GIS

A Computing Perspective

Third Edition

Edited by Matt Duckham, Qian (Chayn) Sun and
Michael F. Worboys

First published 1995

ISBN: 978-1-4665-8719-9 (hbk)
ISBN: 978-1-032-53973-7 (pbk)
ISBN: 978-0-429-16809-3 (ebk)

Chapter 9

Artificial Intelligence and GIS

DOI: 10.1201/9780429168093-9

The funder for this chapter is Matt Duckham

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

Artificial Intelligence and GIS

9

Sections

9.1 *Ontology engineering*

9.2 *Qualitative spatial reasoning*

9.3 *Machine learning and spatial analysis*

9.4 *Deep learning*

Artificial intelligence (AI) encompasses a wide range of different algorithms and techniques commonly used in GIS for creating new knowledge from spatial data. More specifically, through this chapter you will discover how to:

- interpret, use, and design machine-readable **ontologies** to help increase the interoperability of spatial information;
- represent **qualitative spatial relationships**, such as left and right, cardinal directions, and region relations, within automated reasoning systems;
- apply the basic **machine learning** techniques that underpin many common **spatial analyses**; and
- understand the principles and limitations of advanced machine learning techniques, such as **artificial neural networks** and **deep learning**.

Artificial intelligence has been a central to the development of GIS since the earliest days of GIS research.[1] In a nutshell, *artificial intelligence* (AI) concerns the design and development of machines imbued with behaviors and abilities that might otherwise be regarded as requiring human intelligence (see Box 9.1 on the following page).

While the breadth and diversity of AI techniques today is vast, it is possible to identify two major threads that run throughout the history of AI: *knowledge representation and reasoning* and *machine learning*. Knowledge representation and reasoning (KR^2) is concerned with using automated logical reasoning together with symbolic representations of knowledge to solve AI problems. In contrast, machine learning (ML) is concerned with automated learning from data.

In this chapter, we begin by introducing the most successful and widely used KR^2 technique today: ontology engineering (Section 9.1). Whereas the emphasis in ontology engineering is more strongly on the representation of knowledge than reasoning about that knowledge, the emphasis is reversed in qualitative spatial reasoning (Section 9.2). Turning to ML in the second half of this chapter, Section 9.3 shows how many of the most familiar and frequently used spatial analysis tools and techniques are in fact founded on machine learning. Finally, Section 9.4 introduces one of the most important machine learning techniques in the recent decade: artificial neural networks and deep learning.

[1] In 1987, the second ever issue of the *International Journal of Geographical Information Science* (IJGIS), the flagship journal for GIS research, contained a research paper on a knowledge-based GIS called "KBGIS-II" with a facility to apply logical rules to spatial data, and even to learn new rules from data automatically (Smith, Peuquet, Menon, & Agarwal, 1987).

artificial intelligence

KR^2

machine learning

Box 9.1: The Turing test

The definition of AI as, effectively, a machine that can imitate human intelligent behavior is tied most closely to the work of British mathematician and computer scientist Alan Turing. In a landmark 1950 paper, Turing argued that a machine could be called "intelligent" if a human interrogator could not reliably distinguish the machine from a human interlocutor (Turing, 1950). Turing's thought experiment—which Turing called "the imitation game," also the title of the 2014 film about Turing's life and work—is commonly known in computer science as the *Turing test*. Turing made many other enduring contributions to the emerging fields of computing and AI, including an abstract mathematical description of computing machines and the types of problems that can be solved by computers, called a *Turing machine*. Sadly, Alan Turing died at age 41, just two years after he was convicted for "gross indecency"—a euphemism for homosexuality, then a crime in the United Kingdom—and forced to undergo a year of invasive hormone injections resulting in impotence and physical changes to his body. In 2009, the British government officially apologized for Turing's treatment, acknowledging at the same time the appalling treatment of innumerable gay men in Britain during the era.

9.1 *Ontology engineering*

The idea of *ontology* as the philosophical study and classification of what things exist was already encountered in Chapter 4 in the context of how we model geographical spaces. This section introduces the related idea of *an* ontology, which can be defined as an explicit, unambiguous, and machine-readable specification of the meaningful and relevant entities and relationships in an application.[2] The activity of designing and using computerized ontologies is termed *ontology engineering*.

ontology

[2] The topics of conceptual data modeling and entity-relationship models (Section 2.2.2) have clear commonalities with ontologies. However, ontologies are focused more generally on the stable meanings of words used to refer to things that exist in the world. Conceptual data models, by contrast, are slanted towards describing the data related to an application. Nevertheless, a good conceptual data model may also be the basis of a good ontology.

The core idea behind ontologies[3] is that the meaning of words in natural language is relatively stable, while the meaning of items in digital data structures and data models can be highly transient and changeable. An ontology helps to make explicit the link between relatively stable human language words and relatively volatile data item definitions. As we shall see, using ontologies can have significant impacts on making GIS and spatial data more interoperable.

ontology engineering

[3] We consistently use the count nouns—"an ontology" and "ontologies"—to refer to the "explicit specification of human concepts" sense of the word. The abstract, uncountable mass noun—"ontology"—is used to refer to the philosophical study sense of the word.

For example, Figure 9.1 depicts a simple and widely used ontology—or "vocabulary"—for describing locations in terms of spherical coordinates referenced to the WGS84 datum. A *vocabulary* is an informal term for a simple ontology, although there is no strict definition of what counts as "simple" in this context. We will use the term "vocabulary" in this chapter to refer to minimalist ontologies such as depicted in Figure 9.1.

vocabulary

Just as object-oriented classes capture *types* or *kinds* of (object) instances, so too do ontology classes. The vocabulary in Figure 9.1 has just two classes defined: SpatialThing, which has objects with a spatial extent as its instances; and Point which is a (subclass of) SpatialThing. Here the ontology engineering terminology deviates slightly from object-oriented terminology defined in Chapter 4. Classes in ontologies do not possess *behaviors* in the same way as object-oriented classes. However, the "is a" subclass relationship is used exactly as already encountered in the context of inheritance in OO models (Section 4.1.2): a Point "is a" (i.e., specializes a) SpatialThing.

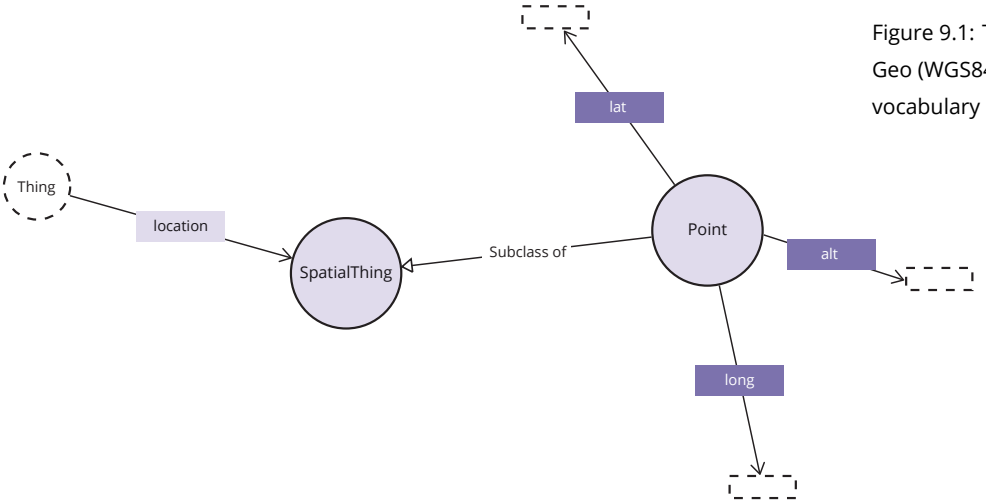


Figure 9.1: The Basic Geo (WGS84 lat/long) vocabulary

Classes may in turn possess *properties* in a similar sense to edge properties in the property graph model and akin to attribute (type) in ER modeling in Chapter 2. In an ontology, a property has the structure of a function in set-based representations (see Section 3.2.3). As for functions, ontology properties have a domain and a range.[4] In the Basic Geo vocabulary in Figure 9.1, the location property has any Thing as its domain (i.e., a location may potentially be a property of anything in an ontology) and SpatialThing as its range. In turn, each point (which is a SpatialThing) may have properties called lat, long, and alt representing the point latitude, longitude, and altitude. These properties have Point as their range (i.e., only points can have coordinates) but, interestingly, no defined range (empty dashed boxes in Figure 9.1).

[4] Recall from Section 3.2.3: the *domain* of a function is the set of things the function maps *from*; the *range* is the set of things the function maps *to*.

The reason for leaving the range undefined is to allow different users of the ontology to define for themselves the precise encoding and data type used for latitude, longitude, and altitude data (including text strings, floating point numbers, or double precision). Properties with no defined range in an ontology are called *annotation properties*, in contrast to *data properties* (which have a data type such as integer or text string as their range) and *object properties* (which have a class as their range, such as location in Figure 9.1).

annotation properties

Ontologies such as the Basic Geo vocabulary in Figure 9.1 can help increase interoperability because they provide a more stable, linguistic basis for describing the *meanings* behind spatial data. Whether I develop a relational database scheme with attributes called "latitude" and "longitude"; a graph database with nodes labeled "coordinate"; or object-oriented software that defines a class of "SpatialPoint" objects, as long as I associate those system entities with the appropriate classes or properties in the Basic Geo vocabulary then everyone should be able to understand what my data means, and link it into other spatial data sets in the right way. Happily, a suite of technologies has been developed to help make it easier to do exactly this—to make

connections between data and the ontologies that describe stable, linguistic meanings for that data.

9.1.1 *Representing data using ontologies*

The discussion above introduces some of the key entities in an ontology including classes, subclasses, instances, and properties. In this section we explore briefly how one can use an ontology to represent the meanings behind data. The *Semantic Web* consists of a collection of technologies to link data to rich, machine-readable ontologies representing the meaning of that data.

Semantic Web

URI

URL

One of the most basic building blocks in the Semantic Web is URIs: *uniform resource identifiers*. A URI is a unique identifier for any digital resource. Web URLs (*uniform resource locators*) are a familiar example of URIs used to identify web pages. Indeed, most URIs today use the machinery of Web addresses to locate the resource. The URI of the SpatialThing class in Figure 9.1, for example, is http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing.

triple

The connections between resources in the Semantic Web are captured using *triples*. Each triple can be thought of as an atomic fact of the form:

subject predicate object

[5] Note than the term "object" is used here in the linguistic sense of "the thing being acted upon in a sentence," rather than the object-oriented sense of "state + behavior."

[6] http://geonames.org

For example, the fact that "The Eiffel Tower is located in Paris" can be represented as a triple: "Eiffel Tower" (subject, "Tour Eiffel" in French), "located in" (predicate), "Paris" (object[5]). Facts can be combined together, linking information from diverse sources. For example, the triple below states a simple fact (triple) from GeoNames[6], an open source database containing myriad facts about millions of toponyms (placenames) from around the world.

```
<https://sws.geonames.org/6254976/> # subject
    <http://www.geonames.org/ontology#name> # predicate
            "Tour Eiffel" . # object
```

Turtle

The triple above is written in a language called *Turtle*. Simple triples in Turtle use the "subject predicate object" format for facts followed by a terminating full stop. Hence, the triple above asserts that the GeoNames instance 6254976 is named "Tour Eiffel".

Rather than rewrite subjects multiple times, Turtle allows multiple predicates to be listed for the same subject by using a semicolon to separate predicates. Thus, the listing below:

```
<https://sws.geonames.org/6254976/> # subject
    <http://www.geonames.org/ontology#name> "Tour Eiffel" ; # predicate object
    <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "48.85832" ; # predicate object
    <http://www.w3.org/2003/01/geo/wgs84_pos#long> "2.29452" . # predicate object
```

asserts the fact that the Eiffel Tower (or more precisely GeoNames instance 6254976 with name "Tour Eiffel") is located at latitude "48.85832" and longitude "2.29452". Note that the Basic Geo (WGS84 lat/long) vocabulary (introduced

in Figure 9.1) is used to ensure that the semantics of the location coordinates are clear.

Further, commas can be used to separate multiple objects for the same predicate. For example, the fact that the Eiffel Tower has different official names in English, German, and French can be asserted using the statement:

```
<https://sws.geonames.org/6254976/> # subject
    <http://www.geonames.org/ontology#name> "Tour Eiffel" ; # predicate object
    <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "48.85832" ; # predicate object
    <http://www.w3.org/2003/01/geo/wgs84_pos#long> "2.29452" ; # predicate object
    <http://www.geonames.org/ontology#officialName> # predicate
        "Eiffel Tower"@en, "Eiffelturm"@de, "Tour Eiffel"@fr . # object object object
```

The Turtle listing in Figure 9.2 below shows 11 of more than 70 facts about the Eiffel Tower in the GeoNames database. The listing contains additional information about a location map and further semantic information about the Eiffel Tower stored on the public data catalog DBpedia[7]. The first listed predicate (simply a) is shorthand for asserting the type of the subject. In this case, the GeoNames instance 6254976 is an instance of type Feature. To shorten the Turtle listings still further, the first three lines of the listing in Figure 9.2 define three *prefixes* (including gn for the GeoNames ontology and geo for the Basic Geo ontology).

[7] DBpedia (https://www.dbpedia.org/) is a Semantic Web database of information from Wikipedia, the crowdsourced encyclopedia.

```
@prefix gn: <http://www.geonames.org/ontology#> . # GeoNames prefix
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> . # WGS84 Basic Geo prefix
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> . # RDFS prefix

<https://sws.geonames.org/6254976/>
    a <http://www.geonames.org/ontology#Feature> ;
    gn:name "Tour Eiffel" ;
    gn:officialName "Eiffel Tower"@en, "Eiffelturm"@de, "Tour Eiffel"@fr ;
    gn:countryCode "FR" ;
    geo:lat "48.85832" ;
    geo:long "2.29452" ;
    gn:locationMap <https://www.geonames.org/6254976/tour-eiffel.html> ;
    rdfs:seeAlso <https://dbpedia.org/resource/Eiffel_Tower> .
```

Figure 9.2: Representation of the Eiffel Tower on GeoNames in Turtle

Eagle-eyed readers may notice that the Turtle in listing Figure 9.2 includes a new third prefix: rdfs. RDFS is a standard ontology engineering vocabulary called "RDF schema," itself based on RDF: the "resource description framework." RDF describes the fundamental data model for triples that underpins the Semantic Web. Indeed, the language Turtle is also based on RDF (Turtle stands for "terse RDF triple language"). Figure 9.3 encodes exactly the same information about the Eiffel Tower as the listing in Figure 9.2 but using RDF/XML format. The RDF/XML format is not as human-readable as Turtle, but it is often used in the Semantic Web because XML is a general purpose document format understood by many different computer applications.

RDF

Figure 9.3: RDF/XML representation of the Eiffel Tower on GeoNames, identical in content to the Turtle representation in Figure 9.2

```
<rdf:RDF xmlns:gn="http://www.geonames.org/ontology#" # GeoNames prefix
         xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#" # WGS84 prefix
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" # RDF prefix
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"> # RDFS prefix

    <rdf:Description rdf:about="https://sws.geonames.org/6254976/">
        <rdf:type rdf:resource="http://www.geonames.org/ontology#Feature"/>
        <gn:name>Tour Eiffel</gn:name>
        <gn:officialName xml:lang="en">Eiffel Tower</gn:officialName>
        <gn:officialName xml:lang="de">Eiffelturm</gn:officialName>
        <gn:officialName xml:lang="fr">Tour Eiffel</gn:officialName>
        <gn:countryCode>FR</gn:countryCode>
        <geo:lat>48.85832</geo:lat>
        <geo:long>2.29452</geo:long>
        <gn:locationMap
            rdf:resource="https://www.geonames.org/6254976/tour-eiffel.html"/>
        <rdfs:seeAlso rdf:resource="https://dbpedia.org/resource/Eiffel_Tower"/>
    </rdf:Description>
</rdf:RDF>
```

The Semantic Web is designed to be agnostic to specific encodings, so several such different languages are commonly used in connection with ontologies. Ultimately, however, all are founded on subject-predicate-object triples, and so they can be represented using graphs, often termed *knowledge graphs*, such as that pictured in Figure 9.4.

knowledge graph

9.1.2 *Querying ontologies*

Section 2.4 introduced the graph database and explored in more detail the property graph model. Although property graph databases can also be used to store RDF data, the addition of node and edge properties in the property graph model is not required for storing triples. Instead, RDF data in the Semantic Web is usually stored in the second major class of graph database: the triple store. A *triple store* is a database specifically designed to store and retrieve only RDF subject-predicate-object triples.

triple store

As we saw in Section 2.4, there is currently no single standard for querying property graph databases, with languages such as Cypher among the leading influences on an emerging future standard. Happily, though, triple stores do enjoy a standard query language: SPARQL.[8] SPARQL shares many superficial similarities with the standard relational database query language SQL. However, in detail SPARQL is more akin to other graph query languages such as Cypher, as might be expected.

SPARQL

[8] SPARQL is an acronym for "SPARQL protocol and RDF query language"—confusingly, "SPARQL" is an abbreviated word in its own acronym.

Echoing SQL, SPARQL queries are constructed around `SELECT` ... `WHERE` statements. Unlike SQL queries, the `SELECT` clause is followed by one or more variables that will appear in the query results. The `WHERE` clause then lists patterns to match against the stored graph, much as is found in other graph

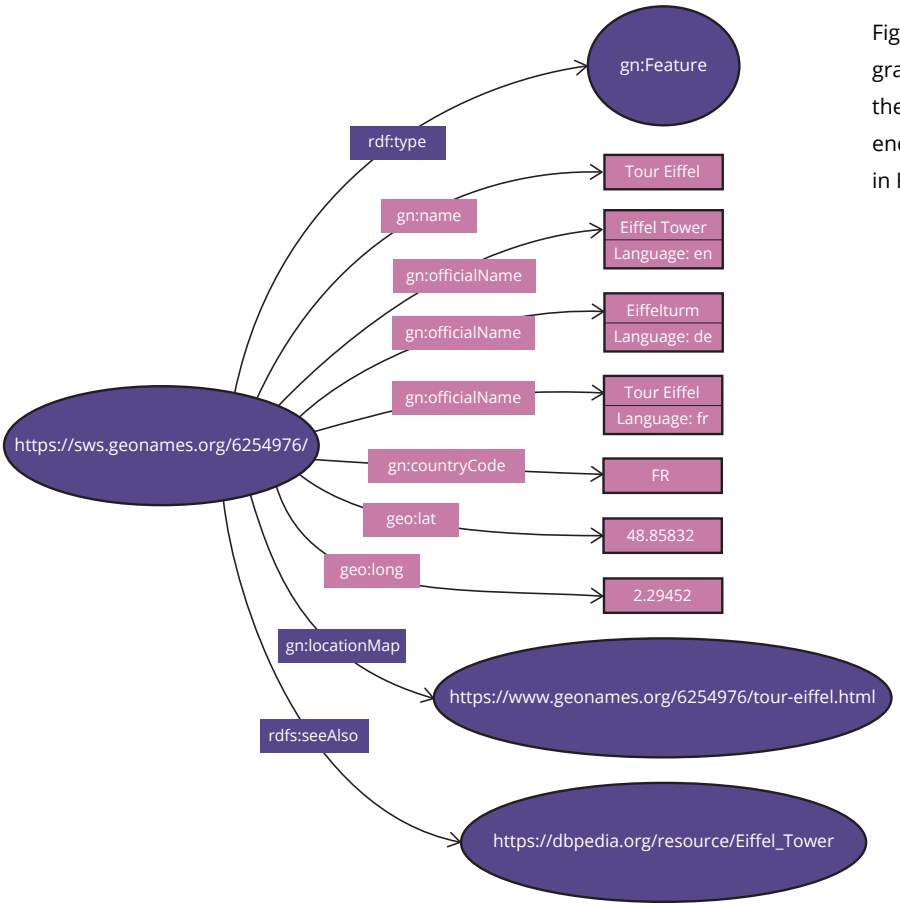


Figure 9.4: The knowledge graph representation of the Eiffel Tower triple data encoded in Turtle and RDF in Figures 9.2 and 9.3

query languages such as Cypher. For example, the SPARQL query in Figure 9.5 retrieves the latitude and longitude of the Eiffel Tower from the graph represented in Figure 9.4 and encoded in Turtle and RDF/XML in Figures 9.2 and 9.3, respectively.

