Artificial Intelligence for Big Data

Complete guide to automating Big Data solutions using Artificial Intelligence techniques

Packt

www.packt.com

By Anand Deshpande and Manish Kumar

Artificial Intelligence for Big Data

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Sunith Shetty
Acquisition Editor: Tushar Gupta
Content Development Editor: Tejas Limkar
Technical Editor: Dinesh Chaudhary
Copy Editor: Safis Editing
Project Coordinator: Manthan Patel
Proofreader: Safis Editing
Indexer: Priyanka Dhadke
Graphics: Tania Dutta
Production Coordinator: Aparna Bhagat

First published: May 2018

Production reference: 1170518

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

ISBN 978-1-78847-217-3

`www.packtpub.com`

Table of Contents

Preface

We are at an interesting juncture in the evolution of the digital age, where there is an enormous amount of computing power and data in the hands of everyone. There has been an exponential growth in the amount of data we now have in digital form. While being associated with data-related technologies for more than 6 years, we have seen a rapid shift towards enterprises that are willing to leverage data assets initially for insights and eventually for advanced analytics. What sounded like hype initially has become a reality in a very short period of time. Most companies have realized that data is the most important asset needed to stay relevant. As practitioners in the big data analytics industry, we have seen this shift very closely by working with many clients of various sizes, across regions and functional domains. There is a common theme evolving toward open distributed open source computing to store data assets and perform advanced analytics to predict future trends and risks for businesses.

This book is an attempt to share the knowledge we have acquired over time to help new entrants in the big data space to learn from our experience. We realize that the field of artificial intelligence is vast and it is just the beginning of a revolution in the history of mankind. We are going to see AI becoming mainstream in everyone's life and complementing human capabilities to solve some of the problems that have troubled us for a long time. This book takes a holistic approach into the theory of machine learning and AI, starting from the very basics to building applications with cognitive intelligence. We have taken a simple approach to illustrate the core concepts and theory, supplemented by illustrative diagrams and examples.

It will be encouraging for us for readers to benefit from the book and fast-track their learning and innovation into one of the most exciting fields of computing so they can create a truly intelligent system that will augment our abilities to the next level.

Who this book is for

This book is for anyone with a curious mind who is exploring the fields of machine learning, artificial intelligence, and big data analytics. This book does not assume that you have in-depth knowledge of statistics, probability, or mathematics. The concepts are illustrated with easy-to-follow examples. A basic understanding of the Java programming language and the concepts of distributed computing frameworks (Hadoop/Spark) will be an added advantage. This book will be useful for data scientists, members of technical staff in IT products and service companies, technical project managers, architects, business analysts, and anyone who deals with data assets.

What this book covers

`Chapter 1`, *Big Data and Artificial Intelligence Systems*, will set the context for the convergence of human intelligence and machine intelligence at the onset of a data revolution. We have the ability to consume and process volumes of data that were never possible before. We will understand how our quality of life is the result of our decisive power and actions and how it translates into the machine world. We will understand the paradigm of big data along with its core attributes before diving into the basics of AI. We will conceptualize the big data frameworks and see how they can be leveraged for building intelligence into machines. The chapter will end with some of the exciting applications of Big Data and AI.

`Chapter 2`, *Ontology for Big Data*, introduces semantic representation of data into knowledge assets. A semantic and standardized view of the world is essential if we want to implement artificial intelligence, which fundamentally derives knowledge from data and utilizes contextual knowledge for insights and meaningful actions in order to augment human capabilities. This semantic view of the world is expressed as ontologies.

`Chapter 3`, *Learning from Big Data*, shows broad categories of machine learning as supervised and unsupervised learning, and we understand some of the fundamental algorithms that are very widely used. In the end, we will have an overview of the Spark programming model and Spark's **Machine Learning library** (Spark **MLlib**).

`Chapter 4`, *Neural Networks for Big Data*, explores neural networks and how they have evolved with the increase in computing power with distributed computing frameworks. Neural networks get their inspiration from the human brain and help us solve some very complex problems that are not feasible with traditional mathematical models.

`Chapter 5`, *Deep Big Data Analytics*, takes our understanding of neural networks to the next level by exploring deep neural networks and the building blocks of deep learning: gradient descent and backpropagation. We will review how to build data preparation pipelines, the implementation of neural network architectures, and hyperparameter tuning. We will also explore distributed computing for deep neural networks with examples using the DL4J library.

`Chapter 6`, *Natural Language Processing*, introduces some of the fundamentals of **Natural Language Processing** (**NLP**). As we build intelligent machines, it is imperative that the interface with the machines should be as natural as possible, like day-to-day human interactions. NLP is one of the important steps towards that. We will be learning about text preprocessing, techniques for extraction of relevant features from natural language text, application of NLP techniques, and the implementation of sentiment analysis with NLP.

`Chapter 7`, *Fuzzy Systems*, explains that a level of fuzziness is essential if we want to build intelligent machines. In the real-world scenarios, we cannot depend on exact mathematical and quantitative inputs for our systems to work with, although our models (deep neural networks, for example) require actual inputs. The uncertainties are more frequent and, due to the nature of real-world scenarios, are amplified by incompleteness of contextual information, characteristic randomness, and ignorance of data. Human reasoning are capable enough to deal with these attributes of the real world. A similar level of fuzziness is essential for building intelligent machines that can complement human capabilities in a real sense. In this chapter, we are going to understand the fundamentals of fuzzy logic, its mathematical representation, and some practical implementations of fuzzy systems.

`Chapter 8`, *Genetic Programming*, big data mining tools need to be empowered by computationally efficient techniques to increase the degree of efficiency. Genetic algorithms over data mining create great, robust, computationally efficient, and adaptive systems. In fact, with the exponential explosion of data, data analytics techniques go on to take more time and inversely affect the throughput. Also due to their static nature, complex hidden patterns are often left out. In this chapter, we want to show how to use genes to mine data with great efficiency. To achieve this objective, we'll introduce the basics of genetic programming and the fundamental algorithms.

`Chapter 9`, *Swarm Intelligence*, analyzes the potential of swarm intelligence for solving big data analytics problems. Based on the combination of swarm intelligence and data mining techniques, we can have a better understanding of the big data analytics problems and design more effective algorithms to solve real-world big data analytics problems. In this chapter, we'll show how to use these algorithms in big data applications. The basic theory and some programming frameworks will be also explained.

`Chapter 10`, *Reinforcement Learning*, covers reinforcement learning as one of the categories of machine learning. With reinforcement learning, the intelligent agent learns the right behavior based on the reward it receives as per the actions it takes within a specific environmental context. We will understand the fundamentals of reinforcement learning, along with mathematical theory and some of the commonly used techniques for reinforcement learning.

`Chapter 11`, *Cyber Security*, analyzes the cybersecurity problem for critical infrastructure. Data centers, data base factories, and information system factories are continuously under attack. Online analysis can detect potential attacks to ensure infrastructure security. This chapter also explains **Security Information and Event Management** (**SIEM**). It emphasizes the importance of managing log files and explains how they can bring benefits. Subsequently, Splunk and ArcSight ESM systems are introduced.

`Chapter 12`, *Cognitive Computing*, introduces cognitive computing as the next level in the development of artificial intelligence. By leveraging the five primary human senses along with mind as the sixth sense, a new era of cognitive systems can begin. We will see the stages of AI and the natural progression towards strong AI, along with the key enablers for achieving strong AI. We will take a look at the history of cognitive systems and see how that growth is accelerated with the availability of big data, which brings large data volumes and processing power in a distributed computing framework.

To get the most out of this book

The chapters in this book are sequenced in such a way that the reader can progressively learn about *Artificial Intelligence for Big Data* starting from the fundamentals and eventually move towards cognitive intelligence. `Chapter 1`, *Big Data and Artificial Intelligence Systems*, to `Chapter 5`, *Deep Big Data Analytics*, cover the basic theory of machine learning and establish the foundation for practical approaches to AI. Starting from `Chapter 6`, *Natural Language Processing*, we conceptualize theory into practical implementations and possible use cases. To get the most out of this book, it is recommended that the first five chapters are read in order. From `Chapter 6`, *Natural Language Processing*, onward, the reader can choose any topic of interest and read in whatever sequence they prefer.

Download the example code files

You can download the example code files for this book from your account at `www.packtpub.com`. If you purchased this book elsewhere, you can visit `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 `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 onscreen 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-for-Big-Data`. 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: `http://www.packtpub.com/sites/default/files/downloads/ArtificialIntelligenceforBigData_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Mount the downloaded `WebStorm-10*.dmg` disk image file as another disk in your system."

A block of code is set as follows:

```
StopWordsRemover remover = new StopWordsRemover()
  .setInputCol("raw")
  .setOutputCol("filtered");
```

