Chapter 2

Concepts and Terminology

2.1 Machine Learning and Data Mining

We will describe a high-level abstract on machine learning and data mining (also additionally bioinformatics), introducing key terms to be used. These terms are explained later more. Readers, particularly entry-level machine learners, are recommended to go through this chapter without thinking to understand the terms fully within this chapter.

Two terms, Machine Learning and Data Mining, can be considered interchangeably, like that they are synonyms. We then use the term "machine learning" mainly, while data mining are more application-oriented, and we use either of these two terms, depending on situations.

Data, input of machine learning, are a set of records, each being called an *instance*, *example* or *sample*. We use the "instance" among them through this book. One instance has *features*, *variables* or *variates*, where these features are visible and unchanged (during learning) and so called *observable variables*.

Occasionally instances can be classified into more than one categories, typically two (binary). Here are two examples:

Example 1: Mobile phone customers of some company can be segmented into at least the following two: *current subscribers* and *former subscribers* (though current subscribers might be categorized into more details, due to the risk of leaving. Also promising subscribers among totally non-subscribers would be also a possible, important category for mobile phone companies).

Example 2: Regarding some disorder, *patients* and *healthy people* (also patients would be categorized into more, due to the seriousness of the disorder).

As such, when instances can be segmented into categories, each category is called a *class*, and each of the above *current subscribers*, *former subscribers*, *patients* and *healthy people*, etc. is called a *label*. The label is the name of a class, and if there are five labels, this means there exist five classes. Here are more

examples:

Example 3: A customer is an instance, where the evaluation by one customer against some item is a label. Also even if a customer buys an item, this is already a label, to fill the corresponding part of the data.

Example 4: A gene is an instance, where its function can be a label.

In Example 3, the evaluation can be binary or some moderate-size number, like a five-stage, while in Example 4, the functions of genes cannot be even by such a small number but a larger number. These two examples imply that labels would not be necessarily assigned to all instances (or functions are not necessarily assigned to all genes), implying that assigning labels or functions to genes would be a possible problem setting. This means that a label is a value of one feature which should be predicted in machine learning.

Again all instances do not necessarily have labels, because obtaining labels is expensive. For example, in Example 3, we need to ask customers to give some evaluation against some item. Also we need some biological experiments to annotate functions to genes in life sciences, which needs cost and also time, which in some cases might not be good enough if it is really hard to find functions of genes.

Thus in machine learning we have a paradigm which do not assume classes and labels. This setting is called *unsupervised learning*, where the objective is to understand the data *distribution* or more generally to summarize data. A typical approach for summarizing data is *clustering*, where instances are grouped into several *clusters*. Clusters are different from observable variable like features and variables where their values can be trained from given data. Thus this variable is called a *latent variable*. Estimating latent variables means clustering.

On the other hand, we call learning using labels *supervised learning*, where inputs are features of instances and estimate a function to predict the label. This function is called a *hypothesis*, *model*, *classifier* or *predictor*. Note that model is not only the concept for supervised learning but also for estimating distribution in unsupervised learning. This function has *parameters*, where the values of parameters are trained (learned or estimated) by using input data, and the trained hypothesis is used to predict labels of unknown instances (or data). The first input data are called *training data* and the second is called *test data*.

Estimating a model from given training data means, for example, to keep the values given by the model closest to those of training data. Thus we can set up an *objective function*, for example, in supervised learning, being consistent with the error of the model from training data, i.e. an *error function* or *loss function*. We can then minimize the objective function, which turns into an optimization problem, like minimizing the error function. The optimization in terms of training data only causes overfitting of the model to the training data. Avoiding that, the model needs to be generalized, which is called *generalization* or *regularization*. In practice, we add one or more constraints to the objective function, where the constraints are, mathematically, terms, called *regularizers* or

Table 2.1: Data in life sciences.

	data types	examples
gene	vector	gene expression
	sequence (string)	nucleic acid sequence
	node in graph	gene regulatory network
protein	sequence (string)	amino acid sequence
	node in graph	protein-protein interaction network
chemical compound	(molecular) graph	chemical structure
(e.g. drug)	node in graph	metabolic pathway
glycan	tree	sequence structure

regularization terms. Thus to solve some problem, we can set up an objective function with regularizers, which we often call problem *formulation*. Usually in machine learning model formulation, model parameters to be estimated are in either the objective function or regularizers. Also model formulation sometimes has *hyperparameters*, which are given arbitrary by users or decided empirically but not trained from input data. For example, in model formulation, the coefficients of regularizers, called *regularization coefficients* are one typical hyperparameter.

