Data Literacy

William John Holden

Contents

Chapter 1

Introduction

1.1 Parameters and statistics

Statistics are the foundation of most data mining, machine learning (ML), and artificial intelligence (AI) methods today. A *statistic* is an estimate of a *parameter*, which is a characteristic of an entire *population*. Statistics are calculated from taking *samples* (subsets) from the population.

For example, suppose we wanted to find the height of the tallest mountain in the world. We might sample $n = 100$ mountains at random from an almanac. Suppose the tallest mountain in our sample is Mount Fuji. Mount Fuji, the tallest mountain in Japan, is 3776 meters tall. We can conclude that the tallest mountain in the world is *at least* 3776 meters tall.

Our estimate is unfortunately quite low. Mount Everest in Nepal, the *highest* mountain in the world, stands 8849 meters above sea level. Mauna Kea in Hawai'i, the *tallest* mountain in the world, stands 4207 meters above sea level and another 6004 meters below. Our estimates of population parameters, *statistics*, generally improve with larger sample sizes, and many statistical methods provide a *margin of error* quantifying sampling error.

One might use statistics to create a *model* to explain a population, based upon sampling data. Models can be useful both for describing the population and also for forming predictions.

1.2 Levels of measurement

There are four distinct *levels of measurement* that a value may fit. *Nominal* data is simply names or categories, with no concept of order or distance. A movie might be animated or live-action: these are simple categories or order.

Another example might be the film's genre (children, comedy, action, romance, documentary, etc).

Ordinal data has ordering but not distance. Ordinal data might be represented as ordered categories or as numerals, though these numerals do not provide meaningful addition and subtraction. The ratings of a film (G, PG, PG-13, R, and so on) form a ranking, but addition is meaningless (does G + PG-13 = R?) and our concept of distance is weak at best. Another example of ordinal might be the rankings the films receive at an awards ceremony, where one film is the winner and another is the runner-up.

Interval data is numerical data with a concept of distance but not multiplication. The year when a film was produced is an example of interval data. If two films were produced in 2000 and 2010, then it makes sense to say one was made ten years later, but we would not say that the latter film is $2010/2000 = 1.005$ times the first.

Ratio data is numerical data with both distance and multiplication. The gross earnings of a film is an example of ratio data. If the 2000 film earned one million dollars and the second earned two million dollars, then it makes sense to say the second film earned double the first.

Name	Operations	Type
Nominal	$=, \neq$	Categories
Ordinal	$<, >$	Ordered categories
Interval	$+, -$	Numbers with distance
Ratio	$\times, \div$	Numbers with meaningful zero

Interval data might be initially confusing to distinguish from ratio data. One indication is the absence of a meaningful zero. Does zero degrees Celsius or Fahrenheit mean the absence of temperature? No, these measurements are simply points along a scale. Twenty degrees Celsius is not "twice" ten degrees Celsius; multiplication is not defined on interval data.

Grid coordinates might be another example of interval data. One can calculate the distance between two grid coordinates, but we would not say that coordinate 1111 is "half" of coordinate 2222.

Data might be represented in numerical formats when some operations do not make sense. Suppose a political scientist encoded voter's political party as "1", "2", "3", and "4". Is "2" an intermediate value between "1" and "3", or are these actually nominal data where the only arithmetic operations are $=$ and $\neq$? AI methods sometimes make incorrect assumptions about data that domain experts can easily prevent.

1.3 Discretization

Measurements with arbitrarily many decimal digits of precision are *continuous*, whereas measurements with finite steps in between (including categories) are *discrete*. For example, when driving along a road, the house numbers (150 2nd Street, 152 2nd Street, 154 2nd Street…) are discrete; there is no intermediate value between 150 and 151. On the other hand, the grid coordinates associated with each address are continuous; one could (theoretically) specify grid coordinates to the nanometer.

It can be useful to combine continuous measurements into discrete categories. An example might be one's birth date and birth year. No one knows their birth *instant* with subsecond precision. Rather, the year, year and month, or year, month, and day are almost almost always enough information. We even combine years into groups when discussing generations and peer groups. Combining a range of birth years into generational categories is an example of *discretization*.

