WEIMAO KE

MACHINE LEARNING: THE QUEST FOR INFORMATION AND MEANING

PUBLISHED BY DRAFT - PLEASE DO NOT DISTRIBUTE

First printing, April 2020

Contents

List of Figures

List of Tables

To Carrie, Lucy, and Brighton

Preface

There have been many good reference resources on the subject of data mining and machine learning. For researchers in the area, it is not a daunting task to search and find relevant articles, online tutorials, and even reference books on specific algorithms and techniques.

As an educator, however, I have struggled to find suitable textbooks for students taking my courses in information retrieval, data mining, and machine learning. Many textbooks are too technical or too inclined to fast-forward to mathematic equations to serve as an introductory text. Students, undergraduate and graduate students alike, are often so overwhelmed by the details of technicality and mathematics that they lose sight of the grand picture and ideas.

I have longed to write a simple book to help my students, a book that strikes a balance between big pictures and necessary technical details; a book with sufficient technical granularity yet easy to access.

This will not serve as a reference book as there have already been plenty of resources available. Rather, it will introduce related topics with rigor and necessary details, opening the door for students and intriguing them to pursue further. The presentation will, in a sense, walk them through the landscape of the field and guide them to see specifics in context.

It is my hope that the book will promote imagination and critical thinking while showing the state of the art. We as educators are not here simply to pass on knowledge but to prepare future researchers and practitioners capable of innovation. I strive to approach academic discussions in the context of real-world applications; on the other hand, I value the process of thought experiments with questions and deliberation. This is something my students enjoyed and appreciated, and what I often employ in the book for important ideas or ways of thinking.

The organization of the book will be driven by major challenges in the quest for relevant information and ways of tackling data with computing methods. Starting with very basic questions and (competing) solutions, it will gradually escalate related problems, with the removal of certain assumptions and consideration of additional

factors, to challenge the reader to look for alternatives and new approaches to innovation.

In the end, each chapter will delve into necessary technical details in the state-of-the-art, connect them back to the original issues that have been raised, and debate about what has been addressed or otherwise. The reader will be left with critical questions and directions for further study and research.

Reading this book is an exercise that will, I hope, help the audience gain insight into computing big data solutions for today and the future. On a personal note, I expect it to assist my students in understanding, questioning, thinking, and innovation in the exciting field of data sciences.

Weimao Ke
Philadelphia, 2019

Introduction

> With great power comes great responsibility.
>
> *Uncle Ben*, Spider-Man

We encounter data and information every day. We live in a world where information appears to be constantly generated and consumed. We have research disciplines where the subject of study is on information. Yet when it comes to the basic definition of what information is, there is hardly any agreement.

Information is ubiquitous and conveyed in all sorts of media. It can be communicated on news papers, radio, or television. It exists in books that can be read and touched, in records that can be played and listened to. Yet information is not the physical medium being used nor the symbols and signals being transmitted. It is a seemingly intangible concept of a tangible reality. With a conscious mind, information can be internalized into knowledge, which in turn gives rise to new ideas and, as it may go on, new information.

Information is power, in so many tangible ways.

The biological coding of DNS dictates the physical growth, development and functioning of living organisms. The very existence of *mass* dictates the curvature of space-time and the physics of moving objects. Is it not a manifestation of information itself?

Information appears to be so universal that a school of prominent physicists have proposed information to be the fundamental element of the universe, with matter and energy as its incidents. "Everything is information," as the late John A. Wheeler put it.

Will it turn out to be true that information is indeed the essence behind "everything," a reality more fundamental than matter and energy, or quantum and light? In other words, is the universe simply a manifestation of its inherent information? If this is the case, we can perhaps regard the natural world as a giant hologram projected from its information core.

In this view, everything we encounter originates from the fundamental reality of information. The pitches of sound we hear, the rays

of light we observe, and the forces that push us together or pull us apart... all are signals of information from deep within. When we record the signals and events, they become our data. They are traces of the underlying information we are yet to discover.

And from here we embark on our quest for *information*, following the footprints of *data* with the vehicle of *computing*.

From Data to Meaning

Suppose we receive the following data:

011 0100 0010 0 100 0101 1001 1 011 1001 0101 0
011 0010 0100 1 100 0001 1000 1 100 0101 0011 0 ... .

Table 1: Raw data

Clearly, the data arrive in the binary form but one has to find out what the numbers actually stand for. If metadata[1] are provided, the values can be properly interpreted and used. Otherwise, one may examine the patterns in the data and identify potential structures.

[1] Metadata is simply data about data.

For example, one may discover that number of digits between each pause (space) repeats in a 3-4-4-1 pattern. If we break each combination of 3-4-4-1 into a line and convert the bits into decimals, the data become:

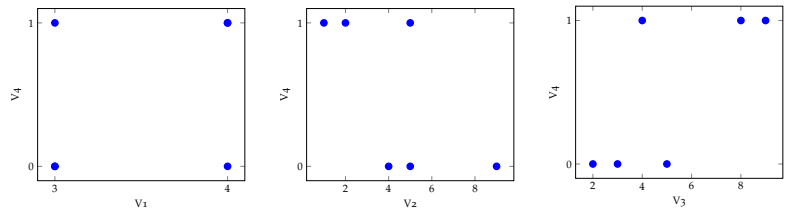
3	4	2	0
4	5	9	1
3	9	5	0
3	2	4	1
4	1	8	1
4	5	3	0
...	....	....	.

Table 2: Raw data, with patterns identified and numbers converted.

Now in Table 2, we are yet to find out what the data mean. There appear to be 4 columns for each row we received. The 4^{th} column is binary, with 0 vs. 1 indicating a boolean answer for one way or the other. We suspect that values of the first 3 columns might have something to do with the 4^{th}.

When examining their relations, it is often helpful to use visualizations. We may start with a scatter plot for the 4^{th} column vs. each of the first 3 columns, as shown in Figure 1.

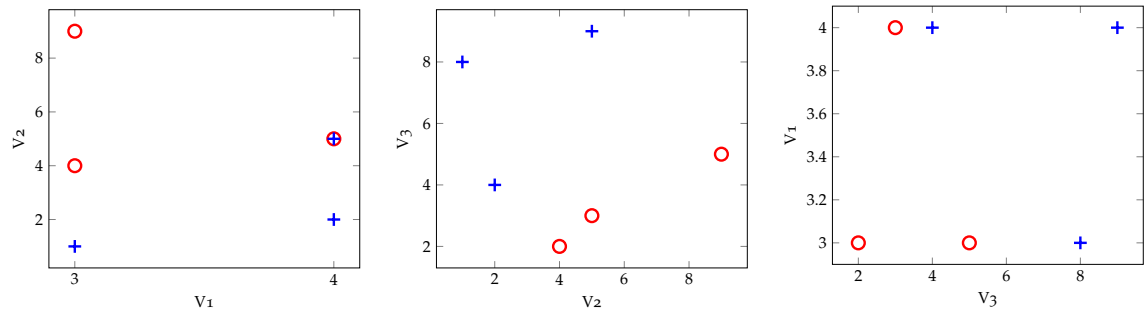
Now that the scatter plots hardly reveal any obvious relation, we may start to look at the interplay of multiple variables. We may ask, for example, "is the boolean value of the 4^{th} column in any way associated with the relation between the first two columns?" To answer this question, we may create a scatter plot of the 1^{st} and 2^{nd} columns



(a) V_4 vs. V_1 (b) V_4 vs. V_2 (a) V_4 vs. V_3

Figure 1: Scatter plots: V_4 vs. V_1, V_2, V_3

while color-coding the 4^{th}. When we do the same for 2^{nd} vs. 3^{rd} and 1^{st} vs. 3^{rd} as well, we get Figure 2.



(a) V_2 vs. V_1 (b) V_3 vs. V_2 (a) V_1 vs. V_3

Figure 2: Scatter plots with color-coded V_4. Mark $\circ$ for 0 and $+$ for 1 values.

It appears the data are nicely separated in terms of 4^{th} column values in Figure 2 (b), showing the scatter plot of the 3^{rd} vs. 2^{nd} column. When the value of the 3^{rd} column is greater than the value of the 2^{nd}, it is 1 in the 4^{th} column; otherwise, it gives 0. This can be written as:

$$V_4 = \begin{cases} 1 & if\ V_3 > V_2 \\ 0 & otherwise \end{cases} \tag{1}$$

This appears to be the pattern so far and can be further verified when we receive more data. Ultimately, if this is consistent with the underlying relation in the data – as far as we can tell from sufficient data we can receive – we can claim to have discovered a pattern or new "knowledge", which is essentially what we have learned in the process of analyzing the data.

In the machine learning context, we refer to the result of learning, i.e. the new "knowledge", as the *concept*. Equation 1 is a rule-based approach to concept representation (descriptor). Sometimes, a (linear) equation may also be used to represent the result of learning. In this example, equation $V_3 - V_2 = 0$ is the straight line that separates the 0 and 1 values of column 4, as shown in Figure 3.

Whether the concept representation is based on the rules in Equation 1 or the linear equation in Figure 3, it can be identified by computing and comparing values in the data. Normally such concept

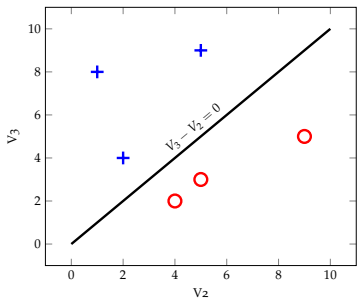


Figure 3: Separating 0 vs. 1 in V_4 with a linear equation

representation is discovered through *training data* and to be applied on testing data, where decisions or predictions will be made.

Concept Generalizability

The example shows that we can potentially identify a concept and gain new knowledge by crunching numbers in the data. This could be done without understanding what the data mean in context. It is important, however, to examine the representativeness of training data and the generalizability of concept representation thus learned in the application domain.

To give the above example some context, the data are in fact about a collection of shapes, each of which has a number of sides, a width, a height, and a code indicating whether the shape is *standing* or *lying*.

# Sides	Width	Height	Standing?
3	4	2	0
4	5	9	1
3	9	5	0
3	2	4	1
4	1	8	1
4	5	3	0
...	....	....	.

Table 3: Shapes data with column information

Data in Table 3 represent the shapes shown in Figure 4. Intuitively, the *concept* of standing can be understood as describing a human-like object – when a person stands, her height is greater than the width; when one lies down, the horizontal measurement (width) becomes relatively greater.

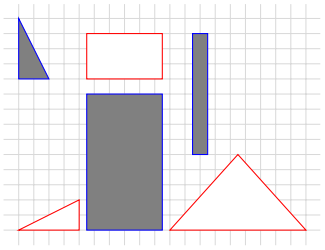


Figure 4: Shapes of different # sides, widths, and heights, based on Table 3. Gray shapes are classified as *standing*.

It is in this context of human-like shapes that the concept of standing is injected into the data and can be discovered by mining. It is also in this context – again the assumption of human-like shapes – that the identified relation of width vs. height can be applied to related data for predictions, decision making, etc.

With related assumptions attached, the concept does not represent universal truth. The same notion of *standing* might be very different with a more complex shape or structure. In Figure 5, for example, the shape of a building may be flat in terms of its height-width ratio but we see it *standing as an architecture*.

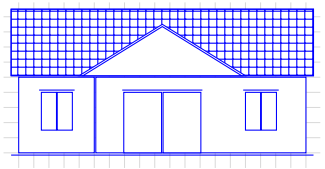


Figure 5: A *standing* architecture

It takes an in-depth analysis with proper evaluation and interpretation to assess the learned concept, its generalizability, and applicability. This cannot be accomplished by crunching the numbers alone. It requires human intelligence along with machine learning.

Final Observation

For the concept of *standing*, it turns out that the number of sides of a shape has no impact on the classification. Variables (features) as such can be safely removed for the analysis. As far as the specific concept goes, these variables are part of the noise in the data and can be identified through feature selection.

The example also shows that it is sometimes (often) insufficient to model a *concept* on *individual* variables. In the shapes data example, the concept of *standing* is a relation between the *width* and *height*. For now, it is a simple, linear function. Nonetheless, a more complex concept with messier real-world data may require a non-linear model.[2] In the following chapters, we will be looking at approaches by which relations of various complexity can be discovered.

Ultimately, machine learning on massive data may help us identify patterns and relations that cannot be eye-spotted, which will then lead to new insight and meaning. And the gained knowledge, if understood properly, will hopefully result in better decision making and improved productivity.

[2] For example, a non-linear model can be a polynomial function. Sometimes, it can also be a linear function based on a non-linear transformation of the original data.

From Datum to Data

As we employ computers to crunch the data to find meaning and insight, we need to understand the basic form of data, their basic types and implications.

Datum

Numerical

To compute is to reckon with numbers. Ultimately, all data should be provided as or transformed into numbers: real numbers, integers or decimal values.

Consider the mileage of a car A, such as 96 thousand miles or, if one demands to be more precise, $96,123.5$ miles. Compared to another car B at 64 thousand miles, one can reasonably make the following statements:

1. Car A has a greater mileage than B has;
2. Car A has 32 thousand more miles than B has;
3. The mileage of car B is $\frac{2}{3}$ of that of A.

The 1^{st} statement can be made because all numbers can be compared, e.g. $96 > 64$. However, the 2^{nd} and 3^{rd} statements make sense because of special characteristics of the *mileage* variable here.

In fact, we can make the 3^{rd} statement because mileage is a *ratio-scaled* variable, for which the zero point is well defined. In this example, a 0 mileage corresponds to the fact that the car has not been driven at all since its production, and is therefore measured absolutely zero. On such a ratio-scaled variable, many mathematical operations such as subtraction, addition, and division can be performed.

Conversely, a temperature variable in Fahrenheit or Celsius is not ratio-scaled, because a 0 value in either $^{\circ}F$ or $^{\circ}C$ is no indication of "zero temperature" or no heat at all. In this case, we cannot perform addition on its values to get a "total temperature" or division to obtain a ratio. Nonetheless, related temperature values are

interval-scaled because they are measured in equal units and a distance (difference) between two values is meaningful.

It is reasonable to say that $100°F$ is 40 degrees higher than $60°F$. As an interval-scaled variable, one can perform subtraction to compute the distance (interval). The sum (addition), however, does not have a clear meaning (except as an intermediary step to calculate the average).

There are also variables for which the unit of measurement is not necessarily equal or consistent. On a likert scale, for example, rating numbers from 1 to 5 may be associated with levels of customer satisfaction, e.g. extremely unsatisfied to extremely satisfied. Arguably, the difference between 1 (extremely unsatisfied) and 2 (unsatisfied) may not be the same as that between 3 (moderately satisfied) and 4 (satisfied). In this case, one can compare the satisfaction values and rank them. But it is not as meaningful – at least not consistent – to compute and compare their intervals (differences).

Such a variable is considered *ordinal* as it can be arranged in a meaningful order but lacks an equal unit of measurement to be interval-scaled. While we can compare its values, one should refrain from performing subtraction on them. Neither is addition meaningful with an ordinal variable.

Categorical

Ordinal variables are in fact categorical. The values are discrete numbers to be compared and ranked; but other than that, no mathematical operations can be performed on them. In fact, they do not have to be numbers. For example, an *Education* variable with values such as *high school*, *college*, and *graduate* can be compared and ranked. These are named values with an order.

A categorical variable can have named values (labels or categories) without an order. We refer to such variables as *nominal*, of which values are from a set of *names* or *labels*. The very basic form of a nominal variable is a binary one, where there are two possible named values: true or false, yes or no, 0 or 1, etc.

For many variables, these labels are discrete, mutually exclusive choices without an explicit relation. For example, a *State* variable can have values such as *PA* and *NY*, and there is no relation between *PA* and *NY*. One cannot establish such relation as $PA > NY$ or $NY > PA$ unless another variable such as state population is considered. This type of variable is purely categorical, not ordinal. With a categorical variable, we can only use the equality operator to determine whether two values are equal or not, e.g. is *PA* the same or different from *NY*.

Data

Sets

We have discussed categorical variables with values from a set of labels. A set is a collection of member objects or elements. The elements can be any unique entities, symbols, names, and/or numbers. The number of elements in a set is referred to as its *cardinality*.

Consider the skill sets of two data science teams A and B:

$$\begin{aligned} A &= \{AI, Math, Python, SQL\} \\ B &= \{InfoVis, Math, R, SQL, Stats\} \end{aligned}$$

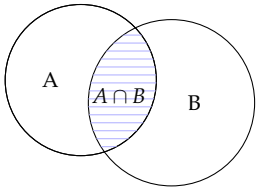


Figure 6: Intersection $A \cap B$

It is easy to count the number of skills in each team and identify their cardinalities: $|A| = 4$ and $|B| = 5$.

The common skill set of the two teams is their *intersection*:

$$\begin{aligned} A \cap B &= \{Math, SQL\} \\ |A \cap B| &= 2 \end{aligned}$$

which is illustrated by the shaded area in Figure 6. The interaction operation is communicative, i.e. $A \cap B = B \cap A$, because the order does not matter.

Now, if we are to put the two team together on one project, the overall skill set of the merged project team will be the *union* of the two:

$$A \cup B = \{AI, InfoVis, Math, Python, R, SQL, Stats\}$$

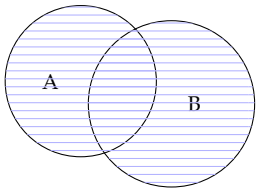


Figure 7: Union $A \cup B$

In this case, the cardinality of the union is:

$$\begin{aligned} |A \cup B| &= |A| + |B| - |A \cap B| \\ &= 4 + 5 - 2 \\ &= 7 \end{aligned}$$

Because a set contains only *unique* elements, those in the intersection $A \cap B$ will have duplicates when two sets are combined and should be removed with $-|A \cap B|$.

The union operation is also communicative, i.e. $A \cup B = B \cup A$, because it does not matter whether we add B to A or A to B. In addition, both the union and intersection operations have the associative property, that is:

$$
\begin{aligned}
(A \cup B) \cup C &= A \cup (B \cup C) \\
(A \cap B) \cap C &= A \cap (B \cap C)
\end{aligned}
$$

for any sets A, B, and C. As illustrated in Figure 8, we will obtain the same union set (the shaded circles) regardless of which two are combined first. The same is true with intersection, as illustrated in Figure 9.

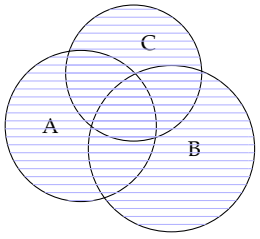


Figure 8: Union of 3 sets

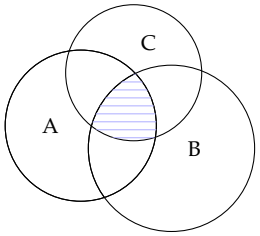


Figure 9: Intersection of 3 sets

The union (∪) operation is similar to *addition* (+), whereas the intersection ∩ operation can be likened to *multiplication* (×). For demonstration, let's look at a couple of special cases. Let 0 denote an empty set, with no elements at all; 1 be the the universal set, containing all elements in the universe.

For a regular set A, we can show the similarity between union (∪) and addition (+):

$$
\begin{aligned}
A \cup 0 &= A + 0 \\
&= A \\
1 \cup 0 &= 1 + 0 \\
&= 1 \\
1 \cup A &= 1 + A \\
&\approx 1
\end{aligned}
$$

The union of a set A and an empty set 0 is the A itself: $A \cup 0 = A$. The same is true about adding an empty set 0 to the universal set 1. $1 \cup A$ is not exactly $1 + A$, but they are roughly the same as long as the universal set 1 is far greater than A.

Likewise, intersection (∩) and multiplication (×) operations are similar in that:

$$
\begin{aligned}
A \cap 0 &= A \times 0 \\
&= 0 \\
1 \cap 0 &= 1 \times 0 \\
&= 0 \\
A \cap 1 &= A \times 1 \\
&= A
\end{aligned}
$$

The intersection between any set (A or 1) and an empty set 0 is always an empty set: $A \cap 0 = 0$ and $1 \cap 0 = 0$. The intersection between a set A and the universal set 1 is A, because the universal includes A as a subset. In fact, for any set A that is a subset of B (see Figure 10), their intersection is A and their union is B:

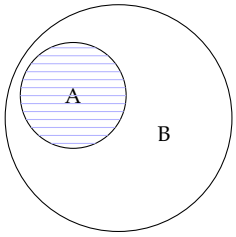


Figure 10: Subset $A \in B$

$$A \cap B \overset{\forall A \in B}{=} A \tag{2}$$

$$A \cup B \overset{\forall A \in B}{=} B \tag{3}$$

Besides union and intersection operators that can be likened to addition and multiplication, there is also the subtraction operator on sets. For two sets A and B, $A - B$ computes the difference between A and B, or the elements of A that do not appear in B. This is essentially the removal of their intersection from set A, as shown in Figure 11.

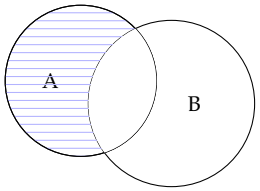


Figure 11: Subtraction $A - B$

The cardinality of the subtracted set can be computed by:

$$|A - B| = |A| - |A \cap B| \tag{4}$$

If A and B have no intersection, $A - B$ is the same as A. If A is a subset of B, then the result is an empty set. Subtraction is not communicative, as $A - B \neq B - A$.

Vectors

In geometrical terms, a *vector* is an entity having both magnitude (length) and direction, in a vector space with multiple dimensions.

Figure 12 shows data v_1 and v_2 in a space with X and Y dimensions. In the vector space, each data point is a vector from the origin $(0, 0)$, as shown as directed lines in Figure 12, and can be represented by its dimensional values:

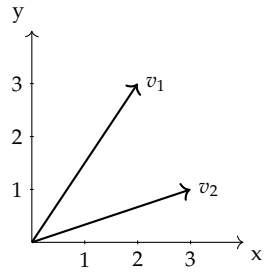


Figure 12: Two-dimensional vector space

$$v_1 = (2, 3)$$
$$v_2 = (3, 1)$$

So in this representation, a vector is simply a list of its coordinate values, each of which represents a feature dimension (variable). The greater the number of dimensions, the longer the list of the values.

A *set* can be represented by a vector. For example, suppose {AI, InfoVis, Math, Python, R, SQL, Stats} is a comprehensive set of skills for data science teams. Given the A and B teams and their skills we discussed earlier, we can have the following tabulated binary representation:

Team	AI	InfoVis	Math	Python	R	SQL	Stats
A	1	0	1	1	0	1	0
B	0	1	1	0	1	1	1

Table 4: Tabulated team skills

where 0 represents absence of a skill and 1 denotes presence of the skill. By populating the 0 and 1 values in a fixed order of the skills, we obtain the vector representations for teams A and B identical to Table 4:

$$\begin{aligned} A^T &= \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \\ B^T &= \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \end{aligned}$$

where the vector space has 7 dimensions, which correspond to seven skills in the comprehensive set, to represent instances of data science teams. These vectors are *row vectors* as they represent data rows.

Note we use superscripted A^T and B^T, where T is the *transpose* operator, because of the convention of using column vectors by default. By transposing the column vectors A and B – with a 90° counter-clockwise rotation – we obtain the row vectors A^T and B^T:

$$A^T = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}^T = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$B^T = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Here, each column vector is a column with 7 rows of *real* numbers. Hence, $A \in \mathbb{R}^m$ and $B \in \mathbb{R}^m$, where $m = 7$. By transposing the column vectors, we obtain row vectors, each of which has m columns of *real* numbers.

Matrix

When we have multiple row vectors of data instances, they form a data matrix. For example, a data matrix of the two teams A^T and B^T can be represented as D:

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

Here D is an $n \times m$ matrix, with $n = 2$ rows and $m = 7$ columns. That is, $D \in \mathbb{R}^{n \times m}$, where $n = 2$ and $m = 7$. In D, we not only have row vectors for the teams but also column vectors for the skills:

$$\begin{matrix} AI \\ = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{matrix} \quad \begin{matrix} InfoVis \\ = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{matrix} \quad \begin{matrix} Math \\ = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \end{matrix} \quad \begin{matrix} Python \\ = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{matrix} \quad \begin{matrix} R \\ = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{matrix} \quad \begin{matrix} SQL \\ = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \end{matrix} \quad \begin{matrix} Stats \\ = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{matrix}$$

Basic Matrix Operation

Consider another data set about the annual revenue and profit of three companies. Let X be last year's data (in million dollars):

$$X = \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} \begin{pmatrix} 3 & 0.5 \\ 20 & 1.5 \\ 12 & 1 \end{pmatrix}$$

where each row is a 2-dimensional vector and can be visualized in Figure 13.

Let Y be this year's data as follow, which, likewise, can be visualized as 3 row vectors in Figure 14:

$$Y = \begin{matrix} y_1 \\ y_2 \\ y_3 \end{matrix} \begin{pmatrix} 6 & 1 \\ 15 & 1 \\ 16 & 2 \end{pmatrix}$$

Apparently, X and Y have the same dimensionality – $X \in \mathbb{R}^{3 \times 2}$ and $Y \in \mathbb{R}^{3 \times 2}$ – as they are about the same three companies (rows) and two variables (columns). We can perform simple computation such as addition and subtraction on the matrices.

For example, it is easy to compute the increase or decrease in revenue and profit with subtraction:

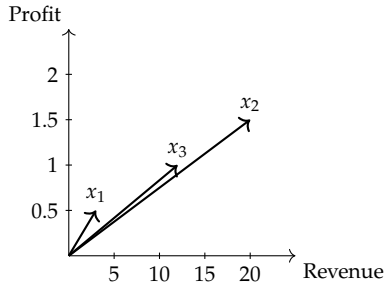


Figure 13: Row vectors of matrix X

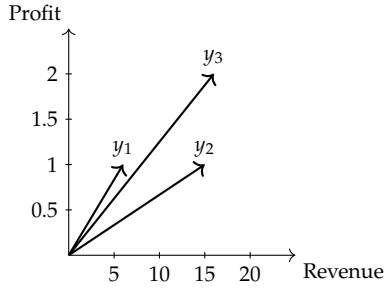


Figure 14: Row vectors of matrix Y

$$
\begin{aligned}
G &= Y - X \\
&= \begin{pmatrix} 6 & 1 \\ 15 & 1 \\ 16 & 2 \end{pmatrix} - \begin{pmatrix} 3 & 0.5 \\ 20 & 1.5 \\ 12 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 6-3 & 1-0.5 \\ 15-20 & 1-1.5 \\ 16-12 & 2-1 \end{pmatrix} \\
&= \begin{matrix} g_1 \\ g_2 \\ g_3 \end{matrix} \begin{pmatrix} 3 & 0.5 \\ -5 & -0.5 \\ 4 & 1 \end{pmatrix}
\end{aligned}
$$

Matrix G, the result of the subtraction, contains data about growth (increases and decreases). The rows of matrix G are vectors connecting respective X vectors to Y vectors. Let's single out the 3^{rd} company from the data, i.e. x_3, y_3, and g_3. As shown Figure 15, subtraction $g_3 = y_3 - x_3$ is the vector that starts at x_3 and ends at y_3.

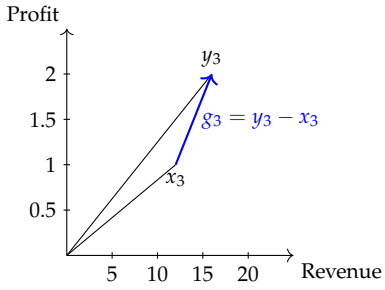


Figure 15: Subtraction geometry

Suppose the rate of growth in G is projected to continue (the same) next year. This will double the growth, which can be computed by:

$$
\begin{aligned}
G \times 2 &= \begin{pmatrix} 3 & 0.5 \\ -5 & -0.5 \\ 4 & 1 \end{pmatrix} \times 2 \\
&= \begin{pmatrix} 3 \times 2 & 0.5 \times 2 \\ -5 \times 2 & -0.5 \times 2 \\ 4 \times 2 & 1 \times 2 \end{pmatrix} \\
&= \begin{pmatrix} 6 & 1 \\ -10 & -1 \\ 8 & 2 \end{pmatrix}
\end{aligned}
$$

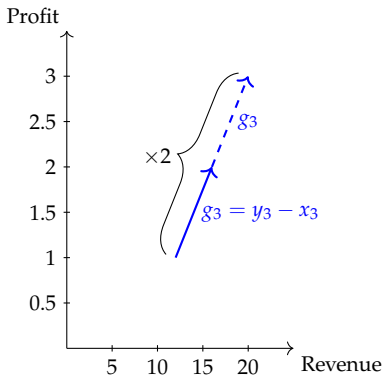


Figure 16: Geometry of $g_3 \times 2$, as part of matrix multiplication $G \times 2$

Here every value in matrix G is doubled; so is every vector – row or column – in the matrix. For example, Figure 16 shows when g_3, the growth vector of the 3^{rd} company (row), multiplies 2, the vector doubles its length (magnitude) in the same direction. Multiplying a matrix with a scalar (single number) results in a matrix of the same dimensionality, in which every value is multiplied by the same scalar. Note the multiplication can be written as $2 \cdot G$, $2G$ or, more generally, aG given a scalar a and a matrix G.

Now with the projected growth (or decrease) data in the $2G$ matrix, we can add them to the last year's data matrix X to compute the final projection for next year Z:

$$
\begin{aligned}
Z &= Y + 2G \\
&= \begin{pmatrix} 3 & 0.5 \\ 20 & 1.5 \\ 12 & 1 \end{pmatrix} + \begin{pmatrix} 6 & 1 \\ -10 & -1 \\ 8 & 2 \end{pmatrix} \\
&= \begin{pmatrix} 3+6 & 0.5+1 \\ 20-10 & 1.5-1 \\ 12+8 & 1+2 \end{pmatrix} \\
&= \begin{matrix} z_1 \\ z_2 \\ z_3 \end{matrix} \begin{pmatrix} 9 & 1.5 \\ 10 & 0.5 \\ 20 & 3 \end{pmatrix}
\end{aligned}
$$

Note that the data matrix Z projected from last year's numbers in X should be the same as that projected based on this year's data in Y, i.e. $Z = X + 2G = Y + G$ given that $G = Y - X$.

Geometrically, for corresponding vectors in the matrices, the addition operation is to connect them head to tail. For example, as shown by the solid lines in Figure 17, for the 3^{rd} row vector, $x_3 + g_3 \times 2$ is to connect vectors x_3 and $2g_3$ and the result is z_3, which draws from the tail of x_3 to the head of $2g_3$. Vector and matrix addition is communicative, the same result can be obtained by $2g_3 + x_3$, as shown by dashed lines in Figure 17.

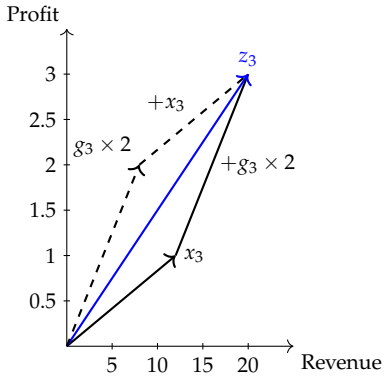


Figure 17: Geometry of addition $z_3 = x_3 + g_3 \times 2$

Sum of Squares

Suppose x is a m-dimensional column vector, which is essentially a matrix of m rows and 1 column: $x \in \mathbb{R}^{m \times 1}$.

$$
x = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_m \end{pmatrix}
$$

Now we can transpose the column vector x into a row vector x^T, which is a matrix of 1 row and m columns, i.e. $x^T \in \mathbb{R}^{1 \times m}$:

$$
x^T = \begin{pmatrix} x_1 & x_2 & ... & x_m \end{pmatrix}
$$

We can multiply x^T with x and get their dot product:

$$
\begin{aligned}
x^T x &= \begin{pmatrix} x_1 & x_2 & ... & x_m \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_m \end{pmatrix} \\
&= x_1^2 + x_2^2 + .. + x_m^2 \\
&= \sum_{i=1}^{m} x_i^2
\end{aligned} \tag{5}
$$

The result is the total sum of squares, a scalar. Sum of squares is often performed on error terms in statistical analysis. For example, given the following errors of three instances:

$$
e = \begin{pmatrix} 0.5 \\ 0.2 \\ 0.7 \end{pmatrix}
$$

We can compute the Error Sum of Squares (ESS) by:

$$
\begin{aligned}
e^T e &= \begin{pmatrix} 0.5 & 0.2 & 0.7 \end{pmatrix} \times \begin{pmatrix} 0.5 \\ 0.2 \\ 0.7 \end{pmatrix} \\
&= 0.5^2 + 0.2^2 + 0.7^2 \\
&= 0.78
\end{aligned}
$$

Vector Magnitude (Length)

Any vector has two components: a direction and a magnitude (length). Taking the square root of the sum of squares, we can compute a vector's magnitude:

$$
\begin{aligned}
\|x\| &= \sqrt{x^T x} \\
&= \sqrt{\sum_{i=1}^{m} x_i^2}
\end{aligned} \tag{6}
$$

which is the length from the origin to the data point x. For example, given a two-dimensional vector v:

$$
v = \begin{pmatrix} 3 \\ 4 \end{pmatrix}
$$

Its magnitude can be computed by:

$$
\begin{aligned}
\|v\| &= \sqrt{3^2 + 4^2} \\
&= 5
\end{aligned} \tag{7}
$$

which is the length shown in Figure 18.

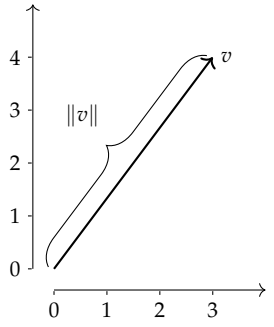


Figure 18: Vector magnitude

Vector Distance

Let y be another column vector with the same dimensionality $y \in \mathbb{R}^{m\times 1}$:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_m \end{pmatrix}$$

The difference between x and y can be computed by:

$$\begin{aligned} x - y &= \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_m \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_m \end{pmatrix} \\ &= \begin{pmatrix} x_1 - y_1 \\ x_2 - y_2 \\ ... \\ x_m - y_m \end{pmatrix} \end{aligned}$$

which, geometrically, is the vector connecting the head of x to that of y. The magnitude of this difference vector can be computed using Equation 6:

$$\begin{aligned} \|x - y\| &= \sqrt{(x-y)^T(x-y)} \\ &= \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2} \end{aligned} \tag{8}$$

Consider two two-dimensional vectors:

$$u = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad v = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

There difference can be computed by:

$$\begin{aligned} v - u &= \begin{pmatrix} 3 \\ 4 \end{pmatrix} - \begin{pmatrix} 2 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 3 \end{pmatrix} \end{aligned}$$

Their Euclidean distance can be computed by the magnitude of their difference:

$$\begin{aligned} \|v - u\| &= \sqrt{1^2 + 3^2} \\ &= \sqrt{10} \end{aligned}$$

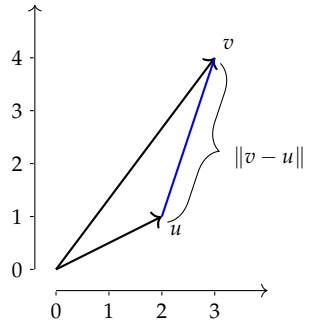


Figure 19: Vector distance

Euclidean distance is symmetric as a metric, i.e. $\|v - u\| = \|u - v\|$. The distance between the two vectors is illustrated in Figure 19.

Vector Angle

For any vector x, we can factor in its magnitude with $\frac{1}{\|x\|}$, we obtain its unit vector:

$$\hat{x} = \frac{x}{\|x\|} \tag{9}$$

of which the length is normalized to 1. We can do the same to another vector y to obtain its unit vector:

$$\hat{y} = \frac{y}{\|y\|} \tag{10}$$

If x and y are both column vectors of the same dimensionality, $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^m$, we can compute their dot product by:

$$\begin{aligned} x^T y &= \begin{pmatrix} x_1 & x_2 & ... & x_m \end{pmatrix} \times \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_m \end{pmatrix} \\ &= x_1 y_1 + x_2 y_2 + .. + x_m y_m \\ &= \sum_{i=1}^{m} x_i y_i \end{aligned} \tag{11}$$

Likewise, we can compute the dot product of their unit vectors:

$$\begin{aligned} \hat{x}^T \hat{y} &= \left(\frac{x}{\|x\|}\right)^T \left(\frac{y}{\|y\|}\right) \\ &= \sum_{i=1}^{m} \frac{x_i}{\|x\|} \frac{y_i}{\|y\|} \end{aligned} \tag{12}$$

This turns out to be the cosine of the angle between the two vectors x and y:

$$\begin{aligned} cosine(x, y) &= \left(\frac{x}{\|x\|}\right)^T \left(\frac{y}{\|y\|}\right) \\ &= \frac{x^T y}{\|x\|\|y\|} \\ &= \sum_{i=1}^{m} \frac{x_i}{\|x\|} \frac{y_i}{\|y\|} \\ &= \frac{\sum_{i=1}^{m} x_i y_i}{\|x\|\|y\|} \end{aligned} \tag{13}$$

Consider the same two vectors discussed earlier:

$$u = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad v = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

Their magnitudes are:

$$\begin{aligned} \|u\| &= \sqrt{2^2 + 1^2} \\ &= \sqrt{5} \\ \|v\| &= \sqrt{3^2 + 4^2} \\ &= 5 \end{aligned}$$

The two vectors can be normalized to unit vectors:

$$\begin{aligned} \hat{u} &= \frac{1}{\sqrt{5}} \cdot \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.894 \\ 0.447 \end{pmatrix} \\ \hat{v} &= \frac{1}{5} \cdot \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 0.6 \\ 0.8 \end{pmatrix} \end{aligned}$$

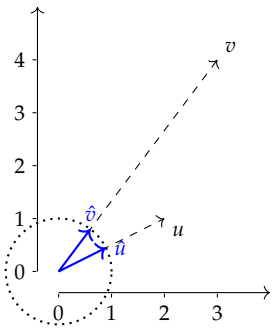


Figure 20: Normalization to unit vectors

As shown in Figure 20, the normalized unit vectors $\hat{u}$ and $\hat{v}$ point to the same directions of their original vectors u and v respectively. However, the normalization produce vectors of equal magnitudes. As the dotted circle shows, the unit vectors are part of a unit circle with radius 1.