We call values, which are clearly different from the data distribution, an *outlier*.

Now let's get back to Example 3, where the situation is an E-commerce site. It would not be the case that an customer has bought almost all items in the E-commerce site, while usually most customers bought only just a few goods in the site. Thus the data from E-commerce, i.e. a matrix of users (rows) and items (columns) has only a small number of elements, filled by some values, showing the goods evaluation by customers or just the user purchase history. More concretely the row vectors or users have only a few number of values, with others being *missing information*. This situation of data is called *sparse* data, and interestingly sometimes this data *sparsity* is useful to solve the problem efficiently even if the given matrix is huge.

2.2 Bioinformatics: Connections to Data Types

Bioinformatics is the study on data in life sciences, particularly molecular biology, so the focus being placed on molecules working in cells, such as *genes*, *proteins*, *chemical compounds*, etc. We can describe data formats of those molecules which are regarded as multiple ways even for one single molecule type, depending on situations.

Gene is the most attention-paid term in life sciences. Recent so-called *high-throughput* techniques allow to measure the expression of thousands of genes in cells simultaneously. The measurement can be performed under many different conditions, which results in numerical *vectors* of genes. At the same time, physically genes are *nucleic acids*, consisting of four *bases* (represented by four letters:

A, T, C, G), which are building blocks of *nucleic acid sequences*. Simply speaking they are *sequences* or *strings* of four letters. Proteins can be generated, following the expression of genes, by which the term "gene" is used in most cases, instead of protein. For example, gene function means the function of the corresponding protein. Physically proteins are also sequences, while building blocks of proteins are 20 different types of *amino acids*, by which sequences of proteins are *amino acid sequences* or strings of 20 letters. Another aspect of genes is gene regulation, which are eventually represented by a *graph*, called *gene regulatory network*, in which *nodes* are genes and an edge shows that a gene is regulated by another gene. Similarly one aspect of proteins is three-dimensional binding between two or more proteins, which are called *protein-protein interactions*, also usually being represented by a graph with nodes of proteins and edges of interactions.

Glycans are called the third molecule in cells, next to nucleic acids and proteins. Building blocks of glycans are *monosaccharides*, and glycan sequences are generated by the connection of monosaccharides. The difference of glycan sequences from gene sequences is that glycans allow to have branches (but no cycles) in their sequences, which results in glycan *trees*.

Nucleic acids of genes, amino acids of proteins and monosaccharides of glycans are all *chemical compounds*, and so molecules. The chemical structure of molecules are *graphs*, so-called *molecular graph*, meaning that chemical compounds can be presented by graphs. On the other hand, for example, *metabolic network* is a collection of chemical reactions in cells and shows the process of generating chemical compounds from other compounds. Thus chemical compounds can be nodes in the graph of metabolic network.

Table 2.1 shows the summary of molecules in cells and their possible data formats. As you can see from this table, bioinformatics data can be five types. When we think about general applications of machine learning and data mining, we can add one more data type, *sets*, to them, resulting in six types: vectors, sets, sequences (strings), trees, graphs and nodes in a graph. These data types are explained in more detail in the next section.

2.3 Six Types of Data

Data can be categorized into six types, partially according to the observation on the life science data. This book is organized according to the six data types, so each chapter being for each data type, as follows:

Chapter 3: Vectors

Chapter 4: Sets

Chapter 5: Sequences (strings)

Chapter 6: Trees

Chapter 7: Graphs

Chapter 8: Nodes in a graph

	Feature 1	Feature 2	Feature 3	Feature 4
Instance 1	A	B	C	D
Instance 2	A	C	C	C
...				

Figure 2.1: Instances, each being a vector, are a matrix.

Instance 1: $\{A, B, C, D\}$, Instance 2: $\{D, C, B, A\}$

Figure 2.2: Two instances, each being a set.

Chapter 9 is then on machine learning for data integration.

2.3.1 Vectors

When each instance has multiple features, the simplest and most frequent data in machine learning and data mining is vectors, where each instance has always a certain number of features. Fig. 2.1 shows an illustrative example of vectors (called *feature vectors*) for each instance, resulting in a matrix for multiple instances. Here are features, terminology and examples:

Building blocks: are feature values, which can be either discrete or continuous.

Fixed length and order: The length of a vector is fixed, and also a sort of feature values are ordered, meaning that for example, if Feature 1 in Fig. 2.1 is fixed to be a binary taking A or B, all first values of instances in this dataset must be A or B.

Example 1: demographic data: One instance is an individual, where features are age, gender, etc.