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

```
$ mkdir css
$ cd css
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Select **System info** from the **Administration** panel."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

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

1

Big Data and Artificial Intelligence Systems

The human brain is one of the most sophisticated machines in the universe. It has evolved for thousands of years to its current state. As a result of continuous evolution, we are able to make sense of nature's inherent processes and understand cause and effect relationships. Based on this understanding, we are able to learn from nature and devise similar machines and mechanisms to constantly evolve and improve our lives. For example, the video cameras we use derived from the understanding of the human eye.

Fundamentally, human intelligence works on the paradigm of *sense*, *store*, *process*, and *act*. Through the sensory organs, we gather information about our surroundings, store the information (memory), process the information to form our beliefs/patterns/links, and use the information to act based on the situational context and stimulus.

Currently, we are at a very interesting juncture of evolution where the human race has found a way to store information in an electronic format. We are also trying to devise machines that imitate the human brain to be able to sense, store, and process information to make meaningful decisions and complement human abilities.

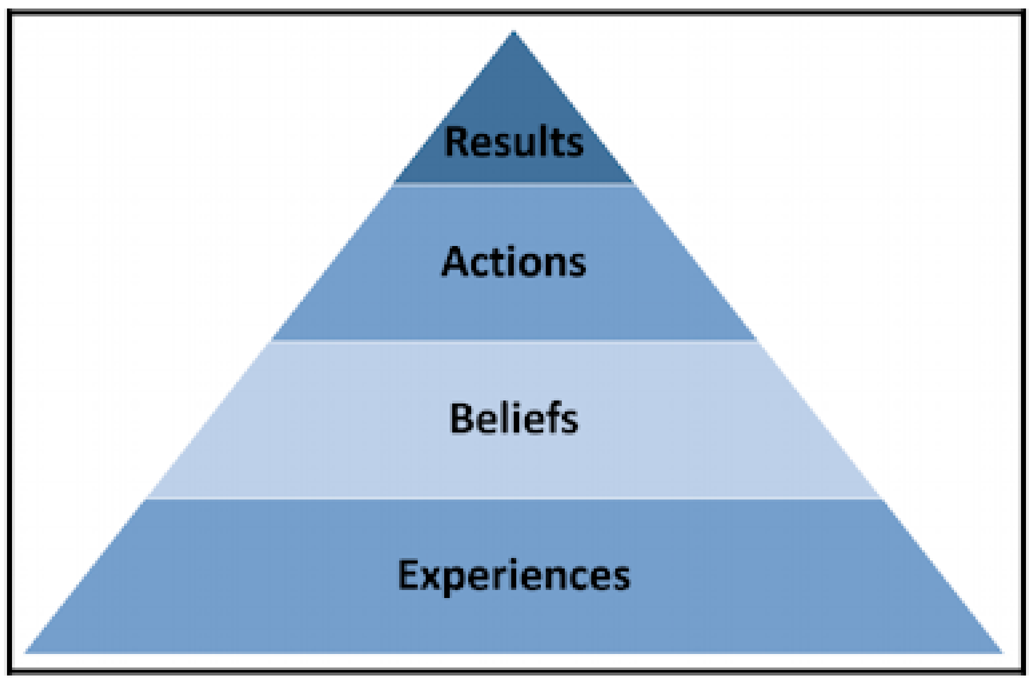
This introductory chapter will set the context for the convergence of human intelligence and machine intelligence at the onset of a data revolution. We have the ability to consume and process volumes of data that were never possible before. We will understand how our quality of life is the result of our decisive power and actions and how it translates to the machine world. We will understand the paradigm of Big Data along with its core attributes before diving into **artificial intelligence** (**AI**) and its basic fundamentals. We will conceptualize the Big Data frameworks and how those can be leveraged for building intelligence into machines. The chapter will end with some of the exciting applications of Big Data and AI.

We will cover the following topics in the chapter:

- Results pyramid
- Comparing the human and the electronic brain
- Overview of Big Data

Results pyramid

The quality of human life is a factor of all the decisions we make. According to Partners in Leadership, the results we get (positive, negative, good, or bad) are a result of our actions, our actions are a result of the beliefs we hold, and the beliefs we hold are a result of our experiences. This is represented as a results pyramid as follows:



At the core of the results pyramid theory is the fact that it is certain that we cannot achieve better or different results with the same actions. Take an example of an organization that is unable to meets its goals and has diverted from its vision for a few quarters. This is a result of certain actions that the management and employees are taking. If the team continues to have same beliefs, which translate to similar actions, the company cannot see noticeable changes in its outcomes. In order to achieve the set goals, there needs to be a fundamental change in day-to-day actions for the team, which is only possible with a new set of beliefs. This means a cultural overhaul for the organization.

Similarly, at the core of computing evolution, man-made machines cannot evolve to be more effective and useful with the same outcomes (actions), models (beliefs), and data (experiences) that we have access to traditionally. We can evolve for the better if human intelligence and machine power start complementing each other.

What the human brain does best

While the machines are catching up fast in the quest for intelligence, nothing can come close to some of the capabilities that the human brain has.

Sensory input

The human brain has an incredible capability to gather sensory input using all the senses in parallel. We can see, hear, touch, taste, and smell at the same time, and process the input in real time. In terms of computer terminology, these are various data sources that stream information, and the brain has the capacity to process the data and convert it into information and knowledge. There is a level of sophistication and intelligence within the human brain to generate different responses to this input based on the situational context.

For example, if the outside temperature is very high and it is sensed by the skin, the brain generates triggers within the lymphatic system to generate sweat and bring the body temperature under control. Many of these responses are triggered in real time and without the need for conscious action.

Storage

The information collected from the sensory organs is stored consciously and subconsciously. The brain is very efficient at filtering out the information that is non-critical for survival. Although there is no confirmed value of the storage capacity in the human brain, it is believed that the storage capacity is similar to terabytes in computers. The brain's information retrieval mechanism is also highly sophisticated and efficient. The brain can retrieve relevant and related information based on context. It is understood that the brain stores information in the form of linked lists, where the objects are linked to each other by a relationship, which is one of the reasons for the availability of data as information and knowledge, to be used as and when required.

Processing power

The human brain can read sensory input, use previously stored information, and make decisions within a fraction of a millisecond. This is possible due to a network of neurons and their interconnections. The human brain possesses about 100 billion neurons with one quadrillion connections known as synapses wiring these cells together. It coordinates hundreds of thousands of the body's internal and external processes in response to contextual information.

Low energy consumption

The human brain requires far less energy for sensing, storing, and processing information. The power requirement in calories (or watts) is insignificant compared to the equivalent power requirements for electronic machines. With growing amounts of data, along with the increasing requirement of processing power for artificial machines, we need to consider modeling energy utilization on the human brain. The computational model needs to fundamentally change towards quantum computing and eventually to bio-computing.

What the electronic brain does best

As the processing power increases with computers, the electronic brain—or computers—are much better when compared to the human brain in some aspects, as we will explore in the following sections.

Speed information storage

The electronic brain (computers) can read and store high volumes of information at enormous speeds. Storage capacity is exponentially increasing. The information is easily replicated and transmitted from one place to another. The more information we have at our disposal for analysis, pattern, and model formation, the more accurate our predictions will be, and the machines will be much more intelligent. Information storage speed is consistent across machines when all factors are constant. However, in the case of the human brain, storage and processing capacities vary based on individuals.

Processing by brute force

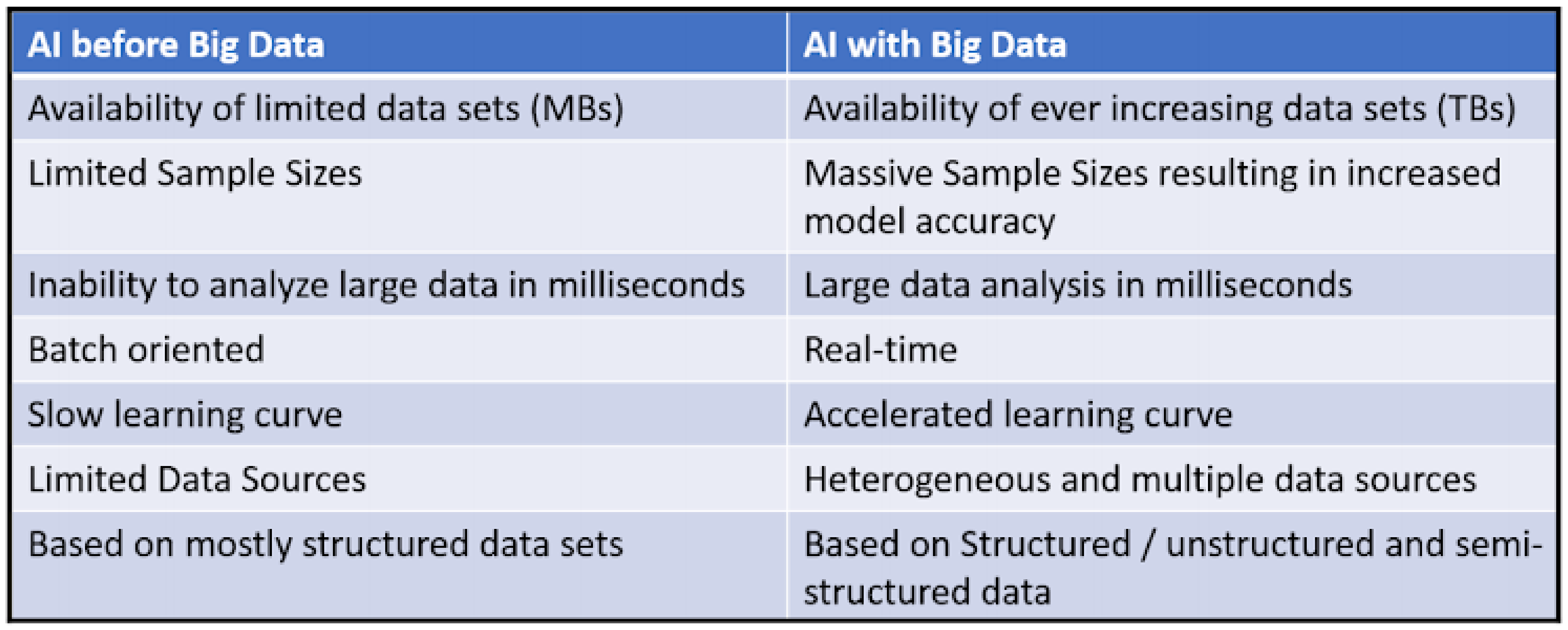
The electronic brain can process information using brute force. A distributed computing system can scan/sort/calculate and run various types of compute on very large volumes of data within milliseconds. The human brain cannot match the brute force of computers.

Computers are very easy to network and collaborate with in order to increase collective storage and processing power. The collective storage can collaborate in real time to produce intended outcomes. While human brains can collaborate, they cannot match the electronic brain in this aspect.

Best of both worlds

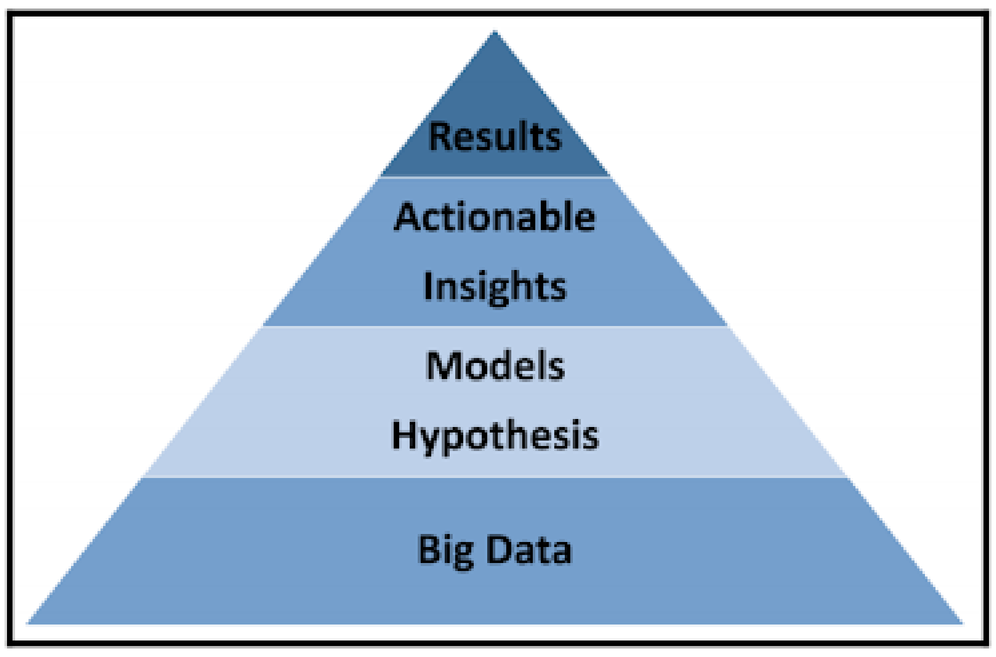
AI is finding and taking advantage of the best of both worlds in order to augment human capabilities. The sophistication and efficiency of the human brain and the brute force of computers combined together can result in intelligent machines that can solve some of the most challenging problems faced by human beings. At that point, the AI will complement human capabilities and will be a step closer to social inclusion and equanimity by facilitating collective intelligence. Examples include epidemic predictions, disease prevention based on DNA sampling and analysis, self driving cars, robots that work in hazardous conditions, and machine assistants for differently able people.

Taking a statistical and algorithmic approach to data in machine learning and AI has been popular for quite some time now. However, the capabilities and use cases were limited until the availability of large volumes of data along with massive processing speeds, which is called Big Data. We will understand some of the Big Data basics in the next section. The availability of Big Data has accelerated the growth and evolution of AI and machine learning applications. Here is a quick comparison of AI before and with with Big Data:



AI before Big Data	AI with Big Data
Availability of limited data sets (MBs)	Availability of ever increasing data sets (TBs)
Limited Sample Sizes	Massive Sample Sizes resulting in increased model accuracy
Inability to analyze large data in milliseconds	Large data analysis in milliseconds
Batch oriented	Real-time
Slow learning curve	Accelerated learning curve
Limited Data Sources	Heterogeneous and multiple data sources
Based on mostly structured data sets	Based on Structured / unstructured and semi-structured data

The primary goal of AI is to implement human-like intelligence in machines and to create systems that gather data, process it to create models (hypothesis), predict or influence outcomes, and ultimately improve human life. With Big Data at the core of the pyramid, we have the availability of massive datasets from heterogeneous sources in real time. This promises to be a great foundation for an AI that really augments human existence:



Big Data

"We don't have better algorithms, We just have more data."

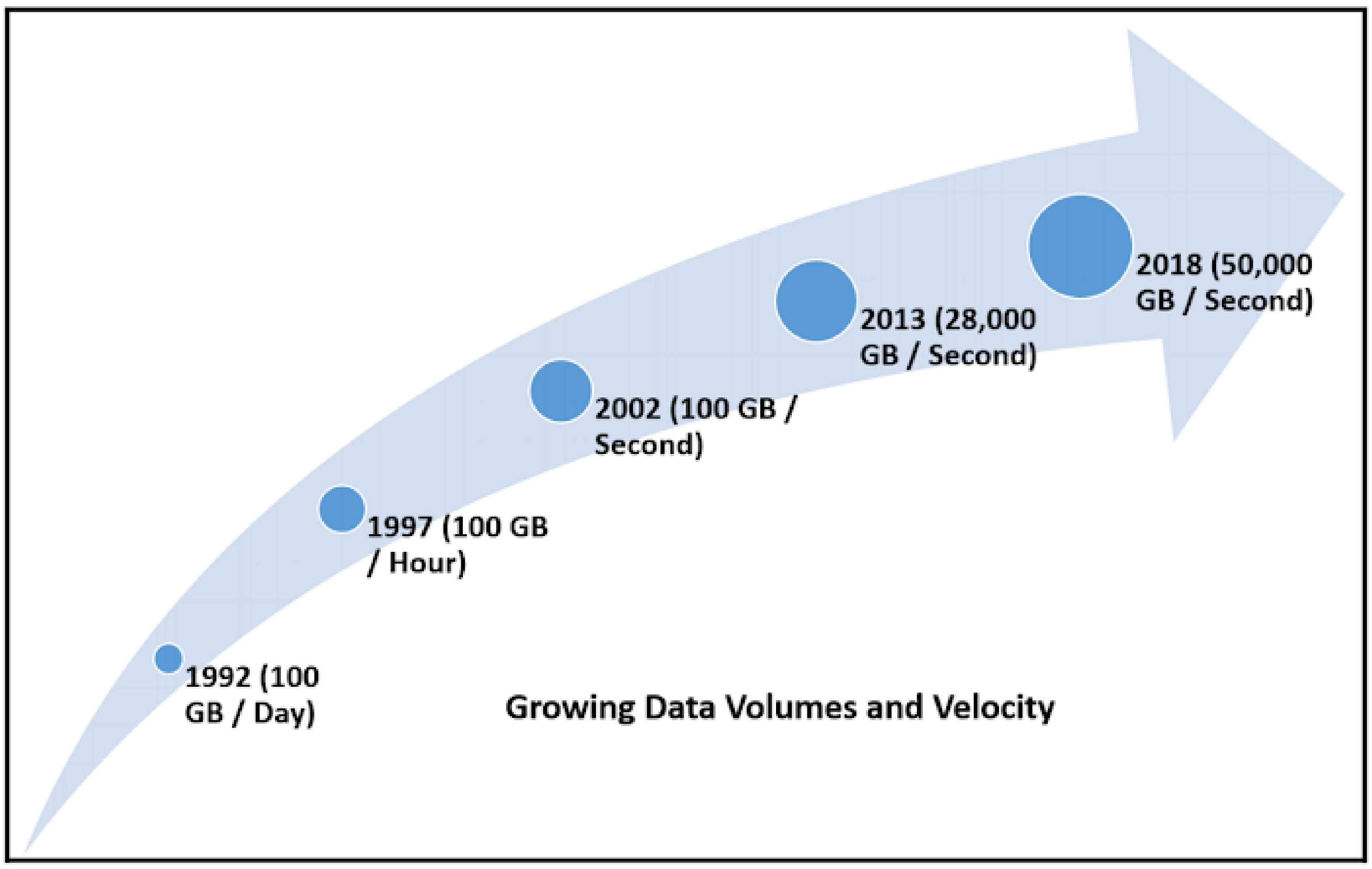
- Peter Norvig, Research Director, Google

Data in dictionary terms is defined as *facts and statistics collected together for reference or analysis*. Storage mechanisms have greatly evolved with human evolution—sculptures, handwritten texts on leaves, punch cards, magnetic tapes, hard drives, floppy disks, CDs, DVDs, SSDs, human DNA, and more. With each new medium, we are able to store more and more data in less space; it's a transition in the right direction. With the advent of the internet and the **Internet of Things** (**IoT**), data volumes have been growing exponentially.

Data volumes are exploding; more data has been created in the past two years than in the entire history of the human race.

The term Big Data was coined to represent growing volumes of data. Along with volume, the term also incorporates three more attributes, velocity, variety, and value, as follows:

- **Volume**: This represents the ever increasing and exponentially growing amount of data. We are now collecting data through more and more interfaces between man-made and natural objects. For example, a patient's routine visit to a clinic now generates electronic data in the tune of megabytes. An average smartphone user generates a data footprint of at least a few GB per day. A flight traveling from one point to another generates half a terabyte of data.
- **Velocity**: This represents the amount of data generated with respect to time and a need to analyze that data in near-real time for some mission critical operations. There are sensors that collect data from natural phenomenon, and the data is then processed to predict hurricanes/earthquakes. Healthcare is a great example of the velocity of the data generation; analysis and action is mission critical:



- **Variety**: This represents variety in data formats. Historically, most electronic datasets were structured and fit into database tables (columns and rows). However, more than 80% of the electronic data we now generate is not in structured format, for example, images, video files, and voice data files. With Big Data, we are in a position to analyze the vast majority of structured/unstructured and semi-structured datasets.

- **Value**: This is the most important aspect of Big Data. The data is only as valuable as its utilization in the generation of actionable insight. Remember the results pyramid where actions lead to results. There is no disagreement that data holds the key to actionable insight; however, systems need to evolve quickly to be able to analyze the data, understand the patterns within the data, and, based on the contextual details, provide solutions that ultimately create value.

Evolution from dumb to intelligent machines

The machines and mechanisms that store and process these huge amounts of data have evolved greatly over a period of time. Let us briefly look at the evolution of machines (for simplicity's sake, computers). For a major portion of their evolution, computers were dumb machines instead of intelligent machines. The basic building blocks of a computer are the **CPU** (**Central Processing Unit**), the RAM (temporary memory), and the disk (persistent storage). One of the core components of a CPU is an **ALU** (**Arithmetic and Logic Unit**). This is the component that is capable of performing the basic steps of mathematical calculations along with logical operations. With these basic capabilities in place, traditional computers evolved with greater and higher processing power. However, they were still dumb machines without any inherent intelligence. These computers were extremely good at following predefined instructions by using brute force and throwing errors or exceptions for scenarios that were not predefined. These computer programs could only answer *specific* questions they were meant to solve.

Although these machines could process lots of data and perform computationally heavy jobs, they would be always limited to what they were programmed to do. This is extremely limiting if we take the example of a self driving car. With a computer program working on predefined instructions, it would be nearly impossible to program the car to handle all situations, and the programming would take forever if we wanted to drive the car on ALL roads and in all situations.

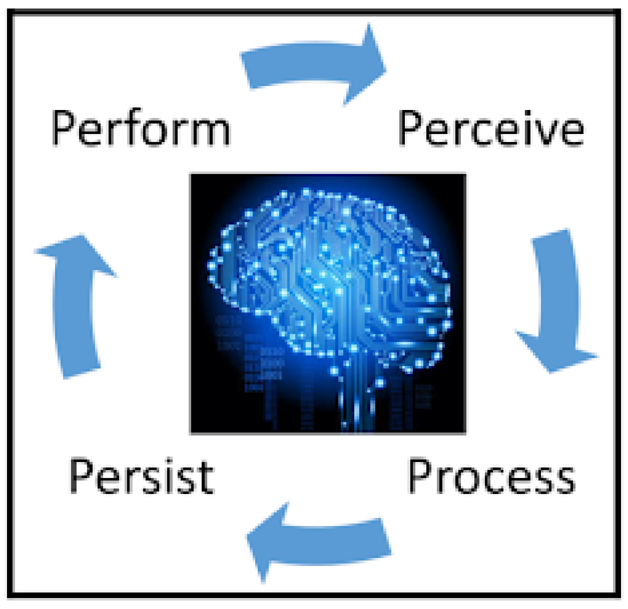
