CS738 Class Notes

Steve Revilak

January 2008 – May 2008
Copyright © 2008 Steve Revilak. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.
Part 1

Mining Frequent Patterns

1.1 Lecture – 1/28/2008

1.1.1 General

- Get in the habit of consulting the course web page once per week (or so). In particular, homework assignments and handouts will be distributed via the course web page.
- At our discretion, we can pick up a copy of Witten’s book. In that text, we’ll mostly use the chapter that covers WEKA. However, WEKA’s documentation also includes that chapter.
- Handouts will be posted for one week only, then pulled. Be sure to take copies.
- We will have one in-class exam, and one final paper/project. There will not be an in-class final exam.

There will be no class on Wed. 1/30/2008.

We will have a makeup class on Saturday 2/23/2008, from 10:00 – 12:30.

1.1.2 What is Data Mining?

Data mining was founded in the field of databases. However, it also draws from the disciplines of AI and statistics. There are the three foundation areas of Data Mining.

DM came about through the desire to extract knowledge from large data sets. However, not all problems involve large data sets. There are notoriously difficult problems that involve only small amounts of data.

The web is arguably the largest database known to mankind. Some estimates put the size of the web at \( \sim 6 \) hexabytes. (1 hexabyte = 1MM terabytes).

As a field “Data Mining” might be better described by the phrases “knowledge mining” or “knowledge discovery”.

The main conference for data mining practitioners is KDD (Knowledge Discovery in Data). PKDD and PAKDD are other significant data mining conferences.

Business Intelligence is data mining applied to business problems. This is one area where AI plays a role.

Statistics gives us a foundation for making conjectures about data. For example, statistics allows us to do hypothesis testing, apply confidence intervals, etc.
1.1.3 A Example of a Data Mining Problem

Suppose we have a grocery store, with 8,000–10,000 items on display. We also have a set of \( n \) customers, \( c_1 \ldots c_n \).

Each time a customer comes into the store, they put a set of items in their shopping basket.

As store owners, we are interested in knowing what items are frequently purchased together. For example, if people commonly buy milk and bread together, we might want to stock bread near to the milk cooler, in the hopes that a person buying milk might be inclined to pick up bread.

The challenge in this sort of basket analysis is the number of combinations of items. If we have 10,000 items, then there are \( \binom{10000}{2} \) combinations of two items and \( \binom{10000}{3} \) combinations of three items. The sheer number of combinations make it difficult to (efficiently) identify those sets that occur frequently.

Another challenge – figuring out which patterns are “interesting”.

1.1.4 The Data Mining Process

The data mining process can be broken down into three phases:

1. Data preparation (takes about 85% of the time)
2. Data Mining Proper (10% of the time)
3. Data Post Processing (5% of the time)

Data Preparation

Data preparation is the most time consuming steps. Data preparation may include any of the following activities:

- Cleaning of data. Data needs to be consistent, and the data needs to be valid. Data Quality is a related issue.
- Data integration. We might take data from several different sources, and merge it into a common schema.
- Data Transformation. Broadly speaking, we deal with two categories of data: numerical data and nominal (categorical) data.

Nominal data is usually unordered – for example, a set of colors.

Numerical data can be turned into categorical data. For example, we might use a set of ranges as categories. Some algorithms only handle categorical data, which necessitates a numerical to categorical transformation.

Data Mining Proper

We choose an algorithm, and apply it to the data.

Data Post-Processing

During the post-processing stage, we evaluate the results of stage 2 (data mining proper). For example, if data mining reveals patterns, we might want to compare these with historic data, to see if the patterns show changes in activity.

During the post-processing stage, we’ll also figure out how to present the data mining results. The key to good presentation is doing it in such a way that the results can be used for decision making.
1.1.5 What Kinds of Data do We Mine

- Relational database data.
- Data in data warehouses and data marts.
- Transaction database data
- Specialized data; for example, textual or multimedia data.
- Data streams. In a stream, data appears only once; you do not have the opportunity to make a second pass to re-examine it. An example: network utilization for a cellular provider.
- Temporal data. For temporal data, we’re usually looking for patterns that appear at regular intervals.

There are specialized methods for each of these different data types.

The most important types of data mining:

- Frequent patterns
- Data classification
- Data clustering

Frequent Patterns

We’ve already seen one example of frequent pattern analysis – the shopping basket problem. We might also be looking for patterns in graphs (i.e., given a collection of graphs, which sub-graphs appear most often?).

Classification

Suppose we worked for a credit card company. We might be interested in finding factors that correspond to an individual’s credit worthiness. What makes a person a good credit risk vs a bad credit risk?

For example:

<table>
<thead>
<tr>
<th>Cust #</th>
<th>annual income</th>
<th>owns house?</th>
<th># kids</th>
<th>married?</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>$80,000</td>
<td>no</td>
<td>2</td>
<td>no</td>
<td>Bad risk</td>
</tr>
<tr>
<td>79</td>
<td>$77,000</td>
<td>yes</td>
<td>0</td>
<td>yes</td>
<td>Good risk</td>
</tr>
</tbody>
</table>

Given a data set like this, we’d like a way to predict values for the “risk” column.

When developing an algorithm, we typically “train” the algorithm on a small portion of the data, then see how well the trained algorithm does on the full data set.

*10x cross validation* is one training technique. We split the data into 10 portions; then see how well each 10% portion predicts the other 90%.

Clustering

Consider Figure 1.1

Figure 1.1 shows data points in three clusters. Intuitively, we have little difficulty forming the groups. However, this turns out to be difficult as an algorithmic problem. In an algorithmic context, we have to give precise definitions for “similar” and “dissimilar”, and the “distance” between points. We also have to give a precise statement as to how different clusters need to be.
When clustering, do not assume that our intuition for 2–3 dimensions holds for larger numbers of dimensions. It doesn’t.

1.1.6 To-Do

- Read the first four chapters of Han and Kamber.
1.2 Notes from Chapter 1 – 1/30/2008

Mining frequent data patterns typically yields a set of association rules, which have the form

\[ \text{buys}(X, \text{computer}) \Rightarrow \text{buys}(X, \text{software}) \quad \text{[support = 1\%, confidence = 50\%]} \]

In this example, \(X\) represents a customer, and the rule states that 50\% of customers that buy computers also buy software.

Confidence denotes how often the implication is true.

Support denotes the percentage of events that support the rule. In this example, the rule occurred in 1\% of purchase transactions.

We can write the rule more concisely as

\[ \text{computer} \Rightarrow \text{software} \quad [1\%, 50\%] \]

More formally, support is the probability that \(X\) and \(Y\) occur in the same transaction.

\[ \text{support}(X \Rightarrow Y) = P(X \cup Y) \]

Confidence is the measure of certainty for a particular association:

\[ \text{confidence}(X \Rightarrow Y) = P(Y|X) \]

Recall that \(P(Y|X)\) is the probability of \(Y\), given \(X\):

\[ P(Y|X) = \frac{P(YX)}{P(X)} = \frac{P(Y) \cdot P(X)}{P(X)} \]
1.3 Lecture – 2/4/2008

1.3.1 Frequent Rule Sets and Association Rules

You have a set of customers, 1 . . . n.

Each customer j, (1 ≤ j ≤ n) puts some set of items into a shopping basket. T(j) is the set of items purchased by customer j.

We use I to denote the store inventory. For example, we might have

\[ I = \{ \text{bread, butter, yogurt, cookies, diapers, chips, beer} \} \]

The set of items for one customer transaction (shopping purchase) might be

\[ T(j) = \{ \text{bread, butter, milk} \} \]

In general, T can be though of as a function that maps sets to the power set of I.

\[ T: \{ \ldots \} \rightarrow \mathcal{P}(I) \]

A supermarket might have a database of customer transactions, like the one shown in Table 1.1.

<table>
<thead>
<tr>
<th>j</th>
<th>T(j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>{ bread, yogurt }</td>
</tr>
<tr>
<td>t2</td>
<td>{ cookies, chips, beer }</td>
</tr>
<tr>
<td>t3</td>
<td>{ butter }</td>
</tr>
<tr>
<td>t4</td>
<td>{ bread, butter, yogurt }</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
</tr>
</tbody>
</table>

Table 1.1: Customer transactions

This database will have the restriction that \( T(j) \neq \emptyset \).

**The Challenge:** Find itemsets which occur with a minimum frequency \( \mu \) in the set \( T \). This is a challenge since the number of customers, \( j \), can be in the millions, and the set \( I \) can have tens of thousands of items.

Let \( L \) be a set of items. The *support count* of \( L \) is the number of transactions such that \( L \subseteq T(j) \):

\[
\text{suppcount}_T(L) = |\{j \mid L \subseteq T(j)\}| \tag{1.1}
\]

Support count leads to the definition of *support*:

\[
\text{support}_T(L) = \frac{\text{suppcount}_T(L)}{N} \tag{1.2}
\]

Where \( N \) denotes the number of transactions.

Note that \( 0 \leq \text{support}_T(L) \leq 1 \).

Suppose we have two itemsets, \( L \) and \( K \). We use the notation

\[
LK = L \cup K \quad \text{(set union)}
\]

Support count gives us an estimation of the conditional probability whereby a transaction that contains \( L \) will also contain \( K \):

\[
0 \leq \frac{\text{suppcount}_T(KL)}{\text{suppcount}_T(L)} \leq 1 \tag{1.3}
\]
We also have a confidence level for each association rule \( L \Rightarrow K \):

\[
\text{confidence} = \frac{\text{support}_T(KL)}{\text{support}_T(L)}
\]  

(1.4)

The confidence level \( c \) tells us how often a transaction that contains \( L \) will also contain \( K \). Note: we assume that \( L \cap K = \emptyset \).

The rule “\( L \Rightarrow K \)” is called an association rule.

Every association rule comes with support and confidence values:

\[
L \Rightarrow K: \begin{cases} \frac{\text{support}_T(L)}{\text{support}_T(KL)} & \text{support of the rule} \\ \frac{\text{support}_T(KL)}{\text{support}_T(L)} & \text{confidence of the rule} \end{cases}
\]  

(1.5)

Finding frequent item sets is the core of finding association rules.

In Table 1.1 we saw one way of representing transactions. In some cases, it will be more convenient to represent the sets as binary data: one row for each customer, and one column for each item. For example:

<table>
<thead>
<tr>
<th></th>
<th>bread</th>
<th>butter</th>
<th>yogurt</th>
<th>cookies</th>
<th>diapers</th>
<th>chips</th>
<th>beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1.2: Binary Set Representation

Let \( I \) be the set of items.

We say that \( L \subseteq I \) is a \( \mu \)-frequent item set in \( T \) if \( \text{support}_T(L) \geq \mu \).

If there are 10,000 items then there are \( 2^{10,000} \) different itemsets. There are too many combinations to handle with brute force.

### 1.3.2 Ryman Trees

Ryman trees give us a way to enumerate subsets with a tree. Figure 1.2 shows a Ryman tree for a set of 4. The numbers 1, 2, 3, 4 represent elements of a set.

![Figure 1.2: Ryman Tree for \(|I| = 4\)](image)

A few things to note about Figure 1.2

- The root is the empty set
- Level one has subsets of size 1. In general, level \( k \) has subsets of size \( k \).
• Each node contains (a) all elements of its parent node and (b) elements that are greater than all elements of its parent node.

For example, node \{1,2\} has children \{1,2,3\} and \{1,2,4\}; while node \{2,3\} has one child, \{2,3,4\}.

**Theorem 1.3.2.1:** If \(K\) is a set at level \(k\) of a Ryman Tree, then there are two sets \(L, M\), at level \((k - 1)\) such that

1. \(K = LM\)
2. \(L \cap M\) is a common ancestor of \(L, M\), at level \(k - 2\).
3. \(L\) and \(M\) are uniquely determined with properties [1] and [2].

**Example 1.3.2.2:** In Figure [1.2] let \(K = \{1,2,4\}\). The set \(L = \{1,2\}\); the set \(M = \{1,4\}\); the set \(L \cap M = \{1\}\).

Assume \(k \geq 2\). We have

\[
|K| = k \\
K = i_{a_1}, i_{a_2} \ldots i_{a_{k-2}} i_{a_{k-1}} i_{a_k} \quad \text{where} \quad a_1 < a_2 < \ldots a_{n-1} < a_n \\
L = i_{a_1}, i_{a_2} \ldots i_{a_{k-2}} i_{a_{k-1}} \\
M = i_{a_1}, i_{a_2} \ldots i_{a_{k-2}} i_{a_k}
\]

Note that

• \(L\) and \(M\) differ by exactly one element
• \(K\) is at level \(k\)
• \(L, M\) are at level \(k - 1\)
• \(L \cap M\) are at level \(k - 2\).

See first course handout for more on Ryman trees.

### 1.3.3 Apriori Algorithm for Finding Frequent Item Sets

Suppose we have itemsets \(U, V \subseteq I\). If \(U \subseteq V\), then \(\text{support}_T(V) \leq \text{support}_T(U)\). This is called the **apriori property**.

Let’s define some notation:

- \(\mathcal{F}_T^\mu\): the set of \(\mu\)-frequent itemsets
- \(\mathcal{F}_T^{\mu,r}\): the set of \(\mu\)-frequent itemsets that have \(r\) elements

\[
\mathcal{F}_T^\mu = \bigcup_{r \geq 1} \mathcal{F}_T^{\mu,r}
\]

Prior to generating the set \(\mathcal{F}_T^\mu\), we’ll need to generate a **candidate set**, \(\mathcal{C}_T^{\mu,r}\).

- \(\mathcal{C}_T^{\mu,r}\): a collection of candidate sets with support \(\mu\), containing \(r\) elements

The general sequence of the apriori algorithm is

\[
\mathcal{C}_T^{\mu,1} \leadsto \mathcal{F}_T^{\mu,1} \leadsto \mathcal{C}_T^{\mu,2} \leadsto \mathcal{F}_T^{\mu,2} \leadsto \cdots \leadsto \mathcal{C}_T^{\mu,r} = \emptyset
\]

The output is \(\mathcal{F}_T^\mu = \bigcup \mathcal{F}_T^{\mu,r-1}\).

The apriori algorithm has two (alternating) phases: **evaluation** and **apriori-gen**.
- Apriori-gen goes from $\mathcal{F}_{T,k}^\mu \rightarrow \mathcal{C}_{T,k+1}^\mu$
- Evaluation goes from $\mathcal{C}_{T,k}^\mu \rightarrow \mathcal{F}_{T,k}^\mu$

The evaluation phase involves a database scan. The apriori-gen phase does not.

Apriori-gen creates $\mathcal{C}_{T,k+1}^\mu$, which is a superset of $\mathcal{F}_{T,k}^\mu$: $\mathcal{F}_{T,k}^\mu \subseteq \mathcal{C}_{T,k+1}^\mu$.

Suppose we have the itemsets shown in Figure 1.3

|   | $k-1$ |   
|---|-------|---
| W |       |   
| U | $k$   |   
| V |       |   
| L | $k+1$ |   

Figure 1.3: Apriori example

Level $k$ shows $\mathcal{F}_{T,k}^\mu = \{U, V\}$.

At level $k + 1$, we have $L \subseteq \mathcal{C}_{T,k+1}^\mu$.

$L$ has two subsets at the prior level: $U$ and $V$.

$U$ and $V$ are direct descendants of $W$.

Suppose $L$ were a frequent set. Then $U$, $V$, which are subsets of $L$ would also be frequent. In addition, every other subset of $L$ would also be frequent.

### 1.3.4 A Quick Summary of the Apriori Algorithm

**Input** Level $k$, confidence $\mu$, and $\mathcal{F}_{T,k}^\mu$.

**Output** $\mathcal{C}_{T,k+1}^\mu$

**Procedure**
- Set $\mathcal{C}_{T,k+1}^\mu = \emptyset$
- For every pair $U, V \in \mathcal{F}_{T,k}^\mu$ such that there is a $W \in \mathcal{F}_{T,k-1}^\mu$, and $U = W_i$ and $V = W'_i$; add $L = UV$ to $\mathcal{C}_{T,k+1}^\mu$.
- Remove from $\mathcal{C}_{T,k+1}^\mu$ all sets which contain a subset with $k$ elements not in $\mathcal{F}_{T,k}^\mu$.

$\mathcal{C}_{T,k+1}^\mu$ may not be a frequent set, but it contains all of the frequent sets. $\mathcal{C}_{T,k+1}^\mu$ becomes the input to the evaluation phase.
1.4 Lecture – 2/6/2008

1.4.1 Apriori Algorithm

The Apriori algorithm’s purpose is to compute $F^\mu_T$ of $\mu$-frequent itemsets. Recall that

$$F^\mu_T = \bigcup_{r>1} F^\mu_{T,r}$$

The Apriori algorithm consists of two alternating processes: generation (apriori_gen) and evaluation. $C^\mu_{T,r}$ is the set of candidate item sets that have $r$ items. Apriori starts with $C^\mu_{T,1}$ - the set of candidate 1-sets.

The apriori algorithm proceeds in the following manner

$$C^\mu_{T,1} \xrightarrow{\text{evaluation}} F^\mu_{T,1} \xrightarrow{\text{apriori-gen}} C^\mu_{T,2} \xrightarrow{\text{evaluation}} F^\mu_{T,2} \xrightarrow{\text{apriori-gen}} \cdots \xrightarrow{\text{evaluation}} F^\mu_{T,r-1} \xrightarrow{\text{evaluation}} C^\mu_{T,r} = \emptyset$$

For each $F^\mu_T$ set, we’ll need to scan the data set in order to determine which candidate items are frequent. The algorithm terminates when the candidate set $C^\mu_{T,r}$ becomes empty.

Example 1.4.1.1 (Apriori Algorithm): Suppose we have the set $I = \{i_1, i_2, i_3, i_4, i_5\}$, along with the following transactions:

<table>
<thead>
<tr>
<th></th>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$i_3$</th>
<th>$i_4$</th>
<th>$i_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(1)$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T(2)$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T(3)$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$T(4)$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$T(5)$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$T(6)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$T(7)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T(8)$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We want to find frequent item sets with $\mu = 0.25$. Here are the steps:

1. We find $C^{0.25}_{T,1}$. This consists of all 1-sets, so

$$C^{0.25}_{T,1} = \{\{i_1\}, \{i_2\}, \{i_3\}, \{i_4\}, \{i_5\}\}$$

2. Next we do an evaluation step to find $F^{0.25}_{T,1}$. All 1-sets occur at least twice ($2/8 = 0.25$), so nothing is removed in this step. We have

$$F^{0.25}_{T,1} = \{\{i_1\}, \{i_2\}, \{i_3\}, \{i_4\}, \{i_5\}\}$$

3. Next, we do a generation step to find $C^{0.25}_{T,2}$. We’ll use a Rymon tree to generate all two-sets. Although we haven’t drawn it, our generation steps are always using Rymon trees.

4. Next, we do an evaluation step to find $F^{0.25}_{T,2}$. We’ll show this with a table:
2-set | support | remarks
--- | --- | ---
\{i_1, i_2\} | 3 |  
\{i_1, i_3\} | 2 |  
\{i_1, i_4\} | 1 | (remove: not enough support)
\{i_1, i_5\} | 3 |  
\{i_2, i_3\} | 5 |  
\{i_2, i_4\} | 2 |  
\{i_2, i_5\} | 3 |  
\{i_3, i_4\} | 2 |  
\{i_3, i_5\} | 3 |  
\{i_4, i_5\} | 2 |  

The first column of our table shows the output of \( \mathcal{C}_{T,0.25} \). As noted above, \( \{i_1, i_4\} \) is not part of \( \mathcal{F}_{T,0.25} \) since that set has support 0.125 < \( \mu \).

5. Next, we have another generate step, producing \( C_{T,3} \).

| 3-sets | remarks |
--- | --- |
\{i_1, i_2, i_3\} | See note 1 |
\{i_1, i_2, i_4\} | See note 1 |
\{i_1, i_3, i_4\} | See note 1 |
\{i_1, i_3, i_5\} |  
\{i_2, i_3, i_4\} |  
\{i_2, i_3, i_5\} |  
\{i_2, i_4, i_5\} |  
\{i_3, i_4, i_5\} |  

Note 1: the sets \( \{i_1, i_2, i_4\} \) and \( \{i_1, i_3, i_4\} \) are not part of \( \mathcal{C}_{T,3} \). Note that \( \{i_1, i_4\} \) was pruned because it did not have enough support. Since \( \{i_1, i_4\} \) does not have sufficient support, no superset of \( \{i_1, i_4\} \) will have sufficient support.

6. Next, we do an evaluation step to find \( \mathcal{F}_{T,3} \). Below, we show the \( \mathcal{C}_{T,3} \) sets along with their support values.

| 3-sets | support | remarks |
--- | --- | --- |
\{i_1, i_2, i_3\} | 2 | remove: 0.125 < 0.25 |
\{i_1, i_2, i_5\} | 1 | remove: 0.125 < 0.25 |
\{i_1, i_3, i_5\} | 1 |  
\{i_2, i_3, i_4\} | 2 |  
\{i_2, i_3, i_5\} | 3 |  
\{i_2, i_4, i_5\} | 2 |  
\{i_4, i_4, i_5\} | 2 |  

Thus, \( \mathcal{F}_{T,3} \) contains 5 elements:

\[
\mathcal{F}_{T,3} = \{\{i_1, i_2, i_3\}, \{i_2, i_3, i_4\}, \{i_2, i_3, i_5\}, \{i_2, i_4, i_5\}, \{i_4, i_4, i_5\}\}
\]

7. Next, we’re back to a generation stage for \( \mathcal{C}_{T,4} \).

| 4-sets | remarks |
--- | --- |
\{i_1, i_2, i_3, i_4\} | see note below |
\{i_1, i_2, i_3, i_5\} |  
\{i_2, i_3, i_4, i_3\} |  

\( \{i_1, i_2, i_3, i_4\} \) is not in \( \mathcal{C}_{T,4} \), since it is a superset of \( \{i_1, i_4\} \). The output of this step is

\[
\mathcal{C}_{T,4} = \{\{i_1, i_2, i_3, i_5\}, \{i_2, i_3, i_4, i_5\}\}
\]
8. Next, an evaluation step for $F_{T,4}^{0.25}$.

<table>
<thead>
<tr>
<th>4-sets</th>
<th>support</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>${i_1, i_2, i_3, i_5}$</td>
<td>1</td>
<td>remove: 0.125 &lt; 0.25</td>
</tr>
<tr>
<td>${i_2, i_3, i_4, i_5}$</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

This gives $F_{T,4}^{0.25} = \{\{i_2, i_3, i_4, i_5\}\}$.

9. Finally, we have a generation step to product $C_{0.25}^{\mu}T, 5$. Because $\{i_2, i_3, i_4, i_5\}$ is not extendable, we have $C_{0.25}^{\mu}T, 5 = \emptyset$

And the algorithm terminates.

10. The final set of frequent items it

$$\bigcup_{r=1}^{4} F_{T,r}^{0.25} = \{\{i_1\}, \{i_2\}, \{i_3\}, \{i_4\}, \{i_5\},$$
    $$\{i_1, i_2\}, \{i_1, i_3\}, \{i_1, i_5\}, \{i_2, i_3\}, \{i_2, i_4\}, \{i_2, i_5\}, \{i_3, i_4\}, \{i_3, i_5\}, \{i_4, i_5\},$$
    $$\{i_1, i_2, i_3\}, \{i_2, i_3, i_4\}, \{i_2, i_3, i_5\}, \{i_2, i_4, i_5\}, \{i_3, i_4, i_5\},$$
    $$\{i_2, i_3, i_4, i_5\}\}$$

There are many opportunities to optimize the Apriori algorithm, which we'll study later on.

As the example illustrates, the Apriori algorithm proceeds level-wise. This will be an important trait to remember.

1.4.2 An Upper Bound on Apriori’s Database Scanning

Suppose we have a collection of itemsets. As noted earlier, if a given itemset is frequent, then every subset of that item set will also be frequent.

Suppose $P \in F^\mu_T$; then $P$ is $\mu$-frequent.

Now suppose $Q \subseteq P$. This tells us that $Q \in F^\mu_T$, whereby $Q$ is $\mu$-frequent.

The property of begin $\mu$-frequent is inherited by subsets; we call this a hereditary property.

**Definition 1.4.2.1 (Border):** Let $C$ be a collection of item sets. The set $BD(C)$ is the border of $C$:

$$BD(C) = \{U \mid T \subset U \rightarrow T \in C \text{ and } U \subset Z \rightarrow Z \notin C\}$$  \hspace{1cm} (1.6)

**Definition 1.4.2.2 (Positive Border):** The positive border of $C$ is defined as

$$BD^+(C) = BD(C) \cap C$$
$$= \{U \in C \mid T \subset U \rightarrow T \in C \text{ and } U \subset Z \rightarrow Z \notin C\}$$  \hspace{1cm} (1.7)

**Definition 1.4.2.3 (Negative Border):** The negative border of $C$ is defined as

$$BD^-(C) = BD(C) - C$$
$$= \{U \notin C \mid T \subset U \rightarrow T \in C \text{ and } U \subset Z \rightarrow Z \notin C\}$$  \hspace{1cm} (1.8)
In the worst possible case, apriori will perform

\[ |F^n_T| + |BD^{-}(F^n_T)| \]

scans. \(|F^n_T|\) comes from the successful queries (the ones with enough support), and \(|BD^{-}(F^n_T)|\) comes from the unsuccessful queries (those without enough support).

### 1.4.3 Partially Ordered Sets

**Definition 1.4.3.1 (Partial Order):** A partial order is a relation \(\leq\) that has three properties

1. \(x \leq x\) for all \(x\) in \(S\). (Reflexivity)
2. \(x \leq y\) and \(y \leq x\) implies \(x = y\). (Anti-Symmetry)
3. \(x \leq y\) and \(u \leq z\) implies \(x \leq z\). (Transitivity)

**Example 1.4.3.2 (Divides as a partial order):** Let \(S\) be the set of natural numbers. We define the relation \(m|n\) as “\(m\) divides \(n\)”. Specifically, \(m|n\) if there is a \(k\) such that \(n = km\).

**Example 1.4.3.3 (Set Inclusion as a Partial Order):** The relation \(\subseteq\) is a partial order.

\[ P \subseteq P \]
\[ P \subseteq Q \text{ and } Q \subseteq P \implies P = Q \]
\[ P \subseteq Q \text{ and } Q \subseteq T \implies P \subseteq T \]

**Example 1.4.3.4 (Infix as a partial order):** We write \(u \leq v\) to mean “\(u\) is an infix of \(v\)”. This relation also satisfies the criteria of a partial order:

\[ u \leq u \]
\[ u \leq v \text{ and } v \leq u \implies u = v \]
\[ u \leq v \text{ and } v \leq w \implies u \leq w \]

We can come up with many other examples. For example \(A \leq B\) to mean “\(A\) is a subgraph of \(B\)”.

**Standard Ways of Representing Partial Orders**

The standard way to represent a partial order is

\[(S, \leq)\]

Where \(S\) is a set and \(\leq\) is the partial order relation. We call \((S, \leq)\) a partially ordered set or poset.

Suppose \(u \leq v\) but \(u \neq v\). We write \(u < v\), which is a strict partial order.

**Definition 1.4.3.5 (Open and Closed Interval):** Let \(u, v \in S\). Then

\[ \{ t \in S \mid u < t < v \} \]
\[ \{ t \in S \mid u \leq t \leq v \} \]

is the open interval \((u, v)\)

is the closed interval \([u, v]\)

**Example 1.4.3.6:** Suppose \(S = \{1, 2, 3, 4, 5, 6, 7, 8\}\), and our poset is \((S, \leq)\).

The interval \([2, 8]\) is \(\{k \mid 2|k \text{ and } k|8\}\). Therefore

\[ [2, 8] = \{2, 4, 8\} \]
\[ (2, 8) = \{4\} \]
**Definition 1.4.3.7 (Cover):** We say that $u$ covers $v$ if $u < v$ and $(u,v) = \emptyset$. In Example 1.4.3.6, 4 covers 2.

We use the notation $u \prec v$ to mean “$u$ covers $v$.”

We can use a Hasse Diagram of a poset to illustrate cover. There is an edge from $u$ to $v$ if $u \prec v$. The Hasse Diagram for example 1.4.3.2 is shown in Figure 1.4.

![Figure 1.4: Hasse Diagram on $\{1\ldots8\}$](image)

As another example, Figure 1.5 shows a Hasse diagram for $(\{a,b,c\}, \subseteq)$. Figure 1.5 also happens to be a lattice.

![Figure 1.5: A lattice](image)

Consider the two sets $S$ and $U$, as shown in Figure 1.6.

![Figure 1.6: A lattice](image)

We say that $t$ is an upper bound of $U$ if $u \leq t$ for every $u \in U$.

The set $U^s$ is the set of upper bounds of $U$. Note that if $U \subseteq V$, then $U^s \supseteq V^s$.

We say that $t$ is a lower bound of $U$ if $t \leq u$ for every $u \in U$.

The set $U^i$ is the set of lower bounds. Note that if $U \subseteq V$, then $U^i \supseteq V^i$.

The set $U \cap U^s$ has at most one element. Likewise for $U \cap U^i$

\[
|U \cap U^s| \leq 1 \\
|U \cap U^i| \leq 1
\]
If \( p \in U \cap U^s \), then \( p \) is the largest element of \( U \). Note that it’s possible to have \( U \cap U^s = \emptyset \).

Similarly, if \( q \in U \cap U^i \), then \( q \) is the smallest element of \( U \).

The set \( U^s \cap (U^s)^i \) contains at most one element. If \( z = U^s \cap (U^s)^i \), then \( z \) is the least upper bound of \( U \). We call \( z \) the supremum of \( U \), written \( \sup U \), or \( \bigvee U \).

If \( p = U^i \cap (U^i)^s \), then \( p \) is the greatest lower bound of \( U \). We call \( p \) the infimum.

### 1.4.4 Logistics

Check the course web site over the weekend. Our second handout should be posted on Sunday, along with our first homework assignment.

Data mining seminar is held on Fridays at 13:00 in the web lab.
1.5 Lecture – 2/11/2008

1.5.1 Posets and Bounds

Let \((S, \leq)\) be a poset, and let \(K \subseteq S\)

\[K^\uparrow: \text{ is an upper bound of } K\]
\[K^\downarrow: \text{ is a lower bound of } K\]
\[K \cap K^\uparrow: \text{ has at most one element}\]
\[K \cap K^\downarrow: \text{ has at most one element}\]

Also

\((a, b): \text{ is the open interval } a < i < b\]
\([a, b]: \text{ is the closed } a \leq i \leq b\]

Example 1.5.1.1: Let \(K = (0, 1)\) over \(\mathbb{R}\).
\(K^\uparrow = [1, \infty].\)
\(K \cap K^\uparrow = \emptyset.\)

Example 1.5.1.2: Let \(K = [0, 1]\) over \(\mathbb{R}\).
\(K^\uparrow = [1, \infty]\]
\(K \cap K^\uparrow = \{1\}\)

Say we form \(K^\uparrow \cap (K^\uparrow)^\downarrow.\) This set contains at most one element.

