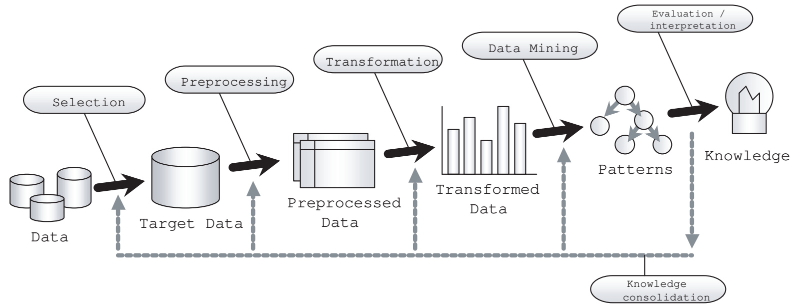

Figure 1: The KDD process by Fayyad *et al.* [9]

1 Introduction

The general flow and components which comprise the knowledge discovery process have come to be recognized in the literature [9, 12]. According to these articles, the KDD (Knowledge Discovery from Databases) process can be, in general, divided into several stages such as: data preparation (selection, preprocessing, transformation) data mining, hypothesis interpretation/evaluation, and knowledge consolidation (Figure 1). Several important elements which are keys to the success of this process are:

- Selection and/or development of appropriate algorithms to solve the problem
- Selection and/or generation of appropriate attributes which represent the data
- Admittance and/or facilitation of human intervention into discovery systems

The first element can be understood as being very important, obviously, represented in the numerous methods in machine learning, statistics, etc., which have been developed and successfully applied to various data sets. The second element is important, especially when dealing with unfamiliar data, where discovering *good* attributes are sometimes the most difficult obstacle in the discovery process [13]. Studies on this topic can be found, for example in the machine learning literature, in methods such as feature construction and constructive induction [4, 16, 18]. The importance of the third element is stressed in [11], where discovery systems are strongly recommended to provide more explicit support for human intervention. The KDD process also consists of numerous trial-and-error iterations [5] which can be related to this element.

Through the $\mathcal{HC}$ project, the concept of *view* and *view operator* has been introduced as *building blocks* of the computational knowledge discovery process [3, 13, 14, 15], and can be related to the afore mentioned elements. A *view* is a generalized form of attribute, a function over objects (data) under consideration which returns values representing certain aspects of the data. A *view operator* is a function that creates new views from existing views and data. Views and view operators can be used to describe the steps of the computational discovery process (Table 1). For example, a machine learning algorithm can be regarded as a view operator which produces new hypotheses (which are also views), from existing views (attributes). The structure of the combination of views and view operator is called the *view design.* The goal of KDD can be restated as the search for *meaningful* views and view design. The "hand-crafting" of good view design can be regarded as an interface for human intervention into the discovery process [3, 13]. The iterative cycle of KDD consists very much of composing and decomposing of views.

Table 1: KDD process in terms of view [3].

Components of the KDD Process	Description in terms of view
1) data selection	classification and filtering of entities according to a certain view, which decides whether the entity is used or not.
2) data preprocessing	Preprocessing of data can be expressed as a function over data, so naturally may be defined by a view.
3) data transformation	Transformation can also be expressed as a function over data, so naturally may be defined by a view.
4) data mining	Data mining can be expressed as a generation of new view. Hypothesis generation algorithms can be considered as view operators.
5) interpretation/evaluation	interpretation/evaluation of a view
6) knowledge consolidation	recursively using newly generated view

Adopting such a *view oriented* perspective, we worked on the problem of characterizing N-terminal protein sorting signals, and have obtained significant results. The knowledge obtained in our investigation has been compiled as an expert system called iPSORT, and experimental service is provided at its website [28]. From this experience, we have also designed a new functional programming language, VML, as an extension to the ML language, which can assist the trial-and-error development of such *view oriented* applications [1]. The two main extensions are the keywords '`view`' and '`vmatch`', which are used for defining and decomposing views, freeing the programmer from the labor of explicitly keeping track of view designs. Formalism and sample implementation of this language has been provided in [26].

In Section 2, we give brief definitions and examples of entity, view, view operator, and the VML language. Section 3 describes our study on the characterization of N-terminal protein sorting signals. Section 4 concludes the paper.

