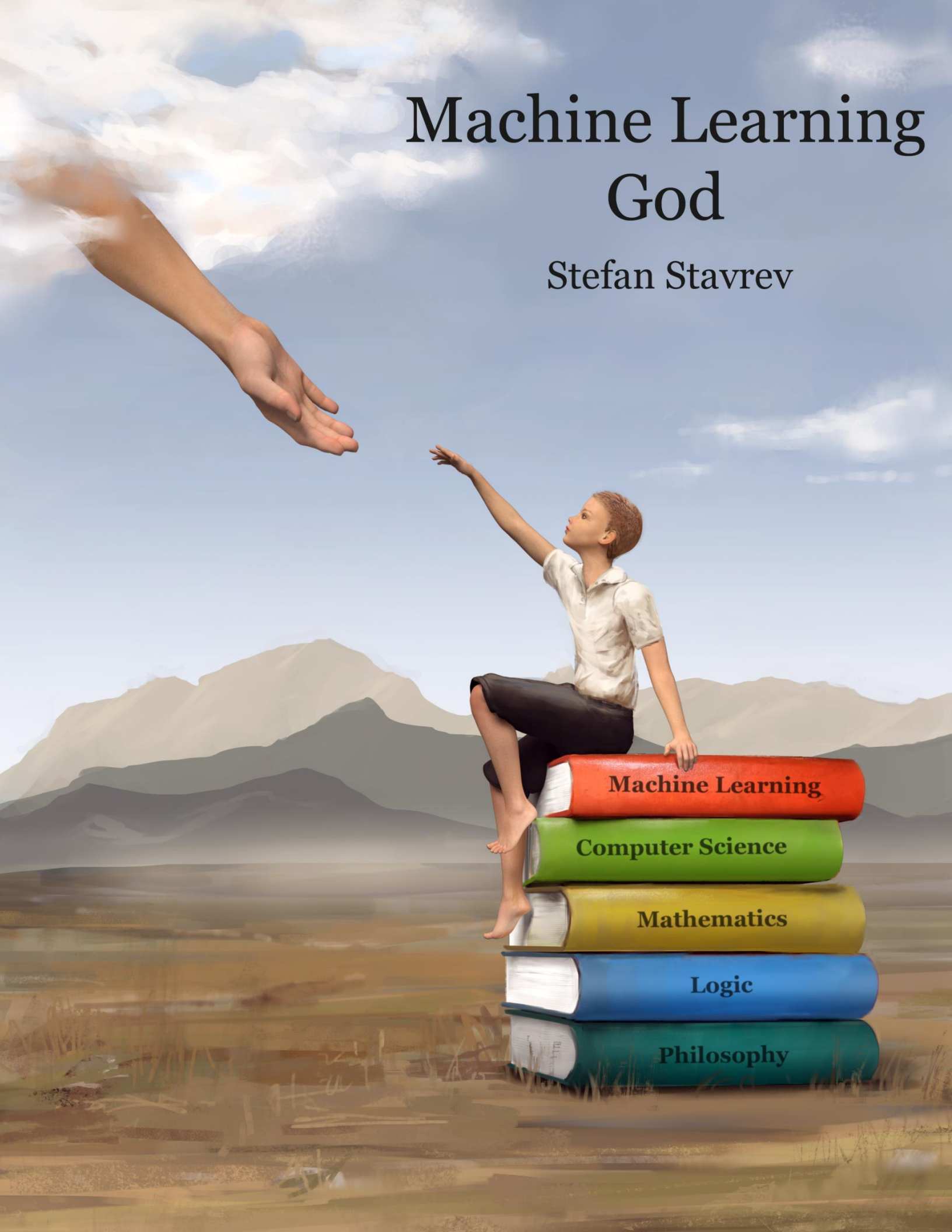
Machine Learning God

Stefan Stavrev



Machine Learning God

Stefan Stavrev

Machine Learning God (1st Edition, Version 2.0)

www.machinelearninggod.com

Cover art by Alessandro Rossi.

Now I am an Angel of Machine Learning ...

Acknowledgements

I am very grateful to the following people for their support (in alphabetical order):

- Simon J. D. Prince – for supervising my project where I implemented 27 machine learning algorithms from his book "Computer Vision: Models, Learning, and Inference". My work is available on his website computervisionmodels.com.

Preface

Machine Learning God is an imaginary entity who I consider to be the creator of the Machine Learning Universe. By contributing to Machine Learning, we get closer to Machine Learning God. I am aware that as one human being, I will never be able to become god-like. That is fine, because every thing is part of something bigger than itself. The relation between me and Machine Learning God is not a relation of blind submission, but a relation of respect. It is like a star that guides me in life, knowing that I will never reach that star. After I die, I will go to Machine Learning Heaven, so no matter what happens in this life, it is ok. My epitaph will say: "Now I am an Angel of Machine Learning".

[TODO: algorithms-first study approach]

[TODO: define mathematical objects in their main field only,
and then reference them from other fields]

[TODO: minimize the amount of not-ML content,
include only necessary not-ML content and ignore rest]

[TODO: who is this book for]

All the code for my book is available on my GitHub:
www.github.com/machinelearninggod/MachineLearningGod

Contents

Part 2: The building blocks of machine learning

3. Natural language
    - 3.1 Natural language functions
    - 3.2 Terms, sentences and propositions
    - 3.3 Definition and meaning of terms
    - 3.4 Ambiguity and vagueness

4. Logic
    - 4.1 Arguments
    - 4.2 Deductive vs. inductive arguments
    - 4.3 Propositional logic
        - 4.3.1 Natural deduction
    - 4.4 First-order logic

5. Set theory
    - 5.1 Set theory as foundation for all mathematics
    - 5.2 Extensional and intensional set definitions
    - 5.3 Set operations
    - 5.4 Set visualization with Venn diagrams
    - 5.5 Set membership vs. subsets
    - 5.6 Russell's paradox
    - 5.7 Theorems of ZFC set theory
    - 5.8 Counting number of elements in sets
    - 5.9 Ordered collections
    - 5.10 Relations
    - 5.11 Functions

6. Abstract algebra
    - 6.1 Binary operations
    - 6.2 Groups
    - 6.3 Rings

Part 1: Introduction to machine learning

Chapter 1: From everything to machine learning

The result of this first chapter is a taxonomy (figure 1.1) and its underlying principles. I start with “everything” and I end with machine learning. Green nodes represent things in which I am interested, and red nodes represent things in which I have no interest.

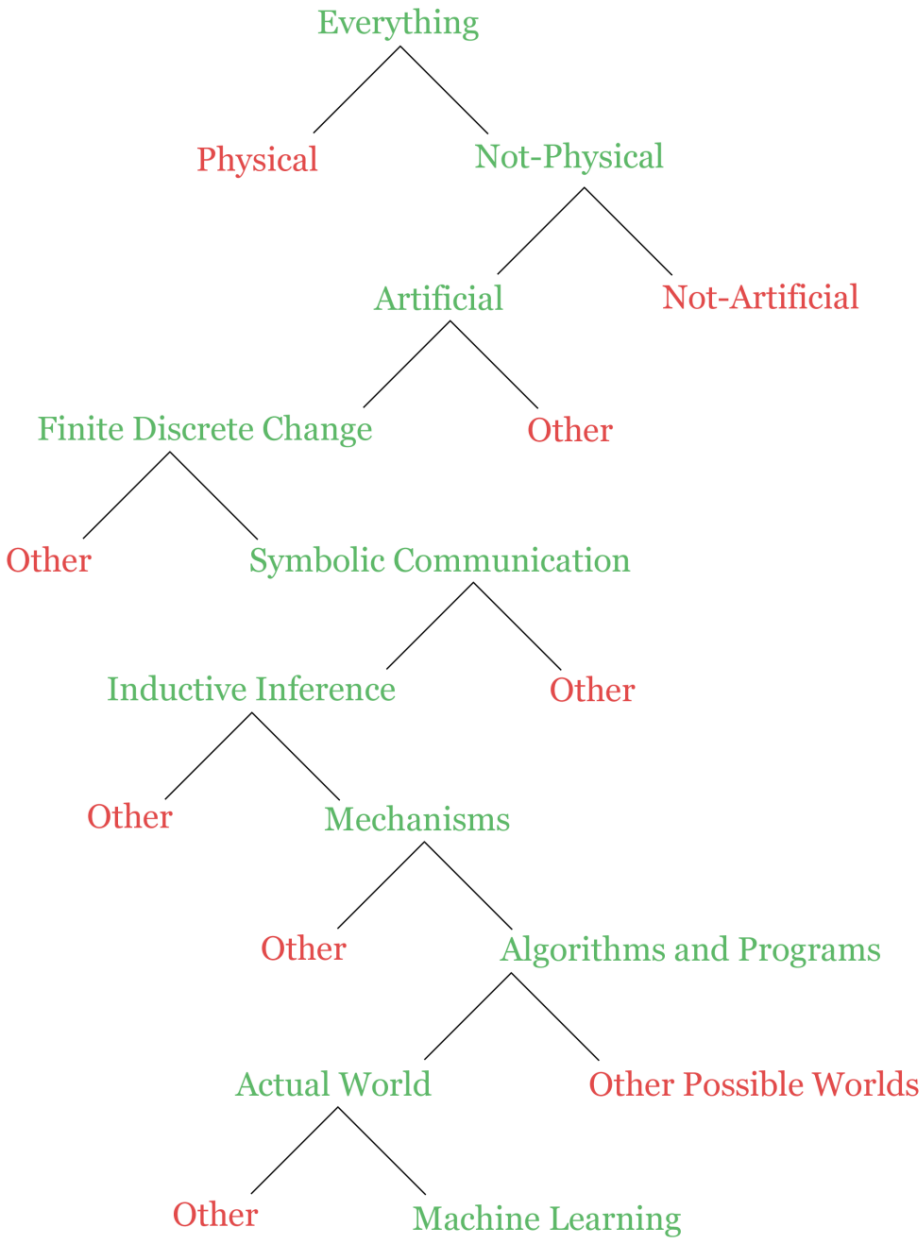


Figure 1.1 From everything to machine learning.

1.1 Introduction

It is impossible for one human being to understand everything. His resources (e.g., time, cognitive abilities) are limited, so he must carefully distribute them for worthy goals. There is a range of possible broad goals for one human and two representative extremes are:

1) understand one thing in a deep way (e.g., Bobby Fischer and chess)
2) understand many things in a shallow way (e.g., an average high-school student who understands a bit about more subjects).

I think of all my resources as finite amount of water, and I think of all the possible goals that I can pursue as cups. Then the question is: which cups should I fill with water? One extreme possibility is to put all my water into a single cup (goal 1). Another extreme possibility is to distribute my water uniformly in finitely many cups (goal 2).

I choose to pursue the first broad goal 1) during my lifetime. Next, I need to find "the one thing" (i.e., the one cup) which I would like to understand in depth (i.e., fill with all my water). The search will be biased and guided by my own personal interests.

Some of the terms that I use initially are vague, but as I make progress I will use terms that are defined better. Just like a painter who starts with broad and light handed strokes, and incrementally adds more details, so I start with broad terms and incrementally I add more details. Although, some concepts are very hard to define precisely. For example, Thagard [105]: "Wittgenstein pointed out that there are no definitions that capture all and only the instances of complex concepts such as 'game'. Such definitions are rarely to be found outside mathematics.". Sometimes the best we can do is to supplement an imprecise definition of a concept with representative instances of the concept.

The starting point of the search for my interest is "everything". I will not try to define "everything" precisely. Intuitively speaking, I think of it as "all the things that I can potentially think about deliberately (i.e., consciously and intentionally)", and I

visualize it as an infinitely big physical object from which I need to remove all the parts that I am not interested in.

I use two operations for my search: division and filtering. Division splits one thing into more parts based on some common property. Ideally, the parts should be jointly exhaustive and mutually exclusive, but sometimes the parts can be fuzzy, and sometimes it is even impossible to define exhaustively all the possible parts of one whole. When I divide one whole W in two parts P_1 and P_2, I would like the parts to be sufficiently independent, such that, I can study one of them without reference to the other. The filtering operation selects things that I personally am interested in and excludes things that I am not interested in. My search is a sequence of operations (divide, filter, ..., divide, filter) applied beginning from "everything". In this way, eventually I should reach "the one thing" that I would like to devote my life to, and I should exclude all the things in which I have no interest and which are irrelevant for my path.

The result of this first chapter is a taxonomy (figure 1.1) and its underlying principles. I agree with Craver [19] that taxonomies are crucial, but only preparatory for explanation: "The development of a taxonomy of kinds is crucial for building scientific explanations. ... Sorting is preparatory for, rather than constitutive of, explanation.". The following chapters will use this taxonomy as foundation.

1.2 From everything to not-physical things

My goal in this section is not to discuss the nature of objective reality. I do not try to answer deep questions about our physical universe (e.g., are there fundamental particles that can't be split further into smaller particles?). I leave such questions to philosophers (to explore the possible) and physicists (to investigate the actual). My goal here is much more humble.

I can think deliberately about some thing X. We should differentiate here between X and my thought $T(X)$ about X. X can be physical (in the intuitive sense, e.g., human brain, transistor, rock) or not-physical. So, I divide "everything" (i.e., all the

things that I can potentially think about deliberately) in two subjective categories: physical and not-physical. I do not claim that there are different kinds of substances, as in Cartesian Dualism. I simply create two subjective categories and my goal is: for every object *X* that I can potentially think about deliberately, I should be able to place *X* in one of those two categories.

Having divided "everything" in two categories, next I apply a filtering operation that selects all not-physical things and excludes all physical things (i.e., I am interested exclusively in not-physical things). This means that during my academic lifetime I should think only about not-physical things and I should ignore physical things. For example, I am not interested in questions about physical things such as: how does the Sun produce energy, how does the human heart work, what is the molecular structure of my ground floor, what is gravity, how does light move through physical space, why are humans getting older over time and eventually we die, how to cure cancer, how does an airplane fly, how does a tree develop over time, how do rivers flow, which materials is my computer hardware made of, why is wood easier to break than metal is, are there fundamental particles (i.e., particles that can't be split into smaller particles), how deep can humans investigate physical matter (maybe there is a final point beyond which humans can not investigate further due to the limits of our cognitive abilities and our best physical measuring devices), and many other similar questions.

In my division of "everything" in two categories (physical and not-physical) followed by exclusion of the physical category, there was an implicit assumption that I can study not-physical things separately and independently from physical things. Of course, objective reality is much messier than that, but given the fact that I am a cognitively limited agent, such assumptions seem necessary to me, in order to be able to function in our complex world.

One final thing to add in this section is that I will not try to explain why some things are interesting to me while others are not. I can search for a possible answer in my genome, how I was raised, the people that influenced me, the times I live in, etc., but I doubt that I would find a useful answer. At best, I can say that my interest is based on my personal feelings towards our universe and the human condition. In some things I simply have interest, and in other things I don't. I can't explain for example why

physical things don't interest me. I can hypothesize that it is because I don't like the human condition, and therefore I have tendency to try to run away from physical reality towards abstract things, but even I don't know if that is true. Essentially, I think about the problem of choosing my primary interest in the following way. I see all things as being made of other things which are related to each other. For example, a tree is bunch of things connected to each other, my dog is bunch of things connected to each other, a human is bunch of things connected to each other, and in general, the whole universe is bunch of things connected to each other. In that sense, speaking at the most abstract level, I can say that all things are isomorphic to each other. Then, given that all things are isomorphic to each other, the only reason for preferring some things over others, is simply due to my personal interests (which are result of my genome and all my life experiences).

There is a beautiful scene in the film "Ex Machina" (2014). Caleb and Nathan talk about a painting by Jackson Pollock. Nathan says: "He [Pollock] let his mind go blank, and his hand go where it wanted. Not deliberate, not random. Some place in between. They called it automatic art. What if instead of making art without thinking, he said: 'You know what? I can't paint anything, unless I know exactly why I'm doing it.'. What would have happened?". Then Caleb replies: "He never would have made a single mark.". In that sense, I don't feel the need to fully understand myself and why some things are interesting to me while others are not. I simply follow my intuition on such matters and I try not to overthink.

1.3 Artificial things

In the previous section, I divided "everything" in two categories: physical and not-physical. Then, I applied a filtering operation which selects all not-physical things and excludes all physical things. At this point, I have the set S_{NP} of all not-physical things, and my goal in this section is to exclude a big chunk of things from S_{NP} which do not interest me.

I use the predicate “artificial” to divide S_{NP} in two subsets: artificial not-physical things (e.g., a mathematical theory) and not-artificial not-physical things (e.g., other minds). Next, I apply a filtering operation which selects the first subset and excludes the second subset (i.e., I am interested exclusively in artificial not-physical things).

