1. What is similarity and dissimilarity? Explain similarity and dissimilarity measures between different attributes based on different types of attributes.

Distance or similarity measures are essential to solve many pattern recognition problems such as classification and clustering. Various distance/similarity measures are available in literature to compare two data distributions. As the names suggest, a similarity measures how close two distributions are. For multivariate data complex summary methods are developed to answer this question.

Similarity Measure

- Numerical measure of how alike two data objects are.
- Often falls between 0 (no similarity) and 1 (complete similarity).

Dissimilarity Measure

- Numerical measure of how different two data objects are.
- Range from 0 (objects are alike) to ∞ (objects are different).

Proximity refers to a similarity or dissimilarity.

Similarity/Dissimilarity for Simple Attributes

Here, *p* and *q* are the attribute values for two data objects.

Common Properties of Dissimilarity Measures

Distance, such as the Euclidean distance, is a dissimilarity measure and has some well known properties:

1. $d(p, q) \geq 0$ for all *p* and *q*, and $d(p, q) = 0$ if and only if $p = q$,
2. $d(p, q) = d(q,p)$ for all *p* and *q*,
3. $d(p, r) \leq d(p, q) + d(q, r)$ for all *p*, *q*, and r, where $d(p, q)$ is the distance (dissimilarity) between points (data objects), *p* and *q*.

A distance that satisfies these properties is called a **metric**.Following is a list of several common distance measures to compare multivariate data. We will assume that the attributes are all continuous.

Euclidean Distance

Assume that we have measurements x_{ik}, $i = 1, \ldots , N$, on variables $k = 1, \ldots , p$ (also called attributes).

The Euclidean distance between the *i*th and *j*th objects is

[Math Processing Error]

for every pair (i, j) of observations.

The weighted Euclidean distance is

[Math Processing Error]

If scales of the attributes differ substantially, standardization is necessary.

Minkowski Distance

The Minkowski distance is a generalization of the Euclidean distance.

With the measurement, x_{ik}, $i = 1, \dots, N$, $k = 1, \dots, p$, the Minkowski distance is

[Math Processing Error]

where $\lambda \geq 1$. It is also called the L_λ metric.

- $\lambda = 1$: L_1 metric, Manhattan or City-block distance.
- $\lambda = 2$: L_2 metric, Euclidean distance.
- $\lambda \rightarrow \infty$: L_∞ metric, Supremum distance.

[Math Processing Error]

Note that λ and p are two different parameters. Dimension of the data matrix remains finite.

Mahalanobis Distance

Let **X** be a N × p matrix. Then the i^th^ row of **X** is

[Math Processing Error]

The Mahalanobis distance is

[Math Processing Error]

where Σ is the p×p sample covariance matrix.

2. Why data preprocessing required in data mining? Explain various steps involved in data preprocessing

Why preprocessing ?

1. Real world data are generally
    - Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
    - Noisy: containing errors or outliers
    - Inconsistent: containing discrepancies in codes or names
2. Tasks in data preprocessing
    - Data cleaning: fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies.
    - Data integration: using multiple databases, data cubes, or files.
    - Data transformation: normalization and aggregation.
    - Data reduction: reducing the volume but producing the same or similar analytical results.
    - Data discretization: part of data reduction, replacing numerical attributes with nominal ones.

Data cleaning

1. Fill in missing values (attribute or class value):
    - Ignore the tuple: usually done when class label is missing.
    - Use the attribute mean (or majority nominal value) to fill in the missing value.
    - Use the attribute mean (or majority nominal value) for all samples belonging to the same class.
    - Predict the missing value by using a learning algorithm: consider the attribute with the missing value as a dependent (class) variable and run a learning algorithm (usually Bayes or decision tree) to predict the missing value.
2. Identify outliers and smooth out noisy data:
    - Binning
        - Sort the attribute values and partition them into bins (see "Unsupervised discretization" below);
        - Then smooth by bin means, bin median, or bin boundaries.
    - Clustering: group values in clusters and then detect and remove outliers (automatic or manual)
    - Regression: smooth by fitting the data into regression functions.
3. Correct inconsistent data: use domain knowledge or expert decision.

Data transformation

1. **Normalization:**
    - Scaling attribute values to fall within a specified range.
        - Example: to transform V in [min, max] to V' in [0,1], apply V'=(V-Min)/(Max-Min)
    - Scaling by using mean and standard deviation (useful when min and max are unknown or when there are outliers): V'=(V-Mean)/StDev
2. Aggregation: moving up in the concept hierarchy on numeric attributes.
3. Generalization: moving up in the concept hierarchy on nominal attributes.
4. Attribute construction: replacing or adding new attributes inferred by existing attributes.

