EXPERT INSIGHT

Artificial Intelligence with Python

Your complete guide to building intelligent apps using Python 3.x

Second Edition

Alberto Artasanchez
Prateek Joshi

Packt

Artificial Intelligence with Python

Second Edition

Your complete guide to building intelligent apps using Python 3.x

Alberto Artasanchez

Prateek Joshi

Packt

BIRMINGHAM - MUMBAI

Artificial Intelligence with Python

Second Edition

Producer: Tushar Gupta
Acquisition Editor – Peer Reviews: Suresh Jain
Content Development Editor: Ian Hough
Technical Editor: Aniket Shetty
Project Editor: Carol Lewis
Proofreader: Safis Editing
Indexer: Tejal Daruwale Soni
Presentation Designer: Sandip Tadge

First published: January 2017

Second edition: January 2020

Production reference: 3020620

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-83921-953-5

`www.packt.com`

Table of Contents

Preface

Recent advances in **artificial intelligence** (**AI**) have placed great power into the hands of humans. With great power comes a proportional level of responsibility. Self-driving cars, chatbots, and increasingly accurate predictions of the future are but a few examples of AI's ability to supercharge humankind's capacity for growth and advancement.

AI is becoming a core, transformative path that is changing the way we think about every aspect of our lives. It is impacting industry. It is becoming pervasive and embedded in our everyday lives. Most excitingly, this is a field that is still in its infancy: the AI revolution has only just begun.

As we collect more and more data and tackle that data with better and faster algorithms, we can use AI to build increasingly accurate models and to answer increasingly complex, previously intractable questions.

From this, it will come as no surprise that the ability to work with and fully utilize AI will be a skill that is set only to increase in value. In this book, we explore various real-world scenarios and learn how to apply relevant AI algorithms to a wide swath of problems.

The book starts with the most basic AI concepts and progressively builds on these concepts to solve increasingly difficult problems. It will use the initial knowledge gleaned during the beginning chapters as a foundation to allow the reader to explore and tackle some of the more complicated problems in AI. By the end of the book, the reader will have gained a solid understanding of many AI techniques and will have gained confidence about when to use these techniques.

We will start by talking about various realms of AI. We'll then move on to discuss more complex algorithms, such as extremely random forests, Hidden Markov Models, genetic algorithms, artificial neural networks, convolutional neural networks, and so on.

This book is for Python programmers looking to use AI algorithms to create real-world applications. This book is friendly to Python beginners, but familiarity with Python programming would certainly be helpful so you can play around with the code. It is also useful to experienced Python programmers who are looking to implement artificial intelligence techniques.

You will learn how to make informed decisions about the type of algorithms you need to use and how to implement those algorithms to get the best possible results. If you want to build versatile applications that can make sense of images, text, speech, or some other form of data, this book on artificial intelligence will definitely come to your rescue!

Who this book is for

This book is for Python developers who want to build real-world artificial intelligence applications. This book is friendly to Python beginners, but being familiar with Python would be useful to play around with the code. It will also be useful for experienced Python programmers who are looking to use artificial intelligence techniques in their existing technology stacks.

What this book covers

Chapter 1, *Introduction to Artificial Intelligence*

This chapter provides some basic definitions and groupings that will be used throughout the book. It will also provide an overall classification of the artificial intelligence and machine learning fields as they exist today.

Chapter 2, *Fundamental Use Cases for Artificial Intelligence*

Artificial Intelligence is a fascinating topic and a vast field of knowledge. In its current state it generates more questions than it answers, but there are certainly many places where artificial intelligence is being applied, in many instances without us even realizing. Before we delve into the fundamental algorithms that drive AI, we will analyze some of the most popular use cases for the technology as of today.

Chapter 3, *Machine Learning Pipelines*

Model training is only a small piece of the machine learning process. Data scientists often spend a significant amount of time cleansing, transforming, and preparing data to get it ready to be consumed by an AI model. Since data preparation is such a time-consuming activity, we will present state-of-the-art techniques to facilitate this activity as well as other components that a well-designed production data pipeline should possess.

Chapter 4, Feature Selection and Feature Engineering

Model performance can be improved by selecting the right dimensions to pass to the model as well as discovering new dimensions that can enrich the input datasets. This chapter will demonstrate how new features can be created from existing ones as well as from external sources. It will also cover how to eliminate redundant or low-value features.

Chapter 5, Classification and Regression Using Supervised Learning

This chapter defines in detail supervised learning. It provides a taxonomy of the various methods and algorithms for problems that fall under this classification.

Chapter 6, Predictive Analytics with Ensemble Learning

Ensemble learning is a powerful technique that allows you to aggregate the power of individual models. This chapter goes over the different ensemble methods as well as guidance on when to use each of them. Finally, the chapter will cover how to apply these techniques to real-world event prediction.

Chapter 7, Detecting Patterns with Unsupervised Learning

This chapter will explore the concepts of clustering and data segmentation and how they are related to unsupervised learning. It will also cover how to perform clustering and how to apply various clustering algorithms. It will show several examples that allow the reader to visualize how these algorithms work. Lastly, it will cover the application of these algorithms to perform clustering and segmentation in real-world situations.

Chapter 8, Building Recommender Systems

This chapter will demonstrate how to build recommender systems. It will also show how to persist user preferences. It will cover the concepts of nearest neighbor search and collaborative filtering. Finally, there will be an example showing how to build a movie recommendation system.

Chapter 9, Logic Programming

This chapter will cover how to write programs using logic programming. It will discuss various programming paradigms and see how programs are constructed with logic programming. It will highlight the building blocks of logic programming and see how to solve problems in this domain. Finally, various Python program implementations will be built for various solvers that tackle a variety of problems.

Chapter 10, Heuristic Search Techniques

This chapter covers heuristic search techniques. Heuristic search techniques are used to search through the solution space to come up with answers. The search is conducted using heuristics that guide the search algorithm. Heuristics allow the algorithm to speed up the process, which would otherwise take a long time to arrive at the solution.

Chapter 11, Genetic Algorithms and Genetic Programming

We will discuss the basics of genetic programming and its importance in the field of AI. We will learn how to solve simple problems using genetic algorithms. We will understand some underlying concepts that are used to do genetic programming. We will then see how to apply this to a real-world problem.

Chapter 12, Artificial Intelligence on the Cloud

The cloud enables us to accelerate AI development, workloads, and deployment. In this chapter, we will explore the different offerings from the most popular vendors that enable and accelerate AI projects.

Chapter 13, Building Games with Artificial Intelligence

This chapter will cover how to build games using artificial intelligence techniques. Search algorithms will be used to develop winning game strategies and tactics. Finally, intelligent bots will be built for a variety of games.

Chapter 14, Building a Speech Recognizer

This chapter will cover how to perform speech recognition. It will show how to process speech data and extract features from it. Finally, it will demonstrate how to use the extracted features to build a speech recognition system.

Chapter 15, Natural Language Processing

This chapter will focus on the important area of AI known as **Natural Language Processing** (**NLP**). It will discuss various concepts such as tokenization, stemming, and lemmatization to process text. It will also cover how to build a Bag of Words model and use it to classify text. It will demonstrate how machine learning can be used to analyze the sentiment of a given sentence. Lastly, it will show topic modeling and go over the implementation of a system to identify topics in a document.

Chapter 16, Chatbots

Chatbots can help to save money and better serve customers by increasing productivity and deflecting calls. In this chapter, we will cover the basics of chatbots and the tools available to build them.

Finally, we will build a full-blown chatbot from scratch that will implement a real-world use case including error handling, connecting it to an external API, and deploying the chatbot.

Chapter 17, Sequential Data and Time Series Analysis

We will discuss the concept of probabilistic reasoning. We will learn how to apply that concept to build models for sequential data. We will learn about the various characteristics of time-series data. We will discuss Hidden Markov Models and how to use them to analyze sequential data. We will then use this technique to analyze stock market data.

Chapter 18, Image Recognition

We will discuss how to work with images in this chapter. We will learn how to detect and track objects in a live video. We will then learn how to apply those techniques to track parts of the human face.

Chapter 19, Neural Networks

We will discuss artificial neural networks. We will learn about perceptrons and see how they are used to build neural networks. We will learn how to build single-layered and multi-layered neural networks. We will discuss how a neural network learns about the training data and builds a model. We will learn about the cost function and backpropagation. We will then use these techniques to perform optical character recognition.

Chapter 20, Deep Learning with Convolutional Neural Networks

We will discuss the basics of deep learning in this chapter. The reader will be introduced to various concepts in convolutional neural networks and how they can be used for image recognition. We will discuss various layers in a convolutional neural network. We will then use these techniques to build a real-world application.

Chapter 21, Recurrent Neural Networks and Other Deep Learning Models

This chapter will continue to cover other types of deep learning algorithms. It will start with coverage of recurrent neural networks and it will then cover newer algorithms such as the Attention, Self-Attention, and Transformer models. This chapter will cover the use cases where these networks are used and the advantages of using these kinds of model architecture, as well as their limitations. Finally, the techniques discussed will be used to build a real-world application.

Chapter 22, Creating Intelligent Agents with Reinforcement Learning

This chapter will define **reinforcement learning** (**RL**) as well as cover the components within an RL model. It will detail the techniques used to build RL systems. Finally, it will demonstrate how to build learning agents that can learn by interacting with the environment.

Chapter 23, Artificial Intelligence and Big Data

This chapter will analyze how big data techniques can be applied to accelerate machine learning pipelines as well as covering different techniques that can be used to streamline dataset ingestion, transformation, and validation. Finally, it will walk the reader through an actual example using Apache Spark to demonstrate the concepts covered in the chapter.

What you need for this book

This book is focused on AI in Python as opposed to Python itself. We have used Python 3 to build various applications. We focus on how to utilize various Python libraries in the best possible way to build real world applications. In that spirit, we have tried to keep all of the code as friendly and readable as possible. We feel that this will enable our readers to easily understand the code and readily use it in different scenarios.

Download the example code files

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `http://www.packtpub.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the on-screen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows

- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Artificial-Intelligence-with-Python-Second-Edition`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781839219535_ColorImages.pdf`.

Conventions used

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. For example: "The `n_estimators` parameter refers to the number of trees that will be constructed."

A block of code is set as follows:

```
# Create label encoder and fit the labels
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
# Create label encoder and fit the labels
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
```

Any command-line input or output is written as follows:

```
$ python3 random_forests.py --classifier-type rf
```

Bold: Indicates a new term, an important word, or words that you see on the screen, for example, in menus or dialog boxes, also appear in the text like this. For example: "**Supervised learning** refers to the process of building a machine learning model that is based on labeled training data."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book we would be grateful if you would report this to us. Please visit, `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Introduction to Artificial Intelligence

In this chapter, we are going to discuss the concept of **artificial intelligence** (**AI**) and how it's applied in the real world. We spend a significant portion of our everyday life interacting with smart systems. This can be in the form of searching for something on the internet, biometric facial recognition, or converting spoken words to text. AI is at the heart of all this and it's becoming an important part of our modern lifestyle. All these systems are complex real-world applications and AI solves these problems with mathematics and algorithms. Throughout the book, we will learn the fundamental principles that can be used to build such applications. Our overarching goal is to enable you to take up new and challenging AI problems that you might encounter in your everyday life.

By the end of this chapter, you will know:

- What is AI and why do we need to study it?
- What are some applications of AI?
- A classification of AI branches
- The five tribes of machine learning
- What is the Turing test?
- What are rational agents?
- What are General Problem Solvers?
- How to build an intelligent agent
- How to install Python 3 and related packages

What is AI?

How one defines AI can vary greatly. Philosophically, what is "intelligence?" How one perceives intelligence in turn defines its artificial counterpart. A broad and optimistic definition of the field of AI could be: "the area of computer science that studies how machines can perform tasks that would normally require a sentient agent." It could be argued from such a definition that something as simple as a computer multiplying two numbers is "artificial intelligence." This is because we have designed a machine capable of taking an input and independently producing a logical output that usually would require a living entity to process.

A more skeptical definition might be more narrow, for example: "the area of computer science that studies how machines can closely imitate human intelligence." From such definition skeptics may argue that what we have today is not artificial intelligence. Up until now, they have been able to point to examples of tasks that computers cannot perform, and therefore claim that computers cannot yet "think" or exhibit artificial intelligence if they cannot satisfactorily perform such functions.

This book leans towards the more optimistic view of AI and we prefer to marvel at the number of tasks that a computer can currently perform.

In our aforementioned multiplication task, a computer will certainly be faster and more accurate than a human if the two numbers are large enough. There are other areas where humans can currently perform much better than computers. For example, a human can recognize, label, and classify objects with a few examples, whereas currently a computer might require thousands of examples to perform at the same level of accuracy. Research and improvement continue relentlessly, and we will continue to see computers solving more and more problems that just a few years ago we could only dream of them solving. As we progress in the book, we will explore many of these use cases and provide plenty of examples.

An interesting way to consider the field of AI is that AI is in some ways one more branch of science that is studying the most fascinating computer we know: the brain. With AI, we are attempting to reflect some of the systems and mechanics of the brain within computing, and thus find ourselves borrowing from, and interacting with, fields such as neuroscience.

Why do we need to study AI?

AI can impact every aspect of our lives. The field of AI tries to understand patterns and behaviors of entities. With AI, we want to build smart systems and understand the concept of intelligence as well. The intelligent systems that we construct are very useful in understanding how an intelligent system like our brain goes about constructing another intelligent system.

Let's look at how our brain processes information:

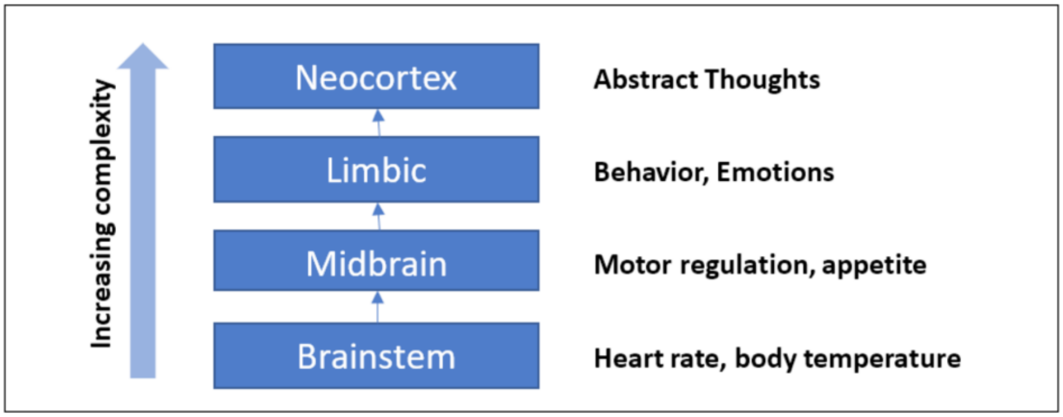


Figure 1: Basic brain components

Compared to some other fields such as mathematics or physics that have been around for centuries, AI is relatively in its infancy. Over the last couple of decades, AI has produced some spectacular products such as self-driving cars and intelligent robots that can walk. Based on the direction in which we are heading, it's obvious that achieving intelligence will have a great impact on our lives in the coming years.

We can't help but wonder how the human brain manages to do so much with such effortless ease. We can recognize objects, understand languages, learn new things, and perform many more sophisticated tasks with our brain. How does the human brain do this? We don't yet have many answers to that question. When you try to replicate tasks that the brain performs, using a machine, you will see that it falls way behind! Our own brains are far more complex and capable than machines, in many respects.

When we try to look for things such as extraterrestrial life or time travel, we don't know if those things exist; we're not sure if these pursuits are worthwhile. The good thing about AI is that an idealized model for it already exists: our brain is the holy grail of an intelligent system! All we have to do is to mimic its functionality to create an intelligent system that can do something similarly to, or better than, our brain.