1.4 Missing values

In practice, sets of data (a *data set*) are often missing values. Different programming languages have substantially different syntax and semantics for representing and handling missing values.

As a small exercise, open Microsoft Excel and enter the values 1, 2, 3, and 5 into cells A1, A2, A3, and A5. Leave cell A4 blank. In cell A6, enter the formula `=PRODUCT(A1:A5)`. The result is $30 = 1 \cdot 2 \cdot 3 \cdot 5$. Excel did *not* treat the missing value as a zero.

Now change cell A4 to `=NA()`. `NA` means "value not available", an explicit indication that a value is not given. The product in cell A6 should update to `#N/A`, which explicitly tells us that there is a problem in the calculation.

Now change cell A4 to `=1/0`. Both cells A4 and A6 should both say `#DIV/0!`, a fault telling us that a division by zero has made further calculation impossible.

Error values propagate from source data through intermediate calculations to final results. If we enter a formula into A7 referencing A6, such as `=SQRT(A6)`, then we will find the same faults in A7 that we see in A6.

Structured Query Language (SQL) databases use the symbol `NULL` to denote missing values. One might build the database *schema* (the structure of the database) to explicitly forbid `NULL` values. For example, `CREATE TABLE Run (Name TEXT NOT NULL, Time INTEGER NOT NULL, Distance REAL NOT NULL)` defines a table *schema* where each of the three columns must be specified. Many programming languages (including C, Java, and JavaScript) also use the term `null` for variables that do not reference any specific value.

Many programming languages support a `NaN` ("not a number") value in error conditions. One might encounter `NaN` when dividing by zero, subtracting infinities, and parsing non-numeric words as numbers. Comparisons with `NaN` can be confusing, such as `NaN == NaN` returning *false*.

Some programming languages will automatically *initialize* variables with some zero value. Other languages give some `Undefined` value to uninitialized variables. Still other languages raise an error if no explicit value is assigned to a variable.

1.5 Strong/weak and static/dynamic typing

Values come in many forms: categorical and numerical, ordered and unordered, discrete and continuous, defined and missing. *Types* can be used to constrain variables to allowable values and applicable operations.

For example, suppose a database indicates how many cars a person owns. It makes no sense to own a fractional or negative car, so we might find an existing type (in this case, whole numbers) or define some new type to model the domain.

Some programming languages offer *dynamic* types that implicitly change the type (*cast*) of values to operate correctly. In Chrome, Edge Chromium, or Firefox press F12 to open the developer console and enter the following into the JavaScript console:

```
>> "5" * 5
<- 25
```

Characters inside quotation marks (`"5"`) are called *strings* and are ordinarily used for text, but JavaScript automatically parses `"5" * 5` as the product of two numerical values and returns `25`.

JavaScript is notoriously inconsistent.

```
>> "5" + 5
<- "55"
```

The resulting string, `"55"`, is the *concatenation* of two strings – perhaps not what one expects.

Many languages and environments seek to automatically parse values. Microsoft Excel and the Python programming language are also dynamic. Other languages, such as Java and Go, are more strict with values and do not automatically change values, especially when the conversion might be "lossy" (where information might be lost, such as approximating the exact value of π as 3.14, or rounding 3.14 to 3, or even changing 3.0 to 3). These languages have both *strong* and *static* typing: the programmer must specify the type of each variable, and lossy type conversions require an explicit cast.

Excel does provide some basic functionality to set number *formats*, but this feature might not stop one from confusing one type of data for another. Excel uses *weak* typing that does prevent one from using unexpected values. Data analysts can benefit greatly by using the appropriate types for the values in their problem.

1.6 Tables, lists, and data frames

Tables of data are structured in *columns* and *rows*, where the rows represent the *individuals* or *observations* in the data set and the columns represent the *features*. For example, a table of employee names might have two columns (the given and surnames) and ten rows, where each row represents one of the ten employees.

In computer science, the terms *list* and *array* both refer to single-column tables, but with different internal memory representation. The distinction is usually unimportant to data analysts.

Scientific languages, such as Julia and R, often use the term *data frame* (or *dataframe*) as their method for representing tables of data. Data frames often provide rich syntax for row-wise and column-wise operations. By contrast, in an object-oriented language, such as Java and JavaScript, the idiomatic representation of a table is likely an array of objects.