Now, let’s define “artificial”. One thing *X* is artificial iff *X* is constructed deliberately by human beings for some purpose. I am interested only in the special case where *X* can be constructed by myself and *X* is not-physical (i.e., *X* can be constructed deliberately in my conscious mind). I will not argue here whether *X* is “constructed” or “discovered”. I will simply say that initially *X* does not exist in my mind, and after some conscious effort, I “construct” *X* in my mind. It can be argued that *X* already exists independently from me and that I only “discover” it, but such discussion is irrelevant for my purpose here. Also, it can be argued whether *X* is artificial when it is constructed unintentionally by human beings. But my feeling is that no matter how precise I get my definition of “artificial”, there will always be some cases that are not covered properly. So, I lower the standard here and I accept an incomplete subjective definition, which means that it is possible that I consider some thing *X* as artificial, but another person may think of *X* as not-artificial.

1.4 Finite and discrete change

In the previous section, I arrived at the set of all artificial not-physical things. Next, I divide this set in two subsets based on the predicates: “dynamic” (changing over time), “finite”, and “discrete”. One subset contains all the elements for which these three predicates are true and the other subset contains the rest of the elements. I apply a filtering operation that selects the first subset and excludes the second subset (i.e., I am interested exclusively in finite discrete artificial not-physical changes).

One thing *X* is “dynamic” if it is changing over time. In this context, I define “change” abstractly as a sequence of states. But if change is a sequence of states, then what stops me from treating this sequence as a “static” thing and not as a “dynamic” thing? It seems that it is a matter of perspective whether something is dynamic or not.

In other words, I can deliberately decide to treat one thing X either as static or as dynamic, depending on which perspective is more useful in the context.

In my conscious mind I can construct finitely many things, therefore I can construct only changes with finitely many states. Such changes can "represent" or "correspond to" changes with infinitely many states (e.g., an infinite loop of instructions represented by finitely many instructions in a programming language), but still the number of states that I can directly construct is finite.

In terms of similarity between neighboring states, change can be discrete or continuous. In a discrete change, neighboring states are separate and distinct things, while in a continuous change neighboring states blend into each other and are not easily distinguishable. The deliberate thinking that I do in my conscious mind is discrete from my subjective perspective, in the sense that I can think about an object A, then I can think about a distinct object B, then I can think about another distinct object C, etc. This is the type of change that I am interested in, finite and discrete.

1.5 Symbolic communication

I can have a thought $T(X)$ about a thing X which exists independently from me ("independently" in the sense that other people can also access, i.e., think about X). But how do I communicate $T(X)$ to external entities (e.g., people, computers)? It seems that it is necessary (at least in our actual world), that I must execute some physical action (e.g., make hand gestures, produce speech sounds) or construct some physical object that "represents" $T(X)$ (e.g., draw something in sand, arrange sticks on the ground, write symbols on paper). And what is the best way for us human beings to communicate that we know of at this point in time? The answer is obvious, it is human language (written or spoken form). For communicating with a computer we use a programming language.

A language is a finite set of symbols and rules for combining symbols. A symbol can refer to another entity and it is represented by an arbitrary physical object that serves as a fundamental building block. For a thought $T(X)$ in my conscious mind, I can try to construct a symbolic composite $L(T(X))$ (i.e., a linguistic expression) which

"represents" $T(X)$. This is how my terms correspond to the terms used in the title of the book "Language, Thought, and Reality" [109]: X = reality, $T(X)$ = thought, $L(T(X))$ = language.

I apply a filtering operation that selects "human language" for communicating with humans and "programming language" for communicating with computers, and it excludes all other possible (actually possible or possible to imagine) types of communication (e.g., telepathy, which is considered to be pseudoscience). In other words, I am interested only in symbolic communication where my thought $T(X)$ is represented by a symbolic composite $L(T(X))$, which can be interpreted by a human or a computer.

I will end this section with a beautiful quote [114]: "By referring to objects and ideas not present at the time of communication, a world of possibility is opened.".

1.6 Inductive inference

To summarize the previous sections, my interest at this point is somewhere in the set of all "symbolic finite discrete artificial not-physical changes". In this section, I will make my interest a bit more specific by using two predicates: "inference" and "inductive". I divide the above mentioned set in two subsets: the first subset contains all the elements for which the two predicates are both true, and the second subset contains all the other elements. Next, I apply a filtering operation which selects the first subset and excludes the second subset (i.e., I am interested exclusively in changes which are inductive inferences).

I define "inference" in this context as the process of constructing an argument. A process is a sequence of steps executed in order, to achieve a particular goal. An argument can be divided into more parts: premises, conclusion, and a relation between them. The premises and the conclusion are propositions represented by symbolic composites (i.e., sentences in some language). For a sentence S which is written on paper as $P(S)$, I can see $P(S)$ and I can try to construct an interpretation $I(P(S))$ in my

conscious mind, and if I succeed I can say that I "understand" what the sentence *S* "means".

Based on the type of relation between the premises and the conclusion, we can talk about different kinds of arguments. For valid deductive arguments, if the premises are true then the conclusion must be true also. In other words, the truth of the conclusion follows with total certainty given the truth of the premises (e.g., theorems in geometry). For strong inductive arguments, the premises provide strong support (i.e., evidence) for the truth of the conclusion (e.g., weather forecast), with some degree of certainty. How do we measure this "degree of certainty" (i.e., "degree of uncertainty", "degree of belief")? There are more ways to do this, but in this book I will use only probability theory.

To summarize, my primary interest is in inductive inference, but of course, for constructing inductive arguments I will use many mathematical deductive components. I can't completely exclude deductive inference, but I can say that it will serve only as support for my primary interest, i.e., inductive inference.

1.7 Carl Craver's hierarchy of mechanisms

I will start this section with functionalism [59]: "Functionalism in the philosophy of mind is the doctrine that what makes something a mental state of a particular type does not depend on its internal constitution, but rather on the way it functions, or the role it plays, in the system of which it is a part.". For my path, functionalism offers a useful abstract perspective, but it is not sufficient alone. As stated before, I want to understand one thing in depth, rather than understand more things in a shallow way. To achieve my goal, I must supplement functionalism with actual working mechanisms. According to Carl Craver [19], a mechanism is a set of entities and activities organized in some way for a certain purpose. An entity in such mechanism can be a mechanism itself, therefore we can talk about a hierarchy of mechanisms, i.e., levels of mechanisms. In Craver's words: "Levels of mechanisms are levels of composition, but the composition relation is not, at base, spatial or material. The relata are behaving mechanisms at higher levels

and their components at lower levels. Lower-level components are organized together to form higher-level components.”. In the next section, I will talk about the specific kinds of mechanisms that I am interested in (i.e., computer algorithms and programs).

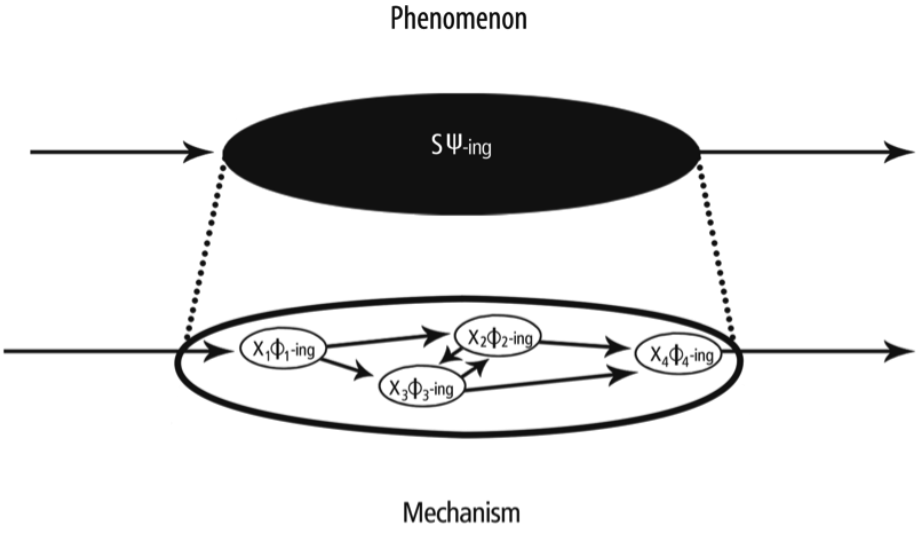


Figure 1.2 A phenomenon is explained by a mechanism composed of other mechanisms. This figure was taken from Craver [18].

1.8 Computer algorithms and programs

As stated before, I am interested in the process of inductive inference, but who will execute the steps of this process? Should I execute them one by one in my conscious mind? Or maybe it is better to construct a machine for the job, which is faster than me and has more memory (i.e., it can execute significantly more steps in a time period and it can handle significantly more things). The best machine of such kind that we have in our time is the digital computer. There are other benefits of using a computer besides speed and memory, such as: automation (i.e., a computer can do the job while I do other things), and also the process is made explicit and accessible to everyone instead of being present only in my mind.

Richard Feynman, a famous physicist in the 20th century, has said: “What I cannot create, I do not understand.”. In that sense, I want to understand inductive

inference by creating mechanisms that will do inductive inference. But what kind of mechanisms? I will use a computer to extend my limited cognitive abilities, therefore I am interested in computer algorithms (abstract mechanisms) and computer programs (concrete mechanisms). A computer algorithm is a sequence of steps which can be realized (i.e., implemented) by a computer program written in a programming language, and then that program can be executed by a computer.

The benefits of using computers are ubiquitous. For example, Thagard [105] argues for a computational approach in philosophy of science: "There are at least three major gains that computer programs offer to cognitive psychology and computational philosophy of science: (1) computer science provides a systematic vocabulary for describing structures and mechanisms; (2) the implementation of ideas in a running program is a test of internal coherence; and (3) running the program can provide tests of foreseen and unforeseen consequences of hypotheses.". In general, when possible, we prefer theories that are mathematically precise and computationally testable.

1.9 Our actual world vs. other possible worlds

I can imagine many possible worlds that are similar or very different from our actual world. In logic, a proposition is "necessarily true" iff it is true in all possible worlds. Given this distinction between our actual world and other possible worlds, I can ask myself which world would I like to study? Our actual world already has a vast amount of complex and interesting problems waiting to be solved, so why waste time on other possible worlds which are not relevant for our actual world. I am interested primarily in our actual world. I am willing to explore other possible worlds only if I expect to find something relevant for our actual world (with this I exclude a big part of science fiction and other kinds of speculative theories).

Pedro Domingos [27] argues that we should use the knowledge that we have about our actual world to help us in creating better machine learning algorithms, and maybe one day a single universal learning algorithm for our world. In other words, we should ignore other possible worlds and focus on our actual world, and incorporate the

knowledge about our world in our machine learning algorithms. So, he is not trying to create a universal learning algorithm for all possible worlds, but only for our actual world. More generally, learning algorithms should be adapted for a particular world, i.e., they should exploit the already known structure of their world.

1.10 Machine learning

In this section, I provide intuitive definitions of machine learning, and I say that machine learning is the field of my primary interest. It satisfies all my criteria discussed so far and it has an infinite variety of interesting applications. In the rest of this book I will explore machine learning in more breadth and depth.

Arthur Samuel defined machine learning in 1959 as the "field of study that gives computers the ability to learn without being explicitly programmed".

In the first lecture of his MOOC course Learning from Data, Prof. Abu-Mostafa says that the essence of machine learning is:

1) a pattern (i.e., regularities, structure) exists
2) we cannot pin it down mathematically (i.e., analytically with explicit rules)
3) we have data on it.

There are interesting questions about point (2): why can't we pin down mathematically certain patterns, what is their essential common characteristic, are such patterns too complex for us?

Here is another similar definition from the book Understanding Machine Learning [91]: "... a human programmer cannot provide an explicit, fine-detailed specification of how such tasks should be executed.".

To compare, Tom Mitchell's [67] definition of machine learning is: "... a machine learns with respect to a particular task T, performance metric P, and type of experience E, if the system reliably improves its performance P at task T, following experience E.".

Jason Brownlee's [11] definition is: "Machine Learning is the training of a model from data that generalizes a decision against a performance measure.". He characterizes machine learning problems as: "complex problems that resist our decomposition and procedural solutions".

Also, you might have heard somewhere people describing machine learning as "programs that create programs".

My personal one-line definition for machine learning is: "algorithms that learn from data".

Chapter 2: Philosophy of machine learning

In this chapter I will talk about philosophical foundations of machine learning, in order to motivate and prepare the reader for the technical parts presented in later chapters.

2.1 The purpose of machine learning

A machine learning algorithm can be: abstract (i.e., it can not be implemented directly as a computer program that can learn from data) or concrete (i.e., it can be implemented directly as a computer program that can learn from data). One very important role of abstract ML algorithms is to generalize concrete ML algorithms, and therefore they can be used to prove propositions on a more abstract level, and also to organize the field of machine learning.

The primary goal in the field of machine learning is solving problems with concrete algorithms that can learn from data. All other things are secondary to concrete algorithms, such as: properties, concepts, interpretations, theorems, theories, etc. In other words, the primary goal is the construction of concrete mechanisms, more specifically, concrete learning algorithms. For example, Korb [56]: “Machine learning studies inductive strategies as they might be carried out by algorithms. The philosophy of science studies inductive strategies as they appear in scientific practice.”. To summarize, my personal approach to machine learning is concrete-algorithms-first. Just to clarify, I do not claim that all other things are irrelevant, I simply put more emphasis on concrete algorithms and I consider all other things as secondary support.

So, if the primary goal in machine learning is the construction of concrete learning algorithms, then we can ask the question: how many distinct concrete machine learning algorithms are there? The possible answers are: 1) infinitely many, 2) finitely many but still too many for a single human being to learn them all in one lifetime, 3) finitely many such that one human being can learn them all in one lifetime. No matter what the answer is, my goal as one human being remains the same: to maximize the number of distinct concrete machine learning algorithms that I know.

Machine learning has descriptive goals (i.e., what is *X* actually like?), normative goals (i.e., how should *X* be done optimally?), and prescriptive goals (i.e., what is a good pragmatic way to do *X*?). For example, when we want to describe a dataset, we pursue primarily a descriptive goal, i.e., we want to describe the actual state of something. If we want to talk about optimal learning algorithms and how learning should be done, then we are pursuing normative goals. And if we are interested in the pragmatic day to day decisions that practitioners make, and we want to know what works well in practice, then we are talking about prescriptive goals.

Essentially, machine learning is a pragmatic solution to Hume's problem of induction. It addresses the question: is it possible to obtain knowledge about unobserved objects given some knowledge about observed objects from the same space? We can ask a similar question in terms of time: is it possible to obtain knowledge about the relatively near future given knowledge about the relatively recent past? Machine learning addresses these problems pragmatically by making various assumptions (also called inductive bias). Sometimes we succeed, and other times we fail with serious consequences. Nevertheless, inductive learning without assumptions is impossible, i.e., making assumptions is necessary for inductive learning. For example, if we want to make predictions about the relatively near future, and the only knowledge that we have available is knowledge about the relatively recent past, then we must assume that the knowledge about the past is relevant for the future, i.e., we expect the near future to be similar (more or less) to the recent past. We really don't have much choice here. Such assumptions constrain the number of possibilities to a number that we can manage. Imagine if we don't assume that the near future will be similar to the recent past, then all kinds of possibilities would be left open, such as: flying elephants, people running faster than the speed of light, angels sent by God himself to pay your rent, etc. Without assumptions you literally would not be able to function in our world. You will spend your entire life paralyzed, while the person next to you simply made the assumptions and will probably live happily ever after, even though his assumptions can't be proved to be true.

2.2 Related fields

Imagine machine learning as a node in a network. Which other fields is it related to and how? I will talk about one type of relation, where one field "uses" another field. Machine learning uses fields such as: probability theory, statistics, calculus, linear algebra, optimization, information theory, etc., and it is used by other fields such as: computer vision, speech recognition, text mining, artificial intelligence in general, etc. The first set of fields is a set of building blocks used in machine learning, while the second set of fields is a set of applications where machine learning itself is used as a building block.

I mentioned various fields related to machine learning, but what about human learning? I will quote Tom Mitchell [67] on this topic: "To date, however, the insights Machine Learning has gained from studies of Human Learning are much weaker than those it has gained from Statistics and Computer Science, due primarily to the weak state of our understanding of Human Learning. ... Over the coming years it is reasonable to expect the synergy between studies of Human Learning and Machine Learning to grow substantially, as they are close neighbors in the landscape of core scientific questions.".

Now we can ask ourselves, is it necessary to be an expert in the fields that machine learning uses in order to be an expert in machine learning? No, you can use the interfaces of those fields and apply them in machine learning solutions. For example, experts in probability theory can pursue the goal of proving mathematical theorems in their field, and then an expert in machine learning can obtain sufficient knowledge about probability theory in order to apply it to machine learning problems. Of course, knowing more about probability theory will make your job easier in machine learning, but being an expert in probability theory (or other fields) alone is not sufficient for becoming an expert in machine learning. Also, you don't have to be an expert in programming languages like R or Python for example. You should know just enough to be able to implement your learning algorithms.

