DEGREE PROJECT, IN SCIENTIFIC COMPUTING , SECOND LEVEL
STOCKHOLM, *SWEDEN 2015*

Epitope Mapping using Local Alignment Features

YUNYI ZHU

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCI SCHOOL OF ENGINEERING SCIENCES

Epitope Mapping using Local Alignment Features

Z H U Y U N Y I

Master's Thesis in Scientific Computing (30 ECTS credits)
Master's Programme in Computer simulation for Science and Engineering
Royal Institute of Technology year 2015
Supervisor at KTH, Science for Life Lab: Lukas Käll
Supervisor at KTH: Christopher Peters
Examiner: Michael Hanke

TRITA-MAT-E 2015:51
ISRN-KTH/MAT/E--15/51--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Abstract

Our immune system uses antibodies to neutralize pathogens such as bacteria and viruses. Antibodies bind to parts of foreign proteins with high efficiency and specificity. We call such parts epitopes. The identification of epitopes, namely epitope mapping, may contribute to various immunological applications such as vaccine design, antibody production and immunological diagnosis. Therefore, a fast and reliable method that can predict epitopes from the whole proteome is highly desirable.

In this work we have developed a computational method that predicts epitopes based on sequence information. We focus on using local alignment to extract features from peptides and classifying them using Support Vector Machine. We also propose two approaches to optimize the features. The results show that our method can reliably predict epitopes and significantly outperforms some most commonly used tools.

Contents

Chapter 1

Introduction

The immune system protects organisms from pathogens (e.g., viruses, bacteria and other things that cause diseases to the hosts). One of the major subdivisions of the immune system is the adaptive immune system . It launches highly specific and long-lasting defences against invading pathogens. B cells, or B lymphocytes, play a crucial role in the adaptive immune system. They can secrete antibodies which neutralize pathogens.

Any substances that trigger the adaptive immune response are referred as antigens. This work only concerns antigens that are proteins. Antibodies neutralize a pathogen by binding to the antigens on its surface, and the part of antigens that antibodies bind to are called B-cell epitopes (Figure 1.1), or epitopes for short in this report.

An antigen usually contains multiple epitopes, and they may overlap with one another. An antibody is not specific to an antigen as a whole, rather, it often recognizes a specific epitope. The specificity of antibody-epitope interaction is the key to distinguishing foreign substance from self substance. It can also be used to develop sensitive diagnostic tests. Besides interacting with antibodies, epitopes also elicit them. An antibody elicited by a certain epitope can bind to any antigens that contain the same or similar epitopes, so it is assumed that immunization with the right epitope provides protection against certain pathogens.

Therefore, the identification of epitopes, namely epitope mapping, is highly desirable for various immunological applications. A number of computational tools have been developed to predict epitopes from antigens. Some of the predictors only rely on sequence information of antigens [1, 6, 9, 10, 22, 26, 29, 34, 40], others require structure information [24, 28, 31, 41]. Sequence-based prediction is more appealing, since the number of sequenced pathogen proteomes is increasing rapidly, while the structure data for antigens is limited. The application of machine learning techniques is a promising direction in sequence-based epitope prediction [42].

The aim of this work is to develop a reliable epitope predictor that only requires sequence information. This predictor should be able to distinguish epitopes from non-epitopes and detect the epitopes in given antigens. We focus on extracting new

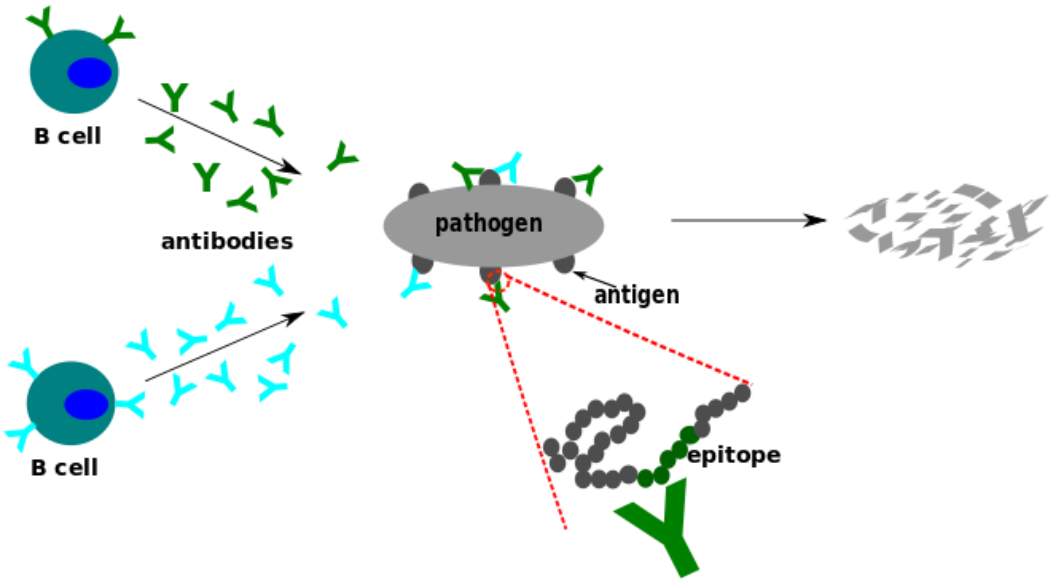


Figure 1.1: The neutralization of a pathogen by antibodies. Antibodies bind to antigens on the surface of a pathogen, and this process leads to the destruction of the pathogen. Epitopes are the part of antigens that antibodies bind to.

features from peptides and classifying them using Support Vector Machine (SVM).

The rest of this thesis is organized as follows.

Chapter 2 provides the background of our work. The biological background and existing methods for epitope mapping are briefly reviewed. A description of SVM is also given.

Chapter 3 describes the dataset used in our work.

Chapter 4 presents our method. We introduced new peptide features and combined them with SVM. In addition, we proposed two approaches to optimize the features.

Chapter 5 shows the results. We evaluated the ability of our models to classify epitopes and non-epitopes as well as detect epitopes from antigens. Then we compared our method with some frequently used tools.

Chapter 6 gives the conclusions of our work and discusses the current issues in developing computational tools for epitope mapping.

Chapter 2

Background

2.1 Biological background

There are two types of epitopes, continuous (Figure 2.1a) and discontinuous (Figure 2.1b). Continuous epitopes are also called linear epitopes, in which the residues are adjacent in a peptide chain. Discontinuous epitopes are also called conformational epitopes. A discontinuous epitope consists of several peptide segments that are separated in sequence but brought close in space as the antigen is folded. The vast majority of known B cell epitopes are discontinuous [33].

Several methods have been designed to map epitopes experimentally. The most accurate approach is X-ray co-crystallography [13] of antigen-antibody complexes which directly visualizes the interaction between the antigen and the antibody. However, purifying such complexes is technically challenging, time-consuming and expensive. To date, the number of solved structures is very small compared to the number of known antigens.

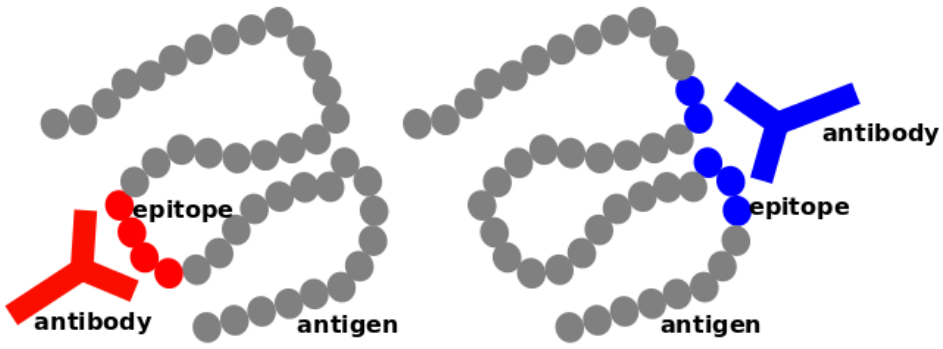


Figure 2.1: Continuous and discontinuous epitopes. (a) A continuous epitope, in which the residues are adjacent. (b) A discontinuous epitope, which consists of several peptide segments that are close in space when an antigen is folded.

Mutation-based techniques, such as site-directed mutugenesis [32], are used to identify residues comprising epitopes. When mutating an epitope residue or a group of epitope residues, a significant loss in antibody binding can be observed. Both continuous and discontinuous epitopes can be mapped this way. The limitation of mutation-based methods is that they are slow and labor-intensive.

Peptide microarray [4] is a high-throughput and relatively inexpensive alternative. In this method, overlapping peptides are extracted from antigens. Then peptides are selected based on whether they can bind to antibodies. These peptides are identified as (part of) epitopes. Discontinuous epitopes can also be mapped with high reliability by combining non-adjacent peptides and maintaining the conformation of the combination [38].

Despite of all the effort, experimental methods are limited in terms of time and expense. The mapping of epitopes cannot catch up with the rapid increase of sequenced pathogen proteomes. In addition, experimental methods lack the ability to explain what makes a good epitope. Therefore, developing different computational methods for epitope mapping is highly desirable.

2.2 Computational methods for epitope mapping

Compared with experimental methods, computational methods are efficient and cheap. Current methods can be divided to two categories. Sequence-based methods only use protein sequence data, while structure-based methods can use both sequence and structure data.