Example 2: gene expression: One instance is a gene, and features are experimental conditions, under which expression of the corresponding gene are measured. Each matrix element is the expression value of the corresponding gene and the corresponding experimental condition.

2.3.2 Sets

Each instance is a set, and input data is a set of sets. Sets are the most flexible machine learning data type. One set, i.e. an instance, can have an arbitrary number of elements, without any order. Fig. 2.2 shows an illustrative examples of sets. Here are features, terminology and examples:

Building blocks: One set has elements, which can be discrete or continuous values.

Instance 1: $DCBA$, Instance 2: $ADBCA$

Figure 2.3: Two instances, each being a sequence.

No order in elements: We focus on discrete values for elements. Elements have no order, and so for example, in Fig. 2.2, {A, B, C, D} and {D, C, B, A} are the same.

No fixed size of elements: Each instance has its own size. We can say that sets are more general than vectors, and vectors are a special case of sets, in the sense that the size and order of features in sets are both fixed in vectors.

Example: Market basket: One typical example is *market basket*, which is a set of items bought by one customer at a department store, a convenience store or an E-commerce site per time. By focusing on a certain number of items, one set can be a vector. In general the number of items is always very large, while each user buys only a few number of items, which makes the data very *sparse*. In fact elements of a matrix of users vs. items are mostly missing.

2.3.3 Sequences and Strings

When elements in one set are ordered, the set becomes a *sequence*. In particular elements are a finite number of letters, such as the alphabet, the sequence is a *string* (see below in some more detail). The size of elements in a set, i.e. the length of a sequence, is changeable over different instances.

Thus again the sequence is a special case of a set, while the length of each sequence is not fixed, meaning that a vector is a further special case of a sequence. Fig. 2.3 shows an illustrative example of sequences. Below we describe the definition and terminology of sequences.

Discrete element for sequences: Discrete elements of sequences are called *letters* or *characters*. The set of letters is called the *alphabet*. A sequence of letters is called a string.

Subsequence and substring: A consecutive part of a sequence is called a *subsequence*. Also a consecutive part of a string is called a *substring*. For example, in Instance 1 of Fig. 2.3, DC and DCB are substrings, while DCA, i.e. the first, second and fourth letters, is not a substring.

Example 1: natural language: Typical sequence examples are natural language, which are the main data of computational linguistics, natural language processing and speech recognition, etc., which are classical applications of machine learning.

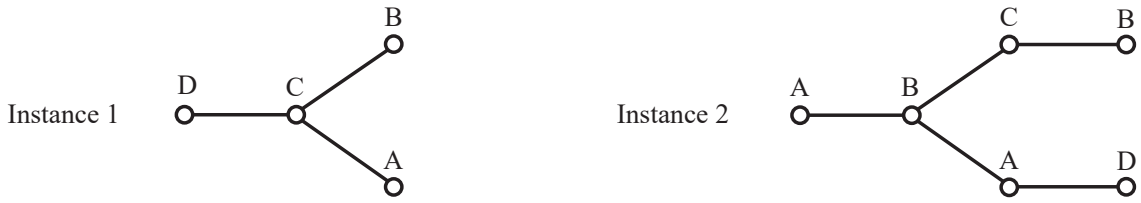

Figure 2.4: Two instances, each being a tree.

Example 2: gene sequence: A nucleic acid sequence can be a string of four letters (corresponding to four types of nucleic acids). Similarly an amino acid sequence can be a string of twenty letters (corresponding to twenty amino acids). The length of one sequence/string can be different.

2.3.4 Trees

Each instance is a tree, and so the input data is a set of trees. Description of trees can be easier if graphs are already defined, and so suppose that graphs are already defined, trees are graphs without any cycling edges (or cycles).

Fig. 2.4 shows illustrative examples of trees, where Instance 1 has a branch from C to B and also to A, and similarly Instance 2 has a branch from B to C and A. Here are more terminology and features of trees:

Building blocks: A tree consists of nodes and edges.

Cycles: No cycles of edges in trees. This discriminates trees from graphs.

Labels: A tree with labels is called a *labeled tree*. For example, in Fig. 2.4, A, B, C and D are labels. We consider labeled trees.

Root: In general, any node of a tree can be a *root*, while a tree is called a *rooted tree*, if one node of the tree is fixed as the root. For example, D of Instance 1 is fixed as the root, and A of Instance 2 is fixed as the root. We focus on rooted trees, and so we consider *labeled rooted trees*.

Leaves and internal nodes: In the rooted tree, except the root, we regard all nodes with only one edge as *leaves*. Nodes except leaves and the root are called *internal nodes*.