Please note that we can use a circle to visualize the unit vectors on the 2-dimensional space (for the 2-dimensional vectors here). For 3 dimensions, we can use a unit sphere for such visualization. For higher dimensionality, however, we can hardly visualize it but the same idea holds true mathematically with the notion of *hypersphere*.

Now the dot product of the two unit vectors is:

$$\begin{aligned} \hat{u}^T \hat{v} &= \begin{pmatrix} 0.894 & 0.447 \end{pmatrix} \cdot \begin{pmatrix} 0.6 \\ 0.8 \end{pmatrix} \\ &= 0.894 \times 0.6 + 0.447 \times 0.8 \\ &= 0.894 \end{aligned}$$

which is the cosine of the angle between u and v, shown in Figure 20. The arc cosine of 0.894 corresponds to an angle of roughly 27° out of a 360° full circle, or 0.465 on a 2π scale. Cosine values range between -1 and 1: it is 1 when two vector are in the exact same direction (parallel) and -1 when they are in the opposite direction (a 180° or π angle). Cosine is 0 when two vectors are orthogonal (perpendicular) to each other (right angle). See Figure .

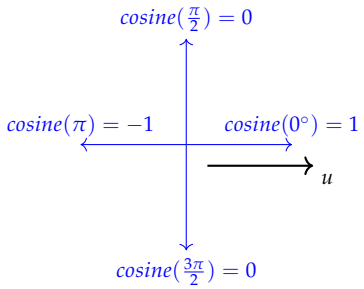


Figure 21: Cosine of special angles

Vector Cross Product

Optimization

Kernel Functions

Graph Analysis

Probabilistic Thinking and Modeling

> Probability theory is nothing but common sense reduced to calculation.
>
> *Pierre-Simon Laplace*

In 2012, a national retailer store broke the news by identifying a teen pregnancy before her father knew about it. After seeing coupons of baby clothes and cribs sent to his daughter, the enraged father went to the store to complain about the incident, only to discover later that he was the one who needed to apologize for his ignorance.

While this story unfolded a spectrum of ethical and privacy-related issues, we will focus on the technical question for now – how did the store figure out the girl was pregnant and it is therefore relevant to send her baby coupons?

As one can imagine, this was based on the evidence of related purchases in the girl's shopping history. But even with that, the store could not be certain about the pregnancy and had to guess how *likely* that was the case. Computing the *likelihood* or *probability* is at the core of the technical problem here. We shall look at the basic theory and rules of *probability* before we can use them to reduce all this to calculation.

Probability

Probability, according to the Oxford Dictionaries, is "the quality or state of being probable; the extent to which something is likely to happen or be the case." While this definition appears common sense and easy to understand, in reality we use the term *probability* or likelihood in two very different ways. In the *Frequentist* view, probability is equivalent to the long-term frequency of an event's occurrence. Flipping a *perfect* coin, for example, has a 50% probability to turn up a head and 50% tail because each side has equal chances (frequencies) if it is repeated for *sufficient* times.

Figure 22: Head vs. Tail

On the other hand, the *Bayesian* view of probability is associated with the degree of belief without complete information or knowledge. Each coin flipping has nothing to do with a long-term repetition and which side it turns up is a result of many factors. Likewise, how likely you are going to get an A on an upcoming test is a belief based on the known and the unknown – for example, how well you are prepared and what questions the test will be comprised of.

These views are rather divided and may lead to different approaches of modeling and interpretation. Nonetheless, many methods and tools of great practical values have been derived from both views. And we can focus on their application without digging deep into the theoretical subtlety.

Probability Basics

We denote the probability of an event A's occurrence as $P(A)$. A probability is a numeric value between 0 and 1:

$$0 \leq P(A) \leq 1 \tag{1}$$

where 0 probability means the event will never occur and 1, or 100%, means it will always occur.

Be A the event of turning up a head in a perfect coin flipping, $P(A) = \frac{1}{2}$; if A is the event of getting number 3 by rolling a dice, we may estimate $P(A)$ to be $\frac{1}{6}$ if we assume six equal sides without additional information.

The probability of the complement of event A, or that A will NOT occur, is denoted by $P(A')$,[3] which can be computed by:

[3] The complementary probability can also be denoted by $P(\bar{A})$, $P(\neg A)$ or $P(A^c)$.

$$P(A') = 1 - P(A) \tag{2}$$

Head and tail on a coin complement each other. Hence:

$$P(Tail) = P(Head') \tag{3}$$
$$= 1 - P(Head) \tag{4}$$

For mutually exclusive events A and B, their union probability, namely the probability that *either one* of them will occur is the sum of their individual probabilities:

$$P(A \cup B) = P(A) + P(B) \tag{5}$$

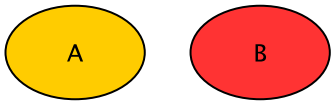


Figure 23: Mutually exclusive events

where the symbol $\cup$ (union) is equivalent to logical OR and implies addition of the events. When two events are mutually exclusive,

they do not occur simultaneously and, as visualized in Figure 23, have *no overlap* or intersection in their probability space. Because of that, we can simply add the two together.

In the case of flipping a coin, Head and Tail are mutually exclusive. By the same rule, we have:

$$P(Head \cup Tail) = P(Head) + P(Tail) \tag{6}$$

which is 1 because Head and Tail complete the entire probability space. Likewise, numbers 1 through 6 on a dice are complete, mutually exclusive events. The same is true that you always get *one* of the six numbers on a dice:

$$\begin{aligned} & P(1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6) && (7)\\ = \; & P(1) + P(2) + P(3) + P(4) + P(5) + P(6) && (8)\\ = \; & 1 && (9) \end{aligned}$$

Joint probability

If events A and B are not mutually exclusive, their union probability should be the sum of their probabilities subtracted by the probability of them occurring simultaneously – that is the joint probability:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{10}$$

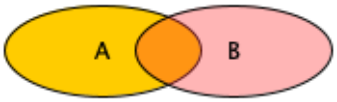


Figure 24: Not mutually exclusive events

The reason for subtracting the joint probability $P(A \cap B)$ can be illustrated in Figure 24. As shown in the figure, when A and B can occur simultaneously, they have an overlap (intersection) which should be discounted so it is not counted *twice* in the addition.

Here is an example of not mutually exclusive events – what is the probability of drawing a card from a deck (52 cards) that is *either* an Ace (4 out of 52) *or* a Spade (13 out of 52). The probability of having each can be estimated by:

$$P(Ace) = \frac{4}{52} \tag{11}$$

$$P(Spade) = \frac{13}{52} \tag{12}$$

Because there is 1 card that is both an Ace and a Spade, the joint probability is $P(Ace \cap Spade) = \frac{1}{52}$ and the union probability can be computed by:

Figure 25: Ace ∩ spade, a joint (AND) event

$$
\begin{aligned}
& P(Ace \cup Spade) && (13)\\
= \; & P(Ace) + P(Spade) - P(Ace \cap Spade) && (14)\\
= \; & \frac{4}{52} + \frac{13}{52} - \frac{1}{52} && (15)
\end{aligned}
$$

Independent events

Again, the joint probability of A and B, denoted by $P(A \cap B)$,[4] is the probability that both A and B will occur.

[4] Joint probability can also be denoted by $P(A, B)$ or P(A AND B).

If A and B are independent, meaning the occurrence of A has nothing to do with the occurrence of B, then the join probability can be computed by the simple multiplication:

$$P(A \cap B) = P(A)P(B) \quad (16)$$

For example, if you flip a coin and roll a dice, how likely will you get a Head *and* number 3. Because the two events are independent, their joint probability can be computed by:

$$
\begin{aligned}
P(Head \cap 3) &= P(Head)P(3) && (17)\\
&= \frac{1}{2} \times \frac{1}{6} && (18)
\end{aligned}
$$

The joint probability is certainly not limited to two events. You can apply the same rule to three or more independent events. For a set of independent events $E = \{E_1, E_2, .., E_m\}$, their joint probability is computed by the product of their probabilities:

$$P(E_1 \cap E_2 \cap .. \cap E_m) = \prod_{i=1}^{m} P(E_i) \quad (19)$$

Again, the equation holds as long as the events are independent – that is, one event's occurrence does not influence the other. While this is often unrealistic, event independence is often assumed for model simplicity.

Dependent Events

Now let's take a step further and look at joint probability of dependent events. If the occurrence of A is dependent on the occurrence of B, we can use $P(A|B)$ to denote the probability of event A *on the condition that* event B has already been observed. This conditional

probability is referred to as the *posterior probability* because it is estimated after the fact (here the observation of event B).

The joint probability that both events A and B will occur is then computed by:

$$P(A \cap B) = P(A|B)P(B) \tag{20}$$

where $P(B)$ is the *prior probability*, which is a priori before the further observation of data. Again, $P(A|B)$ is the *posterior probabilities* conditioned on the observation of the other event and is about the probability that event A will occur if we know event B occurs. Likewise, the joint probability can also be computed by:

$$P(A \cap B) = P(B|A)P(A) \tag{21}$$

This is based on the prior probability of and the posterior probability conditioned on event A. So far the discussion on probability may look quite abstract. So let's take a look at an example to make sense of this.

Suppose you are running a restaurant and you look at your transaction data, and try to learn about the statistics on how to improve your business. Suppose you find out 60% of the customers order burgers, 54% order burgers and fries. Now you want to know – if a customer already orders a burger, how likely he will also order fries. And this question is certainly relevant to your business.

First let's setup proper notation. Let:

- F be the event that a customer orders *fries*, and
- B be the event that a customer orders *burgers*.

In this case, you know 60% of the chance a customer orders burgers, which is $P(B) = 0.60$. With 54% of customers ordering both burgers AND fries, the joint probability of the two events $P(F \cap B) = 0.54$.

Now the question is about the probability of a customer ordering fries (F), *on the condition* that she has already ordered a burger (B), which is the posterior probability $P(F|B)$. By replacing A and B with F and B respectively in Equation 20, we have:

$$P(B)P(F|B) = P(F \cap B) \tag{22}$$

Divide both sides by P(B), we get:

$$P(F|B) = \frac{P(F \cap B)}{P(B)} \quad (23)$$
$$= 0.54/0.60 \quad (24)$$
$$= 0.90 \quad (25)$$

So it appears the customer will probably (with 90% probability) order fries as well if she is ordering a burger.

We can also apply these probability rules to the retailer store story we discussed at the beginning of the chapter. We can ask questions like – if a customer has purchased a crib, how likely will she be interested in coupons of baby clothes? With other related evidence in the shopping history, can we predict whether the customer is pregnant (or how likely she is) and, if so, follow up with customized marketing efforts?

The Bayesian Theorem

To attack the above questions, we need to take one step further and look at the Bayes Theorem.

From equations 20 and 21, we have:

$$P(A|B)P(B) = P(B|A)P(A) \quad (26)$$

Divide both sides by $P(B)$, we get:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \quad (27)$$

And with this, we arrive at a very important probability rule, the *Bayes' Theorem* (rule) on computing posterior probabilities. It is named after Rev. Thomas Bayes[5] and further developed by Pierre-Simon Laplace. The rule provides a foundation for various applications with Bayesian inference and probabilistic modeling.

[5] Richard Price helped publish Bayes' essay with the formula in 1763, after he passed away. Price, a moral philosopher, regarded Bayes Theorem as a means to reason on the probability of miracles and to prove the existence of God.

Naive Bayes' Model

Let us look at an application of the Bayes Theorem to get a sense of how useful the theory is in solving practical problems. One such application is the Naive Bayes model, which is widely adopted in machine learning, data mining, and information retrieval. Pay attention to the two keywords – *Naive* and *Bayes* – the model is *Naive* because the it has naive assumptions about variable *independence*, and is *Bayesian* because the core of the model is the Bayes' Theorem.

Ultimately, the Naive Bayes model is to estimate the probability of an inference (hypothesis H) given observed data (evidence E), that is, in the binary scenario, for example, $P(H|E)$ (true) vs $P(H'|E)$ (false). We can compute these probabilities according to the Bayes' rule:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{28}$$

$$P(H'|E) = \frac{P(E|H')P(H')}{P(E)} \tag{29}$$

where prior probability $P(E)$ is the common denominator that does not differentiate the two and can be ignored for now. $P(H)$ and $P(H')$ can be estimated a priori without evidence E, whereas $P(E|H)$ and $P(E|H')$ can be computed using data observed, in cases when the hypothesis holds or when it does not.

We shall use a specific example to show how it works. In clinical diagnosis, a doctor observes a patient's symptoms (evidence) and determines whether the patient has certain disease (inference / hypothesis). Chickenpox, for example, exhibits symptoms such as fever and blisters (rash). Let:

- E_1 denote the symptom of fever, and
- E_2 denote the symptom of blister (rash).

The question is how likely the patient has developed chickenpox (hypothesis H) if there are symptoms of both fever E_1 and blisters E_2.

Suppose[6] the prevalence of chickenpox is 20 in every $1,000$ individuals, that is $P(H) = 0.02$ and $P(H') = 0.98$; of those *who develop chickenpox*, 90% have fever ($P(E_1|H) = 0.90$) and 55% have blisters ($P(E_2|H) = 0.55$); of those *without chickenpox*, the chances of have these individual symptoms are $P(E_1|H') = 0.10$ for fever and $P(E_2|H') = 0.05$ for blisters.

[6] Numbers in the chickenpox example are hypothetical and only intended for the sake of computational exercise here.

The first probability we need to attack is $P(E|H)$, the probability that a patient exhibiting the observed symptoms *if* she indeed has chickenpox. Because we consider two symptoms here, this is about the joint probability when fever and blisters appear simultaneously. With the *Naive* assumption that the symptoms' occurrences are statistically independent, $P(E|H)$ can be computed by:

$$P(E|H) = P(E_1|H)P(E_2|H) \tag{30}$$

Put this in Equation 28 and we get:

$$P(H|E) = \frac{P(E_1|H)P(E_2|H)P(H)}{P(E)} \quad (31)$$
$$= \frac{0.90 \times 0.55 \times 0.02}{P(E)} \quad (32)$$
$$= \frac{0.0099}{P(E)} \quad (33)$$

Likewise, we can compute the probability of a false hypothesis given the same evidence (symptoms) by:

$$P(H'|E) = \frac{P(E_1|H')P(E_2|H')P(H')}{P(E)} \quad (34)$$
$$= \frac{0.25 \times 0.02 \times 0.98}{P(E)} \quad (35)$$
$$= \frac{0.0049}{P(E)} \quad (36)$$

The result shows $P(H|E) > P(H'|E)$, meaning that it is more likely than not the patient has developed chickenpox. Again, this model is based on the Bayes' theorem and the assumption that evidential events (symptoms) observed are statistically independent. In another word, the occurrence of one symptom has nothing to do with the other. In reality we know this independence assumption is rarely true. For example, one who has blisters is more likely to have fever than those without.

In the above example, we only use two pieces of evidence (symptom observation) to model the likelihood of chickenpox. In reality, this can be extended to any number of predictor variables where E is a set of variables $E_1, E_2, .., E_m$ and m is the size of the set. In this general case, the Naive Bayesian Model in equation 28 can be extended to:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \quad (37)$$
$$= \frac{P(E_1|H)P(E_2|H)..P(E_m|H)P(H)}{P(E)} \quad (38)$$
$$= P(H)\prod_{i=1}^{m} P(E_i|H)/P(E) \quad (39)$$

Likewise, on the probability of a false hypothesis:

$$P(H'|E) = P(H')\prod_{i=1}^{m} P(E_i|H')/P(E) \quad (40)$$

where variables $E_1, E_2, .., E_m$ are assumed to be statistically independent. The main purpose of having this naive assumption is for mathematical simplicity when we compute joint probabilities. Although this assumption is at odds with reality, related models have turned out to perform quite well in experiments. Practically the number of variables (features) m can be very large. For example, in text analysis, it is common to treat unique words as individual features (variables) and, with even a moderate text collection, there can be tens of thousands, if not millions, of variables in the model.

The Bayes Model is not limited to a binary case, e.g. chickenpox vs. not chickenpox. It can also be applied to any classification problem on a number of discrete outcomes. In that case, we simply need to compute the probability of each discrete outcome (hypothesis) and find out which one has the greatest likelihood. It can also be extended to a continuous case where numeric predictions can be made based on certain probability distribution models, which we will discuss later in the chapter.

Probability Estimators

Again, consider Equation 39, in order to compute the probability of each hypothesis, we need to first estimate the probability of each evidence E_i in $\prod_{i=1}^{m} P(E_i|H)$. Where do these probabilities come from? Essentially they are learned from the training data where we can count their frequencies, or the number of times each unique situation occurs. For now, the probability $P(E_i|H)$ is computed by:

$$P(E_i|H) = \frac{F(E_i|H)}{F(H)} \tag{41}$$

where $F(E_i|H)$ is the number of instances with E_i where hypothesis H is true and $F(H)$ is the total number of instances with the true hypothesis.

Imagine you are working on the problem of email spam detection. You have a training data set of $1,000$ emails and 100 have been identified spams (where hypothesis H_{spam} is true). Suppose *prize* is one of the keywords (evidence E_{prize}) that can be used to model the spam detection problem and, of the 100 spam emails (H_{spam}) there are 20 containing that keyword *prize*. In this case, $F(H_{spam}) = 100$ and $F(E_{prize}|H_{spam}) = 20$ so, according to the probability estimator in equation 41, the probability $P(E_{prize}|H_{spam})$ can be computed by:

$$\begin{aligned} P(E_{prize}|H_{spam}) &= \frac{F(E_{prize}|H_{spam})}{F(H_{spam})} \\ &= 20/100 \\ &= 0.2 \end{aligned}$$

This will work just fine as long as the relative frequencies (occurrences) of related keywords in the training data (*sample*) are proportional to those of the same keywords in all spam emails in the world (*population*). In this case, we assume what we observe in the sample data is what will most likely occur in the population.

In reality, however, we may observe data in samples with varied departures from the entire population. Imagine we want to use *prize* as one feature in the Naive Bayes but observe no occurrence at all of this term in the training spam data, that is $F(E_{prize}|H_{spam}) = 0$.

In this case, are we to simply conclude the probability of having *prize* in *any* spam email $P(E_{prize}|H_{spam})$ is 0? This is obviously a premature conclusion – the fact that you have not seen an *elephant* does not lead you to conclude on the non-existence of that very creature. There is variance in the data and we do not want to be misled by data in its departure (skewness) from the overall reality.

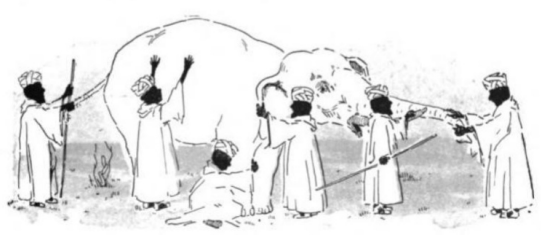


Figure 26: Blind men and elephant. If you have not seen an elephant, does it mean there is no elephant? If you only touch an elephant on its leg, does it mean the elephant is like a tree trunk? These misperceptions are due to our sample data (where you touch the elephant) and should be corrected with external knowledge.

Data may be localized, just as the story of *blind men and elephant* shows (Figure 26). But we should aim to see the whole picture.

In addition, there is also an undesired consequence for having a zero probability in the Naive Bayes model. With E_{prize} as one of the evidence features, its probability $P(E_{prize}|H_{spam})$ is one of the factors in computing the product in *Equation* 39. When $P(E_{prize}|H_{spam}) = 0$, it leads to $P(H|E) = 0$ regardless of the values of other variables.

In short, we have observed that zero probabilities are not desirable, neither theoretically nor practically. One simple remedy to zero probability estimates, as suggested by Pierre Laplace, is to add 1 to each frequency (count) in the sample. So the frequency of having term *prize* in the spams becomes:

$$F(E_{prize}|H_{spam}) + 1 \tag{42}$$

To be fair, we also need to add 1 to other frequency counts, including the frequency of NOT having the term *prize* in the spam subset:

$$F(E'_{prize}|H_{spam}) + 1 \tag{43}$$

The above two are complete and mutually exclusive cases. Hence, the total number of instances in the spams becomes:

$$\begin{aligned} & F(E_{prize}|H_{spam}) + 1 + F(E'_{prize}|H_{spam}) + 1 && (44) \\ = \quad & F(H_{spam}) + 2 && (45) \end{aligned}$$

Note that we add 2 to the total number because we have a binary variable, namely having term *prize* (true) vs. not having it (false). If a variable has k discrete values, then we need to add k to the total. For example, a categorical variable that marks an email's importance may have three different levels such as *low*, *normal*, and *high*. If we add 1 to the frequency of each level, in the end we will have added 3 to the total.

But the question then is why add 1? In fact the idea is that we can add any *constant* c to each count so zero values can be avoided. And the probability of observing a variable value v within the sample where the hypothesis is true can be computed by:

$$P(E_i = v|H) \quad = \quad \frac{F(E_i = v|H) + c}{F(H) + kc} \tag{46}$$

where k is the number of discrete values variable E_i can have, e.g. $k = 3$ for the email priority variable, and c is a chosen constant which represents some prior knowledge about the chance of any value. A great c value will make the probability more predetermined and a lesser value gives relatively more weigh to frequencies observed in the sample data. When a discrete value occurs less frequently in the sample, c will change its probability estimate more dramatically.

In a sense, an added constant c favors the rare values and penalizes the frequent ones. Other than simplicity, is this a rational strategy to estimate the probabilities? The answer depends on how much we know about the probability distributions with or without the sample data. If in fact we know some values are going to occur more frequently regardless – for example, we can safely assume most emails would have been marked as *normal* – then we should add more to its count proportionately in the sample data. The probability can be estimated by:

$$P(E_i = v|H) \quad = \quad \frac{F(E_i = v|H) + cp_v}{F(H) + kc} \tag{47}$$

where p_v is the *a priori* probability of each discrete value v ($v \in [v_1, v_2, .., v_k]$) and they add up to $\sum_v p_v = 1$. The assignment of p_v values depends on *prior knowledge*. Practically it is not always clear how they can be obtained. Besides domain knowledge, training samples are often the major source where we gather related statistics so the simple transformation of adding 1 or another constant is what works in many practical applications.

Probability Distributions and Models

Think about this. The very fact that we have to *normalize* the obtained frequencies – to avoid zeros or for other purposes – indicates that we do not totally trust what observed data (samples) can tell us about. The underlying assumption is that we have certain understanding (prior knowledge) about how data should behave and we will need to *correct* the data whenever they *drift* from that expected behavior. By doing the normalization or smoothing, we impose a *model* of expected *probability distributions* on the analysis.

We shall now prepare us with an introductory discussion of probability distributions, from discrete to continuous cases.

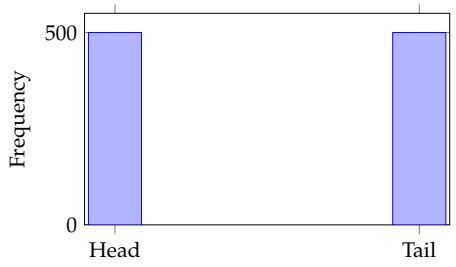


Figure 27: Frequency distribution of flipping a coin, 500 heads vs. 500 tails after 1,000 trials in the ideal world.

First let's look at some discrete cases. Imagine you toss a coin for 1000 times and in the perfect world you get 500 heads and 500 tails. And you can create a simple bar plot to show the frequency distribution of 500 for both heads and tails. This is a uniform distribution because every event, head vs. tail, has an equal chance.

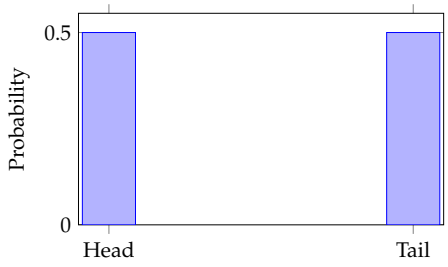


Figure 28: Probability distribution of flipping a coin (500 heads vs. 500 tails).

Now instead of talking about frequencies or counts, we can de-

scribe the distribution in terms of chance (probabilities). And in this case, head and tail have equal chances (probabilities) of 50% or 0.5 in the distribution, as Figure 28 shows.

For the sake of discussion, let's extend the binary case, head vs. tail, to a larger number of exclusive outcomes. For example, imagine you roll a dice. If you roll it for a sufficient number of times, the sides will have equal chances to appear, that is, each side from 1 to 6 has a probability of 1/6 to occur, as shown in Figure 29.

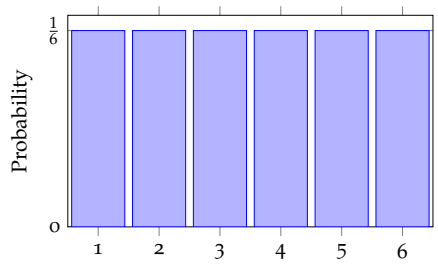


Figure 29: Probability distribution of rolling a dice

Two simple but important observations here:

1. The probabilities of the six sides add up to one because they are exhaustive, mutually exclusive choices;
2. The probability distribution is a uniform distribution as they are all equal. The heights of the bars form a flat, horizontal line that characterizes the uniform distribution.

Continuous Probability Distribution

So far these are discrete distributions. But we can extend this to a continuous distribution. If the outcome x is no longer limited to such integers or specific labels. Perhaps the outcome of the question at hand can be any number in the range of 1 and 6, and you are to predict that number. If it is a uniform distribution in with the continuous variable, it remains a flat line in the range of 1 - 6, shown in Figure 30.

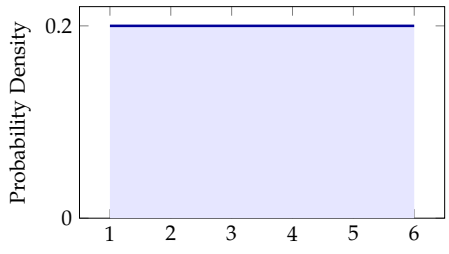


Figure 30: Probability distribution of a continuous variable x

Again, when we add all probabilities up in the distribution, the total should still be 1. But because this is a continuous space, we

are talking about infinite amount of numbers. The outcome can be number 3 or any decimal number such as 3.14159. To add the infinite number of heights (some sort of "probability" values) in the distribution is essentially *integration* of the distribution function (a flat line in this case), that is, the area under the distribution line or curve. The result of the integration or the entire area will *always* be 1, as long as the plotted range includes all possible outcomes.

With a uniform (flat) distribution, the area is a rectangle as shown in Figure 30 and it is easy to calculate the area. In the above case, the area of the rectangle between 1 and 6 can be computed by width times height, which is $(6-1) \times 0.2 = 1$. So this meets the expectation that the sum of the entire probability distribution is 1.

Although the above is discussed as a *probability* distribution, we shall clarify on an important concept. The uniform height 0.2 in Figure 30 is NOT a probability value per se. It is what we refer to as a *probability density*, but not exactly a probability. For now, let us move on from the uniform distribution to a more common distribution in the real world before we come back to elaborate on the concept of *probability density* again.

In the real world, you do not often see uniform distributions[7]. Normal (Gaussian) distributions are much more common. For example, if we collect statistics about adult male heights in the U.S., you will see the heights are not equally likely. Instead, the overwhelming majority of (average) people have the average height around 70 inches. And much rarer are shorter than 62 or taller than 78. The whole distribution is a bell curve as shown in the figure here.

[7] Even in an equal-opportunity society, there is rarely any even (uniform) distribution of goods, wealth, etc.

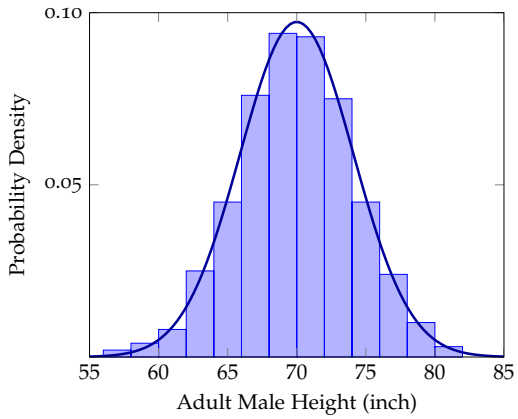


Figure 31: Distribution of adult male heights, approximately a normal distribution (bell curve)

This is again a probability density distribution so if you integrate the curve, the total area under it will be 1 or 100% probability. When you pick a particular number on the x axis such as 70 inches in the middle, you notice it is $y = 0.1$ on the distribution curve. Now be careful that this value 0.1 does NOT mean that there is 10% chance to

be 70 inches height (exactly).

Because they are infinite number of values in the range, each exact height value in fact has a 0 probability to occur. The reason is when we say 70 inches, we mean exactly 70.000000... with an infinite number of decimal zeros. And it is close to *impossible* to get that exact value on a continuous scale.

Back to the question. So what does a value like 0.1 mean on the y axis? It means 0.1 probability density – that is, 0.1 probability *per inch* (range) on the adult height scale. And this value will change if you change the unit of height, for example by switching to centimeters.

What does this probability density entail now that it is not probability and it varies on the measuring unit? Probability density does allow you to compare probabilities of different continuous *values* and *value ranges*. For example, in the normal distribution here, you can still tell that the probability of a height *around* 70 inches is greater than that in the proximity of 80. But to tell the exact probability, you will need to specify a range. You can ask the question of how likely an adult height is 70 *something* (i.e. between 70 and 71) inches. And you can integrate the area under the bell curve in the range to find out.

To reiterate this point – on the simpler example of the uniform distribution presented in Figure 30, the 0.2 value on y is not probability but probability density or probability per unit. You can still tell that all probabilities are equal by looking at the flat line. And you can ask the question about how likely you will get a value between 2 and 4 by multiplying the density which is 0.2 and the range which is 2 and the result turns out to be 0.4 – that is 40% of the chance you will get a value *in that range*.

As always, all probabilities add up to 1 as you can verify by $0.2 \times (6 - 1) = 1$. So if you ask how likely a value occurs in the range between 1 and 6, given the distribution in Figure 30, the answer is that it will *always be true*.

There are several major categories of probability distributions beyond the uniform and normal distributions we discussed here. Some of these are asymmetric, exhibiting a form of imbalance between the lower and higher ends. Power-law distributions and their variations, for example, are very common in many natural and social structures. They are often associated with basic *counts* in these structures and activities.

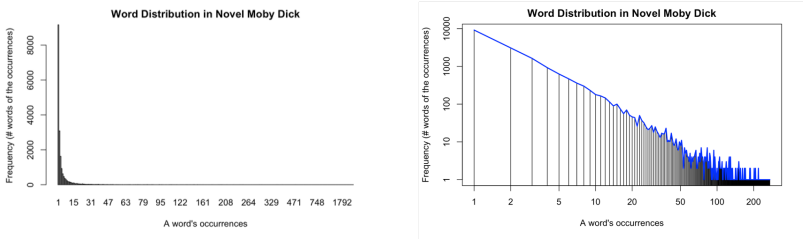
The distribution of wealth or income is an example of the power-law distribution, where the overwhelming majority is *poor* (with relatively little wealth) and a small percent at the top are extremely wealthy. This is a result commonly regarded as due to the *Matthew Effect*[8], in which the rich get rich. In citation and network analysis,

[8] The term *Matthew Effect* is named after the Gospel of Matthew, in which Jesus is recorded as saying, "for to every one that hath shall be given, and he shall abound: but from him that hath not, that also which he seemeth to have shall be taken away" with the parable of the talents.

this effect is also known as the *cumulative advantage*[9] or *preferential attachment*[10].

Spoken language (word frequencies) follows the Zipf law, which can be regarded as a special case of the power-law distribution. Figure 32 shows the distribution of word frequencies taken from the novel *Moby Dick*. As shown in Figure 32 (a), there is a great divide between the rich and the poor – a large number of words have only been used once or twice in the novel (tall bars on the left of the distribution) where very few words have a frequency higher than 100 (the long tail).

It is difficult, however, to examine the long tail on normal coordinates as the values are very small (low) within a long range. Logarithmic transformation is often conducted to better visualize the distribution. The result is Figure 32 (b), where both x (words' occurrences) and y (frequencies of the occurrences) have been transformed with logarithm.[11]



(a) normal coordinates (b) log-log coordinates

Figure 32: Word frequency distribution

One prominent characteristic of a power-law distribution is that it follows a straight line on log-transformed coordinates spanning across a vast range of numeric orders and is therefore considered *scale free*. One should caution though that many distributions may roughly look linear with log-transformation when they are in fact the result of a different distribution. In reality, there are various constraints and capacity limits on related structures where the outcome is not ideally power-law. In social networks, for example, even those who are super active can only maintain up to a certain number of friendly contacts and therefore the outcome of friendship distribution will be subject to such human limits and not scale-free.

Probability Estimators

The first approach we used to compute probability is to follow whatever the sample data tell us. If we take a sample of 1000 emails and 300 of them are spams, the estimated probability is given by $300/1000 = 0.3$. If we analyze 100 emails and *NONE* of them have the term *prize* in them, then we are bound to conclude the probability

[9] Derek De Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American Society for Information Science*, 27(5):292–306

[10] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999

[11] Essentially log-transformation reduces a number to its order, e.g. 1000 to 3 and 100 to 2 with log_{10}.

that we will ever encounter the term is 0. This naive method of trusting whatever in the sample data is part of a spectrum of methods referred to as the *Maximum Likelihood Estimator* (MLE).

Simply put, an MLE chooses as estimates the values that maximize the likelihood of the observed sample. In the case of observing 0 emails with term *prize* in the sample, we shall thus estimate the term never occurs in any email in the whole world (population) so that NOT observing it is the most likely outcome, which is the case of the observed sample. In general, for discrete distributions, the result of an MLE is given by frequencies of related values in the sample without further transformation.

Discrete Probability Estimation

Now if we have additional knowledge about the domain and can assume probabilities follow a particular distribution model, then the question becomes how we can specify parameters of that model to maximize the probability of the observed data. Suppose among all emails you receive, you find out for three particular days there are 5, 3, and 7 emails containing the term *prize* on each day. We assume that:

- The number of such emails you receive one day is independent of that on another day. That is, receiving more or less such email today does not affect the number you will receive tomorrow and in the future.
- During a small time interval, the probability of receiving such an email is proportional to the length of the time interval. That is, the longer the time, proportionally greater the probability.

Under these conditions, the probability of receiving x number of *prize* emails can be modeled by a Poisson probability distribution and the probability mass function is:

$$P(x) = e^{-\mu}\frac{\mu^x}{x!} \tag{48}$$

where:

- e is the Euler constant which is approximately 2.71828;
- x is the number of such emails in question, and $x!$ is the factorial $x! = x \times (x-1) \times (x-2) \times .. \times 2 \times 1$;
- u is the only parameter, the mean number of *prize* emails, to be estimated.

With an MLE, we are to maximize the likelihood of obtaining the 3-day sample, that is the joint probability of having 5, 3, and 7 emails $P(x = 5 \cap x = 3 \cap x = 7)$ on three separate days. Now that we assume the days are independent, the joint probability can be computed by:

$$P(x = 5 \cap x = 3 \cap x = 7) \tag{49}$$
$$= P(x = 5)P(x = 3)P(x = 7) \tag{50}$$

With the Poisson probability function in Equation 48, this becomes:

$$P(x = 5 \cap x = 3 \cap x = 7) \tag{51}$$
$$= e^{-\mu}\frac{\mu^5}{5!} \times e^{-\mu}\frac{\mu^3}{3!} \times e^{-\mu}\frac{\mu^7}{7!} \tag{52}$$
$$= \prod_{x \in [5,3,7]} e^{-\mu}\frac{\mu^x}{x!} \tag{53}$$

Now MLE of the Poisson distribution is to estimate the only parameter, the value of mean μ, so that the above probability in Equation 53 is maximized. Now instead of maximizing the product (multiplications) of the probabilities, we can take the logarithm of Equation 53, which becomes:

$$l(\mu) = \ln(\prod_{x \in [5,3,7]} e^{-\mu}\frac{\mu^x}{x!}) \tag{54}$$
$$= \sum_{x \in [5,3,7]} \ln(e^{-\mu}\frac{\mu^x}{x!}) \tag{55}$$

and try to maximize the sum of log likelihood[12]. We can do this by taking the derivative of Equation 55:

$$l'(\mu) = \frac{1}{\mu}\sum_{x \in [5,3,7]} x - n \tag{56}$$

where n is the number of observations of the sample, which is 3 for 3 days. The maximum can be reached with the derivative at 0, which is:

$$\frac{1}{\mu}\sum_{x \in [5,3,7]} x - 3 \tag{57}$$
$$= \frac{5+3+7}{\mu} - 3 \tag{58}$$
$$= 0 \tag{59}$$

[12] With a positive variable such as a probability, when its value p increases, its $log(p)$ always increases. So when $log(p)$ reaches its maximum, so does the p value. Because of this, we can maximize $log(p)$ in order to maximize p.

And we can conclude that:

$$\begin{aligned} \mu &= (5+3+7)/3 && (60)\\ &= 5 && (61)\end{aligned}$$

maximizes the probability of observing the sample data of 3, 5, 7 on three days following the Poisson distribution. With the estimate for parameter $\mu = 5$, we can now compute the probability for any discrete value with Equation 48. For example, we can find out that it is very unlikely on one day to receive 10 emails containing the term *prize*:

$$\begin{aligned} P(x=10) &= e^{-\mu}\frac{\mu^x}{x!} \\ &= e^{-5}\frac{5^{10}}{10!} && (62)\\ &\approx 0.01 && (63)\end{aligned}$$

We can also make it a general case that, a Poisson MLE shall estimate the paramter μ based on the sample mean:

$$\begin{aligned} \mu &= \bar{x} && (64)\\ &= \frac{\sum_{i=1}^{n} x_i}{n} && (65)\end{aligned}$$

where n is the number of observations in the sample data.

MLE for a Continuous Variable

For a continuous variable, we can also assume a specific functional form of its distribution such as a normal distribution and the objective of an MLE is to estimate related parameters (e.g. mean μ and variance δ^2) of the distribution function, from which the sample data are most likely to be observed. These parameter estimates are then used to compute probability densities at specific values or probabilities within certain ranges.

