This chapter is from *Social Media Mining: An Introduction*.
By Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu.
Cambridge University Press, 2014. Draft version: April 20, 2014.
Complete Draft and Slides Available at: http://dmml.asu.edu/smm

Chapter 5

Data Mining Essentials

Mountains of raw data are generated daily by individuals on social media. Around 6 billion photos are uploaded monthly to Facebook, the blogosphere doubles every five months, 72 hours of video are uploaded every minute to YouTube, and there are more than 400 million daily tweets on Twitter. With this unprecedented rate of content generation, individuals are easily overwhelmed with data and find it difficult to discover content that is relevant to their interests. To overcome these challenges, we need tools that can analyze these massive unprocessed sources of data (i.e., *raw data*) and extract useful patterns from them. Examples of useful patterns in social media are those that describe online purchasing habits or individuals' website visit duration. *Data mining* provides the necessary tools for discovering patterns in data. This chapter outlines the general process for analyzing social media data and ways to use data mining algorithms in this process to extract actionable patterns from raw data.

The process of extracting useful patterns from raw data is known as *Knowledge discovery in databases (KDD)*. It is illustrated in Figure 5.1. The KDD process takes raw data as input and provides statistically significant patterns found in the data (i.e., *knowledge*) as output. From the raw data, a subset is selected for processing and is denoted as *target data*. Target data is *preprocessed* to make it ready for analysis using data mining algorithm. Data mining is then performed on the preprocessed (and transformed) data to extract interesting patterns. The patterns are *evaluated* to ensure their validity and soundness and *interpreted* to provide insights into the data.

Knowledge Discovery in Databases (KDD)

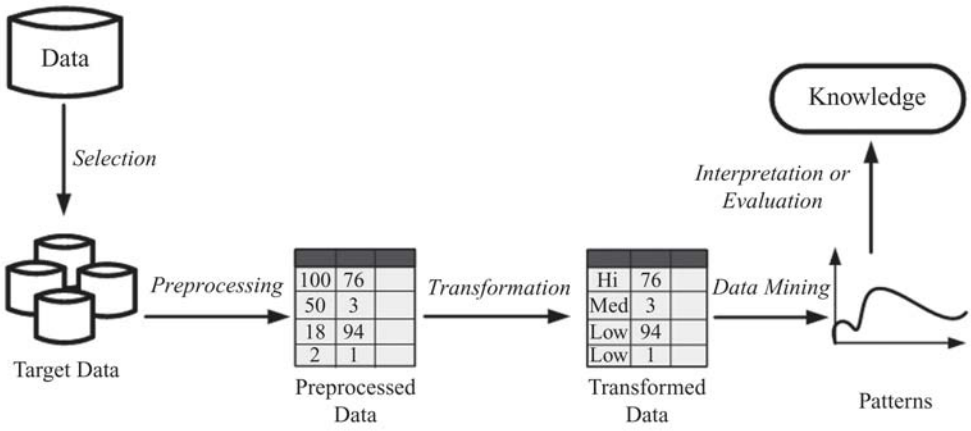


Figure 5.1: Knowledge Discovery in Databases (KDD) process.

In social media mining, the raw data is the content generated by individuals, and the knowledge encompasses the interesting patterns observed in this data. For example, for an online book seller, the raw data is the list of books individuals buy, and an interesting pattern could describe books that individuals often buy.

To analyze social media, we can either collect this raw data or use available repositories that host collected data from social media sites.[1] When collecting data, we can either use APIs provided by social media sites for data collection or scrape the information from those sites. In either case, these sites are often networks of individuals where one can perform graph traversal algorithms to collect information from them. In other words, we can start collecting information from a subset of nodes on a social network, subsequently collect information from their neighbors, and so on. The data collected this way needs to be represented in a unified format for analysis. For instance, consider a set of tweets in which we are looking for common patterns. To find patterns in these tweets, they need to be first represented using a consistent data format. In the next section, we discuss data, its representation, and its types.

[1]See [310] for a repository of network data.

5.1 Data

In the KDD process, data is represented in a *tabular* format. Consider the example of predicting whether an individual who visits an online book seller is going to buy a specific book. This prediction can be performed by analyzing the individual's interests and previous purchase history. For instance, when John has spent a lot of money on the site, has bought similar books, and visits the site frequently, it is likely for John to buy that specific book. John is an example of an *instance*. Instances are also called *points*, *data points*, or *observations*. A *dataset* consists of one or more instances:

Instance, Point, Data Point, or Observation

Attributes				**Class**
Name	Money Spent	Bought Similar	Visits	Will Buy
`John`	`High`	`Yes`	`Frequently`	`?`
`Mary`	`High`	`Yes`	`Rarely`	`Yes`

A dataset is represented using a set of *features*, and an instance is represented using values assigned to these features. Features are also known as *measurements* or *attributes*. In this example, the features are `Name`, `Money Spent`, `Bought Similar`, and `Visits`; feature values for the first instance are `John`, `High`, `Yes`, and `Frequently`. Given the feature values for one instance, one tries to predict its *class* (or *class attribute*) value. In our example, the class attribute is `Will Buy`, and our class value prediction for first instance is `Yes`. An instance such as John in which the class attribute value is unknown is called an *unlabeled* instance. Similarly, a *labeled* instance is an instance in which the class attribute value in known. Mary in this dataset represents a labeled instance. The class attribute is optional in a dataset and is only necessary for prediction or classification purposes. One can have a dataset in which no class attribute is present, such as a list of customers and their characteristics.

Features, Measurements, or Attributes

Labeled and Unlabeled

There are different types of features based on the characteristics of the feature and the values they can take. For instance, `Money Spent` can be represented using numeric values, such as `$25`. In that case, we have a *continuous feature*, whereas in our example it is a *discrete* feature, which can take a number of ordered values: {`High, Normal, Low`}.

Different types of features were first introduced by psychologist Stanley Smith Stevens [265] as "levels of measurement" in the theory of scales. He

Levels of Measurement

claimed that there are four types of features. For each feature type, there exists a set of permissible operations (statistics) using the feature values and transformations that are allowed.

- **Nominal (categorical)**. These features take values that are often represented as strings. For instance, a customer's name is a nominal feature. In general, a few statistics can be computed on nominal features. Examples are the chi-square statistic (χ^2) and the *mode* (most common feature value). For example, one can find the most common first name among customers. The only possible transformation on the data is comparison. For example, we can check whether our customer's name is John or not. Nominal feature values are often presented in a set format.

- **Ordinal**. Ordinal features lay data on an ordinal scale. In other words, the feature values have an intrinsic order to them. In our example, `Money Spent` is an ordinal feature because a `High` value for `Money Spent` is more than a `Low` one.

- **Interval**. In interval features, in addition to their intrinsic ordering, differences are meaningful whereas ratios are meaningless. For interval features, addition and subtraction are allowed, whereas multiplications and division are not. Consider two time readings: 6:16 PM and 3:08 PM. The difference between these two time readings is meaningful (3 hours and 8 minutes); however, there is no meaning to $\frac{6:16\ \ PM}{3:08\ \ PM} \neq 2$.

- **Ratio**. Ratio features, as the name suggests, add the additional properties of multiplication and division. An individual's income is an example of a ratio feature where not only differences and additions are meaningful but ratios also have meaning (e.g., an individual's income can be twice as much as John's income).

In social media, individuals generate many types of nontabular data, such as text, voice, or video. These types of data are first converted to tabular data and then processed using data mining algorithms. For instance, voice can be converted to feature values using approximation techniques such as the fast Fourier transform (FFT) and then processed using data mining algorithms. To convert text into the tabular format, we can use a

process denoted as *vectorization*. A variety of vectorization methods exist. A well-known method for vectorization is the *vector-space model* introduced by Salton, Wong, and Yang [243]. Vectorization

Vector Space Model

In the vector space model, we are given a set of documents D. Each document is a set of words. The goal is to convert these textual documents to [feature] vectors. We can represent document i with vector d_i,

$$d_i = (w_{1,i}, w_{2,i}, \ldots, w_{N,i}), \tag{5.1}$$

where $w_{j,i}$ represents the weight for word j that occurs in document i and N is the number of words used for vectorization.[2] To compute $w_{j,i}$, we can set it to 1 when the word j exists in document i and 0 when it does not. We can also set it to the number of times the word j is observed in document i. A more generalized approach is to use the *term frequency-inverse document frequency (TF-IDF)* weighting scheme. In the TF-IDF scheme, $w_{j,i}$ is calculated as

$$w_{j,i} = tf_{j,i} \times idf_j, \tag{5.2}$$

where $tf_{j,i}$ is the frequency of word j in document i. idf_j is the inverse frequency of word j across all documents, TF-IDF

$$idf_j = \log_2 \frac{|D|}{|\{\text{document} \in D \mid j \in \text{document}\}|}, \tag{5.3}$$

which is the logarithm of the total number of documents divided by the number of documents that contain word j. TF-IDF assigns higher weights to words that are less frequent across documents and, at the same time, have higher frequencies within the document they are used. This guarantees that words with high TF-IDF values can be used as representative examples of the documents they belong to and also, that stop words, such as "the," which are common in all documents, are assigned smaller weights.