2.2.1 Sequence-based methods

Sequence-based methods are used for prediction of continuous epitopes. They are the majority of all prediction methods. Although not able to detect discontinuous epitopes, sequence-based methods give us the information that can be aggregated for better understanding of discontinuous epitopes, since some segments of discontinuous epitopes can independently bind to an antibody and thus be regarded as continuous epitopes. In addition, sequence-based methods have certain advantages over structure-based methods:

- They do not need structural data, which is not currently produced at the same rate as sequence data;
- the implementation of sequence-based methods is usually simpler and the computation is faster.

Propensity scale methods were among the first attempts. They measure the tendency of a residue to be part of an epitope by its propensity score. Propensity scores are based on physcio-chemical properties of amino acids like hydrophilicity, solvent accessibility *etc.*. New propensity scales can be produced by combining different old scales [1]. Blythe *et al.* [3] have exhaustively tested 484 amino acid

propensity scales on their ability to correctly predict epitopes. Their study showed that even the best propensity scale performed only slightly better than random guess.

Improved methods have been developed by using machine learning techniques. For example, Chen *et al.* [6] combined Support Vector Machine (SVM) with amino acid pair (AAP) propensity scales, and BepiPred [26] combines the hidden Markov model (HMM) with propensity scales. Both methods were reported to perform significantly better than random as well as propensity scale methods. ABCPred [34] uses recurrent neural network to predict epitopes of fixed length. BayesB [40] unilized Bayes Feature Extraction and built an SVM classifier to discriminate epitopes from non-epitopes. Instead of focusing on peptide features, El-Manzalawy *et al.* [9] implemented four types of string kernels and built SVM classifiers based on them.

2.2.2 Structure-based methods

Structure-based methods have been developed to predict both continuous and discontinuous epitopes. The integration of structural information overcomes the disadvantages of sequence based methods, as sequentially separated but spatially close residues can be brought together into consideration.

Structure-based methods often incorporate surfaces accessibility of residues [24]. The idea is that epitopes are usually on the surface of a protein structure. Some methods also look for secondary-structures which are favored in epitopes, such as turns and coils [37]. In addition, general-purpose binding site prediction programs can be used [31], since epitopes are the binding sites of antigens and antibodies.

The performance of structure-based methods is to some extend held back by the limited amount of antigen structure information. Yao *et al.* [41] evaluated the performance of several structure-based methods on an independent testing dataset. The result was far from satisfaction. Ponomarenko *et al.* [31] tested two conformational epitope predictors and four general-purpose protein-protein binding site predictors. The best performance was given by ClusPro [8], a binding site predictor.

2.2.3 Ensemble methods

Ensemble methods combine the predictions of several methods and often outperform an individual predictor. The combination of prediction results can be made by ensemble learning, such as majority voting and boosting. For example, EPmeta [28] is an ensemble predictor based on the votes from six discontinuous epitope predictors.

2.2.4 Online tools

Most of the methods mentioned above have software packages or online servers that users can use to obtain predicted epitopes from antigens of interest. Besides,

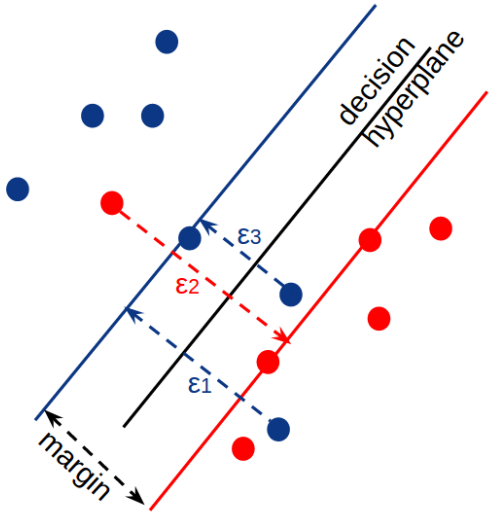


Figure 2.2: An illustration of linear SVM. The red and blue lines are the borders to separate the red and blue dots. $\varepsilon_i = \max\{0, 1 - t_i y_i\}$ is the distance of a data point at the wrong side of a corresponding border to the border

`http://www.cbs.dtu.dk/services/` and `http://www.iedb.org/` provide entries to a series of tools.

2.3 Support Vector Machine

Support Vector Machine (SVM) is a widely used kernel method for supervised classification. It works by finding the hyperplane that separates training data from different classes with a maximum margin (Figure 2.2). In SVM, we need to minimize the following cost function with respect to a set of parameters $\boldsymbol{\beta}$ [5]

$$\mathcal{L}(\boldsymbol{\beta}) = R(\boldsymbol{\beta}) + C \sum_{i=1}^{N} \max\{0, 1 - t_i y_i\}^p, \tag{2.1}$$

where t_i (1 or -1) is the label of the i^{th} data point $\boldsymbol{x}_i$ and $y_i = f(\boldsymbol{x_i})$ is the output of decision function which depends on $\boldsymbol{\beta}$. The output of decision function shows which side of the hyperplane a data point falls to and the distance between the data point to the hyperplane. $\sum_{i=1}^{N} \max\{0, 1 - t_i y_i\}^p$ is the error penalty, where p is either 1 (hinge loss) or 2 (quadratic loss). More specifically, $\max\{0, 1 - t_i y_i\}$ is the distance of a data point at the wrong side of a corresponding border to the border (Figure 2.2). $R(\boldsymbol{\beta})$ is the regularization which is used to generate a "smooth" separation hyperplane and avoid over-fitting. The penalty parameter C controls the trade-off between the two terms.

For a linear classifier, $\boldsymbol{\beta} = (\boldsymbol{w}, b)$ and the decision function is

$$f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} + b. \tag{2.2}$$

And the regularization can be *L1*

$$R(\boldsymbol{\beta}) = \sum |\beta|, \tag{2.3}$$

or *L2*

$$R(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}. \tag{2.4}$$

For non-linear classification, we usually use a kernel function to measure the similarity between two data points. A kernel function $k(.,.)$ takes two arguments and returns a real number. In this case, the decision function is written as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \beta_i k(\boldsymbol{x}_i, \boldsymbol{x}), \tag{2.5}$$

L2 regularization is

$$R(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i,j=1}^{N} \beta_i\beta_j k(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{2.6}$$

Thus Equation 2.1 becomes

$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i,j=1}^{N} \beta_i\beta_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + C\sum_{i=1}^{N} \max\{0, 1 - t_i \sum_{j=1}^{N} \beta_j k(\boldsymbol{x}_j, \boldsymbol{x}_i)\}^p. \tag{2.7}$$

Introducing the *Gram* matrix K where $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{k}_i$ the i^{th} column of K, we can rewrite the cost function in matrix form

$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^T K\boldsymbol{\beta} + C\sum_{i=1}^{N} \max\{0, 1 - t_i\boldsymbol{k}_i^T\boldsymbol{\beta}\}^p. \tag{2.8}$$

Since the Gram matrix K is positive semi-definite, the cost function (2.8) is convex and the minimization problem can be uniquely solved. Standard convex optimizers can be applied directly to obtained the optimal values of $\boldsymbol{\beta}$.

A lot of kernel functions have been developed. Conventional kernels include RBF (Gaussian) kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = exp(-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|^2), \tag{2.9}$$

linear

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T\boldsymbol{x}' + b, \tag{2.10}$$

polynomial

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + b)^p, \tag{2.11}$$

and sigmoid

$$k(\boldsymbol{x}, \boldsymbol{x}') = tanh(\gamma\boldsymbol{x}^T\boldsymbol{x}' + b). \tag{2.12}$$

Parameters in a kernel function such as γ and b are called *hyperparameters*. New kernel functions can be constructed from old ones through linear combination, multiplication, convolution *etc.* [2]. Several kernels have been developed to operate directly on strings [9].

We must decide which kernel function to use first. Generally, RBF kernel is a first choice [17]. First of all, both linear and sigmoid kernel have the same or similar performance as RBF for certain parameters. Secondly, unlike polynomial and sigmoid kernel, there is only one hyperparameter for RBF, which reduces the complexity in model selection. Finally, the output of RBF kernel is bounded by 0 and 1, so the minimization of (2.8) can be easily solved by numerical methods. However, if the dimension of data is very high, we may just use linear kernel.

The choice of the kernel function, the penalty parameter and hyperparameters should be justified by cross-validation, in which a portion of the training data is used to build the model and the rest is used for testing.

Chapter 3

Dataset

We have assembled a large set of well characterized data to train and test our model. Our data was generated by peptide microarray (Figure 3.1). The dataset contains 945 unique antigens. Overlapping peptides were taken by sliding a window along each antigen. The window sizes were between 9 to 12 residues. For each peptide, its affinity with an antibody was measured. The score for affinity ranges from 0 to 65,535. Peptides from epitopes can bind to antibodies, so they are associated with high affinity scores.

We filtered out antigens whose highest scores are lower than 4,000. For those antigens, no epitope was considered identified. 855 antigens remained after the filtering. Then we rescaled the scores of peptides to 0-1:

$$\theta(\boldsymbol{\chi}) = \frac{\theta(\boldsymbol{\chi}) - min}{max - min}, \tag{3.1}$$

where $\theta(\boldsymbol{\chi})$ is the score for peptide $\boldsymbol{\chi}$, max and min are the maximum and minimum among the scores of peptides from the same antigen as $\boldsymbol{\chi}$.

Epitopes were defined as regions in which the score of every peptide is higher than 0.2. The length distribution of the epitopes obtained is shown in Figure 3.2. Regions in antigens that are not covered by epitopes were extracted to form the negative set. Since the length distribution of non-epitopes was significantly different from epitopes, we discarded non-epitopes shorter than 10 residues and evenly split those longer than 20 residues into segments shorter than 20. Eventually we obtained 2,068 epitopes and 4,072 non-epitopes.