2 Basic Concepts in the $\mathcal{HC}$ Project

2.1 Entity, View, View Operator

Here, we review the definitions of entity, view, and view operator, and show how the KDD process can be described in terms of these concepts. An *entity set* E is a set of objects which may be distinguished from one another, representing the data under consideration. Each object $e \in E$ is called an *entity.* A *view* $v : E \to R$ is a function over E. v will take an entity e, and return some aspect (i.e. attribute value) concerning e. A *view operator* is a function which generates new views from existing views and entities. Below are some examples:

Example 1 Given a view $v : E \rightarrow R$, a new view $v' : E \rightarrow R'$ may be created with a function $\psi : R \rightarrow R'$ (i.e. $v' \equiv \psi \circ v : E \xrightarrow{v} R \xrightarrow{\psi} R'$).

We can also consider n-ary functions as views. All arguments except for the argument expecting the entity can be regarded as *parameters* of the view.

Hypothesis generation via machine learning algorithms can also be considered as kind of view operator. The generated hypothesis can also be considered a view.

Example 2 Given a set of data records (entities) and their attributes (views), the ID3 algorithm [23] (view operator) generates a decision tree T. T is also a view because it is a function which returns the class that a given entity is classified to. The generated view T can also be used as an input to other view operators, to create new views, which can be regarded as knowledge consolidation.

Views and view operators are combined to create new views. The structure of such combinations of a compound view, is called the *design* of the view. The task of KDD lies in the search for *good* views which explain the data. Knowledge concerning the data is encapsulated in its design. Human intervention can be conducted through the hand-crafted design of views by domain experts. To successfully assist the expert in the knowledge discovery process, the expert should be able to manipulate and understand the view design with ease. Figure 2 in Section 3 depicts an example of an actual view design which was used in our computational experiments.

2.2 VML

In designing views in practice, the primary difference between a view and a function, is that views must always have an interpretable *meaning*, because the knowledge must be interpretable to be of any use. To lighten the load of the programmer for managing this information, the idea of VML is that the *origin* of a value (i.e. how the value was calculated - the view design of the function which calculated the value) calculated from a function specified by the `view` keyword, is implicitly "remembered" by the runtime system and can be consulted with the `vmatch` keyword.

Here, we give a simple example to illustrate basic syntax and the use of `view` and `vmatch` statements. The syntax and semantics of VML are the same as OCaml [29] except for the added keywords.

A function which calculates the sum of positive integers 1 to n can be written in OCaml as:[1]

```
# let rec sumn n = if n <= 0 then 0 else (n + sumn (n-1));;
val sumn : int -> int = <fun>
```

[1]The expressions starting after '`#`' and ending with '`;;`' is the input by the user, the others are responses from the compiler/interpreter. Comments are written between '`(*`' and '`*)`'.

```
# sumn 10;;                        (* apply 10 to sumn *)
- : int = 55
```

`let` binds the function (value) to the name `sumn`. `rec` specifies that the function is a *recursive* function (a function that calls itself). `n` is an argument of the function `sumn`. `int -> int` is the type of the function `sumn`, which reads as follows: "`sumn` is a function that takes a value of type `int` as an argument, and returns a value of type `int`". Notice that the type of `sumn` is *automatically inferred* by the compiler/interpreter, and need not be specified. Arguments can be applied to functions just by writing them consecutively.

The syntax of the `view` keyword is the same as the `let` statement. If we specify the above function with the `view` keyword in place of `let` (we capitalize the first letter of such functions for convenience, and call them *view functions*):

```
# view rec Sumn n = if n <= 0 then 0 else (n + Sumn (n-1));;
val Sumn : int -> int = <fun>::(n . Sumn n)
# Sumn 10;;
- : int = 55::(Sumn 10)
```

`Sumn` is defined as a view function, and therefore, values calculated from `Sumn` are implicitly remembered. In the above example, the return value is 55, and its representation, shown to the right of the double colon '::', is `(Sumn 10)`. We do not need to see the *inside* of `Sumn`, if we know the meaning of `Sumn`, to understand this value of 55.

The `vmatch` keyword is used to decompose a representation of a value and extract the function and/or any parameters which were used to created the value. Its syntax is the same as the `match` statement of OCaml, which is used for the pattern matching of miscellaneous data structures.

```
# let v = Sumn 10;; (* apply 10 to Sumn and bind value to v *)
val v : int = 55::(Sumn 10)
# vmatch v with    (* Extract parameter used to calculate v *)
    (Sumn x) -> printf "%d was applied to Sumn\n" x
  | _ -> printf "Error: v did not match (Sumn x)\n";;
10 was applied to Sumn
- : unit = ()
```

In the above example, the representation of `v`, which is `(Sumn 10)`, is matched against the pattern `(Sumn x)`. If the match is successful, '`x`' in the pattern is assigned the corresponding value `10`. This value can be used in the expression to the right of '`->`' which is evaluated in case of a match. Multiple patterns matches can be attempted: each pattern and its corresponding expression are separated by '`|`', and the expression for the first matching pattern is evaluated. The underscore '_' represents a *wild card* pattern, matching any representation. The entire `vmatch` expression evaluates to the unit type `()` (similar to `void` in the C language) in this case, because `printf` is a function that executes a side-effect (print a string), and returns `()`.

