Week 5

Data Mining and Knowledge Discovery

Dr John Evans
j.evans8@herts.ac.uk

University of Hertfordshire UH

Plan for today

Data Preprocessing

Measuring Similarity and Dissimilarity

Proximity measures for nominal attributes

Proximity Measures for Binary Attributes

Recap

- We can classify attributes as either qualitative or quantitative.
- We can further classify attributes as Nominal, Ordinal, Interval, Ratio (NOIR).
- We can also classify attributes as continuous or discrete (binary).
- For asymmetric attributes, only presence is regarded as important (i.e. we are interested in non-zero values).
- A binary attribute in which both states are equally valuable and carry the same weight are called symmetric.
- We discussed noise and ways of ‘smoothing out’ data.
- We saw ways to define and compute precision and bias (discussed accuracy also).
- We saw how to deal with missing/inconsistent values.

Preprocessing

Where are we now?

- We know how to load data into Python.
- We have an idea of the different types of data we can encounter, and how to compute meaningful summary statistics on this data.
- We also have an appreciation for the problems that can be encountered, such as data quality, noise and missing values.

Preprocessing

Where are we now?

- We know how to load data into Python.
- We have an idea of the different types of data we can encounter, and how to compute meaningful summary statistics on this data.
- We also have an appreciation for the problems that can be encountered, such as data quality, noise and missing values.

What next?

- With an appreciation for what may be wrong with our data, we now want to ‘fix it’ in some way.
- This is what we call preprocessing.

Preprocessing

There are several preprocessing techniques we can use. In the notes we discuss five, but today will only discuss two:

- Aggregation (notes);
- Sampling (notes);
- Dimensionality reduction (notes);
- Discretization;
- Normalisation.

Roughly speaking, there are two categories: selecting data objects and attributes for the analysis, or for creating/changing the attributes. In both cases, the goal is to improve the data mining analysis with respect to time, cost and quality.

Rough overview

- *Data cleaning* can be applied to remove noise and correct inconsistencies in data.

Rough overview

- *Data cleaning* can be applied to remove noise and correct inconsistencies in data.
- *Data integration* merges data from multiple sources into a coherent data store such as a data warehouse.

Rough overview

- *Data cleaning* can be applied to remove noise and correct inconsistencies in data.
- *Data integration* merges data from multiple sources into a coherent data store such as a data warehouse.
- *Data reduction* can reduce data size by, for instance aggregating, eliminating redundant features, or clustering. Two main strategies include:
  - *dimensionality reduction*
  - *numerosity reduction*

Dimensional/numerosity reduction

- For dimensionality reduction, data encoding schemes are applied so as to obtain a reduced of ‘compressed’ representation of the original data. A key example of this (for us, at least) will be principal component analysis, but other examples include:
  - wavelet transforms
  - attribute subset selection (e.g. removing irrelevant attributes)
  - attribute construction (e.g. where a small set of more useful attributes is derived from the original set)

Dimensional/numerosity reduction

- For dimensionality reduction, data encoding schemes are applied so as to obtain a reduced of 'compressed' representation of the original data. A key example of this (for us, at least) will be principal component analysis, but other examples include:
  - wavelet transforms
  - attribute subset selection (e.g. removing irrelevant attributes)
  - attribute construction (e.g. where a small set of more useful attributes is derived from the original set)
- In numerosity reduction, the data are replaced by alternative, smaller representations using
  - parametric models (e.g. regression or log-linear models); or
  - nonparametric models (e.g. histograms, clusters, sampling or data aggregation).

Data transformation

- *Data transformations* (e.g. normalisation) may be applied, where data are scaled to fall within a smaller range, such as between 0.0 and 1.0. This can improve the accuracy and efficiency of mining algorithms involving distance measurements (more on this later).

Data transformation

- *Data transformations* (e.g. normalisation) may be applied, where data are scaled to fall within a smaller range, such as between 0.0 and 1.0. This can improve the accuracy and efficiency of mining algorithms involving distance measurements (more on this later).
- e.g. Consider some customer data that we want to analyse. In this data, we have the attributes *age* and *annual salary.* The annual salary attribute is going to be much larger than values of age. Thus, if the attributes are left unnormalised, the distance measurements taken on annual salary will generally outweigh distance measurements taken on the age attribute.

Data transformation

- *Data transformations* (e.g. normalisation) may be applied, where data are scaled to fall within a smaller range, such as between 0.0 and 1.0. This can improve the accuracy and efficiency of mining algorithms involving distance measurements (more on this later).
- e.g. Consider some customer data that we want to analyse. In this data, we have the attributes *age* and *annual salary.* The annual salary attribute is going to be much larger than values of age. Thus, if the attributes are left unnormalised, the distance measurements taken on annual salary will generally outweigh distance measurements taken on the age attribute.
- *Discretisation* and *concept hierarchy generation* can also be useful, where raw data values for attributes are replaced by ranges or higher conceptual levels.