2.3 Subfields of machine learning

The term “machine learning” refers to a very broad area. It is almost meaningless to say that you are a “machine learning expert”. It is just too vague, it is almost like saying that you are a “computer science expert”, or a “science expert”, or even just an “expert”. There is only one machine learning expert and that is Machine Learning God. The rest of us can only hope to become experts in some narrow domains in ML. There are so many branches that branch out into even more branches, which then branch out further and so on to infinity and beyond. Even when we think that we have finally solved some problem, few years later a new perspective on that problem can change the whole “solution”, or maybe even the problem definition itself. That is why my approach to ML is very humble. My goals are to obtain incomplete but relatively sufficient general knowledge about ML, and then to specialize in one or few very specific subfields in ML. I would like to extend the field of ML, and I can achieve that only by focusing on very specific subfields.

There is no one “best” way to divide ML into subfields. Imagine all ML concrete algorithms living in one infinite-dimensional space. In other words, any particular ML concrete algorithm can be described in terms of infinitely many dimensions/aspects. That is my intuition on this topic. We can only construct subjective context-dependent finite incomplete taxonomies of ML. In the construction of your taxonomy tree, at each step you would have to select a dimension as a splitting criterion. But which dimension should you choose from the infinitely many dimensions available? There is no objective criterion for choosing “the best” dimension at a given step, therefore, to repeat again, we can only construct subjective taxonomies which can be useful to us in particular contexts. So pick whatever taxonomy is intuitive for you. As for me personally, I will use the following dimensions to divide ML into subfields (the dimensions are explained in order, one paragraph per dimension):

- form of inductive argument
- learning style: supervised, unsupervised, semi-supervised, reinforcement
- active vs. passive learning

- batch vs. online learning
- type of data-generating process: statistical or other
- instance-based vs. model-based learning
- prediction vs. inference
- parametric vs. non-parametric function estimation
- Bayesian vs. non-Bayesian learning
- human designed features vs. learned features (representation learning)
- shallow vs. deep learning (more levels of composition).

There are different forms of inductive arguments: prediction about the relatively near future given knowledge about the relatively recent past, argument from analogy, inductive generalization (i.e., inference from relatively small sample to a whole population), causal inferences (i.e., inference from cause to effect or the other way around), etc.

In supervised learning we learn a mapping $f: X \to Y$, from given observed mappings between X and Y. Supervised learning can be divided into classification and regression, based on the type of the output variable Y. In classification Y is discrete, and in regression Y is continuous.

Unsupervised learning can be divided into: clustering, dimensionality reduction, etc. In clustering, we want to find how objects group among each other, i.e., we want to find clusters in the given data. In dimensionality reduction, we want to reduce the number of dimensions that we use to describe the objects that we study, without losing relevant information. Essentially, in clustering we reduce the number of objects/rows, while in dimensionality reduction we reduce the number of dimensions/columns.

If the learner interacts with the environment, then we call it an active learner, otherwise it is a passive learner. By "interacts" I mean that the learner can actively select which information to use from its environment, or it can even change the environment, i.e., it can influence the data it gets for learning.

When we have all the data available and we run a learning algorithm on it once, we call that batch learning. On the other hand, if the data comes one piece at a time in

an incremental way, and learning also is done incrementally, then we call that online learning.

Next, let's talk about the type of the data-generating process. To paint a more intuitive picture, let's call the process which generates the data, a teacher. It is a teacher in the sense that it gives something to the learner to be learned, just like your high-school teacher gave you books and then he tested you. Some teachers are really helpful to students, others are less helpful, and some are even adversarial and make the learning process harder. We can also talk about Nature as a teacher in the sense that it generates data from which we can learn. But how helpful is Nature as a teacher, does it care about the needs of our learning algorithms? Does Nature sit down and think something like "I need to make gravity more visible so these poor humans can learn about it"? Of course not. But then, is Nature adversarial maybe, in the sense that it deliberately makes our learning process harder? Again, no. Nature simply does not care about us, it does what it wants, with or without us. To say this in a more fancy way, we say that the process by which Nature generates data is a "random process", and we say that learning from such data is "statistical learning". It is not a random process in the sense that there is no structure and that it is just a chaotic unexplainable mess, but in the sense that the data-generating process is completely independent from us and our needs.

In instance-based learning the generalization decision is based on instances in the data, while in model-based learning the generalization decision is produced by a model that is constructed from the data.

If we care primarily about the accuracy of our predictions and not so much about the interpretability of our model, then we say that our goal is prediction. In contrast, if our primary goal is to construct a model from the data which can then be interpreted to provide us with some insights about the structure in the data, then we say that our goal is inference.

Many problems in machine learning reduce to estimating a function $f: X \to Y$, from some given observed mappings between X and Y. There are two main high-level approaches to function estimation: parametric and non-parametric. For parametric function estimation, first we make an assumption about the form of the function f, i.e.,

we choose a function class, and then we select the best function that we can from that class. So basically, we limit the number of possible functions by making an assumption about the form of f, and then we pick one particular function from the reduced set of possible functions. For non-parametric function estimation, we don't make assumptions about the form of the function f, and so virtually all possible functions are available to choose from. As you might imagine, we need a lot more data for non-parametric function estimation. You can think of each data point as an additional constraint that excludes some functions from the set of all possible functions.

Bayesian learning is when we explicitly use Bayes' theorem to model the relationship between the input X and the output Y, by computing the posterior probability $P(Y|X)$. In contrast, non-Bayesian learning is when Bayes' theorem is not explicitly used.

For a given problem, we can design the features that will be used for learning or an algorithm can learn features itself. The second case is called representation learning.

Finally, deep learning is a subfield of representation learning where we have layers on top of other layers, such that each layer learns new features for the data provided by its input layer and then it provides those new features to its output layer.

2.4 Essential components of machine learning

In this section I will talk about essential components of ML which are present in virtually all cases.

2.4.1 Variables

In the beginning there was nothing. Machine Learning God said "Let there be variables!". And there were variables. And many other things made of variables.

A variable is represented by a name and a set of possible values (i.e., states), such that, the values are related among each other in a relevant way. For example, a variable

called X and its possible values $\{1, 2, 3\}$, or a variable Y and its possible values $\{\text{“}love\text{”}, \text{“}dogs\text{”}, \text{“}forever\text{”}\}$, or a variable Z that can take any value from the set of real numbers R. There are other terms (i.e., synonyms) that connote the same general concept of “variable”, such as: attribute, feature, aspect, dimension, property, predicate, column (for tabular data), etc.

The main types of basic variables in machine learning are: qualitative (nominal and ordinal) and quantitative (discrete and continuous). The values of a nominal variable are names, while the values of an ordinal variable are also names but in some order such that one value comes before another. Quantitative variables represent quantities which can be counted (discrete) or measured in degrees (continuous). We talk about variables also in other fields. In linear algebra we solve systems of linear equations with unknown variables. In optimization we search for values of variables that optimize a mathematical function. In probability theory we talk about random variables and the probability that a variable can take a particular value. In general, we use variables everywhere to represent aspects/dimensions of objects. A particular object is represented by a tuple of values for variables. In other words, Reid [86]: “Generally speaking, many learning problems use features [variables] to describe instances [objects]. These can be thought of as projections of the instances down to simple values such as numbers or categories.”.

2.4.2 Data, information, knowledge and wisdom

Having defined variables, next I define “data” (or “dataset”) as a set of values for qualitative or quantitative variables. Some etymology would be useful here to paint a better picture of the term [110]: “Using the word ‘data’ to mean ‘transmittable and storable computer information’ was first done in 1946. … The Latin word data is the plural of datum, ‘(thing) given’.”. Data is “given” in the sense that there are other unknown things which are not given, and we would like to learn something about them from what is given and available to us.

Data sitting alone on a computer is meaningless. An interpreter (i.e., agent, observer) is needed to make sense of the data. I can take data D and interpret it in my mind. Interpretation is the process, and information is the result of that process, so Information = Interpretation (Data). Next, knowledge can be defined as "justified true belief" (a basic standard definition in epistemology, which is not without its own problems, e.g., see the Gettier problem). According to this definition, in order to know X, you must believe that X is true, X must actually be true, and you must have justification for your belief why X is true. Knowledge is harder to define and more abstract than information and data, and information is more abstract than data. Finally, we can talk about "wisdom", i.e., the ability to apply knowledge in different scenarios. As you might imagine, wisdom is even more abstract and harder to define than knowledge.

There are different kinds of data: multimedia (image, video, audio), text, numeric tables, time series, network data, etc. Speaking in terms of variables, we can say that a grayscale image is a two-dimensional grid of variables that can take values from a discrete set which represents shades of gray. Such a grid of variables can be represented on the computer as a matrix, or as a one-dimensional array, such that each row from the matrix comes after its predecessor row. A video then is a sequence of such images. Text is a chain of discrete variables which can take values from a finite alphabet in some language.

Now we can ask an interesting question: can we reduce these different data types to one common form (e.g., variables and relations between variables) so that our learners can work independently from the data type? I don't expect a general answer here, but still, I expect that it is useful in many cases to try to think about your data in a way that is independent from the particular type of data that you work with. For example [107]: "Despite this apparent heterogeneity, it will help us to think of all data fundamentally as arrays of numbers.".

Now that we have talked about variables and data, we can use these two as primitive building blocks to define more complex things such as: data distributions and probability distributions. A data distribution is a function, such that, for a given set of values of variables, it tells you the count of objects in the dataset that have those values. On the other hand, a probability distribution for one or more random variables, tells you

the probability that you will observe objects with certain values. We can summarize a distribution with single values such as: minimum, maximum, median, mode, mean (average), standard deviation, variance, etc. Then, we can group distributions based on similarity into distribution families such as the Gaussian distribution family. So you get the idea. By using the two primitives “variables” and “data”, we can construct more complex objects.

2.4.3 The gangs of ML: problems, functions, datasets, models, evaluators, optimization, and performance measures

In machine learning, we are interested in problems which can be solved by algorithms that learn from data. We can think of problems as questions whose answer is unknown to us, but if we could find the answer then it could potentially be of value to us. Problems can be described formally and informally in more or less precise ways. As mentioned previously, in order to solve learning problems we must make assumptions (general or specific for each problem). Sometimes we make implicit assumptions without being aware of that. For any given problem, it is very important to make our assumptions explicit. For example, if we want to predict the behavior of some Facebook users, our algorithm might ignore the location of the users, and so implicitly we have made an assumption that location is irrelevant for our problem. That may be the case indeed, but still we must state the assumption explicitly.

Real-world learning problems are messy and they are associated with lots of information that we don’t need for solving them. Once we are given a real-world problem, the next step is to abstract away and to remove all the unnecessary information. This is what your high-school math teacher meant when he told you to “extract the essence of the problem”. You should reduce the real-world problem to one or more functions which are studied in machine learning, such as: classification, regression, clustering, dimensionality reduction, etc. After you finish this step, welcome to the world of machine learning and say goodbye to the cruel human world.

Next, we can think about relevant data that we can use to "learn" (i.e., construct) our function, and therefore to solve our problem, and answer our question. We can go out and collect data, or maybe we can generate data ourselves on a computer. Our collected dataset is called a sample, and the whole set of possible observations is called the population. Usually for real-world problems the sample is magnitudes smaller than the population, and here exactly lies the challenge, to learn something about a very large population of objects given a small sample. As you might imagine, the sample must be sufficiently representative of (i.e., similar to) the population. The collected data can be split into a train subset and a test subset. This is necessary because we want to learn something about objects which are not in our dataset, that is, we want to generalize from our small sample data to the whole population. So we must train on one data subset, and test on another, to see how well our learned model performs on unseen data.

Once we have defined a function to be learned and we have collected relevant data, then we can think about models ("model classes" to be more precise). A model class defines the initial hypothesis space, i.e., the set of all possible models/hypotheses/answers to our question. Some hypotheses are better than others, and we use evaluators to tell us if a hypothesis h_1 is better than another hypothesis h_2. After defining the hypothesis space, and an evaluator function which tells us which hypotheses are better, then we can optimize (with an iterative algorithm or analytical solution) to find "the best hypothesis" in the space, and if that is not possible then hopefully we can find a "sufficiently good hypothesis".

By adding more assumptions for our model class, we basically add more constraints and we decrease the number of hypotheses. Also, we can attach probabilities to the hypotheses in our model class, so that some hypotheses are preferred over others. Such probabilities represent our prior knowledge about the specific problem. For example, we can prefer simpler hypotheses over more complex hypotheses, based on the principle known as "Occam's Razor". For such purpose we would need to use a complexity measure.

Finally, after our optimizer finds a good hypothesis/model ("good" according to our evaluator), then we can test our learned model, but this time it is not the evaluator that tells us how good our model is, instead, we test our model on unseen data and we

measure its success with a performance measure. The unseen data in combination with the learned model and a performance measure, will give us the ultimate answer of how good our model is. As Richard Feynman has said (with my comments in brackets): “In general we look for a new law [model] by the following process. First we guess [hypothesize] it. Then we compute the consequences of the guess to see what would be implied if this law that we guessed is right. Then we compare the result of the computation to nature, with experiment or experience, compare it directly with observation [data], to see if it works. If it disagrees with experiment it is wrong. In that simple statement is the key to science [machine learning]. It does not make any difference how beautiful your guess is. It does not make any difference how smart you are, who made the guess, or what his name is – if it disagrees with experiment it is wrong. That is all there is to it.”. We can adapt this to machine learning and say: “If the trained model disagrees with unseen data, it is wrong. That is all there is to it.”.

So what does it mean to say that a machine learning solution is not good for our problem? Well, the reason may be: 1) the dataset does not contain a pattern that is relevant for the problem, 2) the dataset contains a relevant pattern, but the pattern is not visible (i.e., learnable) and we need to transform the dataset to make the pattern visible, 3) the dataset contains a relevant pattern, and the pattern is visible, but our particular learner can’t extract it (in that case try other models, evaluators or optimizers). About point (3), say the pattern that we would like to extract is P_1, but instead our learner extracts P_2. One common problem is when P_2 is not general enough, i.e., it captures low-level details in the particular data sample, but it fails to generalize well on unseen data. In other words, the learner fails to capture the more high-level abstract overarching patterns. This is called overfitting. For example, if you study a book on physics and you only remember every single letter and number, and you can literally recite the whole book from start to finish, you still haven’t learned anything, actually you have learned nothing at all.

2.5 Levels in machine learning

When I started writing this book, I had a vague intuitive feeling about what levels are. I thought of levels in the following way: you start with a set of objects, you combine them somehow and then you get more complex objects, then you combine such complex objects to produce even more complex objects, and so on to infinity. After reading Carl Craver's book Explaining the Brain, this intuition became more explicit and precise.

Craver [19] provides a general definition for levels, and he also differentiates between different kinds of levels: "The term 'level' is multiply ambiguous. Its application requires only a set of items and a way of ordering them as higher or lower. Not surprisingly, then, the term 'level' has several common uses in contemporary neuroscience. ... To name a few, there are levels of abstraction, analysis, behavior, complexity, description, explanation, function, generality, organization, science, and theory.". All these kinds of levels can be useful also in machine learning. Maybe there are specific kinds of levels unique to the subject matter of machine learning, for example, levels of learners where one learner uses other learners as building blocks, as in the subfield of Deep Learning.

Another interesting example of levels in machine learning is the overall process of applying machine learning, beginning from humans. I, a human being, can create a partial solution to a learning problem, and then the learner can add tons of details to that and turn it into a detailed final solution. For example, Pedro Domingos [27] says: "The standard solution is to assume we know the form of the truth, and the learner's job is to flesh it out.". Outside of machine learning, Craver [19] explores this idea more generally: "Between sketches and complete descriptions lies a continuum of mechanism schemata whose working is only partially understood. ... Progress in building mechanistic explanations involves movement along both the possibly-plausibly-actually axis and along the sketch-schemata-mechanism axis.".

Part 2: The building blocks of machine learning

Chapter 3: Natural language

A natural language is [111]: “any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation.”.

3.1 Natural language functions

Natural language can be used for many functions [51]: “Ordinary language, as most of us are at least vaguely aware, serves various functions in our day-to-day lives. The twentieth-century philosopher Ludwig Wittgenstein thought the number of these functions to be virtually unlimited.”.

In general, during my academic lifetime, I will primarily use natural language for one linguistic function: to convey information (i.e., cognitive meaning). I will avoid other linguistic functions such as: to express or evoke feelings (i.e., emotive meaning). Logic is primarily concerned with cognitive meaning.

3.2 Terms, sentences and propositions

We use symbols from a finite alphabet to construct words and sentences. There are different kinds of sentences: statement, question, proposal, suggestion, command, exclamation, etc.