In the query in Figure 9.5, following the definition of the gn and geo prefixes, the SELECT clause specifies two variables whose values will appear in the query output: ?latitude and ?longitude. Variables in SPARQL must always be denoted using the leading ? symbol. In the WHERE clause, three matches are defined. First, the query searches for matches with some variable ?x with gn:name as "Eiffel Tower". The terminating full stop indicates the end of a triple pattern, similarly to the Turtle encoding. Next, two further query matches look for the same subject ?x, but storing the object linked by predicate geo:lat in the variable ?latitude and the object linked by predicate geo:long in the variable ?longitude. The final output of the SPARQL query in Figure 9.5 is given in Table 9.1.

?latitude	?longitude
"48.85832"	"2.29452"

Table 9.1: Output of SPARQL query in Figure 9.5

Figure 9.5: SPARQL query to retrieve the latitude and longitude of the Eiffel Tower

```
PREFIX gn: <http://www.geonames.org/ontology#> # GeoNames prefix
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> # WGS84 Basic Geo prefix

SELECT ?latitude ?longitude WHERE {
    ?x gn:name "Tour Eiffel" . # GeoNames instance named "Tour Eiffel"
    ?x geo:lat ?latitude . # Retrieve latitude
    ?x geo:long ?longitude . # Retrieve longitude
}
```

With the breadth and depth of the Semantic Web growing daily, SPARQL brings the power to query vast networks of spatial knowledge. For example, using the ontology to bridge directly from the GeoNames database to DBpedia, Figure 9.6 gives a SPARQL query to search for other buildings in Paris and sort them by completion date (listed in Table 9.2).

Figure 9.6: SPARQL query to retrieve GeoNames buildings in Paris by construction date (see Table 9.2 on the next page)

```
PREFIX gn: <http://www.geonames.org/ontology#> # GeoNames prefix
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> # WGS84 Basic Geo prefix
PREFIX rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> # RDF prefix
PREFIX dbp: <http://dbpedia.org/property/> # DBpedia property prefix
PREFIX dbo: <http://dbpedia.org/ontology/> # DBpedia ontology prefix
PREFIX dbr: <http://dbpedia.org/resource/> # DBpedia resource prefix

SELECT DISTINCT ?name ?completed ?latitude ?longitude WHERE {
    ?x gn:name ?name . # Retrieve GeoNames name
    ?x geo:lat ?latitude . # Retrieve WGS84 latitude
    ?x geo:long ?longitude . # Retrieve WGS84 longitude
    ?x dbo:location dbr:Paris . # Retrieve location in Paris
    ?x rdf:type dbo:Building . # Retrieve instances of Buildings
    ?x dbp:completionDate ?completed # Retrieve completion dates of Buildings
}
ORDER BY DESC(?completed) # In order of completion date, most recent first
```

Note, however, that the SPARQL query Figure 9.6 does not perform a spatial operation when satisfying the search criterion "in Paris." Instead, the query uses the fact that the DBpedia database includes a predicate dbo:location and the object dbr:Paris. Just like our basic SQL queries, SPARQL does not include spatial operators, although attempts have been made to extend SPARQL with such spatial operations (see Box 9.2 on the facing page).

9.1.3 *Reasoning with ontologies*

To this point, our discussion of ontologies has only addressed the *representation* of human knowledge, with subject-predicate-object triples as the foundation of that representation. An essential ingredient in any KR[2] AI system is the ability also to *reason* with these representations.

Box 9.2: GeoSPARQL

GeoSPARQL is an emerging standard for representing geospatial data on the Semantic Web as well as an extension to SPARQL to enable spatial queries to be constructed. The representation of geospatial data in GeoSPARQL is based on the *simple feature model* already introduced in Section 4.1.3 and summarized in Figure 4.14. The spatial operations available in GeoSPARQL queries include basic spatial filters such as: within-distance queries for retrieving objects within a certain distance of a point; spatial point and range queries; and querying topological relations based on an extension to the 4-intersection model (4IM) already discussed in Chapter 3. The leading triple store databases also offer basic spatial indexes, such as geohashing introduced in Box 6.3 on page 238. However, today's spatial query capabilities for the Semantic Web are overall relatively basic.

?name	?completed	?latitude	?longitude
Tour Eiffel	1889	48.85832	2.29452
Bastille Opera	1990	48.85272	2.36999
Hyatt Regency Paris Etoile	1974	48.88070	2.2845
Tour Montparnasse	1973	48.84193	2.32207
Centre Georges Pompidou	1971	48.86060	2.35237
Hôtel de Ville de Paris	1892	48.85644	2.35244
Arc de Triomphe du Carrousel	1808	48.86173	2.33291
Panthéon	1790	48.84624	2.34613
Élysée Palace	1722	48.87060	2.31709
Palais du Luxembourg	1645	48.84844	2.33726
Hôtel de Ville de Paris	1357	48.85644	2.35244
...	...	...	...

Table 9.2: Output of SPARQL query in Figure 9.6 on the facing page

In fact, behind the scenes reasoning was woven in to the Semantic Web technologies we have already encountered. The SPARQL query in Figure 9.5 can satisfy its query simply by retrieving the required stored data from GeoNames, such as that listed in Figure 9.2 and Figure 9.3. The query in Figure 9.6, however, went much further linking seamlessly to a completely different data source, DBpedia. How did it do that?

The answer lies in the DBpedia knowledge graph. The worldwide network of structured, open access, Semantic Web data (including databases such as DBpedia and GeoNames) is termed *linked open data* (LOD). DBpedia contains over 3 billion such facts, with almost 400 facts about the Eiffel Tower alone, including in particular the (Turtle) fact:

linked open data

```
@prefix owl: <http://www.w3.org/2002/07/owl#> # OWL prefix
@prefix dbr: <http://dbpedia.org/resource/> # DBpedia resource prefix

dbr:Eiffel_Tower owl:sameAs <https://sws.geonames.org/6254976/>
```

OWL (Web ontology language) is an extension of RDF that additionally enables the description of logical relationships and constraints between concepts as triples. In the listing above, the OWL predicate own:sameAs asserts that the DBpedia Eiffel Tower instance dbr:Eiffel_Tower is one and the same as the GeoNames Tour Eiffel instance <https://sws.geonames.org/6254976/>. This simple but critical connection enables a SPARQL query engine to begin to draw inferences that bridge across and span the breadth of knowledge graphs.

OWL

To understand better how we reason with ontologies, and indeed across a swathe of KR[2] systems, we need first to delve into a little classical logic and deduction.

Logic and deduction An important logical distinction is made between *syntax* and *semantics*. Consider the sentence, "The Eiffel Tower is in Paris." A *syntactic* analysis would focus on the terms of the sentence: the sentence contains two proper nouns (starting with uppercase letters), a verb, and a spatial preposition. The *semantics*, or meaning, of the sentence can only be determined by its context. If "Eiffel Tower" is the name of the iconic French landmark, and "Paris" refers to France's capital city, then the sentence has a clear meaning and is in fact true. However, if "Paris" refers to the small town of just over 1000 inhabitants in Monroe County, Missouri in the US, again the semantics are clear although this time the sentence is false. In this *denotational* view of semantics, the meaning of a term is determined by reference to the domain entity or relationship it denotes. The meaning and possibly the truth value of a complex proposition are determined by the meaning and structuring of its terms.

syntax

semantics

denotational semantics

Consider the three sentences:

The Eiffel Tower is a building in Paris.
All buildings in Paris are French.

The Eiffel Tower is French.

The proposition "The Eiffel Tower is French" below the line (the *conclusion*) follows from the sentences above the line (the *premises*) by *deduction*. We can note that the process is quite general, and independent of the meanings of the individual terms. The general form of the deduction above is:

conclusion

premise

x is a *y*.
All *y*s are *z*s.

---

x is a *z*.

This particular deduction is an example of a classical *syllogism*, whose principles were codified by Aristotle in around 350 BC. The deduction is said to be *valid* in the sense that it preserves truth: if the premises of a valid deduction are true, then the conclusion is also expected to be true. Note that not all forms of reasoning are deductively valid, however (see Box 9.3 on page 361).

syllogism

deductive validity

In general our logical premises may be constructed from at least five different building blocks:

Constants which denote specific individuals, such as "Paris" or "France";
Variables which stand as placeholders for individuals, such as x, y, and z;
Connectives such as $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\implies$ (implies), and $\iff$ (equivalence, if and only if);
Quantifiers namely $\forall$ ("for all ...") and $\exists$ ("there exists ..."); and

Predicates such as "Building(x)" (x is a building) or "In(y,z)" (y is located in z), which describe the properties of, or relationships between entities (as already encountered in the context of triples).

Together these building blocks can be combined into *sentences* as the basis of *first order predicate logic*. We can see them in action in our deductive inference rewritten formally as:

sentence

first order predicate logic

$$\frac{\text{Building(Eiffel Tower)} \wedge \text{In(Eiffel Tower, Paris)} \quad \forall x\ \text{Building}(x) \wedge \text{In}(x, \text{Paris}) \implies \text{French}(x)}{\text{French (Eiffel Tower)}}$$

The collection of RDF triples that populate our knowledge graphs constitute a *knowledge base* of simple logical sentences. Using OWL, that knowledge base can be augmented with richer logical rules, such as the OWL owl:sameAs predicate discussed above. Other OWL and RDF predicates, such as owl:disjointWith, owl:ReflexiveProperty, and rdfs:subClassOf likewise lead directly to additional logical sentences in the knowledge base. Deductive reasoning can then be used to derive answers to questions that are not explicitly stored in our knowledge base. Deduction is a purely syntactic process and is therefore amenable to computing and automated reasoning. In fact, automated deductive reasoning capability is integrated into most SPARQL query engines. As a result, reasoning about queries usually happens "on the fly," as a query is executed.

knowledge base

For example, the following query asks the knowledge base "Is the Eiffel Tower a person?" In SPARQL, ASK queries are similar to SELECT queries, but only return "true" or "false" answers.

```
PREFIX gn: <http://www.geonames.org/ontology#> # GeoNames prefix
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> # RDF prefix
PREFIX dbo: <http://dbpedia.org/ontology/> # DBpedia ontology prefix

ASK {?x gn:name "Tour Eiffel"; # Retrieve the GeoNames "Tour Eiffel" instance
        rdf:type dbo:Person . # Is "Tour Eiffel" a dbo:Person?
}
```

The query above evaluates to "false," even though this information is nowhere explicitly stored in our knowledge base. Instead, the query follows a chain of deductive reasoning to arrive at the answer, intuitively as follows:

```
# GeoNames instance 6254976 is named Tour Eiffel
<https://sws.geonames.org/6254976/> gn:name "Tour Eiffel"

# DBpedia Eiffel Tower is the same instance as GeoNames 6254976
dbr:Eiffel_Tower owl:sameAs <https://sws.geonames.org/6254976/>

# DBpedia Eiffel Tower is also an instance of the class of buildings
dbr:Eiffel_Tower rdf:type dbo:Building

# Buildings are a subclass of architectural structures
```

```
dbo:Building rdfs:subClassOf dbo:ArchitecturalStructure

# Architectural structures are never people
dbo:ArchitecturalStructure owl:disjointWith dbo:Person
```

In contrast, if we ask SPARQL the question, "Is the architect of the Eiffel Tower a person?" then we can deduce the affirmative with the query below:

```
PREFIX gn: <http://www.geonames.org/ontology#> # GeoNames prefix
PREFIX rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> # RDF prefix
PREFIX dbo: <http://dbpedia.org/ontology/> # DBpedia ontology prefix

ASK {?x gn:name "Tour Eiffel" ; # Retrieve the GeoNames "Tour Eiffel" instance
        dbo:architect ?y . # Retrieve the "Tour Eiffel" architect
     ?y rdf:type dbo:Person . # Is the "Tour Eiffel" architect a dbo:Person?
}
```

The mechanics of how a SPARQL reasoning engine is able to perform this "on-the-fly" reasoning—required to satisfy queries like those above—is beyond the scope of this book. One may imagine, though, that the number of possible deductions from even a modest set of premises may be very large. The ability to automatically reason with the billions of facts in LOD (linked open data) today represents a major achievement in AI, and indeed in human history. However, first-order logic is sufficiently expressive as to make it provably impossible to define a general automated procedure to answer all questions we might conceivably ask of our first order predicate knowledge base.[9] SPARQL reasoning engines deal with this issue by restricting the scope of automated logical reasoning to carefully constructed "fragments" of first order predicate logic that can be efficiently answered by machines. Thus, it is always possible to construct queries about our knowledge bases that cannot be answered. However, in practice today's Semantic Web technology can be very effective in reasoning about LOD.

[9] Indeed, Alan Turing's work, also discussed in Box 9.1 on page 350, has played an important role in proving this limitation and more generally in understanding the limits of automated logical reasoning.

9.1.4 Principles of ontology engineering

Before moving on to explore more logical reasoning with spatial data, we dwell briefly on the question of what makes a *good* ontology, and how to design one. There are many situations where it is useful to be able to develop an ontology for a specific domain. The Semantic Web is a vast and rapidly expanding knowledge base. In many cases, adding new facts to this global knowledge base is as simple as describing your data using the terms and concepts already defined in existing ontologies. However, in other cases, the required terms and concepts may not yet exist and may necessitate new terms and concepts to be defined.

In general we can identify at least five desirable properties of "good" ontologies:

Box 9.3: Induction and abduction

We have seen the deductive reasoning process "preserves truth" and is said to be (deductively) valid as a consequence. Deductive validity is the main reason why deduction is so important and underpins most automated reasoning today. However, there are two other forms of inference that are not guaranteed to preserve truth but are nevertheless important in automated and in particular human reasoning. *Induction* is the reasoning process of inferring general rules from evidence. Suppose we have as before the premise that "The Eiffel Tower is a building in Paris" and now add as a premise that "The Eiffel Tower is French." An example of induction would be to use those premises as evidence to support the inference that "All buildings in Paris are French." Of course, more evidence beyond a single building—perhaps adding the Arc de Triomphe and the Pompidou Center—may in turn give us more confidence that our inductive inference is correct. However, in most cases no matter how much evidence we add, we can never be *certain* that the induced conclusion is valid. There is always the possibility that a counterexample lurks just around the corner (such as the American Embassy in Paris, which like most embassies has extraterritorial status and consequently is not French). *Abduction* is the process of selecting the most likely explanation from the available premises. For example, if we know that "The Eiffel Tower is French" and "All buildings in Paris are French" we might adduce that "The Eiffel Tower is a building in Paris." Once again, such inference does not preserve truth—consider, for example, the same abductive reasoning using the premises "Marie Antoinette is French" and "All buildings in Paris are French". However, inductive and abductive reasoning are used to great effect almost continuously by humans. So advancing the ability of computers to achieve similar feats is one of the long-term goals of AI.

Congruity The most important principle for any ontology is to *reuse* existing vocabularies and concepts: good ontologies aid in linking data together by building upon established terms, definitions, and concepts.

Modularity Complementing congruity and reuse, good ontologies are *modular* in the sense that they describe carefully scoped and bounded domains in a way that is accessible and understandable by others, not only by the individuals or expert communities who developed them.

Consistency Good ontologies are *consistent*, avoiding circularity in definition and internal contradictions, and helping users to classify instances unambiguously.

Clarity Ontologies should avoid basic *errors* in definition, such as failing to distinguish between entities versus their identifiers (cf. Section 2.2.2); objects in the world versus the digital artifacts that refer to them; and things in the world versus our knowledge of those things.

Openness Ontology engineering is fundamentally about describing the semantics of data in a way that promotes reasoning about linked data. Hence, making this data *openly* available is fundamental to the vision of linked open data (LOD) and the Semantic Web.