Data transformation

- *Data transformations* (e.g. normalisation) may be applied, where data are scaled to fall within a smaller range, such as between 0.0 and 1.0. This can improve the accuracy and efficiency of mining algorithms involving distance measurements (more on this later).
- e.g. Consider some customer data that we want to analyse. In this data, we have the attributes *age* and *annual salary.* The annual salary attribute is going to be much larger than values of age. Thus, if the attributes are left unnormalised, the distance measurements taken on annual salary will generally outweigh distance measurements taken on the age attribute.
- *Discretisation* and *concept hierarchy generation* can also be useful, where raw data values for attributes are replaced by ranges or higher conceptual levels.
- e.g. Raw values for the age attribute may be replaced by higher-level concepts, such as *youth, adult* or *senior*.

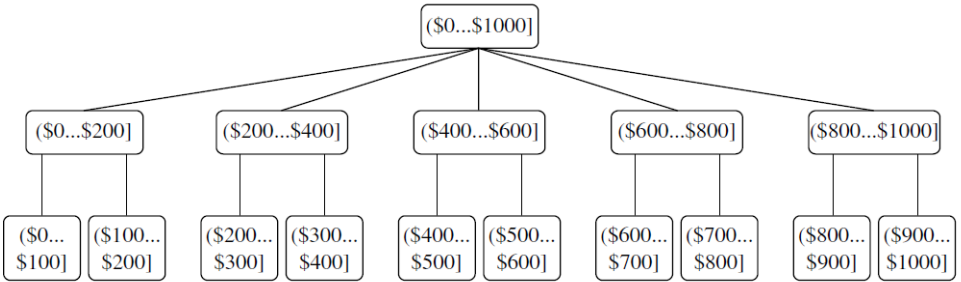
Discretisation

- Discretisation is a form of data transformation where continuous attributes (such as height or potentially age) are transformed into categorical attributes (such as $\{short,\ average,\ tall\}$ or $\{0-10,\ 11-20, \ldots\}$).

Discretisation

- Discretisation is a form of data transformation where continuous attributes (such as height or potentially age) are transformed into categorical attributes (such as $\{short,\ average,\ tall\}$ or $\{0-10,\ 11-20, \ldots\}$).
- We can take this further where continuous and/or discrete attributes may be transformed into one or more binary attributes (binarisation). This is useful for certain classification algorithms, and also for algorithms that find association patterns (which require data that have binary attributes).
- In general, the labels created in this transformation process can be recursively organised into higher-level concepts, resulting in a *concept hierarchy*. This is more often done for numeric attributes and can be used more generally in certain circumstances.

Concept hierarchy for the attribute *price*



Classifying discretisation

- We can categorise discretisation techniques based on how the discretisation is performed, such as whether it uses class information or which direction it proceeds (i.e. top-down v bottom-up).

Classifying discretisation

- We can categorise discretisation techniques based on how the discretisation is performed, such as whether it uses class information or which direction it proceeds (i.e. top-down v bottom-up).
- If the discretisation process uses class information, then we say it is *supervised discretisation*. Otherwise, it is *unsupervised*.

Classifying discretisation

- We can categorise discretisation techniques based on how the discretisation is performed, such as whether it uses class information or which direction it proceeds (i.e. top-down v bottom-up).
- If the discretisation process uses class information, then we say it is *supervised discretisation*. Otherwise, it is *unsupervised*.
- If the process starts by first finding one or a few points (called *split points* or *cut points*) to split the entire attribute range, and then repeats this recursively on the resulting intervals, it is called *top-down discretisation* or *splitting*.

Classifying discretisation

- We can categorise discretisation techniques based on how the discretisation is performed, such as whether it uses class information or which direction it proceeds (i.e. top-down v bottom-up).
- If the discretisation process uses class information, then we say it is *supervised discretisation*. Otherwise, it is *unsupervised*.
- If the process starts by first finding one or a few points (called *split points* or *cut points*) to split the entire attribute range, and then repeats this recursively on the resulting intervals, it is called *top-down discretisation* or *splitting*.
- This contrasts with *bottom-up discretisation* or *merging* which starts by considering all of the continuous values as potential split-points, removes some by merging neighbourhood values to form intervals, and then recursively applies this process to the resulting intervals.
- e.g. Binning is a top-down splitting technique based on a specified number of bins.

