the existing methods like MEMERIS or RNAcontext, we present an approach based on the graph kernel method. By using this method, for some RBPs, we could reach higher prediction performances.

Contents

1 Introduction **3**

2 Related Work **6**

2.1 MEMERIS 6

2.1.1 Single-strandedness characterization 7

2.1.2 MEME 8

2.1.3 Integration of secondary structure information 9

2.2 RNAcontext 10

2.2.1 Motif model of RNAcontext 11

2.2.2 Parameter estimation 12

3 Machine Learning **14**

3.1 SVM and Kernel methods 14

3.1.1 Support vector machine 14

3.1.2 Kernel methods 16

3.2 Convolution Kernels 18

3.3 Neighborhood Subgraph Pairwise Distance Kernel 19

3.4 Combination of SVM and NSPDK 22

3.5 Area under the Precision-Recall curve 24

3.6 Cross-Validation 25

4 Application of graph kernels to RNA secondary structure 27

4.1 Goal of proposed method 27

4.2 Experiment process of proposed method 28

4.2.1 Process of experiment based on RNAfold 29

4.2.2 Process of experiment based on RNAshapes 46

4.2.3 Process of experiment based on RNAplfold 48

5 Experiment 58

5.1 Dataset 58

5.2 Experiment result 61

6 Conclusion 69

Chapter 1

Introduction

Nowadays, scientists have already found hundreds of RNA-binding proteins (RBPs) by using new techniques in bioinformatics. This kind of proteins are vital in living organism, since they play key roles in post-transcriptional regulation (PTR) of gene expression by binding to target RNAs to control splicing, export, stability, localization and translation regulation[1]. In order to know the mechanism of RBPs' binding and the detailed function of RBPs in the post-transcriptional regulation, we must make clear the RBPs' binding preferences. But unfortunately, their binding preferences to the target RNA sequences are not well characterized up to now, so the development of methods for the prediction of RBPs' binding preferences becomes an important theme for us. It is believed that RBPs recognize RNA targets based not only on the RNA sequences' information but also on the structural contexts of the binding sites, this characteristic makes the prediction for the RBPs' binding preferences very challenging[1]. For example, the microarray-based *in vitro* method *RNAcompete assay*[2] and some *in vivo* methods (e.g, RIP-seq[3] and CLIP-seq[4]) can be used together to predict the binding preference of an individual RBP for a large number of RNA sequences, but the binding pref-

erence of RBPs for both specific sequence and secondary structural context makes this method very difficult[1]. Although some motif finding algorithms working only at the sequence level can predict the binding preferences very well for some RBPs, the defects (i.e., ignoring the structure) of themselves will produce misleading results when they meet some RBP with non-trivial structural preferences which the binding preferences will be difficult to detect without consideration of the corresponding structural contexts[1][5]. In order to take both the sequence and structural information into consideration during the process of prediction for RBPs' binding preferences, scientists have developed new motif finding methods, such as *MEMERIS*[5] and *RNAcontext*[1]. These approaches all use the probabilistic idea to compute the binding preferences of RBPs.

Different to the two methods above, in this thesis, we introduce a brand new way to predict the binding preferences of RBPs — *Graph Kernel method*.

By using the graph kernel method, we build different graph models containing heterogeneous features, so that the model can incorporate different biological information and represent complex structures such as sequences and graphs, which is useful for us to look for the mechanism of the binding.

In order to compare our approaches' performance with RNAcontext — the state-of-the-art method for the prediction of RBPs' binding preferences, we use the same dataset with RNAcontext — RNAcompete-derived datasets[2], which is comprised of measured binding preferences of nine RBPs (i.e., HuR, Vts1p, PTB, FUSIP1, U1A, SF2/ASF, SLM2, RBM4 and YB1) to a pool of 213,130 unique short (29- to 38-nt) RNA sequences[1]. Then we use different RNA folding softwares (i.e., *RNAfold*[18], *RNAplfold*[18], *RNAshapes*[21]) to generate the secondary structures for the RNA sequences in the dataset, and use different graph models to convert the secondary structures to the

corresponding gSpans[12], where gSpan is the input of the graph kernel NSPDK[11]. After that we use Support Vector Machine (SVM)[7] to learn our models by using cross-validation[15], and fit each graph model with its corresponding optimal parameters, and then evaluate the model's performance by doing prediction on the test set. The performance is measured with the AUC-PR[14]. At last we compare our models' performances with the performance of RNAcontext.

The rest content of this thesis is organized as follows: in Chapter 2, the related work done by others will be introduced, including MEMERIS and RNAcontext. Then Chapter 3 contains the explanations of some concepts and definitions about machine learning, which are fundamental for our method. Chapter 4 talks mainly about the details of the experiment by using our method, including the different graph models we built for our method and the corresponding experiment process by using them. Chapter 5 introduces of the dataset used in our experiment and does some analysis to the experiment result. At last Chapter 6 will be the conclusion for this thesis.

Chapter 2

Related Work

In this chapter, let's first have a look two related methods for the prediction of RBPs' binding preferences, they are MEMERIS and RNAcontext.

2.1 MEMERIS

This section is mainly based on the work in [5]. Since the RBPs recognize target RNAs in a sequence-specific manner, most of the bindings occur in the single-stranded regions, but it is also shown experimentally that the sequestration of a binding motif in a double-stranded RNA region has a strong negative influence to the binding, so in order to include the secondary structure information of RNA into the motif searching process of a protein, some scientists developed a new approach called *MEMERIS* (MEME in RNAs Including Secondary Structures) as an extension of the widely used motif finding program MEME. MEMERIS searches the sequence motifs in a set of RNA sequences and integrates the secondary structure information simultaneously. It is especially useful for searching the motifs locating preferably in a single-stranded region.

In order to integrate the secondary structure during the process of motif search, the way MEMERIS works can be divided into two steps: the characterization of single-strandedness and integration of secondary structure information.

2.1.1 Single-strandedness characterization

First of all, MEMERIS precomputes values that characterize the single-strandedness of all putative motif occurrences, then uses these values to guide the motif search towards single-stranded regions. For a substring in a given RNA sequence between position a and position b, MEMERIS uses two different ways to characterize the single-strandedness corresponding to different situations respectively:
(1)$PU_{a,b}$: the probability that all the bases in the substring are unpaired, defined as

$$PU_{a,b} = e^{\frac{E^{all} - E^{unpaired}_{a,b}}{RT}},$$

where E^{all} is the free energy of the ensemble of all structures, $E^{unpaired}_{a,b}$ is the free energy of the ensemble of all structures which have the complete substring unpaired, R is the gas constant and T is the temperature. E^{all} and $E^{unpaired}_{a,b}$ can be computed with the partition function version of RNAfold.
(2)$EF_{a,b}$: the expected fraction of bases in the substring are not paired, defined as

$$EF_{a,b} = 1 - \frac{\sum_{i=a}^{b}\sum_{j=1}^{L} p_{i,j}}{b-a+1},$$

where L is the length of the RNA sequence, $p_{i,j}$ is the probability that base i and j make a pair, which can also be computed with RNAfold.

For the given input sequences, MEMERIS first computes the PU and

EF values for all the substrings with length W in the sequences, after computation, MEMERIS gets the secondary structure information of the single-stranded region. Then what MEMERIS does is to integrate the information into the motif finding process.

2.1.2 MEME

In MEME, motif finding is done in a set of unaligned nucleotides or protein sequences, denoted as $X = X_1, X_2, ..., X_n$. And a motif is represented as a position-specific probability matrix (PSPM) $\Theta_1 = (P_1, P_2, ..., P_W)$, where W is the length of the motif and the vector P_i is the probability distribution of the letters (i.e., A/U/C/G) at each position i. A given input sequence X_i can be modeled as consisting of two parts, denoted as $\Theta = (\Theta_0, \Theta_1)$, where Θ_1 means zero, one or more non-overlapping motif occurrences sampled from the matrix Θ_1, and Θ_0 represents the random samples from a background probability distribution for the remaining sequence positions.

The number of motif occurrences depends on the model specified by the user. So in MEME, there are three different models to be considered:

(1) OOPS model: exactly one motif occurrence per sequence

(2) ZOOPS model: zero or one motif occurrence per sequence

(3) TCM model: zero or more motif occurrences per sequence

Given the sequences, MEME uses expectation maximization (EM) algorithm to perform a maximum likelihood (ML) estimation to find a motif. Here the set of given sequences are complete data, while the start positions of the motif occurrences are hidden data. The hidden data are described by variables $Z_{i,j}$, when there is a motif occurrence starting at position j in sequence X_i, we say $Z_{i,j} = 1$, otherwise $Z_{i,j} = 0$. The EM algorithm is often used to perform the maximum likelihood estimation. It consists of two

steps: E (Expectation) step and M (Maximization) step. E step computes the expectation of hidden variables, using current latent variables, while M step computes the parameters of the model on the joint probability of the complete data.

2.1.3 Integration of secondary structure information

In MEME, there is no assumption about the start position of a motif occurrence in a given sequence, so it uses uniform probability distribution $\forall j$ $P(Z_{i,j} = 1) = \frac{1}{m}$, where $m = L$ - W + 1 is the number of possible starting positions for a motif with length W in a sequence of length L. But in MEMERIS, in order to integrate the information of the single-strandedness, MEMERIS replaces the uniform probability distribution by a distribution based on EF and PU values. Since PU values are stricter than EF values, so MEMERIS using PU values favors single-strandedness better than MEMERIS using EF values. Here we take MEMERIS using PU values as an example, the prior probability for the OOPS model in MEMERIS are

$$P(Z_{i,j} = 1 \mid PU_i) = \frac{PU_{i,j} + \pi}{\sum_{k=1}^{m} PU_{i,k} + \pi},$$

where PU_i is the vector of PU values for sequence X_i, j represents the position in the sequence. π is a pseudocount for smoothing the distribution. From the formula we could see that, the higher the PU value for position j, the higher is the prior probability of being a motif start position.

Then, same with MEME, MEMERIS uses EM algorithm to find the motif occurrence iteratively. But the E step and M step have some corresponding change. For simplicity, we don't talk the E step and M step of MEMERIS in detail here.

Above all, MEMERIS searches a motif and integrates information of the secondary structure simultaneously by the pre-computation of a single value for each substring to measure its single-strandedness. The measurement is based on the idea of base pair probabilities. In order to take multiple possible structural contexts into consideration simultaneously, and compute the corresponding preferences for an RBP to each structural context respectively, rather than only a pre-defined structural preference to be queried[1], RNAcontext was raised.

2.2 RNAcontext

This section is mainly based on the work in [1]. *RNAcontext* is an approach for the prediction of RBPs' binding preferences to the RNA sequences. It assumes that the role of RNA secondary structure in the process of binding is to establish a structural context for the RNA sequence recognized by the RBP. Since a given RNA sequence can have multiple different secondary structures, RNAcontext does not focus on the single minimum free energy structure, but on the ensemble of secondary structures that an RNA sequence can form. And for each nucleotide in the sequence, RNAcontext annotates with its structural context (e.g., paired, in a hairpin loop or multiloop). That means RNAcontext uses SFOLD[6] to sampling a large number of structures (1000 structures) for each RNA sequence first, then annotates each base in the sequence with context annotation alphabet , then RNAcontext sets the structural context distribution (*annotation profile*) to be the empirical annotation frequencies for that base across these samples.

2.2.1 Motif model of RNAcontext

Just as mentioned above, RNAcontext takes a set of sequences with their corresponding annotation profiles predicted by SFOLD as its input. Formally, the input sequences can be represented as $S = \{s^1, s^2, ..., s^N\}$, and the annotation profiles for each sequence can be represented as $P = \{p^1, p^2, ..., p^N\}$.

RNAcontext uses A to represent the *annotation alphabet* consisting of structure features, $A = \{P, L, M, U\}$. Here, P means that the nucleotide is paired, L means the nucleotide is in a hairpin loop, U means the nucleotide is in an unstructured region, and M stands for miscellaneous states for a nucleotide since it combines the remaining unpaired contexts including the nucleotide is in a bulge, internal loop or multiloop. And the model parameters are represented as $\Theta = \{\Phi, \Gamma, \beta_s, \beta_p, K\}$, where K is the binding site's width, Φ is a position weight matrix of sequence features with dimensions $4 \times K$, and Γ is a vector of structure annotation parameters represented as $\Gamma = (\Gamma_P, \Gamma_L, \Gamma_M, \Gamma_U)$, with one parameter for each structure annotation in A. And β_s, β_p are the bias terms in the sequence affinity model and structural context model. For a sequence s and its annotation profile p, RNAcontext assigns a score $f(s, p, \Theta)$ to represent the probability that at least one of its subsequences of length K (we call K-mers) is bounded by the RBP, defined as

$$\begin{aligned} f(s,p,\Theta) &= 1 - \prod_{t=0}^{|s|-K} 1 - N(s_{t+1:t+K}, p_{t+1:t+K}, \Theta) \\ &= 1 - \prod_{t=0}^{|s|-K} 1 - N^{seq}(s_{t+1:t+K}, \Theta) \times C(p_{t+1:t+K}, \Theta) \\ &= 1 - \prod_{t=0}^{|s|-K} 1 - \sigma(\beta_s + \sum_{k=1}^{K} \Phi_{s_k,k}) \times \sigma(\beta_p + \sum_{a \in A} \Gamma_a \times \sum_{k=1}^{K} p_{a,k}), \end{aligned}$$

where $N(s_{t+1:t+K}, p_{t+1:t+K}, \Theta)$ is an estimate of the probability that the K-

mer with base content $s_{t+1:t+K}$ and structural context $p_{t+1:t+K}$ is bound. $s_{t+1:t+K}$ represents the subsequence of s between base $t+1$ and $t+K$, $p_{t+1:t+K}$ is a probability profile matrix with its rows representing the annotation alphabet and its columns representing the annotation distributions for each base between base $t+1$ and $t+K$. The term $N^{seq}(s, \Theta)$ is an estimate of the probability that the RBP will bind $s_{t+1:t+K}$ in the ideal structural context, and it is defined with a standard biophysical model $N^{seq}(s, \Theta) = \sigma(\beta_s + \sum_{k=1}^{K} \Phi_{s_k,k})$, where $\sigma(x) = (1 + exp(-x))^{-1}$ is a logistic function. This function saturates to 0 for negative x and 1 for positive x quickly. The structural context term $C(p, \Theta)$ is modeled in the term of $C(p, \Theta) = \sigma(\beta_p + \sum_{a \in A} \Gamma_a \times \sum_{k=1}^{K} p_{a,k})$, where $p_{a,k}$ means the probability that the base at position k of s has structural annotation a.

2.2.2 Parameter estimation

The motif model of RNAcontext has flexibility to represent many different binding modes by tuning the the parameters of the model. For example, when the subsequence $s_{t+1:t+K}$ is in a preferred structural context represented as annotation alphabet a, the corresponding value of Γ_a is positive and large, so that the term $C(p, \Theta)$ equals about 1, and the score $N(s, p, \Theta)$ for the K-mer s equals approximately about $N^{seq}(s, \Theta)$. So the base content s determines the score value. If the subsequence is in a disfavored structural context, the score of Γ_a is a highly negative value, so that the term $C(p, \Theta)$ equals about zero. Since the term $N(s, p, \Theta)$ is modeled as the product of $N^{seq}(s, \Theta)$ and $C(p, \Theta)$, the score of $N(s, p, \Theta)$ will also be equal to about zero regardless of s. In this way, RNAcontext's motif model could adapt itself to different binding situations by regulating the parameters.

Now let's talk about the estimate of parameters for the RNAcontext

model. After building the motif model, RNAcontext estimates parameters from binding data by assigning different values to parameters according to different binding situations. For example, for an RBP binding its favorite sequence with a specific structural context, e.g. unstructured (U), RNAcontext represents this binding mode by setting the sequence parameter Φ to match its sequence binding preference, and sets the elements in the structural parameter Γ to be negative except for the element Γ_U that corresponds to the preferred structural context. For the RBP which has multiple preferred contexts (e.g., hairpin loops or unstructured), RNAcontext sets large positive value to the corresponding structure parameter Γ_H and Γ_U to represent it. For the RBPs without obvious sequence preferences, RNAcontext sets the elements of Φ to be constant values, and sets Γ_P to be large positive value. For RBPs without obvious structure preferences, RNAcontext sets Γ to zero and sets β_p to a large positive value. RNAcontext models the RBPs' binding preferences by associating each RBP with a single preferred structured context which is required for binding.

RNAcontext learns the model parameters Θ by reproducing the binding affinities $R = \{r^1, r^2, ..., r^N\}$ for each of the given sequences $S = \{s^1, s^2, ..., s^N\}$ with the model. RNAcontext models the affinity r^i of a sequence s^i as a linear function of the sequence score $f(s^i, \Theta)$ with unknown slope α and y-intercept b. It searches the combinations of Θ, α and b that minimize the sum of the squared differences between the measured affinity r^i and the reproduced affinity $\hat{\mathrm{r}}^i = \alpha \times f(s^i, \Theta) + b$. RNAcontext evaluates the models by using two-fold cross-validation on RNAcompete-derived datasets, since the construction of datasets provides the advantage to use two-fold cross-validation (see details in Chapter 5).

Chapter 3

Machine Learning

In this chapter, we will introduce some basic concepts about machine learning, which is helpful to understand our method.

3.1 SVM and Kernel methods

3.1.1 Support vector machine

Definition 3.1.1. [7] (**Support Vector Machine**) Support Vector Machines (SVM) are a group of supervised learning methods that can be applied to classification or regression problems. It combines the ideas of statistical learning theory, optimization theory, and is often used together with kernel methods in classification problem.