These principles can help designers in deciding between different options when constructing ontologies and LOD. Figure 9.7 contains a simple example of using linked data to represent a famous meeting in history between nursing pioneers and polymaths Mary Seacole and Florence Nightingale. These contemporaries, whose lives chart many interesting parallels as well as divergent experiences, are known to have met only once, briefly in Üsküdar in Turkey during the Crimea War on 8 March 1855. In constructing an ontology of the meeting, the most important step is to reuse established ontologies. The sim-

[10] http://semanticweb.cs.vu.nl/2009/11/sem/

ple event model[10] (SEM, prefix sem) is a leading ontology for describing the general structure and vocabulary of events (Hage & Ceolin, 2013). Ontologies such as SEM that are designed to be used across many different domains are called *upper level ontologies*. The SEM is built around four key classes: Place, Time, Actor that describe the where, what, and who of an Event. The knowledge graph in Figure 9.7 contains a new class, Meeting, as a subclass of Event (a meeting "is an" event). As in previous examples, the knowledge graph also links to the GeoNames (prefix gn) and Basic Geo (prefix geo) ontologies to describe the meeting location (SEM Place), Üsküdar, and DBpedia (prefix dbr) to identify the historic figures: Mary Seacole and Florence Nightingale.

upper level ontology

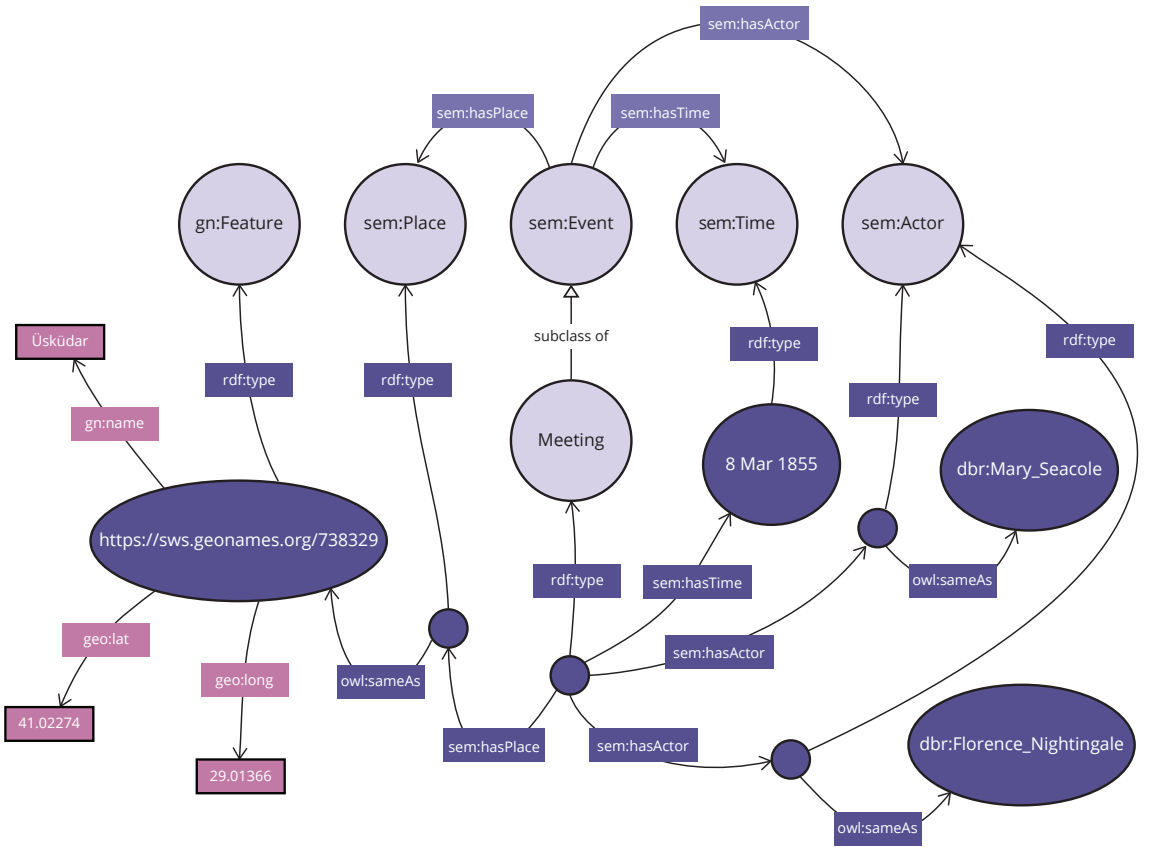


Figure 9.7: A knowledge graph representation of the meeting between Mary Seacole and Florence Nightingale on 8 March 1855 at Üsküdar (Scutari) in Turkey

When building ontologies for specific domains, such as historic meetings, it is generally advised to follow a structured process, such as below, after (Noy & McGuinness, 2001):

1. *Reuse*: Every ontology engineering project should begin by identifying the existing upper-level and domain ontologies relevant to the domain.
2. *Enumerate*: Next, enumerate the terms relevant to the domain, taking special care to be consistent in usage, for example, alert to synonyms and homonyms.

3. *Structure*: In step three, structure the terms into a hierarchy, taking care to ensure each instance of a subclass ***A*** "is an" instance of superclass ***B***.
4. *Relate*: With the structure taking shape, define the properties of each class and the relationships to literals and other classes.
5. *Iterate*: Ontology engineering is an iterative process of revision and refinement, which benefits from a diverse team of developers that includes subject matter experts, ontology engineers, and generalists.

The textbook by Kendall & McGuinness (2019) is highly recommended to those readers interested to delve deeper than would be possible in this book's whistle-stop tour of ontology engineering.

9.2 Qualitative spatial reasoning

A distinction is often made in reasoning between *quantitative* and *qualitative* approaches. An approach is generally referred to as *quantitative* if it is based on analysis of numerical (interval or ratio) data. By contrast, an approach is usually referred to as *qualitative* if it is based on analysis of classifications and ordering (nominal and ordinal data). Hence, qualitative reasoning is concerned with the discrete, imprecise, and non-numerical properties of space and time.

quantitative

qualitative

There are in general three main reasons to be interested in qualitative representation and reasoning with spatial and temporal data:

1. Qualitative properties are *simple*, usually involving small, discrete representations that are inherently tolerant to imprecise human knowledge. For example, navigating instructions are much more understandable using "left" or "right" directions rather than precise bearings or angles (e.g., 276°).
2. Qualitative properties *supervene* on quantitative properties, in the sense that qualities can always be derived from quantities. For example, a bearing of 071° can be represented qualitatively as "East" or to the "right."
3. The boundaries between qualities usually correspond to salient discontinuities for humans. For example, "left" and "right" are intrinsically connected with our embodied physical experience of the world.

supervenience

Qualitative spatiotemporal reasoning (QSR) is an area of study within KR[2] that has developed, over the past 40 years or more, a variety of logical systems for representing and reasoning about qualitative spatial and temporal objects and relations.

qualitative spatial reasoning

9.2.1 Point algebra

The *point algebra* of Vilain & Kautz (1986) is certainly the simplest and arguably the most elegant of all QSR logics. The point algebra describes three qualitative relations between two points in time (Figure 9.8). We write

point algebra

before(A,B) if A comes before B in time; *after*(A,B) if A comes after B; and *same*(A,B) if A and B are the same point in time. In common with most qualitative representations, the different relations are chosen so as to be *jointly exhaustive pairwise disjoint* (JEPD): one and only one of these relations will necessarily hold between any two points in time.

jointly exhaustive pairwise disjoint

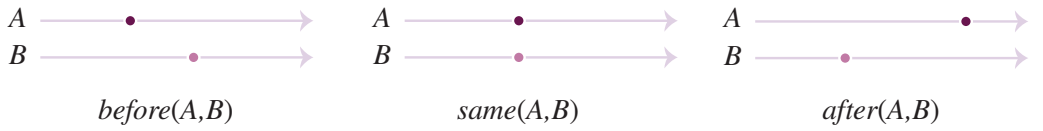


Figure 9.8: The three relations in the point algebra of Vilain & Kautz (1986)

As we will by now be expecting in KR^2, representation is only half the story. The aim of the representation is to enable logical reasoning over qualitative relations. In the point algebra, we may immediately notice that for any three time points A, B, and C if *after*(A,B) and *after*(B,C) then we can deduce that *after*(A,C). We might also notice some other inferences, such as if *after*(A,B) then *before*(B,A). Unfortunately, not all combinations of relations help drive inference. In particular, *after*(A,B) and *before*(B,C) then any of *after*(A,C), *before*(A,C), or *same*(A,C) are possible.

We can capture all such logical inferences in a single table, shown in Table 9.3. For example, if *before*(A,B) (first row of data in the table) then we can deduce the *converse*, that *after*(B,A) from the entry in row 1 column 5 in Table 9.3. If additionally we know that *before*(B,C), then we can infer that *before*(A,C) from row 1 column 2. In general, for three objects X, Y, and Z, the inference $R(X,Y) \wedge P(Y,Z) \implies Q(X,Z)$ is called the *composition* of relations R, P, and Q.

converse

composition

Composition	*before*	*after*	*same*	Converse
before	*before*	*before, after, same*	*before*	*after*
after	*before, after, same*	*after*	*after*	*before*
same	*before*	*after*	*same*	*same*

Table 9.3: Composition and converse for the point algebra

From these foundations, it is possible to construct and reason about more complex configurations of points. For example, Figure 9.9 shows a network of point algebra relations between five points, $A \ldots E$. Based on the supplied relations in Figure 9.9 (solid arrows) it is possible to deduce the other, unknown relations between points, including:

- *after*(B,A), the converse of the supplied *before*(A,B);
- *before*(A,E), the composition of *before*(A,B) and *before*(B,E); and
- *after*(D,A), the composition of *after*(D,C) and inferred converses *after*(C,B) and *after*(B,A).

Some relations though may still be undetermined, such as the relation between E and C in Figure 9.9, which may be any of *before*, *after*, or *same*.

The network of qualitative relations for a specific configuration of objects is called the *constraint network*. The task of automatically reasoning with such

constraint network

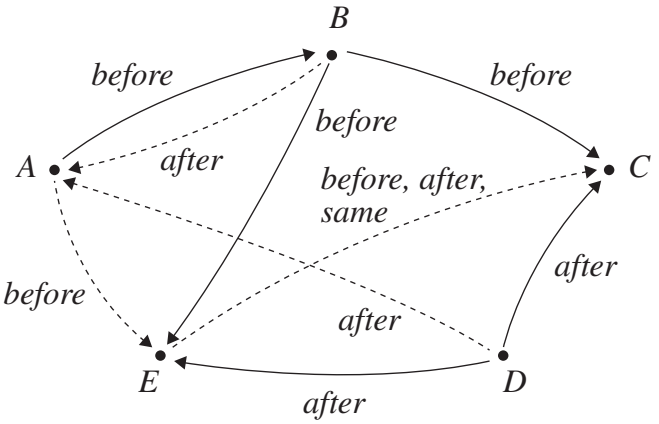


Figure 9.9: Partial constraint network for a point algebra configuration

qualitative relations can be reduced to the question of whether a set of given relations (constraints) is consistent with one another, termed the *constraint satisfaction problem* (CSP), and is computable in $O(n^3)$ time complexity for the point algebra.

constraint satisfaction problem

9.2.2 *Interval algebras*

A natural step following the point algebra is to consider the relations between time *intervals*, rather than points. Allen's *interval algebra* (1983, 1984) defines 13 JEPD relations between two time intervals (Figure 9.10).

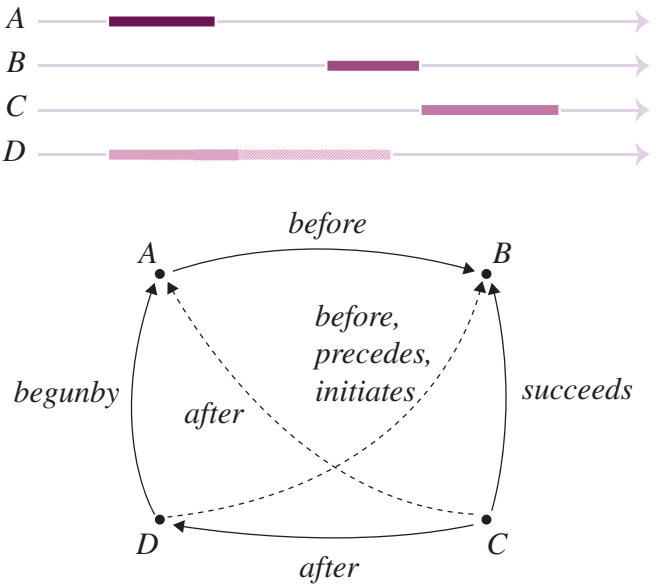
interval algebra

Figure 9.10: The 13 relations in the interval algebra of Allen (1983)

Although Allen's interval algebra defines many more relations than the point algebra, reasoning through constraint networks using composition and converse can proceed in much the same way as for the point algebra.

Figure 9.11 illustrates an example configuration with its associated constraint network. Note that the relationship between B and D is under-determined: it must be one of *before*(D, B), *precedes*(D, B), or *initiates*(D, B), but it is not possible to say which given the available information.

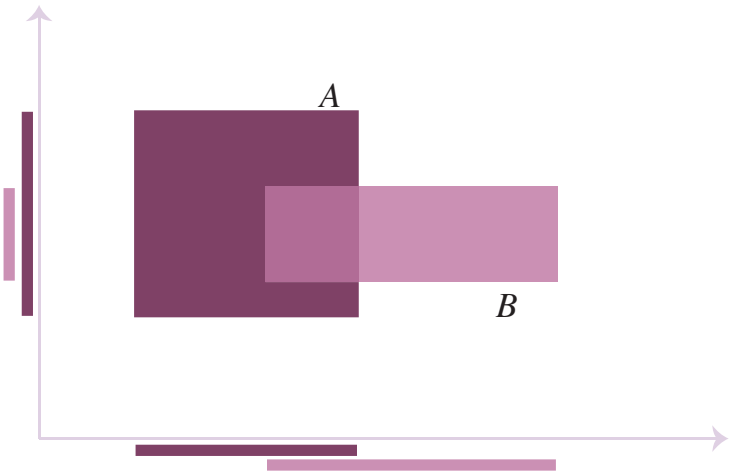
Figure 9.11: Constraint network for an interval configuration



rectangle algebra

The interval algebra can also be applied directly in two dimensions to give a set of JEPD region relations between two axis-parallel rectangles. The result, called the *rectangle algebra*, defines $13 \times 13 = 169$ relations as combined pairs of interval algebra relations. For example, the configuration in Figure 9.12 may be described as *initiates*−*contains*(A, B) (or conversely *terminates*−*during*(B, A)).

Figure 9.12: Example configuration of two regions in the rectangle algebra



Intuitively, such reasoning proceeds as for the point algebra, above. However, in fact the constraint satisfaction problem is not tractable for the full interval and rectangle algebras (nor indeed for the other full qualitative spa-

tiotemporal logics below). A mathematical solution to this problem is to identify restricted subsets of relation sets that lead to logics that are again tractable. However, each qualitative spatiotemporal logic demands its own unique subsets, and the process of identification can present significant challenges to mathematicians. In practice, reasoning with the full logics usually proceeds by computing a "local" constraint satisfaction for all possible sets of three objects instead of seeking the "global" CSP solution. This heuristic approach, termed *path consistency*, is tractable and can be computed in $O(n^3)$ time. Unfortunately, the heuristic is not guaranteed to be correct in all cases.

path consistency

9.2.3 Region connection calculus

Perhaps the most famous of all qualitative spatial logics is RCC, the *region connection calculus*. In the context of RCC, the word "region" may be thought to include any definition of a spatial region, including both continuous areas and discretized representations, such as a polygon or set of pixels in a raster.[11]

region connection calculus

[11] In fact, regions in RCC can have much wider, and possibly non-spatial interpretations, including categories in conceptual spaces (Gärdenfors & Rott, 1995).

RCC is founded on *Clarke's calculus of individuals* which begins with a binary connection relation between regions. We write $C(X, Y)$ as shorthand for "region X is connected to region Y." The connection relation is reflexive and symmetric, satisfying the following axioms:

Clarke's calculus of individuals

1. For each region X, $C(X, X)$
2. For each pair of regions X,Y, if $C(X, Y)$ then $C(Y, X)$

The surprising part of Clarke's calculus is the next step. It turns out that many of the set-oriented and topological relations between spatial objects we encountered in Chapter 3 may be derived using only the minimal machinery above. Table 9.4 shows the construction of various RCC region relations based solely on the connection relation. The final column also highlights the correspondence to the eight 4-intersection model (4IM) relations already summarized in Table 3.3.

The eight RCC relations with corresponding 4IM relations (*DC*, *EQ*, *PO*, *EC*, *TPP*, *TPPi*, *NTTP*, *NTTPi*) form a JEPD set of relations, called RCC-8. Interestingly, there is another JEPD subset of five RCC relations, called RCC-5: *EQ*, *PO*, *PP*, *PPi*, and *DR*. Both RCC-8 and RCC-5 are widely used in spatial reasoning. The RCC-5 relations provide less fine-grained distinctions at the region boundaries (*DR* encompasses both *DC* and *EC*; *PP* encompasses *TPP* and *NTTP*; *PPi* encompasses *TPPi* and *NTTPi*).

Reasoning in RCC is based on composition and converse, shown in Table 9.5, exactly as for all the spatial logics we encounter. Hence, from $PPi(X, Y)$ and $PO(Y, Z)$ we can deduce either $PO(X, Z)$ or $PPi(X, Z)$ holds, for example. Neither RCC-5 nor RCC-8 are tractable in general, and so the path consistency heuristic applied to the constraint network is commonly used in practice.

rowcolorLightestPrimaryRelation	Definition	Description	4IM relation
$DC(X,Y)$	$\neg C(X,Y)$	X and Y are disconnected	X disjoint Y
$P(X,Y)$	$\forall Z.C(Z,X) \implies C(Z,Y)$	X is a part of Y	
$O(X,Y)$	$\exists Z.P(Z,X) \wedge P(Z,Y)$	X overlaps Y	
$EQ(X,Y)$	$P(X,Y) \wedge P(Y,X)$	X equals Y	X equals Y
$PO(X,Y)$	$O(X,Y) \wedge \neg P(X,Y) \wedge \neg P(Y,X)$	X partially overlaps Y	X overlaps Y
$PP(X,Y)$	$P(X,Y) \wedge \neg P(Y,X)$	X is a proper part of Y	
$PPi(X,Y)$	$PP(Y,X)$	Y is a proper part of X	
$DR(X,Y)$	$\neg O(X,Y)$	X is discrete from Y	
$EC(X,Y)$	$C(X,Y) \wedge \neg O(X,Y)$	X is externally connected to Y	X touches Y
$TPP(X,Y)$	$PP(X,Y) \wedge \exists Z.EC(Z,X) \wedge EC(Z,Y)$	X is a tangential proper part of Y	X covers Y
$TPPi(X,Y)$	$TPP(Y,X)$	Y is a tangential proper part of X	X covered by Y
$NTPP(X,Y)$	$PP(X,Y) \wedge \neg\exists Z.EC(Z,X) \wedge EC(Z,Y)$	X is a non-tangential proper part of Y	X inside Y
$NTPPi(X,Y)$	$NTPP(Y,X)$	Y is a non-tangential proper part of X	X contains Y

Table 9.4: Defining region relations in RCC based on the connection relation

9.2.4 *Further qualitative spatiotemporal reasoning*

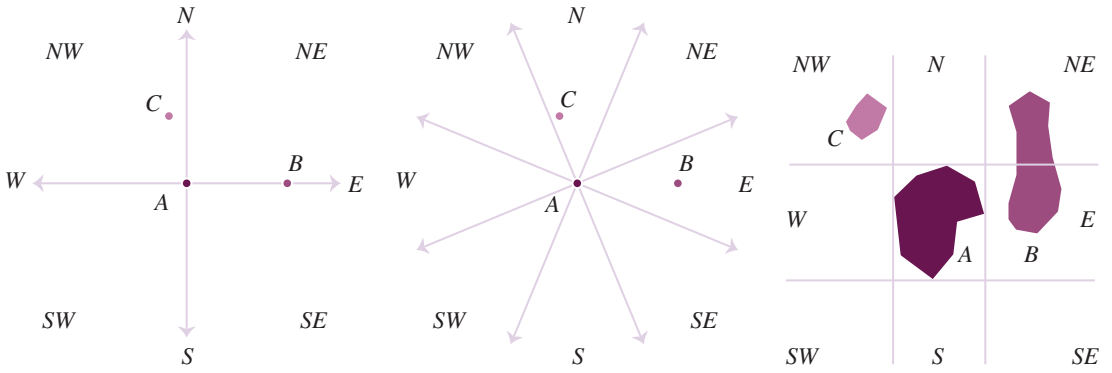
The approach to qualitative spatiotemporal reasoning described above—the definition of a qualitative representation combined with composition and converse operations for reasoning—has led to the development of a rich variety of qualitative spatial and spatiotemporal logics. For example, Figure 9.13 shows three related sets of JEPD qualitative relations for describing cardinal directions. Figure 9.13a is the projection-based *cardinal direction calculus* (CDC) for reasoning about cardinal directions between points. The representation partitions the plane into four semi-open regions (NE, NW, SE, SW) semi-bounded by four halflines (N, S, E, W), and centered on the reference point as the origin. Thus, in Figure 9.13a $E(B,A)$ (B is east of A) and $NW(C,A)$ (C is northwest of A).

cardinal direction calculus

Composition	EQ	DR	PO	PP	PPi	Converse
EQ	EQ	DR	PO	PP	PPi	EQ
DR	DR	EQ, DR, PO, PP, PPi	DR, PO, PP	DR, PO, PP	DR	DR
PO	PO	DR, PO, PPi	EQ, DR, PO, PP, PPi	PO, PP	DR, PO, PPi	PO
PP	PP	DR	DR, PO, PP	PP	EQ, DR, PO, PP, PPi	PPi
PPi	PPi	DR, PO, PPi	PO, PPi	EQ, DR, PO, PP, PPi	PPi	PP

Table 9.5: Composition and converse for RCC-5

Table 9.6 gives the complete table of composition and converse operations for the 9 projection-based CDC relations (eight directions plus the identity relation O). Hence, one can reason from Figure 9.13a and Table 9.6 that, for



a. Projection-based CDC b. Cone-shaped CDC c. CDC for regions

Figure 9.13: Three different qualitative representations of cardinal directions, after Goyal & Egenhofer (2001) and Frank (1992)

example, $NW(C,B)$ since $NW(C,A)$ (given), $W(A,B)$ (converse of supplied $E(B,A)$, row 3 of Table 9.6) and applying the relevant composition rule (NW composed with W, row 8 column 7 of Table 9.6).