Figure 3 in the next section shows an example of an actual view design (written in VML) which was used in our computational experiments.

3 Characterization of N-terminal Protein Sorting Signals

In this section, we describe our approach to the characterization of N-terminal protein sorting signals.

3.1 Problem Statement

Proteins are composed of amino acids, and can be regarded as strings consisting of an alphabet of 20 characters. Most proteins are first synthesized in the cytosol, and carried to specified locations, called *localization sites*. In most cases, the information determining the subcellular localization site is represented as a short amino acid sequence segment called a *protein sorting signal* [19]. Given an amino acid sequence, predicting where the protein will be carried to is an important and difficult problem in molecular biology. Although numerous signal sequences have been found, similarities between these sequence for the same localization site are not yet fully understood. For N-terminal protein signals (signals which are known to be located near the N-terminal of the amino acid sequence), TargetP [8] is currently known as the best predictor. However, although neural networks are "readily available" and "often successful in practice", they are also infamous for the difficulty involved in trying to understand and interpret their meaning [6]. MitoProt [7] and PSORT [21, 20], unlike TargetP, are systems which incorporate existing knowledge about sorting signals as numerical attributes hand-crafted by experts. However, the main idea lies in the linear discriminant analysis of such attributes, which may not be easily interpretable. Also, they are somewhat obsolete and their performance is unsatisfactory compared to TargetP. Our aim was to come up with a predictor which could challenge TargetP [8] in terms of prediction accuracy while not sacrificing the *interpretability* of the classification rule.

3.2 Data

Data available from the TargetP web-site [30] was used, consisting of 940 sequences containing 368 mTP (mitochondrial targeting peptides), 141 cTP (chloroplast transit peptides), 269 SP (signal peptides), and 162 "Other" sequences. The general approach was to: discuss with an expert on how to design the views, conduct computational experiments with those view designs, present results to the expert as feedback, and then repeat the process.

3.3 Approach and Results

We first considered *binary* classifiers, which distinguishes sequences of a certain signal. The entity set is the set of amino acid sequences. The views we look for are of type `string -> bool`: for an amino sequence, return a Boolean value, `true` if the sequence contains a certain signal, and `false` if it does not.

We designed views from two standpoints. First, noticing that although strong consensus patterns do not seem to be present in the signals, there does seem to be a common structure to each of the signals. Therefore, following the work of the BONSAI system [25], an *alphabet indexing* + approximate pattern view was designed. An alphabet indexing [24] can be considered as a discrete, unordered version of amino acid indices, mapping amino acids to a smaller alphabet. After transforming the original amino acid sequence into a sequence of the alphabet indices, an approximate pattern [27] is sought. This view can be regarded as trying to capture "local", structural characteristics of the signals. We tested various parameters for this view, but we were not able to obtain satisfactory results.

We then attempted to capture "global" characteristics of the signals. Since existing knowledge about the signals seemed to depend on biochemical properties of the amino acids contained, we decided to use the AAindex database [10], which is a compilation of 434 *amino acid indices*, where each amino acid index is a mapping from one amino acid to a numerical value, representing various physiochemical and biochemical properties of the amino acids. This turned out to be very effective, especially for the SP set, and was used to distinguish SP from the other signals.

For the remaining signals, we tried the logical combination of views of both designs, which proved to be useful for distinguishing the "Other" set (those which do not have N-terminal signals), from mTP and cTP.

Figure 2 and 3 shows the view designs. For each design, a search for a good view consists of searching a wide range of the parameters.

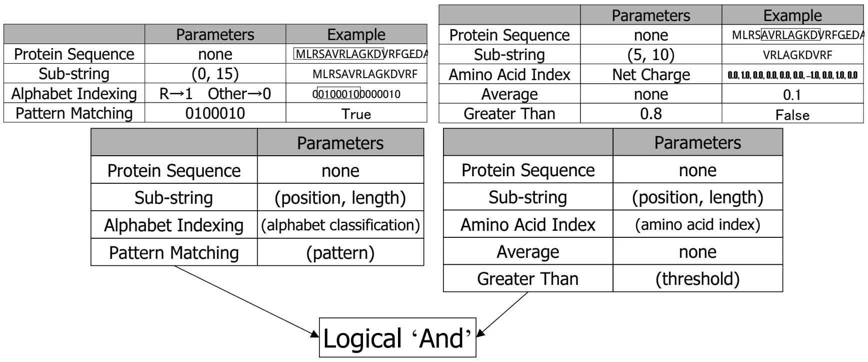

Figure 2: The three types of views and view designs used in the characterization of protein sorting signals.