Suppose X is a numeric variable with a normal distribution, the probability of any value x is given by the following function:

$$f(x) = \frac{1}{\sqrt{2\pi\delta}} e^{-\frac{(x-\mu)^2}{2\delta^2}} \quad (66)$$

This is one of the most recognized formulas in statistics. Although it looks daunting, there are only two parameters and the rest are constants, including π the ratio of a circular circumference to its diameter which is about 3.14159. From a sample data of n instances, the

two parameters, namely mean μ and variance δ^2, can be estimated by the sample mean and variance:

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{67}$$

$$\hat{\delta^2} = \frac{\sum_{i=1}^{n} (x_i - \hat{\mu})^2}{n-1} \tag{68}$$

where x_i is each observed value of x in the sample data.

Normal distributions are symmetric with the highest probability density at the very center of the bell curve. Given the objective of MLE to maximize the likelihood of observed sample, it makes sense that the parameter estimates should ultimately put the sample data at the center (so they are more likely to occur). Now that the center is also the mean of the symmetric distribution, we can now estimate the population mean using directly the mean of the sample, which is what Equation 68 does.

Summary

Because it is probabilistic, we assume there is no absolute certainty. So any probability estimate is a guess based on the data you have and best evidence you can identify. There is always a chance it may go wrong. And so we are not to blindly trust the outcome/prediction of a model (neither probabilistic nor any other models). In many critical settings, human machine collaboration is key – in the medical environment, there have been retrieval and AI systems that can provide necessary assistance to technicians and physicians, but they are not there to take over and do everything for us. That said, we also have to recognize there are human errors as well. When we think and make decisions, even as professionals/experts, we cannot be 100In short, the point is – models are not perfect, as humans are. So the best way is to understand what they are really good at and in what ways we can use their assistance.

Statistical Analysis

Information in the Chaos

It from bit.

John Archibald Wheeler

Imagine taking a quiz consisting of 3 true/false questions. If one has not studied the subject, she can take a guess on each of the questions and there are 8 (i.e. $2 \times 2 \times 2 = 2^3$) possible sets of answers to the entire quiz. Without the ultimate knowledge about what is correct, each set of answers has a likelihood of $p = 1/8$ to be the correct one.

T	T	T	T	F	F	F	F
T	T	F	F	T	T	F	F
T	F	T	F	T	F	T	F
1	2	3	4	5	6	7	8

Table 5: All possible sets of answers to a quiz of 3 binary questions. Each answer set has a probability of 1/8 to be the correct one.

Now if we assume the 3 questions are independent (without an overlap of knowledge), one needs 3 piece of information to answer them. Therefore, the amount of information required to find the correct answer is proportional to:

$$\begin{aligned} H &= 3 \\ &= \log_2 2^3 \\ &= -\log_2 \frac{1}{8} \\ &= -\log_2 p \end{aligned}$$

where p is the probability of one out of 8 possible outcomes and the log base 2 reflects that fact that each question is binary (true or false). In the above 8 mutually exclusive answer sets, each represents a choice and adds to the uncertainty of getting the correct set of answers. Of the overall $-\log_2 \frac{1}{8}$, each answer set i of the 8, if treated equally likely, requires the following amount to be confirmed or eliminated as the ultimate outcome:

$$H_i = -\frac{1}{8}\log_2\frac{1}{8} \quad (1)$$

where $\frac{1}{8}$ can be regarded as the probability of i^{th} outcome p_i:

$$H_i = -p_i \log_2 p_i \quad (2)$$

This quantity can be linked to Shannon's information theory, also known as the Mathematical Theory of Communication, in which the amount of (missing) information can be measured by the *Entropy* of a probability distribution.[13]

[13] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948

Shannon Entropy

Indeed, Shannon views communications as a process through which a sequence of symbols are produced by a random (unpredictable) means. The degree of unpredictability or uncertainty is due to the lack of information. Thus information can be measured by the reduction of entropy.

We can equate the process of communicating a message from a set of symbols with predicting on a set of events or inferences. Given a set of m mutually exclusive events, its Shannon entropy can be computed by:

$$H = \sum_{i=1}^{m} -p_i \log p_i \quad (3)$$

where p_i is the probability of the i^{th} event (inference). The Entropy represents the amount of missing information in the probability distribution. We may discuss the amount in terms of *bits* if 2 is used for the log base. The example of Equation 1 is a special case of its application, where there are 3 pieces (bits) of missing information for 3 binary questions:

$$H = \sum_{i=1}^{8} -\frac{1}{8}\log_2\frac{1}{8} \quad (4)$$

$$= 3 \quad (5)$$

Properties of Shannon Entropy

One may question why this has to be *the* equation for entropy. In fact, Shannon started with several desired properties about such an *entropy* measure before he concluded that this is the only function

that meets the expected properties. We shall discuss some of the important properties.

Figure 33 shows that the entropy function is smooth with regard to changes in the probability p_1. When the two mutually exclusive events are equally likely $p_1 = p_2 = 0.5$, the entropy of the system is at its maximum (the peak in the middle).

It has been proved that, with any number of mutually exclusive events, the greatest entropy is achieved when all events are equally possible. Imagine a number of applicants compete for the same position. When everyone has an equal chance and there is no leading candidate, it is the most uncertain situation with the greatest entropy.

Figure 34 plots entropy H vs. the number of equally likely events m. When the number of events (choices) increases, so does the entropy. With an increasingly larger pool of applicants for a position – supposedly with equal chances – the harder it is to predict who will be hired in the end.

A third property of the Shannon entropy is related to how the entropy of one system can be computed based on entropies of its sub-systems. Let us look at a simple example: Three candidates running for office, with candidate 1 from party A and candidates 2 and 3 from party B. Suppose their probabilities of winning the election are estimated as follow:

$$\begin{aligned} p_1 &= 0.3 \\ p_2 &= 0.2 \\ p_3 &= 0.5 \end{aligned}$$

We can easily compute the overall entropy by:

$$\begin{aligned} H_{overall} &= -0.3 \log_2 0.3 - 0.2 \log_2 0.2 - 0.5 \log_2 0.5 \quad (6) \\ &= 1.485 \quad (7) \end{aligned}$$

We can also look at the sub-systems, i.e. parties A and B, and their entropies. The probability of party A is the same as that of candidate 1, $p_A = p_1 = 0.3$, whereas the probability of party B is the sum of its candidates' probabilities $p_B = 0.2 + 0.5 = 0.7$. Therefore, the two-party system has an entropy of:

$$\begin{aligned} H_{AB} &= -0.3 \log_2 0.3 - 0.7 \log_2 0.7 \quad (8) \\ &= 0.881 \quad (9) \end{aligned}$$

Party A only has one candidate, which has a 100% chance to "win" within the party, and there is no uncertainty:

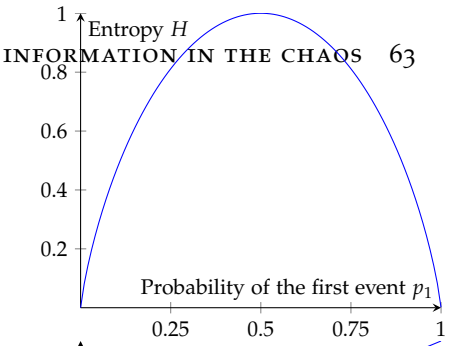


Figure 33: Shannon Entropy of two mutually exclusive events

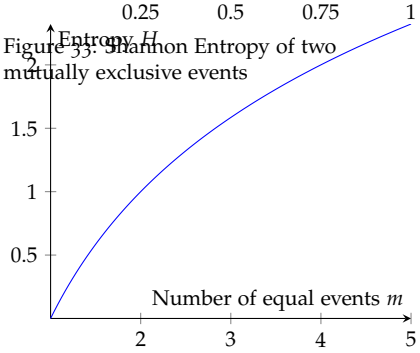


Figure 34: Shannon Entropy of m equally likely events

$$\begin{aligned} H_A &= -1 \times \log_2 1 \\ &= 0 \end{aligned}$$

Of party B, there are two candidates with overall probabilities $p_2 = 0.2$ and $p_3 = 0.5$, which can be translated into the following probabilities *within* the party:

$$\begin{aligned} p_{B2} &= \frac{0.2}{0.2 + 0.5} \\ &= \frac{2}{7} \\ p_{B3} &= \frac{0.2}{0.2 + 0.5} \\ &= \frac{5}{7} \end{aligned}$$

Therefore, the sub-system entropy of party B is:

$$\begin{aligned} H_B &= -\frac{2}{7}\log_2\frac{2}{7} - \frac{5}{7}\log_2\frac{5}{7} && (10) \\ &= 0.863 && (11) \end{aligned}$$

In the end, the entropy of the entire system overall can be computed by the weighted sum of its sub-systems:

$$\begin{aligned} H_{overall} &= H_{AB} + p_A H_A + p_B H_B && (12) \\ &= 0.881 + 0.3 \times 0 + 0.7 \times 0.863 && (13) \\ &= 1.485 && (14) \end{aligned}$$

This is exactly what we got in Equation 7, where we looked at the overall of three candidates regardless of their parties (sub-systems). It can be shown that the overall entropy of any probability distribution is the same as the weighted sum of entropies of its sub-systems.

Entropy in Thermodynamics?

The notion of *entropy* predated Shannon's information theory, and had been used in thermodynamics and statistical mechanics. Historically the information-theoretic entropy has been taken as a completely separated development and is only mathematically analogous to the same notion in thermodynamics. Many regarded the identical name as an accident.[14]

Nonetheless, several prominent theoretical physicists, including the late John A. Wheeler, have argued that the two are equivalent if

[14] According to one story, John von Neumann advised Shannon to use the term *entropy* for two reasons: first because it is something already used in statistical mechanics, and second, "as nobody knows what entropy is, whenever you use the term you will always be at an advantage."

given the same dimensionality.[15] While it is tempting to elaborate on the debate, for now we shall avoid the unsettled issues but emphasize a few useful points drawn from statistical mechanics.

In thermodynamics, entropy measures the degree of uncertainty in the distribution of microscopic states, e.g. the distribution of temperature in a body of gas. This uncertainty, sometimes referred to as some *disorder* – not of the system itself, but according to the human "eye" – can be attributed to the lack of information or knowledge about the states.

The second law of thermodynamics asserts that the *entropy* of a closed system, without external energy input, will never decrease. That is, it requires energy to reduce entropy and restore *order*. While the thought experiment of *Maxwell's demon* appears to suggest that this second law can be violated, some physicists had resorted to the notion of *information* in refuting the paradox between energy and entropy.[16]. One can argue that obtaining information (e.g. about "active" particles) requires energy and the outcome of Maxwell's demon does not violate the second law of thermodynamics. In this reasoning, there appears to be some form of connection between information and entropy even in the physical reality of thermodynamics.

In statistical mechanics, entropy is a measure of uncertainty given the potential number of arrangements or configurations. For example, thermal entropy of gas is due to the lack of knowledge about its particle positions and velocities. It is a macroscopic measure of probability distributions on related microscopic states. Mathematically, this is consistent with quantifying entropy in the arrangements of coding and communication given a probability distribution.

[15] Jacob D. Bekenstein. Information in the holographic universe. *Scientific American*, 289(2):58–65, 2003

[16] The Maxwell's demon goes like this: If we place a demon at the gate between two chambers of gas and it only allows active particles to pass the gate in one direction, then the overall entropy of the two chambers as one closed system will decrease

Applications

Shannon's mathematical theory of communication, commonly known as the information theory, has been used in a wide spectrum of areas including digital coding, communication, and information technology applications.

On binary media such as digital computer systems, for example, Shannon entropy offers the basic unit of *bit* to quantify the amount of information and the necessary space to store and process it. It also enables a fundamental paradigm shift in the study of communication channel capacity, from signal power (energy) to bandwidth (frequency).

Modeling information as reduction of entropy (uncertainty) provides a valuable vehicle in the design and engineering of digital information systems. In information retrieval and text mining, for example, information and probability theories have provided important

guidance to the development of classic methods for representation, ranking, and classification.[17] For example, computing information amounts of words based on their probability distributions enables the weighing and ranking of terms for text representation.

[17] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, April 2009

Let's look at the following example where entropy offers a powerful evaluation tool –

Data mining algorithms often require certain evaluations conducted and decisions made during related training or modeling processes. For example, decision tree construction needs a method to evaluate how good a tree branch is given its membership composition. Data clustering aims to put like entities together and one needs to measure how similar or dissimilar entities are within each cluster. In any of these applications, a good metric of internal homogeneity or heterogeneity (within each decision tree branch or cluster) is critical for the success of related processes.

Suppose we are to group a set of shapes (of types △, ♡, ◯) into related clusters or branches, and Table 6 shows four clusters with different membership compositions.

(1)	(2)	(3)	(4)
△ △ △	△ △ ♡	△ △ △	◯ ◯ ◯
♡ ♡ ♡	♡ ◯ ◯	△ ◯ ◯	◯ ◯ ◯
◯ ◯ ◯	◯ ◯ ◯	◯ ◯ ◯	◯ ◯ ◯

Table 6: Four clusters of shapes

One classic metric is *purity*, which measures the size of the dominant type (class) relative to the entire cluster. That is:

$$Purity(c) = \frac{1}{n_c} \max_t c_t \tag{15}$$

where n_c is the size of cluster c (the total number of members) and c_t is the number of members in c that belong to type t. For cluster (1) in Table 6, the members are equally distributed among three types, each has 3 and the purity is:

$$\begin{aligned} Purity(c_1) &= \frac{1}{9} max(3,3,3) \\ &= \frac{3}{9} \end{aligned}$$

For cluster (4), there is only one type, namely ◯, and its purity is $9/9 = 100\%$. So far, purity appears consistent with our intuitive observation that cluster (4) is pure and cluster (1) is much less so because it is consisted of three equal types.

If we apply purity to clusters (2) and (3) in Table 6, we get:

$$\begin{aligned} Purity(c_2) &= \frac{1}{9} max(2,2,5) \\ &= \frac{5}{9} \\ Purity(c_3) &= \frac{1}{9} max(4,0,5) \\ &= \frac{5}{9} \end{aligned}$$

Both have the same purity score. However, it is clear that cluster (2) has three types (2-2-5) and is more heterogeneous than cluster (3), which only has two types (4-5). Because purity only relies on the dominant type, it ignores the overall distribution and does not differentiate between the two clusters here.

Now that we know Shannon entropy is a property of a probability distribution, it can be applied to the problem here to measure member distribution in the types. By converting the counts (frequencies) into probabilities, we can compute the entropy of each cluster by:

$$H = -\sum_t p_t \log p_t \tag{16}$$

where p_t is the probability of type t and can be estimated by c_t/n_c, i.e. the proportion of cluster c that is in type t. Hence:

$$\begin{aligned} H(c_1) &= -3 \times \frac{3}{9} \log \frac{3}{9} \\ &= 1.585 \\ H(c_2) &= -\frac{2}{9} \log \frac{2}{9} - \frac{2}{9} \log \frac{2}{9} - \frac{5}{9} \log \frac{5}{9} \\ &= 1.436 \\ H(c_3) &= -\frac{4}{9} \log \frac{4}{9} - \frac{5}{9} \log \frac{5}{9} \\ &= 0.991 \\ H(c_4) &= -\frac{9}{9} \log \frac{9}{9} \\ &= 0 \end{aligned}$$

From clusters (1) through (4), the entropy decreases. Cluster (1) has an even distribution among the three types and the most heterogeneous with greatest entropy (disorder). Cluster (4), on the other hand, is the ideal and purest with 0 entropy. Furthermore, cluster (3) is not as broadly distributed as cluster (2) and has less entropy. In terms of the examples in Table 6, Shannon entropy measures the overall distribution better than purity.

Limits and Other Measures

Despite its broad use, there are assumptions that define the boundary of the classic information theory, beyond which its application requires careful examination of the domain context. For one, the basic assumption that information entails the reduction of entropy does not always hold true. There are abundant examples where having more information may lead to confusion and increased uncertainty. Sometimes, less is more.

The original purpose of Shannon's theory, as noted in his master piece, was for engineering communication systems where the "meaning of information was considered irrelevant". It is about technical problems that can be treated as independent the semantic content of messages.[18]

[18] ANATOL RAPOPORT. What is information? *ETC: A Review of General Semantics*, 10(4):247–260, 1953

Shannon entropy is best used in applications where uncertainty or the amount of (missing) information is indeed a *property* of a distribution, no less and no more. In domains where receiving information may lead to revised probabilities (change of beliefs) but not necessarily reduction of uncertainty, Shannon entropy as a function solely based on *a* probability distribution becomes insufficient.

To clarify on this point, let us look at a real world example –

Consider the 2016 presidential election of the U.S.[19] Poll data and analytics had led many people to believe, up until the election night, that Hillary Clinton would win a landslide against Donald Trump. A common prediction would put 90% for Clinton and 10% for Trump[20], resulting in an entropy of:

[19] While this is an example from politics, the analysis here is apolitical.

[20] The data are hypothetical, rough estimates of the election day sentiment. For a simplified discussion, we only focus on candidates of the two major parties rather than the actual four.

$$H_{2016} = -0.9 \log 0.9 - 0.1 \log 0.1 \quad (17)$$

$$= 0.469 \quad (18)$$

which is much less than the entropy of a 50-50 case ($H = 1$). So with this belief, there was very little uncertainty (doubt) about the outcome given one was highly favored (likely) to win.

In terms of Shannon entropy, there was *little* missing information in that belief (probability distribution). This would make sense if the likely candidate eventually won.

However, when the election turned out in the opposite direction, many would immediately ask questions like "what went wrong" and "what did the polls miss?" Clearly there was in fact lots of missing information in the polls. And the surprising result does require tremendous amount of explanation (with additional information).

But the Shannon entropy does not "care" as it only models the estimated distribution, regardless of the outcome or what turns out to be correct.

We can draw from the above example that different amounts of information are needed to explain different (and sometimes opposite) outcomes. The amount of information should depend not only on the uncertainty of inferences but also on the ultimate outcome (the correct inference or the true probability distribution).

We reason that while uncertainty is a property of a specified probability distribution, the amount of information required to explain the outcome and more generally to explain a probability distribution change is more complex than a linear function of uncertainty and should depend on both distributions, i.e. a belief and a changed belief. We will discuss a couple of models that account for this relative entropy change.

Relative Entropy

Relative Entropy, or KL Divergence, models the amount of information it loses when one uses certain *estimates* to approximate a true probability distribution.[21]

[21] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951

Given a probability distribution P and an estimated distribution Q of the same variable, the relative entropy, or information loss due to the estimation, can be computed by:

$$KL(P||Q) = (-\sum_i p_i \log q_i) - (-\sum_i p_i \log p_i) \quad (19)$$

$$= \sum_i p_i \log \frac{p_i}{q_i} \quad (20)$$

where p_i and q_i are, respectively, the (true) probability and estimate of the i^{th} inferences. As shown in the formula, the KL divergence measures the difference in the number of arrangements between Q and P, weighted by the *true* probabilities in P. This is the amount of information (in *bits* with log base 2) lost in the estimates.

We apply this to the 2016 election example. Let Q denote the estimates according to polls: $q_{clinton} = 0.9$ and $q_{clinton} = 0.1$. P denotes the true probabilities (actual result).

Suppose the election went as predicted, i.e. with a Clinton win $p_{clinton} = 1$. KL can be computed by:

$$KL(P_{clinton}||Q) = 1 \times \log \frac{1}{0.9} + 0 \times \log \frac{0}{0.1}$$
$$= 0.152$$

Now that the election actually turned out the opposite, i.e. with a Trump win $p_{trump} = 1$, KL is in fact:

$$\begin{aligned} KL(P_{trump}||Q) &= 0 \times \log \frac{0}{0.9} + 1 \times \log \frac{1}{0.1} \\ &= 3.322 \end{aligned}$$

which indicates a much greater (about 20 times) information loss, compared to the *true* distribution (outcome), in the estimated chances based on poll data. The relative entropy does appear to capture the *surprise* factor.

KL divergence has a few useful properties. KL is non-negative for any distributions. It increases with increasing differences in the P and Q values and can approach infinity with an infinite $\frac{p_i}{q_i}$ ratio, e.g. with a $q_i \to 0$ – that is, when something is mistakenly believed to be impossible when it is in fact possible.

KL divergence cannot, however, be referred to as a distance measure or metric. To qualify as a metric distance, a function has to satisfy the following conditions: 1) it is non-negative, 2) it returns zero for two identical sets of values, 3) it is symmetric, and 4) it satisfies the triangular inequality.

KL does meet the first two requirements on non-negativity and identify of indiscernibles – KL is zero, $KL(P||Q) = KL(Q||P) = 0$, only when the two distributions are identical – but does not satisfy the triangular inequality.

For any distance function d, the triangle inequality requires that, given three points (sets of values or probability distributions) A, B, and C, the sum of two distances (sides) is no less than the third. This can be illustrated in Figure 35, where any two distances (sides) $d(A,B) + d(B,C) \geq d(A,C)$.

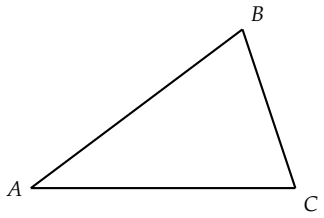


Figure 35: Triangle Inequality. The sum of two distances (sides) is never less than the third, i.e. $d(A,B) + d(B,C) \geq d(A,C)$.

KL divergence does not satisfy the triangle inequality. That is, it does allow $KL(A||B) + KL(B||C) < KL(A||C)$ to happen under certain conditions. In addition, KL is not symmetric and the divergence from Q to P is not the same as that from P to Q, i.e. $KL(P||Q) \neq KL(Q||P)$, where P and Q are not identical.

KL divergence extends the classic Shannon entropy and offers an alternative to measuring information as a linear reduction of entropy. However, the fact that it is not symmetric and not a metric disqualifies it from certain applications where a geometric distance is fitting.

One important consequence of not being a metric is that one cannot add up the amounts of divergence in the *course* of probability changes. It is meaningless to add $KL(A||B)$ and $KL(B||C)$, which has little to do with $KL(A||C)$. In other words, one cannot discuss the sum of information in KL terms. In addition, as discussed earlier, KL divergence can approach infinity and there is no upper bound.

KL divergence has enabled important development in statistical analysis. Mutual information, for example, is the KL relative entropy

between the joint distribution of two random variables and the product of their marginal distributions.

In information retrieval and text mining, the classic IDF (inverse document frequency) term weighting scheme in can be regarded as a function derived from KL divergence. Later in the chapter, we will discuss related problems due to undesirable properties of KL and a remedy developed and tested recently.

Jensen-Shannon Divergence

One approach to make KL divergence symmetric is to compute the sum of its scores in both ways. For example:

$$J(P,Q) \quad = \quad KL(P||Q) + KL(Q||P) \tag{21}$$

However, this does not resolve the problem of no upper bound; nor does it satisfy other desirable properties such as triangle inequality. Another alternative is the Jensen-Shannon divergence, which measures the average of information losses using the mixture of two probability distributions:

$$JS(P||Q) \quad = \quad \frac{KL(P||M) + KL(Q||M)}{2} \tag{22}$$

where M is the mixture probability, i.e. $M = \frac{P+Q}{2}$. It is easy to show that $JS(P||Q) = JS(Q||P)$.

A metric can be derived by taking the square root of Jesen-Shannon divergence:

$$JSR(P||Q) \quad = \quad \sqrt{\frac{KL(P||M) + KL(Q||M)}{2}} \tag{23}$$

Using the discussed 2016 election, for example, JSR can be computed for a Trump win, i.e. $P_{trump} = 1$ and $P_{clinton} = 0$. Given polling data shows chances before election night, $Q_{trump} = 0.1$ and $Q_{clinton} = 0.9$, the mixture probability distribution M is:

$$\begin{aligned} M_{trump} &= \frac{1+0.1}{2} \\ &= 0.55 \\ M_{clinton} &= \frac{0+0.9}{2} \\ &= 0.45 \end{aligned}$$

The Jesen-Shannon divergence is:

$$\begin{aligned} JS(P||Q) &= \frac{KL(P||M) + KL(Q||M)}{2} \\ &= \frac{1 \times \log \frac{1}{0.55} + 0 \times \log \frac{0}{0.45} + 0.1 \times \log \frac{0.1}{0.55} + 0.9 \times \log \frac{0.9}{0.45}}{2} \\ &= 0.758 \end{aligned}$$

Then JSR is the square root of JS:

$$\begin{aligned} JSR(P||Q) &= \sqrt{0.758} \\ &= 0.871 \end{aligned}$$

In the evening of the election, data and the outcome of some states led to an increased likelihood of a Trump win. Suppose the changed probability estimates became Q' where $Q'_{trump} = 0.9$ and $Q'_{clinton} = 0.1$, the opposite of the earlier estimates Q.

We can compute JSR between Q and Q':

$$\begin{aligned} JSR(Q'||Q) &= \frac{0.9 \times \log \frac{0.9}{0.5} + 0.1 \times \log \frac{0.1}{0.5} + 0.1 \times \log \frac{0.1}{0.5} + 0.9 \times \log \frac{0.9}{0.5}}{2} \\ &= 0.729 \end{aligned}$$

We can also compute JSR between Q' and P:

$$\begin{aligned} JS(P||Q') &= \frac{1 \times \log \frac{1}{0.95} + 0 \times \log \frac{0}{0.05} + 0.9 \times \log \frac{0.9}{0.95} + 0.1 \times \log \frac{0.1}{0.05}}{2} \\ &= 0.228 \end{aligned}$$

Clearly, the sum of $JS(P||Q')$ and $JSR(Q'||Q)$ is $0.228 + 0.729 = 0.957$, which is greater than $JS(P||Q) = 0.871$. This does not violate the triangular inequality.

It has been shown that the square root of Jesen-Shannon divergence (JSR) satisfies triangular inequality.[22] And because JS measures the average information loss to the mixture probability, the amount is the same for $JS(P||Q)$ and $JS(Q||P)$ – that is, JS is a symmetric function and so is its square root JSR.

[22] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, September 2006

JSR does satisfy all the requirements for a distance function and can be used where related metric properties are desired. Whereas JS divergence can still be regarded as measuring the amount of information in bits (at least with log base 2), it is not clear what its square root (JSR) actually means.

IDF and KL Divergence

Before we get to an alternative to the divergence measures, we shall discuss an important connection between the classic IDF (inverse doc-

ument frequency) method for text processing and KL divergence. The theoretical connection will enable us to see why certain treatments based on IDF may or may not be a good idea.

As we discussed in chapter Text and Human Language, one important step in text processing is to determine how informative each term is and how much weight should be given. To do this, let's look at probabilities associated with a term.

Suppose we randomly draw a document from a collection of N documents, the likelihood of observing term t in the document can be estimated by:

$$Q(t) = \frac{n_t}{N} \tag{24}$$

where n_t is the number of documents containing term t. Its complementary probability, i.e. the probability that term t does not occur, can be computed by $\bar{Q}(t) = \frac{N-n_t}{N}$.

Now with a specific document where term t actually appears, its probability is $P(t) = 1$ and the complementary $\bar{P}(t) = 0$. We can compute the amount of information loss due to the probability estimate Q using KL divergence:

$$\begin{aligned} KL(P||Q) &= P(t)\log\frac{P(t)}{Q(t)} + \bar{P}(t)\log\frac{\bar{P}(t)}{\bar{Q}(t)} && (25)\\ &= 1\log\frac{1}{\frac{n_t}{N}} + 0\log\frac{0}{\frac{N-n_t}{N}} && (26)\\ &= \log\frac{N}{n_t} && (27) \end{aligned}$$

which is exactly the IDF (inverse document frequency) term weighting scheme, where n_t is document frequency (DF).

Here we see that the classic IDF can be regarded as an application of KL divergence in the text processing context. In related applications, it is common to compute the sum of IDF weights for processes such as document ranking (in information retrieval). However, this might not be the proper treatment for a non-metric.

As we observed earlier, KL divergence does not have an upper bound and can reach infinity. In particular, IDF as an information amount of a term can be extremely large when the term is very rare. When term IDF weights are combined to determine ranking, for example, the rare term will dominate the ranking score and the weights of other terms will no longer matter.

Least Information Theory (LIT)

The Least Information Theory (LIT), a very recent development, has attempted to resolve the problems associated with divergence-related information measures.[23] In particular, its major objective is to create an information quantity that: 1) is a metric, 2) can be interpreted properly such as in terms of bits, and 3) has an upper bound and does not produce infinite values even with extreme probabilities.

[23] Weimao Ke. Information-theoretic term weighting schemes for document clustering and classification. *International Journal on Digital Libraries*, 16(2):145–159, Jun 2015

Based on Shannon entropy, LIT does not assume a linear relation between information and the reduction of entropy. Rather, we acknowledge that receiving information can lead to either an increase or decrease of entropy, and measures the amount of information as the sum of microscopic, absolute entropy changes. Given two probability distributions P and Q on m discrete inferences of the same variable, their Shannon entropies can be computed by:

$$H(P) = -\sum_{i=1}^{m} p_i \log p_i \tag{28}$$

$$H(Q) = -\sum_{i=1}^{m} q_i \log q_i \tag{29}$$

Let dH_i be the amount of entropy change due to a very small change in the probability of the i^{th} inference dp_i. We can compute dH_i based on Shannon entropy:

$$dH_i = -\log p_i dp_i \tag{30}$$

This microscopic entropy represents the change of the weighted (p_i) number of arrangements ($-\log p_i$ or $\log \frac{1}{p_i}$). It is the change in the number of configurations due to a change weight (probability). By integrating (aggregating) the microscopic changes, we can compute the amount of entropy due to a greater, macroscopic probability change from P to Q in the i^{th} inference:

$$I_i(P \to Q) = \int_{p_i}^{q_i} -\log p dp \tag{31}$$

$$= \left| p_i(1 - \ln p_i) - q_i(1 - \ln q_i) \right| \tag{32}$$

The overall entropy change is the sum of entropy changes of all inferences:

$$I(P \to Q) = \sum_{i=1}^{m} \left| p_i(1 - \ln p_i) - q_i(1 - \ln q_i) \right| \tag{33}$$

where m is the total number of mutually exclusive inferences, i.e. the number of discrete values of the probability distribution. Note that in the final formula, the logarithm is always with a natural base (ln). This is the result of the integral of any log function.

The Least Information measure (LIT) in Equation 33 can be interpreted as the minimum amount of information, in Shannon entropic terms, *necessary* for the probability distribution shift, or a change of belief. In this model, information does not always reduce entropy – in fact, it can lead to entropy change in any direction. In addition, it does not consider the removal of information, which can be regarded as new information that refutes prior knowledge.

Figure 36 compares least information (LIT) with Shannon entropy and KL divergence, using a binary example (of two mutually exclusive inferences). The figure assumes the 1^{st} inference to be the ultimate outcome – that is, it starts with a probability distribution P (with some p_1 and p_2) and ends with changed distribution Q, where $q_1 = 1$ and $q_2 = 0$. For example, the reading of $(0.5, 1)$ on the LIT curve indicates the amount of least information is 1 that changes the probability distribution from $P(0.5, 0.5)$ to $Q(1, 0)$.

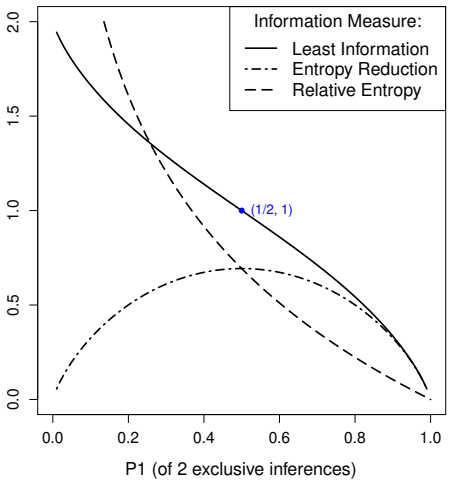


Figure 36: Least Information vs. Shannon Entropy vs. KL Divergence. This shows the amount of information by reducing two mutually exclusive inferences to certainty. The X axis is p_1, the probability of the 1^{st} inferences. The Y axis is the amount of information (I) in terms of each information measure. The 1^{st} inference is assumed to be the ultimate outcome ($q_1 = 1$) in the figure.

The symmetry of Shannon entropy indicates that the amount is the same regardless of the outcome. KL divergence can approach infinity with $p_1 \to 0$. Least Information (LIT), however, is limited to the range of $[0, 2]$ in the binary case.

Further examination of Equation 33 reveals that the Least Information Theory (LIT) possesses several important properties:

1. Non-negativity: $I(P, Q) \geq 0$, the amount of least information is no less than zero.

2. Identity of indiscernibles: $I(P, Q) = 0$, if and only if P and Q are

identical, i.e. no change at all in the probability distribution.

3. Symmetry: $I(P,Q) = I(Q,P)$, the amount of *least information* required for a probability change from P to Q is the same as that from Q to P.

4. Triangular inequality: $I(P,Q) + I(Q,R) \geq I(P,R)$, among three probability distributions, the sum of two least information amount is no less than the third.

5. Addition of continuous change: $I(P,Q) + I(Q,R) = I(P,R)$, when $P \to Q$ and $Q \to R$ are in the same direction of probability changes (i.e. for each individual inference, they either both increase or both decrease its probability). In other words, amounts of *least information* for small, continuous probability changes in the same directions add linearly to the amount responsible for the overall change.

6. Unit Information: In the special case when there are two equally possible inferences, the amount of *least information* needed to explain/interpret an outcome (certainty) is one: $I(p_1 = p_2 = \frac{1}{2} \to q_1 = 1, q_2 = 0) = 1$ (see data point $(\frac{1}{2}, 1)$ in Figure 36).

7. In the special case of reducing uncertain inferences to certainty (the ultimate case):

   - With equally likely inferences, when there are more choices, the least information needed to explain an outcome is larger.
   - The less likely the outcome, the larger the amount of *least information* needed to explain it.

The first four properties listed above mean that the Least Information (LIT) function is a distance metric. The 5^{th} property can be likened to a function such as euclidean distance, where small distances in the same direction can add up to the overall distance. This is illustrated in Figure 37, where $a + b = c$. This is the only situation in which they are equal with the triangle inequality.

We can test the Least Information Theory (LIT) on the example of 2016 election. Given the probability change from election day estimates of $Q_{trump} = 0.1$ and $Q_{clinton} = 0.9$ to adjusted estimates of $Q'_{trump} = 0.9$ and $Q'_{clinton} = 0.1$ to the final election outcome of $P_{trump} = 1$ and $Q_{clinton} = 0$, one can compute the the amounts of LIT on the two stages:

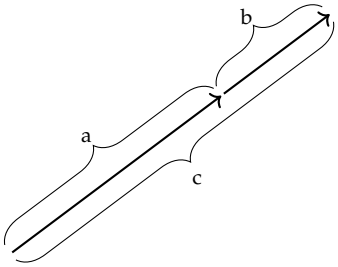


Figure 37: Sum of distances in the same direction, $a + b = c$

$$\begin{aligned} I(Q \to Q') &= \big|0.9(1-\ln 0.9) - 0.1(1-\ln 0.1)\big| \\ &\quad + \big|0.1(1-\ln 0.1) - 0.9(1-\ln 0.9)\big| \\ &= 1.329 \\ I(Q' \to P) &= \big|1(1-\ln 1) - 0.9(1-\ln 0.9)\big| \\ &\quad + \big|0(1-\ln 0) - 0.1(1-\ln 0.1)\big| \\ &= 0.335 \end{aligned}$$

The overall LIT from early estimates Q to the final result P is:

$$\begin{aligned} I(Q \to P) &= \big|1(1-\ln 1) - 0.1(1-\ln 0.1)\big| \\ &\quad + \big|0(1-\ln 0) - 0.9(1-\ln 0.9)\big| \\ &= 1.664 \\ &= 1.329 + 0.335 \end{aligned}$$

which is exactly the sum of the two stages' amounts, i.e. $I(Q \to P) = I(Q \to Q') + I(Q' \to P)$. This equality occurs because $Q \to Q'$ and $Q' \to P$ represent probability changes in the same directions – that is, for each and every inference of the variable, they either both increase or both decrease in that probability.

In the context of text processing and information retrieval, the least information measure has produced superior results. Earlier we have shown that the classic IDF (inverse document frequency) term weighting scheme is a derived method from KL divergence. We can derive a similar method based the least information. Again, the likelihood of having term t in a random document from a text collection is $Q(t) = \frac{n_t}{N}$, where n_t is document frequency (DF)of term t and N the total number of documents in the collection. The amount of least information of actually observing term t in a specific document, with $P(t) = 1$ and $\bar{P}(t) = 0$, can be computed by:

$$\begin{aligned} I(t) &= I(Q_t, P_t) \\ &= \big|1(1-\ln 1) - Q_t(1-\ln Q_t)\big| \\ &\quad + \big|0(1-\ln 0) - \bar{Q}_t(1-\ln \bar{Q}_t)\big| \\ &= \underbrace{\big[1 - Q_t(1-\ln Q_t)\big]}_{Q_t \text{ terms}} + \big[\bar{Q}_t(1-\ln \bar{Q}_t)\big] \end{aligned} \tag{34}$$

We may focus on the Q_t terms (the underbraced in Equation 34) and neglect $\bar{Q}_t$ terms. We can define $LIB(t, d)$, or least information binary, to measure the informativeness of term t for document d

when it appears ($t \in d$) and when it does not ($t \notin d$). Given that $Q_t = \frac{n_t}{N}$, we have:

$$LIB(t,d) = \begin{cases} 1 - \frac{n_t}{N}\left(1 - \ln \frac{n_t}{N}\right) & t \in d \\ -\frac{n_t}{N}\left(1 - \ln \frac{n_t}{N}\right) & t \notin d \end{cases} \tag{35}$$

where n_t is the document frequency of term t and N is the total number of documents. The larger the LIB, the more information the term contributes to the document and should be weighted more heavily in the document representation.

LIB is similar in spirit to IDF and its value represents the discriminative power of the term when it appears in a document. Nonetheless, LIB is formulated based on *least information* of query terms whereas IDF quantifies their KL divergence or the loss of information due to estimates from the entire collection.

Research has shown that these methods derived from LIT worked extremely well for text mining. In a range of experiments conducted on several benchmark data sets, LIT-based methods outperformed their classic counterparts such as TF*IDF in text clustering, classification, and information retrieval.[24]

[24] Weimao Ke. Text retrieval based on least information measurement. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '17, pages 125–132, New York, NY, USA, 2017. ACM

Data Preparation

Trust, but verify.