Figure 9.13b summarizes an alternative cone-shaped CDC representation which partitions the plane into 8 semi-open cone-shaped regions. A different composition table from Table 9.6 is of course required to reason with the cone-shaped CDC representation.[12] Figure 9.13c includes a third version of the CDC designed for reasoning about extended spatial regions. The representation uses four axis-parallel lines to partition the space based on the minimum and maximum x any y coordinates of the reference object. Because located objects may span multiple sectors (such as region B in Figure 9.13c which occupies both the NE and E rectangular areas relative to A) a more complex set of 218 direction relations must be defined that list which of the 9 rectangular regions for a reference object a related object occupies.

[12] A useful self-test exercise is to attempt to complete the composition table for the cone-based CDC yourself, as it raises some additional challenges.

Cardinal directions are defined based on an external frame of reference, independent of any intrinsic orientation of the objects themselves. Such extrinsic reference systems are referred to as *allocentric*. Figure 9.14 illustrates three *egocentric* qualitative direction representations that rely on the intrinsic direction of the reference object.

allocentric

egocentric

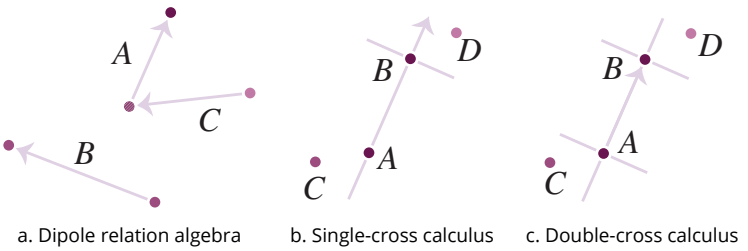
	N	NE	E	SE	S	SW	W	NW	O	Converse
N	N	NE	NE	$\mathcal{E}$	$\mathcal{U}$	$\mathcal{W}$	NW	NW	N	S
NE	NE	NE	NE	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{U}$	$\mathcal{N}$	$\mathcal{N}$	NE	SW
E	NE	NE	E	SE	SE	$\mathcal{S}$	$\mathcal{U}$	$\mathcal{N}$	E	W
SE	$\mathcal{E}$	$\mathcal{E}$	SE	SE	SE	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{U}$	SE	NW
S	$\mathcal{U}$	$\mathcal{E}$	SE	SE	S	SW	SW	$\mathcal{W}$	S	N
SW	$\mathcal{W}$	$\mathcal{U}$	$\mathcal{S}$	$\mathcal{S}$	SW	SW	SW	$\mathcal{W}$	SW	NE
W	NW	$\mathcal{N}$	$\mathcal{U}$	$\mathcal{S}$	SW	SW	W	NW	W	E
NW	NW	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{U}$	$\mathcal{W}$	$\mathcal{W}$	NW	NW	NW	SE
O	N	NE	E	SE	S	SW	W	NW	O	O

Note: $\mathcal{N} = NE, N, NW$; $\mathcal{E} = NE, E, SE$; $\mathcal{S} = SE, S, SW$; $\mathcal{W} = SW, W, NW$; $\mathcal{U} = N, NE, E, SE, S, SW, W, NW, O$

Table 9.6: Composition and converse for the projection-based cardinal direction calculus (CDC) in Figure 9.13a

dipole relation algebra

- The *dipole relation algebra* considers the direction relations between pairs of directed line segments, called *dipoles* (Schlieder, 1995). The relation between a pair of dipoles A and B is determined by whether B begins and ends to the left or right of A, and whether A begins or ends to the left or right of B. For example, with reference to Figure 9.14a, the relative direction of B from A can be described as $rlrr(A, B)$ (B begins to the right and ends to left of dipole A, and A begins and ends to the right of dipole B). The symbols s and e for "start" and "end" complete the possible relation options. For example, relative direction of A from C Figure 9.14a can be described as $errs(C, A)$ (A begins at the end of C and ends to right of C, and C begins to the right of A and ends at the start of A).
- Figure 9.14b shows the single-cross calculus of Freksa (1992b). The single-cross calculus involves a *ternary* (three-place) relation, rather than the binary (two-place) qualitative relations we have so far encountered. The calculus models the qualitative direction of a point (C or D in Figure 9.14b) relative to an observer (at point A in Figure 9.14b) facing some landmark (at point B in Figure 9.14b). For example, the two relations described by Figure 9.14b are $rb(A, B, C)$ (standing at A, C is to the right of and beyond B) and $lf(A, B, D)$ (standing at A, D is to the left and forward of B).
- Figure 9.14c shows the double-cross calculus of Freksa (1992b). The double-cross calculus further refines the single-cross calculus by enabling points that are forward of B but in front of A to be distinguished from points that are forward of B but behind A.



a. Dipole relation algebra b. Single-cross calculus c. Double-cross calculus

Figure 9.14: Three different egocentric direction relations, after Freksa (1992b), Zimmermann & Freksa (1996), and Wolter & Lee (2010)

As before, reasoning using the dipole, single-cross, and double-cross representations is based on the definition of composition and converse operations. Indeed, there are many more logics for qualitative spatiotemporal reasoning, including reasoning about movement with the *qualitative trajectory calculus*, QTC, of Van de Weghe, Cohn, Tré, & De Maeyer (2006).

qualitative trajectory calculus

9.2.5 *Conceptual neighborhoods*

conceptual neighborhood

One final aspect of qualitative spatiotemporal reasoning is worth dwelling on briefly, before we leave KR² for machine learning: *conceptual neighborhoods*. Pairs of relations in a qualitative spatial logic can be distinguished based on whether or not it is possible for an instance of one relation to transition "smoothly" into another. For example, a point moving continuously on the

plane at a distance from the origin can transition directly from cardinal direction relation *N* to *NE*, but not directly from *N* to *E* (without first passing through relation *NE* or the origin *O*, see Figure 9.13). Similarly, two regions may smoothly transition from RCC-8 *NTPP* to *TPP*, but not from *NTTP* to *NTTPi* or *PO* without first passing through some other relations.

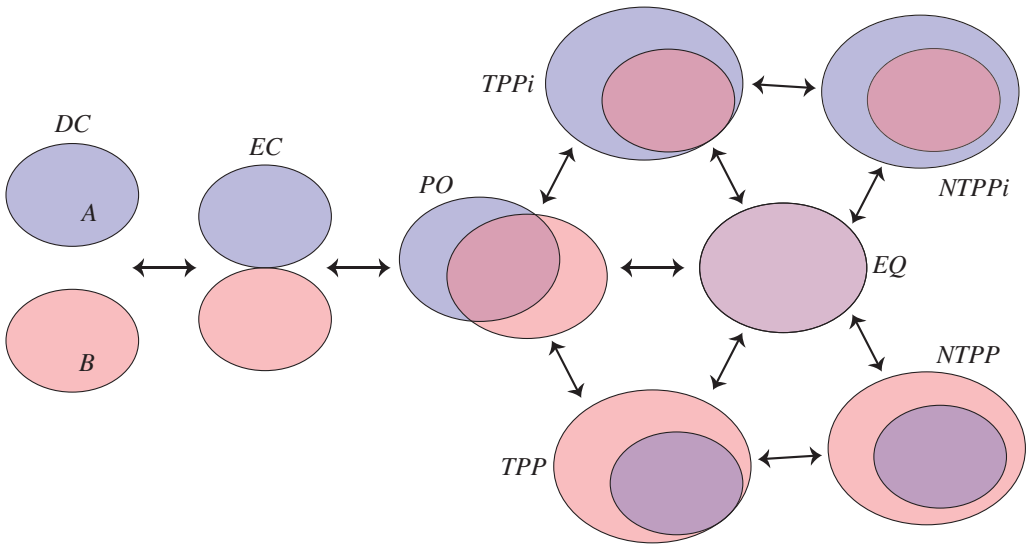


Figure 9.15: A conceptual neighborhood for the RCC-8

The idea of such conceptual neighborhoods was first described by Freksa (1992a): two relations are conceptual neighbors if they can be directly transformed into one another by continuously deforming them. Conceptual neighborhood graphs, such as that for RCC-8 in Figure 9.15, serve two important functions. First, they move qualitative spatial calculi towards describing change. Second, they provide further cognitive structure to underpin qualitative spatiotemporal logics, offering greater ability to capture human expectations about how similar certain spatial situations are.

Together, ontology engineering and qualitative spatial reasoning were selected as the two most important areas of KR² for GIS. However, it is important to be aware that there are many other KR² topics and techniques with historical (such as *expert systems*) or contemporary relevance to GIS (in particular, probabilistic reasoning techniques beyond the scope of this text, such as *Bayesian networks*, *Markov logic networks*, and *causal reasoning*), as well as closely allied topics such as agent-based modeling (see Box 9.4 on the next page). However, here we leave KR² to embark upon an exploration of the other major branch of AI: machine learning.

9.3 Machine learning and spatial analysis

The previous sections show how KR² uses logical reasoning and symbolic representations of data to mimic human problem-solving and decision-

Box 9.4: Agent-based models

Although not usually included in discussions of AI, agent-based models and agent-based modeling (ABM) have much in common with the more formal and logical techniques of KR². Agent-based models comprise three key elements. First, *agents* in an ABM are autonomous units or elements within a computer simulation. Agents in the simulation usually mirror (i.e., model) autonomous units or elements in the real world, such as vehicles, people, animals, trees, or wireless sensor motes. Second, agents in an ABM are placed into an *environment*, which similarly typically mirrors a real-world environment, such as a city or a region of land. Third, an ABM models the *interactions* that arise between agents and between agents and the environment. The ultimate objective of any ABM is a better understanding of interactions in the real-world through simulation. In particular, ABMs are used to model complex systems with emergent behaviors that go beyond our ability to understand using more analytical and numerical techniques. The emergent behavior of ant colonies is a classic example of a complex system that has been studied using ABMs. Understanding the emergent properties of ant colony behavior based on individual ant behavior—seeking food, returning it to the nest, and communicating and coordinating with other ants in the colony to achieve this—is extremely difficult to approach using purely analytical tools, such as mathematical equations. Instead, an ABM simulation of ants (i.e., agents), their environment, and the different ways they can interact with other ants (e.g., communicate) and their environment (e.g., move, leave pheromone trails) has provided significant insights into how relatively simple rules followed by individual ants can give rise to the awe-inspiring complexity of ant colony behavior (Wilensky & Rand, 2015). One of the most popular approaches to developing ABMs is the belief-desire-intention (BDI) approach to agent programming. Using BDI, each agent is equipped with *beliefs* about their environment, other agents, and the agent itself; goals the agent wishes to achieve, termed *desires*; and plans of actions that the agents can select from to achieve their goals, termed *intentions*. There are clear analogies between the knowledge represented by an agent's beliefs and the "reasoning" with desires intentions and intentions of agents in the simulation, and the more formal structures of KR². ABMs are widely used in GI science to help researchers understand many and diverse complex geographical systems (Crooks, Malleson, Manley, & Heppenstall, 2021; Heppenstall, Crooks, See, & Batty, 2012) as well as for understanding emergent computational system behavior, such as in geosensor networks (Duckham, 2013).

making capabilities. Machine learning (ML) applies statistical models to big data sets in order to recognize patterns in data and derive rules that describe data. Machine learning involves learning directly from data, without first imbuing the machine with explicit knowledge about the problem domain, in contrast to KR².

Recent years have seen a huge growth in the combination of machine learning with GIS (in particular, a growth in applications of *deep learning* discussed further in Section 9.4). One reason for this growth in machine learning is the growth in the variety and volume of spatial data. Many ML techniques are "data hungry," in the sense that the more data that is available to the machine to learn from, the better (in principle) the performance of that learning process. More spatial data means more opportunities for machines to learn from that data.

data hungry

Another reason for the growth in machine learning with GIS is the increasing availability and decreasing cost of high performance computing platforms, such as scalable cloud computing. Many ML techniques require significant computing resources to train and apply to data. As a result, the ability to access clusters of computers beyond those directly available on your desktop or in your office has transformed the range of ML techniques available to the broad spectrum of GIS professionals.

Advances in the field of machine learning itself have, of course, also contributed to this rapid growth in recent years. However, the central principles and techniques of machine learning have been well established for many years. In fact, it may be a surprise to learn that many basic spatial analyses found in every GIS are themselves examples of machine learning. Hence, many of the principles and techniques behind machine learning may already be familiar to you if you have already encountered some spatial analyses. Of course, the special nature of spatial data means that care needs to be taken when applying machine learning to spatial data. An understanding of the underlying principles of machine learning helps us to construct robust workflows and meaningful outputs when learning from spatial data.

This section unpacks the link between machine learning and traditional spatial analysis, touching on a range of fundamental spatial analysis and machine learning algorithms for spatial data. Along the way, we will revisit and consolidate some of the fundamental spatial concepts already introduced in previous chapters.

9.3.1 Machine learning tasks

Machine learning aims to equip computers to do something akin to a skill that comes naturally to humans: learning from experience. Specifically, a machine "learns" by identifying patterns in data. When the machine finds a pattern, it updates its model of the solution to reflect this new "experience." In this way, ML techniques aim to adaptively improve their performance as the number of samples available to learn from increases. At some point, when the machine has found enough patterns, it has built a model that is ready to be applied to new data it has not previously seen and to make predictions about that data.

The wide range of problems that machine learning can help solve can be categorized into three main tasks:

- *Clustering* is the task of learning "natural" groupings from within data called *clusters*, such that the similarity of data items within each cluster is greater than the similarity of items between clusters. clustering, cluster
- *Classification* is the task of learning to predict qualitative class labels that can be associated with input data. classification
- *Regression* is the task of learning to predict quantitative values associated with input data. regression

Figure 9.16 summarizes the three techniques graphically, illustrating example inputs and outputs to typical algorithms. Clustering identifies groups of similar items within undifferentiated input data. Classification and regression both construct generalized models from the input data that enable either the qualitative category (classification) or quantitative value (regression) of previously unseen data to be predicted.

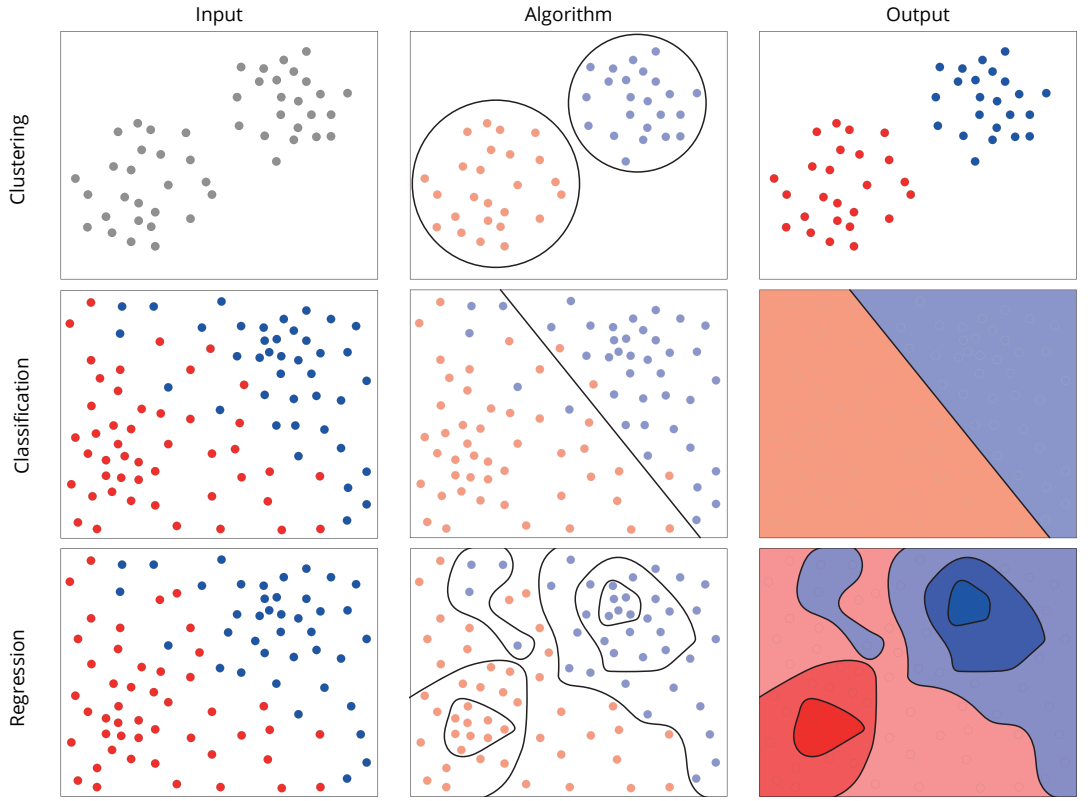


Figure 9.16: Three types of machine learning problems: clustering, regression, and classification

9.3.2 *Machine learning workflows*

As we shall see, there are many different machine learning techniques that each mimic human problem-solving abilities across one or more of the problem areas introduced above. Irrespective of the specific ML technique, we can summarize the machine learning workflow into four basic steps, shown in Figure 9.17.