Let \(t \in K^\uparrow \cap (K^\uparrow)^\downarrow.\) Then \(t = \sup K; t\) is the supremum of \(K.\)

For example, \(\sup[0, 1] = 1.\)
Likewise, \(K^\downarrow \cap (K^\downarrow)^\uparrow\) contains at most one element.
Let \(s \in K^\downarrow \cap (K^\downarrow)^\uparrow.\) Then, \(s = \inf K; s\) is the infimum of \(K.\)

Example 1.5.1.3: Let \(M\) be a set; \(\mathcal{P}(M)\) is the power set of \(M,\) and \((\mathcal{P}(M), \subseteq)\) is a poset.

Suppose \(A, B \in \mathcal{P}(M).\)

\[\sup\{A, B\} = A \cup B\]
\[\inf\{A, B\} = A \cap B\]

Suppose \(t = K^\uparrow \cap (K^\uparrow)^\downarrow.\) Then \(t = \sup K.\) \(t\) has to be an upper bound, and \(t\) has to be the smallest upper bound.

\(A \subseteq A \cup B\) and \(B \subseteq A \cup B.\) Therefore \(A \cup B\) is an upper bound.

Suppose \(A \subseteq C\) and \(B \subseteq C.\) Then \(A \cup B \subseteq C\) and \(A \cup B\) is a least upper bound.

Example 1.5.1.4: Consider the poset \((\mathbb{N}, |).\) Every \(\{m, n\}\) has a supremum and infimum.

Suppose \(p = \sup\{m, n\}.\) Then \(m|p\) and \(n|p.\)
If \(q\) is such that \(m|q\) and \(n|q,\) then \(p|q.\)

\(p\) is the least common multiple of \(m, n.\)

\[\inf\{m, n\} = \gcd(m, n)\]

For some sets, every pair \(\{m, n\}\) has both a supremum and an infimum. However, not all sets have this property.
Definition 1.5.1.5 (Lattice): Let \((S, \leq)\) be a poset such that for every \(x, y\) there both a \(\sup\{x, y\}\) and an \(\inf\{x, y\}\). Then \((S, \leq)\) is said to be a lattice.

\((\mathcal{P}(M), \subseteq)\) and \((\mathbb{N}, |)\) are examples of lattices.

Definition 1.5.1.6 (Complete Lattice): Let \((S, \leq)\) be a poset. If, for every subset \(T\) of \(S\), there is \(\sup T\) and \(\inf T\), then \((S, \leq)\) is a complete lattice.

Lattice theory is an entire branch of mathematics. In this course, we’re only going to skim the surface of it.

Notation

\[
\begin{align*}
\sup T & : \text{ can also be written } \bigvee T \\
\inf T & : \text{ can also be written } \bigwedge T \\
\sup\{x, y\} & : \text{ can also be written } x \lor y \\
\inf\{x, y\} & : \text{ can also be written } x \land y
\end{align*}
\]

1.5.2 Closures

Let \((S, \leq)\) be a complete lattice. A closure is a mapping

\[K: S \to S\]  \hspace{1cm} (1.11)

that satisfies three criteria.

1. \(x \leq K(x)\)
2. \(x_1 \leq x_2\) implies \(K(x_1) \leq K(x_2)\)
3. \(K(K(x)) = K(x)\)

for every \(x_1, x_2, \ldots, x_n \in S\).

Definition 1.5.2.1 (Monotonicity): Suppose we have a function \(f: S \to S\) such that

\[x_1 \leq x_2\] implies \(f(x_1) \leq f(x_2)\)

We say that \(f\) has the property of monotonicity.

Definition 1.5.2.2 (Anti-Monotonicity): Suppose we have the function \(g: S \to S\) such that

\[x_1 \leq x_2\] implies \(g(x_1) \geq g(x_2)\)

Then we say that \(g\) has the property of anti-monotonicity.

Definition 1.5.2.3 (Galois Connection): Suppose we have two anti-monotonic functions

\[
\begin{align*}
f & : S \to S \\
g & : S \to S
\end{align*}
\]

We can observe four properties

1. \(x_1 \leq x_2\) implies \(f(x_1) \geq f(x_2)\).
2. \(x_1 \leq x_2\) implies \(g(x_1) \geq g(x_2)\).
3. \( x \leq f(g(x)) \)
4. \( x \leq g(f(x)) \)

A pair of mappings with these four properties is called a *Galois Connection*.

**Claim 1.5.2.4:** If \( f \) and \( g \) form a Galois Connection, then

\[
K(x) = f(g(x)) \quad \text{and} \quad H(x) = g(f(x))
\]

are closures.

**Example 1.5.2.5:** Suppose we have posets \((S, \leq)\) and \((U, \leq)\) and functions \( f: S \to U \) and \( g: U \to S \).

Let's say that \( f \), \( g \) satisfy the following properties:

1. \( s_1 \leq s_2 \) implies \( f(s_1) \geq f(s_2) \)
2. \( u_1 \leq u_2 \) implies \( g(u_1) \geq g(u_2) \)
3. \( s \leq g(f(s)) \)
4. \( u \leq f(g(u)) \)

In this case,

\[
K(S) = g(f(s)) \text{ is a closure on } S \\
H(U) = f(g(u)) \text{ is a closure on } U
\]

\( f \) and \( g \) are called the Galois Connection between \((S, \leq)\) and \((U, \leq)\).

\[
s \leq g(f(s)) \text{ implies } s \leq K(s) \\
u \leq f(g(u)) \text{ implies } u \leq H(u)
\]

Suppose \( s_1 \leq s_2 \). Then,

\[
f(s_1) \geq f(s_2) \quad \text{anti-monotonicity} \\
g(f(s_1)) \leq g(f(s_2)) \quad \text{anti-monotonicity}
\]

Recalling that \( K(s) = g(f(s)) \), we can say that

\[
x_1 \leq x_2 \text{ implies } K(s_1) \leq K(s_2)
\]

Similarly, suppose \( u_1 \leq u_2 \). Then

\[
g(u_1) \geq g(u_2) \quad \text{anti-monotonicity} \\
f(g(u_1)) \leq f(g(u_2)) \quad \text{anti-monotonicity}
\]

Therefore

\[
H(u_1) \leq H(u_2)
\]

Next, consider \( s \leq g(f(s)) \). From this, we can derive

\[
f(s) \geq f(g(f(s))) \\
f(s) = f(g(f(s))) \\
g(u) = g(f(g(u)))
\]
So,
\[ g(f(s)) = g(f(g(f(s)))) = K(s) = K(K(s)) \]
since \( K(s) = g(f(s))' \).

Suppose \( s \in S \), and \( K(s) = K(K(s)) \). Therefore \( s \) is closed.

If \( t \) is closed and \( s \leq t \), then \( K(s) < t. K(s) \leq K(t) = t \). Therefore \( t \) is closed.

### 1.6 Frequent Item Sets

Now, let’s look at how Galois Connections can be applied to the problem of mining frequent item sets.

Recall our model of a transactional dataset. We have a set of transactions \( T(1), T(2), \ldots, T(n) \). Each \( T(i) \) is associated with a set of items.

Let
\[
D = \{1, 2, \ldots, n\} \quad \text{be the set of transaction ids} \\
I = \{i_1, \ldots\} \quad \text{be the set of items}
\]

Suppose \( E \subseteq D \). \( E \) is a subset of transactions.

\( \{T(k) \mid k \in E\} \) is the set of items that appear in \( E \)’s transactions.

We define
\[
\text{items}_T(E) = \bigcap \{T(k) \mid k \in E\} \quad (1.12)
\]

\( \text{items}_T(E) \) is the set of items that appear in every transaction \( T(k) \in E \).

Now, let \( H \subseteq I \) be a set of items. We define
\[
\text{tids}_T(H) = \{l \in D \mid H \subseteq T(l)\} \quad (1.13)
\]

\( \text{tids}_T(H) \) is the set of transactions that contain the set of items \( H \).

\( \text{tids}_T(H) \) and \( \text{items}_T(E) \) constitute a Galois Connection:

\[
\text{items}_T(E) : \mathcal{P}(D) \rightarrow \mathcal{P}(I) \quad (1.14) \\
\text{tids}_T(H) : \mathcal{P}(I) \rightarrow \mathcal{P}(D) \quad (1.15)
\]

Note that \( \text{tids}_T(H) \) is the set of transactions that contain \( H \). Therefore:
\[
\text{suppcount}_T(H) = |\text{tids}_T(H)| \quad (1.16)
\]

Let’s verify that \( \text{tids}_T(H) \) and \( \text{items}_T(E) \) to indeed form a Galois Connection. We examine the four properties of a Galois connection.

- Suppose \( E_1 \subseteq E_2 \). Then \( \text{items}_T(E_2) \subseteq \text{items}_T(E_1) \). The \( \cap \) causes the anti-monotonicity.
- Suppose \( H_1 \subseteq H_2 \). This implies that \( \text{tids}_T(H_2) \subseteq \text{tids}_T(H_1) \). Since \( H_1 \) is contained in \( H_2 \), the support for \( H_2 \) will be less than the support of \( H_1 \).
- \( E \subseteq \text{tids}_T(\text{items}_T(E)) \). This is pretty straightforward to see.
• $H \subseteq \text{items}_T(\text{tids}_T(H))$. This is also pretty straightforward.

Therefore, the sets \text{items} and \text{tids} form a Galois Connection.

• $\text{tids}_T(\text{items}_T(E))$ is a closure on $\mathcal{P}(D)$.

• $\text{items}_T(\text{tids}_T(H))$ is a closure on $\mathcal{P}(I)$. (This is the more interesting property).

Suppose $H$ is closed. Then $H = \text{items}_T(\text{tids}_T(H))$ and $H \subset L$. (Note: proper subset).

We would like to prove that $\text{suppcount}_T(L) < \text{suppcount}_T(H)$.

For the sake of contradiction, suppose that $\text{suppcount}_T(H) = \text{suppcount}_T(L)$. That would imply

$$|\text{tids}_T(H)| = |\text{tids}_T(L)|$$

which in turn would imply $\text{tids}_T(H) \supseteq \text{tids}_T(L)$, and $\text{tids}_T(H) = \text{tids}_T(L)$.

Because, $H$ is closed

$$H = \text{items}_T(\text{tids}_T(H)) = \text{items}_T(\text{tids}_T(L))$$

This implies the closure of $L = H$, and contradicts $H \subset L$.

Therefore, if $H$ is closed, then any superset of $H$ has less support than $H$ does.

If every superset of $H$ has a smaller level of support, then this implies that $H$ is closed.

If $H$ is such that $H \subset L$, then $\text{suppcount}_T(L) < \text{suppcount}_T(H)$.

Suppose $U$ is an arbitrary set, and we take the closure of $U$:

$$\text{tids}_T(U) = \text{tids}_T(\text{items}_T(\text{tids}_T(U)))$$

Then

$$|\text{tids}_T(U)| = \text{suppcount}_T(U)$$

The support of a set is equal to the support of its closure.
1.7 Lecture – 2/13/2008

1.7.1 The FP-Tree Algorithm

The FP-Tree (Frequent Pattern Tree) algorithm is an alternative to Apriori. Apriori requires several scans of the data set, which can be expensive. FP-Tree, on the other hand, requires only two scans.

FP-Tree was invented by Jiawei Han.

FP-Tree uses a form of data compression. The algorithm’s primary data structure is a compressed representation of the transactional data set.

Basic idea: we build the FP-Tree, and then Mine it for patterns.

Rather than giving a formal definition for FP-Tree, we’ll do it by example.

Consider the following data set, over eight transactions and five data items.

<table>
<thead>
<tr>
<th></th>
<th>( i_1 )</th>
<th>( i_2 )</th>
<th>( i_3 )</th>
<th>( i_4 )</th>
<th>( i_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( T(2) )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( T(3) )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( T(4) )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( T(5) )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( T(6) )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( T(7) )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( T(8) )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We’re looking for frequent patterns with a minimum support count of 3.

**Step 1:** Scan the data set, and determine the support count for each item \( i_j \) (1 ≤ \( j \) ≤ \( n \)).

<table>
<thead>
<tr>
<th>item</th>
<th>support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_1 )</td>
<td>5</td>
</tr>
<tr>
<td>( i_2 )</td>
<td>6</td>
</tr>
<tr>
<td>( i_3 )</td>
<td>5</td>
</tr>
<tr>
<td>( i_4 )</td>
<td>2</td>
</tr>
<tr>
<td>( i_5 )</td>
<td>5</td>
</tr>
</tbody>
</table>

**Step 2:** Sort the items by decreasing support count.

<table>
<thead>
<tr>
<th>item</th>
<th>support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_2 )</td>
<td>6</td>
</tr>
<tr>
<td>( i_1 )</td>
<td>5</td>
</tr>
<tr>
<td>( i_3 )</td>
<td>5</td>
</tr>
<tr>
<td>( i_5 )</td>
<td>5</td>
</tr>
<tr>
<td>( i_4 )</td>
<td>2</td>
</tr>
</tbody>
</table>

**Step 3:** Scan the data set a second time. As we do this scan, we’ll construct the FP-Tree as follows:

- The root of the tree is \( \lambda \)
- For each transaction \( T(k) \), sort \( T(k) \)’s items by descending support count.
- Build a path from the root of the tree, where each node in the path corresponds to \( T(k) \)’s sorted item set.
- When adding a node to the FP-Tree, give it a count of 1. We traversing an existing node, increment its support count.

So, each FP-Tree node corresponds to an item \( i_j \), and each node is ornamented with the number of times \( i_j \) appeared on that particular path.
Below, we have our transactions, along with their sorted data sets:

<table>
<thead>
<tr>
<th>( T(k) )</th>
<th>item set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) )</td>
<td>( {i_2, i_1} )</td>
</tr>
<tr>
<td>( T(2) )</td>
<td>( {i_2, i_3} )</td>
</tr>
<tr>
<td>( T(3) )</td>
<td>( {i_1, i_5} )</td>
</tr>
<tr>
<td>( T(4) )</td>
<td>( {i_1, i_5} )</td>
</tr>
<tr>
<td>( T(5) )</td>
<td>( {i_2, i_3, i_5} )</td>
</tr>
<tr>
<td>( T(6) )</td>
<td>( {i_2, i_1, i_3, i_5, i_4} )</td>
</tr>
<tr>
<td>( T(7) )</td>
<td>( {i_2, i_1, i_3} )</td>
</tr>
<tr>
<td>( T(8) )</td>
<td>( {i_2, i_3, i_5, i_4} )</td>
</tr>
</tbody>
</table>

Figure 1.7 shows the corresponding FP-Tree.

**Step 4:** Along with the FP-Tree, create a two-column table where

- Column one contains the name of an item \( i_j \)
- Column two contains a pointer to a linked list that strings together all occurrences of \( i_j \) in the FP-Tree.

This table is shown in Figure 1.8.

One thing to take note of in Figure 1.8 - if you follow the pointer chain for a particular item, and add the counts together, then you'll get the support count for that item.

Also note: \( i_4 \) does not have sufficient support \((\mu = 3)\), so we can remove \( i_4 \) from the tree immediately.

In general, when we remove infrequent items from an FP-Tree, we'll be removing from the leaves up.

Figure 1.9 shows the FP-Tree with \( i_4 \) removed.

**Step 5:** Build Conditional FP-Trees
Step 5 is where we start mining the information in the FP tree.

From step 4, we see that \( i_5 \) has a support count of 5; therefore \( i_5 \) is a frequent item.

Our next task is to build a conditional FP Tree for \( i_5 \).

We note that \( i_5 \) appears on three paths:

- \( \{i_2, i_1, i_3, i_5\} \). \( i_5 \) leaf has a count of 1.
• \(\{i_2, i_3, i_5\}\). \(i_5\) leaf has a count of 2.
• \(\{i_1, i_5\}\). \(i_5\) leaf has a count of 2.

Next, we take the conditional prefix of these paths (all nodes up to, but not including \(i_5\)). We take the \(i_5\) count, and assign that to each prefix node.
• \(\{i_2, i_1, i_3\}\). All nodes have count 1.
• \(\{i_2, i_3\}\). All nodes have count 2.
• \(\{i_1\}\). All nodes have count 2.

Why do we assign counts in that manner? Consider \(\{i_2, i_3, i_5\}\). \(i_5\) has count two. Therefore, \(i_2\) occurs twice with \(i_5\) on this path, and \(i_3\) occurs twice with \(i_5\) on this path.

Once we have these conditional paths (and their associated counts), then we construct an FP-Tree from them. The FP-Tree construction is exactly what we’ve seen before, except for the fact that some nodes will have counts > 1.

The conditional FP-Tree for \(i_5\) is shown in Figure 1.10.

![Figure 1.10: Conditional FP-Tree for \(i_5\).](image)

The cross-link table in Figure 1.10 shows three items: \(\{i_2, i_1, i_3\}\). There is implicitly an \(i_5\) included in each of these, so we’re really looking at three two-item sets: \(\{i_2, i_5\}\), \(\{i_1, i_5\}\), and \(\{i_3, i_5\}\).

As before, we can sum counts across the pointer chain to find support counts:

<table>
<thead>
<tr>
<th>itemset</th>
<th>support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>({i_2, i_5})</td>
<td>3</td>
</tr>
<tr>
<td>({i_1, i_5})</td>
<td>3</td>
</tr>
<tr>
<td>({i_3, i_5})</td>
<td>3</td>
</tr>
</tbody>
</table>

We can continue, this process, to generate a conditional FP tree for \(\{i_3, i_5\}\).

\(i_3\) gives us two prefix paths:
• \(\{i_2, i_1\}\). Each node has a count of 1.
• \(\{i_2\}\). Each node has a count of 2.

The conditional FP-tree for \(\{i_3, i_5\}\) is shown in Figure 1.11.

From Figure 1.11 we can find support counts for 3-sets.
Of course, \( \{i_1, i_3, i_5\} \) is not frequent, so we wouldn’t include it in the list of frequent item sets. \( \{i_2, i_3, i_5\} \) is frequent. However, a conditional FP-Tree for \( \{i_2, i_3, i_5\} \) has only a \( \lambda \) node, so there’s no sense in continuing along that path.

1.7.2 Vertical Database Formats

So far, we’ve looked at data sets where transactions are represented by rows, and items are represented by columns. This arrangement is called a horizontal format.

There is also a vertical format, where rows represent items. For each item, we keep a list of tids that purchased that item. For example:

<table>
<thead>
<tr>
<th>item</th>
<th>transaction set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_1 )</td>
<td>{T(1), T(3), T(4), T(6), T(7)}</td>
</tr>
<tr>
<td>( i_2 )</td>
<td>{T(1), T(2), T(5), T(6), T(7), T(8)}</td>
</tr>
<tr>
<td>( i_3 )</td>
<td>{T(2), T(5), T(6), T(7), T(8)}</td>
</tr>
<tr>
<td>( i_4 )</td>
<td>{T(6), T(8)}</td>
</tr>
<tr>
<td>( i_5 )</td>
<td>{T(3), T(4), T(5), T(6), T(8)}</td>
</tr>
</tbody>
</table>

Note that this format immediately gives us the support count for each single item.

Of course, we can use separate columns for each transaction.

<table>
<thead>
<tr>
<th></th>
<th>T(1)</th>
<th>T(2)</th>
<th>T(3)</th>
<th>T(4)</th>
<th>T(5)</th>
<th>T(6)</th>
<th>T(7)</th>
<th>T(8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_1 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( i_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( i_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( i_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( i_5 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

This representation has a nice property. With each item \( i_j \), we associate a bitmap that represents the set of transactions purchasing \( i_j \).

Suppose we want to find the support count of a two-set \( \{i_j, i_k\} \). We can do this by AND-ing a pair of bitmaps — this is a very fast operation.

The vertical format was invented by M. Zaki, in a data mining algorithm called ECLAT.
1.7.3 Association Rules

Frequent item sets can be used to find association rules. This is one of the main reasons for finding frequent item sets.

Definition 1.7.3.1 (Association Rule): An association rule is a pair of item sets $X, Y$ such that $X \cap Y = \emptyset$. If $Z$ is a frequent item set, then $Z = X \cup Y$.

Association rules must satisfy two criteria:

\[
\begin{align*}
\text{support}_T(XY) & \geq \mu & \mu & \text{is the minimum support count} \\
\frac{\text{support}_T(XY)}{\text{support}_T(X)} & \geq c & c & \text{is the minimum confidence}
\end{align*}
\]

Association rules are written $X \Rightarrow Y$.

$\text{support}_T(XY)$ is the support of $X \Rightarrow Y$.

$\frac{\text{support}_T(XY)}{\text{support}_T(X)}$ is the confidence of $X \Rightarrow Y$.

Confidence values satisfy the inequality

$0 \leq \text{conf}_T(X \Rightarrow Y) \leq 1$

Confidence works much like conditional probability:

\[
P(Y|X) = \frac{P(XY)}{P(X)}
\]

Suppose we are seeking a set $Z$ such that $Z$ is frequent: $\text{support}_T(Z) \geq \mu$.

To find association rules with high confidence, we need to find $X \subset Z$ such that $\text{support}_T(X)$ is small.

\[
\frac{\text{support}_T(Z)}{\text{support}_T(X)} \geq c
\]

If an association rule $X \Rightarrow Z - X$ has support $\geq c$, then we refer to it as a strong rule.

For an itemset, how many association rules are there? Short answer: lots!

Let $Z = X \cap Y$, where $k = |X|$ and $n = |Z|$.

There are $\binom{n}{k}$ ways to choose $X$, and $2^{n-k}$ ways to choose $Y$. Of course, $k$ can also vary from $1 \leq k \leq n - 1$. The total number of association rules derivable from $Z$ is

\[
\sum_{k=1}^{n-1} \binom{n}{k} \cdot 2^{n-k}
\]

An approximation of this value:

\[
(1 + x)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^{n-k}
\]

a recurrence relation

\[
3^n = 2^n + 1 + \sum_{k=1}^{n-1} \binom{n}{k} \cdot 2^{n-k}
\]

taking $x = 2$

\[
\approx 3^n - 2^n - 1
\]

By choosing appropriate support and confidence levels, we can prevent an explosion of (meaningless) association rules.
1.7.4 Compression and Data Patterns

If a set of data has any kind of embedded structure, then compression will reduce the size of the data. The more structure, the larger the reduction tends to be. Zip, zlib, etc all rely on this property.

If a set of data compresses well, does this mean that it is more likely to contain interesting patterns than a set of data which does not compress well?

One might hypothesize this to be the case, but it is an open question.
1.8 FP-Trees – 2/16/2008

Helpful Pointers

Here's a nice paper about FP-Trees: [http://doi.acm.org/10.1145/335191.335372](http://doi.acm.org/10.1145/335191.335372)

@article{335372,
  author = {Jiawei Han and Jian Pei and Yiwen Yin},
  title = {Mining frequent patterns without candidate generation},
  journal = {SIGMOD Rec.},
  volume = {29},
  number = {2},
  year = {2000},
  issn = {0163-5808},
  pages = {1--12},
  doi = {http://doi.acm.org/10.1145/335191.335372},
  publisher = {ACM},
  address = {New York, NY, USA},
}
1.9 Lecture – 2/20/2008

1.9.1 Optimizations for the Apriori Algorithm

We’ve studied the Apriori algorithm in prior lectures. Apriori is the basis for many algorithms, but it does have a disadvantage: it requires repeated scans of the data set (lots of IO). Fortunately, there are ways to (a) allow apriori to handle larger data sets and (b) make apriori more efficient.

Horizontal Partitionings

Horizontal partitioning won’t make apriori faster, but it will allow you to handle larger data sets. The basic procedure is as follows:

- Split the data set into \( n \) (horizontal) pieces.
- Find frequent items in each individual piece
- Assemble the final set of frequent items.

If \( H \) is frequent in the database as a whole, then \( H \) must be frequent in at least one of the horizontal partitions.

Why is this true? Suppose we have \( p_1, \ldots, p_n \) and \( q_1, \ldots, q_n \), where all \( p_i \) and \( q_i \) are \( \geq 0 \). We have the following inequality

\[
\min \left\{ \frac{p_1}{q_1}, \ldots, \frac{p_n}{q_n} \right\} \leq \frac{p_1 + \cdots + p_n}{q_1 + \cdots + q_n} \leq \max \left\{ \frac{p_1}{q_1}, \ldots, \frac{p_n}{q_n} \right\}
\]

(1.18)

Let

\[
M = \max \left\{ \frac{p_1}{q_1}, \ldots, \frac{p_n}{q_n} \right\}
\]

(1.19)

We know that

\[
\frac{p_i}{q_i} \leq M \quad \text{for } 1 \leq i \leq n
\]

(1.20)

Therefore,

\[
p_i \leq M \cdot q_i \quad \text{for } 1 \leq i \leq n
\]

(1.21)

and

\[
\frac{p_1 + \cdots + p_n}{q_1 + \cdots + q_n} \leq M
\]

(1.22)

\[
\frac{p_1 + \cdots + p_n}{q_1 + \cdots + q_n} \leq M
\]

(1.23)

In terms of horizontal partitioning, that tells us

\[
\frac{\text{supp}_{\text{DB}}(H)}{\text{dbsize}} \leq \max \left\{ \frac{\text{supp}_{\text{DB}}(p_1)}{s_1}, \ldots, \frac{\text{supp}_{\text{DB}}(p_n)}{s_n} \right\} \geq M
\]

(1.24)

Horizontal partitioning allows us to process the data in chunks.

1.9.2 Speeding Up Apriori

Let’s look at three techniques for making Apriori faster.
Sampling

Instead of analyzing the entire data set, we can mine a sample of it. In this case, we are sampling transactions— all columns are included. We’ll use results from the sample to draw conclusions about the full data set.

The danger in sampling: the sample chosen may not accurately reflect the data set as a whole.

Pruning Transactions

We can speed up Apriori by pruning transactions that do not contain \( k \)-frequent itemsets. For example, if \( T(k) \) does not contain a frequent 2-itemset, then there’s no sense in looking for a frequent 3-itemset in \( T(k) \).

This approach allows passes to go successively faster, and it doesn’t compromise accuracy.

Hashing Schemes

We can use hashing to search for several \( k \)-frequent itemsets at once. (In this context, we mean “several values of \( k \)).

Suppose we are looking for \( \mu = 0.5 \) frequent itemsets in the following data:

<table>
<thead>
<tr>
<th>( T(k) )</th>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) )</td>
<td>( {i_1, i_2} )</td>
</tr>
<tr>
<td>( T(2) )</td>
<td>( {i_2, i_3} )</td>
</tr>
<tr>
<td>( T(3) )</td>
<td>( {i_1, i_5} )</td>
</tr>
<tr>
<td>( T(4) )</td>
<td>( {i_1, i_5} )</td>
</tr>
<tr>
<td>( T(5) )</td>
<td>( {i_2, i_3, i_5} )</td>
</tr>
<tr>
<td>( T(6) )</td>
<td>( {i_1, i_2, i_3, i_4, i_5} )</td>
</tr>
<tr>
<td>( T(7) )</td>
<td>( {i_1, i_2, i_3} )</td>
</tr>
<tr>
<td>( T(8) )</td>
<td>( {i_2, i_3, i_4, i_5} )</td>
</tr>
</tbody>
</table>

In this example, we’ll find frequent 1-itemsets and 2-itemsets in a single pass.

The trick: we apply a hashing function. Here, we’ll use

\[ h(i_p, i_q) = (10 \times p) + q \mod 7 \]

For each itemset, we’ll compute \( h(i_p, i_q) \) for all pairs of item. For example, given \( \{i_1, i_2, i_3\} \), we’ll hash

\[ h(i_1, i_2) = (10 \times 1 + 2) \mod 7 = 5 \]
\[ h(i_1, i_3) = (10 \times 1 + 3) \mod 7 = 6 \]
\[ h(i_2, i_3) = (20 + 3) \mod 7 = 2 \]

Since our hash function is \( \mod 7 \), we’ll need seven buckets. Each bucket will have (a) a count of the number of items in that bucket, and (b) a list of the 2-sets.

<table>
<thead>
<tr>
<th>bucket</th>
<th>count</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>{i_3, i_5}, {i_1, i_4}, {i_3, i_5}, {i_3, i_5}</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>{i_1, i_5}, {i_1, i_5}, {i_1, i_5}</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>{i_2, i_3}, {i_2, i_3}, {i_2, i_3}, {i_2, i_3}, {i_2, i_3}</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>{i_2, i_4}, {i_4, i_5}, {i_2, i_4}, {i_4, i_5}</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>{i_2, i_5}, {i_2, i_5}</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>{i_1, i_2}, {i_1, i_2}, {i_1, i_2}</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>{i_1, i_3}, {i_3, i_4}, {i_1, i_3}, {i_3, i_4}</td>
</tr>
</tbody>
</table>
Since we are looking for a support count of $\geq 4$, we can immediately throw out any bucket with less than four items. From there, we’d go through the remaining buckets and figure out the support counts of the other 2-sets.

### 1.9.3 Measures of Interestingness

Often, we find frequent itemsets in order to find association rules. For even a modest number of frequent itemsets, we can generate a huge number of association rules. How can we better understand the association rules? Also, how can we find association rules that are more “interesting”?

#### Interestingness of a Rule

Suppose we have $X \Rightarrow Y$. We already know of two measures:

- $\text{support}_T(X \Rightarrow Y) = \text{support}_T(XY)$
- $\text{conf}_T(X \Rightarrow Y) = \frac{\text{support}_T(XY)}{\text{support}_T(X)}$

When finding association rules, we’re typically look for those which satisfy a minimum confidence $c$ and minimum support $\mu$.

Suppose we treat $X$ and $Y$ as random variables where

$$I_X = \begin{cases} 1 & \text{if a customer buys } X \\ 0 & \text{otherwise} \end{cases}$$

$$I_Y = \begin{cases} 1 & \text{if a customer buys } Y \\ 0 & \text{otherwise} \end{cases}$$

For strong support, $I_X$ and $I_Y$ need not be correlated. $I_X$ and $I_Y$ are referred to as indicator variables.

$I_X$ has a value distribution of

$$I_X : \begin{pmatrix} 1 & 0 \\ p & 1 - p \end{pmatrix}$$

(The top row represents values. The bottom row represent probabilities).

#### A Case Study In Correlation

*This example comes from Section 5.4.1 of the text. Our class example used mustard and sausages instead of games and videos, but the presentation was basically the same.*

Suppose we have 10,000 transactions where

- 6,000 customers purchase computer games
- 7,500 customers purchase videos
- 4,000 customers purchase both
We have
\[
\text{support}_T(\text{games} \Rightarrow \text{videos}) = \frac{4,000}{10,000} = 0.4
\]
\[
\text{conf}_T(\text{games} \Rightarrow \text{videos}) = \frac{4,000}{6,000} = 0.66
\]

\(\text{games} \Rightarrow \text{videos}\) is a strong rule (high support and confidence). But it’s also misleading. Support is only an estimate of correlation between \(\text{games}\) and \(\text{videos}\).

To address this weakness, we can use (several) correlation measures. One example is \(\text{lift}\):
\[
\text{lift}(X,Y) = \frac{P(X \cup Y)}{P(X) \cdot P(Y)} = \frac{P(Y|X)}{P(Y)} = \frac{\text{conf}_T(X \Rightarrow Y)}{\text{support}_T(Y)}
\]

Lift measures the amount by which an increase in \(X\) increases \(Y\). For our video/game example:
\[
\text{lift}(\text{games, videos}) = \frac{\text{conf}_T(\text{games} \Rightarrow \text{videos})}{\text{support}_T(\text{videos})} = \frac{0.66}{0.75} = 0.88
\]

In this case, lift is < 1, which indicates a negative correlation.