The antigens were evenly separated into five groups using the online server CD-hit [18], so that any two antigens from different groups share less than 30% sequence similarity. A model will be trained on epitopes and non-epitopes from four groups and tested on the remaining group.

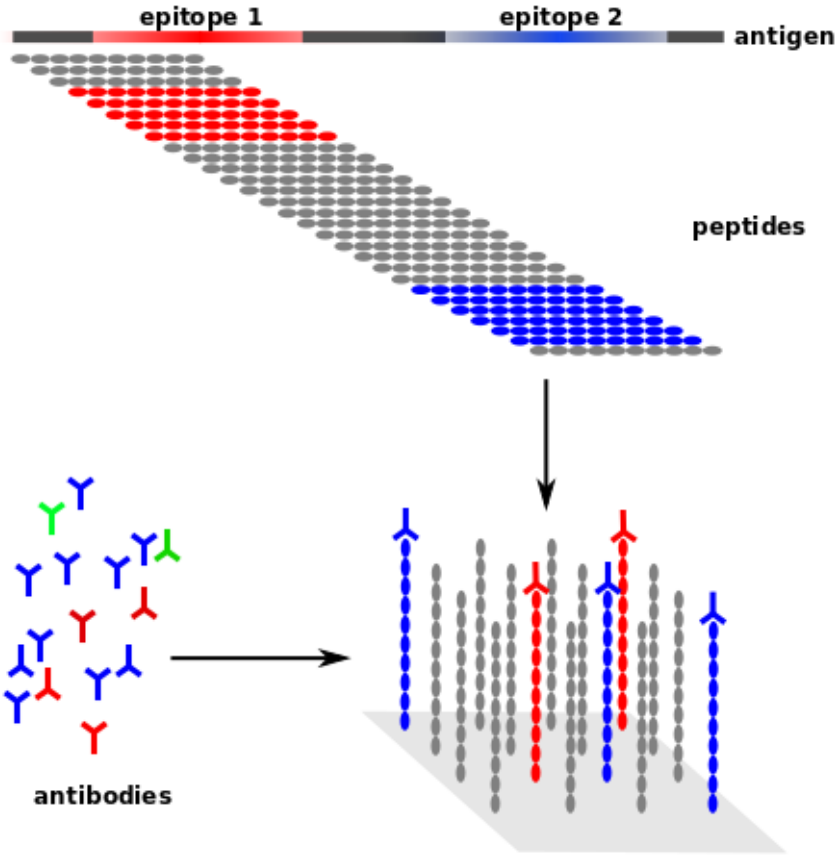


Figure 3.1: Overview of data production. Overlapping peptides are extracted from antigens and their affinities with antibodies are measured. If a peptide has a high affinity with an antibody, then we consider it as part of an epitope.

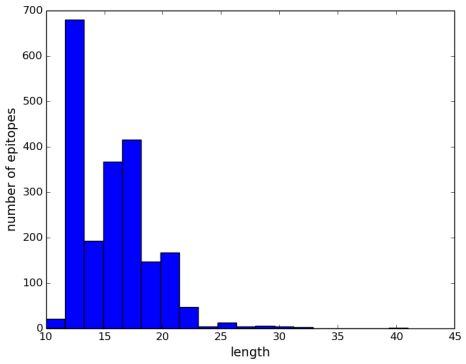


Figure 3.2: Epitope length distribution.

Chapter 4

Methodology

The basic function of our model is to predict if a peptide is an epitope. Modifications can be made so that it will also detect epitopes from an antigen (see § 4.3.2). Known epitope and non-epitope sequences are used for training. We will first map training sequences to a feature space, and then it is straightforward to use an SVM platform to train the model.

4.1 Peptide feature extraction

In this work, we will use a library of short amino acid sequences to characterize a peptide (Figure 4.1).

Let $\boldsymbol{\chi}$ be a peptide, $\boldsymbol{x}$ be its feature vector and $A = \{\boldsymbol{\alpha}_m | m = 1, ..., M\}$ be the library of short amino acid sequences. Then the m^{th} element of $\boldsymbol{x}$ is obtained by:

$$x_m = \Phi(\boldsymbol{\chi}, \boldsymbol{\alpha}_m), \tag{4.1}$$

where Φ is the scoring function.

An example of A is the set of all k-mers, *i.e.* all possible amino acid sequences of length k. Since there are 20 amino acids, the size of A is 20^k.

There are many different ways to define the scoring function Φ. Here we will employ two types of scores.

4.1.1 Bag-of-Words

Bag-of-Words (BoW) model [35] is widely used in image and text classification. In this model, $\Phi(\boldsymbol{\chi}, \boldsymbol{\alpha})$ is the occurrence of $\boldsymbol{\alpha}$ in $\boldsymbol{\chi}$. For example $\Phi('ACACCC', 'AC') = 2$. To rule out the influence of peptide length, the feature vector $\boldsymbol{x}$ generated as in (4.1) is normalized by its maximal element.

4.1.2 Local alignment

BoW model, however, does not capture some important features in peptides. Certain degree of insertion, deletion and substitution of residues may not alter the

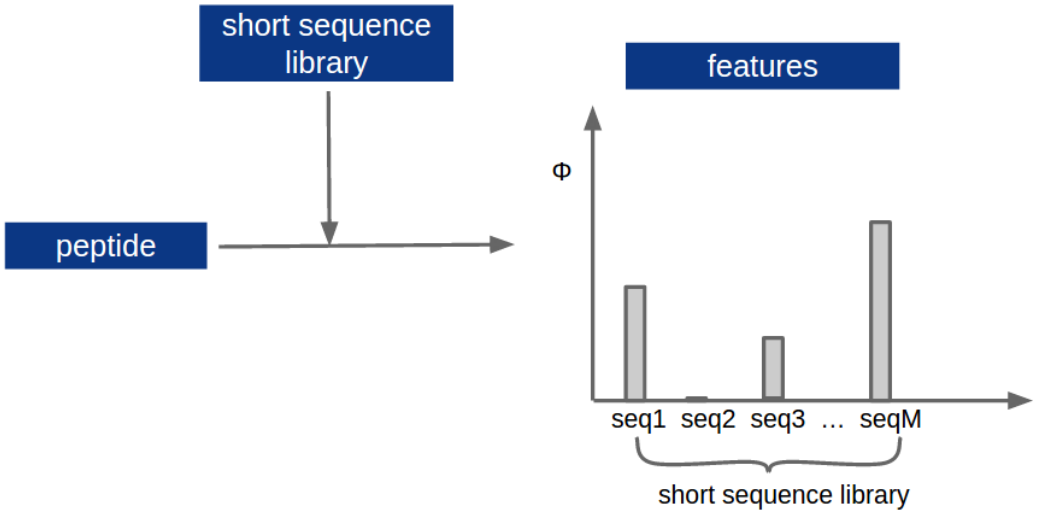


Figure 4.1: Feature extraction. A library of short amino acid sequences is defined. Features of a peptide are extracted by computing the score between the peptide and each short sequence by the scoring function Φ.

function of a protein/peptide. So even if a short sequence does not occur in a peptide, it may still be important for the characterization of the peptide if a similar subsequence can be found.

Thus, we have developed a new feature extraction method that uses local alignment to score a peptide in terms of a short sequence. Local alignment is a way to identify similar regions of two sequences and measure the similarity. Identical or similar residues are aligned and gaps may be inserted. For example, a good alignment between sequence 'GAIGHE' and 'TALH' is

$$\begin{array}{c} A\ I\ G\ H \\ A\ L\ -\ H, \end{array}$$

I (Isoleucine) and L (Leucine) are aligned because they have similar physio-chemical properties. '–' stands for a gap.

The score of an alignment is obtained by summing up the scores of aligned residue pairs and gaps in the alignment. Identical and similar residue pairs have positive scores while dissimilar ones have 0 or negative scores. We can put the scores of all amino acid pairs in a matrix which is called *substitution matrix*. A substitution matrix is 20×20 and symmetric. The score for gaps, usually referred as gap penalty, is always negative. The simplest gap penalty is proportional to the number of consecutive gaps

$$s(g) = -gd, \tag{4.2}$$

where g is the number of gaps and $-d$ is the penalty for a single gap. There are more sophisticated types of gap penalties, but the computation of alignments with those gap penalties is more time consuming than with (4.2). In addition, the short sequences used in our work is shorter than 4. When aligning them to a peptide, the number of gaps is usually small (0-2). In this case different types of gap penalties generate similar result. Therefore, only (4.2) is used in this work.

After choosing the proper substitution matrix and gap penalty, it's easy to calculate the score of a given alignment. Let the substitution score of a residue pair (a, b) be $s(a, b)$, then the total score for the previous alignment between 'GAIGHE' and 'TALH' is $s(A, A) + s(I, L) - d + s(H, H)$.

In most cases we are interested in the best local alignment, that is, the max-pooling on all possible local alignments. Back to peptide feature extraction, $\Phi(\boldsymbol{\chi}, \boldsymbol{\alpha})$ is now the best local alignment score between peptide $\boldsymbol{\chi}$ and short sequence $\boldsymbol{\alpha}$. The score is computed by Smith-Waterman algorithm (see below). Since the local alignment score for totally unrelated sequences is probably larger than 0 (for example if they have one residue in common), it is sensible to put a threshold T on the alignment score, so that every score below T is 0. Similar to BoW, the feature vector $\boldsymbol{x}$ should be normalized.

Smith-Waterman algorithm

Simith-Waterman algorithm is a frequently used dynamic programming algorithm guaranteed to find the best local alignment. It was first proposed by Smith *et al.* [36] and improved by Gotoh [14].