Step 1: Preparation Data is the key to machine learning. However, raw data alone often needs processing, restructuring, and *cleaning*—the process of detecting and fixing or removing incorrect, duplicate, or incomplete records from a data set—before it is ready to serve as input to an ML algorithm. This first preparatory step of transforming the source data into a clean and compatible format and structure is termed *data wrangling* in Figure 9.17.

data cleaning

data wrangling

The preparation step also involves selecting the right ML technique or algorithm for solving the required problem. The latter parts of this section provide a tour of several of the most common ML techniques, starting simple and becoming increasingly complex. However, it is worth emphasizing that a more *complex* technique does not necessarily make a *better* technique. Rather, the challenge is to understand the strengths and limitations of the different choices, and how appropriate each ML technique is for use with different types of data and in solving different problems.

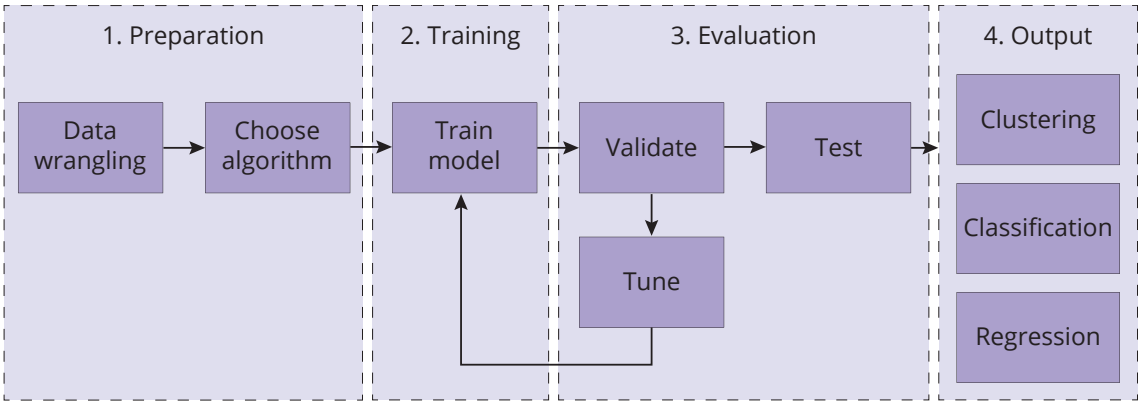


Figure 9.17: The machine learning workflow

Step 2: Training Step 2 in our workflow is at the core of machine learning: *training*. Irrespective of which ML technique is chosen, training involves conditioning an ML *model* to the data provided. Different ML techniques vary widely in how they approach this task. Two major classes of ML technique are often distinguished when discussing training: supervised and unsupervised learning.

training

Some ML techniques begin with a set of *labeled* data items with which to train the ML model. This *training data set* provides the machine with examples of the "right answer" used as the target for the training process. Traditionally, training data sets often needed to be manually labeled by a human operator. As a result, ML techniques that require a labeled data set for training are still termed *supervised learning*. Today, many data sets already include labels that can be used for training. Land cover data sets, for example, include land cover class labels that are often used in spatial applications of supervised machine learning.

training data set

supervised learning

In contrast, other ML techniques are conditioned directly on data without the need for any labels, termed *unsupervised learning*. However, even if human involvement was required at some point to generate the labels needed for supervised learning, in practice human operator intervention is relatively rarely required either for supervised or unsupervised learning techniques. Requiring no labels, unsupervised learning can be applied to a wider range of different data sets than supervised learning. However, without labels guiding the learning process, it is also more challenging to ensure accurate results and achieve meaningful outputs using unsupervised learning.

unsupervised learning

Step 3: Evaluation Whether supervised or unsupervised learning, step 3 *evaluates* how well the training process has performed. Evaluation can be separated into two components: *validation* and *testing*. Validation occurs *while* the model is being trained and is necessary in order to ensure that the learning process is correctly attuned to the specific characteristics of the input data and the problem being addressed. Validation is sometimes

validation data set

run on a reserved portion of the data—typically between 10–20% of the complete data set—called the *validation data set*. In other cases, validation simply involves checking output diagnostic parameters that accompany the training output, and provide a guide to the quality of the training.

As a result of validation, some changes to the ML settings may be indicated to ensure the training process is performing as accurately as possible. Any top-level settings used to control the training process—such as the number of iterations used in training process or the target number of learned clusters or categories—are termed *hyperparameters*. In contrast, the internal parameters of the ML model adapted by the training itself to ensure a good fit to the input data—such as model coefficients or weights—are referred to simply as *model parameters*. The process of adjusting hyperparameters in response to validation is termed *tuning*.

hyperparameters

model parameters

tuning

Finally, testing occurs *after* the model has been built. Testing is used to check that the final output has captured something generalizable about the data, and it is not simply biased towards the specific idiosyncrasies of the input data set. Testing usually aims to compare the output with some external "ground truth" data to evaluate how well the trained model has learned about the salient characteristics of the input data. In most cases, testing is run on another reserved portion of the data, typically 10–30% of the complete data set, called the *test data set*. In order to be able to rely on the results of testing, it is essential that the test data set is kept completely separate and independent from the training data set (and from the validation data set too, if used).

test data set

Step 4: Output A successful machine learning workflow will construct a trained and evaluated model that represents a solution to one of the three general problem types already enumerated: clustering, classification, and regression. In connection with clustering problems, the primary output of the machine learning workflow is the identified clusters, learned from the input data by the tuned ML model. ML techniques for solving classification problems, on the other hand, output not a data set, but the learned model itself. Assuming careful validation and testing, this output model can then be applied to the problem of classifying related data not previously seen by the model. In general, all machine learning techniques generate one of these two types of output: either a data set that encapsulates the solution to the problem, or a trained model that can subsequently be applied to related data to generate a solution automatically.

9.3.3 Evaluation and overfitting

The evaluation step of the machine learning workflow is critical to ensure that machine learning outputs are accurate and meaningful. Machines do not possess the commonsense of humans when they learn. Evaluation provides the safeguards to ensure the machine is not simply learning nonsense, and identifying patterns where none exist. Without careful evaluation, the machine can

learn patterns from noise, inaccuracies, or randomness in data, where in truth no meaningful patterns exist.

Validation metrics provide an indication of the internal quality of the learned model, in terms of the fit between the input and the output. Test metrics provide an indication of the generalizability of the model, in terms of the fit between the output and a reserved test data set. However, similar measures are often used for evaluation, whether validation or testing. Statistical measures of spread such as standard deviation, mean square error (MSE), and R^2 help to evaluate the quality of learning with quantitative data, such as regression and some types of clustering.

When evaluating learning with qualitative output, such as some other types of clustering and classification, commonly used metrics are based on errors of *commission* (items placed in a class X that should not have been) and errors of *omission* (items not placed in a class X that should have been).

errors of commission
errors of omission

Crosstabulating all of these errors by class results in a *confusion matrix*, such as that shown in Table 9.7. In Table 9.7, the confusion matrix shows that 44 red dots were correctly classified, with 3 errors of omission (red dots classified as blue) and 5 errors of commission (blue dots classified as red) in the classification example in Figure 9.16. With just two classes, all the red errors of commission are also blue errors of omission (and mutatis mutandis, all red errors of omission are blue errors of commission). More complex error interrelationships arise with more than two classes.

confusion matrix

	red	blue	total
red	44	3	47
blue	5	35	38
total	49	40	87

Table 9.7: Example confusion matrix for the classification in Figure 9.16

Overall classification accuracy is often captured as the number of correct classifications as a proportion of the total number of classified data items. For instance, taking Figure 9.16 and Table 9.7, the classification accuracy is $(44 + 35)/87 = 90.8\%$. For individual categories, three further metrics are widely used:

- True positive rate (TPR), also called *sensitivity* or *recall*, measures of the data correctly classified as a proportion of all the data classified. For example, the TPR for red in Table 9.7 is $44/47 = 93.6\%$.
- Positive predictive value (PPV), confusingly also called *precision*, is a measure of the data correctly classified as a proportion of the data that should have been so classified. For example, the PPV for red in Table 9.7 is $44/49 = 89.8\%$.
- The F1 score is a mean of TPR and PPV, calculated as $2 * \frac{\text{TPR}*\text{PPV}}{\text{TPR}+\text{PPV}}$. For example, the F1 score for red in Table 9.7 is $2 * \frac{0.936*0.898}{0.936+0.898} = 91.7\%$.

true positive rate

positive predictive value

F1 score

TPR and PPV are always reported together because they complement each other. For example, a model that classifies everything as red will always

achieve a PPV of 100% (since everything that should be classified as red will assuredly *be* classified as red) but a low TPR (since many things that should not be classified as red assuredly will be). Conversely, a model that classifies almost everything as blue with result in a low PPV but a high TPR. The F1 score provides a convenient summary of the combination.

overfitting

The term *overfitting* is used to describe the common error in machine learning where the model corresponds too closely to the input data set, at the expense of learning something generalizable about the data. Overfitting occurs when the model learns about inconsequential noise in the data set, rather than about consequential patterns. Validation and testing at the foundations of detecting overfitting: a model with high validation metrics but low testing metrics is the hallmark of an overfitted ML model.

9.3.4 *Clustering algorithms*

k-means clustering

One of the most frequently used clustering algorithms is *k-means clustering*, which partitions an input data set into a specified number of k clusters. The clusters are constructed such that the arithmetic mean (centroid) of each cluster is closer to every input data point in the cluster than to any data point in another cluster.

The k-means clustering Algorithm 9.1 begins by randomly selecting a set of k "cluster head" points from within the domain of the input point set P. Algorithm 9.1, line 1, uses the MBB (minimum bounding box) to constrain the set of initial cluster heads. The algorithm then computes the subset of points nearest to each cluster head, termed the *assignment* or *allocation step* (Algorithm 9.1, line 3). The set of k subsets of points (blocks) forms a partition of P. Next, the algorithm calculates the arithmetic mean (centroid) of each block, termed the *update step* (Algorithm 9.1, line 4). The alternating assignment and update steps iterate until the blocks stabilize to the optimal solution, found when the cluster heads no longer move after each iteration.

Algorithm 9.1: Computing the k-means of an input point set P

Input Set of points P, desired number of clusters k

1: randomly select k points $c_1, ..., c_k$ (cluster heads) such that $c_i \in \text{MBB}(P)$
2: **repeat**
3: partition P into k blocks, where block i contains all points nearest to c_i
4: recompute cluster heads $c_1, ..., c_k$ as the centroid of points in block i
5: **until** cluster heads $c_1, ..., c_k$ are optimal (i.e., no longer change)

Output Blocks $P_1, ...P_k$ that partition P

Figure 9.18 summarizes the algorithm diagrammatically for an input data set of 37 points and a cluster number $k = 3$ (Figure 9.18a). Three randomly chosen cluster heads are introduced in the assignment step (Figure 9.18b). The update step (Figure 9.18c) then updates cluster heads as the centroid of the identified clusters. Once the cluster heads reach their optimal location, the clusters themselves will stabilize (Figure 9.18d), and further iterations will not result in any further movement of cluster heads. Algorithm 9.1 is known

as *naïve* *k*-means clustering because it is intractable even for 2D sets of input points. Happily, there are many more efficient heuristics used in practice to solve *k*-means.

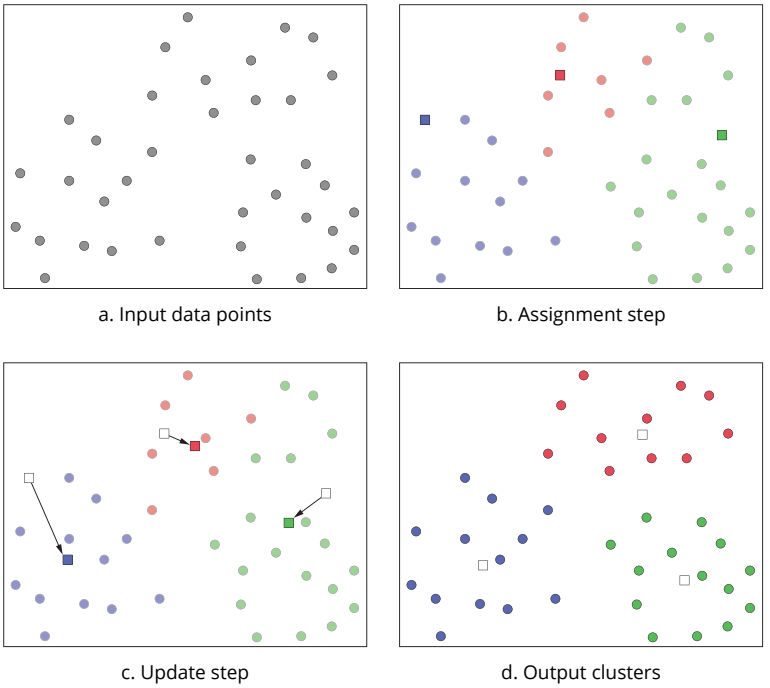


Figure 9.18: Four stages of the naïve *k*-means clustering algorithm, in Algorithm 9.1. Cluster heads are indicated with square points.

The *k*-means clustering algorithm "learns" the optimal location clusters that minimize within-cluster and maximize between-cluster distances. Tuning the algorithm usually involves tweaking the number of clusters *k*, in response to diagnostic parameters such as the statistical spread of points around cluster heads. For example, plotting the sum of squared distances between points and their cluster heads against different *k* can help identify the smallest *k* that captures most of the variation in point locations (termed the *elbow method* of validation). Testing can be performed on a reserved, randomly selected subset of the data, with the output clusters compared in terms of their closeness, giving some indication of the level upon which the identified clusters are dependent on the specific input data points.

elbow method

As might be expected, the results of *k*-means clustering are strongly dependent on the required number of clusters, *k*. A further weakness of *k*-means clustering is a sensitivity to outliers, which will always be allocated to a cluster and so tend to "drag" cluster heads away from a "natural" centroid. Many other clustering algorithms have been proposed that seek to identify clusters with different characteristics. One of the clustering algorithms most frequently used with spatial data is DBSCAN (*density-based spatial clustering of applications with noise*). DBSCAN identifies groups of more densely packed points, and it is able to exclude outlier points that do not fit well in any cluster (Figure 9.19). In addition to the set of input points, DBSCAN requires two input parameters: a minimum number of points in any cluster (to ex-

DBSCAN

clude very small clusters of, say, 2 or 3 points), and a neighborhood size ϵ that controls the maximum threshold distance from a point to some (at least one) other point in the same cluster.

A simplified version of DBSCAN is shown in Algorithm 9.2. A first incomplete cluster is initialized with a single, randomly chosen seed point (Algorithm 9.2, lines 2–3). DBSCAN then finds the yet-unvisited neighbors within distance ϵ of any points in the incomplete cluster (lines 5 and 15–17). Identified neighbors are added to the cluster (line 12), as well as being removed from the set of unvisited nodes (line 13). The cluster is complete when no further neighbors of nodes in the cluster can be found (line 6). Clusters bigger than the minimum cluster size m are stored in the result set before iterating; clusters smaller than m are discarded as outliers (lines 7–8). Finally, the algorithm randomly selects another unvisited point to seed a new cluster (lines 9–10) iterating until all nodes have been visited (line 14).

Algorithm 9.2: DBSCAN algorithm

Input input nodes V, minimum cluster size m, neighborhood distance ϵ

```
1:  set unvisited points U ← V
2:  randomly select points n in U
3:  update U ← U − {n} and set new cluster N = {n}
4:  repeat
5:      M ←GetNeighbors(U,N,ε)
6:      if M = ∅ then
7:          if |N| ≥ m then
8:              store copy of N in result set R
9:          randomly select point n in U
10:         update U ← U − {n} and set new cluster N = {n}
11:     else
12:         update N ← N ∪ R
13:         update U ← U − R
14: until |U| > 0
15: function GetNeighbors(U, N, ε)
16:     return {u ∈ U|∃n ∈ N.δ(u, n) ≤ ε}
17: end function
```

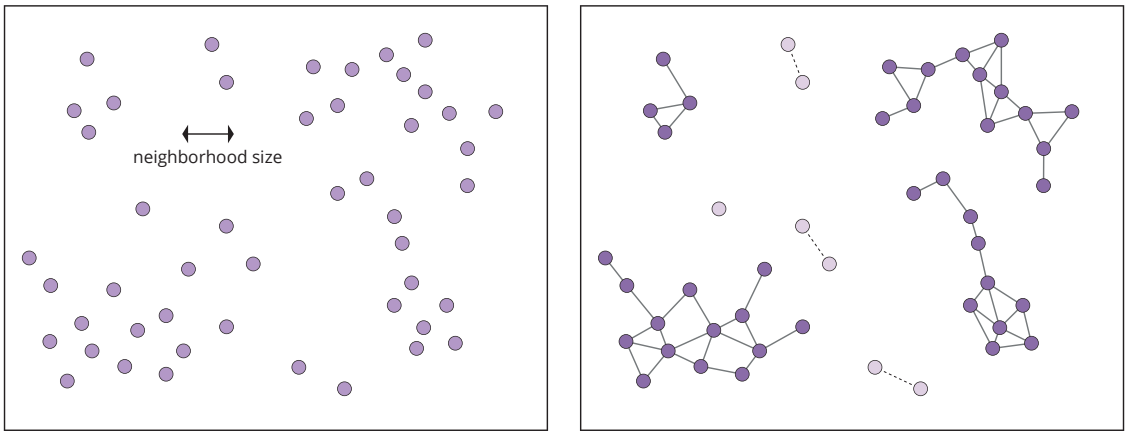
Output Result set of clusters R

9.3.5 *Classification algorithms*

Decision tree learning is a foundational ML technique for classification. Despite being a machine learning technique, decision tree learning has much in common with the knowledge representation approaches to AI encountered earlier in this chapter. *Decision tree learning* involves the construction of a *decision tree* that models the key decisions behind the classification process. A decision tree is a directed acyclic graph (see Section 2.4.1) where:

decision tree learning

- each non-leaf node in the decision tree represents a logical test that can be performed on the data;



a. Input data points

b. DBSCAN clusters and outliers

Figure 9.19: DBSCAN clustering on a set of input points with minimum number of points equal to 3 and neighborhood size indicated

- a directed edge represents one potential outcome of the test indicated by the source node of that edge; and
- each leaf node represents an outcome of the classification process.

For example, Figure 9.20 shows a decision tree related to the thermal comfort of walking tours in a city. In recent years, thermal comfort has become an important issue in urban planning and design, as more walkable cities with greater thermal comfort help promote healthy active lifestyles and lower reliance on car use. The decision tree in Figure 9.20 indicates, for instance, when the weather is cool, then thermal comfort is assured. However, on a hot day, even a shady route cannot provide thermal comfort on a long walk.