Example 5.1. *Consider the words "apple" and "orange" that appear 10 and 20 times in document d_1. Let $|D| = 20$ and assume the word "apple" only appears in*

[2] One can use all unique words in all documents (D) or a more frequent subset of words in the documents for vectorization.

document d_1 and the word "orange" appears in all 20 documents. Then, TF-IDF values for "apple" and "orange" in document d_1 are

$$tf - idf(\text{"apple"}, d_1) = 10 \times \log_2 \frac{20}{1} = 43.22, \tag{5.4}$$

$$tf - idf(\text{"orange"}, d_1) = 20 \times \log_2 \frac{20}{20} = 0. \tag{5.5}$$

Example 5.2. *Consider the following three documents:*

$$d_1 = \text{"social media mining"} \tag{5.6}$$
$$d_2 = \text{"social media data"} \tag{5.7}$$
$$d_3 = \text{"financial market data"} \tag{5.8}$$

The tf values are as follows:

	social	*media*	*mining*	*data*	*financial*	*market*
d_1	*1*	*1*	*1*	*0*	*0*	*0*
d_2	*1*	*1*	*0*	*1*	*0*	*0*
d_3	*0*	*0*	*0*	*1*	*1*	*1*

The idf values are

$$idf_{\text{social}} = \log_2(3/2) = 0.584 \tag{5.9}$$
$$idf_{\text{media}} = \log_2(3/2) = 0.584 \tag{5.10}$$
$$idf_{\text{mining}} = \log_2(3/1) = 1.584 \tag{5.11}$$
$$idf_{\text{data}} = \log_2(3/2) = 0.584 \tag{5.12}$$
$$idf_{\text{financial}} = \log_2(3/1) = 1.584 \tag{5.13}$$
$$idf_{\text{market}} = \log_2(3/1) = 1.584. \tag{5.14}$$

The TF-IDF values can be computed by multiplying tf values with the idf values:

	social	*media*	*mining*	*data*	*financial*	*market*
d_1	*0.584*	*0.584*	*1.584*	*0*	*0*	*0*
d_2	*0.584*	*0.584*	*0*	*0.584*	*0*	*0*
d_3	*0*	*0*	*0*	*0.584*	*1.584*	*1.584*

After vectorization, documents are converted to vectors, and common data mining algorithms can be applied. However, before that can occur, the quality of data needs to be verified.

5.1.1 Data Quality

When preparing data for use in data mining algorithms, the following four data quality aspects need to be verified:

1. **Noise** is the distortion of the data. This distortion needs to be removed or its adverse effect alleviated before running data mining algorithms because it may adversely affect the performance of the algorithms. Many filtering algorithms are effective in combating noise effects.

2. **Outliers** are instances that are considerably different from other instances in the dataset. Consider an experiment that measures the average number of followers of users on Twitter. A celebrity with many followers can easily distort the average number of followers per individuals. Since the celebrities are outliers, they need to be removed from the set of individuals to accurately measure the average number of followers. Note that in special cases, outliers represent useful patterns, and the decision to removing them depends on the context of the data mining problem.

3. **Missing Values** are feature values that are missing in instances. For example, individuals may avoid reporting profile information on social media sites, such as their age, location, or hobbies. To solve this problem, we can (1) remove instances that have missing values, (2) estimate missing values (e.g., replacing them with the most common value), or (3) ignore missing values when running data mining algorithms.

4. **Duplicate data** occurs when there are multiple instances with the exact same feature values. Duplicate blog posts, duplicate tweets, or profiles on social media sites with duplicate information are all instances of this phenomenon. Depending on the context, these instances can either be removed or kept. For example, when instances need to be unique, duplicate instances should be removed.

After these quality checks are performed, the next step is preprocessing or transformation to prepare the data for mining.

5.2 Data Preprocessing

Often, the data provided for data mining is not immediately ready. Data preprocessing (and transformation in Figure 5.1) prepares the data for mining. Typical data preprocessing tasks are as follows:

1. **Aggregation**. This task is performed when multiple features need to be combined into a single one or when the scale of the features change. For instance, when storing image dimensions for a social media website, one can store by image width and height or equivalently store by image area (width $\times$ height). Storing image area saves storage space and tends to reduce data variance; hence, the data has higher resistance to distortion and noise.

2. **Discretization.** Consider a continuous feature such as `money spent` in our previous example. This feature can be converted into discrete values – `High`, `Normal`, and `Low` – by mapping different ranges to different discrete values. The process of converting continuous features to discrete ones and deciding the continuous range that is being assigned to a discrete value is called *discretization*.

3. **Feature Selection**. Often, not all features gathered are useful. Some may be irrelevant, or there may be a lack of computational power to make use of all the features, among many other reasons. In these cases, a subset of features are selected that could ideally enhance the performance of the selected data mining algorithm. In our example, customer's name is an irrelevant feature to the value of the class attribute and the task of predicting whether the individual will buy the given book or not.

4. **Feature Extraction**. In contrast to feature selection, *feature extraction* converts the current set of features to a new set of features that can perform the data mining task better. A transformation is performed on the data, and a new set of features is extracted. The example we provided for aggregation is also an example of feature extraction where a new feature (area) is constructed from two other features (width and height).

5. **Sampling**. Often, processing the whole dataset is expensive. With the massive growth of social media, processing large streams of data

is nearly impossible. This motivates the need for *sampling*. In sampling, a small random subset of instances are selected and processed instead of the whole data. The selection process should guarantee that the sample is representative of the distribution that governs the data, thereby ensuring that results obtained on the sample are close to ones obtained on the whole dataset. The following are three major sampling techniques:

- **Random sampling**. In random sampling, instances are selected uniformly from the dataset. In other words, in a dataset of size n, all instances have equal probability $\frac{1}{n}$ of being selected. Note that other probability distributions can also be used to sample the dataset, and the distribution can be different from uniform.
- **Sampling with or without replacement**. In sampling with replacement, an instance can be selected multiple times in the sample. In sampling without replacement, instances are removed from the selection pool once selected.
- **Stratified sampling**. In stratified sampling, the dataset is first partitioned into multiple bins; then a fixed number of instances are selected from each bin using random sampling. This technique is particularly useful when the dataset does not have a uniform distribution for class attribute values (i.e., *class imbalance*). For instance, consider a set of 10 females and 5 males. A sample of 5 females and 5 males can be selected using stratified sampling from this set.

In social media, a large amount of information is represented in network form. These networks can be sampled by selecting a subset of their nodes and edges. These nodes and edges can be selected using the aforementioned sampling methods. We can also sample these networks by starting with a small set of nodes (*seed nodes*) and sample

(a) the connected components they belong to;

(b) the set of nodes (and edges) connected to them directly; or

(c) the set of nodes and edges that are within n-hop distance from them.

After preprocessing is performed, the data is ready to be mined. Next, we discuss two general categories of data mining algorithms and how each can be evaluated.

5.3 Data Mining Algorithms

Data mining algorithms can be divided into several categories. Here, we discuss two well-established categories: *supervised learning* and *unsupervised learning*. In supervised learning, the class attribute exists, and the task is to predict the class attribute value. Our previous example of predicting the class attribute "will buy" is an example of supervised learning. In unsupervised learning, the dataset has no class attribute, and our task is to find similar instances in the dataset and group them. By grouping these similar instances, one can find significant patterns in a dataset. For example, unsupervised learning can be used to identify events on Twitter, because the frequency of tweeting is different for various events. By using unsupervised learning, tweets can be grouped based on the times at which they appear and hence, identify the tweets' corresponding real-world events. Other categories of data mining algorithms exist; interested readers can refer to the bibliographic notes for pointers to these categories.