The idea of the algorithm is to build up an alignment by extending the optimal alignment of smaller sequences. Suppose we are interested in the best local alignment between sequence $\boldsymbol{x}$ of length m and sequence $\boldsymbol{y}$ of length n, then the best alignment up to the i^{th} residue of $\boldsymbol{x}$ and j^{th} residue of $\boldsymbol{y}$ is chosen from four possibilities:

- extending the best alignment up to x_{i-1} and y_{j-1} by aligning x_i and y_j
- extending the best alignment up to x_i and y_{j-1} by aligning y_j to a gap
- extending the best alignment up to x_{i-1} and y_j by aligning x_i to a gap
- starting a new alignment

To facilitate the building up of the best local alignment, we construct a $(1 + m) \times (1 + n)$ matrix H. The row index of H goes from 0 to m and the column index from 0 to n. $H_{i,j}$ is calculated as best alignment score between $\boldsymbol{x}_{1:i}$ and $\boldsymbol{y}_{1:j}$. We begin by initializing the first row and first column of H to be 0 and then calculate

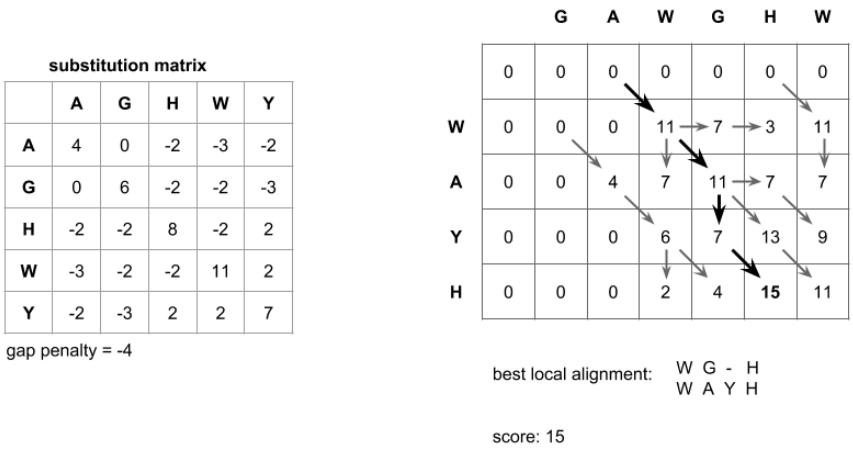


Figure 4.2: A demonstration of Smith-Waterman algorithm. Left, the substitution matrix and gap penalty used in the demonstration. Right, using Smith-Waterman algorithm to find the best local alignment between sequences 'WAYH' and 'GAWGHW'.

the other elements recursively from top left to bottom right.

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + s(x_i, y_j), \\ H_{i-1,j} - d, \\ H_{i,j-1} - d, \\ 0. \end{cases} \tag{4.3}$$

The largest value in H is highest local alignment score, and the best alignment can be obtained by tracing back. The traceback ends when we meet an element with value 0, which indicates the start of an alignment. The above process is demonstrated in Figure 4.2.

Alignment parameters

Entries in substitution matrix and gap penalty are henceforth referred as alignment parameters. They not only determine the score of an alignment, but also play a crucial role in obtaining the best local alignment between two sequences. There are a variety existing substitution matrices, for example the Blosum [16] matrices. However, there is no guarantee that those matrices are suitable in the context of epitope prediction. To address this problem, we developed a gradient descent approach to optimize the parameters.

Normally parameter optimization requires a cost function, and our task is to find the parameters that minimize the function given the training data. Continuously differentiable cost functions are favored in numerical minimization, therefore we will use the SVM cost function (2.1) with *L2* regularization and quadratic loss. For the same reason, the threshold for the alignment score should be 0. For simplicity, we further assume that the classifier is linear. Let $\{(\boldsymbol{\chi}_i, t_i)|i = 1, ..., N\}$ be the peptides and labels of the training data, $\{\boldsymbol{x}_i|n = i, ..., N\}$ be their feature vectors and $\{\boldsymbol{\alpha}_m|m = 1, ..., M\}$ be the set of short amino acid sequences that the peptides are aligned to, then the cost function is

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{w}, b) = \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{w} + b^2) + C\sum_{i=1}^{N}\varepsilon_i^2, \tag{4.4}$$

where

$$\varepsilon_i = \max\{0, 1 - t_i(\boldsymbol{x}_i^T\boldsymbol{w} + b)\}. \tag{4.5}$$

$\boldsymbol{w}$, b, and $\boldsymbol{\theta}$ are the parameters we want to optimize while C is fixed. $\boldsymbol{\theta}$ consists of the scores of all residue pairs (gap penalty can be regarded as the score of any residue paired with a gap). Remember that the score of an alignment is obtained by adding up the scores of residue pairs occurring along it, then the m^{th} element of $\boldsymbol{x}_i$ can be calculated as

$$x_{i,m} = \frac{\sum_{p=1}^{P} O_{i,m}^p\theta_p}{x_i^{max}}, \tag{4.6}$$

in which $O_{i,m}^p$ is the occurrence of the p^{th} residue pair in the best local alignment between $\boldsymbol{\chi}_i$ and $\boldsymbol{\alpha}_m$, and x_i^{max} is the maximum element of $\boldsymbol{x}_i$ before normalization. $O_{i,m}^p$ depends on $\boldsymbol{\theta}$ and is obtained by dynamic programming.

If $\boldsymbol{\theta}$ is fixed, it is easy to optimize $(\boldsymbol{w}, b)$ using existing linear SVM learners such as LIBLINEAR [11]. Thus we will optimize $(\boldsymbol{w}, b)$ and $\boldsymbol{\theta}$ alternately. When fixing $(\boldsymbol{w}, b)$, we take a gradient descent approach to optimize $\boldsymbol{\theta}$.

To perform gradient descent, we need to compute the partial derivatives of the cost function in terms of all the parameters we want to optimize. Unfortunately, there is no explicit form of $O_{i,m}^p$, which means we cannot explicitly express $\mathcal{L}$ as well as its partial derivatives with respect to $\boldsymbol{\theta}$. One way to solve the problem is to approximate derivative by difference. But this is computationally expensive. For example, in order to get the partial derivative of P parameters by central difference, we need to compute the cost function at least $2P$ times, which includes searching for best local alignments for $2PNM$ times. To simplify the problem, we assume that the paths of alignments do not change when there is a small perturbation in $\boldsymbol{\theta}$, i.e. $O_{i,m}^p$ is considered constant when calculating the derivatives. This way, we only need to search for local alignments for NM times to get $\{O_{i,m}^p|i = 1, ..., N, m = 1, ..., M, p = 1, ..., P\}$. And the derivative becomes

$$\frac{\partial\mathcal{L}}{\partial\theta_p} = C\sum_{i=1}^{N}\frac{\partial\varepsilon_i^2}{\partial\theta_p}, \tag{4.7}$$

$$\frac{\partial \varepsilon_i^2}{\partial \theta_p} = -2t_i\varepsilon_i \frac{\partial \boldsymbol{x}_i^T}{\theta_p}\boldsymbol{w}, \tag{4.8}$$

$$\frac{\partial x_{i,m}}{\theta_p} = \frac{O_{i,m}^p - O_{i,\tilde{m}}^p x_{i,m}}{x_i^{max}}. \tag{4.9}$$

In (4.9), x_i^{max} is the largest element in vector $\boldsymbol{x}_i$ before normalization and $\tilde{m}$ is its index.

Algorithm 1 shows the whole process of minimizing (4.4). Here are some remarks on the minimization problem:

- We can fix some of the alignment parameters and optimize the others.
- The cost function is not convex and gradient descent can only find the local minimum. We have tested the optimization process with different initial values, and the parameters converge to different results.
- Conjugate Gradient (CG) algorithm can be implemented to achieve faster convergence. But for our problem, gradient descent is simple and efficient enough. In all tests we have carried out, the parameters converge quickly (within 16 steps).
- The parameters can also be optimized in nonlinear models. In this case, the cost function becomes:

  $$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\beta}) = \boldsymbol{\beta}^T K \boldsymbol{\beta} + C\sum_{i=1}^{N} \varepsilon_i^2, \tag{4.10}$$

  where

  $$\varepsilon_i = \max\{0, 1 - t_i \sum_{j=1}^{N} \beta_j k(\boldsymbol{x}_i, \boldsymbol{x}_j\}. \tag{4.11}$$

  The cost function can be minimized w.r.t $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ using the same idea as in the linear model.

