Geospatial Analytics 24-25

Notes

University of Pisa

M.Sc. in Data Science and Business Informatics

Contents

Chapter 1

Introduction

A geographic information system (GIS) is a computer system used to capture, store, query, analyze, and display geospatial data. **Geospatial data** describes both the location and the characteristics of spatial features: for example, if we want to describe a road, we may refer to its location and its features (length, name, speed limit, etc.). Other than single entities, geospatial data can also describe trajectories, specifying the sequence of locations constituting it.
The following chapter will give an overview of the basic concepts used in GISs and spatial data analysis.

1.1 Geographic Coordinate Systems

When using a GIS, any map layers used together must align spatially; to make sure this is true, we need to use some common spatial reference system for all maps. GIS users normally work with locations expressed on a plane using a coordinate system expressed in x- and y-coordinates, while the actual, real-life locations represented by them are on Earth's surface (which is ellipsoidal). A **map projection** is used to convert the Earth's surface to a plane.

The system used to locate points on Earth is called **geographic coordinate system**. This system is defined by two coordinates: **longitude** and **latitude**. They are angular measures which measure the angle at which the point can be found with respect to the **prime meridian** and the **equator**; longitute represents the angle east or west from the prime meridian, while latitude represents the angle north or south of the equatorial plane.

Meridians are lines of equal longitude. The prime meridian passes through Greenwich, England, and corresponds to 0°. **Parallels** are lines of equal latitude. The equator is the line corresponding to 0° latitude.

In a **plane coordinates** system, longitute and latitude correspond to x and y coordinates respectively. Logitude takes positive values in the easter hemisphere, and negative values in the western hemisphere; latitude takes positive values north of the equator, and negative values south of the equator.

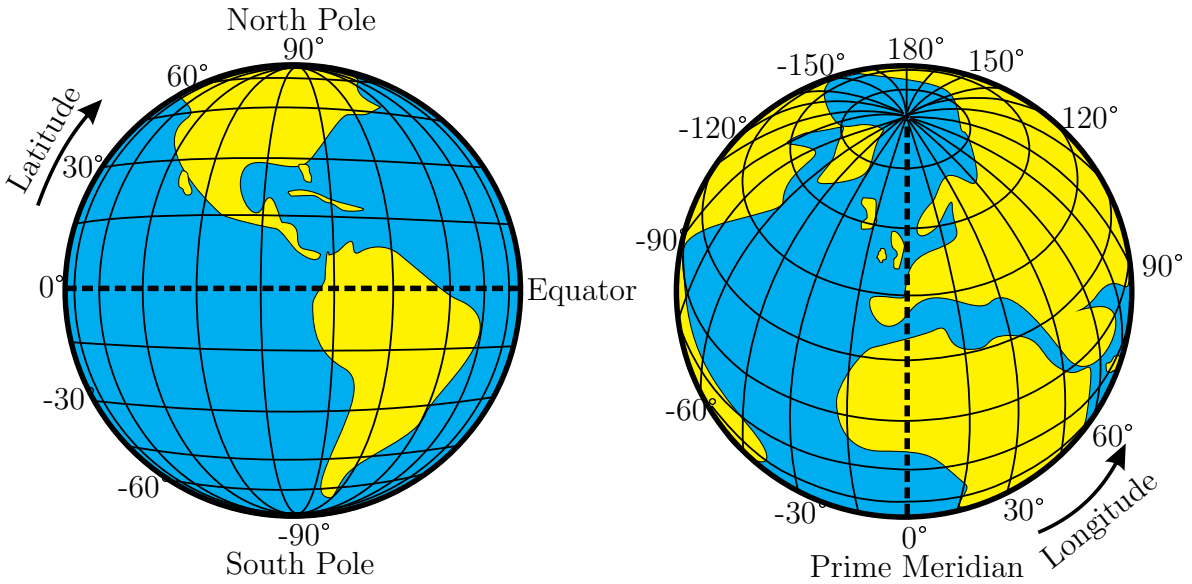


Figure 1.1: Latitude and longitude.

Longitute and latitude values may be expressed in different ways:

- **Decimal degrees** (**DD**): represented by a single decimal value;
- **Degrees-minutes-seconds** (**DMS**): represented by a set of three values, corresponding to degrees, minutes, and seconds. 1 degree corresponds to 60 minutes, and 1 minute corresponds to 60 seconds;
- **Radians** (**rad**): similar to DD, but expressed in radians instead of decimal values: 1 degree is equal to 0.01745 rad.

As mentioned before, planet Earth can be approximated as an **ellipsoid**: this shape is obtained by rotating an ellipse by its shortest axis. Indeed, Earth is wider along the equator (its major axis) than it is between the poles (its minor axis). Another parameter that describes an ellipsoid is the **flattening** (f), calculated as $f = \frac{maj-min}{maj}$, and it describes the difference between the two axes.

A **datum** is a mathematical model of the Earth which is used as the reference to calculate the geographic coordinates of a point (or even the elevation, if we consider vertical datums). A datum is defined as: the pair *longitude, latitude* of coordinates of

an initial point wich will be the origin, an ellipsoid, and the separation of the ellipsiod and the Earth at the origin.

Distances on the Earth's surface are not straight lines; they are instead represented by **geodesics**: through any two points (not antipodal), there is exactly one "great circle" that connects them. The two points separate the great circle in two parts: the shorter of the two is their geodesic distance. Since the Earth is nearly spherical, geodesic distances are correct with an error up to 0.5%. The geodesic distance between points A and B is calculated using the **spherical law of cosines**:

$$\cos(d) = \sin(lat_A)\sin(lat_B) + \cos(lat_B) + cos(lon_A - lon_B),$$

where d is the angular distance between the two points. To convert d to a linear distance measure, it can be multiplied by 111.32 km, which is the length of 1 degree at the equator. This formula is the basis from which the **haversine formula** is derived:

$$\text{hav}(\theta) = \text{hav}(lat_B - lat_A) + \cos(lat_A)\cos(lat_B)\text{hav}(lon_B - lon_A),$$
$$\text{hav}(\theta) = \sin^2(\frac{\theta}{2})$$

1.2 Trajectories, Tessellations, Flows

Let u be an individual. A **trajectory** $T_u = \langle p_1, p_2, \ldots, p_{nu}\rangle$ is a time-ordered sequence composed by the spatio-temporal points visited by u. A spatio-temporal **point** is a pair $p = (t, l)$, where t is the time, and $l = (x, y)$ is the point visited at that time.

Given an area A, a **tessellation** is a set of geographical polygons with the following properties:

- It contains a finite number of polygons called **tiles**:

$$\mathbb{G} = \{g_i : i = 1, \ldots, n\}$$

- The tiles are non overlapping:

$$g_i \cap g_j = \emptyset\,, \forall i \neq j$$

- The union of all tiles completely covers the tessellation:

$$\bigcup_{i=1}^{n} g_i = A$$

A tessellation can be **regular** or **irregular** depending on the shape of its tiles. Regular tessellation may use equilateral triangles, squares, hexagons; irregular tessellation may use buildings, census cells, administrative units. A **spatial join** is used to associate a point with the tile it belongs to. Since the tiles are non overlapping and cover the entire area, each point belongs to one and only one tile. **Voronoi tesselations** are a particular type of tessellation that partition the plane into regions (called **cells**), each closer to a secific point (called **seed**) out of a set. Each of these cells is defined as the set of points that are closest to the seed of the cell itself than any other seed in that area.

Given a tessellation, the **flow**

$$y(g_i.g_j)$$

represents the number of people/objects moving between g_i and g_j. A trajectory refers to a single entity, while a flow refers to the total amount of entities moving between two points. Flows can be derived from a set of trajectories, but the inverse is not true.

1.3 Raster and Vector Data Models

The raster and vector data models are two ways to represent geographic information in GISs. In both cases, data can be stored in several **thematic layers**, each of which contains a set of objects of the same nature. For example, a layer may contain information about buildings, another about streets, another about rivers, and so on.

The **raster data** model divides the space into a regular grid of square cells with a given size (which defines the **resolution**). This format is often used for images, where each element corresponds to a pixel. Depending on how much information is assigned to a cell, data can be **single-band** (one attribute per cell), or **multi-band** (several attributes per cell). Raster data is typically sourced from satellites.

The **vector data** model uses discrete objects (points, lines, polygons) to represent spatial features. Each object can have its own properties and relationship with the others. A **point** is a zero-dimensional object with a *location* property (expressed as x,y coordinates). A **line** in a one-dimensional object with two properties: *location* and *lenght*. It be either straight or curved. A **polygon** is a two-dimensional object with three properties: *location*, *area*, and *perimeter*. These objects are expressed differently depending on the data format used by the software/platform.

Objects in a layer are sometimes also called *spatial features*; for this reasin, the variables associated to them are called *attributes* (and not features). To represent geometric objects in a GIS, we can use one of the following models:

- **Geo-relational data model**, where objects and attributes are stored separately, and associating each object to the corresponding attributes requires a join operation (at the advantage of possibly saving space if a certain attribute(s) is (are) only possessed by few objects);
- **Object-relational data model**, where objects and attributes are stored together in a single table, making retrieval much faster (but possibly increasing the amount of space needed to store everything).

In principle, vectors can model everything; raster data is a discretized view of the same information. Raster data is better suited for "dense" data; it can be more efficient in those cases where a raster representation may need a very high number of objects, but precision is not a concern. Vector data can also be converted to raster data, and vice versa. **Rasterization** is the process of transforming vector data into raster data, and produces a discrete approximation. **Vectorization** is the inverse process: it may be difficult at times, and many algorithms and methods have been developed to perform it.

At times, vectors and raster information can be used together in multi-layer data. Some information is better modeled with one format than the other: for example, street networks or locations of interest are often encoded as vector layers, while things like land usage are encoded as raster layers.

1.4 Spatial Operations

The most important spatial operations are:

- **Intersection**: returns all the points in common with the operands.
- **Union**: returns the union of the two operands. In some tools, They are kept as separate objects, meaning that the result is always a multipolygon. As an alternate operation, the same tools offer the **dissolve** operation, which instead merges the two objects into a single one.
- **Difference**: returns all the points in the first operand which are not in the second.
- **Buffering**: creates a buffer, i.e., an expanded area, around the object. The result is equivalent to replacing each point in the geometry with a circle with a given radius.
- **Spatial join**: like in relational databases, joins merge the information of two objects. THe join can be **inner** (the output contains only pairs in common

with both objects), or **outer/left/right** (the output also contains non matching objects with the *NULL* value in place of the missing attributes).

1.5 Spatial Patterns and Spatial Correlation

Point Pattern Analysis (**PAA**) is the study of point patterns, i.e., the spatial distribution of points in an area. Spatial distributions are typically categorized into three types:

- **Uniform (discrete)**: points are evenly distributed in the area;
- **Random**: points are distributed according to a random process;
- **Clustered**: points appear to be grouped (clustered) in some areas.

A basic form of point pattern analysis consists in determining summary statistics such as mean center, standard distance, and standard deviational ellipse.
Mean center is the average of the x and y coordinate values:

$$\bar{s} = \left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n} \right)$$

Standard distance measures the variance between the average distance of the features to the mean center:

$$d = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu_x)^2 + (y_i - \mu_y)^2}{n}}$$

Similar to standand distance, **standard deviational ellipse** measures the standard distances for each axis:

$$d_x = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu_x)^2}{n}}$$

$$d_y = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \mu_y)^2}{n}}$$

Average Nearest Neighbor (**ANN**) is an algorithm that can be used to study patterns. For each point, its nearest neighbor is found as the point with the smallest distance to it. The average of all points' nearest neighbor distance is calculated as d_{obs}, and normalized with regards to the expected average if the pattern were random (d_{exp}), obtaining a ratio:

$$R = \frac{d_{obs}}{d_{exp}}$$

If $R = 1$, the pattern is random. If $R < 1$, the pattern is clustered, because the distances are smaller than expected; if $R > 1$, the pattern is uniform (or at least more dispersed than random).