1.7 Vectors and matrices

We now quickly mention the terms *vector* and *matrix* here to disambiguate them from other terms already defined.

Arrays, lists, and columns containing numeric data may sometimes be represented with *vectors*. Likewise, tables and data frames might be represented with *matrices*.

A vector is a quantity with both magnitude and direction, often consisting of two or more elements.

$$\mathbf{x} = (x_1, x_2, x_3)$$

The above vector $\mathbf{x}$ has three components and length $\sqrt{x_1^2 + x_2^2 + x_3^2}$.

A matrix is a collection of vectors used for linear transformations. For example, the three-component *identity matrix*

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

has the property

$$\begin{aligned}
I\mathbf{x} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\
&= \begin{pmatrix} 1 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 \\ 0 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 \\ 0 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 \end{pmatrix} \\
&= \mathbf{x}
\end{aligned}$$

Vectors and matrices form the foundations of *linear algebra*, a rich and powerful branch of mathematics that produces many of the results needed for modern statistics, ML, and AI methods.

Remember that the different in ratio and interval data was that *multiplication* is only defined for ratio data. Similarly, multiplication is well-defined for vectors and matrices, but not on tables of data. Depending on the problem domain, it may be inappropriate to use matrices and vectors to represent data where such operations are not necessary.

1.8 Data visualization with plots

Plots allow us to visualize data. Good plots help us to quickly intuit patterns in the data that might otherwise be difficult to understand.

(Note: the term *graph* has different definitions in lower and higher mathematics. We will explain the term "graph" in chapter 8. This text uses the term "plot" as the verb and noun for visualizing data with graphics.)

The *bar plot* helps us to compare the count each category in a discrete (or discretized) variable. The *box plot* helps us to see the center and variation of a numerical variable. The *histogram* also helps us to see the center and variation of a numerical variable, often producing the familiar *bell curve* shape, where the height of the curve indicates the count of observations within the range of each "bin." A histogram is essentially a set of bar plots over discretized numerical values.

A *scatter plot* (sometimes called an XY plot) uses x and y axes to show relationships between two variables. One can also color and shape the points to show third and fourth variables. Three-dimensional XYZ plots are sometimes useful, especially in video and interactive presentations.

As a small exercise to experiment with these four plots, go to https://webr.r-wasm.org/latest/ to use the R language in a web browser. R is a programming language for statistics and data visualization.

R includes several built-in data sets. In the *read-evaluate-print loop* (*REPL*), enter

```
> head(mtcars)
```

to view the column names and first six rows of the Motor Trend Cars (`mtcars`) data set. Now enter the following commands to quickly visualize a few columns in the data set.

```
> barplot(mtcars$cyl)
> boxplot(mtcars$mpg)
> hist(mtcars$mpg)
> plot(mtcars$wt, mtcars$mpg)
```

1.9 Linear and logarithmic scales

Scientists use the term *order of magnitude* to compare values only by the power of 10. One would say $a = 1.6 \times 10^3$ is three orders of magnitude smaller than $b = 8.3 \times 10^6$, which is to say $b/a \approx 1000$.

The *scale* of an axis, such as in bar plot, is the spacing between values. A *linear scale* might show marks at 10, 20, 30, 40, and so on. A *logarithmic scale* might show marks at 10, 100, 1000, 10 000, and so on.

Logarithmic scales can be useful for comparing values that differ by more than one order of magnitude. For example, suppose feature of a data set contains categories a, b, c, and d, and the count of each category is

Category	Count
a	10 736
b	1711
c	398
d	319

Return to https://webr.r-wasm.org/latest/ and plot this data with linear and logarithmic scales:

```
> category_counts <- c(10736, 1711, 398, 319)
> category_counts
[1] 10736  1711   398   319
> barplot(category_counts)
> barplot(category_counts, log="y")
```

1.10 Sets, relations, functions, and algorithms

We now introduce a few terms from *discrete mathematics* that are fundamental to all analysis. A *set* is an unordered collection of *distinct* elements. Sets may be finite or infinite in size. Sets are denoted with curly braces, and the empty set has the special symbol $\emptyset = \{\}$. An example of a set might be

$$W = \{\text{Sunday}, \text{Monday}, \text{Tuesday}, \text{Wednesday}, \text{Thursday}, \text{Friday}, \text{Saturday}\}$$

A *relation* is an association between members of sets. Relations can be used to model any relationship between members any two sets, or even members in the same set. An example might be the relation between integers and elements of W with that many letters, i.e. 6 has a relation on Sunday, Monday, and Friday, 7 has a relation on Tuesday, 8 has a relation on Thursday and Saturday, and 9 has a relation on Wednesday. The term "relation" is seldom used outside of discrete mathematics, but there is a *special case* of a relation that occurs in all mathematical disciplines: *functions*.