Let's see how raw data gets converted into intelligence through various levels of processing:

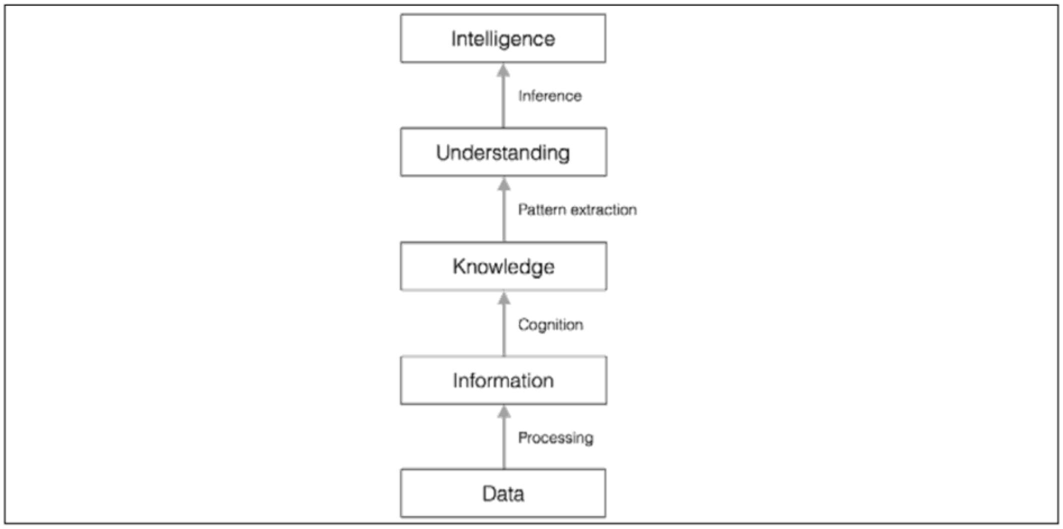


Figure 2: Conversion of data into intelligence

One of the main reasons we want to study AI is to automate many things. We live in a world where:

- We deal with huge and insurmountable amounts of data. The human brain can't keep track of so much data.
- Data originates from multiple sources simultaneously. The data is unorganized and chaotic.
- Knowledge derived from this data must be updated constantly because the data itself keeps changing.
- The sensing and actuation must happen in real-time with high precision.

Even though the human brain is great at analyzing things around us, it cannot keep up with the preceding conditions. Hence, we need to design and develop intelligent machines that can do this. We need AI systems that can:

- Handle large amounts of data in an efficient way. With the advent of Cloud Computing, we are now able to store huge amounts of data.
- Ingest data simultaneously from multiple sources without any lag. Index and organize data in a way that allows us to derive insights.
- Learn from new data and update constantly using the right learning algorithms. Think and respond to situations based on the conditions in real time.
- Continue with tasks without getting tired or needing breaks.

AI techniques are actively being used to make existing machines smarter so that they can execute faster and more efficiently.

Branches of AI

It is important to understand the various fields of study within AI so that we can choose the right framework to solve a given real-world problem. There are several ways to classify the different branches of AI:

- Supervised learning vs. unsupervised learning vs. reinforcement learning
- Artificial general intelligence vs. narrow intelligence
- By human function:
    - Machine vision
    - Machine learning
    - Natural language processing
    - Natural language generation

Following, we present a common classification:

- **Machine learning and pattern recognition**: This is perhaps the most popular form of AI out there. We design and develop software that can learn from data. Based on these learning models, we perform predictions on unknown data. One of the main constraints here is that these programs are limited to the power of the data.

If the dataset is small, then the learning models would be limited as well. Let's see what a typical machine learning system looks like:

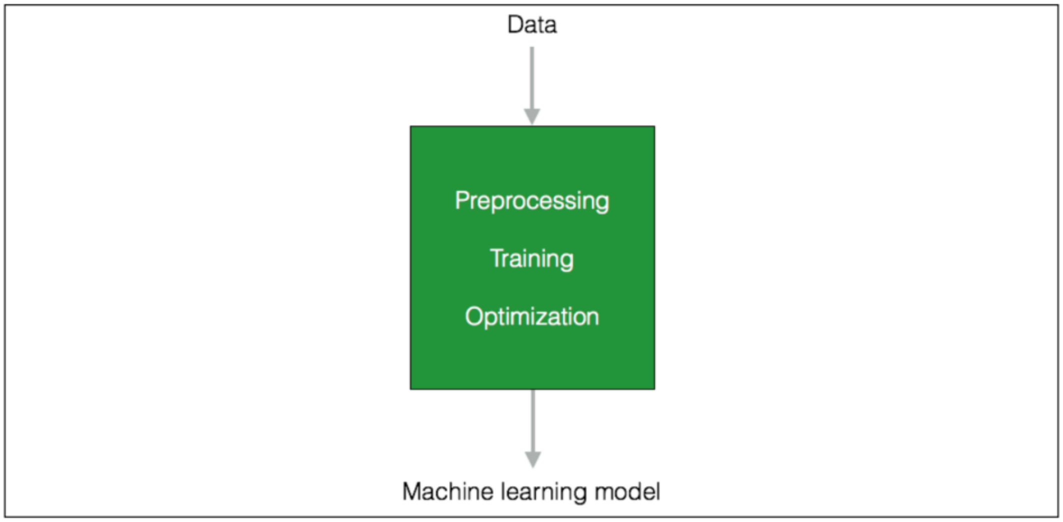


Figure 3: A typical computer system

When a system receives a previously unseen data point, it uses the patterns from previously seen data (the training data) to make inferences on this new data point. For example, in a facial recognition system, the software will try to match the pattern of eyes, nose, lips, eyebrows, and so on in order to find a face in the existing database of users.

- **Logic-based AI**: Mathematical logic is used to execute computer programs in logic-based AI. A program written in logic-based AI is basically a set of statements in logical form that expresses facts and rules about a problem domain. This is used extensively in pattern matching, language parsing, semantic analysis, and so on.
- **Search**: Search techniques are used extensively in AI programs. These programs examine many possibilities and then pick the most optimal path. For example, this is used a lot in strategy games such as chess, networking, resource allocation, scheduling, and so on.
- **Knowledge representation**: The facts about the world around us need to be represented in some way for a system to make sense of them. The languages of mathematical logic are frequently used here. If knowledge is represented efficiently, systems can be smarter and more intelligent. Ontology is a closely related field of study that deals with the kinds of objects that exist.

It is a formal definition of the properties and relationships of the entities that exist in a domain. This is usually done with a taxonomy or a hierarchical structure of some kind. The following diagram shows the difference between information and knowledge:

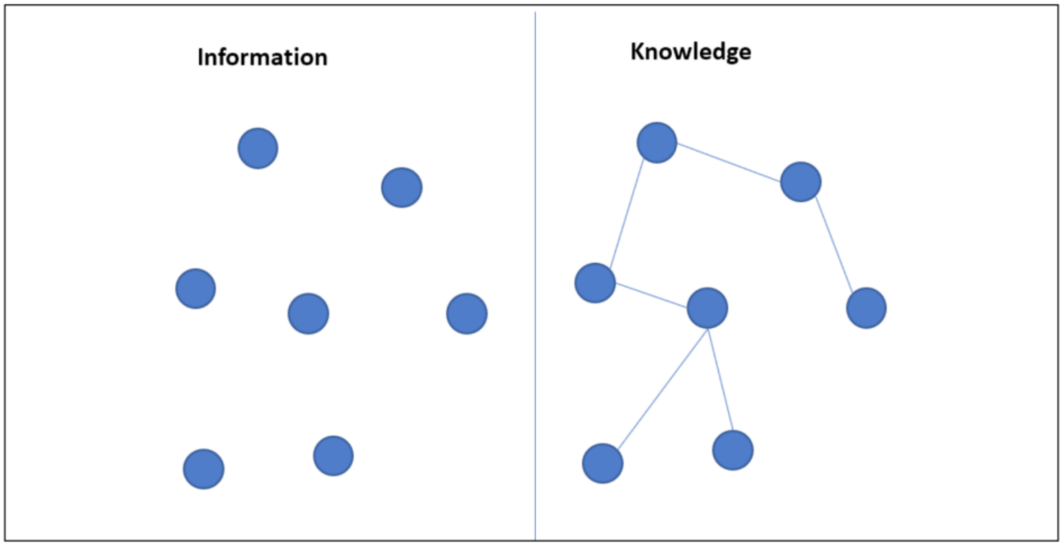


Figure 4: Information vs. Knowledge

- **Planning**: This field deals with optimal planning that gives us maximum returns with minimal costs. These software programs start with facts about the situation and a statement of a goal. These programs are also aware of the facts of the world, so that they know what the rules are. From this information, they generate the most optimal plan to achieve the goal.
- **Heuristics**: A heuristic is a technique used to solve a given problem that's practical and useful in solving the problem in the short term, but not guaranteed to be optimal. This is more like an educated guess on what approach we should take to solve a problem. In AI, we frequently encounter situations where we cannot check every single possibility to pick the best option. Thus, we need to use heuristics to achieve the goal. They are used extensively in AI in fields such as robotics, search engines, and so on.
- **Genetic programming**: Genetic programming is a way to get programs to solve a task by mating programs and selecting the fittest. The programs are encoded as a set of genes, using an algorithm to get a program that can perform the given task well.

The five tribes of machine learning

Machine learning can be further classified in a variety of ways. One of our favorite classifications is the one provided by Pedro Domingos in his book *The Master Algorithm*. In his book, he classifies machine learning by the field of science that sprouted the ideas. For example, genetic algorithms sprouted from Biology concepts. Here are the full classifications, the name Domingos uses for the tribes, and the dominant algorithms used by each tribe, along with noteworthy proponents:

Tribe	**Origins**	**Dominant algorithm**	**Proponents**
Symbolists	Logic and Philosophy	Inverse deduction	Tom Mitchell Steve Muggleton Ross Quinlan
Connectionists	Neuroscience	Backpropagation	Yan LeCun Geoffrey Hinton Yoshua Bengio
Evolutionaries	Biology	Genetic programming	John Koza John Holland Hod Lipson
Bayesians	Statistics	Probabilistic inference	David Heckerman Judea Pearl Michael Jordan
Analogizers	Psychology	Kernel machines	Peter Hart Vladimir Vapnik Douglas Hofstadter

Symbolists – Symbolists use the concept of induction or inverse deduction as their main tool. When using induction, instead of starting with a premise and looking for conclusions, inverse deduction starts with a set of premises and conclusions and works backwards to fill in the missing pieces.

An example of deduction:

Socrates is human + All humans are mortal = What can be deduced? (Socrates is mortal)

An example of induction:

Socrates is human + ?? = Socrates is mortal (Humans are mortal?)

Connectionists – Connectionists use the brain, or at least our very crude understanding of the brain, as their primary tool – mainly neural networks. Neural networks are a type of algorithm, modeled loosely after the brain, which are designed to recognize patterns. They can recognize numerical patterns contained in vectors. In order to use them, all inputs, be they images, sound, text, or time series need to be translated into these numerical vectors. It is hard to open a magazine or a news site and not read about examples of "deep learning." Deep learning is a specialized type of a neural network.

Evolutionaries – Evolutionaries focus on using the concepts of evolution, natural selection, genomes, and DNA mutation and applying them to data processing. Evolutionary algorithms will constantly mutate, evolve, and adapt to unknown conditions and processes.

Bayesians – Bayesians focus on handling uncertainty using probabilistic inference. Vision learning and spam filtering are some of the problems tackled by the Bayesian approach. Typically, Bayesian models will take a hypothesis and apply a type of "a priori" reasoning, assuming that some outcomes will be more likely. They then update a hypothesis as they see more data.

Analogizers – Analogizers focus on techniques that find similarities between examples. The most famous analogizer model is the *k-nearest neighbor* algorithm.

Defining intelligence using the Turing test

The legendary computer scientist and mathematician, *Alan Turing*, proposed the Turing test to provide a definition of intelligence. It is a test to see if a computer can learn to mimic human behavior. He defined intelligent behavior as the ability to achieve human-level intelligence during a conversation. This performance should be enough to trick an interrogator into thinking that the answers are coming from a human.

To see if a machine can do this, he proposed a test setup: he proposed that a human should interrogate the machine through a text interface. Another constraint is that the human cannot know who's on the other side of the interrogation, which means it can either be a machine or a human. To enable this setup, a human will be interacting with two entities through a text interface. These two entities are called respondents. One of them will be a human and the other one will be the machine.

The respondent machine passes the test if the interrogator is unable to tell whether the answers are coming from a machine or a human. The following diagram shows the setup of a Turing test:

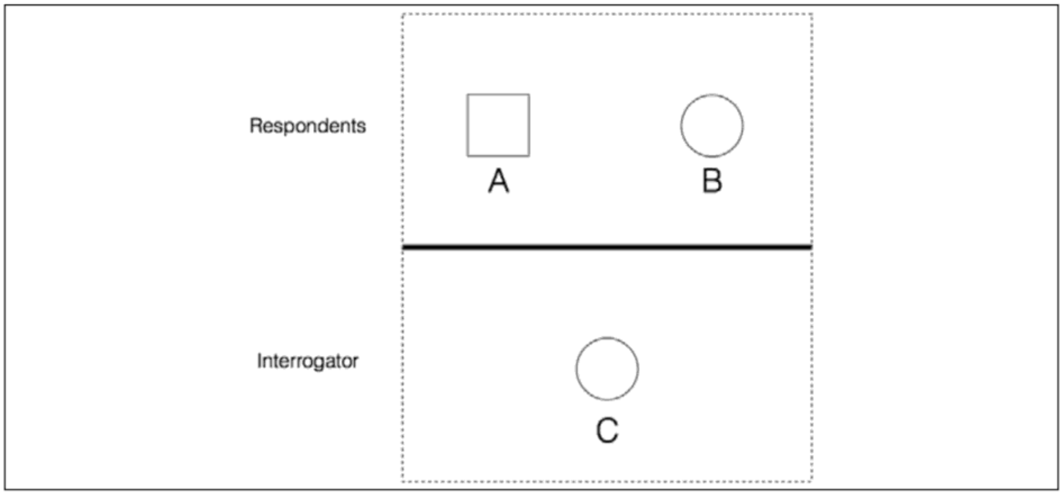


Figure 5: The Turing Test

As you can imagine, this is quite a difficult task for the respondent machine. There are a lot of things going on during a conversation. At the very minimum, the machine needs to be well versed with the following things:

- **Natural language processing**: The machine needs this to communicate with the interrogator. The machine needs to parse the sentence, extract the context, and give an appropriate answer.
- **Knowledge representation**: The machine needs to store the information provided before the interrogation. It also needs to keep track of the information being provided during the conversation so that it can respond appropriately if it comes up again.
- **Reasoning**: It's important for the machine to understand how to interpret the information that gets stored. Humans tend to do this automatically in order to draw conclusions in real time.
- **Machine learning**: This is needed so that the machine can adapt to new conditions in real time. The machine needs to analyze and detect patterns so that it can draw inferences.

You must be wondering why the human is communicating with a text interface. According to Turing, physical simulation of a person is unnecessary for intelligence. That's the reason the Turing test avoids direct physical interaction between the human and the machine.

There is another thing called the Total Turing Test that deals with vision and movement. To pass this test, the machine needs to see objects using computer vision and move around using robotics.

Making machines think like humans

For decades, we have been trying to get the machine to think more like humans. In order to make this happen, we need to understand how humans think in the first place. How do we understand the nature of human thinking? One way to do this would be to note down how we respond to things. But this quickly becomes intractable, because there are too many things to note down. Another way to do this is to conduct an experiment based on a predefined format. We develop a certain number of questions to encompass a wide variety of human topics, and then see how people respond to it.

Once we gather enough data, we can create a model to simulate the human process. This model can be used to create software that can think like humans. Of course, this is easier said than done! All we care about is the output of the program given an input. If the program behaves in a way that matches human behavior, then we can say that humans have a similar thinking mechanism.

The following diagram shows different levels of thinking and how our brain prioritizes things:

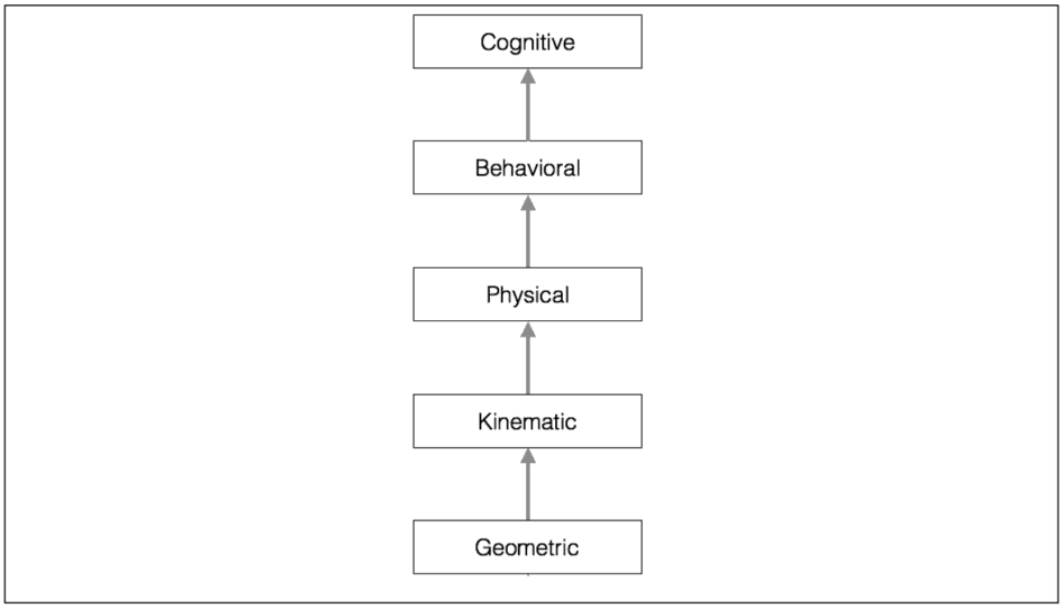


Figure 6: The levels of thought

Within computer science, there is a field of study called **Cognitive Modeling** that deals with simulating the human thinking process. It tries to understand how humans solve problems. It takes the mental processes that go into this problem-solving process and turns it into a software model. This model can then be used to simulate human behavior.