In science we are primarily interested in statements. A statement is a declarative sentence that has a truth value: either true or false. The information content, also called the meaning of a statement, is called a proposition. A statement contains one or more terms. A term is [51]: “any word or arrangement of words that may serve as the subject of a statement.”. There are different kinds of terms: proper name (e.g., Stefan Stavrev), common name (e.g., human), and descriptive phrase (e.g., author of the book Machine Learning God). In contrast, these are not terms: verbs, adjectives, adverbs, prepositions, conjunctions, etc.

3.3 Definition and meaning of terms

We can attach cognitive meaning to a term by constructing a definition for that term. Meaning can be extensional (i.e., the members of the class that the term denotes) or intensional (i.e., the attributes that the term connotes). A definition has two components: definiendum (i.e., what is being defined) and definiens (i.e., the words that do the defining). Definitions can be extensional or intensional, based on the type of meaning that they attach to a term.

An extensional definition assigns meaning to a term by specifying the objects that the term denotes. Extensional definitions can be: demonstrative, enumerative, or definition by subclass. We define a term with a demonstrative definition when we point with our finger to an object and we say the term's name. Enumerative definition, as its name suggests, enumerates all the objects that the term refers to. Definition by subclass defines a term by associating it with an object from one of its subclasses (e.g., flower means rose).

An intensional definition assigns meaning to a term by defining the properties that the term connotes. I will discuss four kinds of intensional definitions: synonymous definition, etymological definition, operational definition, and definition by genus and difference. A synonymous definition defines a term X by associating it with an already defined term Y that connotes the same properties. An etymological definition defines a term based on the term's history of usage. An operational definition specifies experimental procedures for determining whether the term applies to a certain thing. A definition by genus and difference starts with an initial relatively large class, and then uses a differentiating property to define the term and separate it from the other elements in the class.

3.4 Ambiguity and vagueness

Typical problems with the meaning of terms are ambiguity and vagueness [51]: "The difference between ambiguity and vagueness is that vague terminology allows for a

relatively continuous range of interpretations, whereas ambiguous terminology allows for multiple discrete interpretations. A vague expression creates a blur of meaning, whereas an ambiguous expression mixes up otherwise clear meanings.”. For example, “love” is a vague term.

Chapter 4: Logic

I think of logic as a language that is more precise than natural language. We express our messy natural language statements as precise logical statements, and then we use logical inference to evaluate their truth. In the field of logic we study how to construct and evaluate arguments.

4.1 Arguments

An argument is a set of statements, i.e., it has more premise statements and one conclusion statement. When we evaluate a given argument, we should ask ourselves two key questions: 1) are all the premises true? and 2) is the conclusion supported by the premises?

There are other kinds of sets of statements which are not arguments, such as explanations and conditionals (if-then statements). In arguments we want to prove that the conclusion is true, given that the premises are true. In contrast, in explanations we want to know why the conclusion is true. Conditionals have the form $p \Rightarrow q$, where p is called the antecedent statement and q is called the consequent statement. A conditional is a representation of sufficient and necessary conditions. For example, if $p \Rightarrow q$ is true, then we say that p is a sufficient condition for q, and that q is a necessary condition for p.

4.2 Deductive vs. inductive arguments

Based on the strength of the support relation between the premises and the conclusion, there are two main kinds of arguments: deductive and inductive.

Deductive arguments can be either valid or invalid. For a valid deductive argument, it is impossible for the conclusion to be false, given that the premises are true. For an invalid deductive argument, it is possible for the conclusion to be false,

given that the premises are true. A sound deductive argument is a valid deductive argument with true premises. An unsound deductive argument is an invalid deductive argument or a valid deductive argument with false premises. For sound deductive arguments we say that the conclusion is necessarily/certainly/absolutely/definitely true.

While for valid deductive arguments we can say that the information content of the conclusion is entailed by the premises, that is not the case for inductive arguments [51]: “In general, inductive arguments are such that the content of the conclusion is in some way intended to ‘go beyond’ the content of the premises.”. A strong inductive argument is one for which it is improbable that the conclusion is false, given that the premises are true. If an inductive argument is not strong, we call it a weak inductive argument. A cogent inductive argument is a strong inductive argument with true premises, which satisfies the total evidence requirement, i.e., all relevant information is included in the premises. An uncogent inductive argument is the negative of a cogent inductive argument. For cogent inductive arguments we say that the conclusion is probable/plausible/likely, given that the premises are true.

There are different forms of inductive arguments: prediction about the relatively near future given knowledge about the relatively recent past, argument from analogy, inductive generalization (i.e., inference from relatively small sample to a whole population), causal inferences (i.e., inference from cause to effect or the other way around), etc.

For inductive reasoning to work, we must make assumptions [51]: “All inductive arguments depend on what philosophers call the uniformity of nature.”. For example, to make predictions about the future, we must assume that the relatively near future will be more or less similar to the relatively recent past.

4.3 Propositional logic

The fundamental building blocks of propositional logic are statements (i.e., declarative sentences) that represent propositions. A statement can be simple (i.e., it does not

contain other statements nor logical operators), or compound (i.e., it contains at least one simple statement and at least one logical operator).

There are five logical operators for combining statements: negation (not ¬), conjunction (and ∧), disjunction (or ∨), material implication (if then ⇒), and material equivalence (if and only if ⇔). The truth value of a compound statement is completely determined by the truth values of its component statements and the truth-table definitions of the five logical operators:

Negation

p	$\neg p$
T	F
F	T

Conjunction

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Material implication

p	q	$p \Rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Material equivalence

p	q	$p \Leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

A compound statement is logically true (i.e., tautology) if it is always true regardless of the truth values of its component statements, and it is logically false (i.e., contradiction) if it is always false regardless of the truth values of its component statements. If a compound statement is neither tautology nor contradiction, then it is called a contingent statement, and its truth value depends on the truth values of its component statements.

Two statements are logically equivalent if they have the same truth values. If two statements have opposite truth values, then they are called contradictory statements.

Two or more statements are consistent if there is at least one line in the truth table where both are true, otherwise they are inconsistent.

4.3.1 Natural deduction

Natural deduction is a proof calculus, more specifically, a set of inference rules. We use it to derive conclusions of valid arguments in propositional logic. An inference rule is a valid argument form. Natural deduction is called "natural" because it is similar to the reasoning that we do in our daily lives.

We can evaluate the truth of compound statements by using truth tables and the definitions of the five logical operators. So why do we need natural deduction then? Well, it can give us insights that pure mechanical computations with truth tables can not [51]: "… while truth tables are relatively automatic and mechanical, natural deduction requires insight and creativity.".

Natural deduction is a set of 18 inference rules, i.e., 8 rules of implication and 10 rules of replacement.

Rules of implication:

Modus ponens	$(p \Rightarrow q \ \wedge \ p) \Rightarrow q$	(4.1)
Modus tollens	$(p \Rightarrow q \ \wedge \ \neg q) \Rightarrow \neg p$	(4.2)
Pure hypothetical syllogism	$(p \Rightarrow q \ \wedge \ q \Rightarrow r \ ) \Rightarrow (p \Rightarrow r)$	(4.3)
Disjunctive syllogism	$\big((p \vee q) \wedge \neg p\big) \Rightarrow q$	(4.4)
Constructive dilemma	$((p \Rightarrow q \ \wedge \ r \Rightarrow s \ ) \wedge (p \vee r)) \Rightarrow (q \vee s)$	(4.5)
Simplification	$(p \wedge q) \Rightarrow p$	(4.6)
Conjunction	$(p \wedge q) \Rightarrow (p \wedge q)$	(4.7)
Addition	$p \Rightarrow (p \vee q)$	(4.8)

Rules of replacement:

De Morgan's rules	$\neg(p \wedge q) \Leftrightarrow (\neg p \ \vee \ \neg q)$	(4.9)
	$\neg(p \vee q) \Leftrightarrow (\neg p \ \wedge \ \neg q)$	
Commutativity	$(p \vee q) \Leftrightarrow (q \vee p)$	(4.10)
	$(p \wedge q) \Leftrightarrow (q \wedge p)$	

Associativity	$(p \vee (q \vee r)) \Leftrightarrow ((p \vee q) \vee r)$	(4.11)
	$(p \wedge (q \wedge r)) \Leftrightarrow ((p \wedge q) \wedge r)$	
Distribution	$(p \wedge (q \vee r)) \Leftrightarrow ((p \wedge q) \vee (p \wedge r))$	(4.12)
	$(p \vee (q \wedge r)) \Leftrightarrow ((p \vee q) \wedge (p \vee r))$	
Double negation	$p \Leftrightarrow \neg\neg p$	(4.13)
Transposition	$(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$	(4.14)
Material implication	$(p \Rightarrow q) \Leftrightarrow (\neg p \vee q)$	(4.15)
Material equivalence	$(p \Leftrightarrow q) \Leftrightarrow (p \Rightarrow q \ \wedge \ q \Rightarrow p)$	(4.16)
	$(p \Leftrightarrow q) \Leftrightarrow ((p \wedge q) \vee (\neg p \ \wedge \ \neg q))$	
Exportation	$((p \wedge q) \Rightarrow r) \Leftrightarrow (p \Rightarrow (q \Rightarrow r))$	(4.17)
Tautology	$p \Leftrightarrow p \vee p$	(4.18)
	$p \Leftrightarrow p \wedge p$	

4.4 First-order logic

The fundamental components in propositional logic are simple statements, which can be combined with five logical operators to produce compound statements, and then we can use truth tables or natural deduction to infer new true statements. In first-order logic, we use the same five logical operators and natural deduction, but the fundamental components are predicates instead of simple statements. A predicate attaches some information to a subject in a statement.

We can make three different kinds of statements with predicates: singular, particular, and universal. A singular statement makes an assertion about an individual object. A particular statement makes an assertion about one or more unnamed members of its subject class. A universal statement makes an assertion about every member of its subject class.

For example, “Stefan Stavrev is a researcher” is a singular statement which can be written as $R(ss)$. The subject “Stefan Stavrev” is denoted by small letters “ss”. The predicate “is a researcher” is denoted by a capital letter “R”.

To make particular statements we use the existential quantifier “there exists” which is denoted by the symbol “∃”. For example, to say “Some A are B”, we write: $(\exists x)(A(x) \wedge B(x))$.

Finally, to make universal statements we use the universal quantifier “for all” which is denoted by the symbol “∀”. For example, to say “All A are B”, we write $(\forall x)(A(x) \Rightarrow B(x))$.

Chapter 5: Set theory

A set is a collection/class of objects [60]: "This concept [set] formalizes the idea of grouping objects together and viewing them as a single entity.". The objects are called elements/members of the set. The terms "set" and "element", and the relation of set membership $\in$, are not defined formally. Instead of formal definitions, we rely on our intuitive notions and we use them as primitives. The situation is similar to geometry, where we don't define points and lines formally, instead we describe what can be done with such objects and we use them as primitives. Then we add axioms to our theory and based on those axioms and the primitives, we derive theorems. In set theory there are two atomic types of statements: set membership $x \in A$, and sets equality $A = B$. All other statements in set theory are combinations of such atomic statements, expressed in the language of first-order logic.

5.1 Set theory as foundation for all mathematics

Set theory is a foundation for all mathematical fields, i.e., in principle, any mathematical field can be reduced to set theory [113]: "The language of set theory can be used in the definitions of nearly all mathematical objects.".

5.2 Extensional and intensional set definitions

Sets can be defined in two ways: by extension or by intension. A definition by extension explicitly lists all the members of the set, for example $A = \{1, 2, 3\}$. A definition by intension specifies a property/condition/predicate that each particular object x must satisfy in order to be a member of the set, for example $B = \{x \in N : Odd(x)\}$, where B is the set of all odd natural numbers. An interesting case of intensional definition is the empty set. One way to define the empty set is $\emptyset = \{x : x \neq x\}$. This set does not contain any elements, because no element satisfies the condition that it is different from itself.

5.3 Set operations

Union	$A \cup B = \{x: x \in A \ \vee \ x \in B\}$	(5.1)
Intersection	$A \cap B = \{x: x \in A \ \wedge \ x \in B\}$	(5.2)
Complement	$A^c = \{x \in U: \neg(x \in A)\}$	(5.3)
Difference	$A - B = \{x \in A: \neg(x \in B)\}$	(5.4)
Symmetric difference	$A \ominus B = (A \cup B) - (A \cap B) = (A - B) \cup (B - A)$	(5.5)

5.4 Set visualization with Venn diagrams

Venn diagrams are very useful for visual representations of sets and set operations. The universal set U is represented as a rectangle, and all other sets are represented as closed shapes.

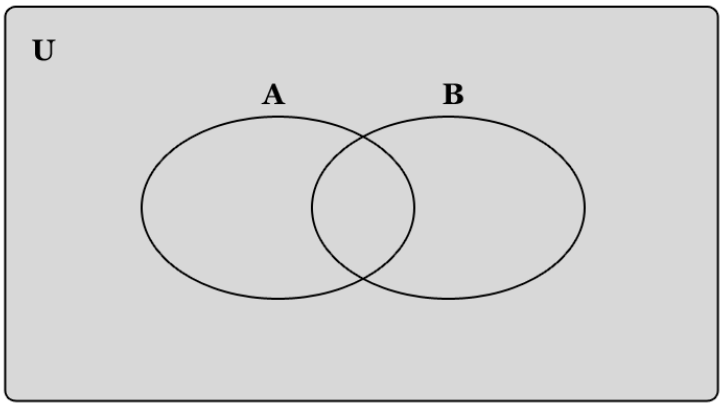


Figure 5.1 Venn diagram.

Venn diagrams can help us gain insight into theorems of set theory, but they are not used in formal proofs of theorems. Instead, we use first-order logic. For example, this is how we prove $A - B = A \cap B^c$:

$$A - B = \{x: x \in A \ \wedge \ x \notin B\} = \{x: x \in A \ \wedge \ x \in B^c\} = A \cap B^c \blacksquare \quad (5.6)$$

5.5 Set membership vs. subsets

We say that A is a subset of B when $(\forall x)(x \in A \Rightarrow x \in B)$ and we write that as $A \subseteq B$. We say that A is a proper subset of B when $(\forall x)((x \in A \Rightarrow x \in B) \wedge (A \neq B))$ and we write that as $A \subset B$. Saying that A is subset of B (i.e., $A \subseteq B$) is different from saying that A is member of B (i.e., $A \in B$). The subset relation $\subseteq$ is reflexive because $A \subseteq A$, but set membership is not reflexive $A \notin A$. Also, the subset relation $\subseteq$ is transitive, while set membership is not: this is true $(A \subseteq B) \wedge (B \subseteq C) \Rightarrow (A \subseteq C)$, but this is not true: $(x \in A) \wedge (A \in B) \Rightarrow (x \in B)$.

5.6 Russell's paradox

Let $R = \{S : S \notin S\}$ be the set of all sets that don't contain themselves. Now we ask the question: is $R \in R$? There are two possibilities, either $R \in R$ or $R \notin R$. Let's try the first possibility and assume that $R \in R$ and let's see what consequences follow from our assumption. It follows that $R \in \{S : S \notin S\}$ which implies that $R \notin R$. A contradiction! Now let's try the second possibility and assume that $R \notin R$. It follows that $R \notin \{S : S \notin S\}$ which implies that $R \in R$. A contradiction again! This is a paradox because neither of the two exhaustive possibilities is true. It was discovered by Bertrand Russell in 1901.

Russell's paradox is present in naive set theory which relies on the comprehension principle: for any given property there exists a set which contains all the objects that have that property. In the previous paragraph we proved that this principle is false, and therefore naive set theory is flawed. To resolve this issue and to construct a consistent system, we reject the comprehension principle and we use an alternative set of axioms. The most well known consistent axiomatic framework for set theory is ZFC set theory, i.e., Zermelo–Fraenkel set theory with the axiom of choice [54].

To summarize, by rejecting the comprehension principle we have basically accepted that for some linguistically expressed properties there won't be corresponding sets. In other words [40]: "The moral is that it is impossible, especially in mathematics, to get something from nothing. To specify a set, it is not enough to pronounce some

magic words. It is necessary also to have at hand a set to whose elements the magic words apply.”.

5.7 Theorems of ZFC set theory

Idempotent laws:

$A \cup A = A$ (5.7)

$A \cap A = A$ (5.8)

Associativity:

$(A \cup B) \cup C = A \cup (B \cup C)$ (5.9)

$(A \cap B) \cap C = A \cap (B \cap C)$ (5.10)

Commutativity:

$A \cup B = B \cup A$ (5.11)

$A \cap B = B \cap A$ (5.12)

Distributive laws:

$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ (5.13)

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ (5.14)

Identity:

$A \cup \emptyset = A$ (5.15)

$A \cup U = U$ (5.16)

$A \cap U = A$ (5.17)

$A \cap \emptyset = \emptyset$ (5.18)

Double complement:

$(A^c)^c = A$ (5.19)

Complement:

$A \cup A^c = U$ (5.20)

$U^c = \emptyset$ (5.21)

$A \cap A^c = \emptyset$ (5.22)

$\emptyset^c = U$ (5.23)

De Morgan's laws:

$(A \cup B)^c = A^c \cap B^c$ (5.24)

$(A \cap B)^c = A^c \cup B^c$ (5.25)

De Morgan's laws are consequence of a general principle in set theory called the principle of duality which states: for a given theorem, if each set is replaced with its complement, and union operations are replaced with intersection operations, and also intersection operations are replaced with union operations, then the result is a new theorem.