This limitation of traditional computers to respond to unknown or non-programmed situations leads to the question: Can a machine be developed to *think* and evolve as humans do? Remember, when we learn to drive a car, we just drive it in a small amount of situations and on certain roads. Our brain is very quick to learn to react to new situations and trigger various actions (apply breaks, turn, accelerate, and so on). This curiosity resulted in the evolution of traditional computers into artificially intelligent machines.

Traditionally, AI systems have evolved based on the goal of creating *expert systems* that demonstrate intelligent behavior and learn with every interaction and outcome, similar to the human brain.

In the year 1956, the term **artificial intelligence** was coined. Although there were gradual steps and milestones on the way, the last decade of the 20th century marked remarkable advancements in AI techniques. In 1990, there were significant demonstrations of machine learning algorithms supported by case-based reasoning and natural language understanding and translations. Machine intelligence reached a major milestone when then World Chess Champion, Gary Kasparov, was beaten by Deep Blue in 1997. Ever since that remarkable feat, AI systems have greatly evolved to the extent that some experts have predicted that AI will beat humans at *everything* eventually. In this book, we are going to look at the specifics of building intelligent systems and also understand the core techniques and available technologies. Together, we are going to be part of one of the greatest revolutions in human history.

Intelligence

Fundamentally, intelligence in general, and human intelligence in particular, is a constantly evolving phenomenon. It evolves through four Ps when applied to sensory input or data assets: **Perceive**, **Process**, **Persist**, and **Perform**. In order to develop artificial intelligence, we need to also model our machines with the same cyclical approach:



Types of intelligence

Here are some of the broad categories of human intelligence:

- **Linguistic intelligence**: Ability to associate words to objects and use language (vocabulary and grammar) to express meaning
- **Logical intelligence**: Ability to calculate, quantify, and perform mathematical operations and use basic and complex logic for inference
- **Interpersonal and emotional intelligence**: Ability to interact with other human beings and understand feelings and emotions

Intelligence tasks classification

This is how we classify intelligence tasks:

- Basic tasks:
    - Perception
    - Common sense
    - Reasoning
    - Natural language processing
- Intermediate tasks:
    - Mathematics
    - Games
- Expert tasks:
    - Financial analysis
    - Engineering
    - Scientific analysis
    - Medical analysis

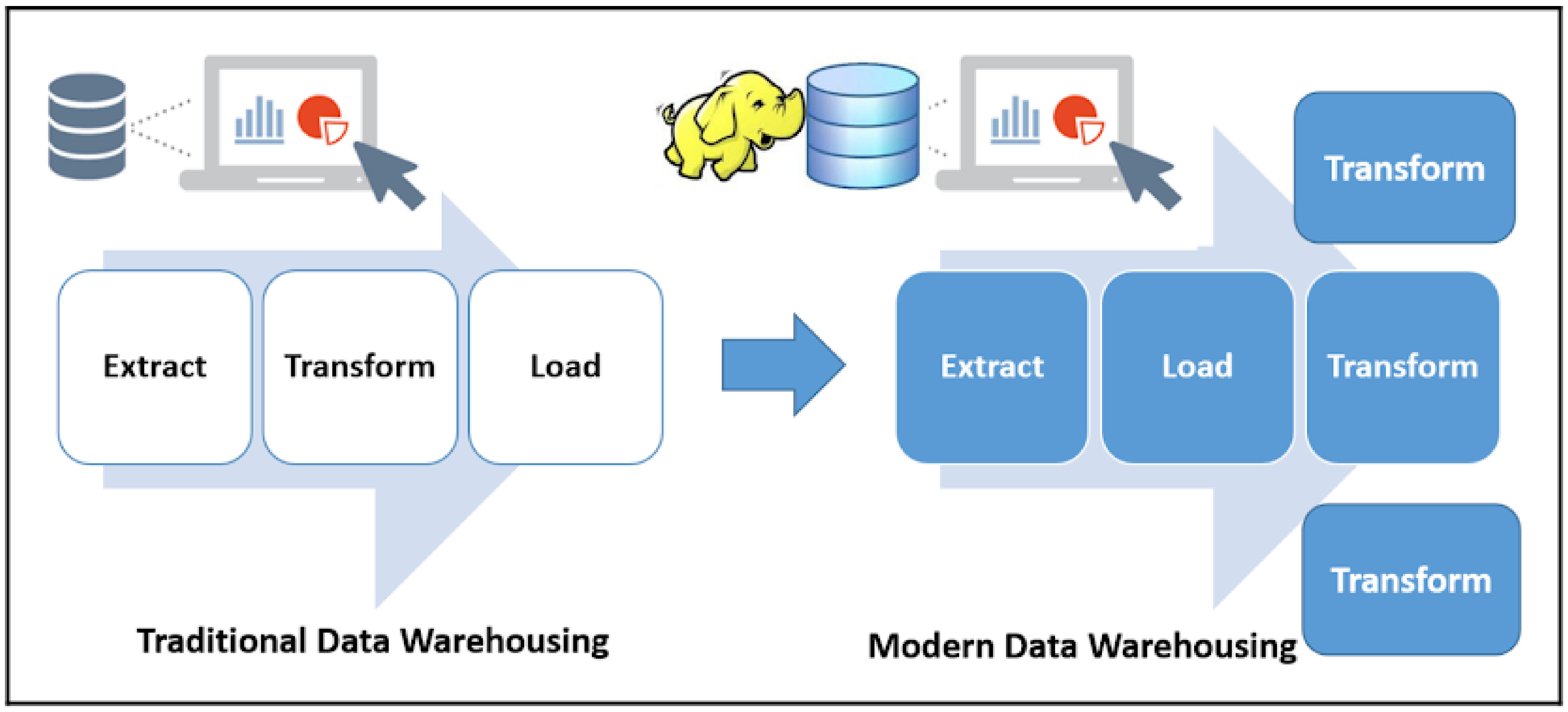
The fundamental difference between human intelligence and machine intelligence is the handling of basic and expert tasks. For human intelligence, basic tasks are easy to master and they are hardwired at birth. However, for machine intelligence, perception, reasoning, and natural language processing are some of the most computationally challenging and complex tasks.

Big data frameworks

In order to derive **value** from data that is high in **volume**, **varies** in its form and structure, and is generated with ever increasing **velocity**, there are two primary categories of framework that have emerged over a period of time. These are based on the consideration of the differential time at which the event occurs (data origin) and the time at which the data is available for analysis and action.

Batch processing

Traditionally, the data processing pipeline within data warehousing systems consisted of **Extracting**, **Transforming**, and **Loading** the data for analysis and actions (**ETL**). With the new paradigm of file-based distributed computing, there has been a shift in the ETL process sequence. Now the data is **Extracted**, **Loaded**, and **Transformed** repetitively for analysis (**ELTTT**) a number of times:



In batch processing, the data is collected from various sources in the staging areas and loaded and transformed with defined frequencies and schedules. In most use cases with batch processing, there is no critical need to process the data in real time or in near real time. As an example, the monthly report on a student's attendance data will be generated by a process (batch) at the end of a calendar month. This process will extract the data from source systems, load it, and transform it for various views and reports. One of the most popular batch processing frameworks is **Apache Hadoop**. It is a highly scalable, distributed/parallel processing framework. The primary building block of Hadoop is the **Hadoop Distributed File System**.

As the name suggests, this is a wrapper filesystem which stores the data (structured/unstructured/semi-structured) in a distributed manner on data nodes within Hadoop. The processing that is applied on the data (instead of the data that is processed) is sent to the data on various nodes. Once the compute is performed by an individual node, the results are consolidated by the master process. In this paradigm of data-compute localization, Hadoop relies heavily on intermediate I/O operations on hard drive disks. As a result, extremely large volumes of data can be processed by Hadoop in a reliable manner at the cost of processing time. This framework is very suitable for extracting value from Big Data in batch mode.

Real-time processing

While batch processing frameworks are good for most data warehousing use cases, there is a critical need for processing the data and generating actionable insight as soon as the data is available. For example, in a credit card fraud detection system, the alert should be generated as soon as the first instance of logged malicious activity. There is no value if the actionable insight (denying the transaction) is available as a result of the end-of-month batch process. The idea of a real-time processing framework is to reduce latency between **event time** and **processing time**. In an ideal system, the expectation would be zero differential between the event time and the processing time. However, the time difference is a function of the data source input, execution engine, network bandwidth, and hardware. Real-time processing frameworks achieve low latency with minimal I/O by relying on in-memory computing in a distributed manner. Some of the most popular real-time processing frameworks are:

- **Apache Spark**: This is a distributed execution engine that relies on in-memory processing based on fault tolerant data abstractions named **RDDs** (**Resilient Distributed Datasets**).
- **Apache Storm**: This is a framework for distributed real-time computation. Storm applications are designed to easily process unbounded streams, which generate event data at a very high velocity.
- **Apache Flink**: This is a framework for efficient, distributed, high volume data processing. The key feature of Flink is automatic program optimization. Flink provides native support for massively iterative, compute intensive algorithms.

As the ecosystem is evolving, there are many more frameworks available for batch and real-time processing. Going back to the machine intelligence evolution cycle (Perceive, Process, Persist, Perform), we are going to leverage these frameworks to create programs that work on Big Data, take an algorithmic approach to filter relevant data, generate models based on the patterns within the data, and derive actionable insight and predictions that ultimately lead to **value** from the data assets.

Intelligent applications with Big Data

At this juncture of technological evolution, where we have the availability of systems that gather large volumes of data from heterogeneous sources, along with systems that store these large volumes of data at ever reducing costs, we can derive value in the form of insight into the data and build intelligent machines that can trigger actions resulting in the betterment of human life. We need to use an algorithmic approach with the massive data and compute assets we have at our disposal. Leveraging a combination of human intelligence, large volumes of data, and distributed computing power, we can create expert systems which can be used as an advantage to lead the human race to a better future.

Areas of AI

While we are in the infancy of developments in AI, here are some of the basic areas in which significant research and breakthroughs are happening:

- **Natural language processing**: Facilitates interactions between computers and human languages.
- **Fuzzy logic systems**: These are based on the degrees of truth instead of programming for all situations with IF/ELSE logic. These systems can control machines and consumer products based on acceptable reasoning.
- **Intelligent robotics**: These are mechanical devices that can perform mundane or hazardous repetitive tasks.
- **Expert systems**: These are systems or applications that solve complex problems in a specific domain. They are capable of advising, diagnosing, and predicting results based on the knowledge base and models.

Frequently asked questions

Here is a small recap of what we covered in the chapter:

Q: What is a results pyramid?

A: The results we get (man or machine) are an outcome of our experiences (data), beliefs (models), and actions. If we need to change the results, we need different (better) sets of data, models, and actions.

Q: How is this paradigm applicable to AI and Big Data?

A: In order to improve our lives, we need intelligent systems. With the advent of Big Data, there has been a boost to the theory of machine learning and AI due to the availability of huge volumes of data and increasing processing power. We are on the verge of getting better results for humanity as a result of the convergence of machine intelligence and Big Data.

Q: What are the basic categories of Big Data frameworks?

A: Based on the differentials between the event time and processing time, there are two types of framework: batch processing and real-time processing.

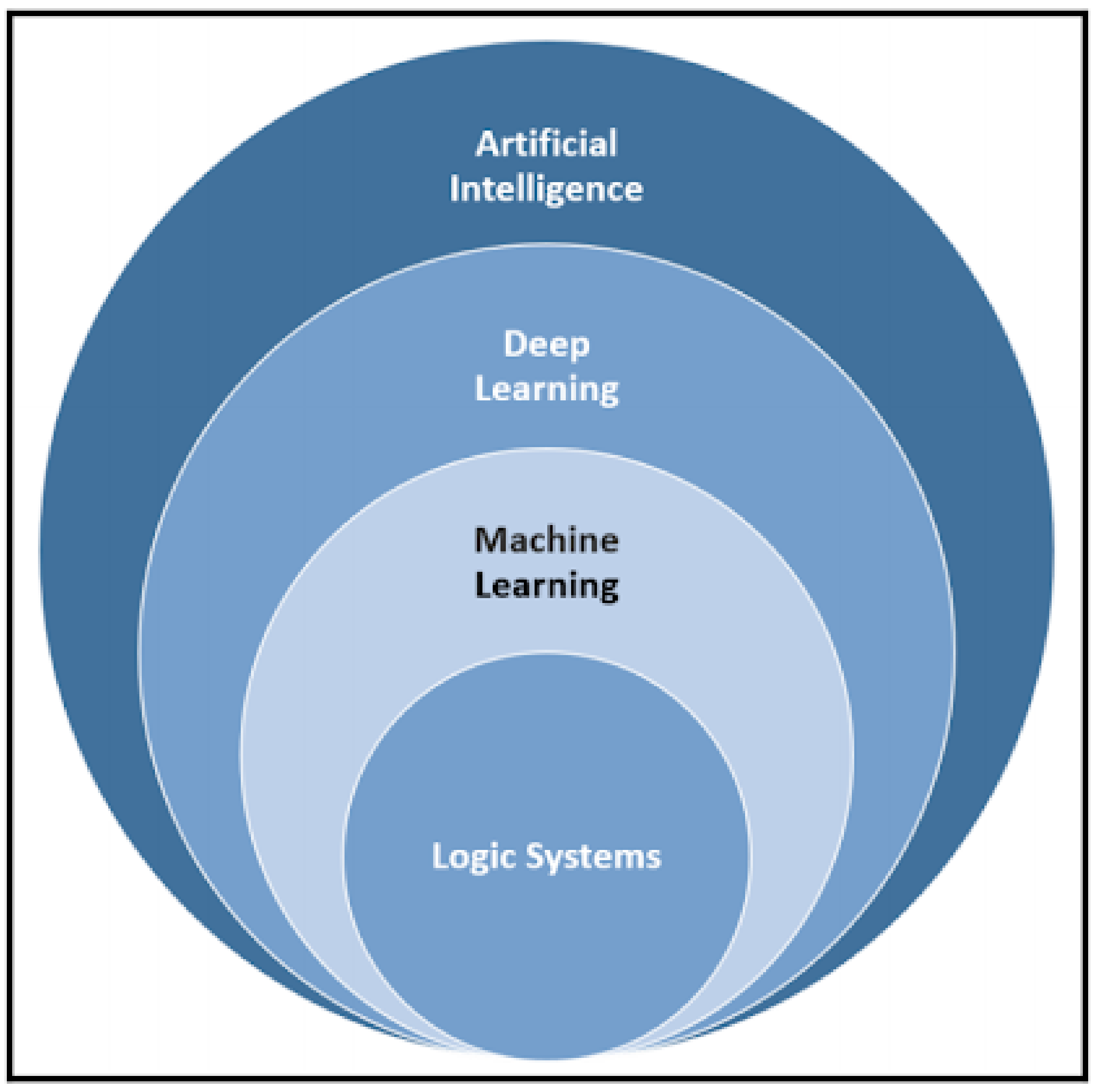
Q: What is the goal of AI?

A: The fundamental goal of AI is to augment and complement human life.

Q: What is the difference between machine learning and AI?

A: Machine learning is a core concept which is integral to AI. In machine learning, the conceptual models are trained based on data and the models can predict outcomes for the new datasets. AI systems try to emulate human cognitive abilities and are context sensitive. Depending on the context, AI systems can change their behaviors and outcomes to best suit the decisions and actions the human brain would take.

Have a look at the following diagram for a better understanding:



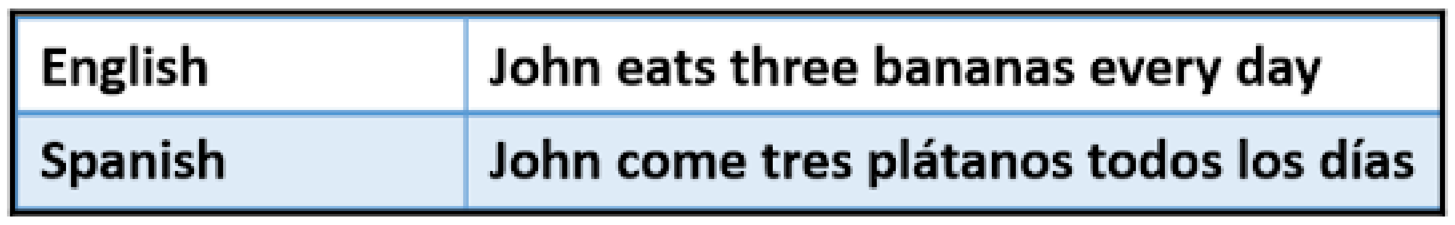
Summary

In this chapter, we understood the concept of the results pyramid, which is a model for the continuous improvement of human life and striving to get better results with an improved understanding of the world based on data (experiences), which shape our models (beliefs). With the convergence of the evolving human brain and computers, we know that the best of both worlds can really improve our lives. We have seen how computers have evolved from dumb to intelligent machines and we provided a high-level overview of intelligence and Big Data, along with types of processing frameworks.

With this introduction and context, in subsequent chapters in this book, we are going to take a deep dive into the core concepts of taking an algorithmic approach to data and the basics of machine learning with illustrative algorithms. We will implement these algorithms with available frameworks and illustrate this with code samples.

2
Ontology for Big Data

In the introductory chapter, we learned that big data has fueled rapid advances in the field of artificial intelligence. This is primarily because of the availability of extremely large datasets from heterogeneous sources and exponential growth in processing power due to distributed computing. It is extremely difficult to derive value from large data volumes if there is no standardization or a common language for interpreting data into information and converting information into knowledge. For example, two people who speak two different languages, and do not understand each other's languages, cannot get into a verbal conversation unless there is some translation mechanism in between. Translations and interpretations are possible only when there is a semantic meaning associated with a keyword and when grammatical rules are applied as conjunctions. As an example, here is a sentence in the **English** and **Spanish** languages:



English	**John eats three bananas every day**
Spanish	**John come tres plátanos todos los días**

Broadly, we can break a sentence down in the form of objects, subjects, verbs, and attributes. In this case, **John** and **bananas** are subjects. They are connected by an activity, in this case eating, and there are also attributes and contextual data—information in conjunction with the subjects and activities. Knowledge translators can be implemented in two ways:

- **All-inclusive mapping**: Maintaining a mapping between *all* sentences in one language and translations in the other language. As you can imagine, this is impossible to achieve since there are countless ways something (object, event, attributes, context) can be expressed in a language.
- **Semantic view of the world**: If we associate semantic meaning with every entity that we encounter in linguistic expression, a standardized semantic view of the world can act as a centralized dictionary for all the languages.

A semantic and standardized view of the world is essential if we want to implement artificial intelligence which fundamentally derives knowledge from data and utilizes the contextual knowledge for insight and meaningful actions in order to augment human capabilities. This semantic view of the world is expressed as **Ontologies**. In the context of this book, Ontology is defined as: a set of concepts and categories in a subject area or domain, showing their properties and the relationships between them.

In this chapter, we are going to look at the following:

- How the human brain links objects in its interpretation of the world
- The role Ontology plays in the world of Big Data
- Goals and challenges with Ontology in Big Data
- The Resource Description Framework
- The Web Ontology Language
- SPARQL, the semantic query language for the RDF
- Building Ontologies and using Ontologies to build intelligent machines
- Ontology learning

Human brain and Ontology

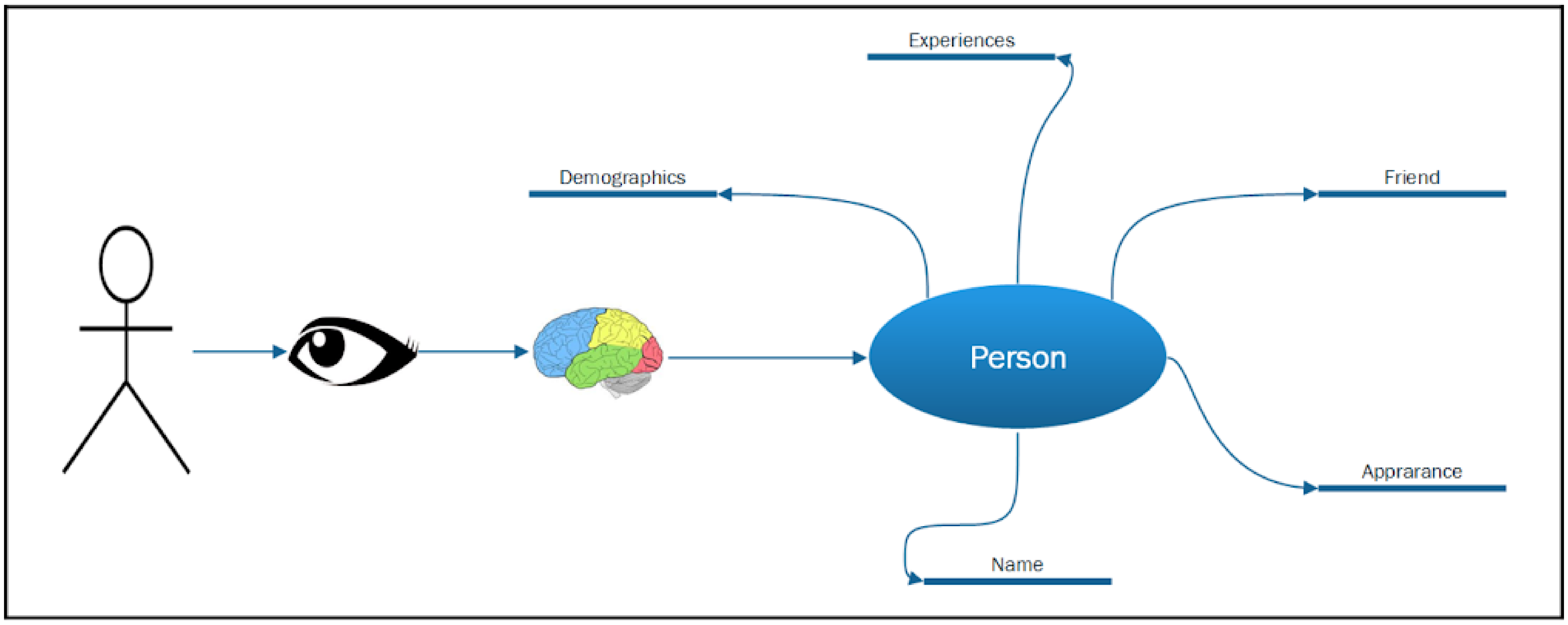
While there are advances in our understanding of how the human brain functions, the storage and processing mechanism of the brain is far from fully understood. We receive hundreds and thousands of sensory inputs throughout a day, and if we process and store every bit of this information, the human brain will be overwhelmed and will be unable to understand the context and respond in a meaningful way. The human brain applies filters to the sensory input it receives continuously. It is understood that there are three compartments to human memory:

- **Sensory memory**: This is the first-level memory, and the majority of the information is flushed within milliseconds. Consider, for example, when we are driving a car. We encounter thousands of objects and sounds on the way, and most of this input is utilized for the function of driving. Beyond the frame of reference in time, most of the input is forgotten and never stored in memory.

- **Short-term memory**: This is used for the information that is essential for serving a temporary purpose. Consider, for example, that you receive a call from your co-worker to remind you about an urgent meeting in room number D-1482. When you start walking from your desk to the room, the number is significant and the human brain keeps the information in short-term memory. This information may or may not be stored beyond the context time. These memories can potentially convert to long-term memory if encountered within an extreme situation.
- **Long-term memory**: This is the memory that will last for days or a lifetime. For example, we remember our name, date of birth, relatives, home location, and so many other things. The long-term memory functions on the basis of patterns and links between objects. The non-survival skills we learn and master over a period of time, for example playing a musical instrument, require the storage of connecting patterns and the coordination of reflexes within long-term memory.

Irrespective of the memory compartment, the information is stored in the form of patterns and links within the human brain. In a memory game that requires players to momentarily look at a group of 50-odd objects for a minute and write down the names on paper, the player who writes the most object names wins the game. One of the tricks of playing this game is to establish links between two objects and form a storyline. The players who try to independently memorize the objects cannot win against the players who create a linked list in their mind.

When the brain receives input from sensory organs and the information needs to be stored in the long-term memory, it is stored in the form of patterns and links to related objects or entities, resulting in mind maps. This is shown in the following figure:



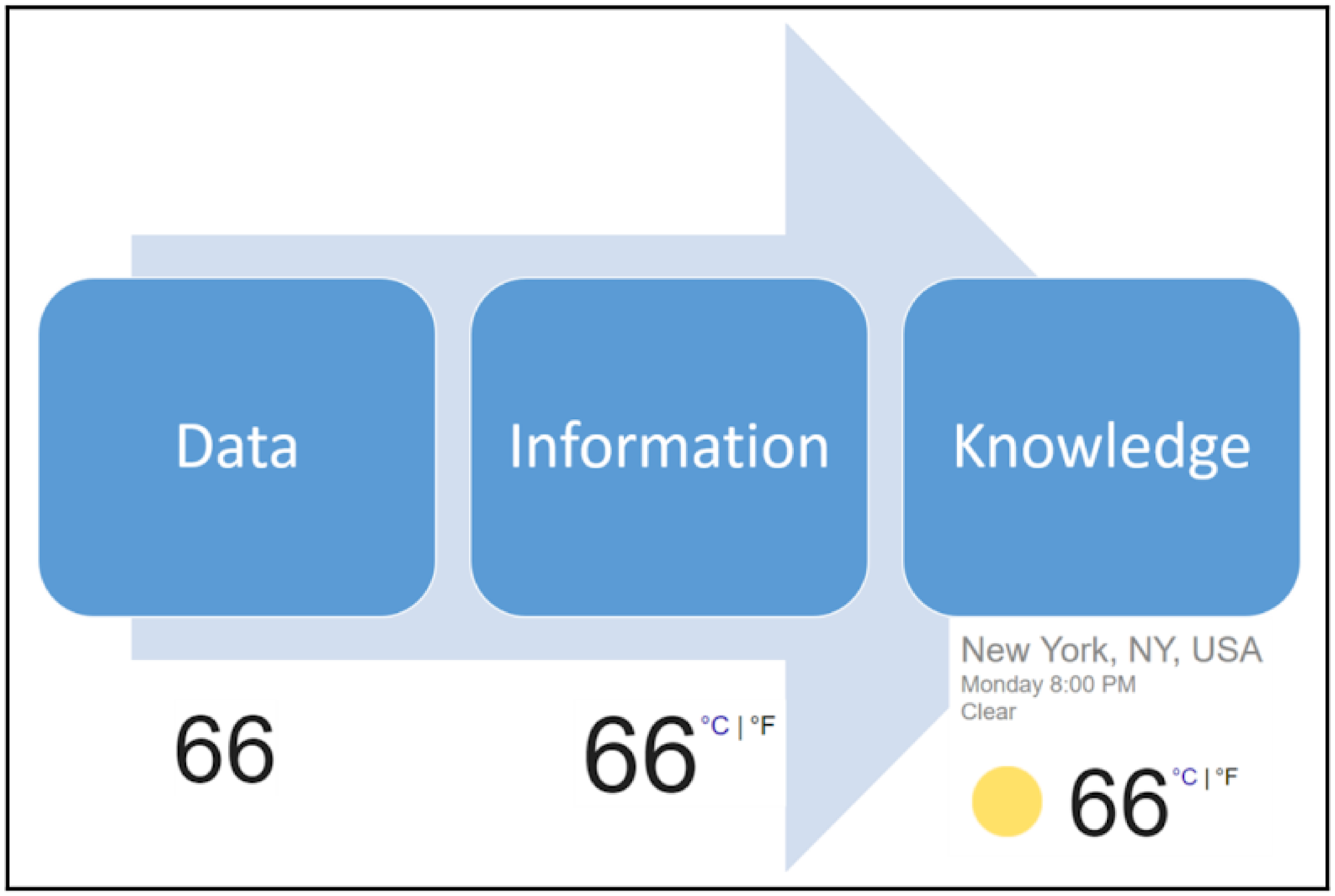
When we see a person with our eyes, the brain creates a map for the image and retrieves all the context-based information related to the person.

This forms the basis of the Ontology of information science.

Ontology of information science

Formally, the Ontology of information sciences is defined as: *A formal naming and definition of types, properties, and interrelationships of the entities that fundamentally exist for a particular domain*.

There is a fundamental difference between people and computers when it comes to dealing with information. For computers, information is available in the form of **strings** whereas for humans, the information is available in the form of **things**. Let's understand the difference between strings and things. When we add metadata to a string, it becomes a thing. Metadata is data about data (the string in this case) or contextual information about data. The idea is to convert the data into knowledge. The following illustration gives us a good idea about how to convert data into knowledge:



The text or the number **66** is **Data**; in itself, **66** does not convey any meaning. When we say **66^0 F**, 66 becomes a measure of temperature and at this point it represents some **Information**. When we say **66^0 F** in **New York** on 3rd October 2017 at **8:00 PM**, it becomes **Knowledge**. When contextual information is added to **Data** and **Information**, it becomes **Knowledge**.

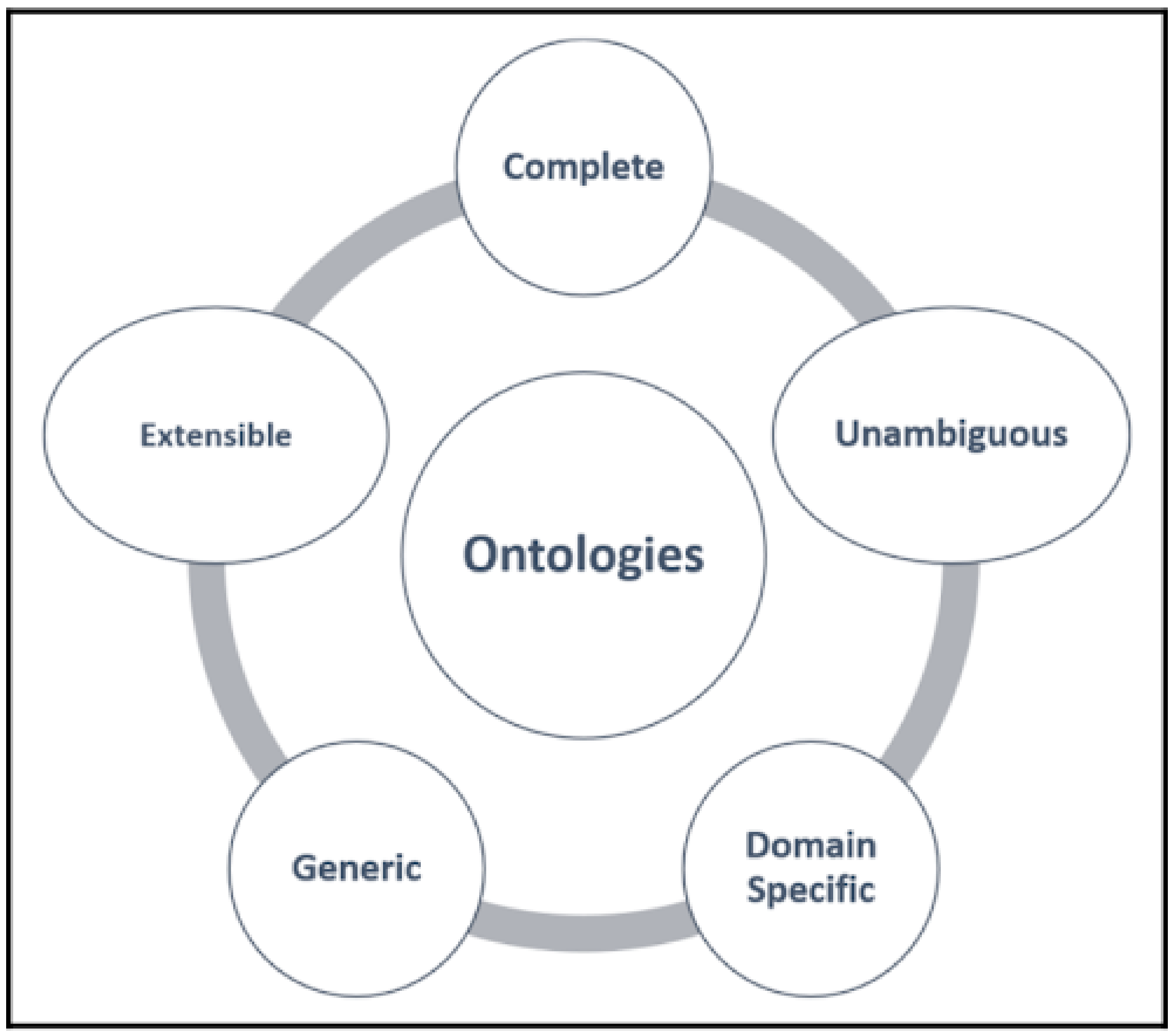
In the quest to derive knowledge from data and information, Ontologies play a major role in standardizing the worldview by precisely defined terms that can be communicated between people and software applications. They create a shared understanding of objects and their relationships within and across domains. Typically, there are schematic, structural, and semantic differences, and hence conflict arises between knowledge representations. Well-defined and governed Ontologies bridge the gaps between the representations.

Ontology properties

At a high level, Ontologies should have the following properties to create a consistent view of the universe of data, information, and knowledge assets:

- The Ontologies should be complete so that all aspects of the entities are covered.
- The Ontologies should be unambiguous in order to avoid misinterpretation by people and software applications.
- The Ontologies should be consistent with the domain knowledge to which they are applicable. For example, Ontologies for medical science should adhere to the formally established terminologies and relationships in medical science.
- The Ontologies should be generic in order to be reused in different contexts.
- The Ontologies should be extensible in order to add new concepts and facilitate adherence to the new concepts, that emerge with growing knowledge in the domain.
- The Ontologies should be machine-readable and interoperable.

Here is an illustration to better explain properties of Ontologies:



The most important advantage of Ontological representation for real-world concepts and entities is that it facilitates the study of concepts independently of programming language, platforms, and communication protocols. This enables loose coupling, and at the same time, tight integration between the concepts, which enables the software development process to reuse the software and knowledge base as modular concepts.

Advantages of Ontologies

The following are the advantages of Ontologies:

- Increased quality of entity analysis
- Increased use, reuse, and maintainability of the information systems
- Facilitation of domain knowledge sharing, with common vocabulary across independent software applications

Those who are familiar with the object-oriented programming paradigm or database design can easily relate the Ontological representation of the domain entities to classes or database schemas. The classes are generic representations of the entities that encapsulate properties and behaviors. One class can inherit behavior and properties from another class (*is-a* relationship). For example, a cat is an animal.

In this case, Animal is an abstract superclass of Cat. The Cat class inherits properties from the Animal class and adds/overrides some of the attributes and behaviors specific to a cat. This paradigm is applicable in Ontologies. Similarly, relational databases have schematic representations of the domain entities within an organization.

There are some fundamental differences between databases and Ontologies, as follows:

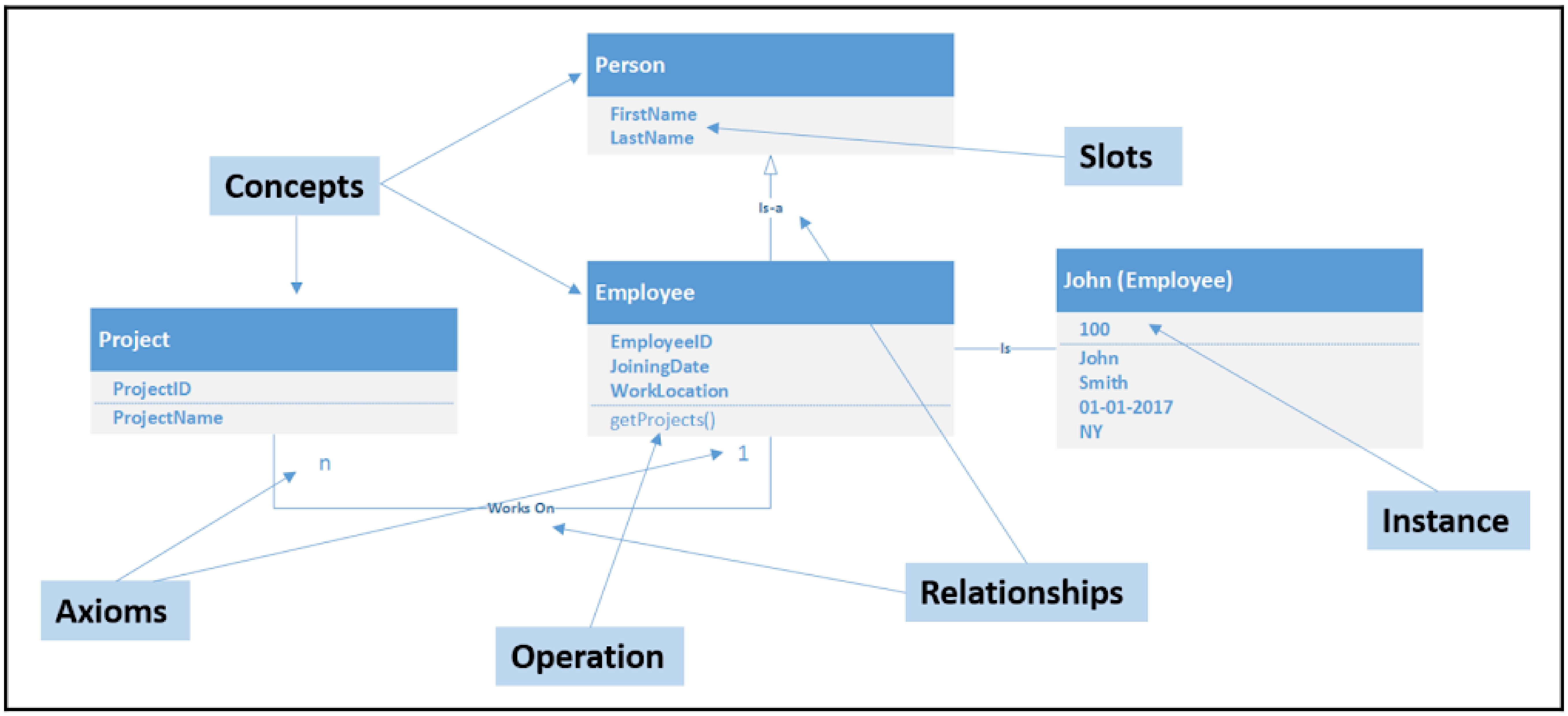
- Ontologies are semantically richer than the concepts represented by databases
- Information representation in an Ontology is based on semi-structured, natural language text and it is not represented in a tabular format
- The basic premise of Ontological representation is globally consistent terminology to be used for information exchange across domains and organizational boundaries
- More than defining a confined data container, Ontologies focus on generic domain knowledge representation

Components of Ontologies

The following are the components of Ontologies:

- **Concepts**: These are the general things or entities similar to classes in object-oriented programming, for example, a person, an employee, and so on.
- **Slots**: These are the properties or attributes of the entities, for example, gender, date of birth, location, and so on.
- **Relationships**: These represent interactions between concepts, or *is-a*, *has-a* relationships, for example, an employee is a person.
- **Axioms**: These are statements which are *always* true in regards to concepts, slots and relationships, for example, a person is an employee if he is employed by an employer.
- **Instances**: These are the objects of a class in object-oriented terms. For example, John is an instance of the Employee class. It is a specific representation of a concept. Ontology, along with instances, fully represents knowledge.
- **Operations**: These are the functions and rules that govern the various components of the Ontologies. In an object-oriented context, these represent methods of a class.

The following diagram explains the components of Ontologies:



The development of Ontologies begins with defining classes in the Ontology. These classes represent real-world entities. Once the entities are clearly identified and defined, they are arranged in a taxonomic hierarchy. Once the hierarchy is defined, the Slots and Relationships are defined. Filling in the values for slots and instances completes the development of a domain-specific Ontology.

The role Ontology plays in Big Data

As we saw in the introductory chapter, data volumes are growing at a phenomenal rate and in order to derive value from the data, it is impossible to model the entire data in a traditional **Extract**, **Transform**, and **Load** (**ETL**) way. Traditionally, data sources generate the datasets in structured and unstructured formats. In order to store these data assets, we need to manually model the data based on various entities. Taking an example of Person as an entity in the relational database world, we need to create a table that represents Person. This table is linked to various entities with foreign key relationships. However, these entities are predefined and have a fixed structure. There is manual effort involved in modeling the entities and it is difficult to modify them.

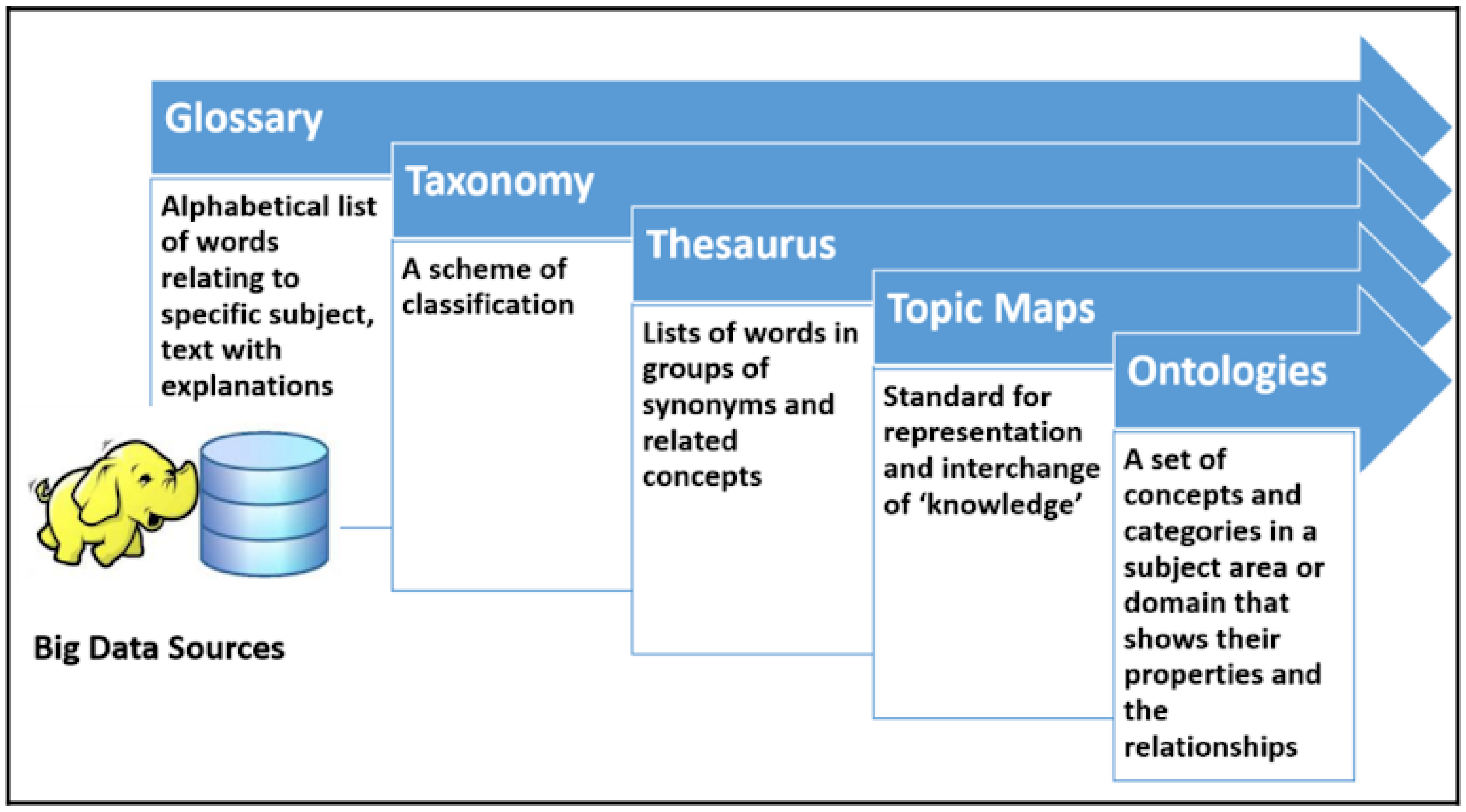
In the big data world, the schema is defined at read time instead of write time. This gives us a higher degree of flexibility with the entity structure and data modeling. Even with flexibility and extensible modeling capabilities, it is very difficult to manage the data assets on an internet scale if the entities are not standardized across domains.

In order to facilitate web search, Google introduced the **knowledge graph** which changed the search from keyword statistics based on representation to knowledge modeling.

This was the introduction of the searching by things and not strings paradigm. The knowledge graph is a very large Ontology which formally describes objects in the real world. With increased data assets generated from heterogeneous sources at an accelerating pace, we are constantly headed towards increased complexity. The big data paradigm describes large and complex datasets that are not manageable with traditional applications. At a minimum, we need a way to avoid false interpretations of complex data entities. The data integration and processing frameworks can possibly be improved with methods from the field of semantic technology. With use of *things* instead of text, we can improve information systems and their interoperability by identifying the context in which they exist. Ontologies provide the semantic richness of domain-specific knowledge and its representation.

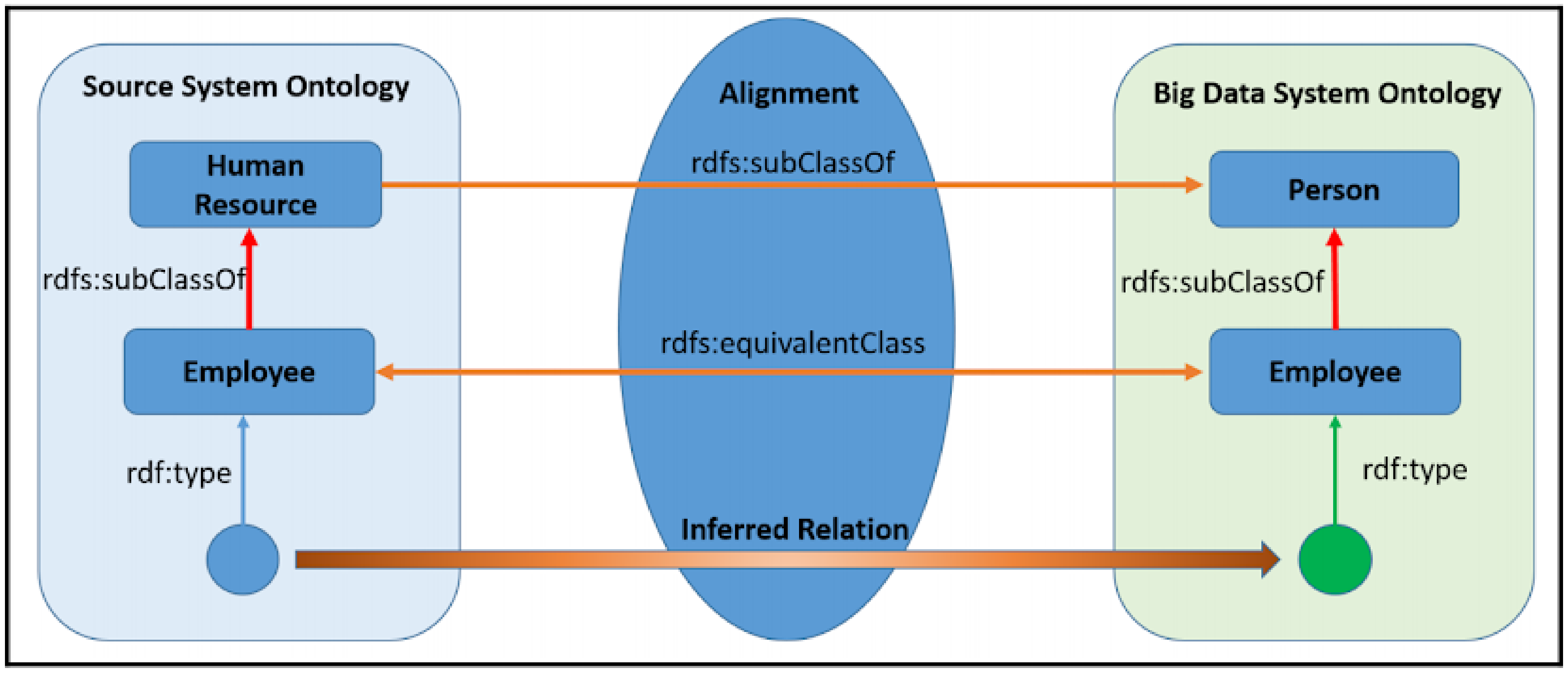
With big data assets, it is imperative that we reduce the manual effort of modeling the data into information and knowledge. This is possible if we can create a means to find the correspondence between raw entities, derive the generic schema with taxonomical representation, and map the concepts to topics in specific knowledge domains with terminological similarities and structural mappings. This implementation will facilitate automatic support for the management of big data assets and the integration of different data sources, resulting in fewer errors and speed of knowledge derivation.

We need an automated progression from **Glossary** to **Ontologies** in the following manner:


Glossary
Alphabetical list of words relating to specific subject, text with explanations
Taxonomy
A scheme of classification
Thesaurus
Lists of words in groups of synonyms and related concepts
Topic Maps
Standard for representation and interchange of 'knowledge'
Ontologies
A set of concepts and categories in a subject area or domain that shows their properties and the relationships
Big Data Sources

Ontology alignment

Ontology alignment or matching is a process of determining one-to-one mapping between entities from heterogeneous sources. Using this mapping, we can infer the entity types and derive meaning from the raw data sources in a consistent and semantic manner:



Goals of Ontology in big data

The following are the goals of Ontology in big data:

- Share a common understanding of information structures across software applications
- Make ETL faster, easier, and more accurate
- Eliminate the need for customized, situation-specific ETL pipelines
- The automatic incorporation of new data sources
- Enhance information extraction from text and convert it into knowledge assets
- Enrich existing data with structural and semantic information
- Translate business knowledge into machine-usable software
- Build once, use many times

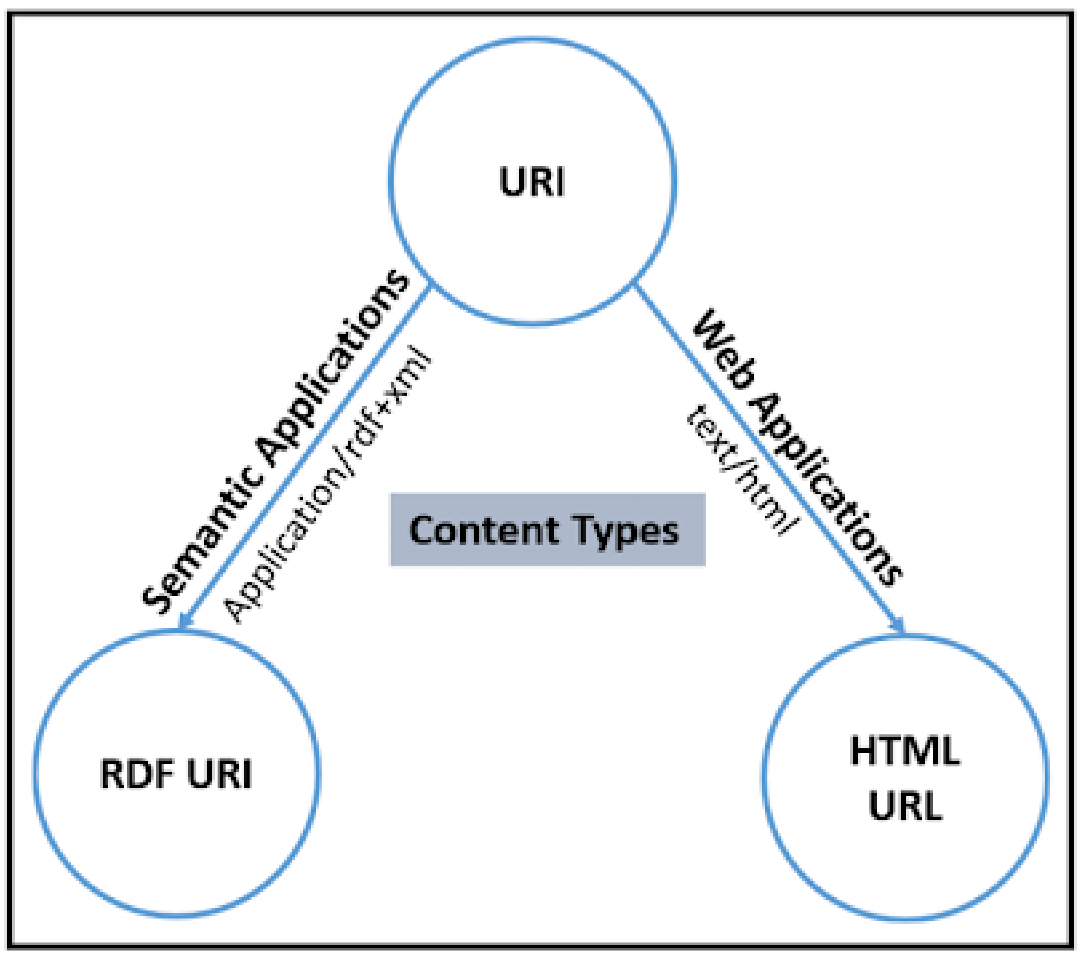
Challenges with Ontology in Big Data

We face the following challenges when using Ontology in big data:

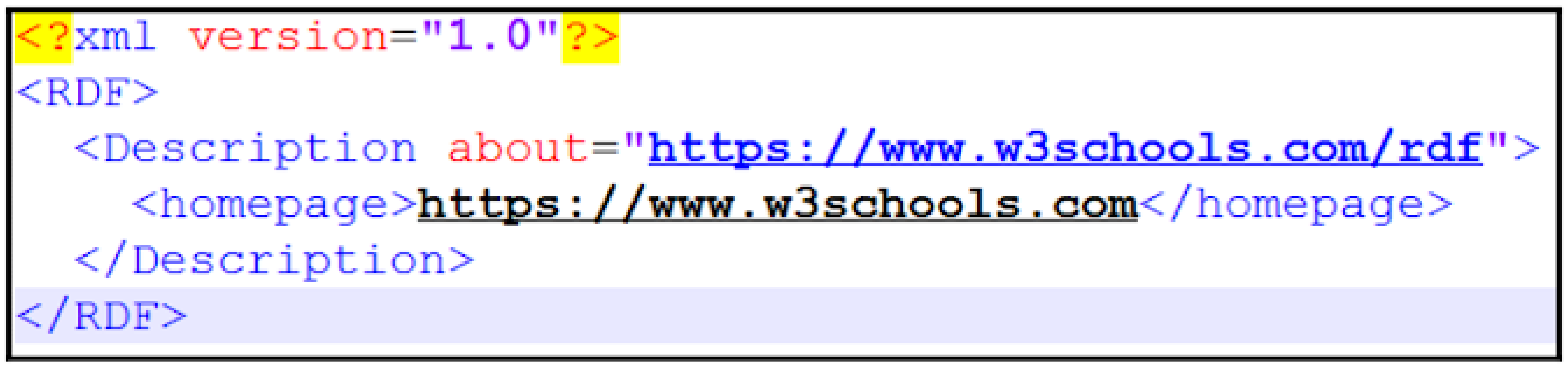
- Generating entities (converting strings to things)
- Managing relationships
- Handling context
- Query efficiency
- Data quality

RDF—the universal data format

With the background of Ontologies and their significance in the big data world, let us look at a universal data format that defines the schematic representations of the Ontologies. One of the most adopted and popular frameworks is the **Resource Description Framework** (**RDF**). RDF has been a W3C recommendation since 2004. RDF provides a structure for describing identified things, entities, or concepts designed to be read and interpreted by computers. There is a critical need to uniquely identify an entity or concept universally. One of the most popular ways in the information science field is the use of **Universal Resource Identifiers** (**URIs**). We are familiar with website addresses, which are represented as **Universal Resource Locators** (**URLs**). These map to a unique IP address and hence a web domain on the internet. A URI is very similar to a URL, with the difference that the URIs may or may not represent an actual web domain. Given this distinction, the URIs that represent the real-world objects must be unambiguous. Any URI should be exclusive to either a web resource or a real-world object and should never be used to represent both at the same time, in order to avoid confusion and ambiguity:



Here is a basic example that describes the `https://www.w3schools.com/rdf` resource:

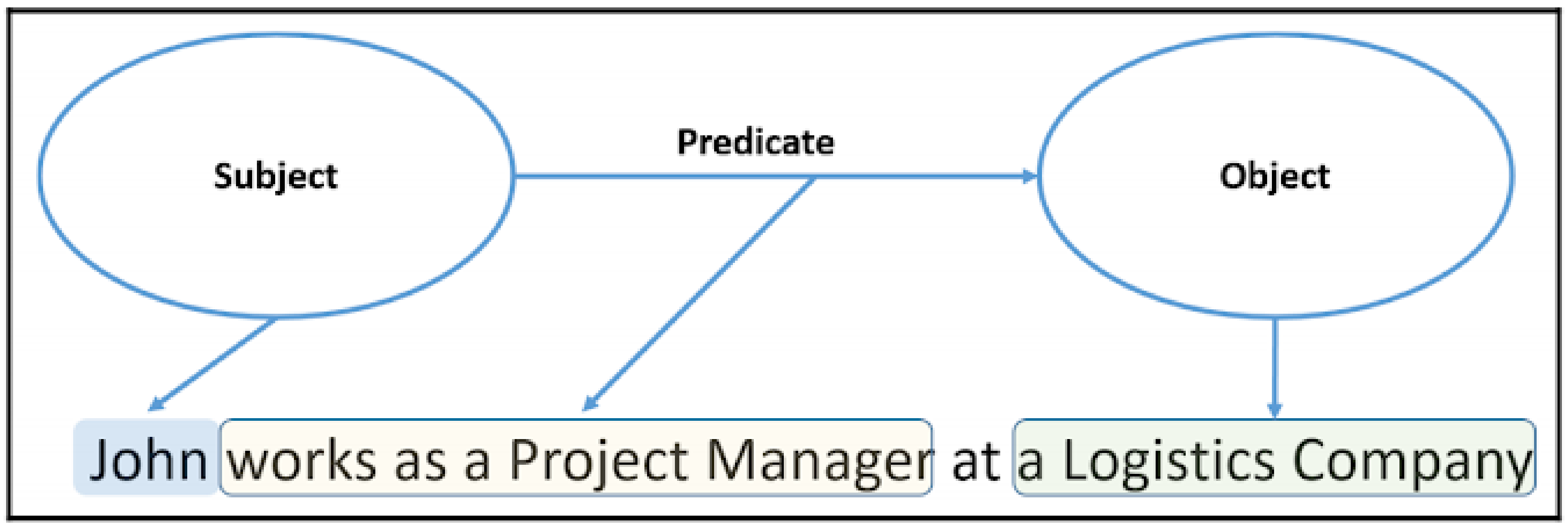


