5-Distances-Timeseries

September 22, 2016

1 Distances and Timeseries

Today we will start building some tools for making comparisons of data objects, with particular attention to timeseries.

Working with data, we can encounter a wide variety of different data objects:

- Records of users
- Graphs
- Images
- Videos
- Text (webpages, books)
- Strings (DNA sequences)
- Timeseries
- ...

How can we compare them?

1.1 Feature space representation

Usually a data object consists of a set of attributes.

These are also commonly called **features.**

- ("J. Smith", 25, $ 200,000)
- ("M. Jones", 47, $ 45,000)

If all d dimensions are real-valued then we can visualize each data object as a point in a vector space.

- (25, USD 200,000) $\rightarrow \begin{bmatrix} 25 \\ 200000 \end{bmatrix}$.

Likewise If all features are binary then we can think of each data object as a binary vector in vector space.

The space is called **feature space.**

We then are naturally interested in how **similar** or **dissimilar** two objects are.

A dissimilarity function takes two objects as input, and returns a large value when then two objects are not very similar.

Often we put restrictions on the dissimilarity function.

One of the most common is that it be a **metric.**

The dissimilarity $d(x, y)$ between two objects x and y is a **metric** if

- $d(i, j) = 0 \leftrightarrow i == j$ (identity of indiscernables)
- $d(i, j) = d(j, i)$ (symmetry)

- $d(i,j) \leq d(i,h) + d(h,j)$ (triangle inequality)

A metric is also commonly called a **distance**.

Sometimes we will use "distance" informally, ie, to refer to a dissimilarity function even if we are not sure it is a metric. We'll try to say "dissimilarity" in those cases though.

Why is it important or valuable for a dissimilarity to be a metric?

The additional constraints allow us to reason about and more easily visualize the data.

The main way this happens is through the triangle inequality.

The triangle inequality basically says, if two objects are "close" to another object, then they are "close" to each other.

This is not always the case for real data, but when it is true, it can really help.

Definitions of distance or dissimilarity functions are usually diferent for real, boolean, categorical, and ordinal variables.

Weights may be associated with diferent variables based on applications and data semantics.

1.2 Matrix representation

Very often we will manage data conveniently in matrix form.

The standard way of doing this is:

$$\overbrace{\begin{bmatrix} x_{11} & \dots & x_{1j} & \dots & x_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & \dots & x_{ij} & \dots & x_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mj} & \dots & x_{mn} \end{bmatrix}}^{n \text{ features}} \left.\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right\} m \text{ data objects}$$

Where we typically use symbols m for number of rows (objects) and n for number of columns (features).

When we are working with distances, the matrix representation is:

$$\overbrace{\begin{bmatrix} 0 & & & & \\ d(1,2) & 0 & & & \\ d(1,3) & d(2,3) & \ddots & & \\ \vdots & \vdots & \vdots & & \\ d(1,m) & d(2,m) & \dots & 0 & \end{bmatrix}}^{m \text{ data objects}} \left.\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right\} m \text{ data objects}$$

1.3 Norms

Assume some function $p(\mathbf{v})$ which measures the "size" of the vector $\mathbf{v}$.

$p()$ is called a **norm** if:

- $p(a\mathbf{v}) = |a|\, p(\mathbf{v})$ (absolute scaling)
- $p(\mathbf{u} + \mathbf{v}) \leq p(\mathbf{u}) + p(\mathbf{v})$ (subadditivity)
- $p(\mathbf{v}) = 0 \leftrightarrow \mathbf{v}$ is the zero vector (separates points)

Norms are important for this reason, among others:

Every norm defines a corresponding metric.

That is, if $p()$ is a norm, then $d(\mathbf{x}, \mathbf{y}) = p(\mathbf{x} - \mathbf{y})$ is a metric.

1.4 Useful Norms

ℓ_p Norm, where $p \geq 1$.

Defines the *Minkowski distance.*

$$L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

A special case of this isthe ℓ_2 norm that we've already studied, which defines the **Euclidean** norm.