Cognitive modeling is used in a variety of AI applications such as deep learning, expert systems, natural language processing, robotics, and so on.

Building rational agents

A lot of research in AI is focused on building rational agents. What exactly is a rational agent? Before that, let us define the word *rationality* within the context of AI. Rationality refers to observing a set of rules and following their logical implications in order to achieve a desirable outcome. This needs to be performed in such a way that there is maximum benefit to the entity performing the action. An agent, therefore, is said to act rationally if, given a set of rules, it takes actions to achieve its goals. It just perceives and acts according to the information that's available. This system is used a lot in AI to design robots when they are sent to navigate unknown terrains.

How do we define what is *desirable*? The answer is that it depends on the objectives of the agent. The agent is supposed to be intelligent and independent. We want to impart the ability to adapt to new situations. It should understand its environment and then act accordingly to achieve an outcome that is in its best interests. The best interests are dictated by the overall goal it wants to achieve. Let's see how an input gets converted to action:

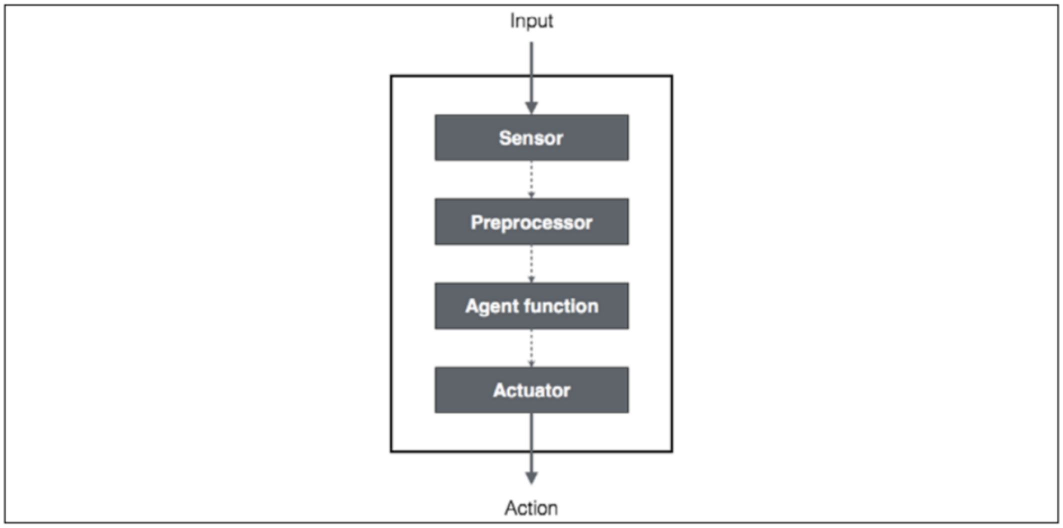


Figure 7: Converting input into action

How do we define the performance measure for a rational agent? One might say that it is directly proportional to the degree of success. The agent is set up to achieve a task, so the performance measure depends on what percentage of that task is complete. But we must think as to what constitutes rationality in its entirety. If it's just about results, we don't consider the actions leading up to the result.

Making the right inferences is a part of being rational, because the agent must act rationally to achieve its goals. This will help it draw conclusions that can be used successively.

But, what about situations where there are no provably right things to do? There are situations where the agent doesn't know what to do, but it still must do something.

Let's set up a scenario to make this last point clearer. Imagine a self-driving car that's going at 60 miles an hour and suddenly someone crosses its path. For the sake of the example, assume that given the speed the car is going, it only has two choices. Either the car crashes against a guard rail knowing that it will kill the car occupant, or it runs over the pedestrian and kills them. What's the right decision? How does the algorithm know what to do? If you were driving, would you know what to do?

We now are going to learn about one of the earliest examples of a rational agent – the General Problem Solver. As we'll see, despite the lofty name, it really wasn't capable of solving any problem, but it was a big leap in the field of computer science nonetheless.

General Problem Solver

The **General Problem Solver** (**GPS**) was an AI program proposed by Herbert Simon, J.C. Shaw, and Allen Newell. It was the first useful computer program that came into existence in the AI world. The goal was to make it work as a universal problem-solving machine. Of course, there were many software programs that existed before, but these programs performed specific tasks. GPS was the first program that was intended to solve any general problem. GPS was supposed to solve all the problems using the same base algorithm for every problem.

As you must have realized, this is quite an uphill battle! To program the GPS, the authors created a new language called **Information Processing Language** (**IPL**). The basic premise is to express any problem with a set of well-formed formulas. These formulas would be a part of a directed graph with multiple sources and sinks. In a graph, the source refers to the starting node and the sink refers to the ending node. In the case of GPS, the source refers to axioms and the sink refers to the conclusions.

Even though GPS was intended to be a general purpose, it could only solve well-defined problems, such as proving mathematical theorems in geometry and logic. It could also solve word puzzles and play chess. The reason was that these problems could be formalized to a reasonable extent. But in the real world, this quickly becomes intractable because of the number of possible paths you can take. If it tries to brute force a problem by counting the number of walks in a graph, it becomes computationally infeasible.

Solving a problem with GPS

Let's see how to structure a given problem to solve it using GPS:

1. The first step is to define the goals. Let's say our goal is to get some milk from the grocery store.
2. The next step is to define the preconditions. These preconditions are in reference to the goals. To get milk from the grocery store, we need to have a mode of transportation and the grocery store should have milk available.
3. After this, we need to define the operators. If my mode of transportation is a car and if the car is low on fuel, then we need to ensure that we can pay the fueling station. We need to ensure that you can pay for the milk at the store.

An operator takes care of the conditions and everything that affects them. It consists of actions, preconditions, and the changes resulting from taking actions. In this case, the action is giving money to the grocery store. Of course, this is contingent upon you having the money in the first place, which is the precondition. By giving them the money, you are changing your money condition, which will result in you getting the milk.

GPS will work if you can frame the problem like we did just now. The constraint is that it uses the search process to perform its job, which is way too computationally complex and time consuming for any meaningful real-world application.

In this section we learned what a rational agent is. Now let's learn how to make these rational agents more intelligent and useful.

Building an intelligent agent

There are many ways to impart intelligence to an agent. The most commonly used techniques include machine learning, stored knowledge, rules, and so on. In this section, we will focus on machine learning. In this method, the way we impart intelligence to an agent is through data and training.

Let's see how an intelligent agent interacts with the environment:

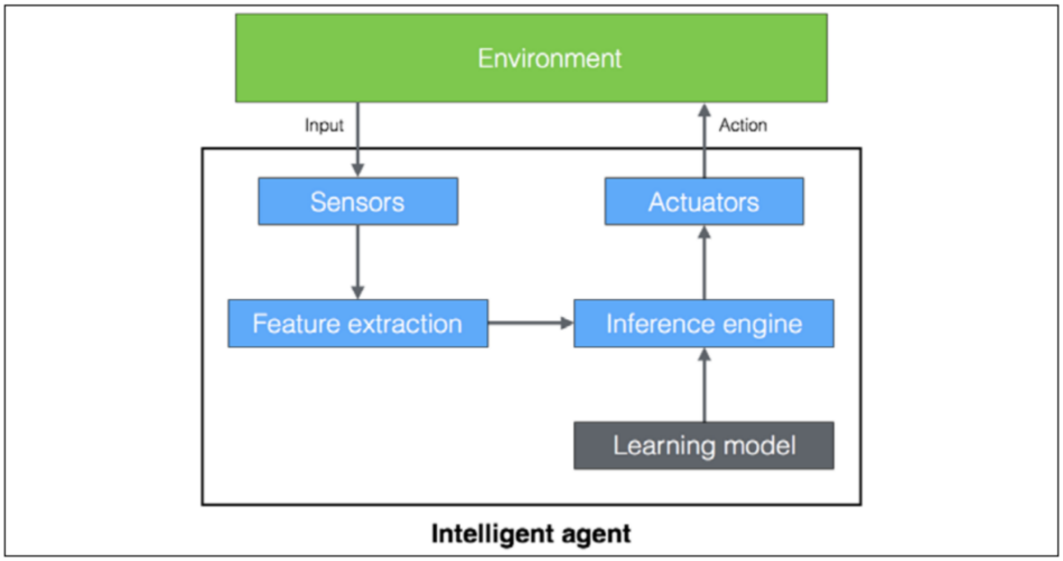


Figure 8: An intelligent agent interaction with its environment

With machine learning, sometimes we want to program our machines to use labeled data to solve a given problem. By going through the data and the associated labels, the machine learns how to extract patterns and relationships.

In the preceding example, the intelligent agent depends on the learning model to run the inference engine. Once the sensor perceives the input, it sends it to the feature extraction block. Once the relevant features are extracted, the trained inference engine performs a prediction based on the learning model. This learning model is built using machine learning. The inference engine then takes a decision and sends it to the actuator, which then takes the required action in the real world.

There are many applications of machine learning that exist today. It is used in image recognition, robotics, speech recognition, predicting stock market behavior, and so on. In order to understand machine learning and build a complete solution, you will have to be familiar with many techniques from different fields such as pattern recognition, artificial neural networks, data mining, statistics, and so on.

Types of models

There are two types of models in the AI world: Analytical models and learned models. Before we had machines that could compute, people used to rely on analytical models.

Analytical models were derived using a mathematical formulation, which is basically a sequence of steps followed to arrive at a final equation. The problem with this approach is that it was based on human judgment. Hence, these models were simplistic and often inaccurate, with just a few parameters. Think of how Newton and other scientists of old made calculations before they had computers. Such models often involved prolonged derivations and long periods of trial and error before a working formula was arrived at.

We then entered the world of computers. These computers were good at analyzing data. So, people increasingly started using learned models. These models are obtained through the process of training. During training, the machines look at many examples of inputs and outputs to arrive at the equation. These learned models are usually complex and accurate, with thousands of parameters. This gives rise to a very complex mathematical equation that governs the data that can assist in making predictions.

Machine learning allows us to obtain these learned models that can be used in an inference engine. One of the best things about this is the fact that we don't need to derive the underlying mathematical formula. You don't need to know complex mathematics, because the machine derives the formula based on data. All we need to do is create the list of inputs and the corresponding outputs. The learned model that we get is just the relationship between labeled inputs and the desired outputs.

Installing Python 3

We will be using Python 3 throughout this book. Make sure you have installed the latest version of Python 3 on your machine. Type the following command to check:

```
$ python3 --version
```

If you see something like Python 3.x.x (where x.x are version numbers) printed out, you are good to go. If not, installing it is straightforward.

Installing on Ubuntu

Python 3 is already installed by default on Ubuntu 14.xx and above. If not, you can install it using the following command:

```
$ sudo apt-get install python3
```

Run the check command like we did earlier:

```
$ python3 --version
```

You should see the version number as output.

Installing on Mac OS X

If you are on Mac OS X, it is recommended that you use Homebrew to install Python 3. It is a great package installer for Mac OS X and it is really easy to use. If you don't have Homebrew, you can install it using the following command:

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
install/master/install)"
```

Let's update the package manager:

```
$ brew update
```

Let's install Python 3:

```
$ brew install python3
```

Run the check command like we did earlier:

```
$ python3 --version
```

You should see the version number printed as output.

Installing on Windows

If you use Windows, it is recommended that you use a `SciPy-stack` compatible distribution of Python 3. Anaconda is pretty popular and easy to use. You can find the installation instructions at: `https://www.continuum.io/downloads`.

If you want to check out other `SciPy-stack` compatible distributions of Python 3, you can find them at `http://www.scipy.org/install.html`. The good part about these distributions is that they come with all the necessary packages preinstalled. If you use one of these versions, you don't need to install the packages separately.

Once you install it, run the check command like we did earlier:

```
$ python3 --version
```

You should see the version number printed as output.

Installing packages

Throughout this book, we will use various packages such as NumPy, SciPy, scikit-learn, and matplotlib. Make sure you install these packages before you proceed.

If you use Ubuntu or Mac OS X, installing these packages is straightforward. All these packages can be installed using a one-line command. Here are the relevant links for installation:

- NumPy: `http://docs.scipy.org/doc/numpy-1.10.1/user/install.html`
- SciPy: `http://www.scipy.org/install.html`
- scikit-learn: `http://scikit-learn.org/stable/install.html`
- matplotlib: `http://matplotlib.org/1.4.2/users/installing.html`

If you are on Windows, you should have installed a `SciPy-stack` compatible version of Python 3.

Loading data

In order to build a learning model, we need data that's representative of the world. Now that we have installed the necessary Python packages, let's see how to use the packages to interact with data. Enter the Python command prompt by typing the following command:

```
$ python3
```

Let's import the package containing all the datasets:

```
>>> from sklearn import datasets
```

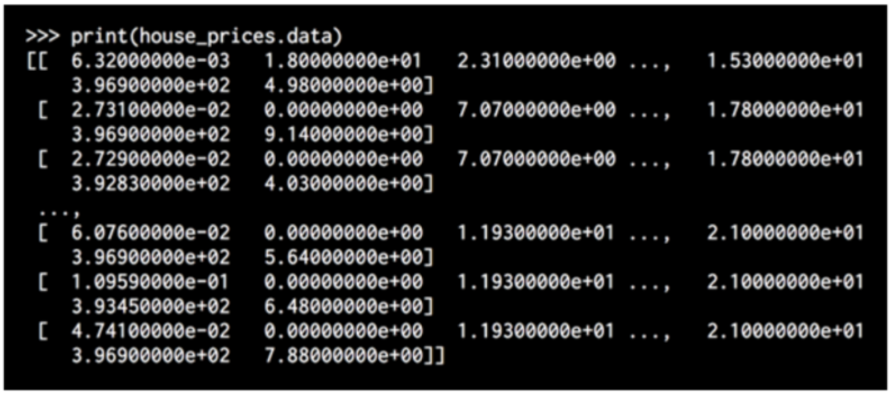
Let's load the house prices dataset:

```
>>> house_prices = datasets.load_boston()
```

Print the data:

```
>>> print(house_prices.data)
```

You will see an output similar to this:

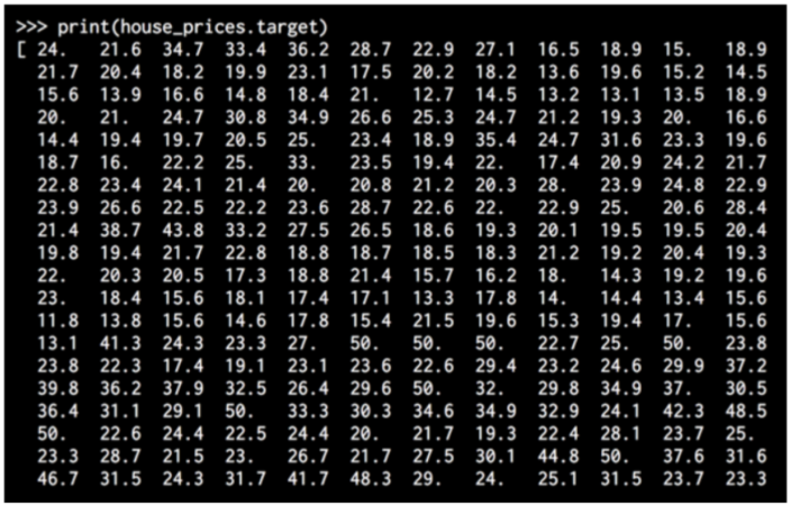