Ripley's K-function is another popular method for analyzing point patterns. Usually, its normalized version, called **L function**, is used. Given n points in an area of size A, and a distance d, the L function is calculated as follows:

1. Compute all $n * (n-1)$ distances between each pair of points;
2. Compute ϕ, the fraction of distances that are $\leq d$;
3. Compute

$$L(d) = \sqrt{\frac{A}{\pi}\phi}\,.$$

$L(d) = d$ for random distributions. If $L(d)$ is higher, the data is more clustered; if it is lower, the data is more dispersed. Different values of d can be explored to understand patterns at different spatial granularities.

Another important aspect is **density based analysis**. Density measurements can be either global or local. **Global density** is simply calculated as the ratio of observed points and the study region's area:

$$\hat{\lambda} = \frac{n}{A}$$

Density can also be measured at different locations of the study region. **Local density** is computed over a single tessellation cell; the chosen resolution will affect the resulting density calculation. **Kernel density** is another method of calculating density per-cell which considers also the points found in its neighborhood. Usually, given a cell c, the 8 adjacent cells are considered as the neighborhood N_c, and so the kernel density becomes:

$$\textit{Kernel density}\ (c) = \hat{\lambda}(c \cup N_c)$$

A variant is **weighted kernel density**, which assigns to each point a weight inversely proportional to the distance from the cell's center. Different weight functions can be used; a common one is the Gaussian function.

Autocorrelation is the correlation of the values of a same variable measured at different points in time (**temporal autocorrelation**)/space (**spatial autocorrelation**). An example of spatial autocorralation may be checking how much the temperature values in the points of a layer are influenced by the neighboring values. According to **Tobler's first law of geography**, "everything is related to everything else, but near things are more related than distant things": this is the fundamental assumption in spatial analysis.

Popular measures of spatial autocorrelation are:

- **Moran's I**, which calculates the autocorrelation between values of each point against all other points in its neighborhood:

$$I = \frac{\sum_{i=1}^{n} \sum_{j:x_j \in N_{x_i}} w_{ij}(x_i - \mu_x)(x_j - \mu_x)}{s^2 \sum_{i=1}^{n} \sum_{j:x_j \in N_{x_i}} w_{ij}}$$

  where s^2 is the variance of the x values, and w_{ij} is a weight, typically defined as the inverse of the distance between the two points. Positive values mean positive correlation, negative values mean negative correlation;

- **Geary's C**:

$$C = \frac{n-1 \sum_{i=1}^{n} \sum_{j:x_j \in N_{x_i}} w_{ij}(x_i - x_j)^2}{2(\sum_{i=1}^{n} \sum_{j:x_j \in N_{x_i}} w_{ij}) * \sum_{i=1}^{n}(x_i - \mu_x)^2}$$

  The higher it is, the more different are nearby values (less correlation), the lower it is the closer they are (more correlation).

Both measures can also be interpreted as the average of local I/C values calculated across neighborhoods. These local values can be studied individually as well.

1.6 Spatial Interpolation

Spatial interpolation refers to the process of using points with known values (called **control points**) to estimate values at others. For example, we could estimate the temperature at a point with no recorded data by approximating it from known temperatures at nearby points. Ideally, control points should be well distributed across the study area, although this situation is rare in real-world applications since a study area oftentimes also contains data-poor areas.

Interpolation methods can be divided in two groups: **deterministic** and **stochastic**. The first group assumes that the known values are exact, with no assessment of errors for predicted values. The second group considers the presence of some random error in the known data and offers some assessment of prediction error with an estimated variance.

1.6.1 Deterministic Methods

Proximity Interpolation

Proximity interpolation (also known as **Thiessen interpolation**) is one of the simplest and oldest interpolation methods. The goal is to assign to all unsampled locations the value of the closest sampled location, producing a Voronoi tessellation over the study

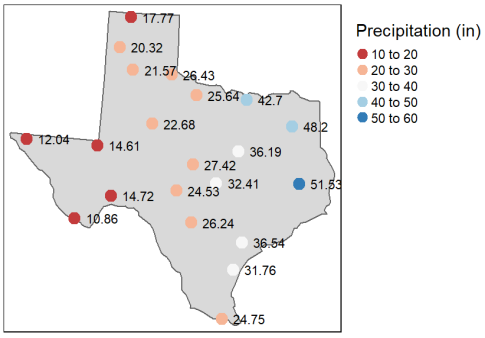


Figure 1.2: An example of samples corresponding to yearly precipitation recorded in different sites in Texas.

area. All the points within the same cell have the same value. A problem of this approach is that surface values change abruptly across the perimeter of adjacent cells, which is not realistic.

Inverse Distance Weighted Interpolation

Inverse Distance Weighted (IDW) interpolation calculates an average value using nearby weighted locations. The weight of each sample location is inversely proportional to the distance; the value at location j is given by:

$$\hat{Z}_j = \frac{\sum_i Z_i / d_{ij}^n}{\sum_i 1 / d_{ij}^n}$$

Here, d_{ij} is the distance between points i and j, and n is a hyperparameter that controls the irrelevance of a point as the distance increases/decreases: the larger n is, the less far away samples influence the interpolated value. For $n \to \infty$, the result is equivalent to proximity interpolation.
Values returned by this method are always within the range of the known values: $[Z_{min}, Z_{max}]$.

1.6.2 Stochastic Methods

Trend Surface Interpolation

Trend surface analysis approximates points with known values using a polynomial equation. The same equation can then be used to predict values at other points. Depending on the order of the polynomial, the approximation can be more or less complex:

- A 0^{th} order surface is described by $Z = z$, where c is the average value of all samples;

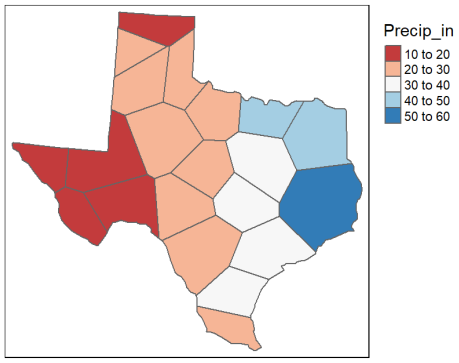


Figure 1.3: Interpolated values obtained by proximity interpolation.

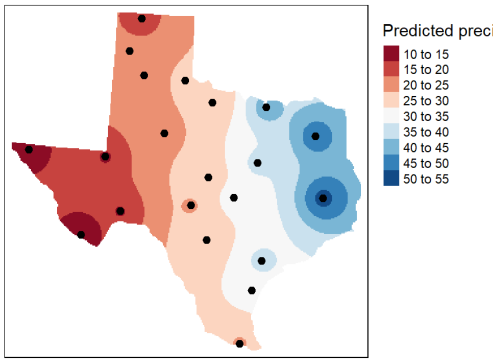


Figure 1.4: Interpolated values obtained by IDW interpolation.

- A 1^{st} order surface is described by $Z = aX + bY + z$, where X, Y are the coordinate pairs and c is a constant;
- A 2^{nd} order surface is described by $Z = aX^2 + bY^2 + cXY + dX + eY + z$;

and so on. Changing the order allows the model to better capture the complexity of the data, but using a value that is too high may result in overfitting the data, meaning that the model is too dependant on the known information and does not provide a useful prediction.

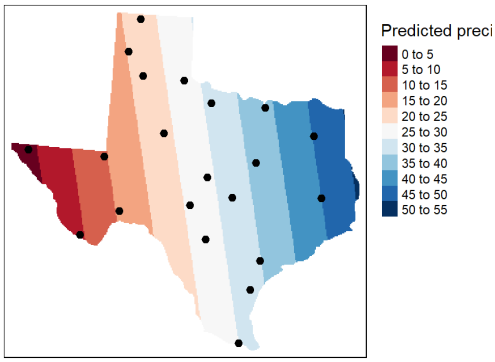


Figure 1.5: Interpolated values obtained by a 1^{st} order trend surface. The model is too rigid.

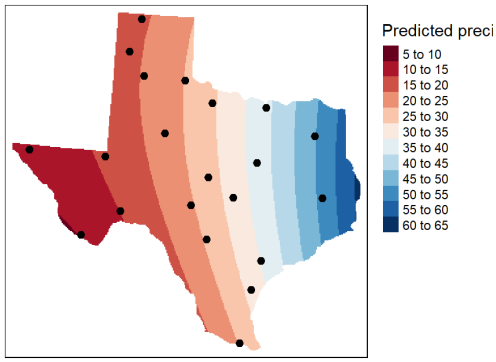


Figure 1.6: Interpolated values obtained by a 2^{nd} order trend surface. The model is better than before, but still not a good fit.

Kriging

Kriging differs from other methods in that it can assess the quality of prediction with estimated prediction errors. It assumes that the spatial variation of an attribute is neither random nor deterministic; instead, it is a combination of some spatially correlated component, a "drift" (assumed for now to be null), and a random error term.

The first step is **de-trending the data**, so that mean and variance of the data are constant across the whole study area. This can be done by using any trend model and subtracting the predictions from the data. From this point on, the method will focus on residuals, i.e., the remaining variability in te data that is not explained by the global trend. The second step is **constructing a (semi)variogram**. For each pair of points i and j, their semivariance is calculated as:

$$\gamma = \frac{(Z_i - Z_j)^2}{2}$$

The variogram is obtained as the plot of all the semivariances, using the x axis to represent distances, and the y axis to represent the semivariances. Usually, it is simplified by binning over the distance values and averaging the semivariances within each bin, obtaining a **sample experimental variogram**. The third step is to fit an **experimental variogram model** to find the parameters that best describe the spatial variance; there are many model variants, with the main ones being the Gaussian, linear, and spherical ones. Finally, **interpolation** can be performed. The general equation for estimating the residual at a point j is:

$$z_j = \sum_{i=1}^{s} z_i W_i$$

where s is the number of sample points used in the estimation, and W_i is the weight assigned to point i. These weights are derived by solving a set of simultaneous equations.

1.6.3 Spatial Regression

Regression has the goal of modeling the relationship between a target variable and a (set of) independent variable(s). The difference with interpolation is that in the latter, only the target variable data and the spatial coordinates are used, while in regression there is information about other non-spatial attributes to make predictions.

Spatial regression extends the typical regression method to include information

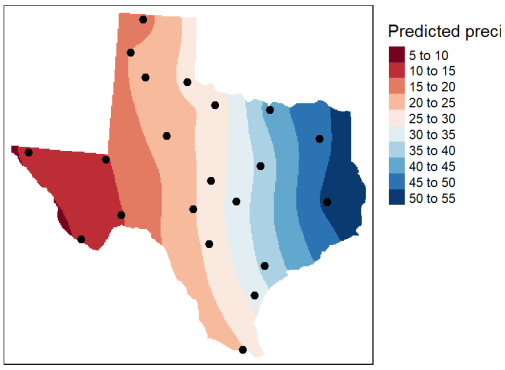


Figure 1.7: Interpolated values obtained by kriging.

about a point's neighborhood:

$$(P_i) = \underbrace{\alpha + \beta X_i + \epsilon_i}_{\text{Standard regression model}} + \underbrace{\delta \sum_j w_{ij} X_j}_{\text{Summary information about neighbors}}$$

1.7 Co-location Pattern Mining

Co-location pattern mining is the process of finding subsets of spatial features whose instances are frequently located together in space. It is similar to traditional frequent itemset mining, although defining "transactions" is not as immediate, since there is no explicit way to isolate instances into groups.