It is equal to the euclidean distance between the vectors.

$$L_1(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

Another important special case is the ℓ_1 norm.

This defines the **Manhattan** distance, or (for binary vectors), the **Hamming** distance:

$$L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1} |x_i - y_i|$$

If we take the limit of the ℓ_p norm as p gets large ($\lim_{p \to \infty} L_p$) we get the ℓ_∞ norm.

The value of the ℓ_∞ norm is simply the **largest element** in a vector.

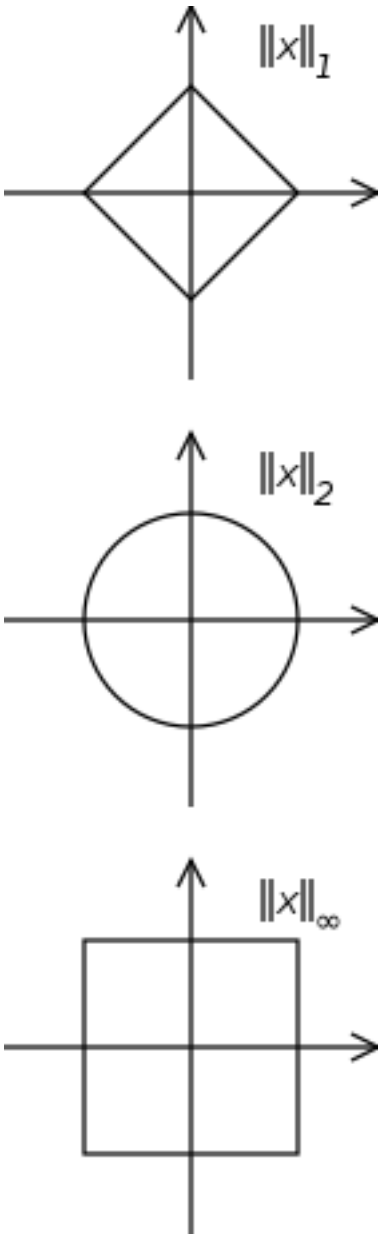
What is the metric that this norm induces?

Another related idea is the ℓ_0 "norm," which is not a norm, but is in a sense what we get from the p-norm for $p = 0$.

Note that this is **not** a norm, but it gets called that anyway.

This "norm" simply counts the number of **nonzero** elements in a vector.

This is called the vector's **sparsity.**



1.5 Similarity and Dissimilarity

We've already seen that the inner product of two vectors can be used to compute the **cosine of the angle** between them:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

Note that this value is **large** when $\mathbf{x} \approx \mathbf{y}$. So it is a **similarity** function.

We often find that we have a similarity function and need to convert it to a dissimilarity function. Two straightforward ways of doing that are:

$$d(x, y) = 1/s(x, y)$$

$$d(x, y) = k - s(x, y)$$

For some properly chosen k.
For cosine similarity, one often uses:

$$d(\mathbf{x}, \mathbf{y}) = 1 - \cos(\mathbf{x}, \mathbf{y})$$

Note however that this is **not a metric!**
However if we recover the actual angle beween $\mathbf{x}$ and $\mathbf{y}$, that is a metric.