5.4 Supervised Learning

The first category of algorithms, supervised learning algorithms, are those for which the class attribute values for the dataset are known before running the algorithm. This data is called *labeled* data or *training* data. Instances in this set are tuples in the format $(\mathbf{x}, y)$, where $\mathbf{x}$ is a vector and y is the class attribute, commonly a scalar. Supervised learning builds a model that maps $\mathbf{x}$ to y. Roughly, our task is to find a mapping $m(.)$ such that $m(\mathbf{x}) = y$. We are also given an unlabeled dataset or *test* dataset, in which instances are in the form $(\mathbf{x}, ?)$ and y values are unknown. Given $m(.)$ learned from training data and $\mathbf{x}$ of an unlabeled instance, we can compute $m(\mathbf{x})$, the result of which is prediction of the label for the unlabeled instance.

Consider the task of detecting spam emails. A set of emails is given where users have manually identified spam versus non-spam (training

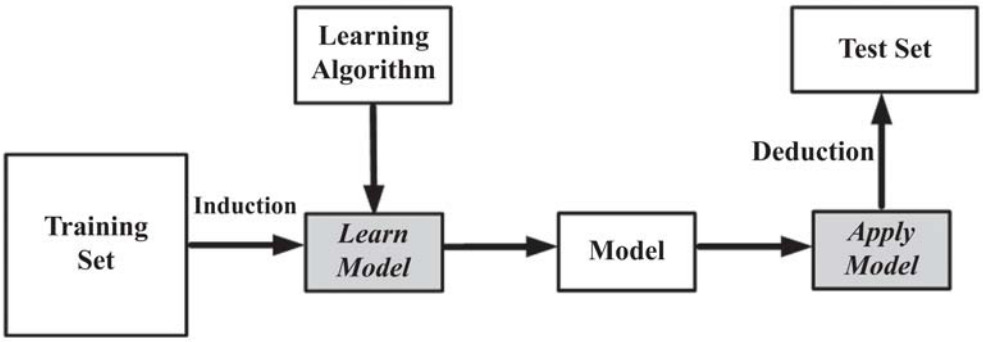


Figure 5.2: Supervised Learning.

data). Our task is to use a set of features such as words in the email ($\mathbf{x}$) to identify the spam/non-spam status (y) of unlabeled emails (test data). In this case, $y = \{spam, non\text{-}spam\}$.

Supervised learning can be divided into *classification* and *regression*. When the class attribute is discrete, it is called classification; when the class attribute is continuous, it is regression. We introduce classification methods such as *decision tree learning*, *naive Bayes classifier*, *k-nearest neighbor classifier*, and *classification with network information* and regression methods such as *linear regression* and *logistic regression*. We also introduce how supervised learning algorithms are evaluated. Before we delve into supervised learning techniques, we briefly discuss the systematic process of a supervised learning algorithm.

This process is depicted in Figure 5.2. It starts with a training set (i.e., labeled data) where both features and labels (class attribute values) are known. A supervised learning algorithm is run on the training set in a process known as *induction*. In the induction process, the *model* is generated. The model maps the feature values to the class attribute values. The model is used on a *test set* in which the class attribute value is unknown to predict these unknown class attribute values (*deduction* process).

5.4.1 Decision Tree Learning

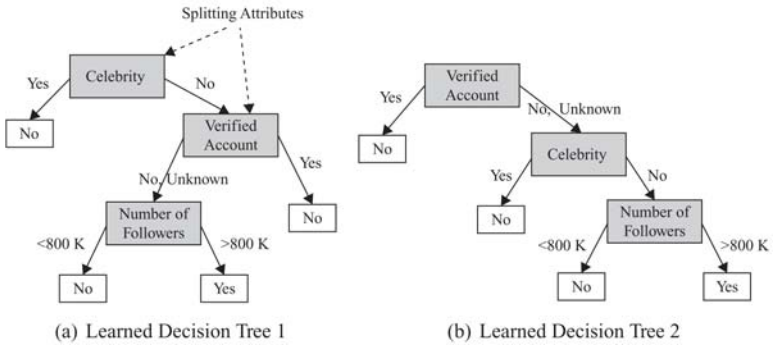
Consider the dataset shown in Table 5.1. The last attribute represents the class attribute, and the other attributes represent the features. In decision tree classification, a decision tree is learned from the training dataset, and

Table 5.1: A Sample Dataset. In this dataset, features are characteristics of individuals on Twitter, and the class attribute denotes whether they are influential or not

ID	Celebrity	Verified Account	# Followers	Influential?
1	Yes	No	1.25 M	No
2	No	Yes	1 M	No
3	No	Yes	600 K	No
4	Yes	Unknown	2.2 M	No
5	No	No	850 K	Yes
6	No	Yes	750 K	No
7	No	No	900 K	Yes
8	No	No	700 K	No
9	Yes	Yes	1.2 M	No
10	No	Unknown	950 K	Yes

that tree is later used to predict the class attribute value for instances in the test dataset. As an example, two learned decision trees from the dataset shown in Table 5.1 are provided in Figure 5.3. As shown in this figure, multiple decision trees can be learned from the same dataset, and these decision trees can both correctly predict the class attribute values for all instances in the dataset. Construction of decision trees is based on heuristics, as different heuristics generate different decision trees from the same dataset.

Decision trees classify examples based on their feature values. Each



(a) Learned Decision Tree 1 (b) Learned Decision Tree 2

Figure 5.3: Decision Trees Learned from Data Provided in Table 5.1.

nonleaf node in a decision tree represents a feature, and each branch represents a value that the feature can take. Instances are classified by following a path that starts at the root node and ends at a leaf by following branches based on instance feature values. The value of the leaf determines the class attribute value predicted for the instance (see Figure 5.3).

Decision trees are constructed recursively from training data using a top-down greedy approach in which features are sequentially selected. In Figure 5.3(a), the feature selected for the root node is `Celebrity`. After selecting a feature for each node, based on its values, different branches are created: For Figure 5.3(a), since the `Celebrity` feature can only take either `Yes` or `No`, two branches are created: one labeled `Yes` and one labeled `No`. The training set is then partitioned into subsets based on the feature values, each of which fall under the respective feature value branch; the process is continued for these subsets and other nodes. In Figure 5.3(a), instances 1, 4, and 9 from Table 5.1 represent the subset that falls under the `Celebrity=Yes` branch, and the other instances represent the subset that falls under the `Celebrity=No` branch.

When selecting features, we prefer features that partition the set of instances into subsets that are more *pure*. A pure subset has instances that all have the same class attribute value. In Figure 5.3(a), the instances that fall under the left branch of the root node (`Celebrity=Yes`) form a pure subset in which all instances have the same class attribute value `Influential?=No`. When reaching pure subsets under a branch, the decision tree construction process no longer partitions the subset, creates a leaf under the branch, and assigns the class attribute value for subset instances as the leaf's predicted class attribute value. In Figure 5.3(a), the instances that fall under the right branch of the root node form an impure dataset; therefore, further branching is required to reach pure subsets. Purity of subsets can be determined with different measures. A common measure of purity is *entropy*. Over a subset of training instances, T, with a binary class attribute (values $\in \{+, -\}$), the entropy of T is defined as

$$entropy(T) = -p_+ \log p_+ - p_- \log p_-, \tag{5.15}$$

where p_+ is the proportion of instances with + class attribute value in T and p_- is the proportion of instances with – class attribute value.

Example 5.3. *Assume that there is a subset T, containing 10 instances. Seven instances have a positive class attribute value, and three instances have a negative*

class attribute value $[7+, 3-]$. *The entropy for subset* T *is*

$$entropy(T) = -\frac{7}{10}\log\frac{7}{10} - \frac{3}{10}\log\frac{3}{10} = 0.881. \tag{5.16}$$

Note that if the number of positive and negative instances in the set are equal ($p_+ = p_- = 0.5$)*, then the entropy is 1.*

In a pure subset, all instances have the same class attribute value and the entropy is 0. If the subset being measured contains an unequal number of positive and negative instances, the entropy is between 0 and 1.