In a classification problem, there are usually two kinds of set: training set and test set. In the training set, each training instance is represented as one "target value" (the class label) plus its corresponding "attributes" (the features). The goal of classification by using SVM is to produce a model by training on the training set, then use this model to predict the target values

of the data in the test set by using the given attributes of test set[8]. The data used in SVM are in *sparse representation*, which means the attributes of a data are all contained in a vector called *sparse vector*, so that each data can be represented as a vector of real numbers, with each item of the vector representing an category of attribute for the data[8]. In our method, the conversion work from attributes to numeric vector is done by a graph kernel called NSPDK (Neighborhood Subgraph Pairwise Distance Kernel)[11].

The SVM classifies the training data with hyperplanes, but among all the hyperplanes, there exists an optimal one which makes the distance between the data and the separating hyperplane itself largest. What SVM does in the classification is trying to make a balance between searching the optimal hyperplane that maximizes the margin and minimizing the training error. As shown in Figure 3.1, the vectors located on the dashed lines are called *support vectors*, the two dashed lines are *supporting hyperplane*, the *separating hyperplane* locates in the middle of the two supporting hyperplanes. In mathematics, the maximization of the margin between two supporting hyperplanes is a *dual optimization problem*[9], and we don't talk it here in detail. But in reality, the situations that all the data with class label +1 locating on one side of the optimal hyperplane and all the data with class label -1 locating on the other side are very rare. There are usually some data locating in the wrong side, illustrated as in Figure 3.1. So there should be some penalties for the training error, SVM uses parameter C to deal with this problem for the linear classification. Penalty parameter C means the tradeoff between margin maximization and training error minimization[9]. In our method, we also use parameter C for the linear classification.

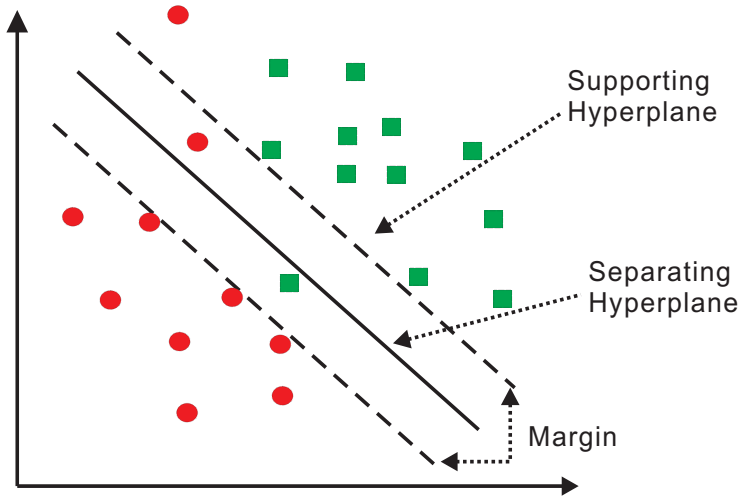


Figure 3.1: example figure for SVM: separate balls from diamonds

3.1.2 Kernel methods

In reality, we often meet some problems that the original input data could not be separated linearly in the input space. So we need kernel methods[10] to help us. Kernel methods use a function to map the original input data from input space to a vector space called *feature space*. Then SVM still uses the same way talked above to classify the mapped data in the feature space with linear classifier.

The way kernel methods work can be divided into two procedures[10]:

(i): we use kernel function to map the original data items from input space to feature space.

(ii): In the feature space, we use learning algorithms (e.g., SVM) to seek the linear function to classify the mapped data items.

The mapping function is defined as *kernel function*, represented as $\Phi : D \to F, K(d_i, d_j) = \langle \phi(d_i), \phi(d_j) \rangle$. The kernel function maps the data into feature space, then computes the inner products between mapped data in the

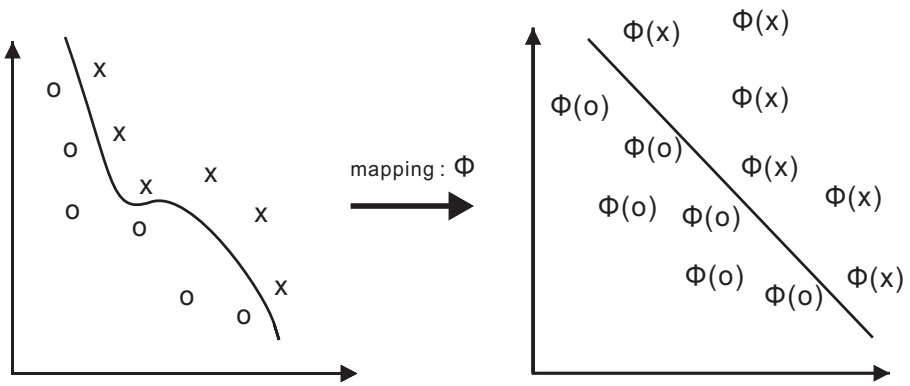


Figure 3.2: the kernel function maps the original data into a feature space, and the nonlinear pattern becomes linear in the feature space.

feature space. However, the computation of inner product can be replaced by the direct computation of function in the original input space, so that we need not even know the concrete mapping function. This substitution is also called as *kernel trick*[10].

Definition 3.1.2. [10] (**Kernel Function**) A kernel is a function k that for all x, $z \in X$ satisfies

$$k(x, z) = \langle \phi(x), \phi(z) \rangle,$$

where ϕ is a mapping from X to an (inner product) feature space F

$$\phi : x \to \phi(x) \in F.$$

The advantage of kernel function is that we can compute the inner product between two mapped data by calculating the kernel function in the original input space, we need not even know the explicit mapping function. After having the kernel function, we use it to map the original data into a feature space, then we get the corresponding kernel matrix, and we process the matrix with a pattern analysis algorithm (e.g., SVM) to generate a pattern

function, the pattern function will be used for the prediction of unexpected new examples[10]. Figure 3.3 shows the steps of applying kernel function.

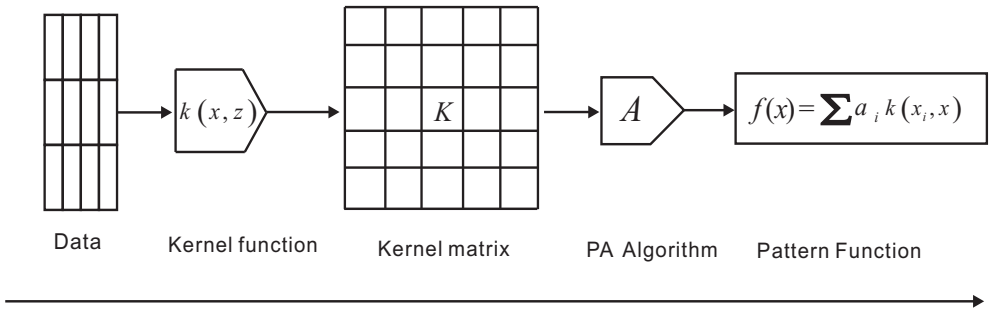


Figure 3.3: stages of applying kernel method

3.2 Convolution Kernels

Graph is the most general data structure. It is used in many aspects since it could model the relationships between objects. The most general form of a graph consists of a vertex set V and an edge set E. The kernels related with graphs are called *graph kernels*, which are a kind of *convolution kernels*.

Definition 3.2.1. [11] (**Convolution Kernels**) Let $x \in X$ be an composite object which could be divided into many parts represented as $x = (x_1, x_2, ..., x_d)$ with each $x_i \in X_i$ for $d = 1, ..., D$ with $D \geq 1$. Let R be the relation defined on the set $X_1 \times X_2 \times ... \times X_D \times X$ such that $R(x_1, ..., x_D, x)$ is true iff $x_1, ..., x_D$ are the parts of x. We define the inverse relation $R^{-1}(x)$ yielding the parts of x, which means $R^{-1}(x) = x_1, ..., x_D : R(x_1, ..., x_D, x)$. $R^{-1}(x)$ represents the set of all valid decompositions of an object, and it is countable. If there exists a kernel K_d over $X_d \times X_d$ for each $d = 1, ..., D$, and if two objects $x, y \in X$ can be decomposed into $x_1, ...x_d$ and $y_1, ..., y_d$, then

the convolution kernel is defined as

$$K(x,y) = \sum_{\substack{x_1,\ldots,x_d \in R^{-1}(x) \\ y_1,\ldots,y_d \in R^{-1}(y)}} \prod_{d=1}^{D} K_d(x_d, y_d).$$

So in a word, the convolution kernel is a sum of the product of valid kernels over the parts of object.

Example 3.2.1. [10] In a directed graph $G = (V, E)$, there is a source vertex s with the in-degree 0 and a sink vertex t with the out-degree 0. P_{st} is the set of directed paths from vertex s to t, and each edge of the graph is labeled with a kernel K_e, so for a path $p = (u_1, u_2, ..., u_d)$ we have the product of the kernels related with the edges of path p

$$K_p(x,z) = \prod_{i=1}^{d} K_{(u_{i-1} \to u_i)}(x,z),$$

then the graph kernel associated with the graph G could be defined as

$$K_G(x,z) = \sum_{p \in P_{st}} K_p(x,z) = \sum_{p \in P_{st}} \prod_{i=1}^{d} K_{(u_{i-1} \to u_i)}(x,z).$$

In next section, we will introduce a graph kernel called Neighborhood Subgraph Pairwise Distance Kernel (NSPDK), which is a convolution kernel used in our method.

3.3 Neighborhood Subgraph Pairwise Distance Kernel

In order to know the principle of NSPDK, we must make clear the concepts below. The content of this section refers mainly to the work in [11].

Definition 3.3.1. (**Distance**) The distance between two vertices is the length of the shortest path between those two vertices, denoted as $D(u, v)$.

Definition 3.3.2. (**Neighborhood**) The neighborhood of radius r of a vertex v is the set of vertices at a distance less than or equal to r from v, denoted as $N_r(v)$.

Definition 3.3.3. (**Induced-subgraph**) The induced-subgraph in a graph G is a graph on a set of vertices $W = w_1, ..., w_k$ that has W as its vertex set and contains every edge of G whose two endpoints are both in W.

Definition 3.3.4. (**Neighborhood Subgraph**) The neighborhood subgraph of radius r of a vertex v is the subgraph induced by the neighborhood of radius r of v, denoted as N_r^v.

Definition 3.3.5. (**Isomorphism**) Isomorphism is a structure-preserving bijection. Two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, denoted as $G_1 \simeq G_2$, if there is a bijection: $\phi : V_1 \to V_2$, such that for any two vertices $u, v \in V_1$, there is an edge uv if and only if there is an edge $\phi(u)\phi(v)$ in G_2. For two labeled graphs, if they are isomorphic, then except satisfying the condition above, there should be an isomorphism that preserves also the label information, i.e., $L(\phi(v)) = L(v)$.

Definition 3.3.6. (**Graph Invariant**) A graph invariant (isomorphism invariant) is a graph property that is identical for two isomorphic graphs (e.g., the number of vertices and/or edges in the two graphs).

For two rooted graph A^v and B^u and a graph G, we define a relation $R_{r,d}(A^v, B^u, G)$ among them. If both A^v and B^u are in $N_r^{v'}$, where $v' \in V(G)$ and $D(u, v) = d$, then we say the relation $R_{r,d}$ is true. So in a word, the relation $R_{r,d}$ selects in a graph G all pairs of neighborhood graphs of radius r whose roots are at a distance d (see Figure 3.4).

Definition 3.3.7. (**NSPDK**) We define $k_{r,d}$ over $G \times G$ as a decomposition kernel on the relation $R_{r,d}$:

$$k_{r,d}(G, G') = \sum_{\substack{A_v, B_u \in R_{r,d}^{-1}(G) \\ A'_{v'}, B'_{u'} \in R_{r,d}^{-1}(G')}} \delta(A_v, A'_{v'})\delta(B_u, B'_{u'}),$$

where $\delta(x, y)$ is the *exact matching kernel*. $\delta(x, y)$ equals 1 if $x \simeq y$, otherwise 0. So from the perspective of function, $k_{r,d}$ counts the number of identical pairs of neighboring graphs of radius r whose roots are at a distance d between two graphs. And the Neighborhood Subgraph Pairwise Distance Kernel is defined as:

$$K(G, G') = \sum_r \sum_d k_{r,d}(G, G').$$

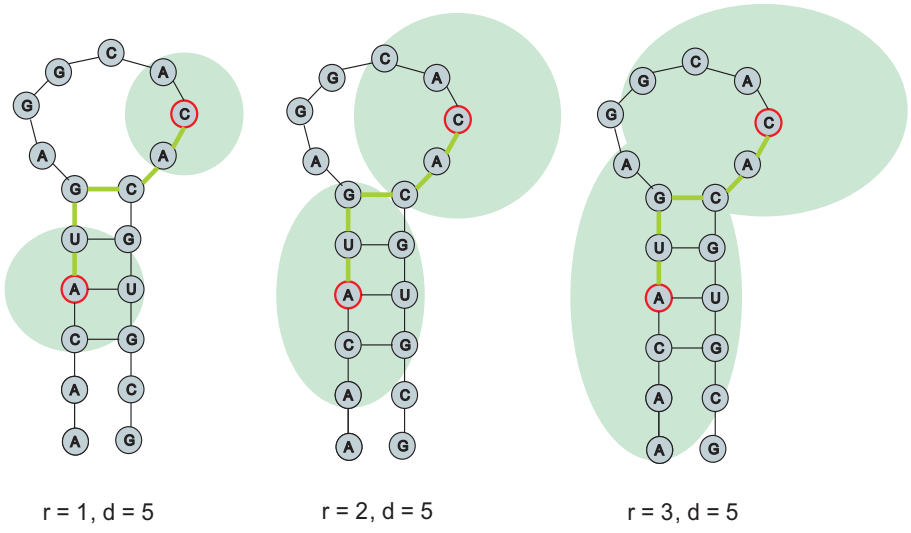


Figure 3.4: pairs of neighborhood graphs for radius $r = 1, 2, 3$ and distance $= 5$ (the neighborhood graphs can overlap)

Since the computation of the exact matching kernel $\delta(G_h, G'_{h'})$ is equivalent to solving the graph isomorphism problem (ISO), for which the polynomial algorithms is still an open problem, NSPDK uses an approximate

way to convert the isomorphism test between two graphs to a fast numerical identity test. That means first NSPDK uses a label function $L^g : G_h \longrightarrow \Sigma^*$ to produce a string encoding for the graph invariant in G_h and $G'_{h'}$ including vertices and edges, so that we could convert the set of rooted graphs G_h to the set of strings Σ^*. Then NSPDK uses a hash function $H : \Sigma^* \longrightarrow N$ to map the strings to natural numbers.

Concretely speaking, in the process of graph encoding, the label function for vertices and edges are denoted as L^n and L^e respectively, where $L^n(v)$ assigns to vertex v the concatenation of the lexicographically sorted list of distance-label pairs $\langle D(v,u), L(u)\rangle$ for all $u \in G_h$ including the root node; $L^e(uv)$ assigns to edge uv the label $\langle L^n(u), L^e(uv), L^n(v)\rangle$. So, above all, the graph encoding $L^g(G_h)$ assigns to the rooted graph G_h the concatenation of the lexicographically sorted list of $L^e(uv)$ for all edges $uv \in E(G_h)$. After the graph encoding, NSPDK uses a Merkle-Damgård construction based hash function to map the graph encoding string to a 32-bit integer at last.

3.4 Combination of SVM and NSPDK

NSPDK runs on the input files in the gSpan format[12] represented as follows:
"$t\#N$" means the Nth graph;
"v M L" means the Mth vertex in this graph has label L;
"e P Q L" means the edge connecting the Pth and Qth vertex has label L.
A graph consists of vertices and edges, so a labeled graph could be represented with a corresponding gSpan (shown in Figure 3.5).

The output of NSPDK is the sparse vector file taken as the input of SVM, since SVM accepts the input file in sparse representation. In our experiment, we use the software *SVMlight* (Joachims 1999).

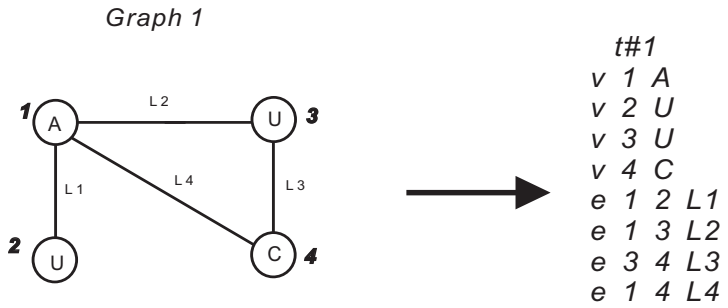


Figure 3.5: convert a labeled graph to the corresponding gSpan

The SVMlight works in two modes[13]: learning (training) and classification (prediction). In the learning mode, it takes a lot of training examples to train the model, and generates a model file for the classification use. In the NSPDK generated sparse vector file, each instance is represented as a numeric vector as mentioned before. We plus the target value in front of each instance so that the file can be processed by SVMlight. The detailed format of the sparse vector file after adding the target values is as follows[13]:

<line >.=. <target><feature>:<value><feature>:<value>

...<feature>:<value># <info >

<target >.=. +1 | -1 | 0 | <float >

<feature >.=. <integer >| "qid"

<value >.=. <float >

<info >.=. <string >

where "target" means the class label (target value) of the corresponding training instance. The "feature" means the position number of the sparse vector represented with integer values, while the "value" represents the element value of the sparse vector denoted with float values. In the learning mode, SVMlight first generates a model file by taking the input sparse vector

file, then it uses this model file to generate a prediction file consisting of the predicted class labels in the classification mode. After the two steps above, we will use the software *AUCCalculator* to calculate the AUC-PR by using the prediction file, since we use AUC-PR to measure the performance of our method.