A simple technique for binarisation

The following is a simple technique to binarise a categorical attribute:

1. Suppose there are m categorical values in a set X.
2. Define an injective map,

$$X \to \{0, 1, 2, \ldots, m-1\} \subset \mathbb{Z}$$

   so that each value is assigned a unique integer in the interval $[0, m-1]$. Note, that if the attribute is ordinal, then the order must be maintained by the assignment.
3. Convert each of these m integers to a binary number. Note we need $n = \lceil \log_2(m) \rceil$ binary digits to represent these integers, where we use the ceiling function since not all integers can be represented as a power of 2. We then represent these binary numbers using n binary attributes.

Reminder of log

Recall, $\log_2(N)$ is the exponent value of 2 which gives N, i.e. if $\log_2(N) = k$ then this is the same as saying $2^k = N$. More generally, if $\log_a(N) = b$, then $a^b = N$. All the usual log rules apply, regardless of a. So,

Reminder of log

Recall, $\log_2(N)$ is the exponent value of 2 which gives N, i.e. if $\log_2(N) = k$ then this is the same as saying $2^k = N$. More generally, if $\log_a(N) = b$, then $a^b = N$. All the usual log rules apply, regardless of a. So,

- Product: $\log_a(N_1 N_2) = \log_a(N_1) + \log_a(N_2)$.
- Quotient: $\log_a(N_1/N_2) = \log_a(N_1) - \log_a(N_2)$.
- Power: $\log_a(N_1^x) = x \log_a(N_1)$.
- Identity: $\log_a(a) = 1$.
- Log of exponent: $\log_a(a^x) = x$.
- Exponent of log: $a^{\log_a(N)} = N$.
- Relationship with *ln*: $\log_b(x) = \frac{\ln(x)}{\ln(b)}$.

Reminder of log

Recall, $\log_2(N)$ is the exponent value of 2 which gives N, i.e. if $\log_2(N) = k$ then this is the same as saying $2^k = N$. More generally, if $\log_a(N) = b$, then $a^b = N$. All the usual log rules apply, regardless of a. So,

- Product: $\log_a(N_1 N_2) = \log_a(N_1) + \log_a(N_2)$.
- Quotient: $\log_a(N_1/N_2) = \log_a(N_1) - \log_a(N_2)$.
- Power: $\log_a(N_1^x) = x \log_a(N_1)$.
- Identity: $\log_a(a) = 1$.
- Log of exponent: $\log_a(a^x) = x$.
- Exponent of log: $a^{\log_a(N)} = N$.
- Relationship with *ln*: $\log_b(x) = \frac{\ln(x)}{\ln(b)}$.

$\log_2(128) = \log_2(64) + \log_2(2) = \log_2(32) + 2\log_2(2) = \log_2(16) + 3\log_2(2) = 4 + 3 = 7.$

Example

Consider a categorical variable with five values: $\{awful, poor, okay, good, great\}$.
First, define f by $f(awful) = 0$, $f(poor) = 1$, $f(okay) = 2$, $f(good) = 3$, $f(great) = 4$.

Example

Consider a categorical variable with five values: $\{awful, poor, okay, good, great\}$. First, define f by $f(awful) = 0$, $f(poor) = 1$, $f(okay) = 2$, $f(good) = 3$, $f(great) = 4$. Next, since $\log_2(4) = 2$ and $log_2(8) = 3$, we need three binary variables x_1, x_2, x_3. By converting 0, 1, 2, 3, 4 to binary, we will get our representations for x_1, x_2, x_3 for free. This is shown in the following table:

Categorical Value	Integer Value	x_1	x_2	x_3
awful	0	0	0	0
poor	1	0	0	1
okay	2	0	1	0
good	3	0	1	1
great	4	1	0	0

A problem

Such transformations can cause complications, such as creating unintended relationships among the transformed attributes.

For example, in the previous example, we have a correlation between x_2, x_3 arising from the information about the *good* value being encoded using both attributes.

Furthermore, association analysis requires asymmetric binary attributes, where only the presence of the attribute (value = 1) is important. For association problems, it is therefore necessary to introduce one asymmetric binary attribute for each categorical value, as shown in the table on the next slide:

Example continued

Categorical Value	Integer Value	x_1	x_2	x_3	x_4	x_5
awful	0	1	0	0	0	0
poor	1	0	1	0	0	0
okay	2	0	0	1	0	0
good	3	0	0	0	1	0
great	4	0	0	0	0	1

If the number of resulting attributes is too large, then other techniques can be used to reduce the number of categorical values before binarisation.

Discretisation of continuous attributes

- This is typically applied to attributes that are used in classification or association analysis. The transformation involves two subtasks:
  1. deciding how many categories *n* to have, and
  2. determining how to map the values of the continuous attribute to these categories.