5.4.2 Naive Bayes Classifier

Among many methods that use the Bayes theorem, the naive Bayes classifier (NBC) is the simplest. Given two random variables X and Y, Bayes theorem states that

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}. \tag{5.17}$$

In NBC, Y represents the class variable and X represents the instance features. Let X be $(x_1, x_2, x_3, \ldots, x_m)$, where x_i represents the value of feature i. Let $\{y_1, y_2, \ldots, y_n\}$ represent the values the class attribute Y can take. Then, the class attribute value of instance X can be calculated by measuring

$$\arg\max_{y_i} P(y_i|X). \tag{5.18}$$

Based on the Bayes theorem,

$$P(y_i|X) = \frac{P(X|y_i)P(y_i)}{P(X)}. \tag{5.19}$$

Note that $P(X)$ is constant and independent of y_i, so we can ignore the denominator of Equation 5.19 when maximizing Equation 5.18. The NBC also assumes conditional independence to make the calculations easier; that is, given the class attribute value, other feature attributes become conditionally independent. This condition, though unrealistic, performs well in practice and greatly simplifies calculation.

$$P(X|y_i) = \Pi_{j=1}^{m} P(x_j|y_i). \tag{5.20}$$

Table 5.2: Naive Bayes Classifier (NBC) Toy Dataset

No.	Outlook (O)	Temperature (T)	Humidity (H)	Play Golf (PG)
1	sunny	hot	high	N
2	sunny	mild	high	N
3	overcast	hot	high	Y
4	rain	mild	high	Y
5	sunny	cool	normal	Y
6	rain	cool	normal	N
7	overcast	cool	normal	Y
8	sunny	mild	high	?

Substituting $P(X|y_i)$ from Equation 5.20 in Equation 5.19, we get

$$P(y_i|X) = \frac{(\Pi_{j=1}^{m} P(x_j|y_i))P(y_i)}{P(X)}. \tag{5.21}$$

We clarify how the naive Bayes classifier works with an example.

Example 5.4. *Consider the dataset in Table 5.2.*

We predict the label for instance 8 (i_8) using the naive Bayes classifer and the given dataset. We have

$$\begin{aligned} P(PG = Y|i_8) &= \frac{P(i_8|PG = Y)P(PG = Y)}{P(i_8)} \\ &= P(O = Sunny, T = mild, H = high|PG = Y) \\ &\quad \times \frac{P(PG = Y)}{P(i_8)} \\ &= P(O = Sunny|PG = Y) \times P(T = mild|PG = Y) \\ &\quad \times P(H = high|PG = Y) \times \frac{P(PG = Y)}{P(i_8)} \\ &= \frac{1}{4} \times \frac{1}{4} \times \frac{2}{4} \times \frac{\frac{4}{7}}{P(i_8)} = \frac{1}{28P(i_8)}. \end{aligned} \tag{5.22}$$

Similarly,

$$P(PG = N|i_8) = \frac{P(i_8|PG = N)P(PG = N)}{P(i_8)}$$

Algorithm 5.1 k-Nearest Neighbor Classifier

Require: Instance i, A Dataset of Real-Value Attributes, k (number of neighbors), distance measure d

1: **return** Class label for instance i
2: Compute k nearest neighbors of instance i based on distance mea- sure d.
3: l = the majority class label among neighbors of instance i. If more than one majority label, select one randomly.
4: Classify instance i as class l

$$
\begin{aligned}
&= P(O = Sunny, T = mild, H = high|PG = N) \\
&\quad \times \frac{P(PG = N)}{P(i_8)} \\
&= P(O = Sunny|PG = N) \times P(T = mild|PG = N) \\
&\quad \times P(H = high|PG = N) \times \frac{P(PG = N)}{P(i_8)} \\
&= \frac{2}{3} \times \frac{1}{3} \times \frac{2}{3} \times \frac{\frac{3}{7}}{P(i_8)} = \frac{4}{63P(i_8)}. \qquad (5.23)
\end{aligned}
$$

Since $\frac{4}{63P(i_8)} > \frac{1}{28P(i_8)}$, *for instance* i_8, *and based on NBC calculations, we have Play Golf* = *N.*

5.4.3 Nearest Neighbor Classifier

As the name suggests, k-nearest neighbor or kNN uses the k nearest instances, called neighbors, to perform classification. The instance being classified is assigned the label (class attribute value) that the majority of its k neighbors are assigned. The algorithm is outlined in Algorithm 5.1. When $k = 1$, the closest neighbor's label is used as the predicted label for the instance being classified. To determine the neighbors of an instance, we need to measure its distance to all other instances based on some distance metric. Commonly, Euclidean distance is employed; however, for higher dimensional spaces, Euclidean distance becomes less meaningful and other distance measures can be used.

Example 5.5. *Consider the example depicted in Figure 5.4. As shown, depending on the value of k, different labels can be predicted for the circle. In our example,*

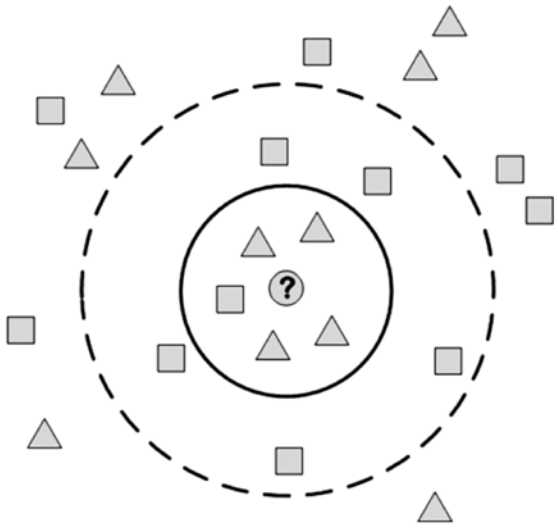


Figure 5.4: k-Nearest Neighbor Example. In this figure, our goal is to predict the label for the instance shown using a circle. When $k = 5$, the predicted label is ▲ and when $k = 9$ the predicted label is ■.

$k = 5$ and $k = 9$ generate different labels for the instance (triangle and square, respectively).

As shown in our example, an important issue with the k-nearest neighbor algorithm is the choice of k. The choice of k can easily change the label of the instance being predicted. In general, we are interested in a value of k that maximizes the performance of the learning algorithm.

5.4.4 Classification with Network Information

Consider a friendship network on social media and a product being marketed to this network. The product seller wants to know who the potential buyers are for this product. Assume we are given the network with the list of individuals who decided to buy or not buy the product. Our goal is to predict the decision for the undecided individuals. This problem can be formulated as a classification problem based on features gathered from individuals. However, in this case, we have additional friendship information that may be helpful in building more accurate classification models. This is an example of *classification with network information*.

Assume we are not given any profile information, but only connections and class labels (i.e., the individual bought/will not buy). By using the rows of the adjacency matrix of the friendship network for each node as features and the decision (e.g., buy/not buy) as a class label, we can predict the label for any unlabeled node using its connections; that is, its row in the adjacency matrix. Let $P(y_i = 1|N(v_i))$ denote the probability of node v_i having class attribute value 1 given its neighbors. Individuals' decisions are often highly influenced by their immediate neighbors. Thus, we can approximate $P(y_i = 1)$ using the neighbors of the individual by assuming that

$$P(y_i = 1) \approx P(y_i = 1|N(v_i)). \tag{5.24}$$

Weighted-Vote Relational-Neighbor Classifier

We can estimate $P(y_i = 1|N(v_i))$ via different approaches. The weighted-vote relational-neighbor (wvRN) classifier is one such approach. It estimates $P(y_i = 1|N(v_i))$ as

$$P(y_i = 1|N(v_i)) = \frac{1}{|N(v_i)|} \sum_{v_j \in N(v_i)} P(y_j = 1|N(v_j)). \tag{5.25}$$

In other words, the probability of node v_i having class attribute value 1 is the average probability of its neighbors having this class attribute value. Note that $P(y_i = 1|N(v_i))$ is *only* calculated for v_i's that are unlabeled. For node v_k, which is labeled 1, $p(y_k = 1|N(v_k)) = 1$ and the probability is never estimated. Similarly, if v_k will not buy the product, $p(y_k = 0|N(v_k)) = 1$. Since the probability of a node having class attribute value 1 depends on the probability of its neighbors having the same value, the probability of the node is affected if the probabilities of its neighbors change. Thus, the order of updating nodes can affect the estimated probabilities. In practice, one follows an order sequence for estimating node probabilities. Starting with an initial probability estimate for all unlabeled nodes and following this order, we estimate probabilities until probabilities no longer change (i.e., converge). We can assume the initial probability estimate to be $P(y_i = 1|N(v_i)) = 0.5$ for all unlabeled nodes.[3] We show how the wvRN classifier learns probabilities using the following example.