3.5 Area under the Precision-Recall curve

In machine learning, the PR (Precision-Recall) curves are often used to measure the performance of a learning algorithm on a given dataset, which contains fixed number of positive and negative examples[14]. Since in a binary decision problem, a classifier labels the examples as either positive or negative, the performance of it can be represented as a *Confusion Matrix*, in which there are four categories[14]: True positives (TP), False positives (FP), True negatives (TN), and False negatives (FN), where TP means the positive examples that are correctly labeled as positive, FP means the negative examples that are incorrectly labeled as positive, and FN and TN represent the positive examples incorrectly labeled as negative and negative examples correctly labeled as negative respectively.

	actual positive	actual negative
predicted positive	TP	FP
predicted negative	FN	TN

Table 3.1: confusion matrix

In our method, since we combine graph kernel and SVM, the classifier's performance is related with the graph model we use. We prefer to use AUC-PR to represent the predictive performances of our models, because we will

compare our method with RNAcontext, which represents its performance with the AUC-PR metric. AUC-PR means the area under the curve of Precision-Recall, it is a simple metric to define how an algorithm performs over the whole space[14]. In PR space, the y-axis represents the precision, the x-axis represents the recall. The related definitions are as follows:

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

3.6 Cross-Validation

Definition 3.6.1. [15] (**Cross-Validation**) Cross-validation is a statistical method often used for the performance evaluation of learning algorithms, when one wants know the performance of a predictive model on an unknown dataset. The dataset is often divided into several equally sized subsets, then we train the learning model on some subsets of them, called as *training sets*, and we test the model after training on the remaining subsets, we call *test sets*. According to the number of subsets partitioned, there are several kinds of cross-validation, e.g., k-fold cross-validation, leave-one-out cross-validation, etc.

Generally speaking, for k-fold cross-validation, we often use $k-1$ folds to do the training, and leave 1 fold for the testing. In the first round, we train the model on the $k-1$ folds, then test the fit model on the remaining 1 fold. This process is repeated k times, with each of the k subsets used exactly once as the validation data. At last, the testing (validation) results are averaged over the k rounds. And this average value is taken as the final

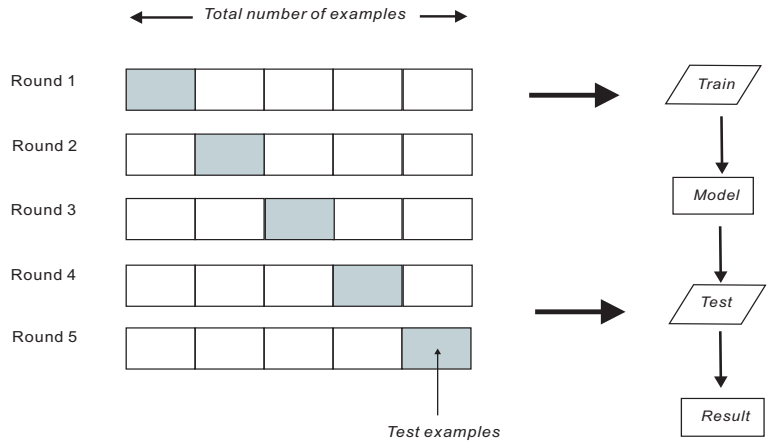


Figure 3.6: five-fold cross-validation

result of k-fold cross-validation. Figure 3.6 illustrates the process of five-fold cross-validation.

In our experiment, for the training of our models, we prefer to use five-fold cross-validation, so that we can make sure that there are enough training instances to learn the model. But for the final performance evaluation of our models, we use two-fold cross-validation, since the construction of the dataset we used in our experiment provides us the advantage to use this strategy. We will introduce this in detail in following chapters.

Chapter 4

Application of graph kernels to RNA secondary structure

In this chapter, we will introduce how we apply the graph kernel method to the prediction of RBPs' binding preferences, including the different graph models we built and the processes of our experiment based on different folding softwares and graph models.

4.1 Goal of proposed method

Before the introduction of our method, we should first make clear the goal by using it. Generally speaking, there are three goals we want to achieve through our experiment:

(1) train a graph model that has better performances (i.e., higher AUC-PR) for the corresponding RBPs than RNAcontext.

(2) get to know whether the structural context has positive influence to the binding through the performance comparison between models with structural context and models without structural context.

(3) except the structural context, wonder whether there are some other factors that influence the binding.

4.2 Experiment process of proposed method

The process of the experiment by using our method consists of many steps. Briefly speaking, we start from the original dataset (i.e., RNAcompete-derived datasets, see details in Chapter 5), after filtering, splitting and converting operation on the original data, we get sequence files in FASTA format[16], then we use different RNA folding softwares to predict the secondary structures for each RNA sequence in the FASTA files, and we will get the output files containing the secondary structures for the FASTA sequences in dot-bracket format[17], then we use our java programs to convert the dot-bracket files to the corresponding gSpan files, so that the graph kernel NSPDK can run on it and output files in sparse representation, then we combine the original split label files and the sparse vector files together to take as the input of SVM-light, and use SVMlight to do cross-validation on the training set to train our graph models, then we choose the optimal combination of model parameters with highest average AUC-PR for each model, and test the fit models' performances by doing prediction on the corresponding test set. We use AUCCalculator to compute AUC-PR to measure the models' performances. The AUC-PR values for RBPs computed with the fit models are the final result of our experiment, which represent the predictive performances for the corresponding RBPs' binding preferences by using our method.

In our method, we use three different RNA folding softwares: *RNAfold*[18], *RNAplfold*[18] and *RNAshapes*[21]. By using each of them, we generated several graph models, so altogether we built 11 different graph models in our

whole experiment.

4.2.1 Process of experiment based on RNAfold

Now, let's start from the models based on RNAfold first. For RNAfold, we built three models, in which one is based on the MFE secondary structure (without parameter when we use RNAfold), the second one is based on the centroid structure (with parameter *-p* in the command), and the last one is a modified model derived from the MFE structure model.

Definition 4.2.1. [19] (**Centroid Structure**) The centroid for a given set of structures is the structure that has the minimum total base-pair distance to the entire structures in the set. It could be considered as a single structure in the ensemble that best represents the central tendency of the set.

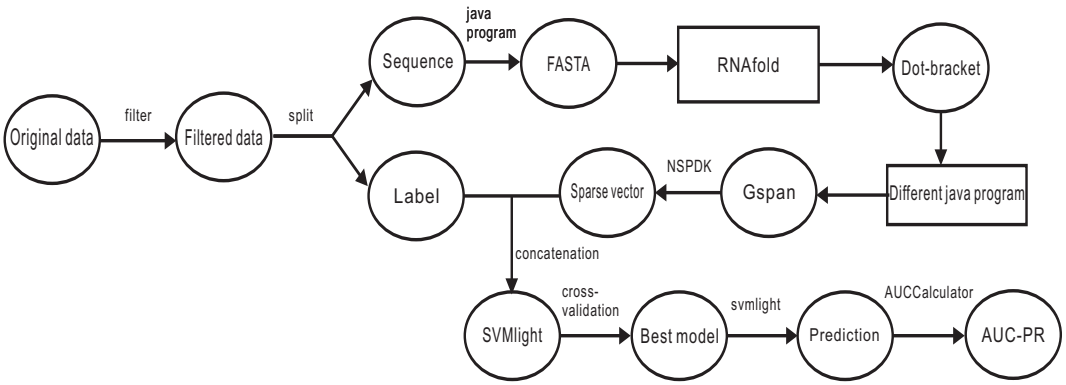


Figure 4.1: process of experiment by using RNAfold

Figure 4.1 shows the process of experiment by using RNAfold, and the description of each step is as follows:

step (1): Filter the original dataset

In this step, we start from the original RNAcompete-derived datasets, convert the dataset to a "filtered" dataset with our java program. Since

we will use five-fold cross-validation to train our model and search the best combination of model parameters on SVMlight later, we must assign class labels to each training and testing instance (i.e., each RNA sequence in the dataset). We prefer to use a similar way as in RNAcontext to filter the original dataset and assign the class label for each RNA sequence.

In RNAcontext, it uses different cutoff for each of the nine RBPs, i.e., *FUSIP1*: 2, *RBM4*: 3, *vts1p*: 3.5, *YB1*: 1.3, *U1A*: 0.5, *SLM2*: 2.5, *SF2*: 3, *HuR*: 4, *PTB*: 2. By using these cutoff values, it filters the original dataset, so that if a sequence in the dataset with estimated binding affinity larger than the corresponding cutoff, it will be put into the "positive" group. On the other side, if a sequence with estimated binding affinity below the median value of the training set where it is in, it will be put into the "negative" group. Here the training set means full or weak set for each of the nine RBPs in the RNAcompete-datasets. In our experiment, we use a similar way as follows:

(i) First we get the cutoff values for the nine RBPs from RNAcompete-derived datasets, and let the sequences with their estimated binding affinities above the corresponding cutoff to be "positive" sequences in the training set.

(ii) Then for each of the nine RBPs, we calculate the median values for the relative set A and set B of the full set and weak set respectively, while in RNAcontext they only calculate median values for full and weak set of each RBP. We let the sequences with their estimated binding affinities below the corresponding median value to be "negative" sequences. For example, for RBP FUSIP1, we compute the median values for set *FUSIP1_weak_A*, *FUSIP1_weak_B*, *FUSIP1_full_A*, *FUSIP1_full_B* respectively. So, for each of the nine RBPs, we have four median values computed from the relative *weak_A*, *weak_B*, *full_A*, *full_B* set. Since in the original RNAcompete-derived datasets, the sequence file is comprised of the RNA sequences with their esti-

mated binding affinities, and the affinities are in random order, so according to the definition of median, we first sort the affinities in ascending order, then calculate the median: If the number of sequences in the file is odd, then the median is the middle value $((n+1)/2)$ among all the sorted affinity values. If the number of sequences is even, then the median is the average between the $n/2$ and the $n/2+1$ values.

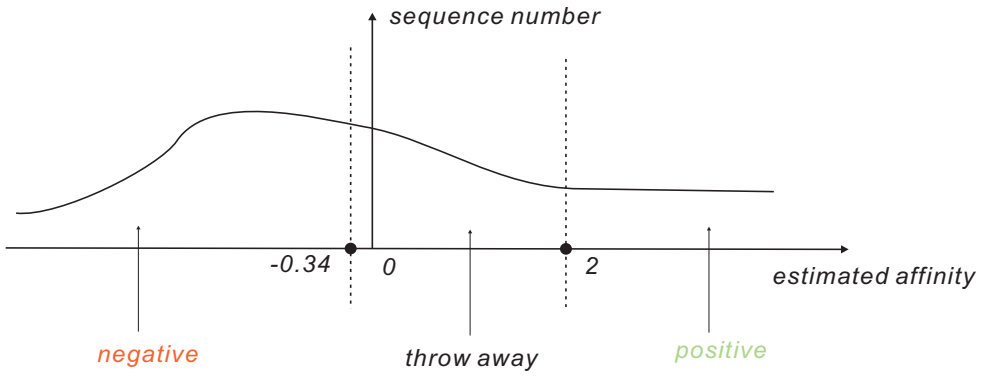


Figure 4.2: the cutoff and median value for filtering a dataset: the cutoff equals 2, the median equals -0.34

(*iii*) After we get the cutoffs and medians for dividing the positive and negative sequences, we start to filter the original data. If one sequence's estimated binding affinity is larger than the cutoff, then we replace its binding affinity with label $+1$; when the estimated binding affinity of a sequence is lower than the median value of the corresponding set, we replace the affinity with label -1. The sequences with estimated binding affinities between the cutoff and median will be thrown away from the dataset (the reason we call "filtered"). Figure 4.2 is an illustration for the filtering by using cutoff and median on a dataset. Figure 4.3 shows the difference between the original data and the filtered data after filtering.

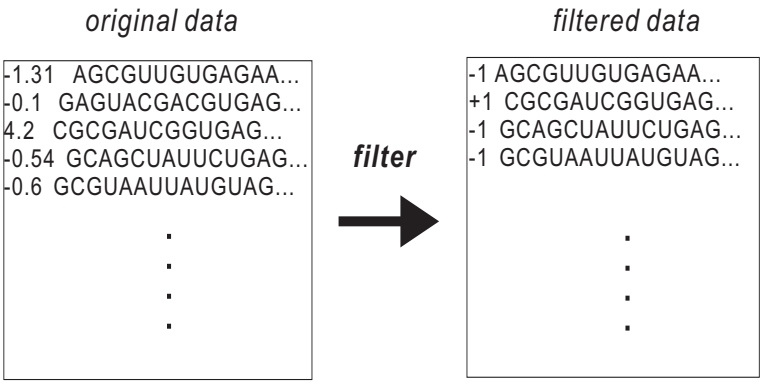


Figure 4.3: the difference between original data and filtered data

step (2) Convert the filtered data to FASTA format

Definition 4.2.2. [16] (**FASTA Format**) FASTA format is often used to represent nucleotide sequences in bioinformatics. A sequence in FASTA format begins with a symbol ">", and the subsequent content following the symbol will be taken as description for the sequence. The sequence information starts in another line, and all lines should be shorter than 80 characters.

Example 4.2.1. FASTA sequence

>Description of the sequence

UGAUGCGUGAUGUUAUGCGUAGGUCGAUUGCGUAUGUGG

CGUAGUCGAUCGUAUGCUGAGUGCUGUUGCGAUGCGAUG

AUCGGAUGUCGUGAGUUCUAUGUCUGAUGGUG

Now let's convert the sequences in the filtered dataset to the FASTA format. Take a data file as instance, first we split the file into two parts: the label part and the sequence part. Then we convert the sequence part to the FASTA format by adding a sequence description above each sequence in the file, since the RNA folding softwares used in our experiment all accept input files in FASTA format.

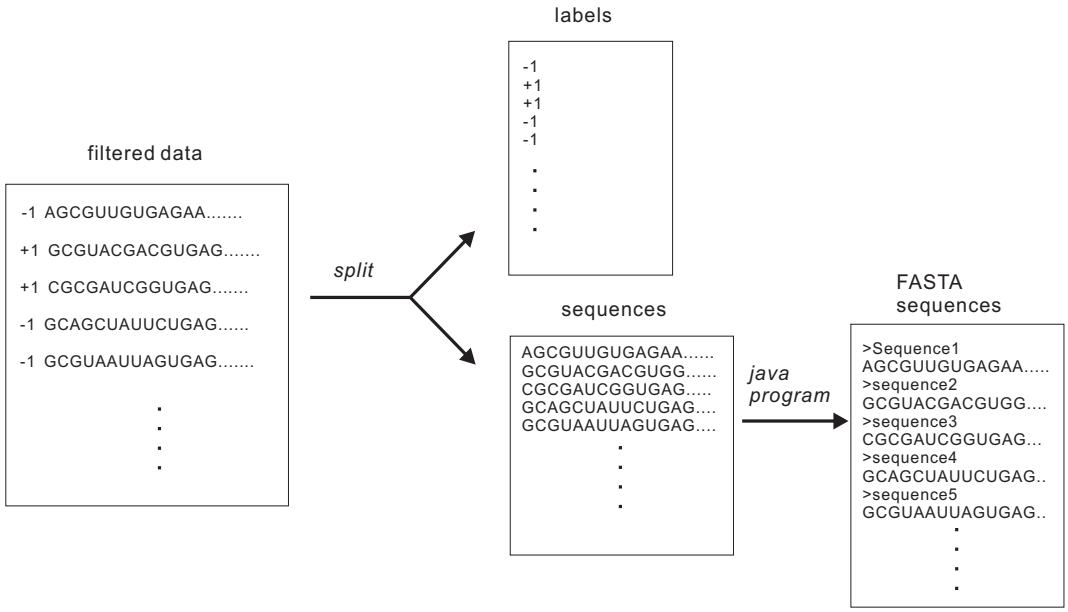


Figure 4.4: process of splitting and converting

After the filtering, splitting and converting step, we have the RNA sequences in FASTA format (see Figure 4.4). Then from this time on, we start to use different folding softwares to process the data and use different graph models to work on it.

step (3): Convert FASTA to dot-bracket notation

By using different parameters of RNAfold, we could get the secondary structures computed with different folding algorithms for the corresponding RNA sequences in the FASTA file, such as the MFE structures (without parameter) or the centroid structures (with parameter *-p*). For the MFE structure, the generated output file includes the sequences in FASTA format, the corresponding secondary structures of the sequences in dot-bracket notation, and also the corresponding minimum free energy for each sequence. We throw the MFE away, and leave the sequence and its structure information for our experiment use. Figure 4.5 illustrates the conversion process

from FASTA format to dot-bracket notation.

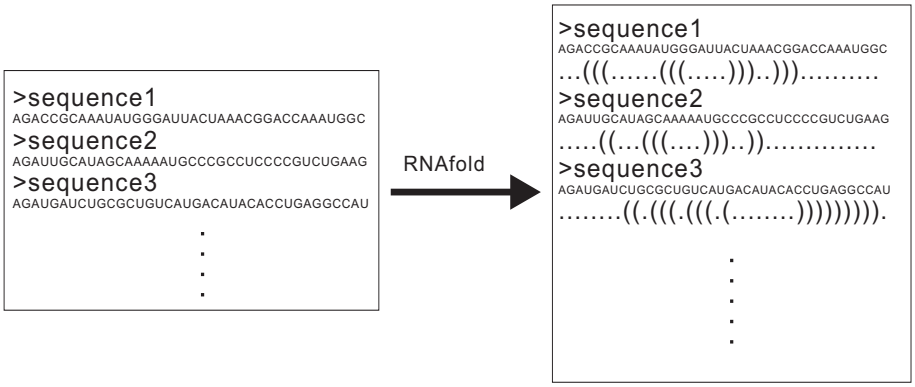


Figure 4.5: convert FASTA format to dot-bracket notation with RNAfold

step (4): Convert dot-bracket notation to gSpan

In this step, we will use our java program to convert the output of RNAfold to gSpan format. This step is crucial in our experiment, since we could convert a dot-bracket file to different gSpan files by using different graph models.