Ronald Reagan

There is no perfect data. In fact, a Kaggle 2017 survey showed that *dirty data* "is the most common problem for workers in the data science realm."[25] About 50% of survey responses cited it as a barrier at work.

[25] Kaggle. The state of data science & machine learning 2017. https://www.kaggle.com/surveys/2017. Accessed: 2019-09-10

Binary Questions

To be, or not to be,
that is the question...

Hamlet, William Shakespeare

A binary question represents the basic decision making: yes or no, true or false, etc. So how do we build a model to answer basic questions of such? In particular, what does a model *learn* from training data to make predictions?

We shall continue to use a simple data set we encountered earlier, namely the instances of shapes on the concept of *standing* vs. *lying*. Suppose we have identified *width* and *height* as relevant attributes to the concept and Table 7 shows the data:

Width	Height	Standing?
4	2	No
5	9	Yes
9	5	No
2	4	Yes
1	8	Yes
5	3	No

Table 7: Shapes data

If the shape instances here can serve as training data, how can an algorithm identify the relation between width and height as a predictor for the concept of standing?

Given this simple problem – simple to the human eye – it may be tempting to hardcode a selection structure such as $height > width$? into an algorithm and make predictions without the training data. This, however, defeats the purpose of data-driven machine learning. While we can easily identify the classification rule with this small data set, it is impossible to do so with massive data and more complex relations.

We should instead design a general strategy by which related rules and relations can be captured from training data and then used in predictions. What potential strategies can we propose? Let's look at a

few proposals.

Lazy Learning

The easiest approach to learning is not to learn at all, or simply memorize all given materials without trying to understand them. We can refer to this as *lazy learning*, which is similar to what some "lazy" students may do in reality.

But in the real world, how does a student perform on a test (examination) if she has not understood anything at all? The solution is to search, whether it is searching one's memory or materials and notes. Without understanding, one will try to find notes and sections *most closely related* to a test question and develop an answer based on the search result.

Likewise, in machine learning, a lazy learning method does not build any *model* at all but simply remembers (stores) training instances. To predict on a test instance, the method searches its memory (training data), finds the *most closely related* instances to the test, and predict the answer based on related instances it has found.

K Nearest Neighbors (kNN)

We normally refer to the most closely related instances as the *nearest neighbors*.[26] A classic lazy learning algorithm is the *k nearest neighbors* (kNN) approach, which makes predictions based on k closest training instances.

[26] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967

But how does the algorithm find out which ones are the nearest? It needs a function to measure the closeness or distance between two instances. An example of such a distance function is the *Euclidean distance*, which the length a straight line connecting the endpoints of two instances u and v in dimensional space:

$$D(u,v) = \sqrt{\sum_{i=1}^{m}(u_i - v_i)^2} \tag{36}$$

where m is the number of dimensions (features) to represent each data instance. Figure 38 shows the shapes data on two dimensions, i.e. *width* and *height*. Suppose a new instance at $(4,7)$, i.e. $width = 4$ and $height = 7$, is to be tested. The dashed lines are the euclidean distances from the test instance to the training instances.

If we set $k = 1$ for the kNN algorithm, we will use only 1 nearest neighbor in prediction. The instance at $(5,9)$, which belongs to the *standing* class, has a distance from the test instance:

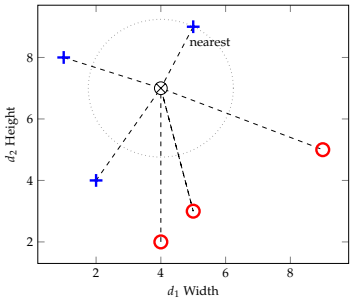


Figure 38: Nearest neighbors $k = 1$, $+$ denotes *standing* (positive) and $\circ$ denotes *lying* (negative). $\otimes$ is the test instance.

$$D[(4,7),(5,9)] = \sqrt{(4-5)^2+(7-9)^2} = \sqrt{5}$$

This turns out to be the nearest in terms of Euclidean distance and the test instance will be predicted *standing* (positive) as well.

If we change $k = 3$, the 3 nearest neighbors will all be positive (+) instances and the prediction for the above test instance will remain positive.

For a different test instance, the k nearest neighbors might have conflicting answers. For binary classification, one strategy is to pick an odd number for k and let the nearest neighbors vote for the final prediction.

Figure 39 shows another test instance at $(3,4)$. With $k = 3$, it turns out that the three nearest neighbors include 1 positive and 2 negatives. Using a majority vote strategy, the test instance will be mistakenly classified into the negative (lying) class.

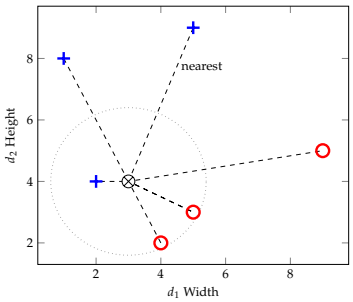


Figure 39: Nearest neighbors $k = 3$, + denotes *standing* (positive) and ○ denotes *lying* (negative). ⊗ is the test instance.

Two observations on the misclassification. First, the selection of k does have an impact on the classification result. This is especially true when training instances are sparse. With $k = 1$, the method relies on one single instance for each test and is hardly robust. A larger k can reduce the likelihood of predicting with atypical neighbors but it introduces noise, e.g. instances from across the class boundary as illustrated in Figure 39. Finding the optimal k may require experimentation with data.

Second, more important, a lazy learner such as kNN does not have a general function (model) to represent a concept of learning. It purely relies on individual training instances for predictions and has *no generalizability*. Its success or failure depends on where the test instance hit and how training data distribute around it. This is analogous to a *lazy* student who memorizes everything without understanding – she will not be able to answer a question for which no similar examples have been provided in the materials.

Other Distance Metrics

Euclidean distance is a special case of a category of distance metrics known as the *Minkowski distance*, which is defined as:

$$MD(u,v) = \left(\sum_{i=1}^{m} |u_i - v_i|^p\right)^{1/p} \tag{37}$$

where $p \geq 1$ and m is the number of dimensions. When $p = 2$, this is the Euclidean distance. When $p = 1$, it becomes the Manhattan distance:

$$MD_1(u,v) = \sum_{i=1}^{m} |u_i - v_i| \tag{38}$$

which is similar to counting the number of blocks in a city grid.

Another metric is similar to (but not exactly) Minkowski distance with $p = 0$:

$$MD_0(u,v) = \sum_{i=1}^{m} (u_i - v_i)^0 \tag{39}$$

$$= \sum_{u_i \neq v_i} 1 \tag{40}$$

which counts the number of dimensions where there is a difference and is often referred to as *edit distance*. This is equivalent to treating every dimension as a binary variable and the overall distance is the amount of change in the bits.

Linear Classifiers

As we observed, a lazy learner such as the k Nearest Neighbors (kNN) does not actually *learn* and build a model (function) that can be generalized. Although kNN has decent performances in many applications, it suffers in domains where, for example, test data appear on a different scale from that of training data.

If we assume the classes can be separated by a *function* (model) with certain parameters, then building the model requires identification of those parameters from training data. Surely there are way too many different functional forms one can choose the model from. We start with the simplest, a linear function.

With the shapes data in Figure 38, there are two variables *width* (v_1 for the x axis) and *height* (v_2 for y axis). y as a linear function of x can be written as $f(x) = a + bx$ or $a + bx - y = 0$, with two parameters a and b for two variables.

To make it extendable, We can rewrite this two-variable linear equation as:

$$w_0 + w_1 v_1 + w_2 v_2 = 0 \tag{41}$$

where the w parameters (instead of a and b) are to be estimated. When there are more variables (dimensions), the function can be extended to:

$$0 = w_0 + w_1 v_1 + w_2 v_2 + ... + w_m v_m \quad (42)$$

$$= w_0 + \sum_{i=1}^{m} w_i v_i \quad (43)$$

where m is the number of variables (dimensions).

With $m = 2$ for the shapes data, how do we estimate the parameters w_0, w_1, and w_2 in Equation 41 to linearly separate the two classes *standing* vs. *lying*? Note that a linear equation in a 2-dimensional space is a straight line. It is a plane (flat surface)) in a 3-dimensional space and a *hyperplane* in a higher-dimensional space.[27] We will discuss several approaches of the linear model.

[27] A hyperplane in more three dimensions is hard to visualize. Mathematically, it shares the same functional form of Equation 43 and is a natural extension of the lower-dimensional representation.

Between Centroids

We start with finding the *centroid* for instances in each class. A centroid is the center of the mass for a group of objects and can be computed as the average of values on each dimension. The j^{th} dimension of the centroid for class c is computed by:

$$\mu_j(c) = \frac{\sum_{i=1}^{n_c} v_{ij}}{n_c} \quad (44)$$

where v_{ij} is the value of the j^{th} dimension of the i^{th} instance in class c. Following this equation, the centroids for the positive (standing) and negative (lying) classes can be obtained:

$$\begin{aligned}
\mu(\oplus) &= (\mu_1(\oplus), \mu_2(\oplus)) \\
&= (\frac{5+2+1}{3}, \frac{9+4+8}{3}) \\
&= (\frac{8}{3}, 7) \\
\mu(\ominus) &= (\frac{4+5+9}{3}, \frac{2+3+5}{3}) \\
&= (6, \frac{10}{3})
\end{aligned}$$

Figure 40 shows the two centroids and a dashed line with an arrowhead connecting them. The arrow represents a vector in the direction $\vec{\mu} = (6 - \frac{8}{3}, \frac{10}{3} - 7) = (\frac{10}{3}, -\frac{11}{3})$, whereas the middle of the dashed line is $\bar{\mu} = (\frac{13}{3}, \frac{31}{6})$.

Now if we can find a linear equation, i.e. another straight line, that: 1) passes through the middle of the dashed line (equal to each centroid), and 2) is perpendicular to the dashed line, then any point on the linear equation (line) should have an equal distance from both

centroid. In this case, the line can serve as the *boundary* between the two classes.

Now, identifying the linear equation turns out to be very straightforward. For any point on the line (v_1, v_2), it can be seen as a vector originated from the midle point $\bar{\mu} = (\frac{13}{3}, -\frac{31}{6})$:

$$\vec{v} = (v_1 - \frac{13}{3}, v_2 - \frac{31}{6})$$

For the vector to be perpendicular to the dashed vector $\vec{\mu}$, their dot product should be 0. That is:

$$\begin{aligned} \vec{v} \cdot \vec{\mu} &= \frac{10}{3}(v_1 - \frac{13}{3}) - \frac{11}{3}(v_2 - \frac{31}{6}) \\ &= 0 \end{aligned}$$

which is:

$$-1 - \frac{20}{27}v_1 + \frac{22}{27}v_2 = 0$$

or:

$$v_2 = 1.227 + 0.909v_1$$

Comparing this to Equation 41, we have identified the parameters $w_0 = -1$, $w_1 = -\frac{20}{27}$, and $w_2 = \frac{22}{27}$ for the linear classification function. This is the solid straight line separating the two classes (centroids) in Figure 40.

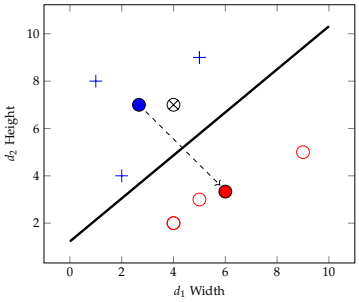


Figure 40: Linear classification with centroids, + denotes *standing* (positive) and ∘ denotes *lying* (negative). • represents a centroid of the respective class. ⊗ is a test instance.

With the identified linear function $f(v) = 1 + \frac{20}{27}v_1 - \frac{22}{27}v_2$, one can plug in values of a test instance and make a prediction. If the result is positive, it will predict positive (standing); otherwise, it will predict negative (lying). For example, given a test instance at $(4, 7)$ ($\otimes$ on Figure 40), we can put the values in the linear function and get:

$$\begin{aligned} f(4,7) &= -1 - \frac{20}{27} \times 4 + \frac{22}{27} \times 7 \\ &= \frac{47}{27} \\ &> 0 \end{aligned}$$

The final result is positive – the same positivity with the positive class centroid – and hence the test instance $\otimes$ will be classified into the *standing* class.

Theoretically, we understand that the concept of *standing* is on the relation of $height - width > 0$ or, in terms of the linear function, $0 -$

$v_1 + v_2 > 0$. The identified equation is a rough approximation of this theoretical function. The model assumption about a linear separation holds true in this example. But how precise we can establish the exact linear function depends largely on the training data. The data distribution influences the identification of related parameters.

In particular, the centroid approach is greatly impacted by the density of the training set and where most data instances are. For example, having more data instances with smaller width and height in the negative class change its centroid, which will, in turn, lead to a very different linear function, illustrated in Figure 41 (compare to Figure 40).

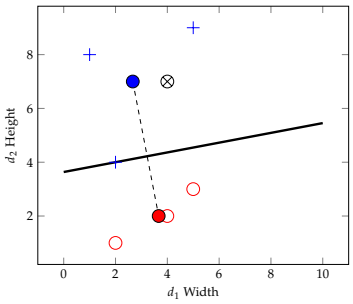


Figure 41: Linear classification with centroids, with smaller values in the negative class

The centroid-based approach to linear classification is greatly influenced by all data points in the training set and how they are distributed. The basic assumption underlying the above modeling is that the two classes are of the same size roughly and the "border" lies halfway between their centers. This is often untrue and its performance suffers.

Support Vector Machines

Imagine settling the border between two countries, for example, the United States and Canada. It does not matter where the centers of the two countries are. Even if the two countries are of the same size, the border is rarely halfway between the centers.

Rather, it is much more relevant to find out where cities, landmarks, and populations are *on the border*. For U.S. and Canada, cities such as Seattle vs. Detroit vs. Vancouver, Detroit vs. Toronto help define their border.

For classification, it matters not where the centroids are but where the *borderline* data are. The Support Vector Machines, or SVM, is a method focused on the border populations and draws the "border" based on the so-called *support vectors*, data points that are located on the boundary and together form the border.

We start by connecting a number of data points within the same class to construct a linear function, i.e. a line on a 2-dimensional space, a plane for 3 dimensions, and a hyperplane for higher dimensions. On the 2-dimensional shapes data, this means using any two points in each class to form a straight line that, when used as the linear classification function, satisfies two conditions: 1) all data points in the same class fall on *one size* of the linear function, and 2) most if not all data points of the other class reside on the other side of the line. In other words, if a line is drawn in the positive (standing class), then 1) data in it will test all positive (or all negative), and 2) data in the other class will test the opposite, negative (or conversely

positive).

Figure 42 illustrates the idea with the shapes data. [28] Pairwise lines drawn in the figure all satisfy the 1^{st} condition and collectively form a *convex hull* for each class. However, not all lines meet the 2^{nd} condition. In fact, only the lines marked 1, 2, and 3 meet both conditions. They are candidate *support vectors*.

For each of the three candidate lines, we can draw a line perpendicular to it from any point in the other class. From this process, we can find out the margin between that candidate support vector line and the other class (convex hull), which is where the boundary can be defined. For example, in Figure 42, the two dotted lines show the margin between line #3 and the negative class.

Yet a greater margin – in fact the greatest margin of all – can be found between line #1 and point $(2, 4)$ in the positive class, as the two dotted lines in Figure 43 show. The greatest margin will be chosen as the decision boundary, which is defined by a few data points known as the *support vectors*. As shown as circled data points in Figure 43, support vectors are those that determine the maximum margin between the two *convex hulls* of the two classes.

With this, it is now easy to identify the linear equation in the margin to separate the two classes. In Figure 43, for example, the solid line at $v_2 = 1.1 + 0.667v_1$ can be used as the linear classifier. One can certainly use the linear functions on the support vectors for classification as well, e.g. to predict one class vs. another if a test instance is within the support vector boundary or to estimate a probability if it is on the margin.

Note that with the shapes data example, we only have to deal with two dimensions and any linear equation is a straight line. With a higher dimensionality, the process remains the same mathematically but a linear function becomes a plane or hyperplane. On three dimensions, for example, three data points should be connected to construct a linear equation (plane) to identify the convex hulls and support vectors. Chapter Matrix has details on the mathematics for SVM.

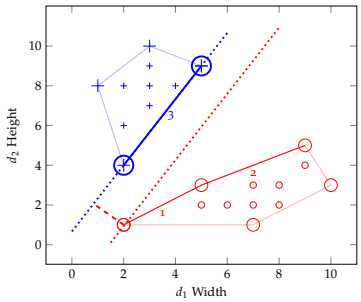


Figure 42: Linear classification with SVM, one candidate margin

[28] Please note additional data points than previously discussed in the illustration.

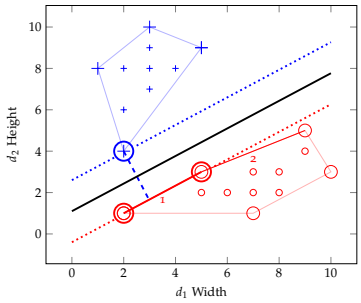


Figure 43: Linear classification with SVM, support vectors and maximum margin

Perceptron: Toward a Neural Network

The linear function $f(v) = w_0 + w_1v_1 + w_2v_2$ can be visualized as a *perceptron* in Figure 44, where the node at $\sum$ computes the weighted sum of input variables. Obviously, this can be extended to include any number of variables m for which $f(v) = \sum_{i=1}^{m} w_iv_i$.

A perceptron is a simple, one-layer neural network – besides the input variables, there is only one layer of nodes (only one $\sum$ node) in the model, as shown in Figure 44. The final output of the perceptron

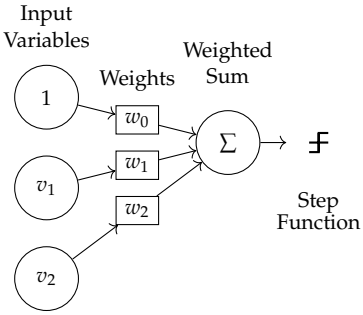


Figure 44: Linear classification with a Perceptron, where the output is a step function based on the weighted sum of input variables

is binary 0 or 1 by using a step function, which simply converts values higher than a threshold to 1 and those lower to 0. For example, one can use 0 as the threshold for a step function $f(s)$:

$$f(s) = \begin{cases} 1 & \text{if s>o} \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

which is visualized in Figure 45.

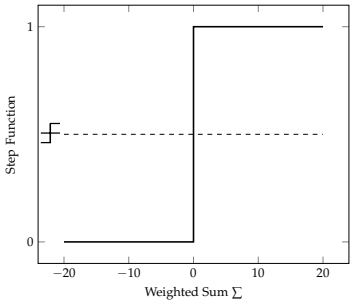


Figure 45: Example of a step function. Output is 0 for any value below zero, and is 1 if above zero.

Before training, the weights are unknown and should be initialized with certain values, e.g. all 0. For each training instance, its values v_1 and v_2 will be fed into the input layer and multiplied with their weights w_1 and w_2 respectively.

Note that in Figure 44, the input with weight w_0 is always 1 for every training instance. This is referred to as the *bias*, which adds the constant w_0 to the linear function. Once every input is multiplied by its weight, the output computes the sum of the weighted values.

For the shapes data, we can decide that the model will predict positive (standing) if the output is greater than 0; negative (lying) if otherwise. During training, if a correct prediction is made, the model will have no need to adjust its weights (for now) and will simply move to the next training instance.

If a misclassification occurs, however, the model will have to adjust its weights so it can potentially make more correct predictions in the future. When the model predicts negative on an actual positive instance, the weights should be adjusted so the overall output (sum) will move toward the positive side; and vice versa.

For example, for a specific shape instance with v_1 (width) and v_2 (height) values, which is $(1, v_1, v_2)$ with the bias input included. Suppose $v_2 > v_1$ and this instance actually belongs to the positive (standing) class. Now if the model finds out $w_0 + w_1 v_1 + w_2 v_2 < 0$ and predicts negative (lying). We will add the input values to their current weights:

$$\begin{aligned} w_0' &\leftarrow w_0 + 1 \\ w_1' &\leftarrow w_1 + v_1 \\ w_2' &\leftarrow w_2 + v_2 \end{aligned}$$

where w' is the updated weight. Because $v_2 > v_1$, the weight for height w_2 will increase more than the weight for width w_1.

Conversely, for a training instance with $v_1 > v_2$ that actually belongs to the negative (lying) class, if it is misclassified as positive, then the current weights will be reduced by the input values:

$$\begin{aligned} w_0' &\leftarrow w_0 - 1 \\ w_1' &\leftarrow w_1 - v_1 \\ w_2' &\leftarrow w_2 - v_2 \end{aligned}$$

Now that $v_1 > v_2$, the weight for width w_1 will be reduced more than w_2. The training continues for a number of iterations or until the weights stabilize. With more training instances and misclassifications, the weights will be adjusted further, leading to relatively a greater (positive) value of w_2 and a smaller (negative) value of w_1.

Theoretically, we know that the classification function for *standing* vs. *lying* should be in the form of $f(v) = 0 - v_1 + v_2$, with w_1 and w_2 having opposite signs. Overall, the adjustment of weights discussed above is moving in the correct direction. As for the *bias*, it will simply be adjusted back and forth with positive and negative misclassifications. For the shapes data, w_0 will be more or less canceled out and its absolute value will become much smaller than those of w_1 and w_2.

The *perception* model is learning by *mistakes*, as it updates the weights only when there is misclassification. For classes that are linearly separable, the updating strategy is guaranteed to converge on a set of weights.

Non-Linear Models

Many machine learning problems in the real world cannot be solved by a linear equation. Consider the car evaluation data in Table 8.

In the data, the concept is about whether a car is *acceptable* or *unacceptable* given its *buying price* and *maintenance cost*.[29] The price (cost) variables are on an ordinal scale: low, medium, high, and very high, which we convert into an equal-interval numeric scale from 1 (low) to 4 (very high).

[29] The tiny data set was inspired by the UCI's Car Evaluation collection. However, this simplified version is intended for demonstration only and does not represent the actual data in the UCI repository.

While it is possible to include other variables such as the number of doors and safety features, we will focus on the cost factors in the discussion here. Figure 46 plots *maintenance cost* vs. *buying price*, with the $+$ indicating an *acceptable* car and $\circ$ unacceptable. The two classes appear not separable linearly on the two-dimensional space – that is, there is no straight line that can be drawn to separate the two.

Again, if one can introduce other predictor variables and the classes might be separable linearly on a higher-dimensional space. But even with the only two variables, namely *maintenance cost* and *buying price*, the plot seems to suggest that a car will be evaluated *unacceptable* with either a high buying price or a high maintenance cost.

Buying Price	Maintenance Cost	Evaluation
2	2	Acceptable
2	1	Acceptable
1	2	Acceptable
1	1	Acceptable
4	4	Unacceptable
4	3	Unacceptable
4	2	Unacceptable
4	1	Unacceptable
3	4	Unacceptable
3	3	Unacceptable
3	2	Unacceptable
3	1	Unacceptable
2	4	Unacceptable
2	3	Unacceptable
1	4	Unacceptable
1	3	Unacceptable

Table 8: Car evaluation data. Buying price and maintenance cost are on a scale of 1 to 4, with 1 meaning the lowest price (cost) and 4 the highest.

Perhaps one can draw two straight lines, one to separate the classes on the price dimension and the other on the maintenance cost. However, this is no longer one linear equation of the two variables that can fit in with a linear model. Alternatively, we can still draw a curve (a non-linear function) to separate them. For example, as shown as the dashed line in Figure 46, a circular function such as $(v_1 - 1)^2 + (v_2 - 1)^2 = 3$ can set the boundary between the two classes. The question is: how can we identify such as a non-linear function from the training data?

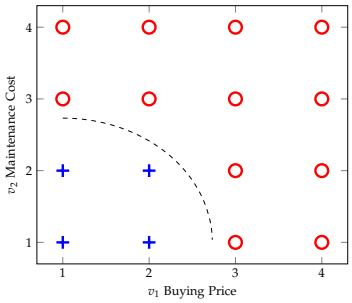


Figure 46: Car evaluation data. + is the positive (acceptable) class and ○ is the negative (unacceptable).

Kernel SVM

We have discussed the *support vector machines*, which, in its original form, is a linear model. How can we apply SVM to non-linear data? Now instead of finding out a non-linear function, we can think about how the data can be transformed in such a way that they are linearly separable.

In particular, we can map the data onto a higher dimensional space with richer features, where a linear classifier is applicable. Now that we do not have additional variables, how can we create a higher-dimensional space? The basic idea is to find non-linear combinations of existing variables, via a so-called *kernel* function. We will have an in-depth discussion of *kernel* functions in Chapter Matrix. But in the nutshell, a *kernel* function can be computed by a dot product with expanded features and makes it extremely efficient to identify the decision hyperplane in the higher dimensional space.

Among families of kernel functions, a polynomial function is often used. For example, we may consider a quadratic transformation with the car evaluation data. Given the two existing variables, any data instance with $v = (v_1, v_2)$ (a two-dimensional vector) can be expanded onto a vector with more dimensions:

$$\phi(v) = (v_1^2, v_2^2, \sqrt{2}v_1v_2) \tag{46}$$

Note that the quadratic expansion should result in 5 dimensions including the original dimensions of v_1 and v_2. We focus on the 3 additional features and thus limit the number of dimensions to three, which can be visualized.

With the v_1^2 and v_2^2 transformation, data in the two classes already appear to be linearly separable. As shown in Figure 47, the solid line at $v_1 + v_2 - 9 = 0$ can separate the data we have so far. However, this does not necessarily represent the general idea of car evaluation and new data may violate the rule (cross the decision boundary).

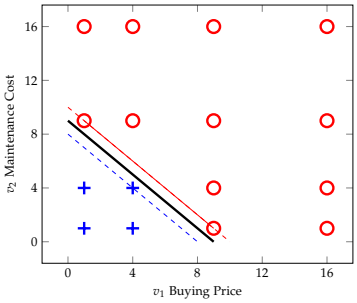
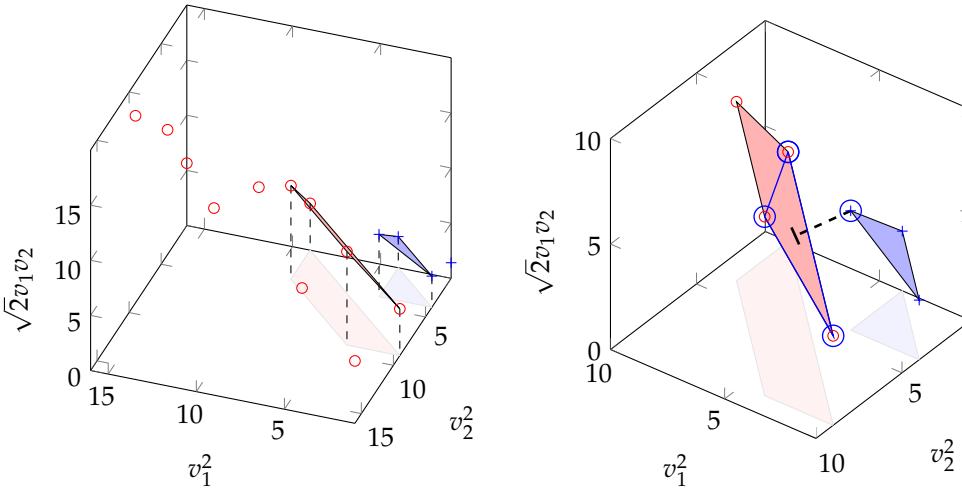


Figure 47: Car evaluation data, with transformed v_1^2 and v_2^2.

Whether a car is acceptable or not may depend on some form of interaction between the buying price and maintenance cost. With the quadratic expansion, the additional dimension of $\sqrt{2}v_1v_2$ can be seen as an interaction variable in the form of multiplication.

When all three expanded features are considered, the data are mapped to a three-dimensional space shown in Figure 48, where they may be separated linearly. Of all linear equations (planes) formed by any three points in each class, we have identified two planes closest to their opposite classes.



(a) All data points

(b) Zoom in on support vectors

Figure 48: Car evaluation data, mapped to higher dimensions with v_1^2, v_2^2, and $\sqrt{2}v_1v_2$. The two planes are closest to the opposite class. The projected shadows are for reference only. (a) shows all data points in the three dimensional space, and (b) is a closer view of the two planes with support vectors. Dashed line is the gap (margin) between the two classes. Circled data points are support vectors.

Figure 48 (a) shows all data points in the expanded dimensional space with quadratic mapping. The two planes formed by data

points are the closest to the opposite class, which are candidate support vectors. Shadows of the planes are projected to the "ground" (i.e. v_2^2 vs. v_1^2 dimensions) for easier reading of related data points.

Figure 48 (b) zooms in on the planes formed by candidate support vectors in the three-dimensional space, where a decision boundary can be identified to maximize the margin between the two classes (convex hulls). The dashed line connects the closest points of data in the two classes (convex hulls), by drawing a line from the circled positive point perpendicular to the plane formed by three circled negative points. The four circled points (one positive and three negative) are the identified support vectors, based on which the margin is established and classification can be performed.

Besides polynomial kernels, other popular kernel functions for SVM include radial basis and string kernels. A string kernel, for example, operates directly on symbol sequences without vectorized features and is useful in applications such as text classification and gene sequence analysis.

Multi-Layer Neural Networks

We know the single perceptron can only be applied to linearly separable data. The car evaluation data, as shown in Figure 46, cannot be separated by one linear function. However, with a step function, one can transform the data into something linearly separable. Specifically, for a value s on the v_1 or v_2 dimension, we use the following step function:

$$f(s) = \begin{cases} 1 & \text{if s>2.5} \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

So it returns 1 for a value above 2.5, and 0 for a value below. Applying the step function to both v_1 (buying price) and v_2 (maintenance cost), we have transformed data in Figure 49. All positive (acceptable) data have been mapped to the $(0,0)$ point whereas the negative (unacceptable) ones are at $(0,1)$, $(1,1)$, and $(1,0)$. As the solid line in Figure 49 shows, the transformed data are now linearly separable.

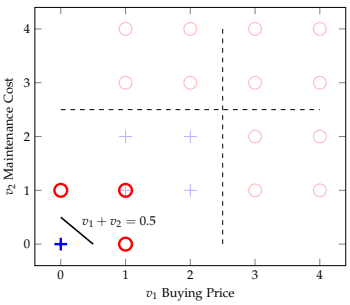


Figure 49: Car evaluation data, transformed by a step function. A step function with a threshold 2.5 converts the original data (+ and ∘ in faded colors) to those with 0 and 1 values (+ and ∘ in bright colors). The solid line at $v_1 + v_2 = 0.5$ separate the two classes.

This non-linear classification with step function transformation can be represented with a multi-layer neural network. As shown in Figure 50, each of the input variables v_1 and v_2 is normalized by a step function and then multiplied by -1 before feeding into the final output, which is combined with the bias weighted 0.5. This is equivalent to $0.5 \times 1 - 1 \times f(v_1) - 1 \times f(v_2)$, where f is the step function for v_1 and v_2 values.

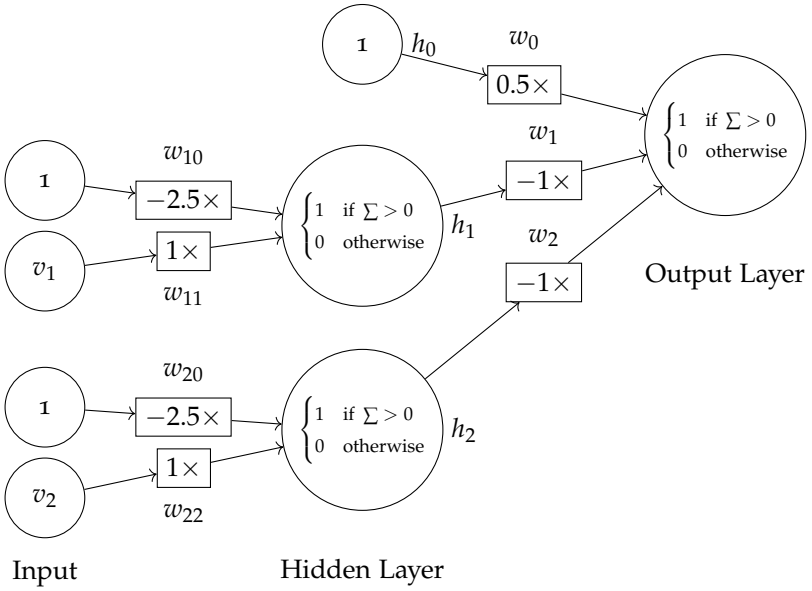


Figure 50: Multi-layer neural network with step functions. $\sum$ represents the weighted sum of input values leading to the present node.

For example, with data point $(2, 2)$, the step function will convert them into $(0, 0)$. With the bias 0.5×1, the weighted sum is $0.5 \times 1 - 1 \times 0 - 1 \times 0 = 0.5$ and it will be classified positive (acceptable). With data point $(3, 1)$, it is transformed into $(1, 0)$ by the step function and the final weighted sum is $0.5 \times 1 - 1 \times 1 - 1 \times 0 = -0.5$, which will be classified negative (unacceptable).

Using the multi-layer neural network, based on the additional hidden layer with step functions between the input and final output, it is now possible to establish a non-linear model such as the one in Figure 50.

But the question remains – how can the weights of the model, together with the step function, be identified properly? While it is tempting to use a similar strategy as with the single perceptron, the non-linearity of the model rules out such a linear strategy for weight adjustment. That is, simply adding or subtracting weights (linear combinations) does not help when the model is non-linear.

Can we adjust the weights in a non-linear manner during training? In other words, can we adjust the weights based on how much each variable contributes to the final output via the hidden layer? The general optimization technique called *gradient descent* makes it possible to find optimal values (weights) by searching in the derivatives of related functions. However, a step function, as illustrated in Figure 45, has a sudden jump (not smooth transition) between 0 and 1 and is not differentiable.

If we are to retain the idea of a step function, can we develop one that functions in the like manner and yet smooth and differentiable?

A candidate is the *sigmoid* function, which is defined by:

$$f(v) = \frac{1}{1+e^{-v}} \tag{48}$$

for a value v. Figure 51 shows the functional curve of sigmoid, which transforms the weighted sum of a node to a value approaching 0 or 1. The threshold is 0 by default but we can use any threshold for the transformation. For example, with a threshold of 5, the sigmoid function becomes:

$$f(v) = \frac{1}{1+e^{-(v-5)}} \tag{49}$$

which shifts the functional curve on the x coordinates by $+5$ (dashed line in Figure 51). $v - 5 = 1 \times v + (-5) \times 1$ is equivalent to the weighted sum of variable v (with weight $w = 1$) and bias 1 (with weight $w_0 = -5$). It can be generalized as a linear function of any number of input variables with bias in the form of $\sum = w_0 + \sum_i w_i v_i$.

Now if we replace the step function with the sigmoid function in Figure 50, we have the multi-layer neural network in Figure 52, in which each layer (node) is differentiable and the weight can be learned by gradient descent.

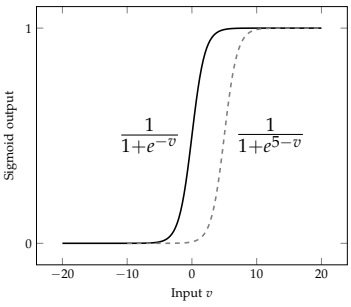


Figure 51: Sigmoid function

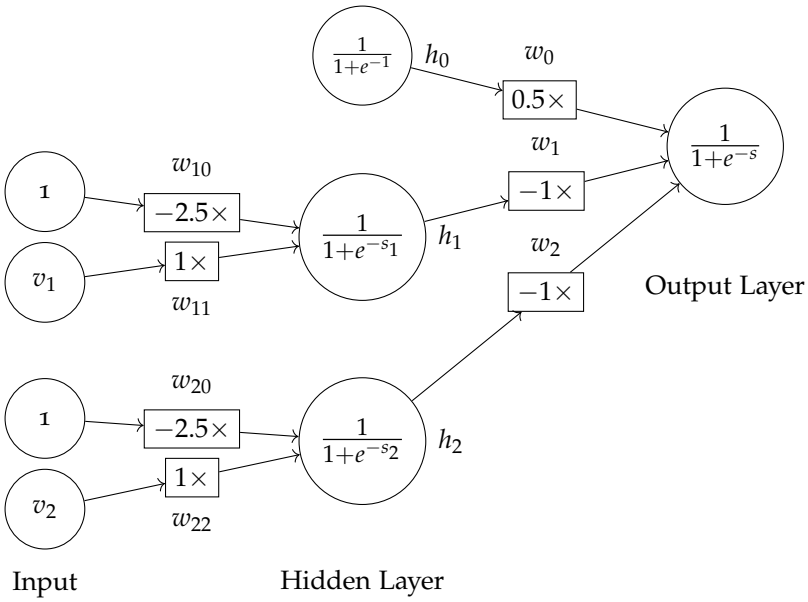


Figure 52: Multi-layer neural network with the sigmoid function. s_1 and s_2 are the weighted sum of input values leading to the respective hidden nodes whereas s is the weighted sum of input values leading to the output layer.

The weighted sum for each hidden node i can be computed by:

$$s_i = \sum_{j=0}^{m} w_{ij} v_j \tag{50}$$

where m is the number of variables and v_0 is the bias, which is always 1. Given the sigmoid function f, the output of each node in the hidden layer is:

$$\begin{aligned} h_i &= f(s_i) && (51)\\ &= f(\sum_{j=0}^{m} w_{ij} v_j) && (52) \end{aligned}$$

In Figure 52, this can be written as:

$$\begin{aligned} h_1 &= f(\sum_{j=0}^{m} w_{1j} v_j)\\ &= f(w_{10} v_0 + w_{11} v_1 + w_{12} v_2)\\ &= f(-2.5 \times 1 + 1 \times v_2 + 0 \times v_2)\\ h_2 &= f(\sum_{j=0}^{m} w_{2j} v_j)\\ &= f(w_{20} v_0 + w_{21} v_1 + w_{22} v_1)\\ &= f(-2.5 \times 1 + 0 \times v_2 + 1 \times v_2) \end{aligned}$$

The final output, predicted value $\hat{y}$, is the sigmoid of the weighted sum of values from the hidden layer:

$$\begin{aligned} \hat{y} &= f(s) && (53)\\ &= f(\sum_{i=0}^{m} w_i h_i) && (54)\\ &= f\Big(\sum_{i=0}^{m} w_i f(\sum_{j=0}^{m} w_{ij} v_j)\Big) && (55) \end{aligned}$$

which, for the model in Figure 52, can be computed by:

$$\begin{aligned} \hat{y} &= f(w_0 h_0 + w_1 h_1 + w_2 h_2)\\ &= f\Big(w_0 f(1)\\ &\quad + w_1 f\big(w_{10} + w_{11} v_1 + w_{12} v_2\big)\\ &\quad + w_2 f\big(w_{20} + w_{21} v_1 + w_{22} v_2\big)\Big) \end{aligned}$$

Let y be the actual outcome (class 0 or 1) for input from the hidden layer h. If we define the error E as a squared function:

$$E(h) = \frac{1}{2}(y - \hat{y})^2 \quad (56)$$

For a prediction based on the sigmoid function f, it can be shown that the differential with respect to a weight w_i to the output layer is:

$$\frac{dE(h)}{dw_i} = \big(f(s) - y\big) f'(s) f(s_i) \quad (57)$$

$$= \big(f(s) - y\big) f'(s) h_i \quad (58)$$

where s is the weighted sum at the output layer and $f'(s)$ is the derivative of the sigmoid function $f(s)$, which is as simple as $f(s)(1 - f(s))$. This gives the differentials needed to optimize w_i weights from the hidden layer to the output, i.e. w_0, w_1, and w_2 weights in Figure 52.