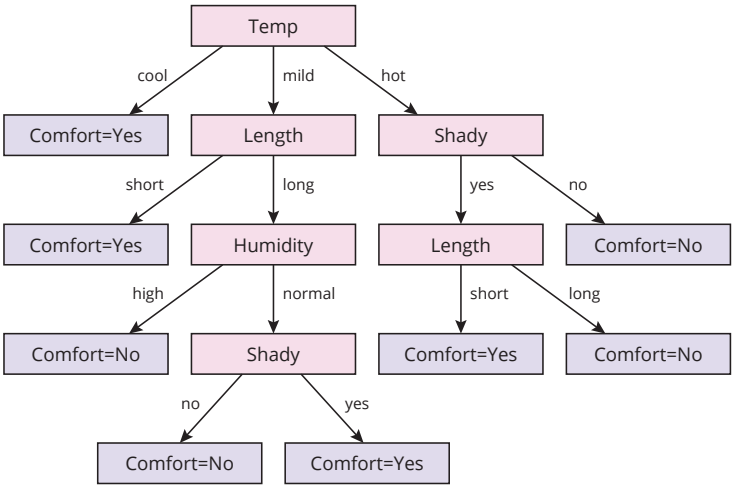


Figure 9.20: A decision tree for thermal comfort in city walking tours

Although the decision tree is a symbolic representation of knowledge that sits comfortably in the KR² branch of AI, decision trees can be learned from data. Table 9.8 provides the data that was used to automatically construct the decision tree in Figure 9.20. Each record in Table 9.8 provides information

about the thermal comfort of individual instances of different walks under different weather conditions.

Table 9.8: Example data about thermal comfort of walking tours through a city

Temp	Humidity	Shady	Length	Comfort
Cool	Normal	Yes	Long	Yes
Mild	Normal	Yes	Short	Yes
Cool	Normal	No	Long	Yes
Mild	High	No	Long	No
Mild	High	Yes	Long	No
Hot	High	Yes	Short	Yes
Mild	Normal	Yes	Long	Yes
Hot	High	Yes	Long	No
Mild	High	No	Short	Yes
Mild	Normal	No	Long	No
Cool	Normal	Yes	Short	Yes
Hot	High	No	Short	No
Mild	Normal	No	Short	Yes

ID3

The decision tree learning algorithm used to create the decision tree in Figure 9.20 from the data in Table 9.8 is among the most famous of all machine learning algorithms, called *ID3* and developed by Quinlan (1986). The ID3 algorithm listed in Algorithm 9.3 takes as input a table of data (such as that in Table 9.8) and a target attribute, which indicates the classification outcomes (in this case the `Comfort` attribute). Algorithm 9.3 initializes by creating a new, empty root node (line 1) before calling the recursive `BuildSubtree` function to build the subtree underneath that root. The `BuildSubtree` function first checks if the table contains multiple values for the target attribute (line 4). If the table contains only a single target attribute value, then the algorithm has identified decision tree leaf node that classifies with that value. If instead the table contains multiple target attribute values, then the algorithm needs some way of selecting the best attribute in the table to use next to classify the data. The ingenious and elegant answer to this problem is at the heart of the ID3 algorithm and uses the important concept of *information entropy*.

It is not difficult to appreciate that data and information are distinct concepts (refer to Section 1.3.2). The string 01101101 contains 8 bits (binary digits) of data, but no information unless we have some means of interpreting it. A red light at a railroad crossing can convey a lot of information to a motorist with only a small amount of data (assuming that the light is either on or off, then only a single bit of data). Data is easy to measure: it may be measured in bits and bytes. A more difficult question is, "How do we measure information?" Indeed, "What is information?"

information entropy

One way to describe information is as a *signal* communicated on a *channel* from a *transmitter* to a *receiver*. This simple model, termed Shannon-Weaver information theory, can help in constructing a metric for the *quantity* of information communicated, not just the quantity of data. *Information entropy*, also called *Shannon entropy*, is a measure of the quantity of information in a signal in terms of its capacity to "surprise" the recipient. "Surprise" in this

context is used to mean the smallness of the chance that a particular signal will be received. The smaller the chance, the greater the surprise, and the higher the information is valued.

For example, in Maine in the winter, there is almost always snow on the ground. A weather forecast of winter snow in Maine, therefore, is not unexpected and so does not contain much information. However, the same forecast of winter snow in Melbourne would be unusual, to say the least, and therefore might be said to contain much more information (and see Box 9.5 on the following page).

Algorithm 9.3: ID3 decision tree learning algorithm

```
Input table tab, target attribute tar
create empty root node n
BuildSubtree(tab, tar, n)
function BuildSubtree(table tab, target tar, node n)
    if only one row in: SELECT DISTINCT tar FROM tab then
        set n as: leaf node with value from SELECT DISTINCT tar FROM tab
    else
        find attribute att in table tab with the largest information gain for tar
        set n as: non-leaf node with decision attribute att
        set lst as: the list of all attributes in table tab excluding tar
        for all attribute values val in: SELECT DISTINCT att FROM tab do
            set sub as: SELECT lst FROM tab WHERE att=val
            set n' as: a new empty child node
            create new edge from n to n' labeled with decision att=val
            BuildSubtree(sub, tar, n')
end function
Output Decision tree with root node n
```

Calculating information entropy, then, involves calculating how surprising a particular signal is expected to be, to a receiver. For some variable X with values $\{x_1, ..., x_i\}$ the information entropy $H(X)$ is calculated in Equation 9.1 as follows:

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x) \tag{9.1}$$

In Equation 9.1, $p(x)$ is the probability of x, interpreted in ID3 as the proportion of all records in the table with attribute value x. For example, in Table 9.8 the attribute Comfort (8 Yes values, 5 No values, 13 records in total) has information entropy $8/13 \log_2(8/13) + 5/13 \log_2(5/13) = 0.946$. Deciding which attribute is most useful to use as a decision node in the decision tree can then be reformulated as the question of which attribute is associated with the largest *information gain* (i.e., the greatest reduction in surprisal value). Using information entropy, information gain $IG(X, a)$ associated with attribute a and variable X is calculated in Equation 9.2 as follows:

information gain

$$IG(X, a) = H(X) - \sum_{T \in \tau} p(T) H(T) \tag{9.2}$$

where τ is interpreted as the set of tables resulting from a split using attribute a and $p(T)$ the number of records in table $T \in \tau$ as a proportion of those in X.

Box 9.5: Information quantity and value

The Shannon-Weaver concepts of transmitters and receivers of signals via channels are of course metaphors. The image of information leaving the source, being transmitted through a medium (channel) to a receiver, perhaps also being the subject of degradation through noise, is compelling. In fact, it is so compelling that sometimes it is easy to forget that it is only an image, and an image with limitations. In particular, in Shannon and Weaver's elaboration of the metaphor, "value" becomes synonymous with "quantity" of information, which is measured by information entropy. However, it is easy to find intuitive examples where value is not measured this way. Suppose a burglar has determined by observation that almost every night the owners of a house forget to set their house alarm system and only remember on those rare occasions when they take their dog for a walk. On the night planned for the burglary, the lack of a dog being walked provides unsurprising but highly valuable information to the burglar. Context is the key here, and an understanding of context needs to contribute to any metric of information value. A stronger position would be that context is a necessary facilitator of information flow, and without context there is no flow (see Sperber & Wilson, 1995).

For example, the information gained by selecting attribute Temp is:

$$\left(\frac{8}{13}\log_2\frac{8}{13}+\frac{5}{13}\log_2\frac{5}{13}\right)-$$
$$\left(\frac{3}{13}\left(\frac{3}{3}\log_2\frac{3}{3}+\frac{0}{3}\log_2\frac{0}{3}\right)+\right.$$
$$\frac{7}{13}\left(\frac{3}{7}\log_2\frac{3}{7}+\frac{4}{7}\log_2\frac{4}{7}\right)+$$
$$\left.\frac{3}{13}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)\right)=0.204$$

By contrast, none of the other attributes in Table 9.8 exhibit such high information gains (the information gain for Humidity is 0.046, Shady is 0.076, Length is 0.115). Algorithm 9.3 uses such calculations of information gain (line 7) as the basis for selecting the attribute to use in each decision node (lines 8–9). Then, by splitting the table into smaller tables using the selected attribute (Algorithm 9.3, line 10–11) the algorithm continues building the decision tree recursively with a new decision edge and child node (lines 12–14).

Decision trees, such as that in Figure 9.20, are powerful and intuitive ML classification tools. However, decision trees also tend to suffer from overfitting, especially if grown to their fullest extent, necessary to classify every data item from the input data. To address this limitation, a *random forest* is a collection of decision trees grown from randomized samples of the input data, illustrated in Figure 9.21. Each sample typically uses about two-thirds of input data. However, samples are often "topped up" to be the same size as the input data set with randomly sampled duplicate entries, a sampling process called *bootstrapping*. Validation of each tree can use the data omitted from its generating sample, termed *out-of-bag estimation* (OOB).

random forest

bootstrap sampling

out-of-bag estimation

A random forest may take many hundred such random samples in order to grow hundreds of trees. After training this way, classification of unseen data can proceed using majority voting from across all the trees in the forest. Each new record encountered is classified by all the trees in parallel, with the class most frequently represented across the forest being assigned to the output.

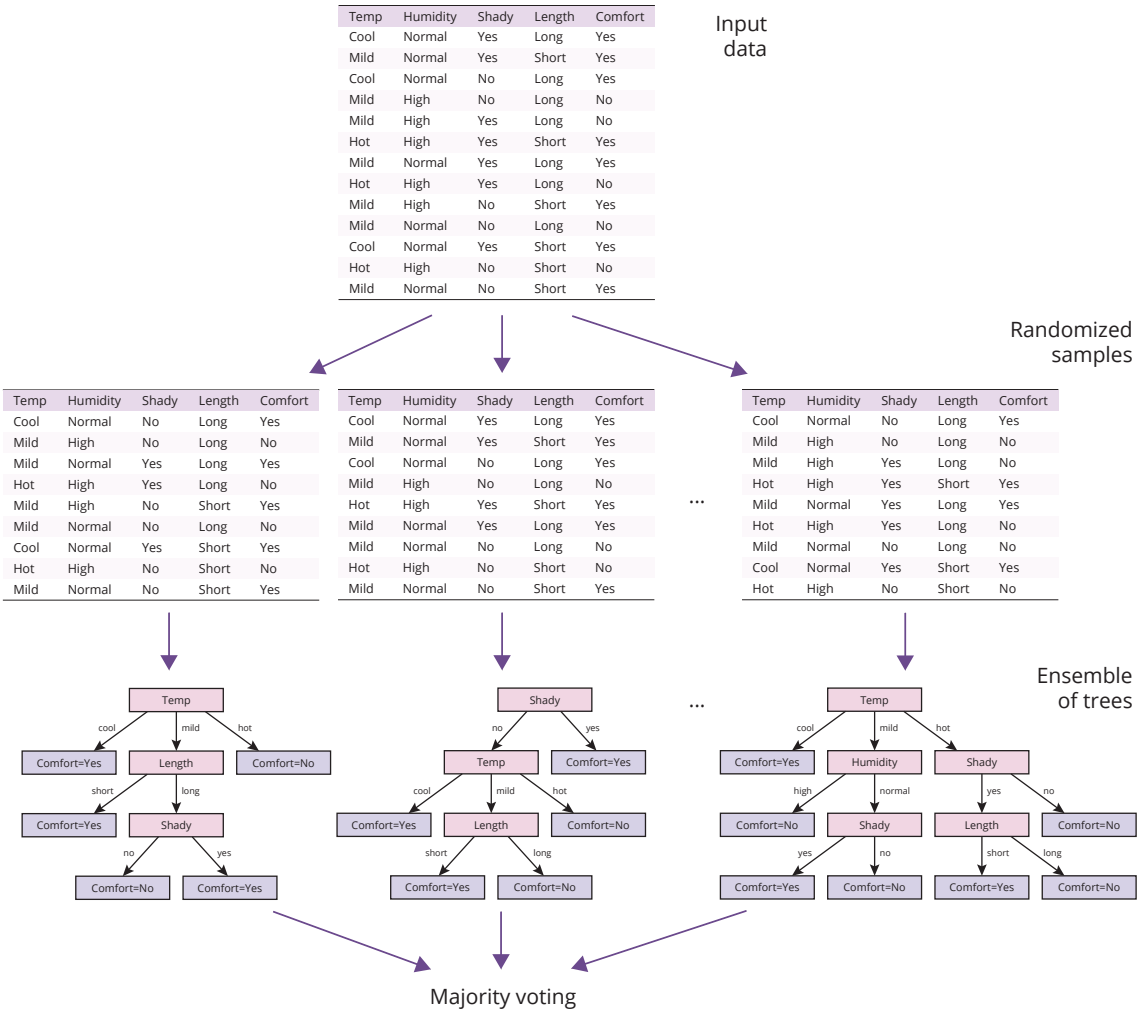


Figure 9.21: Random forest ensemble of decision trees without bootstrapping

The approach of learning multiple models from data and then recombining the set of model outputs into a final output is a common technique in machine learning, termed *ensemble learning*. In general, ensemble learning tends to increase overall stability and accuracy of learning outputs. Accordingly, random forests are an example of an ensemble learning technique, with the particular objective of reducing the tendency to overfitting common in single decision trees.

ensemble learning

Decision tree learning can also be adapted to handle continuous-valued data, by classifying quantities using data classification techniques such as quantiles, equal intervals, and mean-standard deviation encountered in the previous chapter. Moving beyond classifying quantitative data to true predictions with continuous values, however, requires regression algorithms.

9.3.6 *Regression algorithms*

Regression aims to fit a model to quantitative data, such that the model can subsequently be used to predict values that are not part of the input training data set. One of the most important techniques for spatial regression is *Kriging*, named after mining engineer and statistician Danie Krige.[13] At the root of Kriging is a construction called the *semivariogram*. The semivariogram provides a model of the spatial autocorrelation (degree of spatial interdependence) in a data set (Section 4.2.1). The semivariogram captures the degree to which "near things are more related" (Tobler's first law), and it uses that as its basis for regression.

Kriging

[13] Both the name "Krige" and the technique "Kriging" are pronounced in English with a hard "g."

semivariogram

Figure 9.22a summarizes the key features of the semivariogram as follows:

- On the x-axis, the semivariogram measures the distance between pairs of points in spatial data, termed *lag*.
- On the y-axis, the semivariogram measures the average of variation between points at that corresponding lag. Variation is measured as *semivariance*: half the average squared difference between pairs of points at a specified lag.
- The intercept of the curve with the y-axis is termed the *nugget*, and it represents the variability between points at zero lag. The curve will not usually intercept the origin. The non-zero nugget is interpreted as the level of error in measurements, or of variation below the smallest measurement granularity.
- Semivariance is expected to increase with increasing lag (because near things are more related), but only up to a certain level, termed the *sill*.
- The lag beyond which semivariance no longer appreciably increases is called the *range*. The range is the limit of autocorrelation, beyond which distance all point pairs are equally uncorrelated.

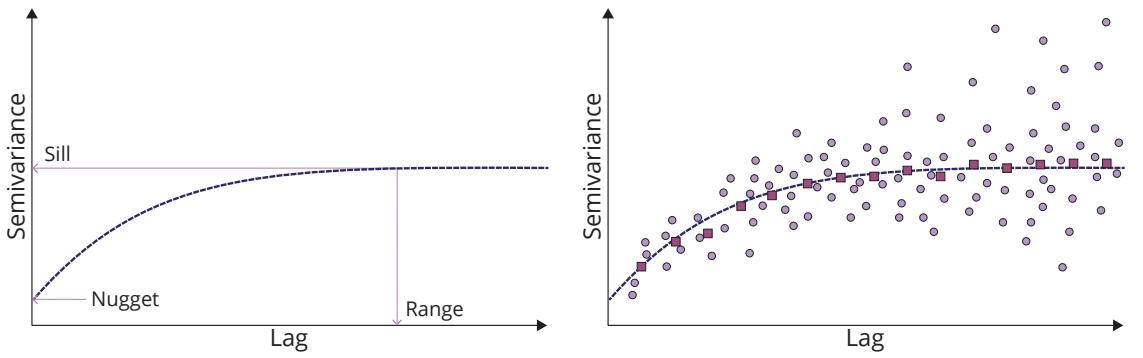
lag

semivariance

nugget

sill

semivariogram range



a. Semivariogram features

b. Training with data

Figure 9.22: The semivariogram, the basis of Kriging

Fitting (i.e., training) a semivariogram to a data set N of points involves first computing the semivariance between the sets of points at different lag

distances. For a set of pairs of points $N(h)$ at approximate lag distance h (i.e., within some small tolerance), the semivariance γ is computed as:

$$\gamma(h) = \frac{1}{2N(h)} \sum_{(i,j)\in N(h)} \left(Z(p_i) - Z(p_j)\right)^2$$

where $Z(p_x)$ is the observed value at point p_x. Figure 9.22b plots the semivariance of a set of points against lag (circular data points) with the curve (dashed line) fitted to the data using regression. Often the curve may be fitted to averaged values (square data points in Figure 9.22b) for clarity. There are many different types of curve that may be fitted to the data, from simple linear relationships to sophisticated Gaussian models and sigmoid functions. One of the most commonly used models is a modified quadratic, called a *spherical model* (as used in Figure 9.22b). The objective in all cases is to achieve the best fit between model and data. However, the choice of curve type typically has a significant impact on the training results.

The output of this training process is a model of the autocorrelation in the data, based on the structure of the semivariogram. Specifically, the predicted value $\hat{Z}$ at some new location p_0 is estimated as the sum of known values at all the other input data points, weighted by inverse distance and according to the semivariogram (i.e., so nearby data points have a much stronger influence than more distal data points). This estimate is captured by the equation:

$$\hat{Z}(p_0) = \sum_{k=1}^{|N|} \lambda_k Z(p_k)$$

where λ_k is the set of weights derived from the semivariogram based on the distance of each input data point p_k from p_0. The model can be evaluated in the usual way, for example, by reserving a test data set unused in the training and computing the MSE or R^2 deviations between test data and predictions.

Empirical Bayesian Kriging The approach described above is the most basic form of Kriging, called *simple Kriging*. An important limitation of simple Kriging is that it assumes *statistical stationarity*: that the statistical properties of the training data are constant across the study area. In contrast, we know to *expect* spatial data to exhibit *nonstationarity*—where variability itself varies spatially and there exists no such thing as an "average location" (see Box 1.7 on page 34).

statistical stationarity

Several more sophisticated extensions of simple Kriging have been developed to provide more sophisticated predictions that relax the strict requirement for stationarity, such as *ordinary Kriging*. One of the most widely used such extensions is called *empirical Bayesian Kriging* (EBK). EBK generates not one semivariogram, but many semivariograms from overlapping, randomly selected subsets of input data.

empirical Bayesian Kriging

Thus, training generates a *spectrum* of semivariograms, each one an estimate of the "true" semivariogram. The spectrum of semivariograms forms an ensemble that can be recombined into a single output model. As before, the

ensemble approach tends to provide more reliable predictions for each new location. Figure 9.23 shows the results of EBK regression analysis of point-based observations of temperature across the state of Oregon, US.