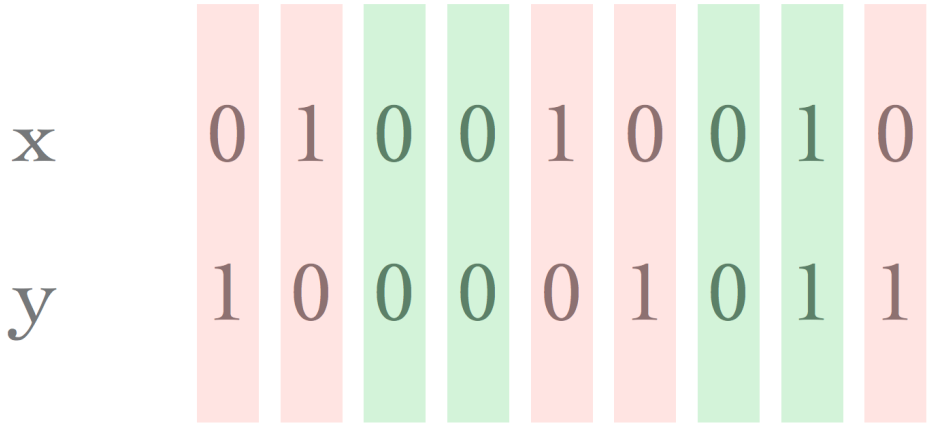
1.6 Bit vectors and Sets

When working with bit vectors, the ℓ_1 metric is commonly used and called the **Hamming** distance.

x	0	1	0	0	1	0	0	1	0
y	1	0	0	0	0	1	0	1	1

$$L_1(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i| \right)$$

This has a natural interpretation: "how well do the two vectors match?"
Or: "What is the smallest number of bit flips that will convert one vector into the other?"

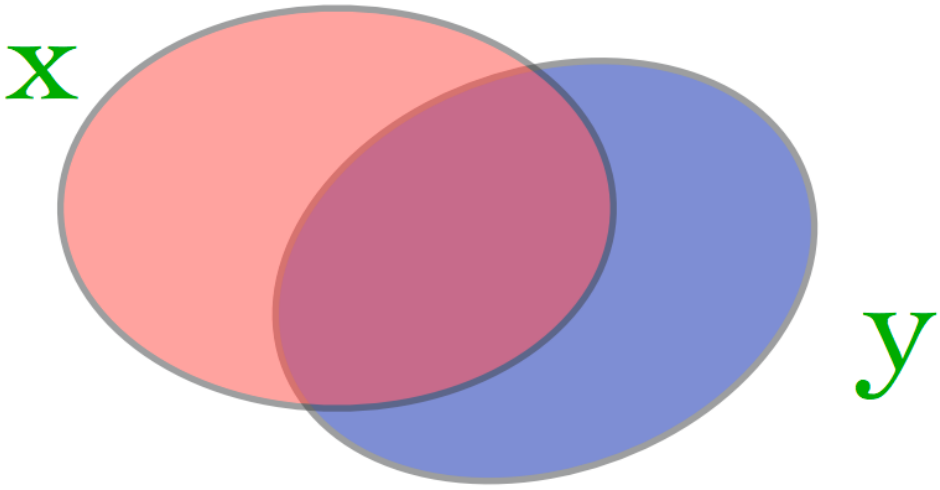


$$L_1(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i| \right)$$

In other cases, the Hamming distance is not a very appropriate metric.
Consider the case in which the bit vector is being used to represent a set.
In that case, Hamming distance measures the **size of the set difference.**
For example, consider two documents. We will use bit vectors to represent the sets of words in each document.

- Case 1: both documents are large, almost identical, but differ in 10 words.
- Case 2: both documents are small, disjoint, have 5 words each.

The situation can be represented as this:



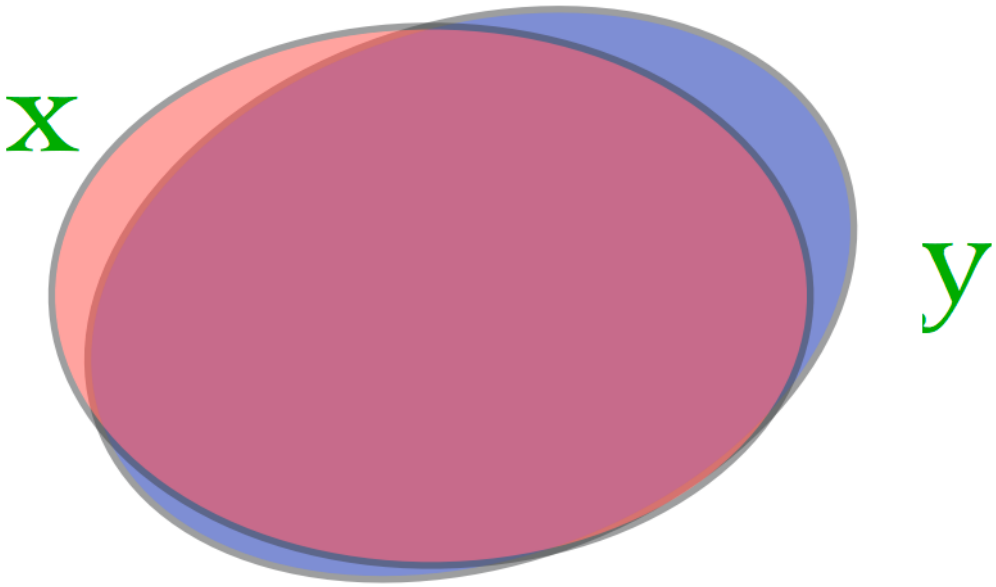
What matters is not just the size of the set difference, but the size of the intersection as well. This leads to the *Jaccard* similarity:

$$JSim(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x} \cap \mathbf{y}|}{|\mathbf{x} \cup \mathbf{y}|}$$

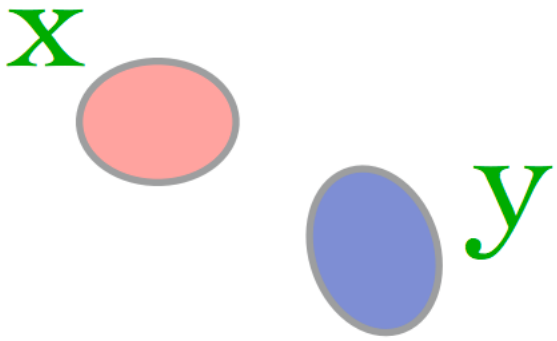
This takes on values from 0 to 1, so a natural dissimilarity metric is $1 - JSim()$.
In fact, this is a **metric!**:

$$JDist(\mathbf{x}, \mathbf{y}) = 1 - \frac{|\mathbf{x} \cap \mathbf{y}|}{|\mathbf{x} \cup \mathbf{y}|}$$

Consider our two cases:
Case 1: (very large almost identical documents)



Here $JSim(\mathbf{x}, \mathbf{y})$ is almost 1.
Case 2: (small disjoint documents)



Here $JSim(\mathbf{x}, \mathbf{y})$ is 0.

1.7 Time Series

A time series is a sequence of real numbers, representing the measurements of a real variable at (equal) time intervals.

- Stock prices
- Volume of sales over time
- Daily temperature readings

A time series database is a large collection of time series.

1.8 Similarity of Time Series

How should we measure the "similarity" of two timeseries?
We will assume they are the same length.
Examples:

- Find companies with similar stock price movements over a time interval
- Find similar DNA sequences
- Find users with similar credit usage patterns

Two Problems:

1. Defining a meaningful similarity (or distance) function.
2. Finding an efficient algorithm to compute it.

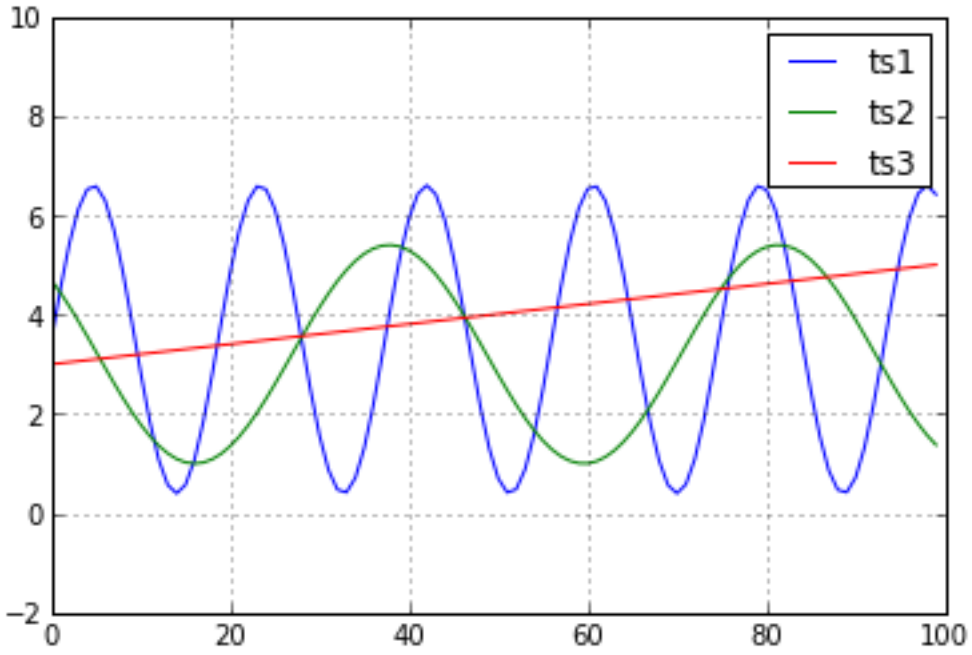
1.9 Norm-based Similarity Measures

We could just view each sequence as a vector.
Then we could use a p-norm, eg ℓ_1, ℓ_2, or ℓ_p to measure similarity.
Advantages:

1. Easy to compute - linear in the length of the time series (O(n)).
2. It is a metric.

Disadvantage:

1. May not be **meaningful!**



It is clear that ts1 and ts2 are "more similar."
However, according to Euclidean distance: d(ts1, ts2) = 26.9, while d(ts1, ts3) = 23.2.
Not good!

1.10 Feature Engineering

In general, there may be different aspects of a timeseries that are important in different settings.

The first step therefore is to ask yourself "what is important about timeseries in my application?"

This is an example of **feature engineering.**

Namely, the art of computing some derived measure from your data object that makes its important properties usable in a subsequent step.

A reasonable approach may then to be:

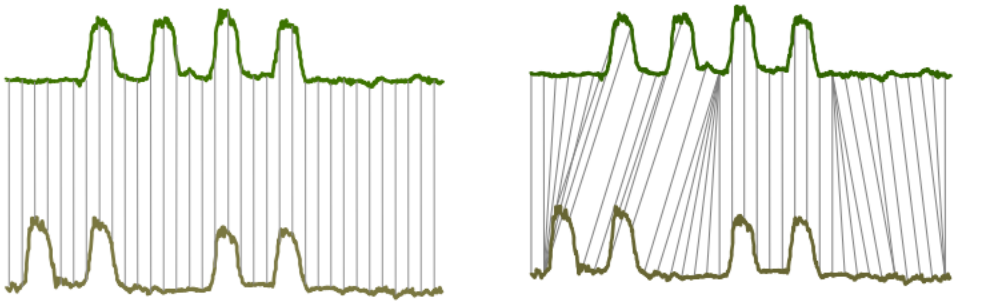
- extract the relevant features
- use a simple method (eg, a norm) to define similarity over those features.

In the case above, one might think of using

- Fourier coefficients (to capture periodicity)
- Histograms
- Or something else!

1.11 Dynamic Time Warping

One case that arises often is something like the following: "bump hunting"