4.2 Multiple kernel learning

If several sets of feature mapping functions and kernels perform well in epitope prediction, we can combine them into a new kernel function. Let $\{(\phi_l(.), k_l(.,.))|l = 1, ..., L\}$ be the set of feature mapping and kernel functions and $k(.,.)$ be the new kernel, then

$$k(\boldsymbol{\chi}, \boldsymbol{\chi}') = \sum_{l=1}^{L} \lambda_l k_l(\phi_l(\boldsymbol{\chi}), \phi_l(\boldsymbol{\chi}')). \tag{4.12}$$

Algorithm 1: Optimize alignment parameters

Input : $\{(\boldsymbol{\chi}_i, t_i)|i = 1, ..., N\}, \{\alpha_m|m = 1, ..., M\}, C$
1 . **Output**: $\boldsymbol{w}$,b, and $\boldsymbol{\theta}$
2 .
3 Assign initial value to $\boldsymbol{\theta}$.
4 Compute $\{\boldsymbol{x}_i|i = 1, ..., N\}$ and $\{O^p_{i,m}|i = 1, ..., N, m = 1, ..., M, p = 1, ..., P\}$.
5 **do**
6 | `/* Update` $\boldsymbol{w}$ `and` b`.` `*/`
7 | Optimize $\boldsymbol{w}$ and b by linear SVM learner.
8 | Compute $\mathcal{L}$.
9 | `/* Update` $\boldsymbol{\theta}$`.` `*/`
10 | **do**
11 | | Compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$
12 | | $\boldsymbol{\theta_{new}} = \boldsymbol{\theta} - \gamma \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$.
13 | | Update $\{\boldsymbol{x}_i\}$ and $\{O^p_{i,m}\}$.
14 | | Compute $\mathcal{L}_{new}$.
15 | | **if** $\mathcal{L}_{new} <= \mathcal{L}$ **then**
16 | | | $\boldsymbol{\theta} = \boldsymbol{\theta}_{new}$.
17 | | **else**
18 | | | Decrease γ, go back to step 10.
19 | | **end**
20 | **while** *not converge*;
21 **while** *not converge*;

Weights $\lambda_1, ..., \lambda_L$ show the contribution of corresponding terms. Without loss of generality, we assume

$$\sum_{l=1}^{L} \lambda_l = 1, \qquad \lambda_l \geq 0, l = 1, ..., L. \tag{4.13}$$

Here we propose a supervised multiple kernel learning approach to determine $\lambda_1, ..., \lambda_L$. Given the training sequences and labels $\{(\boldsymbol{\chi}_i, t_i)|i = 1, ..., N\}$, we first compute the Gram matrices $K_1, ..., K_L$ so that $K_{l,(i,j)} = k_l(\phi_l(\boldsymbol{\chi}_i), \phi_l(\boldsymbol{\chi}_j))$. Let $\boldsymbol{k}_{l,i}$ be the i^{th} column of K_l, then we can obtain $\lambda_1, ..., \lambda_L$ by minimizing the following cost function w.r.t $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$:

$$\mathcal{L}(\boldsymbol{\beta}, \lambda_1, ..., \lambda_L) = \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{\beta} + C\sum_{i=1}^{N} \varepsilon_i^2, \tag{4.14}$$

$$\text{subject to} \sum_{l=1}^{L} \lambda_l = 1, \quad \lambda_l \geq 0, l = 1, ..., L,$$

where

$$\varepsilon_i = \max\{0, 1 - t_i \boldsymbol{k}_i^T \boldsymbol{\beta}\}$$

and

$$\boldsymbol{k}_i = \sum_{l=1}^{L} \lambda_l \boldsymbol{k}_{l,i}.$$

Again, we can optimize $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$ alternately. When optimizing $\boldsymbol{\lambda}$, the cost function is convex, bounded and constrained, minimizers such as SLSQP [23] can be used to solve the problem. But first we need to calculate the gradient:

$$\frac{\partial \mathcal{L}}{\partial \lambda_l} = -C \sum_{i=1}^{N} \varepsilon_i t_i \boldsymbol{k}_{l,i}^T \boldsymbol{\beta}. \tag{4.15}$$

4.3 Performance evaluation

An epitope predictor built in our work produces a prediction score for a peptide and the score indicates how likely it is (part of) an epitope. A model can be evaluated on many aspects. Evaluation on peptide level assesses the ability to correctly predict whether a peptide is an epitope. Evaluation on residue level looks at if the residues in epitopes are correctly distinguished from residues in non-epitopes. The end purpose of an epitope predictor is to detect the regions from an antigen that are likely to be epitopes, so the quality of the detections should also be assessed.

4.3.1 Evaluation on peptide level

Peptides which are either epitopes or non-epitopes are used to test a model. We need to set a threshold on the prediction scores, so that peptides whose scores are above the threshold are predicted as an epitope and otherwise is not. By comparing with the true labels, we can obtain the number of TP (true positive), FP (false positive), TN (true negative) and FN (false negative). Some statistics can be calculated afterwards:

- *Recall* or *true positive rate* (TPR): $\frac{TP}{TP+FN}$;
- *Precision*: $\frac{TP}{TP+FP}$;
- *False positive rate* (FPR): $\frac{FP}{TN+FP}$.

None of the statistics alone yields a comprehensive picture about the performance. It is always possible to increase TPR by decreasing the threshold, at the expense of increasing FPR and decreasing precision. Receiver operating characteristic (ROC) curve and precision-recall curve can evaluate the performance independent of the threshold. An ROC curve is obtained by varying the threshold and plotting TPR against FPR. The curve is monotonously increasing and TPR should

be larger than FPR. *Area under curve* (AUC) is a useful statistic to compare different ROC curves [12]. For a random classifier AUC=0.5 and for a perfect classifier AUC=1. A classifier performs better than random should have an AUC between 0.5 and 1. Another non-parametric evaluation metric is the precision-recall curve. It is obtained by plotting precision against recall. A random classifier results in a horizontal precision-recall curve with precision $\equiv \frac{\text{number of actual positive data}}{\text{number of all data}}$, while a perfect classifier has a horizontal precision-recall curve with precision $\equiv 1$. For a reasonable classifier, precision should tends to decrease when recall increases, although the curve may not be totally monotonous. Area under the precision-recall curve, *average precision*, is also a measurement of how good a classifier is.

4.3.2 Evaluation on epitope detection

In § 4.3.1, evaluation is made on the assumption that an antigen is already segmented into epitopes and non-epitopes, and we only need to predict the labels of the segments. But in epitope mapping, an antigen sequence is given as a whole. To detect epitopes in an antigen, we first extract overlapping peptides using a slide window. Again the model assigns precision scores to peptides and we need to put a threshold on the scores. Then we merge successive peptides whose scores are higher than the threshold to form detected epitopes. To facilitate further analysis, we calculated the score of a detected epitope as the average prediction score of all peptides within it.

Next, we need to distinguish true detections from false detections. It is rare that a detection covers exactly the same region as an actual epitope, but they may overlap. The extent of overlap is calculated as the length of overlap region divided by the maximum of the detected epitope length and the actual epitope length. If the overlap between a detection and any actual epitope is larger than a predefined value δ, then the detection is true, otherwise it is false.

We expect a good predictor to assign high scores to true detections and low scores to false detections. One way to examine this is to set a threshold on the scores of detections, and define

- TP: true predictions with scores higher than the threshold;
- FP: false predictions with scores higher than the threshold;
- FN: true predictions with scores lower than the threshold and undetected epitopes.

By adjusting the threshold, we can obtain the precision-recall curve as in § 4.3.1.

4.3.3 Evaluation on residue level

A lot of existing methods assign scores to individual residues and predict a residue with a high score to be part of an eptope. To make direct comparisons and avoid involving too many parameters as in § 4.3.2, we made the score of a peptide as the

score of its center residue. A residue at either ends of a antigen is not the center of any peptide, then its score is the same as the closest residue. Similarly, the label of a peptide becomes the label of the center residue. Then, ROC curves and precision-recall curved can be constructed as in § 4.3.1.

Chapter 5

Results

5.1 Experimental setting

A feature extraction class was written in python. Then, we used the python package *scikit-learn* [30] to build SVM classifiers. In particular, class *sklearn.svm.LinearSVC* was used for linear SVM and *sklearn.svm.SVC* for nonlinear SVM. Algorithm 1 was implemented in python. The minimization problem (4.14) for multiple kernel learning was solved using class *scipy.optimize.minimize* with the SLSQP algorithm [20]. Other python packages used in this work include *NumPy* [39] and *Matplotlib* [19].

5.2 Impact of feature extraction and kernel functions

In the first set of experiments, we built SVM models with different features and kernel functions.

Peptide features were extracted with respect to 1-, 2- and 3-mers. The feature extraction method was either BoW or local alignment. Two substitution matrices were used for local alignment. One was BLOSUM62 (Figure 5.1), the other was a modification of BLOSUM62 in which all diagonal elements were set to 11. The gap penalty was set to -6 in both cases. If features were generated with respect to 3-mers, the number of features would be $20^3 = 8000$, which is too large compared to the number of training data (about 4000). To reduce the number, we summed up each feature across the positive training sequences and only selected features whose sums were the largest.

Then the features were associated with linear or RBF kernel function.

We tuned other parameters using grid search. The threshold T for alignment score was searched over the range $2, 4, ..., 8k$ when k-mers were aligned to. If 3-mers were used, the number of features was searched over the range $500, 1000, 1500, ..., 4000$. The penalty parameter C was selected from $10^{-3}, 10^{-2}, ..., 10^3$. γ in the RBF kernel was searched over $10^{-2}, 10^{-2}, ..., 10^2$. We exhaustively tested all combinations of these parameters, and the best combinations were selected based on five-fold