In general:
- If \(\text{lift}(X,Y) < 1\), then \(X\) and \(Y\) are \textit{negatively} correlated.
- If \(\text{lift}(X,Y) > 1\), then \(X\) and \(Y\) are \textit{positively} correlated.
- If \(\text{lift}(X,Y) = 1\), the \(X\) and \(Y\) are independent.

This is pretty easy to understand with a little manipulation:
\[
\text{lift}(X,Y) = \frac{P(X \cup Y)}{P(X) \cdot P(Y)}
\]
\[
\text{lift}(X,Y) \cdot (P(X)P(Y)) = P(XY)
\]

Lift is kind of like a scaling factor that associates the independent probabilities to the join probability.

\textit{Note:} in class we got a different definition of lift; the reciprocal of what’s in the text:
\[
\text{lift}(X,Y) = \frac{P(X) \cdot P(Y)}{P(X \cup Y)}
\]
**Indicator Variables**

If $X$ as an event, then the indicator variable for $X$ is

$$I_X = \begin{cases} 
1 & \text{if } X \text{ occurs} \\
0 & \text{otherwise}
\end{cases}$$

If $P(X) = p$, then

$$I_X : \begin{pmatrix} 
1 \\
p \\
1-p 
\end{pmatrix}$$

Suppose $I_X$ follows a binomial distribution.

$$E(I_X) = p \quad \text{the expected probability}$$

Variance is

$$\text{var}(I_X) = E((I_X - E(I_X))^2)$$

Covariance is

$$\text{covar}(I_X, I_Y) = E(I_X I_Y) - E(I_X)E(I_Y)$$

$$= P(XY) - P(X)P(Y)$$

Correlation is

$$\text{corr}(I_X, I_Y) = \frac{\text{covar}(I_X I_Y)}{\sqrt{\text{var}(I_X)\text{var}(I_Y)}}$$

**Multi-Level Association Rules**

Suppose we have a database of courses and students:

<table>
<thead>
<tr>
<th>S1</th>
<th>C1</th>
<th>C2</th>
<th>...</th>
<th>C500</th>
</tr>
</thead>
</table>

And, we'd like to find association rules of the form $X_1X_2 \Rightarrow Y_1Y_2Y_3$: if a student takes $X$ courses, what $Y$ courses to they take?

With 500 courses, there are a huge number of rules: $\binom{500}{2} \cdot \binom{498}{3}$.

Aside from the sheer number of rules, the level of detail is also hard to manage.

However, we can classify courses according to a hierarchy – computer science courses, physics courses, management courses, etc. This gives us a concept hierarchy like the one shown in Figure 1.12.

When mining, we can replace courses with their classifications. For example, “CS110” becomes “Computer Science”, and we do the mining with “Computer Science”.

These are called multi-level rules.

Going from the leaves to the root, we typically expect the level of support to increase.
Figure 1.12: Concept Hierarchy for University Courses
1.10 Notes from Handout #2 – 2/25/2008

1.10.1 Association Rules

For the association rule $X \Rightarrow Y$,
- Both $X$ and $Y$ must be nonempty: $X \neq \emptyset$, $Y \neq \emptyset$.
- $X$ and $Y$ must be disjoint: $X \cap Y = \emptyset$.

An association rule holds if we have $\text{support}_T(X \Rightarrow Y) \geq \mu$ and $\text{conf}_T(X \Rightarrow Y) \geq c$. Rules must meet our requirements for support and confidence.

Given a Frequent item set $Z$, we find association rules by examining all proper subsets $X$ of $Z$. Rules have the form

$$X \Rightarrow Z - X$$

such that $X \Rightarrow Z - X$ has the desired confidence $c$ and support $\mu$.

An association rules with 100% confidence is referred to as an exact association rule.
1.11 Lecture – 2/25/2008

1.11.1 Clarifications on Lift

(This is a second presentation of the material given at the end of our last lecture. See page 35 or so).

For an association rule $X \Rightarrow Y$, we define lift as

$$\text{lift}(X, Y) = \frac{\text{support}_T(XY)}{\text{support}_T(X) \cdot \text{support}_T(Y)}$$  \hfill (1.25)

Covariance is

$$\text{covar}(I_X, I_Y) = E(I_X, I_Y) - E(I_X)E(I_Y)$$  \hfill (1.26)

Say we have an event $X$, and $X$ occurs with probability $p$. The indicator variable $I_X$ is

$$I_X : \begin{pmatrix} 1 & 0 \\ p & 1 - p \end{pmatrix}$$  \hfill (1.27)

The expected value of $I_X$ is $E(I_X) = p$.

The variance of $I_X$ is

$$\text{var}(I_X) = p \cdot (1 - p)$$  \hfill (1.28)

The support of an itemset $X$ is the probability of having $X \subseteq T(i)$ for a randomly chosen transaction $T(i)$.

Suppose we have

- $\text{support}_T(X) = p$
- $\text{support}_T(Y) = q$
- $\text{support}_T(XY) = r$

The correlation of $I_X$ and $I_Y$ is

$$\text{corr}(I_X, I_Y) = \frac{\text{covar}(I_X, I_Y)}{\sqrt{\text{var}(I_X) \cdot \text{var}(I_Y)}}$$  \hfill (1.29)

If $I_{XY}$ is

$$I_{XY} : \begin{pmatrix} 1 & 0 \\ r & 1 - r \end{pmatrix}$$  \hfill (1.30)

then,

$$\text{corr}(I_X, I_Y) = \frac{r - pq}{\sqrt{p(1 - p) \cdot q(1 - q)}} = \frac{pq \cdot \text{lift}(X \Rightarrow Y) - 1}{\sqrt{p(1 - p) \cdot q(1 - q)}}$$

note that lift is $\frac{r}{pq}$

If lift$(X \Rightarrow Y) > 1$, then we have positive correlation.

If lift$(X \Rightarrow Y) < 1$, then we have negative correlation.
Part 2

Clustering

2.1 Lecture – 2/25/2008

Clustering groups similar objects into sets named clusters, such that dissimilar objects belong to different sets.

This definition is somewhat vague. For example, it doesn’t say how dissimilarity is measured. It also doesn’t say if clusters can overlap, or if they have to be disjoint.

2.1.1 Measuring Dissimilarity

Suppose we have two objects, \( o_1, o_2 \in S \), and we want to measure the dissimilarity between \( o_1 \) and \( o_2 \).

Formally, a dissimilarity measure is a function

\[
d : S \times S \to \mathbb{R}_{\geq 0}
\]  

(2.1)

So, \( d(o_1, o_2) \) is the dissimilarity between \( o_1 \) and \( o_2 \).

Requirements for a dissimilarity function:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(DISS1)</td>
<td>( d(o, o) = 0 )</td>
</tr>
<tr>
<td>(DISS2)</td>
<td>( d(o_1, o_2) = d(o_2, o_1) )</td>
</tr>
</tbody>
</table>

These are the minimum requirements for a dissimilarity measure. It’s common to impose two additional requirements:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(DISS3)</td>
<td>( d(o_1, o_2) \leq d(o_1, o') + d(o', o_2) )</td>
</tr>
<tr>
<td>(DISS4)</td>
<td>( d(o_1, o_2) = 0 ) implies ( o_1 = o_2 )</td>
</tr>
</tbody>
</table>

(DISS3) is the triangular axiom.

If \( d \) satisfies (DISS1), (DISS2), and (DISS3), then we call \( d \) a semi-metric.

If \( d \) satisfies (DISS1)–(DISS4), then \( d \) is a metric.

For clustering, we’ll typically prefer metrics or semi-metrics.
Data That is Typically Clustered

- Vectors in $\mathbb{R}^n$
- Sets which are subsets of a given set. The representations used for sets tend to work for general binary data (i.e., you can represent a set as a bit string)
- Sequences of symbols over a given alphabet
- Categorical data. Categorical data is data without a natural value order (e.g., color).

2.1.2 Examples of Dissimilarity Measures

You can think of dissimilarity measures as distance measures – we’re measuring how “far apart” two objects are.

Euclidean Distance

Let $x$ and $y$ be two vectors:

$$x = (x_1, \ldots, x_n)$$

$$y = (y_1, \ldots, y_n)$$

The euclidean distance between $x$ and $y$ is

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$  \hspace{1cm} (2.2)

This is a fairly common distance measure.

Minkovski Distance

Minkovski distance is

$$d_p(x, y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$  \hspace{1cm} (2.3)

Minkovski distance is a metric for $p \geq 1$.

Note that $d_2(x, y)$ (taking $p = 2$) is the same as the Euclidean distance.

Manhattan Distance

The Manhattan Distance is Minkovski distance for $p = 1$

$$d_1(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$  \hspace{1cm} (2.4)

How would this look for vectors of length 2?

$$d_1(x, y) = |x_1 - y_1| + |x_2 - y_2|$$
Figure 2.1: Manhattan Distance

Figure 2.1 shows the Manhattan distance between \( x \) and \( y \). The distance is the length of the dashed line between \( x \) and \( y \).

Why is this called Manhattan Distance? If you were in Manhattan and wanted to go from \( x \) to \( y \), you’d have to walk along a path like the dotted line.

**Canberra Distance**

The Minkovski distance does something interesting as \( p \) approaches infinity. Let’s look at \( \lim_{p \to \infty} d_p(x, y) \).

Starting with

\[
d_p(x, y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}
\]

Let’s factor out the largest \( |x_i - y_i| \) value:

\[
d_p(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \cdot \left( 1 + \sum r_j^p \right)^{\frac{1}{p}}
\]

Above, \( r_j \) has the form \( |x_j - y_j| \) (but not the maximum value). All \( r_j \) are \( \leq 1 \).

As \( p \) tends to infinity, \( \frac{1}{p} \) tends to zero, and \( \left( \sum r_j^p \right) \) tends to 1.

So, we have the Canberra Distance:

\[
d_\infty = \max_{1 \leq i \leq n} |x_i - y_i|
\]

(2.5)

**2.1.3 Spheres and Distance Measures**

We can define spheres in terms of \( d_p(x, y) \) measures.

If \( c \) is the center of the sphere and \( r \) is the radius, then we can define a sphere as

\[
S_d_2(c, r) = \{ x \mid d(c, x) \leq r \}
\]

(2.6)

\( d_2(x, y) \) corresponds to the way we normally think of a sphere. The outer shell has distance \( r \) from \( c \).

Figure 2.2 shows a sphere for \( S_d_2(0, r) \).

For \( d_1(x, y) \) our sphere is

\[
S_d_1(c, r) = \{(x_1, x_2) \mid |x_1| + |x_2| \leq r \}
\]

(2.7)
Figure 2.2: Sphere defined by $d_2(0, r)$

Figure 2.3: Sphere defined by $d_1(0, r)$

This is shown in Figure 2.3.

For $d_\infty(x, y)$ our sphere is

$$S_{d_\infty}(0, r) = \{ (x_1, x_2) \mid \max \{x_1, x_2\} \leq r \}$$

(2.8)

This sphere is shown in Figure 2.4.

Figure 2.4: Sphere defined by $d_\infty(0, r)$

Moral of the story – the way we intuitively think of things in two or three dimensions does not hold for larger numbers of dimensions!

2.1.4 Dissimilarity Measures for Sets

Most dissimilarity measures for sets will also work for binary data. When working with binary data, 1 is like being in the set, and 0 is like being out of the set.

A common dissimilarity measure for sets is symmetric difference

$$d(P,Q) = |P \oplus Q|$$

$$= |(P - Q) \cup (Q - P)|$$

In Figure 2.5, the symmetric difference is represented by the shaded area. Note that $P \cap Q$ is not part of the symmetric difference.
Is \(d(P, Q)\) a similarity? It’s pretty easy to see that

\[
\begin{align*}
d(P, P) &= 0 \\
d(P, Q) &= d(Q, P)
\end{align*}
\]

But \(d(P, Q) \leq d(P, R) + d(R, Q)\) – that’s a little trickier to prove.

First, we note that \(\oplus\) is associative: \(P \oplus P = \emptyset\)

Now,

\[
\begin{align*}
d(P, Q) &\leq d(P, R) + d(R, Q) \\
|P \oplus Q| &\leq |P \oplus R| + |R \oplus Q|
\end{align*}
\]

This is what we need to prove

\[
P \oplus Q = (P \oplus R) \oplus (R \oplus Q)
\]

\[
\subseteq (P \oplus R) \cup (R \oplus Q)
\]

therefore

\[
|P \oplus Q| \leq |P \oplus R| + |R \oplus Q|
\]

**Steinhaus Transform of \(d\)**

Suppose we pick a fixed point \(u\). The Steinhaus Transform of \(d\) is

\[
d_u(x, y) = \frac{d(x, y)}{d(x, y) + d(x, u) + d(u, y)}
\]

(2.9)

Above, we’ll have \(0 \leq d_u(x, y) \leq 1\).

\(d_u(x, y)\) is a dissimilarity, since

\[
\begin{align*}
d_u(x, x) &= 0 \\
d_u(x, y) &= d_u(y, x)
\end{align*}
\]

Let’s prove that \(d_u(x, y)\) also satisfies the triangular axiom.

First observe that if \(a \leq b\), then

\[
\frac{a}{a + k} \leq \frac{b}{b + k} \quad \text{for } k \geq 1
\]

Why?

\[
ab + ak \leq ba + bk \quad \text{cross multiply}
\]

\[
ak \leq bk
\]
Let's look at
\[ d(x, y) \leq d(x, z) + d(z, y) \]

Applying the definition of \( d_u(x, y) \):
\[
d_u(x, y) = \frac{d(x, y)}{d(x, z) + d(z, y)} \leq \frac{d(x, y)}{d(x, z) + d(z, y) + d(x, u) + d(u, y)}
\]
\[
= \frac{d(x, z)}{d(x, z) + d(z, y) + d(x, u) + d(u, y)} + \frac{d(z, y)}{d(x, z) + d(z, y) + d(x, u) + d(u, y)}
\]
\[
\leq \frac{d(x, z) + d(x, u) + d(u, y)}{d(x, z) + d(x, u) + d(u, y)} + d(z, y) + d(z, y) + d(u, y)
\]
\[
= d_u(x, z) + d_u(z, y)
\]

Note: going from the third line to the fourth line, we make use of our earlier observation: \( \frac{a}{a+k} \leq \frac{b}{b+k} \) for \( a \leq b \) and \( k \geq 1 \).

We can apply the Steinhaus Transform to our set dissimilarity measure:
\[
d(x, y) = |x \oplus y|
\]
\[
d_u(x, y) = \frac{|x \oplus y|}{|x \oplus y| + |x \oplus u| + |u \oplus y|}
\]
Suppose we choose \( u = \emptyset \).
\[
d_u(x, y) = \frac{|x \oplus y|}{|x \oplus y| + |x| + |y|} = \frac{|x \oplus y|}{2 \cdot |x \cup y|}
\]
This measure satisfies the triangular axiom, and takes values.
\[
0 \leq d_u(x, y) = \frac{|x \oplus y|}{2 \cdot |x \cup y|} \leq 0.5
\]

A variation on this, called the Jaccard Coefficient
\[
\frac{|x \oplus y|}{|x \cup y|}
\]
takes values in the range \([0.0, 1.0]\).

Finally, we can also measure dissimilarity relative to the set \( S \) itself. This is like the Steinhaus Transform, but we're using the set \( S \) instead of a single element \( u \).
\[
d_S(x, y) = \frac{|x \oplus y|}{|x \oplus y| + |x| + |y|} = \frac{|x \oplus y|}{2|S - (x \cap y)|}
\]

2.1.5 Logistics

- Download R and WEKA. Start getting familiar with them.
- Our makeup class will tentatively be the first Saturday after spring break. We'll finalize the date later.
2.2 Lecture – 2/27/2008

A dissimilarity needs to satisfy two requirements:

\[ d(x, x) = 0 \]  
\[ d(x, y) = d(y, x) \]  

(2.11)
(2.12)

However, it’s common to impose additional requirements:

\[ d(x, y) \leq d(x, z) + d(z, y) \]  
\[ d(x, y) = 0 \text{ implies } x = y \]

triangular inequality
definedness property

(2.13)
(2.14)

- If \( d \) satisfies (2.11) and (2.12), then \( d \) is a dissimilarity.
- If \( d \) satisfies (2.11), (2.12), and (2.13), then \( d \) is a semi-metric (also called a quasi-metric)
- If \( d \) satisfies (2.11), (2.12), (2.13), and (2.14), then \( d \) is a metric.

Let \( d \) be a dissimilarity (not a semi-metric). We define \( D_\alpha \) as

\[ D_\alpha(x, y) = d(x, y)^\alpha \]

There is always an \( \alpha > 0 \) such that \( D_\alpha \) is a semi-metric.

We won’t give a proof here. For \( \alpha = 0 \), it is trivially true. We can also find \( \alpha \) by

\[ \alpha = \min_{x, y \in S} d(x, y) \]

Such that \( d(x, y)^\alpha \leq d(x, z)^\alpha + d(z, y)^\alpha \) for every \( z \in S \).

\( \alpha \) tells us how much \( d \) behaves like a semi-metric. (semi-metrics have \( \alpha = 1 \).)

2.2.1 Ultrametrics

An ultrametric is a function

\[ d: S \times S \to \mathbb{R}_{\geq 0} \]

Ultrametrics satisfy three properties:

\[ d(x, y) = 0 \text{ iff } x = y \]  
\[ d(x, y) = d(y, x) \]  
\[ d(x, y) \leq \max\{d(x, z), d(z, y)\} \]

(2.15)
(2.16)
(2.17)

Equation (2.17) is referred to as the ultrametric inequality. The ultrametric inequality implies the triangular inequality.

Suppose we have a tree where all leaves are equidistant from the root – an equidistant tree. The length of the path between any two trees is an ultrametric.

Consider the tree in Figure 2.6

We have

\[ d(x, y) = d(x, u) + d(u, y) \]  
\[ d(x, z) = d(x, u) + d(u, v) + d(v, z) \]  
\[ d(y, z) = d(y, u) + d(u, v) + d(v, z) \]
In Figure 2.6, \( a \) and \( b \) represent distances. So

\[
\begin{align*}
  d(x, u) &= d(u, y) \\
  d(v, x) &= d(v, z) \\
  d(x, y) &= 2a \\
  d(x, z) &= 2a + 2b \\
  d(y, z) &= 2a + 2b \\
  d(x, y) &\leq \max\{d(x, z), d(z, y)\}
\end{align*}
\]

Notice that two of our three distances, \( d(x, y) \), \( d(x, z) \), and \( d(y, z) \), are the same. This always happens with an ultrametric.

Every triangle in an ultrametric-space is an isosceles triangle. Consider Figure 2.7.

There are three distances: \( d(p, q) \), \( d(p, r) \), and \( d(q, r) \). Let \( d(p, q) \) be the smallest distance. If \( d(p, q) \) is the smallest, then

\[
\begin{align*}
  d(p, q) &\leq d(p, r) \\
  d(p, q) &\leq d(q, r)
\end{align*}
\]

Using the ultrametric property,

\[
\begin{align*}
  d(p, r) &\leq \max\{d(p, q), d(q, r)\} = d(q, r) \quad \text{since } d(p, q) \text{ is smallest} \\
  d(q, r) &\leq \max\{d(p, q), d(p, r)\} = d(p, r) \quad \text{since } d(p, q) \text{ is smallest}
\end{align*}
\]

So, \( d(p, r) \leq d(q, r) \leq d(p, r) \). Therefore \( d(p, r) = d(q, r) \).

### 2.2.2 Clustering

When it comes to clustering, there is no free lunch. There are lots of different approaches – each are suited to different tasks, or to different kinds of data.
A Taxonomy of Clustering Algorithms

Clustering algorithms may be

**Exclusive or Non-Exclusive** Exclusive algorithms do not allow groups to overlap. Non-Exclusive algorithms allow groups to overlap (typically at the periphery of groups).

**Supervised or Unsupervised** Unsupervised algorithms have no operator involvement. Supervised algorithms allow an operator to specify constraints for the clustering process. For example, a constraint might be “a and b cannot be part of the same cluster” or “a and b must be part of the same cluster”.

There are also semi-supervised clustering algorithms.

In addition to these categories, there are also

- Hierarchical algorithms. Some work bottom-up; some work top-down.
- Means-based (and medoid-based) algorithms.
- Density-based algorithms.

Some Clustering Formalisms

For clarity, we should define a few terms.

**Definition 2.2.2.1 (Clustering):** A clustering on a set of objects is partition of that set of objects.

**Definition 2.2.2.2 (Partition):** A partition on a set $S$ is a finite collection of non-empty subsets of $S$: $\{B_1, B_2, \ldots, B_n\}$, such that

$$B_1 \cup B_2 \cup \ldots \cup B_n = S$$

$$B_i \cap B_j = \emptyset \text{ if } i \neq j$$

The $B_i$ are called blocks of the partition. A partition is a collection of blocks.

Figure 2.8 shows an example of a partition.

![Figure 2.8: A partition with six blocks](image)

For a given set $S$, there are a large number of possible clusterings (Stirling’s number).

Let $\text{parts}(S)$ be the set of all partitions on $S$. How can we define an ordering on clusters?

Let $\pi, \delta \in \text{parts}(S)$. We say that $\pi < \delta$ if every block of $\pi$ is included in a block of $\delta$. In other words, blocks of $\delta$ are unions of blocks of $\pi$.

Let $c$ be a block of $\delta$:

$$c = \{x_{i_1}, \ldots, x_{i_p}\}$$

such that

$$x_{i_1} \in B_{i_1}, \ldots, x_{i_p} \in B_{i_p}$$
Then $c \subseteq \bigcup_{i=1}^{p} B_i$, and each $B_i \in c$.

The smallest partition has one element per block. The largest partition is a single block that contains all elements.

Let $S = \{x_1, \ldots, x_n\}$.

$$\alpha_S = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\} \quad \text{the smallest partition}$$

$$\omega_S = \{S\} \quad \text{the largest partition}$$

### 2.2.3 Hierarchical Clustering Algorithms

Hierarchical clustering algorithms build clusters with either an increasing, or a decreasing sequence of partitions. In other words, hierarchical algorithms start with $\alpha_S$ or $\omega_S$. The algorithms produce a sequence of clusters (typically, there will be a parameter that says when to stop).

Hierarchical algorithms can be

- **Agglomerative** These start with $\alpha_S$ and build larger clusters. In R, agglomerative clustering is done by `agnes`.
- **Divisive** These start with $\omega_S$ and divide it into smaller clusters. In R, divisive clustering is done by `diana`.

Aside – in older literature, clusters are sometimes called “OTUs” – OTU stands for “Operational Taxonomical Unit”. This term comes from the field of biology.

Hierarchical clustering uses a matrix of distances (between clusters). Given a set $S$ with $n$ elements, an agglomerative algorithm starts with an $n \times n$ matrix $M$. $M$ is symmetric, and has zeros along the diagonal. The zeros come from the dissimilarity rule $d(x, x) = 0$.

With each step, we combine (or divide) clusters, forming a new matrix $M$. We have to rebuild $M$ in each step.

### 2.2.4 Measuring Distance Between Clusters

There are several ways to measure the distance between clusters.

- **Single-Link Technique**

With the single-link technique, distance between clusters $C_1$ and $C_2$ is given by the two closest objects, $x \in C_1$ and $y \in C_2$. We use $\min d(x, y)$ as the distance between clusters.

Figure 2.9 shows the single-link technique.

![Figure 2.9: Single-Link Technique](image_url)

The single-link technique tends to produce elongated clusters.
Complete-Link Technique

The complete technique uses the two furthest objects, \( x \in C_1, y \in C_2 \). The distance between clusters is \( \max d(x, y) \).

Figure 2.10 shows the complete-link technique.

![Figure 2.10: Complete-Link Technique](image)

Other Techniques

We can determine the distance between clusters using an average distance. Suppose \( x_i \in C_1 \) and \( y_i \in C_2 \). The distance is

\[
\frac{\sum d(x_i, y_i)}{|C_1| \cdot |C_2|}
\]

Centroids provide another measure of cluster distance. Let \( S \) be a set \( S \in \mathbb{R}^n \). The centroid of set \( B \) a vector

\[
c = \frac{\sum \{ x | x \in B \}}{|B|}
\]

The centroid is the block’s “center of gravity”. Typically, the centroid will not correspond to a real object.

Medoids provide yet another measure. A medoid is the object in a cluster \( C \) whose average distance to all other objects in \( C \) is minimal. The mediod always corresponds to an object.
2.3 Lecture – 3/3/2008

2.3.1 Hierarchical Clustering

Hierarchical clustering works in one of two ways:

- We start with small clusters and build larger and larger ones (Agglomerative clustering)
- We start with a large cluster and build smaller and smaller ones (Divisive clustering)

We will focus on Agglomerative clustering first. We start with singleton sets, and combine them to form larger clusters.

If our set of objects is \( S = \{x_1, \ldots, x_n\} \), we will start with \( \alpha_S = \{\{x_1\}, \ldots, \{x_n\}\} \).

Suppose we are on iteration \( t \) of the clustering process. We select two clusters, \( U \) and \( V \) to combine. In iteration \( t + 1 \), \( U \) and \( V \) will be replaced with a new cluster \( W \), where \( W = U \cup V \).

For this discussion, we’ll assume that our objects line in a metric space \( \mathbb{R}^n \).

2.3.2 A quick Aside

For reference, I’ve dug up a few formulas from one of my old Linear Algebra books.

\[
\| (x_1, \ldots, x_n) \| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \quad \text{length or absolute value of a vector}
\]

\[
p - q = \|p - q\| \quad \text{distance between two points in } n \text{ space}
\]

\[
p \cdot q = p_1 q_1 + p_2 q_2 + \cdots + p_n q_n \quad \text{dot product}
\]

\[
p \cdot q = \|p\| \cdot \|q\| \cos \theta \quad \text{another way to find dot product}
\]

\[
\|x\|^2 = x \cdot x \quad \text{for any vector } x
\]

\[
\|x + y\|^2 = \|x\|^2 + 2x \cdot y + \|y\|^2
\]

2.3.3 Now, Back to Clustering ...

We treat points as vectors in \( n \)-space. The euclidean distance between \( u \) and \( v \) is

\[
d(u, v) = \|u - v\|
\]

\[
d^2(u, v) = \|u - v\|^2
\]

\[
= (u - v, u - v)
\]

\[
u \cdot v = \|u\| \cdot \|v\| \cdot \cos \theta
\]

\[
u \cdot u = \|u\|^2
\]

Our basic goal is to find two clusters, \( U \) and \( V \), and combine them to form a new cluster \( W \). Figure 2.11 shows the general idea.

How do we choose \( U \) and \( V \)?
We define $c_U$ as the centroid of $U$. The centroids of $U$, $V$, $W$ are

$$
c_U = \frac{\sum \{x \mid x \in U\}}{|U|}
$$

$$
c_V = \frac{\sum \{x \mid x \in V\}}{|V|}
$$

$$
c_W = \frac{\sum \{x \mid x \in W\}}{|W|}
$$

When combining clusters, one of the measures that interests us is the sum of the squared errors of the cluster, or $sse(W)$. $sse$ tells us how “tight” the cluster is.

$$
sse(U) = \sum \{d^2(x, c_U) \mid x \in U\} \quad (2.18)
$$

$sse(U)$ is small for a tight cluster.

We can manipulate $c_W$ in the following way:

$$
c_W = \frac{\sum \{x \mid x \in W\}}{|W|} = \frac{\sum \{x \mid x \in U\} + \sum \{x \mid x \in V\}}{|U| + |V|} = \frac{|U|}{|W|} \sum \{x \mid x \in U\} + \frac{|V|}{|W|} \cdot \sum \{x \mid x \in V\}
$$

$$
= \frac{|U|}{|W|}c_U + \frac{|V|}{|W|}c_V
$$

When combining $U$ and $V$, we are interested in minimizing

$$
sse(W) - sse(U) - sse(V) \quad (2.19)
$$

$sse(W)$ is

$$
sse(W) = \sum \{d^2(x, c_W) \mid x \in W\}
$$

$$
d^2(u, v) = (u, u) - 2(u, v) + (v, v) = \|u\|^2 - 2(u \cdot v) + \|v\|^2
$$

Using the fact that $W = U \cup V$, we can re-write (2.19) as

$$
sse(W) - sse(U) - sse(V)
$$

$$
= \sum \{d^2(o, c_W) \mid o \in W\} - \sum \{d^2(o, c_U) \mid o \in U\} - \sum \{d^2(o, c_V) \mid o \in V\}
$$

Figure 2.11: Forming $W$ from $U$ and $V$
Looking only at the terms

\[ \sum \{ d^2(o, c_W) \mid o \in W \} - \sum \{ d^2(o, c_U) \mid o \in U \} \]

We can do the following

\[
\begin{align*}
\sum \{ d^2(o, c_W) \mid o \in W \} - \sum \{ d^2(o, c_U) \mid o \in U \} \\
= \sum \{ ||o||^2 \mid o \in U \} - 2 \sum_{o \in U} o \cdot c_W + |U| \cdot ||c_W||^2 - \left( \sum \{ ||o||^2 \mid o \in U \} - 2 \sum_{o \in U} o \cdot c_U + |U| \cdot ||c_U||^2 \right) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - \sum_{o \in U} (o \cdot c_W - c_U) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - 2|U|(c_U, c_W - c_U) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - 2|U|(c_U, c_W - ||c_U||^2)
\end{align*}
\]

We can do a similar thing for \( V \).

Putting the \( U \) and \( V \) parts (\( V \) not shown) together, we have

\[
\begin{align*}
\sum \{ ||o||^2 \mid o \in U \} - 2 \sum_{o \in U} o \cdot c_W + |U| \cdot ||c_W||^2 - \left( \sum \{ ||o||^2 \mid o \in U \} - 2 \sum_{o \in U} o \cdot c_U + |U| \cdot ||c_U||^2 \right) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - \sum_{o \in U} (o \cdot c_W - c_U) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - 2|U|(c_U, c_W - c_U) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - 2|U|(c_V, c_W)
\end{align*}
\]

Note that

\[
\begin{align*}
|U|(c_U, c_W) + |V|(c_V, c_W) \\
= (|U|c_U + |V| \cdot c_V, c_W) \\
= (|W|, c_W, c_W) \\
= |W| \cdot ||c_W||^2
\end{align*}
\]

Going back to (2.19), we have

\[
\begin{align*}
\sum \{ ||o||^2 \mid o \in U \} - 2 \sum_{o \in U} o \cdot c_W + |U| \cdot ||c_W||^2 - \left( \sum \{ ||o||^2 \mid o \in U \} - 2 \sum_{o \in U} o \cdot c_U + |U| \cdot ||c_U||^2 \right) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - \sum_{o \in U} (o \cdot c_W - c_U) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - 2|U|(c_U, c_W - c_U) \\
= |U| \cdot (||c_W||^2 - ||c_U||^2) - 2|U|(c_V, c_W)
\end{align*}
\]

After all of this derivation, the important equation to remember is

\[ sse(W) - sse(U) - sse(V) = \frac{|U| \cdot |V|}{|U| + |V|} (c_U - c_V)^2 \] (2.20)
2.3.4 Hierarchical Clustering - a high-level overview

We will look at several hierarchical clustering algorithms. These follow a common pattern:

**Input**: A set of objects \( S \), where \( |S| = n \), and a distance matrix \( D \).

**Output**: A hierarchy of clusters

**Method**:
- Define \( D^1 = D \)
- For \( k = 1 \) to \( n \), do
  - Choose clusters \( U, V \) according to a specific criteria
  - Fuse clusters \( U \) and \( V \) into \( W \)
  - Create the distance matrix \( D^{k+1} \) by computing the distances \( d(W, Q) \) for \( Q \neq U, Q \neq V \)

With each iteration, we’ll
- remove the matrix rows and columns for \( U \) and \( V \)
- add a row and column for \( W \)

In this process, our interesting choices are
- The criteria for choosing \( U \) and \( V \)
- The method for computing distances

We will study five methods for computing distances:

1. Single Link (\( sl \))
2. Complete Link (\( cl \))
3. Group Average Method (\( gav \))
4. Centroid Distance Method (\( cd \))
5. Ward method (\( ward \))

These distance measures are defined as follows:

\[
sl(u, v) = \min\{d(u, v) \mid u \in U, v \in V\} \tag{2.21}
\]
\[
cl(u, v) = \max\{d(u, v) \mid u \in U, v \in V\} \tag{2.22}
\]
\[
gav(u, v) = \frac{\sum\{d(u, v) \mid u \in U, v \in V\}}{|U| \cdot |V|} \tag{2.23}
\]
\[
cd(u, v) = (c_U - c_V)^2 \tag{2.24}
\]
\[
ward(u, v) = \frac{|U| \cdot |V|}{|U| + |V|} (c_U - c_V)^2 \tag{2.25}
\]
2.4 Lecture – 3/5/2008

2.4.1 Agglomerative Cluster Example with $sl$

Let’s work out an agglomerative clustering example. In this example, we will use the set of objects $S = \{o^1, \ldots, o^7\}$, where $o^i \in \mathbb{R}^2$

Figure 2.12 shows our set of objects.