Discretisation of continuous attributes

- This is typically applied to attributes that are used in classification or association analysis. The transformation involves two subtasks:
  1. deciding how many categories n to have, and
  2. determining how to map the values of the continuous attribute to these categories.
- In the first step, after the values of the continuous attribute are sorted, they are then divided into n intervals by specifying $n - 1$ split points.
- In the second, all values in one interval are mapped to the same categorical value.

Discretisation of continuous attributes

- This is typically applied to attributes that are used in classification or association analysis. The transformation involves two subtasks:
  1. deciding how many categories n to have, and
  2. determining how to map the values of the continuous attribute to these categories.
- In the first step, after the values of the continuous attribute are sorted, they are then divided into n intervals by specifying $n - 1$ split points.
- In the second, all values in one interval are mapped to the same categorical value.

Thus, the problem of discretisation is one of deciding how many split points to choose and where to place them. The result can be represented either as a set of intervals

$$\{(x_0, x_1], (x_1, x_2], \ldots, (x_{n-1}, x_n)\},$$

where x_0, x_n can be $\pm\infty$, or equivalently, as a series of inequalities

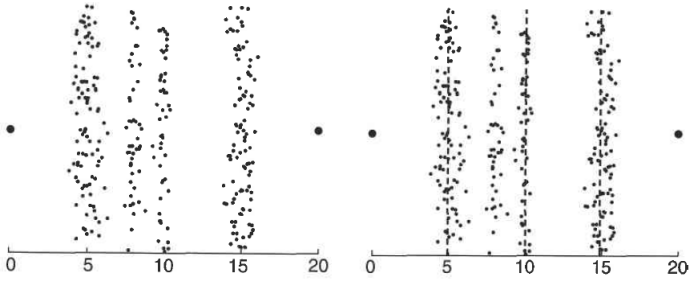
$$x_0 < x \leq x_1, \ldots, x_{n-1} < x < x_n.$$

Case 1: Class information not used

This is an *unsupervised discretisation* task. In this case, relatively simple approaches are common. We consider a few here.

- The *equal width* approach divides the range of the attribute into a user-specified number of intervals, each having the same width. Such an approach can be badly affected by outliers.
- An *equal frequency/equal depth* approach tries to put the same number of objects into each interval. Due to the equal width's weakness with outliers, this method is often preferred.
- A clustering method such as K-means can also be used. We will see this later in the course.
- Visually inspecting the data can sometimes be an effective approach.

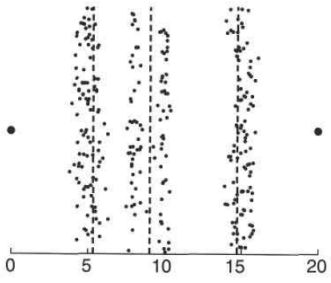
Example



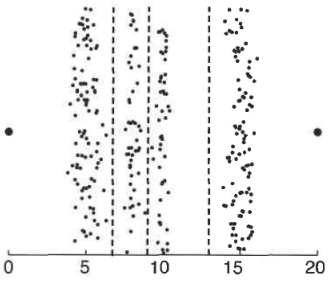
(a) Original data.

(b) Equal width discretization.

Example



(c) Equal frequency discretization.



(d) K-means discretization.

Case 2: Introduce class information

This is called *supervised discretisation* and often produces better classification.

Case 2: Introduce class information

This is called *supervised discretisation* and often produces better classification. A conceptually simple approach is to place the splits in a way that maximises the purity of the intervals, i.e. the extent to which an interval contains a single class label.

In practice, however, such an approach requires potentially arbitrary decisions about the purity of an interval and the minimum size of an interval.

Case 2: Introduce class information

- To overcome this, we can allow statistics to guide us.
- We start with each attribute value in a separate interval and create larger intervals by merging adjacent intervals that are similar according to a statistical test (recall, this is called a bottom-up approach).

Case 2: Introduce class information

- To overcome this, we can allow statistics to guide us.
- We start with each attribute value in a separate interval and create larger intervals by merging adjacent intervals that are similar according to a statistical test (recall, this is called a bottom-up approach).
- If we wanted a top-down approach then we can start by bisecting the initial values so that the resulting two intervals give minimum entropy. This only needs to consider each value as a possible split point because it is assumed that intervals contain ordered sets of values. The splitting process is then repeated with another interval, typically choosing the interval with the worst (highest) entropy, until a user-specified number of intervals is reached, or a stopping criterion is satisfied.

Entropy-based approaches are one of the most promising approaches to discretisation, whether bottom-up or top-down, and is used throughout data science.

Entropy: Surprise

Idea: Surprise and probability are inversely related. Higher the probability, lower the surprise. lower the probability, higher the surprise.