[3]Note that in our example, the class attribute can take two values; therefore, the initial guess of $P(y_i = 1|N(v_i)) = \frac{1}{2} = 0.5$ is reasonable. When a class attribute takes n values, we can set our initial guess to $P(y_i = 1|N(v_i)) = \frac{1}{n}$.

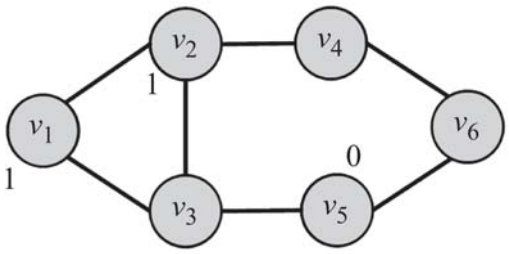


Figure 5.5: Weighted-Vote Relational-Neighbor (wvRN) Example. Labeled nodes have their class attribute values next to them. The goal is to predict labels for other nodes in the network.

Example 5.6. *Consider the network given in Figure 5.5. Labeled nodes have their class attribute values next to them. Therefore,*

$$P(y_1 = 1|N(v_1)) = 1, \tag{5.26}$$
$$P(y_2 = 1|N(v_2)) = 1, \tag{5.27}$$
$$P(y_5 = 1|N(v_5)) = 0. \tag{5.28}$$

We have three unlabeled nodes $\{v_3, v_4, v_6\}$. We choose their natural order to update their probabilities. Thus, we start with v_3:

$$\begin{aligned} &P(y_3|N(v_3)) \\ &= \frac{1}{|N(v_3)|} \sum_{v_j \in N(v_3)} P(y_j = 1|N(v_j)) \\ &= \frac{1}{3}(P(y_1 = 1|N(v_1)) + P(y_2 = 1|N(v_2)) + P(y_5 = 1|N(v_5))) \\ &= \frac{1}{3}(1 + 1 + 0) = 0.67. \end{aligned} \tag{5.29}$$

$P(y_3|N(v_3))$ does not need to be computed again because its neighbors are all labeled (thus, this probability estimation has converged). Similarly,

$$P(y_4|N(v_4)) = \frac{1}{2}(1 + 0.5) = 0.75, \tag{5.30}$$
$$P(y_6|N(v_6)) = \frac{1}{2}(0.75 + 0) = 0.38. \tag{5.31}$$

We need to recompute both $P(y_4|N(v_4))$ and $P(y_6|N(v_6))$ until convergence.

Let $P_{(t)}(y_i|N(v_i))$ denote the estimated probability after t computations. Then,

$$P_{(1)}(y_4|N(v_4)) = \frac{1}{2}(1 + 0.38) = 0.69, \tag{5.32}$$

$$P_{(1)}(y_6|N(v_6)) = \frac{1}{2}(0.69 + 0) = 0.35, \tag{5.33}$$

$$P_{(2)}(y_4|N(v_4)) = \frac{1}{2}(1 + 0.35) = 0.68, \tag{5.34}$$

$$P_{(2)}(y_6|N(v_6)) = \frac{1}{2}(0.68 + 0) = 0.34, \tag{5.35}$$

$$P_{(3)}(y_4|N(v_4)) = \frac{1}{2}(1 + 0.34) = 0.67, \tag{5.36}$$

$$P_{(3)}(y_6|N(v_6)) = \frac{1}{2}(0.67 + 0) = 0.34, \tag{5.37}$$

$$P_{(4)}(y_4|N(v_4)) = \frac{1}{2}(1 + 0.34) = 0.67, \tag{5.38}$$

$$P_{(4)}(y_6|N(v_6)) = \frac{1}{2}(0.67 + 0) = 0.34. \tag{5.39}$$

After four iterations, both probabilities converge. So, from these probabilities (Equations 5.29, 5.38, and 5.39), we can tell that nodes v_3 and v_4 will likely have class attribute value 1 and node v_6 will likely have class attribute value 0.

5.4.5 Regression

In classification, class attribute values are discrete. In regression, class attribute values are real numbers. For instance, we wish to predict the stock market value (class attribute) of a company given information about the company (features). The stock market value is continuous; therefore, regression must be used to predict it. The input to the regression method is a dataset where attributes are represented using $x_1, x_2, \ldots, x_m$ (also known as *regressors*) and class attribute is represented using Y (also known as the *dependent variable*), where the class attribute is a real number. We want to find the relation between Y and the vector $X = (x_1, x_2, \ldots, x_m)$. We discuss two basic regression techniques: linear regression and logistic regression.

Linear Regression

In linear regression, we assume that the class attribute Y has a linear relation with the regressors (feature set) X by considering a linear error ϵ. In other words,

$$Y = XW + \epsilon, \tag{5.40}$$

where W represents the vector of regression coefficients. The problem of regression can be solved by estimating W using the training dataset and its labels Y such that fitting error ϵ is minimized. A variety of methods have been introduced to solve the linear regression problem, most of which use least squares or maximum-likelihood estimation. We employ the least squares technique here. Interested readers can refer to the bibliographic notes for more detailed analyses. In the least square method, we find W using regressors X and labels Y such that the square of fitting error *epsilon* is minimized.

$$\epsilon^2 = ||\epsilon^2|| = ||Y - XW||^2. \tag{5.41}$$

To minimize ϵ, we compute the gradient and set it to zero to find the optimal W:

$$\frac{\partial ||Y - XW||^2}{\partial W} = 0. \tag{5.42}$$

We know that for any X, $||X||^2 = (X^T X)$; therefore,

$$\begin{aligned}
\frac{\partial ||Y - XW||^2}{\partial W} &= \frac{\partial (Y - XW)^T (Y - XW)}{\partial W} \\
&= \frac{\partial (Y^T - W^T X^T)(Y - XW)}{\partial W} \\
&= \frac{\partial (Y^T Y - Y^T XW - W^T X^T Y + W^T X^T XW)}{\partial W} \\
&= -2X^T Y + 2X^T XW = 0.
\end{aligned} \tag{5.43}$$

Therefore,

$$X^T Y = X^T XW. \tag{5.44}$$

Since $X^T X$ is invertible for any X, by multiplying both sides by $(X^T X)^{-1}$, we get

$$W = (X^T X)^{-1} X^T Y. \tag{5.45}$$

Alternatively, one can compute the singular value decomposition (SVD) of $X = U\Sigma V^T$:

$$\begin{aligned} W &= (X^T X)^{-1} X^T Y \\ &= (V\Sigma U^T U\Sigma V^T)^{-1} V\Sigma U^T Y \\ &= (V\Sigma^2 V^T)^{-1} V\Sigma U^T Y \\ &= V\Sigma^{-2} V^T V\Sigma U^T Y \\ &= V\Sigma^{-1} U^T Y, \end{aligned} \tag{5.46}$$

and since we can have zero singular values,

$$W = V\Sigma^{+} U^T Y, \tag{5.47}$$

where Σ^{+} is the submatrix of Σ with nonzero singular values.