For this example, we’ll use Manhattan distance. We will join clusters using the single link measure

$$d(u, v) = \min\{d(u, v) \mid u \in U, v \in V\}$$

First we find $D^1$, the distance matrix for $oS$.

$$
\begin{array}{cccccc}
D^1 & o^1 & o^2 & o^3 & o^4 & o^5 & o^6 \\
\hline
o^1 & 0 & 1 & 3 & 6 & 8 & 11 & 10 \\
o^2 & 1 & 0 & 2 & 5 & 7 & 10 & 9 \\
o^3 & 3 & 2 & 0 & 3 & 5 & 8 & 7 \\
o^4 & 6 & 5 & 3 & 0 & 2 & 5 & 4 \\
o^5 & 8 & 7 & 5 & 2 & 0 & 3 & 4 \\
o^6 & 11 & 10 & 8 & 5 & 3 & 0 & 3 \\
o^7 & 10 & 9 & 7 & 4 & 4 & 3 & 0 \\
\end{array}
$$

$o^1$ and $o^2$ are the closest clusters (smallest distance), so we’ll fuse them to form $o^{12}$. Next, we construct the next distance matrix $D^2$

$$
\begin{array}{cccccc}
D^2 & o^{12} & o^3 & o^4 & o^5 & o^6 & o^7 \\
\hline
o^{12} & 0 & 2 & 5 & 7 & 10 & 9 \\
o^3 & 2 & 0 & 3 & 5 & 8 & 7 \\
o^4 & 5 & 3 & 0 & 2 & 5 & 4 \\
o^5 & 7 & 5 & 2 & 0 & 3 & 4 \\
o^6 & 10 & 8 & 5 & 3 & 0 & 3 \\
o^7 & 9 & 7 & 4 & 4 & 3 & 0 \\
\end{array}
$$

For our next step, we have several options for picking clusters to merge. For example $o^{12}$ and $o^3$ have distance 2, but so do $o^4$ and $o^5$. We’ll merge $o^{12}$ with $o^3$. Next, we construct the distance matrix $D^3$
In our next step, we have one choice: merge $o^4$ and $o^5$. We merge these clusters and form $D^4$.

$$
\begin{array}{c|ccccc}
D^3 & o^{123} & o^4 & o^5 & o^6 & o^7 \\
\hline
o^{123} & 0 & 3 & 5 & 8 & 7 \\
o^4 & 3 & 0 & 2 & 5 & 4 \\
o^5 & 5 & 2 & 0 & 3 & 4 \\
o^6 & 8 & 5 & 3 & 0 & 3 \\
o^7 & 7 & 4 & 4 & 3 & 0 \\
\end{array}
$$

In our next step, we have choices for which clusters to merge (several have distance 3). We’ll merge $o^6$ and $o^7$, and construct the distance matrix $D^5$.

$$
\begin{array}{c|cc}
D^4 & o^{123} & o^{45} \\
\hline
o^{123} & 0 & 3 \\
o^{45} & 3 & 0 \\
o^6 & 8 & 3 \\
o^7 & 7 & 4 \\
\end{array}
$$

Next, we’ll join $o^{45}$ and $o^{67}$, and form the distance matrix $D^6$.

$$
\begin{array}{c|c}
D^5 & o^{123} & o^{4567} \\
\hline
o^{123} & 0 \\
o^{4567} & 3 \\
\end{array}
$$

Finally, we merge these to form the last cluster, yielding matrix $D^7$.

$$
\begin{array}{c|c}
D^6 & o^{1234567} \\
\hline
o^{1234567} & 0 \\
\end{array}
$$

A few things to note about this process:

- When we merge two clusters, we remove two rows and two columns, and add one row and one column. The rest of the matrix is unchanged.
- With each step, the minimum distance (smallest matrix cell value) is non-decreasing.

Why are distances non-decreasing? Let $c_i$ and $c_j$ be two clusters, which we fuse to form $c_r$. Using a single link distance:

$$c_{r_{ij}} = \min\{c_{it}, c_{jt}\}$$

The distances are non-decreasing, because you can’t get values which are smaller than the values you already have.

We can use this clustering to construct a dendrogram. The dendrogram for this cluster is shown in Figure 2.13.

Figure 2.13 shows how the clusters were formed. It’s worth noting the role that height plays. For example, the cluster between $o^1$ and $o^2$ appears at height 1 – that was the distance between $o^1$ and $o^2$ when they were merged. Similarly, $o^{12}$ is joined with $o^3$ at height 2, because that was the distance between $o^{12}$ and $o^3$ when they were merged.

Let’s define $\delta(u,v)$ as the height of the least common ancestor of $u$ and $v$. For example, $\delta(o^1,o^3) = 2$ and $\delta(o^1,o^6) = 3$. 

δ is an ultrametric. δ is also the largest ultrametric such that \( \delta(u, v) \leq d(u, v) \).

δ is called the sub-dominant ultrametric of \( d \).

### 2.4.2 Agglomerative Cluster Example with \( cl \)

Let’s repeat the example using the complete-link criteria for computing distance. Recall that complete link is

\[
cl(u, v) = \max\{d(u, v) \mid u \in U, v \in V\}
\]

Here is \( D^1 \) (it’s the same as \( D^1 \) in our last example).

\[
\begin{array}{c|cccccccc}
D^1 & o^1 & o^2 & o^3 & o^4 & o^5 & o^6 & o^7 \\
\hline
o^1 & 0 & 1 & 3 & 6 & 8 & 11 & 10 \\
o^2 & 1 & 0 & 2 & 5 & 7 & 10 & 9 \\
o^3 & 3 & 2 & 0 & 3 & 5 & 8 & 7 \\
o^4 & 6 & 5 & 3 & 0 & 2 & 5 & 4 \\
o^5 & 8 & 7 & 5 & 2 & 0 & 3 & 4 \\
o^6 & 11 & 10 & 8 & 5 & 3 & 0 & 3 \\
o^7 & 10 & 9 & 7 & 4 & 4 & 3 & 0 \\
\end{array}
\]

We use the same criteria for joining clusters: pick the pair with the smallest distance. We join \( o^1 \) and \( o^2 \) (distance = 1) and form \( D^2 \).

\[
\begin{array}{c|ccccc}
D^2 & o^{12} & o^3 & o^4 & o^5 & o^6 & o^7 \\
\hline
o^{12} & 0 & 3 & 6 & 8 & 11 & 10 \\
o^3 & 3 & 0 & 3 & 5 & 8 & 7 \\
o^4 & 6 & 3 & 0 & 2 & 5 & 4 \\
o^5 & 8 & 5 & 2 & 0 & 3 & 4 \\
o^6 & 11 & 8 & 5 & 3 & 0 & 3 \\
o^7 & 10 & 7 & 4 & 4 & 3 & 0 \\
\end{array}
\]

Next, we join \( o^4 \) and and \( o^5 \) (distance 2) and form \( D^3 \).

\[
\begin{array}{c|ccccc}
D^3 & o^{12} & o^3 & o^{45} & o^6 & o^7 \\
\hline
o^{12} & 0 & 3 & 8 & 11 & 10 \\
o^3 & 3 & 0 & 5 & 8 & 7 \\
o^{45} & 8 & 5 & 0 & 5 & 4 \\
o^6 & 11 & 8 & 5 & 0 & 3 \\
o^7 & 10 & 7 & 4 & 3 & 0 \\
\end{array}
\]

Next, we join \( o^6 \) and \( o^7 \) (distance = 3) and form \( D^4 \).
Next, we join \(o^{12}\) and \(o^{3}\) (distance = 3), and form \(D^5\).

\[
\begin{array}{c|c|c|c}
D^5 & o^{123} & o^{45} & o^{67} \\
\hline
o^{12} & 0 & 8 & 11 \\
o^{3} & 3 & 0 & 5 \\
o^{45} & 8 & 0 & 5 \\
o^{67} & 11 & 5 & 0 \\
\end{array}
\]

Next, we join \(o^{45}\) and \(o^{67}\) (distance = 5), and form \(D^6\).

\[
\begin{array}{c|c|c}
D^6 & o^{123} & o^{4567} \\
\hline
o^{123} & 0 & 11 \\
o^{4567} & 11 & 0 \\
\end{array}
\]

And finally, we join the last pair of clusters (distance 11) to form \(D^7\).

\[
\begin{array}{c|c}
D^7 & o^{1234567} \\
\hline
o^{1234567} & 0 \\
\end{array}
\]

The dendrogram for this clustering is shown in Figure 2.14.

![Dendrogram](image)

**Figure 2.14: Dendrogram for cl cluster**

Once again, we see a property of **monotonicity** of cluster distances. If we have \(c_i\) and \(c_j\) of \(D^k\), and we merge \(c_i, c_j\) into \(c_r\) in \(D^{k+1}\), then

\[c_{rl} = \max\{c_{il}, c_{jl}\}\]

\(c_{il}\) and \(c_{jl}\) appears in \(D^k\), and \(c_{rl}\) appears in \(D^{k+1}\). These are arbitrary points, so no distance in \(D^{k+1}\) can be less than a distance in \(D^k\).

One interesting aspect of dendrograms in the way they give us flexibility for choosing the granularity of clusters.
Consider the dendrogram in Figure 2.15.
The horizontal lines (a), (b), and (c) are “between” unit heights. If we “cut” the dendrogram at (a), we have seven clusters. If we cut the dendrogram at (b) we have 6 clusters. If we cut the dendrogram at (c), we have four clusters.

2.4.3 k-Means Clustering Algorithm

k-Means is another popular clustering algorithm. Unlike agglomerative clustering, k-Means requires the number of clusters as input to the algorithm.

Input  A set of points $S$, a distance function $d$, and a number of clusters $k$ (where $k \leq |S|$).

Method
- Randomly choose $k$ points $c_1, \ldots, c_k$ in $\mathbb{R}^n$. These are the initial “centroids” of the clusters. There is no connection between $c_1, \ldots, c_k$ and the set of objects in $S$.
- Repeat until no change
  - Assign each of the objects $o_1, \ldots, o_n$ to one of the “centroids”. This forms an initial clustering.
  - Recompute the centroids, based on distances from the real objects.

We can construct an assignment matrix. The rows are objects $o_1, \ldots, o_n$, and the columns are clusters $c_1, \ldots, c_k$. A cell of the matrix, $b_{ij}$ is

$$
b_{ij} = \begin{cases} 
1 & \text{if } o_i \text{ is assigned to } c_j \\
0 & \text{otherwise}
\end{cases}
$$

Note: when choosing a distance function for k-Means, you must choose a real distance measure. k-Means will not work if $d$ is not a real distance.

2.4.4 Misc.

Try drawing four points on a sheet of paper, such that the distance between each pair of points is one. It’s not possible to do. $\therefore$ it’s hard to represent multi-dimensional data on flat paper.
2.5 Lecture – 3/10/2008

2.5.1 The k-Means Clustering Algorithm

One of the inputs to k-means is $k$, the number of clusters. In some cases, you’ll know $k$ in advance. In other cases, you may arrive at $k$ by trial and error – try several $k$ and take the value which produces the best results.

Assume we have objects $\{o^1, \ldots, o^n\}$, and centroids $\{c^1, \ldots, c^k\}$. Both $o$ and $c$ are vectors in $\mathbb{R}^m$.

Initially $\{c^1, \ldots, c^k\}$ are chosen at random (they don’t correspond to real objects).

We use a matrix $B$ to keep track of assignments of objects to clusters. In $B$, we have one row per object, and one column cluster. Matrix cells have the value

$$b_{ij} = \begin{cases} 
1 & \text{if } o^i \text{ is assigned to cluster } U^k, \text{ having centroid } c^k \\
0 & \text{otherwise}
\end{cases}$$

Once we make an initial assignment of objects to clusters, we

- Recompute the centroids, based on assignment of objects to clusters
- Re-assign objects to the closest centroid

In pseudocode,

```
while (stopping criteria is not met) {
    assign object to the cluster with the closest centroid
    recompute the centroids
}
```

We can measure the “tightness” of a cluster using the sum of the squared error

$$\sum_{o \in U^j} \|o - c^j\|^2$$

The more central $c^j$ is, the smaller the squared error will be.

We can also measure the squared error for all clusters

$$F(c^j) = \sum_{j=1}^{k} \sum_{o \in U^j} \|o - c^j\|^2 \quad (2.26)$$

We can rewrite (2.26) to use $B$ instead of set membership.

$$F(c^j) = \sum_{j=1}^{k} \sum_{i=1}^{n} b_{ij} \|o^i - c^j\|^2 \quad (2.27)$$

Our goal is to choose centroids that minimize $F(c^j)$. Our optimality criteria is

$$\frac{\partial F(c^j)}{\partial c^j} = 0$$

\footnote{The superscripts denote different set members, not exponentiation.}
Let \( o^i = (o^i_1, \ldots, o^i_m) \) and \( c^j = (c^j_1, \ldots, c^j_m) \). \( o^i \) and \( c^j \) are both vectors in \( m \)-space. The squared error between \( o \) and \( c \) is

\[
\| o^i - c^j \|^2 = \sum_{l=1}^{m} (o^i_l - c^j_l)^2
\]

We can rewrite \( F(c^j_l) \) further

\[
F(c^j_l) = \sum_{j=1}^{k} \sum_{i=1}^{n} b_{ij} \sum_{l=1}^{m} (o^i_l - c^j_l)^2
\]

\[
= \sum_{j=1}^{k} \sum_{i=1}^{n} \sum_{l=1}^{m} b_{ij} (o^i_l - c^j_l)^2
\]

To minimize, we fix \( j \) and \( l \) (\( i \) is variable).

\[
\frac{\partial F(c^j_l)}{\partial c^j_l} = -n \sum_{i=1}^{n} 2b_{ij} (o^i_l - c^j_l) = 0
\]

\[
\sum_{i=1}^{n} b_{ij} \cdot o^i_l = \sum_{i=1}^{n} b_{ij} \cdot c^j_l
\]

\[
\sum_{i=1}^{n} b_{ij} \cdot o^i_l = c^j_l \sum_{i=1}^{n} b_{ij}
\]

The final equality is valid for \( 1 \leq l \leq m \).

\( c^j \sum_{i=1}^{n} b_{ij} = \sum_{i=1}^{n} b_{ij} \cdot o^i \) gives us equality on components. To find the new centroid, we use

\[
c^j = \frac{\sum_{i=1}^{n} b_{ij} \cdot o^i}{\sum_{i=1}^{n} b_{ij}}
\]

(2.28)

By (2.28) the new centroid is just the average (center of gravity) of the cluster.

Once we have found new centroids, then we reassign objects to clusters.

Common stopping criteria for \( k \)-means:

- We stop after a specific number of iterations
- We stop when \( F(c^j_l) \) falls below a certain threshold

The time complexity of \( k \)-means is \( \Theta(nkt) \) – where \( n \) is the number of objects, \( k \) is the number of clusters, and \( t \) is the number of iterations.

Note again that \( k \)-means groups around centroids, and centroids are not real objects.

\( k \)-means yields good clusters (they’re locally optimal), but not necessary the best (globally optimal) clusters.

### 2.5.2 Partitions Around Medoids (PAM)

The PAM (partitions around medoids) algorithm is superficially like \( k \)-means. The most obvious difference is that \( k \)-means uses centroids (not real objects) as cluster centers, while PAM uses medoids (which are real objects). Like \( k \)-means, PAM requires \( k \) to be given in advance.

PAM works in two phases:
**Building Phase** During the building phase, we’ll try to find the most central objects. We aim to choose \( k \) objects that are most centrally located.

If \( S \) is the set of objects, and \( L \) is the set of objects chosen by the building phase, then \( R = S - L \) is the set of unchosen objects.

**Swapping Phase** During the swapping phase, we swap selected \( (L) \) and unselected \( (R) \) objects. We aim to improve the centrality of medoids.

The swapping phase continues as long as we see improvement.

Figure 2.16 shows the set \( S \), the selected set \( L \), and one object \( x \in R \).

![Figure 2.16: Set S with selected objects L and x ∈ R](image)

For a discrete set, the distance between \( L \) and \( x \) can be given by

\[
d(L, x) = \min_{t \in L} d(t, x)
\]

\[
d(L, x) = 0 \quad \text{if} \ x \in L
\]

How do we choose the first member of \( L \)? We can find the object \( o^i \) with the minimum distance to all other objects. If we have a distance matrix \( D \), then \( o^i \) is the object whose row has the smallest sum. In other words \( o^i \) is the object that minimizes \( \sum_{j=1}^{n} d_{ij} \). \( (d_{ij} \) is a cell of \( D) \).

Now, we have \( L = \{o^i\} \). Next, we pick \( o \in R \) such that \( \sum_{o' \in R-\{o\}} d(o, o') \) is minimal. This is basically what we did with \( o^i \), but we disregard \( o^i \) when choosing \( o \).

During the swapping phase, we need to decide whether to swap \( o_i \) and \( o_h \). We’ll need to consider every other object \( o_j \) and consider the cost of the swap.

Let \( c_{ihj} \) be the cost to swap \( o^i \) and \( o^h \) with respect to \( o^j \). The total cost of the swap is \( \sum_{j=1}^{n} c_{ihj} \). We will only make swaps whose cost is negative.

Note that \( d(o^j, L) \leq d(o^j, o^i) \). We have the following cases to consider:

- \( d(L, o^j) < d(o^h, o^j) < d(o^i, o^j) \). In this case \( o^j \) is closer to something else (not \( L \)). So \( c_{ihj} = 0 \).
- \( d(o^h, o^j) < d(L, o^j) < d(o^i, o^j) \). \( o^j \) is closer to \( o^i \) than \( L \). \( c_{ihj} = d(o^h, o^j) - d(L, o^j) \).
- \( d(L, o^j) = d(o^i, o^j) \). Let \( e(o^j) \) between the distance between \( o^j \) and the second closest object in \( L \).

\[
c_{ihj} = \min\{d(o^h, o^j), e(o^j)\} - d(L, o^j).
\]

2.5.3 Misc.

What is an infimum?

Consider the interval \([0, 1]\) this set has a least element, and that least element is also the infimum.

Now consider the interval \([0, 1]\). This set has no least element. \( \inf([0, 1]) = 0 \). If we think of the interval as a continuous set, \( 0 \notin (0, 1) \).
2.6 PAM Notes – 3/11/2008

2.6.1 PAM notes, from Prof. Simovici’s handout

PAM is a fairly robust algorithm. Where k-means minimizes the sum of the squared errors, PAM tries to minimize the sum of the errors.

For PAM, we start with

- A set $S$ of objects ($|S| = n$).
- A parameter $k$, which specifies the number of clusters
- An $n \times n$ distance matrix $D$.

Medoids are supposed to assume the most central positions in their respective clusters.

During the building phase, we choose an initial set of $k$ medoids. Let $L$ be the set of medoids, and $R = S – L$ be the set of non-medoids.

1. The first medoid is the one with a minimal $\sum_{i=1}^{n} d_{ij}$. Once we find this medoid, we add it to $L$ ($L$ now has one element).

2. We select the rest of the medoids in $L$ by using a merit function $M(o)$.

$$M(o) = \sum_{o' \in R - \{o\}} \max\{d(L, o') - d(o, o'), 0\}$$  \hspace{1cm} (2.29)

- How does this work. Suppose $d(o, o') < d(L, o')$. Adding $o$ to $L$ (the medoid set) benefits clustering from the point of view of $o'$, because $d(L, o')$ will diminish.
- On the other hand, if $d(o, o') \geq d(L, o')$, then there is no benefit from the point of view of $o'$.

The tricky thing here: for each object $o \notin L$, we’re examining the distance between $o$ and every other $o'$ in $R - \{o\}$.

The building phase halts when $|L| = k$.

During the swapping phase, we try to improve the clustering by exchanging objects in $L$ with objects in $R = S – L$.

For the swapping phase:

- $o_i$ is an object in $L$
- $o_h$ is an object not in $L$
- $o_j$ is an arbitrary object not in $L$

We consider the cost $C(o_i, o_h)$ of swapping $o_i$ and $o_h$. The contribution $c_{ihj}$ of $o_j$ to the cost of swapping $o_i$ and $o_h$ is as follows:

- If $d(o_i, o_j) > d(o, o_j)$ and $d(o_h, o_j) > d(o, o_j)$ for any $o \in L - \{o_i\}$, then the cost is zero.
- If $d(o_i, o_j) = d(L, o_j)$, then we define $e(o_j)$ to be object in $S$ that is second-closest to $o_j$.\footnote{Is this really $S$?} The cost contribution is

$$c_{ihj} = \min\{d(o_h, o_j), e(o_j)\} - d(o_i, o_j)$$

- If (a) $d(o_i, o_j) > d(L, o_j)$ (there is some object in $L$ that is closer to $o_j$ than $o_i$), and (b) $d(o_h, o_j) < d(L, o_j)$ ($o_j$ is closer to $o_h$ than any medoid object in $L$), then the cost is given by\footnote{Why $S$ again. Isn’t $d(S, o_j) = 0$?}

$$c_{ihj} = d(o_h, o_j) - d(S, o_j)$$
The overall cost of the swap is

\[ C(o_i, o_h) = \sum_{o_j \in R} c_{ihj} \]

The pair that minimizes \( C(o_i, o_h) \) is selected. If \( C(o_i, o_h) < 0 \), then the swap is carried out. All potential swaps are considered.

In pseudocode:

```
construct the set \( L \) of \( k \) medoids
    do
        compute the cost \( C(o_i, o_h) \) for all \( o_i \in L \) and \( o_h \in R \).
        find the pair \( (o_i, o_h) \) that produces the minimum \( m = C(o_i, o_h) \)
        swap \( (o_i, o_h) \)
    until \( (m > 0) \)
```

PAM’s runtime complexity is \( \Theta(n^4) \), which makes it impractical for large data sets.

### 2.6.2 PAM Notes – from Han

Han’s swapping criteria appear below. In his presentation, \( o_j \) is a representative node (\( o_j \in L \)), \( p \) is non-representative object being examined, and \( o_{random} \) is a random non-representative object.

**Case 1** \( p \) belongs to \( o_j \). If \( o_j \) is replaced by \( o_{random} \) as a representative object, and \( p \) is closer to some other representative object \( o_i \), then reassign \( p \) to \( o_i \).

**Case 2** \( p \) belongs to \( o_j \). If \( o_j \) is replaced by \( o_{random} \) as a representative object and \( p \) is closest to \( o_{random} \), then reassign \( p \) to \( o_{random} \).

**Case 3** \( p \) belongs to \( o_j \). If \( o_j \) is replaced by \( o_{random} \) as a representative object, and \( p \) is still closer to \( o_j \), then no change of assignment takes place.

**Case 4** \( p \) belongs to \( o_i \). If \( o_j \) is replaced by \( o_{random} \) as a representative object, and \( p \) is closest to \( o_{random} \), then \( p \) is assigned to \( o_{random} \).
2.7 Lecture – 3/12/2008

2.7.1 Graph Clustering

We can use a complete graph \( G = (V,E) \) to represent distances between objects. Each edge \( (u,v) \) of \( G \) is labeled with the distance between \( u \) and \( v: k = d(u,v) \).

Since \( G \) is a complete graph, there is an edge between every \( (u,v) \) pair.

Let \( S = \{o_1, \ldots, o_n\} \) be our set of objects. We’ll use the same distance matrix that we’ve used for some prior examples:

<table>
<thead>
<tr>
<th></th>
<th>( o_1 )</th>
<th>( o_2 )</th>
<th>( o_3 )</th>
<th>( o_4 )</th>
<th>( o_5 )</th>
<th>( o_6 )</th>
<th>( o_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( o_1 )</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>( o_2 )</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>( o_3 )</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>( o_4 )</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>( o_5 )</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( o_6 )</td>
<td>11</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>( o_7 )</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.1: Distance matrix for Graph Clustering Examples

We say that \( u \sim v \) if there is a path from \( u \) to \( v \), including the path of length zero. \( \sim \) is an equivalence relationship on vertices:

\[
\begin{align*}
&u \sim u \\
&u \sim v \rightarrow v \sim u \\
&u \sim v, v \sim w \rightarrow u \sim w
\end{align*}
\]

This equivalence class is called the connected component of \( u \).

A few more definitions:

**Definition 2.7.1.1 (Clique):** \( K \) is a clique if, for every \( u, v \in K \) we have (1) \( (u, v) \in E \), and (2) \( K \) is maximal.

**Definition 2.7.1.2 (Threshold Graph):** A threshold graph \( G_k \) is a graph consisting of

\[
G_k = (V, \{(u, v) \mid d(u,v) \leq k\})
\]

\( G_k \) contains all vertices, but only those edges that are \( \leq k \). Of course, as \( k \) increases, we include more edges.

We can use threshold graphs for clustering. We’ll describe two approaches: single-link and complete-link.

2.7.2 Single-Link Graph Clustering

- For our first iteration, we form \( G_0 \). This graph has no edges, just seven nodes (see Table 2.1).
  
  This gives seven clusters:

\[
G_0: \{o_1\}, \{o_2\}, \{o_3\}, \{o_4\}, \{o_5\}, \{o_6\}, \{o_7\}
\]

\( G_0 \) is shown in Figure 2.17.
• For $G_1$, we add one edge: $(o_1, o_2)$. This gives us a strongly connected component, and a new cluster.

$G_1: \{o_1, o_2\}, \{o_3\}, \{o_4\}, \{o_5\}, \{o_6\}, \{o_7\}$

$G_1$ is shown in Figure 2.18.

• Next, we form $G_2$, using all edges of length $\leq 2$. This adds $(o_2, o_3)$ and $(o_4, o_5)$. The new set of clusters is

$G_2: \{o_1, o_2, o_3\}, \{o_4, o_5\}, \{o_6\}, \{o_7\}$

$G_2$ is shown in Figure 2.19.

• Next, we form $G_3$, using all edges of length $\leq 3$. This adds $(o_1, o_3)$, $(o_3, o_4)$, $(o_5, o_6)$, and $(o_6, o_7)$. $G_3$ has one strongly connected component, so the process terminates:

$G_3: \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$

$G_3$ is shown in Figure 2.20.
Single-link graph clustering continues as long as the number of connected components decreases.

### 2.7.3 Complete-Link Graph Clustering

With single-link graph clustering, we formed clusters from connected components. In complete link graph clustering, we form clusters from cliques.

The general idea: if \( P, Q \) are disjoint cliques in \( G_k \), and \( P \cup Q \) is a clique in \( G_{k+1} \), then \( P, Q \) are fused together.

Because \( G \) is a complete graph, we are guaranteed to wind up with a single link. When we form \( G_m \) for \( m = \max d_{ij} \) we have all edges of the complete graph.

- \( G_0 \) is the same as the single link case: each vertex is in a cluster by itself.
  \[ G_0: \{o_1\}, \{o_2\}, \{o_3\}, \{o_4\}, \{o_5\}, \{o_6\}, \{o_7\} \]

- \( G_1 \) gives a single clique (Figure 2.21).

  \[ G_1: \{o_1, o_2\}, \{o_3\}, \{o_4\}, \{o_5\}, \{o_6\}, \{o_7\} \]

- \( G_2 \) has two cliques (Figure 2.22)
  \[ G_2: \{o_1, o_2\}, \{o_3\}, \{o_4, o_5\}, \{o_6\}, \{o_7\} \]

- Next, we form \( G_3 \) (Figure 2.23).
  \[ G_3: \{o_1, o_2, o_3\}, \{o_4, o_5\}, \{o_6, o_7\} \]

  Figure 2.21: \( G_1 \) for complete-link
• $G_4$ adds edges $(o_4, o_7)$ and $(o_5, o_7)$ (Figure 2.24). Note that we cannot join \{o_4, o_5\}, \{o_6, o_7\}, since there is no $(o_4, o_6)$ edge.

$G_4 = G_3$: \{o_1, o_2, o_3\}, \{o_4, o_5\}, \{o_6, o_7\}

• $G_5$ is next. We add edges $(o_2, o_4)$, $(o_3, o_5)$, and $(o_4, o_6)$ (Figure 2.25). $G_5$ gives us a new cluster from the clique \{o_4, o_5, o_6, o_7\}.

$G_5$: \{o_1, o_2, o_3\}, \{o_4, o_5, o_6, o_7\}

• $G_6$ comes next (Figure 2.26). We add the edge $(o_1, o_4)$, but there’s no change in the clusters. In order to fuse \{o_1, o_2, o_3\} with \{o_4, o_5, o_6, o_7\}, we’ll need a clique containing all seven nodes.

• $G_7$ comes next (Figure 2.27). We add the edges $(o_2, o_5)$ and $(o_3, o_7)$.

• $G_8$ is next (Figure 2.28). We add edges $(o_1, o_5)$ and $(o_3, o_6)$. 

Figure 2.22: $G_2$ for complete-link

Figure 2.23: $G_3$ for complete-link

Figure 2.24: $G_4$ for complete-link

Figure 2.25: $G_5$ for complete-link

Figure 2.26: $G_6$ for complete-link

Figure 2.27: $G_7$ for complete-link

Figure 2.28: $G_8$ for complete-link
• $G_9$ is next (Figure 2.29). We add edge $(o_2, o_7)$.

• $G_{10}$ is next (Figure 2.30). We add edge $(o_2, o_6)$ and $(o_1, o_7)$. 
Finally, we form $G_{11}$, giving us a single clique (Figure 2.31).

$$G_{11} = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$$

With complete-link graph clustering, clusters tend to appear later in the process (we need more edges to form the cliques).

### 2.7.4 Density-Based Clustering Algorithms

The clustering algorithms we’ve studied so far tend to produce circular (or ellipsoidal) clusters. Density-based clustering can discover irregularly shaped clusters.