Entropy: Surprise

Idea: Surprise and probability are inversely related. Higher the probability, lower the surprise. lower the probability, higher the surprise.

1. Suppose we have a bunch of puppies (say there are 32 of them) and some are black, some are brown.
2. We gather them up into three separate areas A_1, A_2, A_3. Let's say A_1 has 8 (7 brown, 1 black) puppies, A_2 has 10 (5 black, 5 brown) and A_3 has 14 (4 brown, 10 black).

Entropy: Surprise

Idea: Surprise and probability are inversely related. Higher the probability, lower the surprise. lower the probability, higher the surprise.

1. Suppose we have a bunch of puppies (say there are 32 of them) and some are black, some are brown.
2. We gather them up into three separate areas A_1, A_2, A_3. Let's say A_1 has 8 (7 brown, 1 black) puppies, A_2 has 10 (5 black, 5 brown) and A_3 has 14 (4 brown, 10 black).
3. Next, we pick up a a puppy in A_1. The probability of this being a brown puppy is much higher than it being a black puppy because there are 7 brown and just 1 black puppy).
4. Thus, if we picked up a brown puppy, it would not be particularly surprising. Conversely, if we picked up a black puppy, we would be surprised.

Entropy: Surprise

5. Likewise, in A_2 (which has an equal number of brown and black puppies), we can pick up a puppy. However, whether it is brown or black we would be equally surprised.
6. Finally, in A_3 we can do the same thing and pick up a puppy. Here there are more black puppies than brown, so the probability of picking a black puppy is higher than picking a brown puppy. As such, if we picked up a black puppy, we wouldn't be very surprised.

Entropy: Computing surprise

To actually compute surprise, the obvious thing would be to compute $1/\textit{probability}$, but this isn't quite right.

Entropy: Computing surprise

To actually compute surprise, the obvious thing would be to compute $1/\textit{probability}$, but this isn't quite right.

Go back to A_1 and this time suppose it has no black puppies, i.e. $p(\textit{brown}) = 1$, $p(\textit{black}) = 0$. In this case, if we pick up a brown puppy, we should not be surprised at all (i.e. surprise should be 0), but $1/1 = 1 \neq 0$.

Entropy: Computing surprise

To actually compute surprise, the obvious thing would be to compute $1/\textit{probability}$, but this isn't quite right.

Go back to A_1 and this time suppose it has no black puppies, i.e. $p(\textit{brown}) = 1$, $p(\textit{black}) = 0$. In this case, if we pick up a brown puppy, we should not be surprised at all (i.e. surprise should be 0), but $1/1 = 1 \neq 0$.

Answer: Take logarithms (i.e. take $\log(1/\textit{prob})$).

Now this works for $\textit{prob} = 1$ since $\textit{log}(1) = 0$.

Entropy: Computing surprise

Another problem: There is a problem if $prob = 0$ since $\log(1/0) = \log(1) - \log(0)$ and $\log(0)$ is undefined.

Fortunately this isn't too bad since this scenario codes for an event that can never happen. It also means that $\log(n)$ for n very close to 0 is a large number and so $\log(1/prob)$ is very large (and negative!) meaning we are very surprised.

Entropy: Computing surprise

Another problem: There is a problem if $prob = 0$ since $\log(1/0) = \log(1) - \log(0)$ and $\log(0)$ is undefined.

Fortunately this isn't too bad since this scenario codes for an event that can never happen. It also means that $\log(n)$ for n very close to 0 is a large number and so $\log(1/prob)$ is very large (and negative!) meaning we are very surprised.

Something to keep in mind: We use $\log_2$ when we are working with two outcomes (such as heads/tails, or inside/outside an interval). So, the surprise in this case is $\log_2(1/prob)$.

Example

If we have a coin where $p(H) = 0.9$ and $p(T) = 0.1$, and we flip this coin 3 times, then the surprise of getting a heads, then tails, then heads is

$$\textit{surprise} = \log_2\left(\frac{1}{0.9 \times 0.1 \times 0.9}\right) = \log_2(1) - 2\log_2(0.9) - \log_2(0.1) = 3.62.$$

In particular, note this is just the sum of individual surprises.

Example

If we have a coin where $p(H) = 0.9$ and $p(T) = 0.1$, and we flip this coin 3 times, then the surprise of getting a heads, then tails, then heads is

$$\textit{surprise} = \log_2(\frac{1}{0.9 \times 0.1 \times 0.9}) = \log_2(1) - 2\log_2(0.9) - \log_2(0.1) = 3.62.$$

In particular, note this is just the sum of individual surprises.