Before we use different graph models to convert the dot-bracket file to gSpan file, we could first build a model called *Plain_sequence model* based on the FASTA file (shown in Figure 4.6). In this model, we generate for each base in the RNA sequence a vertex v with the corresponding nucleotide (i.e., A/U/C/G) as the vertex's label. And between each neighboring two bases in the sequence, there is an edge with label b connecting them. Since this model has no structural context, we can compare the performance of this model with the performances of the other models with structural context, so that we could know whether the structural context of the binding site has positive influence to the binding or not. And the process of the experiment by using this model has some differences compared with the process by using RNAfold, that instead of using RNAfold and our java program, we directly

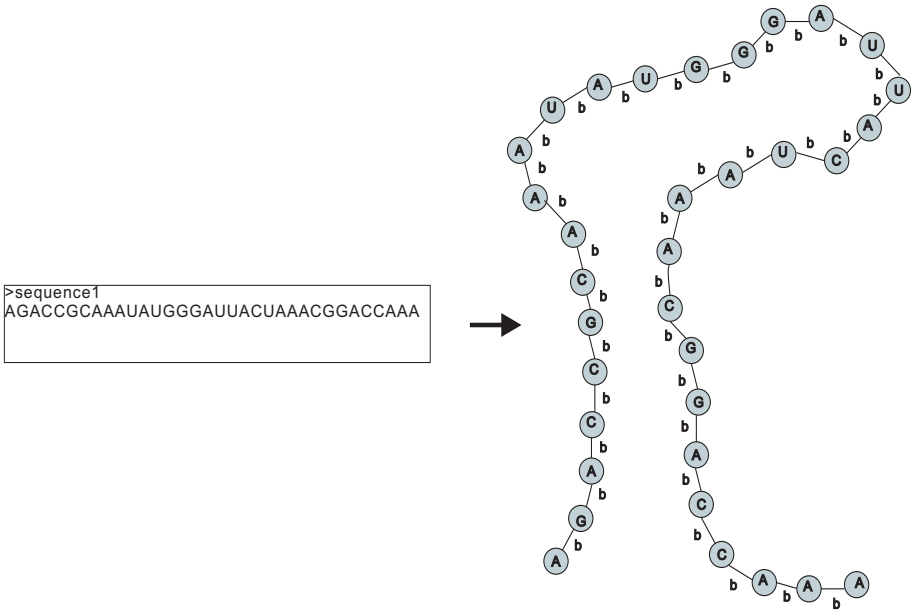


Figure 4.6: Plain_sequence model based on FASTA file

convert the FASTA sequences to the gSpans by using the Plain_sequence model. The red line in Figure 4.7 illustrates the difference.

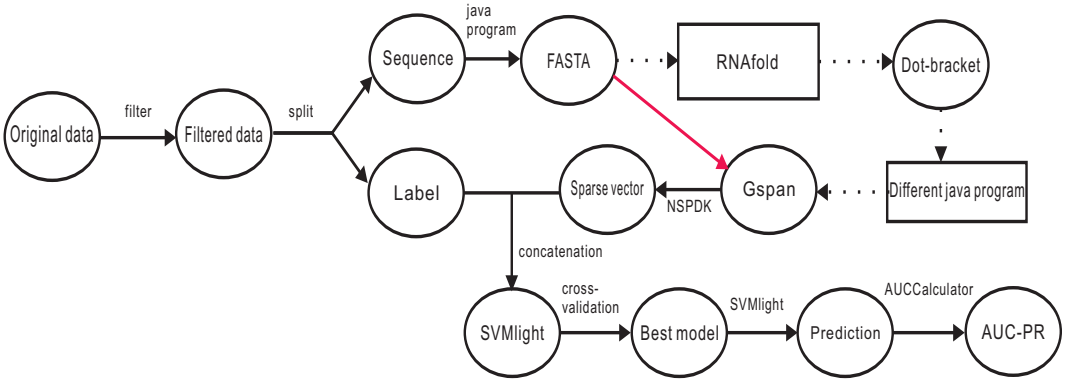


Figure 4.7: process of experiment based on Plain_sequence model

Now, let's have a look the three graph models we built based on the dot-bracket files generated by RNAfold:

(1) convert the MFE structure to the gSpan (*RNAfold_mfe model*).

(2) convert the centroid structure to the gSpan (*RNAfold_centroid model*).

(3) replace the labels of the vertices which are paired in the RNAfold_mfe model with label N (*RNAfold_covered model*).

The first and second model are used to see the influence of structural context to the binding, the only difference between them is just the folding idea. Similar with the Plain_sequence model, we take each nucleotide in the sequence as a vertex v with the corresponding nucleotide as its label, and between every two neighboring nucleotides, there is an edge with label b connecting them. And for the base pair formed by the corresponding two paired nucleotides, we also model it as an edge too, with label p. This model is used for both MFE structure and centroid structure generated by RNAfold to convert the dot-bracket notation to the corresponding gSpan (see Figure 4.8).

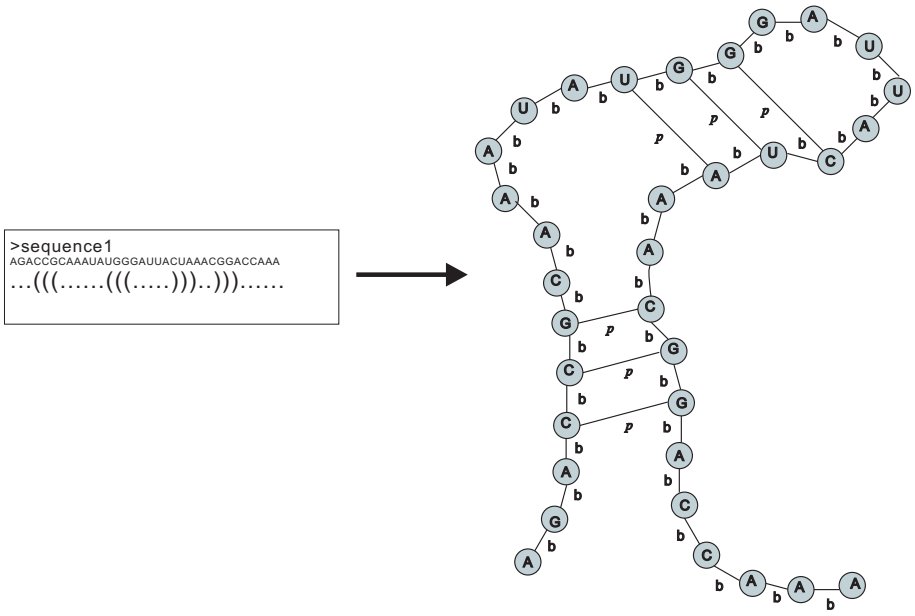


Figure 4.8: RNAfold_mfe model and RNAfold_centroid model

For the third model (shown in Figure 4.9), we replace the labels of paired

nucleotides in the RNAfold_mfe model with label N (we call this process "cover"). By comparing the performance of this covered model with the original RNAfold_mfe model, we could know whether the change of detailed nucleotides' information on the paired region will influence the binding or not. Since we already know experimentally that, most of the bindings occur in the single-stranded regions, but the sequestration of a motif in double-stranded region will cause strong negative influence to the binding or even abolishes the binding[5]. So by building this model, we cover the nucleotides paired in the latent binding regions (double-stranded region), and we want to see if the experiment result by using this model is consistent with the experimentally detected phenomenon — bindings are strongly negatively influenced or even abolished. Figure 4.10 shows the conversion process from dot-bracket notation to gSpan format by using our models.

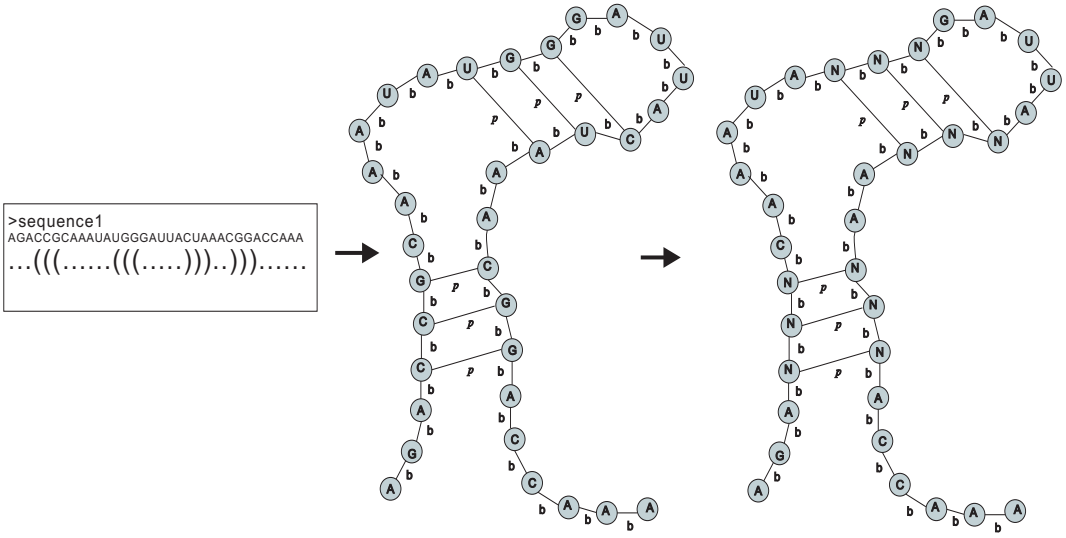


Figure 4.9: RNAfold_covered model

step (5): Convert gSpan to sparse vector

In Chapter 3, we have said that the graph kernel NSPDK is responsible for the conversion from gSpans to the sparse vectors. So after we have the

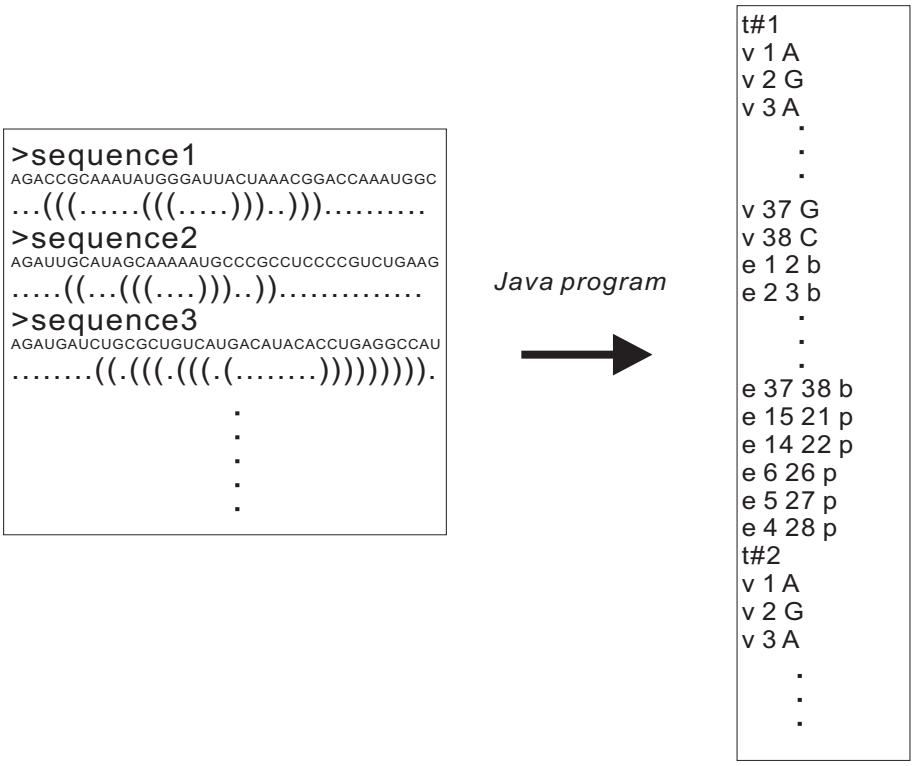


Figure 4.10: conversion from dot-bracket to gSpan

corresponding gSpans for each RNA sequence, we input them into NSPDK by using command "*./NSPDK -fg file -of -R 1 -D 1 -b 25*", where *-R* and *-D* are two parameters representing the radius and the distance as we have introduced in Chapter 3, and *-b* is used to control the length of the sparse vectors, since there are too many attributes. In order to incorporate more information from the input gSpan file, we set b to be 25, that is the largest value NSPDK can accept. In our experiment, R, D and the penalty parameter C are the three parameters used for the learning of our graph models. We will choose the optimal combination of these three variables in the training process, and evaluate the performance of the model fit with the optimal parameters on the test set. In our experiment, we set the range of R from 1 to 4, the range of D from 1 to 6, and the value of C within 1, 10, 100, 500.

The "file" in the command represents the gSpan file we input into NSPDK.

In step (2), we have split the filtered data into sequence part and label part, now we need to paste the original labels and the corresponding sparse vectors together, so that the pasted file can be processed by SVMlight.

step (6) Parameter optimization

In this step, we use the pasted file as input for SVMlight to do the parameter optimization (training) for our model by using five-fold cross-validation, which can be divided into two steps: training and validation, corresponding to the two modes of SVMlight: learn and classification. After the whole training process, we evaluate the performance of our model fit with the optimal parameters by doing prediction on the test set, we call this testing process. Corresponding to the two modes above, the testing process can also be divided into two steps: first we train our fit model on the whole training set to generate a model file, then we do prediction on the whole test set to get a prediction file consisting of predicted class labels. Figure 4.11 illustrates the relationships between the training process, testing process and the training and validation steps in the training process.

Take the RBP FUSIP1 as an example, we first train our model on the dataset *FUSIP1_weak_A*, by using different combinations of parameter R, D and C. In our experiment, we set the range of R in [1, 4], D in [1, 6], C in (1, 10, 100, 500). The optimal parameter search by using cross-validation is called *Grid-search*, it is straightforward but very naive, so it is time-consuming[9]. In order to save time but still achieve the same goal, we don't search the optimal parameters by training our model on the whole training set *FUSIP1_weak_A*, but on a subset of it. For example, we choose top 2000 sequences in the pasted sequence file as a subset for training (including training and validation steps), and after we find the optimal parameters,

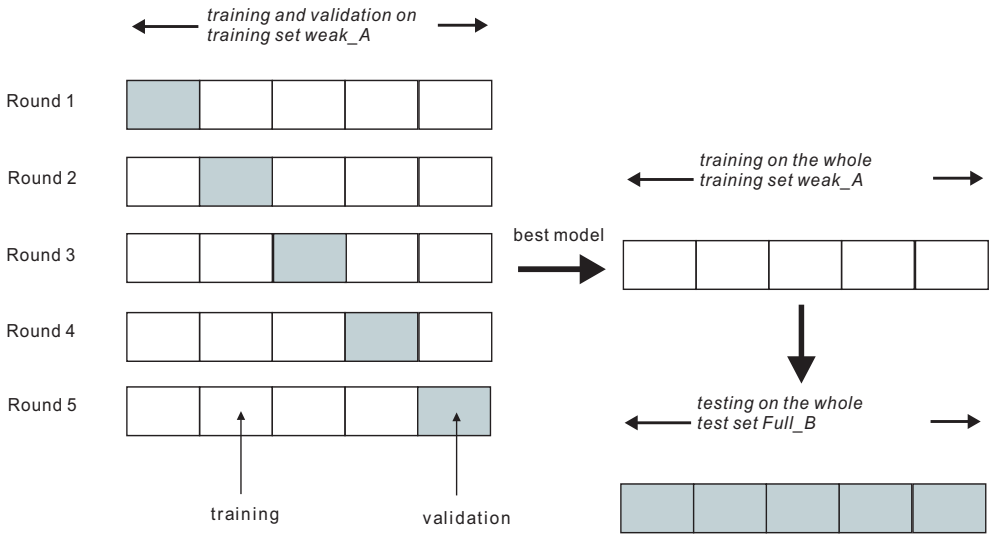


Figure 4.11: relation of training, validation and testing

we test our model fit with the parameters on the test set. But there may exist such a problem, that the sequences in the chosen subset have some structural similarities, while the remaining sequences in the pasted file have a quite different structure tendency. When we use the chosen subset to do cross-validation, it means we do training and validation on a set with certain similarities, the AUC-PR value could be very good. And the model trained on this set will reduce the reliability of our experiment. In order to avoid it, first we assign random numbers at the beginning of each sparse vector line in the input sparse vector file of SVMlight, then we sort the sparse vectors with ascending order by using the random numbers at the beginning of each line, so that we achieve the goal of shuffling the original sequences, and then we choose the top 2000 lines as a subset for cross-validation use.

We do the parameter optimization in this way: in each round, we let one of the three parameters to be variable once with the other two parameters being constant simultaneously. For example, in the first round, first we let R to be the variable and R changes from 1 to 4, D and C are constant

(e.g., $D = 1$, $C = 100$). So there are four parameter combinations, they are $R = 1$, $D = 1$, $C = 100$; $R = 2$, $D = 1$, $C = 100$; $R = 3$, $D = 1$, $C = 100$; $R = 4$, $D = 1$, $C = 100$. For each combination, we use five-fold cross-validation to do the training and validation, since in the training, we must make sure that there are enough examples to learn the model, that's why we prefer five-fold cross-validation. And we get five AUC-PR values for each combination after training and validation. Then we compute the average AUC-PR for each combination, and we get four average AUC-PR values for the four combinations respectively, then we choose the parameter combination with the maximum average AUC-PR. We assume that after computing the average AUC-PR for the four combinations, we have $R = 2$, $D = 1$, $C = 100$ that has the highest average AUC-PR value among the four. Then still in this round, we set $R = 2$ to be a constant, and we still keep $C = 500$ fixed, but we let D to be the variable, and D is from 1 to 6. In the same way, we get new optimal combination of R, D and C, from which we can pick the best value for parameter D, e.g., $D = 4$. At the end of the first round, we set C to be the variable, and R and D are constant this time, that means $R = 2$, $D = 4$, and we get $C = 10$. So after the first round, we get an optimal combination of the three parameters $R = 2$, $D = 4$, $C = 10$ by using the way above. And in the next round, we start to set variable from R again, R is still from 1 to 4, but this time, the relative constant value for D and C remain the optimal value of last round, i.e., $D = 4$, $C = 10$. Figure 4.12 shows the way we do the parameter optimization by using cross-validation. We do this until the values of R, D and C do not change any more, this usually takes about two or three rounds.