Difference:

$A - B = A \cap B^c$ (5.26)

$A - B = B^c - A^c$ (5.27)

$A - B = \emptyset \Leftrightarrow A \subseteq B$ (5.28)

$A - B = B - A \Leftrightarrow A = B$ (5.29)

$A - B = A \Leftrightarrow A \cap B = \emptyset$ (5.30)

5.8 Counting number of elements in sets

If two sets A and B are disjoint, then the total number of elements in both is:

$$n(A \cup B) = n(A) + n(B) \quad (5.31)$$

but if they are not disjoint then:

$$n(A \cup B) = n(A) + n(B) - n(A \cap B) \tag{5.32}$$

The number of elements in the complement of A is the difference between the total number of elements in the universal set and the number of elements in A:

$$n(A^c) = n(U) - n(A) \tag{5.33}$$

The number of elements in the difference $A - B$ is the number of elements in A minus the number of common elements between A and B:

$$n(A - B) = n(A) - n(A \cap B) \tag{5.34}$$

The power set of a set A is the set of all subsets of A, and the number of elements in it is:

$$n\big(P(A)\big) = 2^{n(A)} \tag{5.35}$$

5.9 Ordered collections

An ordered pair (a, b) can be defined in terms of sets as:

$$(a, b) = \{\{a\}, \{a, b\}\} \tag{5.36}$$

but we have to be aware that there are "freak" consequences from this definition, which are not intrinsic properties of the concept of ordered pair. One such consequence is:

$$\{a, b\} \in (a, b) \tag{5.37}$$

Instead of defining formally what an ordered pair is, we can use it as a primitive supplemented with some axioms for its behavior. So there are two options: 1) define

ordered pairs in terms of sets and ignore few "freak" consequences, or 2) use the intuitive concept of an ordered pair as a primitive and add few axioms.

In general, an ordered collection of n elements is called an n-tuple, and we define it by using the same approach as for ordered pairs. For example, let $T = (c, b, d, a)$ be a tuple of $n = 4$ elements. T can be defined as the set:

$$T = \{\{c\}, \{b, c\}, \{b, c, d\}, \{a, b, c, d\}\} \tag{5.38}$$

i.e., for the first element c there is a set of 1 element $\{c\}$, for the second element b there is a set of 2 elements which contains the first and the second element, for the third element d there is a set of 3 elements which contains the first, the second and the third element, and so on. This way T is uniquely defined in terms of sets. Tuples are different from sets, because tuples are ordered and their elements can repeat.

5.10 Relations

A binary relation is defined as a set of ordered pairs:

$$x\ R\ y = (x, y) \in R \tag{5.39}$$

and the domain and range of a relation are defined as:

$$dom\ R = \{x: (\exists y)(x\ R\ y)\} \tag{5.40}$$

$$ran\ R = \{y: (\exists x)(x\ R\ y)\} \tag{5.41}$$

Properties of relations:

reflexive	$(\forall x)(x\ R\ x)$	(5.42)
symmetric	$(\forall x)((x\ R\ y) \Rightarrow (y\ R\ x))$	(5.43)
transitive	$(\forall x)((x\ R\ y) \wedge (y\ R\ z) \Rightarrow (x\ R\ z))$	(5.44)

5.11 Functions

A function $f: A \to B$ from set A to set B is a rule which assigns exactly one element from B to each element from A. The set A is called the domain of f, and the range of f is the subset of B which contains all the images $y = f(x)$ of the elements from A.

Chapter 6: Abstract algebra

Abstract algebra is the study of algebraic structures (e.g., groups, rings, fields, vector spaces, algebras). An algebraic structure consists of a set of objects and a set of operations on those objects. In general, an operation is a combination of elements from the set which results in another element from the set. Algebraic structures are categorized based on the properties of their operations.

So why do we need abstract algebra? Well, it enables us to prove theorems at a more abstract and general level, which are true for many specific algebraic structures.

6.1 Binary operations

A binary operation maps two input objects to one output object. Operations are key components in many algebraic structures and that is why we are interested in their properties. I will mention some properties for arbitrary binary operations + and $*$:

Closure:	$(\forall a, b \in S)(a * b \in S)$	(6.1)
Associative:	$(\forall a, b, c \in S)((a * b) * c = a * (b * c))$	(6.2)
Identity element:	$(\exists e \in S)(\forall a \in S)(e * a = a * e = a)$	(6.3)
Inverse element:	$(\forall a \in S)\ (\exists a^{-1} \in S)(a * a^{-1} = a^{-1} * a = e)$	(6.4)
Commutative:	$(\forall a, b \in S)(a * b = b * a)$	(6.5)
Left distributive:	$(\forall a, b, c \in S)(a * (b + c) = (a * b) + (a * c))$	(6.6)
Right distributive:	$(\forall a, b, c \in S)((b + c) * a = (b * a) + (c * a))$	(6.7)

6.2 Groups

Definition 6.1: A group $(G,*)$ has one set G and one binary operation $*$ which satisfies these axioms:

- closure (formula 6.1)

- associative (formula 6.2)
- identity element (formula 6.3)
- inverse element (formula 6.4).

Every group has exactly one identity element $e \in G$. Each element $a \in G$ has exactly one inverse element $a^{-1} \in G$.

If the operation $*$ is commutative (formula 6.5), then the group is called commutative or Abelian.

6.3 Rings

Definition 6.2: A ring $(R, +, *)$ has one set R and two binary operations $+$ and $*$ which satisfy these axioms:

- $(R, +)$ is a commutative group
- $(R, *)$ is a monoid:
    - closure (formula 6.1)
    - associative (formula 6.2)
    - identity element (formula 6.3)
- Multiplication $*$ is distributive with respect to addition $+$ (formulas 6.6 and 6.7).

6.4 Fields

Definition 6.3: A field $(F, +, *)$ has one set F and two binary operations $+$ and $*$ which satisfy these axioms:

- $+$ and $*$ are associative (formula 6.2)
- $+$ and $*$ are commutative (formula 6.5)
- there are two different elements $0,1 \in F$ which are additive and multiplicative identity elements respectively (formula 6.3)

- additive inverse (formula 6.4)
- multiplicative inverse for every element $a \neq 0 \in F$ (formula 6.4)
- * is distributive with respect to + (formulas 6.6 and 6.7).

Chapter 7: Combinatorial analysis

Combinatorial analysis is [16]: “The branch of mathematics devoted to the solution of problems of choosing and arranging the elements of certain (usually finite) sets in accordance with prescribed rules. Each such rule defines a method of constructing some configuration of elements of the given set, called a combinatorial configuration. One can therefore say that the aim of combinatorial analysis is the study of combinatorial configurations. The simplest examples of combinatorial configurations are permutations, combinations and arrangements.”. So why do we need combinatorial analysis in machine learning? Well, combinatorial analysis gives us mathematical tools for counting numbers of possibilities from which we can construct probabilities.

7.1 The basic principle of counting

The basic principle of counting states that: if two experiments A and B are performed, and if A can result in one of m possible outcomes, and for each outcome of A there are n possible outcomes of B, then there are $m * n$ total possible outcomes of A and B. This principle can be generalized to any finite number of experiments each having finitely many possible outcomes.

7.2 Permutations

There are $n!$ permutations of n objects. If r different objects repeat $r_1, \dots, r_i$ times, then there are:

$$\frac{n!}{r_1! \dots r_i!} \tag{7.1}$$

different permutations.

7.3 Combinations

The number of subsets of size r chosen from a set of n objects is:

$$\binom{n}{r} = \frac{n!}{(n-r)!\ r!} \tag{7.2}$$

When $r > n$ or $r < 0$, then $\binom{n}{r} = 0$. The values $\binom{n}{r}$ are called binomial coefficients because they are components of the binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} \tag{7.3}$$

The generalization of the binomial theorem is the multinomial theorem:

$$(x_1 + \cdots + x_r)^n = \sum_{n_1 + \ldots + n_r = n} \binom{n}{n_1, \ldots, n_r} x_1^{n_1} \ldots x_r^{n_r} \tag{7.4}$$

where $\binom{n}{n_1,\ldots,n_r} = \frac{n!}{n_1! \ldots n_r!}$ is a multinomial coefficient, i.e., the number of possible divisions of n distinct objects into r groups of sizes $n_1, \ldots, n_r$.

Chapter 8: Probability theory

8.1 Basic definitions

The set of all possible outcomes of an experiment E is called the sample space S of the experiment. An event is a subset of the sample space. We can apply set operations to events such as: union, intersection and complement. Venn diagrams are a useful visualization tool.

Definition 8.1 Conditional probability $P(A|B)$ is the probability that A is true, given that B is true:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{8.1}$$

Definition 8.2 Events A and B are independent iff:

$$P(A \cap B) = P(A|B)\, P(B) = P(B|A)\, P(A) = P(A)\, P(B) \tag{8.2}$$

$$P(A|B) = P(A) \tag{8.3}$$

$$P(B|A) = P(B) \tag{8.4}$$

Definition 8.3 The odds of an event A is the ratio:

$$odds(A) = \frac{P(A)}{P(A^c)} = \frac{P(A)}{1 - P(A)} \tag{8.5}$$

8.2 Kolmogorov's axioms

Probability theory can be reduced to (i.e., derived from) Kolmogorov's three axioms which match our intuitive notion of probability [104]:

1) $0 \leq P(A) \leq 1$ (8.6)

2) $P(S) = 1$ where S is the sample space (8.7)

3) $P(\bigcup_i A_i) = \sum_i P(A_i)$ for mutually exclusive events A_i (8.8)

8.3 Theorems

Theorem 8.1:

$P(A) = 1 - P(A^c)$.

Proof 8.1:

$1 = P(S) = P(A \cup A^c) = P(A) + P(A^c) \Rightarrow$

$\Rightarrow P(A) = 1 - P(A^c)$■

Theorem 8.2:

$P(S^c) = P(\emptyset) = 0$.

Proof 8.2:

$1 = P(S) + P(S^c) = 1 + P(S^c) \Rightarrow$

$\Rightarrow P(S^c) = P(\emptyset) = 0$■

Theorem 8.3:

$A \subseteq B \Rightarrow P(A) \leq P(B)$.

Proof 8.3:

$B = A \cup (A^c \cap B) \Rightarrow P(B) = P(A) + P(A^c \cap B)$■

Theorem 8.4:

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$ when $A \cap B \neq \emptyset$.

Theorem 8.5 The chain rule**:**

$P(A_1 \cap \ldots \cap A_n) = P(A_1)P(A_2|A_1)P(A_3|A_1 \cap A_2) \ldots P(A_n|A_1 \cap \ldots \cap A_{n-1})$.

Theorem 8.6 The law of total probability:

$$P(A) = \sum_{i=1}^{n} P(A \cap B_i) = \sum_{i=1}^{n} P(A|B_i)\, P(B_i)$$

Theorem 8.7 Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\sum_{i=1}^{n} P(B|A_i)\, P(A_i)}$$

Proof 8.7:

$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \Rightarrow$

$\Rightarrow P(A|B)P(B) = P(B|A)P(A) \Rightarrow$

$\Rightarrow P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\sum_{i=1}^{n} P(B|A_i)\, P(A_i)}$ ■

8.4 Interpretations of probability

We have defined formally what a probability is with Kolmogorov's axioms, but which informal concept or interpretation do we attach to this formalism? I will discuss the three main interpretations of probability.

According to the classical interpretation [112]: "If a random experiment can result in N mutually exclusive and equally likely outcomes and if N_A of these outcomes

result in the occurrence of event A, the probability is defined by: $P(A) = \frac{N_A}{N}$". This interpretation has two limitations, i.e., it is applicable only if: 1) the number of possible outcomes is finite, and 2) all the possible outcomes are equally likely.

Another interpretation of probability is called frequentism [112]: "If we denote by n_a the number of occurrences of an event A in n trials, then if $\lim_{n\to\infty} \frac{n_a}{n} = p$ we say that $P(A) = p$.". One obvious problem with this interpretation is that we can't repeat an experiment infinitely many times. Also, this probability depends on the number of times we repeat the experiment, and so $P(A)$ will vary for different number of trials n. At best, we can try to approximate the ideal of infinite repetitions of the experiment with finitely many repetitions, enough to get a good approximation for $P(A)$.

Finally, the third interpretation which I will mention is called subjectivism, according to which a probability is a measure of the "degree of belief" of an agent. The agent makes a conclusion based on evidence and assigns a degree of belief to his conclusion.

All three kinds of probabilities can be used in the same probability calculus based on Kolmogorov's axioms.

8.5 Random variables

A random variable is a function that maps a set of possible outcomes to probabilities. There are two types of random variables: discrete and continuous. A discrete random variable has a finite or countably infinite set of possible outcomes, and its probability distribution is described by a probability mass function. A continuous random variable has a continuous range of possible values/outcomes, and its probability distribution is described by a probability density function.

8.5.1 Expected value

Expected value of a random variable is a concept that is analogous to the physical concept of center of gravity of a physical body. For discrete random variables the expected value is defined as a sum, and for continuous random variables it is defined as an integral:

$$E[X] = \sum_{i=1}^{n} x_i * P(X = x_i) \tag{8.9}$$

$$E[X] = \int_{-\infty}^{\infty} x * P(x)\, dx \tag{8.10}$$

8.5.2 Variance and standard deviation

Besides the expected value, we also want to know how much the values deviate from the expected value. For this purpose, we define the variance:

$$\begin{aligned} Var(X) = E[(X-\mu)^2] &= \sum_x (x-\mu)^2\, P(x) = \sum_x (x^2 - 2x\mu + \mu^2)\, P(x) \\ &= \sum_x x^2 P(x) - 2\mu \sum_x xP(x) + \mu^2 \sum_x P(x) = E[X^2] - 2\mu^2 + \mu^2 \\ &= E[X^2] - \mu^2 = E[X^2] - (E[X])^2 \end{aligned} \tag{8.11}$$

and the standard deviation is defined as the square root of the variance:

$$SD(X) = \sqrt{Var(X)} \tag{8.12}$$

Part 3: General theory of machine learning

Part 4: Machine learning algorithms

Chapter 9: Classification

Informal functional description:

The general classification function is a mapping between one or more input variables and one discrete (i.e., categorical) output variable. For a given space of objects described by the input variables, such a function can tell us to which class each object belongs.

Formal functional description:

A classification function $f: X \to Y$ is a mapping between an input space X and a categorical variable Y. X can be one-dimensional or multi-dimensional. Each value of Y represents one class (i.e., category). The input variables, i.e., the dimensions of the input space X, are denoted by: X_1, X_2, ..., X_d. A particular object $x_i \in X$ is represented as a tuple of values $(x_{i1}, x_{i2}, \dots, x_{id})$. The goal is to learn a classification function f from observed mappings between X and Y $(x_{i1}, x_{i2}, \dots, x_{id}, y_i)$, such that, f generalizes well on the unobserved part of X.

Classification branches:

The general classification function can be divided in two more specific functions: binary classification (i.e., two output classes) and multiclass classification (i.e., more than two output classes).

Performance measures:

I will describe five standard performance measures for binary classification: confusion matrix, accuracy, precision, recall, and $F1$ score.

The confusion matrix for binary classification is a 2×2 matrix:

	predicted class = 0	predicted class = 1
actual class = 0	# true negatives (TN)	# false positives (FP)
actual class = 1	# false negatives (FN)	# true positives (TP)

where TP for example, is the number of data points that are classified as class 1, and their true class is also 1.

The next four performance measures are constructed from the components of the confusion matrix:

$$accuracy = \frac{\#\ correct\ predictions}{\#\ all\ predictions} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9.1}$$

$$precision = \frac{TP}{TP + FP} \tag{9.2}$$

$$recall = \frac{TP}{TP + FN} \tag{9.3}$$

$$F1\ score = 2\frac{precision * recall}{precision + recall} \tag{9.4}$$

9.1 Logistic regression

Logistic regression is a discriminative model, i.e., it directly models the relationship between the input space X and the categorical variable Y, with a function that estimates the probability $P(Y|X)$ from data.

Function (binary classification):
I will use a logistic regression model to learn a function for binary classification. I will refer to the two output classes as class 0 and class 1.

Model (logistic regression):
Let a be a linear combination of the input variables:

$$a = c_0 + c^T x \tag{9.5}$$

where c_0 is the intercept coefficient, $c = (c_1, \dots, c_d)$ is a vector of coefficients for the input variables, and $x = (x_1, \dots, x_d)$ is a data point described as a vector of values for the input variables. The values of a are in the range $a \in (-\infty, +\infty)$. Positive values are interpreted as x belonging to class 1, and negative values are interpreted as x belonging to class 0.