Data reduction

1. Reducing the number of attributes
    - Data cube aggregation: applying roll-up, slice or dice operations.
    - Removing irrelevant attributes: attribute selection (filtering and wrapper methods), searching the attribute space (see Lecture 5: Attribute-oriented analysis).
    - Principle component analysis (numeric attributes only): searching for a lower dimensional space that can best represent the data..
2. Reducing the number of attribute values
    - Binning (histograms): reducing the number of attributes by grouping them into intervals (bins).
    - Clustering: grouping values in clusters.
    - Aggregation or generalization
3. Reducing the number of tuples
    - Sampling

Discretization and generating concept hierarchies

1. Unsupervised discretization - class variable is not used.
    - Equal-interval (equiwidth) binning: split the whole range of numbers in intervals with equal size.
    - Equal-frequency (equidepth) binning: use intervals containing equal number of values.
2. Supervised discretization - uses the values of the class variable.
    - Using class boundaries. Three steps:
        - Sort values.
        - Place breakpoints between values belonging to different classes.

        - If too many intervals, merge intervals with equal or similar class distributions.
    - Entropy (information)-based discretization. Example:
        - Information in a class distribution:
            - Denote a set of five values occurring in tuples belonging to two classes (+ and -) as [+,+,+,-,-]
            - That is, the first 3 belong to "+" tuples and the last 2 - to "-" tuples
            - Then, Info([+,+,+,-,-]) = -(3/5)*log(3/5)-(2/5)*log(2/5) (logs are base 2)
            - 3/5 and 2/5 are relative frequencies (probabilities)
            - Ignoring the order of the values, we can use the following notation: [3,2] meaning 3 values from one class and 2 - from the other.
            - Then, Info([3,2]) = -(3/5)*log(3/5)-(2/5)*log(2/5)
        - Information in a split (2/5 and 3/5 are weight coefficients):
            - Info([+,+],[+,-,-]) = (2/5)*Info([+,+]) + (3/5)*Info([+,-,-])
            - Or, Info([2,0],[1,2]) = (2/5)*Info([2,0]) + (3/5)*Info([1,2])
        - Method:
            - Sort the values;
            - Calculate information in all possible splits;
            - Choose the split that minimizes information;
            - Do not include breakpoints between values belonging to the same class (this will increase information);
            - Apply the same to the resulting intervals until some stopping criterion is satisfied.

3. Generating concept hierarchies: recursively applying partitioning or discretization methods.

3. What is Apriori algorithm? How its is used to find frequent item sets? Explain.

The *apriori principle* can reduce the number of itemsets we need to examine. Put simply, the apriori principle states that if an itemset is infrequent, then all its subsets must also be infrequent. This means that if {beer} was found to be infrequent, we can expect {beer, pizza} to be equally or even more infrequent. So in consolidating the list of popular itemsets, we need not consider {beer, pizza}, nor any other itemset configuration that contains beer.

```
$\text{Apriori}(T, \epsilon)$
    $L_1 \leftarrow \{\text{large } 1 - \text{itemsets}\}$
    $k \leftarrow 2$
    **while** $L_{k-1} \neq \emptyset$
        $C_k \leftarrow \{a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a\} - \{c \mid \{s \mid s \subseteq c \wedge |s| = k-1\} \nsubseteq L_{k-1}\}$
        **for** transactions $t \in T$
            $C_t \leftarrow \{c \mid c \in C_k \wedge c \subseteq t\}$
            **for** candidates $c \in C_t$
                $count[c] \leftarrow count[c] + 1$
        $L_k \leftarrow \{c \mid c \in C_k \wedge \ count[c] \geq \epsilon\}$
        $k \leftarrow k + 1$
    **return** $\bigcup_k L_k$
```

Finding itemsets with high support

Using the apriori principle, the number of itemsets that have to be examined can be pruned, and the list of popular itemsets can be obtained in these steps:
Step 0. Start with itemsets containing just a single item, such as {apple} and {pear}.
Step 1. Determine the support for itemsets. Keep the itemsets that meet your minimum support threshold, and remove itemsets that do not.
Step 2. Using the itemsets you have kept from Step 1, generate all the possible itemset configurations.
Step 3. Repeat Steps 1 & 2 until there are no more new itemsets.