In the process of parameter optimization, since we use five-fold cross-validation, in each round of the five, we use 80% of the 2000 sparse vectors

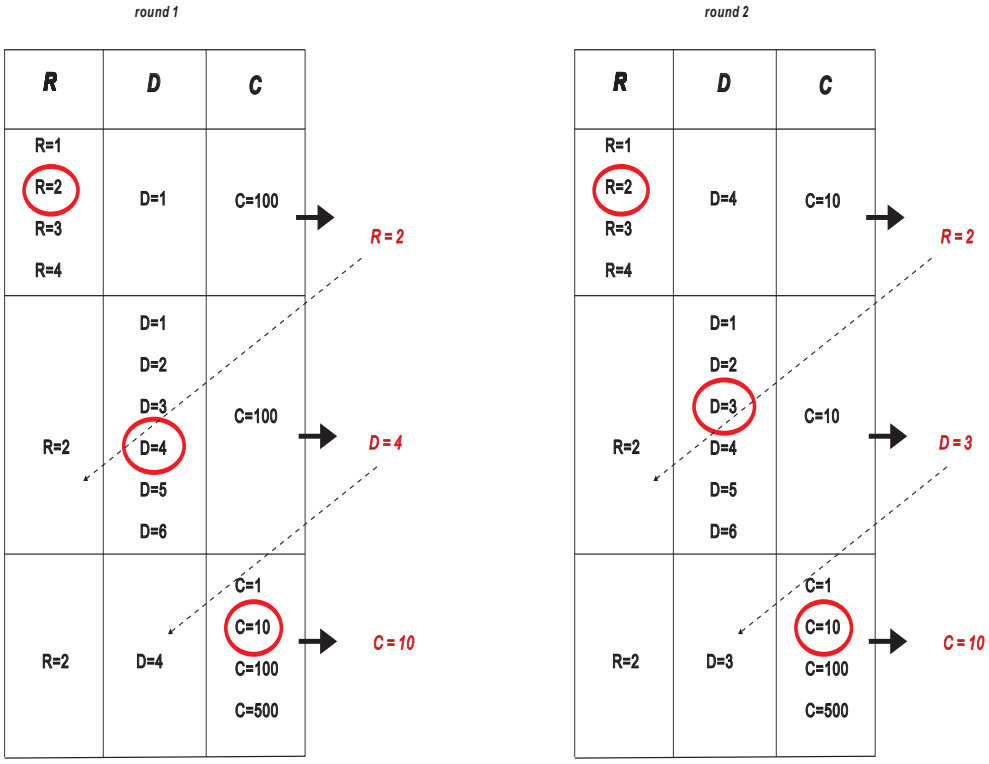


Figure 4.12: parameter optimization

for training and 20% for validation, and make sure that each 20% of the subset data will be taken as validation set only once for each combination of the three parameters. After the training and validation, we get the prediction file generated by SVMlight, it consists of 400 (20% $\times$ 2000 = 400) predicted target values, and this file will be used in the following step to compute the AUC-PR.

First we paste the prediction file and the 400 validation sequences' original labels together as a file, then we convert the pasted file to the format accepted by the AUCCalculator. That means, we set all the label +1 in the original label file to be 1, and all the label -1 to be 0, so that this file can be taken as the input of AUCCalculator (shown in Figure 4.13). After computation, the output of AUCCalculator is the AUC-PR value. This AUC-PR value

represents the performance of the model fit with the current parameters running on the current validation set. After the parameter optimization for two or three rounds, we find the optimal parameters for our model.

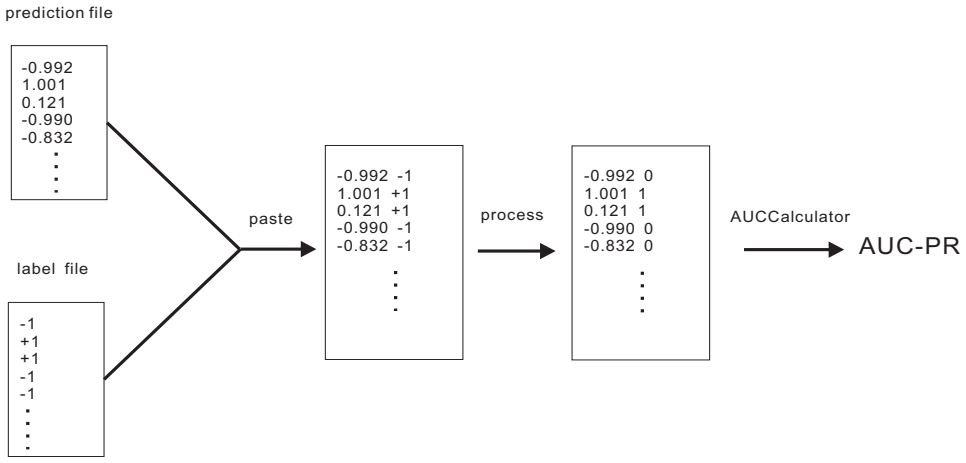


Figure 4.13: AUC-PR computation by using AUCCalculator

Above all, the process of parameter optimization (training process) consists of two steps: training and validation. These two steps are done by using five-fold cross-validation on a subset of the whole training set in our experiment, with each fold being the validation set exactly one for each combination of the three parameters. After we find the best parameters for the model, we will use this optimal parameter combination to fit our model and evaluate its performance in testing process.

step (7): Performance evaluation.

After we get the optimal combination of the three parameters R, D and C after training and validation on a subset, we will compute the performance of our fit model in testing process. We first train the fit model on the whole training set *FUSIP1_weak_A* to get a model file generated by SVM-light, then we do classification on the whole test set *FUSIP1_full_B* (not *FUSIP1_weak_B*) to generate the prediction file. In the training process on

the whole *FUSIP1_weak_A* set, in order to enhance the performance, except using parameter *C*, we also use parameter *-j*, which means how much the training errors on positive examples outweight errors on negative examples (default 1), it is set to be the ratio of negative to positive examples in the whole training set *FUSIP1_weak_A*. Since for each full and weak set of an RBP in the dataset, they are comprised of set *A* and set *B*, this provides an advantage for us to use two-fold cross-validation, so that we can train our model on one set, and test on the other, vice verse. Take FUSIP1 as an example, the process of the two-fold cross-validation is as shown in Figure 4.14.

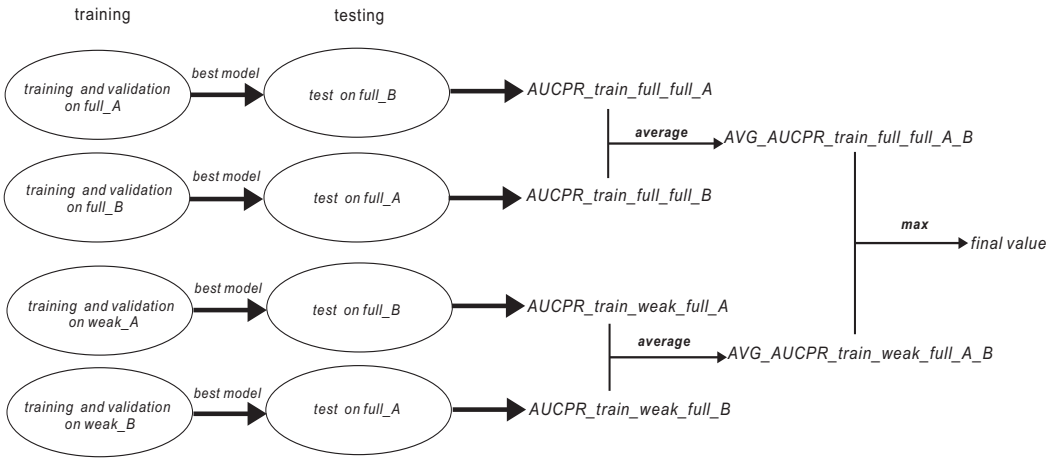


Figure 4.14: two-fold cross-validation for training and testing in our experiment

(*i*) First we train our model on the relative subset of *FUSIP1_full_A*, *FUSIP1_full_B*, *FUSIP1_weak_A* and *FUSIP1_weak_B* respectively as introduced in step (6), and find their corresponding optimal parameters by using five-fold cross-validation after two or three rounds. Then we use SVMlight to train the four fit models on the corresponding whole training sets to get the model files used for the classification. For example, after we get the

optimal parameters for the model trained and validated on the subset of *FUSIP1_full_A*, we train this model on the whole *FUSIP1_full_A* set to generate a model file.

(*ii*) After we get the model file from SVMlight, we do classification (prediction) on the whole test set *FUSIP1_full_B* by using this model file, and we get the prediction file which will be used together with the labels of the test set *FUSIP1_full_B* later for the model's performance evaluation. For the performance, we evaluate it with AUC-PR, which is computed by AUCCalculator in the way illustrated in Figure 4.13. For the AUC-PR generated by training on *FUSIP1_full_A*, and testing on *FUSIP1_full_B*, we call this value *AUCPR_train_full_full_A*. For the AUC-PR trained on *FUSIP1_full_B* and tested on *FUSIP1_full_A*, we call this value *AUCPR_train_full_full_B*. But for the models trained on *FUSIP1_weak_A* and *FUSIP1_weak_B*, we test them on *FUSIP1_full_B* and *FUSIP1_full_A* respectively, rather than on *FUSIP1_weak_B* and *FUSIP1_weak_A*. we call these two AUC-PR values *AUCPR_train_weak_full_A* and *AUCPR_train_weak_full_B*.

(*iii*) After the four parallel training and testing steps, we will get four AUC-PR values. Then we average the AUC-PR trained on *FUSIP1_full_A* and *FUSIP1_full_B*, and we call this average value *AVG_AUCPR_train_full_full_A_B*. Also we average the AUC-PR trained on *FUSIP1_weak_A* and *FUSIP1_weak_B*, and we call this average value *AVG_AUCPR_train_weak_full_A_B*.

(*iv*) After having the two average values, the last thing we will do is to compare them and choose the maximum one as the final AUC-PR value for *FUSIP1*, which means this value is the predictive performance of binding preference for *FUSIP1* by using the model in our approach.

4.2.2 Process of experiment based on RNAshapes

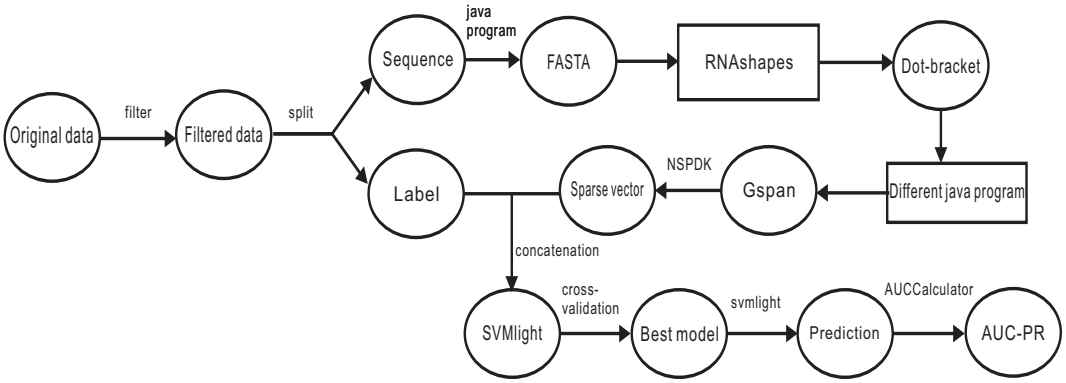


Figure 4.15: process of experiment by using RNAshapes

The experiment process by using RNAshapes is similar to RNAfold, the only difference is just replacing RNAfold with RNAshapes (shown in Figure 4.15). And by using RNAshapes, we compute the secondary structure of RNA sequence by using different parameters, so that we could get corresponding structures based on different folding ideas. In our experiment, we first use parameter *-s* to get the *complete suboptimal folding*[20] structures for each FASTA sequence. Different to MFE structure, the suboptimal folding structures for a sequence are not a single structure, but the structures within a certain range of the minimum free energy (default 10%). After we get the complete suboptimal folding structures for a sequence, we convert the structures to a big graph. For example, we assume that a sequence has two different secondary structures in the complete suboptimal folding, we convert the first structure to a graph in the same way as RNAfold_mfe model, that means we generate a vertex for each nucleotide in the sequence with the corresponding nucleotide's information as the vertex's label, and the sequence numbers of the vertices are from 1 to the sequence length. Between each two neighboring nucleotides, there is an edge with label b connecting them.

For the base pair formed by two nucleotides, we model it as an edge with label p. Then for the second secondary structure of the same sequence, we model it by using the same graph model as above, but with the sequence numbers of vertices starting from "the sequence length + 1" to "2 × the sequence length", so that the two secondary structures of the sequence form an integrated graph. We call this graph model based on complete suboptimal folding *RNAshapes_sub model* (see Figure 4.16). We build this model in order to know, if the complete suboptimal folding can have better performance than MFE or centroid structure folding.

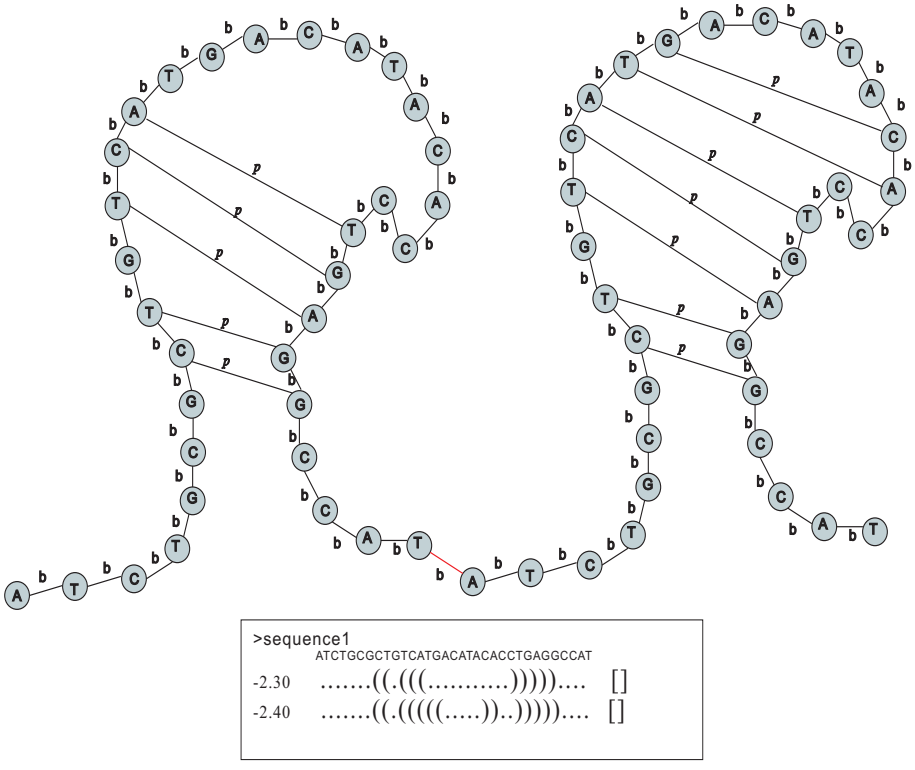


Figure 4.16: convert the suboptimal folding structures into a graph

Except using parameter -s, we could also get the shape probabilities based on partition function and the corresponding structures that fall into that shape by using parameter -q[20]. Same with model RNAshapes_sub, we

convert all the structures of a sequence into an integrated graph. Since for one RNA sequence, there could be many shapes with a lot of secondary structures falling into each of them, we can use parameter -T to omit the shapes with low probabilities. In our experiment, We set the value of -T to be 0.1, that means we only leave the shapes with probabilities larger than or equal to 0.1. And another important factor we have to take into consideration is that there are five different shape types in RNAshapes representing the different levels of shape abstraction[21]. In our experiment, we use level 1 (the most accurate level) and level 5 (the most abstract level), and the parameter -t is used to control the level. We call the models generated by these two shape levels *RNAshapes_q_1 model* and *RNAshapes_q_5 model*. By building these two models, we want to know whether the factor of shape probability has positive influence to the binding, and we also want to know the binding preference of the RBPs to the shape abstraction level.

4.2.3 Process of experiment based on RNAplfold

Different to RNAfold and RNAshapes, RNAplfold does not return the secondary structure for each sequence directly, but computes average pair probabilities for local base pair i and j within a span L (default 70)[18]. In our experiment, we use the parameter -u (default 31) to compute the unpaired probability of x consecutive nucleotides from position i in the sequence. Then the output of RNAplfold is a matrix with each line consisting of a position i and the unpaired probability of base i, [i - *1,...,i*], [i - *2,...,i*] until [i - X + *1,...,i*]. In our experiment, in order to see whether the unpaired probability of each individual nucleotide has positive influence to the binding or not, we set the value of -u to be 1. And the output matrix is saved in a "*_lunp*" file. Figure 4.17 shows the probability matrix of a FASTA sequence.