Now that values in the hidden layer h are based on weighted sums of the input layer, we also need to optimize w_{ij} weights connecting the input to the hidden layer. Given input values v, the differential with respect to a weight w_{ij} to hidden node h_i can be computed by:

$$\frac{dE(v)}{dw_{ij}} = \big(f(s) - y\big) f'(s) f'(s_i) v_j \quad (59)$$

The differentials will be used to update weights to both layers during the training process. In particular, the differential $\frac{dE(h)}{dw_i}$ in Equation 58 will be used to subtract from existing weights of w_0, w_1, and w_2. The differential $\frac{dE(v)}{dw_{ij}}$ in Equation 59 will be used to subtract from existing weights of w_{10} and w_{11} for h_1; w_{20} and w_{22} for h_2 in Figure 52. This approach of revising weights to the two layers can thought of as propagating differentials of squared errors *backward* in the feed-forward neural network, and is referred to as *backpropagation*.

To understand how backpropagation actually operates, let's walk through the training process with the first data instance in Table 9.

Buying Price	Maintenance Cost	Evaluation
2	**2**	**1**
..	..	..

Table 9: Car evaluation, training data for multi-layer neural network. Evaluation value 1 for *acceptable* and 0 for unacceptable.

Before training, we don't know the weights and may initialize them with all 0 values. Note that we use 0 initial weights for easy demonstration. This is not necessarily the best strategy for initialization.

Now with the model and initial weights in Figure 53, given the first training instance of $(2, 2)$, we can compute:

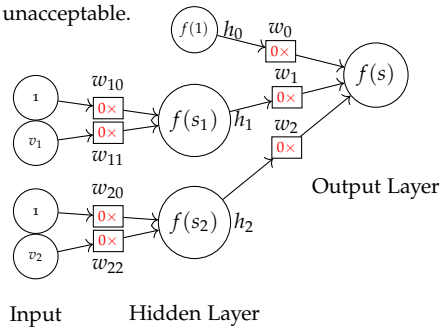


Figure 53: Backpropagation step 1 with 0 initial weights.

$$\begin{aligned} h_0 &= f(1) \\ &= \frac{1}{1+e^{-1}} \\ &= 0.731 \\ h_1 &= f(s_1) \\ &= f(0 \times 1 + 0 \times 2) \\ &= 0.5 \\ h_2 &= f(s_2) \\ &= f(0 \times 1 + 0 \times 2) \\ &= 0.5 \end{aligned}$$

The final prediction $\hat{y}$ is computed by:

$$\begin{aligned} f(s) &= f(0 \times 0.731 + 0 \times 0.5 + 0 \times 0.5) \\ &= f(0) \\ &= 0.5 \end{aligned}$$

We get:

$$\begin{aligned} \big(f(s) - y\big) f'(s) &= \big(f(s) - y\big) f(s) \big(1 - f(s)\big) \\ &= (0.5 - 1) \times 0.5 \times (1 - 0.5) \\ &= -0.125 \end{aligned}$$

With the actual class being 1, we can compute the differentials. For each w_i weight leading to the output layer:

$$\begin{aligned} \frac{dE(h)}{dw_0} &= \big(f(s) - y\big) f'(s) h_0 && (60) \\ &= -0.125 \times h_0 && (61) \\ &= -0.125 \times 0.73 && (62) \\ &= -0.0914 && (63) \\ \frac{dE(h)}{dw_1} &= -0.125 \times 0.5 && (64) \\ &= -0.0625 && (65) \\ \frac{dE(h)}{dw_2} &= -0.0625 && (66) \\ & && (67) \end{aligned}$$

Subtracting these (negative) values from their current weights respectively, we obtain the new weights $w_0 = 0.0914$, $w_1 = 0.0625$, and $w_2 = 0.625$.

Now we can further back-propagate the error to the w_{ij} weights from the input layer:

$$\begin{aligned}
\frac{dE(v)}{dw_{10}} &= \big(f(s)-y\big)f'(s)f'(s_1)v_0 && (68)\\
&= -0.125 \times f(s_1)(1-f(s_1))v_0 && (69)\\
&= -0.125 \times 0.5 \times (1-0.5) \times 1 && (70)\\
&= -0.0313 && (71)\\
\frac{dE(v)}{dw_{11}} &= \big(f(s)-y\big)f'(s)f'(s_1)v_1 && (72)\\
&= -0.125 \times 0.5 \times (1-0.5) \times 2 && (73)\\
&= -0.0625 && (74)\\
\frac{dE(v)}{dw_{20}} &= -0.0313 && (75)\\
\frac{dE(v)}{dw_{22}} &= -0.0625 && (76)
\end{aligned}$$

Subtracting these from current 0 values, we obtain new weights for of all w_{ij}, namely $w_{10} = 0.0313$ and $w_{11} = 0.0625$ to hidden node h_1, and $w_{20} = 0.0313$ and $w_{22} = 0.0625$ for hidden node h_2.

Figure 54 shows the model with updated weights after training by the first instance $(2, 2)$ with a known class label 1. We can continue to train the model with additional data instances in Table 9. This can be repeated with the same data until weights are stabilized and the error is reduced to a certain level.

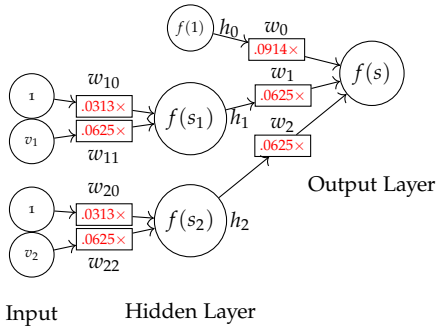
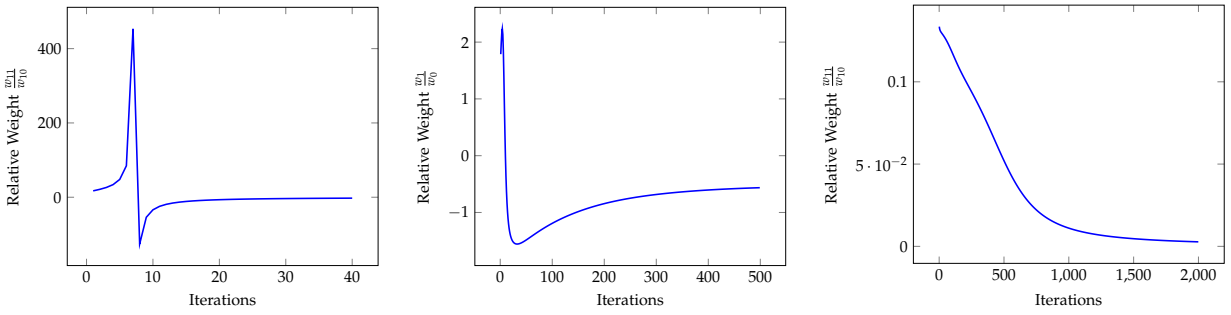


Figure 54: Backpropagation step 2 with updated weights



(a) Weight $\frac{w_{11}}{w_{10}}$ to hidden layer (b) Weight $\frac{w_1}{w_0}$ to output (c) Error $\frac{(y-\hat{y})^2}{2}$

Figure 55: Neural network training iterations and convergence

Figure 55 shows relative weights and error over time, via iterations of repeated training with data in Table 9. Note that in linear equations, absolute values of related weights (coefficients) are not important. Rather, it is their relative weight to one another that matters. Hence, we present weights relative to w_0, i.e. divided by the *bias* weight.

Overall, the figures show the weights and error all appear to converge. However, they seem to differ in the number of iterations

needed for their convergence. In Figure 55 (a), the relative weight of w_{11} (the weight of v_1 to the hidden node h_1) fluctuates a lot in the first 10 iterations but stabilizes quickly after a few dozen iterations. In Figure 55 (b), w_1 – the weight from the hidden node h_1 to the final output – appears to take hundreds of iterations to converge. The squared error, as shown in Figure 55 (c), takes even even more iterations to be minimized.

In the end, we obtain the weights in Table 10 with minimized squared error.

Weights	w_0	w_1	w_2	w_{10}	w_{11}	w_{20}	w_{22}	Error
Absolute	-18.4	9.47	9.44	7.35	-2.93	7.43	-2.96	0.0027
Relative	-1	0.52	0.52	1	-0.40	1	-0.40	0.0027
Relative	-1.92	1	1	2.5	-1	2.5	-1	0.0027

Table 10: Neural network final weights

The result of Table 10 can be represented as the final (converged) model in Figure 56, which is highly similar to the one in Figure 52. For example, the weighted sum for hidden node h_1 is $2.5 - v_1$ in the model here, which is simply the negative of what is in Figure 52, where the sum is $v_1 - 2.5$. We observe the same with the hidden node h_2. This leads to the different signs of w_1, w_2 vs. w_0 to the output layer.

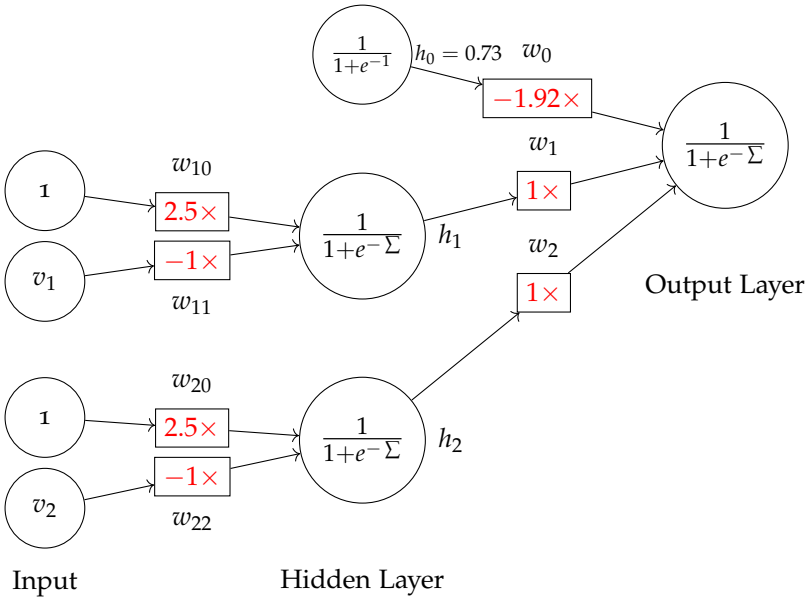


Figure 56: Neural network final model. Compare to Figure 52. Note that we transform the bias node on the hidden layer with the sigmoid as well, i.e. $h_0 = f(1) = \frac{1}{1+e^{-1}} = 0.73$. In the final model, $w_0 h_0 = -1.92 \times 0.73 = -1.4$.

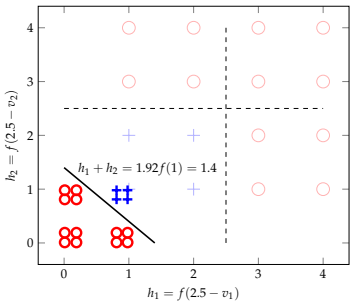


Figure 57: Visualization of the final model on car evaluation. Data transformed by the sigmoid function $h_i = f(s_i) = \frac{1}{1+e^{-s_i}}$, where s_i is the weighted sum of each hidden node: $s_1 = 2.5 - v_1$ and $s_2 = 2.5 - v_2$. The solid line at $h_1 + h_2 - 1.92 f(1) = 0$ is where the sigmoid of the output layer makes the cut.

Figure 57 visualizes how the model transforms the original data into something separable. A sigmoid function with a weighted sum of $2.5 - v$ (for both v_1 and v_2) converts the original data ($+$ and $\circ$ in faded colors) to values close to 0 (if $v > 2.5$) and 1 values (if $v < 2.5$).

Given the limited training data we have, there are many non-linear solutions to the classification problem. In the end, we obtain the model in Figure 56 which is different from what we proposed in Figure 52. Both models will make perfect classifications on the training data presented in Table 9. However, a model thus obtained has not necessarily found the global optimal. In fact, the back-propagation mechanism via gradient descent can converge to a local optimum, which depends on many factors such as the starting point (initial values) and the sequence of training data. The learning rate also impacts the convergence of weights and, occasionally, may also influence where they converge.

Multiple Choices

Numeric Predictions

Text and Human Language

> The big print giveth, and the fine print taketh away.
>
> *Fulton J. Sheen*

Text has been the primary form of documentation in human history. Today large volumes of unstructured text data are being generated and disseminated at an unprecedented pace, via such media as tweets, emails, and scholarly publications. The magnitude of these data has far exceeded our human capacity to read and digest. How can we possibly sift through the world of written text and access what we need? In what ways can computers, which have helped create this ever-growing pile of data, be harnessed to also help us discover relevance and meaning?

Text Vectorization

Let's start with some basic issues and ideas for processing text.

Imagine having a long legal document, it can be stored digitally along with other documents and files. In its raw text format, you can hardly use the document to compute and compare – for example, to find other documents related to the same or similar cases – unless we can convert the text into some numbers to be used in the calculation.

Indeed, the first step of text processing involves a general process called *vectorization*, which identifies elements in the text and assigns numbers (weights) to them as its numerical representation. The result of this is a list of values, which can then be used for searching, ranking, and association.

Tokenization

Through *tokenization*, the text can be split into basic elements (tokens) such as paragraphs, sentences, phrases, and/or words. Taking single words as basic elements, the following text:

John is quicker than Mary.

can be split into individual tokens *John*, *is*, *quicker*, *than*, and *Mary*. In a simple classic approach, we only try to capture what tokens (keywords) occur in the text and pay no heed to the order in which they appear. This is referred to as the *bag of words* model, which will lead to some loss of information. Apparently, with this treatment, tokenization of the above text will result in the identical *bag of words* with the following:

Mary is quicker than John.

where the only difference is the order and the meaning is simply the opposite. Indeed, for analyses where the semantic interpretation of a text is critical, the model will suffer from the loss of order. In many practical applications, nonetheless, the *bag of words* works sufficiently well where the central problem is about finding associations through the keywords. For now, we stick with this basic model and revisit the issue of word order later.

Now that we identify the tokens (words), one may consider normalize them so that the same word types in various forms can be unified and treated as the same. In English, one can apply normalization techniques such as:

- *Case folding*, which changes all tokens into lower-case, e.g. *John* and *JOHN* $\rightarrow$ *john*;
- *Stemming*, which reduces tokens to their *roots*, e.g. *compression* and *compressed* $\rightarrow$ *compress*;
- *Lemmatization*, which converts word variants to their base form, e.g. *are* and *is* $\rightarrow$ *be*;

These techniques are language and domain-dependent. Aside from potential benefits of unifying tokens of the same type, token normalization may also lead to undesired consequences, e.g. to treat *CAT* (the company) and *cat* as the same. So careful examination of data and application is necessary to determine whether related normalization techniques may be instrumental or detrimental.

Also, note that tokens are not limited to single words. One can use a moving window to take two, three, or more consecutive words at a time to identify all possible phrases or n-grams. Likewise, you can use a moving window to identify any n consecutive characters as tokens. Character n-grams have shown its utility in applications such as spam detection, where junk emails may pad normal keywords with special characters to avoid being identified.

Term Weighting

Once tokens – single words or phrases – have been identified, they can be sorted and merged into a set of unique tokens, which we refer to as the *dictionary*. Suppose we have a book collection of *The Chronicles of Narnia*, for which we only store their titles[30]:

[30] Bear in mind that full-text is often considered and analyzed in real-world text data. We use the title only here for the simplicity of illustration.

1. The Lion, the Witch and the Wardrobe
2. The Voyage of the Dawn Treader
3. The Horse and His Boy

Based on tokenization of single words with case-folding, the following dictionary (alphabetically ordered) can be identified from this small collection:

1	2	3	4	5	6	7	8	9	10	11
and	boy	dawn	his	horse	lion	the	treader	voyage	wardrobe	witch

Now we can use terms in this dictionary to represent each document based on their occurrences, 0 if one does not occur and 1 if it does. The following matrix shows such a *binary representation*:

	1	2	3	4	5	6	7	8	9	10	11
DOC	and	boy	dawn	his	horse	lion	the	treader	voyage	wardrobe	witch
1	1	0	0	0	0	1	1	0	0	1	1
2	0	0	1	0	0	1	1	1	1	0	0
3	1	1	0	1	1	0	1	0	0	0	0

Table 11: Matrix representation with binary weights

The size of the dictionary will increase with the size of the text collection and the length of each document. However, one can imagine that, for each document, not all dictionary terms will occur and therefore some (many in fact) of vector values are 0s. A more efficient representation is a sparse vector, where only non-zero values (1 in the binary case here) are retained. For example, document 3 in the above matrix can be represented as:

(	1	2	4	5	7	)
	and	boy	his	horse	the	

which only includes terms that appear in the document and have 1 values.

The binary representation only whether or not a term occurs. This is a reasonable treatment for small text documents such as the above example and tweets. With longer text documents, when full-text is used for vectorization, it is useful to also consider *term frequency* (TF).

TF_{dt} is the number of times a term t appears in document d. It is often observed that a term with a higher term frequency in a document is more relevant to the document's topical theme and should, therefore, be given a higher weight. One can simply use the term frequency as term weight:

$$w_{dt} = TF_{dt} \tag{1}$$

One can also normalize the TF value with logarithm:

$$w_{dt} = \begin{cases} 1 + \log TF_{dt}, & \text{if } TF_{dt} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

A basic assumption of counting occurrences in the term frequency weight is that all occurrences are equal and should count equally. In light of the Zipf law, however, we know words are used unequally.

Some words appear more frequently simply because of the effect of *least effort*. Some are stop-words like *a* and *the* that contribute little in meaning. For many of these, there is one thing in common – they appear frequently not only in the current document but also in other documents as well.

In general, we observe that some terms are used so *broadly* in so many documents that they can hardly help differentiate one document from another. They are too broad to be *specific* and informative.

On the other extreme, some terms are used in only a very few documents and they very clearly distinguish these documents from others. These terms are rarer, more specific, and should be given more weight according to the observation here.

Imagine we randomly draw a document from the entire text collection, there is a probability that term t will appear, which can be estimated by:

$$p(t|C) = \frac{n_t}{N} \tag{3}$$

$$p(t'|C) = 1 - \frac{n_t}{N} \tag{4}$$

where n_t is the number of documents containing term t and N is the total number of documents in the collection C. $p(t'|C)$ is the complimentary probability that t does NOT occur.

For a specific document d where term t appears, it is certain so the probability of observing the term is:

$$p(t|d) = 1 \tag{5}$$

$$p(t'|d) = 0 \tag{6}$$

The amount of *information gain* or relative entropy, according to chapter , by seeing the term in document d out of collection C, can be computed by:

$$\begin{aligned} Info(t) &= DL[P(t|d)||P(t|C)] && (7)\\ &= -\sum_{x\in(t,t')} p(x|d)\log\frac{p(x|C)}{p(x|d)} && (8)\\ &= -p(t|d)\log\frac{p(t|C)}{p(t|d)} - p(t'|d)\log\frac{p(t'|C)}{p(t'|d)} && (9)\\ &= -1\cdot\log\frac{n_t}{N} - 0\cdot\log\frac{p(t'|C)}{p(t'|d)} && (10)\\ &= -\log\frac{n_t}{N} && (11) \end{aligned}$$

This information gain of term t is in fact the exactly the classic *Inverse Document Frequency* (IDF) weighting scheme:

$$\begin{aligned} IDF_t &= Info(t) && (12)\\ &= \frac{N}{n_t} && (13) \end{aligned}$$

where n_t, the number of documents with term t, is commonly referred to as *document frequency* (DF).

IDF measures how informative a term is when it appears in a document. For a term that appears in all documents in the collection – the stop-words for example – n_t is the same as N and its IDF weight becomes:

$$\begin{aligned} IDF_t &= \log\frac{N}{N} && (14)\\ &= \log 1 && (15)\\ &= 0 && (16) \end{aligned}$$

For a term that appears in very few documents in a large collection, $n_t << N$ and the IDF weight is great. In this way, the IDF weight penalizes broader terms used in many documents and rewards more selective, rare terms.

Please note that IDF is a collection-wide statistic for each term whereas term frequency (TF) is term statistic specific to a document. They address two separate aspects of term weighting and are often combined for document representation. The combined weighting scheme, namely the classic *TF*IDF*, assigns a term's weight by:

$$w_{dt} = TF_{dt}\log\frac{N}{n_t} \quad (17)$$

or with log-transformed TF weight:

$$w_{dt} = \begin{cases} (1 + \log TF_{dt}) \log \frac{N}{n_t}, & \text{if } TF_{dt} > 0 \\ 0, & \text{otherwise} \end{cases} \quad (18)$$

where TF_{dt} is the number of times term t occurs in document d (TF), n_t is the number of documents having t (DF), and N is the total number of document in the collection.

In certain applications, TF may be normalized by document length to alleviate potential favoritism toward longer documents. This is one of the most widely used term weighting methods in information retrieval and text mining.

Similarity Measures

Once each document is represented as a vector of terms with weights, we can conduct various operations, such as clustering and ranking, on them. The basic unit of these computational tasks is to measure how similar or dissimilar one document is to another.

A simple method to compute the similarity of any two documents d_u and d_v is by the dot product of their vector values:

$$Sim(d_u, d_v) = \sum_{t \in T}^{m} w_{ut} w_{vt} \quad (19)$$

where for each term t in dictionary T, $w_{u,t}$ is the weight of t in document d_u, and $w_{v,t}$ is the weight of t in document d_v. For example, the similarity of doc 1 and 3 given the binary matrix of Table 11 can be computed by:

$$Sim(d_1, d_3) = 1 \times 1 + 0 \times 1 + 0 \times 0 + ... + 1 \times 0 \quad (20)$$

$$= 2 \quad (21)$$

The dot product coefficient in Equation 19 can be used with any term weighting schemes including TF and TF*IDF. However, this similarity measure is not normalized by any size factor and its score can be inflated by document lengths and favor those longer documents. A remedy of this is the classic cosine similarity, which we will discuss in depth.

A binary sparse vector can also be regarded as a *set*, that is, a set including all unique terms that occur in the document. Several similarity coefficients can be computed based on sets. The *Dice Coefficient* computes the ratio between two sets' intersection size and the mean of their sizes:

$$\begin{aligned} Dice(d_u, d_v) &= \frac{|d_u \cap d_v|}{(|d_u| + |d_v|)/2} \\ &= \frac{2|d_u \cap d_v|}{|d_u| + |d_v|} \end{aligned} \tag{22}$$

where $|d_u|$ and $|d_v|$ are cardinalities (sizes) of the two sets, and $d_u \cap d_v$ is their interaction (the set of common terms).

A similar method, *Jaccard Coefficient*, measures the same intersection but as a fraction of the union of the two sets:

$$Jaccard(d_u, d_v) = \frac{|d_u \cap d_v|}{|d_u \cup d_v|} \tag{23}$$

Beyond binary representation, document vectors can be treated as data points in a dimensional space where each term is an independent dimension (coordinate). To simplify the discussion and visualize the dimensional space, suppose we have only two terms in the dictionary, *Data* and *Machine*.[31] Suppose we use TF term weighting and the following is a matrix representation of two documents where the terms appear:

[31] In real world applications, the number of terms can be very large and we may be dealing with thousands, if not millions, of dimensions. While mathematically possible, we cannot visualize more than three dimension due to the limit of our visible, three-dimensional world.

	1	2
DOC	Machine	Data
1	1	5
2	4	1

Table 12: Two documents represented by term frequencies (TF) of two terms

These two documents can be plotted on the two-dimensional space, e.g. *machine* for the x dimension and *data* for the y dimension, shown in Figure 58:

With this dimensional representation, pair-wise similarity or difference can be measured by the geometric distance between two document data points. A classic metric of such is the *Euclidean Distance*, which measures the length of the straight line connecting the two data points in question, as shown in Figure 58. In an m-dimensional space, the general *Euclidean Distance* function is:

$$ED(u, v) = \sqrt{\sum_{i=1}^{m} (u_i - v_i)^2} \tag{24}$$

where u_i and v_i are the i^{th} dimensional value of data points u and v respectively. Given documents d_1 and d_2 in Figure 58, their euclidean distance is:

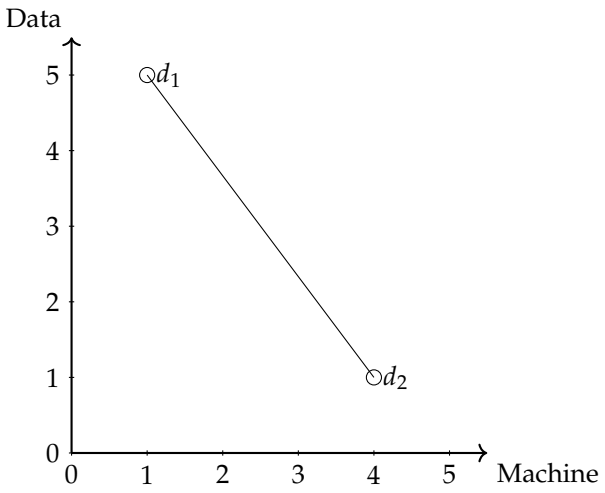


Figure 58: Documents in a two-dimensional space

$$\begin{aligned} ED(d_1, d_2) &= \sqrt{(5-1)^2 + (1-4)^2} \\ &= 5 \end{aligned} \tag{25}$$

While *Euclidean Distance* is widely used in text mining, its performance may suffer in applications where there is a wide distribution of document sizes. A thought experiment will help illustrate the problem.

Imagine we have two documents $d_1 = (4, 5)$ and $d_2 = (5, 4)$ (with TF weights) that are highly similar and have a very small Euclidean distance between them, as show in Figure 59:

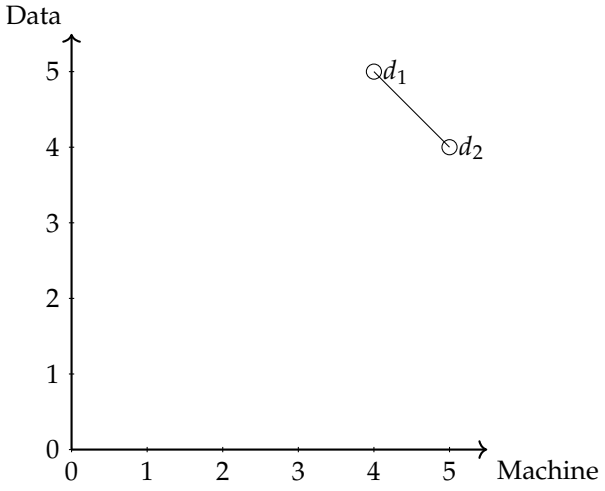


Figure 59: Thought experiment with two documents

For now, one may think the line distance in Figure 59 is a reasonable estimate of their difference. But imagine make a double-copy of d_2 and treat the new copy as d_3, resulting in Figure 59:

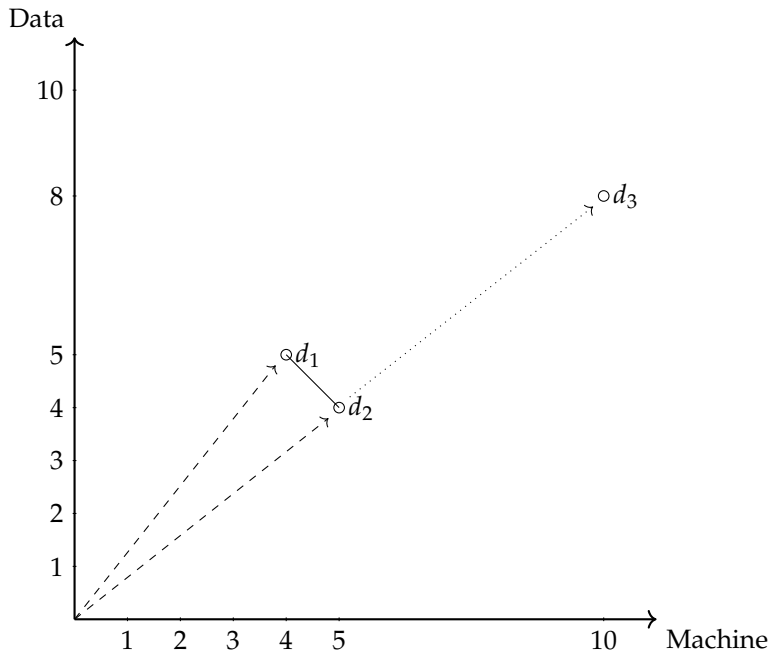


Figure 60: Thought experiment a third document, where d_3 doubles the content of d_2

Because d_3 is copy of d_2 – the only difference is that it doubles everything in d_2 – its semantics should remain the same as d_2. A fair distance measure should give a relatively small value, indicating d_3 is *close* to d_1 and especially d_2. However, as shown in Figure 59, d_3 is much farther away from d_1 and consequently the euclidean distance is much greater.

In short, Euclidean Distance treats documents differently simply because they are of different sizes (magnitude). In this light, we should somehow disregard the sizes when measuring their similarities.

Another perspective on similarity vs. distance in the dimensional space is by measuring the *direction* of a vector. In physics, the notion of vector has been used extensively for quantities such as velocity, which has both a *magnitude* (speed) and direction. Likewise, as shown as dashed lines in Figure 60, documents such as d_1 and d_2 can be treated as vectors from the origin, with a magnitude (line length) and direction (arrow). The difference between the two documents can be measured by their angle distance.

Also in Figure 60, d_3 as copy of d_2 is pointing in the exact same direction of d_2 and their angle distance is 0. Consequently, the angle distance between d_1 and d_3 is the same as that of d_1 and d_2.

All this makes sense with angle distance as d_2 and d_3 essentially have the same content and should be treated equivalent. In this *vector space* representation, a vector's direction is dependent on the *distri-*

bution of term weights in the corresponding text document. While the magnitude is about the absolute values of term weights, the direction is a rather relative measure – that means, in the case of Figure 60, to what degree a document vector is more or less about *machine* (to the east) vs. *data* (to the north).

To measure an *angle distance*, one can calculate the degree of the angle between two vectors. Nonetheless, a classic approach is to measure the *cosine* of the angle as their *similarity*.

As shown in Figure , with the help line perpendicular to vector d_1, the cosine of d_1 and d_2 is:

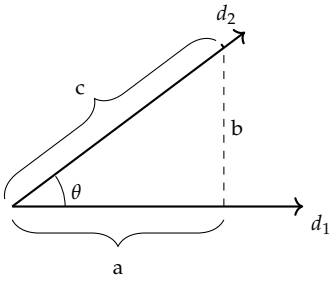


Figure 61: Cosine of an angle

$$Cosine(d_1, d_2) = \frac{a}{c} \tag{26}$$

With the vector representations of any two documents u and v, the same cosine can be computed by:

$$Cosine(d_u, d_v) = \frac{\sum_{i=1}^{m} u_i v_i}{\sqrt{\sum_{i=1}^{m} u_i^2}\sqrt{\sum_{i=1}^{m} v_i^2}} \tag{27}$$

where u_i is the weight of the i^{th} term in document u and v_i is the same term's weight in document v. For the three documents in the thought experiments, $d_1 = (4, 5)$, $d_2 = (5, 4)$, and $d_3 = (10, 8)$, we can compute their pairwise similarities:

$$\begin{aligned} Cosine(d_1, d_2) &= \frac{4*5+5*4}{\sqrt{4^2+5^2}\sqrt{5^2+4^2}} \\ &\approx 0.98 \\ Cosine(d_1, d_3) &= \frac{4*10+5*8}{\sqrt{4^2+5^2}\sqrt{10^2+8^2}} \\ &\approx 0.98 \\ Cosine(d_1, d_2) &= \frac{4*8+5*10}{\sqrt{4^2+5^2}\sqrt{10^2+8^2}} \\ &= 1 \end{aligned} \tag{28}$$

In this case, d_1 and d_2 are identical with cosine 1 whereas d_2 and d_3 have the same similarity compared to d_1.

Cosine is inversely related to the angle θ (distance) and is a similarity measure. When two vectors are close in their direction, with a small angle shown in Figure , a is almost the same as c and cosine similarity is nearly 1. The two vectors are highly similar in their directions, with 1 indicating the same direction and perfect match.

On the other hand, when two vectors point to directions nearly perpendicular to one another, with a large angle shown in Figure ,

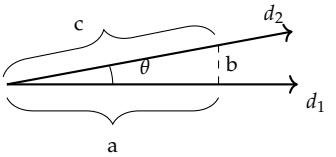


Figure 62: Cosine of a small angle, i.e. vectors of close directions

a becomes extremely small and cosine similarity is close to 0. For vectors based on text documents, their vector values are normally positive (in the first quadrant in the two-dimensional space). Therefore, the smallest cosine similarity value one can obtain is 0 when two vectors are exactly perpendicular (orthogonal directions).

In short, cosine is a similarity measure in the vector space model, which ranges from 0 and 1. A greater cosine value indicates a higher similarity of two vectors.

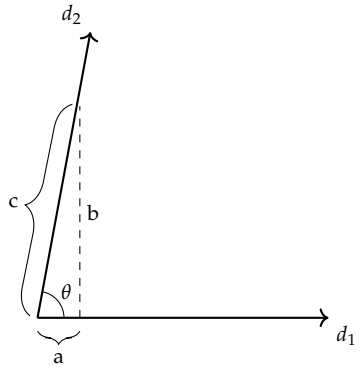


Figure 63: Cosine of a large angle, i.e. vectors with nearly perpendicular directions

Probabilities and Implications

Zipf's Law

The notion of *least action* or *least effort* has long been observed in the natural world as well as human behavior. In spoken language, we tend to minimize our efforts by reusing old words and expressions, as if they are old tools meant for many applications. The results of this: words tend toward "a minimum in number and size" and "the older the tool (word) the more different jobs it can perform."[32]

[32] George K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, 1949

Under the *Principle of Least Effort*, there appears to be a great divide between a small number of words so frequently used and those rarely mentioned. In particular, if we sort words by their frequencies, a term t's frequency f_t is proportional to the inverse of its rank k_t:

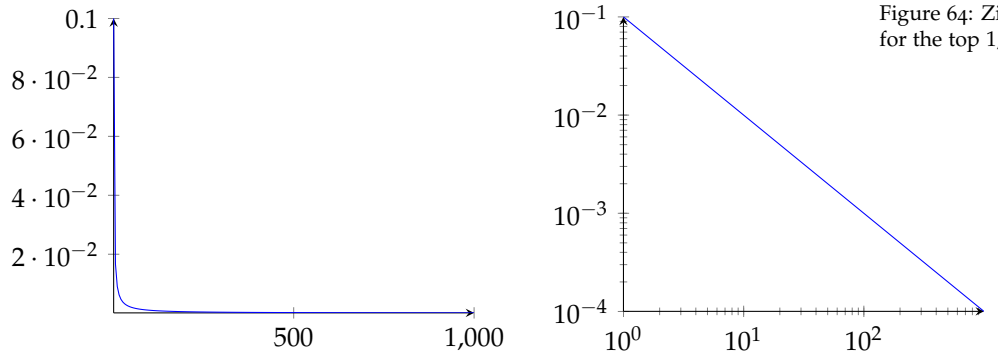
$$f_t \propto \frac{1}{k_t} \tag{29}$$

where word t is ranked the k^{th} most frequent. If we consider probabilities rather than raw frequencies, the above can be written as:

$$p(t) = \frac{c}{k_t} \tag{30}$$

where c is a constant. In the English language, empirical data have shown $p(t) \approx 0.1/k_t$ within the top 1,000 rank. This relation can be visualized in Figure 64 (a).

With a long tail, Figure 64 (a) on normal coordinates is difficult to examine and it is often useful to apply the logarithmic transformation. By taking the log of both x (rank) and y (frequency) values, this can be transformed into Figure 64 (b). As shown, Zipf law is a linear function on log-log coordinates and is a special case of *power-law* distributions, which, as we discussed in Chapter Probabilistic Thinking and Modeling, are results of the *Matthew effect* (the rich get richer). In fact, there is an intrinsic connection between the *Principle of Least Effort* and the *Matthew effect* – as we tend to a minimum set of vocab-



(a) normal coordinates

(b) log-log coordinates

Figure 64: Zipf law function, roughly for the top 1,000 in English

ulary, what we have frequently used (old "rich" words) will be used even more frequently (richer).

Feature Selection

We have observed that any text processing may involve a very large number of features (terms) and it is a common practice to perform *feature selection* in such processes as clustering, classification, and retrieval (search).

In light of Zipf's law, there are a large number of words that are rarely used, only in a few documents. While these terms might be informative when they occur, their rare occurrences make them less useful in such tasks as clustering and classification where documents are grouped by terms they have in common. Given that there are many low frequent words – the long tail of the Zipf function – removing them will reduce the dictionary size (feature space) significantly.