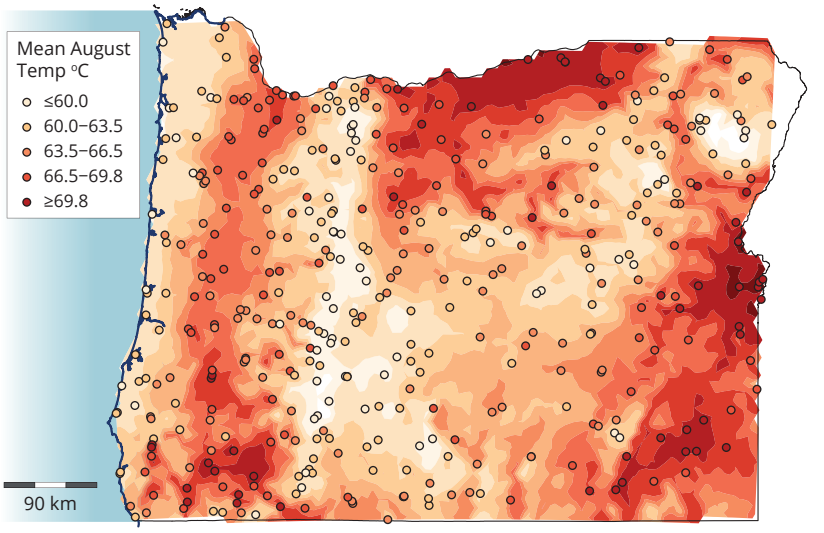


Figure 9.23: Empirical Bayesian Kriging of temperature points across the state of Oregon, US

Like all Kriging, EBK accommodates autocorrelation in spatial data. The temperature surface output by EBK in Figure 9.23 captures the spatially autocorrelated continuous variation in temperature points.[14] However, unlike simple Kriging, EBK also allows for moderate levels of nonstationarity, because the estimate of the value at a location can be based on recombining a subset of the spectrum of semivariograms from nearby locations. Hence, the level of autocorrelation used to generate the surface in Figure 9.23 from the input temperature points may vary somewhat across the area.

[14] The input points and the output surface have been classified in Figure 9.23 for cartographic clarity, but the underlying input and output data is continuous.

Geographic weighted regression One further, important regression technique capable of accommodating both autocorrelation and nonstationarity is called *geographically weighted regression* (GWR). Whereas Kriging trains a model to estimate a variable at unknown *locations*, GWR trains a model to estimate the spatially varying and unknown *relationship* between different variables.

geographically weighted regression

GWR is a spatial extension of multiple linear regression, where training data is used to construct the relationship between a dependent variable and one or more independent variables. Predicting a dependent variable y based on p independent variables $x_1 ... x_p$ with multiple linear regression has the general form:

$$y = \beta_0 + \sum_{k=1}^{p} \beta_k x_k + \epsilon$$

where ϵ is an error term and β_k are the parameters of the regression that need to be estimated. GWR additionally accounts for local, spatial variations in the relationship between variables, using the form:

$$y(u) = \beta_0(u) + \sum_{k=1}^{p} \beta_k(u) x_k + \epsilon$$

where $\beta(u)$ indicates the estimated parameter at a location u.

Multiple linear regression estimates a single, *global* relationship between the dependent variable and each independent variable based on the entire training data set. By contrast, GWR estimates the *local* relationship between dependent and independent variables for each location in the training data set. Figure 9.24 shows an example of the output of a GWR model for a regression on the Oregon temperature data already encountered in Figure 9.23. It is expected that temperature is not only autocorrelated but correlated with other environmental variables, such as elevation. Further, the degree to which temperature is correlated with elevation is expected to vary spatially, with some locations exhibiting stronger correlation than others. The GWR output in Figure 9.24 shows the spatially varying strength of this relationship. Higher coefficients (stronger relationships) are found at higher elevations; lower coefficients tend to cluster nearer to the coast, where proximity to the sea tends to dominate local weather conditions rather than elevation.

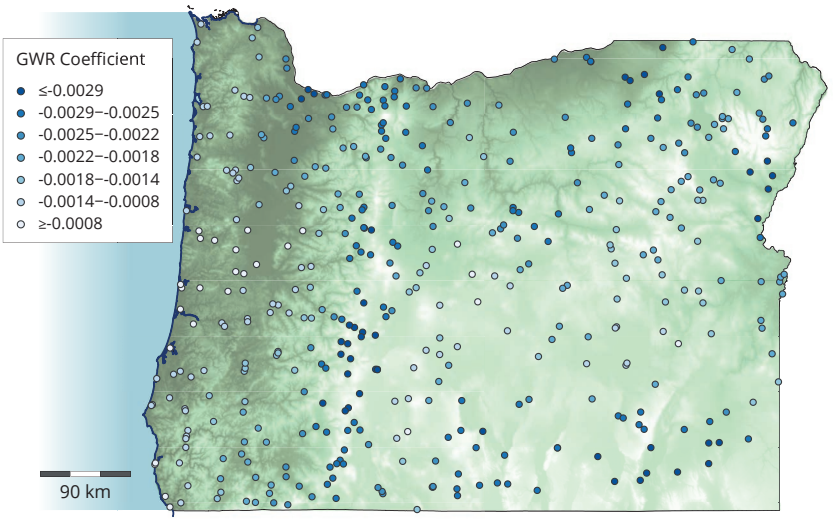


Figure 9.24: Output coefficient for geographically weighted regression of mean August temperature (dependent variable) against elevation (independent variable, lighter green indicates higher elevation) across the state of Oregon, US (cf. Figure 9.23)

The extent to which the prediction at a location is influenced by other nearby spatial locations is controlled by the *kernel bandwidth* or simply *bandwidth*. Larger bandwidths take into account more observations in the prediction, in the extreme approaching the global linear regression results. Smaller bandwidths are more strongly influenced by nearby locations, in the extreme tending towards extreme overfitting. The predictions at a particular bandwidth are usually calculated using a distance decay function, such as a Gaussian or bisquare distribution. In that way, closer locations within the bandwidth influence the estimate for a location more than more distal locations within the bandwidth.

kernel bandwidth

Bandwidths themselves can be expressed in (Euclidean) distance, termed a *fixed kernel*, or in terms of the number of neighbors, termed an *adaptive kernel* (since in this case the distance used will adapt to the density of input data points). Figure 9.25 shows an example of an adaptive bisquare kernel containing 19 points at two locations a and b. Figure 9.24 was constructed using an adaptive bandwidth of 30 neighbors, for example.

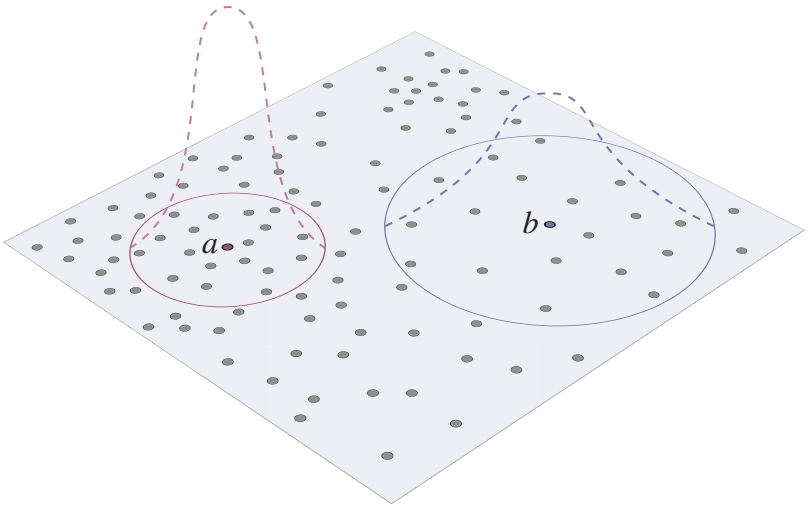


Figure 9.25: Bisquare adaptive kernel containing 19 neighbors at estimated locations a and b

The first step of any GWR modeling, then, is to select an appropriate bandwidth. Bandwidth may be estimated by an experienced human analyst or an optimized bandwidth can be computed. The most common optimization procedure is *cross-validation*, where the chosen bandwidth minimizes the sum of the square differences between the estimated and known dependent variable. More formally, the selected bandwidth h minimizes CV—the difference between observed dependent variable at each location y_i and that estimated from the neighbors within distance h, excluding the observation y_i itself, denoted $y_{\neq i}(h)$:

$$CV = \sum_{i-1}^{n} \left(y_i - \hat{y}_{\neq i}(h)\right)^2$$

The R^2 value is commonly used for evaluating the quality of a spatial regression such as GWR. The overall R^2 for the regression in Figure 9.24 is 0.74—moderately high, but perhaps might be improved using an optimized bandwidth procedure. In addition, Figure 9.26 shows the spatial variation in R^2 values output by the geographically weighted regression in Figure 9.24. Figure 9.24 highlights the spatially varying quality of the regression, of lowest quality in the southern central region, but performing well in other areas, especially the northwest of the state.

It lies beyond the scope of this book to attempt a more complete treatment of spatial analysis. The leading textbook specifically on this fascinating

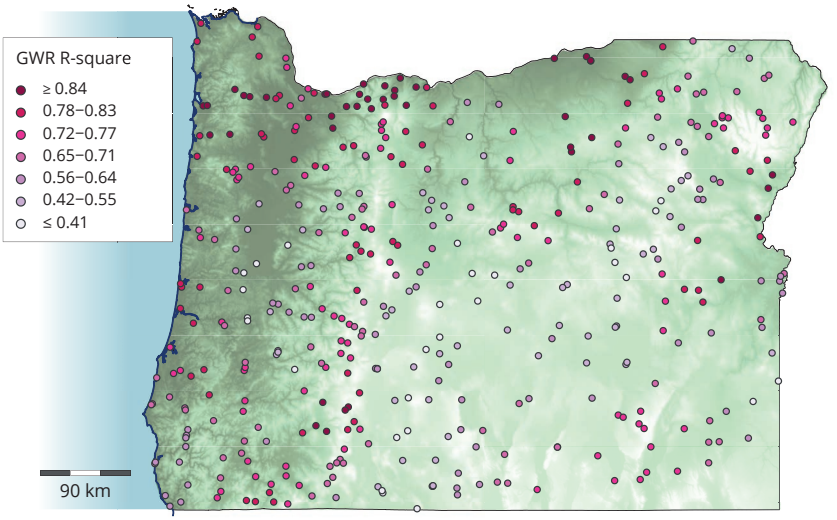


Figure 9.26: Output R^2 for geographically weighted regression of mean August temperature (dependent variable) against elevation (independent variable, lighter green indicates higher elevation) across the state of Oregon, US (cf. Figure 9.24)

topic is O'Sullivan & Unwin (2010). Other GIS textbooks, such as Burrough et al. (2015) and Longley et al. (2015), devote significant portions of the contents to the topic. A great number of other, application-oriented texts adopt a discipline-specific view, such as spatial analysis for social scientists (e.g., Rey & Franklin, 2022), or a technology specific perspective, such as spatial analysis with R (e.g., Comber & Brunsdon, 2020). The central message of our treatment of this topic, however, is that the most venerable and celebrated spatial analysis techniques are also machine learning techniques. Learning patterns from data is at the root of spatial analysis too.

9.4 *Deep learning*

The previous section provides an introduction to the huge variety of machine learning techniques that exist today, by focusing on that subset of clustering, regression, and classification techniques most frequently used with spatial data. There is one further family of machine learning techniques, however, that has had arguably more impact on GIS than any other: deep learning.

9.4.1 *Artificial neural networks*

Deep learning is founded on *artificial neural networks*. An artificial neural network (ANN), or simply a *neural network*, is a machine learning classification or regression algorithm and is commonly represented using a weighted, directed graph. The simplest possible ANN has multiple input nodes connected to a single output node by weighted directed edges as in Figure 9.27. The output node computes the weighted sum of inputs, and it then applies an *activation function* to determine what output is generated. A simple activation

neural network

activation function

function is to apply a threshold, such that a 1 is output if the weighted sum plus some bias b is above the threshold, -1 otherwise.

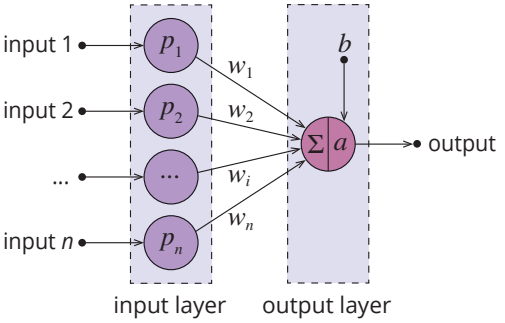


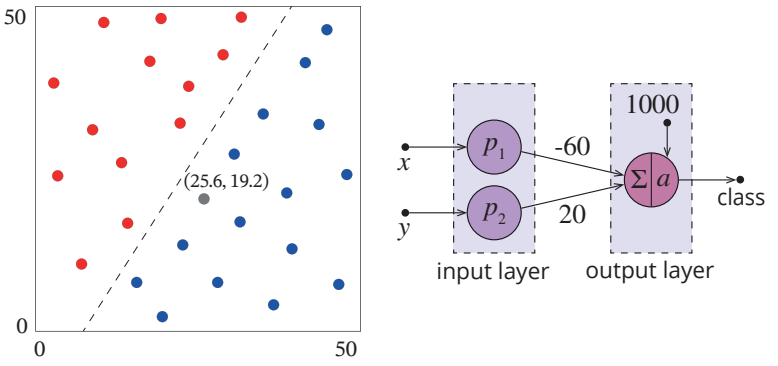
Figure 9.27: A single-layer feed-forward ANN or perceptron

The structure of the ANN in Figure 9.27 has just one layer of activating (output) nodes. The structure also only ever feeds data forward from inputs towards outputs, known as a *feed-forward network*. Accordingly, this structure is known as a *single-layer feed-forward neural network*, or more simply as a *perceptron*. The terms "perceptron" and "neural network" were coined to highlight the biological inspiration of animal neurons and brains behind these computational structures.

feed-forward network

perceptron

Despite their simplicity, even perceptrons alone can provide basic machine learning classification capabilities. Consider the classification problem in Figure 9.28a, where a training data set of points in Euclidean space has been classified into two categories shown in red (corresponding to -1) or blue (corresponding to 1) based on an implicit linear boundary between them.



a. Classified points

b. Classifying perceptron

Figure 9.28: Simple classification using a trained perceptron

The perceptron in Figure 9.28b has two inputs, corresponding to the x and y coordinates of input points. The activation function, $a : \mathbb{R} \to \{-1, 1\}$, combines the weighted sum with the bias as follows:

$$a(v) \mapsto \begin{cases} -1 \text{ if } v + b > 0 \\ 1 \text{ otherwise} \end{cases}$$

For example, applying the weights $w_1 = -60$ and $w_2 = 20$ and a threshold of $b = 1000$, the perceptron in Figure 9.28b can correctly classify all the points

in Figure 9.28a. A new point (shown in gray) with coordinate (25.6, 19.2) is classified by the perceptron as $a(-1152) = 1$ (i.e., classified as blue since $-60 * 25.6 + 20 * 19.2 + 1000 \leq 0$).

Implicitly, the perceptron is performing binary classification based on the dashed line indicated in Figure 9.28a. Training such a perceptron is effectively a matter of using the training data to learn the weights and bias needed to correctly classify the data. There are many sophisticated strategies used to train neural networks efficiently. However, a simple training strategy that illustrates the key features of neural network training is shown in Algorithm 9.4.

Algorithm 9.4: Simple perceptron training

Input n perceptron weights $w_1, ..., w_n$, activation threshold b, training data table T with $n + 1$ attributes (n features, 1 classification)

1: initialize weights $w_1, ..., w_n \leftarrow 0.0$
2: initialize activation threshold $b \leftarrow 0.0$
3: **repeat**
4: randomly select and remove one row $\langle p_1, ..., p_n, o \rangle \in T$
5: compute weighted sum $v \leftarrow \sum_{i=1}^{n} w_i.p_i$
6: classify output as $o' \leftarrow \begin{cases} 1 \text{ if } v \leq b \\ -1 \text{ otherwise} \end{cases}$
7: **if** $o' \neq o$ **then**
8: update all weights $w_i \leftarrow w_i + o'.p_i$
9: update threshold $b \leftarrow b - o'.v$
10: **until** all rows in T have been visited

Output Trained weights $w_1, ..., w_n$ and threshold b

The simple training strategy in Algorithm 9.4 begins by initializing the weights and bias with arbitrary values (lines 1–2). Next, we iterate through the training data set, selecting and removing data items to process in random order (lines 3–4). The selected training data item is classified using the (initially arbitrary) weighted sum and output in lines 5–6. If the computed classification o' does not match the training data ground truth classification o each weight w_i is updated to $w_i + o'.p_i$ (line 8) and the bias b is updated to $b - o'.v$ (line 9). Otherwise, the weights and bias are left unchanged.

Table 9.9 steps through the first 13 iterations of Algorithm 9.4 using the training data set and untrained perceptron from Figure 9.28. The example rapidly converges on the approximate weights and threshold close to those used above ($w_1 = -58.2$, $w_2 = 20.3$, $b = 926.1$).

9.4.2 *Multilayer perceptron networks*

The perceptron discussed above illustrates several features common to all ANNs, such as the graph-based structure, the classification itself using weighted sums, and the learning of weights and biases from training data. However, as might be expected, such simple ANNs have significant limitations. In particular, perceptrons are only able to learn a certain specific type

Table 9.9: Example training of perceptron in Figure 9.28b using data in Figure 9.28a and Algorithm 9.4

x	y	o	class	w_1	w_2	b	v	o'	predicted
13.1	26.4	-1	red	0.0	0.0	0.0	0.0	1	blue
19.1	48.3	-1	red	13.1	26.4	0.0	1525.3	-1	red
19.3	3.0	1	blue	13.1	26.4	0.0	332.0	-1	blue
39.0	13.3	1	blue	-6.2	23.4	332.0	69.4	-1	red
47.3	24.6	1	blue	-45.2	10.1	401.5	-1889.5	1	blue
15.4	6.5	1	blue	-45.2	10.1	401.5	-630.4	1	blue
31.1	17.4	1	blue	-45.2	10.1	401.5	-1230.0	1	blue
2.8	38.5	-1	red	-45.2	10.1	401.5	262.3	-1	red
31.3	46.8	-1	red	-45.2	10.1	401.5	-942.1	1	blue
44.3	36.6	1	blue	-13.9	56.9	-540.6	1466.8	-1	red
43.1	32.3	1	blue	-58.2	20.3	926.1	-1852.7	1	blue
23.3	38.0	-1	red	-58.2	20.3	926.1	-584.7	-1	red
3.4	22.7	-1	red	-58.2	20.3	926.1	263.0	-1	red
...	...	...	...	...	...	...	...	...	...

of classification (i.e., those where classification can be achieved with a single straight line or hyperplane, termed *linearly separable*) and are not able to classify reliably in the presence of noisy data.