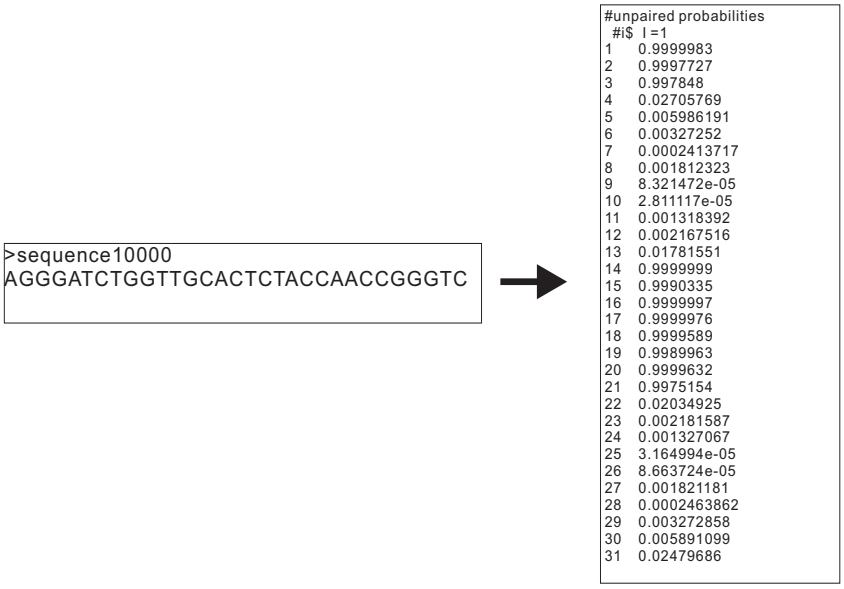


Figure 4.17: the probability matrix of a FASTA sequence generated by RNAplfold

Since the output of RNAplfold for each FASTA sequence is a probability matrix, the process of the experiment based on it has some difference compared with RNAfold. As the RNAfold, we also start from the original dataset, we filtered the original data with different cutoff and median values for set A and set B of both weak and full set of the nine RBPs. Then we split the filtered data into two parts: labels and sequences.

On one side, for the sequence part, we still convert it to FASTA format first, this is exactly the same as RNAfold. Then we use RNAplfold to process the FASTA files. But the output of RNAplfold are many *_lunp* files such like *sequence1_lunp*, *sequence2_lunp*,...,*sequence1000_lunp*, each file contains the corresponding probability matrix of the FASTA sequence with same sequence number. In the probability matrix, it contains the unpaired probability of each nucleotide in the corresponding sequence by using option *-u*,

and the number of *_lunp* files equals the number of the RNA sequences in the FASTA file. Then we use our java program to write each FASTA sequence's information (sequence description and sequence) to the corresponding *_lunp* file which has the same sequence number. As an example, we assume that in a FASTA file, there are 100 RNA sequences, then the number of generated *_lunp* files is 100 (*sequence1_lunp, sequence2_lunp,...,sequence100_lunp*). We write the corresponding sequence's information into the *_lunp* file with the same sequence number, which means we write the information of *sequence1, sequence2,...,sequence100* into the file *sequence1_lunp*, *sequence2_lunp,...,sequence100_lunp* respectively. Then we concatenate all of these *_lunp* files after writing, so that we get a single *_lunp* file containing each FASTA sequence's unpaired probability matrix with the corresponding FASTA sequence's description and sequence information following the matrix. Then we can use java programs to convert this file to different gSpan files by using different graph models. And the left steps remain the same as in the process of RNAfold. Figure 4.18 illustrates the experiment process by using RNAplfold.

On the other side, for the label part, if we still paste the sparse vector file with the original label file to do the cross-validation on SVMlight directly, there will be an error. Since during the process of concatenation, we use the Linux command "cat" to concatenate all the *_lunp* files in an ascending order, e.g., you will find that in the final concatenated *_lunp* file, the content of file *sequence13_lunp* will be in front of the content of file *sequence2_lunp*, but will be behind the content of file *sequence02_lunp*. So if we still use the original label file directly in the following steps, the order of labels is different to the sequence order in the concatenated *_lunp* file, and this will cause an error. In order to solve it, we first use our java program to split

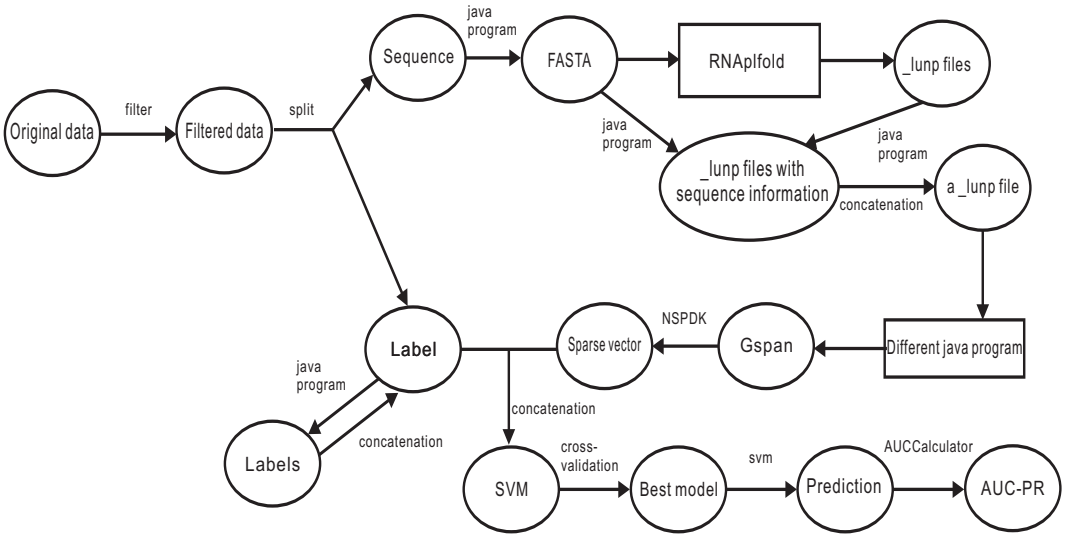


Figure 4.18: process by using RNAplfold (including RNAplfold_original model and RNAplfold_NP model)

the label file into thousands small label files with each containing only one label. For example, in the original label files, it has 100 labels (+1/-1) with each label in one line. We then split the original label file, so that the labels in the *line1, line2,...,line100* will be written into the generated label files with names *1.label*, *2.label*,...,*100.label* respectively. The number of generated small label files equals the number of labels in the original label file. Then we concatenate these small files again to form a new label file, but the order of labels in it has already changed, and this order is same with the sequence order of the concatenated *_lunp* file. Then we could use them to do the cross-validation on SVMlight in the following steps of our experiment.

By using RNAplfold, we design four models: In the first model (*RNAplf-old_original model*), for each FASTA sequence, we first model the sequence as the Plain_sequence model introduced before. That means each nucleotide in the sequence will be represented as a vertex with the corresponding nucleotide

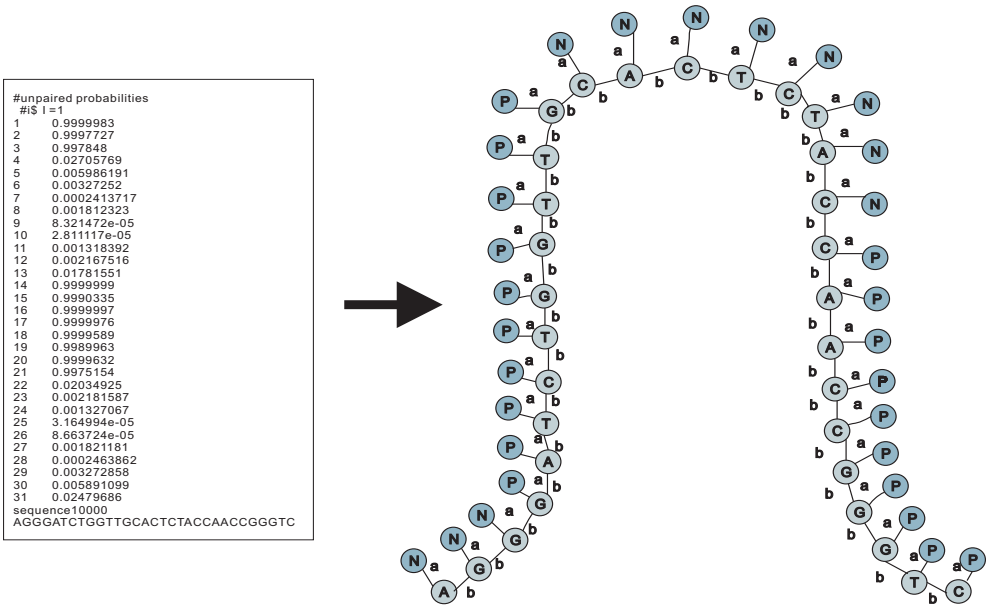


Figure 4.19: RNAplfold_original model

as its label, and between each two neighboring vertices, there is an edge with label b connecting them. Since we have the unpaired probability of each nucleotide in the sequence, if the probability of one nucleotide is larger than or equal to 0.5, we say this nucleotide has higher probability to be unpaired than being paired. So we generate a new vertex above the corresponding nucleotide's vertex in the model by using an edge with label a connecting them, and the label of the generated vertex is N. On the other side, if the unpaired probability is less than 0.5, we set the label of the new vertex to be P, which means the nucleotide has higher probability to be paired than being unpaired. The model is shown in Figure 4.19.

The second graph model (*RNAplfold_NP model*) has the same idea with the first model, we just do a tiny change, that if the unpaired probability of a nucleotide is larger than or equal to 0.5, except adding the vertex N above the vertex of the nucleotide in the Plain_sequence model, we also add a vertex P

above the added vertex N by using an edge with label a connecting these two added vertices. To the opposite, if the unpaired probability of a nucleotide is below 0.5, we just replace the positions of the corresponding vertex N and P (see Figure 4.20). By building the two models above, we could test if the unpaired probabilities of the bases in RNA sequence have positive influence to the binding or not.

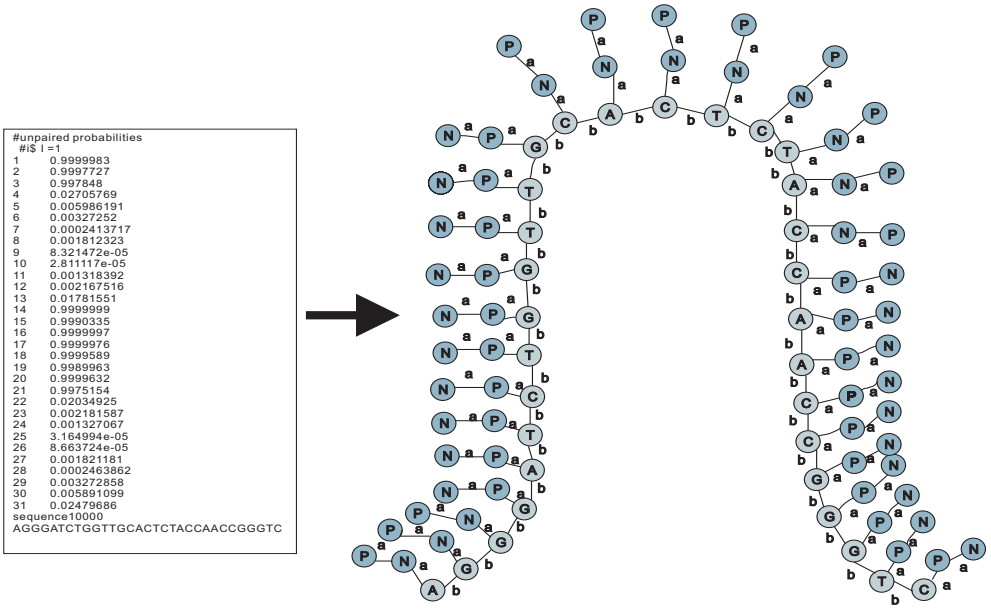


Figure 4.20: RNAplfold_NP model

For the third model (*RNAplfold_modified model*), we use a modified version of RNAplfold. For a FASTA sequence, except generating a *_lunp* file, this modified RNAplfold also generates another four files: *_lunp_H*, *_lunp_I*, *_lunp_M*, *_lunp_E*. These four files contain the probability of each nucleotide in the sequence to be in the state hairpin loop, interior loop, multiloop and external region respectively. That means the unpaired probability for each position in the sequence is split up by the structural contexts. This is similar with the annotation alphabet in RNAcontext (i.e., P, L, M, U). And for each

nucleotide, the sum of the corresponding probabilities in the four files should be equal to the probability of the corresponding position in the original *_lunp* file, because if a nucleotide is not paired, then it must be in one of the four states above. After we get these files, for a given sequence, if the probability of a position i in the original *_lunp* file is larger than or equal to 0.5, except converting the sequence to a model in the way illustrated as the model RNAplfold_NP, we still compare the four probabilities at the same position in the other corresponding four generated files. If the maximum value among the four locates in the *_lunp_H* file, then we know that the probability for this nucleotide to be in the hairpin loop is biggest among the four states, then we add a vertex H to connect the vertex N in the model RNAplfold_NP by using an edge with label a between them. If the probability of a position i in the original *_lunp* file is less than 0.5, we don't do any change to the corresponding vertices compared with RNAfold_NP model. Figure 4.21 shows the RNAplfold_modified model.

The change in the experiment process based on RNAplfold_modified model is as follows: For each sequence, the outputs generated by the modified RNAplfold are five probability matrices contained in the *_lunp, _lunp_H, _lunp_I, _lunp_M, _lunp_E* file respectively. Then we write the corresponding sequence's information into its corresponding five output files, so that in each of the five output files, there is a probability matrix followed by the corresponding sequence description and sequence information. Then, we concatenate all the relative *_lunp, _lunp_H, _lunp_I, _lunp_M, _lunp_E* files respectively to get a *_lunp* file, a *_lunp_H* file, a *_lunp_I* file, a *_lunp_M* file and a *_lunp_E* file. At last we paste all of these five concatenated files into a final file. In this file, for each sequence, there are five corresponding probability matrices followed by the sequence description and information. The

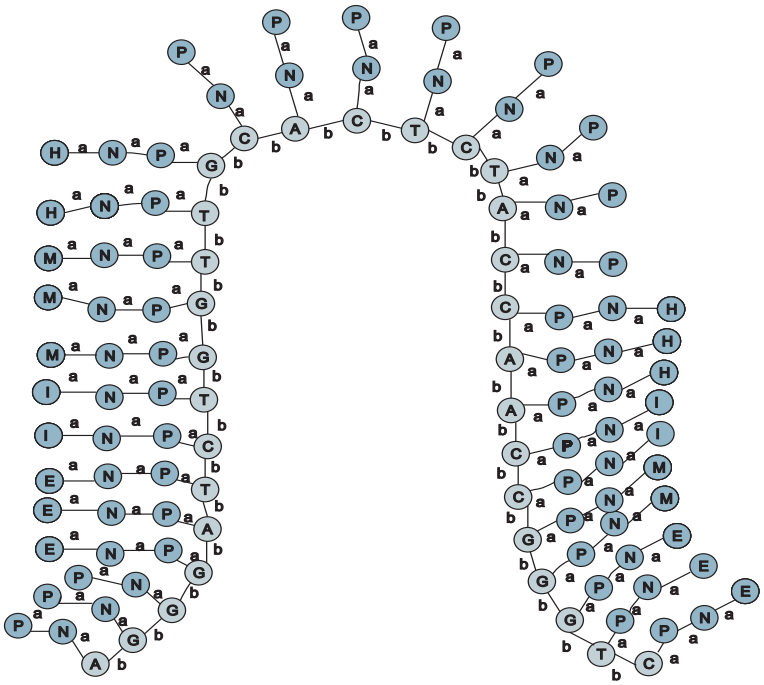


Figure 4.21: RNAplfold_modified model

five matrices represent the relative probabilities that the nucleotides in the sequence are unpaired, in a hairpin loop, in an interior loop, in a multiloop or in an external region. Then we could use our java program to work on this file, and if the unpaired probability for a position in the first matrix is larger than or equal to 0.5, we compare the other four probability values in the same position of the other four matrices, so that we could know the state of the unpaired nucleotide. Then we use RNAplfold_modified model to convert the pasted file into a gSpan file. Figure 4.22 shows the experiment process by using RNAplfold_modified model.

For the last model by using RNAplfold, we combine the RNAfold and the modified RNAplfold. That means except building the RNAplfold_modified model, we also check if some nucleotides in the sequence form base pairs by using RNAfold. We call this model *RNAplfold_combine model* (see Figure

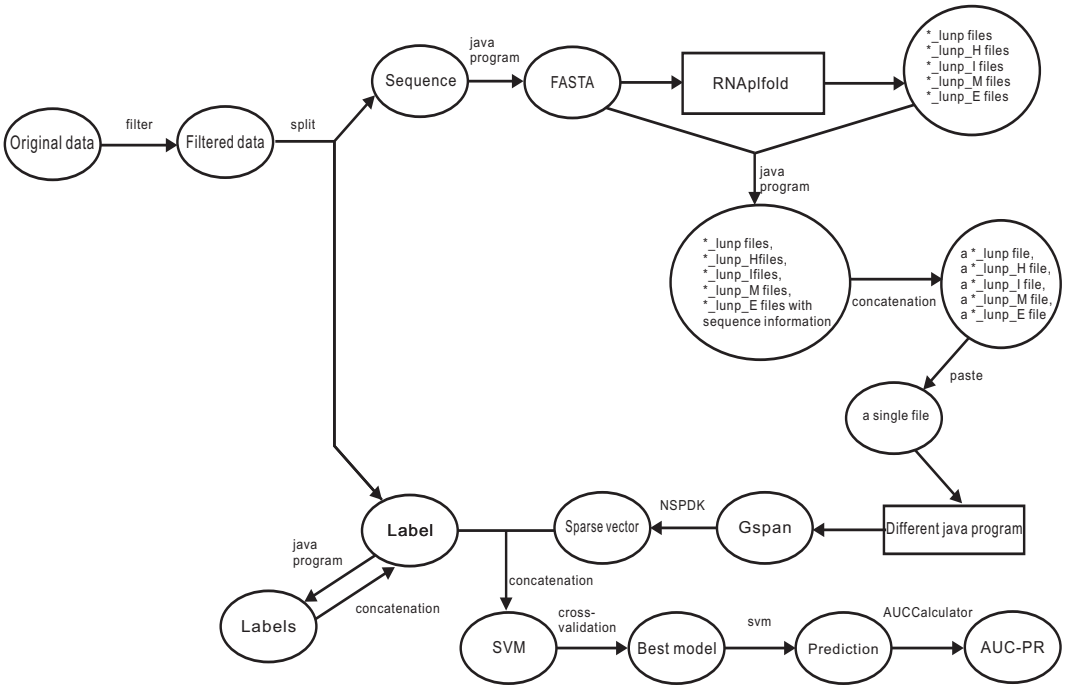


Figure 4.22: process of experiment based on RNAplfold_modified model

4.23). The process of the experiment based on this model has only a little difference to the process based on RNAplfold_modified, that except using modified RNAplfold to get the corresponding five files for each sequence, we also use RNAfold to compute the corresponding MFE structure for each sequence. Then we use our java program to write the sequence and its MFE structure information into the corresponding five generated files, then use the same way to convert it and the left steps remain the same. Figure 4.24 illustrates the experiment process by using RNAplfold_combine model.