In terms of document frequency (DF), on the other hand, we know highly frequent words such as stop words are not informative and can be safely removed in certain applications. In practice, simple feature selection techniques such as DF (document frequency) thresholding has performed very effectively with little computational cost.[33] It has been shown that having the right number of features is key to the success of text clustering and classification – using all features not only slows down the processing but also contributes more noise than information for related tasks.[34]

[33] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc

[34] Weimao Ke. Information-theoretic term weighting schemes for document clustering and classification. *International Journal on Digital Libraries*, 16(2):145–159, Jun 2015

Text Classification

With a distance or similarity measure in the vector representation, it is now possible to compare documents to one another and asso-

ciate them with data groups or labels. For example, one may use an instance-based lazy learning method such as kNN (see Chapter) with cosine similarity to perform classification and prediction based on the nearest data vectors.

Another successful approach is to model the probability that a document belongs to each class (label) and assign the document to the class that maximizes the probability. That is, given a document representation d, we need to estimate the probability of each class $p(c_i|d)$, where c_i is one of the given classes to be predicted.

Say we are working on email *spam* filtering based on subject titles. This is a binary classification problem, for which we have two classes (labels) – an email is either a *ham* (normal) or a *spam* (junk). Suppose we have a small collection of email subjects which we have labeled *ham* or *spam* to train our model (learning):

DOC	Email Subject	Class
1	You have been selected!	SPAM
2	Have a prize for you!	SPAM
3	You won!	SPAM
4	Project meeting schedule	HAM
5	Nice to meet you!	HAM
6	Project update	HAM
7	Update on research	HAM

Table 13: Training data with email subject and class label

Naive Bayes with Binary Term Occurrences (Bernoulli)

Given the short subject titles, let us first look at how we can model probabilities in text if we only consider binary occurrences of terms. Assume we remove some of stop-words (e.g. have, been) and normalize some others (e.g. meeting → meet, won → win), we can represent the email using the following dictionary:

	1	2	3	4	5	6	7	8	9	10	
DOC	meet	nice	prize	project	research	schedule	select	update	win	you	Class
1							1			1	SPAM
2			1							1	SPAM
3									1	1	SPAM
4	1			1		1					HAM
5	1	1								1	HAM
6				1				1			HAM
7					1			1			HAM

Table 14: Binary representation of email subjects. Note that empty cells are in fact 0 values and they are left out for better readability.

Within a class c, there is a probability that a term t will occur in a (random) document $p(t|c)$. To estimate the probability, one can

simply count the ratio between the number of emails t appears and the total number of emails in class c.

However, as we discussed in Chapter Probabilistic Thinking and Modeling, this approach may lead to skewed estimates and potential zero probability values due to data variance in the sample. Now that we have a very small data set here, we will use the Laplace estimator with an added constant μ to any count, that is:

$$p(t|c) = \frac{n(t|c) + \mu}{N(c) + 2\mu} \tag{31}$$

where $n(t|c)$ is the number of documents in class c that contain term t (document frequency) and $N(c)$ is the total number of documents belonging to c. μ is a constant added to avoid zero probability estimates and 2μ in the denominator is to account for μ being added to the binary cases (for when t occurs and when it does not).

For example, with $\mu = 0.05$, some of the term probabilities in the SPAM class can be computed by:

$$\begin{aligned} p(meet|spam) &= (0 + 0.05)/(3 + 0.1) \\ &= 0.016 \\ p(prize|spam) &= (1 + 0.05)/(3 + 0.1) \\ &= 0.339 \\ p(you|spam) &= (3 + 0.05)/(3 + 0.1) \\ &= 0.984 \end{aligned}$$

The same probabilities in the HAM class are:

$$\begin{aligned} p(meet|ham) &= (2 + 0.05)/(4 + 0.1) \\ &= 0.5 \\ p(prize|ham) &= (0 + 0.05)/(4 + 0.1) \\ &= 0.012 \\ p(you|ham) &= (1 + 0.05)/(4 + 0.1) \\ &= 0.256 \end{aligned}$$

In the binary representation, $w_t = 1$ when term t occurs in the document and $w_t = 0$ if it does not. Given $p(t|c)$ probability term t occurs in a (random) document in class c, $1 - p(t|c)$ is the probability that it does not occur. That is, the probability of observing a binary term weight w_t can be written as the *Bernoulli* probability mass function:

$$p(w_t|c) = p(t|c)^{w_t}(1-p(t|c))^{1-w_t} \quad (32)$$

$$= \begin{cases} p(t|c), & \text{if } w_t = 1 \\ 1-p(t|c), & \text{otherwise } w_t = 0 \end{cases} \quad (33)$$

For term *prize* in the SPAM class, the above can be written as:

$$p(w_{prize}|spam) = p(prize|spam)^{w_{prize}}(1-p(prize|spam))^{1-w_{prize}}$$

$$= \begin{cases} 0.339, & \text{if prize occurs } w_{prize} = 1 \\ 0.661, & \text{otherwise } w_{prize} = 0 \end{cases}$$

Similarly, probability of a term weight for *you* in the HAM class:

$$p(w_{you}|ham) = p(you|ham)^{w_{prize}}(1-p(you|spam))^{1-w_{you}}$$

$$= \begin{cases} 0.256, & \text{if it occurs } w_{you} = 1 \\ 0.744, & \text{otherwise } w_{you} = 0 \end{cases}$$

Following this, we can estimate the probability of *any* term weight w_t in the classes HAM vs. SPAM. Now suppose we receive a new email and try to determine whether it is a *ham* or a *spam*:

DOC	Email Subject	Class
d	Your prize!	?

Table 15: A new email to be classified

which can be represented as:

	1	2	3	4	5	6	7	8	9	10	
DOC	meet	nice	prize	project	research	schedule	select	update	win	you	Class
d			1							1	?

Table 16: Binary representation of the new email. 0 values are omitted for readability.

The question here is how we can estimate the probability of the email d being *ham* or *spam*, $p(ham|d)$ vs. $p(spam|d)$. Using the Bayes theorem, they can be computed by:

$$p(ham|d) = \frac{p(ham)p(d|ham)}{p(d)} \quad (34)$$

$$p(spam|d) = \frac{p(spam)p(d|spam)}{p(d)} \quad (35)$$

Both equations above have $p(d)$ as the denominator, which does not affect the ratio of the two values (classification outcome) and can be ignored in the calculation.

$p(ham)$ and $p(spam)$ can be estimated by counting the number of hams vs. spams. With a Laplace estimator, they are:

$$\begin{aligned} p(ham) &= (4+0.05)/(7+0.1) \\ &= 0.57 \\ p(spam) &= (3+0.05)/(7+0.1) \\ &= 0.43 \end{aligned}$$

For $p(d|spam)$, email d can be viewed as a collection of its term weights w_t. Given the 10 terms in the dictionary, $p(d|spam)$ is their joint probabilities $p(w_t|spam)$ in the *spam* class. Assume the term probabilities are independent (naive assumption), the conditional probability can be computed by:

$$\begin{aligned} p(d|spam) &= \prod_t p(w_{dt}|spam) \\ &= p(w_{meet}=0|spam)p(w_{nice}=0|spam)p(w_{prize}=1|spam)...p(w_{you}=1|spam) \\ &= 0.984 \times 0.984 \times 0.339... \times 0.984 \\ &= 0.132 \end{aligned}$$

Likewise, we can compute the probability conditioned on the *ham* class:

$$\begin{aligned} p(d|ham) &= \prod_t p(w_{dt}|ham) \\ &= p(w_{meet}=0|ham)p(w_{nice}=0|ham)p(w_{prize}=1|ham)..p(w_{you}=1|ham) \\ &= 0.5 \times 0.744 \times 0.012... \times 0.256 \\ &= 0.013 \end{aligned}$$

Plug the numbers back in Equations 34 and 35, we have:

$$
\begin{aligned}
p(spam|d) &= \frac{p(spam)p(d|spam)}{p(d)} \\
&= \frac{0.43 \times 0.293}{p(d)} \\
&= \frac{0.132}{p(d)} \\
p(ham|d) &= \frac{p(ham)p(d|ham)}{p(d)} \\
&= \frac{0.57 \times 0.013}{p(d)} \\
&= \frac{0.007}{p(d)}
\end{aligned}
$$

Clearly $p(ham|d) < p(spam|d)$ and the new email d will be predicted as a SPAM.

Two important notes here. First, although the example only has two classes, the Naive Bayes classifier can be applied to more than two classes. Given a document d and a set of classes $C = [c_1, c_2, .., c_k]$, where $k \geq 2$, the general Naive Bayes model with Bernoulli distribution is to identify a class $\hat{c}$ such that the posterior probability $p(c|d)$ is maximized:

$$
\begin{aligned}
\hat{c} &= \arg\max_{c \in C} p(c|d) && (36) \\
&= \arg\max_{c \in C} \frac{p(c)p(d|c)}{p(d)} && (37) \\
&= \arg\max_{c \in C} \frac{p(c)\prod_t p(w_{dt}|c)}{p(d)} && (38)
\end{aligned}
$$

where w_{dt} is the binary weight of term t in document d (to be classified). $p(d)$ is the common denominator for all classes and can be ignored in the calculation.

Second, the product of joint probabilities (with $\prod_t$ in the above equation) in the model can become a very small value, especially when there are a great number of terms in the dictionary. To avoid this computational difficulty, it is a better practice to convert this into a log-likelihood (by taking the logarithm of the probabilities):

$$
\begin{aligned}
\hat{c}_{log} &= \arg\max_{c \in C} \log \frac{p(c)\prod_t p(w_{dt}|c)}{p(d)} && (39) \\
&= \arg\max_{c \in C} \log p(c) + \sum_t \log p(w_{dt}|c) - \log p(d) && (40)
\end{aligned}
$$

In this way, we perform the addition of log-likelihood and can maintain a better precision with float points. Again $\log p(d)$ is a constant across the classes and can be omitted in the computing.

Naive Bayes with Term Frequencies (Multinomial)

Now let us look beyond binary term occurrences for text classification. As we discussed earlier in the chapter, term frequency (TF) is a useful statistic, which we should also take advantage of in this task.

Suppose we extend the email representation from subject title to include email body (content) as well. Assume the dictionary (vocabulary) remains the same as in Table 14 but obviously terms may occur multiple times in the email text. The following is a hypothetical TF representation of the same emails:

	1	2	3	4	5	6	7	8	9	10	
DOC	meet	nice	prize	project	research	schedule	select	update	win	you	Class
1							1			3	SPAM
2			3							2	SPAM
3									2	2	SPAM
4	2			1		3					HAM
5	1	1								2	HAM
6				2				1			HAM
7					3			1			HAM

Table 17: Term frequency (TF) representation of email subjects. Note that empty cells are 0 values and they are left out for better readability.

The probability of term t in class c, $p(t|c)$ can be computed by its overall term frequency (in all documents) over the total term frequency of documents in that class. Given term frequencies in Table 17, the following terms' probabilities in the SPAM class can be estimated by[35]:

[35] Again, we add a Laplace constant of 1, which is 1 added to the numerator and 2 added to the denominator.

$$\begin{aligned} p(nice|spam) &= (0+1)/[(1+3+3+2+2+2)+2] \\ &= 1/15 \\ p(prize|spam) &= (3+1)/15 \\ &= 4/15 \\ p(you|spam) &= (3+2+2+1)/15 \\ &= 8/15 \end{aligned}$$

Likewise, these terms' probabilities in the HAM class can be estimated by:

$$
\begin{aligned}
p(nice|ham) &= (1+1)/[(2+1+3+1+1+2+2+1+3+1)+2] \\
&= 2/19 \\
p(prize|ham) &= (0+1)/19 \\
&= 1/19 \\
p(you|ham) &= (2+1)/19 \\
&= 3/19
\end{aligned}
$$

Now given the following new document to be classified:

	1	2	3	4	5	6	7	8	9	10	
DOC	meet	nice	prize	project	research	schedule	select	update	win	you	Class
d		1	3							2	?

Table 18: TF representation of the new email. 0 values are omitted for readability.

Document d can be regarded as a sequence of terms that are drawn from their probability distributions. And the probability of generating the document $p(d)$ is proportional to the joint probability of drawing the individual terms, following the Multinomial distribution[36]:

$$
\begin{aligned}
p(d) &\propto \prod_{k=1}^{n_d} p(t_k) && (41) \\
&= \prod_{t \in T} p(t)^{TF_{dt}} && (42)
\end{aligned}
$$

[36] With the bag-of-words approach, the Multinomial probability mass function has to include as a factor $\frac{n_d!}{\prod_t TF_{dt}!}$ because of the permutations the terms can appear in. This Multinomial coefficient, however, is a constant given a fixed document d, is the same for all classes and can be omitted.

where n_d is the length of document d (the total number of terms including duplicates) and t_k is term t at position of k in the document. In the bag-of-words approach, positional information is irrelevant so $p(t_k) = p(t)$ for any position k. T is the dictionary of unique terms, and TF_{dt} is term frequency of term t in document d.

If document d belongs to be particular class c, the posterior probability can be computed in the same manner:

$$
p(d|c) = \prod_{t \in T} p(t|c)^{TF_{dt}} \quad (43)
$$

For document d in Table 18, IF it belongs to the SPAM class, then the log-likelihood of having document d is[37]:

[37] Note that in the calculation we omit terms that do not occur in document because a 0 TF value will lead to 0 in the log-likelihood and does not affect the result.

$$\begin{aligned}
\log p(d|spam) &= \log \prod_{t \in T} p(t|spam)^{TF_{dt}} \\
&= \sum_{t \in T} TF_{dt} \log p(t|spam) \\
&= 1 \log p(nice|spam) + 3 \log p(prize|spam) + 2 \log p(you|spam) \\
&= \log \frac{1}{15} + 3 \log \frac{4}{15} + 2 \log \frac{8}{15} \\
&= -3.444
\end{aligned}$$

Likewise, we can compute the log-likelihood of d IF it belongs to the *ham* class instead:

$$\begin{aligned}
\log p(d|ham) &= \sum_{t \in T} TF_{dt} \log p(t|ham) \\
&= 1 \log p(nice|ham) + 3 \log p(prize|ham) + 2 \log p(you|ham) \\
&= \log \frac{2}{19} + 3 \log \frac{1}{19} + 2 \log \frac{3}{19} \\
&= -6.417
\end{aligned}$$

Finally, we can estimate the posterior probability of document d belonging to either spam or ham by using the Bayes theorem. Again, the log-likelihood is used:

$$\begin{aligned}
\log p(spam|d) &= \log \frac{p(spam)p(d|spam)}{p(d)} \\
&= \log p(spam) + \log p(d|spam) - \log p(d) \\
&= \log 0.43 - 3.444 - \log p(d) \\
&= -3.811 - \log p(d) \\
\log p(ham|d) &= \log \frac{p(ham)p(d|ham)}{p(d)} \\
&= \log p(ham) + \log p(d|ham) - \log p(d) \\
&= \log 0.57 - 6.417 - \log p(d) \\
&= -6.66 - \log p(d)
\end{aligned}$$

In this case, $\log p(spam|d) > \log p(ham|d)$ so $p(spam|d) > p(ham|d)$ and document d should be labeled a SPAM.

Back to the general model of Naive Bayes with a Multinomial distribution, the idea is to consider term frequencies in the model and identify the class that maximizes the log-likelihood of the document's membership to that class. That is[38]:

$$\hat{c} = \arg\max_{c \in C} \log p(c) \prod_{t} p(t|c)^{TF_{dt}} \quad (44)$$

$$= \arg\max_{c \in C} \log p(c) + \sum_{t \in T} TF_{dt} \log p(t|c) \quad (45)$$

[38] Here we omit $\log p(d)$ as it is constant for all classes.

where c is each class in the set of classes C, t is each term in dictionary T, $p(t|c)$ is the probability of term t in class c, and TF_{dt} is the term frequency of t in document d, the document to be classified.

Summary

The basic method for text processing is the *bag of words* approach, in which we disregard the order of *tokens*. In practice, for the English language, tokens are often single words which can be obtained using word boundaries such as space and punctuation. Nonetheless, a token can be a phrase or any number of characters in a sequence (character n-gram). For example, you may treat every four consecutive characters (4-gram) in a document as a token and use all unique 4-gram tokens as your vocabulary.

Some form of token normalization can be helpful. However, the extent of its use depends on the language, data domain, and use cases, e.g. how users express their queries vs. words used in written documents of the data collection.

Once tokens or terms are obtained and normalized, we can then assign certain weights to them for the representation of text documents. We have discussed term weighting schemes including the binary, term frequency (TF), and TF*IDF (Inverse Document Frequency). TF*IDF is a classic treatment where a term's weight increases with the increase of term frequency in the current document and its rarity in other documents. IDF, or a term's rarity, indicates the degree of *specificity* and *informativeness* of the term.

Text documents can be treated as vectors based on the term weights. When comparing two vectors in the vector space model, we noticed that vector direction is more important than vector magnitude (length). Whereas the magnitude is a function of the absolute values of term weights, the vector direction is due to the distribution of these (relative) weights. Because of this nature, we often use an angle-based distance function such as cosine similarity to compare two vectors.

The probabilistic view provides an important approach to text processing. The Zipf's is an empirical generalization of the probabilistic occurrences of spoken or written words. The Bayes theorem lays the foundation for probabilistic modeling of problems such as text classification and retrieval.

Clustering and Organization

We as human organize things. An important aspect of human development is to recognize the commonality of objects and group them together, for example, in terms of shapes and colors.

A related category of machine learning algorithms is called *clustering*, which aims to bring like entities together and, by doing so, discovers major groups, patterns, and themes in the data. Clustering is often referred to as unsupervised learning because its outcome is not predetermined (unsupervised) but whatever the data shall reveal.

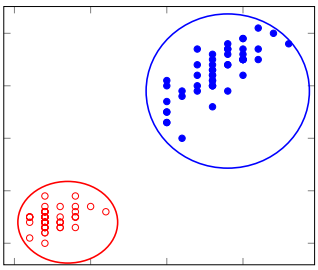


Figure 65: Hard, flat partitioning

Clustering helps in the organization (grouping) of information and makes it easier to access and navigate. For example, when a student organizes course materials in folders of subject areas, it becomes more efficient for her to locate related materials when she needs to review them.

Clustering is also an important step toward information aggregation and pattern recognition of data at hand. By discovering major groups and themes, a high-level summarization and overview can be identified with data instances representative of these groups. This can often be integrated with interactive information visualization for graphical representation and navigation of the data space.

We often think of clustering as *hard* and *flat partitioning* – *hard* because cluster membership is concrete and every data instance belongs to only one group and *flat* because there is only one level of clusters. Figure 65 shows an example of two flat partitions, where data belong to either one of them. However, the result of clustering can be a more complex structure than that.

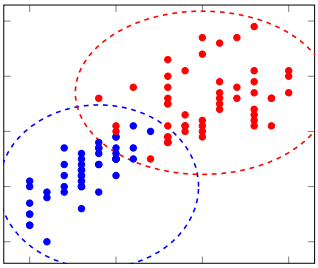


Figure 66: Illustration of soft partitioning, where data may belong to multiple clusters

There are such methods for *soft* or *fuzzy* clustering, where each data instance can belong to multiple clusters. For example, a book on quantum computing can group with other books on quantum physics as well as those in computer science/engineering. The membership is not a concrete assignment but a degree of association. One can certainly view such cluster associations as probabilities. Soft clustering can be very helpful in applications where one may need different *routes*, so to speak, to data of interest.

In addition, there are clustering algorithms that produce a hier-

archical structure with levels of nested partitions, i.e. a tree with branches and branches of branches. Figure 68 illustrates hierarchical clustering, where a circle represents a cluster and a nested circle is a child (branch) of the one containing it. This can be achieved by either recursively merging data instances until the whole structure has emerged (the bottom-up approach) or recursively dividing the entire data into subsets until they can no longer be broken down (the top-down approach).

The result of hierarchical clustering in Figure 67 can also be represented as a tree structure in Figure 68. A hierarchy or dendrogram thus constructed may reveal the underlying taxonomy of themes in the data. It is a great tool for information organization and data exploration.

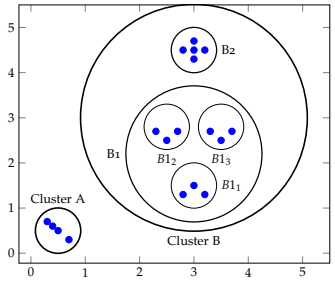


Figure 67: Hierarchical clustering. A nested circle represents child (branch) of the enclosing cluster.

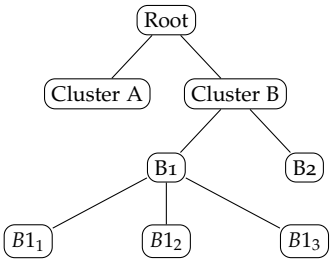


Figure 68: Tree representation of the hierarchical structure in Figure 67

Example Data

Now that we have had an overview of what clustering does and what it can produce, we shall examine how related algorithms perform clustering on data. We will look at a few classic methods and illustrate the clustering process with a well-known dataset, namely the IRIS flower data created by British statistician and geneticist Ronald Fisher. Table shows a small sample of the data set:

Sepal Length	Sepal Width	Petal Length	Petal Width	Class
5.1	3.5	1.4	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.3	3.3	6.0	2.5	Virginica
..	..	..	..	..

Table 19: IRIS flowers sample data

Each instance is an Iris flower observation, with measurements of its sepal length, sepal width, petal length, and petal width in centimeters, as well as classification of three Iris species (setosa, versicolor, or virginica). For demonstration, we use only 5 data instances from each species (15 total) and focus on two variables in the analysis, namely *petal length* vs. *petal width*. In Figure 69, we plot *petal length* vs. *petal width* and color code data in each class (species).

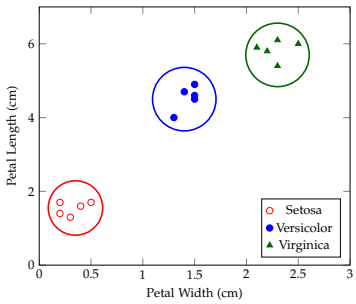


Figure 69: IRIS flowers data: Petal Length vs. Petal Width

There appear to be three clusters, as shown in Figure 69, consistent with the three species. But how can we design algorithms to identify these clusters? Bear in mind that any clustering method, as unsupervised learning, should be data driven and should aim to discover inherent patterns from rather than imposing a structure on the data.

Although clustering does not predict on the class label, we will use the classes of the IRIS data to help evaluate (at least visually) how

well a clustering method has performed. In the end, if each identified cluster can be related to one of the species, then the clustering process must have done something right with regard to the data.

Hierarchical Clustering

Hierarchical clustering is primarily used to create a tree structure of the data. Nonetheless, one can cut the tree at a specific level to obtain a desired number of clusters. As we discussed, there are two ways to construct a hierarchical structure: 1) the top-down divisive approach and 2) the bottom-up approach. We will first discuss a classic bottom-up method, namely Hierarchical Agglomerative Clustering (HAC), and come back to the top-down approach in the next section on K-means Partitioning.

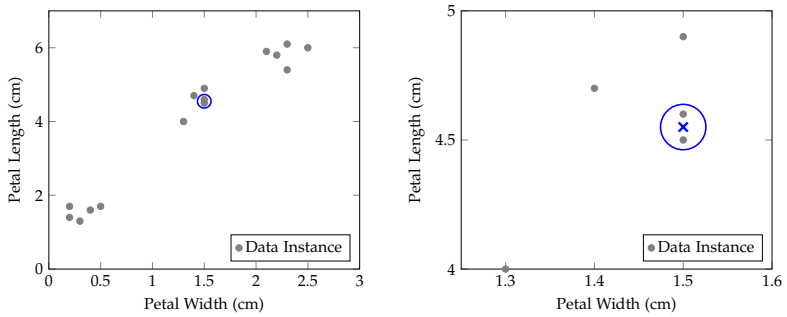
In the bottom up approach, we start with each data instance as a separate cluster and repeatedly merge the closest pair of clusters. Given N instances, the initial number of clusters will be N and this number will be reduced by 1 whenever two clusters are joined together as one. The process will continue until the number of clusters have been reduced to a (predefined) desired number k , or, if the objective is to build the entire hierarchical structure, until all clusters have been merged into one (the root).

This process is simple but we should clarify on two aspects before a demonstration with the IRIS data. First, how do we measure distances in order to determine which two are the *closest*? There are many distance and similarity functions one can use. Among others, we discuss the Euclidean distance as a commonly used measure in Chapter Binary Questions and cosine similarity for text analysis in Chapter . One should carefully evaluate how suitable a measure is with the data before adopting it.

Second, when two clusters are merged to become one, how do we represent the new cluster so that we can measure its distance (or similarity) to other clusters? A basic strategy is to compute the centroid of the new cluster using vector values of member data instances. And this is the strategy that we will use in the following demonstration as it is easy to visualize. We will discuss several other strategies after that.

Figure 70 (a) plots the IRIS data and visualizes the 1^{st} step of HAC clustering. Using Euclidean distance, the closest pair of instances can be identified, as circled on the plot, and joined as a new cluster.

Figure 70 (b) offers a closer look at the new cluster. Once the new cluster is formed, it is represented by the average values of its members, i.e. the geometric center of the two data instances on the plot (see $\times$ in the circled cluster). Now this new cluster with a centroid at

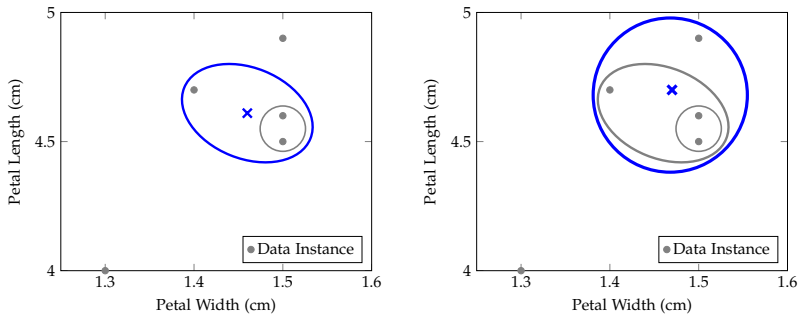


(a) all data instances (b) zoomed in on the closest pair

Figure 70: Step 1 of HAC clustering on IRIS data. The circle indicates a new cluster formed by the closest pair of instances. The centroid of the cluster is marked $\times$ in (b).

$\times$ is simply treated as another data instance that can be compared to and merged with others.

Now we can continue to find the next closest pairs. In Figure 71 (a), the 1^{st} merged cluster is found to be closest to yet another data instance, and they are joined to form a new cluster again. This continues to merge with another instance, as shown in Figure 71 (b).



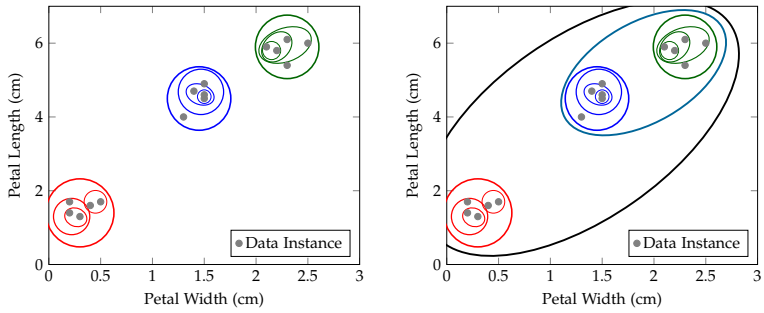
(a) step 2 (b) step 3

Figure 71: Further iterations of HAC clustering on IRIS data

Repeat the merging process, we can get results in Figure 72. If the desired number of clusters k is predefined, one can stop the iterative process once k clusters have been obtained. In Figure 72 (a), for example, the clustering process stops at 3 clusters ($k = 3$). If one desires to build the entire hierarchy of the data, as Figure 72 (b) shows, the process can continue until all clusters have been merged into the root ($k = 1$).

Issues of HAC

In the above demonstration, we have relied on the notion of *centroid* (with average values of cluster members) when we compute the distance between clusters. Regardless of the distance or similarity function being used, there are several alternative strategies for measuring intra-cluster distances or similarities.



(a) merged to 3 clusters (b) merged to 1 cluster

Figure 72: Result of HAC clustering on IRIS data

Average link, for example, compare each data instance in one cluster to another instance in the other cluster, and computes the average distance (or similarity) of all pair-wise distances.

Single link does the same pair-wise comparisons between members of two clusters, but singles out the closest or most similar pair. The shortest distance (or the greatest similarity) is then used as the distance (or similarity) of the two clusters. *Complete link*, on the other hand, identifies the farthest pair and uses the greatest distance (or the least similarity) as the distance (or similarity) of the two clusters.

Hierarchical Agglomerative Clustering (HAC) is not an efficient method and does not scale well with large data sets. Consider the amount of computation during each iteration of HAC. Given N data instances – which will later include merged clusters as pseudo-instances – it requires $N \times (N-1)/2$ pair-wise comparison to identify the closest pair, which is $O(N^2)$ complexity. When we factor in the number of iterations, which is proportional to $\log_2 N$ as each iteration reduces two clusters into one, the time complexity of the entire process is of $O(N^2 \log N)$.

Suppose we can manage to perform HAC on $N = 1000$ data instances, say in 1 minute. For $N = 10^6$ data instances, it will take $(\frac{10^6}{10^3})^2 \log \frac{10^6}{10^3} \approx 10,000,000$ minutes (about 19 years). On large volumes of data, HAC can be costly to perform and one has to find an alternative to hierarchical clustering in a cost-effective way.

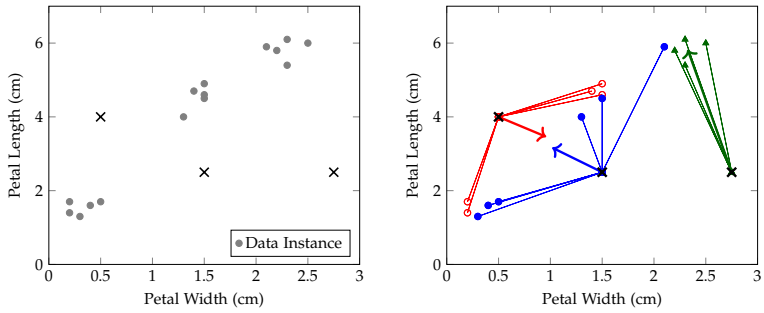
K-means Partitioning

We shall now look at *k-means*, a more efficient method for flat partitioning but, as we will discuss later, can also be used for hierarchical clustering as well. In the nutshell, k-means is a cluster optimization technique that improves on initial (random) clusters through iterations of cluster membership assignment and adjustment.

In the beginning when clusters are unknown, it picks a number of random cluster centroids and assigns every data instance to a

centroid depending on its proximity (similarity). Once all cluster membership has been determined, the centroid (center) of each cluster can be re-computed based on its assigned member instances. With the re-computed centroids, data instances will be reassigned, which will lead to further change of cluster centroids. So the process will continue on with centroid adjustment and cluster membership re-assignment, until the centroids no longer move (or move very little) or it has gone through a sufficient number of iterations.

Now let's look at how k-means may identify clusters from the IRIS data. Again, we set the number of desired clusters $k = 3$ to see whether it may produce clusters consistent with the three Iris species.

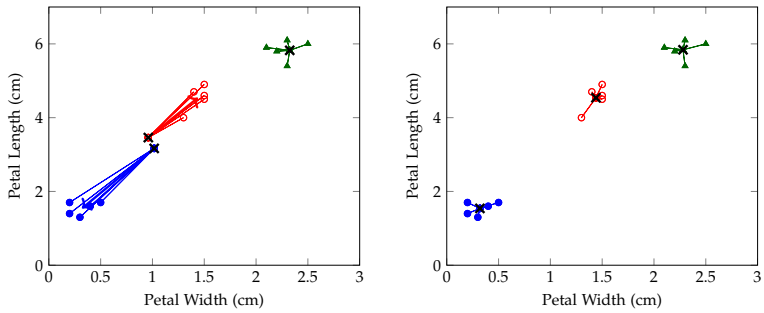


(a) initial (random) centroids (b) membership assignment

Figure 73: K-means initial steps. A $\times$ represents an initial centroid. Undirected lines represent cluster membership. Directed lines $\nearrow$ indicates how the centroid will move (change) after re-computation based on assigned members.

With data plotted in Figure 73 (a), we start with 3 random cluster centroids, shown as $\times$ in the plot. For each data instance, we compute its distance to each of the three initial centroids and identify the closest centroid, to which it will be assigned as a member. Figure 73 (b) shows lines connecting data instances to their corresponding closest centroids.

Once membership of each cluster is assigned, the center of cluster can be re-computed based on averaging member instances. After this computation, the centroids will likely change (move). Directed lines (arrows) in Figure 73 (b) represent the changes of cluster centroids.

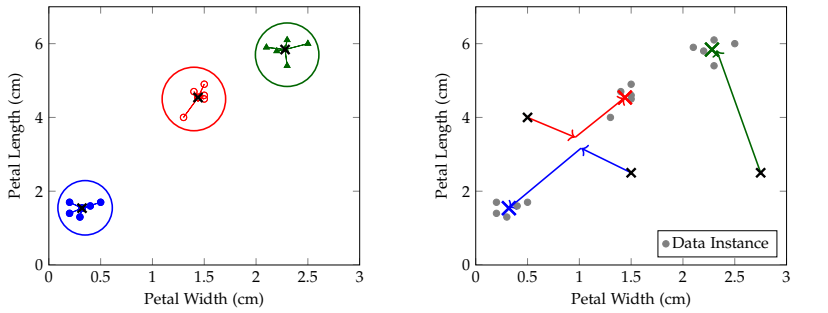


(a) 2^{nd} iteration (b) 3^{nd} iteration

Figure 74: K-means further steps. A $\times$ represents an centroid before it is updated. Undirected lines represent cluster membership. Directed lines $\nearrow$ indicates how the centroid will move (change) after re-computation based on assigned members.

With the new cluster centroids, the distance of each data instance has also changed and should be measured again to find out which

cluster one belongs to now. By doing so, some data instances may move from one cluster to another, as shown in the 2^{nd} iteration in Figure 74 (a), which, in turn, leads to further changes of cluster centroids. See the 3^{rd} iteration in Figure 74 (b). By repeating the iterations of cluster member assignment and centroid re-computation, the cluster centroids will continue to move and gradually converge. One may stop the k-means optimization process when cluster centroid does not move or moves very little, or after a certain number of iterations.



(a) Final result (b) Overview of centroid movement

Figure 75: K-means final clusters and steps

For the small IRIS data set here, it only takes less than 5 iterations for the cluster centroids to converge (with little further movement). Figure 75 (a) shows the final 3 clusters thus identified and they are consistent with the three species of Iris flowers in the data. Figure 75 (b) gives an overview of the movement of cluster centroids through the iterations. Even though we started with random centroids, k-means gradually moves the centroids toward more representative segments of the data and, in the end, reaches an optimal outcome.

Issues of K-means

K-means can be interpreted as to minimize member instances' mean squared distance from their centroid. The iterative process can always converge to an optimum, but there is no guarantee that it will be globally optimal. In each iteration, it compares k clusters to N data instances, which is $O(kN)$. Because k is usually much smaller than N, k-means is much more efficient and scalable than HAC.

Clustering is understood as unsupervised learning. Theoretically, it is data driven without human input. In practice, however, it is not a clear cut. There are many ways in which one can influence the outcome of clustering, making it somewhat supervised. With the k-means method, for example, one has to predefine the number of clusters to be generated, which makes the outcome rather arbitrary. The data at hand does not necessarily have that supposed k num-

ber of clusters. One may have to test and evaluate clustering with different numbers of clusters to find out what an ideal k may be.

The distance function also plays an important role in clustering. For text clustering, feature (term) distributions matter more than the absolute weights of terms. In that domain of applications, cosine (angle) similarity works better than Euclidean distance and like measures. Furthermore, even with the same distance function, the weights and scales of variables (features) make a major difference. If all variables are to be treated equally, then they should be normalized to the same scale, e.g. within $[0, 1]$ for positive values. Otherwise, they should be properly weighted by their discriminative power based on domain theory and/or statistics.

Probabilistic Clustering

While k-means discovers cluster centroids, the Euclidean distance, for example, essentially defines a cluster boundary as a sphere of a certain radius in a dimensional space, e.g. circles on two dimensions in Figure 75 (a). This implies two basic assumptions: 1) the distribution of member instances in a cluster is spherical, and 2) each data instance only belongs to one and only one cluster.

Is it possible to assume a different data distribution for clustering? In the same time, can there be soft (fuzzy) associations between data instances and clusters? The soft cluster membership can perhaps be thought of as probabilities.

Indeed, there is a general probabilistic approach to clustering, namely a model-based clustering. The term *model* means that data are assumed to follow a distribution function with certain parameters (model) to be discovered. So based on the model, one can estimate the probabilities of cluster membership and try to maximize the probabilities in the process of identifying the clusters.

For simpler illustration and explanation, let's first consider only one variable, *petal width*, in the IRIS data. Suppose data instances with petal width values v follows the normal distribution, based on which, as we discussed in Chapter Probabilistic Thinking and Modeling, the probability density of value v is computed by:

$$f(v) = f(v|\mu, \delta^2) \tag{46}$$

$$= \frac{1}{\sqrt{2\pi\delta^2}} e^{-\frac{(v-\mu)^2}{2\delta^2}} \tag{47}$$

In the context of clustering, however, we assume the same normal distribution *within* each cluster c. The above density function can be extended to include terms related to the cluster c and the probability

of a value v in cluster c can bey computed by:

$$P(v|c) \propto f_c(v) \quad (48)$$
$$= f(v|\mu_c, \delta_c^2) \quad (49)$$
$$= \frac{1}{\sqrt{2\pi\delta_c^2}} e^{-\frac{(v-\mu_c)^2}{2\delta_c^2}} \quad (50)$$

where μ_c and δ_c^2 are the mean and variance of values associated with cluster c, the only two parameters of the model for each cluster.

The above formula shows that, in order to compute related probabilities, we need to know parameter values, namely mean μ_c and variance δ_c^2, for each cluster c. The problem is, we are yet to identify these values, which, in fact, are based on related probabilities.

With the chicken or the egg dilemma, we shall start with some initial values for the μ_c and δ_c^2 so that we can feed them into the calculation of probabilities, which then help in the identification of better μ_c and δ_c^2 values. This is similar to k-means where we start with initial centroids, which, through membership assignment and reassignment, change and improve.