A *function* is a relation that uniquely relates members of one set (the *domain*) to another set (the *range*). An example of some functions might be:

$$\begin{aligned}
\text{Translate}\left(\text{Friday}, \text{English}, \text{German}\right) &= \text{Freitag} \\
\text{Length}\left(\text{Wednesday}\right) &= 9 \\
\text{Distance}\left(\text{Thursday}, \text{Tuesday}\right) &= -2 \\
\text{DaysOfLength}\left(6\right) &= \{\text{Sunday}, \text{Monday}, \text{Friday}\} \\
\text{Sunday} &= \text{Next}\left(\text{Saturday}\right) \\
&= \text{Previous}\left(\text{Monday}\right) \\
&= \text{Previous}\left(\text{Previous}\left(\text{Tuesday}\right)\right) \\
&= \text{Previous}\left(\text{Next}\left(\text{Sunday}\right)\right)
\end{aligned}$$

Each of these functions accepts one or more *parameters* as *arguments* and returns the unique corresponding value (if any) from its range. It may appear that the third function, DaysOfLength, has returned three values, but actually this function has returned a single set which contains three values.

Many programming languages use the term "function" as a synonym for *procedure*, *subroutine*, and *method*. Functions are "pure" if they have no side-effects, such as mutating a value outside of the function.

The mathematical definition of the term *algorithm* is the set of instructions necessary to solve a problem. Long division, a procedure for manually dividing numbers, is an example of an algorithm. The term "algorithm" has recently entered the popular lexicon in relation to AI systems. Here, the instructions of the algorithm are part of a model, which is created from data.

1.11 Discussion prompts

1. Who owns knowledge management?
2. What are good and bad uses for spreadsheets?

3. What is reproducibility? Why would this be important for scientific inquiry?
4. Like a barplot, a pie chart shows the relative sizes of categorical values. What are some disadvantages of using pie charts?

1.12 Practical exercises

1. Given a dataset, plot the data and explain why this plot technique is appropriate.
2. Given a noisy and poorly structured dataset, propose a method of restructuring the data.
3. Discretize the values of a dataset and explain the reasoning.
4. Be creative and construct intentionally misleading plots that deliberately distort information presented.

Chapter 2

Data Operations

2.1 Forms and input validation

2.2 Relational algebra

Codd's *relational algebra* is the framework theory describing all modern *database management systems* (DBMS). The relational algebra can be described with five primitives: *selection* (σ), *projection* (π), the *Cartesian product* ($\times$; also known as the *cross product*), set *union* ($\cup$), and set *difference* ($\setminus$).

Selection takes all or a subset of a table's rows. Projection takes all or a subset of a table's columns. For example, suppose a table's schema is defined as `CREATE TABLE WeightliftingMeet (Athlete TEXT, Lift TEXT, Mass REAL, Good BOOLEAN)`. The query `SELECT Athlete FROM WeightliftingMeet WHERE Mass >= 100 AND Good == TRUE` performs both a selection (specified in the `WHERE` clause) and a projection (the columns specified immediately after `SELECT`, in this case `ATHLETE`).