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	**4**	0	-2	-1	-2	0	-2	-1	-1	-1	-1	-2	-1	-1	-1	1	0	0	-3	-2
C	0	**9**	-3	-4	-2	-3	-3	-1	-3	-1	-1	-3	-3	-3	-3	-1	-1	-1	-2	-2
D	-2	-3	**6**	2	-3	-1	-1	-3	-1	-4	-3	1	-1	0	-2	0	-1	-3	-4	-3
E	-1	-4	2	**5**	-3	-2	0	-3	1	-3	-2	0	-1	2	0	0	-1	-2	-3	-2
F	-2	-2	-3	-3	**6**	-3	-1	0	-3	0	0	-3	-4	-3	-3	-2	-2	-1	1	3
G	0	-3	-1	-2	-3	**6**	-2	-4	-2	-4	-3	0	-2	-2	-2	0	-2	-3	-2	-3
H	-2	-3	-1	0	-1	-2	**8**	-3	-1	-3	-2	1	-2	0	0	-1	-2	-3	-2	2
I	-1	-1	-3	-3	0	-4	-3	**4**	-3	2	1	-3	-3	-3	-3	-2	-1	3	-3	-1
K	-1	-3	-1	1	-3	-2	-1	-3	**5**	-2	-1	0	-1	1	2	0	-1	-2	-3	-2
L	-1	-1	-4	-3	0	-4	-3	2	-2	**4**	2	-3	-3	-2	-2	-2	-1	1	-2	-1
M	-1	-1	-3	-2	0	-3	-2	1	-1	2	**5**	-2	-2	0	-1	-1	-1	1	-1	-1
N	-2	-3	1	0	-3	0	1	-3	0	-3	-2	**6**	-2	0	0	1	0	-3	-4	-2
P	-1	-3	-1	-1	-4	-2	-2	-3	-1	-3	-2	-2	**7**	-1	-2	-1	-1	-2	-4	-3
Q	-1	-3	0	2	-3	-2	0	-3	1	-2	0	0	-1	**5**	1	0	-1	-2	-2	-1
R	-1	-3	-2	0	-3	-2	0	-3	2	-2	-1	0	-2	1	**5**	-1	-1	-3	-3	-2
S	1	-1	0	0	-2	0	-1	-2	0	-2	-1	1	-1	0	-1	**4**	1	-2	-3	-2
T	0	-1	-1	-1	-2	-2	-2	-1	-1	-1	-1	0	-1	-1	-1	1	**5**	0	-2	-2
V	0	-1	-3	-2	-1	-3	-3	3	-2	1	1	-3	-2	-2	-3	-2	0	**4**	-3	-1
W	-3	-2	-4	-3	1	-2	-2	-3	-3	-2	-1	-4	-4	-2	-3	-3	-2	-3	**11**	2
Y	-2	-2	-3	-2	3	-3	2	-1	-2	-1	-1	-2	-3	-1	-2	-2	-2	-1	2	**7**

Figure 5.1: Substitution matrix BLOSUM62.

cross-validation.

For each set of features and kernel function, the model was built five times, and in each time four groups of our data were used for training and one group for testing. Table 5.1 shows the peptide-level evaluation of models built with different sets of parameters. The models are sorted first by AUC and then by average precision. The results show that feature extraction by local alignment significantly outperforms BoW, and the modified BLOSUM62 matrix is better than the original BLOSUM62. In additionn, features extracted in terms of 2-mers and 3-mers often perform better than single amino acids (1-mers).

5.3 Optimization

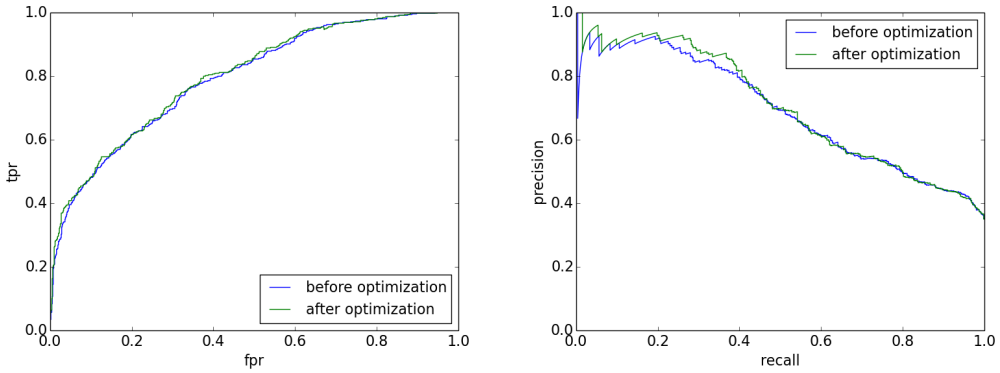
In this section, we carried out two optimization procedures to improve the settings in Table 5.1.

5.3.1 Multiple kernel learning

We combined the first 10 models in Table 5.1 using multiple kernel learning as proposed in § 4.2. The penalty parameter C was set to 1, the same as Model 1 in Table 5.1. We initialized the weights of Model 1 to Model 10 to $[1, 0, 0, ..., 0]$. After optimization, the weights of Model 1 and Model 2 became 0.684 and 0.316, while the

Model No.	k-mers	feature generation	substitution matrix	kernel	other parameters	average precision	AUC
Model 1	2	alignment	modified BLOSUM62	RBF	T=12 C=1 γ=0.1	0.68 ± 0.01	0.79 ± 0.01
Model 2	3	alignment	modified BLOSUM62	RBF	T=24 4000 features C=10 $\gamma = 0.1$	0.69 ± 0.01	0.77 ± 0.01
Model 3	2	alignment	modified BLOSUM62	linear	T=12 C=10	0.66 ± 0.02	0.76 ± 0.01
Model 4	3	alignment	modified BLOSUM62	linear	T=15 1500 features C=0.01	0.63 ± 0.02	0.75 ± 0.02
Model 5	1	alignment	BLOSUM62	RBF	T=0 C=1 γ=0.1	0.60 ± 0.03	0.75 ± 0.01
Model 6	1	alignment	modified BLOSUM62	linear	T=0 C=1	0.61 ± 0.02	0.74 ± 0.02
Model 7	2	alignment	BLOSUM62	RBF	T=0 C=1 γ=0.1	0.59 ± 0.02	0.74 ± 0.01
Model 8	1	alignment	modified BLOSUM62	RBF	T=0 C=0.1 γ=0.1	0.57 ± 0.04	0.74 ± 0.02
Model 9	1	alignment	BLOSUM62	linear	T=0 $C = 0.1$	0.58 ± 0.02	0.73 ± 0.01
Model 10	3	alignment	BLOSUM62	linear	T=6 4000 features C=0.01	0.58 ± 0.01	0.73 ± 0.01
Model 11	2	alignment	BLOSUM62	linear	T=0 C=1	0.58 ± 0.02	0.72 ± 0.01
Model 12	3	alignment	BLOSUM62	RBF	T=2 2000 features C=100 γ=0.1	0.55 ± 0.01	0.72 ± 0.01
Model 13	3	bow		RBF	1500 features C=0.01 γ=0.1	0.56 ± 0.01	0.69 ± 0.02
Model 14	3	bow		linear	4000 features C=0.01	0.55 ± 0.02	0.69 ± 0.01
Model 15	2	bow		RBF	C=0.1 γ=0.1	0.51 ± 0.03	0.68 ± 0.02
Model 16	2	bow		linear	C=100	0.50 ± 0.02	0.68 ± 0.02
Model 17	1	bow		RBF	C=100 γ=1	0.48 ± 0.02	0.67 ± 0.02
Model 18	1	bow		linear	C=1	0.53 ± 0.02	0.64 ± 0.02

Table 5.1: Performance of different settings of parameters. Evaluations were made on peptide level. The statictiscs were averaged over five-fold cross-validation. The settings are sorted first by AUC and then by average precision.



(a) **(b)**

Figure 5.2: Optimization by multiple kernel learning. (a) ROC curves. AUC was 0.79 before optimization and 0.80 after optimization. (b) Precision-recall curves. Average precision was 0.69 before optimization and 0.70 after optimization.

others remained 0. AUC was increased from 0.79 to 0.80 (Figure 5.2a), and average precision was increased from 0.69 to 0.70 (Figure 5.2b). Since the improvement was not obvious and the model was simpler before optimization, we prefer Model 1 to the combination of Model 1 and Model 2.

The improvement brought by the optimization is not obvious. A possible explanation is that the features we combined are too similar. After all, they are the same type of features extracted using different parameters. To overcome this problem and further improve the effectiveness of optimization, we may involve other types of features such as propensity scales. Moreover, string kernels may be included to increase the diversity of kernels.

5.3.2 Optimization of alignment parameters

Model 6, 9 and 11 in Table 5.1 are linear models that use local alignment scores without threshold as features, so we tried to improve them by optimizing the alignment parameters.

Prior to carrying out gradient descent, we need to make sure that the analytical calculation of gradient as proposed in (4.9) is correct. This can be tested by comparing the analytical and numeric derivatives of the parameters. Figure 5.3 shows the derivatives of some parameters in Model 11 and indicates that the gradient as we calculated is correct.

In Model 6 and 9, only positive substitution scores appear in the local alignments, so we only need to optimize them. For the same reason, we only optimized positive substitution scores and the gap penalty in Model 11. Table 5.2 compares the performance of the models before and after optimization. There was a significant improvement in Model 11.