Initialization

With the IRIS data on only one dimension, i.e. petal width, we randomize μ_c and δ_c for $k = 3$ clusters: $\mu_1 = 1$ and $\delta_1 = 0.25$ for cluster c_1, $\mu_2 = 1.6$ and $\delta_2 = 0.25$ for cluster c_2, and $\mu_3 = 2.8$ and $\delta_1 = 0.25$ for cluster c_3. Figure 76 shows the distribution of IRIS data instances on the petal width v dimension, with a normal (bell) curve based on each set of initial parameters.

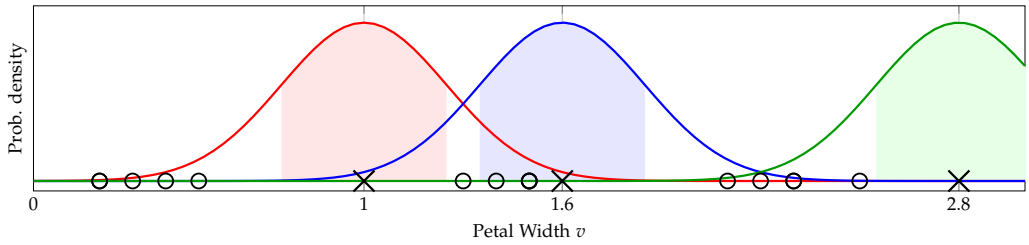


Figure 76: Initial parameters of normal distributions. ○ are IRIS data instances. × marks each an initial (random) mean μ (similar to a centroid), around which a normal distribution curve is plotted. The shaded area under curve shows the range of deviation from mean, i.e. $[\mu - \delta, \mu + \delta]$.

In addition, we don't know the prior probability $P(c)$ of each cluster c. If we assume equal chances, we have for the 3 clusters in the IRIS data: $P(c_1) = P(c_2) = P(c_3) = 1/3$.

Expectation

Given each cluster model with the initial parameter, we can estimate $P(v|c)$, how likely an instance v can be observed – generated from the

model – in that cluster c.

For now, however, we are more interested in finding the probability of an instance's membership association with a cluster. That is, given an instance v, how likely it is associated with each cluster c, $P(c|v)$. Following the Bayes Theorem, $P(c|v)$ can be estimated using $P(v|c)$:

$$\begin{aligned} P(c|v) &= \frac{P(v|c)P(c)}{P(v)} \\ &= \frac{f(v|\mu_c, \delta_c^2)P(c)}{\sum_{i=1}^{k} P(c_i) f(v|\mu_{c_i}, \delta_{c_i}^2)} \end{aligned} \tag{51}$$

where k is the number of clusters and the denominator is the sum of probabilities for the data instance v to be generated by distribution models of all k clusters.

Computing $P(c|v)$ is similar to the membership assignment of data instances to clusters in k-means. But instead of a *hard* membership, we have a probabilistic (soft) assignment here. We refer to the process of estimating the posterior probabilities of cluster membership as *Expectation*.

Let's apply this to the IRIS data. We use the first data instance with petal width $v = 0.2$ (cm) (see the $\circ$ closest to the zero in Figure 76). First, we compute $P(v|c)$, probability that each cluster model will generate the instance $v = 0.2$:

$$\begin{aligned} P(v = 0.2|c_1) &\propto f_{c_1}(v = 0.2) \\ &= f(v = 0.2|\mu_1 = 1, \delta_1 = 0.25) \\ &= \frac{1}{\sqrt{2 \times 0.25^2 \pi}} e^{-\frac{(0.2-1)^2}{2 \times 0.25^2}} \\ &= 0.00954 \\ P(v = 0.2|c_2) &\propto 2.5 \times 10^{-7} \\ P(v = 0.2|c_3) &\propto 5.2 \times 10^{-24} \end{aligned}$$

As you can see from the Figure 76, $v = 0.2$ is on the far left side of the first (red) bell curve centered on 1, and has a very small probability density $P(v = 0.2|c_1) \propto 0.00954$. This is further remove from the centers of the second (blue) and third (green) normal curves, and probability densities corresponding to c_2 and c_3 are much smaller.

With these values, we can now compute $P(v)$, which is the weighted sum of probabilities that the 3 cluster models will generate data $v = 0.2$:

$$\begin{aligned} P(v=0.2) &= \sum_{i=1}^{3} P(c_i) f(v=0.2|\mu_{c_i}, \delta_{c_i}^2) \\ &= 1/3 \times 0.00954 + 1/3 \times 2.5 \times 10^{-7} + 1/3 \times 5.2 \times 10^{-24} \\ &= 0.00318 \end{aligned}$$

Plug these values into Equation 51, we can compute the probability of data instance $v = 0.2$ belonging to each cluster for the *Expectation* step:

$$\begin{aligned} P(c_1|v=0.2) &= \frac{P(v=0.2|c_1)P(c_1)}{P(v=0.2)} \\ &= \frac{0.00954 \times 1/3}{0.00318} \\ &\approx 1 \\ P(c_2|v=0.2) &= 2.59 \times 10^{-5} \\ P(c_3|v=0.2) &= 5.46 \times 10^{-22} \end{aligned}$$

Apparently, data instance $v = 0.2$ is much more likely to belong to the cluster c_1 (the red bell curve on the left) than to the other clusters, which are far away away as shown in Figure 76.

Now that we have identified the membership probabilities for the first instance, we shall continue to do the same for all other data instances ($N = 15$ for the IRIS data) and conclude the *Expectation* step with all probabilistic membership "assignment." For example, here are related probabilities for two more data instances in IRIS: $v = 0.4$ (2^{nd} instance) and $v = 2.3$ (15^{th} instance).

$$\begin{aligned} P(c_1|v=0.4) &\approx 1 \\ P(c_2|v=0.4) &= 0.000\,177 \\ P(c_3|v=0.4) &= 1.7 \times 10^{-19} \\ P(c_1|v=1.3) &= 8.66 \times 10^{-6} \\ P(c_2|v=1.3) &= 0.128 \\ P(c_3|v=1.3) &= 0.872 \end{aligned}$$

Figure 77 shows the color-coded result of the first *Expectation* step on the IRIS data. A data instance's color indicates the cluster with the highest probability – that is, the bell curve that best "covers" the data instance.

Maximization

The objective of clustering is to find clusters such that the overall probability of observed data instances is maximized. That is to maxi-

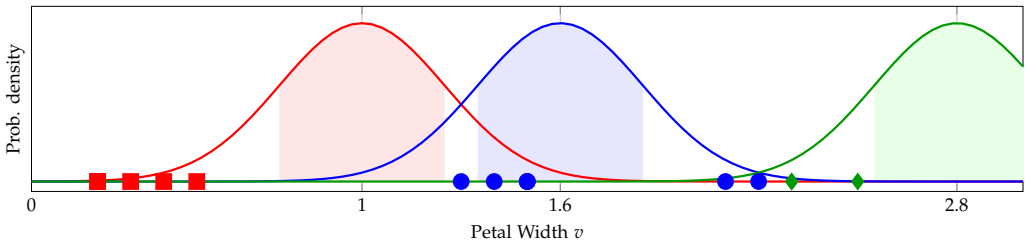


Figure 77: First expectation result. Color of a data instance indicates the cluster membership with the highest probability.

mize the joint probability for all data instances, which is the product of all $P(v|c)$ if we assume statistical independence:

$$\arg\max_{\theta_c} \prod_v P(v|c) \tag{52}$$

where θ are parameters of the distribution models and, with the normal distribution assumption, are μ and δ^2 of each cluster distribution. This is equivalent to maximizing the log-likelihood:

$$\log \prod_v P(v|c) = \log \prod_v f(v|\mu_c, \delta_c^2) \tag{53}$$

$$= \sum_v \log f(v|\mu_c, \delta_c^2) \tag{54}$$

The parameters can be estimated by:

$$\hat{\mu_c} = \frac{\sum_v vP(c|v)}{\sum_v P(c|v)} \tag{55}$$

$$\hat{\delta_c^2} = \frac{\sum_v (v - \hat{\mu_c})^2 P(c|v)}{\sum_v P(c|v)} \tag{56}$$

where v is each one of N data instances ($N = 15$ in the small IRIS data subset here) and $P(c|v)$ is the probability that v is associated with cluster c.

It turns out that there is not a simple direct solution to maximizing the log-likelihood. However, if we look at the result from the *Expectation* step, there is a way in which we can maximize this by following the data and improving the *representative* of the cluster models, similar to how k-means iterates.

Now that data instances have been assigned, probabilistically, to each cluster, the original model for each cluster is no longer representative of data associated with them. In Figure 77, for example, the first (red) bell curve is not centered on the data instances most highly associated with it.

This is similar to k-means where the original centroid does no longer represent the center of member instances. And the models

– with parameters mean μ_c and variance δ_c^2 for each cluster c – will have to be recomputed. This re-computation of parameters, in the context of model-based clustering, is the *Maximization* step.

Now back to the IRIS data. With the assigned posterior probabilities $P(c|v)$, we can now update the mean and variance for each cluster model. For cluster c_1 and $N = 15$ data instances, we have:

$$\begin{aligned}
\hat{\mu_1} &= \frac{\sum_v vP(c_1|v)}{\sum_v P(c_1|v)} \\
&= \frac{0.2 \times 1 + 0.4 \times 1 + .. + 1.3 \times 8.66 \times 10^{-5}}{1 + 1 + .. + 8.66 \times 10^{-5}} \\
&= 0.522 \\
\hat{\delta_1} &= \sqrt{\frac{\sum_v (v - \hat{\mu_1})^2 P(c_1|v)}{\sum_v P(c_1|v)}} \\
&= \sqrt{\frac{(0.2 - 0.522)^2 \times 1 + (0.4 - 0.522)^2 \times 1 + .. + (1.3 - 0.522)^2 \times 8.66 \times 10^{-5}}{1 + 1 + .. + 8.66 \times 10^{-5}}} \\
&= 0.433
\end{aligned}$$

Likewise, we can update models for cluster c_2 with $\hat{\mu_2} = 1.67$ and $\hat{\delta_2} = 0.332$, and cluster c_3 with $\hat{\mu_3} = 2.34$ and $\hat{\delta_3} = 0.116$.

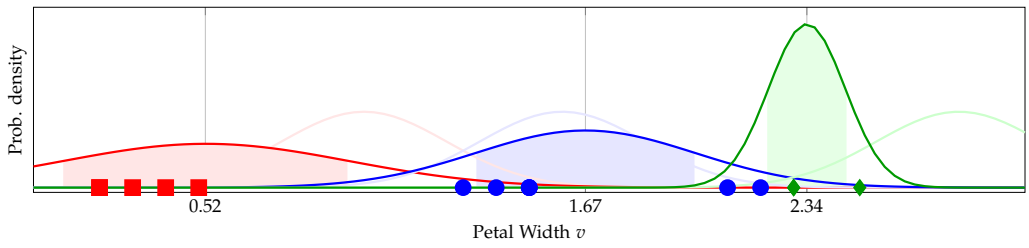


Figure 78: First Maximization result. A bell curve with with a shaded area ($\pm delta$) is the new model with updated parameters for the cluster.

Figure 78 shows the result of this *Maximization* step. The models (normal curves) have moved closer to their more highly associated data instances. This is very similar to re-computation of centroids – thus causing them to move – in k-means clustering. With Maximization, more than one parameters are needed to specify a model. For normal distribution models, both mean μ and variance δ^2 are to be updated. If one assumes a different probability distribution, then a different set of parameters will be maximized.

Iterations of Expection-Maximization

Now that the cluster models have changed, we need to go back to the *Expectation* step and re-compute the posterior probabilities $P(c|v)$. Once the probabilities are changed, the model parameters will be updated again in the next *Maximization* step. The process

of Expectation-Maximization will repeat itself until the models no longer changed, where the overall probability of models generating observed data instances has been maximized.

Figure 79 shows further iterations of EM on the IRIS data. Visually, the cluster models (curves) move increasingly closer to the data instances they represent and, in the end, get well aligned with them.

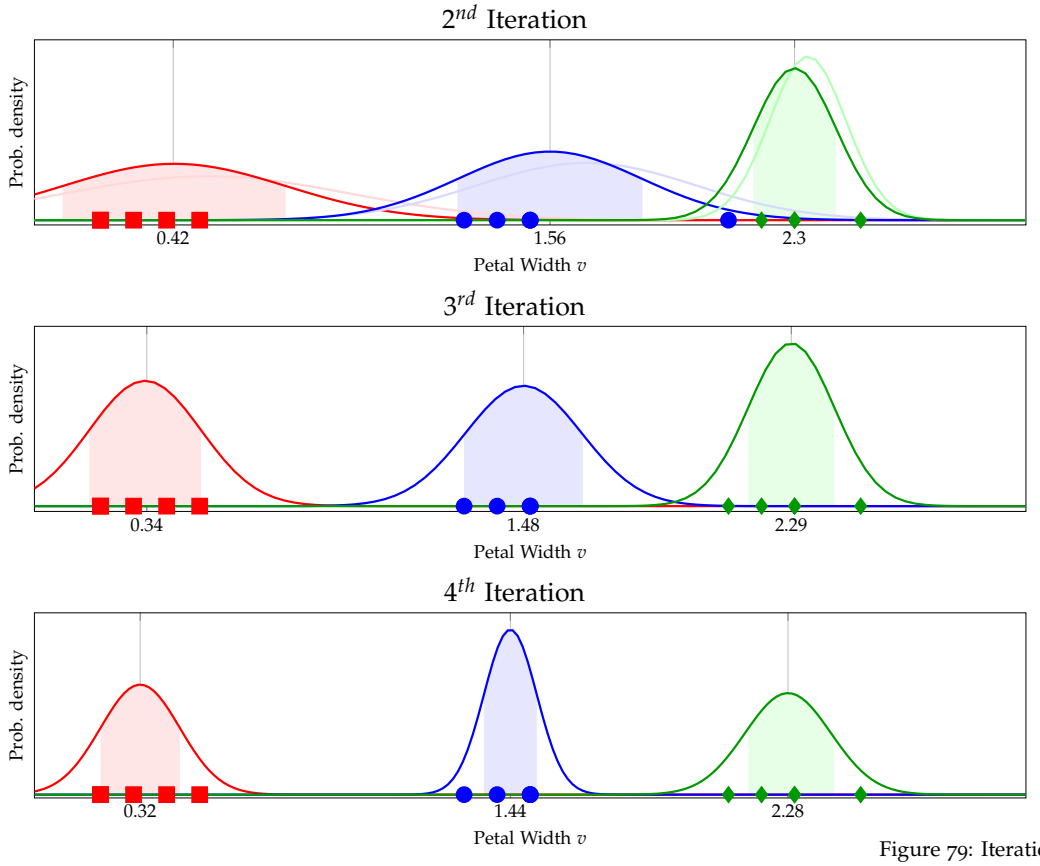


Figure 79: Iterations of Expectation and Maximization. Note that there are 5 data instances in each iris species. But because their petal width values overlap on the only dimension, there appear to be 3 or 4 instances in each cluster, which, in fact, has 5.

EM with Higher Dimensionality

Our discussion has been focused on the it Expectation-Maximization method on one dimension with the univariate normal distribution. In real-world applications, however, there is hardly any clustering problem that can be resolved with this simplified version.

Normally EM has to take into account multiple, if not thousdands or even millions of, variables. Instead of univariate distributions, one will have to rely on multivariate distributions. For example, rather than using univariate normal distributions with means and variances, one will instead rely on co-variances in a mixture Gaussian model.

In addition, the normal (Gaussian) distribution is only one of many distribution functions that can be used with EM clustering.

In text clustering application, for example, it is very common to use poisson distributions. Regardless of the assumed distribution, the idea and process of expectation and maximization for the optimization of log-likelihood of observed data remain the same.

Search and Retrieval

> Ask, and it shall be given you;
> seek, and you shall find;
> knock, and it shall be opened
> to you.
>
> *The Gospel of Matthew*

We are in constant pursuit of information, knowledge, and truth. We have questions and we ask. Today, we ask questions and seek information with the help of technologies. We search for information and web pages by Googling. We pose questions to digital assistants such as Amazon's Alexa. Regardless of differences in the specifics, the essence of these interactions is to find the right (relevant) piece of information that can satisfy us (the user) – be it a web page, a short answer, a prediction or some kind of recommendation.

This is about search and the general area of research is commonly referred to as *Information Retrieval*. In 1951 – when searches for information were primarily conducted in the libraries with the assistance of reference librarians without a computer – Calvin Mooers coined the term *Information Retrieval* as "the investigation of information description and specification for search and techniques for search operations."[39] A library catalog with index cards was the state of the art to help people seek and find quickly.

[39] Calvin N. Mooers. Zatocoding applied to mechanical organization of knowledge. *American Documentation*, 2(1):20–32

With the support of computers, we can perform the same type of search tasks even more efficiently, yet with ease on the user's side and sophistication on the system's side. Now that computers are essential to the various search systems we have in place, the notion of *Information Retrieval* (IR) may be redefined, in a narrower sense, as:[40]

> finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)

[40] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008

With this narrower definition, a basic task of IR is, given a user query, to find *relevant* documents from a collection of text materials. Traditionally, the data collection such as a library of books is

assumed to be static, and certain pre-computation and indexing can be conducted before querying processing. In reality, this assumption does rarely hold as data can grow to include new materials which in turn should be incorporated into the index.

Web search engines are *Information Retrieval* (IR) systems and the Web is a huge collection of data that constantly grow and change. The magnitude, dynamics, and decentralization of data on the Web pose great challenges, which have led to waves of innovation on Web IR[41]. We will discuss some of these challenges and related ideas later in the chapter.

[41] First came Lycos and Yahoo, which borrowed traditional text indexing and reference catalog/directory techniques from libraries. AltaVista enhanced the technologies with a larger index and a responsive interface. Google and Baidu came after them and revolutionized search ranking by using *citation analysis*, another method that had been studied extensively in library science. Now very few people know AltaVista, not to mention Lycos. Looking forward, what will come in the next ten or twenty years? Will Google remain relevant then.

Basic Paradigm

In information retrieval, the basic problem is to search and find matched documents against a query representation based on a user's information need. Documents need to be stored in proper representation so that related matches can be computed (numerically). An index is also necessary to conduct the matching process timely.

As shown in Figure 80, the user with an information need issues a query to the retrieval system, which in turn match it against the document representation in the store to identify relevant documents in the search result. The result can also be ranked (sorted) in terms of predicted relevance of each document.

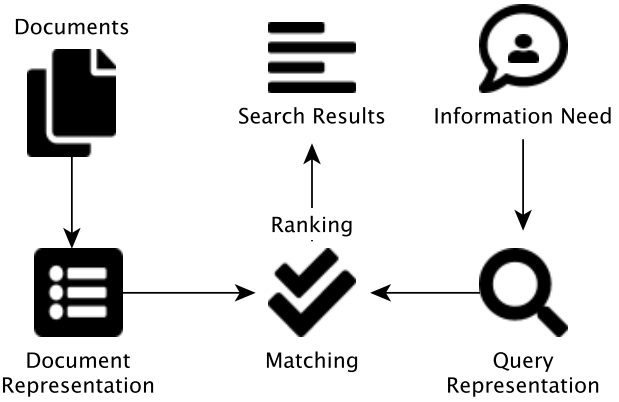


Figure 80: Basic paradigm of an *Information Retrieval* system

The notion of *relevance* is key in information retrieval research. Yet it remains one of the least understood concepts in the field. Whether a piece of information is relevant depends on many factors such as the user's objectives and the context of the current task. Sometimes, these may be difficult for the user to articulate; and even if she does, the query expression thus constructed may hardly be precise.

Retrieval systems have to attack the issue of uncertainty often

associated with understanding the user's need and what is truly relevant in context. We can factor this in when building a retrieval model. With probabilistic models, for example, the degree of uncertainty can be quantified for proper probabilistic ranking in light of probability and information theories.

However, uncertainty is often due to lack of information (data) and requires further data collection. Techniques that support relevance feedback and interactive, exploratory searching may help engage the user in order to understand more precisely what she is looking for.

Indexing

For now, let's assume the basic model presented in Figure 80 and walk through some of the major ideas for finding matched (and potentially relevant) documents.

In Figure 80, the data collection can be very large with a great number of documents, e.g. a library of a million books. In order to conduct searches with matching and ranking responsively, some amount of data pre-processing is necessary. This is also practically sensible if the collection is relatively static (no dramatic changes frequently).

In chapter Text and Human Language, we discuss basic processes for text vectorization. The result of vectorizing a text collection can be represented by a matrix with documents as rows and terms as columns. Each document here is a vector of values corresponding to terms in the dictionary. These values can be obtained using methods such as binary, term frequency (TF), or TF*IDF term weighting scheme.

Assume we start with the binary representation, which is very simple with 0s and 1s, for a collection of $1,000$ documents each containing a dozen words. In terms of the Heaps' law, we probably obtain somewhere close to $1,000$ unique terms, leading to a large matrix in the order of million values. With an even larger collection, the entire matrix representation can become too expensive to be stored and computed.[42]

Now if we examine actual values of each document (row) in the matrix, we will quickly realize that most values are 0 (zero). This is due to the fact that even though we have a larger number of unique terms in the entire collection – thus many columns in the matrix – each document (row) only contains a small number of these terms. So instead of storing the entire matrix with all values, we can use a *Sparse Matrix* representation where only non-zero values are stored.

On the document level, this means each document is a *Sparse Vector* that only stores data about what terms actually appear in

[42] On the benchmark Reuters RCV1 text collection, the observed Heaps' function indicates that the vocabulary size M, i.e. the number of unique terms, is proportional to $T^{0.5}$, where T is the total number of tokens. Given that T is about linearly associated with the number of documents N, the number of values in the matrix representation is $N \cdot M$, which is $\propto N^{1.5}$.

it. With the sparse representation, we can significantly reduce the amount of storage needed for document representation (i.e. better space efficiency).

However, the use of sparse vectors does not necessarily address the issue of query processing efficiency. That is, even with document sparse vectors, it still requires lots of computational effort on the system side to sift through all documents to find matches for a query. The amount of time to perform this matching task increases linearly with the increasing number of the documents. For searching a very large data collection such as the Web, this will become too slow to serve users on the fly.

Efficient query processing requires a better data structure that supports: 1) fast lookup of individual query terms with their associated documents, and 2) fast merging of all terms in the query expression. The 1st objective can be achieved if the terms are sorted where a binary search can be performed. Each of the terms on the sorted list can then point to the list of documents (using their IDs) containing it. We refer to such a sorted structure of terms with references to documents as an *Inverted Index*. Each term is now a vector of document IDs, inverted from the original document vectors of terms.

When presented with a single-term query, the system can conduct a binary search to quickly identify all documents matching (containing) the term from the inverted index. With a query expression of multiple terms, each term can be looked up in the same manner.

But how can we merge documents associated with the individual terms to find out which documents actually satisfy the entire query, where, either explicitly or implicitly, terms are used in a logical context with AND, OR, and/or NOT?

To achieve the 2nd objective of fast merging, documents for each term in the inverted index should also be sorted (by their IDs). So in the end when an inverted index is built, all the terms will be sorted, each of which in turn contains a list of sorted document IDs. In the next section, we will look at a couple of methods to see how we can take advantage of the sorted lists to merge query terms for document-query matching.

Matching

One who is interested in reading about the U.S. revolution may issue a query "*The Declaration of Independence*" on a collection of historical artifacts to find that particular document. If we assume the system treats words *the* and *of* as stop-words, they will be disregarded in the query processing. In this case, the query has two terms, namely *Declaration* and *Independence*, which can be further interpreted as

Declaration AND *Independence* assuming the user is looking for both terms.

Given an inverted index with a sorted list of all unique terms in the collection, it takes binary search logarithmic time to find each term's entry. That is, given M terms in the index, the system makes $O(\log M)$ comparisons to find *Declaration* and *Independence* in the worse case. This is logarithmic time complexity is rather efficient and scalable. An example of the two term entries found in the inverted index may look like this:

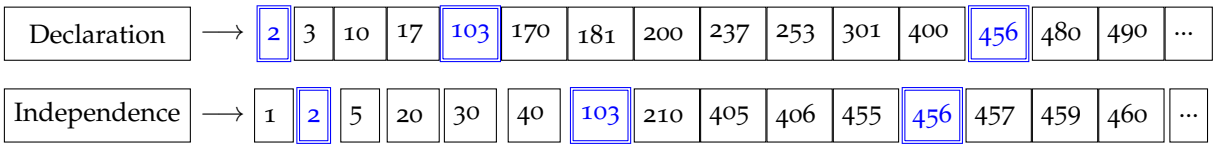


Figure 81: Two postings from the inverted index. Highlighted document IDs with double border indicate matched documents that satisfy *Declaration* AND *Independence*.

We now refer to each term's list of documents – in fact only their IDs – as the *posting* of the term. An inverted index is thus a collection of *postings*. After examining the two postings, we can conclude that documents #2, #103, and #456 in Figure 81 contain both terms. But how can an algorithm merge the two postings to identify these matched documents?

A very simple approach is to go through the documents IDs in the posting for term *Declaration* and, for each of them, we compare it with every document ID in the second posting for term *Independence*. Suppose the first posting has n_1 documents and the second has n_2, this will conduct $n_1 \cdot n_2$ comparisons. This approach is going to be very slow with large postings and it has yet to take advantage of the sorting (pre-computation) that has been performed on the document IDs.

Apparently, we can improve on the above brute force method is by conducting a binary search on the second posting for each document on the first posting. In this case, the time complexity of the matching process will become $O(n_1 \log n_2)$ as the binary search is of $O(\log n_2)$ complexity. Assume $n_2 > n_1$ is a large number, this will significantly reduce query processing time.

So far we have used one posting as the primary to iterate through and, in the process, compare each value there to those in the second posting. In this approach, the two postings play unequal roles in the matching process.

A different approach is to take the postings equally and iterate through them simultaneously. Assuming document IDs are sorted in the ascending order, we start at the beginning of each posting. In each step, we compare the current values of the two postings and

the one with a small value will move to the next (greater) value. This continues until it has been found a value greater or equal to the current value of the other posting. In that case, we switch to the other posting to go next.

For the above example, we start at first document ID in each posting shown in Figure 82, where the *Declaration* posting has a value 2 and the *Independence* posting has 1 (see step 1 in the figure). As the one with a smaller document ID, the *Independence* posting will move to the next document ID, which is 2 and matches the the current value in *Declaration* (see step 2).

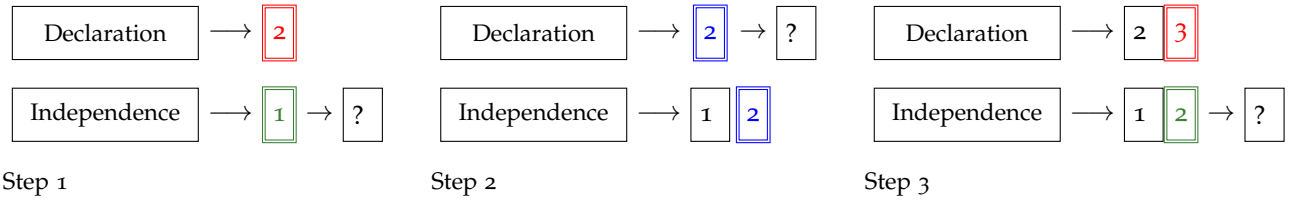


Figure 82: Simultaneous walking to merge postings, starting at the first document ID. The posting with the smaller value will move to the next ID.

Now that the second posting (*Independence*) is already with an equal or greater document ID, the process will shift to the first posting (*Declaration*) to catch up, which, as shown in Figure 82 step 3, then goes to document #3 and switches the turn back as it surpasses 2. This process will continue until no more matches can possibly be found.

In the worse case scenario, it iterates through the two postings from start to end, which is $O(n_1 + n_2)$ time complexity. Practically, this is better than the previous method with $O(n_1 \log n_2)$ complexity where $\log n_2$ may remain a large number ($\gg 2$).

These techniques are particularly useful to merge postings for logical AND, i.e. for document containing both or all query terms. They also work well with the OR query expressions, where documents containing any of the terms should be brought together after the removal of duplicate IDs.

The simultaneous iterative process can be further improved with skip pointers that the iterations faster by skipping a significant amount of document IDs in each posting. However, the skip pointers technique is of little use for OR queries as it skips documents that should be included with a logical OR.

Ranking

With techniques presented in the previous section, we know how to process a query with an inverted index and identify matched documents by merging postings for the query terms. This can be

conducted rather efficiently if proper preprocessing is already in place, e.g. with sorting and skip pointers.

If the user can articulate her need for information with relevant query terms in a precise boolean expression, then the system should be able to serve the user's need via matching. There is potentially a long list of documents that match the query. In the old days, this perhaps worked just fine for scholars who would like to find a list of research publications for a comprehensive literature review.

In the context of today's World Wide Web and online social media, it will not be helpful by retrieving an extremely long list of pages and presenting them all at once to the user. For one, the query expression of any ordinary user is most likely imprecise and a lot of matched documents will turn out to be non-relevant.

More important, many information seekers on the web are not looking for comprehensive literature. Often they are in need of the one page, if not a few, that can satisfy their need or at least something they can start with. In these cases, the challenge is not about the thoroughness of the result, but how precise and relevant the result is.

The common practice of today's web search engines is to present a ranked list of matched documents where, ideally, the most relevant documents will appear early on so the user can see them and follow with a clickthrough. So in addition to query matching, document ranking (sorting) in the retrieved result is essential for the success of a search engine.

Content-based Ranking

So in what order shall we rank documents or pages according to a query? Among the matched ones, what documents should be presented to the user early in the result?

As the key to *information retrieval* (IR) is relevance, the first reaction to the questions here is to rank documents according to how *relevant* they are to the query. But how do we quantify relevance? Without the user's subjective feedback on what is relevant vs. what is not, there is simply no data to quantify relevance. The alternative is to assume that documents that match the user's query expression to a higher degree are more relevant to her information need.

Based on this assumption, there are several approaches we can take. For example, using the *Term Frequency* (TF) scheme discussed in chapter Text and Human Language, we can count the total number of query term occurrences in each document. The matching score of a document d with regard to query q can be therefore computed by:

$$score(q,d) = \sum_{t \in q \cap d} TF_{t,d} \quad (1)$$

where t is each term appearing in both the query and the document, and $TF_{t,d}$ is the term frequency of t in d, i.e. the number of times it appears in the document.

Now one would argue that this scoring function will not work well as it give equal importance to each query term. With this method, one who issues a query like "the declaration" may retrieve a bunch of documents with lots of "the" in the writing (suppose "the" has not been removed as a stop-word from being indexed). We may therefore consider the relevance of individual terms by replacing the TF with TF-IDF weights (Term Frequency * Inverse Document Frequency):

$$score(q,d) = \sum_{t \in q \cap d} TF_{t,d} \cdot IDF_t \quad (2)$$

where IDF_t is IDF weight of term t.

Both TF and TF-IDF scoring functions above are dependent on the terms' occurrences. Apparently, longer documents (e.g. books) are more likely to contain more instances of the same terms than the shorter documents. To remedy for this bias, one can normalize term frequency by the document length:

$$score(q,d) = \sum_{t \in q \cap d} \frac{TF_{t,d}}{L_d} \cdot IDF_t \quad (3)$$

where L_d is the length of document d, i.e. the number of terms in the documents including duplicates. Now that we factor in document length, short documents will have *equal* chances to be ranked and long documents will no longer dominate[43]. After all, for any document d, $\sum_t \frac{TF_{t,d}}{L_d} = 1$ regardless of its length. This is after the fact that the overall probability of *any term* drawn from the document is always 1.

The document-length-normalized TF is a a probability estimate. Imagine if we put all the terms of document d in a black box and randomly draw a term from it, the likelihood of getting term t can be estimated by $\frac{TF_{t,d}}{L_d}$. The length-normalized scoring function here is essentially modeled on the probability of each query term's occurrence in the document.

Like document-length normalized TF, IDF is also based on a probability estimate. As discussed in chapter Text and Human Language, a term's IDF weight is also a measure based on the term's likelihood

[43] An alternative to document length normalization is to use *cosine similarity* instead of a sum of term weights. Cosine is focused on the angle proximity (vector direction) in a vector space, which is essentially about terms' distributions rather than their absolute occurrences. See related discussions in chapter Text and Human Language.

to appear in any document drawn randomly from the entire data collection. The component $\frac{n_t}{N}$ in the IDF function, where N is the total number of documents and n_t the number of documents containing term t, is to estimate the term's collection-wide probability. The logarithmic transformation:

$$IDF_t = -\log \frac{n_t}{N} \quad (4)$$

$$= \log \frac{N}{n_t} \quad (5)$$

does convert dramatically varied probability values into a more comparable scale and dampens their differences. However, this formulation is not accidental but has in fact a concrete theoretical root in the information theory. IDF_t can be derived from measuring the amount of *relative entropy* (KL divergence) when term t is observed in a document from the collection.[44]

In light of uncertainties involved in understanding user needs to quantify relevance, one may also think of ranking documents in terms of the *probability of relevance*. Indeed, early probabilistic IR research followed the *Probability Ranking Principle* (PRP), which states that documents should be ranked in order of the probability of relevance, or how likely documents will be useful to the user.[45] The principle can be operationalized by estimating those probability from data, e.g. data on term distributions and user relevance feedback.

Based on PRP and a logarithmic transformation of the Bayes rule, one can model the odds of a document's relevance given a query and derive a weighting function similar to IDF. This is known as the *Binary Independence Model* (BIM), where the notion of relevance is binary (either relevant or not relevant)[46] and documents are assumed to be independent of one another [47]. The relevance weight w_t^{RSJ}[48] of a term t in BIM is computed by:

$$w_t^{RSJ} = \log \frac{(r_t + 0.5)(N - R - n_t + r_t + 0.5)}{(n_t - r_t + 0.5)(R - r_t + 0.5)} \quad (6)$$

where R is the number of relevant documents, n_t is document frequency of term t, and r_t is the number of *relevant* ones in the n_t documents. The weight approximates IDF when both the number of relevant documents r_t is much fewer n_t, which in turn is much smaller compared to the entire collection N. That is, $w_t^{RSJ} \approx IDF_t$ when $r_t \ll n_t \ll N$.

Further development of this probabilistic model led to another well-known method called *BM25*[49], where a term's weight is computed by:

[44] Akiko Aizawa. An information-theoretic perspective of tf-idf measures. *Information Processing and Management*, 39(1):45–65, January 2003

[45] Stephen Robertson. The probability ranking principle in IR. *Journal of Documentation*, 33:294–304, 12 1977

[46] This is not to confuse with the fact that the probability (of relevance) is still a continuous measure, not a binary one. In other words, even though the final answer is either yes (1) or no (0), the probability of yes vs. no remains a continuous value between 0 and 1.

[47] Stephen Robertson and Karen Sparck-Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27:129–146, 05 1976

[48] RSJ stands for Robertson and Spark-Jones, the authors of the weighting scheme.

[49] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, April 2009

$$w_{t,d}^{BM25} = \frac{TF_{t,d}}{k_1(1-b) + b\frac{L_d}{\bar{L}} + TF_{t,d}} \cdot w_t^{RSJ} \quad (7)$$

where $\bar{L}$ is the average document length in the collection, and k_1 and b are parameters to be optimized (tuned) through experiments. b adjusts the degree of document normalization, with full normalization at $b = 1$ and no normalization at $b = 0$. This highly resembles TF*IDF, where w_t^{RSJ} corresponds to IDF_t and the rest of the formula is $TF_{t,d}$ with document-length normalization. The scoring (ranking) function for BM25 is then the sum of term weights:

$$score(q,d) = \sum_{t \in q \cap d} w_{t,d}^{BM25} \quad (8)$$

Popularity-based Ranking

With document ranking methods such as those based on TF*IDF and BM25, an *information retrieval* system will be able to sort documents according to their match scores, especially in terms of important keywords in the user query. This worked well in the traditional library context, where documents were in fact selected publications such as books and articles. Because of the editorial processes these publications went through, the contents of the documents were trustworthy in general. Ranking based on document content is a sensible solution in that setting.

When the World Wide Web started to grow in the 1990s, the first generation of web search engines adopted the same technologies developed in the libraries and continued to rely on text contents in their ranking. It soon became clear that this is easily susceptible to spamming.

The Web is a decentralized space for publication without a central editorial process. Everyone is a publisher who writes anything he wants, in his own way of expression. A ranking method based on TF*IDF, for example, is impacted by TF and IDF. You probably can exert very little influence on the IDF part as it is a *global* statistic of the entire web collection. However, if you want to be ranked number one for queries like "The Declaration of Independence," you simply repeat those keywords in your page contents.

By simple repetition in the text, you increase the terms' frequencies (TFs) and therefore the ranking of your page to related queries. The point here is that page contents on the web are not necessarily trustworthy and we cannot simply rely on the *claims* of authors in this rather decentralized, autonomous environment.

Without a central authority that we can trust on individual pages, we may perhaps consider input from their *peers* – those who know these pages and what they have to say about them. On the web, pages reference one another via *hyperlinks*. Figure 83 shows an example of a web page, from which the user can click on a label (text), a button, or an image to visit another page.

Figure 83: Page references via hyperlinks. Page A links to several other pages, including B and C. The user can click on a description (anchor text), a button, or anything else such as an image to follow a link.

With hyperlinks, we pages form a directed network on which structural analysis can be performed. And when a text label (i.e. anchor text) is provided as part of the hyperlink, it provides useful data about what the target page is about.

In Figure 83, when page A links to page B, descriptions or keywords page A uses on the hyperlink can be considered for the representation of page B. Assuming the pages are on different web sites with separate ownership, then this piece of anchor text represents an independent evaluation of page B's contents. By aggregating anchor text data from all other pages linking to page B, its inverted index can be adjusted with the crowd-sourced data, which are potentially more reliable than self-claimed page contents.[50]

Based on the classic BM25, BM25F is an example of related methods that go beyond the body text of a web page to include other structural elements (fields) and relevant data streams (particularly anchor text data).[51] Among others, the use of anchor text data for indexing and matching is a common practice of web search engines.

When the user issues a search query, the question remains what is more *relevant* and should be ranked higher in the result. For web searches, *relevance* does not merely mean *on-topic* because a very large number of pages will be considered on topic in terms of their contents (anchor text included). The notion of *relevance* also entails certain degree of *importance* while on topic.