By combining the views and parameters thus obtained for each signal type into a single decision list, we were able to create a rule which competes fairly well with TargetP in terms of prediction accuracy. The scores of a 5-fold cross-validation is shown in Table 2 [2].

The knowledge encapsulated in the view design was consistent with widely believed (but vague) characteristics of each signal, and the expert was surprised that such a simple rule could describe the sorting signals with such accuracy. Biological interpretations of the knowledge discovered is given

[2]The score used for comparison is the Matthews correlation coefficient (MCC) [17], which assumes that the cost of a false positive is the same as cost of a false negative. The rationale for adopting this score is that 1) different score penalties may arise in different situations, and choosing an appropriate penalty for comparison is a difficult issue, and 2) MCC has been used in the literature for comparing the scores, and we have followed the tradition.

```
# let h1 pat mm ind pos len str =
    Astrstr mm pat (AlphInd ind (Substring pos len str));;
val h1 : string -> astr_mismatch -> (char -> char) -> int -> int ->
         string -> bool = <fun>
# let h2 thr ind pos len str =
    GT (Average (AAindex ind (Substring pos len str))) thr;;
val h2 : float -> string -> int -> int -> string -> bool = <fun>
# let h3 thr aaind pos1 len1 pat mm alphind pos2 len2 str =
    And (h1 pat mm alphind pos1 len1 str)
        (h2 thr aaind pos2 len2 str);;
val h3 : float -> string -> int -> int -> string ->
    astr_mismatch -> (char -> char) -> int -> int -> string -> bool = <fun>
```

Figure 3: Views and view designs used in the characterization of protein sorting signals written in VML [1]. Notice the types of each view function ends in `string->bool`, meaning that after appropriate arguments are applied, it is a function which returns a Boolean value given a string.

in [2]. A system called iPSORT was built based on these rules, and an experimental web service is provided at the iPSORT web-site [28].

Table 2: The Prediction Accuracy of the Final Hypothesis (scores of TargetP [8] in parentheses)

True category	# of seqs	Predicted category: cTP	mTP	SP	Other	Sensitivity	MCC
cTP	141	96 (120)	26 (14)	0 (2)	19 (5)	0.68 (0.85)	0.64 (0.72)
mTP	368	25 (41)	309 (300)	4 (9)	30 (18)	0.84 (0.82)	0.75 (0.77)
SP	269	6 (2)	9 (7)	244 (245)	10 (15)	0.91 (0.91)	0.92 (0.90)
Other	162	8 (10)	17 (13)	2 (2)	135 (137)	0.83 (0.85)	0.71 (0.77)
Specificity		0.71 (0.69)	0.86 (0.90)	0.98 (0.96)	0.70 (0.78)		

The score is defined by: $\frac{(tp \times tn - fp \times fn)}{\sqrt{(tp+fn)(tp+fp)(tn+fp)(tn+fn)}}$ where tp, tn, fp, fn are the number of true positive, true negative, false positive, and false negative, respectively (Matthews correlation coefficient (MCC) [17]).

4 Conclusion and Future Work

In this paper, we discussed various issues we encountered in the $\mathcal{HC}$ project through the concept of views. We should admit that we have not yet conducted a large amount of computational experiments for various data under this approach, but our experience so far has been promising. We have also done experiments concerning the detection of putative gene regulatory sites, partially described in [1]. Our next target is to investigate the *alternative splicing* mechanism of human RNA, using an Aberrant Splicing Database [22].

5 Acknowledgements

This research was supported in part by Grant-in-Aid for Encouragement of Young Scientists and Grant-in-Aid for Scientific Research on Priority Areas (C) "Genome Information Science" from the Ministry of Education, Sports, Science and Technology of Japan, and the Research for the Future Program of the Japan Society for the Promotion of Science.

References

[1] H. Bannai, Y. Tamada, O. Maruyama, and S. Miyano. VML: A view modeling language for computational knowledge discovery. In *Discovery Science*, Lecture Notes in Artificial Intelligence, 2001. To appear.