Images from: Eamonn Keogh's slides

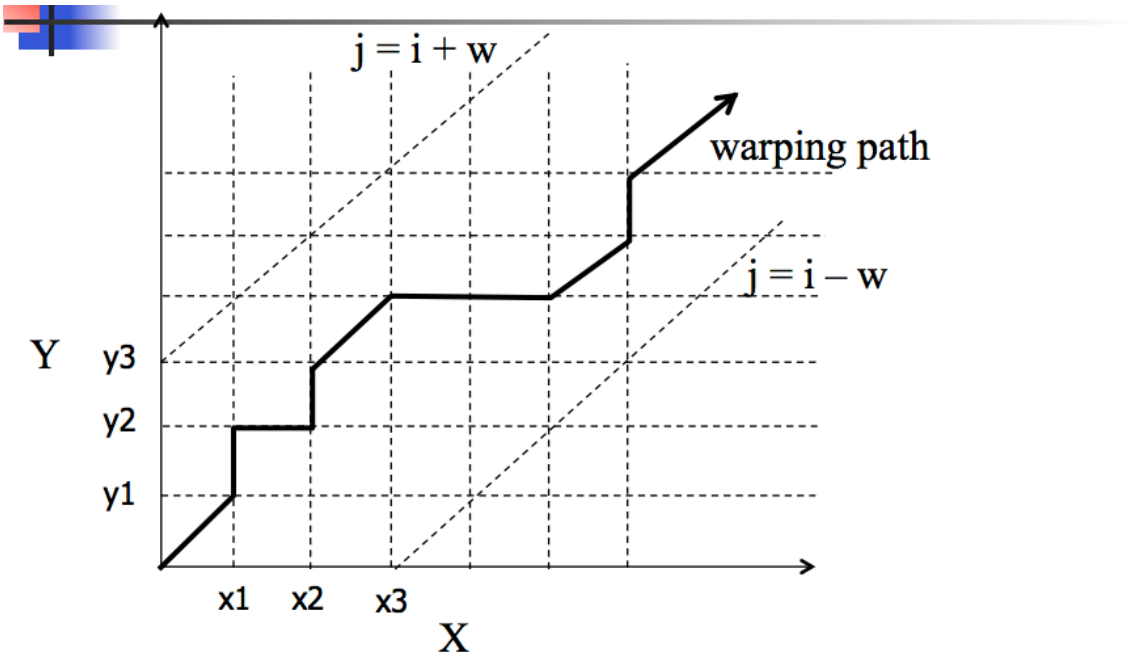
Both timeseries have the same key characteristics: four bumps.
But a one-one match (ala Euclidean distance) will not detect this.
A solution to this is called **dynamic time warping.**
The basic idea is to allow acceleration or deceleration of signals along the time dimension.
Classic applications:

- Speech recognition
- Signature recognition

Specifically:

- Consider $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$.
- We are allowed to extend each sequence by repeating elements to form X' and Y'.
- We then calculate, eg, Euclidean distance between the extended sequnces X' and Y'

There is a simple way to visualize this algorithm.
Consider a matrix M where $M_{ij} = |x_i - y_j|$ (or some other error measure).



M measures the amount of error we get if we match x_i with y_j.
So we seek a **path through M that minimizes the total error.**
We need to start in the lower left and work our way up via a continuous path.
Basic restrictions on path:

- Montonicity
  - path should not go down or to the left
- Continuity
  - No elements may be skipped in sequence

This can be solved via dynamic programming. However, the algorithm is still quadratic in n.
Hence, we may choose to put a restriction on the amount that the path can deviate from the diagonal.
The basic algorithm looks like this:

```
D[0, 0] = 0 for i in range(n):     for j in range (m):         D[i,j] = M[i,j] +
min( D[i-1, j],  # insertion                                  D[i, j-1],  # deletion
D[i-1, j-1]) # match
```

Unfortunately, the algorithm is still quadratic in n ($O(nm)$).
Hence, we may choose to put a restriction on the amount that the path can deviate from the diagonal.
This is implemented by not allowing the path to pass through locations where $|i - j| > w$.
Then the algorithm is $O(nw)$.

1.12 A related notion

A closely related idea concerns strings.
(Strings are just sequences, like timeseries).
Given two strings, one way to define a 'distance' between them as the minimum number of edit operations that are needed to transform one string into the other.
Edit operations are insertion, deletion, and substitution of single characters.
This is called **edit distance** or **Levenshtein distance.**
A very similar dynamic programming algorithm can be used to find this distance.