One of the most prominent density-based algorithms is DBSCAN. The algorithm was introduced in KDD96, in a paper by M. Ester.

DBSCAN uses two parameters, $\epsilon$ and $\mu$. 
Let \( q \) be a point. The *neighborhood* of \( q \) is the set of objects within distance \( \epsilon \) of \( q \). We denote this with \( N_\epsilon(q) \). The neighborhood is a circle, with \( q \) at the center, and \( \epsilon \) as the radius. This is illustrated in Figure 2.32.

\[
N_\epsilon(q)
\]

Figure 2.32: \( N_\epsilon(q) \)

Density-based clustering recognizes three types of objects:

**Core Objects** \( o \) is a core object if the number of points in \( N_\epsilon(o) \geq \mu \).

**Border Objects** \( p \) is a border object if \( p \in N_\epsilon(o) \) for some object \( o \). \( p \) itself is *not* a core object.

**Noisy Objects** Noisy objects are those which are neither core nor border objects. In general, we don’t include noisy objects in clusters.

**Definition 2.7.4.1 (Directly Reachable):** \( p \) is *directly reachable* from \( q \) if \( p \) is a core object, and \( p \in N_\epsilon(q) \).

Note that \( p \) is directly reachable from \( q \), \( q \) may not be directly reachable from \( p \). Directly reachable is a reflexive relation, but not a symmetric one.

**Definition 2.7.4.2 (Density Reachable):** \( p \) is *density reachable* from \( q \) if there exists a sequence of objects \( r_1, \ldots, r_n \) such that \( r_1 = q \), \( r_{i+1} \) is directly reachable from \( r_i \), and \( r_n = p \).

This is a sequence of core objects, with the possible exception of the last one. Density reachable is not symmetric, but it is transitive.

**Definition 2.7.4.3 (Density Connected):** Two objects \( u, v \) are *density connected* if there is an objects \( o \), such that \( u, v \) are density reachable from \( o \).

A set \( C \) of core objects is a cluster if

1. For all \( u, v \), if \( u \in C \) and \( v \) is density reachable from \( u \), then \( v \in C \).
2. For all \( u, v \in C \), \( u \) and \( v \) are density connected

The general idea behind density-based clustering is the following: we find a core object. We then build a cluster from the core object, extending the cluster as long as there are density reachable objects. When there are no more density reachable points, we stop.

This process makes the selection of \( \epsilon \) very important. \( \epsilon \) must be smaller than the smallest distance between clusters (otherwise, we’ll wind up combining the clusters).

For more detailed clusters, smaller \( \epsilon \) are better.

### 2.7.5 To-Do

Find (and read) a paper on DBSCAN. (We’ll need this for hw2 problem 3).
2.8 Notes on Density-Based Clustering – 3/23/2008

Notes from

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96).

2.8.1 Density-clustering concepts

Within each cluster, the point density is higher than for areas outside the cluster. Areas of low density are called noise; noisy areas are not part of any clusters.

Basic idea - for each point in a cluster, the neighborhood for a given radius needs to contain at least a minimum number of points.

The shape of a neighborhood is determined by a distance function \(d(p, q)\). For example, euclidean distance creates spheres and Manhattan distance creates rectangles.

**Definition 2.8.1.1 (\(\epsilon\)-neighborhood):** The \(\epsilon\)-neighborhood of a point \(p\), denoted \(N_\epsilon(p)\) is the set of points

\[
N_\epsilon(p) = \{q \in D \mid d(p, q) \leq \epsilon\}
\]

(2.30)

So, all points \(q\) within radius \(\epsilon\) of \(p\).

Within a cluster, there are two kinds of points: core points and border points.

- **core point** A core point is a point \(p\) within a cluster such that \(|N_\epsilon(p)| \geq \text{minpts}\).
- **border point** A border point is a point \(p\) within a cluster such that \(|N_\epsilon(p)| < \text{minpts}\).

Intuitively, border points sit on the edge of clusters, and core points sit somewhere in the middle.

**Definition 2.8.1.2 (directly density-reachable):** A point \(p\) is directly density-reachable from point \(q\) with respect to \(\epsilon\) and \(\text{minpts}\) if

1. \(p \in N_\epsilon(q)\), and
2. \(|N_\epsilon(q)| \geq \text{minpts}\) (core point condition)

Directly density reachable is symmetric for core points, but asymmetric for core points and border points.

**Definition 2.8.1.3 (Density-reachable):** A point \(p\) is density reachable from a point \(q\) with respect to \(\epsilon\) and \(\text{minpts}\) if there is a chain of points \(p_1, \ldots, p_n\), where

1. \(p_1 = q\),
2. \(p_n = p\), and
3. \(p_i\) is directly density-reachable from \(p_{i+1}\).

Density reachable is transitive. Density reachable is not symmetric in general, but it is symmetric for core points.

Two border points of the same cluster may not be density reachable from each other, because the core point condition may not hold for both of them. However, there must be a core point of the cluster such that both border points are density reachable from the core point.

**Definition 2.8.1.4 (Density-Connected):** A point \(p\) is density connected to a point \(q\), with respect to \(\epsilon\), \(\text{minpts}\) if there is a point \(o\), such that

1. \(p\) is density-reachable from \(o\) with respect to \(\epsilon\) and \(\text{minpts}\), and
2. $q$ is density-reachable from $o$ with respect to $\epsilon$ and $\text{minpts}$.

**Definition 2.8.1.5 (Cluster):** A cluster $C$ is a non-empty subset of points that meets the following conditions

1. For all points $p, q$; if $p \in C$ and $q$ is density-reachable from $p$, then $q \in C$. (Maximality property)
2. For all $p, q \in C$; $p$ is density connected to $q$. (Connectivity)

Note that each cluster must have at least $\text{minpts}$ points, because the second condition requires the existence of a core point.

**Lemma 2.8.1.6:** Let $p$ be a point in the database $D$, where $|N_\epsilon(p)| \geq \text{minpts}$ (a core point). Let $O$ be a set of points such that for each $o \in O, o \in D$ and $o$ is density-reachable from $p$. $O$ is a cluster with respect to $\epsilon$ and $\text{minpts}$.

**Lemma 2.8.1.7:** Let $C$ be a cluster, and let $p$ be any point in $C$ with $|N_\epsilon(p)| \geq \text{minpts}$ (again, $p$ is core point). Then, $C$ equals the set $O$, where $\forall o \in O, o$ is density-reachable from $p$.

### 2.8.2 DBSCAN Algorithm

The basic idea of the DBSCAN algorithm:

- we pick a point $p$, and find all points that are density-reachable from $p$. If $p$ is a core point, the we’ve found a cluster.
- if $p$ is a border point, then no points are density reachable from $p$, and DBSCAN visits the next point in the database.

The DBSCAN algorithm consists of two main procedures.

```plaintext
procedure DBSCAN(SetofPoints, Eps, minpts) {
    ClusterId = nextId(NOISE);
    for i in (1 to SetOfPoints.size()) {
        Point = SetOfPoints.get(i)
        if (Point.ClusterId == UNCLASSIFIED) {
            if (expandCluster(SetOfPoints, Point, ClusterId, Eps, minpts)) {
                ClusterId = nextId(ClusterId);
            }
        }
    }
}
```
procedure ExpandCluster(SetOfPoints, Point, ClusterId, Eps, minpts) {
    seeds = SetOfPoints.regionQuery(Point, Eps)
    if (seeds.size < minpts) {
        /* not a core point */
        SetOfPoints.changeClusterId(point, NOISE)
        return FALSE
    }
    /* Point is a core point */
    SetOfPoints.changeClusterId(seeds, ClusterId)
    seeds.delete(Point)
    while (seeds is not EMPTY) {
        currentP = seeds.first();
        result = SetOfPoints.regionQuery(currentP, Eps)
        if (result.size() >= minpts) {
            for (i = 1 to result.size) {
                resultP = result.get(i)
                if (resultP.clusterId in (UNCLASSIFIED, NOISE)) {
                    seeds.append(resultP)
                }
            }
            SetOfPoints.changeClusterId(resultP, ClusterId)
        }
        seeds.delete(currentP)
    }
    return TRUE
}

The call SetofPoints.regionQuery(Point, Eps) returns the set of points in \( N_{\epsilon}(p) \).

If a point \( p \) is assigned a clusterId of NOISE, we allow \( p \)'s clusterId to be changed later, if \( p \) is density-reachable from some other point in the database.

### 2.8.3 Determining \( \epsilon \) and \( \text{minpts} \)

- We define a function \( k\text{-dist}(p) \). Given a point \( p \), \( k\text{-dist} \) returns the distance between \( p \) and \( p \)'s \( k \)-th nearest neighbor.
- We sort points in descending order of their \( k\text{-dist} \) values.
- Say we choose an arbitrary point \( p \), and set \( \epsilon = k\text{-dist}(p) \) and \( \text{minpts} = k \). Then all points \( q \) with \( k\text{-dist}(q) \leq k\text{-dist}(p) \) will be core points.
- If we can find a threshold point with the maximal \( k\text{-dist} \) value in the thinnest cluster, then we have the desired values for \( \epsilon \) and \( \text{minpts} \).

For 2-dimensional data, the authors recommend the following:

- Use 4 as the value for \( \text{minpts} \).
- Compute and display the 4-dist graph of points in the database.
- The user can select a threshold point from the graph (which defines \( \text{minpts} \) and \( \epsilon \)); or, the user can specify what percentage of the points are noise, and the system can find a threshold point based on the noise percentage.
2.9 Lecture – 3/25/2008

2.9.1 DBSCAN and Density-Based Clustering

DBSCAN starts with a set of points and a distance metric \( d \). DBSCAN recognizes three types of points
1. Core points
2. Border points
3. Noise

Clusters consist of core points and border points. Noise is not included in any cluster.

DBSCAN uses two parameters:
1. \( \epsilon \) - the radius of a point’s neighborhood
2. \( \mu \) - this is \( \text{minpts} \) in Ester’s paper

\( N_\epsilon(p) \) is a closed sphere of radius \( \epsilon \) centered around point \( p \). Informally, \( p \) belongs to a cluster if \( N_\epsilon(p) \) contains a minimum number of points.

The core point condition is \( N_\epsilon(p) \geq \mu \).

The choice of \( \mu, \epsilon \) is largely heuristic.

A cluster \( C \) can have several core points, but not all \( p \in C \) need be core points.

If \( q \) is a point in a cluster and \( N_\epsilon(q) < \mu \), then \( q \) is a border point.

A point \( q \) is directly reachable from \( p \) if \( q \in N_\epsilon(p) \).

A point \( q \) is reachable from \( p \) if there a a sequence of points \( r_1, \ldots, r_n \) such that (1) \( r_1 = p \), (2) \( r_n = q \), and \( r_{i+1} \) is directly reachable from \( r_i \) for \( 1 \leq i \leq n - 1 \). Note that points \( r_1, \ldots, r_{n-1} \) must be core points; \( r_n \) may be a core point or a border point.

Figure 2.33 illustrates the difference between directly-reachable and reachable points.

Two points, \( p \) and \( p' \) are density connected if there is a point \( q \) such that \( p \) is reachable from \( q \) and \( p' \) is reachable from \( q \). Figure 2.34 shows an example of density connected points.

A cluster is a set of points \( C \) that satisfies two conditions:
1. If \( u \in C \) and \( v \) is reachable from \( u \), then \( v \in C \).
2. If \( u, v \in C \), then \( u, v \) are density connected.
The distance between two clusters $C, C'$ is
\[ d(C, C') = \min\{d(x, y) \mid x \in C, y \in C'\} \quad (2.31) \]

**Choosing $\mu$ and $\epsilon$**

Let $\delta(p)$ be the distance from $p$ to $p$’s $k$-th nearest neighbor. $\delta(p)$ is the radius of a sphere that includes $k$ points.

Next, we arrange the points by decreasing of $\delta(p)$. If we graph $\delta(p)$ vs points, we can look for an “elbow” in the curve. The elbow gives us $\epsilon$. Points where $\delta(p) < \epsilon$ are core points. The rest are noise (and maybe some border points). Figure 2.35 illustrates the basic idea.

![Figure 2.35: Illustration of selecting $\epsilon$](image)

The choice of $k$ becomes $\mu$. In practice, $k = \mu = 4$ tends to work well.
Part 3

Classification

3.1 Lecture – 3/24/2008

3.1.1 An overview of Classification

Classification is inherently based on past experience.

A model is an algorithm that does classification.

A training set is a set of existing data that is used to train the model.

The process of constructing a model is called “learning the classifier” (via the training set).

After using the model with the training set, we then move to test data sets. We apply the model to the test set, and see if it correctly classifies members of the test set. This procedure is called supervised learning.

Often, we do tenfold cross validation; we repeat the testing process ten times, each with a different ten percent of the data. Typically, the average of these ten runs becomes the model parameters.

Once we have a model, then we can apply it to real data.

3.1.2 Entropy of a Random Distribution

Suppose we have a set of objects, and split up the set into classes. Figure 3.1 shows a division where the majority of objects fall into two regions. Figure 3.2 shows a division where objects are evenly distributed among regions.

Figure 3.1: Objects concentrated in two groups
Suppose we have a random variable $X$:

$$X: \begin{pmatrix} x_1 & \cdots & x_n \\ p_1 & \cdots & p_n \end{pmatrix}$$  \hspace{1cm} (3.1)

Each $p_i \in [0, 1]$ (or more likely, $(0, 1)$). Since $X$ is a probability distribution, $\sum_{i=1}^{n} p_i = 1$.

The entropy of the distribution is

$$H(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i$$  \hspace{1cm} (3.2)

The formula for $H$ is credited to Claude Shannon.

For example, suppose $p_1 = p_2 = 0.5$. Then $H(p_1, p_2) = 0.5 + 0.5 = 1$.

Now, suppose $p_1 = p_2 = \cdots = p_n = \frac{1}{n}$. Then

$$H\left(\frac{1}{n}, \frac{1}{n}, \ldots\right) = -n \cdot \frac{1}{n} \log_2 \frac{1}{n}$$

$$= -\log_2 \frac{1}{n}$$

This represents the maximum amount of non-determinism you can have in a classification question. $\lceil \log_2 n \rceil$ is the maximum number of questions needed to classify an object.

### 3.1.3 Convex Functions

A convex function is one that “holds water”.

Convex functions have positive second derivatives. If $f(x)$ is a convex function, then $f''(x) > 0$.

Figure 3.3 shows a convex function $f$.

Let’s pick two points, $x_1, x_2$. If we pick a point $u$ such that $x_1 \leq u \leq x_2$, then $u$ can be written as

$$u = tx_1 + (1-t)x_2$$

$$u = t(x_1 - x_2) + x_2$$

$$t = \frac{x_2 - u}{x_2 - x_1}$$

for $t \in [0, 1]$.

$u$ is called a convex combination of $x_1, x_2$. 
Figure 3.3: A convex function $f$

The convex curve of $f$ is always below the chord $(x_1, f(x_1))$ to $(x_2, f(x_2))$.

An example of a convex function:

\[
\begin{align*}
  f(x) &= x \ln(x) \\
  f'(x) &= \ln(x) + 1 \\
  f''(x) &= \frac{1}{x}
\end{align*}
\]
3.2 Lecture – 3/26/2008

3.2.1 Convex Functions

Given a probability distribution $X$,

$$X: \begin{pmatrix} x_1 & \cdots & x_n \\ p_1 & \cdots & p_n \end{pmatrix}$$

The entropy of $X$ is

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \quad (3.3)$$

**Note:** in what follows, any time we say log $n$, we really mean log$_2 n$; base two is implicit.

The more concentrated the values of $X$ are, the lower the entropy.

The highest entropy is $H(X) = -\log \frac{1}{n}$, which occurs when all $p_i$ are equal.

If $f''(x) > 0$, then $f$ is a **convex function** (it holds water).

Let’s take two points, $x_1$, and $x_2$, and a point in the middle, $tx_1 + (1-t)x_2$ (for $t \in [0,1]$):

$$x_1 \leq tx_1 + (1-t)x_2 \leq x_2$$

For a convex function $f$, we have the inequality

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \quad (3.4)$$

Equation (3.4) is the base case for the **Jensen Inequality**.

Let’s say we have a convex function $f$, and a probability distribution $(t_1, \ldots, t_n)$. All $t_i > 0$ and $\sum t_i = 1$. We can prove

$$f \left( \sum_{i=1}^{n} t_i x_i \right) \leq \sum_{i=1}^{n} t_i f(x_i) \quad (3.5)$$

Equation (3.5) is the Jensen Inequality.

We prove the Jensen Inequality using induction on $n$.

**Proof:** For the base case, $n = 2$, which is just the definition of a convex function. Assuming that the inequality holds for $n$, we just need to prove that it holds for $n+1$.

For $n+1$, we have $t_1, t_2, \ldots, t_{n-1}, t_n, t_{n+1} = 1$. Along with $t_i$, we also have values $x_1, x_2, \ldots, x_{n-1}, x_n, x_{n+1}$.

Looking at the left side of the inequality, we have

$$f \left( \sum_{i=1}^{n+1} t_i x_i \right) = f(t_1 x_1 + \ldots + t_{n-1}x_{n-1} + t_n x_n + t_{n+1}x_{n+1})$$

$$= f \left( t_1 x_1 + \ldots + t_{n-1}x_{n-1} + (t_n + t_{n+1}) \cdot \left( \frac{t_n x_n + t_{n+1}x_{n+1}}{t_n + t_{n+1}} \right) \right)$$

In the second line, we’ve done a little factoring of the last two terms, nothing more.

Using the inductive hypothesis, we can write

$$f \left( t_1 x_1 + \ldots + t_{n-1}x_{n-1} + (t_n + t_{n+1}) \cdot \left( \frac{t_n x_n + t_{n+1}x_{n+1}}{t_n + t_{n+1}} \right) \right)$$

$$\leq t_1 f(x_1) + t_2 f(x_2) + \ldots + t_{n-1} f(x_{n-1}) + (t_n + t_{n+1}) f \left( \frac{t_n x_n + t_{n+1}x_{n+1}}{t_n + t_{n+1}} \right)$$

This completes the proof.
Let’s focus on the last term:
\[
f\left(\frac{t_n x_n + t_{n+1} x_{n+1}}{t_n + t_{n+1}}\right) = f\left(\frac{t_n x_n + t_{n+1} x_{n+1}}{t_n + t_{n+1}}\right)
\leq \frac{t_n}{t_n + t_{n+1}} f(x_n) + \frac{t_{n+1}}{t_n + t_{n+1}} f(x_{n+1})
\leq t_n f(x_n) + t_{n+1} f(x_{n+1})
\]

□

Let’s look at a convex function:
\[
f(x) = x \log x
= x \frac{\ln x}{\ln 2}
\]
\[
f'(x) = \frac{1}{\ln 2} (\ln x + 1)
\]
\[
f''(x) = \frac{1}{\ln 2} \frac{1}{x}
\]
\[
f''(x) > 0, \text{ so } x \text{ is concave.}
\]

Let’s apply the Jensen Inequality to \(f(x)\), where \(t_1 = t_2 = \ldots = t_n = 1/n\).

Let \(X\) be a probability distribution where \(x_1 = p_1, \ldots, x_n = p_n\).

\[
f(1/n) \leq \sum_{i=1}^{n} p_i \log p_i
\]
\[
n f(1/n) \leq \sum_{i=1}^{n} p_i \log p_i
\]
\[
- \sum_{i=1}^{n} p_i \log p_i \leq - n \frac{1}{n} \log \frac{1}{n} = - \log n
\]

This represents the maximum entropy. (For the last line, we multiplied each side by \(-1\) and flipped the inequality.)

### 3.2.2 Two-Probability Distributions

Let’s consider the case of two probability distributions, \((p_1, \ldots, p_n), (q_1, \ldots, q_n)\), where \(p_i, q_i > 0\) and \(\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i = 1\).

We have
\[
\sum_{i=1}^{n} p_i \log \frac{p_i}{q_i} > 0 \tag{3.6}
\]

Equation \(3.6\) is called the Kullback-Leibler Inequality.

First, note the graph of \(y = \ln x\), and the tangent line \(y = 1 \cdot (x - 1)\). \(\ln x\) is always below the tangent line. (See Figure 3.4.)
Figure 3.4: $y = \ln x$ and tangent

Since $\ln x$ always falls below the tangent, we know that $\ln x \leq x - 1$. Let’s replace $x$ with $\frac{q_i}{p_i}$

\[
\ln \left( \frac{q_i}{p_i} \right) \leq \frac{q_i}{p_i} - 1
\]

\[
p_i \ln \left( \frac{q_i}{p_i} \right) \leq q_i - p_i
\]

\[
\sum_{i=1}^{n} p_i \ln \left( \frac{q_i}{p_i} \right) \leq \sum_{i=1}^{n} q_i - \sum_{i=1}^{n} p_i = 0
\]

\[
\sum_{i=1}^{n} p_i \ln \left( \frac{q_i}{p_i} \right) \leq 0
\]

Therefore,

\[
\sum_{i=1}^{n} p_i \ln \left( \frac{p_i}{q_i} \right) \geq 0
\]

(Above, pay attention to $\frac{q_i}{p_i}$ vs $\frac{p_i}{q_i}$.)

If each $p_i = q_i$, then $\sum p_i \ln \frac{p_i}{q_i} = 0$.

Back to entropy. Recall our definition $H(X) = -\sum_{i=1}^{n} p_i \log p_i$.

Conditional probability is

\[
P(A|B) = \frac{P(A \land B)}{P(B)}
\]

If $B$ is fixed (known), then $P(A|B)$ acts like an ordinary probability distribution.
Say we have two probability distributions, X and Y

\[
X: \begin{pmatrix} x_1 & \ldots & x_n \\ p_1 & \ldots & p_n \end{pmatrix} \\
Y: \begin{pmatrix} y_1 & \ldots & y_m \\ q_1 & \ldots & q_m \end{pmatrix}
\]

Note that X and Y have different numbers of elements.

Of course \( P(X = x_i) = p_i \) and \( P(Y = y_j) = q_j \).

If we pair X and Y up, then we have \((X, Y) = (x_i, y_j)\).

How does \((X, Y)\) look as a probability distribution?

\[
XY: \begin{pmatrix} x_1y_1 & \ldots & x_iy_j & \ldots & x_ny_m \\ r_{11} & \ldots & r_{ij} & \ldots & r_{nm} \end{pmatrix}
\]

\( r_{ij} \) is the probability of the conjunction of two events, so \( r_{ij} = P(X = x_i \land Y = y_j) \), and

\[
\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} = 1
\]

If \( P(X = x_i \land Y = y_j) = P(X = x_i) \cdot P(Y = y_j) \) for every \( i, j \), then we say that \( x, y \) are independent.

However, in real world data, it's quite common to have some level of dependency between X and Y.

The question we consider next: how do we evaluate entropy when X and Y are not independent?

**Definition 3.2.2.1 (Conditional Entropy):** The conditional entropy of X dependent on Y is

\[
H(X|Y = y_j) = - \sum_{i=1}^{n} P(X = x_i|Y = y_j) \log P(X = x_i|Y = y_j) \tag{3.7}
\]

This measures the entropy of X, assuming that Y is known.

If we take (3.7) and sum across the different Y values, then we have

\[
H(X|Y) = \sum_{j=1}^{m} P(Y = y_j) \cdot H(X|Y = y_j) \tag{3.8}
\]
Let’s work with $H(X|Y)$.

$$H(X|Y) = \sum_{j=1}^{m} P(Y = y_j) \cdot H(X|Y = y_j)$$

$$= - \sum_{j=1}^{m} P(Y = y_j) \cdot \sum_{i=1}^{n} P(X = x_i|Y = y_j) \log P(X = x_i|Y = y_j)$$

$$= - \sum_{i=1}^{n} \sum_{j=1}^{m} P(Y = y_j) \cdot P(X = x_i|Y = y_j) \log P(X = x_i|Y = y_j)$$

$$= - \sum_{i=1}^{n} \sum_{j=1}^{m} P(Y = y_j) \cdot \frac{P(X = x_i \land Y = y_j)}{P(Y = y_j)} \log \frac{P(X = x_i \land Y = y_j)}{P(Y = y_j)}$$

$$= - \sum_{i=1}^{n} \sum_{j=1}^{m} P(X = x_i \land Y = y_j) \cdot \log \left( \frac{P(X = x_i \land Y = y_j)}{P(Y = y_j)} \right)$$

$$= - \left( \sum_{i=1}^{n} \sum_{j=1}^{m} P(X = x_i \land Y = y_j) \cdot \log (P(X = x_i \land Y = y_j)) \right)$$

$$+ \left( \sum_{i=1}^{n} \sum_{j=1}^{m} P(X = x_i \land Y = y_j) \cdot \log (P(Y = y_j)) \right)$$

$$= H(X, Y) + \sum_{i=1}^{n} \sum_{j=1}^{m} P(X = x_i \land Y = y_j) \cdot \log P(Y = y_j)$$

$$= H(X, Y) + \sum_{j=1}^{m} \left( \sum_{i=1}^{n} P(X = x_i \land Y = y_j) \right) \cdot \log P(Y = y_j)$$

$$= H(X, Y) - \sum_{j=1}^{m} P(Y = y_j) \cdot \log P(Y = y_j)$$

$$= H(X, Y) - H(Y)$$

Therefore, $H(X|Y) = H(X, Y) - H(Y)$.

Also

$$H(X, Y) = H(X|Y) + H(Y)$$

$$= H(Y|X) + H(X)$$

$H(X, Y)$ is the **joint entropy** of $X$ and $Y$.

Next, we’ll prove that

$$H(X, Y) \leq H(X) + H(Y)$$
Proof: First note that $\sum_{j=1}^{m} r_{ij} = p_i$ and $\sum_{i=1}^{n} r_{ij} = q_j$.

$$- \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log r_{ij} \leq - \sum_{i=1}^{n} p_i \log p_i - \sum_{j=1}^{m} q_j \log q_j$$

$$\leq - \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log p_i - \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log q_j$$

$$= - \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log p_i q_j$$

Therefore

$$- \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log r_{ij} \leq - \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log p_i q_j$$

and

$$\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log r_{ij} - \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log p_i q_j \geq 0$$

and

$$\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log \frac{r_{ij}}{p_i q_j} \leq 0$$

The last line is equivalent to the Kullback-Leibler Inequality. \hfill \Box

3.2.3 To-Do

- Do a little research on the topic of “Information Theory”. It will provide some useful background information.


- Track down the “Play Tennis Data Set”. We’ll be using this for classification problems.
### 3.3 Lecture – 3/31/2008

#### 3.3.1 Classification

Table 3.1 shows a set of data called the ‘play tennis’ data set. We’ll use this to work out a classification example.

<table>
<thead>
<tr>
<th></th>
<th>Outlook</th>
<th>Temp</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 3.1: Play Tennis Data Set

Our objective: given this training set of tennis data, we’d like to understand what days are good (and what days are not good) for playing tennis.

Given the cardinality of the different data attributes, there are 36 possible values. We have 14 values – these 14 values will serve as a *training set*. We use use the training set to construct a classifier that can be applied to the more general data set.

In this example, “Play” is the *class attribute*. Play is the attribute that we’re trying to understand.

Our example will use *supervised classification*. An expert has provided class assignments for each row of the training set.

If we find that one attribute determines the class, then our job is easy. That attribute becomes the predictor and we’re done. In real life, this is extremely rare.

Also, if we find that the class attribute only takes on one value, then our job is also done. With a single-valued class attribute, there’s no way to split the data set. Again, this is rarely the case.

Our example will use the ID3 classification algorithm. C4.5 is another popular classification algorithm. In Weka, these are called “ID3” and “J48” respectively.

Recall our definition of *entropy*

\[
H(p_1, \ldots, p_n) = - \sum_{i=1}^{n} p_i \log p_i
\]  

(When we say log, log_2 is implicit).

Suppose we had the probability distribution

\[
\text{Play: } \begin{pmatrix} \text{yes} & \text{no} \\ 9/14 & 5/14 \end{pmatrix}
\]
Then, the entropy is
\[
H(9/14, 5/15) = - \left( \frac{9}{14} \log \frac{9}{14} + \frac{5}{15} \log \frac{5}{14} \right)
\]
\[= 0.94\]

Note: given a zero probability, we’ll adopt the convention that \(0 \cdot \log 0 = 0\).

3.3.2 ID3 Algorithm Example

ID3 picks one attribute, and splits the decision tree along that attribute. With each split, we seek to maximize the entropy gain. For example, we choose the attribute \(A\) that maximizes
\[
H(\text{play}) - H(\text{play}|A)
\]
The equation we want to maximize is
\[
H(\text{play}) - H(\text{play}|A) = H(\text{play}) + H(A) - H(\text{play} \land A)
\]
On the right hand side, \(H(\text{play})\) is a constant, so we only need to work with \(H(A) - H(\text{play} \land A)\).

Split 1

We have four attributes \(A\) to consider: outlook, temperature, humidity, and wind. First, we calculate the entropy of the individual attributes.
\[
H(\text{outlook}) = H(\text{sunny, overcast, rain})
= H(5/14, 4/14, 5/14)
= 1.577
\]
\[
H(\text{temp}) = H(\text{hot, mild, cool})
= H(4/15, 6/15, 4/15)
= 1.556
\]
\[
H(\text{humidity}) = H(\text{high, normal})
= H(7/14, 7/14)
= 1
\]
\[
H(\text{wind}) = H(\text{strong, weak})
= H(6/14, 8/14)
= 0.985
\]

Note: for these entropy calculations, all that we have done is to count the number of times that each attribute value appears in the data set.

Next, we compute the joint entropy \(H(A \land \text{play})\) for each of the four \(A\) above.

First, we count up the joint frequencies

<table>
<thead>
<tr>
<th>outlook</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>sunny</td>
<td>no</td>
<td>3</td>
</tr>
<tr>
<td>overcast</td>
<td>yes</td>
<td>4</td>
</tr>
<tr>
<td>overcast</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>rain</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>rain</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>temp</td>
<td>play</td>
<td>freq</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>hot</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>hot</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>mild</td>
<td>yes</td>
<td>4</td>
</tr>
<tr>
<td>mild</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>cool</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>cool</td>
<td>no</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>humidity</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>high</td>
<td>no</td>
<td>4</td>
</tr>
<tr>
<td>normal</td>
<td>yes</td>
<td>6</td>
</tr>
<tr>
<td>normal</td>
<td>no</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wind</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td>yes</td>
<td>6</td>
</tr>
<tr>
<td>weak</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>strong</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>strong</td>
<td>no</td>
<td>3</td>
</tr>
</tbody>
</table>

From these frequencies, we compute the joint entropies:

\[
H(\text{outlook} \land \text{play}) = H(2/14, 3/14, 4/14, 1/14) = 2.270
\]

\[
H(\text{temp} \land \text{play}) = H(2/14, 2/14, 4/14, 2/14, 1/14) = 2.467
\]

\[
H(\text{humidity} \land \text{play}) = H(3/14, 4/14, 6/14, 1/14) = 1.788
\]

\[
H(\text{wind} \land \text{play}) = H(6/14, 2/14, 3/14, 3/14) = 1.877
\]

Finally, we compute \( H(A) - H(\text{play} \land A) \):

<table>
<thead>
<tr>
<th>attribute</th>
<th>entropy gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>outlook</td>
<td>1.577 - 2.270 = -0.693</td>
</tr>
<tr>
<td>temp</td>
<td>1.566 - 2.467 = -0.911</td>
</tr>
<tr>
<td>humidity</td>
<td>1 - 1.788 = -0.788</td>
</tr>
<tr>
<td>wind</td>
<td>0.985 - 1.877 = -0.892</td>
</tr>
</tbody>
</table>

Outlook has the largest entropy gain, so we'll perform the split on that attribute. Notice that this forms a partition of the data. This is shown in Figure 3.5.