```
>>> print(house_prices.data)
[[  6.32000000e-03   1.80000000e+01   2.31000000e+00 ...,   1.53000000e+01
    3.96900000e+02   4.98000000e+00]
 [  2.73100000e-02   0.00000000e+00   7.07000000e+00 ...,   1.78000000e+01
    3.96900000e+02   9.14000000e+00]
 [  2.72900000e-02   0.00000000e+00   7.07000000e+00 ...,   1.78000000e+01
    3.92830000e+02   4.03000000e+00]
 ...,
 [  6.07600000e-02   0.00000000e+00   1.19300000e+01 ...,   2.10000000e+01
    3.96900000e+02   5.64000000e+00]
 [  1.09590000e-01   0.00000000e+00   1.19300000e+01 ...,   2.10000000e+01
    3.93450000e+02   6.48000000e+00]
 [  4.74100000e-02   0.00000000e+00   1.19300000e+01 ...,   2.10000000e+01
    3.96900000e+02   7.88000000e+00]]
```

Figure 9: Output of input home prices

Let's check out the labels.

You will see this output:



```
>>> print(house_prices.target)
[ 24.   21.6  34.7  33.4  36.2  28.7  22.9  27.1  16.5  18.9  15.   18.9
  21.7  20.4  18.2  19.9  23.1  17.5  20.2  18.2  13.6  19.6  15.2  14.5
  15.6  13.9  16.6  14.8  18.4  21.   12.7  14.5  13.2  13.1  13.5  18.9
  20.   21.   24.7  30.8  34.9  26.6  25.3  24.7  21.2  19.3  20.   16.6
  14.4  19.4  19.7  20.5  25.   23.4  18.9  35.4  24.7  31.6  23.3  19.6
  18.7  16.   22.2  25.   33.   23.5  19.4  22.   17.4  20.9  24.2  21.7
  22.8  23.4  24.1  21.4  20.   20.8  21.2  20.3  28.   23.9  24.8  22.9
  23.9  26.6  22.5  22.2  23.6  28.7  22.6  22.   22.9  25.   20.6  28.4
  21.4  38.7  43.8  33.2  27.5  26.5  18.6  19.3  20.1  19.5  19.5  20.4
  19.8  19.4  21.7  22.8  18.8  18.7  18.5  18.3  21.2  19.2  20.4  19.3
  22.   20.3  20.5  17.3  18.8  21.4  15.7  16.2  18.   14.3  19.2  19.6
  23.   18.4  15.6  18.1  17.4  17.1  13.3  17.8  14.   14.4  13.4  15.6
  11.8  13.8  15.6  14.6  17.8  15.4  21.5  19.6  15.3  19.4  17.   15.6
  13.1  41.3  24.3  23.3  27.   50.   50.   50.   22.7  25.   50.   23.8
  23.8  22.3  17.4  19.1  23.1  23.6  22.6  29.4  23.2  24.6  29.9  37.2
  39.8  36.2  37.9  32.5  26.4  29.6  50.   32.   29.8  34.9  37.   30.5
  36.4  31.1  29.1  50.   33.3  30.3  34.6  34.9  32.9  24.1  42.3  48.5
  50.   22.6  24.4  22.5  24.4  20.   21.7  19.3  22.4  28.1  23.7  25.
  23.3  28.7  21.5  23.   26.7  21.7  27.5  30.1  44.8  50.   37.6  31.6
  46.7  31.5  24.3  31.7  41.7  48.3  29.   24.   25.1  31.5  23.7  23.3
```

Figure 10: Output of predicted home prices

The actual array is larger, so the image represents the first few values in that array.

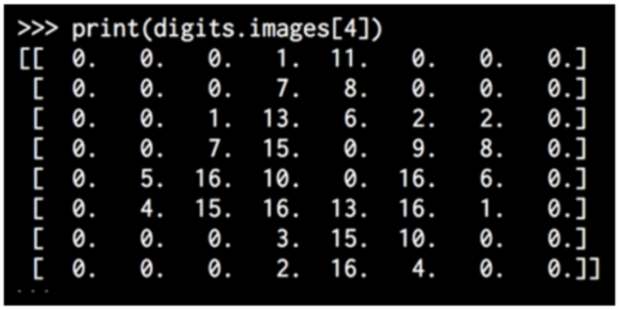
There are also image datasets available in the scikit-learn package. Each image is of shape 8×8. Let's load it:

```
>>> digits = datasets.load_digits()
```

Print the fifth image:

```
>>> print(digits.images[4])
```

You will see this output:



```
>>> print(digits.images[4])
[[ 0.   0.   0.   1.  11.   0.   0.   0.]
 [ 0.   0.   0.   7.   8.   0.   0.   0.]
 [ 0.   0.   1.  13.   6.   2.   2.   0.]
 [ 0.   0.   7.  15.   0.   9.   8.   0.]
 [ 0.   5.  16.  10.   0.  16.   6.   0.]
 [ 0.   4.  15.  16.  13.  16.   1.   0.]
 [ 0.   0.   0.   3.  15.  10.   0.   0.]
 [ 0.   0.   0.   2.  16.   4.   0.   0.]]
...
```

Figure 11: Output of scikit-learn array of images

As you can see, it has eight rows and eight columns.

Summary

In this chapter, we discussed:

- What AI is all about and why we need to study it
- Various applications and branches of AI
- What the Turing test is and how it's conducted
- How to make machines think like humans
- The concept of rational agents and how they should be designed
- General Problem Solver (GPS) and how to solve a problem using GPS
- How to develop an intelligent agent using machine learning
- Different types of machine learning models

We also went through how to install Python 3 on various operating systems, and how to install the necessary packages required to build AI applications. We discussed how to use these packages to load data that's available in scikit-learn.

In the next chapter, we will learn about supervised learning and how to build models for classification and regression.

2
Fundamental Use Cases for Artificial Intelligence

In this chapter, we are going to discuss some of the use cases for **Artificial Intelligence** (**AI**). This by no means is an exhaustive list. Many industries have been impacted by AI, and the list of those industries not yet impacted gets shorter every day. Ironically, some of the jobs that robots, automation, and AI will not be able to take over are jobs with a low pay rate that require less "brain" power. For example, it will be a while until we are able to replace hair stylists and plumbers. Both of these jobs require a lot of finesse and detail that robots have yet to master. I know it will be a long time before my wife trusts her hair to anyone else other than her current hair stylist, let alone a robot.

This chapter will discuss:

- Some representative AI use cases
- The jobs that will take the longest to replace by automation
- The industries that will be most impacted by AI

Representative AI use cases

From finance to medicine, it is difficult to find an industry that is not being disrupted by Artificial Intelligence. We will focus on real-world examples of the most popular applications of AI in our everyday life. We will explore the current state of the art as well as what is coming soon. Most importantly, maybe this book will spark your imagination and you will come up with some new and innovative ideas that will positively impact society and we can add it to the next edition of our book.

Artificial Intelligence, cognitive computing, machine learning, and deep learning are only some of the disruptive technologies that are enabling rapid change today. These technologies can be adopted quicker because of advances in cloud computing, **Internet of Things** (**IoT**), and edge computing. Organizations are reinventing the way they do business by cobbling together all these technologies. This is only the beginning; we are not even in the first inning, we haven't even recorded the first strike!

With that, let's begin to look at some contemporary applications of AI.

Digital personal assistants and chatbots

Unfortunately, it is still all too common for some call centers to use legacy **Interactive Voice Response** (**IVR**) **systems** that make calling them an exercise in patience. However, we have made great advances in the area of natural language processing: chatbots. Some of the most popular examples are:

- **Google Assistant**: Google Assistant was launched in 2016 and is one of the most advanced chatbots available. It can be found in a variety of appliances such as telephones, headphones, speakers, washers, TVs, and refrigerators. Nowadays, most Android phones include Google Assistant. Google Home and Nest Home Hub also support Google Assistant.
- **Amazon Alexa**: Alexa is a virtual assistant developed and marketed by Amazon. It can interact with users by voice and by executing commands such as playing music, creating to-do lists, setting up alarms, playing audiobooks, and answering basic questions. It can even tell you a joke or a story on demand. Alexa can also be used to control compatible smart devices. Developers can extend Alexa's capabilities by installing skills. An Alexa skill is additional functionality developed by third-party vendors.
- **Apple Siri**: Siri can accept user voice commands and a natural language user interface to answer questions, make suggestions, and perform actions by parsing these voice commands and delegating these requests to a set of internet services. The software can adapt to users' individual language usage, their searches, and preferences. The more it is used the more it learns and the better it gets.
- **Microsoft Cortana**: Cortana is another digital virtual assistant, designed and created by Microsoft. Cortana can set reminders and alarms, recognize natural voice commands, and it answers questions using information.

All these assistants will allow you to perform all or at least most of these tasks:

- Control devices in your home
- Play music and display videos on command
- Set timers and reminders
- Make appointments
- Send text and email messages
- Make phone calls
- Open applications
- Read notifications
- Perform translations
- Order from e-commerce sites

Some of the tasks that might not be supported but will start to become more pervasive are:

- Checking into your flight
- Booking a hotel
- Making a restaurant reservation

All these platforms also support 3rd party developers to develop their own applications or "skills" as Amazon calls them. So, the possibilities are endless.

Some examples of existing Alexa skills:

- **MySomm**: Recommends what wine goes with a certain meat
- **The bartender**: Provides instructions on how to make alcoholic drinks
- **7-minute workout**: Will guide you through a tough 7-minute workout
- **Uber**: Allows you to order an Uber ride through Alexa

All the preceding services listed continue to get better. They continuously learn from interactions with customers. They are improved both by the developers of the services as well as by the systems taking advantage of new data points created daily by users of the services.

Most cloud providers make it extremely easy to create chatbots and for some basic examples it is not necessary to use a programming language. In addition, it is not difficult to deploy these chatbots to services such as Slack, Facebook Messenger, Skype, and WhatsApp.

Personal chauffeur

Self-driving or driverless cars are vehicles that can travel along a pre-established route with no human assistance. Most self-driving cars in existence today do not rely on a single sensor and navigation method and use a variety of technologies such as radar, sonar, lidar, computer vision, and GPS.

As technologies emerge, industries start creating standards to implement and measure their progress. Driverless technologies are no different. SAE International has created standard J3016, which defines six levels of automation for cars so that automakers, suppliers, and policymakers can use the same language to classify the vehicle's level of sophistication:

Level 0 (No automation)

The car has no self-driving capabilities. The driver is fully involved and responsible. The human driver steers, brakes, accelerates, and negotiates traffic. This describes most current cars on the road today.

Level 1 (Driver assistance)

System capability: Under certain conditions, the car controls either the steering or the vehicle speed, but not both simultaneously.

Driver involvement: The driver performs all other aspects of driving and has full responsibility for monitoring the road and taking over if the assistance system fails to act appropriately. For example, Adaptive cruise control.

Level 2 (Partial automation)

The car can steer, accelerate, and brake in certain circumstances. The human driver still performs many maneuvers like interpreting and responding to traffic signals or changing lanes. The responsibility for controlling the vehicle largely falls on the driver. The manufacturer still requires the driver to be fully engaged. Examples of this level are:

- Audi Traffic Jam Assist
- Cadillac Super Cruise
- Mercedes-Benz Driver Assistance Systems
- Tesla Autopilot
- Volvo Pilot Assist

Level 3 (Conditional automation)

The pivot point between levels 2 and 3 is critical. The responsibility for controlling and monitoring the car starts to change from driver to computer at this level. Under the right conditions, the computer can control the car, including monitoring the environment. If the car encounters a scenario that it cannot handle, it requests that the driver intervene and take control. The driver normally does not control the car but must be available to take over at any time. An example of this is Audi Traffic Jam Pilot.

Level 4 (High automation)

The car does not need human involvement under most conditions but still needs human assistance under some road, weather, or geographic conditions. Under a shared car model restricted to a defined area, there may not be any human involvement. But for a privately-owned car, the driver might manage all driving duties on surface streets and the system takes over on the highway. Google's now defunct Firefly pod-car is an example of this level. It didn't have pedals or a steering wheel. It was restricted to a top speed of 25 mph and it was not used in public streets.

Level 5 (Full automation)

The driverless system can control and operate the car on any road and under any conditions that a human driver could handle. The "operator" of the car only needs to enter a destination. Nothing at this level is in production yet but a few companies are close and might be there by the time the book is published.

We'll now review some of the leading companies working in the space:

Google's Waymo

As of 2018, Waymo's autonomous cars have driven eight million miles on public roads as well as five billion miles in simulated environments. In the next few years, it is all but a certainty that we will be able to purchase a car capable of full driving autonomy. Tesla, among others, already offers driver assistance with their Autopilot feature and possibly will be the first company to offer full self-driving capabilities. Imagine a world where a child born today will never have to get a driver's license! The disruption caused in our society by this advance in AI alone will be massive. The need for delivery drivers, taxi drivers, and truckers will be obviated. Even if there are still car accidents in a driverless future, millions of lives will be saved because we will eliminate distracted driving and drunk driving.

Waymo launched the first commercial driverless service in 2018 in Arizona, USA with plans to expand nationally and worldwide.

Uber ATG

Uber's **Advanced Technology Group** (**ATG**) is an Uber subsidiary working on developing self-driving technology. In 2016, Uber launched an experimental car service on the streets of Pittsburgh. Uber has plans to buy up to 24,000 Volvo XC90 and equip them with their self-driving technology and start commercializing them in some capacity by 2021.

Tragically, in March 2018, Elaine Herzberg was involved in an incident with an Uber driverless car and died. According to police reports, she was struck by the Uber vehicle while trying to cross the street, while she was watching a video on her phone. Ms. Herzberg became one of the first individuals to die in an incident involving a driverless car. Ideally, we would like to see no accidents ever happen with this technology, yet the level of safety that we demand needs to be tempered with the current crisis we have with traffic accidents. For context, there were 40,100 motor vehicle deaths in the US in 2017; even if we continue to see accidents with automated cars, if this death toll was slashed by say, half, thousands of lives would be saved each year.

It is certainly possible to envision a driverless vehicle that looks more like a living room than the interior of our current cars. There would be no need for steering wheels, pedals or any kind of manual control. The only input the car would need is your destination, which could be given at the beginning of your journey by "speaking" to your car. There would be no need to keep track of a maintenance schedule as the car would be able to sense when a service is due or there is an issue with the car's function.

Liability for car accidents will shift from the driver of the vehicle to the manufacturer of the vehicle doing away with the need to have car insurance. This last point is probably one of the reasons why car manufacturers have been slow to deploy this technology. Even car ownership might be flipped on its head since we could summon a car whenever we need one instead of needing one all the time.

Shipping and warehouse management

An Amazon sorting facility is one of the best examples of the symbiotic relationship that is forming between humans, computers, and robots. Computers take customer orders and decide where to route merchandise, the robots act as mules carrying the pallets and inventory around the warehouse. Humans plug the "last mile" problem by hand picking the items that are going into each order. Robots are proficient in mindlessly repeating a task many times as long as there is a pattern involved and some level of pretraining is involved to achieve this. However, having a robot pick a 20-pound package and immediately being able to grab an egg without breaking it is one of the harder robotics problems.

Robots struggle dealing with objects of different sizes, weights, shapes, and fragility; a task that many humans can perform effortlessly. People, therefore, handle the tasks that the robots encounter difficulty with. The interaction of these three types of different actors translates into a finely tuned orchestra that can deliver millions of packages everyday with very little mistakes.

Even Scott Anderson, Amazon's director of robotics fulfillment acknowledged in May 2019 that a fully automated warehouse is at least 10 years away. So, we will continue to see this configuration in warehouses across the world for a little longer.

Human health

The ways that AI can be applied in health science is almost limitless. We will discuss a few of them here, but it will by no means be an exhaustive list.

Drug discovery

AI can assist in generating drug candidates (that is, molecules to be tested for medical application) and then quickly eliminating some of them using constraint satisfaction or experiment simulation. We will learn more about constraint satisfaction programming in later chapters. In a nutshell, this approach allows us to speed up drug discovery by quickly generating millions of possible drug candidates and just as quickly rejecting them if the candidates do not satisfy certain predetermined constraints.

In addition, in some cases we can simulate experiments in the computer that otherwise would be much more expensive to perform in real life.

Furthermore, in some instances researchers still conduct real-world experiments but rely on robots to perform the experiments and speed up the process with them. These emerging fields are dubbed **high throughput screening** (**HTS**) and **virtual high throughput screening** (**VHTS**).

Machine learning is starting to be used more and more to enhance clinical trials. The consulting company of Accenture has developed a tool called **intelligent clinical trials** (**ITP**). It is used to predict the length of clinical trials.

Another approach that can surprisingly be used is to apply to drug discovery is **Natural Language Processing** (**NLP**). Genomic data can be represented using a string of letters and the NLP techniques can be used to process or "understand" what the genomic sequences mean.

Insurance pricing

Machine learning algorithms can be used to better price insurance by more accurately predicting how much will be spent on a patient, how good a driver an individual is, or how long a person will live.

As an example, the *young.ai* project from Insilico Medicine can predict with some accuracy how long someone will live from a blood sample and a photograph. The blood sample provides 21 biomarkers such as cholesterol level, inflammation markers, hemoglobin counts and albumin level that are used as input to a machine learning model. Other inputs into the model are ethnicity and age, as well as a photograph of the person.

Interestingly, as of now, anyone can use this service for free by visiting young.ai (`https://young.ai`) and providing the required information.

Patient diagnosis

Doctors can make better diagnosis on their patients and be more productive in their practice by using sophisticated rules engines and machine learning. As an example, in a recent study at the University of California in San Diego conducted by Kang Zhang [1], one system could diagnose children's illnesses with a higher degree of accuracy than junior pediatricians. The system was able to diagnose the following diseases with a degree of accuracy of between 90% and 97%:

- Glandular fever
- Roseola
- Influenza
- Chicken pox
- Hand, foot, and mouth disease

The input dataset consisted of medical records from 1.3 million children visits to the doctor from the Guangzhou region in China between 2016 and 2017.

Medical imaging interpretation

Medical imaging data is a complex and rich source of information about patients. CAT scans, MRIs, and X-rays contain information that is otherwise unavailable. There is a shortage of radiologists and clinicians that can interpret them. Getting results from these images can sometimes take days and can sometimes be misinterpreted. Recent studies have found that machine learning models can perform just as well, if not better, than their human counterparts.

Data scientists have developed AI enabled platforms that can interpret MRI scans and radiological images in a matter of minutes instead of days and with a higher degree of accuracy when compared with traditional methods.

Perhaps surprisingly, far from being concerned, leaders from the American College for Radiology see the advent of AI as a valuable tool for physicians. In order to foster further development in the field, the **American College for Radiology Data Science Institute** (**ACR DSI**) released several AI use cases in medical imaging and plans to continue releasing more.

Psychiatric analysis

An hour-long session with a psychiatrist can costs hundreds of dollars. We are on the cusp of being able to simulate the behavior with AI chatbots. At the very least, these bots will be able to offer follow-up care from the sessions with the psychiatrist and help with a patient's care between doctor's visits.

One early example of an automated counselor is Eliza. It was developed in 1966 by Joseph Weizenbaum. It allows users to have a "conversation" with the computer mimicking a Rogerian psychotherapist. Remarkably, Eliza feels natural, but its code is only a few hundred lines and it doesn't really use much AI at its core.

A more recent and advanced example is Ellie. Ellie was created by the Institute for Creative Technologies at the University of Southern California. It helps with the treatment of people with depression or post-traumatic stress disorder. Ellie is a virtual therapist (she appears on screen), responds to emotional cues, nods affirmatively when appropriate and shifts in her seat. She can sense 66 points on a person's face and use these inputs to read a person's emotional state. One of Ellie's secrets is that she is obviously not human and that makes people feel less judged and more comfortable opening up to her.

Smart health records

Medicine is notorious for being a laggard in moving to electronic records. Data science provides a variety of methods to streamline the capture of patient data including OCR, handwriting recognition, voice to text capture, and real-time reading and analysis of patient's vital signs. It is not hard to imagine a future coming soon where this information can be analyzed in real-time by AI engines to take decisions such as adjusting body glucose levels, administering a medicine, or summoning medical help because a health problem is imminent.

Disease detection and prediction

The human genome is the ultimate dataset. At some point soon, we will be able to use the human genome as input to machine learning models and be able to detect and predict a wide variety of diseases and conditions using this vast dataset.

Using genomic datasets as an input in machine learning is an exciting area that is evolving rapidly and will revolutionize medicine and health care.

The human genome contains over 3 billion base pairs. We are making progress on two fronts that will accelerate progress:

- Continuous advancements in the understanding of genome biology
- Advances in big data computing to process vast amounts of data faster

There is much research applying deep learning to the field of genomics. Although it is still in early stages, deep learning in genomics has the potential to inform fields including:

- Functional genomics
- Oncology
- Population genetics
- Clinical genetics
- Crop yield improvement
- Epidemiology and public health
- Evolutionary and phylogenetic analysis

Knowledge search

We have gotten to a point where, in some cases, we don't even realize we are using artificial intelligence. A sign that a technology or product is good is when we don't necessarily stop to think how it's doing what it is doing. A perfect example of this is Google Search. The product has become ubiquitous in our lives and we don't realize how much it relies on artificial intelligence to produce its amazing results. From its Google Suggest technology to its constant improvement of the relevancy of its results, AI is deeply embedded in its search process.

Early in 2015, as was reported by Bloomberg, Google began using a deep learning system called RankBrain to assist in generating search query responses. The Bloomberg article describes RankBrain as follows:

> *"RankBrain uses artificial intelligence to embed vast amounts of written language into mathematical entities – called vectors – that the computer can understand. If RankBrain sees a word or phrase it isn't familiar with, the machine can make a guess as to what words or phrases might have a similar meaning and filter the result accordingly, making it more effective at handling never-before-seen search queries."*
>
> *– Clark, Jack [2]*

As of the last report, RankBrain plays a role in a large percentage of the billions of Google Search queries. As one can imagine, the company is tight lipped about how exactly RankBrain works, and furthermore even Google might have a hard time explaining how it works. You see, this is one of the dilemmas of deep learning. In many cases, it can provide highly accurate results, but deep learning algorithms are usually hard to understand in terms of why an individual answer was given. Rule-based systems and even other machine learning models (such as Random Forest) are much easier to interpret.

The lack of explainability of deep learning algorithms has major implications, including legal implications. Lately, Google and Facebook among others, have found themselves under the microscope to determine if their results are biased. In the future, legislators and regulators might require that these tech giants provide a justification for a certain result. If deep learning algorithms do not provide explainability, they might be forced to use other less accurate algorithms that do.

Initially, RankBrain only assisted in about 15 percent of Google queries, but now it is involved in almost all user queries.

However, if a query is a common query, or something that the algorithm understands, the RankBrain rank score is given little weight. If the query is one that the algorithm has not seen before or it does not know its meaning, RankBrain score is much more relevant.

Recommendation systems

Recommendation systems are another example of AI technology that has been weaved into our everyday lives. Amazon, YouTube, Netflix, LinkedIn, and Facebook all rely on recommendation technology and we don't even realize we are using it. Recommendation systems rely heavily on data and the more data that is at their disposable, the more powerful they become. It is not coincidence that these companies have some of the biggest market caps in the world and their power comes from them being able to harness the hidden power in their customer's data. Expect this trend to continue in the future.

What is a recommendation? Let's answer the question by first exploring what it is not. It is not a definitive answer. Certain questions like "what is two plus two?" or "how many moons does Saturn have?" have a definite answer and there is no room for subjectivity. Other questions like "what is your favorite movie?" or "do you like radishes?" are completely subjective and the answer is going to depend on the person answering the question. Some machine learning algorithms thrive with this kind of "fuzziness." Again, these recommendations can have tremendous implications.

Think of the consequences of Amazon constantly recommending a product versus another. The company that makes the recommended product will thrive and the company that makes the product that was not recommended could go out of business if it doesn't find alternative ways to distribute and sell its product.

One of the ways that a recommender system can improve is by having previous selections from users of the system. If you visit an e-commerce site for the first time and you don't have an order history, the site will have a hard time making a recommendation tailored to you. If you purchase sneakers, the website now has one data point that it can start using as a starting point. Depending on the sophistication of the system, it might recommend a different pair of sneakers, a pair of athletic socks, or maybe even a basketball (if the shoes were high-tops).

An important component of good recommendation systems is a randomization factor that occasionally "goes out on a limb" and makes oddball recommendations that might not be that related to the initial user's choices. Recommender systems don't just learn from history to find similar recommendations, but they also attempt to make new recommendations that might not be related at first blush. For example, a Netflix user might watch "The Godfather" and Netflix might start recommending Al Pacino movies or mobster movies. But it might recommend "Bourne Identity," which is a stretch. If the user does not take the recommendation or does not watch the movie, the algorithm will learn from this and avoid other movies like the "Bourne Identity" (for example any movies that have Jason Bourne as the main character).

As recommender systems get better, the possibilities are exciting. They will be able to power personal digital assistants and become your personal butler that has intimate knowledge of your likes and dislikes and can make great suggestions that you might have not thought about. Some of the areas where recommendations can benefit from these systems are:

- Restaurants
- Movies
- Music
- Potential partners (online dating)
- Books and articles
- Search results
- Financial services (robo-advisors)

Some notable specific examples of recommender systems follow:

Netflix Prize

A contest that created a lot of buzz in the recommender system community was the Netflix Prize. From 2006 to 2009, Netflix sponsored a competition with a grand prize of one million US dollars. Netflix made available a dataset of 100 million plus ratings.

Netflix offered to pay the prize to the team that offered the highest accuracy in their recommendations and was 10% more accurate than the recommendations from Netflix's existing recommender system. The competition energized research for new and more accurate algorithms. In September 2009, the grand prize was awarded to the BellKor's Pragmatic Chaos team.

Pandora

Pandora is one of the leading music services. Unlike other companies like Apple and Amazon, Pandora's exclusive focus is as a music service. One of Pandora's salient service features is the concept of customized radio stations. These "stations" allow users to play music by genre. As you can imagine, recommender systems are at the core of this functionality.

Pandora's recommender is built on multiple tiers:

- First, their team of music experts annotates songs based on genre, rhythm, and progression.
- These annotations are transformed into a vector for comparing song similarity. This approach promotes the presentation of "long tail" or obscure music from unknown artists that nonetheless could be a good fit for individual listeners.
- The service also heavily relies on user feedback and uses it to continuously enhance the service. Pandora has collected over 75 billion feedback data points on listener preferences.
- The Pandora recommendation engine can then perform personalized filtering based on a listener's preferences using their previous selections, geography, and other demographic data.

In total, Pandora's recommender uses around 70 different algorithms, including 10 to analyze content, 40 to process collective intelligence, and about another 30 to do personalized filtering.

Betterment

Robo-advisors are recommendation engines that provide investment or financial advice and management with minimal human involvement. These services use machine learning to automatically allocate, manage, and optimize a customer's asset mix. They can offer these services at a lower cost than traditional advisors because their overhead is lower, and their approach is more scalable.

There is now fierce competition in this space with well over 100 companies offering these kinds of services. Robo-advisors are considered a tremendous breakthrough. Formerly, wealth management services were an exclusive and expensive service reserved for high net worth individuals. Robo-advisors promise to bring a similar service to a broader audience with lower costs compared to the traditional human-enabled services. Robo-advisors could potentially allocate investments in a wide variety of investment products like stocks, bonds, futures, commodities, real estate, and other exotic investments. However, to keep things simple investments are often constrained to **exchange traded funds** (**ETFs**).

As we mentioned there are many companies offering robo-advice. As an example, you might want to investigate Betterment to learn more about this topic. After filling out a risk questionnaire, Betterment will provide users with a customized, diversified portfolio. Betterment will normally recommend a mix of low-fee stock and bond index funds. Betterment charges an administration fee (as a percentage of the portfolio) but it is lower than most human-powered services. Please note that we are not endorsing this service and we only mention it as an example of a recommendation engine in the financial sector.

The smart home

Whenever you bring up the topic of AI to the common folk on the street, they are usually skeptical about how soon it is going to replace human workers. They can rightly point to the fact that we still need to do a lot of housework around the house. AI needs to become not only technologically possible, but it also needs to be economically feasible for adoption to become widespread. House help is normally a low-wage profession and, for that reason, automation to replace it needs to be the same price or cheaper. In addition, house work requires a lot of finesse and it comprises tasks that are not necessarily repetitive. Let's list out some of the tasks that this automaton will need to perform in order to be proficient:

- Wash and dry clothes
- Fold clothes
- Cook dinner

- Make beds
- Pick up items off the floor
- Mop, dust and vacuum
- Wash dishes
- Monitor the home

As we already know, some of these tasks are easy to perform for machines (even without AI) and some of them are extremely hard. For this reason and because of the economic considerations, the home will probably be one of the last places to become fully automated. Nonetheless, let's look at some of the amazing advances that have been made in this area.