Given:

- A set of features $F = \{f_1, \ldots, f_m\}$;
- A set of instances $O = \{o_1, \ldots, o_n\}$;
- A neighbor relation R between pairs of instances, s.t. $R(o_1, o_2) \iff d(o_1, o_2) < \mathit{thresh.}$;

a **co-location pattern** $CL = \{f_1, \ldots, f_k\} \subseteq F$ is a subset of features, and an **instance of a pattern** $I = \{o_1, \ldots, o_k\} \subseteq O$ is a subset of objects s.t.:

- For each $f \in F$, there is exactly one object $o \in O$ of type f, anc vice versa;
- I forms a clique w.r.t. R (i.e., all pairs of objects are connected to each other).

A co-location pattern that contains k spatial features is called size-k co-location pattern.

To measure the prevalence of a given pattern, we use two measures; the **participation ratio** (of a feature within a pattern):

$$PR(CL, f_i) = \frac{\#|\pi_{f_i}(instances(CL))|}{\#|instances(CL)|}$$

and the **participation index** (of the entire pattern):

$$PI(CL) = \min_{i=1}^{k} PR(CL, f_i)$$

A co-location pattern is called **prevalent** if its partecipation index is greater or equal than a given threshold *minprev*. The participation index is anti-monotone, which means that the participation index of a co-location pattern is always lower or equal to that of its sub-patterns.

1.8 Trend Detection

Spatial trend analysis is the process of finding patterns of change of non-spatial attributes in the neighborhood of some object in the study area. It is analogous to trend analysis in time series, but applied to spatial data, where the linear direction of time is replaced by the many possible paths that can be formed in space. The most notable paths of interest usually start from a common location (e.g., a city center), have meaningful shape, and show a statistically significant trend.

Let g be the neighborhood graph that represents all the neighborhood relationships among objects in the study area. Let o be one such object, which is considered the starting point of paths. Let a be a subset of all non-spatial attributes, in which paths must be found in. Let t be a type of function (e.g., linear, exponential) used for regression. Then, the goal of spatial trend detection is to discover the set of all neighborhood paths in g, starting from o, having a trend of type t in attributes a, with a correlation greater or equal to a given threshold *minconf*.

To isolate these relevant paths, special filters are used to select subsets of all paths. There are many ways to define such filters; some examples are reported in Fig 1.8.

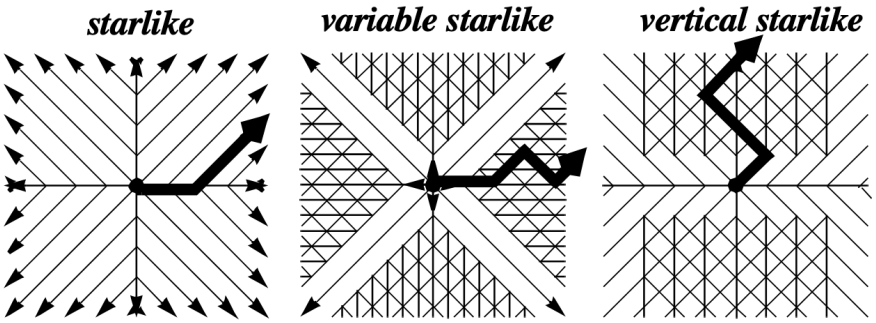


Figure 1.8: Examples of possible filters. The *stralike* filter allows paths that deviate in a specific diagonal direction; the *variable starlike* filter allows paths that can deviate diagonally multiple times; the *vertical starlike* filter restricts horizontal deviations in favor of vertical ones.

Chapter 2

Spatial and Mobility Data

The study of human mobility data has a wide range of applications, such as urban planning, epidemic control, transportation management, immigration monitoring. The last decade has seen a rapid increase of digital traces that represent human movements. Examples of this data are those generated by GPS devices embedded in phones or cars, mobile phone records collected by telco companies, and geotagged social media posts. This chapter will describe the main characteristics of such data.

2.1 GPS Data

Global Navigation Satellite Systems (**GNSS**) use satellites to provide geo-spatial positioning of objects on the Earth's surface in terms of longitude, latitude, and altitude. There are currently multiple systems managed by different countries of the world. The most famous GNSS is the US **Global Positioning System** (**GPS**), and receivers are now embedded in many commonly used tools such as smartphones, cars, and wearable devices.

GPS can be thought of a system of three main components: a space segment, a control segment, and a user segment. The **space segment** consist of a constellation of 24+ satellites orbiting around the planet, circling it twice a day in six equally spaced orbits. Each orbit has four satellites, and any user will always see at least four satellites from any point on Earth. The **control segment** consist of all the ground facilities used to track/command the satellites and monitor their transmissions. The **user segment** is represented by the GPS receivers in the user devices. On mobile phones, GPS receivers are activated by software that requires the user's position. On cars, receivers automatically turn on as the car starts. Signals are sent to a server every few seconds up to every few minutes; their precision can range from a few centimeters to meters, depending on the quality of the device.

To identify a point on Earth, a total of four satellites are used (three to identify the point, a fourth one for redundancy): each satellite broadcasts a radio signal at the speed of light that contains information about its location, status, and time of the on-board atomic clock. When the signal hits the device receiver, the time of arrival is compared to that of the send to calculate the distance d from the satellite.

Using this distance, we can imagine a sphere of radius d centered around the satellite. The intersection of all the spheres centered around the involved satellites is the exact location of the device.
The GPS traces format is as follows:

$$(u, lat, lng, alt, t)$$

where:

- u is the user/receiver ID;
- lat is the latitude;
- lng is the longitude;
- alt is the altitude;
- t is the timestamp.

Pros Ubiquitous, produces high resolution, dense traces.

Cons Rarely publicly available, no semantic information (e.g., what the user is doing, if he/she is in a building or outside), prone to errors when signal is noisy or absent, and small samples.

2.2 Mobile Phone Records

Mobile phone coverage reaches almost 100% of the population of most countries. Telco companies collect the activity of phone users for billing purposes, storing information about where users are located, when they communicate with others, and who they communicate with. This type of information can be split in the following three groups.

Call Detail Records CDR are generated each time a user makes/receives a call or an SMS. A CDR is a tuple

$$(n_o, n_i, t, A_o, A_i, d)$$

where:

- n_o is the caller's ID (can be the phone number, but it should be anonymized for privacy reasons);
- n_i is the receiver's ID (same as above);
- t is the timestamo if the call;
- A_o is the ID of the antenna the caller is connected to;
- A_i is the ID of the antenna the receiver is connected to;
- d is the duration of the call.

eXtended Detail Records XDR are generated when a device uploads or downloads data from the Internet. These connections can be both human-triggered and device-triggered (e.g., software updates, background synchronization). An XDR is a tuple

$$(n, t, A, k)$$

where:

- n is the user's ID;
- t is the timestamp of the connection;
- A is the ID of the antenna the user is connected to;
- k is the amount of KB of data transferred.

Control Plane Records CPR are used to check the status of the network, and are generated each time a device connects to a new antenna. These records are exclusively network-triggered. A CPR is a tuple

$$(n, t, A, e_1, e_2, \ldots, e_n)$$

where

- n is the user's ID;
- t is the timestamp of the connection;
- A is the ID of the antenna the user is connected to;
- e_i are the "events" that happen in the network.

Pros Ubiquitous, huge sample size, rich and multidimensional (give information about space, time, and social connections among users, since we can know who calls/messages who).

Cons Not publicly available, much more sparse and low resolution than GPS, suffers from ping-pong effect (since areas of influence of antennas are not perfectly regular and may overlap, a device may rapidly flip between them).

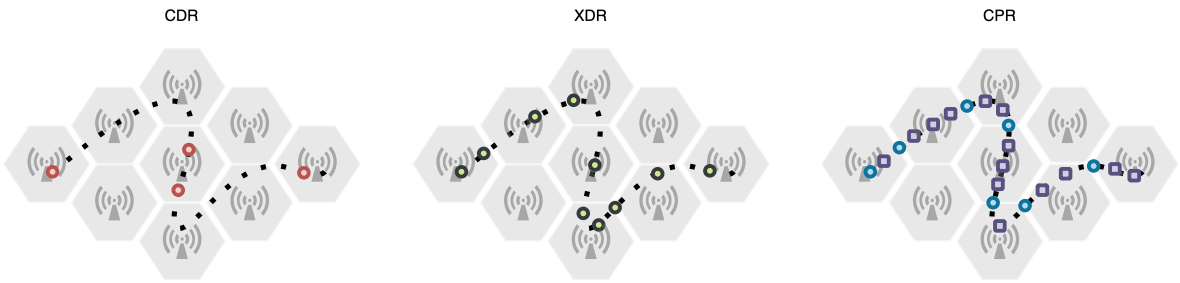


Figure 2.1: Comparison of densities of different data types.

2.3 Location-based Social Networks

Certain social media platforms allow users to geotag their posts, i.e., assign geographic and temporal information. This information can be represented by absolute data (latitude, longitude), relative data (distance from some point of interest), or symbolic (home, office, stadium, shopping mall). This geospatial information can then be easily coupled with the social network structures of connected users, allowing us to consider the social dimension in our analysis. A LBSN record is a tuple

$$(u, t, lid, lpos, post)$$

where:

- u is the user's ID;
- t is the timestamp of the post;
- lid is the location ID;
- $lpos$ is the location position;
- $post$ is the content of the text post.

Pros Ubiquitous, can provide precise information with little error, carries semantic information.

Cons Is very sparse (even more than mobile phone records), suffers from self-selection bias (users that frequently produce such content tend to belong to a specific demographic).

2.4 Road Networks and Points of Interest

Road networks describe streets and their intersections. They are usually represented by multigraphs, where a node is an intersection and an edge is a road. Nodes and edges

are geometries, encoded as Point, Lines, or Polygons. A road network is described by a two sets of records: the first contains all the nodes, the second all the edges. A node record is a tuple

$$(id, lat, lng, streets, geom)$$

where:

- id is the node ID;
- lat, lng are the geographic coordinates;
- $streets$ number of streets crossing in that intersection;
- $geom$ is the geometric shape of the node (usually a Point).

An edge record is a tuple

$$(id, u, v, f_1, \ldots, f_k, geom)$$

where:

- id is the edge ID;
- u is the origin node;
- v is the destination node;
- $f_i, \ldots, f_k$ are the road characteristics (e.g., if the road is one-way only, its length, number of lanes, name, speed limit);
- $geom$ is the geometric shape of the edge (usually a Line).

Road networks help associate GSP traces to actual streets, and can find routes between two places. These structuresa are also aften used in simulations for traffic management and urban planning.

Points of interest describe certain geospatial entities in a city, such as grocery stores, transit stops, restaurants, etc. They carry semantic information about activities in cities and associated human behaviour. A POI record is a tuple

$$(id, f_1, \ldots, f_n, geom)$$

where:

- id is the POI ID;
- $f_i, \ldots, f_n$ are the POI characteristics (e.g., name, type of building, road nodes involved);
- $geom$ is the geometric shape of the POI (usually a Point or a Polygon).

Chapter 3

Data Preprocessing

Data that represents trajectories of moving entities is often affected by errors. Their presence can greatly influence the result of analysis, so it is important to identify and remove them. Additionally, datasets tend to contain huge amounts of objects, meaning that the time and space complexity that may bee too high. Finally, individual trajectories that are too long and complex may cause the failure of data mining algorithms due to the inability of extracting details from them. Summarizing the information carried by trajectories via a preprocessing phase is a necessary step to make sure subsequent analysis can be carried out efficiently and effectively.