In Figure 3.5 the numbers in the boxes represent row numbers. Below each box are the Play assignments. For outlook = overcast, we have homogeneity of the class attribute values (all yes). This tells us that “outlook = overcast” is a leaf, and we don’t need to divide it further.
For reference, we’ll refer to these sets as \( S_{\text{sunny}} \), \( S_{\text{overcast}} \), and \( S_{\text{rain}} \).

**Split 2A**

We apply ID3 recursively to the tree. At each step in the recursion, we ignore attributes that have already been used to partition a parent node.

For our next step, we’ll split \( S_{\text{sunny}} \). It’s essentially the same process, but we don’t consider “outlook”.

For convenience, we’ll isolate the subset of data that applies to \( S_{\text{sunny}} \). This data is in Table 3.2

<table>
<thead>
<tr>
<th>Temp</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hot</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>hot</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>8</td>
<td>mild</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>cool</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>mild</td>
<td>strong</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 3.2: \( S_{\text{sunny}} \) data

First, we find the entropy of the attributes Temp, Humidity, and Wind (using only the data in table 3.2).

\[
H(\text{temp}) = H(\text{hot, mild, cool}) \\
= H(2/5, 2/5, 1/5) \\
= 1.521 \\
H(\text{humidity}) = H(\text{high, normal}) \\
= H(3/5, 2/5) \\
= 0.970 \\
H(\text{wind}) = H(\text{weak, strong}) \\
= H(3/5, 2/5) \\
= 0.970
\]

Next, we find the joint entropy \( H(A \land \text{play}) \) for each of these three attributes.

<table>
<thead>
<tr>
<th>temp</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>yes</td>
<td>0</td>
</tr>
<tr>
<td>hot</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>mild</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>mild</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>cool</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>cool</td>
<td>no</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>humidity</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>yes</td>
<td>0</td>
</tr>
<tr>
<td>high</td>
<td>no</td>
<td>3</td>
</tr>
<tr>
<td>normal</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>normal</td>
<td>no</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wind</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>weak</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>strong</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>strong</td>
<td>no</td>
<td>1</td>
</tr>
</tbody>
</table>
The joint entropies are:

\[
H(\text{temp} \land \text{play}) = H(2/5, 1/5, 1/5, 1/5) = 1.921
\]

\[
H(\text{humidity} \land \text{play}) = H(3/5, 2/5) = 0.970
\]

\[
H(\text{wind} \land \text{play}) = H(1/5, 2/5, 1/5, 1/5) = 1.921
\]

Finally, we compute the entropy gain, \( H(A) - H(\text{play} \land A) \).

<table>
<thead>
<tr>
<th>attribute</th>
<th>entropy gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp</td>
<td>1.521 - 1.921 = -0.4</td>
</tr>
<tr>
<td>humidity</td>
<td>0.970 - 0.970 = 0</td>
</tr>
<tr>
<td>wind</td>
<td>0.970 - 1.921 = -0.951</td>
</tr>
</tbody>
</table>

Here, “humidity” has the highest entropy gain, so we use that to split. The split is shown in Figure 3.6.

Note that both of the new nodes in Figure 3.6 are homogeneous with respect to their Play values. Therefore, we are done with those.

**Split 2B**

Next, we’ll work on splitting \( S_{\text{rain}} \). Again, we’ll isolate ourselves to the subset of data that interests us. This subset appears in Table 3.3.

<table>
<thead>
<tr>
<th>Temp</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>mild</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
</tr>
<tr>
<td>6</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
</tr>
<tr>
<td>10</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
</tr>
<tr>
<td>14</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
</tr>
</tbody>
</table>

Table 3.3: \( S_{\text{rain}} \) data subset
As before, we’ll start by finding the entropy for the three attributes Temperature, Humidity, and Wind.

\[
H(\text{temp}) = H(\text{mild}, \text{cool}) = H(3/5, 2/5) = 0.970
\]

\[
H(\text{humidity}) = H(\text{high}, \text{normal}) = H(2/5, 3/5) = 0.970
\]

\[
H(\text{wind}) = H(\text{weak}, \text{strong}) = H(3/5, 2/5) = 0.970
\]

Next, we’ll determine the frequencies for the joint entropy computations:

<table>
<thead>
<tr>
<th>temp</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>mild</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>mild</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>cool</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>cool</td>
<td>no</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>humidity</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>high</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>normal</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>normal</td>
<td>no</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wind</th>
<th>play</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>weak</td>
<td>yes</td>
<td>3</td>
</tr>
<tr>
<td>weak</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>strong</td>
<td>yes</td>
<td>0</td>
</tr>
<tr>
<td>strong</td>
<td>no</td>
<td>2</td>
</tr>
</tbody>
</table>

Next, we find the joint entropies

\[
H(\text{temp} \land \text{play}) = H(2/5, 1/5, 1/5, 1/5) = 1.921
\]

\[
H(\text{humidity} \land \text{play}) = H(1/5, 1/5, 2/5, 1/5) = 1.921
\]

\[
H(\text{wind} \land \text{play}) = H(3/5, 2/5) = 0.970
\]

Finally, we compute the entropy gain.

<table>
<thead>
<tr>
<th>attribute</th>
<th>entropy gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp</td>
<td>0.970 – 1.921 = −0.951</td>
</tr>
<tr>
<td>humidity</td>
<td>0.970 – 1.921 = −0.951</td>
</tr>
<tr>
<td>wind</td>
<td>0.970 – 0.970 = 0</td>
</tr>
</tbody>
</table>

Here, “wind” gives us the highest entropy gain, so we’ll split on that. The new tree is shown in Figure 3.7.

All leaves of Figure 3.7 are homogeneous. So, we’re done.
Rule Generation

This decision tree gives us several rules for classification. For example:

(outlook = overcast) ⇒ (play = yes)
(outlook = sunny) ∧ (humidity = high) ⇒ (play = no)

3.3.3 ID3 Caveats

When splitting, ID3 tends to favor attributes with more values. The decisions at the lower levels can be dicey, since you work with smaller and smaller sample sizes.

Models that fit the training set well may work badly for real data, due to over-fitting (a.k.a “fitting to the training set”). This problem is not unique to ID3.
3.4 Lecture – 4/2/2008

3.4.1 Decision Trees and ID3

Last class, we worked through an example of decision tree construction (see page 85). In our example, all leaves were homogeneous. This will not always be the case – some data sets will produce leaves that are not homogeneous.

Our example used an algorithm called ID3, which was the first algorithm developed for constructing decision trees. ID3 is based on the notion of comparing entropy gain of attributes vs the entropy of the class.

\[ H(C) - H(C|A) \]
\[ = H(C) + H(A) - H(A \land C) \]

Also recall

\[ H(C \land A) = H(C|A) + H(A) \]

Let’s rewrite entropy gain, using the definition of conditional entropy

\[ H(C) - H(C|A) \]
\[ = H(C) - \sum_j P(A = a_j) \cdot H(C|A = a_j) \]

If there are many attributes, then the \( \sum_j P(A = a_j) \cdot H(C|A = a_j) \) term will be large. Therefore, ID3 favors attributes that generate lots of splits (high cardinality).

3.4.2 C4.5

C4.5 is another decision tree algorithm. For the most part, C4.5 is an improved version of ID3. It was created by Quinlan.

C4.5 uses (3.10) instead of entropy gain.

\[ \frac{H(C) - H(C|A)}{H(A)} \] (3.10)

3.4.3 Weka

Weka uses an input file format called ARFF. For our next homework assignment, we’ll need to write an ARFF format converter.

By default, Weka assumes that the last attribute is the class attribute.

Alongside decision tree construction, one of the by-products that Weka produces is a confusion matrix. For example, when constructing a decision tree for the tennis set with 10x cross validation, Weka gives the confusion matrix

\[
\begin{pmatrix}
a & b \\
8 & 1 \\
1 & 4
\end{pmatrix}
\]

What is this telling us? For the ‘a’ (yes) values, cross validation got eight of them right, and one of them wrong. The wrong value (b) is called a false negative from a’s perspective (or a false positive from b’s perspective).

Likewise, four of the b’s were right, and one was wrong.
3.4.4 Numeric vs. Nominal Attributes

ID3 and C4.5 only work with nominal attributes. A *nominal attribute* takes on a discrete set of unordered values. Of course, *numeric attributes* are ordered and not discrete.

We’ll discuss numeric attributes in a little while.

3.4.5 Entropy and the Gini Index

Entropy is

\[
H(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i
\]

Why is entropy useful? Entropy measures how values are distributed. Entropy is at its maximum when all \( p_i \) are equal. Entropy is zero if some \( p_i = 1 \) and all other \( p_j = 0 \) (for \( j \neq i \)).

There are other measures that accomplish the same goal.

Consider, \( H_\beta \), which is called *generalized entropy*.

\[
H_\beta(p_1, \ldots, p_n) = \frac{1}{1 - 2^{1-\beta}} \left( 1 - \sum_{i=1}^{n} p_i^\beta \right)
\]

for \( \beta > 1 \) (3.11)

General entropy is the invention of Havrda and Charvat.

If we take the limit of \( H_\beta \) as \( \beta \to 0 \),

\[
\lim_{\beta \to 0} H_\beta(p_1, \ldots, p_n) = \lim_{\beta \to 0} -\sum_{i=1}^{n} p_i^\beta \ln p_i = -\sum_{i=1}^{n} p_i \ln p_i - \sum_{i=1}^{n} p_i \log_2 p_i
\]

L’Hospital’s Rule

The term \( -\sum_{i=1}^{n} p_i \log_2 p_i \) is our definition of entropy.

Our definition of entropy \( H \) is really just a special case of generalized entropy.

Next, let’s look at \( H_\beta \) for \( \beta = 2 \).

\[
H_2(p_1, \ldots, p_n) = 2 \left( 1 - \sum_{i=1}^{n} p_i^2 \right)
\]

Gini Index (3.12)

Equation (3.12) is called the *gini index*.

The Gini index works like entropy. It’s largest when values are uniform, and zero when \( p_i = 1 \) and \( p_j = 0 \) for \( j \neq i \).

3.4.6 CART Method

The CART method uses the Gini index. Cart can handle both numeric and nominal attributes.
Some manipulation of $H_\beta$:

$$H_\beta(p_1, \ldots, p_n) = \frac{1}{1 - 2^{1 - \beta}} \cdot \left(1 - \sum_{i=1}^{n} p_i^\beta\right)$$

$$= \frac{1}{1 - 2^{1 - \beta}} \cdot \left(\sum_{i=1}^{n} p_i - \sum_{i=1}^{n} p_i^\beta\right)$$

since sum of $p_i = 1$

$$= \frac{\sum_{i=1}^{n} p_i - p_i^\beta}{1 - 2^{1 - \beta}}$$

CART was invented by Breiman.

CART’s splitting is similar to ID3 and entropy gain, but it’s based on a different principle.

CART always does binary splits.

For numerical splits, we’ll try a series of possible split values within the range of the attribute. For each possible split value, we compute a Gini index. The Gini indexes determine what the split point is.

The gini index for a class attribute:

$$\text{Gini}(C) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2$$

(3.13)

For Gini,

$$\text{Gini}(C) = \sum_{j=1}^{n} \left(P(A = a_j)^2 \cdot \text{Gini}(C|A = a_j)\right)$$

(3.14)

CART chooses split points that maximize the Gini gain.

Nominal attribute take a little more work, since we have to consider all possible split points.

### 3.4.7 Convex Functions

Consider the function

$$h(x) = x - x^\beta \quad \text{for } x \geq 0, \beta \geq 1$$

$$h'(x) = 1 - \beta x^{\beta - 1}$$

The function $h(x)$ is shown in figure 3.8

$h(x)$ is a concave function that crosses the $x$-axis at $(0, 0)$ and $(1, 0)$. Note: if we make a chord between any pair of points in the range $0 < x < 1$, $h(x)$ will lie above that chord.

Convex functions have properties that are similar to convex functions (just upside down). For one,

$$h(p_1 x_1, \ldots, p_n x_n) \geq p_1 h(x_1) + \ldots + p_n h(x_n)$$

Say $p_1 = p_2 = \ldots = p_n = 1/n$. Then

$$\frac{h(x_1 + \ldots + x_n)}{n} \geq \frac{1}{n} (h(x_1) + \ldots + h(x_n))$$

$$h(1/n) \geq \frac{1}{n} (h(p_1) + \ldots + h(p_n))$$

$$n \cdot h(1/n) \geq h(x_1) + \ldots + h(x_n)$$
3.4.8 Misc Notes

Derivatives for exponents

\[(a^x)' = a^x \ln x\] (3.15)

3.4.9 Next Class

- Start reading over Naïve Bayes classification, Perceptrons, and Neural Networks.
- Between now and Monday, check the course web site for new handouts.
3.5 Lecture – 4/7/2008

3.5.1 Naïve Bayes Classifiers

Bayes classification is based on work by Thomas Bayes (1702-1761).

We have a definition of conditional probability

\[ P(A|B) = \frac{P(A \land B)}{P(B)} \quad \text{for } P(B) \neq 0 \]

If \( B \) is fixed, this behaves like any other probability.

Like \( P(A|B) \), we can have \( P(B|A) \), the probability of \( B \) conditioned by \( A \):

\[ P(B|A) = \frac{P(B \land A)}{P(A)} \]

Equation (3.16) is known as the Bayes Formula

\[ P(B|A) \cdot P(A) = P(A|B) \cdot P(B) \quad (3.16) \]

There is also a “chain rule” for conditional probability. Consider

\[ P(A|B \land C) \cdot P(B|C) \cdot P(C) \quad (3.17) \]

If we expand these terms, using the definitional of conditional probability, we have

\[
\begin{align*}
P(A|B \land C) & \cdot P(B|C) \cdot P(C) \\
= & \frac{P(A \land B \land C)}{P(B \land C)} \cdot \frac{P(B \land C)}{P(C)} \cdot P(C) \\
= & P(A \land B \land C)
\end{align*}
\]

The chain rule of conditional probability is

\[ P(A \land B \land C) = P(A|B \land C) \cdot P(B|C) \cdot P(C) \quad (3.18) \]

Of course, (3.18) can be carried out to arbitrary lengths.

If \( A, B \) are independent, then \( P(A \land B) = P(A)P(B) \).

Also,

\[
\begin{align*}
P(A|B) & = P(A) \quad \text{if } A, B \text{ independent} \\
P(B|A) & = P(B) \quad \text{if } A, B \text{ independent}
\end{align*}
\]

3.5.2 Bayes Classifiers and Data Mining

For data mining, assume that we have a dataset \( D \) (for classification), and a set of classes \( C = \{c_1, \ldots, c_n\} \).

Our data set has attributes \( A_1, \ldots, A_n \), along with a class attribute.

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>( \ldots )</th>
<th>( A_n )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( \ldots )</td>
<td>( a_n )</td>
<td>( c_i )</td>
</tr>
</tbody>
</table>
In principle, $D$ is a subset of the cartesian product of attributes. $D$ does not contain every tuple of $(A_1 \times \ldots \times A_n)$.

Given a tuple $(b_1, \ldots, b_n) \notin D$, our job is to classify it. In the context of Bayes theorem, we need to choose a class $c$ which maximizes

$$P(C|b_1, \ldots, b_n)$$

Note again that $(b_1, \ldots, b_n)$ is not part of the data set (if it were part of the dataset, then the class would be known, and we could simply look it up).

We can say

$$P(C|b_1, \ldots, b_n) \cdot P(b_1, \ldots, b_n) = P(b_1, \ldots, b_n|C) \cdot P(C)$$

or equivalently

$$P(C|b_1, \ldots, b_n) = \frac{P(b_1, \ldots, b_n|C) \cdot P(C)}{P(b_1, \ldots, b_n)}$$

For a given row (i.e., the one we are attempting to classify), $P(b_1, \ldots, b_n)$ is fixed. Therefore, we need to find a $C$ that maximizes

$$P(b_1, \ldots, b_n|C) \cdot P(C)$$

To perform the classification, we make the following hypothesis.

**Hypothesis 3.5.2.1:** $b_1, \ldots, b_n$ are independent in the presence of $C$. In other words,

$$P(b_1, \ldots, b_n|C) = P(b_1|C) \cdot P(b_2|C) \cdot \ldots \cdot P(b_n|C)$$

Equation (3.23) is the fundamental assumption of Naïve Bayes Classifiers. This assumption is not always realistic, but it tends to work well in practice.

**Example 3.5.2.2 (Naïve Bayes Classification):** Recall our “play tennis” data set.

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temp</th>
<th>Humidity</th>
<th>Wind</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

We wish to classify a new tuple: (sunny, cool, high, strong).

First, we find the probabilities of the class attribute:

$$P(play = yes) = \frac{9}{14}$$  
$$P(play = no) = \frac{5}{14}$$
Next, we find the conditional probability of each attribute:

\[
\begin{align*}
P(\text{outlook} = \text{sunny} | \text{play} = \text{yes}) &= \frac{2}{9} \\
P(\text{outlook} = \text{sunny} | \text{play} = \text{no}) &= \frac{3}{5} \\
P(\text{temp} = \text{cool} | \text{play} = \text{yes}) &= \frac{3}{9} \\
P(\text{temp} = \text{cool} | \text{play} = \text{no}) &= \frac{1}{5} \\
P(\text{humidity} = \text{high} | \text{play} = \text{yes}) &= \frac{3}{9} \\
P(\text{humidity} = \text{high} | \text{play} = \text{no}) &= \frac{4}{5} \\
P(\text{wind} = \text{strong} | \text{play} = \text{yes}) &= \frac{3}{5} \\
P(\text{wind} = \text{strong} | \text{play} = \text{no}) &= \frac{3}{5}
\end{align*}
\]

Next, we combine the conditional probabilities:

\[
\begin{align*}
P(\text{sunny} \land \text{cool} \land \text{high} \land \text{strong} | \text{play} = \text{yes}) \cdot P(\text{play} = \text{yes}) &= \frac{2}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{9}{14} \\
&= \frac{2}{378} \\
&= 0.005291
\end{align*}
\]

\[
\begin{align*}
P(\text{sunny} \land \text{cool} \land \text{high} \land \text{strong} | \text{play} = \text{yes}) \cdot P(\text{play} = \text{no}) &= \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} \cdot \frac{5}{14} \\
&= \frac{36}{1750} \\
&= 0.020571
\end{align*}
\]

The probability for “play = no”, is larger, so we classify this tuple as “no”. □

### 3.5.3 Weaknesses in Naïve Bayes Classification

Suppose we had an attribute that wasn’t in our data set. For example, outlook = snow. In this case, our data set would give us

\[
P(\text{outlook} = \text{snow} | \text{play}) = 0
\]

Because ‘snow’ does not appear in the data set, we are guaranteed to get a probability of zero.

One strategy for handling this is Laplace Correction. Suppose we have a class \(c\) and a value \(v\), which occurs \(n_v\) times. Then

\[
P(v | c) = \frac{n_v}{n}
\]
If $v$ is underrepresented in the data set, then $P(v|c)$ will be artificially small. However, prior knowledge may indicate a larger probability: $v$ is not rare – but $v$ happens to be rare in our data set.

Instead, we can use

$$
\frac{n_v + mp}{n + m}
$$

(3.24)

where $m$ is the simulated sample size (obtained from prior knowledge). If $m > n$, then (3.24) will tend to $p$.

This problem appears in other classification techniques, but the effect is (often) less pronounced than what we see with Naïve Bayes classifiers.

Naïve Bayes classifiers are easier to implement than decision trees (ID3, etc), and they tend to produce very good results.

### 3.5.4 Bayesian Networks

Suppose we have a set of factors, $x_1, \ldots, x_n$, along with an associated set of probabilities. Bayesian networks are a graphical way to represent this using conditional probabilities.

Recall our chain rule formula

$$
P(A \land B \land C) = P(A|B \land C) \cdot P(B|C) \cdot P(C)
$$

This is represented in Figure 3.9.

Figure 3.9: A very simple Bayesian Network

You can read this as “$C$ determines $B$”, and “$B, C$ determines $A$”.

If $A$ is independent of $B$ in the presence of $C$, then

$$
P(A|B \land C) = P(A|C)
$$

**Example 3.5.4.1:** Figure 3.10 shows another example of a Bayesian Network. The boxes are events, and the arrows indicate causality. For example “lightning causes thunder”, and “storm causes lightning”.

Figure 3.10 is qualitative: given $x$, it shows how $x$ may have been caused by $y$.

To make the graph quantitative, we need to add more information. For example: Fire has three “parent” nodes: Lightning, Storm, and Campfire. Campfire has two parent nodes: Tourist and Storm.

To turn this graph into a Bayesian Network, we need to assign conditional probabilities to each edge.
Figure 3.10: A Bayesian Network for Forest Fires

For example:

\[ P(\text{campfire } = \text{yes}|\text{storm } = \text{yes} \land \text{tourist } = \text{yes}) \]
\[ P(\text{campfire } = \text{no}|\text{storm } = \text{yes} \land \text{tourist } = \text{yes}) \]
\[ P(\text{campfire } = \text{yes}|\text{storm } = \text{no} \land \text{tourist } = \text{yes}) \]
\[ P(\text{campfire } = \text{no}|\text{storm } = \text{no} \land \text{tourist } = \text{yes}) \]
\[ P(\text{campfire } = \text{yes}|\text{storm } = \text{yes} \land \text{tourist } = \text{no}) \]
\[ P(\text{campfire } = \text{no}|\text{storm } = \text{yes} \land \text{tourist } = \text{no}) \]
\[ P(\text{campfire } = \text{yes}|\text{storm } = \text{no} \land \text{tourist } = \text{no}) \]
\[ P(\text{campfire } = \text{no}|\text{storm } = \text{no} \land \text{tourist } = \text{no}) \]

Above, campfire acts as our class. There are two attribute (storm and tourist) and \(2^{n+1}\) different probabilities.

The process of finding these probabilities is iterative, and uses “gradient descent”.

Let \(h\) be a network. We want to find \(P(D|h)\).
3.6 Notes on Laplacian Correction

(This comes from p. 315 of Han)

Suppose we have 1000 tuples where

- 10 tuples have income = high
- 990 tuples have income = medium
- 0 tuples have income = low

With no correction, we have probabilities

\[ P(\text{income = high}) = \frac{10}{1000} = 0.01 \]
\[ P(\text{income = med}) = \frac{990}{1000} = 0.99 \]
\[ P(\text{income = low}) = \frac{0}{1000} = 0 \]

To perform Laplacian correction, we assume that there is one additional tuple per category. For this example, we’d assume three additional tuples: one low, one medium, and one high. This brings our total number of tuples to 1,003.

The corrected probabilities are

\[ P(\text{income = high}) = \frac{10 + 1}{1000 + 3} = \frac{11}{1003} = 0.011 \]
\[ P(\text{income = med}) = \frac{990 + 1}{1000 + 3} = \frac{991}{1003} = 0.988 \]
\[ P(\text{income = low}) = \frac{0 + 1}{1000 + 3} = \frac{1}{1003} = 0.001 \]

The corrected probabilities are “close” to their uncorrected counterparts, but they avoid the zero probability issue.
3.7 ARFF Notes – 4/9/2008

(These notes come from pp. 53–58 of Witten and Frank.)

In ARFF files, the character % denotes a comment.

The header of an ARFF file contains

- The relation name (@relation)
- A set of attributes (@attribute)
- Data section (@data)

There are four types of attributes:

- Nominal
- Numeric
- String
- Date

Nominal attribute declarations contain a brace-enclosed list of attribute values.

Date attributes must be given in ISO-8601 format (in strftime terms, that’s %Y-%m-%dT%H:%M:%S).

String attributes are enclosed in quotes. Embedded quotation marks should be backslash-escaped.

ARFF uses one line per record.

ARFF uses a single question mark to represent missing values.

Here’s an example of an ARFF File

% Here is some weather data
@relation weather

@attribute outlook { sunny, overcast, rainy }
@attribute temperature numeric
@attribute humidity numeric
@attribute windy { true, false }
@attribute play? { yes, no }
@data
%
% Data rows follow
%
sunny, 85, 85, false, no
sunny, 80, 80, true, no
overcast, 83, 86, false, no
rainy, 70, 96, false, yes
3.8 Lecture – 4/9/2008

3.8.1 Bayesian Networks

A Bayesian network is a directed graph whose nodes are random variables $X_1, \ldots, X_n$.

For each node $X_i$, there is a set of nodes $X_{j_1}, \ldots, X_{j_p}$ which are the immediate predecessors of $X_i$, such that $j_1, \ldots, j_p < i$. In other words, if you were to number the nodes of a Bayesian Network, then the numbering would be consistent with a topological sort.

Also, for each node $X_i$ and each set of parents $X_{j_1}, \ldots, X_{j_p}$, we are given the conditional probability

$$P(X_i = x_i | X_{j_1} = x_{j_1}, \ldots, X_{j_p} = x_{j_p})$$

Figure 3.11 shows a simple example of a Bayesian Network, where

- $R$ means “it rained”,
- $F$ means “forgot to turn the sprinkler off”
- $W_W$ means “Watson’s grass is wet”, and
- $W_H$ means “Holmes’ grass is wet”.

(Holmes has a sprinkler system, Watson does not).

![Bayesian Network Diagram](image)

Let’s consider $W_H$. The parent nodes give us four combination of variables: $R \land F$, $R \land \overline{F}$, $\overline{R} \land F$, and $\overline{R} \land \overline{F}$. $W_H$ itself can take on two values: $W_H$ and $\overline{W_H}$.

This gives us two probability tables

<table>
<thead>
<tr>
<th></th>
<th>$R \land F$</th>
<th>$R \land \overline{F}$</th>
<th>$\overline{R} \land F$</th>
<th>$\overline{R} \land \overline{F}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_H$</td>
<td>0.5</td>
<td>0.6</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>$\overline{W_H}$</td>
<td>0.5</td>
<td>0.4</td>
<td>0.7</td>
<td>0.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$R$</th>
<th>$\overline{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_H$</td>
<td>0.9</td>
<td>0.05</td>
</tr>
<tr>
<td>$\overline{W_H}$</td>
<td>0.1</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Note that each column sums to one. This captures (for example) the notion that

$$P(W_H | R \land F) + P(\overline{W_H} | R, F) = 1$$

For further discussion, we’ll re-draw our little Bayesian Network with more abstract names.

In this model,

$$P(U = u \land X = x \land Z = z \land Y = y) = P(Z = z | X = x \land U = u) \cdot P(Y = y | U = u) \cdot P(U = u) \cdot P(X = x)$$
In other words, the values of \( Y \) and \( Z \) are conditionally independent of nodes which are not their ancestors. In other words,

\[
P(Z = z | U = u \land X = x \land Y = y) = P(Z = z | X = x \land U = u)
\]

Or equivalently,

\[
P(Z = z | X = x \land Y = y \land U = u) = \frac{P(Z = z \land X = x \land U = u \land Y = y)}{P(X = x \land U = u)} = \frac{P(Z = z) \cdot P(X = x \land U = u \land Y = y)}{P(X = x \land U = u)}
\]

Let’s look at \( Y \):

\[
P(U = u \land X = x \land Z = z \land Y = y) = \frac{P(Z = z \land X = x \land U = u \land Y = y)}{P(X = x \land U = u)} \cdot \frac{P(Y = y \land U = u)}{P(U = u)} \cdot P(X = x)
\]

Since \( U \) and \( X \) are parent nodes and independent of one another,

\[
P(U = u \land X = x \land Z = z \land Y = y) = \frac{P(Y = y \land U = u)}{P(U = u)} \cdot P(X = x)
\]

From the last line, it follows that

\[
P(Y = y | X = x \land U = u \land Z = z) = P(Y = y | U = u)
\]

### 3.8.2 Finding Probabilities in Bayesian Networks

The previous example give us an idea of how the probabilities work. But how do we find these numbers when we have a bunch of real data? (A real Bayesian Network might have hundreds of nodes.)

One of the simplest methods is something called the gradient ascent method (sometimes called “gradient descent method” in the literature).

Suppose you have \( y = f(x) \) where \( f(x) \) is both continuous and differentiable, like the one shown in Figure 3.12. The function shown in Figure 3.12 has four points of inflection, and four local extremum.

Suppose \( f \) is a very complex function, where it will be difficult to find all of the roots. We will estimate where local extreme points are.

- We pick an \( x_0 \), and find \( f'(x_0) \).
- \( f'(x_0) \) is the slope of a tangent line
- Next, pick an \( x_1 \), where \( x_1 = x_0 - \eta \cdot f'(x_0) \)
- Find \( f(x_1) \) and \( f'(x_1) \). Now, we have another tangent line.
- We keep repeating the process until \( f'(x_i) \approx f'(x_{i+1}) \). That’s a local minimum.
Figure 3.12: A continuous, differentiable function

Figure 3.13: $f(x)$ with $x_0$, $x_1$, and tangent lines

Figure 3.13 shows $f(x)$ with points $x_0$, $x_1$, and their tangent lines. Let’s look at another example (an ellipse)

$$f(x, y) = \frac{x^2}{4} + \frac{y^2}{9} + 15$$

We’ll pick a pair of points $(x_0, y_0)$, and another pair $(x_1, y_1)$. Expanding the derivative as a Taylor Polynomial gives

$$f(x, y) = f(x_0, y_0) + \frac{\partial f}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0)(y - y_0) + \Theta((x - x_0)^2 + (y - y_0)^2)$$

The growth is

$$f(x, y) - f(x_0, y_0) = \left(\frac{\partial f}{\partial x}(x_0, y_0)\frac{\partial f}{\partial y}(x_0, y_0)\right) \cdot (x - x_0, y - y_0) + R((x - x_0)^2 + (y - y_0)^2)$$

($R$ is a remainder that tends towards zero.)

The gradient of $f$ is

$$\text{grad } f_{(x_0, y_0)} = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)_{(x_0, y_0)}$$

So,

$$\Delta f = \text{grad } f_{(x_0, y_0)} \cdot (x - x_0, y - y_0) + R$$

For functions of one argument, the gradient is the derivative.

Gradient descent gives approximate solutions, but they tend to work pretty well.

\[1\] Hopefully I’ve gotten this right
3.8.3 Finding Probabilities of a Bayesian Network

We start with a data set $D$ and a hypothesis $h$. $h$ is the totality of all coefficients of the network.

We want to maximize $P(D \land h)$, where

$$P(D \land h) = \prod_{d \in D} P(D = d|h)$$

Assume our network consists of nodes $Y_i$ whose parents are nodes $U_i$, as shown in Figure 3.14.

![Figure 3.14: Node $Y_i$ with parents $U_i$](image)

We also have probabilities

$$P(Y_i = y_{ij}|U_i = u_{ik})$$

Above, note that $U_i$ really represents a tuple of parents. (We could write them out explicitly, but the notation is cleaner if we just treat $U_i$ like a tuple). $Y_i$ represents a single node.