A Cartesian product is the multiplication of sets. If $A = \{i, j\}$ and $B = \{x, y, z\}$, then $A \times B = \{(i, x), (i, y), (i, z), (j, x), (j, y), (j, z)\}$. The Cartesian product produces the set of all possible pairwise combinations of elements in each set. These composite values are called *tuples*. Tuples may contain more than two values. If $C = \{c\}$, then

$$A \times B \times C = \{(i, x, c), (i, y, c), (i, z, c), (j, x, c), (j, y, c), (j, z, c)\}.$$

As an exercise, go to https://sqlime.org to use a DBMS named SQLite. Enter the following commands to reproduce the above Cartesian product.

```
CREATE TABLE A (a text);
CREATE TABLE B (b text);
```

```
CREATE TABLE C (c text);

INSERT INTO A(a) VALUES ('i'), ('j');
INSERT INTO B(b) VALUES ('x'), ('y'), ('z');
INSERT INTO C(c) VALUES ('c');

SELECT * FROM A CROSS JOIN B CROSS JOIN C;
```

This text views tuples as unordered and "flattened" sets, and therefore Cartesian products are both *commutative* ($R \times S = S \times R$) and *associative* ($R \times (S \times T) = (R \times S) \times T$). Some mathematical texts use a stricter definition for the Cartesian product where the result is a set, which does not "flatten" and therefore provides neither commutivity nor associativity. This text uses the looser definition for compatibility with practical DBMSs, including SQLite. Mathematics is partly discovered and partly invented.

Set union, $\cup$, combines two sets. Sets definitionally contain only distinct elements. If $A = \{i, j, k\}$ and $B = \{k, l, m\}$, then

$$A \cup B = \{i, j, k, l, m\} .$$

Set difference, $\setminus$, retains the elements of the left set that are not present in the right set.

$$A \setminus B = \{i, j, k\} \setminus \{k, l, m\} = \{i, j\} .$$

2.3 Join

The *join* ($\bowtie$) is a combination of the Cartesian product and selection. For example, suppose we have a tables named `Swim`, `Bike`, and `Run`. Each table has a column that uniquely identifies an athlete. To get a triathletes (the athletes who participate in swimming, cycling, and running), we use an *equijoin* to find the product where the names are equal. Return to https://sqlime.org to demonstrate experiment with the `JOIN` operator.

```
CREATE TABLE IF NOT EXISTS Swim (sn TEXT UNIQUE);
CREATE TABLE IF NOT EXISTS Bike (bn TEXT UNIQUE);
CREATE TABLE IF NOT EXISTS Run (rn TEXT UNIQUE);

INSERT OR IGNORE INTO Swim (sn) VALUES
    ('John'), ('Jane'), ('Luke'), ('Phil');
INSERT OR IGNORE INTO Bike (bn) VALUES
    ('Mary'), ('Alex'), ('Jane'), ('Levi');
INSERT OR IGNORE INTO Run (rn) VALUES
    ('Mike'), ('John'), ('Jane'), ('Sven');
```

```
SELECT * FROM Swim, Bike, Run WHERE sn = bn AND sn = rn;
```

There are other syntaxes which achieve the same result using the `ON` and `USING` clauses. As an exercise, try to predict how many rows will return from `SELECT * FROM Swim, Bike, Run` without a `WHERE` clause.

2.4 Grouping and aggregation

DBMSs provide robust *grouping* functions for operating on related rows. Return to https://sqlime.org and create a small table of hypothetical marathon times.

```
CREATE TABLE Marathon (rn TEXT UNIQUE, time INTEGER,
  gender TEXT CHECK( gender IN ('M', 'F') ));

INSERT INTO Marathon (rn, time, gender) VALUES
  ('Kyle', 2*60*60 + 14*60 + 22, 'M'),
  ('Hank', 2*60*60 + 10*60 + 45, 'M'),
  ('Lily', 2*60*60 + 24*60 + 47, 'F'),
  ('Emma', 2*60*60 + 22*60 + 37, 'F'),
  ('Elle', 2*60*60 + 25*60 + 16, 'F'),
  ('Fred', 2*60*60 + 6*60 + 17, 'M');

SELECT MIN(time) FROM Marathon GROUP BY (gender);
```

`MIN` is one of the *aggregate functions* in SQLite. The `GROUP BY` clause tells the DBMS to split the rows into groups on the `gender` column.

One might be tempted to find the names of our male and female champions with `SELECT rn, MIN(time) FROM Marathon GROUP BY (gender)`. This may work in some DBMSs but there is a subtle bug. It might be obvious that we want the `rn` associated with the `MIN(time)` value, but suppose we change the query to also include `MAX(time)`:

```
SELECT rn, MIN(time), MAX(time) FROM Marathon GROUP BY (gender);
```

Now it is no longer clear which `rn` the query should return. Should the DBMS return the `rn` associated with the `MIN(time)`, the `MAX(time)`, or some other `rn` from the group?