Preprocessing techniques can be divided into two families: context-based filtering, and movement-based filtering. **Context-based filtering** relies on contextual geographical information provided by the environment, such as road networks, to remove points that are deemed to be errors. These methods may not always be effective, since they assume that trajectories must always be on a road (or adapt in some way to the available data); this may not always be true, for example for vehicles moving across the sea or open fields, or for pedestrians. **Movement-based filtering** uses only the geometry and dynamics of the movement described by the trajectory to find points that do not match the expected behaviour of the moving entity.

3.1 Movement-based Filtering

3.1.1 Speed-based Filtering

This filtering is based on the idea that tha speed of the moving entity must always be within a certain range of allowed values, without any abrupt change. Initially, the first point of the trajectory is found and set as valid; then, each subsequent point is scanned in order according to the timestamp, and the speed v between the current and

the previous point is calculated:

- If v is larger than a given threshold, the point is marked as invalid and removed from the trajectory;
- If v is within the threshold, the point is marked as valid and kept.

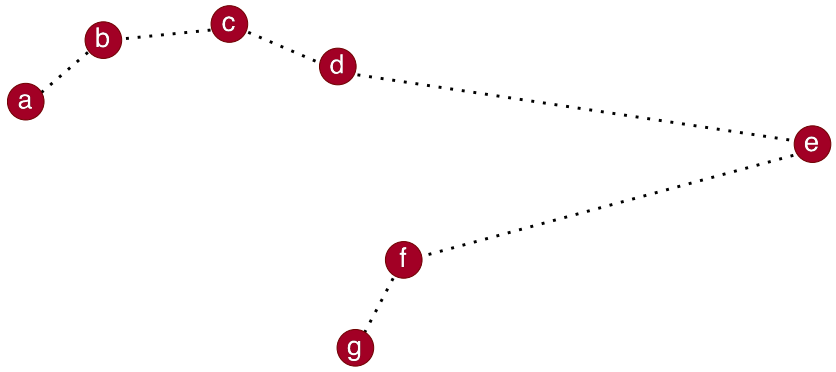


Figure 3.1: Example of speed-based filtering: point e is removed from the trajectory as it would imply a speed higher than the threshold.

3.2 Context-based Filtering

3.2.1 Point Map Matching

Map matching is the procedure of determining which road on a map a vehicle is using, given a set of GPS points. This approach both filters out noise and outliers, and adds new information to the dataset by associating each point with an identifier of the road it is matched to. Once points have been mapped to a specific road, any point farther than a certain threshold can be removed as noise.

The main issues of this approaches are:

- Projecting individual points requires a comparison for each point and each road segment, which can be computationally expensive;
- In some cases, the same point could be potentially assigned to multiple road segments, but limiting ourselves to consider the closest one may not always be the best choice;
- Matching points individually can lead to inconsistencies, especially when the road network is very dense.

3.2.2 Route Reconstruction and Trajectory Map Matching

Route Reconstruction Route reconstruction tries to infer the path taken by a moving entity by connecting the available points and possibly creating new ones; this technique is useful in cases where data is generally sparse or has large missing gaps. Usually, we assume the entity always takes the shortest (or best) path between two consecutive points; if an area allows free movement, the entity is assumed to move in a straight line. If there are limitations, such as the presence of buildings, the path must adapt to them. Since we often have attributes describing the characteristics of roads, the optimal path can be found by trying to maximize or minimize certain features: the total path duration in time, fuel consumption, CO_2 emissions, etc.

Common algorithms used for route reconstruction are **Dijkstra's minimum cost path algorithm**, which is efficient and can handle cycles, although it requires non-negative costs, and **Bellman-Ford's algorithm**, which is less efficient and can't handle cycles, but allows negative costs.

Trajectory map matching This is a more advanced version of point map matching that takes as input a whole trajectory (wheter it comes from raw data or from a reconstruction step), and adapts it to a road network considering spatial, temporal, and contextual constraints for the whole path. Common approaches are shortest-path based, and probability-based.

Shortest-path based Map Matching starts by matching only the first and last point of the path to the road segments, as in point map matching, and then finding the optimal path that connects them. A threshold is set as the maximum distance allowed between a segment and a GPS point for the two to be considered matching; using this threshold, **cutting points** are found. A cutting point can be of two types: the first type is the point that is the farthest from the current optimal path, and the second type are the points whose distance fall within the threshold. If a cutting point of the first type is found, the path is split in two, and a new optimal path that passes by that point is found. These steps run recursively on each half until all the points are reasonably close to the optimal path.

Probability-based Map Matching considers the probability that an entity may move from one road segment to another. State-of-the-art implementations use Hidden Markov Models to model the trajectory as a sequence of state transitions.

3.3 Trajectory Simplification

Many algorithms on trajectories tend to be very expensive, and their complexity is tied to the number of points in the dataset. Trajectory simplification is used to reduce the number of points without affecting the quality of results. Typical cases in which simplification can be used are for straight line movement (the entity keeps moving in a straight line, so the whole trajectory/subtrajectory can be represented as a single segment), or negligible movement (the entity stays still for some time, accumulating GPS traces in the same spot). Some standard methods for polygonal curve simplification include Ramer-Douglas-Peucker, and Driemel-HarPeled-Wenk.

Ramer-Douglas-Peucker is one of the most successful simplification algorithms, used in many fields other than GIS, such as computer vision and pattern recognition. A pseudocode of the algorithm is found below.

Algorithm 1 Ramer-Douglas-Peucker pseudocode.

```
1: function RamerDouglasPeucker(P, ε, p_i, p_j)        ▷ P = ⟨p_1, ..., p_n⟩
2:     Find vertex p_f ∈ ⟨p_i, ..., p_j⟩ farthest from the line p_i p_j.
3:     dist ← distance between p_f and p_i p_j.
4:     if dist < ε then
5:         RamerDouglasPeucker(P, p_i, p_f)
6:         RamerDouglasPeucker(P, p_f, p_j)
7:     end if
8:     Return line p_i p_j.
9: end function
```

The time complexity is $T(n) = O(n) + T(n-1) = O(n^2)$ in the worst case, but it is usually much faster in practice.

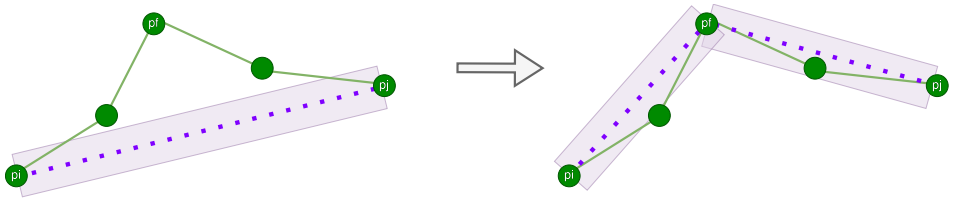


Figure 3.2: Example of one step Ramer-Douglas-Peucker simplification.

Driemel-HarPeled-Wenk is not recursive, unlike the previous, but instead simplifies the line starting from the very first point and iteratively removes points that fall within a certain radius from the current one, substituting all of them with a single line

that goes from the current point to the first point in the trajectory more distant than that radius. The pseudocode is below.

Algorithm 2 Driemel-HarPeled-Wenk pseudocode.

```
1: function DriemelHarPeledWenk(P, ε)                    ▷ P = ⟨p_1, ..., p_n⟩
2:     P' ← p_1.
3:     i ← 1.
4:     while i ≤ n do
5:         curr ← p_i.
6:         j ← i + 1.
7:         while dist(p_i, p_j ≤ ε) do
8:             j ← j + 1.
9:             if j > n then
10:                i ← n.
11:                Break.
12:            end if
13:        end while
14:        Add p_j to P'.
15:    end while
16:    Return P'.
17: end function
```

Either way, simplification has the side effect of modifying average speed estimates, as simpler, straighter paths imply a slower average speed compared to more complex, chaotic ones.

3.4 Stop Detection for Trajectory Segmentation

GPS data is normally formed by a continuous stream of points, but there is no explicit separation between the trajectories of the same entity. **Stop detection** is a technique used to identify moments in which the entity is staying still (or moving very slowly) to be used as endpoints of different trajectories. For example, if we consider a person moving through a regular work day, we may observe that person move away from a spot (likely his/her house), then stop at what is either school or work, then make another stop at the gym, and finally return back home. Every movement between each of these stops can be isolated as its own trajectory.

The general criteria is: the estimated speed from a set of points is very low (i.e., very little movement over a long time interval), the set of points represent a stop.

The sequence of points between stops is a trajectory. To check if a point belongs to a stop, typical space and time thresholds are $Th_S \in [50, 250]$ (meters), and $Th_T \in [1, 20]$ (minutes), but depending on the context, these values may change: thresholds for a car are different from those for a pedestrian, for example.

When it comes to vehicles specifically, stops can often be detected by simply checking when the engine is on or off. This is generally good, but may not detect stops that are very short. A more general approach is to use a density-based approach: the faster an entity is, the less of its own points are found around it as it is moving.

After trajectory segmentation has been performed, we can use them to study the behaviour of the individual, and eventually find deviations from the norm. A way to represent this information is building the entity's **Individual Mobility Graph**, where each location is a node and each movement is an edge; this means that for each trajectory, only the starting and ending locations are recorded in the graph. Naturally, denser areas with more nodes will represent important places. These locations are identified using a clustering algorithm, and are each associated with their frequency; trips between points belonging to the same cluster (location) are then aggregated as edges between the entire clusters. The basic assumption is that the location with the highest frequency is home, the second most frequent location is work, and so on. An alternative approach also considers the actual time of the day the traces were recorded at: points recorded during nighttime or early morning are more likely to be home, while those recorded during the morning and early afternoon are more likely to be work.

Other than GPS, phone data can also be used to infer important locations. By analyzing the calls and messages sent or received by the user, we can identify what are called **personal anchor points**, which are the most frequently contacted antennas. These points can also be labeled as home/work by considering the time of the day the calls were made or received.

3.5 Activity Labeling

After locations have been identified, an ulterior step is to assign to each of them a specific activity the user carries out once it arrives there. The first step is to assign a location to a set of actual point of interest that are close by. The POIs are then also associated with an activity, for example: "eating out", "shopping", "working", "education". At the end, the available data will be:

- For each POI, the category, the opening hours, the location;
- For each trajectory, the stop location, the stop duration, the time of day;

- For each user, the max walking distance (as in, how far the person is likely to walk after parking).

Each POI is associated to a certain probability of being the reason for the stop, calculated as a function of the information above. Finally, a stop is annotated with the activities and relative probabilities of the POIs close by.

Chapter 4

Individual and collective human mobility

Human mobility refers to the movement in space and time of human beings. Individual mobility is the movement of a single person and is represented by trajectories, while collective mobility is the movement of groups of people, and is represented by flows. In this chapter, a few basic concepts about human mobility are introduced: the laws of individual human mobility, abd the techniques used to model it.

4.1 Individual Human Mobility

4.1.1 Metrics

Travel Distance (Jump Length) **Travel distance** is the distance between two consecutive locations visited by the individual. Travel distance may also be called in different ways depending on the context it is used in; the term **"jump length"** is is used to refer to different appearances of the same object in different locations, without necessarily implying a direct movement between them. An early example of mobility analysis involved information about the spatial trajectories of bank-notes. The data was sourced from a US-based website where users could input the identifier of their dollar bills, specifying where they are located. In this sense, the distance between two different locations in which a bank-note appears in is a jump length.