The probabilities are represented as a contingency table

<table>
<thead>
<tr>
<th>$U_i = u_{ik}$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y = y_j$</td>
<td>$w_{ijk}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

Maximizing $P(D|h)$ is equivalent to maximizing $\ln P(D|h)$. Instead of multiplying probabilities, we add logarithms.

$$\ln P(D|h) = \ln \prod_{d \in D} P(d|h)$$

$$= \sum_{d \in D} \ln P(d|h)$$

To find the gradients, we’ll need to compute

$$\frac{\partial \ln P(D|h)}{\partial w_{ijk}}$$

where $w_{ijk} = P(Y_i = y_{ij}|U_i = u_{ik})$

$$\frac{\partial P(D|h)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P(D|h)} \frac{\partial P(d|h)}{\partial w_{ijk}}$$

We’ll continue this in the next class.
3.8.4 Logistics

- hw3 is due on 4/24/2008
- We will have a test, either during the last class in April, or the first class in May
- Shortly after 4/24/2008, the final project will be assigned.
3.9 Lecture – 4/14/2008

3.9.1 Weight Tables and Bayesian Networks

A Bayesian network is a directed acyclic graph whose nodes correspond to random variables.

For a moment, let’s make the simplifying assumption that all variables are binary. We have a node $X_i$ whose parents are $U_1, \ldots, U_n$.

Each cell in $X_i$’s probability is

$$w_{ijk} = P(X_i = x_{ij} | U_i = u_{ik} \land w)$$

where $w$ represents the other weights of the network.

If $X_i$ has $r_i$ parents, then $X_i$’s table of probabilities will have $2^{1+r_i}$ elements. This comes from $r_i$ parents (each of which represents a binary variable), and the binary value of $X_i$ itself.

Therefore, the number of probabilities in a Bayesian network may be exponential in the number of nodes. For a “real” network, it’s practically impossible to solve for all of these weights directly.

Suppose our data set is $D = \{d_1, \ldots, d_m\}$. Assume that the data elements appear randomly and independently, such that

$$P(D|w) = \prod_{l=1}^{m} P(d_l|w)$$

We will try to find weights that maximize $P(D|w)$ locally.

However, since we’ll be working with very small numbers, it’s often better to work with logarithms instead:

$$\ln P(D|w) = \sum_{l=1}^{m} \ln P(d_l|w)$$

We’ll use a method called *gradient ascent*. This involves finding the partial derivative:

$$\frac{\partial}{\partial w_{ijk}} \ln P(D|w)$$

Let’s work with Equation (3.25) a little.

$$\frac{\partial}{\partial w_{ijk}} \ln P(D|w) = \sum_{l=1}^{m} \frac{\partial}{\partial w_{ijk}} \ln P(d_l|w)$$

$$= \sum_{l=1}^{m} \frac{1}{P(d_l|w)} \frac{\partial P(d_l|w)}{\partial w_{ijk}}$$

Note that $w_{ijk} = P(x_{ij} | w_{ij} \land w)$.

Next, let’s work with the term in isolation.
\[
\frac{\partial P(d_i|w)}{\partial w_{ijk}} = \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P(d_i|w) \cdot P(x_{ij'} \land u_{ik'} \land w) \cdot P(x_{ij'} \land u_{ik'}|w)
\]

\[
= \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P(d_i|x_{ij} \land u_{ij} \land w) \cdot P(x_{ij} \land u_{ik}|w) \quad \text{chain rule for conditional probability}
\]

\[
= \frac{\partial}{\partial w_{ijk}} \left( P(d_i|x_{ij} \land u_{ij} \land w) \cdot P(x_{ij} \land u_{ik}|w) \right)
\]

\[
= \frac{\partial}{\partial w_{ijk}} P(d_i|x_{ij} \land u_{ij} \land w) \cdot \frac{P(x_{ij} \land u_{ik} \land w)}{P(w)} \quad \text{def. of conditional probability}
\]

\[
= \frac{\partial}{\partial w_{ijk}} P(d_i|x_{ij} \land u_{ij} \land w) \cdot \frac{P(u_{ik} \land w)}{P(w)} \cdot \frac{w_{ijk} P(w)}{P(w)} \quad \text{(where did } w_{ijk} \text{ come from?)}
\]

Next, let’s combine the two groups of equations:

\[
\sum_{l=1}^{m} 1 \cdot \frac{\partial P(d_i|w)}{\partial w_{ijk}} = \sum_{l=1}^{m} 1 \cdot P(d_i|x_{ij} \land u_{ik} \land w) \cdot \frac{P(u_{ik} \land w)}{P(w)} \cdot \frac{w_{ijk} P(w)}{P(w)} \quad \text{From earlier}
\]

\[
= \sum_{l=1}^{m} 1 \cdot P(d_l|x_{ij} \land u_{ik} \land w) \cdot \frac{P(u_{ik} \land w)}{P(w)} \cdot \frac{P(u_{ik} \land w)}{P(w)} \quad \text{(where did } P(w) \text{ go?)}
\]

Therefore,

\[
\frac{\partial}{\partial w_{ijk}} \ln P(D|w) = \sum_{l=1}^{m} \frac{P(x_{ij} \land u_{ik}|d_l \land w)}{w_{ijk}} \quad \text{(3.26)}
\]

For gradient ascent, we use

\[
w_{ijk} \leftarrow w_{ijk} + \eta \sum_{l=1}^{m} \frac{P(x_{ij} \land u_{ik}|d_l \land w)}{w_{ijk}} \quad \text{(3.27)}
\]

The probabilities of a Bayesian network are computed iteratively. They will converge to a local maximum. Bayesian Networks can give better results than Naïve Bayes classifiers. However, it’s usually not worth the trouble.

**Further Reading on Bayesian Networks**

There is a paper by Heckerman which gives a very good tutorial of Bayesian Networks. It appears to be this document: [ftp://ftp.research.microsoft.com/pub/tr/tr-95-06.pdf](ftp://ftp.research.microsoft.com/pub/tr/tr-95-06.pdf)

A 1995 Book by Cowell (pub. Springer) is also very good.
3.9.2 Neural Networks

Where Bayesian Networks deal with discrete values, Neural Networks can handle data in \( \mathbb{R}^n \).

Let’s start by considering a simple example.

![Figure 3.15: Height vs Weight](image)

Figure 3.15 show a scatterplot of height vs. weight. Suppose we are trying to determine which subjects are obese. The simplest form of classification is to draw a line, \( h = aw + b \). Subjects above the line are classified as obese, while subjects below the line are not.

This method of linear separation is simple, but it does not always work. Consider the following: if you have three points, you can form all possible types of classifications by drawing a line. However, this does not hold for four points.

Consider the function \( f(x_1, x_2) = x_1 \oplus x_2 \) for \( x_1, x_2 \in \{0, 1\} \). If \( f(x_1, x_2) = 1 \) might be categorized as positive examples, while \( f(x_1, x_2) = 0 \) might be categorized as negative example. This is shown in Figure 3.16.

In Figure 3.16 there’s no way to divide the (-) and (+) points using a single line.

The Vapnik-Chervanenkis dimension of classification describes this metric. We have \( \text{VCdimension(linear)} = 3 \).

A **perceptron** is the simplest piece that goes into a neural network. Neural networks are models of neuron. Figure 3.17 shows a (crude) drawing of a pair of neurons.

Neurons are cells. Thick strands, called *axons* extend from the nucleus. Thin strands, called *dendrites* extend from the axons. The space between dendrites is called a *synapse*.

Neurons fire if the sum of their input signal exceeds a certain threshold, \( \lambda \):

- if \( w_1 x_1 + \ldots + w_n x_n \geq \lambda \), the neuron fires.
- if \( w_1 x_1 + \ldots + w_n x_n < \lambda \), the neuron does not fire.
Neural networks try to emulate this behavior.

In general, a neural network has three layers:

1. Input Units
2. A hidden layer
3. Output Units

Figure 3.18 illustrates how these layers are connected.

In Figure 3.18, the $x_i$'s represent different input nodes. Each input node has a weight, $w_i$.

The input layer feeds into the hidden layer. In the hidden layer, input nodes and weights are summed together and fed into a function $f$. The different hidden layer nodes pass their signals to the output nodes.

The idea behind the hidden layer:

$$f(x) = \begin{cases} 
1 & \text{if } \mathbf{w} \times \mathbf{x} > \lambda \\
-1 & \text{if } \mathbf{w} \times \mathbf{x} \leq \lambda 
\end{cases}$$

or equivalently,

$$f(x_1, \ldots, x_n) = \text{sign}(\mathbf{w} - \mathbf{x} - \lambda)$$

$$\text{sign}(z) = \begin{cases} 
1 & \text{if } z > 0 \\
-1 & \text{if } z \leq 0 
\end{cases}$$

$\text{sign}$ is not used very often. It is not continuous, and not differentiable. $\text{sign}$ is shown in Figure 3.19.

A more common function is the sigmoid function:

$$h(z) = \frac{1}{1 + e^{-z}} \quad (3.28)$$
Figure 3.19: Graph of $\text{sign}(z)$

Figure 3.20 shows a graph of Equation (3.28).

Figure 3.20: Sigmoid function $h(z)$
3.10 Some Notes on Derivatives – 4/16/2008

\[
\frac{d}{dx} x^a = ax^{a-1} \quad \text{power rule}
\]

\[
\frac{d}{dx} (f(x) \cdot g(x)) = f'(x)g(x) + f(x)g'(x) \quad \text{product rule}
\]

\[
\frac{d}{dx} \left( \frac{f(x)}{g(x)} \right) = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2} \quad \text{quotient rule}
\]

\[
\frac{d}{dx} e^x = e^x
\]

\[
\frac{d}{dx} \ln x = \frac{1}{x}
\]

\[
\frac{d}{dx} \log_a x = \frac{1}{\ln a \cdot x}
\]

\[
\frac{d}{dx} a^x = a^x \ln a
\]

\[
\frac{d}{dx} (g(x)^k) = kg(x)^{k-1} \cdot \frac{d}{dx} g(x)
\]

If \( y = f(u) \) and \( u = g(x) \), then \( \frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \quad \text{chain rule} \)
3.11 Lecture – 4/16/2008

3.11.1 Neural Networks

A perceptron is a device with a set of inputs \( x_1, \ldots, x_n \), a set of weights \( w_1, \ldots w_n \) (one for each input), a summation device, and a threshold function.

Figure 3.21 shows a perceptron. Perceptrons were predecessors of neural networks.

```
\[
\begin{array}{c}
\sum \\
\end{array}
\]
```

Figure 3.21: Diagram of a perceptron

“net” denotes the output of the summation function

\[
\text{net} = w \cdot x = \sum (x_i w_i)
\]

The original output function was

\[
o(x_1, \ldots, x_n) = \begin{cases} 
1 & \text{if } w \cdot x \geq 0 \\
-1 & \text{if } w \cdot x < 0
\end{cases}
\]

A later variation added a parameter \( w_0 \), as illustrated in Figure 3.22.

```
\[
\begin{array}{c}
\sum \\
w_0
\end{array}
\]
```

Figure 3.22: Diagram of a perceptron with bias

For Figure 3.22 the output function is

\[
o(x_1, \ldots, x_n) = \begin{cases} 
1 & \text{if } w \cdot x \geq w_0 \\
-1 & \text{if } w \cdot x < w_0
\end{cases}
\]

The signum function mimics the firing of a neuron (the neuron fires or it doesn’t). However, signum has some disadvantages: it is neither continuous nor differentiable.
But, we can replace signum with functions that are continuous and differentiable. Some common example:

\[ f(x) = \frac{1}{1 + e^{-x}} \]  
sigmoid or “squashing” function

\[ f(x) = \frac{1}{1 + e^{-kx}} \]  
sigmoid with an extra parameter

\[ \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \]  
hyperbolic tangent

A plot of \( \tanh(x) \) is shown in Figure 3.23.

\[ \frac{-10}{-5} \quad 0 \quad 5 \quad 10 \]  
\[ -1.0 \quad -0.5 \quad 0.0 \quad 0.5 \quad 1.0 \]  
\[ \text{tanh}(x) \]  
\[ x \]  
Figure 3.23: Plot of \( \tanh(x) \)

Our discussion of neural networks assumes the sigmoid function.

One trains a neural network as follows:

- We supply a set of training data in the form \( (x_1, \ldots, x_n, t) \), where \( t \) is the target output.
- We measure the error at the outputs.
- We adjust the parameters of the network and try again.

Neural networks have three layers: input, hidden and output.
Figure 3.24 shows a diagram of a Neural Network.

In Figure 3.24:
- $x_1, \ldots, x_n$ are inputs.
- $j$ is a node in the hidden layer.
- $k$ is a node in the output layer.
- $w_{ji}$ is the weight between input node $x_i$ and hidden node $j$. When giving weights, the first subscript denotes the destination, and the second subscript denotes the source.
- $w_{jk}$ is the weight between internal node $j$ and output node $k$.
- $o_j$ denotes the output of node $j$. Similarly, $o_k$ denotes the output of node $k$.
- $net_j$ and $net_k$ denote the summation functions of nodes $j$ and $k$ respectively.

For training, we define the error $E$ as follows:

$$E = \frac{1}{2} \sum_k (t_k - o_k)^2$$

Our goal is to minimize $E$.

How many parameters are we working with. Let’s assume a small neural network with 10 inputs, 4 hidden nodes, and 6 output nodes. This give us $(10 \times 4) + (4 \times 6) = 40 + 24 = 64$ different parameters.

We will minimize $E$ by seeking local minimums, using gradient descent:

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E}{\partial w_{ji}}$$

From the input to the hidden layer

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E}{\partial net_k} w_i$$

since $net_j$ depends only upon the weights and inputs.
If \( j \) is an output unit (i.e., a \( k \) instead of a \( j \)), we have

\[
\frac{\partial E}{\partial \text{net}_j} = \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial \text{net}_k}
\]

\[
= -(t_k - o_k) \cdot o_k(1 - o_k)
\]

Where does \( o_k(1 - o_k) \) come from? \( o_k \) comes from node \( k \)’s output function:

\[
o_k = \frac{1}{1 + e^{-\text{net}_k}}
\]

Taking the derivative of this:

\[
f(x) = \frac{1}{1 + e^{-x}}
\]

\[
f'(x) = \frac{-1}{(1 + e^{-x})^2} \cdot e^{-x}
\]

\[
= \frac{e^{-x}}{(1 + e^{-x})^2}
\]

\[
= \frac{1}{1 + e^{-x} - \frac{1}{(1 + e^{-x})^2}}
\]

\[
= f(x) - f(x)^2
\]

\[
= f(x)(1 - f(x))
\]

Next, let’s look at the hidden layer

Let \( DS(j) \) denote the set of nodes that are “downstream” from node \( j \)

\[
\frac{\partial E}{\partial \text{net}_j} = \sum_{k \in DS(j)} \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial \text{net}_j}
\]

\[
= - \sum_{k \in DS(j)} (t_k - o_k) \cdot \frac{\partial o_k}{\partial \text{net}_j}
\]

Let’s look at \( \frac{\partial o_k}{\partial \text{net}_j} \)

\[
\frac{\partial o_k}{\partial \text{net}_j} = \frac{\partial o_k}{\partial \text{net}_k} \cdot \frac{\partial \text{net}_k}{\partial \text{net}_j}
\]

\[
= o_k(1 - o_k)
\]

\[
\frac{\partial \text{net}_k}{\partial \text{net}_j} = \frac{\partial \text{net}_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial \text{net}_j}
\]

\[
= w_{kj} \cdot o_j(1 - o_j)
\]

So,

\[
\frac{\partial o_k}{\partial \text{net}_j} = o_k(1 - o_k) \cdot w_{kj} \cdot o_j(1 - o_j)
\]

We computed the partial derivatives from the output layer backwards. That’s why the approach is called “back propagation”. 
The important equation to remember is
\[
\frac{\partial E}{\partial \text{net}_j} = \sum_{k \in D_S(j)} (t_k - o_k) \cdot o_k (1 - o_k) \cdot w_{kj} \cdot o_j (1 - o_j)
\]

To train a neural network, we do the following:

- We apply training data to the inputs
- We observe the outputs
- We calculate the partial derivatives
- We propagate the updates backwards, from output to input
- We repeat the process, until it converges.

Training is usually a time consuming process – there could be hundreds of inputs and dozens of hidden nodes.

There is no real relationship among weights. For example, they don’t form a probability distribution. The weights become what the training process makes them become.

3.11.2 To-Do

- Take a look at R’s neural network package (named “neural”)
- Look over vector support machines. We start discussing those in our next lecture.

3.12.1 Instance-Based Classification

We’ve studied several techniques for classification. Up to now, they’ve had the following things in common

- We start with a training set.
- We build a model from the training set.
- We use the model to classify data

Most of the work happens before we use the model to answer queries. In other words, the work goes into building the model.

By contrast, an instance-based classifier has the following traits:

- We start with a training set
- No model is built explicitly. The classifier simply stores the training set.
- Classification is done when queries are presented
- Most of the work is done during the query phase

Instance-based classifiers are also referred to as lazy classifiers, since they postpone work to the last possible moment.

3.12.2 Nearest-Neighbor Classifiers

Nearest Neighbor Classifiers are a type of instance-based classifier. The more general case are $k$-nearest neighbor classifiers, or $k$-NN for short.

$k$-NN classifiers work on data in $\mathbb{R}^n$.

In $\mathbb{R}^n$ we work with distances between vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$. Euclidean distance is the most common distance measure

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (3.29)$$

$k$-NN assumes that points belong to the same class if they are “close” together.

$k$-NN classifiers can be thought of as a function $f: D \rightarrow \{v_1, \ldots, v_s\}$, such that $f(d) = v_i$ if $v_i$ is the class of $d$.

A training set $T$ is a set of data containing (1) a set of tuples, and (2) the class associated with each tuple.

$$T = \{(x^k, v_k) \mid v_k \text{ is the class of } x^k\} \quad (3.30)$$

The general technique goes like this:

- We have a query $q$ and a number $k$.
- We look for the $k$ points of the training set that are nearest to $q$.
- We assign $q$ the predominating class of the $k$ nearest neighbors.

In other words, if $\delta(a, b)$ is

$$\delta(a, b) = \begin{cases} 
1 & \text{if } a = b \\
0 & \text{otherwise} 
\end{cases} \quad (3.31)$$
Then $k$-NN is

$$\text{arg max}_v \sum_{j=1}^k \delta(v, f(x^j))$$

(3.32)

### 3.12.3 Good and Bad Points of $k$-NN

One issue with $k$-NN is the way that $k$ influences the outcome. Consider Figure 3.25

![Figure 3.25: Sample $k$-NN Query](image)

Figure 3.25 contains a query $q$, and training set points. Two of the training set points have class 0 and three have class 1.

If we have $k = 1$, then $q$ will be assigned a class of 0, since the 1-nearest neighbor has class 0. However, if we have $k = 5$, then $q$ will be given a class of 1, since there are three 1’s but only two 0’s.

Usually $k = 1$ gives good results.

$k$-NN is not sensitive to outliers and noise.

$k$-NN does not perform well if the number of dimensions is too high. Suppose our training set has 50 attributes $A_1, \ldots, A_{50}$, along with a class attribute.

It’s quite possible that the class is predominantly determined by 2–3 attributes, while the other 47–48 have little impact. $k$-NN will treat all attributes as having equal influence.

In other words, objects of the same class can be very far apart geometrically.

Reducing the number of attributes would help . . . but which ones can we eliminate? (This is called the attribute selection problem).

We could try to bias (weight) the attributes. Bias gives us a distance measure like this:

$$d(x, y, a) = \sqrt{\sum_{i=1}^n a_i(x_i - y_i)^2}$$

(3.33)

Above, $a$ is a vector of weights, such that $\sum_{i=1}^n a_i = 1$.

### 3.12.4 The Dimensionality Curse

In this section, we’ll look at some of the issues inherent in dealing with data from $\mathbb{R}^n$ for large $n$. 
Squares in $\mathbb{R}^n$

Consider a unit square:

- In $\mathbb{R}^2$, this is the familiar square. Each side has length 1.
- In $\mathbb{R}^3$, this is a cube, where each side has length 1.
- In $\mathbb{R}^1$, this is a line, with length 1.

In general, a unit cube of dimension $n$ is

$$Q_n(1) = [0, 1] \times [0, 1] \times \ldots \times [0, 1] \quad (n \text{ times})$$

(3.34)

The center of such a cube is

$$(0.5, 0.5, \ldots, 0.5) \quad \text{(again, } n \text{ times)}$$

Next, let's take a look at pairs of opposing corners – the two vertices that are furthest apart.

- In $\mathbb{R}^2$, the diagonal of a unit square has length $\sqrt{2}$.
- In $\mathbb{R}^3$, the longest diagonal of a unit cube has length $\sqrt{3}$. (We take the $\sqrt{2}$ diagonal from $\mathbb{R}^2$ and go “up” in one dimension).
- In $\mathbb{R}^n$, the longest diagonal has length $\sqrt{n}$.

In $\mathbb{R}^n$, the distance from the center to a face of the square is 0.5. In other words, the faces stay close to the center, but the corners become very elongated ($\frac{\sqrt{n}}{2}$ from the center).

Sphere’s in $\mathbb{R}^n$

In $\mathbb{R}^2$ a sphere is a circle. The circle is all points within distance $r$ of the center. The “volume” of such a sphere is $\pi r^2$.

In $\mathbb{R}^3$, we have the familiar sphere, whose volume is $\frac{4\pi r^3}{3}$.

In $n$ dimensions, the volume of a sphere is

$$V_s = \frac{\pi^{\frac{n}{2}} \cdot r^n}{\Gamma\left(\frac{n}{2} + 1\right)}$$

Where

$$\Gamma(a) = \int_0^{\infty} x^{a-1} e^{-x} dx$$

$\Gamma$ has the property that $\Gamma(a + 1) = a \cdot \Gamma(a)$.

$$\begin{align*}
\Gamma(1) &= 1 \\
\Gamma(2) &= 1 \\
\Gamma(3) &= 2 \\
\Gamma(4) &= 6
\end{align*}$$

$\Gamma(n)$ is undefined for $n \leq 0$

$\Gamma(1/2) = \sqrt{\pi}$
Or more generally,
\[ \Gamma(n + 1) = n! \]

\( \Gamma \) generalizes the factorial function.

Suppose we have two concentric spheres, one with radius \( r \), and one with radius \( r_1 \), such that \( r = r_1 + \delta \). This is shown in Figure 3.26.

What portion of the outer sphere lies outside the inner sphere? Let \( V \) be the volume of the outer sphere and \( v \) be the volume of the inner sphere. The difference is

\[
\frac{V - v}{V} = \frac{\pi \frac{n}{2} r^n}{\Gamma\left(\frac{n}{2} + 1\right)} - \frac{\pi \frac{n}{2} r_1^n}{\Gamma\left(\frac{n}{2} + 1\right)} \\
= \frac{r^n - r_1^n}{r^n} \\
= 1 - \left(\frac{r_1}{r}\right)^n \\
= 1 - \left(\frac{r - \delta}{r}\right)^n \\
= 1 - \left(1 - \frac{\delta}{r}\right)^n
\]

As \( n \) grows, \( \frac{V - v}{V} \) goes to 1. This means that most of the volume of the outer sphere is concentrated near the outer edge.

**Another Cube Example**

Consider the two squares in Figure 3.27.

In Figure 3.27 we have a square with a diagonal. A point on the diagonal is used to define a smaller square. What proportion of the larger square’s volume lies outside the inner square?

Assuming in unit square in \( \mathbb{R}^2 \), the difference is

\[
\frac{p^2}{(1 - p)^2}
\]
In three dimensions, this ratio is
\[ \frac{p^3}{(1-p)^3} \]
In \( n \) dimensions,
\[ \frac{p^n}{(1-p)^n} \]
Suppose we move \( p \) to \( p + \alpha \). And, let’s choose \( \alpha \) such that ratio doubles
\[ 2 \cdot \frac{p^n}{(1-p)^n} = \frac{(p + \alpha)^n}{(1 - p - \alpha)^n} \]
We have
\[ \alpha = \frac{p(1-p) \sqrt{2} \cdot p(1-p)}{1 - p + p \sqrt{2}} \]
\[ = \frac{p(1-p)(\sqrt{2} - 1)}{\approx 1} \]
Therefore, a tiny change in \( \alpha \) produces a very large change in the volume ratio.

**The Moral of this Story**

The moral of this story, as \( n \) grows, \( k \)-NN tends to become extremely unreliable. Your query point will tend to be near to everything, or near to nothing.

\( n = 13 \) is about as far as you want to push \( k \)-NN.

**3.12.5 Logistics**

- We will have a take home final rather than an in-class exam. The final will be distributed on 5/12/2008, and due on 5/14/2008.
- Our final project will involve doing analysis on a large data set, then writing an essay of the results. We’ll probably use the Mushroom data set (but not necessarily). The final project will be posted within the next few days.
### 3.13 Lecture – 4/28/2008

#### 3.13.1 Evaluating Classifiers

We’ve looked at several different types classifiers. How can we determine when one classifier is better than another?

Suppose we are working with a binary class, for example:

- Does a patient have disease \( x \), or not?
- Is the guy walking through the airport a terrorist, or not?

In both of these cases, the classifier has a positive class and a negative class.

Let \( P \) be the number of (real) positive examples; let \( N \) be the number of (real) negative examples; and, let \( n = N + P \) be the number of subjects to be classified.

If we compare the actual classes to the output of our classifier, we get a matrix like this:

<table>
<thead>
<tr>
<th>Classifier Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

Above,

- TP denotes a “true positive”,
- FN denotes a “false negative”,
- FP denotes a “false positive”, and
- TN denotes a “true negative”

The **accuracy** of classifier \( M \) is the number of correct responses.

\[
\text{accuracy}(M) = \frac{TP + TN}{P + N} 
\]

The **recall** (also known as “true positive rate”, or “sensitivity”) is

\[
\text{recall}(M) = \frac{TP}{P} = \frac{TP}{TP + FN} 
\]

The **specificity**, or “true negative rate” is

\[
\text{specificity}(M) = \frac{TN}{N} = \frac{TN}{FP + TN} 
\]

The **precision** of the classifier is

\[
\text{precision}(M) = \frac{TP}{TP + FP} 
\]

Ideally, precision and recall should be close to one.

Some common rearrangements of accuracy:

\[
\text{accuracy}(M) = \frac{TP}{P + N} + \frac{TN}{P + N} \\
= \frac{TP}{P} \cdot \frac{P}{P + N} + \frac{TN}{N} \cdot \frac{N}{P + N} \\
= \text{recall}(M) \cdot \frac{P}{P + N} + \text{specificity}(M) \cdot \frac{N}{P + N} 
\]
Another common measure is the *F-measure*, which is the harmonic average of precision and recall.

\[
F(M) = \frac{2 \times \text{precision}(M) \times \text{recall}(M)}{\text{precision}(M) + \text{recall}(M)}
\]  

(3.39)

It’s desirable for \( F(M) \) to be as large as possible.

Why use the harmonic average, rather than the arithmetic or euclidean average? Given \( a \leq b \), we have

\[
a \leq \frac{2ab}{a+b} \leq \sqrt{ab} \leq \frac{a+b}{2} \leq b
\]

Thus, if the harmonic average is raised, the other averages are raised along with it.

Suppose we were to take several classifiers, and plot their true positive rates against their false negative rates. Figure 3.28 shows an example.

![Figure 3.28: Comparing Classifiers](image)

In Figure 3.28 we’ve normalized \( TP \) and \( FP \) by dividing them by \( P \).

The really lousy classifiers will lie on the diagonal line. On the diagonal, you’re no better off than flipping a coin.

Classifier \( M_1 \) is better than \( M_0 \). \( M_1 \) has a higher rate of true positives, and a lower rate of false positives. In general, moving “northwest” means the classifier is better.

How do \( M_2 \) and \( M_3 \) compare to \( M_0 \)? This is less cut and dry. \( M_2 \) has fewer false positives, but fewer true positives (a more conservative classifier). On the other hand \( M_3 \) has more true positives, but also more false positives. In this case, there’s no general answer. The choice depends on the cost of making mistakes.

Classifiers with high recall produce a small number of false negatives.

### 3.13.2 ROC Curves

If the output of a classifier is a probability, then there is another comparison technique we can use: ROC curves.

ROC stands for “receiver operating characteristic”. In Weka, they’re called “threshold graphs”.

Naïve Bayes classifiers are built around the formula

\[
P(C_+|x_1, \ldots, x_n) \geq \lambda
\]
A Bayes classifier makes a positive choice if the probability is $\geq \lambda$. The decision is based on the parameter $\lambda$.

The basic idea: we plot how the performance of the classifier varies for different values of $\lambda$.

Let’s do an example. Suppose we have 20 data instances. Half are classified as positive, and half are classified as negative.

| Instance | True Class | score $= P(C_+|x) \geq \lambda$ |
|----------|------------|-----------------------------------|
| 1        | P          | 0.900                             |
| 2        | P          | 0.800                             |
| 3        | N          | 0.700                             |
| 4        | P          | 0.600                             |
| 5        | P          | 0.550                             |
| 6        | P          | 0.540                             |
| 7        | N          | 0.530                             |
| 8        | N          | 0.520                             |
| 9        | P          | 0.510                             |
| 10       | N          | 0.505                             |
| 11       | P          | 0.490                             |
| 12       | N          | 0.390                             |
| 13       | P          | 0.390                             |
| 14       | N          | 0.370                             |
| 15       | N          | 0.360                             |
| 16       | N          | 0.350                             |
| 17       | P          | 0.340                             |
| 18       | N          | 0.330                             |
| 19       | P          | 0.300                             |
| 20       | N          | 0.100                             |

The classifier outputs “P” when the score is greater than $\lambda$.

Note that the scores are sorted in descending order. This makes it easy to see the effect of changing thresholds.

- For $\lambda = 0.9$, there is one true positive, and no false positives. Likewise, for $\lambda = 0.8$.
- For $\lambda = 0.7$, there are two true positives, and one false positive.

Figure 3.29 shows the ROC curve corresponding to this example. The curve was constructed like this:

- We start at $(0, 0)$.
- If the class for the current $\lambda$ is $P$, then the classifier has made a true positive judgement. We move up and plot the point for $\lambda$.
- If the class for the current $\lambda$ is $N$, then the classifier has made a false negative judgement. We move right and plot the point for $\lambda$.
- We repeat the prior two steps until we’re out of $\lambda$ values.

Figure 3.30 illustrates how ROC Curves vary according to the quality of the classifier. Curves near the diagonal are bad. Curves that are primarily in the northwest corner are good.

In general, the larger the area under the curve, the better the classifier.
Figure 3.29: ROC Curve for 20-element data set
Figure 3.30: ROC Curve vs. Classifier Quality
3.13.3 Ways of Improving Classifiers

Here we are concerned with improving the performance of a group of classifiers, also known as a “classifier ensemble”.

Boosting is a way to improve performance with small data sets – sets that are too small to use with 10x cross-validation. There are several variations of this technique, but the one that’s most popular is called “0.632-boosting”.

Suppose there are \( d \) elements in our training set. The probability of choosing an element is \( \frac{1}{d} \).

With boosting, we select a random element, and this random element becomes part of our training set. But we’ll also put the random element back into the data set (thereby allowing us to pick it again).

After selecting \( d \) random elements with repetition, we’ll have \( <d \) elements for our training set.