We can, of course, extend this. Suppose we now flipped this coin 100 times. Then the surprise of getting a heads in 100 coin flips is just
$(0.9 \times 100) \times 0.15 = (\textit{prob} \times 100) \times \textit{surprise}$. Likewise, the probability of getting tails is $(0.1 \times 100) \times 3.32$. If we wanted the total surprise, then we just add the outcome of this. In this case, it is 46.7.

Shannon's Entropy

If we take the above example and divide by 100 (so we get 0.467), we get the average amount of surprise, per coin toss. This is the entropy of the coin, i.e. the expected surprise every time we flip the coin.

Shannon's Entropy

If we take the above example and divide by 100 (so we get 0.467), we get the average amount of surprise, per coin toss. This is the entropy of the coin, i.e. the expected surprise every time we flip the coin. In reality, we can simply our formula above since we are multiplying and dividing by 100. In reality,

$$E(\textit{surprise}) = (0.9 \times 0.15) + (0.1 \times 3.32) = 0.467.$$

In general,

$$E(\textit{surprise}) = \sum_x P(x) \log_2(1/P(x)) = -\sum_x P(x) \log_2(P(x)).$$

Example

Return to the example of the puppies above. The entropy for A_1 is

$$-\frac{7}{8}\log_2(7/8) - \frac{1}{8}\log_2(1/8)) = 0.54$$

Likewise, the entropy for A_2 is 1 and the entropy for A_3 is 0.86.

Entropy: Tidying things up

Let k be the number of different class labels, m_i the number of values in the i^{th} interval of a partition, and m_{ij} be the number of values of j in the interval i. Then the entropy e_i of the i^{th} interval is,

$$e_i = \sum_{j=1}^{k} p_{ij} \log_2(p_{ij}),$$

where $p_{ij} = \frac{m_{ij}}{m_i}$ is the probability of class j in the i^{th} interval.

Entropy: Tidying things up

Let k be the number of different class labels, m_i the number of values in the i^{th} interval of a partition, and m_{ij} be the number of values of j in the interval i. Then the entropy e_i of the i^{th} interval is,

$$e_i = \sum_{j=1}^{k} p_{ij} \log_2(p_{ij}),$$

where $p_{ij} = \frac{m_{ij}}{m_i}$ is the probability of class j in the i^{th} interval.

The total entropy e of the partition is the weighted average of the individual entropies,

$$e = \sum_{i=1}^{n} w_i e_i,$$

where m is the number of values, $w_i = m_i/m$ is the fraction of values in the i^{th} interval and n is the number of intervals.

Entropy: Intuition

Entropy is a measure of the purity of an interval. If an interval contains only values of one class (is perfectly pure) then our expected surprise/entropy is 0. It therefore contributes nothing to the overall entropy. Likewise, if the classes of values in an interval occur equally often (the interval is as impure as possible), then as we saw above, the entropy is at a maximum.

Measuring similarity and dissimilarity: Getting started

Idea: Formalise ways to measure how alike two objects are.

Measuring similarity and dissimilarity: Getting started

Idea: Formalise ways to measure how alike two objects are.

How do we do this?

1. Jaccard and cosine similarity measures for sparse data such as documents.
2. For non-sparse data such as time series or multi-dimensional points we can use correlation and Euclidean distance.

N.B. Often, we normalise similarity so that it is measured between 0 and 1 with 0 meaning no similarity and 1 meaning complete similarity, although sometimes the range $[0, \infty)$ is used. In reality, the range can be anything we like (within reason) and different algorithms may require different ranges.

Measuring similarity and dissimilarity: Getting started

Idea: Formalise ways to measure how alike two objects are.

How do we do this?

1. Jaccard and cosine similarity measures for sparse data such as documents.
2. For non-sparse data such as time series or multi-dimensional points we can use correlation and Euclidean distance.

N.B. Often, we normalise similarity so that it is measured between 0 and 1 with 0 meaning no similarity and 1 meaning complete similarity, although sometimes the range $[0, \infty)$ is used. In reality, the range can be anything we like (within reason) and different algorithms may require different ranges.

Exercise: Read pp. 62-63 of the notes to understand transformations.

Similarity/Dissimilarity matrix

Similarity and dissimilarity are naturally related[1], e.g. we can perform a simple transformation on $[0, 1]$ by $d = 1 - s$ or on $[-1, 1]$ by $d = -s$, say.

[1] It is typical to speak of *proximity* as well.

Similarity/Dissimilarity matrix

Similarity and dissimilarity are naturally related[1], e.g. we can perform a simple transformation on $[0, 1]$ by $d = 1 - s$ or on $[-1, 1]$ by $d = -s$, say.

We can represent similarity/dissimilarity by a similarity/dissimilarity matrix (it is usual to work with the latter). Suppose we have m objects and each has n attributes. We can represent each object as a row of a data matrix,

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}$$

[1] It is typical to speak of *proximity* as well.