Regardless of how it is defined, distance, denoted as Δr, is calculated as

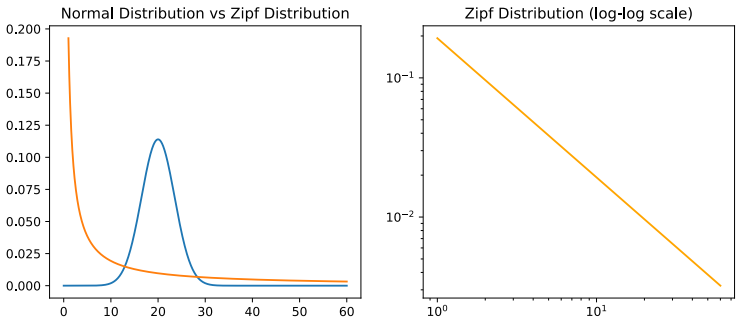
$$\Delta r = \|r(t) - r(t + dt)\|_2$$

i.e., the Euclidean distance between the locations recorded at intervals t and $t + dt$. It is interesting to study the distribution of Δr within a population, approximating the

probability distribution function of distances/jump lengths, $P(\Delta r)$. Empirical studies observed how this distribution follows a *power-law*: a scale-free distribution (i.e., with no/meaningless mean) with a heavy tail:

$$P(r) \sim r^{-(1+\beta)}$$

where β is the scaling exponent, which is chosen depending on the context.



Power law behaviour has been observed in the trajectories of mobile phone users as well, specifically following a truncated power law:

$$P(r) = (r + r_0)^{-\beta} \exp(-r/\kappa)$$

where r_0 and κ are the cutoffs of smaller and larger values of Δr.

What this means is that there are a lot of individuals who tend to move short distances; as the distance increases, the number of individual sharply decreases. However, there are still a few individuals who move very long distances (the distribution is not exponential).

Radius of Gyration Humans tend to habitually move a characteristic distance away from their starting locations. This distance can be quantified by the radius of gyration, r_g, defined as the root mean square distance of a set of points:

$$r_g = \sqrt{\frac{\sum_{i=1}^{N} \|r_i - r_{cm}\|^2}{N}}$$

where r_i are the coordinates of the N individual locations the person visited, and r_{cm} is the center of mass of the set of points (their centroid):

$$r_{cm} = \frac{\sum_{i=1}^{N} r_i}{N} \tag{4.1}$$

It was observed that the distribution of the radii of gyration of a population of individuals also follows a truncated power law.

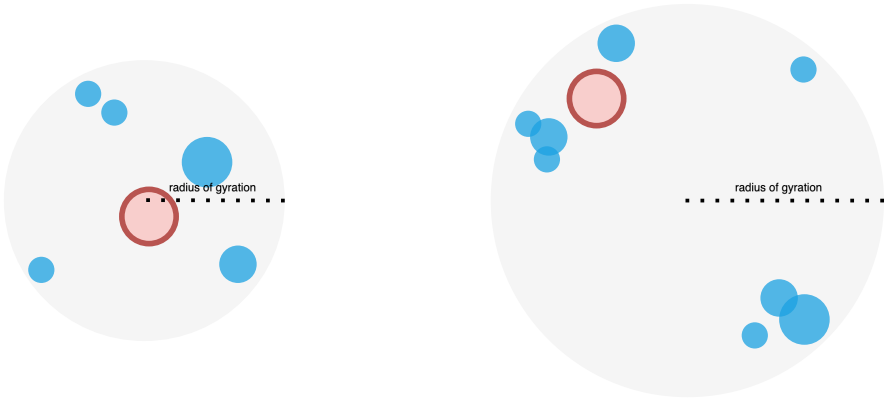


Figure 4.1: Radii of gyration of two different individuals. The red circle indicates their house, while the blue circles are the locations they visited.

What can be observed from data is that people with a small radius of gyration tend to travel mostly over small distances, while those with a large radius tend to have a combination of a lot of short travels and a few long ones.

An interesting aspect to consider is the phenomenon of **saturation**, observed during the temporal evolution of r_g; the average r_g of a person shows a logaritmic increase with time, which can be attributed to the fact that people tend to follow regular travel patterns and after a certain amount of time, they have visited all the places they usually go to, leading to a stable r_g.

By itself, the radius of gyration does not carry any information about the relevance of a certain location in an individual's movements. An individual who spends most of his/her time in his/her most visited locations (e.g., home and work) will have a large r_g if the two are far away; if instead they are close to each other, the r_g may still be large, since there are likely other longer movements that contribute to the radius of gyration. To address this issue, one can consider the frequency of visits to a location, ranking them accordingly, and then calculate the **k-radius of gyration**:

$$r_g^{(k)} = \sqrt{\frac{\sum_{j=1}^{k} n_j (r_j - r_{cm^{(k)}})^2}{N_k}}$$

where N_k is the number of visits to the k most frequented locations, $r_{\text{cm}}^{(k)}$ is the centroid of those top k locations, and n_j is the number of visits to the j^{th} location. The frequency of visits also follows a power law distribution, since people will devote most of their time to few places (home, work, school, etc.), and less and less time to others.

The analysis of CDR and GPS traces revealed that there are two distinct groups of individuals: the *k-returners*, whose radius of gyration is well approximated by their k-radius of gyration for $k \geq 2$, and the *k-explorers*, whose k-radius is very small compared to their overall r_g.

4.1.2 Predictability

A way to represent the movements of an individual is by constructing its **individual mobility network**, i.e., a graph where each node is a visited location and an edge is a movement between them. A measure that can be calculated on this network to quantify how predictable the movements are is **entropy**:

$$S^{rand} = \log_2 N \qquad \text{(Random entropy)}$$

$$S^{unc} = -\sum_{i=1}^{n} p_i \log_2 p_i \qquad \text{(Uncorrelated entropy)}$$

$$S = -\sum_{T' \subset T} p_{T'} \log_2 p_{T'} \qquad \text{(Real entropy)}$$

Random entropy assumes that each location is visited with equal probability. (Temporal) Uncorrelated entropy assumes heterogeneous visitation patterns, associating each term in the sum with the probability it was visited by the user. Real entropy depends not only on the frequency of visitation, but also on the order in which nodes were visited and the time spent at each of them: $T = \{X_1, \ldots, X_n\}$ is a sequence of locations at which the user is observed at a certain time interval.

4.1.3 Modeling

The **Exponential-preferential return** (**EPR**) model is based on two key mechanisms observed in human trajectories: *exploration* and *preferential return*. Most movements tend to frequently return to a few previously known locations (*preferential return*), but every once in a while, the individual may decide to explore a new location (*exploration*). This method models both the distance and the wait time between subsequent visits to the same location as a power law with exponential cutoff.

The steps of the algorithm that generates the trajectories are the following:

1. The individual starts at a random location, and the location's visit counter is initialized to 1. Since it's the initial point, its time of visit is set to 0.

2. The individual *explores* a new location. First, the distance between the current location and the new one is extracted from a power law. Then, a random location is chosen at that distance: its visit counter is initialized to 1, while its time of visit is set to the previous location's visit time plus some value extracted from a power law (where small wait times are much more likely than large wait times).

3. After the new location has been added to the trajectory, the phase is randomly chosen between *exploration* and *return*.

4. If the phase is *exploration*, another distance is chosen from a power law and a new location is added to the trajectory, again setting its counter to 1 and its visit time to the previous location's time plus a value extracted from a power law.

5. If the phase is *return*, a previously visited location is chosen; the probability that a location is chosen as the return is proportional to the number of times it has already been visited in the past. Once visited, the location's visit time is incremented by 1 and its visit time is updated as before.

6. Steps 3-5 are repeated until the trajectory reaches the desired length.

A variant of this model has been developed to also have a *recency* return phase, where the return is not proportional to the number of previous visits, but to the visit time. When a return phase is chosen, the algorithm decides between a frequency based return with probability α, or a recency based return with probability $1 - \alpha$, where α is chosen from empirical data.

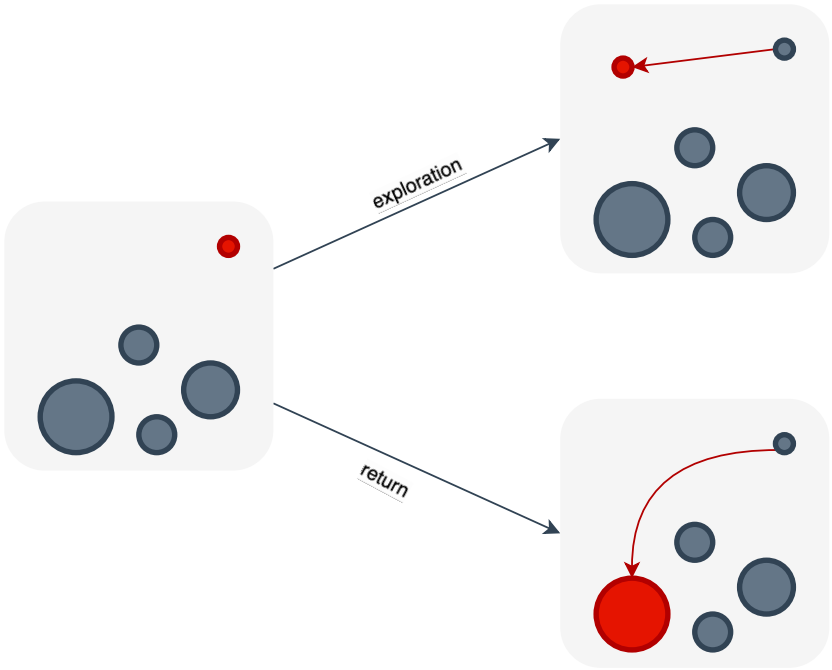


Figure 4.2: Example of the two phases of the EPR model: the red dot is the current location, while the gray dots are previously visited locations. The size of the dots is proportional to the number of visits.

4.2 Collective Human Mobility

Mobility information of individuals can be aggregated to study flows of individuals traveling from one location to another. These flows can be organized into a **Origin-Destination** (**OD**) matrix: each cell of indices i, j contains the number of trips that start in the i^{th} location and arrive in the j^{th} location, quantifying the flow T_{ij} of people between those two extremes. Locations are typically defined as tessellations of the space with a certain granularity, such as provinces, cities, neighborhoods, regions, states, etc. The OD matrix normally excludes self-loops, i.e., trips that start and end in the same location. From this matrix, we can derive the following:

- **Total out-flow** from location i:

$$O_i = \sum_j T_{ij}$$

- **Total in-flow** to location j:

$$D_j = \sum_i T_{ij}$$

- **Total flow**:

$$N = \sum_{ij} T_{ij}$$

There exist different models that can be used to infer/generate flows from OD matrices. The problem can be interpreted as a classification task, where locations are the classes: given a starting point, we want to predict which location is goind to be the ending point of the trip. Each model will estimate this probability from a different set of variables that characterize the locations, with the goal of maximizing the likelihood of the observed data. Models may also be **constrained** or **uncostrained**, imposing certain conditions on the flow matrix:

$$\sum_{ij} T_{ij} = N \qquad \text{(Unconstrained/ globally constrained)}$$

$$\sum_j T_{ij} = O_i \ \forall i \qquad \text{(Origin constrained)}$$

$$\sum_i T_{ij} = D_i \ \forall j \qquad \text{(Destination constrained)}$$

$$\sum_j T_{ij} = O_i \ , \ \sum_i T_{ij} = D_i \qquad \text{(Doubly constrained)}$$

Over time, two main schools of though have emerged: the first one assumes that the number of trips between two locations is inversely proportional to their distance

(**gravity model**), while the other instead focuses on **intervening opportunities**, meaning that locations with a lot of opportunities of some kind (e.g., jobs, schooling, services, etc.) will attract more people.