Ordered tree: For a rooted tree, nodes can be ordered from the root to leaves, and the tree is an *ordered tree*. Ordered trees are generated by allowing branches in sequences. In other words, sequences are a special case of ordered trees, by not allowing any branches.

Parent-children: In the rooted tree, for two nodes connected by an edge, the node closer to the root is called *parent* and the other node is called a *child*. Comparing with sequences, a feature of trees is a parent can have more than one children, while in sequences (and strings), a parent can have only one. Also the root cannot have a parent. In ordered trees, the parent to child direction is regarded as the order.

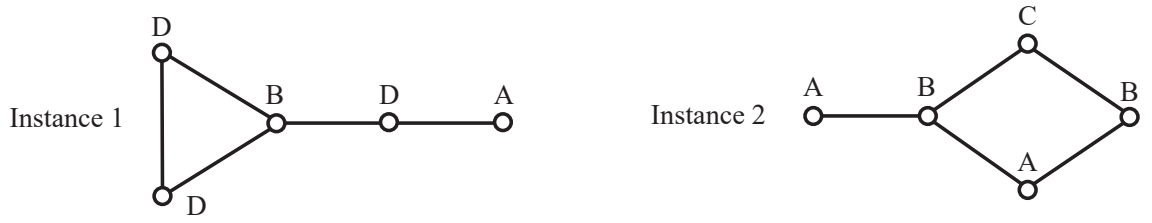

Figure 2.5: Two instances, each being a graph.

Siblings, ancestors and descendants: Nodes with the same parent are called *siblings*. For a child, its parent and any node of the parent side are all *ancestors*. Similarly for a parent, its children and any node of the children side are all *descendants*.

Depth: The number of edges from the root is called the *depth*. Usually the depth of the root is zero. The nodes with the same depth are called nodes at the same *layer*. On the other hand, the number of edges from a leaf is called the *height*. Usually the height of an leaf is zero.

Direction of edge: In ordered trees, usually the direction is from the root to leaves.

Subtree: We explain regular definition of *subtree* below, while in frequent subtree mining, subtree means any connected part of a tree. This definition is different from the regular definition (see Section 6.3 for more detail). In fact the regular definition is based on the definition of subsequence in sequences/strings, while the above definition (any part of a tree) is derived from the definition of subgraph in graphs.

A subtree, a part of a tree, has some node (of the original tree) as the root (of this subtree) and also all nodes and edges of the leaf side of the root. For example, in Instance 2 of Fig. 2.4, if the root is the most left-hand side A, CB and AD of the most right-hand side can be subtrees, while BAD cannot be a subtree, because if B is specified as the root of a subtree, the subtree must be B(AD)CB.

Example: glycan (carbohydrate sugar chain): Building blocks of glycans are around 10 to 15 types of monosaccharides, which can be letters (labels). Glycan is generated by the connection (binding) of these monosaccharides, and the connection allows branches, resulting in trees with labels of monosaccharides. Glycans will be explained more in Section 6.4.

2.3.5 Graphs

One instance is a graph and input data are a set of graphs. Fig. 2.5 shows illustrative examples of graphs. Comparing with trees, one important feature of graphs is that graphs have cycles. For example, in Instance 1 of Fig. 2.5, two D

and B are connected each other in the left-hand side, which turns into a cycle. This is not allowed in a tree, and trees are a special case of graphs. Below are features, terminology and examples of graphs:

Building blocks: A graph consists of nodes and edges connecting nodes.

Degree: The number of edges connecting to one node is called the *degree* of the node.

Labels: A graph with labels is called a *labeled graph*. Instances in Fig. 2.5 are two labeled graphs.

Direction of edges: If edges have some direction, the graph is called a *directed graph*, while if no directions, the graph is called an *undirected graph*.

Cycles: In graphs, any connection is allowed, and so it is possible to start with some node and come back to the same node along with edges. This is called a *cycle*. Also the cycle allows to go from a node to another node by using more than one routes (edge connections). Again graphs with no cycles are trees, meaning that there is always only one route in a tree if going from one node to another.

Subgraph: A part of a graph is called a *subgraph*. If all nodes and edges are connected each other in a subgraph, the subgraphs is called a *connected subgraph*. We consider only connected subgraphs as subgraphs.

Spanning tree: A tree with all nodes of a undirected graph is called a *spanning tree*. This is an important concept to deal with a graph efficiently.

Isomorphism: If two graphs have the same structure in terms of nodes and edge connectivity, they are called *isomorphic*.