[2] H. Bannai, Y. Tamada, O. Maruyama, K. Nakai, and S. Miyano. Extensive feature detection of n-terminal protein sorting signals. *Bioinformatics*, 2001. To appear.

[3] H. Bannai, Y. Tamada, O. Maruyama, K. Nakai, and S. Miyano. Views: Fundamental building blocks in the process of knowledge discovery. In *Proceedings of the 14th International FLAIRS Conference*, pages 233–238. AAAI Press, 2001.

[4] E. Bloedorn and R. S. Michalski. Data-driven constructive induction. *IEEE Intelligent Systems*, pages 30–37, March/April 1998.

[5] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In *Advances in Knowledge Discovery and Data Mining.* AAAI Press/MIT Press, 1996.

[6] K.-C. Chou. Using subsite coupling to predict signal peptides. *Protein Engineering*, 14(2):75–59, 2001.

[7] M. G. Claros and P. Vincens. Computational method to predict mitochondrially imported proteins and their targeting sequences. *Eur. J. Biochem.*, 241(3):779–786, November 1996.

[8] O. Emanuelsson, H. Nielsen, S. Brunak, and G. von Heijne. Predicting subcellular localization of proteins based on their N-terminal amino acid sequence. *J. Mol. Biol.*, 300(4):1005–1016, July 2000.

[9] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54, 1996.

[10] S. Kawashima and M. Kanehisa. AAindex: Amino Acid index database. *Nucleic Acids Res.*, 28(1):374, 2000.

[11] P. Langley. The computer-aided discovery of scientific knowledge. In *Lecture Notes in Artificial Intelligence*, volume 1532, pages 25–39, 1998.

[12] P. Langley and H. A. Simon. Applications of machine learning and rule induction. *Communications of the ACM*, 38(11):54–64, 1995.

[13] O. Maruyama and S. Miyano. Design aspects of discovery systems. *IEICE Transactions on Information and Systems*, E83-D:61–70, 2000.

[14] O. Maruyama, T. Uchida, T. Shoudai, and S. Miyano. Toward genomic hypothesis creator: View designer for discovery. In *Discovery Science*, volume 1532 of *Lecture Notes in Artificial Intelligence*, pages 105–116, 1998.

[15] O. Maruyama, T. Uchida, K. L. Sim, and S. Miyano. Designing views in HypothesisCreator: System for assisting in discovery. In *Discovery Science*, volume 1721 of *Lecture Notes in Artificial Intelligence*, pages 115–127, 1999.

[16] C. J. Matheus and L. A. Rendell. Constructive induction on decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 645–650, 1989.

[17] B. W. Matthews. Comparison of predicted and observed secondary structure of t4 phage lysozyme. *Biochim. Biophys. Acta*, 405:442–451, 1975.

[18] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161, 1983.

[19] K. Nakai. Protein sorting signals and prediction of subcellular localization. In P. Bork, editor, *Analysis of Amino Acid Sequences*, volume 54 of *Advances in Protein Chemistry*, pages 277–344. Academic Press, San Diego, 2000.

[20] K. Nakai and P. Horton. PSORT: a program for detecting the sorting signals of proteins and predicting their subcellular localization. *Trends Biochem. Sci.*, 24:34–35, 1999.

[21] K. Nakai and M. Kanehisa. A knowledge base for predicting protein localization sites in eukaryotic cells. *Genomics*, 14:897–911, 1992.

[22] K. Nakai and H. Sakamoto. Construction of a novel database containing aberrant splicing mutations of mammalian genes. *Gene*, 141:171–177, 1994. http://www.hgc.ims.u-tokyo.ac.jp/~knakai/asdb.html.

[23] J. Quinlan. Induction of decision trees. *Machine Learning 1*, 1:81–106, 1986.

[24] S. Shimozono. Alphabet indexing for approximating features of symbols. *Theor. Comput. Sci.*, 210:245–260, 1999.

[25] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, and S. Arikawa. Knowledge acquisition from amino acid sequences by machine learning system BONSAI. *Trans. Information Processing Society of Japan*, 35(10):2009–2018, 1994.

[26] E. Sumii and H. Bannai. VMlambda: A functional calculus for scientific discovery. http://www.yl.is.s.u-tokyo.ac.jp/~sumii/pub/, 2001.

[27] S. Wu and U. Manber. Fast text searching allowing errors. *Commun. ACM*, 35:83–91, 1992.

[28] iPSORT - http://www.hypothesiscreator.net/iPSORT/.

[29] Objective Caml - http://caml.inria.fr/ocaml/.

[30] TargetP - http://www.cbs.dtu.dk/services/TargetP/.