4.2.1 Gravity Model

In the gravity model, the way mobility flows are calculated is inspired by Newton's law of universal gravitation:

$$T_{ij} \propto \frac{P_i P_j}{r_{ij}}$$

where P_i and P_j are the populations of locations i and j, and r_{ij} is the distance between them. This idea is generalized in the following equation:

$$T_{ij} = K m_i m_j f(r_{ij})$$

where K is a constant, m_i and m_j are the masses of the two locations, which relate to the number of trips leaving i and arriving in j respectively, and $f(r_{ij})$ is the **friction factor**, a decreasing function of the distance (usually a power law or exponential function). The masses are typically modeled as:

$$m_i = P_i^{\alpha} \qquad\qquad m_j = P_j^{\beta}$$

where α and β are the free parameters of the model.

This model is very popular, especially for transport planning and spatial economics. Still, it grossly simplifies flows and may not be able to capture empirical observations. Since it is trained on a dataset to adapt its parameters, it is also rather sensitive to fluctuations or incomplete data.

Some issues can be solved using constrained versions of the model. The number of people arriving to or coming from a location can be fixed to a known quantity, and the model can be used to esitmate the other end of the flow:

$$T_{ij} = K_i O_i m_j f(r_{ij}) \qquad K_i = \frac{1}{\sum_k m_k f(r_{ik})}$$

$$T_{ij} = K_j m_i D_j f(r_{ij}) \qquad K_j = \frac{1}{\sum_k m_k f(r_{kj})}$$

Note that there is a different proportionality constant for each origin/destination. Alternatively, both origin and destination can be doubly constrained:

$$T_{ij} = K_i O_i L_j D_j f(r_{ij}) \qquad K_i = \frac{1}{\sum_j L_j D_j f(r_{ij})}, \quad L_j = \frac{1}{\sum_i K_i O_i f(r_{ij})}$$

Choosing the right constrained or unconstrained model depends on the information available, and on the objective: if the goal is to approximate the flows from indirect socio-economic variables, unconstrained models are preferable; if instead we have outgoing and/or in-going flows available, and the goal is to estimate the values of an OD matrix, constrained models are appropriate.

4.2.2 Intervening Opportunities Model

This model is based on the idea that movements across locations are not simply controlled by distance, but by th number of opportunities available. In the original paper this approach was first presented in, the author did not provide a precise definition of "opportunity", but it can be interpreted differently depending on the context.

The assumption is that, given a starting point, the likelihood that a certain location will be the destination of a trip depends on two factors: the **opportunities** represented by the destination, and the number of **intervening opportunities**, i.e., alternative potential destination that are closer to the starting point but are rejected by the trip maker. The probability that a trip will end in a certain location is then modeled as equal to the probability that it offers an acceptable opportunity, multiplied by the probability that another acceptable opportunity in a closer location has not been chosen. The flow between two locations is:

$$T_{ij} = O_i \frac{e^{-LV_{ij-1}} - e^{-LV_{ij}}}{1 - e^{-LV_{in}}}$$

O_i is the number of trips originating from i; V_{ij} is the cumulative number of opportunities up to the j^{th} location, in order of distance from the origin i, and L is a hyperparameter adjusted on empirical data to be equal to the probability of accepting an opportunity destination. The denominator is a normalization factor that ensures that the probabilities sum to 1 (n is the total number of locations in the region considered).

4.2.3 Radiation Model

The radiation model elaborates the hypothesis of intervening opportunities, and assumes that the choice of a traveler's destination consists in two steps:

- Each opportunity in every location of the region considered is assigned a fitness number z, extracted from a distribution $p(z)$, whose value represents how good the opportunity is for the traveler.
- The traveler ranks all opportunities according to their distance from the origin location, and chooses the closest destination with a fitness number higher than some threshold (which is also a random number extracted from $p(z)$).

The average size of the flow between two location is then estimated as:

$$T_{ij} = O_i \frac{1}{1 - \frac{m_i}{M}} \frac{m_i m_j}{(m_i + s_{ij})(m_i + m_j + s_{ij})}$$

O_i is again the number of trips originating from i, m_i and m_j are the number of opportunities at the origin and the destination respectively, s_{ij} is the number of opportunities in a circle of radius r_{ij} centered in i (excluding source and destination). The second term is for normalization, and M is the sum of all the opportunities in the region.

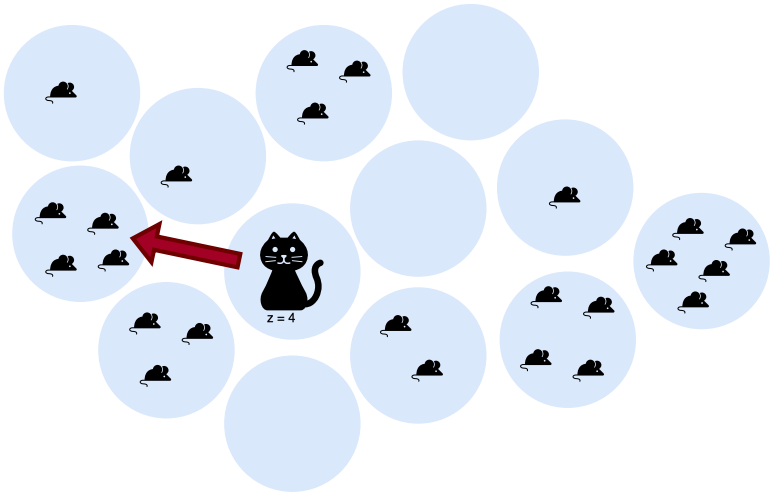


Figure 4.3: Example of how an individual chooses its destination in the radiation model. The cat has a threshold of $z = 4$, so it chooses the closest location with a fitness number equal or higher than 4.

This model is parameter-free, so it is immediately adapted to the available data qith no training phase; in particular, the flows do not depend from the distribution $p(z)$ of the fitness of the locations. This can also be a disadvantage, however, as the model does not seem to be very robust with changes in the spatial scale. An extended version has been proposed that also considers the spatial distribution of opportunities, and introduces a parameter α that regulates the effect of the number of intervening opportunities.

4.2.4 Other Models

Most other models developed for the task are variants of the previous three:

- **Rank-distance model**: the probability of traveling from a location to the other

is inversely proportional to the rank of the destination:

$$p_{ij} \propto \frac{1}{(m_i + s_{ij})^{\alpha}} = \frac{1}{rank_{ij}^{\alpha}}$$

- **Population-weighted opportunities model**: it considers the opportunities centered at the destination instead of the origin:

$$p_{ij} \propto m_j \left(\frac{1}{m_i + m_j + s_{ji}} - \frac{1}{M} \right)$$

- **Deep gravity model**: based on deep neural networks, it is capable of capturing non-linear relationships between the input and output variables. Locations are also better characterized and enriched with data collected from external sources (e.g., land use, transportation networks, POIs). It also uses explainable AI techniques to interpret the results of the training, and find the most influential variables.

  The input layer of the network receives all the input features (of the origin, of the destination, and their distance), concatenates them, and passes them to the hidden layers. Each destination is assigned a score using the softmax function on the output layer.

  Other models also use deep learning, such as MoGAN (Generative Adversarial Networks) and LLM-based approaches.

4.2.5 Comparison between models

All the models presented until now will generate values for an Origin-Destination matrix. There exist different ways to evaluate and compare their performance:

- **Common part of commuters** (**CPC**):

$$\text{CPC} = \frac{\sum_{i,j} \min(T_{ij}^m, T_{ij}^e)}{\sum_{ij} T_{ij}^e}$$

  T_{ij}^m is the model's prediction, T_{ij}^e is the empirical value;

- Root mean square error;

- Cosine similarity;

- Correlation coefficients.

Chapter 5

Mobility Patterns

Mobility patterns are an abstraction of the movement of individuals or groups of individuals that show their cumulative behaviour. They can be divided into **global patterns**, which describe the movement of sets of individuals, and **local patterns**, which refer to the movement of a single individual. This chapter will show various methods to extract both types of patterns from mobility data.

5.1 Global Patterns

A simple way to extract global patterns is to use a clustering algorithm on trajectory data. However, the choice of clustering algorithm and distance measure is crucial, since trajectories cannot be treated as regular tabular data. Also, interpreting clusters as patterns is not immediate: if we, for example, ran k-means on a trajectory dataset, would the centroids of the clusters represent meaningful patterns?

5.1.1 Distances

First, let's look at some distance measures. Trajectory distances can be divided in three categories, depending on how the trajectories themselves are treated:

- Those who treat trajectories as **sets** of points, such as Hausdorff distance;
- Those who treat trajectories as **sequences** of points, such as Fréchet distance;
- Those who treat trajectories as **time-stamped sequences** of points, such as the Average Euclidean distance.

As sets Two distances of this type are **common destination** and **common origin** distances, which both reduce a trajectory to its end/start point, respectively. The distance between a pair of trajectories is defined as the distance (e.g., Euclidean) between their end/start point only.

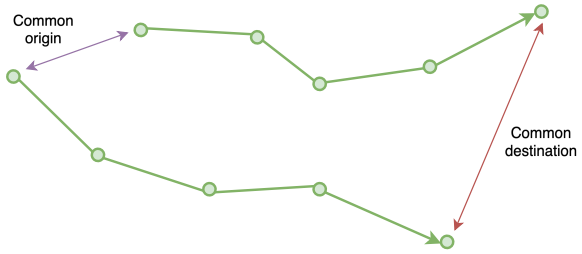


Figure 5.1: Common origin vs Common destination distances.

Hausdorff distance measures the maximum distance of all distance values from a point of one trajectory T_1 to the nearest in another trajectory T_2. Another way to interpret Hausdorff distance is to consider it the smallest buffer to build around T_1 to include all points of T_2.

$$\text{Hausdorff}(T_1, T_2) = \sup_i \inf_j d(T_1(i), T_2(j))$$

$$\text{symm. Hausdorff}(T_1, T_2) = \max\{\sup_i \inf_j d(T_1(i), T_2(j)), \sup_j \inf_i d(T_1(i), T_2(j))\}$$

However, when applied to trajectories, Hausdorff can yield counter-intuitive results; for example, when the points of the two trajectories are spatially close to each other, but they make different detours, such as in the figure below.

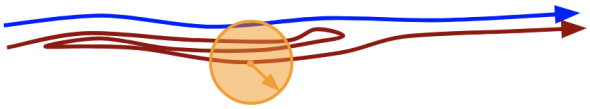


Figure 5.2: Hausdorff distance between two trajectories, visualized in orange. Even though they are very different, their distance is misleadingly small.

As sequences **Frèchet distance** can be seen as the equivalent of Dynamic Time Warping but applied to continuous curves. Formally, it is calculated as:

$$\text{Frechet}(T_1, T_2) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \{d(T_1(\alpha(t)), T_2(\beta(t)))\}$$

where α, β are non-decreasing mappings from $[0, 1]$ to the points along T_1 and T_2 in forward order. In other words, it takes in consideration the direction of the trajectories,

and in which order their points appear in each. Dynamic Time Warping and Edit Distance with real values can also be used on trajectories. It is important to note that most of these methods assume constant sampling rates.