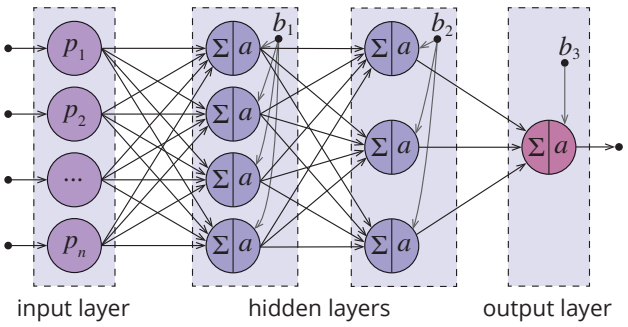
More sophisticated ANNs, however, can overcome these limitations, and indeed provide the basis of "universal" classifiers, able to approximate any continuous function. There are four main features that more sophisticated ANNs add to the perceptrons already encountered.

multilayer perceptron network

hidden layer

First, many perceptrons can be connected together into a single, combined neural network with multiple layers, termed a *multilayer perceptron network* (MLP). Figure 9.29 shows an MLP with three activating layers. In between the (inactive) input layer and the output layer, two "hidden" layers provide many more connections leading to many more possible paths between input and output. Hidden layers stage the processing of input data, with successive layers usually having fewer nodes—termed the *width* of the layer—than its predecessor. The proliferation of paths made possible by the multilayer structure enables much more sophisticated classifications to be learned than achievable with a single perceptron.

Figure 9.29: A three-layer perceptron network (weights omitted)

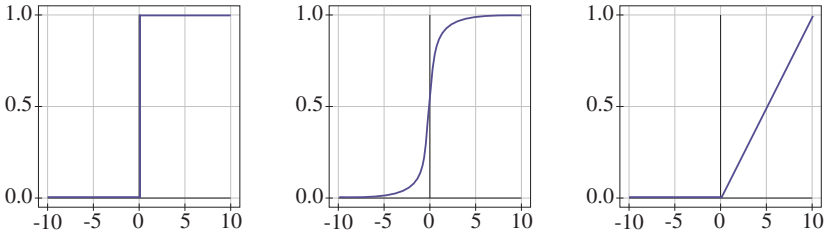


Second, MLPs can employ more sophisticated activation functions than the simple step function encountered in the previous section. Figure 9.30 shows the two most common activation functions used in preference to the

simple step function. The *sigmoid function* in Figure 9.30b is effectively a continuous differentiable analog of the step function. Returning a continuous value in the range [0.0, 1.0], rather than binary values such as {0.0, 1.0}, ensures small changes to weight and bias parameters do not lead to sudden dramatic changes in outputs, as can happen with binary step functions. The *rectified linear unit function* (ReLU) function in Figure 9.30c is also continuous, but not differentiable. Both sigmoid and ReLU functions are nonlinear, enabling the neural network to approximate nonlinear classifications. Being piecewise linear, however, the ReLU activation function can be computationally more efficient in practice.

sigmoid function

rectified linear unit function



a. step function b. sigmoid function c. ReLU function

Figure 9.30: Three common activation functions

Third, the added hidden layers in MLPs demand more sophisticated mechanisms for updating weight and bias parameters during training. In a single-layer perceptron, all classification errors can be attributed to the parameters associated with the output layer. In MLPs, updates to parameters across the hidden layers are required in proportion to the contributions of hidden nodes to output errors. A process known as *back-propagation* is used to achieve this and effect parameter updates in the hidden layers too during training.

back-propagation

Fourth, the neural networks we have encountered thus far are feed-forward and fully connected, with every node in a layer with forward connections to every node in the next layer. As we shall see in the next section, these features too are extended by true deep learning techniques.

9.4.3 *Deep learning models*

We have seen above that the number of nodes in a layer provides a measure of the *width* of a neural network. The *depth* of an ANN is defined by the number of layers in a feed-forward network, and more generally by the length of the path from input to output in neural networks where feedback loops are possible. *Deep* learning is particularly concerned with the construction, training, and use of *deep* neural networks. While there is no universally agreed upon definition of what counts as "deep," generally *deep learning* is taken to mean any ANN with paths of three or more activating nodes from input to output. Hence, at a minimum a feed-forward MLP with two hidden layers would count as deep learning.

deep learning

convolutional neural network

Convolutional neural networks Perhaps the most important family of deep neural network structures for spatial data is the *convolutional neural network* (CNN), particularly used with image-based and raster spatial data. Large images may contain millions of pixels, which would translate into billions of weights in a fully connected MLP. Even a small 8 channel, 1024 pixel square satellite image would require more than 8 million weights ($8 \times 1024 \times 1024 = 8{,}366{,}608$) for each node in an MLP.

Such wide, fully connected MLPs are not computationally efficient. More importantly, nor are they statistically efficient because we expect that most of the meaningful weights will relate nearby pixels, due to autocorrelation. CNNs achieve increase both computational and statistical efficiency by effectively restricting the connections between neural network nodes to those representing nearby localities. To achieve this, two new types of layers are introduced by CNNs:

convolutional layer
kernel
pooling layer

1. a *convolutional layer* which combines values from pixels within a specified neighborhood using an image processing *kernel*; and
2. a *pooling layer* which combines values from nearby pixels using maximum, minimum, median, or other aggregate function of all the pixel values within a specified neighborhood.

Figure 9.31 illustrates the operations embedded into the structure of these two types of layers. Starting with a simple 8×8 pixel image on the left of Figure 9.31, and placing the 3×3 kernel over each pixel in turn, a new 6×6 image is computed as the pixel-wise weighted sum of kernel and pixel values.[15] In Figure 9.31, the kernel configuration shown is known as a *high-pass filter* and is tuned to detect edges in the image. Other kernel configurations will tend to highlight other features, allowing each convolutional layer to be tuned to activate in response to certain types of image features.

[15] For example, placing the kernel in the top left of the input image in Figure 9.31, the top left pixel of the convolutional layer is computed as: $4 \times 4 + 4 \times -1 + 4 \times -1 + 3 \times -1 + 3 \times -1 = -2$.

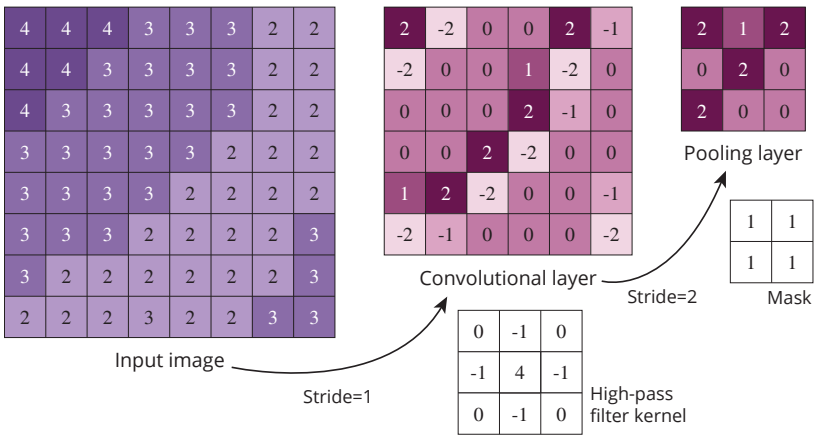


Figure 9.31: CNNs: convolution of an image using a 3×3 high-pass filter with stride 1 followed by **max** pooling using a 2×2 mask with stride 2

stride

The progress of the kernel across the image is controlled by a model parameter, termed *stride*. A stride of 1 means the kernel is applied to every

available pixel; a stride of 2 applies the kernel to available pixels in every other row and column; a stride of 3, available pixels in every third column and row, and so on.

Similar to convolution, pooling is also based on aggregation of neighboring pixels. Pooling applies an aggregation operation—such as maximum, minimum, or modal pixel value—to the set of all pixels in a neighborhood, called a *mask*. In Figure 9.31, the result of the **max** pooling operation using a 2×2 mask with stride 2 is shown in the final pooling layer. Pooling assists in summarizing and downsampling an image, reducing the size of subsequent layers and thus the number of parameters to train.

mask

Figure 9.32 summarizes a generalized CNN architecture. The input image is used to construct a set of convolutional layers (just three in Figure 9.32), each derived using different kernels. Pooling layers downsample into a set of smaller layers. Most CNNs apply a series of convolution and pooling layer pairs, not shown in Figure 9.32. The final pooling layers are then recombined by feeding into one or more fully connected hidden layers and ultimately the output layer, as we have seen already used with MLPs. Training a CNN begins in the usual way with back-propagation through the hidden layers. Pooling has no trainable parameters, and so no learning takes place with pooling layers. Importantly, back-propagation through convolutional layers updates the *kernel* weights rather than individual ANN edge weights, which dramatically reduces the number of parameters to learn, when compared with a fully connected MLP for example.

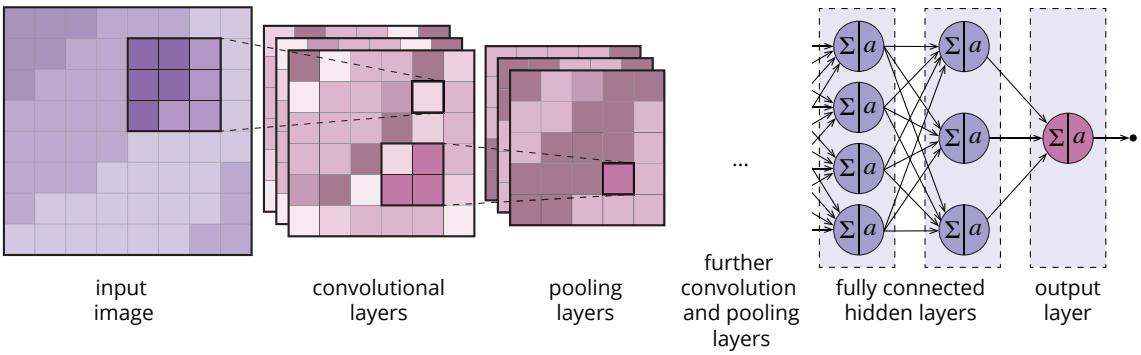


Figure 9.32: Summary architecture of a CNN, with three (3×3) convolutional filters, 2×2 pooling filters, and two fully connected hidden layers

Graph neural networks CNNs operate on the implicit neighborhood structure of raster images. The kernels and masks that underpin the convolutional and pooling layers in CNNs are the mechanism by which that neighborhood is encoded within a CNN. *Graph neural networks* (GNNs) generalize the concept of the CNN to the arbitrary neighborhood structures found in abstract graphs. Rather than use pixel neighborhoods, GNNs are structured around learning with encoded graph neighborhood structures.

graph neural network

Graphs are fundamental to GIS, as we have seen in Chapter 2, Chapter 3, and indeed in this chapter too (in connection with knowledge graphs, Sec-

tion 9.1). New applications of GNNs are emerging every day, and we do not delve further into this fast-moving area. Nevertheless, it is possible to identify three broad types of tasks tackled using GNNs:

- node-level tasks, such as node classification and clustering;
- edge-level tasks, such as edge classification and link prediction; and
- graph-level tasks, such as graph classification and graph matching.

Recurrent neural networks Recurrent neural networks are our final stop on this tour of deep learning. Unlike feed-forward networks, a *recurrent* neural network (RNN) contains feedback loops, where node outputs may connect back into the current or previous layers as well as forward towards the output (see Figure 9.33). Looped connections equip the neural network with an analog of working memory, where previously processed training data may influence subsequent training steps. As a result, RNNs are especially well adapted for learning from data where the sequence or timing of data items is significant.

recurrent neural network

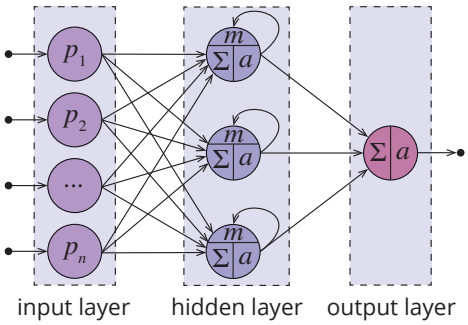


Figure 9.33: RNN: recurrent neural networks allow loops, where feed-forward and feedback links between nodes are permitted

In practice, to enable recurrence, nodes accepting feedback connections are equipped with a memory to store the node state from one iteration to the next. The capacity to store node state between iterations is indicated with an $\boldsymbol{m}$ in the nodes with feedback links in Figure 9.33. As for CNNs and GNNs, the back-propagation algorithm has to be modified to accommodate successive parameter updates, termed *back-propagation through time* (BPTT). In short, BPTT effectively "unrolls" the RNN into a sequence of MLPs (as shown in Figure 9.34), back-propagating parameter updates through the chain, before rolling back up the RNN ready for the next iteration.

back-propagation through time

Deep learning has become fertile ground for research and development of new models and techniques, of which GNNs and RNNs are just two examples. Other examples include GANs—*generative adversarial networks*—which pit two neural networks one against the other. The two networks compete against each other in training: one (the *generator*) learns to generate new data with the same statistical properties as the training data; the second (the *discriminator*) learns to spot the differences between real and generated data sets. Further innovations and new applications are to be expected in the coming decade, especially as bigger and bigger data sets containing billions of

generative adversarial network

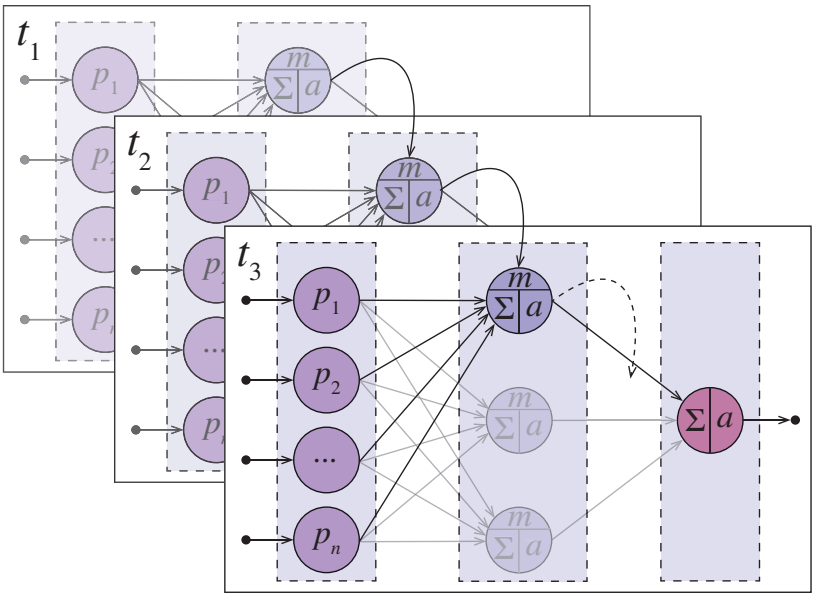


Figure 9.34: Back-propagation in RNNs can be achieved by unrolling the network into a chain of MLPs

items, such as the data sets used to train neural networks for natural language processing (NLP) like GPT (Brown et al., 2020), become more and more common.[16]

Reflections

As we have seen, AI is not *only* about machine learning; and neither is machine learning—nor AI more broadly—a *new* frontier for GIS. This chapter unpicks the deep and longstanding connection between AI and GIS. The strong link between machine learning and spatial analysis, highlighted in Section 9.3, is especially significant. In addition to the spatial analysis textbooks already cited (including Burrough et al., 2015; O'Sullivan & Unwin, 2010), Li et al. (2016), Li (2020), and Li & Arundel (2022) offer accessible, recent introductions and summaries of the variety of spatial analysis techniques in the context of AI and big data. The synergies between GIS and AI, and the unique problems and opportunities thrown up by their combination, have led to its emergence as a distinct area of inquiry, sometimes called *GeoAI* (Mao, Hu, Kar, Gao, & McKenzie, 2017).

Undoubtedly, recent years have seen unprecedented growth in the development and practical application of deep learning in particular. These advances are enabling similarly rapid advances in applications to GIS. CNNs (convolutional neural networks), for example, already have well-established spatial applications to feature extraction from satellite imagery (e.g., Romero, Gatta, & Camps-Valls, 2016). Such applications are helping to automate spatial analyses that previously required many months of manual labor (for example, to

[16] ChatGPT—released after this sentence was written, but before this book was published—serves to underline the rapid pace of innovation in deep learning. The remarkable ability of ChatGPT to generate seemingly intelligent text guarantees that its release in late 2022 will forever remain a milestone in the history of machine learning. Trained on vast corpora of human written text and tasked with prediction (regression) of word frequencies, ChatGPT and similar tools are, nevertheless, statistical models constructed using deep learning techniques fundamentally no different to those explored in Section 9.4. The ability of such statistical models of word frequencies to mimic human writing led renowned AI researcher Emily Bender and coauthors to coin the term "stochastic parrots" (Bender, Gebru, McMillan-Major, & Shmitchell, 2021).

GeoAI

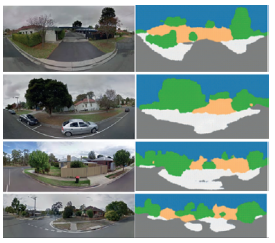


Figure 9.35: Deep learning to segment street panoramas into landscape types, after Sun et al. (2021)

automatically segment street view panoramas into landscape types—sky, trees, buildings, roads, grass pixels, Figure 9.35). Further spatial application areas of deep learning—such as urban analytics (see Li, Zhao, & Zhong, 2022) and "GeoQA" (geographic question-answering systems, e.g., Mai, Janowicz, Zhu, Cai, & Lao, 2021)—are likewise certain to continue to expand.

Nevertheless, the inherent structure and rich semantics of spatial and temporal information also demands abstract and symbolic representations and reasoning, such as ontologies and linked open data (Mai, Janowicz, Yan, & Scheider, 2019). While the ML and KR^2 branches of AI adopt markedly different approaches to intelligence, both are well suited to GIS applications. Indeed, combinations of ML and KR^2 are possible and offer the potential to capitalize on the advantages of both (e.g., Du, Wang, Ye, Sinton, & Kemp, 2022; Duckham et al., 2022; Li, Ouyang, & Zhang, 2022). However, as we witness wider use of powerful GeoAI techniques in society, the challenges are not only technical, they are ethical. The next chapter tackles head-on not simply questions of what we *can* we do with GeoAI, addressed in this chapter, but also questions of what *should* we do.