Similarity/Dissimilarity matrix continued

We now want to measure the dissimilarity (or difference) between objects i and j, denoted $d(i, j)$.

[2]We have ignored entries above the main diagonal for ease.

Similarity/Dissimilarity matrix continued

We now want to measure the dissimilarity (or difference) between objects i and j, denoted $d(i, j)$. In general, we want $d(i, j)$ to be non-negative and 0 when objects i and j are completely similar, and we want $d(i, j) = d(j, i)$. Given this, we end up with a symmetric matrix (i.e. $A^T = A$) written as follows[2]:

$$\begin{pmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & & \ddots & \\ d(m,1) & d(m,2) & \cdots & \cdots & 0 \end{pmatrix}.$$

[2]We have ignored entries above the main diagonal for ease.

Proximity measures for one nominal attribute

Recall: A nominal attribute is one which involves equality and inequality only (i.e. they only convey information about the distinctness of objects). Examples include postcodes, employee ID, and so forth.

First, suppose our objects are described by just the one nominal attribute which can take N different states.

Proximity measures for one nominal attribute

Recall: A nominal attribute is one which involves equality and inequality only (i.e. they only convey information about the distinctness of objects). Examples include postcodes, employee ID, and so forth.

First, suppose our objects are described by just the one nominal attribute which can take N different states.

- It is typical to define similarity to be 1 if the attribute values match, and 0 otherwise.
- Likewise, dissimilarity could be matched the opposite way, i.e. 0 if the attribute values match and 1 if they do not.

Proximity measures for multiple nominal attributes

Next, suppose we have multiple nominal attributes.

- In this case, we can define dissimilarity between two objects x_i, x_j based on the ratio of mismatches,

$$d(i, j) = \frac{p - m}{p},$$

where

- m is the number of matches (i.e. the number of attributes for which x_i, x_j are in the same state), and
- p is the total number of attributes describing the objects.

Proximity measures for multiple nominal attributes

Next, suppose we have multiple nominal attributes.

- In this case, we can define dissimilarity between two objects x_i, x_j based on the ratio of mismatches,

$$d(i, j) = \frac{p - m}{p},$$

  where
  - m is the number of matches (i.e. the number of attributes for which x_i, x_j are in the same state), and
  - p is the total number of attributes describing the objects.
- Weights can also be assigned to increase the effect of m or to assign greater weight to the matches in attributes having a larger number of states.
- Of course, we can instead work with similarity using the formula,

$$sim(i, j) = 1 - d(i, j) = \frac{m}{p}.$$

Example

Suppose we have the following data in which only the first two columns are available:

Object Identifier	Test 1 - Nominal	Test 2 - Ordinal	Test 3 - Numeric
1	Code A	Excellent	45
2	Code B	Fair	22
3	Code C	Good	64
4	Code A	Excellent	28

Example

Suppose we have the following data in which only the first two columns are available:

Object Identifier	Test 1 - Nominal	Test 2 - Ordinal	Test 3 - Numeric
1	Code A	Excellent	45
2	Code B	Fair	22
3	Code C	Good	64
4	Code A	Excellent	28

In this case $p = 1$ since we just have the one nominal attribute (*Test 1*). So, $d(i, j) = 0$ if the attributes match, and 1 if they differ. The dissimilarity matrix is as follows:

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Proximity measures for binary attributes

Recall: A binary attribute has only one of two states - 0 and 1. Usually, 0 means that the attribute is absent, and 1 means that it is present. For example, the attribute might be *smoker* which describes whether or not a patient is a smoker. In this case, 1 might indicate that the patient smokes, while 0 indicates that the patient does not.

Proximity measures for binary attributes

Recall: A binary attribute has only one of two states - 0 and 1. Usually, 0 means that the attribute is absent, and 1 means that it is present. For example, the attribute might be *smoker* which describes whether or not a patient is a smoker. In this case, 1 might indicate that the patient smokes, while 0 indicates that the patient does not.

There are methods specific to binary data which are necessary for computing dissimilarity. One approach involves computing a dissimilarity matrix from the given binary data. Suppose all binary attributes are thought of as having the same weight and define the following:

- f_{11} = number of attributes that equal 1 for both objects.
- f_{10} = number of attributes that equal 1 for object x_i but 0 for object x_j.
- f_{01} = number of attributes that equal 0 for object x_i but 1 for object x_j.
- f_{00} = number of attributes that equal 0 for both objects.