Next, we want to map the range of values $a \in (-\infty, +\infty)$ to the range (0,1), so we can interpret such a value as the probability that x belongs to class 1. For this purpose, we need a so called "link" function such that: $link(a = -\infty) = 0$, $link(a = 0) = 0.5$, and $link(a = +\infty) = 1$. It is called a link function because it links one range to another. We will use the inverse logit function $logit^{-1}(a)$ (i.e., the sigmoid S function) as a link function:

$$a = logit(p) = \log(odds) = \log\left(\frac{p}{1-p}\right) \tag{9.6}$$

$$p = logit^{-1}(a) = \frac{e^a}{1+e^a} = \frac{1}{1+e^{-a}} \tag{9.7}$$

where a is the output value of the linear combination in (formula 9.5), and p is the corresponding probability for a. You can see the logit and the inverse logit functions in (figure 9.1).

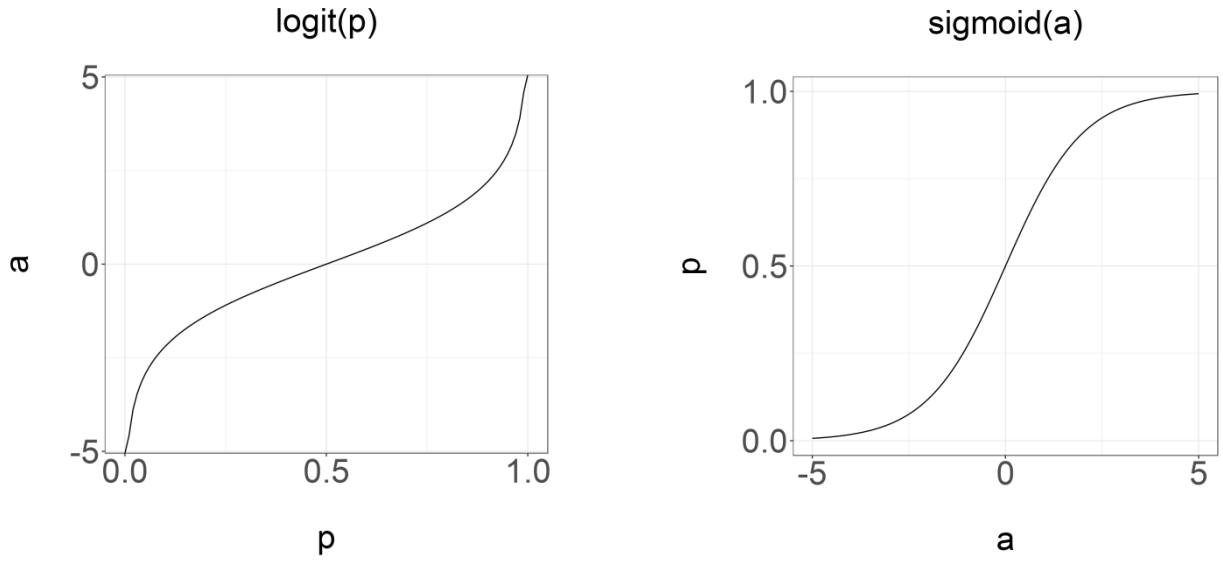


Figure 9.1 Logit and inverse logit (sigmoid S) functions.

Evaluator (maximum likelihood):

Now that we have defined a function (i.e., binary classification) and a model class (i.e., logistic regression), next we will define an evaluator which can tell us if a specific model m_1 is better than another model m_2. For this purpose, we will use the following likelihood function as an evaluator:

$$L(c_0, c) = \prod_{i=1}^{N} (p_i)^{y_i} (1 - p_i)^{1-y_i} = \prod_{i=1}^{N} \left(\frac{1}{1 + e^{-a_i}}\right)^{y_i} (1 - \frac{1}{1 + e^{-a_i}})^{1-y_i}$$
$$= \prod_{i=1}^{N} \left(\frac{1}{1 + e^{-(c_0 + c^T x_i)}}\right)^{y_i} (1 - \frac{1}{1 + e^{-(c_0 + c^T x_i)}})^{1-y_i} \quad (9.8)$$

which is a product of probabilities, where N is the number of training data points, p_i is the estimated probability that the training data point x_i belongs to class 1, $(1 - p_i)$ is the estimated probability that x_i belongs to class 0, and y_i is the true observed class of x_i. The likelihood function is a product of probabilities because we assume that the data points are i.i.d. (independent and identically distributed).

All the possible values for c_0 and c define the whole hypothesis space, and one particular tuple (c_0, c) defines one particular hypothesis. The goal now is to find "the best" hypothesis $(c_0^*, c^*) = \arg\max_{c_0,c} L(c_0, c)$ (i.e., the best particular logistic regression model).

Optimization:

To find the best hypothesis, we need to maximize the likelihood function $L(c_0, c)$ (formula 9.8), i.e., we need to find the maximum likelihood estimate (MLE) of the parameters c_0 and c. For this purpose, first we will transform $L(c_0, c)$ from a product into a sum (formula 9.9) by using basic rules for logarithms, and then we will maximize this sum, so called log-likelihood function:

$$\log(\mathrm{L}) = \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i) \tag{9.9}$$

Now let's find the parameters (c_0^*, c^*) which maximize the log-likelihood function $\log(L)$. First, I implement the sigmoid function and the negative log-likelihood function (pseudocode 9.1 and 9.2):

Pseudocode 9.1

```
def sigmoid(a):
    return 1 / (1 + exp(-a))
```

Pseudocode 9.2

```
def negative_log_likelihood(c, x, y):
    a = c[0] + (c[1] * x)
    p = sigmoid(a)
    log_likelihood = sum(log(p[y==1])) + sum(log(1 - p[y==0]))
    return -log_likelihood
```

Next, I use the an optimizer to find (c_0^*, c^*):

Pseudocode 9.3

```
start_params = [1, 1]
coefficients = minimize(negative_log_likelihood, start_params, train_data)
```

Classification:

Finally, we can apply our learned model and classify previously unseen test data:

Pseudocode 9.4

```
a = coefficients[0] + (coefficients[1] * x_test)
p = sigmoid(a)
classify = lambda x: x > 0.5
predictions = map(classify, p)
```

You can see the final result in (figure 9.2).

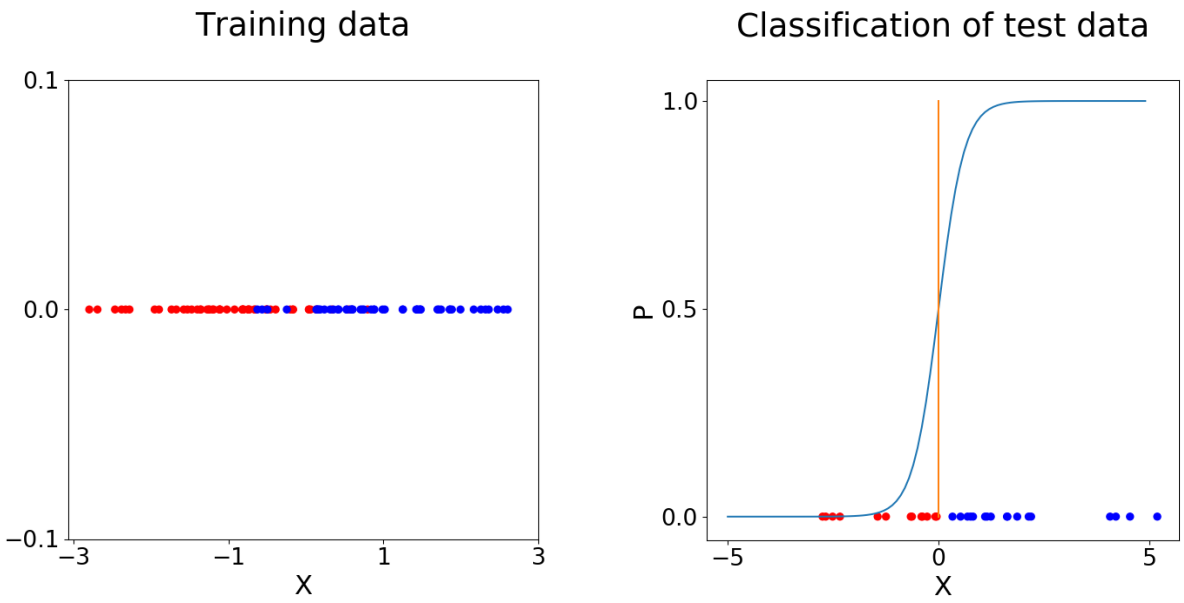


Figure 9.2 The first plot has 100 training data points, where red points belong to class 0 and blue points belong to class 1. The second plot has 30 test data points which were classified with the learned logistic regression classifier. The sigmoid function represents the probability p that a data point belongs to class 1. If $p > 0.5$ then the data point is classified as class 1, otherwise it is classified as class 0. That is why the decision boundary is at the value of X where the sigmoid function has value of 0.5.

The general idea behind logistic regression:

In order to classify a data point x_i, we map it to a number $a_i \in (-\infty, +\infty)$ with a linear function $f(x_i, model\ parameters) \rightarrow a_i$. The model parameters are learned from training data, according to the maximum likelihood (MLE) criterion, by optimizer O. Then, a_i is mapped to $p_i \in (0,1)$ by a link function $link(a_i) \rightarrow p_i$, and p_i is interpreted as the probability that $x_i \in class\ 1$.

9.2 Gaussian naive Bayesian classifier

Function (multiclass classification):

The input variables X_j can be discrete or continuous. The output variable Y is discrete/categorical and the number of categories is greater than two. Our goal here is to learn a classification function $f: X \rightarrow Y$.

Model (Gaussian naive Bayesian model):

We want to learn a model for $P(Y|X)$ which can then be used for classification, such that, $class(x_i) = \arg max_k P(Y = k|x_i)$. According to Bayes' theorem, $P(Y|X)$ can be expressed as:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{9.10}$$

which is interpreted in the classification context as:

$$P(class|data\ point) = \frac{P(data\ point|class)P(class)}{P(data\ point)} \tag{9.11}$$

and more generally it can be interpreted as:

$$P(hypothesis|evidence) = \frac{P(evidence|hypothesis)P(hypothesis)}{P(evidence)} \tag{9.12}$$

The components in Bayes' theorem are referred to as:

$$Posterior = \frac{Likelihood * Prior}{Marginal} \tag{9.13}$$

The likelihood $P(X|Y)$ can be computationally intractable if the number of input variables leads to combinatorial explosion. In order to simplify the computation of the

likelihood, we make an assumption that the input variables are conditionally independent given the class:

$$P(X_1|Y, X_2) = P(X_1|Y) \tag{9.14}$$

which significantly reduces the number of parameters of the model. Although this assumption simplifies the model, it is not necessarily true, and that is why this model is called “naive”. Under this assumption, the likelihood can be expressed as:

$$P(X|Y = k) = \prod_{j=1}^{J} P(X_j|Y = k) \tag{9.15}$$

This naive Bayesian model corresponds to the Bayesian network in (figure 9.3). The nodes in a Bayesian network represent variables, and the directed edges represent causal relationships or influences in general. The class variable Y influences the input variables $\{X_j\}$, which are conditionally independent from each other given Y, and that is why the input variables $\{X_j\}$ are not connected in the network. This is a generative model because it can be used to generate new data, by first sampling the prior distribution $P(Y)$, followed by sampling the conditional distributions $P(X_j|Y = k)$.

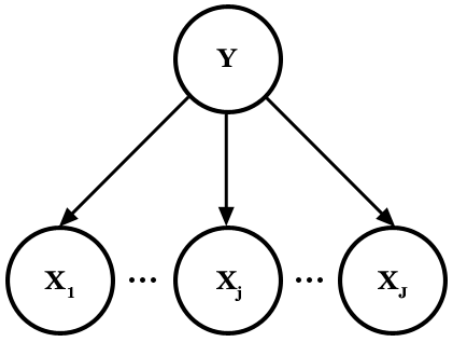


Figure 9.3 Bayesian network for the naive Bayesian model.

Next, we make an assumption that the probability distributions $P(X_j|Y = k)$ are univariate Gaussians $N(\mu_{jk}, \sigma_{jk}^2)$. Each Gaussian is defined by two parameters: mean μ_{jk} and variance σ_{jk}^2. We will see later how to learn these parameters from data.

Evaluator (maximum a posteriori - MAP):

For two particular Gaussian naive Bayesian models m_1 and m_2, how do we decide which one is better given the observed data? A model m_1 is better than another model m_2, if m_1 results in a larger posterior probability $P(Y|X)$.

Optimization (Gaussian density estimation):

We learn the "best" model by maximizing the posterior probability:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{9.10}$$

We can exclude $P(X)$ because it is constant:

$$P(Y|X) \propto P(X|Y)P(Y) \tag{9.16}$$

so maximizing the posterior $P(Y|X)$ is equivalent to maximizing the likelihood $P(X|Y)$ and the prior $P(Y)$.

The prior $P(Y)$ is very easy to compute. We simply count the number of data points in each class:

$$P(Y = k) = \frac{count(data\ points\ in\ class\ k)}{total\ number\ of\ data\ points} \tag{9.17}$$

The likelihood $P(X|Y)$ is computed as a product of univariate Gaussians:

$$P(X|Y=k) = \prod_{j=1}^{J} P(X_j|Y=k) = \prod_{j=1}^{J} Gaussian_{jk}\,(X_j|\mu_{jk}, \sigma_{jk}^2) \tag{9.18}$$

To compute the likelihood $P(X|Y)$, we need to estimate $(J * K)$ univariate Gaussian distributions, one Gaussian for each pair (X_j, k). Each Gaussian is defined by two parameters, mean μ_{jk} and variance σ_{jk}^2, so the total number of parameters is $(J * K * 2)$. For example, if there are two input variables $\{X_1, X_2\}$ and three output classes $k \in \{1, 2, 3\}$, then we need to learn $(2 * 3 * 2 = 12)$ parameters for 6 Gaussians. The MAP estimates for the Gaussian parameters are computed as follows:

$$mean_{jk} = \mu_{jk} = \frac{\sum_{i=1}^{I} X_j^i\, \delta(Y^i = k)}{\sum_{i=1}^{I} \delta(Y^i = k)} \tag{9.19}$$

$$variance_{jk} = \sigma_{jk}^2 = \frac{\sum_{i=1}^{I} (X_j^i - \mu_{jk})^2\, \delta(Y^i = k)}{(\sum_{i=1}^{I} \delta(Y^i = k)) - 1} \tag{9.20}$$

where j indexes the input variables $\{X_j\}$, k indexes the output classes, i indexes the training data points, and δ is an indicator function.

Now that our model is defined mathematically, we can implement it in pseudocode. You can see the final visual result in (figure 9.4).

First, we set the number of input variables J and the number of output classes K (pseudocode 9.5).

Pseudocode 9.5

```
J = 2 # two input variables
K = 3 # three output classes
```

Now let's estimate the prior probability for each class k, given the observed training data (pseudocode 9.6).

Pseudocode 9.6

```
def learn_prior(data, K):
    p = zeros(K)
    for k in range(K):
        p[k] = sum(data[:, -1] == k) / nrow(data)
    return p
learned_prior = learn_prior(train_data, 3)
```

Next, we need to learn $(J * K)$ univariate Gaussians, one Gaussian for each pair (X_j, k) of input variable X_j and class k (pseudocode 9.7).

Pseudocode 9.7

```
def learn_gaussians(train_data, J, K):
    mean = zeros(J * K).reshape(J, K)
    variance = zeros(J * K).reshape(J, K)
    classes = train_data[:, -1]
    for j in range(J):
        for k in range(K):
            mean[j, k] = sum(train_data[classes==k, j]) / sum(classes == k)
            variance[j, k] = sum((train_data[classes==k, j] - mean[j, k])^2) /
                (sum(classes == k) - 1)
    return mean, variance

learned_gaussians = learn_gaussians(train_data, J, K)
gauss_mean = learned_gaussians[0]; gauss_variance = learned_gaussians[1]
```

Next, we implement the likelihood function $P(x_i|Y = k)$ which tells us how likely it is for class k to generate the data point x_i (pseudocode 9.8).

Pseudocode 9.8

```
def learn_likelihood(xi, k):
    product_of_gaussians = 1
    for j in range(J):
        gaussian_ = norm.pdf(xi[j], gauss_mean[j,k], sqrt(gauss_variance[j,k]))
        product_of_gaussians = product_of_gaussians * gaussian_
    return product_of_gaussians
```

Finally, we implement the posterior probability function $P(Y = k|x_i)$ which will assign probabilities to each class k for a given data point x_i (pseudocode 9.9).