Home Monitoring

Home monitoring is one area where great solutions are generally available already. The Ring video doorbell from Amazon and the Google Nest thermostat are two inexpensive options that are widely available and popular. These are two simple examples of smart home devices that are available for purchase today.

The Ring video doorbell is a smart home device connected to the internet that can notify the homeowner of activity at their home, such as a visitor, via their smartphone. The system does not continuously record but rather it activates when the doorbell is pressed, or when the motion detector is activated. The Ring doorbell can then let the home owner watch the activity or communicate with the visitor using the built-in microphone and speakers. Some models also allow the homeowner to open the door remotely via a smart lock and let the visitor into the house.

The Nest Learning Thermostat is a smart home device initially developed by Nest Labs, a company that was later bought by Google. It was designed by Tony Fadell, Ben Filson, and Fred Bould. It is programmable, Wi-Fi-enabled, and self-learning. It uses artificial intelligence to optimize the temperature of the home while saving energy.

In the first weeks of use you set the thermostat to your preferred settings and this will serve as a baseline. The thermostat will learn your schedule and your preferred temperatures. Using built-in sensors and your phones' locations, the thermostat will shift into energy saving mode when no one is home.

Since 2011, the Nest Thermostat has saved billions of kWh of energy in millions of homes worldwide. Independent studies have shown that it saves people an average of 10% to 12% on their heating bills and 15% on their cooling bills so in about 2 years it may pay for itself.

Vacuuming and mopping

Two tasks that have been popular to hand off to robots are vacuuming and mopping. A robotic vacuum cleaner is an autonomous robotic vacuum cleaner that uses AI to vacuum a surface. Depending on the design, some of these machines use spinning brushes to reach tight corners and some models include several other features in addition to being able to vacuum, such as mopping and UV sterilization. Much of the credit for popularizing this technology goes to the company (not the film), *iRobot*.

iRobot was started in 1990 by Rodney Brooks, Colin Angle, and Helen Greiner after meeting each other while working in MIT's Artificial Intelligence Lab. iRobot is best known for its vacuuming robot (Roomba), but for a long time they also had a division devoted to the development of military robots. The Roomba started selling in 2002. As of 2012 iRobot had sold more than eight million home robots as well as creating more than 5,000 defense and security robots. The company's PackBot is a bomb-disposal robot used by the US military that has been used extensively in Iraq and Afghanistan. PackBots were also used to gather information under dangerous conditions at the Fukushima Daiichi nuclear disaster site. iRobot's Seaglider was used to detect underwater pools of oil after the Deepwater Horizon oil spill in the Gulf of Mexico.

Another iRobot product is the Braava series of cleaners. The Braava is a small robot that can mop and sweep floors. It is meant for small spaces like bathrooms and kitchens. It sprays water and uses an assortment of different pads to clean effectively and quietly. Some of the Braava models have a built-in navigation system. The Braava doesn't have enough power to remove deep-set stains, so it's not a complete human replacement, but it does have wide acceptance and high ratings. We expect them to continue to gain popularity.

The potential market for intelligent devices in the home is huge and it is all but certain that we will continue to see attempts from well established companies and startups alike to exploit this largely untapped market.

Picking up your mess

As we learned in the shipping use case, picking objects of different weights, dimensions, and shapes is one of the most difficult tasks to automate. Robots can perform efficiently under homogeneous conditions like a factory floor where certain robots specialize in certain tasks. Picking up a pair of shoes after picking up a chair, however, can be immensely challenging and expensive. For this reason, do not expect this home chore to be pervasively performed by machines in a cost-effective fashion any time soon.

Personal chef

Like picking up items off the floor, cooking involves picking up disparate items. Yet there are two reasons why we can expect "automated cooking" to happen sooner:

- Certain restaurants may charge hundreds of dollars for their food and be paying high prices for skilled chefs. Therefore, they might be open to using technology to replace their high-priced staff if this should work out to be more profitable. An example for this is a five-star sushi restaurant.
- Some tasks in the kitchen are repetitive and therefore lend themselves to automation. Think of a fast food joint where hamburgers and fries might have to be made by the hundreds. Thus, rather than having one machine handle the entire disparate cooking process, a series of machines could deal with individual repetitive stages of the process.

Smart prosthetics are great examples of artificial intelligence augmenting humans rather than replacing them. There are more than a few chefs that lost their arm in an accident or were born without a limb.

One example is chef Michael Caines who runs a two Michelin star restaurant and lost his arm in a horrific car accident. Chef Caines was head chef of Gidleigh Park in Devon in England until January 2016.[3] He is currently the executive chef of the Lympstone Manor hotel between Exeter and Exmouth. He now cooks with a prosthetic arm, but you'd never know it given the quality of his food.

Another example is Eduardo Garcia who is a sportsman and a chef – both of which are made possible by the most advanced bionic hand in the world.

On October 2011, while bow-hunting elk he was electrocuted in the Montana backcountry. Eduardo was hunting by himself in October 2011. He was in back country when he saw a dead baby black bear. He stopped to check it out, knelt, and used his knife to prod it.

While doing so, 2,400 volts coursed through his body – the baby bear had been killed by a buried, live electrical wire. He survived but lost his arm during the incident.

In September 2013, Garcia was fitted by Advanced Arm Dynamics with a bionic hand designed by Touch Bionics. The bionic hand is controlled by Garcia's forearm muscles and can grip in 25 different ways. With his new hand, Garcia can perform tasks that normally require great dexterity. His new hand still has some limitations. For example, Garcia cannot lift heavy weights. However, there are things that he can perform now that he couldn't before. For example, he can grab things out of a hot oven and not get burnt and it is impossible to cut his fingers.

Conversely, rather than augmenting humans, robots may replace humans in the kitchen entirely. An example of this is Moley, the robotic kitchen. Moley is not currently in production but the most advanced prototype of the Moley Robotic Kitchen consists of two robotic arms with hands equipped with tactile sensors, a stove top, an oven, a dishwasher, and a touchscreen unit. These artificial hands can lift, grab, and interact with most kitchen equipment including knives, whisks, spoons, and blenders.

Using a 3D camera and a glove it can record a human chef preparing a meal and then upload detailed steps and instructions into a repository. The chef's actions are then translated into robotic movements using gesture recognition models. These models were created in collaboration with Stanford University and Carnegie Mellon University. After that Moley can reproduce the same steps and cooks the exact same meal from scratch.

In the current prototype, the user can operate it using a touchscreen or smartphone application with ingredients prepared in advance and placed in preset locations. The company's long-term goal is to allow users to simply select an option from a list of more 2,000 recipes and Moley will have the meal prepared in minutes.

Gaming

There is perhaps no better example to demonstrate the awe-inspiring advances in Artificial Intelligence than the progress that has been made in the area of gaming. Humans are competitive by nature and having machines beat us at our own games is an interesting yardstick to measure the breakthroughs in the field. Computers have long been able to beat us in some of the more basic, more deterministic, less compute-intensive games like say checkers. It's only in the last few years that machines have been able to consistently beat the masters of some of the harder games. In this section we go over three of these examples.

StarCraft 2

Video games have been used for decades as a benchmark to test the performance of AI systems. As capabilities increase, researchers work with more complex games that require different types of intelligence. The strategies and techniques developed from this game playing can transfer to solving real-world problems. The game of StarCraft II is considered one of the hardest, though it is an ancient game by video game standards.

The team at DeepMind introduced a program dubbed AlphaStar that can play StarCraft II and was for the first time able to defeat a top professional player. In matches held in December 2018, AlphaStar whooped a team put together by Grzegorz "MaNa" Komincz, one of the world's strongest professional StarCraft players with a score of 5-0. The games took place under professional match conditions and without any game restrictions.

In contrast to previous attempts to master the game using AI that required restrictions, AlphaStar can play the full game with no restrictions. It uses a deep neural network that is trained directly from raw game data using supervised learning and reinforcement learning.

One of the things that makes StarCraft II so difficult is the need to balance short- and long-term goals and adapt to unexpected scenarios. This has normally posed a tremendous challenge for previous systems.

While StarCraft is just a game, albeit a difficult one, the concepts and techniques coming out of AlphaStar can be useful in solving other real-world challenges. As an example, AlphaStar's architecture is capable of modeling very long sequences of likely actions – with games often lasting up to an hour with tens of thousands of moves – based on imperfect information. The primary concept of making complicated predictions over long sequences of data can be found in many real-world problems, such as:

- Weather prediction
- Climate modelling
- Natural Language Understanding

The success that AlphaStar has demonstrated playing StarCraft represents a major scientific breakthrough in one of the hardest video games in existence. These breakthroughs represent a big leap in the creation of artificial intelligence systems that can be transferred and that can help solve fundamental real-world practical problems.

Jeopardy

IBM and the Watson team made history in 2011 when they devised a system that was able to beat two of the most successful Jeopardy champions.

Ken Jennings has the longest unbeaten run in the show's history with 74 consecutive appearances. Brad Rutter had the distinction of winning the biggest prize pot with a total of $3.25 million.

Both players agreed to an exhibition match against Watson.

Watson is a question-answering system that can answer questions posed in natural language. It was initially created by IBM's DeepQA research team, led by principal investigator David Ferrucci.

The main difference between the question-answering technology used by Watson and general search (think Google searches) is that general search takes a keyword as input and responds with a list of documents with a ranking based on the relevance to the query. Question-answering technology like what is used by Watson takes a question expressed in natural language, tries to understand the question at a deeper level, and tries to provide the precise answer to the question.

The software architecture of Watson uses:

- IBM's DeepQA software
- Apache UIMA (Unstructured Information Management Architecture)
- A variety of languages, including Java, C++, and Prolog
- SUSE Linux Enterprise Server
- Apache Hadoop for distributed computing

Chess

Many of us remember the news when Deep Blue famously beat chess grand master Gary Kasparov in 1996. Deep Blue was a chess-playing application created by IBM.

In the first round of play Deep Blue won the first game against Gary Kasparov. However, they were scheduled to play six games. Kasparov won three and drew two of the following five games thus defeating Deep Blue by a score of 4–2.

The Deep Blue team went back to the drawing board, made a lot of enhancements to the software, and played Kasparov again in 1997. Deep Blue won the second round against Kasparov winning the six-game rematch by a score of 3½–2½. It then became the first computer system to beat a current world champion in a match under standard chess tournament rules and time controls.

A lesser known example, and a sign that machines beating humans is becoming common place, is the achievement in the area of chess by the AlphaZero team.

Google scientists from their AlphaZero research team created a system in 2017 that took just four hours to learn the rules of chess before crushing the most advanced world champion chess program at the time called *Stockfish*. By now the question as to whether computers or humans are better at chess has been resolved.

Let's pause for a second and think about this. All of humanity's knowledge about the ancient game of chess was surpassed by a system that, if it started learning in the morning, would be done by lunch time.

The system was given the rules of chess, but it was not given any strategies or further knowledge. Then, in a few hours, AlphaZero mastered the game to the extent it was able to beat Stockfish.

In a series of 100 games against Stockfish, AlphaZero won 25 games while playing as white (white has an advantage because it goes first). It also won three games playing as black. The rest of the games were ties. Stockfish did not obtain a single win.

AlphaGo

As hard as chess is, its difficulty does not compare to the ancient game of Go.

Not only are there more possible (19 x 19) Go-board positions than there are atoms in the visible universe and the number of possible chess positions is negligible to the number of Go positions. But Go is at least several orders of magnitude more complex than a game of chess because of the large number of possible ways to let the game flow with each move towards another line of development. With Go, the number of moves in which a single stone can affect and impact the whole-board situation is also many orders of magnitude larger than that of a single piece movement with chess.

There is great example of a powerful program that can play the game of Go also developed by DeepMind called AlphaGo. AlphaGo also has three far more powerful successors, called AlphaGo Master, AlphaGo Zero, and AlphaZero.

In October 2015, the original AlphaGo became the first computer Go program to beat a human professional Go player without handicaps on a full-sized 19 x 19 board. In March 2016, it beat Lee Sedol in a five-game match. This became the first time a Go program beat a 9-dan professional without handicaps. Although AlphaGo lost to Lee Sedol in the fourth game, Lee resigned in the final game, giving a final score of 4 games to 1.

At the 2017 Future of Go Summit, the successor to AlphaGo called AlphaGo Master beat the master Ke Jie in a three-game match. Ke Jie was ranked the world No.1 ranked player at the time. After this, AlphaGo was awarded professional 9-dan by the Chinese Weiqi Association.

AlphaGo and its successors use a Monte Carlo tree search algorithm to find their moves based on knowledge previously "learned" by machine learning, specifically using deep learning and training, both playing with humans and by itself. The model is trained to predict AlphaGo's own moves and the winner's games. This neural net improves the strength of tree search, resulting in better moves and stronger play in following games.

Movie making

It is all but a certainty that within the next few decades it will be possible to create movies that are 100% computer generated. It is not unfathomable to envision a system where the input is a written script and the output is a full-length feature film. In addition, some strides have been made in natural generators. So, eventually not even the script will be needed. Let's explore this further.

Deepfakes

A deepfake is a *portmanteau*, or blend, of "deep learning" and "fake." It is an AI technique to merge video images. A common application is to overlap someone's face onto another. A nefarious version of this was used to merge pornographic scenes with famous people or to create revenge porn. Deepfakes can also be used to create fake news or hoaxes. As you can imagine, there are severe societal implications if this technology is misused.

One recent version of similar software was developed by a Chinese company called Momo who developed an app called *Zao*. It allows you to overlap someone's face over short movie clips like Titanic and the results are impressive. This and other similar applications do not come without controversy. Privacy groups are complaining that the photos submitted to the site per the terms of the user agreement become property of Momo and then can later be used for other applications.

It will be interesting to see how technology continues to advance in this area.

Movie Script Generation

They are not going to win any Academy Awards any time soon, but there are a couple projects dedicated to producing movie scripts. One of the most famous examples is Sunspring.

Sunspring is an experimental science fiction short film released in 2016. It was entirely written by using deep learning techniques. The film's script was created using a **long short-term memory** (**LSTM**) model dubbed Benjamin. Its creators are BAFTA-nominated filmmaker Oscar Sharp and NYU AI researcher Ross Goodwin. The actors in the film are Thomas Middleditch, Elisabeth Grey, and Humphrey Ker. Their character names are H, H2, and C, living in the future. They eventually connect with each other and a love triangle forms.

Originally shown at the Sci-Fi-London film festival's 48hr Challenge, it was also released online by technology news website Ars Technica in June 2016.

Underwriting and deal analysis

What is underwriting? In short, underwriting is the process by which an institution determines if they want to take a financial risk in exchange for a premium. Examples of transactions that require underwriting are:

- Issuing an insurance policy
  - Health
  - Life
  - Home
  - Driving
- Loans
  - Installment loans
  - Credit cards
  - Mortgages
  - Commercial lines of credit
- Securities underwriting and Initial Public Offerings (IPOs)

As can be expected, determining whether an insurance policy or a loan should be issued and at what price can be very costly if the wrong decision is made. For example, if a bank issues a loan and the loan defaults, it would require dozens of other performing loans to make up for that loss. Inversely, if the bank passes up on a loan where the borrower was going to make all their payments is also detrimental to the bank finances. For this reason, the bank spends considerable time analyzing or "underwriting" the loan to determine the credit worthiness of the borrower as well as the value of the collateral securing the loan.

Even with all these checks, underwriters still get it wrong and issue loans that default or bypass deserving borrowers. The current underwriting process follows a set of criteria that must be met but specially for smaller banks there is still a degree of human subjectivity in the process. This is not necessarily a bad thing. Let's visit a scenario to explore this further:

A high net worth individual recently came back from a tour around the world. Three months ago, they got a job at a prestigious medical institution and their credit score is above 800.

Would you lend money to this individual? With the characteristics given, they seem to be a good credit risk. However, normal underwriting rules might disqualify them because they haven't been employed for the last two years. Manual underwriting would look at the whole picture and probably approve them.

Similarly, a machine learning model would probably be able to flag this as a worthy account and issue the loan. Machine learning models don't have hard and fast rules but rather "learn by example."

Many lenders are already using machine learning in their underwriting. An interesting example of a company that specializes in this space is Zest Finance. Zest Finance uses AI techniques to assist lenders with their underwriting. AI can help to increase revenue and reduce risk. Most importantly well applied AI in general and Zest Finance in particular can help companies to ensure that the AI models used are compliant with a country's regulations. Some AI models can be a "black box" where it is difficult to explain why one borrower was rejected and another one was accepted. Zest Finance can fully explain data modeling results, measure business impact, and comply with regulatory requirements. One of Zest Finance's secret weapons is the use of non-traditional data, including data that a lender might have in-house, such as:

- Customer support data
- Payment histories
- Purchase transactions

They might also consider nontraditional credit variables such as:

- The way a customer fills out a form
- The method a customer uses to arrive at the site or how they navigate the site
- The amount of time taken to fill out an application

Data cleansing and transformation

Just as gas powers a car, data is the lifeblood of AI. The age-old adage of "garbage in, garbage out" remains painfully true. For this reason, having clean and accurate data is paramount to producing consistent, reproducible, and accurate AI models. Some of this data cleansing has required painstaking human involvement. By some measures, it is said that a data scientist spends about 80% of their time cleaning, preparing, and transforming their input data and 20% of the time running and optimizing their models. Examples of this are the ImageNet and MS-COCO image datasets. Both contain over a million labeled images of various objects and categories. These datasets are used to train models that can distinguish between different categories and object types. Initially, these datasets were painstakingly and patiently labeled by humans. As these systems become more prevalent, we can use AI to perform the labeling. Furthermore, there is a plethora of AI-enabled tools that help with the cleansing and deduplication process.

One good example is Amazon Lake Formation. In August 2019, Amazon made its service Lake Formation generally available. Amazon Lake Formation automates some of the steps typically involved in the creation of a data lake including the collection, cleansing, deduplication, cataloging, and publication of data. The data then can be made available for analytics and to build machine models. To use Lake Formation, a user can bring data into the lake from a range of sources using predefined templates. They can then define policies that govern data access depending on the level of access that groups across the organization require.

Some automatic preparation, cleansing, and classification that the data undergoes uses machine learning to automatically perform these tasks.

Lake Formation also provides a centralized dashboard where administrators can manage and monitor data access policies, governance, and auditing across multiple analytics engines. Users can also search for datasets in the resulting catalog. As the tool evolves in the next few months and years, it will facilitate the analysis of data using their favorite analytics and machine learning services, including:

- Databricks
- Tableau
- Amazon Redshift
- Amazon Athena
- AWS Glue
- Amazon EMR
- Amazon QuickSight
- Amazon SageMaker

Summary

This chapter provided a few examples of the applications of AI. That said, the content here doesn't begin to scratch the surface! We tried to keep the use cases to either technology that is widely available, or at least that has the potential to become available soon. It is not difficult to extrapolate how this technology is going to continue to improve, become cheaper, and be more widely available. For example, it will be quite exciting when self-driving cars start becoming popular.

However, we can all be certain that the bigger applications of AI have not yet even been conceived. Also, advances in AI will have wide implications for our society and at some point, we will have to deal with these questions:

- What happens if an AI became so evolved that it became conscious? Should it be given rights?
- If a robot replaces a human, should companies be required to continue paying payroll tax for that displaced worker?
- Will we get to a point where computers are doing everything, and if so, how will we adapt to this; how will we spend our time?
- Worse yet, does the technology enable a few individuals to control all resources? Will a universal income society emerge in which individuals can pursue their own interests? Or will the displaced masses live in poverty?

Bill Gates and Elon Musk have warned about AIs either destroying the planet in a frenzied pursuit of their own goals or doing away with humans by accident (or not so much by accident). We will take a more optimistic "half-full" view of the impact of AI, but one thing that is certain is that it will be an interesting journey.

References

1. Willingham, Emily, *A Machine Gets High Marks for Diagnosing Sick Children*, Scientific American, October 7th, 2019, `https://www.scientificamerican.com/article/a-machine-gets-high-marks-for-diagnosing-sick-children/`
2. Clark, Jack, *Google Turning Its Lucrative Web Search Over to AI Machines*, Bloomberg, October 26th, 2015, `https://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines`
3. `https://www.michaelcaines.com/michael-caines/about-michael/`

3
Machine Learning Pipelines

Model training is only a small piece of the machine learning process. Data scientists often spend a significant amount of time cleansing, transforming, and preparing data to get it ready to be consumed by a machine learning model. Since data preparation is such a time-consuming activity, we will present state of the art techniques to facilitate this activity as well as other components that together form a well-designed production machine learning pipeline.

In this chapter, we will cover the following key topics:

- What exactly is a machine learning pipeline?
- What are the components of a production-quality machine learning pipeline?
- What are the best practices when deploying machine learning models?
- Once a machine learning pipeline is in place, how can we shorten the deployment cycle?

What is a machine learning pipeline?

Many young data scientists starting their machine learning training immediately want to jump into model building and model tuning. They fail to realize that creating successful machine learning systems involves a lot more than choosing between a random forest model and a support vector machine model.

From choosing the proper ingestion mechanism to data cleansing to feature engineering, the initial steps in a machine learning pipeline are just as important as model selection. Also being able to properly measure and monitor the performance of your model in production and deciding when and how to retrain your models can be the difference between great results and mediocre outcomes. As the world changes, your input variables change, and your model must change with them.

As data science progresses, expectations get higher. Data sources become more varied, voluminous (in terms of size) and plentiful (in terms of number), and the pipelines and workflows get more complex. It doesn't help that more and more of the data we are expected to process is real-time in nature. Think of web logs, click data, e-commerce transactions, and self-driving car inputs. The data from these systems comes in fast and furious and we must have methods that can process the information faster than it is received.

Many machine learning solutions exist to implement these pipelines. It is certainly possible to set up basic machine learning pipelines using just the Python or R languages. We'll begin to build up our understanding by laying out an example of a pipeline using Python. In this chapter we will explore in detail a few architectures that utilize some of the most popular tools out there today. Some of the tools that data pipelines commonly leverage are:

- Hadoop
- Spark
- Spark Streaming
- Kafka
- Azure
- AWS
- Google Cloud Platform
- R
- SAS
- Databricks
- Python

As we'll see, some of these are more appropriate for certain stages of the pipeline. Let's perform a quick overview of the minimum steps required to set up a machine learning pipeline.

One important item to consider is that each step in the pipeline produces an output that becomes the input for the next step in the pipeline. The term *pipeline* is somewhat misleading as it implies a one-way flow of data. In reality, machine learning pipelines can be cyclical and iterative. Every step in the pipeline might be repeated to achieve better results or cleaner data. Finally, the output variable might be used as input the next time the pipeline cycle is performed.

The main steps in a machine learning pipeline are:

1. **Problem Definition**: Define the business problem.
2. **Data Ingestion**: Identify and collect the dataset.
3. **Data Preparation**: Process and prepare the data using techniques such as:
    - Impute missing values
    - Remove duplicate records
    - Normalize values (change numeric values in a dataset to use a common scale)
    - Perform another type of cleanup or mappings
    - Complete feature extraction
    - Eliminate correlated features
    - Perform feature engineering
4. **Data Segregation**: Split the data into a training set, validation set, and testing set.
5. **Model Training**: Train the machine models against the training dataset. This is the core of data science. In this chapter, we will only scratch the surface of this step and the steps that follow. There are other chapters in the book that will cover model training in more detail. It is listed here mostly to give the reader a full picture of the complete pipeline.
6. **Candidate Model Evaluation**: Measure the performance of the models using test and validation subsets of data to determine model accuracy.
7. **Model Deployment**: Once a model is chosen, deploy it into production for inference.

8. **Performance Monitoring**: Continuously monitor model performance, retrain, and calibrate accordingly. Collect new data to continue to improve the model and prevent it from becoming stale:

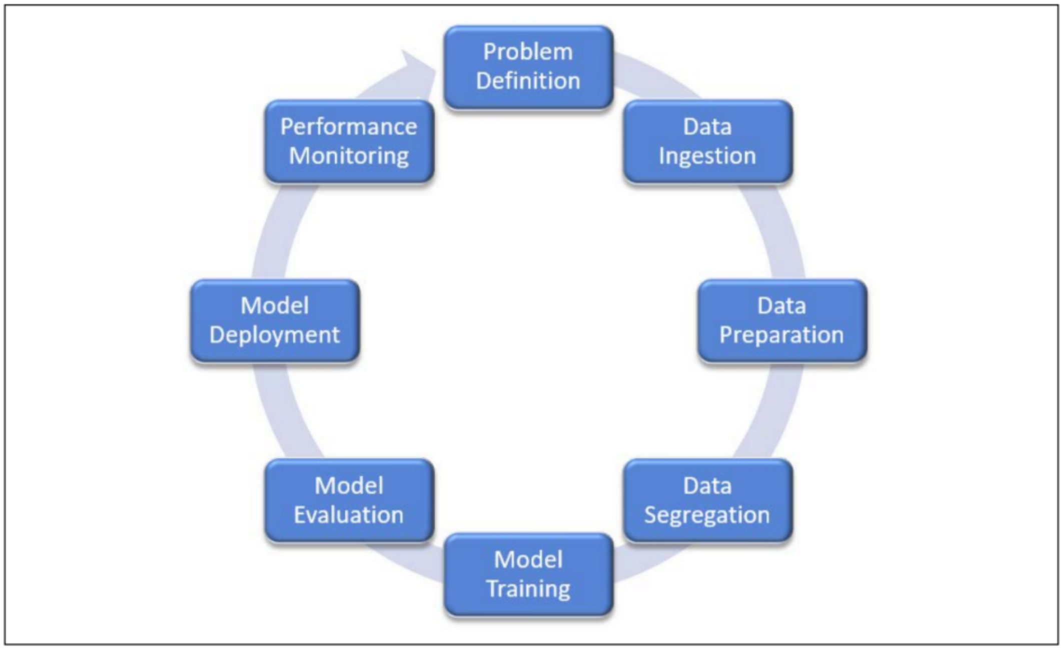


Figure 1: The machine learning pipeline

Let's explore further and dive into the components of the pipeline.

Problem definition

This might be the most critical step when setting up your pipeline. Time spent here can save you orders of magnitude of time on the later stages of the pipeline. It might mean the difference between making a technological breakthrough or failing, or it could be the difference between a startup company succeeding or the company going bankrupt. Asking and framing the right question is paramount. Consider the following cautionary tale:

> *"Bob spent years planning, executing, and optimizing how to conquer a hill. Unfortunately, it turned out to be the wrong hill."*

For example, let's say you want to create a pipeline to determine loan default prediction. Your initial question might be:

For a given loan, will it default or not?

Now, this question does not distinguish between a loan defaulting in the first month or 20 years into the loan. Obviously, a loan that defaults upon issuance is a lot less profitable than a loan that stopped performing 20 years in. So, a better question might be:

When will the loan default?