Logistic Regression

Logistic regression provides a probabilistic view of regression. For simplicity, let us assume that the class attribute can only take values of 0 and 1. Formally, logistic regression finds probability p such that

$$P(Y = 1|X) = p, \tag{5.48}$$

where X is the vector of features and Y is the class attribute. We can use linear regression to approximate p. In other words, we can assume that probability p depends on X; that is,

$$p = \beta X, \tag{5.49}$$

where β is a vector of coefficients. Unfortunately, βX can take unbounded values because X can take on any value and there are no constraints on how β's are chosen. However, probability p must be in range $[0, 1]$. Since βX is unbounded, we can perform a transformation $g(.)$ on p such that it also becomes unbounded. Then, we can fit $g(p)$ to βX. One such transformation $g(.)$ for p is

$$g(p) = \ln \frac{p}{1-p}, \tag{5.50}$$

which for any p between $[0, 1]$ generates a value in range $[-\infty, +\infty]$. The function $g(.)$ is known as the *logit* function. The transformed p can be approximated using a linear function of feature vector X,

$$g(p) = \beta X. \tag{5.51}$$

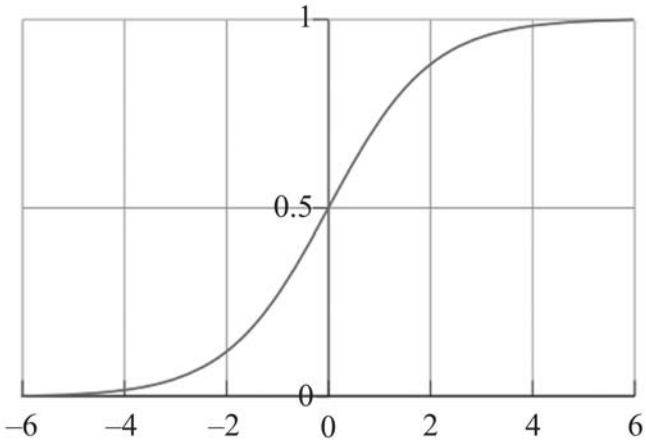


Figure 5.6: Logistic Function.

Combining Equations 5.50 and 5.51 and solving for p, we get

$$p = \frac{e^{\beta X}}{e^{\beta X} + 1} = \frac{1}{e^{-\beta X} + 1}. \tag{5.52}$$

This function is known as the logistic function and is plotted in Figure 5.6. An interesting property of this function is that, for any real value (negative to positive infinity), it will generate values between 0 and 1. In other words, it acts as a probability function.

Our task is to find β's such that $P(Y|X)$ is maximized. Unlike linear regression models, there is no closed form solution to this problem, and it is usually solved using iterative maximum likelihood methods (See Bibliographic Notes).

After β's are found, similar to the Naive Bayes Classifier (NBC), we compute the probability $P(Y|X)$ using Equation 5.52. In a situation where the class attribute takes two values, when this probability is larger than 0.5, the class attribute is predicted 1; otherwise, 0 is predicted.

5.4.6 Supervised Learning Evaluation

Supervised learning algorithms often employ a *training-testing* framework in which a training dataset (i.e., the labels are known) is used to train a model and then the model is evaluated on a test dataset. The performance

of the supervised learning algorithm is measured by how accurate it is in predicting the correct labels of the test dataset. Since the correct labels of the test dataset are unknown, in practice, the training set is divided into two parts, one used for training and the other used for testing. Unlike the original test set, for this test set the labels are known. Therefore, when testing, the labels from this test set are removed. After these labels are predicted using the model, the predicted labels are compared with the masked labels (*ground truth*). This measures how well the trained model is generalized to predict class attributes. One way of dividing the training set into train/test sets is to divide the training set into k equally sized partitions, or *folds*, and then using all folds but one to train, with the one left out for testing. This technique is called *leave-one-out* training. Another way is to divide the training set into k equally sized sets and then run the algorithm k times. In round i, we use all folds but fold i for training and fold i for testing. The average performance of the algorithm over k rounds measures the *generalization accuracy* of the algorithm. This robust technique is known as *k-fold cross validation*.

Leave-one-out

k-fold Cross Validation

To compare the masked labels with the predicted labels, depending on the type of supervised learning algorithm, different evaluation techniques can be used. In classification, the class attribute is discrete so the values it can take are limited. This allows us to use *accuracy* to evaluate the classifier. The accuracy is the fraction of labels that are predicted correctly. Let n be the size of the test dataset and let c be the number of instances from the test dataset for which the labels were predicted correctly using the trained model. Then the accuracy of this model is

$$accuracy = \frac{c}{n}. \tag{5.53}$$

In the case of regression, however, it is unreasonable to assume that the label can be predicted precisely because the labels are real values. A small variation in the prediction would result in extremely low accuracy. For instance, if we train a model to predict the temperature of a city in a given day and the model predicts the temperature to be 71.1 degrees Fahrenheit and the actual observed temperature is 71, then the model is highly accurate; however, using the accuracy measure, the model is 0% accurate. In general, for regression, we check if the predictions are highly correlated with the ground truth using correlation analysis, or we can fit lines to both ground truth and prediction results and check if these lines

Table 5.3: Distance Measures

Measure Name	Formula	Description
Mahalanobis	$d(X,Y) = \sqrt{(X-Y)^T\Sigma^{-1}(X-Y)}$	X, Y are features vectors and Σ is the covariance matrix of the dataset
Manhattan (L_1 norm)	$d(X,Y) = \sum_i \lvert x_i - y_i \rvert$	X, Y are features vectors
L_p-norm	$d(X,Y) = (\sum_i \lvert x_i - y_i \rvert^n)^{\frac{1}{n}}$	X, Y are features vectors

are close. The smaller the distance between these lines, the more accurate the models learned from the data.

5.5 Unsupervised Learning

Unsupervised learning is the unsupervised division of instances into groups of similar objects. In this topic, we focus on *clustering*. In clustering, the data is often unlabeled. Thus, the label for each instance is not known to the clustering algorithm. This is the main difference between supervised and unsupervised learning. Clustering

Any clustering algorithm requires a distance measure. Instances are put into different clusters based on their distance to other instances. The most popular distance measure for continuous features is the *Euclidean distance*:

$$\begin{aligned} d(X,Y) &= \sqrt{(x_1-y_1)^2 + (x_2-y_2)^2 + \cdots + (x_n-y_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}, \end{aligned} \tag{5.54}$$

where $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ are n-dimensional feature vectors in $\mathbb{R}^n$. A list of some commonly used distance measures is provided in Table 5.3.

Once a distance measure is selected, instances are grouped using it. Clusters are usually represented by compact and abstract notations. "Cluster centroids" are one common example of this abstract notation. Finally, clusters are evaluated. There is still a large debate on the issue of evaluating clustering because of the lack of cluster labels in unsupervised learning. Clustering validity and the definition of valid clusters are two of the challenges in the ongoing research.

5.5.1 Clustering Algorithms

There are many clustering algorithm types. In this section, we discuss *partitional* clustering algorithms, which are the most frequently used clustering algorithms. In Chapter 6, we discuss two other types of clustering algorithms: spectral clustering and hierarchical clustering.

Partitional Algorithms

Partitional clustering algorithms partition the dataset into a set of clusters. In other words, each instance is assigned to a cluster exactly once, and no instance remains unassigned to clusters. k-means [135] is a well-known example of a partitional algorithm. The output of the k-means algorithm ($k = 6$) on a sample dataset is shown in Figure 5.7. In this figure, the dataset has two features, and instances can be visualized in a two-dimensional space. The instances are shown using symbols that represent the cluster to which they belong. The pseudocode for k-means algorithm is provided in Algorithm 5.2.

The algorithm starts with k initial centroids. In practice, these centroids are randomly chosen instances from the dataset. These initial instances form the initial set of k clusters. Then, we assign each instance to one of these clusters based on its distance to the centroid of each cluster. The calculation of distances from instances to centroids depends on the choice of distance measure. Euclidean distance is the most widely used distance measure. After assigning all instances to a cluster, the centroids, are recomputed by taking the average (mean) of all instances inside the clusters (hence, the name k-means). This procedure is repeated using the newly computed centroids. Note that this procedure is repeated until convergence. The most common criterion to determine convergence is to check whether centroids are no longer changing. This is equivalent to clustering

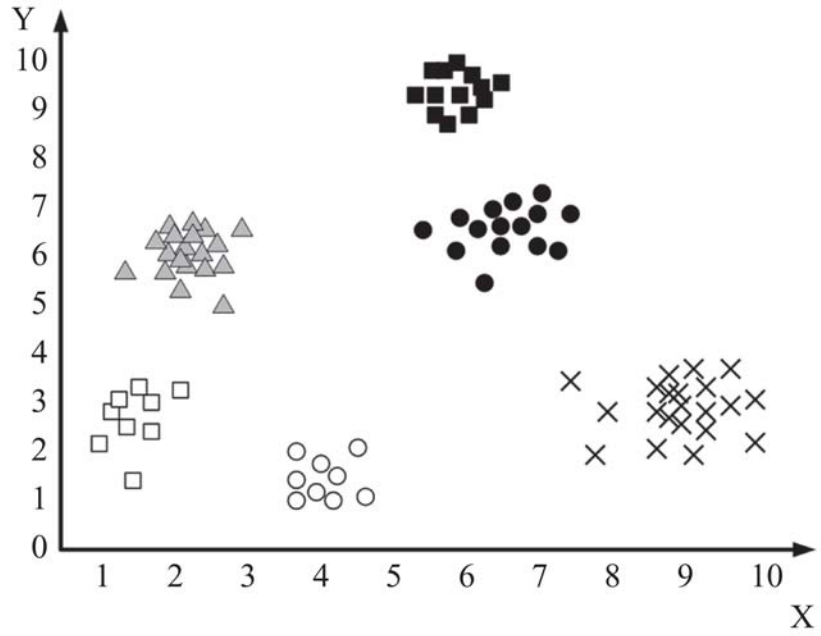


Figure 5.7: *k*-Means Output on a Sample Dataset. Instances are two-dimensional vectors shown in the 2-D space. *k*-means is run with $k = 6$, and the clusters found are visualized using different symbols.