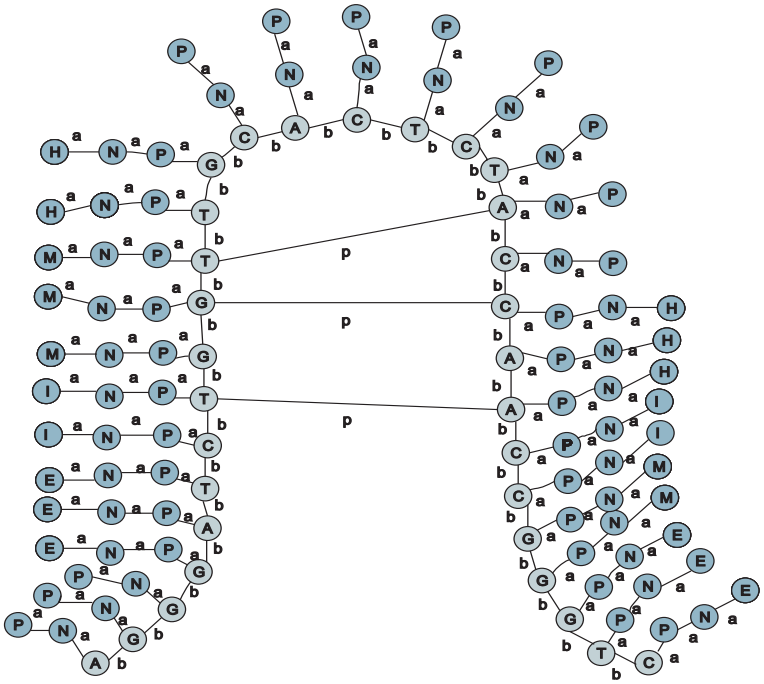


Figure 4.23: RNAplfold_combine model

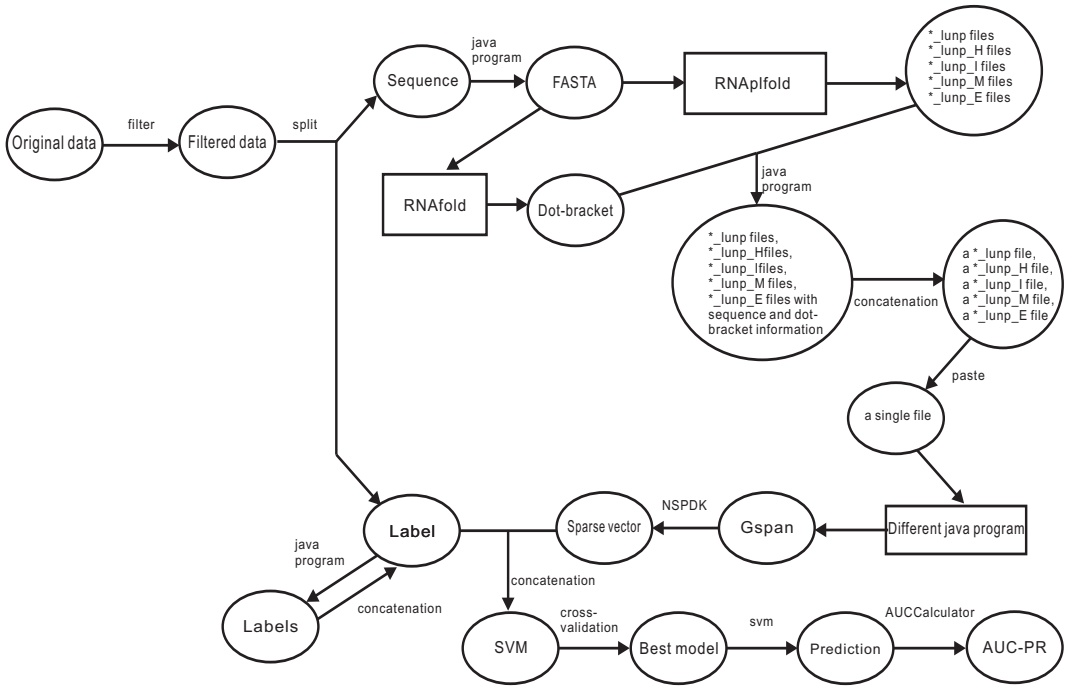


Figure 4.24: process of experiment based on RNAplfold_combine model

Chapter 5

Experiment

5.1 Dataset

As RNAcontext, we use RNAcompete-derived datasets in our experiment. The datasets are an RNA pool consisting of 213,130 unique short RNA sequences, with each sequence's length between 29- to 38-nt. The datasets are comprised of nine RBPs (i.e., HuR, Vts1p, PTB, FUSIP1, U1A, SF2/ASF, SLM2, RBM4, YB1) with their corresponding measured binding affinities computed by RNAcompete and other *vivo* method[1].

The RNA pool can be divided into two sets: set A and set B. Both of these two sets satisfy the following two constraints[1]:

(1) each loop with length between 3 and 7 (inclusive) is represented on at least one sequence flanked by RNA stems of 10 bases;

(2) each possible 7-mer appears in at least 64 different sequences with high folding free energy, so the sequences are linear or form weak secondary structures.

We call the sequences satisfying the first condition *stem-loop* sequences, while the sequences fulfilling the second constraint *weakly-structured* sequences.

There is no overlap between these two sequence groups. And the sequences in the pool have two characteristics[1]: (1) there are some unintended sequences in the pool (e.g., some weakly-structured sequences contain stem-loops). (2) no two sequences in the pool have a common subsequence with length more than 12-nt long. The design of the RNA pool and the sequence properties can be found in [2].

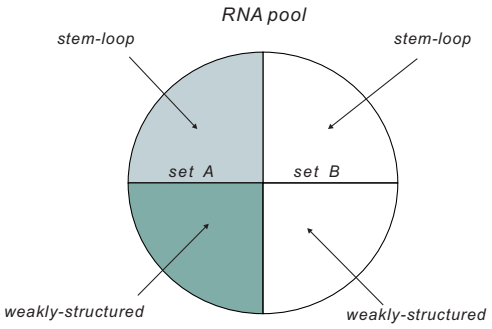


Figure 5.1: the division of the RNA pool

From Figure 5.1, we could see that there are two different stem-loop groups and two different weakly-structured sequences groups in set A and set B respectively. This construction provides us the advantage to train our graph model on one set, then test our model's performance on the other set, and vice verse, so that we can get two independent performance measurements[1]. This strategy is called *two-fold cross-validation*.

As the way in RNAcontext, for each of our graph models, we also trained two sets of models. One set was trained on the full training set comprised of all the RNA sequences in the training set. The other set of models was trained with the weakly-structured sequences in the training set, which means the stem-loop sequences are removed from the set. So, in the RNAcompete-

derived datasets, for each RBP, there are two training sets: full set and weak set. Figure 5.2 shows the construction of full and weak set for one RBP in RNAcompete-derived datasets.

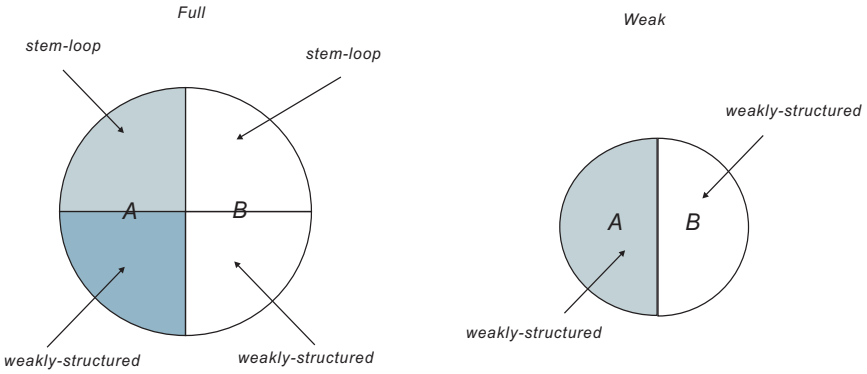


Figure 5.2: the construction of full set and weak set for an RBP

Above all, in RNAcompete-derived datasets, the full set data is comprised of the full sets of all the nine RBPs. And for each RBP, the full set consists of set A and set B. In the same way, the weak set consists of all the weak sets of the nine RBPs with each containing set A and set B respectively. Figure 5.3 illustrates the construction of full and weak sets of RNAcompete-derived dataset. The full and weak sets are comprised of the relative full and weak

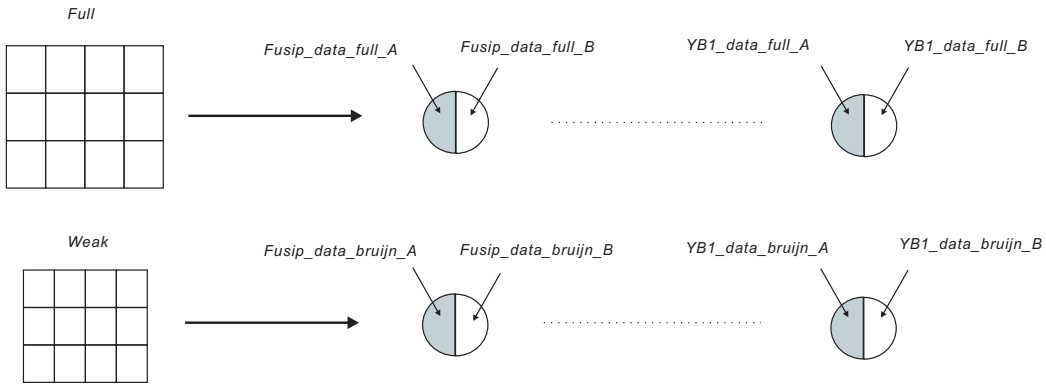


Figure 5.3: the construction of RNAcompete-derived datasets

sets of the nine RBPs. And both the full and weak set of an RBP can be divided into set A and set B.

5.2 Experiment result

We implement our experiment in the way introduced in Chapter 4 on the dataset in Linux environment. And after computation, we compare our method's performance with RNAcontext — the current best method for the prediction of RBPs' binding preferences. The nine RBPs' AUC-PR values by using our different models are as follows:

Proteins	Plain_sequence	RNAfold_mfe	RNAfold_centroid	RNAfold_covered
RBM4	0.62	0.68	0.67	0.22
FUSIP1	0.73	0.84	0.84	0.18
vts1p	0.36	0.38	0.33	0.26
YB1	0.20	0.21	0.22	0.08
SLM2	0.70	0.67	0.66	0.47
SF2	0.82	0.76	0.78	0.40
U1A	0.47	0.62	0.62	0.48
HuR	0.92	0.91	0.91	0.72
PTB	0.70	0.68	0.68	0.54

Table 5.1: Performances of the Plain_sequence model and the models related with RNAfold

Proteins	RNAcontext	RNAshapes_sub	RNAshapes_q_1	RNAshapes_q_5
RBM4	0.91	0.63	0.65	0.65
FUSIP1	0.53	0.84	0.83	0.84
vts1p	0.65	0.43	0.37	0.35
YB1	0.17	0.24	0.25	0.21
SLM2	0.81	0.65	0.66	0.64
SF2	0.70	0.77	0.77	0.77
U1A	0.30	0.57	0.63	0.58
HuR	0.96	0.90	0.90	0.91
PTB	0.69	0.66	0.68	0.68

Table 5.2: Performances of RNAcontext and the models related with RNAshapes

Proteins	RNAplfold_original	RNAplfold_NP	RNAplfold_modified	RNAplfold_combine
RBM4	0.38	0.38	0.62	0.61
FUSIP1	0.28	0.26	0.83	0.84
vts1p	0.21	0.20	0.49	0.44
YB1	0.11	0.20	0.23	0.21
SLM2	0.60	0.60	0.67	0.65
SF2	0.80	0.79	0.79	0.72
U1A	0.32	0.31	0.67	0.64
HuR	0.91	0.91	0.91	0.89
PTB	0.60	0.60	0.65	0.64

Table 5.3: Performances of the models related with RNAplfold

From table 5.1, we could see that for the RBPs RBM4, FUSIP1, vts1p, YB1, U1A, the AUC-PR values in model RNAfold_mfe are larger than the corresponding values in model Plain_sequence. This tells us that, for these five RBPs, the structural context folded by using MFE idea in the binding region has positive influence to the binding, while for the other four RBPs, the structural context has negative influence or no obvious positive influence. This result is consistent with the result of RNAcontext experiment. In RNAcontext, using the structural context leads to a significant improvement in AUC-PR for eight of the nine RBPs, among which the AUC-PR of RBPs vts1p, RBM4, FUSIP1, U1A, YB1 increase dramatically[1]. So, to some extent, our experiment's result is consistent with the result of RNAcontext. And still in this table, we could find that the performance of RNAfold_covered model is much worse than the other three models. Since model RNAfold_covered replaces all the paired nucleotides' labels with label *N*, this change may destroy the structural context as well as the specific sequence of a latent binding motif in the double-stranded region. And we already know that the sequestration of a motif locating in the double-stranded region will have strong negative influence to the binding, so the experiment result of model RNAfold_covered is consistent with the experimentally detected phenomenon.

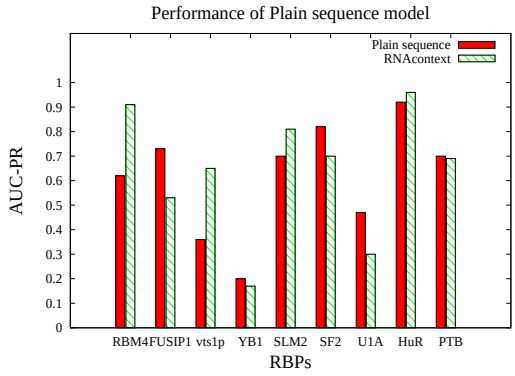
In table 5.2, the AUC-PR values of nine RBPs in the three models RNAshapes_sub, RNAshapes_q_1 and RNAshapes_q_5 have no big difference, and their performances have no obvious improvement compared with the model RNA_mfe or RNA_centroid. So we think that, the RBPs recognize the target RNA from the aspects of structural context and specific sequence, rather than the shape probability of the secondary structures. By comparing the performances of model RNAshapes_q_1 and model RNAshapes_q_5,

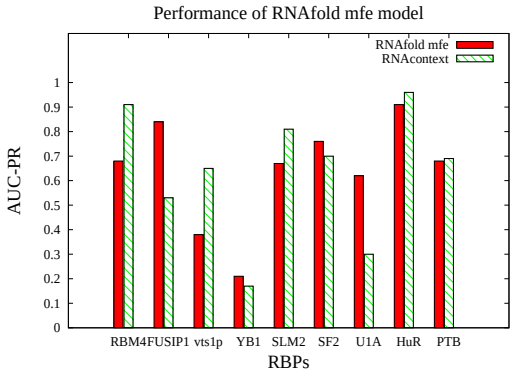
we find that, four of the nine RBPs' AUC-PR values in the former one are better than the latter one, two are a little worse. So we can think that the RBPs recognize the target RNA sequence in accurate shape level, the reason is probably that the RBPs bind the sequence in a sequence-specific manner.

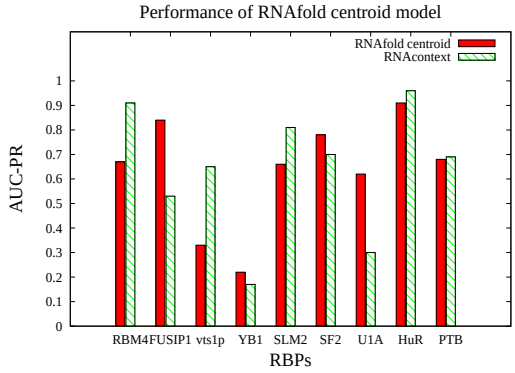
In table 5.3, for the model RNAplfold_original and RNAplfold_NP, seven of the nine AUC-PR values are much worse than the corresponding values in RNAcontext (except SF2, U1A), and all of the nine AUC-PR values are worse than the corresponding values in Plain_sequence model. So can we say that, for all of the tested RBPs, the unpaired probability of each individual nucleotide in an RNA sequence has no positive influence to the binding. And the performances of model RNAplfold_modified and RNAplfold_combine are much better than the model RNAplfold_original and RNAplfold_NP, which tells us that the detailed structural context of each individual nucleotide in the RNA sequence has positive influence to the binding.