This is a more valuable question to answer. Can we make it better? It is sometimes possible that a borrower will not send in the full due payment every month. Sometimes, a borrower might send sporadic payments. To account for this, we might refine the question further:

How much money will be received for a given loan?

Let's improve it even more. A dollar today is worth more than a dollar in the future. For this reason, financial analysts use a formula to calculate the present value of money. Just as important as to how much a borrower pays on their loan is the question of when do they pay it. Also, you have the issue of prepayment. If a borrower prepays a loan, that might make the loan less profitable since less interest will be collected. Let's change the question again:

What will be the profit made on a given loan?

Are we done crafting the question? Maybe. Let's consider one more thing. There are certain input variables that by law are not allowed to be used to determine default rates. For example, race and sex are two factors that cannot be used to determine loan eligibility. One more attempt:

What will be the profit made on a given loan without using disallowed input features?

We will leave it to the reader to further refine the question. As you can see, a lot of thought needs to be given to the first and critical step in the machine learning pipeline.

Data ingestion

Once you have crafted and polished your question to a degree to which you are satisfied with, it is now time to gather the raw data that will help you answer the question. This doesn't mean that your question cannot be changed once you go on to the next steps of the pipeline. You should continuously refine your problem statement and adjust it as necessary.

Collecting the right data for your pipeline might be a tremendous undertaking. Depending on the problem you are trying to solve, obtaining relevant datasets might be quite difficult.

Another important consideration is to decide how will the data be sourced, ingested, and stored:

- What data provider or vendor should we use? Can they be trusted?
- How will it be ingested? Hadoop, Impala, Spark, just Python, and so on?
- Should it be stored as a file or in a database?
- What type of database? Traditional RDBMS, NoSQL, graph.
- Should it even be stored? If we have a real-time feed into the pipeline, it might not even be necessary or efficient to store the input.
- What format should the input be? Parquet, JSON, CSV.

Many times, we might not even have control of the input sources to decide what form it should take, and we should take it as is and then decide how it needs to be transformed. Additionally, we might not have a sole data source. There might be multiple sources that need to be consolidated, merged, and joined before we can feed them into the model (more on that later).

As much as we would like it, and even though artificial intelligence makes the long-term promise to replace human intelligence, deciding what variables should be contained in the input datasets still requires human intelligence and maybe even some good old human intuition.

If you are trying to predict stock prices, the price of the stock the previous day seems like an obvious input. Maybe not so obvious might be other inputs like interest rates, company earnings, news headlines, and so forth.

For restaurant daily sales, the previous day's sales are probably also important. Others might include: Day of the week, holiday or not holiday, rain or no rain, daily foot traffic, and so on.

For game-playing systems like chess and Go, we might provide previous games or successful strategies. As an example, one of the best ways for humans to learn chess is to learn opening and gambits that master players have used successfully in the past as well as watching completed games from past tournaments. Computers can learn in the same way by using this previous knowledge and history to decide how to play in the future.

As of now, picking relevant input variables and setting up successful models still requires the data scientist to have domain knowledge. And in some cases intimate and deep domain knowledge. Let's explore an example further.

Staying on the loan default example, let's think of some of the most important features that are relevant in order to make accurate predictions. This is a first stab at that list. Due to space limitations, we're not going to list all the features that would normally be used. We'll add and remove items as we learn from our data:

Feature Name	Feature Description	Why is it useful?
Delinquent Accounts	# of accounts on which the borrower is now delinquent.	If a borrower is having trouble paying their bills, they will probably have trouble paying new loans.
Trade Accounts	# of trades opened in past 24 months.	This is only a problem if there are too few.
Borrower Address	The address provided by the borrower in the loan application.	Drop this. Addresses are unique. Unique variables do not provide predictive ability.
Zip Code	The zip code provided by the borrower in the loan application.	This is not unique and can have predictive power.
Annual Income	The self-reported annual income provided by the borrower during registration.	More income allows the borrower to handle bigger payments more easily.
Current Balance	Average current balance of all accounts.	Not valuable in isolation. Needs to be relative.
Charge-offs	Number of charge-offs within 12 months.	Indicative of borrower's previous default behavior.
Past Due Amount	The past-due amount owed for the accounts on which the borrower is now delinquent.	Indicative of borrower's previous default behavior.
Oldest Account	Months since the oldest revolving account opened.	Indicative of a borrower's experience borrowing money.
Employment Length	Employment length in years.	Indicative of borrower's stability.
Loan Amount	The total amount committed to that loan at that point in time.	Not valuable in isolation. Needs to be relative.
Number of Inquiries	Number of personal finance inquiries.	Borrower looking for credit.
Interest Rate	Interest rate on the loan.	If a loan has a high interest rate, the payments will be more and might be harder to pay back.
Maximum Balance	Maximum current balance owed on all revolving accounts.	If it's close to 100%, this might indicate the borrower is having financial difficulties.

Months Since Last Public Record	The number of months since the last public record.	Indicative of previous financial difficulties
Number of accounts past due	Number of accounts 120 or more days past due	Indicative of current financial difficulties
Public Records	Number of derogatory public records	Indicative of previous financial difficulties
Term	The number of monthly payments on the loan.	The longer the loan, potentially the more possibility for default.
Total Current Balance	Total current balance of all accounts	Not valuable in isolation. Needs to be relative.

As we saw, some of these variables do not provide meaning on their own and they need to be combined to become predictive. This would be an example of feature engineering. Two examples of new variables are:

Credit Utilization	Balance to credit limit on all trades. Current balance compared to the credit limit.	A high percentage indicates that the borrower is "maxed out" and is having trouble obtaining new credit.
Debt to Income	Calculated using the total monthly debt payments on the total debt obligations, excluding mortgage and the requested loan, divided by the borrower's self-reported monthly income.	A low debt-to-income ratio indicates that the borrower has ample resources to pay back their obligations and should not have issues meeting them.

Data preparation

The next step is a data transformation tier that processes the raw data; some of the transformations that need to be done are:

- Data Cleansing
- Filtration
- Aggregation
- Augmentation
- Consolidation
- Storage

The cloud providers have become the major data science platforms. Some of the most popular stacks are built around:

- Azure ML service
- AWS SageMaker
- GCP Cloud ML Engine
- SAS
- RapidMiner
- Knime

One of the most popular tools to perform these transformations is Apache Spark, but it still needs a data store. For persistence, the most common solutions are:

- Hadoop Distributed File System (HDFS)
- HBase
- Apache Cassandra
- Amazon S3
- Azure Blob Storage

It's also possible to process data for machine learning in-place, inside the database; databases like SQL Server and SQL Azure are adding specific machine learning functionality to support machine learning pipelines. Spark has that built in with Spark Streaming. It can read data from HDFS, Kafka, and other sources.

There are also other alternatives like Apache Storm and Apache Heron. Whatever else is in the pipeline, initial exploration of the data is often done in interactive Jupyter notebooks or R Studio.

Some of the real-time data processing solutions out there provide fault-tolerant, scalable, low-latency data ingestion. Some of the favorite ones are:

- Apache Kafka
- Azure Event Hubs
- AWS Kinesis

Let's now explore one of the critical operations of data preparation – data cleansing. We need to ensure that the data is clean. More likely than not, the data will not be perfect, and the data quality will be less than optimal. The data can be unfit for several reasons:

Missing values

Quite often our data contains missing values or missing values are replaced by zeros or N/A. How do we deal with this problem? Following are six different ways to deal with missing values:

- **Do Nothing**: Sometimes the best action is no action. Depending on the algorithm being used, it is not always the case that we need to do anything with missing values. XGBoost is an example of an algorithm that can gracefully handle missing values.
- **Imputation using median values**: When values are missing, a reasonable value to assign to the missing data is the median of all the rest of the non-missing values for that variable. This alternative is easy and fast to calculate, and it works well for small datasets. However, it does not provide much accuracy and it doesn't consider correlations with other variables.
- **Imputation using the most frequent value or a constant**: Another option is to assign the most frequent value or a constant like zero. One advantage of this method is that it works for non-numerical variables. Like the previous method, it doesn't factor correlations with other variables and, depending on the frequency of the nulls, it can introduce a bias into the dataset.

Duplicate records or values

If two values are truly identical, it is easy to create a query or a program that can find duplicate values. The trouble starts if two records or values are supposed to identify the same entity but there is a slight difference between the two values. A traditional database query for duplicates might not find spelling errors, missing values, address changes, or people who left out their middle name. Some people use aliases.

Until recently, finding and fixing duplicate records has been a manual process that is time-intensive and resource-consuming. However, some techniques and research are starting to emerge that use AI to find duplicates. Unless all the details match exactly, it is difficult to determine whether different records refer to the same entity. Additionally, often most duplicates are false positives. Two individuals might share the same name, address, and date of birth but still be different people.

The solution to identifying duplicates is to use fuzzy matching instead of exact matching. Fuzzy matching is a computer-assisted technique to score data similarity. It is used extensively to perform fuzzy matching. Discussing fuzzy matching is beyond the scope of this book, but it may be useful for the reader to investigate this topic further.

Feature scaling

Datasets often contain features with varying magnitudes. This kind of variation in the magnitudes in the features often has a detrimental effect on the accuracy of predictions (but not always; for example, Random Forest does not need feature scaling). Many machine learning algorithms use Euclidean distance between the data points for their calculations. If we don't make this adjustment, features with a high order of magnitude will have an over-weighted impact on the results.

The most common methods for feature scaling are:

- Rescaling (min-max normalization)
- Mean normalization
- Standardization (Z-score normalization)
- Scaling to unit length

Inconsistent values

Data can contain often contain inconsistent values. Furthermore, data can be inconsistent in a variety of ways. An example of inconsistent data is a street address modifier. Consider these data points:

- Fifth Avenue
- Fifth Ave
- Fifth Av
- Fifth Av.

As humans, we can quickly determine that all these examples are truly the same value. Computers have a harder time in drawing this conclusion.

Two approaches to handle this are rule-based and example-based. A rule-based system will work better when there is less variability in the data, and it doesn't change quickly. The rule-based approach breaks when we have fast moving data.

Consider a spam filter. We could create a rule that marks as spam anything that has the word "Viagra," but spammers might get smart and start changing the data to bypass the rule ("Vi@gra"). A machine learning example-based cleanser would work better in this case.

Sometimes, we might want to consider a hybrid approach and use both methods. For instance, a person's height should always be a positive value. So, we could write a rule for that. For other values with more variability, we can use a machine learning approach.

Inconsistent date formatting

- 11/1/2016
- 11/01/2016
- 11/1/16
- Nov 1 16
- November 1st, 2016

These are all the same value. So, we need to standardize dates.

This is not a comprehensive list of data preparation but instead is designed to give a taste of the different transformations that need to be done to cleanse and prepare data in order to be useful.

Data segregation

In order to train a model using the processed data, it is recommended to split the data into two subsets:

- Training data
- Testing data

and sometimes into three:

- Training data
- Validation data
- Testing data

You can then train the model on the training data in order to later make predictions on the test data. The training set is visible to the model and it is trained on this data. The training creates an inference engine that can be later applied to new data points that the model has not previously seen. The test dataset (or subset) represents this unseen data and it now can be used to make predictions on this previously unseen data.

Model training

Once we split the data it is now time to run the training and test data through a series of models and assess the performance of a variety of models and determine how accurate each candidate model is. This is an iterative process and various algorithms might be tested until you have a model that sufficiently answers your question.

We will delve deeper into this step within later chapters. Plenty of material is provided on model selection in the rest of the book.

Candidate model evaluation and selection

After we train our model with various algorithms comes another critical step. It is time to select which model is optimal for the problem at hand. We don't always pick the best performing model. An algorithm that performs well with the training data might not perform well in production because it might have overfitted the training data. At this point in time, model selection is more of an art than a science but there are some techniques that are explored further to decide which model is best.

Model deployment

Once a model is chosen and finalized, it is now ready to be used to make predictions. It is typically exposed via an API and embedded in decision-making frameworks as part of an analytics solution.

How it gets exposed and deployed should be determined by the business requirements. Some questions to consider in the deployment selection:

- Does the system need to be able to make predictions in real-time (if yes, how fast: in milliseconds, seconds, minutes, hours?)
- How often do the models need to be updated?
- What amount of volume or traffic is expected?
- What is the size of the datasets?
- Are there regulations, policies and other constraints that need to be followed and abided by?

Once you've solidified the requirements, we can now consider a high-level architecture for the deployment of the model. Following are a variety of options. This is not an exhaustive list by any means, but it does encompass some of the more popular architectures:

	RESTful API Architecture	**Shared DB Architecture**	**Streaming Architecture**	**Mobile App Architecture**
Training Method	Batch	Batch	Streaming	Streaming
Prediction Method	Real-time	Batch	Streaming	Real-time

Result Delivery	Via RESTful API	Via Shared Database	Streaming via Message Queue	Via in-process API on mobile device
Prediction Latency	Low	High	Very Low	Low
System Maintainability	Medium	Easy	Difficult	Medium

As summarized in the table, each of these four options has its pros and cons. Many more considerations need to be accounted for as we drill down into the specifics of the architecture. As an example, each of these architectures can be implemented using a modularized microservice architecture or in a monolithic manner. Again, the choice should be driven by the business requirements. For example, the monolithic approach might be picked because we have a very limited use case that required an extremely low latency.

Regardless of the architecture chosen for the model deployment it is a good idea to use the following principles:

- **Reproducibility**: Store all the model inputs and outputs, as well as all relevant metadata such as configuration, dependencies, geography, and time zones. required to explain a past prediction. Ensure the latest versioning for each of these deployment bundles is available, which should also include the training data. This is especially important for domains that are highly regulated, banking, for example.
- **Automation**: As early as possible, automate as much as possible of the training and model publishing.
- **Extensibility**: If models need to be updated on a regular basis, a plan needs to be put in place from the beginning.
- **Modularity**: As much as possible, modularize your code and make sure that controls are put in place to faithfully reproduce the pipelines across environments (DEV, QA, TEST).
- **Testing**: Allocate a significant part of your schedule for testing the machine learning pipeline. Automate the testing as much as possible and integrate into your process from the beginning. Explore **Test Driven Development** (**TDD**) and **Behavior Driven Development** (**BDD**).

Performance monitoring

Once a model makes it into production, our work is not finished. Moving a model into production may not be easy, but once the model is deployed it must be closely monitored to make sure that the model is performing satisfactorily. There are various steps involved in getting the model into production. The model is continuously monitored to observe how it behaved in the real world and calibrated accordingly. New data is collected to incrementally improve it. Similarly, monitoring a deployed machine learning model requires attention from various perspectives to make sure that the model is performing. Let's analyze these different metrics that need to be considered when monitoring machine learning models, and why each one of them is important:

Model performance

Performance in the data science context does not mean how fast the model is running, but rather how accurate are the predictions. Data scientists monitoring machine learning models are primarily looking at a single metric: drift. Drift happens when the data is no longer a relevant or useful input to the model. Data can change and lose its predictive value. Data scientists and engineers must monitor the models constantly to make sure that the model features continue to be like the data points used during the model training. If the data drifts, the prediction results will become less accurate because the input features are out of date or no longer relevant. As an example, think of stock market data. Thirty years ago, the market was dramatically different. Some ways in which it was different include the following:

- Volume in the stock exchanges was significantly lower than it is today
- High-frequency trading was not even an idea
- Passive index funds were much less popular

As you can imagine, these characteristics make stock performance significantly different. If we train our models with 30-year-old data, they are more than likely not going to be able to perform with today's data.

Operational performance

Machine learning pipelines at the end of the day are still software systems. For this reason, it's still important to monitor resource consumption, including:

- **CPU utilization**: Identifies spikes and whether or not they can be explained.
- **Memory usage**: How much memory is being consumed.
- **Disk usage**: How much disc space is our application consuming.

- **Network I/O traffic**: If our application spans across instances, it is important to measure the network traffic.
- **Latency**: The amount of time it takes for a data transfer to occur.
- **Throughput**: The amount of data successfully transferred.

If these metrics change, they need to be analyzed to understand why these changes are happening.

Total cost of ownership (TCO)

Data scientists need to monitor their model performance in terms of records per second. Although this gives some insight into the efficiency of the model, companies should also be focused on the benefit they gain from the model versus the cost. It is recommended to monitor the cost of all the steps of the machine learning pipeline. If this information is closely tracked, the business can make smart decisions on how to keep costs down and how to take advantage of new opportunities or whether certain pipelines are not providing enough value and they need to be changed or shut down.

Service performance

Technology not in the context of a business problem is useless. Businesses often have, or at least should have, **service level agreements** (**SLAs**) in place with the technology department. Examples of SLAs:

- Fix all critical bugs within one day
- Ensure that an API responds within 100 ms
- Process at least a million predictions per hour
- Complex models must be designed, developed, and deployed within 3 months

For the business to perform optimally it's important to establish, monitor, and meet previously agreed upon SLAs.

Machine learning models can be mission critical to a business. A key to ensure that they do not become a bottleneck is to properly monitor the deployed models. As part of your machine learning pipelines, make sure that deployed machine learning models are monitored and compared against SLAs to ensure satisfactory business results.

Summary

This chapter laid out in detail what are the different steps involved in creating a machine learning pipeline. This tour should be considered an initial overview of the steps involved. As the book progresses you will learn how to improve your own pipelines, but we did learn some of the best practices and most popular tools that are used to set up pipelines today. In review the steps to a successful pipeline are:

- Problem definition
- Data ingestion
- Data preparation
- Data segregation
- Candidate model selection
- Model deployment
- Performance monitoring

In the next chapter we'll delve deeper into one of the steps of the machine learning pipeline. We'll learn how to perform feature selection and we'll learn what is feature engineering. These two techniques are critically important to improve model performance.

4
Feature Selection and Feature Engineering

Feature selection – also known as variable selection, attribute selection, or variable subset selection – is a method used to select a subset of features (variables, dimensions) from an initial dataset. Feature selection is a key step in the process of building machine learning models and can have a huge impact on the performance of a model. Using correct and relevant features as the input to your model can also reduce the chance of overfitting, because having more relevant features reduces the opportunity of a model to use noisy features that don't add signal as input. Lastly, having less input features decreases the amount of time that it will take to train a model. Learning which features to select is a skill developed by data scientists that usually only comes from months and years of experience and can be more of an art than a science. Feature selection is important because it can:

- Shorten training times
- Simplify models and make them easier to interpret
- Enhances testing set performance by reducing overfitting

One important reason to drop features is the high correlation and redundancy between input variables or the irrelevancy of certain features. These input variables can thus be removed without incurring much loss of information. Redundant and irrelevant are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

Feature engineering in some ways is the opposite of feature selection. With feature selection, you remove variables. In feature engineering, you create new variables to enhance the model. In many cases, you are using domain knowledge for the enhancement.

Feature selection and feature engineering is an important component of your machine learning pipeline, and that's why a whole chapter is devoted to this topic.

By the end of this chapter, you will know:

- How to decide if a feature should be dropped from a dataset
- Learn about the concepts of collinearity, correlation, and causation
- Understand the concept of feature engineering and how it differs from feature selection
- Learn about the difference between manual feature engineering and automated feature engineering. When is it appropriate to use each one?

Feature selection

In the previous chapter, we explored the components of a machine learning pipeline. A critical component of the pipeline is deciding which features will be used as inputs to the model. For many models, a small subset of the input variables provide the lion's share of the predictive ability. In most datasets, it is common for a few features to be responsible for the majority of the information signal and the rest of the features are just mostly noise.

It is important to lower the amount of input features for a variety of reasons including:

- Reducing the multi collinearity of the input features will make the machine learning model parameters easier to interpret. *Multicollinearity* (also *collinearity*) is a phenomenon observed with features in a dataset where one predictor feature in a regression model can be linearly predicted from the other's features with a substantial degree of accuracy.
- Reducing the time required to run the model and the amount of storage space the model needs will allow us to run more variations of the models leading to quicker and better results.
- The smaller number of input features a model requires, the easier it is to explain it. When the number of features goes up, the explainability of the model goes down. Reducing the amount of input features also makes it easier to visualize the data when reduced to low dimensions (for example, 2D or 3D).
- As the number of dimensions increases, the possible configurations increase exponentially, and the number of configurations covered by an observation decreases. As you have more features to describe your target, you might be able to describe the data more precisely, but your model will not generalize with new data points – your model will overfit the data. This is known as the *curse of dimensionality*.

Let's think about this intuitively by going through an example. There is a real estate site in the US that allows real estate agents and homeowners to list homes for rent or for sale. Zillow is famous, among other things, for its Zestimate. The Zestimate is an estimated price using machine learning. It is the price that Zillow estimates a home will sell for if it was put on the market today. The Zestimates are constantly updated and recalculated. How does Zillow come up with this number? If you want to learn more about it, there was a competition on Kaggle that has great resources on the Zestimate. You can find out more here:

`https://www.kaggle.com/c/zillow-prize-1`

The exact details of the Zestimate algorithm are proprietary, but we can make some assumptions. We will now start to explore how we can come up with our own Zestimate. Let's come up with a list of potential input variables for our machine learning model and the reasons why they might be valuable:

- **Square footage**: Intuitively, the bigger the home, the more expensive it will be.
- **Number of bedrooms**: More rooms, more cost.
- **Number of bathrooms**: Bedrooms need bathrooms.
- **Mortgage interest rates**: If rates are low, that makes mortgage payments lower, which means potential homeowners can afford a more expensive home.
- **Year built**: In general, newer homes are typically more expensive than older homes. Older homes normally need more repairs.
- **Property taxes**: If property taxes are high, that will increase the monthly payments and homeowners will only be able to afford a less expensive home.
- **House color**: At first glance, this might not seem like a relevant variable, but what if the home is painted lime green?
- **Zip code**: Location, location, location. In real estate, where the home is located is an important determinant of price. In some cases, a house in one block can be hundreds of thousands of dollars more than a house on the next block. Location can be that important.
- **Comparable sales**: One of the metrics that is commonly used by appraisers and real estate agents to value a home is to look for similar properties to the "subject" property that have been recently sold or at least are listed for sale, to see what the sale price was or what the listing price currently is.
- **Tax assessment**: Property taxes are calculated based on what the county currently thinks the property is worth. This is publicly accessible information.

These could all potentially be variables that have high predictive power, but intuitively we can probably assume that square footage, the number of bedrooms, and number of bathrooms are highly correlated. Also, intuitively, square footage provides more precision than the number of bedrooms or the number of bathrooms. So, we can probably drop the number of bedrooms and the number bathrooms and keep the square footage and don't lose much accuracy. Indeed, we could potentially increase the accuracy, by reducing the noise.

Furthermore, we can most likely drop the house color without losing precision.

Features that can be dropped without impacting the model's precision significantly fall into two categories:

- **Redundant**: This is a feature that is highly correlated to other input features and therefore does not add much new information to the signal.
- **Irrelevant**: This is a feature that has a low correlation with the target feature and for that reason provides more noise than signal.

One way to find out if our assumptions are correct is to train our model with and without our assumptions and see what produces the better results. We could use this method with every single feature, but in cases where we have a high number of features the possible number of combinations can escalate quickly.

As we mentioned previously, exploratory data analysis can be a good way to get an intuitive understanding and to obtain insights into the dataset we are working with. Let's analyze three approaches that are commonly used to obtain these insights. They are:

- Feature importance
- Univariate selection
- Correlation matrix with heatmap

Feature importance

The importance of each feature of a dataset can be established by using this method.

Feature importance provides a score for each feature in a dataset. A higher score means the feature has more importance or relevancy in relation to the output feature.

Feature importance is normally an inbuilt class that comes with *Tree-Based Classifiers*. In the following example, we use the *Extra Tree Classifier* to determine the top five features in a dataset:

```
import pandas as pd
```

```
from sklearn.ensemble import ExtraTreesClassifier
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv("train.csv")
X = data.iloc[:,0:20]  #independent columns
y = data.iloc[:,-1]    # pick last column for the target feature

model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class
#feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.
columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.show()
```

You should see this as output:

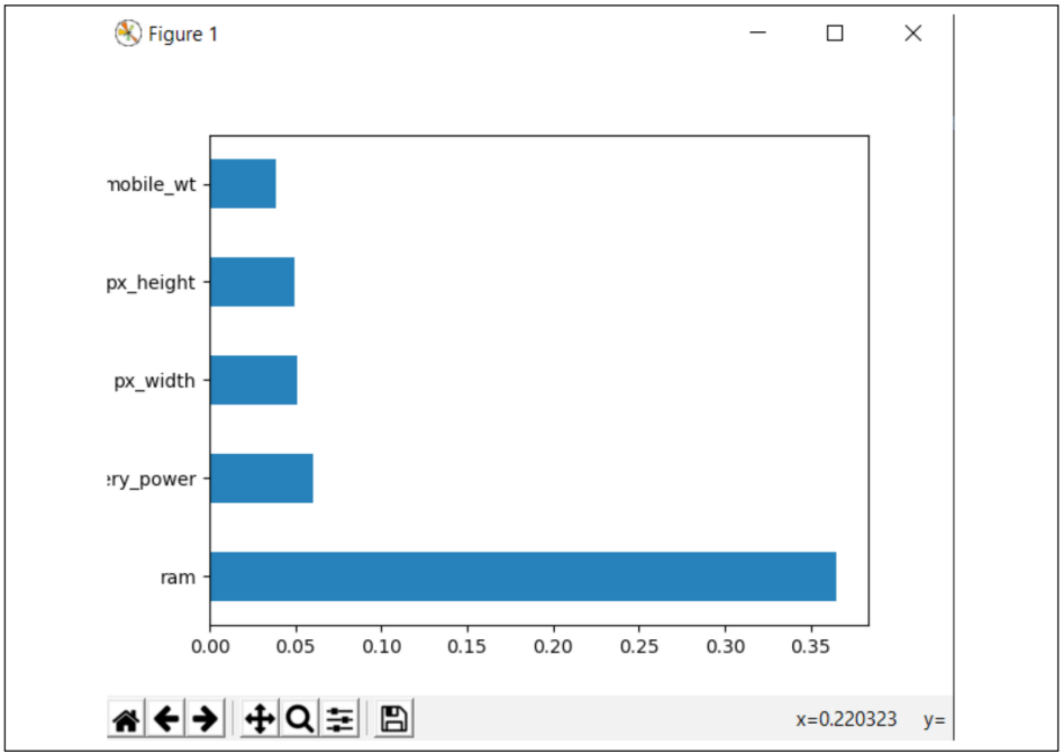


Figure 1: Feature importance graph

Index