- \( \frac{1}{d} \) is the probability of picking a random element.
- \( 1 - \frac{1}{d} \) is the probability of not picking a random element.
- \( (1 - \frac{1}{d})^d \) is the probability of not a picking an element after \( d \) attempts.

As \( d \) tends to infinity

\[
\lim_{d \to \infty} \left(1 - \frac{1}{d}\right)^d = \frac{1}{e} = \frac{1}{2.718} \approx 0.368
\]  

(3.40)

Therefore, 0.368 of our samples are not picked, but \( 1 - 0.368 = 0.632 \) of our samples are picked. This is where the name “0.632 boosting” comes from.

The randomly-chosen 63.2% becomes our training set, and we do cross validation on the rest.

The accuracy of such a classifier will be

\[
\text{accuracy}(M) = 0.632 \times \text{accuracy developed in the 63.2%} + 0.368 \times \text{accuracy from the rest of the data set}
\]

Bagging and Adaboost

Bagging and Adaboost are two techniques based on boosting.

In bagging, classification is done by consensus. (The classifier ensemble takes a vote.)

In adaboost, the classification is sequential. The output of classifier \( n \) influences the decision of classifier \( n - 1 \).
3.14 Notes from Han, Chapters 6.12 – 6.15

3.14.1 Accuracy Measures

Accuracy measures the percentage of tuples that are correctly classified by a classifier. The error rate means the percentage of tuples that are incorrectly classified by a classifier. Sensitivity is the true positive recognition rate. Specificity is the true negative recognition rate.

What does one do if the class of a tuple cannot be uniquely determined from the data set (i.e., two tuples with the same values have different classes)? In this case, the classifier could return a probability distribution instead of a discrete answer.

3.14.2 Ways to Improve Accuracy

Holdout Method In the holdout method, 2/3’s of the data are used for training, and the remaining 1/3 is used to test and evaluate the classifier. Holdout is a pessimistic method, since the classifier is not trained on the full data set.

Cross Validation The data is partitioned on k folds of equal size. There are k training iterations, where 1/k of the data is reserved for testing.

Bootstrap Method The data is sampled d times with replacement, and the sampled data is used for training. The d samples will usually cover 62.8% of the data.

3.14.3 Ensemble Methods

Ensemble methods involve several classifiers. There are two common ensemble methods: bagging and boosting.

Bagging relies on the majority vote of a set of classifiers.

Boosting is an serial process, where data is analyzed by one classifier at a time. Each tuple is assigned a weight. After classifier $M_i$ is learned, the weights are adjusted, such that classifier $M_{i+1}$ “pays more attention” to the misclassified tuples. After training, boosting assigns a weight to each classifier, based on how well that classifier performed.

3.14.4 Model Selection

The $t$-test is a common method for determining whether one model is better than another.

The null hypothesis: if the two classifiers perform equally well, then the difference in their mean errors will be zero.

$$t = \frac{\text{err}(M_1) - \text{err}(M_2)}{\sqrt{\text{var}(M_1 - M_2)/k}}$$ (3.41)

For $k$-fold sampling, we use $k - 1$ degrees of freedom.
3.14.5 $k$-NN Classifiers

How can $k$-NN work for nominal data? Here’s one approach:

- If two attributes are the same, then the distance is zero.
- If two attributes are different, then the distance is one.
- If one or both attributes of a pair are missing, then the distance is one.

(This assumes distances are normalized to the range $[0,1]$).

How can we handle missing numeric data? Here’s one approach:

- If both numeric attributes are missing, the distance is one (max distance).
- If one attribute is missing, then the distance is $\max(1-v,v)$ where $v$ is the attribute that is present.
3.15 Lecture – 4/30/2008

3.15.1 Bagging

In bagging, we
- Generate (by bootstrapping) a multiplicity of classifiers.
- Present a query to each classifier.
- The class is decided by a majority vote.

The term “bagging” comes from “bootstrapping” and “aggregation”.

We start with a data set $D$. With bootstrapping, we build training sets $T_i$, and testing sets $U_i$. These in turn are used to learn classifiers $M_i$.

The $M_i$ classifiers are similar, but independent. One does not affect any of the others.

3.15.2 Adaboost

Adaboost uses a series of classifiers, where $M_i$ influences $M_{i+1}$. $M_{i+1}$ focuses on $M_i$’s mistakes.

If our data set $D$ has $|D| = d$, then the probability of choosing a random element is $1/d$. When going from $M_i$ to $M_{i+1}$, we renormalize the probabilities:

$$(p_1, p_2, \ldots, p_d) \Rightarrow \left( \frac{w_1 p_1}{\sum w_i}, \frac{w_2 p_2}{\sum w_i}, \ldots, \frac{w_d p_d}{\sum w_i} \right)$$

The purpose of the weights $w_i$ is to make the incorrectly classified elements more prominent. (To actually implement the weights, we can create copies of tuples, to mirror the weighting.)

The error rate of an Adaboost ensemble will be smaller than the error rate of any individual classifier in the ensemble.

Adaboost will work with any type of classifier.

Let’s use $\text{err}(M_i)$ to denote the error rate of $M_i$. The Adaboost algorithm is as follows.

1. set $i = 1$.
2. Extract from $D$ a set of tuples, using equal weightings. The extraction is done with bootstrapping.
3. Construct $M_i$.
4. if $\text{err}(M_i) > 0.5$; then
   (a) set the weights to $1/d$
   (b) if $i > k$, then stop. Else go back to step 2
5. if $\text{err}(M_i) \leq 0.5$; then
   (a) Take the tuples that $M_i$ classified correctly, and multiply their weights by $\frac{\text{err}(M_i)}{1-\text{err}(M_i)}$.
   (b) Resample $D$
   (c) increment $i$
   (d) Go back to step 3

Note that $0 \leq \text{err}(M_i) \leq 1$. We reject “bad” classifiers – those whose error rate is no better than random guessing. It’s not okay to invert the decision of a bad classifier. In most cases, the contingency matrix will not be symmetric. Better to just get rid of the bad ones.

How does the function $\frac{\text{err}(M_i)}{1-\text{err}(M_i)}$ behave? We can get an idea by looking at similar function $\frac{x}{1-x}$, which is shown in Figure 3.31.
Note what’s happening: if the error rate is high, the weight will be high (asymptotically close to 1). If the error rate is low, the weight will be low (tending towards zero). This places more weight on incorrectly classified tuples.

Once we’ve constructed our set of classifiers, \{M_1, \ldots, M_k\}, the next step is to “weight them”.

We have classes \{c_1, \ldots, c_l\}. With each class, we associate a variable \(v_i\).

To classify \(x\),

1. For each classifier \(M_i, 1 \leq i \leq k\)
   
   (a) We present \(x\) to classifier \(M_i\).
   
   (b) \(M_i\) chooses a class \(c_j\).
   
   (c) We set \(v_j = v_j + \log \frac{1 - \text{err}(M_i)}{\text{err}(M_i)}\).

2. We output the class that corresponds to the largest \(v\).

What does \(\log \frac{1 - \text{err}(M_i)}{\text{err}(M_i)}\) do? To get an idea, let’s look at a graph of \(\ln \frac{1-x}{x}\), which is shown in Figure 3.32. Classifiers with the lowest error rates have the largest contributions, and classifiers with the highest error rates have the smallest contributions.

The Adaboost algorithm was developed by Freund and Shapire.
Figure 3.32: Plot of $\ln \left(\frac{1-x}{x}\right)$
Part 4

Graph Mining

4.1 Lecture – 4/30/2008

In the remainder of this section, we’ll take a look at the general problem of graph mining and some of the challenges that it poses.

Organic molecules can be represented as graphs. For example, Figure 4.1 shows a benzene ring.

![Figure 4.1: A Benzene Ring](image)

Figure 4.1 shows 12 nodes. Six are labelled ‘H’ (hydrogen), and six are labelled ‘C’ (carbon). The edges also have labels, which denote single or double bonds.

Figure 4.2 shows another example, this time with Nucleotides. The challenge of graph mining – to notice that the structure on the right is a subset of the structure on the left.

4.1.1 Graph Isomorphism

Suppose we have two graphs: $G = (V, E)$ and $G' = (V', E')$. We also have a function $f: V \rightarrow V'$, such that $f$ is a bijection.

$f$ is an isomorphism if $(x, y) \in E \Leftrightarrow (f(x), f(y)) \in E'$.

If there is an isomorphism between $G = (V, E)$ and $G' = (V', E')$, then $G$ and $G'$ are identical from the point of view of graph theory.
However, determining whether two graphs are isomorphic is not an easy problem. If $|V| = |V'| = n$, then there can be up to $n!$ isomorphisms. Not a good problem for brute force.

For example, Figure 4.3 shows a pair of isomorphic graphs.

By contrast, Figure 4.4 shows a pair of graphs that are not isomorphic. The graphs in Figure 4.4 cannot be isomorphic, because the left graph has one connected component, and the right graph has two.

Isomorphism is a $\#P$-complete problem, which is more intractable than NP-complete.

### 4.1.2 Data Sets of Graphs

Recall our discussion of frequent item sets: we had a set of transactions, where each transaction was associated with a set of items.

In a graph database, we have a set of graphs. Each graph has an identifier, and some form of representation. Suppose we wanted to find frequent subgraphs. This kind of problem might arise if we were trying to identify a molecular structure in a database of chemical compounds.

With frequent item sets, we might notice that $a, b, c$ and $b, c, d$ were both frequent, and this would lead us to test whether $a, b, c, d$ was frequent.
This sort of thing is very difficult to do with graphs. We could start with small pieces – for example, subgraphs consisting of two nodes – and then try to build larger subgraphs from them. However, it's not always obvious how to assemble larger subgraphs from the smaller ones.

Consider the structures shown in Figure 4.5.

![Figure 4.5: Two similar subgraphs](image)

In figure 4.5, we can see that the H–C–H portions are common, but how would we go about putting them together?

In the next class, we’ll look at some graph mining algorithms.
4.2 Lecture – 5/5/2008

4.2.1 Finding Frequent Subgraphs

Finding frequent subgraphs is a computationally difficult problem, largely because finding isomorphic graphs is computationally difficult.

Our challenge here: given a set set which consists of graphs, we wish to find graphs $G$ that contain structures $g$ as sub-graphs.

There are a variety of graph mining algorithms. We will examine the $gSpan$ algorithm. $gSpan$ was developed by Yan and Han; it appeared in a book called *Graph Data Mining*.

Frequent item sets can be thought of as a trivial special case of graph mining. Transactions are graphs with isolated vertices. Each item is a vertex, and there are no edges in the graph.

The first question we’ll need to consider: how does one encode a graph so that the code can be used by a mining algorithm? We will encode graphs using DFS trees. More specifically, the DFS tree of a graph will be used to construct the code.

Given a graph $G$, $dfs(G)$ is the DFS tree for $G$.

For any given $G$, several $dfs(G)$’s are possible. Having several representations makes the problem a little more difficult.

4.2.2 DFS Trees

In constructing a DFS tree, we will visit nodes in a systematic way.

Before giving the DFS tree construction algorithm, let’s look at a basic adjacency list representation of a graph. Consider the undirected graph in Figure 4.6.

![Figure 4.6: An Undirected Graph](image-url)
An adjacency-list representation of Figure 4.6 would look like this:

\[
\begin{align*}
L[v_1] &= \{v_2, v_3\} \\
L[v_2] &= \{v_1, v_3, v_4, v_5\} \\
L[v_3] &= \{v_1, v_2, v_4\} \\
L[v_4] &= \{v_2, v_3\} \\
L[v_5] &= \{v_2\}
\end{align*}
\]

We could represent \(G\) by concatenating the edge lists. Of course, different concatenation orders would produce different representations, but they’d all be isomorphic. For a small graph like this, there would be \(5! = 120\) different representations. That’s too many to work with.

Instead, we will construct a set \(T\) of edges in the DFS tree. The construction algorithm is shown in Figure 4.7. In this algorithm, \(\text{SEARCH}(v)\) will be called no more than once per edge.

```plaintext
procedure dfs(G)
    \(T \rightarrow \emptyset\)
    for \(v \in V\); do
        mark \(v\) as “new”
    done
    while (there is a vertex \(v\) marked “new”); do
        \(\text{SEARCH}(v)\)
    done
end procedure

procedure SEARCH(v)
    mark \(v\) as “old”
    for \(w \in L[v]\); do
        if \(w\) is “new”; then
            add \((v, w)\) to \(T\)
            \(\text{SEARCH}(w)\)
        endif
    done
end procedure
```

Figure 4.7: \(\text{dfs}(G)\) construction

For example, if we applied this algorithm to the graph in Figure 4.6 starting from vertex \(v_1\), we’d have

\[
T = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_2, v_5)\}
\]

Notice that \(T\) contains a subset of the edges in \(G\). Figure 4.8 illustrates this. Edges in \(T\) are numbered, and drawn with solid lines. Edges \(\not\in T\) are drawn with dashed lines. The solid edges are called forward edges and the dashed edges are called back edges.

Forward edges are represented as a pair \((v_i, v_j)\) where \(i < j\). All edges in \(T\) are forward edges.

Back edges are represented as a pair \((v_i, v_j)\) where \(i > j\). The back edges of Figure 4.8 are \((v_4, v_2)\) and \((v_3, v_1)\).

The first vertex in \(\text{dfs}(G)\) is called the root.

The last vertex in \(\text{dfs}(G)\) is called the rightmost vertex.
Of course, any vertex can be chosen as the root. Let’s work through another example using Figure 4.9 starting with $v_0$. In Figure 4.9, the subscripts denote the order in which nodes were visited. In addition, both nodes and edges have labels.

Starting from $v_0$, $\text{dfs}(G)$ gives

$$\text{dfs}(G) = \{(v_0, v_1), (v_1, v_2), (v_2, v_3), (v_0, v_4)\}$$

The back edges are $\{(v_3, v_0), (v_2, v_0)\}$. Figure 4.10 shows the graph with dashed back-edges.

So far, we’ve been represented edges in the form $(v_i, v_j)$. However, we really want to use a 5-tuple, $(v_i, v_j, l(v_i), e_{ij}, l(v_j))$, where

- $v_i$ and $v_j$ are vertices. The subscripts $i$ and $j$ denote the order in which the vertices were visited.
- $l(v_i)$ and $l(v_j)$ are the labels of $v_i$ and $v_j$.
- $e_{ij}$ is the edge label for $(v_i, v_j)$.

The 5-tuple notation for Figure 4.10 is
Figure 4.10: Ending Graph, with Forward and Back edges

<table>
<thead>
<tr>
<th>edge</th>
<th>5-tuple notation</th>
<th>Forward/Back edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(v_0, v_1)$</td>
<td>$(0, 1, Y, a, X)$</td>
<td>Forward</td>
</tr>
<tr>
<td>$(v_1, v_2)$</td>
<td>$(1, 2, X, a, X)$</td>
<td>Forward</td>
</tr>
<tr>
<td>$(v_2, v_3)$</td>
<td>$(2, 3, X, c, Z)$</td>
<td>Forward</td>
</tr>
<tr>
<td>$(v_0, v_4)$</td>
<td>$(0, 4, Y, d, Z)$</td>
<td>Forward</td>
</tr>
<tr>
<td>$(v_3, v_0)$</td>
<td>$(3, 0, Z, b, Y)$</td>
<td>Back</td>
</tr>
<tr>
<td>$(v_2, v_0)$</td>
<td>$(2, 0, X, b, Y)$</td>
<td>Back</td>
</tr>
</tbody>
</table>

Of course, different choices of starting nodes give us different representations, but the number of representations here is far fewer than a brute force permutation of edges. Doing this much, the number of representations will be no more than the number of nodes.

Next, we’ll show how to turn this into a canonical representation, by taking the tree from $\text{dfs}(G)$ and turning it into a code.

**procedure** code_dfs_tree  
  repeat  
  add a new vertex  
  add a forward edge that connects the new vertex with the vertex in the previous code  
  add all backward edges that connect the new $v$ with the previous code  
  until (all edges are included)  
  end procedure

In pair notation, this gives

$$(v_0, v_1), (v_1, v_2), (v_2, v_0), (v_2, v_3), (v_3, v_0), (v_0, v_4)$$
The equivalent 5-tuple notation is

\[(0, 1, Y, a, X)\]
\[(1, 2, X, a, X)\]
\[(2, 0, X, b, Y)\]
\[(2, 3, X, c, Z)\]
\[(3, 0, Z, b, Y)\]
\[(0, 4, Y, d, Z)\]

The 5-tuples can be ordered lexicographically. Therefore, there is a total order on the set of 5-tuples, and a total order on the coded representations.

Relative to this total order, our canonical representation will be

\[\text{min dfs}(G)\]

In other words, compute all of the codes, sort them, and throw away everything but the first one. (The first code in sorted order is our canonical representation.)

**Theorem 4.2.2.1:** Two graphs \(G\) and \(G'\) are isomorphic iff \(\text{min dfs}(G) = \text{min dfs}(G')\).

The proof of this is very long. We won’t give it here.

A **Prüfer Code** is a similar technique for coding trees. Prüfer codes are simpler, but cannot handle arbitrary graphs.
4.3 Lecture – 5/7/2008

4.3.1 Social Networks

Social networks are usually represented by undirected graphs, having anywhere from thousands to millions of nodes. Social networks are interesting in many areas: sociology, economics, power distribution systems, and so forth.

4.3.2 Random Graphs

Given an undirected graph of \( n \) vertices, there are \( \binom{n}{2} \) possible edges. Suppose we add edges, choosing the edges randomly, such that each edge is added with probability \( p \). The end result is a random graph.

The degree of a vertex \( v \), \( \text{deg}(v) \) is the number of edges that are incident to \( v \).

Given a random graph, we can look at the number of edges with degree 1, degree 2, \ldots, degree \( k \). The probability that a node is connected to \( n \) vertices follows a poisson distribution.

\[
\frac{\lambda^k}{k!} e^{-\lambda} \\
\text{poisson distribution}
\]  

(4.1)

If you add enough edges, the degree of nodes will eventually (and asymptotically) become equalized.

One of the best references on this subject is the book Random Graphs by Bella Bollobas.

The US highway system could be regarded as a random graph.

4.3.3 Scale-Free Networks

People have conjectured that links on the internet would form a random graph. This conjecture turns out to be completely wrong. The internet’s link structure is far from uniform. Instead, a small number of nodes have very high degrees (“hubs”), and most of the nodes have very low degrees.

The degree of hubs is very high when compared to non-hubs. The distribution is more like the curve shown in Figure 4.11.

Figure 4.11: Distribution of Node Degrees on the Internet

Figure 4.11 shows the number of nodes with degree \( n \approx \frac{1}{n^\alpha} \) (for \( \alpha \approx 2 \)). The higher the degree, the fewer nodes there are with that degree.
This type of edge distribution appears in other areas. For example, the C. elegans worm contains about 5,000 proteins. These proteins are connected in a scale free network – a few of the proteins are highly connected, but most are not.

Citation links of scientific papers are another example of a scale-free network.

Suppose we wanted to classify a set of web pages. We’d need to look at the content of the pages, but that alone is not sufficient. We’d also need to look at the number of links to the pages, as well as the properties of the referring pages.

4.3.4 Modeling Scale Free Networks

How are scale-free networks created? A popular model is the *Forest Fire Model*. The Forest Fire Model works something like this:

- Lightning strikes a tree, and starts the tree on fire.
- The fire spreads to neighboring trees.
- The fire eventually burns out.

In terms of a mathematical construction:

- We start with some set of nodes.
- We add a new node $v$. $v$ randomly selects another node $w$, which serves as an *ambassador* to $v$.
- We add a set of edges from $v$ to neighbors of $w$. In a directed graph, more weight is given to edges that lead out of $w$.

Notice how this differs from the construction of a random graph. In a random graph, we had all of the nodes at the beginning. In the Forest Fire Model, we add nodes as the graph is created.

4.3.5 Centrality Problems

One of the most significant problems in social networks is determining how “central” a given node is. In other words, given a social network, we’d like to identify the hubs. If a hub is “compromised”, the entire network could be severely damaged.

Consider the graph in Figure 4.12

![Figure 4.12: A Star](image)

Figure 4.12: A Star

Figure 4.12 shows a star, whose center node is $x$. $x$ is clearly a hub.

- $x$ has the lowest average communications cost.
- $x$ controls access to the other nodes in the graph. In other words, to get from one outer node to another, you have to pass through $x$.

This is a simple case; we’d like a more general way to identify hubs.
A graph is fully defined by its incidence matrix. For example, the graph shown in Figure 4.13 is defined by the incidence matrix \( A \).

\[
A = \begin{pmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix}
\]

Figure 4.13: A Graph with Four Nodes

Let’s suppose we took \( A \) and raised it to the \( k \)-th power, forming \( A^k \). \( a_{ij}^k \) tells us the number of distinct paths of length \( k \) between \( v_i \) and \( v_j \). Put another way,

\[
A^{k+1} = A^k \times A
\]

\[
a_{ij}^{k+1} = \sum_{l=1}^{n} a_{il}^k \times a_{lj}^1
\]

What happens if you multiply \( A \) by the vector \( I \), where

\[
I = \begin{pmatrix}
1 \\
1 \\
1 \\
1
\end{pmatrix}
\]

This gives

\[
A \times I = \begin{pmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix} \times \begin{pmatrix}
1 \\
1 \\
1 \\
1
\end{pmatrix} = \begin{pmatrix}
2 \\
2 \\
3 \\
1
\end{pmatrix}
\]

\( A \times I \) gives us the degree of each \( v_i \). \( \text{deg}(v_i) = (AI)_i \) is a marginalization.

### 4.3.6 Measures of Centrality

**Closeness Centrality and Graph Centrality**

Let’s say we have a node \( v_i \) that is connected to some other nodes \( v_j \). The *closeness centrality* of \( v_i \), or \( C_{CL}(v_i) \) is

\[
C_{CL}(v_i) = \frac{1}{\sum \text{deg}(v_j)}
\]

**graph centrality**, or \( C_{GR}(v_i) \).

\[
C_{GR}(v_i) = \frac{1}{\max \text{deg}(v_j)}
\]
Betweenness Centrality

Suppose we have nodes $u$ and $v$. There can be several paths from $u$ to $v$, $u \sim v$, and one or more of these paths will be the shortest. Let us denote the length of the shortest path as $d(u, v)$. A few properties of $d(u, v)$:

$$d(u, u) = 0$$
$$d(u, v) = d(v, u)$$
$$d(u, w) \leq d(u, v) + d(v, w)$$

$d(u, v)$ is a metric.

For the special case where there is no path from $u$ to $v$, we say that $d(u, v) = \infty$.

Betweenness centrality, or $B(v_i)$ is a way to measure a node’s degree of “control”.

$$V(v_i) = \frac{\sum_{v_j \neq v_k, v_j \in \{V\} - v_i} d_{v_i}(v_j, v_k)}{\sum_{v_j \neq v_k, v_j \in \{V\} - v_i} d(v_j, v_k)}$$  \hspace{1cm} (4.4)

Equation (4.4) is a little messy, but here’s what it means. The numerator

$$\sum_{v_j \neq v_k, v_j \in \{V\} - v_i} d_{v_i}(v_j, v_k)$$

adds the lengths of the paths from distinct nodes $v_j$ to $v_k$, where the path passes through $v_i$. The denominator

$$\sum_{v_j \neq v_k, v_j \in \{V\} - v_i} d(v_j, v_k)$$

does the same thing, but it includes paths from $v_j$ to $v_k$ that do not pass through $v_i$.

Eigenvalue Centrality

Eigenvalue Centrality is another centrality measure. Google uses it for their PageRank algorithm.

Let $A$ be an $n \times n$ incidence matrix, and let $x$ be a vector of length $n$ that denotes the centrality of each vertex. How does $Ax$ look?

$$
\begin{pmatrix}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \ldots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{pmatrix}
$$

The centrality of a node $v$ is a function of the centrality of the nodes to which $v$ is connected.

$$x_i = \lambda \sum_{j=1}^{n} a_{ij} x_j$$
or equivalently

\[ x = \lambda A \cdot x \]
\[ Ax = \frac{1}{\lambda} x \]

\( \frac{1}{\lambda} \) is an eigenvalue, and \( x \) is an eigenvector.

We’ll look at this more during our next lecture.
4.4 Lecture – 5/12/2008

Most of the material in our last class dealt with undirected graphs. Today’s material will involve directed graphs.

4.4.1 Scientific Bibliographies

There’s a trend to look at co-citations in scientific papers. Suppose we have a set of papers \( x_1, \ldots, x_n \), and we want to determine whether two papers are closely related (in content).

Given two papers \( x_i \) and \( x_j \), we say that \( x_i \) and \( x_j \) are co-cited if there is a third paper \( x_l \) such that \( x_l \) cites both \( x_i \) and \( x_j \). Figure 4.14 shows a graphical representation of co-citation.

![Figure 4.14: Co-citation as a directed graph](image)

If two papers are co-cited many times, then they probably have something in common (similar content). Let’s start with an incidence matrix \( A = (a_{ij}) \) where

\[
a_{ij} = \begin{cases} 
1 & \text{if } x_i \text{ cites } x_j \\
0 & \text{otherwise}
\end{cases}
\]

Because we deal with directed edges, \( A \) will not be symmetric.

For a pair of papers, we can determine the number of co-citations as follows:

\[
\text{ct}(x_i, x_j) = \sum_{l=1}^{n} a_{li} \cdot a_{lj} = \sum_{l=1}^{n} (a_{il})^T \cdot a_{lj}
\]

In general, citations can be found via \( A^T \cdot A \).

\( \text{ct}(x_i, x_j) \) is a similarity (not a distance). We can apply clustering techniques using this similarity. It will be preferable to use a clustering algorithm that permits intersecting clusters.

This method has been shown to work well for a corpus of a few hundred papers.

4.4.2 Mining Pages on the Web

Directed graphs can also represent pages on the web. There are two popular algorithms for mining web pages.

**PageRank** Google’s algorithm, invented by Larry Page and Sergei Brin. PageRank is based on the “prestige” of a page.

**HITS** Invented by John Kleinberg. HITS is based on the notions of “authority” and “hub quality.”
Suppose we have a set of pages $x_1, \ldots, x_n$, and an incidence matrix $A = (a_{ij})$.

Let $p_o$ be a vector that denotes the probability of a user visiting any of the pages in $x_1, \ldots, x_n$. We denote the probability of visiting page $x_i$ as $p_o[x_i]$. Of course, $\sum_{i=1}^n p_o[x_i] = 1$.

Let’s consider one row of the matrix $A$

$$A = \begin{pmatrix} a_{i1} & a_{i2} & \ldots & a_{in} \end{pmatrix}$$

The sum

$$\sum_{j=1}^n a_{ij}$$

is the number of edges that depart from $x_i$ – the out-degree of $x_i$, or $\text{outdeg}(x_i)$.

We can form another matrix $E$, where each element is

$$e_{ij} = \frac{a_{ij}}{\text{outdeg}(x_i)}$$

Each row of $E$ will sum to 1. In other words, $E$ is a stochastic matrix.

Suppose we have two probability distributions $p$ and $p'$, where

$$\begin{pmatrix} p_1 & p_2 & \ldots & p_n \end{pmatrix} \cdot \begin{pmatrix} e_{11} & \cdots & e_{1n} \\ e_{i1} & \cdots & e_{in} \\ e_{n1} & \cdots & e_{nn} \end{pmatrix} = \begin{pmatrix} p'_1 & p'_2 & \ldots & p'_n \end{pmatrix}$$

$p'_i = \sum_{j=1}^n e_{ij} p_j$

If $p$ is a stochastic vector, then $p'$ will also be a stochastic vector.

$$p'_1 = \sum_{j=1}^n p_j e_{ji}$$

$$\sum_{i=1}^n p'_i = \sum_{i=1}^n \sum_{j=1}^n p_j e_{ji}$$

$$= \sum_{j=1}^n \sum_{i=1}^n p_j e_{ji}$$

$$= \sum_{j=1}^n p_j \sum_{i=1}^n e_{ji}$$

$$= \sum_{j=1}^n p_j$$

So,

$$p_1 = p_o E$$

$$p_2 = p_1 E = p_o E^2$$

$$p_n = p_o E^n$$

$$p_{n+t} = p_mE^n$$
When \( t \to \infty \), then \( p = p \cdot E^n \). \( p \) becomes an eigenvector.

### 4.4.3 The HITS Algorithm

HITS recognizes two types of papers:

- **authority papers**, which generate new ideas;
- **survey papers**, which reference authority papers

In reality, any paper is partially an authority paper, and partially a survey paper.

In terms of the web,

- There are authority pages. These have many in edges.
- There are hub pages. These have many out edges.

How would the HITS algorithm process a query \( q \)?

- \( q \) is presented to an information retrieval (IR) system. The IR system returns pages based on textual content only.
- The set of pages returned by the IR system forms the core set. The core set contains pages with textual components of \( q \).
- The core set is used to generate an extended core. If \( p \) is a page in the extended core, then there is a link from \( p \) to some page in the core set; or, some page in the core set has a link to \( p \).

Figure 4.15 shows the relationship between the Extended and Core sets.

![Figure 4.15: HITS Extended and Core Sets](image)

We attribute two characteristics to each page:

- \( a_i \) which is the authority degree of page \( x_i \), and
- \( h_i \) which is the hub degree of page \( x_i \).

Next, we apply an iterative algorithm to the extended set.

Let \( A = (a_{ij}) \) be an incidence matrix, which represents edges \( x_i \to x_j \). In this context, \( x_i \) is a hub, and \( x_j \) is an authority.

For each \( x_i \), we compute

\[
\begin{align*}
    a &= (a_1, \ldots, a_n) \\
    h &= (h_1, \ldots, h_n) \\
    a_j &= \sum_{i=1}^{n} h_i a_{ij} \quad \text{or, } a = h \cdot A \\
    h_i &= \sum_{j=1}^{n} a_j a_{ij} \quad \text{or, } h = a \cdot A^T
\end{align*}
\]
So,

\[
\mathbf{a} = \mathbf{a} \cdot (A^T \cdot A)
\]

\[
\mathbf{h} = \mathbf{h} \cdot (A \cdot A^T)
\]

\(\mathbf{h}\) and \(\mathbf{a}\) are eigenvectors.
GNU Free Documentation License

Version 1.2, November 2002

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, TeXinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose
the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent
copy of the Document, and likewise the network locations given in the Document for previous
versions it was based on. These may be placed in the “History” section. You may omit a network
location for a work that was published at least four years before the Document itself, or if the
original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section,
and preserve in the section all the substance and tone of each of the contributor acknowledgements
and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles.
Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified
Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any
Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary
Sections and contain no material copied from the Document, you may at your option designate some
or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the
Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your
Modified Version by various parties—for example, statements of peer review or that the text has been
approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as
a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of
Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any
one entity. If the Document already includes a cover text for the same cover, previously added by you
or by arrangement made by the same entity you are acting on behalf of, you may not add another; but
you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their
names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms
defined in section 4 above for modified versions, provided that you include in the combination all of the
Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections
of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections
may be replaced with a single copy. If there are multiple Invariant Sections with the same name but
different contents, make the title of each such section unique by adding at the end of it, in parentheses,
the name of the original author or publisher of that section if known, or else a unique number. Make
the same adjustment to the section titles in the list of Invariant Sections in the license notice of the
combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents,
forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and
any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS
You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.