Pseudocode 9.9

```
def learn_posterior(xi, K):
    post = zeros(K)
    for k in range(K):
        post[k] = learned_prior[k] * learn_likelihood(xi, k)
    return post
```

Classification:

To classify a particular data point x_i, first we compute the posterior probabilities for all the K classes, and then we select the class with the highest probability:

$$class(x_i) = \arg max_k \, P(Y = k|x_i) \tag{9.21}$$

The final result of applying the learned Gaussian naive Bayesian classifier can be seen in (figure 9.4).

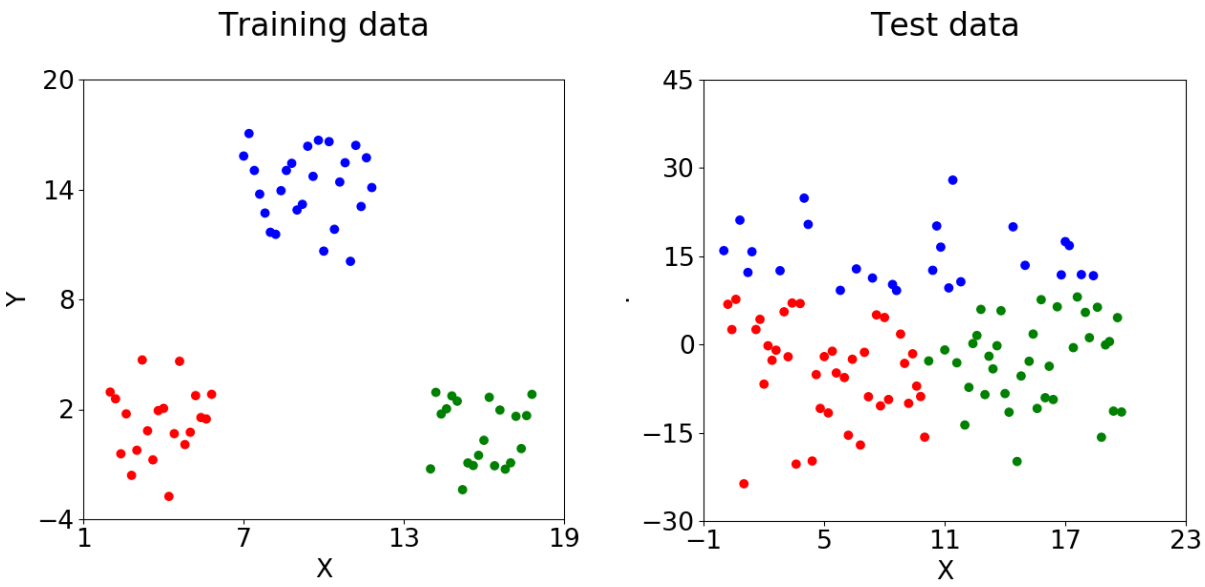


Figure 9.4 Gaussian naive Bayesian classifier.

The general idea behind the Gaussian naive Bayesian classifier:

The relationship between the input variables $\{X_j\}$ and the output variable Y is represented by the posterior probability $P(Y|X)$, such that, for a given data point x_i we assign the class k_{max} with the highest posterior probability $P(Y = k_{max}|x_i)$. The key question then is how to compute these posterior probabilities?

By using Bayes' theorem, the posterior $P(Y|X)$ is expressed in terms of: the likelihood $P(X|Y)$, the prior $P(Y)$, and the marginal $P(X)$. Maximizing the posterior is equivalent to maximizing the product of the likelihood and the prior, because the marginal is constant and can be ignored. The prior is estimated from data by counting the number of data points in each class. The key component of the whole model is the likelihood, which can be expressed as a product, due to the conditional independence assumption. We assume that each component in that product is a univariate Gaussian, which is estimated from the training data.

Chapter 10: Regression

Informal functional description:

The general regression function estimates the relationship between the input variables (also called independent variables, predictor variables, regressor variables, explanatory variables, etc.) and the output variables (also called dependent variables, predicted variables, regressand variables, explained variables, etc.). An input variable can be qualitative or quantitative.

Formal functional description:

A regression function is a mapping between input variables and output variables: $(Y_1, \dots, Y_e) = f(X_1, \dots, X_d)$.

Regression branches:

Regression is a broad area which can be divided in three branches based on the number of input and output variables:

# input variables	# output variables	regression branch
1	1	simple regression
> 1	1	multiple univariate regression
1	> 1	multivariate regression
> 1	> 1	multiple multivariate regression

I join multivariate regression and multiple multivariate regression in one branch, i.e., multivariate regression. So we have three branches: simple regression, multiple univariate regression, and multivariate regression. Next, each branch is divided in two sub-branches based on the type of the regression function: linear and non-linear.

Performance measures:

A standard performance measure for simple linear regression is the root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2} \tag{10.1}$$

This measure is relative and not absolute, i.e., it can be meaningfully interpreted only when comparing two different learned models, such that, the model with smaller RMSE is considered to be better.

10.1 Simple linear regression

Function (simple linear regression):

A simple linear regression function is a linear function of one input variable and one output variable.

Model (line):

We assume that the relationship between the input variable and the output variable is linear. Then, we want to know the type of the linear relationship. Positive linear relationship means that as the input variable increases, so does the output variable. Negative linear relationship means that as the input variable increases, the output variable decreases. Also, we want to know precisely how much the output variable changes as the input variable changes.

We can use a linear correlation measure to find out if the relationship between the input variable and the output variable is linear, to what degree is it linear, and is it positive or negative. For example, we can use the Pearson correlation measure:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}} \in [-1, 1] \tag{10.2}$$

where n is the number of data points, $\bar{x}$ is the average of the x values, and $\bar{y}$ is the average of the y values. The sign of r corresponds to the type of the linear relationship between x and y, i.e., positive or negative. The magnitude (i.e., the absolute value) of r corresponds to the strength of the linear relationship, such that higher magnitude of r corresponds to stronger linear relationship. If $r = 0$, x and y are not linearly related.

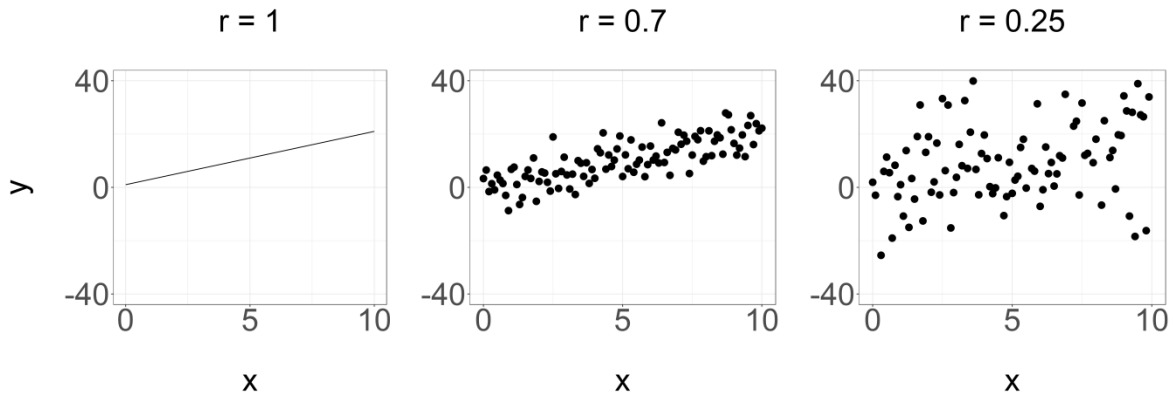


Figure 10.1 Pearson correlation coefficient r.

The model class for simple linear regression is a line (formula 10.3). The parameters c_0 and c_1 are learned from data.

$$y = c_0 + c_1 x \tag{10.3}$$

Evaluator (ordinary least squares - OLS):
Now that we have defined a model class (i.e., line), we need to find the best particular linear model according to the ordinary least squares (OLS) criterion:

$$(c_0^*, c_1^*) = \arg\min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \arg\min \sum_{i=1}^{n} (y_i - (c_0 + c_1 x_i))^2 \tag{10.4}$$

where y_i are the true observed values and $\hat{y}_i$ are our predicted values. The differences $(y_i - \hat{y}_i)$ are called residuals (i.e., errors). We square them in order to achieve two goals: 1) to get positive values so that it does not matter on which side of the line the data point lies, and 2) to emphasize larger errors.

Optimization (analytical solution of OLS):
Iterative optimization algorithms can be used (e.g., gradient descent), but in this case an analytical solution is available for (formula 10.4).

I will only sketch the optimization process here. First we find the partial derivatives of (formula 10.4) with respect to the two parameters c_0 and c_1. Then we set those derivatives to 0, and finally we solve for c_0 and c_1:

$$c_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} = r\frac{s_y}{s_x} \tag{10.5}$$

$$c_0 = \bar{y} - c_1\bar{x} \tag{10.6}$$

where r is the Pearson correlation coefficient (formula 10.2), and s_x and s_y are the standard deviations for x and y:

$$s_x = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}} \tag{10.7}$$

You can see in (formula 10.5) how the correlation coefficient r is related to the slope coefficient c_1. They carry different information, i.e., r tells you how close x and y are to a perfect linear relationship, and c_1 tells you how much y will change if you increase x by 1 unit.

Finally, we can implement a simple linear regression learner in (pseudocode 10.1). You can see the final visual result in (figure 10.2).

Pseudocode 10.1

```
mx = mean(x)
my = mean(y)
c1 = sum((x - mx) * (y - my)) / sum((x - mx)^2)
c0 = my - (c1 * mx)
```

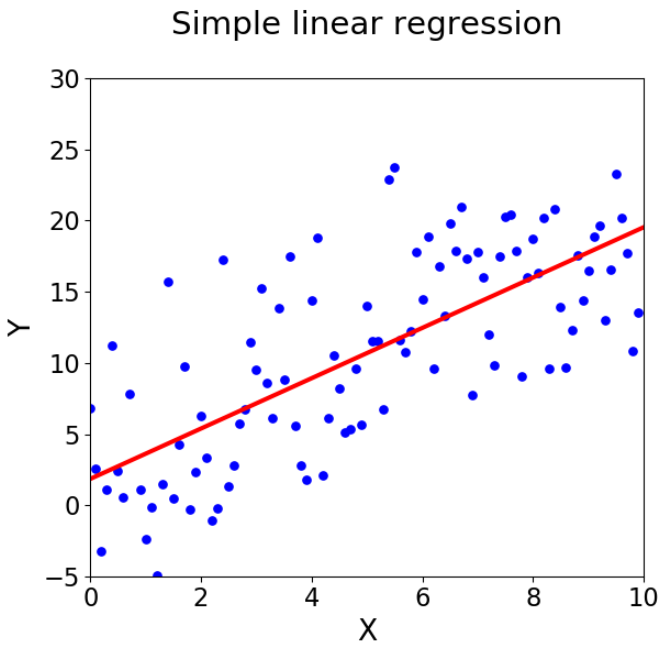


Figure 10.2 Simple linear regression.

The general idea behind simple linear regression:

First we make an assumption about the form of the relationship between x and y, i.e., we assume a linear relationship. Then we find the best particular linear model that minimizes the squared differences between the true values and the predicted values.

Chapter 11: Clustering

Informal functional description:
The general clustering function takes as input a set of objects, and it outputs a set of clusters and cluster memberships for each object. Clustering is an area in unsupervised machine learning where the goal is to discover structure in data without guidance on what to search for, unlike supervised machine learning where guidance is provided in the form of labels in the training data. We want to know if the initial population can be split into sub-populations and how these sub-populations relate to each other.

Formal functional description:
A clustering function is a mapping between input objects $\{x_i\}$ and output clusters $\{c_k\}$. In general, one input object x_i can belong to zero or more clusters $\{c_1, \ldots, c_k\}$ with corresponding degrees of membership $\{m_{i1}, \ldots, m_{ik}\}$.

Clustering branches:
Clustering is a broad area which can be divided into more branches and sub-branches:

- flat or hierarchical clustering
- hard or soft clustering

In flat clustering there is only one level of clusters, while in hierarchical clustering we can keep dividing clusters into more clusters. In hard clustering, a data point either belongs to a cluster or it does not, while in soft clustering a data point can belong to a cluster to a certain degree.

Performance measures:
There are intrinsic and extrinsic performance measures for clustering results. An extrinsic measure tells us how good the clustering result is relative to some context (e.g., another problem that uses the clustering result and provides basis for interpretation of the clustering result). An intrinsic measure tells us how good the clustering result is by

itself, independent from external entities. Standard intrinsic measures are intra-cluster similarity and inter-cluster dissimilarity. Intra-cluster similarity measures how similar are the data points inside each cluster, while inter-cluster dissimilarity measures how dissimilar are the data points from different clusters.

11.1 K-means clustering

Function (hard flat clustering):

The input is a set of data points $\{x_i\}$ and the number of clusters K to find in the data. The output for a given input data point x_i is the cluster it belongs to.

Model (centroid-based):

In centroid-based models, a cluster is represented by a data point called a centroid which is in the same space as the input data points. A data point x_i belongs to the cluster with the most similar centroid. We will use Euclidean distance to measure similarity between two data points:

$$d(p,q) = \sqrt{\sum_{d=1}^{D}(p_d - q_d)^2} \tag{11.1}$$

Evaluator (sum of squared residuals - SSR):

The evaluator, i.e., the objective function that we want to minimize is the sum of squared residuals, where the residuals are Euclidean distances between data points and their cluster centroids:

$$\sum_{k=1}^{K}\sum_{b=1}^{count(k)} d\left(x_b^k, centroid(k)\right) \tag{11.2}$$

where K is the number of clusters, $count(k)$ is the number of data points in cluster k, x_b^k is the bth data point in cluster k, and $centoid(k)$ is the centroid of cluster k.

Optimization (k-means iterative algorithm):

To minimize (formula 11.2) we will use an iterative algorithm called k-means. It converges to a local minimum, relative to the initial centroid points it starts with. We

can run the algorithm more times with different initial centroid points, and then we can pick the best solution.

The algorithm is very simple. First, we choose initial centroid data points. For example, we can choose random distinct K data points from our data. Then, two main steps are repeated until convergence. In the first step, we associate each data point x_i with its closest cluster centroid. In the second step, we recompute each cluster centroid as the average of its data points. The algorithm stops when there are no more changes to the cluster centroids. K-means clustering is implemented in (pseudocode 11.1) and the final visual result can be seen in (figure 11.1).

Pseudocode 11.1

```
def k_means(X, K):
    nrow = nrow(X); ncol = ncol(X)

    initial_centroids = sample(nrow, K, replace=False)
    centroids = X[initial_centroids]

    centroids_old = zeros(K, ncol)
    cluster_assignments = zeros(nrow)

    while (centroids_old != centroids):
        centroids_old = centroids

        # compute distances between data points and centroids
        dist_matrix = distance_matrix(X, centroids)

        # step 1: find the closest centroid for each data point
        for i in range(nrow):
```

```
        closest_centroid = min_index(dist_matrix[i])
        cluster_assignments[i] = closest_centroid

    # step 2: recompute centroids
    for k in range(K):
        centroids[k] = rows_mean(X[cluster_assignments == k])

return (centroids, cluster_assignments)
```

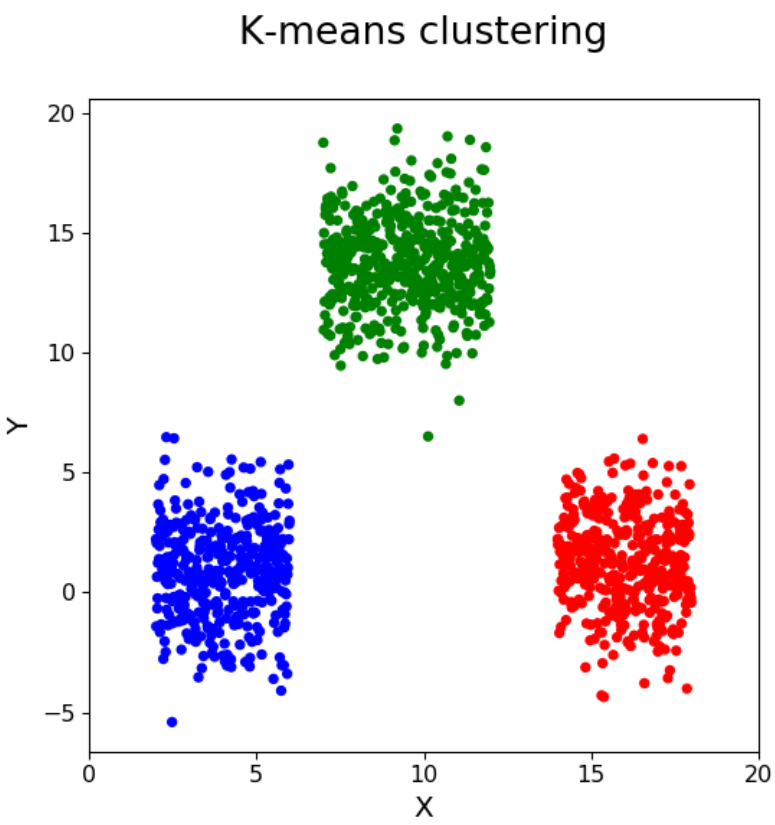


Figure 11.1 K-means clustering with $K = 3$.