Example: chemical structure of chemical compounds: The types of (physical) elements of chemical compounds are limited and so can be labels. Then the chemical structure (called a *molecular graph*) of a chemical compound can be regarded as a labeled graph.

2.3.6 Nodes in a Graph

Input data is a graph, and one instance is a node in the graph. Fig. 2.6 shows a simple, illustrative example. Nodes are instances, meaning that nodes are all unique (Note that this does not mean that labels of nodes are all different). Also all nodes in one graph is a set of all instances.

Example 1: social network: A social network is a graph with nodes for individuals and edges for some relationship between the individuals connected by the corresponding edges. Note that nodes are all unique, and they can be assigned by a limited number of labels, such as a male or a female.

Example 2: gene regulatory network: A gene regulatory network is a collection of molecular-level biological knowledge on gene regulations, such as that gene A is regulated by gene C. Regarding gene regulations as binary relations, i.e. two nodes connected by one edge, a graph with all binary relations of gene regulations is a gene regulatory network, where nodes are all unique genes. Again in this case also, genes can be with a limited number of labels, such as gene functions.

2.3.7 Notes on Data Types

Structured and Semi-structured Data

The most typical data in machine learning, i.e. vectors, are called *structured data*, while others are called *semi-structured data*. We think that the term *structured* data are derived from the fact that vectors can be organized well into a matrix with lows for instances and columns for features. However, sometimes even in machine learning and data mining publications, these two terms, i.e. structured and semi-structured data, are used in different ways, such as structured data for graphs. Thus you can be careful when you come across the term "structured data".

Structured data or vectors were long-time the major data in machine learning, while semi-structured data are relatively new in machine learning, except for sequences and strings (which have been used in applications of machine learning, such as speech recognition, natural language processing and bioinformatics).

Inclusion Relations among Data Types

Let $\mathcal{S}_V$, $\mathcal{S}_S$, $\mathcal{S}_Q$, $\mathcal{S}_T$ and $\mathcal{S}_G$ be all possible data of vectors, sets, sequences, trees and graphs, respectively.

If elements of a set are ordered, the elements become a sequence (or a string), and so sequences are part of sets. Also the size (length) of elements in a set (sequence) is fixed over all instances, vectors are part of sets (sequences). This observation leads to the following inclusion relations:

$$\mathcal{S}_S \supseteq \mathcal{S}_Q \supseteq \mathcal{S}_V \tag{2.1}$$

Trees are graphs without cycles, indicating that trees are a special case of graphs. Ordered trees are a subset of trees. Sequences are ordered trees without any branches, meaning that sequences are a special case of ordered trees. Thus in summary the following relations come out:

$$\mathcal{S}_G \supseteq \mathcal{S}_T \supseteq \mathcal{S}_{OT} \supseteq \mathcal{S}_Q, \tag{2.2}$$

where $\mathcal{S}_{OT}$ is all data of ordered trees.

Data Transformation

In the above relations, when one data type, say A, can be defined by some constraint on another data type, say B, if the constraint is removed, A becomes B.

For example, if the order of elements in sequences is removed, the sequences can be sets. More practically, two sequences, ATCG and ACGT, can be the same set {A, C, G, T} if we remove the order of letters in sequences,

In general however this type of transformation are not performed, because clearly this transformation loses some important information. For example, in the above toy example, originally ATCG and ACGT are different, and the order of letters is definitely good information to discriminate them.

Thus simply speaking some appropriate machine learning method should be developed for each data type.

Machine Learning for Different Data Types

However machine learning algorithms for some data type with less constraints can be applied to another data type with more constraints rather easily sometimes. On the other hand, applying some algorithm with more constraints to those with less constraints is not easy or impossible. For example, an algorithm for graphs can be applied to ordered trees or sequences just by considering the order of nodes of simpler graphs, while an algorithm for trees or sequences cannot be applied to graphs so easily.

Also even if one algorithm can be applied to another data type, like that an algorithm for graphs can be applied to sequences, this would never be the best, because the algorithm for sequences can be designed as a more efficient one usually. Thus again, thinking about the method most proper for each data type would be an important aspect of machine learning and data mining.

2.4 Structure of This Book

Machine learning and data mining methods/ algorithms are summarized for each of the above six data types, and so there are six chapters on vectors, sets, sequences, trees, graphs and nodes on a graph. Then after the six chapters, we have one chapter which focuses on more than one data types, which are called by many ways in machine learning, such as *data fusion*, *multiview learning*, etc. On the other hand, we call all methods for data integration and machine learning *data-integrative machine learning* methods.