The solution in this particular case is to nest our `MIN(time)` aggregation as a *subquery*.

```
SELECT * FROM Marathon
  WHERE time IN (
    SELECT MIN(time) FROM Marathon GROUP BY (gender));
```

2.5 Filter, map, and reduce

2.6 Vectorized functions

2.7 Concurrency

2.8 Consistency, availability, and partition-tolerance (CAP) theorem

2.9 Discussion prompts

1. How does the CAP theorem impact intelligence and fires in relation to the command and control (C2) warfighting function (WfF)?
2. Where should unclassified data be stored and processed?
3. What are some methods to prevent conflicts among concurrent writes in a shared database?
4. What could possibly go wrong when altering database schema?

2.10 Practical exercises

1. Create a custom list in SharePoint that provides multiple views showing grouped and aggregated values.
2. Given a noisy dataset, identify problems in each column that could influence inclusion and exclusion criteria.
3. Implement filter and map in terms of reduce using a programming language which provides reduce.
4. Define an "embarrassingly parallel" problem and provide both examples and counterexamples.

Chapter 3

Measures of Central Tendency

3.1 Mode

3.2 Median

3.3 Arithmetic Mean

3.4 The four moments: mean, variance (and standard deviation), skewness, and kurtosis

3.5 Exponential moving averages (EMA)

3.6 Covariance

3.7 Outliers

3.8 Unbalanced data sets

3.9 Discussion prompts

Is four a lot?

First battalion has an average ACFT score of 482 while second battalion has an average ACFT score of 491. Which is better?

What do we do when statistics show us something that contradicts our values? For example, suppose we discover that Soldiers of a specific demographic have much lower promotion rates than their peers.

Is it more important for an organization to think about variance or the 99th percentile?

Given a sample set [Equation], what is the estimate of the mean ([Equation]), and what is the sample variance?

3.10 Practical exercises

Calculate the influence that outliers have on different-sized datasets that contain outliers.

Calculate the exponential moving average in a small dataset.

Given a dataset and experimental result, identify problems caused by analyzing categorical data represented in a numeric form.

Given multiple datasets with identical mean and standard deviation, use kurtosis to identify the dataset with more outliers.

Design or implement an algorithm to incrementally calculate standard deviation, where the estimate of the sample standard deviation is updated with each additional value.

Chapter 4

Linear Models

4.1 Sum of squared errors

4.2 Linear regression

4.3 Polynomial models

4.4 Pearson correlation coefficient

4.5 Prediction

4.6 Overfitting

4.7 Correlation and causation

4.8 Discussion prompts

What are confounding factors and how can we identify them?

When is it useful to generate a linear model with no previous understanding of the data?

Some data mining techniques fail if the data set contains colinear columns. How might one identify these columns?

4.9 Practical exercises

Design a novel linear model, using any programming language, that considers time when fitting [Equation] values and creates a biased model (like an EMA).

Given linear models and their associated [Equation] values among many different columns in a data set, identify the strongest and weakest models, then use plots to support these conclusions.

Using the Goal Seek feature of Microsoft Excel, estimate the mean of a column by minimizing the sum of squared errors. Compare this value to the arithmetic mean.

Use more than one tool (such as both Excel and R) to find the line of best fit on a single dataset. Compare and contrast the linear regression generated by each program.

Chapter 5

Hypothesis Testing

5.1 Combinatorics

5.2 The Binomial Distribution

5.3 The Normal Distribution

5.4 The Central Limit Theorem

5.5 Null hypotheses ([Equation]), [Equation]-values, and [Equation]-values

5.6 Probability density function (PDF) and cumulative distribution function (CDF)

5.7 Student's [Equation]-test

5.8 Pearson's [Equation]-test

5.9 Analysis of Variance (ANOVA)

5.10 Discussion prompts

How does the "curse of combinatorics" create effectively infinite event spaces?