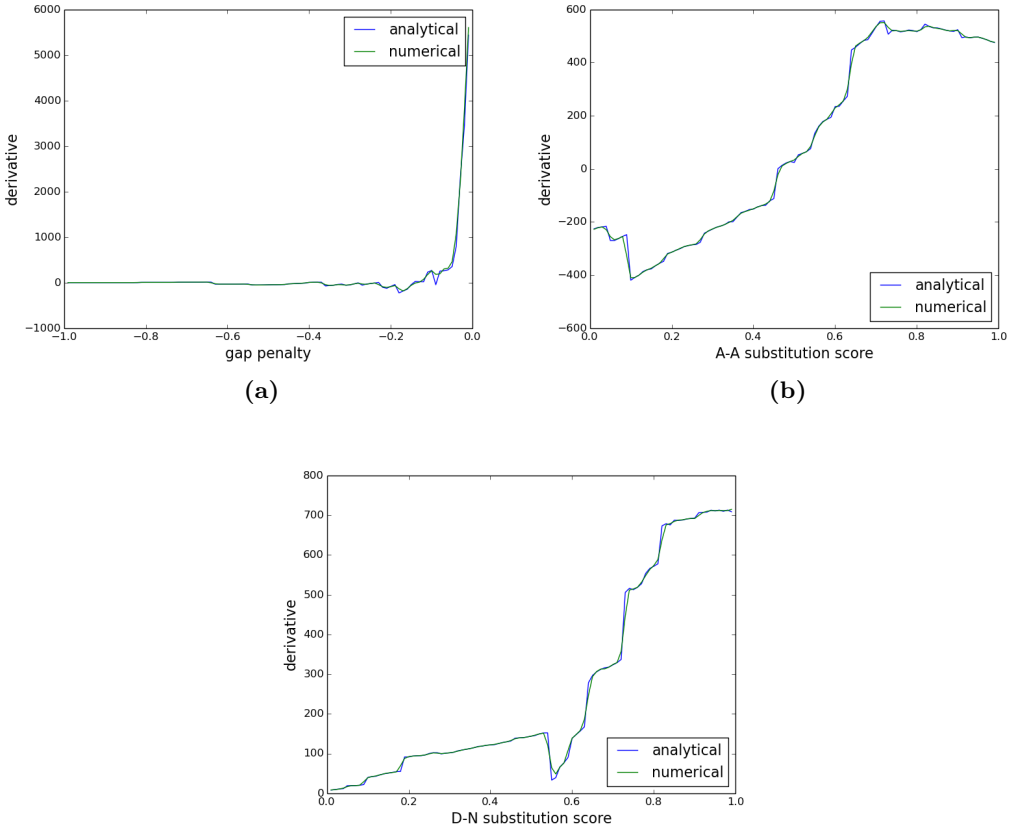


Figure 5.3: Comparison between analytical and numerical derivatives. When calculating the derivative of a parameter, all other parameter remained the same as in Model 11.

Model	before optimization		after optimization	
	average precision	AUC	average precision	AUC
6	0.59	0.73	0.59	0.74
9	0.56	0.72	0.58	0.73
11	0.56	0.71	0.64	0.75

Table 5.2: Results for alignment parameter optimization.

5.4 Epitope detection

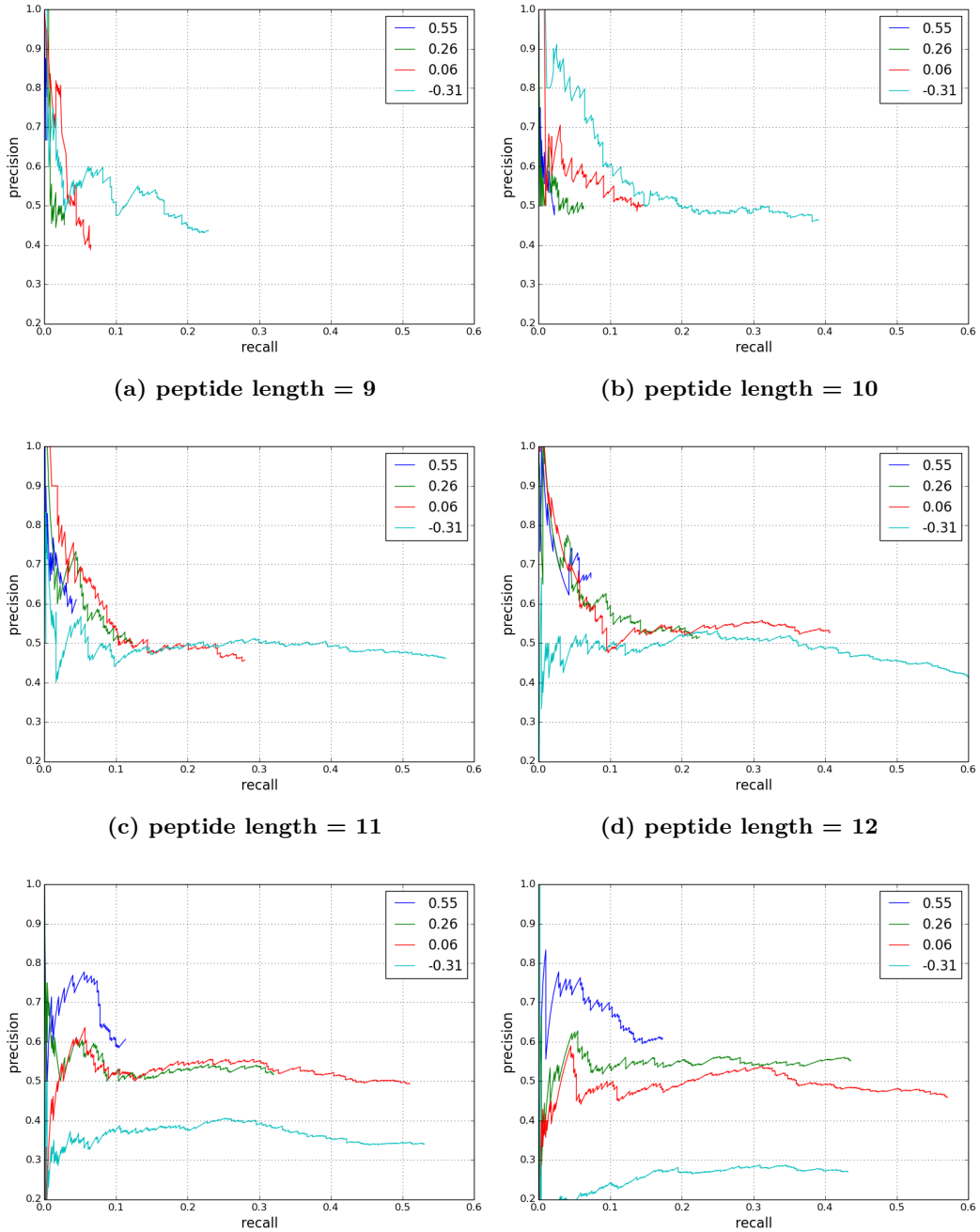
The best model we obtained so far is Model 1 in Table 5.1. In the rest of this chapter, all results are concerned with this model.

In this section, we evaluated the ability of our model to detect epitopes as described in § 4.3.2. We first set that if the overlap between a detection and any actual epitope is larger than 0.5, then the detection is true, otherwise it is false. Before making detections, we need to the set the length of extracted peptides and the threshold on their prediction scores. Figure 5.4 shows the impact of these two parameters on the detection result. 110, 11 and 12 are suitable lengths. For those lengths, the top scored detections are likely to be true epitopes. Lower threshold results in higher maximum recall, but if it is too low the precision of top detections will be compromised. We will use 10 as the default peptide length and -0.31 as the default threshold, which yields a maximum recall around 0.4 while maintaining the precision above or around 0.5. The lengths of detected epitopes were slightly shorter than actual epitopes (Figure 5.5).

5.5 Comparison with other epitope prediction tools

In this section, we compared our model with tools provided on `http://tools.immuneepitope.org/bcell/`. They are abbreviated as Chou & Fasman [7], Emini [10], Karplus & Schulz [21], Kolaskar & Tongaonkar [22], Parker [29] and Bepipred. Bepipred combines propensity scale with HMM, while all others are based only on propensity scales. One group of antigens in our dataset were used to test all models, and the other four groups were used to train our model. Since all the other models assign scores to individual residues, we made comparisons on residue level. Figure 5.6 and Table 5.3 shows that our model performs significantly better than all the other models.

The training data for Bepipred is publicly available, so we also used it to train our model to make more convincing comparison with Bepipred. This dataset contains only 124 antigens, from which 437 epitopes and 3239 non-epitopes were extracted. The comparison result is also shown in Figure 5.6 and Table 5.3. Although the performance of our model was decreased, it was still better than Bepipred. The decrease in performance may be due to the small size of training data, and the fact that they are not fully annotated.



(a) peptide length = 9

(b) peptide length = 10

(c) peptide length = 11

(d) peptide length = 12

(e) peptide length = 13

(f) peptide length = 14

Figure 5.4: Impact of peptide length and threshold of prediction score on epitope detection. In each plot, peptide length does not change, and different colors stand for different thresholds.

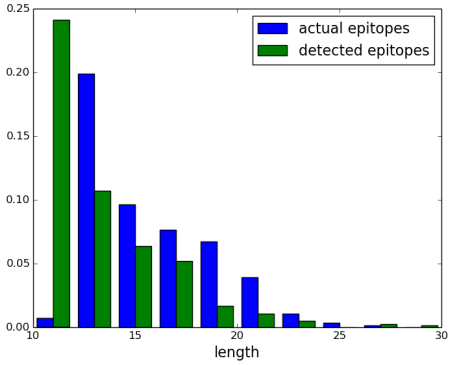
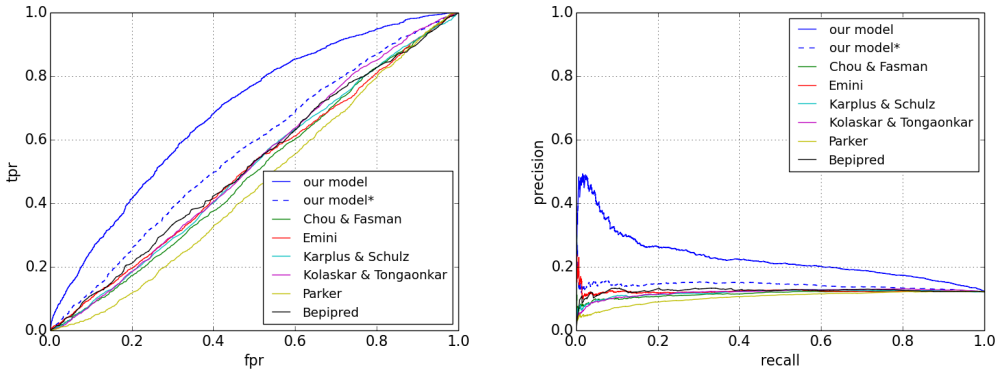


Figure 5.5: The length distributions of detected epitopes and actual epitopes.