```
<?xml version="1.0"?>
<RDF>
  <Description about="https://www.w3schools.com/rdf">
    <homepage>https://www.w3schools.com</homepage>
  </Description>
</RDF>
```

When defining RDFs, there are the following considerations:

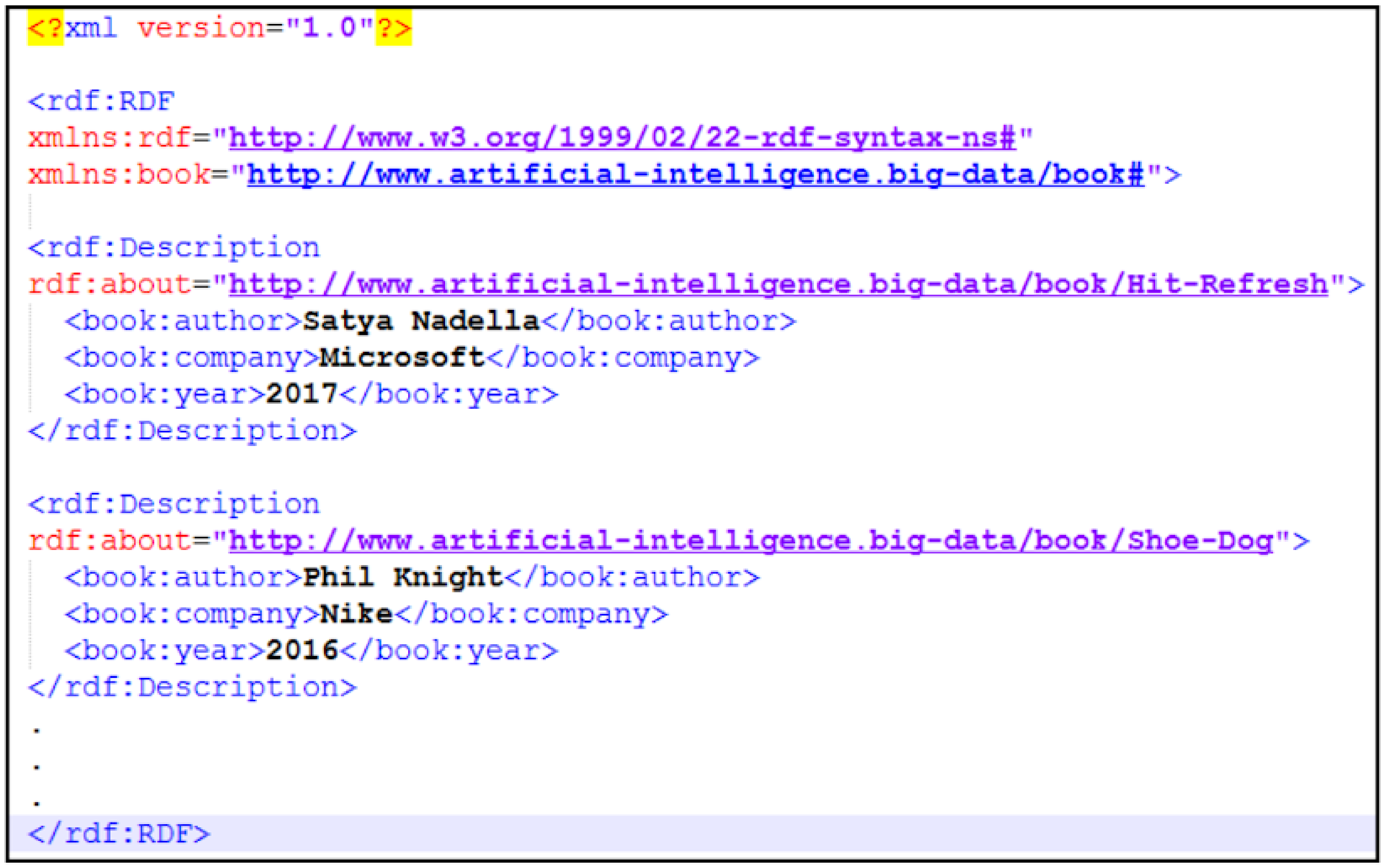
- Define a simple data model
- Define formal semantics
- Use extensible URI-based vocabulary
- Preferably use an XML-based syntax

The basic building block of the RDF is a triple that consists of a **Subject**, **Predicate**, and an **Object**. The set of triples constitutes an RDF graph:



Let us look at an example of a database of books and represent it with RDF XML:

Book Name	Author	Company	Year
Hit Refresh	Satya Nadella	Microsoft	2017
Shoe Dog	Phil Knight	Nike	2016