Suppose a daycare has 1000 toys in the toybox. Each time a child takes a toy from the toybox, a worker records the toy and the (seemingly independent) result of a fair coin toss. After each toy has been pulled ten times, the worker discovers that the coin always landed on heads for the toy shark. How surprising is this outcome?

A study shows [Equation]. What does this result mean?

Describe a workplace situation where the Central Limit Theorem applies.

5.11 Practical exercises

One group generates a random data set that should not fit a normal distribution. Another group takes samples and applies the Central Limit Theorem to estimate the population mean.

Given a multivariant dataset, use ANOVA to identify the strongest linear relationship between a dependent variable and many independent variables.

An obscure Filipino superstition claims that the gender of the firstborn child predicts the first parent who will die (e.g., if the eldest child is a son, then the father will pass away before the mother). Create a small data set by surveying the class, then use the [Equation]-test to analyze the strength of the superstition.

Use a CDF/PDF implementation to convert a uniform random number generator (RNG) into a normal distribution.

Chapter 6

Supervised Learning

6.1 Learning from data

6.2 Test-train split

6.3 Confusion tables and model accuracy

6.4 Decision trees

6.5 Artificial neural networks (ANN)

6.6 Activation functions

6.7 Backpropagation

6.8 Discussion prompts

Why might a voice recognition model more reliably understand one accent than another in the same language? How might the model be improved?

What happens when a model is trained on information that the model itself produced?

Unlike linear models, neural networks are often initialized with random values. As a result, training two models on the same data might not lead to identical results. What are the ethical implications of having different results in different models?

6.9 Practical exercises

In any programming language, train an ANN to recognize handwritten characters. Use a test-train split to evaluate the accuracy of the model.

Create a classification model for an extremely unbalanced data set, then assess the accuracy of the model.

Create a classification tree by hand to identify Russian military vehicles.

As a group, create an “ensemble” of different models that vote on the outcome of a classification task.

Chapter 7

Unsupervised Learning

7.1 Data mining

7.2 Principal component analysis (PCA)

7.3 Hierarchical clustering

7.4 [Equation]-Nearest Neighbors (kNN)

7.5 [Equation]-Means Clustering

7.6 Constraint solvers

7.7 Embeddings

7.8 Word2Vec

7.9 Discussion prompts

Suppose a dataset contains a sequential numeric identifier for each row. An unsupervised learning method unexpectedly uses this feature to predict an outcome with good accuracy. What does this mean?

Supposes an embeddings model produces “queen” from “king – man + woman”, but unexpectedly produces “king” from “queen – English + Turkish – Turkish + English”. Why might this occur?

7.10 Practical exercises

Compile raw ACFT data for the class, perform PCA, and attempt to interpret the first and second components.

Using the same ACFT data, use cluster analysis to attempt to predict Army component (RA, AR, NG). Use the diagonal of the resulting confusion table to assess model accuracy.

Model a scheduling problem using a constraint solver.

Chapter 8

Graph Theory

8.1 Vertices and edges; [Equation]

8.2 Directed and undirected graphs

8.3 Directed acyclic graphs (DAG) and topological sorting

8.4 Weighted graphs

8.5 Breadth-first search (BFS) and depth-first search (DFS)

8.6 Dijkstra's algorithm

8.7 Computational complexity and Big-[Equation]

8.8 Graph databases

8.9 Power Law Distribution

8.10 Discussion prompts

A graph can be represented with an adjacency list or a matrix. What are the advantages and disadvantages of each approach?

What algorithm can be used to solve the “seven ways to Kevin Bacon” problem?

Is a Gantt chart a graph? How can one find the critical path of a project if represented as a graph?

Which is bigger, [Equation] or [Equation]?

8.11 Practical exercises

Compare two different heuristic functions in a provided A* informed search implementation on the 8-piece puzzle problem.

Convert currency exchange rates from multiplication to addition using a logarithm, then prove that infinite arbitration is impossible given a set of exchange rates and Bellman-Ford implementation.

Define a topological sorting and relate it to a workplace problem.

Define the Traveling Salesman Problem (TSP) and explain the computational difficulty of this problem.

Determine the minimum paving needed to fully connect a tent complex using a list of coordinates and a Prim or Kruskal implementation.

Simulate an infection model in a dense social graph where edge weights represent probability of infection.