Importance, however, can be subjective and hard to quantify. To borrow an idea from *scientometrics*, we can instead substitute *importance* with *impact* or *popularity*. While in scholarly communication scientific research is evaluated based on the amount of citations, web pages can be treated likewise in terms of hyperlinks.

If we regard each hyperlink as a citation to the target page, we can now measure a page's impact or popularity by counting the number of incoming links, i.e. *in-degree*. That is, the impact score R_d of a target document (page) d is computed by:

$$r_d = \sum_{p \in P_d} 1 \tag{9}$$

where P_d is the collection of pages linking to d. For each linking page p, the contribution is considered equal with 1. In Figure 84, for

[50] Anchor text can also be hacked and abused to boost one's search engine ranking. There has been known practice of self- or mutual-linking from sites set up by the same group of people. On the other hand, websites can team up to *bomb* a target with off-topic anchor text in order to "promote" its ranking for those keywords, e.g. for political activism.

[51] M. Taylor S. Saria H. Zaragoza, N. Craswell and S. E. Robertson. Microsoft cambridge at trec 2004: Web and hard track. In *The Thirteenth Text Retrieval Conference (TREC 2004)*, pages 1–7. NIST Special Publication, 2004

example, page t has two in-links from pages p_1 and p_2 and its impact score is therefore 2.

But are all hyperlinks created equal? Let's consider an extreme case where we have more data for the two linking pages in Figure 84. Suppose p_1 is an ordinary page that is linked to by another page but links to many others (including page d). On the other hand, p_2 is a very popular page that is referenced by many pages but has chosen to link to only one page (i.e. page d). Figure 85 illustrates this scenario.

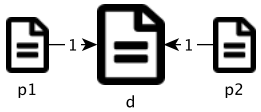


Figure 84: Two links being equal. Unless we have more data about these links, we assume they make equal contributions to measuring the target page's impact.

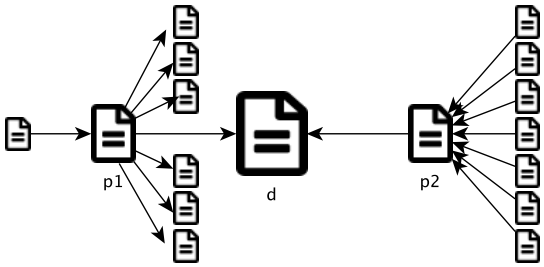


Figure 85: Two links being different. When we know more about the pages linking to the target page, we can no longer assume their contributions are equal.

Based on the above hypothetical scenario, it is apparent that the links from p_1 and p_2 to the page t are not equal. For one, p_2 is much more popular and has more impact than p_1 given the number of in-links it receives. Moreover, it gives out the only one link to page d, making its contribution (endorsement) even stronger. On the contrary, p_1 receives very little (from only one page) and gives out a lot, further diluting its already small share.

In reality, p_2 can be the page of an influential figure like the Pope who receives lots of media attention. If he chooses to post only one link and that goes to your page, you must be thrilled. We can also think of p_1 as the total opposite, a page of yours or mine with a long list of notes and references. The longer the list goes, the less significant each reference may become.

This discussion sheds lights on two important aspects for scoring a page's impact or popularity. First, it matters who gives the link and how impactful the *link giver* is. A link from an important figure should carry more weight in itself. Second, it also matters how many links the giver *gives out*. When one gives out to more recipients, the share of each link is diluted and the weight should be divided by the number of out-links. Mathematically, the second aspect can be easily quantified by counting the out-links for each page in P.

On the first point, however, the solution does not appear straightforward. Again in Figure 85, the impact of document d depends on the impact of linking pages p_1 and p_2, which in turn depend on the ones that link to them. And this will go on. How can we proceed to

solve this problem of *chicken and egg* without going into a loop? But the solution is indeed a loop, or a recursive process.

PageRank

The *directed graph* of web pages pointing to one another can be thought of as a city of one-way streets (hyperlinks) you can follow and locations (pages) that you can visit via these streets. If you want to simply explore the city, you can randomly pick a street to get where it leads to. Then from there, you can pick a random street again and continue on.

Imagine if you continue this over and over again, ultimately you may be able to reach every corner of the city (if every location is connected). Over time, you will be able to visit some locations for many times. And chances are, you will visit some locations more often than others – in other words, some locations are more *popular* than others on your tireless, random tour of the city.

So which ones will be visited more frequently by chance? Apparently, those with more incoming streets, which in turn have more incoming streets, and so on. And this appears to match the problem we had earlier, where we wanted to calculate the impact or popularity score of a page, which depends on the impact of linking pages, which in turn depends on their linking pages, and so on.

In this thinking, the impact or popularity score of a web document (similar to a location in the above imaginary city) is equivalent to its *chance* (frequency) of being visited on a random walk over time – that is, its *long-term visit rate*.

This is essentially the *Random Surfer* (random "walker" on the web) model proposed for the Google's original *PageRank*[52] function.[53] In this model, the computer simulates a user following the link structure of the web after pages have been collected (crawled) and parsed. It starts at any page and, at each step, goes out along one (randomly chosen) of the links on page to reach another. Ultimately, each page will have a long-term visit rate if this simulation is conducted with sufficient time (to reach the steady state).

There is one problem, however. Some pages do not have out-links – they are dead ends where the surfer will get stuck. To avoid this issue, the surfer should be allowed to jump (*teleport*) out of the current page to visit another random page. Also for pages with few or no in-links to be visited a bit more frequently – so that they are no totally disregarded in the ranking model – the surfer will *teleport* periodically regardless of out-links. The probability associated with this periodic teleporting is referred to as the *damping factor*. With this model, the impact score r_d of a page d is equivalent to the long-term

[52] While PageRank is indeed about ranking *pages* on the web, the name is in fact after its primary author and Google co-founder, Larry Page.

[53] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120

visit rate, which is computed by:

$$r_d = \alpha \sum_{p \in P_d} \frac{r_p}{n_p} + \beta \quad (10)$$

where n_p is the out-degree (number of out-links) of page p, which links to d and is having a current score of r_p. On the right side of the equation, the first part $\alpha \sum r_p / n_p$ is the score one receives from in-links and the term β represents the amount each page is given "for free." Both α and β can be regarded as parameters related to the damping factor.

For example, with a periodic teleporting probability of 0.15, we can set $\alpha = 1 - 0.15 = 0.85$ for the earned portion (from links) and $\beta = 0.15\frac{1}{N}$ for the "free" portion, where N is the total number of documents (pages) in the collection.[54] In this way, β represents the chance each page will be visited with teleporting.

[54] The original PageRank paper used 0.15 for the damping factor based on earlier experimental results.

With equation 10, one needs to follow the recursive approach of a random surfer to calculate the ultimate PageRank scores for all pages. Because we do not know these values (i.e. the r_p values), which are needed to start the computation (i.e. to calculate r_d), we will kick off with arbitrary initial values, for example, by setting every page's score to $1/N$ – that is, they are treated equally likely to be visited before we proceed to find out the actual values. In each iteration, we update the all pages' scores using equation 10 until there is very little or no change at all to the scores.

In the end, the PageRank score of a page corresponds to the probability that the page will be visited by the random surfer. The more prominent a page is in the web graph, the more likely (frequently) it will be visited and therefore will have a higher PageRank score. PageRank scores are query independent as they are only determined by the network structure. When a search engine ranks pages with regard to a specific user query, results should first limited to those meeting query criteria before PageRanks are combined with query-related weights such as those based on TF*IDF.

Although the basic version of PageRank computes its scores regardless of a query, one can easily modify equation 10 to make it query-dependent. For example, the β does not have to be a universal constant for all the pages. If we assume each document (page) can have its individual β_d value, then the PageRank function becomes:

$$r_d = \alpha \sum_{p \in P_d} \frac{r_p}{n_p} + \beta_d \quad (11)$$

where we may assign different β_d to pages based on, for example, other indicators about their relevance to a search query.

PageRank, which is essentially a *centrality* measure in network analysis, is an extension of computing eigenvector and Katz centrality.[55] The Hyperlink-Induced Topic Search algorithm, or HITs, can be regarded as a further extension of PageRank.[56] While PageRank only considers contributions from in-links, HITS gives credit to pages that point to others (out-links) as well. With *hub* (centrality based on out-links) and *authority* (centrality based on in-links), the algorithm propagates the scores in a similar iterative process until stabilized.

With HITS, the analysis is performed on a sub-collection including pages relevant to a query (text-wise) as well as other pages with in-links from and/or out-links to the relevant ones. This setup makes HITS query-dependent and the relevance to a query is reinforced through the propagation of *hubs* and *authorities*.

[55] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010

[56] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, sep 1999

Information Filtering

Information Filtering (IF) looks at the same information seeking problem from the opposite angle of IR – instead of reaching out to find information, IF is to eliminate the non-relevant so that the user only receives the relevant. IR and IF are considered to be "two sides of the same coin."[57]

Information filtering usually operates on an ongoing information stream such as emails. Look at how many junk emails or spams[58] you receive daily and you see the importance of having a good spam filter. Filtering for recommendation, e.g. on news and movies, is another popular application of information filtering.

One major challenge in information filtering is the information stream that constantly feeds and changes in its content. Patterns that you have discovered from past spam emails, for example, may turn out to be useless for future spam filtering as spammers learn and change their behavior. News that you used to follow closely may become irrelevant in the next news cycle or as your interests move on. It requires constant learning of the user and the environment to adapt to the ongoing changes.

[57] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Commun. ACM*, 35(12):29–38, December 1992

[58] We refer to relevant emails as *hams* and the irrelevant as *spams*.

Questions

I do want to stress the key advantage AND disadvantage of normalization, which is to treat terms of different forms as the same.

1) There are situations in which the different variations are indeed about the same and should be treated as one term. This way, users who uses one variation in the query may be able to find documents containing terms in another.

2) But there are also situations where normalization is an over-

treatment and can lead to results that are in fact not relevant (false positives). There are plenty of examples in both 1) and 2). In specific domains, there might be more in 1) vs. 2), or vise versa. By examining your domain data and use cases, you may find whether it is better to do certain normalization or not (e.g. casefolding, but not stemming).

Now, we have increasingly more storage and computing power at a cheaper price. So often we can do more with these technologies. For example, it is common to have multiple indices even for the same data, e.g. one with certain normalization and one without. Then at search time, you combine evidence from all indices to final matching and ranking of results.

We know terms have different discriminative powers – that some terms are more useful than the others. We know stopwords (highly frequent terms) are generally less useful. What about rare terms, i.e., terms that are used in a very tiny portion of documents? What about terms in between (neither rare or too frequent)? If you have to choose a limited number of terms to index your documents, what kind of terms would you pick?

In the vector space model, cosine similarity is often used to measure how close (or distant) two vectors are? For example, you can compare a query (vector) with each document (vector), and rank them according to their cosine similarities. Given that cosine is about the angle (vector directions) and disregards vector magnitude (length), why is vector direction more important than magnitude here (for term-based text representation)?

Your analogue is good in that both "generators" are in a dynamic process of generating something, which can trap a crawler in their space (in both worlds).

The generator here in the context of the web is a general reference to dynamic web pages that generate page contents based on different given parameters.

An example of this is the "next page" link to help users navigate to additional lists of information. An ordinary user may click on "next", and "next", ... for a few iterations.

But a spider, without sufficient intelligence and design, may go on and on by simply "clicking" on (following) the "next page" link nonstop... The notion of "trap" goes both ways – it traps the crawler on the site gathering data on and on; but it also leads to intensive load on the generator (as there appear to be lots of clicks and page loads to the web server). Years ago when I was a graduate student, I developed a crawler with an aim to collect all web pages of the entire university (where I was studying) and, the first night I ran it, it was able to collect 2 million+ pages with a cost. The crawler was

trapped by another college/school's site with a page "generator" and, in the end, brought the entire server down... You can imagine what happened the next morning when the IT manager found out... And it did take me some time to settle the issue as as a student. ;)

Good question.

For website or pages that no one knows and links to, crawlers won't be able to collect them. A crawler starts with seed websites (those already known) and, ultimately, will get to those where there is a path. But there are "islands" to which there is no path. There are also pages that can be reached from certain parts of the web and cannot be reached either unless you have seeds to crawl from specific locations...

There is always a part of the web where crawlers cannot penetrate. We generally refer to this as the deep web. There are private sites, sites with access control, and sites that can be interacted with but too difficult for crawler to collect... they represent a space much much greater than the regular web covered by search engines.

Exercise

Assume we have a tiny collection of two documents with the following terms: Doc 1: apple, pear, dessert, cream, cook, work, berry Doc 2: work, computer, disk, apple, network, system Given a free text query "apple computer," rank the documents using Dice's coefficient. Please provide the formula and list calculation steps.

Which ones (more than one) of the following statements about Euclidean distance are true? It measures that distance between the end points of two vectors. It measures that angle between two vectors. Euclidean distance is affected by vector magnitudes. Euclidean distance is zero when two vectors point to the same direction.

Which ones (more than one) of the following statements about Cosine similarity are true? It measures that distance between the end points of two vectors. It measures that angle between two vectors. Cosine is affected by vector magnitudes. Cosine is 1 (max value) when two vectors point to the same direction.

Does vector length normalization (dividing a vector by its length/magnitude) have any impact on Cosine ranking?

Structural Analysis

Does It Really Work?

Imagine you are asked to design an algorithm to help detect credit card fraud. Suppose you are told that your algorithm will be tested for its accuracy and, as a way to motivate you, your boss promises a bonus based on its accuracy. Accuracy, as it turns out, can be roughly defined as the fraction of answers that are correct. In the context of credit card fraud detection, one can produce a confusion matrix by cross-examining predictions (model) vs. real outcomes (ground truth):

	Predicted Positive $\hat{\oplus}$	Predicted Negative $\hat{\ominus}$
Actual Positive $\oplus$	True Positive (TP)	False Negative (FN)
Actual Negative $\ominus$	False Positive (FP)	True Negative (TN)

Table 20: A confusion matrix

Based on the counts in Table 20, accuracy can be computed by:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

which is again the total number of correct answers (true positive and true negative) divided by the total number of answers.

Given the evaluation based on accuracy, how would you do to maximize your reward? Surely you can develop a very sophisticated algorithm with massive data and effort. There is, nonetheless, a simple response to this challenge. If you look at statistics on credit cards, fraud transactions represent only a very small fraction. Suppose the likelihood of a fraud is 10 out of every 1000. One may simply classify all 1000 transactions as non-fraud and the confusion matrix will become:

	Predicted Fraud	Predicted Non-Fraud
Actual Fraud	0	10
Actual Non-Fraud	0	990

Table 21: Confusion matrix based on the zero-effort solution, by classifying all transactions to non-fraud.

And the accuracy is:

$$Accuracy = \frac{0+990}{1000} \quad (2)$$
$$= 0.99 \quad (3)$$

This solution requires virtually zero effort but its accuracy appears to be pretty good: 99%. There are several reasons why an overall accuracy can be rather misleading, if not completely useless here. For one, the evaluation metric ignores the data distribution among class labels (e.g. fraud vs. non-fraud) and does not account for potential *chance* accuracy. In the above example, it is very easy to cheat and get rewarded just *by chance*. Here you simply favor the dominant class (non-fraud) to boost accuracy.

Second, an evaluation metric should align properly with the purpose of the assigned task. For credit card fraud detection, one can argue that missing a *fraud* transaction is much more costly and undesirable than misclassifying a normal transaction as fraud. In other words, one should be motivated to identify most if not *all* fraud transactions; and the result should be heavily penalized if real frauds are identified as non-fraud (false negative). We shall look into these issues to see what alternatives may offer.

Precision

Given a skewed distribution of data in the classes, it may be useful to evaluate accuracy within each predicted class. This is equivalent to a metric commonly known as *precision*, which is the fraction of those predicted positive to a class that actually belong to that class. In terms of the confusion matrix Table 20, precision is computed by:

$$Precision = \frac{TP}{TP+FP} \quad (4)$$

When both TP and FP are zero, precision is 1 (100%) by definition. Based on the numbers in Table 21, we can compute precision for those predicted to be fraud (in the first column of numbers):

$$Precision(fraud) = \frac{0}{0+0}$$
$$= 1$$

This does not seem to resolve the issue with accuracy, to which precision is very similar. This is due to the second reason we discussed, that there should be a greater emphasis and penalty on the *false negative*. Now instead of looking at the columns to compute precision, we can examine the rows and evaluate the same result from a very different perspective.

Recall

Recall, a classic evaluation metric, refers to the fraction of actual positive that have been classified positive. In the example here, it is the fraction of real fraud transactions that have been identified as such. Recall can be computed in terms of the confusion matrix Table 20:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

where false negative (FN) is part of the denominator. Applying this to data in Table 21, we get a recall of:

$$\begin{aligned} Recall(fraud) &= \frac{0}{0+10} \\ &= 0 \end{aligned}$$

By focusing on true positive vs. false negative, recall gives a better indication than precision on the poor performance in fraud detection. Precision and recall are often used together in evaluation. It is attempting to compute an average score of the two so there is only one score to represent the overall performance. The arithmetic mean of precision and recall here is $(1+0)/2 = 0.5$, which still appears too good for a zero-effort solution.

Arithmetic mean gives equal weights to both metrics and is not very indicative of the overall performance. We suggest that such an "averaging" measure show an alarming score if one of the scores is extremely low – that is, we will regard a method as bad if it performs miserably in one metric. One function that leans more heavily on the lower score is the F measure of precision and recall:

$$F_\beta = (1+\beta^2)\frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} \tag{6}$$

where β value can be adjusted in favor of precision or recall. When $\beta = 1$, this becomes the F_1 measure, i.e. the *harmonic mean*:

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{7}$$

With $Precision = 1$ and $Recall = 0$ in the above example, the harmonic mean is $F_1 = 0$ – here, the one zero score (recall in this case) completely dominates and the average suffers.

In applications where a model can be adjusted or tuned, we can evaluate the model based on more than one pairs of precision and recall scores. A common practice is to plot the precision-recall

curve. Figure 86 shows precision-recall curves of two methods, where method 1 appears to perform better as its curve is closer to the top-right (higher precision and recall).

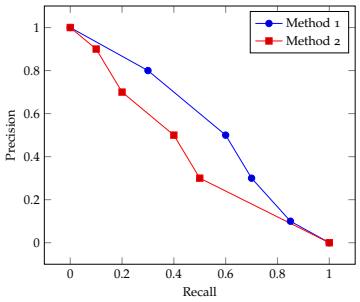


Figure 86: Precision-recall curve

Sensitivity and Specificity

Precision and recall are both focused on the fraction of true positive. Recall is also referred to as the *true positive rate* (TPR), sensitivity, or the probability of detection. It measures how *sensitive* a method is in identifying the actual positives.

Conversely, one can also measure the *true negative rate* (TNR) or the fraction of actual negatives that have been correctly identified as such. This is often referred to as *specificity*, which is computed by:

$$Specificity = \frac{TN}{TN + FP} \tag{8}$$

Sensitivity and specificity are concepts relative to the class in question. Sensitivity for the *fraud* class is specificity for the *non-fraud* class; and vise versa.

The false negative rate (FNR) is complementary to specificity or the true negative rate (TNR), i.e. $FNR = 1 - TNR$, and is often referred to as fall-out or *probability of false alarm*. Similar to precision vs. recall, sensitivity and fall-out (i.e. 1 - specificity) are often plotted to produce an *ROC*[59] curve.

ROC analysis has been widely used and well studied for model performance assessment in many fields including defense and medicine. It not only enables the evaluation of model effectiveness in terms of true positive and true negative rates, but can also help identify best practice with cost constraints.

Figure 87 illustrates a plot of an ROC curve, where the diagonal line (dashed) defines the acceptable performance. To understand what the diagonal line means, imagine we create a random model that takes a 50-50 guess during classification. As a result of this random guessing, there will be an equal split between the predicted positives $\hat{\oplus}$ and predicted negatives $\hat{\ominus}$. Hence, column values of the same row in Table 20 will be identical:

[59] ROC stands for *receiver operating characteristic* and was initially used in the military to evaluate the performance of a radar detector.

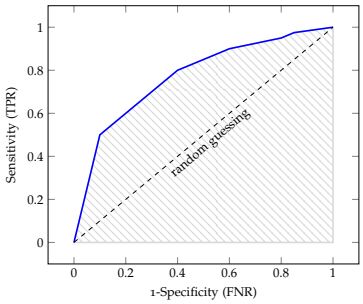


Figure 87: ROC curve and area under curve (AUC)

$$\begin{aligned} N_{\hat{\oplus}} &= N_{\hat{\ominus}} \\ TP &= FN \\ FP &= TN \end{aligned}$$

where $N_{\hat{\oplus}}$ and $N_{\hat{\ominus}}$ are the numbers of predicted positives and negatives respectively. It follows that $TPR = FNR$, i.e. $Sensitivity =$

$1 - Specificity$ or $Sensitivity + Specificity = 1$. This equation defines the diagonal line in Figure 87 and is the result of random guessing.

Below the diagonal line $Sensitivity + Specificity = 1$, the model performs worse than random and is undesirable. The farther the ROC curve is above the diagonal line, the better the performance with a greater area under curve. Therefore, the *area under curve* (AUC), i.e. the shaded area in the plot, is a single score that can be computed to measure the overall performance in terms of ROC.

Kappa and Chance Agreement

So far the metrics are focused on different aspects of the confusion matrix in evaluation. However, we are yet to address one important problem in assessing the agreement between the predicted and the ground truth – chance agreement. A method may hit the correct answer simply by chance. The evaluation cannot be fairly conducted unless we find a way to discount the chance agreement.

Let's first look at some of the chances or probabilities. In binary classification, for example, the outcome is either positive (e.g. fraud) or negative (e.g. non-fraud). Based on the confusion matrix Table 20, the likelihood of a positive prediction vs. a negative prediction can be estimated:

$$\begin{aligned} P(\hat{\oplus}) &= \frac{N_{\hat{\oplus}}}{N} \\ &= \frac{TP + FP}{TP + FN + FP + TN} \qquad (9) \\ P(\hat{\ominus}) &= \frac{N_{\hat{\ominus}}}{N} \\ &= \frac{TN + FN}{TP + FN + FP + TN} \qquad (10) \end{aligned}$$

where $N_{\hat{\oplus}}$ and $N_{\hat{\ominus}}$ are the number of predicted positive and negative answers respectively. N is the total number of instances. $TP + FP$ is the sum of the *Predicted Positive* $\hat{\oplus}$ column where as $TN + FN$ is the sum of the *Predicted Negative* $\hat{\ominus}$ column in Table 20. These are probabilities of model predictions.

Likewise, we can estimate probabilities of having a positive vs. negative outcome in the ground truth:

$$\begin{aligned} P(\oplus) &= \frac{N_\oplus}{N} \\ &= \frac{TP+FN}{TP+FN+FP+TN} \qquad (11) \\ P(\ominus) &= \frac{N_\ominus}{N} \\ &= \frac{TN+FP}{TP+FN+FP+TN} \qquad (12) \end{aligned}$$

where $N_\oplus$ and $N_\ominus$ are the numbers of actual positive and negative results respectively. $TP + FN$ is the sum of the *Actual Positive* row and $TN + FP$ is the sum of the *Actual Negative* row in Table 20. These are probabilities based on the ground truth.

With these probability, it is now possible to estimate chance agreement. If we assume the model and ground truth are independent, then the (hypothetical) probabilities that they will reach the same conclusion, namely the joint probability of both saying "yes" (positive) and the joint probability of both saying "no" (negative) can be computed by:

$$P_e(\oplus \cap \hat{\oplus}) = P(\oplus)P(\hat{\oplus}) \qquad (13)$$

$$P_e(\ominus \cap \hat{\ominus}) = P(\ominus)P(\hat{\ominus}) \qquad (14)$$

The overall chance agreement, the probability of either both "yes" or both "no" is:

$$\begin{aligned} P_e &= P_e[(\oplus \cap \hat{\oplus}) \cup (\ominus \cap \hat{\ominus})] \\ &= P_e(\oplus \cap \hat{\oplus}) + P_e(\oplus \cap \hat{\oplus}) \\ &= P(\oplus)P(\hat{\oplus}) + P(\ominus)P(\hat{\ominus}) \qquad (15) \end{aligned}$$

Note that we add the two probabilities to compute the OR ($\cup$) probability because *positive* and *negative* are mutually exclusive.

Now that we know the chance agreement P_e, we can use it to discount the overall agreement P_o:

$$\kappa = \frac{P_o - P_e}{1 - P_e} \qquad (16)$$

where P_o is the relative agreement of two, e.g. two raters such as a model and an expert (the ground truth), and is identical to the accuracy metric in our discussion $P_o = Accuracy$. This is commonly known as the *Kappa* coefficient, a statistic that measures the agreement of two sets of results.

As Equation 16 shows, Kappa takes into account potential agreement by chance and discounts the overall coefficient by the chance agreement. This makes it harder for a model to "cheat" on the result by simply favoring the dominant classes (e.g. classifying all transactions into non-fraud).

With the result in Table 21, the chance agreement P_e is:

$$\begin{aligned} P_e &= P(\oplus)P(\hat{\oplus}) + P(\ominus)P(\hat{\ominus}) \\ &= \frac{10}{1000} \times 0 + \frac{990}{1000} \times \frac{1000}{1000} \\ &= 0.99 \end{aligned}$$

By discounting this chance agreement, we get a *Kappa* coefficient of zero:

$$\begin{aligned} \kappa &= \frac{P_o - P_e}{1 - P_e} \\ &= \frac{0.99 - 0.99}{1 - 0.99} \\ &= 0 \end{aligned}$$

A kappa score of 0 does not mean there is no agreement. There is in fact an agreement between the model prediction and the ground truth. However, this agreement is purely due to chance and is reduced to 0 after discounted for the chance agreement.

Indeed, $\kappa = 0$ means completely chance agreement. Its minimum value, $\kappa = -1$, indicates total disagreement whereas a maximum value $\kappa = 1$ shows the two are in full agreement even after accounting for chances.

Averaging

When evaluating model performance on multiple classes – even binary classification entails two classes – there are two major strategies to compute an overall score (average).

Suppose precision is the evaluation metric, one can construct a confusion matrix for each class c and compute its precision by:

$$P_c = \frac{TP_c}{TP_c + FP_c} \tag{17}$$

And the overall (average) precision can be computed by:

$$\bar{P}_{macro} = \frac{\sum_{c \in C} P_c}{|C|} \tag{18}$$

where C is the set of unique classes and $|C|$ is the size of the set or the number of classes, e.g. 2 for binary classification. This is *macro averaging*, which computes the average based on class-level scores. It gives equal treatment to each class and disregard their relative sizes (significance).

The alternative is micro averaging which takes into account the composition of each classes and produces a final average weighted by class sizes. With micro averaging, all related counts are summed up before a ratio is taken. For example, average precision can be computed by:

$$\bar{P}_{micro} = \frac{\sum_{c \in C} TP_c}{\sum_{c \in C} TP_c + FP_c} \tag{19}$$

This turns out to be the overall fraction of predictions that are correct – in fact, a micro averaged precision is identical to *accuracy* in binary classification.

Whether one should use macro or micro averaging depends on the relative importance of classes in the final assessment. If all classes are equally important regardless of their sizes, macro-averaging should be preferred. If one aims to emphasize the absolute number of correct predictions, then micro-averaging is appropriate.

Rank Evaluation

There are applications where the focus is not about definitive correct vs. incorrect answer. In information retrieval, for example, a model may return a sorted list of search results. Such a rank list requires a different approach to evaluation.

Surely one can regard a rank list as the set of correct answers and use the discussed metrics such as precision and recall. More precisely, we can create a cut-off at a certain position k in the rank and use the top k as the set to be evaluated. For example, precision at $k = 10$ is useful to evaluate early results in web searches, where the emphasis is often on the first result page (top 10).

If one knows the total number of relevant document r (recall) for a query, the cut-off can be set at position r to measure it precision. This is referred to as *R-Precision* because at this particular position precision is equal to recall. It is also possible to measure average precision by starting with the top position $k = 1$ and increasing it until recall reaches 100%.

Several other metrics can be derived from variations of these strategies. However, they only attack the problem of rank list evaluation by indirectly addressing positions in the list.

A direct attack on this issue is to assign a *different score* (amount of reward) for a correct (relevant) answer in a *different position*. A relevant answer at the top should be highly rewarded, which should be discounted down the rank.

This is the idea behind the *Normalized Discounted Cumulative Gain*, or nDCG, which has been widely used in the information retrieval community since its proposal.[60] Given the relevance rel_i[61] of an answer at position i, the Discounted Cumulative Gain at position k can be computed by:

$$DCG_k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{20}$$

$$= rel_i + \sum_{i=2}^{k} \frac{rel_i}{\log_2(i+1)} \tag{21}$$

where a relevant (correct) answer at position 1 will receive its full credit and others will be discounted by their positions down the rank, with $\log_2(i+1)$.

Among all possible arrangements of the rank list for a query, there exists at least one ideal (perfect) list in which answers are sorted by their relative relevance scores in the descending order. This produces the best possible DCG score, which we call the ideal DCG or IDCG. Comparing a DCG score against its corresponding ideal score, we can obtain the *normalized* DCG or nDCG:

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{22}$$

Normalized by the ideal score, an nDCG score is always between $[0, 1]$, with 1 indicating a perfect ranking and 0 no relevant answer at all in the list[62]. nDCG is a popular metric for web search evaluation and one can adjust the position k to emphasize relevant results at the very top, e.g. with $nDCG_3$.

Evaluation of Numeric Predictions

When the outcome of a model is a on a continuous scale rather than discrete values, the evaluation is no longer about *correct* answers because it is rarely possible to make predictions that are *exactly* the actual outcome. Instead, the focus should be on how closely predictions are to the actual values or how much apart they are (errors).

[60] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002

[61] Relevance scores can be on any scale dictated by the evaluation protocol. For example, they can be as simple as 1 for a relevant (correct) answer and 0 if otherwise.

[62] Here assume relevance scores are non-negative, e.g. 0 no reward for non-relevance. In certain applications, it might be reasonable to assign negative scores for non-relevant answers to penalize a ranking with any non-relevant results.

Error Metrics

Many evaluation metrics for numeric predictions are indeed based on the error terms. The mean absolute error (MAE), mean squared error (MSE), and root mean-squared error (RMSE), for example, compute mean errors in their absolute, squared, and root squared forms:

$$MAE = \frac{\sum_{i=1}^{N} |\hat{v}_i - v_i|}{N} \quad (23)$$

$$MSE = \frac{\sum_{i=1}^{N} (\hat{v}_i - v_i)^2}{N} \quad (24)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (\hat{v}_i - v_i)^2}{N}} \quad (25)$$

where N is the total number of instances to test the numeric predictions, $\hat{v}_i$ is the predicted value on the i^{th} instance, and v_i is the actual value of the i^{th} instance. Relatively speaking, metrics based on squared errors such as MSE and RMSE are amplify the impact of large errors and are more sensitive to outliers.

Each of these error metrics can be normalized by the corresponding error of a pseudo-model predicting the mean value of training data, to obtain its *relative* error. For example, relative squared error (RSE) can be computed by:

$$RSE = \frac{\sum_{i=1}^{N} (\hat{v}_i - v_i)^2}{\sum_{i=1}^{N} (\bar{\mu} - v_i)^2} \quad (26)$$

where $\bar{\mu}$ is the estimated mean based on training data. Root relative squared error (RRSE) and relative absolute error (RAE) can be established in the like manner.

Correlation

While RSE measures the deviation of predicted values $\hat{v}_i$ from actual values v_i, standard deviations of predicted and actual values can be computed by:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (v_i - \mu)^2}{N-1}} \quad (27)$$

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{N} (\hat{v}_i - \hat{\mu})^2}{N-1}} \quad (28)$$

where μ is the mean of actual values and $\hat{\mu}$ the mean of predicted values. The covariance between the predicted and actual is:

$$cov(\hat{v}, v) = \frac{\sum_{i=1}^{N}(\hat{v}_i - \hat{\mu})(v_i - \mu)}{N-1} \quad (29)$$

By normalizing the covariance by the product of both standard deviations (also a value of variance), we can obtain the Pearson correlation between the predicted and actual values:

$$cor(\hat{v}, v) = \frac{cov(\hat{v}, v)}{\hat{\sigma}\sigma} \quad (30)$$

$$= \frac{\sum_{i=1}^{N}(\hat{v}_i - \hat{\mu})(v_i - \mu)}{\sqrt{\sum_{i=1}^{N}(v_i - \mu)^2}\sqrt{\sum_{i=1}^{N}(\hat{v}_i - \hat{\mu})^2}} \quad (31)$$

Pearson correlation measures the similarity in their variances from means and is equivalent to cosine of v and $\hat{v}$ as vectors from their means. Figure 88 illustrates the connection using a 2-dimensional space. To simplify the discussion, suppose there are only $N = 2$ instances on which the actual outcomes are (v_1, v_2) and predicted values are $(\hat{v_1}, \hat{v_2})$, with means μ and $\hat{\mu}$ respectively.

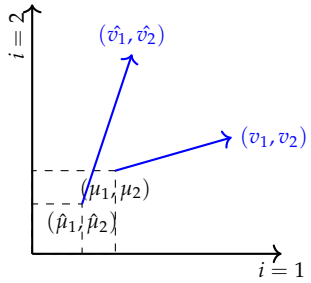


Figure 88: Pearson correlation as cosine of vectors originated from means, with $N = 2$ for a 2-dimensional visualization

As shown in Figure 88, the actual values can be regarded as a vector pointing in the $\vec{v} = (v_1 - \mu, v_2 - \mu)$ direction, whereas the predicted is represented by the vector $\vec{\hat{v}} = (\hat{v_1} - \hat{\mu}, \hat{v_2} - \hat{\mu})$. In this representation, Pearson correlation is identical to the cosine of the two vectors, i.e. $cov(\hat{v}, v) = cosine(\vec{\hat{v}}, \vec{v})$. Chapter Text and Human Language has more on the cosine similarity.

Because of this connection, Pearson correlation shares important characteristics with cosine. It reaches its maximum value of 1 when errors from their mean, i.e. $\hat{v}_i - \hat{\mu}$ vs. $v_i - \mu$ for all instances, are distributed identically (in the same vector direction) – they are perfectly correlated in this case. When the errors are distributed in absolutely *orthogonal* directions, the two are perfectly independent and has no correlation. When their errors distribute in the opposite direction, they are negatively correlated, with the minimum value of -1 indicating perfectly opposite.

Experimentation

Now that we know some of the metrics for model evaluation, how do we actually conduct the assessment with data? In the ideal world, one builds a model using training data with known outcomes and rolls out the model to be tested in the operating environment. In the real world, however, we have to make sure a model works reliably well for its purpose before it is used in production for important predictions and decision making.

We often have to use existing data to test models with experiments. It is a common practice to split the data into training and testing subsets – one to build and train the model, and the other to evaluate its performance.

Random sampling can be used to create the training and testing subsets. Once a model is trained, it can be experimented with the test data and evaluated using metrics suitable for related objectives. This process can be repeated to test different models or the same model with different parameters or configurations. It can be done with different random samples.

Along the process, one may be able to find out what leads to best performances in terms of certain metrics. Given various model configurations, it is also possible to create plots such as precision-recall or ROC curves, where one can examine related parameters and trade-offs.

The ratio to split training vs. testing depends the amount of available data and the amount required to sufficiently train the specific model. One may be tempted to conduct a 50-50 split. In this case, you use half of the data (sample) for training and the other half for testing. Afterwards, it also makes sense to switch the two subsets, testing data now for training and training for testing.

This is cross validation. And you can do it on 50-50, a 2-fold validation, you can certainly do it on more folds. Practically, it is popular to conduct a 10-fold cross validation, where the data is split into 10 random subsets. Each time, 1 subset is held out for testing and the other 9 are used for training. This is repeated 10 times so the model is tested on every fold of the data.

On Efficiency

In this chapter, we have focused on the evaluation of model *effectiveness* or, loosely speaking, how satisfactory predictions are compared to the ground truth. This is about the *quality* of a model.

The other side of evaluation is on the *efficiency* of operations. This is about the time and cost necessary to train a model and to perform assigned tasks. It depends on the space and time complexity of the model, and the structure and magnitude of data the model has to work on in the production settings.

A thorough evaluation should include an assessment of both effectiveness and efficiency. We will elaborate on important aspects of efficiency as well as *scalability* in Chapter How Does It Scale?.

Is It Fit?

> Torture the data, and it will confess to anything.
>
> —*Ronald Coase*

Under- and over-fitting...
Bias, variance, and error...
Combined, ensemble learning...

How Does It Scale?

Bibliography

[1] Akiko Aizawa. An information-theoretic perspective of tf-idf measures. *Information Processing and Management*, 39(1):45–65, January 2003.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] Jacob D. Bekenstein. Information in the holographic universe. *Scientific American*, 289(2):58–65, 2003.

[4] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Commun. ACM*, 35(12):29–38, December 1992.

[5] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.

[6] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, September 2006.

[7] M. Taylor S. Saria H. Zaragoza, N. Craswell and S. E. Robertson. Microsoft cambridge at trec 2004: Web and hard track. In *The Thirteenth Text Retrieval Conference (TREC 2004)*, pages 1–7. NIST Special Publication, 2004.

[8] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[9] Kaggle. The state of data science & machine learning 2017. https://www.kaggle.com/surveys/2017. Accessed: 2019-09-10.

[10] Weimao Ke. Information-theoretic term weighting schemes for document clustering and classification. *International Journal on Digital Libraries*, 16(2):145–159, Jun 2015.

[11] Weimao Ke. Text retrieval based on least information measurement. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '17, pages 125–132, New York, NY, USA, 2017. ACM.

[12] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, sep 1999.

[13] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[14] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[15] Calvin N. Mooers. Zatocoding applied to mechanical organization of knowledge. *American Documentation*, 2(1):20–32.

[16] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.

[17] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[18] Derek De Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American Society for Information Science*, 27(5):292–306.

[19] ANATOL RAPOPORT. What is information? *ETC: A Review of General Semantics*, 10(4):247–260, 1953.

[20] Stephen Robertson. The probability ranking principle in IR. *Journal of Documentation*, 33:294–304, 12 1977.

[21] Stephen Robertson and Karen Sparck-Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27:129–146, 05 1976.

[22] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, April 2009.

[23] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[24] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[25] George K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, 1949.

Index