As time-stamped sequences **Average Euclidean distance** transforms trajectories into continuous spatio-temporal curves via linear interpolation of pairs of subsequent points. The distance between two trajectories is calculated as:

$$D(T_1, T_2)|_T = \frac{\int_T d(T_1(t), T_2(t))dt}{|T|}$$

where T is the time interval over which the trajectories have been recorded. This distance only considers pairs of contemporary points, and does not use any matching/shifting of subsequences to align the two trajectories.

5.1.2 Algorithms

In principle, any distance-based clustering can be used, with the following requirements: it must be able to identify non-globular clusters, it must tolerate noisy data, it must be computationally cheap, and can be applied to structured data. A common choice is to use density-based clustering algorithms such as DBSCAN or OPTICS. These algorithms label points as either core points (on the inside of a cluster), border points (on the edge of a cluster), or noise points (not belonging to any cluster) on the basis of two hyperparameters, *eps* and *minPts*. A core point has *minPts* points within *eps* distance, a border point in whithin *eps* distance to a core one but has less than *minPts* points in its neighborhood, and finally noise points are those who are too far away to be connected to any core point. After the labeling, clusters can be formed by connecting core points and border points close to each other in a single cluster.

5.2 Local Patterns

A simple way to extract local patterns is to transform the trajectories in the dataset to a sequence of areas, building a tessellation over the area of interest and assigning each point of a trajectory to the area it falls in; the resulting sequences can be then used as input for sequential pattern mining algorithms that will identify frequent subsequences. The results are influenced by how the tessellation is constructed and how fine it is.

Trajectory flocks are groups of objects that move together close to each other for a certain time interval. A flock can be formally defined as follows: given a set of n trajectories of entities, a flock in a time interval I, whose is at least k, consists of at

least m entities such that for every point in time $t \in I$ there is a disk of radius r that contains all m entities. m, k, and r are hyperparameters that can be tuned to control how flocks are extracted.

Convoys are similar to flocks, but to count as a convoy, entities must be density-connected. Formally: given a set of trajectories of entities, a distance threshold r, a time threshold k, and a size m, a convoy is a set of m entities that are found to be density-connected for at least k consecutive time points w.r.t. r and m. Convoys can capture groups of connected entities of irregular shapes. **Swarms** are defined similarly to convoys, but do not require the entities to appear in k consecutive time points: the total number of times in which the entities are connected must be at least k, but an entity may disappear for a few time points and then reunite with the rest while still counting as part of the swarm.

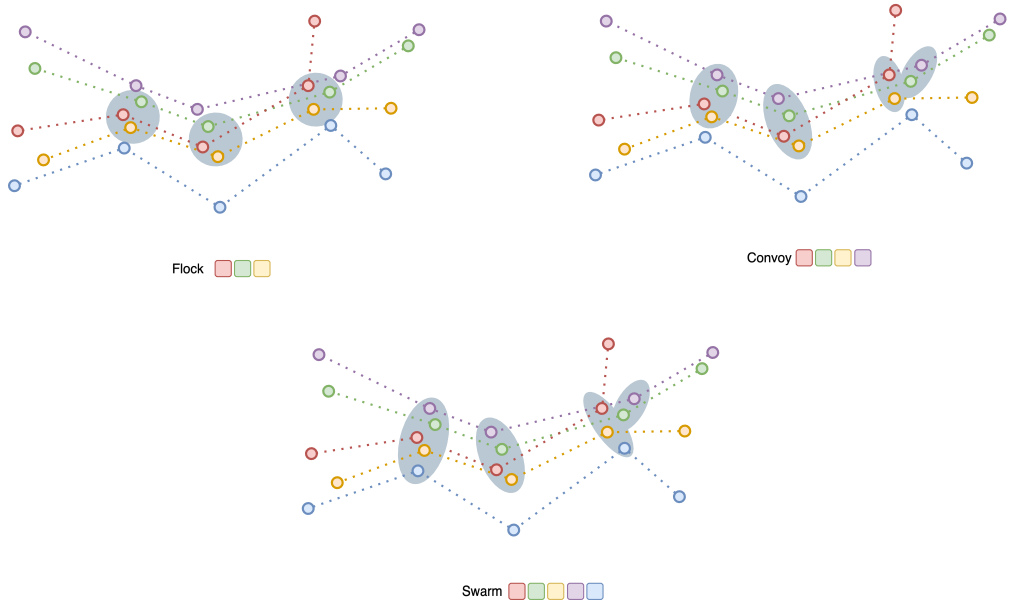


Figure 5.3: Differences between flocks, convoys, and swarms.

A **moving cluster** is yet another type of local pattern, which does not require the entities in it to be the same from start to finish. Let $H = \{t_1, t_2, \ldots t_n\}$ be a timestamped history, and $S = \{o_1, o_2, \ldots o_m\}$ a collection of objects that have moved during H, each with a variable lifetime. A snapshot S_i of H is the set of objects and their locations that exist at time t_i. Clusters of density-connected entities can be found at each snapshot. Let $g = c_1, c_2, \ldots, c_k$ be a sequence of snapshot clusters such that the timestamp of c_i is exactly before that of c_{i+1}; then g is a moving cluster w.r.t. to a threshold θ $(0 < \theta \leq 1)$ if:

$$\text{Jaccard}(c_i, c_{i+1}) = \frac{|c_i \cap c_{i+1}|}{|c_i \cup c_{i+1}|} \geq \theta, \quad \forall 1 < i \leq k$$

This means that as long as the Jaccard coefficient is high enough between every two consecutive shanpshots, the cluster is considered a pattern, and entities can move in and out of it.

In practice, the algorithm used to find moving clusters scans through each snapshot and runs DBSCAN/OPTICS on the entities in that snapshot, identifying clusters. At each step, a set $\mathcal{G}$ is used to keep track of current moving clusters, so that the clusters found in the current snapshot can be compared to the existing ones. If a moving cluster in $\mathcal{G}$ is extended by some new cluster, the two are merged and added to $\mathcal{G}_{new}$. If an existing moving cluster is not extended by any cluster, it is outputted. Any new cluster that does not extend any old moving cluster is also added to $\mathcal{G}_{new}$, as it may be the start of a new moving cluster. Finally, $\mathcal{G}$ is replaced by $\mathcal{G}_{new}$ and the process is repeated for the next snapshot.

Trajectory patterns, or **T-patterns**, are sequences of regions frequently visited in a specific order and with similar transition times. A T-pattern is defined as a pair (S, A), where $S = \langle (x_0, y_0), \ldots, (x_k, y_k) \rangle$ is a sequence of points in R^2, and $A = \langle \alpha_1, \ldots, \alpha_k \rangle \in R_+^k$ is the temporal annotation of the sequence. A T-pattern is frequent if it has support higher than some threshold s_{min}, where the support is calculated as the number of input trajectories that "spatio-temporally" contain the pattern. Containment does not require spatial or temporal data to perfectly match between the pattern and the input sequence to contribute to the support. Temporal containment requires that each element of the pattern is contained in an element of the sequence in the same order they appear, and that the time annotation between each pair of subsequent elements is similar for both the pattern and the sequence (the difference should be less than some threshold τ). Spatial containment requires that all the points of the pattern are contained, in order, in neighborhoods of points of the sequence.

Trajectories can also be treated as sequences of **regions of interest**, so that spatial information can be discarded after a preprocessing phase that assigns each pair of coordinates in the dataset to a RoI. If RoIs are not known a priori, they may be automatically identified by analyzing the dataset to find the most visited areas and/or using information found in external datasets.

5.3 Deep Learning for Trajectory Clustering

Recent approaches use deep learning models to facilitate detection and analysis of mobility patterns. A model can be trained to produce embeddings of trajectiories, automatically learning the best representation to capture similarity and differences between them, producing a clustering over this latent space. An example of such

framework is **Deep Embedded TrajEctory ClusTering network** (**DETECT**). It operates in three parts:

1. First, it summarizes the critical parts of the trajectories, and augments them with additional context. Stay areas are found by identifying segments in the trajectory where there is no movement, and a buffer is created around them. The PoIs falling inside the buffer aroud a stay area contribute to its **geographical context features**, represented by a vector of feature values, each indicating the contribution to a PoI category. The trajectory data is then transformed as a sequence of pairs: the coordinates of the stay areas, and their respective context features.

2. The enriched data is given as input to a LSTM-based encoder-decoder model to learn embeddings of trajectories. In general, training an encoder-decoder model has the objective of minimizing the difference between the encoder input (which is the original augmented data produced by the previous step) and the encoder output (what it recontructs from the embeddings produced by the encoder). The result should be a model that can compress the input data into a lower-dimensional space, and then reconstruct it with minimal loss.

3. The model is jointly optimized to also produce a good clustering of the embeddings, using the clustering error as an additional term for the loss function. The error is the distance between two probability distributions Q and P: Q is the distribution of the current embeddings and each term q_{ij} is the probability of assigning the i^{th} embedding to the j^{th} cluster centroid, while P is the target distribution. Their distance is calculated using the Kullback-Leibler divergence.

Chapter 6

Mobility Prediction

Mobility prediction is the task of predicting mobility information of individuals or collections of individuals given their historical mobility data. The target of prediction may be the trajectory of an individual (either a full path or the next location), the destination of a trip, the happening of certain events (e.g., the probability of a car crash occurring), or a combination of the above. For collective targets, the prediction may produce aggregate flows described by an Origin-Destination matrix, or events that involve multiple individuals at once (e.g., a traffic jam).

Approaches are driven by two perspective: the first considers movement as continuous, expressing points as latitude-longitude pairs, and interprets prediction as reconstructing the precise movements of individuals; the second considers movement as discrete, where points are interpreted as visited areas (PoIs, mobile phone cells, tessellation cells), and prediction becomes predicting which cell ID or sequence of cell IDs describes the future movement of the individual.

6.1 Prediction Tools

6.1.1 Hidden Markov Models

A **Markov chain** is a standard way to model sequential stochastic processes. It is used to estimate the probability of sequences of events, called states, that take values from a finite set (e.g., words, weather conditions, stock prices). The assumption is that the probability of an event happening depends only on the previous event ("Markov assumption"):

$$P(q_i|q_1q_2\dots q_{i-1}) = P(q_i|q_{i-1})$$

The model can be represented as a simple **transition matrix** P_{ij}, containing the probability of transitioning from each state i to each state j. Formally, a Markov chain

is defined by:

- A set of states $Q = \{q_1, q_2, \ldots, q_n\}$;
- A vector of prior probabilities $\Pi = \{\pi_1, pi_2, \ldots, \pi_n\}, \quad \pi_i = P(S_0 = q_i)$;
- A transition matrix $P = \{a_{ij}\}, \quad i, j \in [1, n]$, such that $a_{ij} = P(q_j|q_i)$.

Learning Π and Q is straightforward, as they can be estimated from the data by counting the frequency of each state in Q.

A **Hidden Markov Model** (**HMM**) calculates the probability of sequences of events by separating them between observed and hidden. The observed states (the ones that actually constitute the sequence) are "emitted" by the hidden ones, meaning that their probability depends only on the current hidden state. Formally, a hidden Markov model is defined by:

- A set of states $Q = \{q_1, q_2, \ldots, q_n\}$;
- A set of observables $O = \{o_1, o_2, \ldots, o_m\}$
- A vector of prior probabilities $\Pi = \{\pi_1, pi_2, \ldots, \pi_n\}, \quad \pi_i = P(S_0 = q_i)$;
- A transition matrix $P = \{a_{ij}\}, \quad i, j \in [1, n]$, such that $a_{ij} = P(q_j|q_i)$;
- A matrix of emission probabilities $P_E = \{p_{s \to o}\}, s \in Q, o \in O$, such that $p_{s \to o} = p(o|s)$.