We have seen how the apriori algorithm can be used to identify itemsets with high support. The same principle can also be used to identify item associations with high confidence or lift. Finding rules with high confidence or lift is less computationally taxing once high-support itemsets have been identified, because confidence and lift values are calculated using support values.
ake for example the task of finding high-confidence rules. If the rule

{beer, chips -> apple}

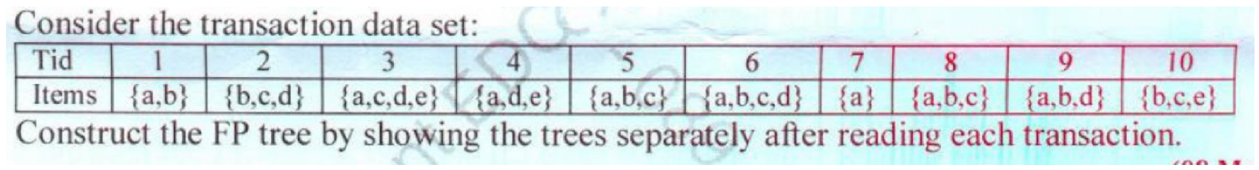
has low confidence, all other rules with the same constituent items and with apple on the right hand side would have low confidence too. Specifically, the rules

{beer -> apple, chips}
{chips -> apple, beer}

would have low confidence as well. As before, lower level candidate item rules can be pruned using the apriori algorithm, so that fewer candidate rules need to be examined.

4. Define the terms with proper example: Support and confidence

A consequent is an item that is found in combination with the antecedent. Association rules are created by analyzing data for frequent if/then patterns and using the criteria **support and confidence** to identify the most important relationships. **Support** is an indication of how frequently the items appear in the database.



Consider the transaction data set:

Tid	1	2	3	4	5	6	7	8	9	10
Items	{a,b}	{b,c,d}	{a,c,d,e}	{a,d,e}	{a,b,c}	{a,b,c,d}	{a}	{a,b,c}	{a,b,d}	{b,c,e}

Construct the FP tree by showing the trees separately after reading each transaction.

8. What is association analysis? Explain with example

Association rule mining is a popular and well researched method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, RakeshAgrawal et al. introduced association rules.

Problem Definition: The problem of association rule mining is defined as:

Let be a set of binary attributes called items.

Let be a set of transactions called the database.

Each transaction in has a unique transaction ID and contains a subset of the items in .

A rule is defined as an implication of the form

whereand .

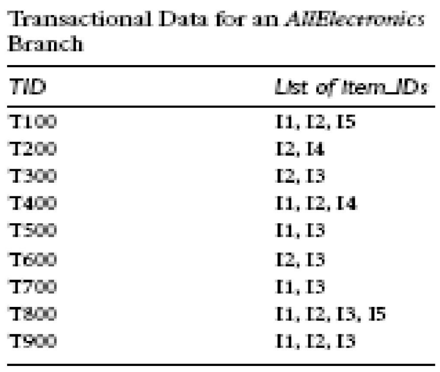
The sets of items (for short itemsets) and are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively.To illustrate the concepts, we use a small example from the supermarket domain. The set of items is and a small database containing the items (1 codes presence and 0 absence of an item in a transaction) is shown in the table.

An example rule for the supermarket could be meaning that if butter and bread are bought, customers also buy milk.

9. List and explain factors affecting the complexity of Apriori algorithm

12. Explain feature subset selection as away to reduce the dimensionality

14. Consider the following transaction data set:



Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

a) Construct FP tree

b) Generate list of frequent item set ordered by their corresponding suffixes -12

15. Find frequent item sets for the following table using Apriori algorithm:
Minimum support= 3

Transaction ID	**Items Bought**
T1	{M, O, N, K, E, Y }
T2	{D, O, N, K, E, Y }
T3	{M, A, K, E}
T4	{M, U, C, K, Y }
T5	{C, O, O, K, I, E}

Let's start with a non-simple example,

Transaction ID	**Items Bought**
T1	{Mango, Onion, Nintendo, Key-chain, Eggs, Yo-yo}
T2	{Doll, Onion, Nintendo, Key-chain, Eggs, Yo-yo}
T3	{Mango, Apple, Key-chain, Eggs}
T4	{Mango, Umbrella, Corn, Key-chain, Yo-yo}
T5	{Corn, Onion, Onion, Key-chain, Ice-cream, Eggs}

Now, we follow a simple golden rule: we say an item/itemset is frequently bought if it is bought at least 60% of times. So for here it should be bought at least 3 times.