A 2×2 contingency table

		Object x_j		
		1	0	Sum
Object x_i	1	f_{11}	f_{10}	$f_{11} + f_{10}$
	0	f_{01}	f_{00}	$f_{01} + f_{00}$
	Sum	$f_{11} + f_{01}$	$f_{10} + f_{00}$	p

Simple matching coefficient

A commonly used similarity coefficient is the *simple matching coefficient* (SMC), which is defined as follows:

$$SMC = \frac{\textit{Number of matching attribute values}}{\textit{Number of attributes}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

Simple matching coefficient

A commonly used similarity coefficient is the *simple matching coefficient* (SMC), which is defined as follows:

$$SMC = \frac{\textit{Number of matching attribute values}}{\textit{Number of attributes}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

- This measure counts both presences and absences equally.
- Consequently, the SMC could be used to find students who had answered questions similarly on a test that consisted only if true/false questions.

Symmetric binary dissimilarity

Recall: Symmetric binary attributes are those in which each state is equally valuable.

Dissimilarity that is based on symmetric binary attributes is called *symmetric binary dissimilarity*.

Symmetric binary dissimilarity

Recall: Symmetric binary attributes are those in which each state is equally valuable.

Dissimilarity that is based on symmetric binary attributes is called *symmetric binary dissimilarity*.

If objects x_i, x_j are described by symmetric binary attributes, then the dissimilarity between x_i, x_j is given by,

$$d(i, j) = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

Asymmetric Binary Dissimilarity

For asymmetric binary attributes, the two states are *not* equally important (for instance, in the case where positive (1) and negative (0) are outcomes of a disease test).

Asymmetric Binary Dissimilarity

For asymmetric binary attributes, the two states are *not* equally important (for instance, in the case where positive (1) and negative (0) are outcomes of a disease test).

Given two asymmetric binary attributes, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary attributes are often considered 'monary' (having one state). The dissimilarity based on these attributes is called *asymmetric binary dissimilarity*, where the number of negative matches f_{00} is considered unimportant and is thus ignored:

$$d(i, j) = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01}}.$$

Jaccard Coefficient

Complementing this, we can measure the difference between two binary attributes based on the notion of similarity instead of dissimilarity. This can be computed as,

$$sim(i, j) = \frac{f_{11}}{f_{11} + f_{10} + f_{01}} = 1 - d(i, j).$$

This coefficient is called the *Jaccard coefficient* and is popular in the literature.

Example 1

Suppose that we have the following patient record table containing the attributes *name, sex, fever, cough, Test 1, Test 2, Test 3, Test 4*:

name	sex	fever	cough	Test 1	Test 2	Test 3	Test 4
Tom	M	Y	N	P	N	N	N
Dick	M	Y	Y	N	N	N	N
Mary	F	Y	N	P	N	P	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Here, *name* is an object identifier, *sex* is a symmetric attribute and the remaining attributes are asymmetric binary.

Example 1

Suppose that we have the following patient record table containing the attributes *name, sex, fever, cough, Test 1, Test 2, Test 3, Test 4*:

name	sex	fever	cough	Test 1	Test 2	Test 3	Test 4
Tom	M	Y	N	P	N	N	N
Dick	M	Y	Y	N	N	N	N
Mary	F	Y	N	P	N	P	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Here, *name* is an object identifier, *sex* is a symmetric attribute and the remaining attributes are asymmetric binary.

For the asymmetric values we let *Y* (yes) and *P* (positive) be set to 1, and the value *N* (no or negative) be set to 0.

Example 1 cont.

Suppose that the distance between objects (patients) is computed based only on the asymmetric attributes. Then, the distance between each pair of the three patients (Tom, Dick and Mary) is as follows:

$$
\begin{aligned}
d(\mathit{Tom}, \mathit{Dick}) &= \frac{1+1}{1+1+1} = \frac{2}{3} \\
d(\mathit{Tom}, \mathit{Mary}) &= \frac{0+1}{2+0+1} = \frac{1}{3} \\
d(\mathit{Dick}, \mathit{Mary}) &= \frac{1+2}{1+1+2} = \frac{3}{4}.
\end{aligned}
$$

Example 2

Suppose we want to compute the Jaccard and SMC similarity measures for the following two binary vectors,

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then $f_{11} = 0,\ f_{01} = 2,\ f_{10} = 1,\ f_{00} = 7$

Example 2

Suppose we want to compute the Jaccard and SMC similarity measures for the following two binary vectors,

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then $f_{11} = 0$, $f_{01} = 2$, $f_{10} = 1$, $f_{00} = 7$ and $SMC = \frac{0+7}{2+1+0+7} = 0.7$, $J = \frac{0}{2+1+0} = 0$.

Summary

- We have seen discretisation and entropy.
- We have seen measures of similarity and dissimilarity.
- In particular, we have seen the SMC and Jaccard Coefficient.