assignments of the data instances stabilizing. In practice, the algorithm execution can be stopped when the Euclidean distance between the centroids in two consecutive steps is bounded above by some small positive ϵ. As an alternative, *k*-means implementations try to minimize an *objective function*. A well-known objective function in these implementations is the squared distance error:

$$\sum_{i=1}^{k} \sum_{j=1}^{n(i)} \|x_j^i - c_i\|^2, \tag{5.55}$$

where x_j^i is the jth instance of cluster i, $n(i)$ is the number of instances in cluster i, and c_i is the centroid of cluster i. The process stops when the difference between the objective function values of two consecutive iterations of the *k*-means algorithm is bounded by some small value ϵ.

Note that *k*-means is highly sensitive to the initial k centroids, and different clustering results can be obtained on a single dataset depending on

Algorithm 5.2 *k*-Means Algorithm

Require: A Dataset of Real-Value Attributes, *k* (number of Clusters)

1: **return** A Clustering of Data into *k* Clusters
2: Consider *k* random instances in the data space as the initial cluster centroids.
3: **while** centroids have not converged **do**
4: Assign each instance to the cluster that has the closest cluster centroid.
5: If all instances have been assigned then recalculate the cluster centroids by averaging instances inside each cluster
6: **end while**

the initial *k* centroids. This problem can be mitigated by running *k*-means multiple times and selecting the clustering assignment that is observed most often or is more desirable based on an objective function, such as the squared error. Since *k*-means assumes that instances that belong to the same cluster are the ones that found the cluster's centroid closer than other centroids in the dataset, one can safely assume that all the instances inside a cluster fall into a hyper-sphere, with the centroid being its center. The radius for this hyper-sphere is defined based on the farthest instance inside this cluster. If clusters that need to be extracted are nonspherical (globular), *k*-means has problems detecting them. This problem can be addressed by a preprocessing step in which a transformation is performed on the dataset to solve this issue.

5.5.2 Unsupervised Learning Evaluation

When clusters are found, there is a need to evaluate how accurately the task has been performed. When ground truth is available, we have prior knowledge of which instances should belong to which cluster, as discussed in Chapter 6 in detail. However, evaluating clustering is a challenge because ground truth is often not available. When ground truth is unavailable, we incorporate techniques that analyze the clusters found and describe the quality of clusters found. In particular, we can use techniques that measure *cohesiveness* or *separateness* of clusters.

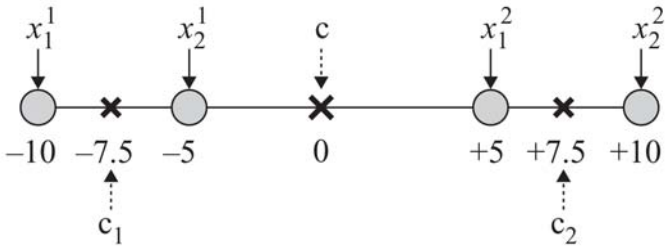


Figure 5.8: Unsupervised Learning Evaluation.

Cohesiveness

In evaluating clustering, we are interested in clusters that exhibit *cohesiveness*. In cohesive clusters, instances inside the clusters are close to each other. In statistical terms, this is equivalent to having a small standard deviation (i.e., being close to the mean value). In clustering, this translates to being close to the centroid of the cluster. So cohesiveness is defined as the distance from instances to the centroid of their respective clusters,

$$cohesiveness = \sum_{i=1}^{k} \sum_{j=1}^{n(i)} \|x_j^i - c_i\|^2, \tag{5.56}$$

which is the squared distance error (also known as SSE) discussed previously. Small values of cohesiveness denote highly cohesive clusters in which all instances are close to the centroid of the cluster.

Example 5.7. *Figure 5.8 shows a dataset of four one-dimensional instances. The instances are clustered into two clusters. Instances in cluster 1 are x_1^1 and x_2^1, and instances in cluster 2 are x_1^2 and x_2^2. The centroids of these two clusters are denoted as c_1 and c_2. For these two clusters, the cohesiveness is*

$$\begin{aligned} cohesiveness &= |-10-(-7.5)|^2 + |-5-(-7.5)|^2 + |5-7.5)|^2 \\ &\quad +|10-7.5|^2 = 25. \end{aligned} \tag{5.57}$$

Separateness

We are also interested in clustering of the data that generates clusters that are well separated from one another. To measure this distance between clusters, we can use the *separateness* measure. In statistics, separateness

can be measured by standard deviation. Standard deviation is maximized when instances are far from the mean. In clustering terms, this is equivalent to cluster centroids being far from the mean of the entire dataset:

$$separateness = \sum_{i=1}^{k} ||c - c_i||^2, \tag{5.58}$$

where $c = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the centroid of all instances and c_i is the centroid of cluster i. Large values of separateness denote clusters that are far apart.

Example 5.8. *For the dataset shown in Figure 5.8, the centroid for all instances is denoted as c. For this dataset, the separateness is*

$$separateness = |-7.5 - 0|^2 + |7.5 - 0|^2 = 112.5. \tag{5.59}$$

In general, we are interested in clusters that are both cohesive and separate. The silhouette index combines both these measures.

Silhouette Index

The *silhouette index* combines both cohesiveness and separateness. It compares the average distance value between instances in the same cluster and the average distance value between instances in different clusters. In a well-clustered dataset, the average distance between instances in the same cluster is small (cohesiveness), and the average distance between instances in different clusters is large (separateness). Let $a(x)$ denote the average distance between instance x of cluster C and all other members of C:

$$a(x) = \frac{1}{|C| - 1} \sum_{y \in C, y \neq x} ||x - y||^2. \tag{5.60}$$

Let $G \neq C$ denote the cluster that is closest to x in terms of the average distance between x and members of G. Let $b(x)$ denote the average distance between instance x and instances in cluster G:

$$b(x) = \min_{G \neq C} \frac{1}{|G|} \sum_{y \in G} ||x - y||^2. \tag{5.61}$$

Since we want distance between instances in the same cluster to be smaller than distance between instances in different clusters, we are interested in $a(x) < b(x)$. The silhouette clustering index is formulated as

$$s(x) = \frac{b(x) - a(x)}{\max(b(x), a(x))}, \tag{5.62}$$

$$\textit{silhouette} = \frac{1}{n}\sum_{x} s(x). \tag{5.63}$$

The silhouette index takes values between $[-1, 1]$. The best clustering happens when $\forall x\ a(x) \ll b(x)$. In this case, $\textit{silhouette} \approx 1$. Similarly when $\textit{silhouette} < 0$, that indicates that many instances are closer to other clusters than their assigned cluster, which shows low-quality clustering.