(a) **(b)**

Figure 5.6: Comparison of different methods. (a) ROC curves. (b) Precision-recall curves. 'Our model' was trained using our dataset, and 'Our model*' was trained using the training data of Bepipred.

Model	average precision	AUC
our model	0.23	0.69
our model*	0.14	0.56
Bepipred	0.12	0.52
Chou & Fasman	0.11	0.49
Emini	0.12	0.51
Karplus & Schulz	0.12	0.51
Kolaskar & Tongaonkar	0.12	0.52
Parker	0.10	0.46

Table 5.3: Comparison of different methods.'Our model' was trained using our dataset, and 'Our model*' was trained using the training data of Bepipred.

Chapter 6

Conclusions and discussions

In this work, we explored different combinations of feature mapping methods and kernel functions for predicting continuous epitopes. On this basis, we built an predictor that can reliably distinguish epitopes from non-epitopes. In particular, we evaluated its ability to make epitope detections from whole antigen sequences, and the top ranked detections are highly likely to be actual epitopes. Results also show that our predictor greatly outperforms some other predictors.

Our results show that feature extraction plays a major role in building a good predictor. Local alignment scores can well characterize epitopes. The alignments of peptides to 2-mers and 3-mers often outperform 1-mers. The simple procedure for feature selection may be the reason that 3-mers do not perform as good as 2-mers, and we may use more sophisticated feature selection methods, such as metric learning [25], to improve the results.

We also developed two optimization approaches. The improvement brought by multiple kernel learning is not obvious. The reason may be that the features that we combined are too similar. After all, they are just the same type of features with different parameters. One possibility to improve the result of optimization is to involve other types of features (such as propensity scales). We may also incorporate other types of kernels (such as spectrum kernel [27]).

On the other hand, the optimization of alignment parameters brought significant improvement in one of the three models we tested. Although we have only implemented optimization for linear models, the target function 4.4 and its gradient can be modified so that a non-linear model can also be optimized.

It is difficult to compare existing methods due to several issues. First of all, there are no consensus metrics for evaluation. Non-parametric statistics, such as AUC, are generally preferred in the literature [15, 42]. However, little attention has been paid to the level (peptide or residue) of evaluation. Our results show that the same statistic changes a lot when measured on a different level, but existing methods were evaluated on either level. To make direct comparisons, either we should agree on which to choose, or evaluations on both levels should be reported.

Another problem is the lack of consensus benchmark dataset. The reported per-

formance of some models was biased because they were trained and/or tested using incompletely annotated dataset. Both [24] and our results show that incomplete annotations may lead to underestimation of the performance of predictors. To address this problem, a high-quality dataset should be assembled, and existing models should be evaluated against it.

Bibliography

[1] Alain JP Alix. Predictive estimation of protein linear epitopes by using the program people. *Vaccine*, 18(3):311–314, 1999.

[2] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[3] Martin J Blythe and Darren R Flower. Benchmarking b cell epitope prediction: underperformance of existing methods. *Protein Science*, 14(1):246–248, 2005.

[4] Inmaculada Cerecedo, Javier Zamora, Wayne G Shreffler, Jing Lin, Ludmilla Bardina, Ma Carmen Dieguez, Julie Wang, Alfonso Muriel, Belén de la Hoz, and Hugh A Sampson. Mapping of the ige and igg4 sequential epitopes of milk allergens with a peptide microarray–based immunoassay. *Journal of Allergy and Clinical Immunology*, 122(3):589–594, 2008.

[5] Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[6] J Chen, H Liu, J Yang, and K-C Chou. Prediction of linear b-cell epitopes using amino acid pair antigenicity scale. *Amino acids*, 33(3):423–428, 2007.

[7] P Y Chou and G D Fasman. Prediction of the secondary structure of proteins from their amino acid sequence. *Adv Enzymol Relat Areas Mol Biol*, 47:45–148, 1978.

[8] Stephen R Comeau, David W Gatchell, Sandor Vajda, and Carlos J Camacho. Cluspro: an automated docking and discrimination method for the prediction of protein complexes. *Bioinformatics*, 20(1):45–50, 2004.

[9] Yasser EL-Manzalawy, Drena Dobbs, and Vasant Honavar. Predicting linear b-cell epitopes using string kernels. *Journal of molecular recognition*, 21(4):243–255, 2008.

[10] EMILIO A Emini, JOSEPH V Hughes, DoS Perlow, and J Boger. Induction of hepatitis a virus-neutralizing antibody by a virus-specific synthetic peptide. *Journal of virology*, 55(3):836–839, 1985.

[11] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[12] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[13] Damien Fleury, Rod S Daniels, John J Skehel, Marcel Knossow, and Thierry Bizebard. Structural evidence for recognition of a single epitope by two distinct antibodies. *Proteins: Structure, Function, and Bioinformatics*, 40(4):572–578, 2000.

[14] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708, 1982.

[15] Jason A Greenbaum, Pernille Haste Andersen, Martin Blythe, Huynh-Hoa Bui, Raul E Cachau, James Crowe, Matthew Davies, AS Kolaskar, Ole Lund, Sherrie Morrison, et al. Towards a consensus on datasets and evaluation metrics for developing b-cell epitope prediction tools. *Journal of Molecular Recognition*, 20(2):75, 2007.

[16] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[17] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[18] Ying Huang, Beifang Niu, Ying Gao, Limin Fu, and Weizhong Li. Cd-hit suite: a web server for clustering and comparing biological sequences. *Bioinformatics*, 26(5):680–682, 2010.

[19] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[20] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online].

[21] PA Karplus and GE Schulz. Prediction of chain flexibility in proteins. *Naturwissenschaften*, 72(4):212–213, 1985.

[22] AS Kolaskar and Prasad C Tongaonkar. A semi-empirical method for prediction of antigenic determinants on protein antigens. *FEBS letters*, 276(1):172–174, 1990.

[23] Dieter Kraft et al. *A software package for sequential quadratic programming*. DFVLR Obersfaffeuhofen, Germany, 1988.

[24] Jens Vindahl Kringelum, Claus Lundegaard, Ole Lund, and Morten Nielsen. Reliable b cell epitope predictions: impacts of method development and improved benchmarking. *PLoS computational biology*, 8(12):e1002829, 2012.

[25] Brian Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2012.

[26] JE Larsen, Ole Lund, and Morten Nielsen. Improved method for predicting linear b-cell epitopes. *Immunome Res*, 2(2):1–7, 2006.

[27] Christina S Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific symposium on biocomputing*, volume 7, pages 566–575. World Scientific, 2002.

[28] Shide Liang, Dandan Zheng, Daron M Standley, Bo Yao, Martin Zacharias, and Chi Zhang. Epsvr and epmeta: prediction of antigenic epitopes using support vector regression and multiple server results. *BMC bioinformatics*, 11(1):381, 2010.

[29] JMR Parker, D Guo, and RS Hodges. New hydrophilicity scale derived from high-performance liquid chromatography peptide retention data: correlation of predicted surface residues with antigenicity and x-ray-derived accessible sites. *Biochemistry*, 25(19):5425–5432, 1986.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] Julia V Ponomarenko and Philip E Bourne. Antibody-protein interactions: benchmark datasets and prediction tools evaluation. *BMC Structural Biology*, 7(1):64, 2007.

[32] LATA Prasad, SADHANA Sharma, M Vandonselaar, JW Quail, JS Lee, EB Waygood, KS Wilson, Z Dauter, and LT Delbaere. Evaluation of mutagenesis for epitope mapping. structure of an antibody-protein antigen complex. *Journal of Biological Chemistry*, 268(15):10705–10708, 1993.

[33] Marc H.V. Van Regenmortel. What is a b-cell epitope? In Ulrich Reineke and Mike Schutkowski, editors, *Epitope Mapping Protocols*, pages 3–20. Humana Press, New York, second edition, 2008.

[34] Sudipto Saha and GPS Raghava. Prediction of continuous b-cell epitopes in an antigen using recurrent neural network. *Proteins: Structure, Function, and Bioinformatics*, 65(1):40–48, 2006.

[35] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[36] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[37] Eric J. Sundberg. Structural basis of antibodyâĂŞantigen interactions. In Ulrich Reineke and Mike Schutkowski, editors, *Epitope Mapping Protocols*, pages 23–36. Humana Press, New York, second edition, 2008.

[38] Peter Timmerman, Wouter C Puijk, and Rob H Meloen. Functional reconstruction and synthetic mimicry of a conformational epitope using clipsâĎć technology. *Journal of Molecular Recognition*, 20(5):283–299, 2007.

[39] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[40] Lawrence JK Wee, Diane Simarmata, Yiu-Wing Kam, Lisa FP Ng, and Joo Chuan Tong. Svm-based prediction of linear b-cell epitopes using bayes feature extraction. *BMC genomics*, 11(Suppl 4):S21, 2010.

[41] Bo Yao, Dandan Zheng, Shide Liang, and Chi Zhang. Conformational b-cell epitope prediction on antigen protein structures: a review of current algorithms and comparison with common binding site prediction methods. *PLoS One*, 8(4):e62249, 2013.

[42] EM Yasser and Vasant Honavar. Recent advances in b-cell epitope prediction methods. *Immunome Res*, 6(Suppl 2):S2, 2010.

TRITA -MAT-E 2015:51
ISRN -KTH/MAT/E--15/51--SE