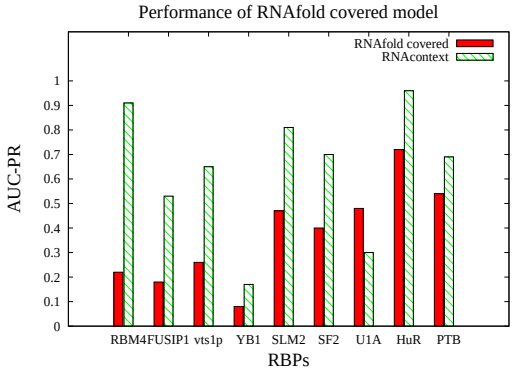
From the three tables above, we could find that, by using the model Plain_sequence, RNAfold_mfe, RNAfold_centroid, RNAshapes_sub, RNAshapes_q_1, RNAshapes_q_5, RNAplfold_modified and RNAplfold_combine, the AUC-PR values of RBPs FUSIP1, SF2, U1A, YB1 are always better than the corresponding values in RNAcontext. In model Plain_sequence, even PTB's AUC-PR is higher than in RNAcontext. So we can say that, by using our graph model without structural context, the performances of binding preference prediction for RBPs FUSIP1, SF2, U1A, YB1 and PTB have reached the state-of-the-art level, while by using our model with structural context we can reach the state-of-the-art performance for four of the five RBPs above except PTB.

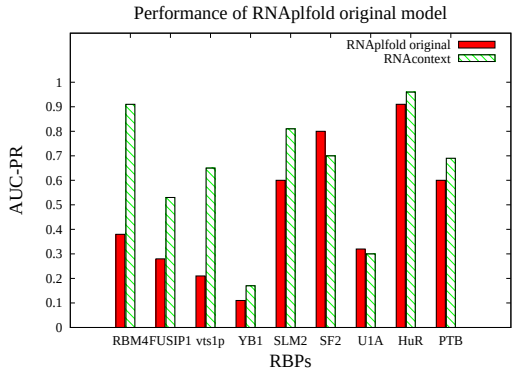
From the perspective of models, the performance comparisons between our models and RNAcontext are as follows:

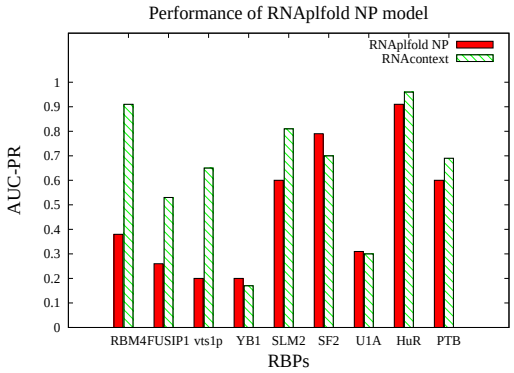

Performance of Plain sequence model
Plain sequence
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

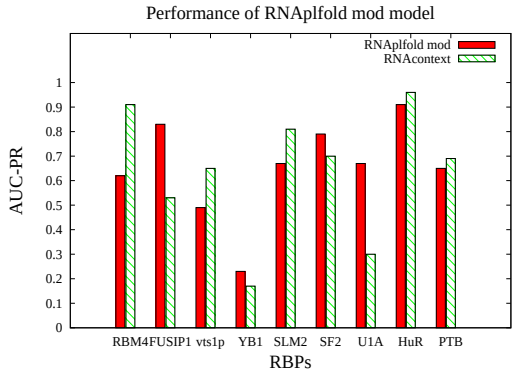

Performance of RNAfold mfe model
RNAfold mfe
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

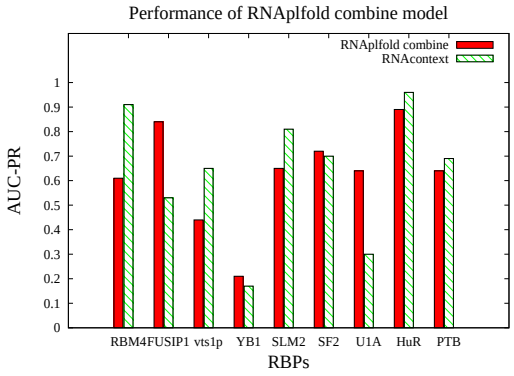

Performance of RNAfold centroid model
RNAfold centroid
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

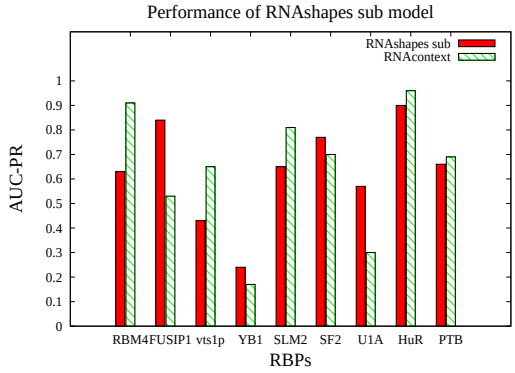

Performance of RNAfold covered model
RNAfold covered
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

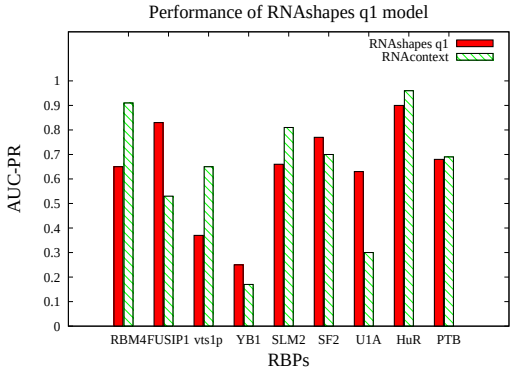

Performance of RNAplfold original model
RNAplfold original
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

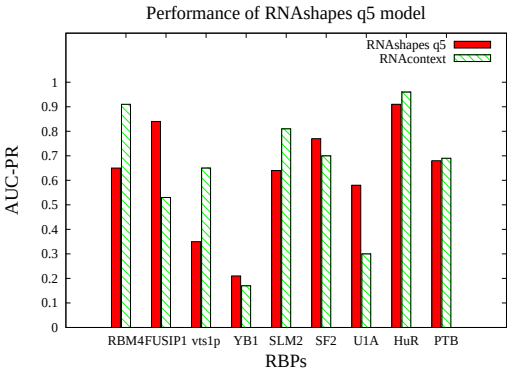

Performance of RNAplfold NP model
RNAplfold NP
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs


Performance of RNAplfold mod model
RNAplfold mod
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

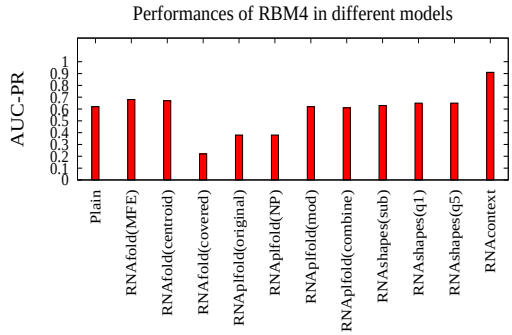

Performance of RNAplfold combine model
RNAplfold combine
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs


Performance of RNAshapes sub model
RNAshapes sub
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

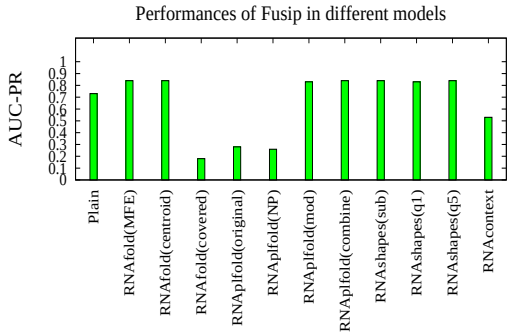

Performance of RNAshapes q1 model
RNAshapes q1
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs


Performance of RNAshapes q5 model
RNAshapes q5
RNAcontext
AUC-PR
RBM4FUSIP1 vts1p YB1 SLM2 SF2 U1A HuR PTB
RBPs

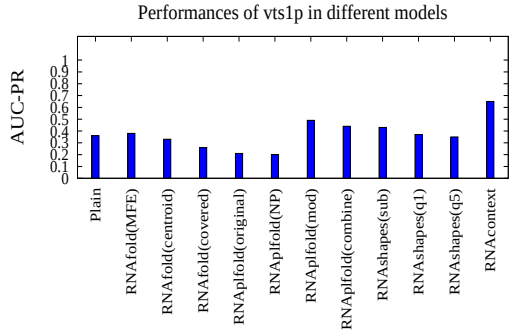
From the perspective of RBPs, the performance comparisons for each of the nine RBPs in different models are as follows:



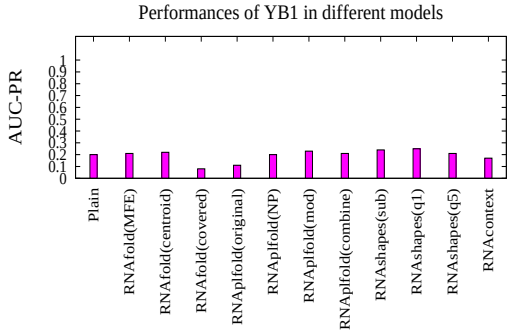
RBM4



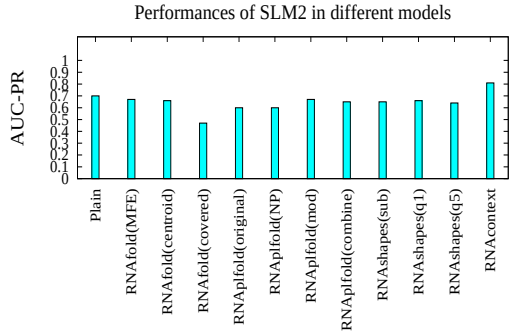
Fusip



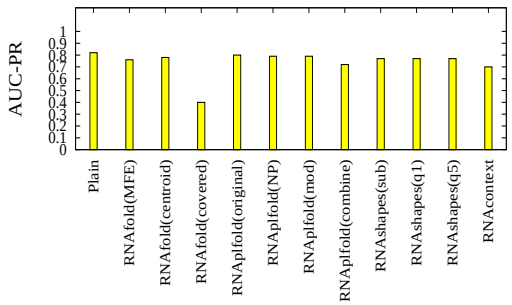
vts1p



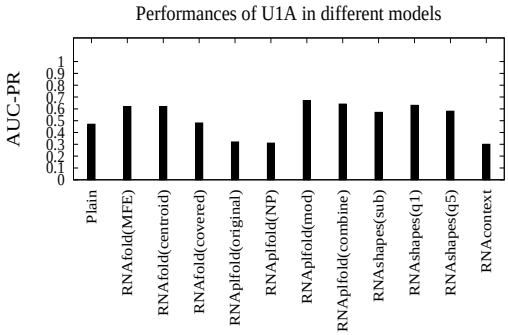
YB1



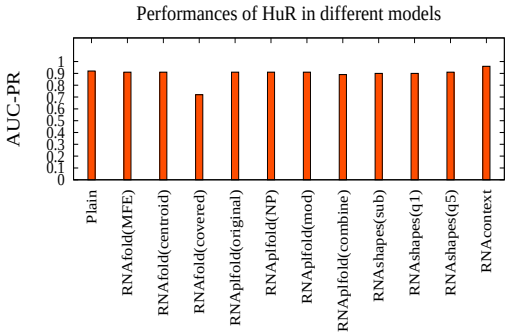
SLM2



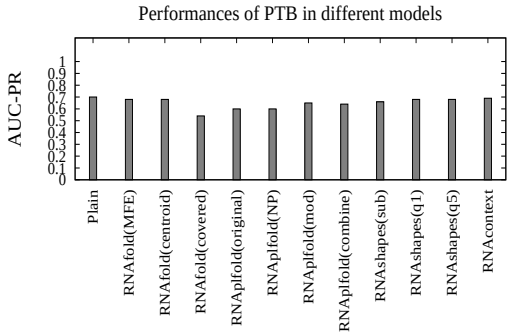
SF2


Performances of U1A in different models
AUC-PR
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Plain
RNAfold(MFE)
RNAfold(centroid)
RNAfold(covered)
RNAplfold(original)
RNAplfold(NP)
RNAplfold(mod)
RNAplfold(combine)
RNAshapes(sub)
RNAshapes(q1)
RNAshapes(q5)
RNAcontext

U1A


Performances of HuR in different models
AUC-PR
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Plain
RNAfold(MFE)
RNAfold(centroid)
RNAfold(covered)
RNAplfold(original)
RNAplfold(NP)
RNAplfold(mod)
RNAplfold(combine)
RNAshapes(sub)
RNAshapes(q1)
RNAshapes(q5)
RNAcontext

HuR


Performances of PTB in different models
AUC-PR
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Plain
RNAfold(MFE)
RNAfold(centroid)
RNAfold(covered)
RNAplfold(original)
RNAplfold(NP)
RNAplfold(mod)
RNAplfold(combine)
RNAshapes(sub)
RNAshapes(q1)
RNAshapes(q5)
RNAcontext

PTB

Chapter 6

Conclusion

Since RBPs play an important role in the post-transcriptional regulation of living organisms, the development of methods for the prediction of RBPs' binding preferences is important for us to understand the binding mechanisms. In this thesis, we introduced a brand new approach for the prediction of RBPs' binding preferences — graph kernel method. Different to the existing motif finding methods like MEMERIS and RNAcontext, which use probabilistic idea, our method is based on the machine learning method like kernel method and support vector machine. During the process of our experiment, we designed 11 graph models with each containing different attributes. And the experiment result shows that by using the graph model without structural context in our method, the AUC-PR values of five RBPs are better than the corresponding performances in RNAcontext, while by using the graph models with structural context such like RNAfold_mfe model, four RBPs' performances are better than in RNAcontext. So we can say that, for the five RBPs FUSIP1, SF2, U1A, YB1 and PTB, our method is the state-of-the-art method for the prediction of binding preferences. By comparing the performances between the models with structural context and model without

structural context, we could see that, not every RBP's AUC-PR increases under the structural context. And for the RBPs whose AUC-PR increase obviously under the structural context, our experiment result is consistent with the experiment result of RNAcontext, which means, for the following five RBPs: FUSIP1, RBM4, U1A, YB1, vts1p, the structural context in the binding sites has positive influence to the binding, but for the other four RBPs, the structural context in the binding sites seems to be not so helpful for the binding. From the experiment result, we could also conclude that the unpaired probability for each nucleotide in the RNA sequence has no positive influence to the binding, but the structural context of each nucleotide in the sequence will influence the binding positively. And the shape probability of the secondary structures is not a decisive factor in the binding. Our experiment result also proved the experimentally detected phenomenon, that the sequestration of a binding motif in a double-stranded region will cause strong negative influence to the binding.

References

[1] Hilal Kazan, Debashish Ray, Esther T. Chan, Timothy R.Hughes, Quaid Morris (2010) RNAcontext: a new method for learning the sequence and structure binding preferences of RNA-binding proteins. *Computational biology.*

[2] Ray D, Kazan H, Chan ET, Castillo LP, Chaudhry S, et al. (2009) Rapid and systematic analysis of the RNA recognition specificities of RNA-binding proteins. *Nature biotechnology* 27(7) 667-670.

[3] Keene JD, Komisarow JM, Friedersdorf MB (2006) RIP-Chip: the isolation and identification of mRNAs, microRNAs and protein components of ribonucleoprotein complexes from cell extracts. Nat Protoc 1: 302-307.

[4] Ule J, Jensen K, Mele A, Darnell RB (2005) CLIP: a method for identifying protein-RNA interaction sites in living cells. Methods 37: 376-386.

[5] Michael Hiller, Rainer Pudimat, Anke Busch, Rolf Backofen (2006) Using RNA secondary structures to guide sequence motif finding towards single-stranded regions. *Nucleic Acids Research.*

[6] Ye Ding, Charles E. Lawrence (2003) A statistical sampling algorithm for RNA secondary structure prediction. *Nucleic Acids Research.*

[7] O. Ivanciuc, Applications of Support Vector Machines in Chemistry, Rev. Comput. Chem. 2007, 23, 291-400.

[8] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin (2010) A practical guide to support vector classification.

[9] Bernhard Schlkopf and Alexander J. Smola (2001) Learning with Kernels. *The MIT Press.*

[10] John Shawe-Taylor and Nello Cristianini (2004) Kernel Methods for Pattern Analysis. *Cambridge University Press.*

[11] Fabrizio Costa, Kurt De Grave (2010) Fast neighborhood subgraph pairwise distance kernel. *Appearing in Proceeding of the* 27^{th} *International Conference on machine learning, Haifa, Israel.*

[12] Xifeng Yan and Jiawei Han (2002) gSpan: Graph-Based Substructure Pattern Mining. *Proc. 2002 International Conference on Data Mining (ICDM'02), Maebashi, Japan.*

[13] Thorsten Joachims, SVM-Light Support Vector Machine. http://svmlight.joachims.org/

[14] Jesse Davis, Mark Goadrich (2006) The relationship between Precision-Recall and ROC curves. *Appearing in Proceeding of the* 23^{th} *International Conference on machine learning, Pittsburgh, PA.*

[15] Ling Liu, Özsu, M. Tamer (2009) Encyclopedia of Database Systems. *Springer-Verlag.*

[16] http://www.g2l.bio.uni-goettingen.de/blast/fastades.html

[17] Ramlan, E. I. and Zauner, K. P. (2008) An Extended Dot-Bracket-Notation for Functional Nucleic Acids. In: International Workshop on Computing with Biomolecules, Wien, Austria, August 27, 2008, pp. 75-86, Österreichische Computer Gesellschaft. ISBN 978-3-85403-244-1

[18] Vienna RNA Package. http://www.tbi.univie.ac.at/∼ivo/RNA/

[19] Ye Ding, Chi Yu Chan, Charles E. Lawrence (2011) RNA secondary structure prediction by centroids in a Boltzmann weighted ensemble. *Published by Cold Spring Harbor Laboratory Press.*

[20] Peter Steffen, Robert Giegerich, Björn Voß, Marc Rehmsmeier, Jens, Reeder (2008) RNAshapes 2.1.5 manual.

[21] Robert Giegerich, Björn Voß, Marc Rehmsmeier (2004) Abstract shapes of RNA. *Nucleic Acids Research.*