Example 5.9. *In Figure 5.8, the $a(.)$, $b(.)$, and $s(.)$ values are*

$$a(x_1^1) = |-10-(-5)|^2 = 25 \tag{5.64}$$

$$b(x_1^1) = \frac{1}{2}(|-10-5|^2 + |-10-10|^2) = 312.5 \tag{5.65}$$

$$s(x_1^1) = \frac{312.5-25}{312.5} = 0.92 \tag{5.66}$$

$$a(x_2^1) = |-5-(-10)|^2 = 25 \tag{5.67}$$

$$b(x_2^1) = \frac{1}{2}(|-5-5|^2 + |-5-10|^2) = 162.5 \tag{5.68}$$

$$s(x_2^1) = \frac{162.5-25}{162.5} = 0.84 \tag{5.69}$$

$$a(x_1^2) = |5-10|^2 = 25 \tag{5.70}$$

$$b(x_1^2) = \frac{1}{2}(|5-(-10)|^2 + |5-(-5)|^2) = 162.5 \tag{5.71}$$

$$s(x_1^2) = \frac{162.5-25}{162.5} = 0.84 \tag{5.72}$$

$$a(x_2^2) = |10-5|^2 = 25 \tag{5.73}$$

$$b(x_2^2) = \frac{1}{2}(|10-(-5)|^2 + |10-(-10)|^2) = 312.5 \tag{5.74}$$

$$s(x_2^2) = \frac{312.5-25}{312.5} = 0.92. \tag{5.75}$$

Given the $s(.)$ values, the silhouette index is

$$\textit{silhouette} = \frac{1}{4}(0.92 + 0.84 + 0.84 + 0.92) = 0.88. \tag{5.76}$$

5.6 Summary

This chapter covered data mining essentials. The general process for analyzing data is known as knowledge discovery in databases (KDD). The first step in the KDD process is data representation. Data instances are represented in tabular format using features. These instances can be labeled or unlabeled. There exist different feature types: nominal, ordinal, interval, and ratio. Data representation for text data can be performed using the vector space model. After having a representation, quality measures need to be addressed and preprocessing steps completed before processing the data. Quality measures include noise removal, outlier detection, missing values handling, and duplicate data removal. Preprocessing techniques commonly performed are aggregation, discretization, feature selection, feature extraction, and sampling.

We covered two categories of data mining algorithms: supervised and unsupervised learning. Supervised learning deals with mapping feature values to class labels, and unsupervised learning is the unsupervised division of instances into groups of similar objects.

When labels are discrete, the supervised learning is called classification, and when labels are real numbers, it is called regression. We covered, these classification methods: decision tree learning, naive Bayes classifier (NBC), nearest neighbor classifier, and classifiers that use network information. We also discussed linear and logistic regression.

To evaluate supervised learning, a training-testing framework is used in which the labeled dataset is partitioned into two parts, one for training and the other for testing. Different approaches for evaluating supervised learning such as leave-one-out or k-fold cross validation were discussed.

Any clustering algorithm requires the selection of a distance measure. We discussed partitional clustering algorithms and k-means from these algorithms, as well as methods of evaluating clustering algorithms. To evaluate clustering algorithms, one can use clustering quality measures such as cohesiveness, which measures how close instances are inside clusters, or separateness, which measures how separate different clusters are from one another. Silhouette index combines the cohesiveness and separateness into one measure.

5.7 Bibliographic Notes

A general review of data mining algorithms can be found in the machine learning and pattern recognition [40, 75, 199, 234, 235, 163], data mining [92, 120, 304, 270, 120], and pattern recognition [39, 75] literature.

Among preprocessing techniques, feature selection and feature extraction have gained much attention due to their importance. General references for feature selection and extraction can be found in [175, 64, 65, 117, 313, 175, 176]. Feature selection has also been discussed in social media data in [275, 276, 277]. Although not much research is dedicated to sampling in social media, it plays an important role in the experimental outcomes of social media research. Most experiments are performed using sampled social media data, and it is important for these samples to be representative samples of the site that is under study. For instance, Morstatter et al. [203] studied whether Twitter's heavily sampled Streaming API, a free service for social media data, accurately portrays the true activity on Twitter. They show that the bias introduced by the Streaming API is significant.

In addition to the data mining categories covered in this chapter, there are other important categories in the area of data mining and machine learning. In particular, an interesting category is *semi-supervised* learning. In semi-supervised learning, the label is available for some instances, but not all. The model uses the labeled information and the feature distribution of the unlabeled data to learn a model. *Expectation maximization* (EM) is a well-established technique from this area. In short, EM learns a model from the data that is partially labeled (expectation step). Then, it uses this model to predict labels for the unlabeled instances (maximization step). The predicted labels for instances are used once again to refine the learned model and revise predictions for unlabeled instances in an iterative fashion until convergence in reached. In addition to supervised methods covered, neural networks deserve mention [123]. More on regression techniques in available in [208, 40].

Clustering is one of the most popular areas in the field of machine learning research. A taxonomy of clustering algorithms can be found in [35, 136, 305, 197]. Among clustering algorithms, some of which use data density of cluster data, DBSCAN [86], GDBSCAN [245], CLARANS [218], and OPTICS [14] are some of the most well known and practiced algorithms. Most of the previous contributions in the area of clustering

consider the number of clusters as an input parameter. Early literature in clustering had attempted to solve this by running algorithms for several Ks (number of clusters) and selecting the best K that optimizes some coefficients [196, 35]. For example, the distance between two cluster centroids normalized by a cluster's standard deviation could be used as a coefficient. After the coefficient is selected, the coefficient values are plotted as a function of K (number of clusters) and the best K is selected.

An interesting application of data mining is *sentiment analysis* in which the level of subjective content in information is quantified; for example, identifying the polarity (i.e., being positive/negative) of a digital camera review. General references for sentiment analysis can be found in [227, 174], and examples of recent developments in social media are available in [131, 132].

5.8 Exercises

1. Describe how methods from this chapter can be applied in social media.

2. Outline a framework for using the supervised learning algorithm for unsupervised learning.

Data

3. Describe methods that can be used to deal with missing data.

4. Given a continuous attribute, how can we convert it to a discrete attribute? How can we convert discrete attributes to continuous ones?

5. If you had the chance of choosing either instance selection or feature selection, which one would you choose? Please justify.

6. Given two text documents that are vectorized, how can we measure document similarity?

7. In the example provided for TF-IDF (Example 5.1), the word "orange" received zero score. Is this desirable? What does a high TF-IDF value show?

Supervised Learning

8. Provide a pseudocode for decision tree induction.

9. How many decision trees containing n attributes and a binary class can be generated?

10. What does zero entropy mean?

11.
    - What is the time complexity for learning a naive Bayes classifer?
    - What is the time complexity for classifying using the naive Bayes classifier?

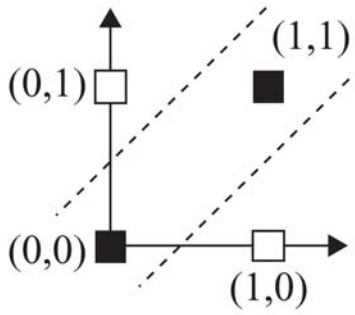


Figure 5.9: Nonlinearity of XOR Function.

- **Linear separability**: Two sets of two-dimensional instances are linearly separable if they can be completely separated using one line. In n-dimensional space, two set of instances are linearly separable if one can separate them by a hyper-plane. A classical example of nonlinearity is the XOR function. In this function, the two instance sets are the black-and-white instances (see Figure 5.9), which cannot be separated using a single line. This is an example of a nonlinear binary function. Can a naive Bayes classifier learn nonlinear binary functions? Provide details.
- What about linear separability and K-NN? Are K-NNs capable of solving such problems?

12. Describe how the least square solution can be determined for regression.

Unsupervised Learning

13. (a) Given k clusters and their respective cluster sizes s_1, s_2, ..., s_k, what is the probability that two random (with replacement) data vectors (from the clustered dataset) belong to the same cluster?

    (b) Now, assume you are given this probability (you do not have s_i's and k), and the fact that clusters are equally sized, can you find k? This gives you an idea how to predict the number of clusters in a dataset.

14. Give an example of a dataset consisting of four data vectors where there exist two different optimal (minimum SSE) 2-means (k-means, $k = 2$) clusterings of the dataset.

    - Calculate the optimal SSE value for your example.
    - In general, how should datasets look like geometrically so that we have more than one optimal solution?
    - What defines the number of optimal solutions?

    Perform two iterations of the k-means algorithm in order to obtain two clusters for the input instances given in Table 5.4. Assume that the first centroids are instances 1 and 3. Explain if more iterations are needed to get the final clusters.

Table 5.4: Dataset

Instance	X	Y
1	12.0	15.0
2	12.0	33.0
3	18.0	15.0
4	18.0	27.0
5	24.0	21.0
6	36.0	42.0

16. What is the usual shape of clusters generated by k-means? Give an example of cases where k-means has limitations in detecting the patterns formed by the instances.

17. Describe a preprocessing strategy that can help detect nonspherical clusters using k-means.