For simplicity
M = Mango
O = Onion
And so on……

So the table becomes

Original table:

Transaction ID	Items Bought
T1	{M, O, N, K, E, Y }
T2	{D, O, N, K, E, Y }
T3	{M, A, K, E}
T4	{M, U, C, K, Y }
T5	{C, O, O, K, I, E}

Step 1: Count the number of transactions in which each item occurs, Note 'O=Onion' is bought 4 times in total, but, it occurs in just 3 transactions.

Item	No of transactions
M	3
O	3
N	2
K	5
E	4
Y	3

D	1
A	1
U	1
C	2
I	1

Step 2: Now remember we said the item is said frequently bought if it is bought at least 3 times. So in this step we remove all the items that are bought less than 3 times from the above table and we are left with

Item	Number of transactions
M	3
O	3
K	5
E	4
Y	3

This is the single items that are bought frequently. Now let's say we want to find a pair of items that are bought frequently. We continue from the above table (Table in step 2)

Step 3: We start making pairs from the first item, like MO,MK,ME,MY and then we start with the second item like OK,OE,OY. We did not do OM because we already did MO when we were making pairs with M and buying a Mango and Onion together is same as buying Onion and Mango together. After making all the pairs we get,

Item pairs
MO

MK
ME
MY
OK
OE
OY
KE
KY
EY

Step 4: Now we count how many times each pair is bought together. For example M and O is just bought together in {M,O,N,K,E,Y}
While M and K is bought together 3 times in {M,O,N,K,E,Y}, {M,A,K,E} AND {M,U,C, K, Y}
After doing that for all the pairs we get

Item Pairs	Number of transactions
MO	1
MK	3
ME	2
MY	2
OK	3
OE	3
OY	2
KE	4

KY	3
EY	2

Step 5: Golden rule to the rescue. Remove all the item pairs with number of transactions less than three and we are left with

Item Pairs	Number of transactions
MK	3
OK	3
OE	3
KE	4
KY	3

These are the pairs of items frequently bought together.
Now let’s say we want to find a set of three items that are brought together.
We use the above table (table in step 5) and make a set of 3 items.

Step 6: To make the set of three items we need one more rule (it’s termed as self-join),
It simply means, from the Item pairs in the above table, we find two pairs with the same first Alphabet, so we get

· OK and OE, this gives OKE
· KE and KY, this gives KEY

Then we find how many times O,K,E are bought together in the original table and same for K,E,Y and we get the following table

Item Set	Number of transactions
OKE	3

KEY	2

While we are on this, suppose you have sets of 3 items say ABC, ABD, ACD, ACE, BCD and you want to generate item sets of 4 items you look for two sets having the same first two alphabets.

- ABC and ABD -> ABCD
- ACD and ACE -> ACDE

And so on … In general you have to look for sets having just the last alphabet/item different.

Step 7: So we again apply the golden rule, that is, the item set must be bought together at least 3 times which leaves us with just OKE, Since KEY are bought together just two times.

Thus the set of three items that are bought together most frequently are O,K,E.

16. Explain Rule generation in Apriori algorithm with example (pseudocode expected)

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)     $L_1$ = find_frequent_1-itemsets(D);
(2)     for ($k = 2; L_{k-1} \neq \phi; k{+}{+}$) {
(3)         $C_k$ = apriori_gen($L_{k-1}$);
(4)         for each transaction $t \in D$ { // scan $D$ for counts
(5)             $C_t$ = subset($C_k, t$); // get the subsets of $t$ that are candidates
(6)             for each candidate $c \in C_t$
(7)                 c.count++;
(8)         }
(9)         $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10)    }
(11)    return $L = \cup_k L_k$;
```

```
procedure apriori_gen($L_{k-1}$:frequent $(k-1)$-itemsets)
(1)     for each itemset $l_1 \in L_{k-1}$
(2)         for each itemset $l_2 \in L_{k-1}$
(3)             if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge ... \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then {
(4)                 $c = l_1 \bowtie l_2$; // join step: generate candidates
(5)                 if has_infrequent_subset($c, L_{k-1}$) then
(6)                     delete $c$; // prune step: remove unfruitful candidate
(7)                 else add $c$ to $C_k$;
(8)             }
(9)     return $C_k$;
```

```
procedure has_infrequent_subset($c$: candidate $k$-itemset;
            $L_{k-1}$: frequent $(k-1)$-itemsets); // use prior knowledge
(1)     for each $(k-1)$-subset $s$ of $c$
(2)         if $s \notin L_{k-1}$ then
(3)             return TRUE;
(4)     return FALSE;
```