The input of the model is a sequence of observations drawn from a finite vocabulary. The model can be used to solve three problems: likelihood, decoding, and learning.

Likelihood Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$. The naive approach to calculating the likelihood of a sequence is to sum the probabilities of that sequence being generated by all possible hidden state sequences. Since the number of hidden and observable states tends to be large in real tasks, this approach is infeasible as it has exponential complexity ($O(|Q|^{|O|})$). Insteadm an efficient algorithm called **forward algorithm** is used. It is a kind of dynamic programming algorithm with a $O(|Q|^2|O|)$ complexity.

Decoding Given as input an HMM $\lambda = (A, B)$, and a sequence of observations $O = o_1, o_2, \ldots, o_T$, find the most probable sequence of states $Q = q_1, q_2, \ldots, q_T$. The most common decoding algorithm is the **Viterbi algorithm**, which is also a dynamic programming algorithm.

Learning Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B. The standard algorithm for HMM training is **forward-backward algorithm** (or **Baum-Welch algorithm**), a special case of the

Expectation-Maximization algorithm, which is an iterative method used to find the maximum likelihood estimates of parameters in statistical models. A trained HMM can be used to predict the next value in a sequence using:

- **Point estimation**, by simply appending the state that, once appended to the sequence, maximizes the likelihood:

$$\begin{aligned} o_T &= \arg\max_o P(O_T = o | o_1, o_2, \ldots, o_{T-1}) = \\ &= \arg\max_o \frac{P(o_1, o_2, \ldots, o_{T-1}, O_T = o)}{P(o_1, o_2, \ldots, o_{T-1})} = \\ &= \arg\max_o P(o_1, o_2, \ldots, o_{T-1}, O_T = o) \end{aligned}$$

- **Conditional expectation** (for numerical data only), which calculates the expected value of the next observation given the current sequence:

$$\mathbb{E}[O_T | o_1, o_2, \ldots, o_{T-1}] = \sum_o o \cdot P(O_T = o | o_1, o_2, \ldots, o_{T-1})$$

When applied to mobility data, usually the first step is a discretization phase, in which each trajectory is converted into a sequence of cell/area IDs (the observations), and the HMM is trained on these sequences. Alternatively, data is kept as is, making emissions 2-dimensional (latitude and longitude pairs), and assuming a Gaussian distribution for the emissions.

6.1.2 Pattern-based Prediction

Mobility patterns can be used for predictions other than simply highlighting frequent sequences of visits/events. For example, after finding some frequent sequences of visits, and observing a trajectory up until a certain point, we may estimate the destination of the individual's trip by looking at the pattern that best matches the trajectory so far, and taking the last location in it. Patterns can be either treated as Booleans (the pattern either does or does not occur in the trajectory), or as numerical values that represent how strongly a pattern occurs. Standard Machine Learning models can be then applied to automatically learn from the patterns and predict the desired target variable.

Patterns can be extracted using various criteria; the most common is frequency, where the goal is to find sequences that express significant features, but can also be extracted based on their discriminatory power, which may find infrequent patterns that are useful.

The typical pipeline for pattern-based prediction is:

1. The user history is collected, representing the sequence of spatio-temporal points the user visited within a certain time frame;
2. The data is split into several trips, by identifying the stop points and the movements among them;
3. The resulting individual trajectories are grouped using a clustering algorithm with an appropriate spatio-temporal distance function;
4. Clusters with few elements are pruned as they do not represent significant patterns;
5. The medoid of each cluster is calculated, and the resulting set of medoids becomes the user's routine movements.

This is repeated for multiple users to form a pool of patterns. When a prediction is done for a specific user, the algorithm first tries to match the user's trajectory with its own patterns, and if none of them are good enough, those of the general population are analyzed.

6.1.3 Neural Networks

The three main architectures used are **recurrent neural networks** (RNNs), **convolutional neural networks** (CNNs), and **generative adversarial networks** (GANs).

RNNs are neural networks where each hidden unit has an additional self-connection that allows the net to maintain a state. The state is updated each time an instance goes through the net, and is connected to its corresponding unit with a weighted edge that is trained along with the other weights. The model is especially useful fr sequential data, as it uses past information to influence the prediction of the current instance (e.g., predicting a word in a sentence knowing which words came before it). Standard RNNs have issues with capturing long-term dependencies between input and output, suffering from the "vanishing gradient" problem, so an advanced variant of RNNs called **Long Short-Term Memory networks** (LSTMs) are used instead. LSTM units have a more sophisticated structure with gates that control the flow of information. A typical LSTM scheme is the Encoder-Decoder architecure, where training is done in two steps: the **encoder** step, during which the sequence data is transformed in a latent representation, and the **decoder** step, where the output of the encoder is processed to generate a sequence until a special token is generated to signal the end.

CNNs are used mostly for image processing, and can be used in conjuction wit LSTM or other sequence-based models: CNNs capture spatial relations and are able to identify objects and features, while LSTM captures temporal relation and movement.

GANs are a type of model where two neural networks are trained to "compete" with

each other in a zero-sum game. The **generator** tries to produce synthetic data that is as similar as possible to the real data, while the **discriminator** tries to distinguish between real and fake data. Initially, the generator produces a set of synthetic instances by processing a random input (e.g., a latent representation of a trajectory), and the discriminator is trained on a dataset composed of those fake instances and real instances from an actual dataset. The two models are then trained in parallel, trying to make the generator capable of producing convincingly realisting data, and the discriminator capable of telling if an instance comes from the generator or not. Both generator and discriminator can be any suitable model, depending on the type of data to learn from.

Appendix A

Useful Tools and Libraries

A.1 File Formats

Shapefile Popular geospatial vector data format for GIS software. It can describe any vector feature using points, lines, and polygons. The format is actually composed of multiple files, but the shapefile itself has the filename extension `.shp`.

Specifies axis coordinates assuming a Cartesian coordinate system, using the ordering (x, y): x is longitude, and y is latitude.

GeoJSON Open format used to describe collections of spatial objects using JSON. Possible geometries are points, lines, polygons, and collections of these types. It uses the filename extension `.geojson` or `.json`.

A.2 Python Libraries

Shapely Used to analyze and perform set operations on planar geometric objects. Represents objects as either `Point`s, `LineString`s, or `Polygon`s.

- `Points` can be 2- or 3-dimensional, and is defined from a coordinate tuple.
- `LineString` is built from a list of at least two coordinate tuples.
- `Polygon` is a filled area; must be defined by at least three coordinate tuples that speciy the outer perimeter, and optionally a list of inner holes.

Various attributes can be directly accessed from an instantiated object to analyze them (e.g., length, centroid, area, bounding box, etc.). For each of these types, a `Multi-` variant exists, used to have collections of objects of the same type.

GeoPandas Extends the datatypes used by pandas to allow spatial operation on geometric types. It's built on top of Shapely for geometric opeations. Its main data structures are `GeoSeries` and `GeoDataFrame`, which respectively extend pandas' `Series` ad `DataFrame` types. One constraint is that a `GeoDataFrame` must contain a `geometry` column that specifies the geometry of the objects held in the dataframe; this column must be of type `GeoSeries`. The `.plot()` function (built on top of matplotlib's `.plot()`) can be used to easily visualize geometric data on a 2D map. Supports the shapefile data format, with methods for reading (`read_file()`) and writing (`to_file()`) shapefiles.

The main operations that can be performed on `GeoDataFrame`s are spatial joins: the method used is `.sjoin()`, where parameter `how` accepts a string that specifies the type: `"left"`, `"right"`, or `"inner"`. The direction refers to the dataframe on which the method is called; for example:

```
df1.sjoin(df2, how="left")
```

performs a join that preserves all the rows of `df1`.

Scipy General library for scientific computing. Contains the module `scipy.spatial` that provides spatial algorithms and data structures. Provides a `cKDTree` class that implements efficient methods for nearest-neighbor calculations in spatial data.

Scikit-Learn The most popular machine learning library in Python.
Its `KNeighborRegressor` class can be used to perform k-NN regression on spatial data.

Scikit-Mobility Library for human mobility analysis. The library manages mobility data of various formats (CDR, GPS, social media data, etc.), and can represent trajectories and mobility flows using the data structures `TrajDataFrame` and `FlowDataFrame`, respectively. A `TrajDataFrame` requires three columns: latitude, longitude, and datetime, while a `FlowDataFrame` requires three columns: origin, destination, and flow.

NetworkX Package for creation and manipulation of complex networks. Has implementations of many graph algorithms. Useful to operate on street networks, since they are usually represented as graphs with arbitrary attributes on nodes/edges.

OSMnx Package used to download, model, and visualize street networks from OpenStreetMap. It can retrieve walking, driving, and biking networks, as well as points of interests, transit stops, street orientations, travel time, elevation data, and routing. It contains several methods that automatically download data by providing some spatial reference; for example, `graph_from_address()` downloads and returns the desired type

of network within some distance to a specific provided address; `graph_from_point()` is similar to the previous, but downloads the network centered around a point expressed as a tuple *(lat, lon)*; `graph_from_bbox()` allows the user to specify a bounding box to download the network within.

Geovoronoi Can create and plot voronoi tessellations. Has a main function called `voronoi_regions_from_coords()`, which takes as input a list of coordinates representing the centroids of the cells and an area to bound the cells, and returns a list of polygons representing the tessellation. It uses Scipy to perform the tessellation, and Shapely to cut the edge polygons to fit into the provided shape (since they would be otherwise infinite).

PySal Provides various methods for spatial analysis. It has four main modules:

- `Lib` contains the core spatial data structure and file IO functionalities;
- `Explore` contains methods for exploratory analisys. It contains the subpackage `esda`, which provides specific methods for spatial autocorrelation (Moran's I, Geary's C);
- `Model` contains different models to estimate spatial relationships, both linear and non-linear;
- `Viz` contains visualization tools.

Pykrige Implements various kriging algorithms, both for 2D and 3D data.

A.3 Links

opencellid.org - World's largest open database of cell towers.
geojson.io - Browser-based interactive tool for creating and editing GeoJSON files.

Bibliography

[1] Kang-Tsung Chang. *Introduction to geographic information systems*. Mcgraw-hill Boston, 2018.

[2] Manuel Gimond. Intro to gis and spatial analysis. `https://mgimond.github.io/Spatial/`.

[3] Massimiliano Luca, Gianni Barlacchi, Bruno Lepri, and Luca Pappalardo. A survey on deep learning for human mobility, 2021.

[4] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

[5] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 336–343, 2009.

[6] Lei Zhu, Jacob R. Holden, and Jeffrey D. Gonder. Trajectory segmentation map-matching approach for large-scale, high-resolution gps data. *Transportation Research Record*, 2645(1):67–75, 2017.

[7] Hugo Barbosa, Marc Barthelemy, Gourab Ghoshal, Charlotte R James, Maxime Lenormand, Thomas Louail, Ronaldo Menezes, José J Ramasco, Filippo Simini, and Marcello Tomasini. Human mobility: Models and applications. *Physics Reports*, 734:1–74, 2018.

[8] Joachim Gudmundsson and Marc van Kreveld. Computing longest duration flocks in trajectory data. In *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, GIS '06, page 35–42. Association for Computing Machinery, 2006.

[9] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. Discovery of convoys in trajectory databases, 2010.

[10] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. In *Advances in Spatial and Temporal Databases: 9th International Symposium, SSTD 2005, Angra dos Reis, Brazil, August 22-24, 2005. Proceedings 9*, pages 364–381. Springer, 2005.

[11] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, page 330–339. Association for Computing Machinery, 2007.