The general idea behind k-means clustering:

We start with a set of data points that we want to cluster, and a number of clusters K that we want to find in the data. A cluster is represented by a data point in the same space as the input data points. The goal is to find clusters such that overall the data points have high intra-cluster similarity and high inter-cluster dissimilarity. Similarity between two data points can be measured in different ways (e.g., we used Euclidean distance). An iterative algorithm is used for optimization, which finds a local minimum relative to its starting point.

Chapter 12: Dimensionality reduction

Informal functional description:

The general dimensionality reduction function maps a set of variables to another smaller set of variables. Obviously, some of the initial information will be lost in this process, but we can try to minimize that loss and preserve a sufficient amount of the initial information.

Formal functional description:

A dimensionality reduction function is a mapping from a set of initial variables $\{X_j\}$ to a smaller set of new variables $\{X_k^{new}\}$.

Dimensionality reduction branches:

The broad area of dimensionality reduction can be divided in two branches: feature selection and feature extraction. In feature selection, some of the initial variables are excluded if they don't satisfy certain criteria. In feature extraction, a smaller set of new variables $\{X_k^{new}\}$ is constructed from the initial variables: $X_k^{new} = f(\{X_j\})$. Feature extraction can then be divided in two sub-branches based on the type of the function f: linear and non-linear feature extraction.

Performance measures (context-dependent):

The performance of a dimensionality reduction algorithm can be measured relative to other dimensionality reduction algorithms. Also, if a dimensionality reduction algorithm is a component of another problem (e.g., classification, regression, clustering), then its performance can be measured according to how much it contributes to the solution of that problem. In summary, performance is measured based on context.

12.1 Principal component analysis (PCA)

Function (linear feature extraction):

We learn a linear function f which maps the original variables to new variables:

$$X_k^{new} = f(\{X_j\}) = c_{k1}X_1 + \cdots + c_{kd}X_d \tag{12.1}$$

such that $1 \leq j, k \leq d$, where d is the dimensionality of the original input space.

Model (linear combination of the original variables):

What do we learn with dimensionality reduction? Well, first let's revisit the previous machine learning functions that were discussed in this book. In classification, we learn a model which tells us the class of each data point. In regression, we learn a model that can produce numeric values for an output variable, given values for input variables. In clustering, we learn a model that groups data points into clusters based on similarity. Now to get back to our initial question, in dimensionality reduction we learn a new smaller set of variables which describe the data sufficiently well. More specifically, in feature extraction, we learn a function that maps the original variables to new variables.

In this context, I use the term "model" to refer to the underlying principles and assumptions which define the form of the function f. In principal component analysis (PCA), the model is a linear combination of the original variables, i.e., each new variable is constructed as a linear combination of the original variables (formula 12.1). There are d new variables (same as the original space), but some are more important than others, so we can exclude the less important ones and we can reduce the dimensionality of the new space to $d^{new} < d$ dimensions. The new variables form a new space where each variable corresponds to a coordinate axis that is orthogonal to all the other axes.

Evaluator (variance of new variables):

For each new variable X_k^{new} we learn the coefficients $\{c_{k1}, \dots, c_{kd}\}$ in (formula 12.1) that maximize the variance of X_k^{new}. In other words, we want to preserve the maximum variance possible:

$$\arg max_{c_k} \left(var(X_k^{new})\right) = \arg max_{c_k} \left(var(c_k^T X)\right) \tag{12.2}$$

where c_k is a vector of coefficients for the linear combination for the new variable X_k^{new}, and X is the whole dataset represented in terms of the original variables.

It turns out that the "maximum variance" criterion is equivalent to the "minimum reconstruction error" criterion (i.e., minimizing the sum of the squared distances between the original data points and the projected data points).

Optimization (eigen decomposition):

The goal of the optimization process in PCA is to find "the best" coefficients of the linear combinations for constructing the new variables. This process can be described as eigen decomposition of the covariance matrix of the data, so essentially PCA is a linear algebra problem. The optimization process consists of the following steps:

1) Normalize the initial variables:
   a) Center the data (subtract the data mean from each data point):

$$x_i^{new} = x_i - \bar{x} \tag{12.3}$$

   b) Scale the data (divide the centered variables with their standard deviations):

$$x_{ij}^{new} = \frac{x_{ij}^{new}}{sd(X_j)} \tag{12.4}$$

2) Compute the covariance matrix of the normalized data.
3) Compute the eigenvectors and the eigenvalues of the covariance matrix.
4) Order the eigenvectors according to the eigenvalues in decreasing order.
5) Choose the number of new dimensions and select the first d^{new} eigenvectors.
6) Project the normalized data points onto the first d^{new} eigenvectors.

The eigenvectors of the covariance matrix are the orthonormal axes of the new coordinate system of the data space. Once we compute the eigenvectors, we order them by decreasing eigenvalues, and we select the first d^{new} eigenvectors. An eigenvalue indicates how much variance is preserved by projecting the data on the corresponding eigenvector. The larger the eigenvalue, the larger the variance preserved.

The implementation in (pseudocode 12.1) is very simple. I applied PCA to reduce the dimensionality of two-dimensional data to one dimension (figure 12.1).

Pseudocode 12.1

```
data_normal = normalize(data) # steps 1.a, 1.b
w, v = eigen(covariance(data_ normal)) # steps 2, 3, 4
pca_main_axis = v[:,0] # step 5
projected_data = data_ normal * pca_main_axis # step 6
```

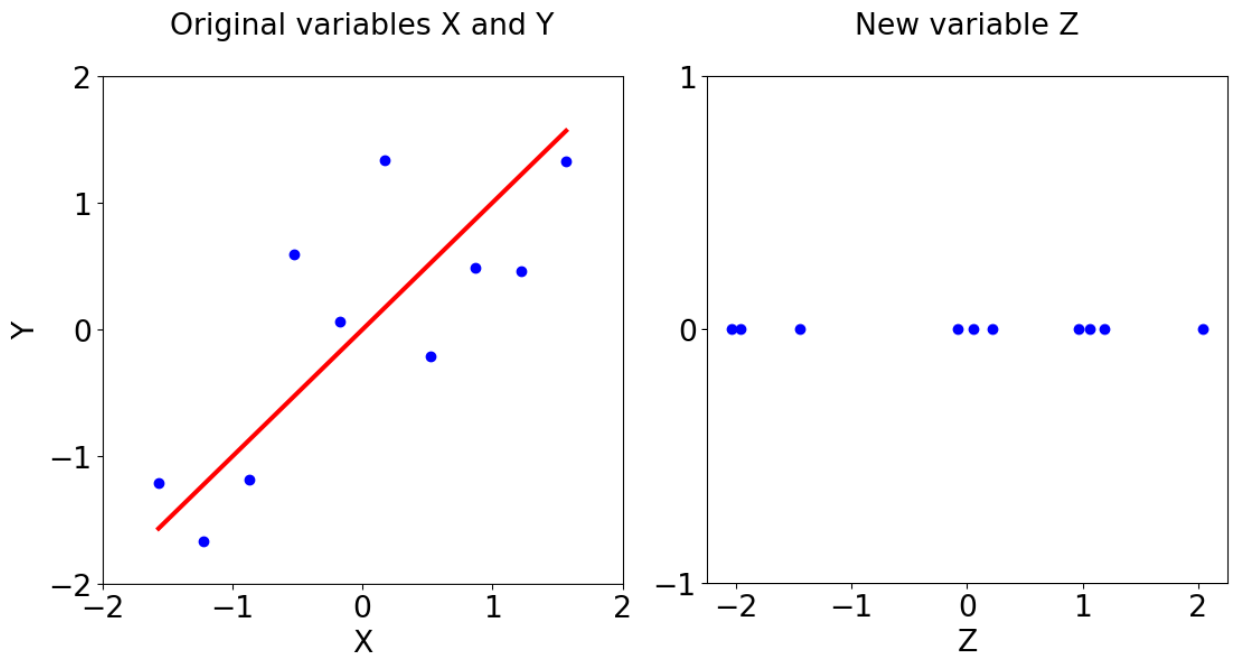


Figure 12.1 Principal component analysis (PCA) was used to reduce two-dimensional data to one-dimensional data.

The general idea behind PCA:

The new variables are constructed as linear combinations of the original variables. The coefficients of those linear combinations are learned from data by maximizing the variance of the new variables. Essentially, PCA rotates the coordinate system of the original data space. The new axes are ordered by importance (i.e., how much variance is preserved when the data is projected on them). The goal is to exclude as many of the new variables as possible, while preserving enough variance in the data. As we exclude more variables, our linear approximation of the data gets worse. So we have to find some middle ground, such that, a sufficient number of variables are excluded and a sufficient amount of variance is preserved.

Bibliography

[1] Balaguer, Mark. Platonism and Anti-Platonism in Mathematics.

[2] Barber, David. Bayesian Reasoning and Machine Learning.

[3] Bennett, Jonathan. A Philosophical Guide to Conditionals.

[4] Binmore, Ken. Game Theory: A Very Short Introduction.

[5] Blei, Dave. Principal Component Analysis (PCA).

[6] Booth, Wayne C. Colomb, Gregory G. Williams, Joseph M. The Craft of Research.

[7] Bradley, Darren. A Critical Introduction to Formal Epistemology.

[8] Brockman, John. This Idea Must Die.

[9] Brockman, John. This Will Make You Smarter.

[10] Brockman, John. What to Think About Machines that Think.

[11] Brownlee, Jason. Machine Learning Mastery Blog.

[12] Carnap, Rudolf. An Introduction to the Philosophy of Science.

[13] Carter, Matt. Minds and Computers.

[14] Cheng, Eugenia. Mathematics, Morally.

[15] Chrisman, Matthew. Philosophy for Everyone.

[16] Combinatorial analysis. Encyclopedia of Mathematics. URL: http://www.encyclopediaofmath.org/index.php?title=Combinatorial_analysis&oldid=35186

[17] Conery, John S. Computation is Symbol Manipulation.

[18] Craver, Carl and Tabery, James. Mechanisms in Science. The Stanford Encyclopedia of Philosophy (Spring 2017 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/spr2017/entries/science-mechanisms/>.

[19] Craver, Carl F. Explaining the Brain.

[20] Davis, Philip J. Hersh, Reuben. The Mathematical Experience.

[21] Descartes, René. Discourse on the Method.

[22] Devlin, Keith. Introduction to Mathematical Thinking.

[23] Devlin, Keith. Mathematics Education for a New Era.

[24] Devlin, Keith. The Language of Mathematics.

[25] Devlin, Keith. The Math Gene.

[26] Domingos, Pedro. A Few Useful Things to Know about Machine Learning.
[27] Domingos, Pedro. The Master Algorithm.
[28] Effingham, Nikk. An Introduction to Ontology.
[29] Einstein, Albert. Induction and Deduction in Physics.
[30] Floridi, Luciano. Information: A Very Short Introduction.
[31] Gaukroger, Stephen. Objectivity: A Very Short Introduction.
[32] Ghodsi, Ali. Dimensionality Reduction A Short Tutorial.
[33] Giaquinto, Marcus. Visual Thinking in Mathematics.
[34] Giordano, Frank. A First Course in Mathematical Modeling.
[35] Goertzel, Ben. Engineering General Intelligence, Part 1.
[36] Goldreich, Oded. On our duties as scientists.
[37] Grayling, A. C. Russell: A Very Short Introduction.
[38] Grayling, A. C. Wittgenstein: A Very Short Introduction.
[39] Hadamard, Jacques. The Psychology of Invention in the Mathematical Field.
[40] Halmos, Paul R. Naive Set Theory.
[41] Hamming, Richard W. A Stroke of Genius: Striving for Greatness in All You Do.
[42] Hardy, G. H. A Mathematician's Apology.
[43] Harnad, Stevan. Computation is just interpretable symbol manipulation: cognition isn't.
[44] Hawking, Stephen. The Grand Design.
[45] Herbert, Simon. The Sciences of the Artificial.
[46] Hindley, Roger J. Seldin, Jonathan P. Lambda-Calculus and Combinators: An Introduction.
[47] Hofstadter, Douglas. Gödel, Escher, Bach.
[48] Holland, John H. Complexity: A Very Short Introduction.
[49] Hornik, Kurt. Principal Component Analysis using R.
[50] Horst, Steven. Beyond Reduction.
[51] Hurley, Patrick J. A Concise Introduction to Logic.
[52] Jaynes, E. T. Bretthorst, Larry G. Probability Theory: The Logic of Science.
[53] Jesseph, Douglas M. Berkeley's Philosophy of Mathematics.
[54] Kapetosu. Definition: Russell's paradox. YouTube, 12 October 2017.

[55] Kirk, Roger E. Statistics: An Introduction.
[56] Korb, Kevin B. Machine Learning as Philosophy of Science.
[57] Kuhn, Thomas. The Structure of Scientific Revolutions.
[58] Lemos, Noah. An Introduction to the Theory of Knowledge.
[59] Levin, Janet. Functionalism. The Stanford Encyclopedia of Philosophy (Winter 2016 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/win2016/entries/functionalism/>.
[60] Lipschutz, Seymour. Schaum's Outline of Set Theory and Related Topics.
[61] Lockhart, Paul. A Mathematician's Lament.
[62] Matloff, Norman. The Art of R Programming.
[63] McGrayne, Sharon Bertsch. The Theory That Would Not Die.
[64] McInerny, D. Q. Being Logical.
[65] Mitchell, Melanie. Complexity.
[66] Mitchell, Tom. Generative and Discriminative Classifiers: Naive Bayes and Logistic Regression.
[67] Mitchell, Tom. The Discipline of Machine Learning.
[68] Mohri, Mehryar. Foundations of Machine Learning.
[69] Morrison, Robert. The Cambridge Handbook of Thinking and Reasoning.
[70] Mumford, Stephen. Causation: A Very Short Introduction.
[71] Mumford, Stephen. Metaphysics: A Very Short Introduction.
[72] Neuhauser, C. Estimating the Mean and Variance of a Normal Distribution.
[73] Oaksford, Mike. Bayesian Rationality.
[74] Okasha, Samir. Philosophy of Science: A Very Short Introduction.
[75] Papineau, David. Philosophical Devices.
[76] Peterson, Martin. An Introduction to Decision Theory.
[77] Pinter, Charles C. A Book of Abstract Algebra (2nd edition).
[78] Poincaré, Henri. Science and Method.
[79] Pólya, George. How to solve it.
[80] Pólya, George. Induction and Analogy in Mathematics.
[81] Pólya, George. Patterns of Plausible Inference.
[82] Popper, Karl. The Logic of Scientific Discovery.
[83] Prince, Simon J. D. Computer Vision: Models, Learning, and Inference.

[84] Pritchard, Duncan. What is this thing called Knowledge?
[85] Rapaport, William. Philosophy of Computer Science.
[86] Reid, Mark. Anatomy of a Learning Problem.
[87] Russell, Bertrand. Introduction to Mathematical Philosophy.
[88] Russell, Bertrand. Principles of Mathematics.
[89] Russell, Bertrand. The Problems of Philosophy.
[90] Sawyer, W. W. Prelude to Mathematics.
[91] Shalev-Shwartz, Shai. Ben-David, Shai. Understanding Machine Learning: From Theory to Algorithms.
[92] Shapiro, Stewart. Thinking about Mathematics: The Philosophy of Mathematics.
[93] Shmueli, Galit. To Explain or to Predict?
[94] Smith, Leonard. Chaos: A Very Short Introduction.
[95] Smith, Lindsay. A tutorial on Principal Components Analysis.
[96] Smith, Peter Godfrey. Theory and Reality.
[97] Smith, Peter. An Introduction to Gödel's Theorems.
[98] Sober, Elliott. The Multiple Realizability Argument against Reductionism.
[99] Stone, James. Bayes's Rule: A Tutorial Introduction to Bayesian Analysis.
[100] Suber, Peter. Formal Systems and Machines.
[101] Tao, Terence. Solving Mathematical Problems.
[102] Tao, Terence. What is Good Mathematics?
[103] Tarski, Alfred. Introduction to Logic.
[104] Taylor, Courtney. What Are Probability Axioms? ThoughtCo, Sep. 28, 2017, thoughtco.com/what-are-probability-axioms-3126567.
[105] Thagard, Paul. Computational Philosophy of Science.
[106] Thurston, William. On Proof and Progress in Mathematics.
[107] VanderPlas, Jake. Python Data Science Handbook.
[108] Westerhoff, Jan. Reality: A Very Short Introduction.
[109] Whorf, Benjamin Lee. Language, Thought and Reality.
[110] Wikipedia. Data.
[111] Wikipedia. Natural language.
[112] Wikipedia. Probability Interpretations.
[113] Wikipedia. Set theory.

[114] Wikipedia. Symbolic Communication.