```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:book="http://www.artificial-intelligence.big-data/book#">

<rdf:Description
rdf:about="http://www.artificial-intelligence.big-data/book/Hit-Refresh">
  <book:author>Satya Nadella</book:author>
  <book:company>Microsoft</book:company>
  <book:year>2017</book:year>
</rdf:Description>

<rdf:Description
rdf:about="http://www.artificial-intelligence.big-data/book/Shoe-Dog">
  <book:author>Phil Knight</book:author>
  <book:company>Nike</book:company>
  <book:year>2016</book:year>
</rdf:Description>
.
.
.
</rdf:RDF>
```

The first line of the RDF document is the XML declaration. The XML declaration is followed by the root element of the RDF documents, `<rdf:RDF>`.

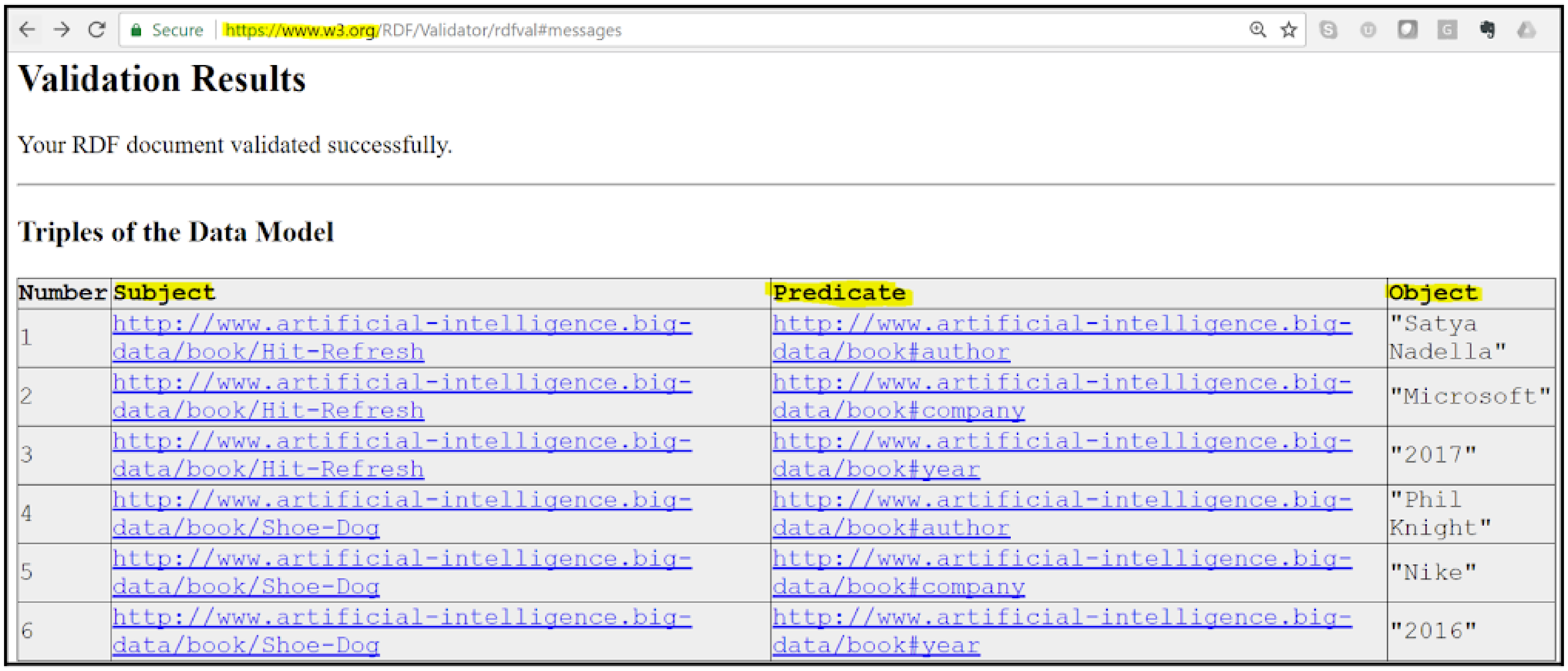
The `xmlns:rdf` namespace specifies that the elements with the `rdf` prefix are from the `http://www.w3.org/1999/02/22-rdf-syntax-ns#` namespace. The XML namespaces are used to provide uniquely named elements and attributes in an XML document.

The `xmlns:book` namespace specifies that the elements with the `book` prefix are from the – `http://www.artificial-intelligence.big-data/book#` namespace.

The `<rdf:Description>` element contains the description of the resource identified by the `rdf:about` attribute.

The elements `<book:author>`, `<book:company>`, `<book:year>`, and so on are properties of the resource.

W3C provides an online validator service (`https://www.w3.org/RDF/Validator/`), which validates the RDF in terms of its syntax and generates tabular and graphical views of the RDF document:



Validation Results

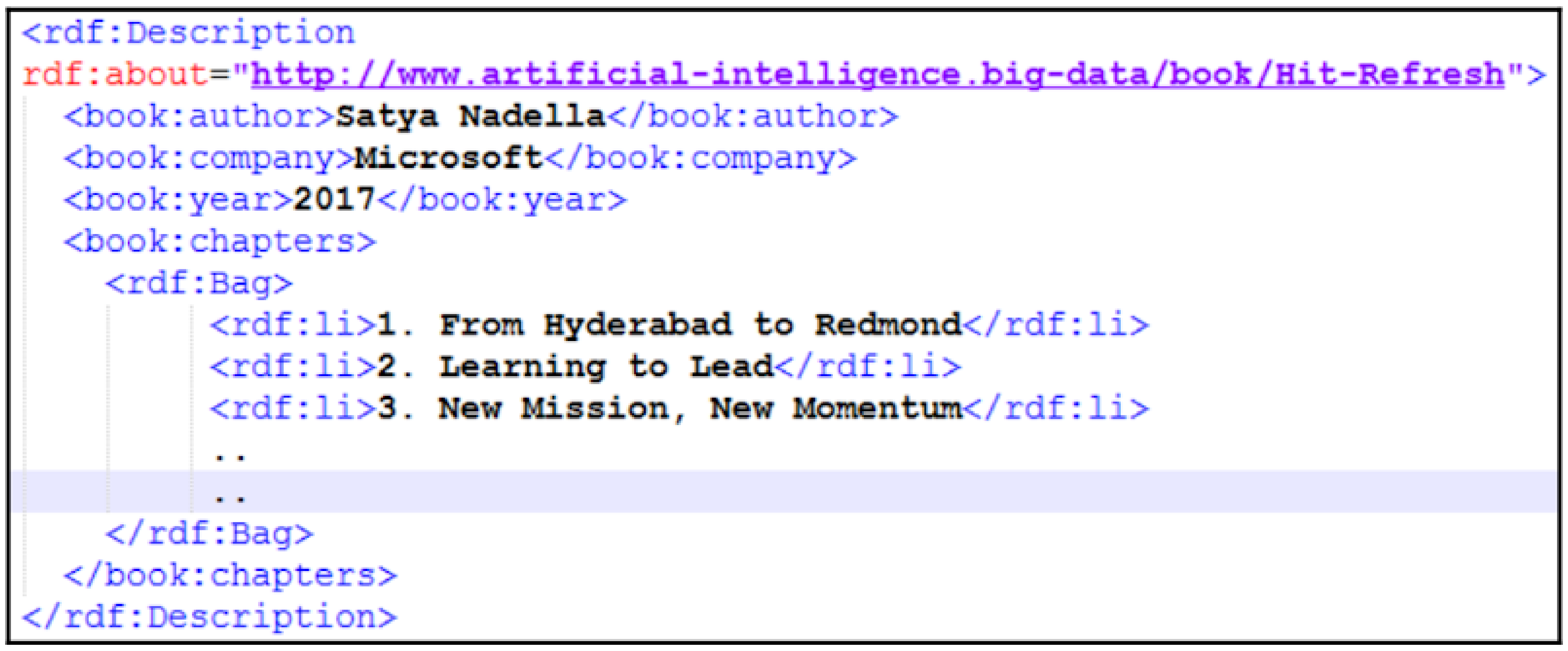
Your RDF document validated successfully.

Triples of the Data Model

Number	Subject	Predicate	Object
1	http://www.artificial-intelligence.big-data/book/Hit-Refresh	http://www.artificial-intelligence.big-data/book#author	"Satya Nadella"
2	http://www.artificial-intelligence.big-data/book/Hit-Refresh	http://www.artificial-intelligence.big-data/book#company	"Microsoft"
3	http://www.artificial-intelligence.big-data/book/Hit-Refresh	http://www.artificial-intelligence.big-data/book#year	"2017"
4	http://www.artificial-intelligence.big-data/book/Shoe-Dog	http://www.artificial-intelligence.big-data/book#author	"Phil Knight"
5	http://www.artificial-intelligence.big-data/book/Shoe-Dog	http://www.artificial-intelligence.big-data/book#company	"Nike"
6	http://www.artificial-intelligence.big-data/book/Shoe-Dog	http://www.artificial-intelligence.big-data/book#year	"2016"

RDF containers

RDF containers are used to describe groups of things. Here is an example:



```
<rdf:Description
rdf:about="http://www.artificial-intelligence.big-data/book/Hit-Refresh">
  <book:author>Satya Nadella</book:author>
  <book:company>Microsoft</book:company>
  <book:year>2017</book:year>
  <book:chapters>
    <rdf:Bag>
         <rdf:li>1. From Hyderabad to Redmond</rdf:li>
         <rdf:li>2. Learning to Lead</rdf:li>
         <rdf:li>3. New Mission, New Momentum</rdf:li>
         ..
         ..
    </rdf:Bag>
  </book:chapters>
</rdf:Description>
```

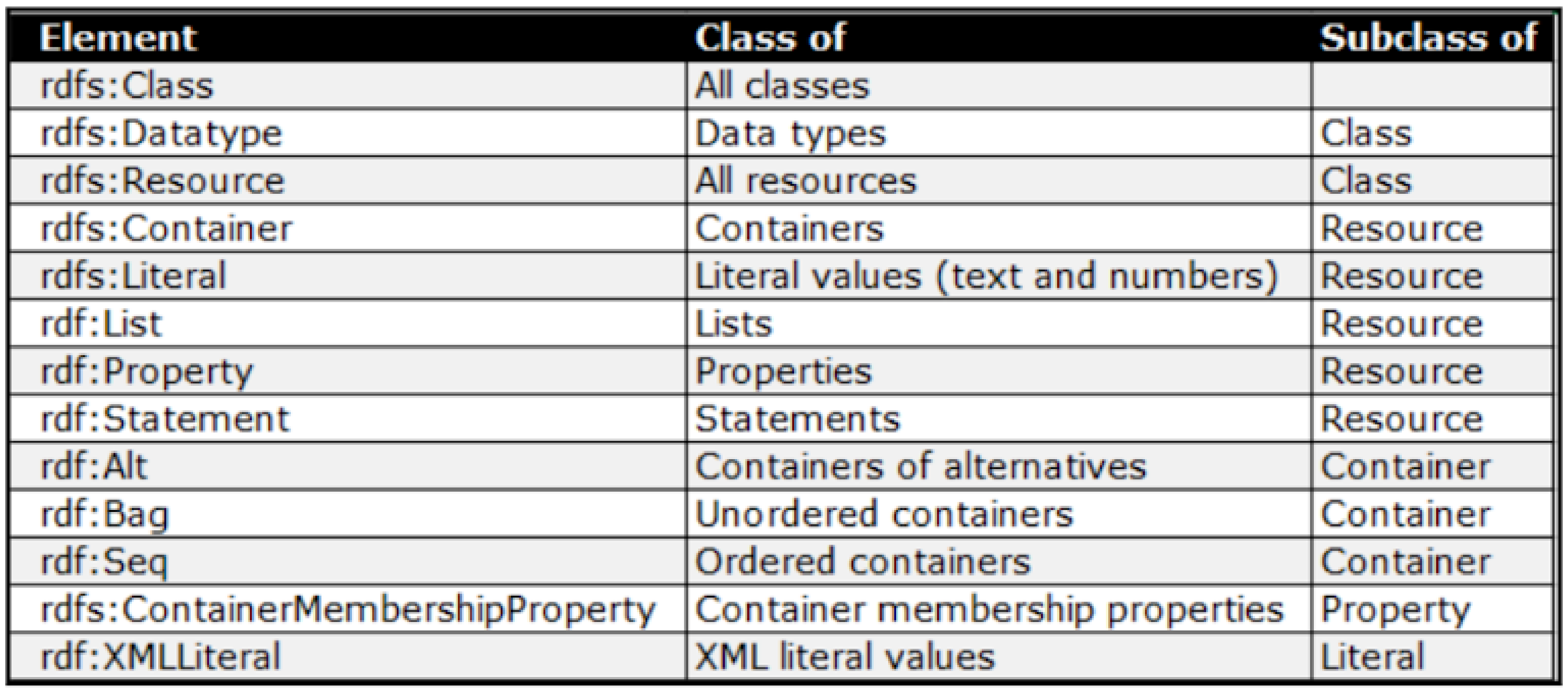
The `<rdf:Bag>` element is used to describe a list of values that do not have to be in a specific order.

`<rdf:Seq>` is similar to `<rdf:Bag>`. However, the elements represent an ordered list.

`<rdf:Alt>` is used to represent a list of alternate values for the element.

RDF classes

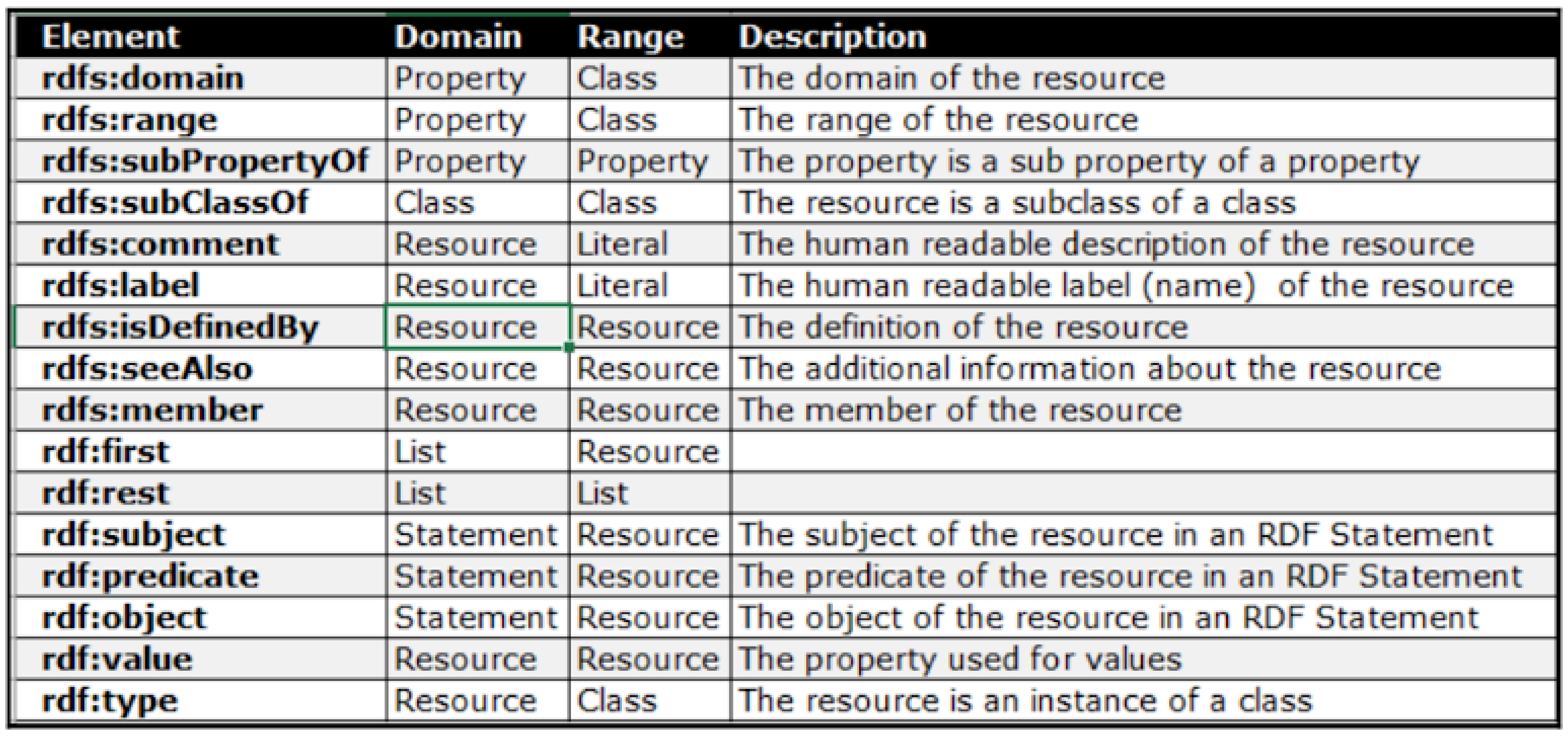
The RDF classes are listed in the following images:



Element	Class of	Subclass of
rdfs:Class	All classes	
rdfs:Datatype	Data types	Class
rdfs:Resource	All resources	Class
rdfs:Container	Containers	Resource
rdfs:Literal	Literal values (text and numbers)	Resource
rdf:List	Lists	Resource
rdf:Property	Properties	Resource
rdf:Statement	Statements	Resource
rdf:Alt	Containers of alternatives	Container
rdf:Bag	Unordered containers	Container
rdf:Seq	Ordered containers	Container
rdfs:ContainerMembershipProperty	Container membership properties	Property
rdf:XMLLiteral	XML literal values	Literal

RDF properties

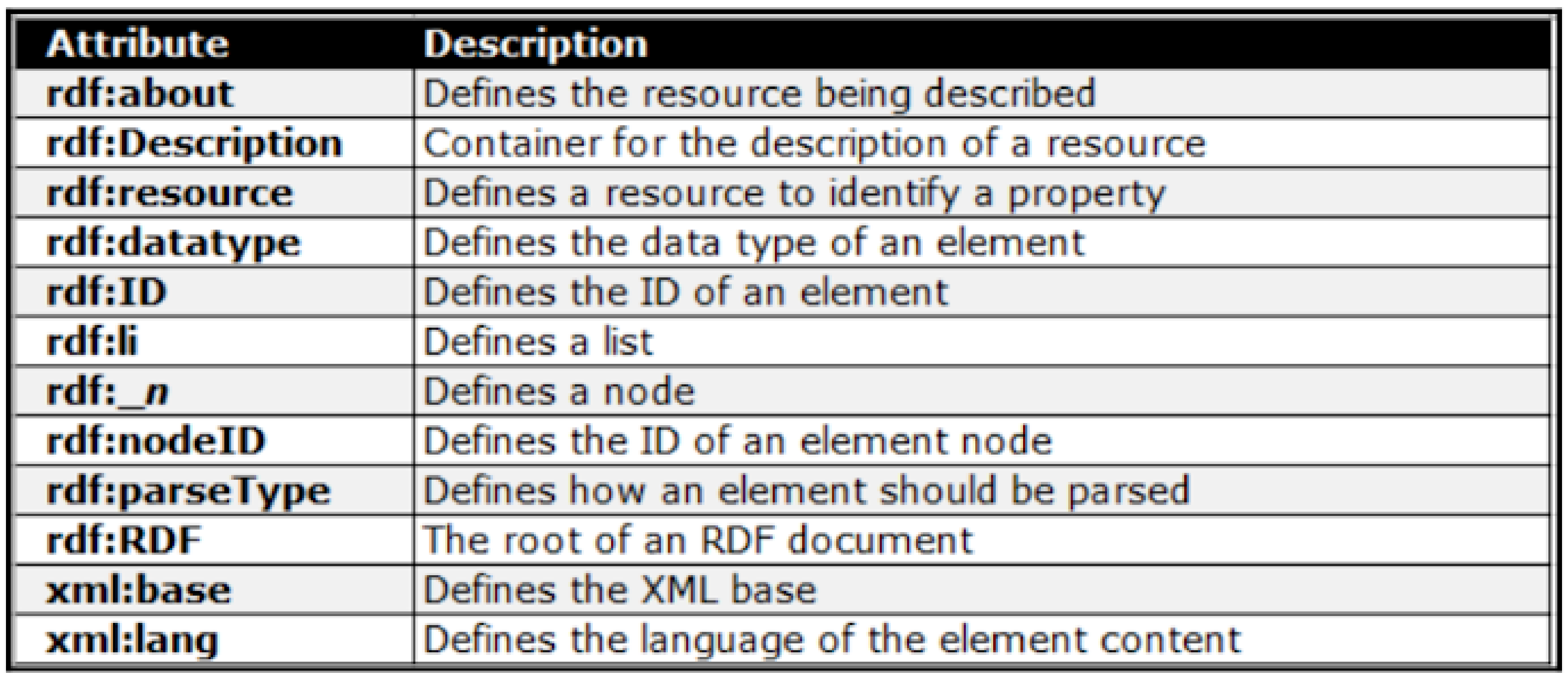
The RDF properties are listed as follows:



Element	Domain	Range	Description
rdfs:domain	Property	Class	The domain of the resource
rdfs:range	Property	Class	The range of the resource
rdfs:subPropertyOf	Property	Property	The property is a sub property of a property
rdfs:subClassOf	Class	Class	The resource is a subclass of a class
rdfs:comment	Resource	Literal	The human readable description of the resource
rdfs:label	Resource	Literal	The human readable label (name) of the resource
rdfs:isDefinedBy	Resource	Resource	The definition of the resource
rdfs:seeAlso	Resource	Resource	The additional information about the resource
rdfs:member	Resource	Resource	The member of the resource
rdf:first	List	Resource	
rdf:rest	List	List	
rdf:subject	Statement	Resource	The subject of the resource in an RDF Statement
rdf:predicate	Statement	Resource	The predicate of the resource in an RDF Statement
rdf:object	Statement	Resource	The object of the resource in an RDF Statement
rdf:value	Resource	Resource	The property used for values
rdf:type	Resource	Class	The resource is an instance of a class

RDF attributes

The various RDF attributes are listed as follows:



Attribute	Description
rdf:about	Defines the resource being described
rdf:Description	Container for the description of a resource
rdf:resource	Defines a resource to identify a property
rdf:datatype	Defines the data type of an element
rdf:ID	Defines the ID of an element
rdf:li	Defines a list
rdf:_*n*	Defines a node
rdf:nodeID	Defines the ID of an element node
rdf:parseType	Defines how an element should be parsed
rdf:RDF	The root of an RDF document
xml:base	Defines the XML base
xml:lang	Defines the language of the element content

Using OWL, the Web Ontology Language

While the **RDF** and **corresponding schema definitions** (**RDFS**) provide a structure for the semantic view of the information assets, there are some limitations with RDFS. RDFS cannot describe the entities in sufficient detail. There is no way to define localized ranges for the entity attributes, and the domain-specific constraints cannot be explicitly expressed. The existence or non-existence of a related entity, along with cardinality constraints (one-to-one, one-to-many, and so on), cannot be represented with RDFS. It is difficult to represent transitive, inverse, and symmetrical relationships. One of the important aspects of real-world entity relationships is logical reasoning and inferences, without explicit mention of the relationship. RDFS cannot provide reasoning support for the related entities.

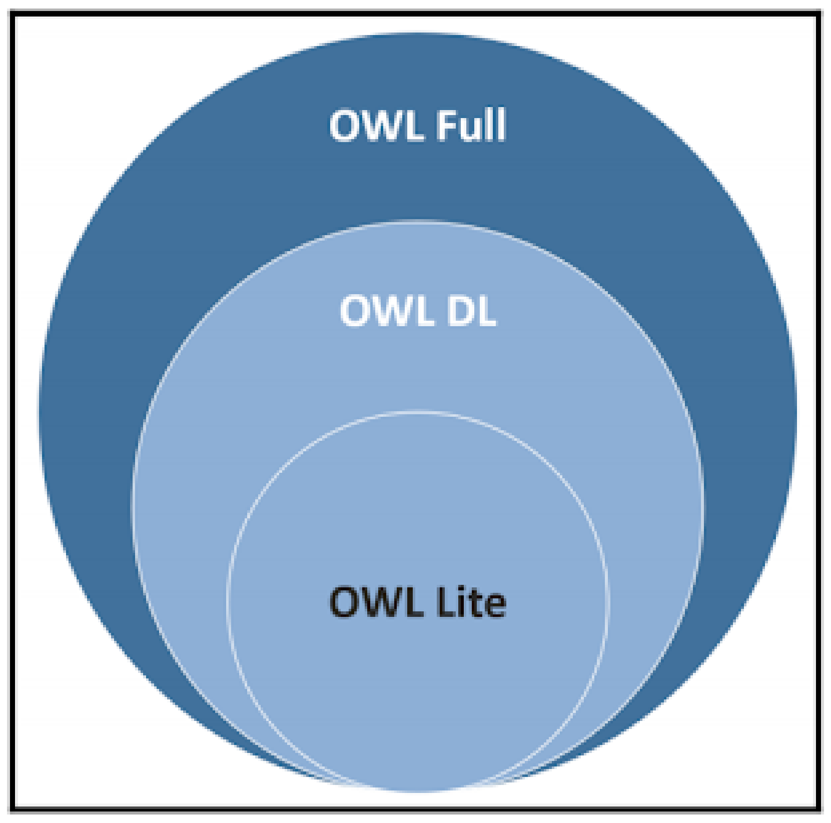
The **Web Ontology Language** (**OWL**) extends and builds on top of RDF/RDFS. OWL is a family of knowledge representation languages for authoring Ontologies.

TIP

Actually, OWL is not a real acronym. The language started out as WOL. However, the working group disliked the acronym WOL. Based on conversations within the working group, OWL had just one obvious pronunciation that was easy on the ear, and it opened up great opportunities for a logo—owls are associated with wisdom!

For building intelligent systems that can communicate across domains, there is a need to overcome the limitations of RDFS and equip the machines with access to structured collections of knowledge assets and sets of inference rules that can be used for automated reasoning. OWL provides formal semantics for knowledge representation and attempts to describe the meaning of the entities and their relationships and reasoning precisely.

There are three species of OWL:



- **OWL DL**: This is used for supporting description logic. This supports maximum expressiveness and logical reasoning capabilities. This is characterized by:
    - Well-defined semantics
    - Well-understood formal properties for the entities
    - The ease of implementation of known reasoning algorithms
- **OWL Full**: This is based on RDFS-compatible semantics. It complements the predefined RDF and OWL vocabulary. However, with OWL Full, the software cannot completely reason and inference.
- **OWL Lite**: This is used for expressing taxonomy and simple constraints such as zero-to-one cardinality.

OWL represents entities as classes. For example, let's define an entity of `PlayGround` with OWL:

```
<owl:Class rdf:ID="PlayGround">
```

Now, define `FootballGround` and state that `FootballGround` is a type of `PlayGround`:

```
<owl:Class rdf:ID="FootballGround">
    <rdf:subClassOf rdf:resource="#PlayGround"/>
</owl:Class>
```

OWL provides several other mechanisms for defining classes:

- `equivalentClass`: Represents that the two classes (across Ontologies and domains) are synonymous.
- `disjointWith`: Represents that an instance of a class cannot be an instance of another class. For example, `FootballGround` and `HockyGround` are stated as disjointed classes.
- Boolean combinations:
    - `unionOf`: Represents that a class contains things that are from more than one class
    - `intersectionOf`: Represents that a class contains things that are in both one and the other
    - `complementOf`: Represents that a class contains things that are not other things

SPARQL query language

With a generic understanding of Ontologies, the RDF, and OWL, we are able to fundamentally understand how intelligent systems can communicate with each other seamlessly with a semantic view of the world. With a semantic worldview, the entities come to life by translating data assets into information and information assets into knowledge. It is imperative that there is a common language to leverage a semantic worldview so that heterogeneous systems can communicate with each other. SPARQL is a W3C standard that is attempting to be the global query language with the primary goal of interoperability. SPARQL is a recurring acronym and stands for **SPARQL Protocol and RDF Query Language**. As the name indicates, it is a query language for querying knowledge (as triples) stored in RDF format. Traditionally, we stored the information in relational databases in tabular format. The relational database view of the entities can easily be represented as triples. For example, let us once again consider the `BOOK` table:

`Book_ID`	`Title`	`Author`	`Company`	`Year`
`1`	`Hit Refresh`	`Satya Nadella`	`Microsoft`	`2017`
`2`	`Shoe Dog`	`Phil Knight`	`Nike`	`2016`

Here, the row identifier (`Book_ID` and `Title`) is the subject, the column name is the predicate, and the column value is the object. For example:

A Triple:

{1: Hit Refresh}	{Author}	{Satya Nadella}
Subject (Entity Name)	Predicate (Attribute Name)	Object (Attribute Value)

The subjects and predicates are represented using URIs which universally identify specific subjects and predicates as resources:

```
http://www.artificial-intelligence.big-data/book#
http://www.artificial-intelligence.big-data/book#author "Satya Nadella"
```

Turtle syntax allows an RDF graph to be completely written in a compact and natural text form. It provides abbreviations for common usage patterns and datatypes. This format is compatible with the triple pattern syntax of SPARQL.

Let us use the turtle syntax to represent the `book` table in RDF format:

```
@prefix book: <http://www.artificial-intelligence.big-data/book#>

book:1 book:Title "Hit Refresh"
book:1 book:Author "Satya Nadella"
book:1 book:Company "Microsoft"
book:1 book:Year "2017"

book:2 book:Title "Shoe Dog"
book:2 book:Author "Phil Knight"
book:2 book:Company "Nike"
book:2 book:Year "2016"
```

Let us use a simple SPARQL query for getting a list of books published in the year `2017`:

```
PREFIX book: <http://www.artificial-intelligence.big-data/book#>

SELECT ?books
WHERE
{
    ?books book:year "2017" .
}
```

We have the following result:

```
?books
book:1
```

Here is another `SELECT` query, which fetches more data elements from the dataset:

```
PREFIX book: <http://www.artificial-intelligence.big-data/book#>

SELECT ?books ?bookName ?company
WHERE
{
    ?books book:year "2017" .
    ?books book:title ?bookName .
    ?books book:company ?company .
}
```

The result is as follows:

```
?books    ?bookName     ?company
book:1    Hit Refresh   Microsoft
```

While we are discussing role of Ontologies in the context of *Artificial Intelligence for Big Data*, a complete reference to OWL and SPARQL is outside of the scope of this book. In the following subsections, we will introduce a generic SPARQL language reference, which will help us leverage Ontologies to build artificial intelligence.

Generic structure of an SPARQL query

The generic structure of SPARQL is as follows:

- `PREFIX`: Similar to the declaration of namespaces in the context of XML, and package in the context of Java, or any similar programming languages, PREFIX is the SPARQL equivalent, which ensures uniqueness among entity representations and eliminates the need for typing long URI patterns within SPARQL code.
- `SELECT` / `ASK` / `DESCRIBE` / `CONSTRUCT`:
    - `SELECT`: This is an equivalent of SQL's `SELECT` clause. It defines the attributes that are required to be fetched from the RDF triples that fulfill the selection criteria.
    - `ASK`: This returns a Boolean value of true or false depending on the availability of the RDF triples, and based on the selection criteria within the RDF knowledge base.

    - DESCRIBE: This query construct returns a graph containing all the available triples from the RDF knowledge base which match the selection criteria.
    - CONSTRUCT: This is very handy when creating a new RDF graph from an existing RDF based on selection criteria and filter conditions. This is the equivalent of XSLT in the context of XML. XSLT transforms XML in the intended format.
- FROM: Defines the data source of the RDF endpoint, against which the query will be run. This is the SQL equivalent of the FROM <TABLE_NAME> clause. The endpoint can be a resource on the internet or a local data store accessible to the query engine.
- WHERE: Defines the part of the RDF graph we are interested in. This is the equivalent of the WHERE SQL clause which defines filter conditions to fetch specific data from the entire dataset.

Additional SPARQL features

The additional SPARQL features are as follows:

- Optional matching: Unlike traditional relational data stores, where the database schemas and constraints are predefined for the structured representation of data, in the big data word we deal with unstructured datasets. The attributes of the two resources of the same type may be different. Optional matching comes in handy when handling heterogeneous representations of the entities. The OPTIONAL block is used to select the data elements if they exist.
- Alternative matching: Once again, considering the unstructured nature of knowledge assets, alternating matching provides a mechanism to return whichever properties are available.
- UNION: This is in contrast to the OPTIONAL pattern. In the case of UNION, at least one of the datasets must find a match given the query criteria.
- DISTINCT: This is the equivalent of the DISTINCT SQL clause, which excludes multiple occurrences of the same triple within the result.

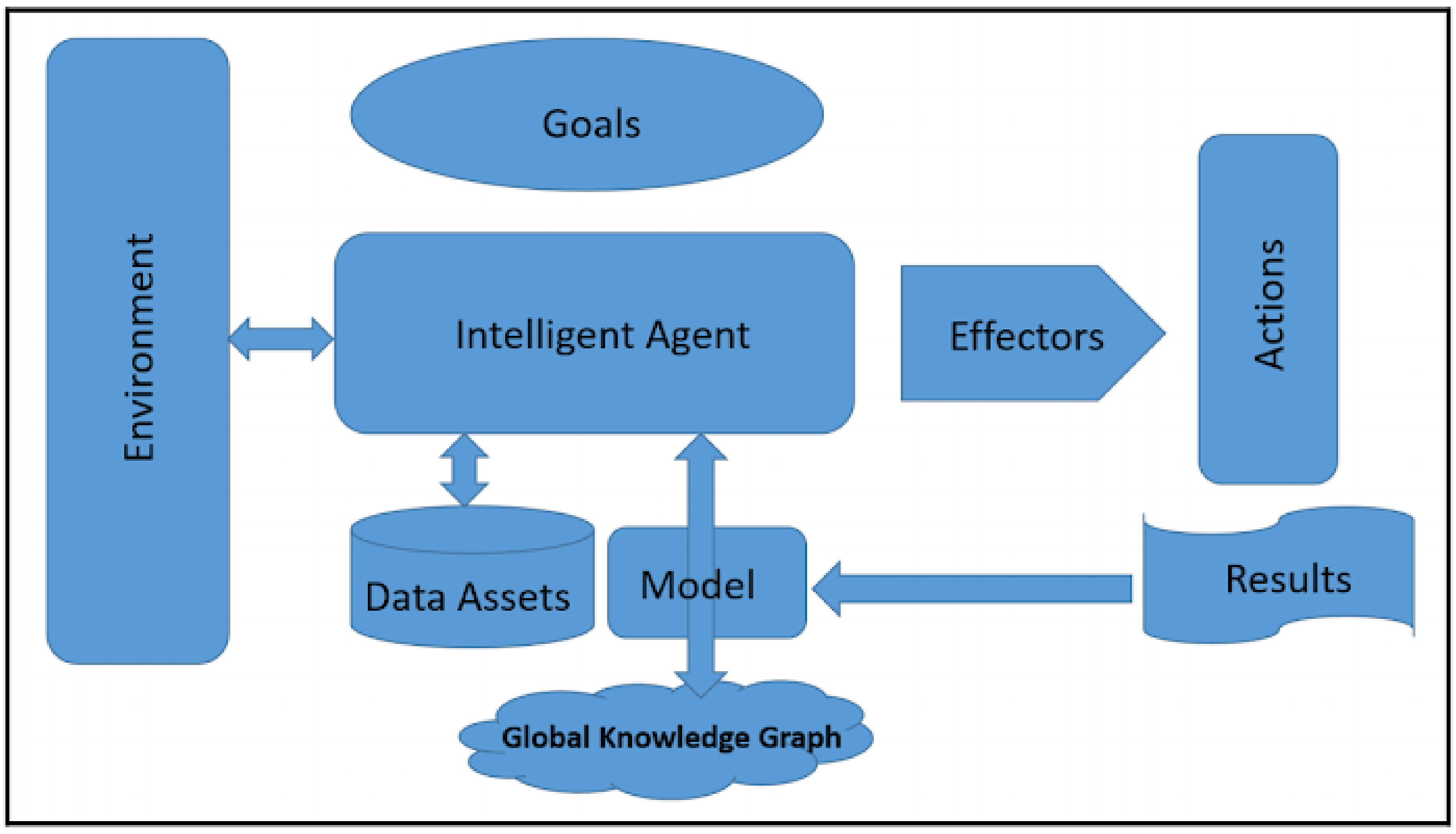
- ORDER BY: Instructs the query to sequence results by a specific variable either in ascending or descending order. This is also equivalent to ORDER BY clause in SQL.
- FILTERS and regular expressions: SPARQL provides features to restrict the result set triples by using expressions. Along with mathematical and logical expressions, SPARQL allows for the use of regular expressions to apply filters on datasets based on textual patterns.
- GROUP BY: This allows the grouping of the resulting RDF triples based on one or more variables.
- HAVING: This facilitates a selection of the query results at the group level.
- SUM, COUNT, AVG, MIN, MAX, and so on are the functions available to be applied at the group level.

Building intelligent machines with Ontologies

In this chapter, we have looked at the role of Ontology in the management of big data assets as knowledge repositories, and understood the need for computational systems to perceive the data as things instead of strings. Although some of the big systems and web search engines use a semantic world view, the adoption of Ontology as a basis for systems is slow. The custodians of data assets (governments and everyone else) need to model knowledge assets in a consistent and standardized manner in order for us to evolve current computational systems into intelligent systems.

Let us consider a use case that leverages Ontology-based knowledge graphs in order to simplify the flight boarding process. We have all experienced a hugely manual and time-consuming process when boarding a flight. From the time we enter the airport to the time we board the flight, we go through a number of security checks and experience document verification. In a connected world where all the knowledge assets are standardized and defined as domain-specific Ontologies, it is possible to develop intelligent agents to make the flight boarding process hassle free and seamless.

Let us define the generic characteristics of an intelligent agent:



A little expansion on the characteristics is as follows:

- **Goals**: Every intelligent system should have a well defined set of goals. These goals govern the rational decisions taken by the intelligent system and drive actions and hence results. For example, in the case of an intelligent agent that is responsible for the flight boarding process, one of the goals is to restrict access to anyone who does not pass all security checks, even if the person has a valid air ticket. In defining the goals for intelligent agents, one of the prime considerations should be that the AI agent or systems should complement and augment human capabilities.
- **Environment**: The intelligent agent should operate within the context of the environment. Its decisions and actions cannot be independent of the context. In our example use case, the environment is the airport, the passenger gates, flight schedules, and so on. The agents perceive the environment with various sensors, for example video cameras.
- **Data Assets**: The intelligent agent needs access to historical data in terms of the domain and the context in which it operates. The data assets can be available locally and globally (internet endpoints). These data assets ideally should be defined as RDF schema structures with standardized representations and protocols. These data assets should be queryable with standard languages and protocols (SPARQL) in order to ensure maximum interoperability.

- **Model**: This is where the real intelligence of the agent is available as algorithms and learning systems. These models evolve continuously based on the context, historical decisions, actions, and results. As a general rule, the model should perform better (more accurately) over a period of time for similar contextual inputs.
- **Effectors**: These are the tangible aspects of the agent which facilitate actions. In the example of an airline passenger boarding agent, the effector can be an automated gate opening system which opens a gate once all the passengers are fully validated (having a valid ticket, identity, and no security check failures). The external world perceives the intelligent agent through effectors.
- **Actions and Results**: Based on the environmental context, the data assets, and the trained models, the intelligent agent makes decisions that trigger actions through the effectors. These actions provide results based on the rationality of the decision and accuracy of the trained model. The results are once again fed into model training in order to improve accuracy over a period of time.

At a high level, the method of the intelligent agent, which facilitates the flight boarding process, can be depicted as follows:

1. When a passenger walks into the airport, a video camera reads the image and matches it to the data assets available to the agent. These data assets are Ontology objects which are loosely coupled and have flexibility of structure and attributes. Some of the inferences are made at the first level of matching to correctly identify the person who has entered the airport.
2. If the person cannot be identified with the video stream, the first airport gate does not open automatically and requires a fingerprint scan from the passenger. The fingerprint scan is validated against the dataset, which is once again an Ontology object representation of the person entity. If the person is not identified at this stage, they are flagged for further manual security procedures.
3. Once the person is correctly identified, the agent scans the global active ticket directory in order to ensure that the person has a valid ticket for a flight that departs from the airport in a reasonable time window. The global ticket directory and the flight database is also available as Ontology objects for the agent to refer to in real time.
4. Once ticket validity is ensured, a boarding pass is generated and delivered to the passenger's smartphone, once again by referring to the person Ontology to derive personal details in a secure manner. The real-time instructions for directions to the gate are also sent to the device.

The agent can seamlessly guide the passenger to the appropriate boarding gate. The system can be built easily once all the heterogeneous data sources are standardized and have Ontological representation, which facilitates maximum interoperability and eliminates a need to code diverse knowledge representations. This results in an overall reduction of complexity in the agent software and an increase in efficiency.

Ontology learning

With the basic concepts on Ontologies covered in this chapter, along with their significance in building intelligent systems, it is imperative that for a seamlessly connected world, the knowledge assets are consistently represented as domain Ontologies. However, the process of manually creating domain-specific Ontologies requires lots of manual effort, validation, and approval. Ontology learning is an attempt to automate the process of the generation of Ontologies, using an algorithmic approach on the natural language text, which is available at the internet scale. There are various approaches to Ontology learning, as follows:

- **Ontology learning from text**: In this approach, the textual data is extracted from various sources in an automated manner, and keywords are extracted and classified based on their occurrence, word sequencing, and patterns.
- **Linked data mining**: In this processes, the links are identified in the published RDF graphs in order to derive Ontologies based on implicit reasoning.
- **Concept learning from OWL:** In this approach, existing domain-specific Ontologies are leveraged for expand the new domains using an algorithmic approach.
- **Crowdsourcing**: This approach combines automated Ontology extraction and discovery based on textual analysis and collaboration with domain experts to define new Ontologies. This approach works great since it combines the processing power and algorithmic approaches of machines and the domain expertise of people. This results in improved speed and accuracy.

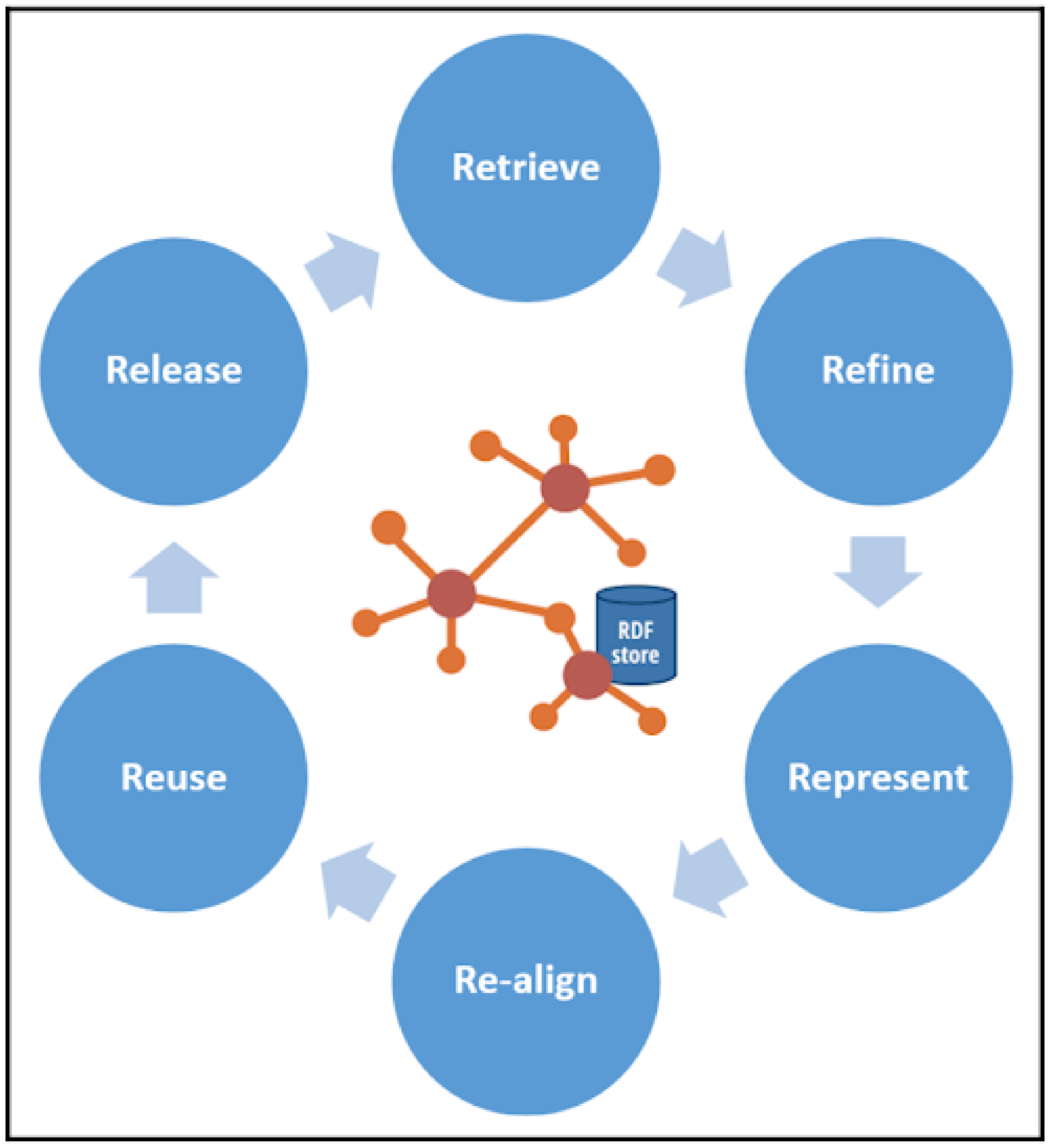
Here are some of the challenges of Ontology learning:

- **Dealing with heterogeneous data sources**: The data sources on the internet, and within application stores, differ in their forms and representations. Ontology learning faces the challenge of knowledge extraction and consistent meaning extraction due to the heterogeneous nature of the data sources.

- **Uncertainty and lack of accuracy**: Due the the inconsistent data sources, when Ontology learning attempts to define Ontology structures, there is a level of uncertainty in terms of the intent and representation of entities and attributes. This results in a lower level of accuracy and requires human intervention from domain experts for realignment.
- **Scalability**: One of the primary sources for Ontology learning is the internet, which is an ever growing knowledge repository. The internet is also an unstructured data source for the most part and this makes it difficult to scale the Ontology learning process to cover the width of the domain from large text extracts. One of the ways to address scalability is to leverage new, open source, distributed computing frameworks (such as Hadoop).
- **Need for post-processing**: While Ontology learning is intended to be an automated process, in order to overcome quality issues, we require a level of post-processing. This process need to be planned and governed in detail in order to optimize the speed and accuracy of new Ontology definitions.

Ontology learning process

The Ontology learning process consists of six Rs:



They are explained as followed:

- **Retrieve**: The knowledge assets are retrieved from the web and application sources from the domain specific stores using web crawls and protocol-based application access. The domain specific terms and axioms are extracted with a calculation of TF/IDF values and by the application of the C-Value / NC Value methods. Commonly used clustering techniques are utilized and the statistical similarity measures are applied on the extracted textual representations of the knowledge assets.
- **Refine**: The assets are cleansed and pruned to improve signal to noise ratio. Here, an algorithmic approach is taken for refinement. In the refinement step, the terms are grouped corresponding to concepts within the knowledge assets.
- **Represent**: In this step, the Ontology learning system arranges the concepts in a hierarchical structure using the unsupervised clustering method (at this point, understand this as a machine learning approach for the segmentation of the data; we will cover the details of unsupervised learning algorithms in the next chapter).
- **Re-align**: This is a type of post-processing step that involves collaboration with the domain experts. At this point, the hierarchies are realigned for accuracy. The Ontologies are aligned with instances of concepts and corresponding attributes along with cardinality constraints (one-to-one, one-to-many, and so on). The rules for defining the syntactic structure are defined in this step.
- **Reuse**: In this step, similar domain-specific Ontologies with connection endpoints are reused, and synonyms are defined in order to avoid parallel representations of the same concepts, which are finalized across other Ontology definitions.
- **Release**: In this step, the Ontologies are released for generic use and further evolution.

Frequently asked questions

Let's have a small recap of the chapter:

Q: What are Ontologies and what is their significance in intelligent systems?

A: Ontology as a generic term means the knowledge of everything that exists in this universe. As applicable to information systems, Ontologies represent a semantic and standardized view of the world's knowledge assets. They are domain-specific representations of knowledge and models related to real world entity representations. The intelligent systems that link heterogeneous knowledge domains need to have access to consistent representations of knowledge in order to interoperate and understand contextual events to make inferences and decisions, which trigger actions and hence results, in order to complement human capabilities.

Q: What are the generic properties of Ontologies?

A: Ontologies should be complete, unambiguous, domain-specific, generic, and extensible.

Q: What are the various components of Ontologies?

A: Various Ontology components are Concepts, Slots, Relationships, Axioms, Instances, and Operations.

Q: What is the significance of a universal data format in knowledge management systems?

A: The **Resource Description Format** (**RDF**) intends to be the universal format for knowledge representation, allowing heterogeneous systems to interact and integrate in a consistent and reliable manner. This forms the basis of the semantic view of the world.

Q: How is it possible to model the worldview with Ontologies? Is it possible to automate the Ontology definition process considering vast and ever-increasing knowledge stores in the universe?

A: Knowledge assets are growing exponentially in size with time. In order to create an Ontological representation of these assets, we need an automated approach, without which it will be difficult to catch up with the volume. Ontology learning takes an algorithmic approach by leveraging distributed computing frameworks to create a baseline model of the worldview. The Ontology learning process retrieves textual, unstructured data from heterogeneous sources, refines it, and represents it in a hierarchical manner. This is realigned with post-processing by reusing existing domain-specific knowledge assets, and finally released for generic consumption by intelligent agents.

Summary

In this chapter, we have explored the need for a standardized and consistent representation of the world's knowledge for the evolution of intelligent systems, and how these systems are modeled against the human brain. Ontologies, as applied to information systems, is a W3C standard that defines the generic rules for knowledge representation.

This chapter introduced the basic concepts of the RDF, OWL, and a query language to extract the knowledge representations within Ontology instances through SPARQL.

In this chapter, we have explored how to use Ontologies to build intelligent agents by looking at the generic characteristics of the intelligent agents. In the end, we learned how Ontology learning facilitates the speedy adoption of Ontologies for the worldview, with consistent knowledge assets and representations.

In the next chapter, we will get introduced to fundamental concepts of Machine Learning and how Big Data facilitates the learning process.

3 Learning from Big Data

In the first two chapters, we set the context for intelligent machines with the big data revolution and how big data is fueling rapid advances in artificial intelligence. We also emphasized the need for a global vocabulary for universal knowledge representation. We have also seen how that need is fulfilled with the use of ontologies and how ontologies help construct a semantic view of the world.

The quest is for the knowledge, which is derived from information, which is in turn derived from the vast amounts of data that we are generating. Knowledge facilitates a rational decision-making process for machines that complements and augments human capabilities. We have seen how the **Resource Description Framework** (**RDF**) provides the schematic backbone for the knowledge assets along with **Web Ontology Language** (**OWL**) fundamentals and the query language for RDFs (SPARQL).

In this chapter, we are going to look at some of the basic concepts of machine learning and take a deep dive into some of the algorithms. We will use Spark's machine learning libraries. **Spark** is one of the most popular computer frameworks for the implementation of algorithms and as a generic computation engine on big data. Spark fits into the big data ecosystem well, with a simple programming interface, and very effectively leverages the power of distributed and resilient computing frameworks. Although this chapter does not assume any background with statistics and mathematics, it will greatly help if the reader has some programming background, in order to understand the code snippets and to try and experiment with the examples.

In this chapter, we will see broad categories of machine learning in **supervised** and **unsupervised** learning, before taking a deep dive, with examples, into:

- Regression analysis
- Data clustering
- K-means

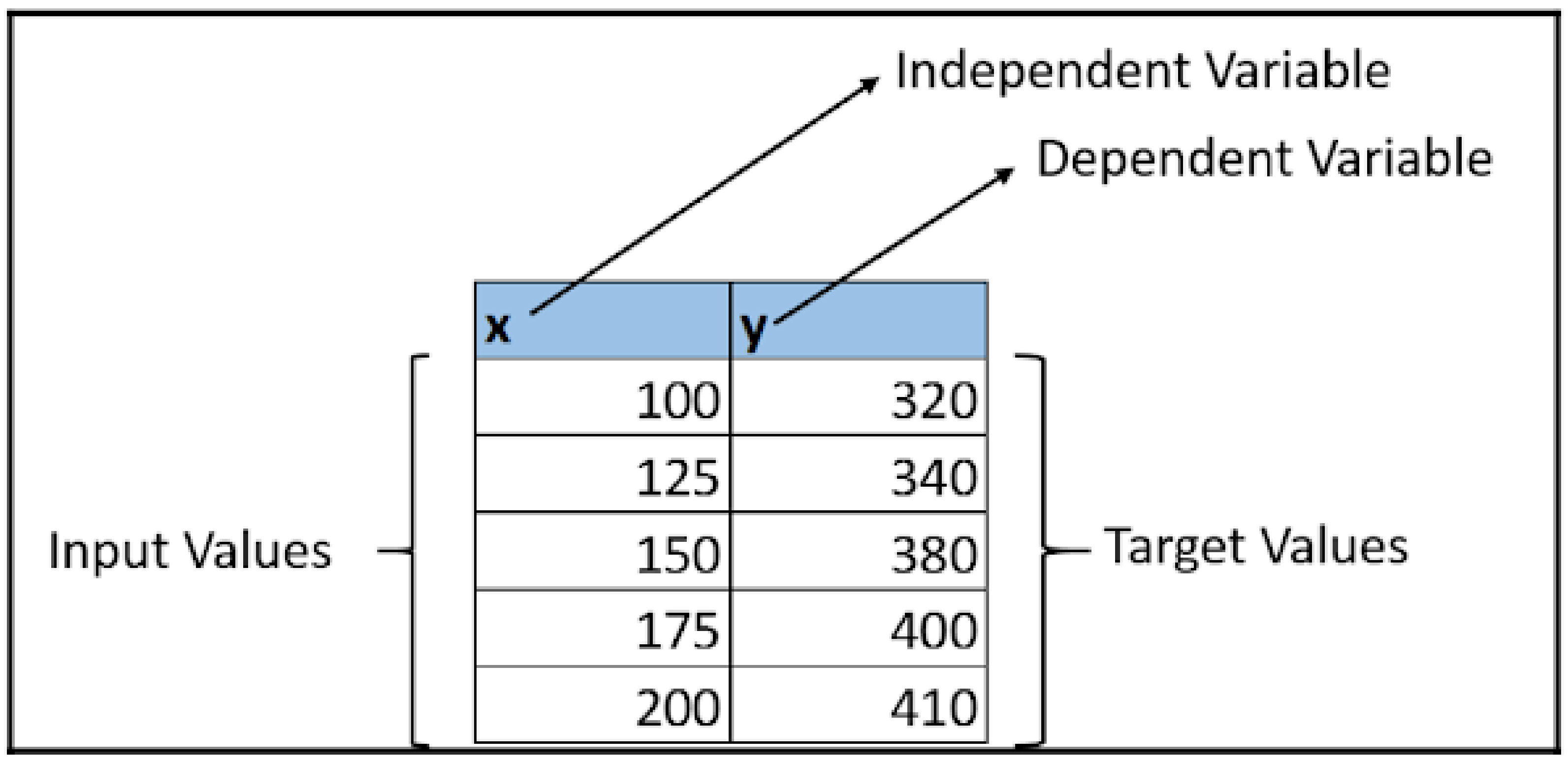
- Data dimensionality reduction
- Singular value decomposition
- Principal component analysis (PCA)

In the end, we will have an overview of the Spark programming model and **Spark's Machine Learning library** (**Spark MLlib**). With all this background knowledge at our disposal, we will implement a recommendation system to conclude this chapter.

Supervised and unsupervised machine learning

Machine learning at a broad level is categorized into two types: supervised and unsupervised learning. As the name indicates, this categorization is based on the availability of the historical data or the lack thereof. In simple terms, a supervised machine learning algorithm depends on the trending data, or version of truth. This version of truth is used for generalizing the model to make predictions on the new data points.

Let's understand this concept with the following example:



x	y
100	320
125	340
150	380
175	400
200	410

Figure 3.1 Simple training data: input (independent) and target (dependent) variables

Consider that the value of the **y** variable is dependent on the value of **x**. Based on a change in the value of **x**, there is a proportionate change in the value of **y** (think about any examples where the increase or decrease in the value of one factor proportionally changes the other).

Based on the data presented in the preceding table, it is clear that the value of **y** increases with an increase in the value of **x**. That means there is a direct relationship between **x** and **y**. In this case, **x** is called an independent, or input, variable and **y** is called a dependent, or target, variable. In this example, what will be the value of **y** when **x** is **220**? At this point, let's understand a fundamental difference between traditional computer programming and machine learning when it comes to predicting the value of the **y** variable for a specific value of **x=220**. The following diagram shows the traditional programming process:

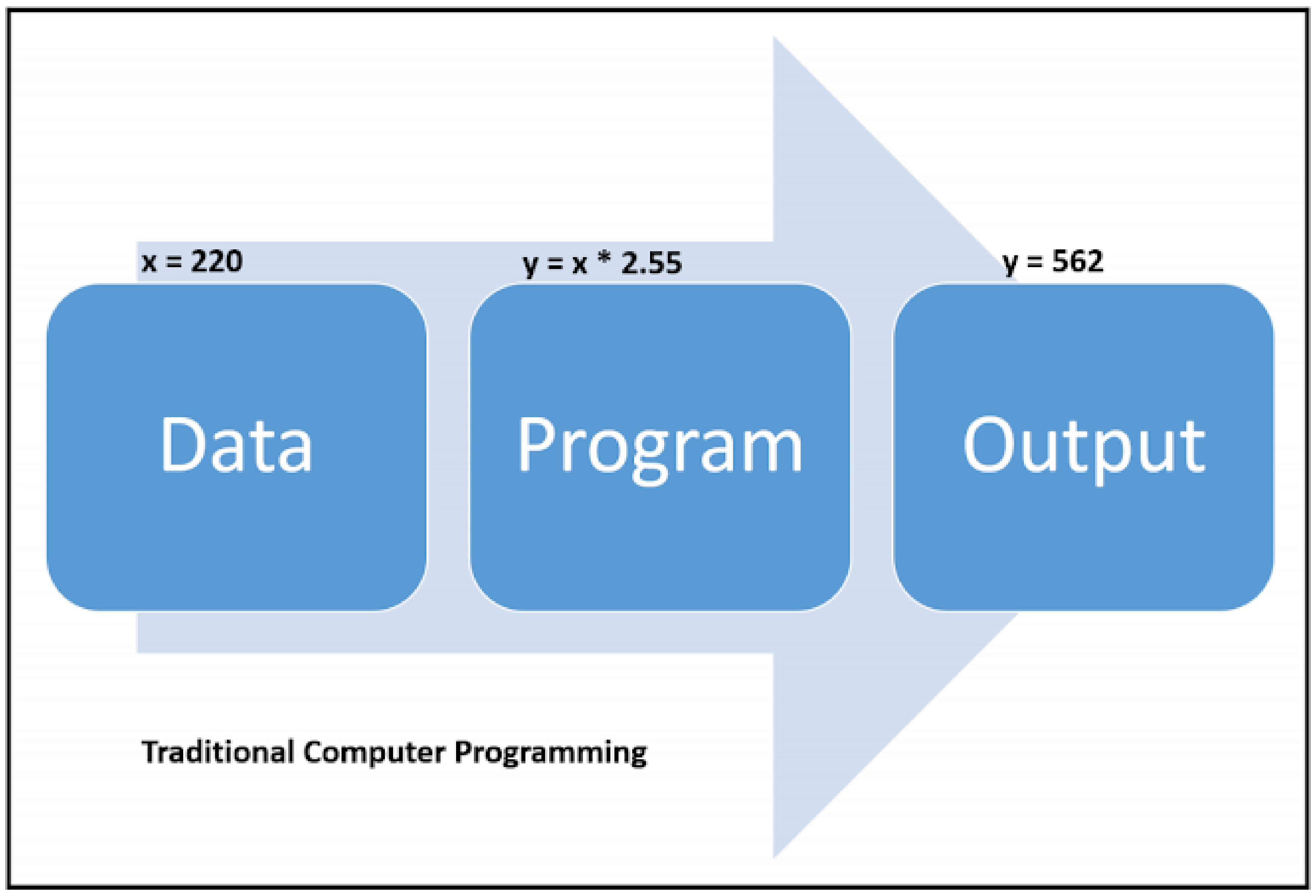
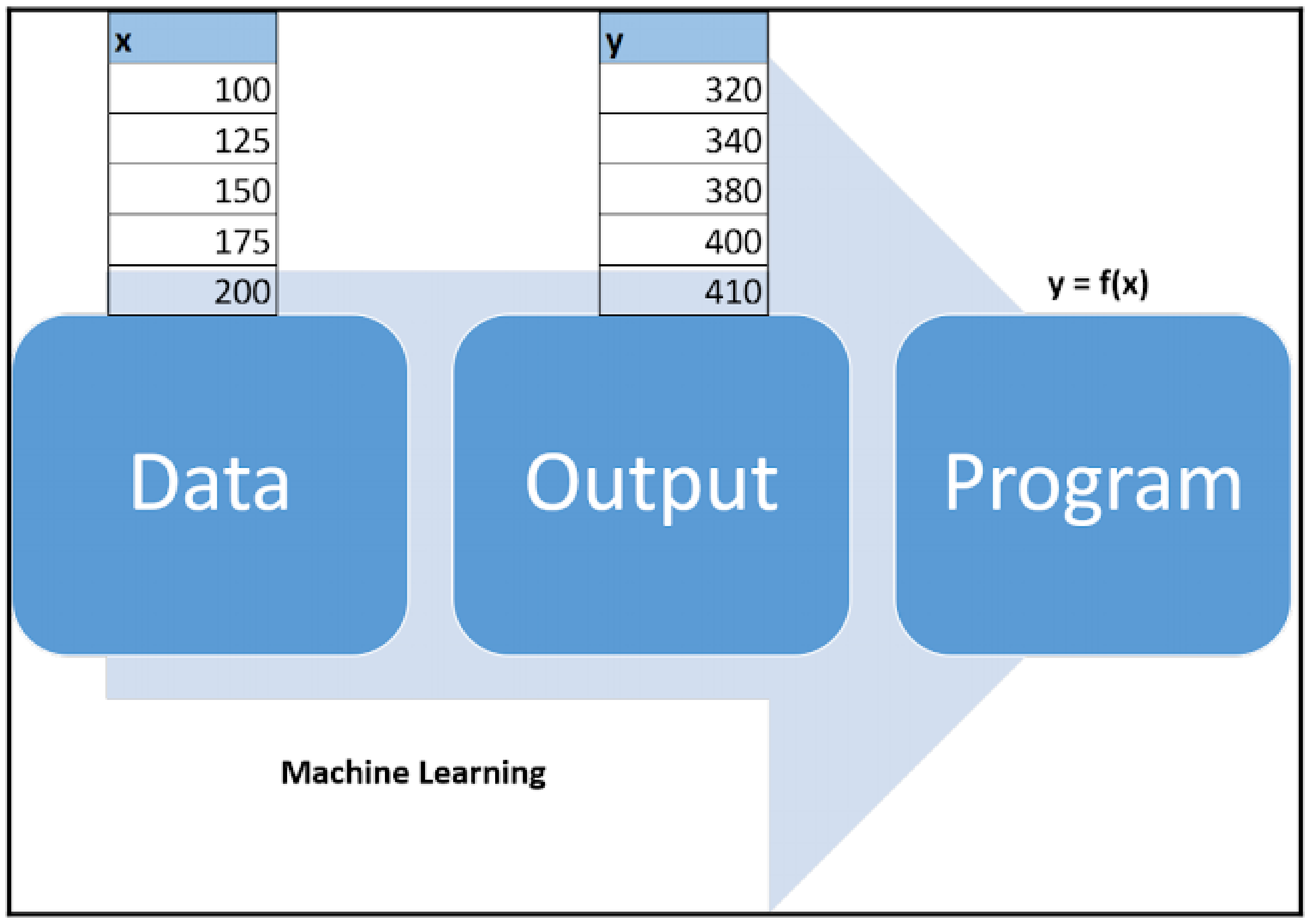


Figure 3.2 Traditional computer programming process

The traditional computer program has a predefined function that is applied on the input data to produce the output. In this example, a traditional computer program calculates the value of the (**y**) output variable as **562**.

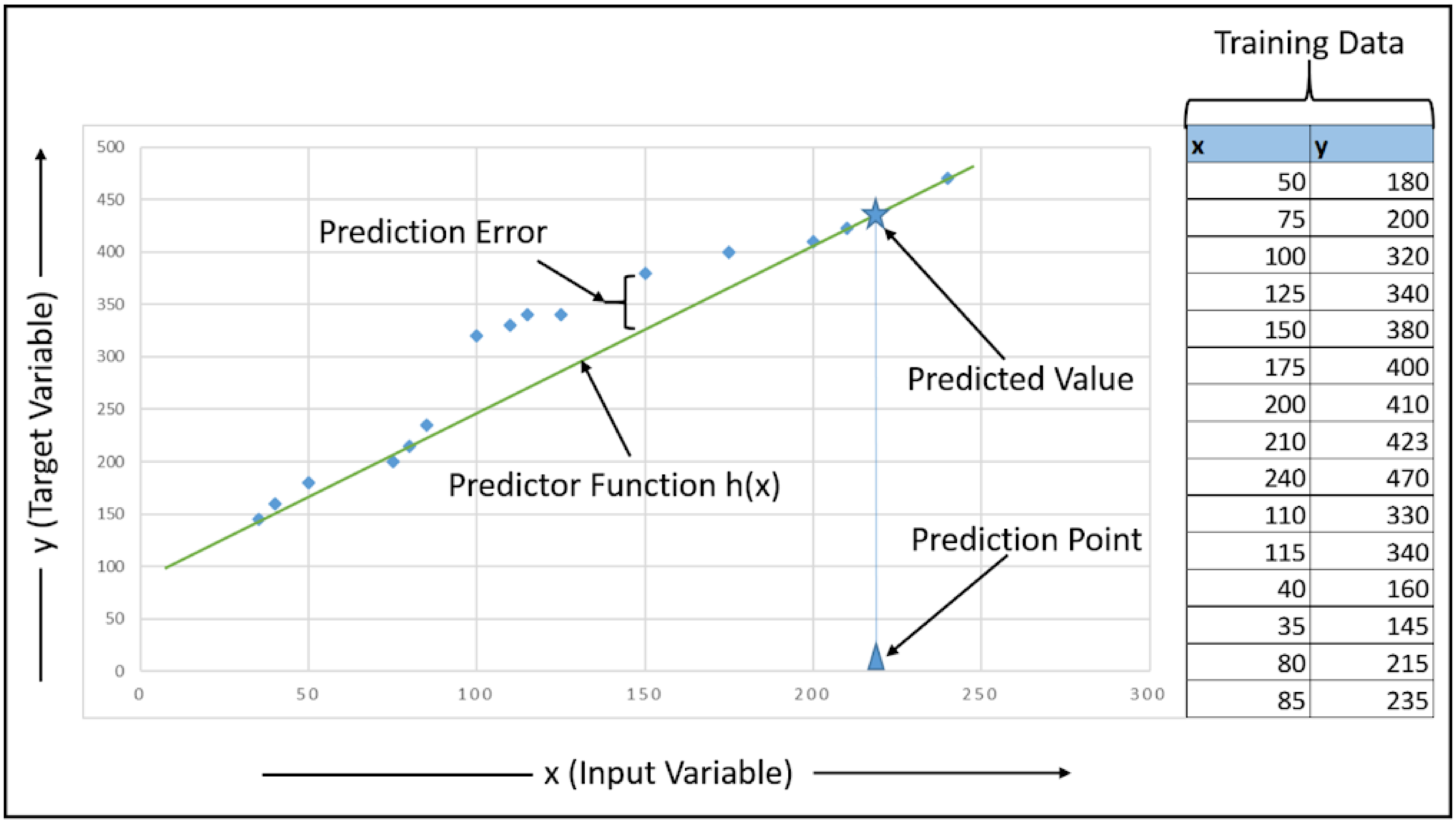
Have a look at the following diagram:



x	y
100	320
125	340
150	380
175	400
200	410

Figure 3.3 Machine learning process

In the case of supervised machine learning, the input and output data (training data) are used to create the program or the function. This is also termed the predictor function. A predictor function is used to predict the outcome of the dependent variable. In its simplest form, the process of defining the predictor function is called **model training**. Once a generalized predictor function is defined, we can predict the value of the target variable (y) corresponding to an input value (**x**). The goal of supervised machine learning is to develop a finely-tuned predictor function, **h(x)**, called **hypothesis**. Hypothesis is a certain function that we believe (or hope) is similar to the true function, the target function that we want to model. Let's add some more data points and plot those on a two-dimensional chart, like the following diagram:



x	y
50	180
75	200
100	320
125	340
150	380
175	400
200	410
210	423
240	470
110	330
115	340
40	160
35	145
80	215
85	235

Figure 3.4 Supervised learning (linear regression)

We have plotted the input variable on the *x* axis and the target variable on the *y* axis. This is a general convention used and hence the input variable is termed *x* and the output variable is termed *y*. Once we plot the data points from the training data, we can visualize the correlation between the data points. In this case, there seems to a direct proportion between *x* and *y*. In order for us to predict the value of *y* when *x = 220*, we can draw a straight line that tries to characterize, or model, the truth (training data). The straight line represents the predictor function, which is also termed as a hypothesis.

Based on the hypothesis, in this case our model predicts that the value of *y* when *x = 220* will be *~430*. While this hypothesis predicts the value of *y* for a certain value of *x*, the line that defines the predictor function does not cover all the values of the input variable. For example, based on the training data, the value of *y = 380* at *x = 150*. However, as per the hypothesis, the value comes out to be *~325*. This differential is called prediction error (*~55* units in this case). Any input variable (*x*) value that does not fall on the predictor function has some prediction error based on the derived hypothesis. The sum of errors for across all the training data is a good measure of the model's accuracy. The primary goal of any supervised learning algorithm is to minimize the error while defining a hypothesis based on the training data.

A straight-line hypothesis function is as good as an illustration. However, in reality, we will always have multiple input variables that control the output variable, and a good predictor function with minimal error will never be a straight line. When we predict the value of an output variable at a certain value of the input variable it is called **regression**. In certain cases, the historical data, or version of truth, is also used to separate data points into discrete sets (class, type, category). This is termed **classification**. For example, an email can be flagged as spam or not based on the training data. In the case of classification, the classes are known and predefined. The following image shows the classification with the **Decision Boundary**:

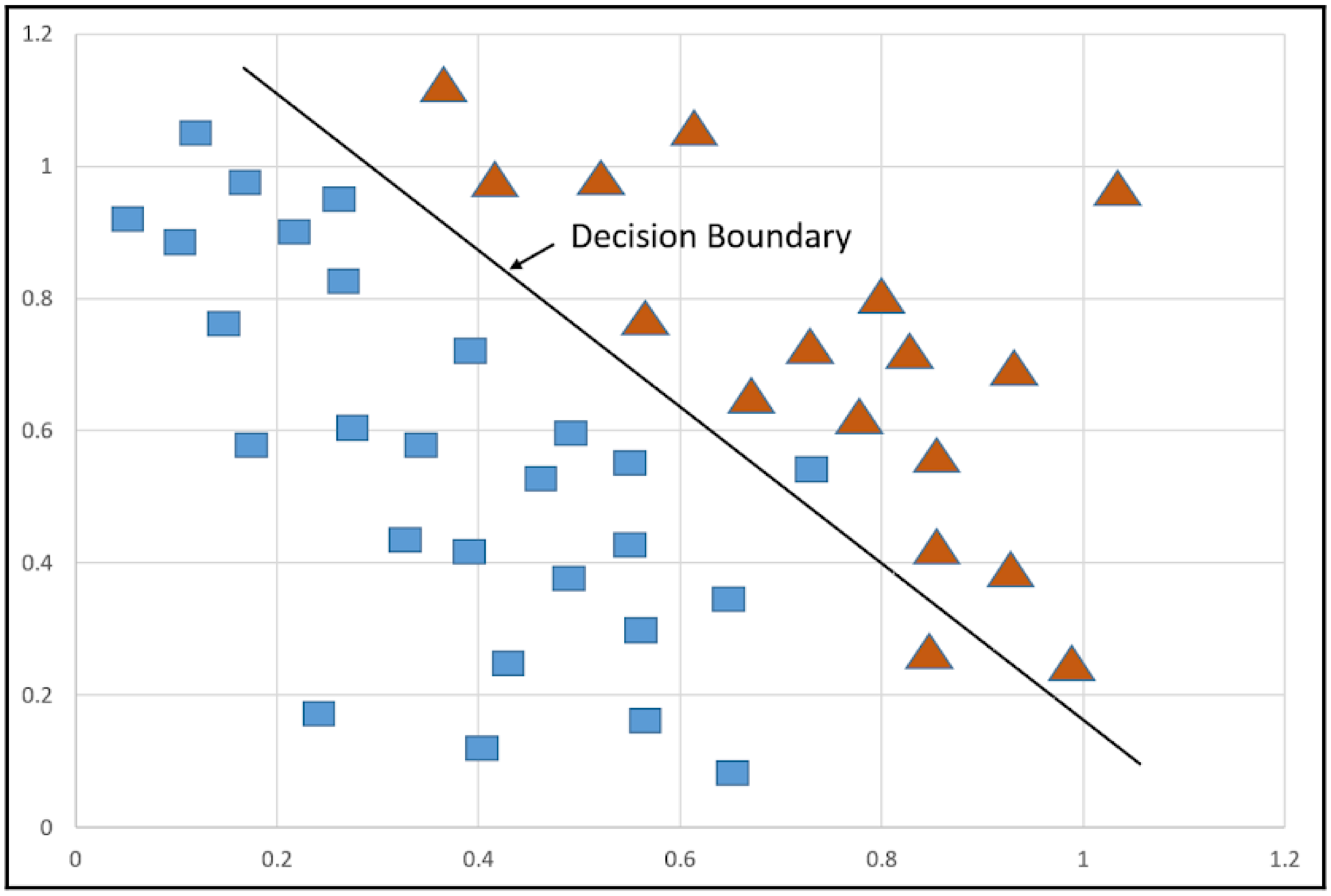


Figure 3.5 Classification with Decision Boundary

Here is a two-dimensional training dataset, where the output variables are separated by a Decision Boundary. Classification is a supervised learning technique that defines the Decision Boundary so that there is a clear separation of the output variables.

Regression and classification, as discussed in this section, require historical data to make predictions about the new data points. These represent *supervised* learning techniques. The generic process of supervised machine learning can be represented as follows:

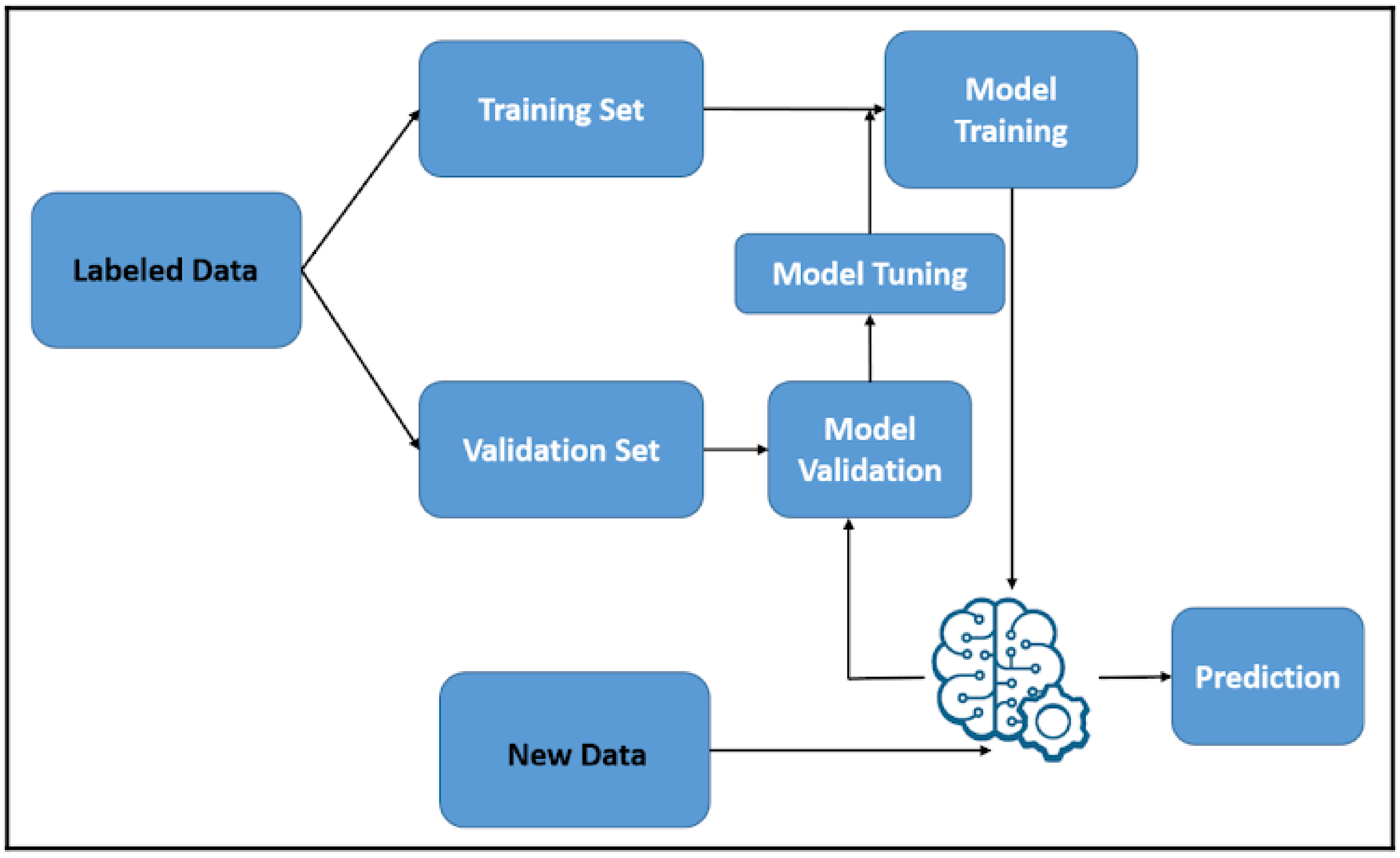


Figure 3.6 Generic supervised learning process

The labeled data, or the version of truth, is split into training and validation sets with random sampling. Typically, an 80-20 rule is followed with the split percentage of the training and validation sets. The training set is used for training the model (curve fitting) to reduce overall error of the prediction. The model is checked for accuracy with the validation set. The model is further tuned for the accuracy threshold and then utilized for the prediction of the dependent variables for the new data.

With this background in machine learning, let's take a deep dive into various techniques of supervised and unsupervised machine learning.

The Spark programming model

Before we deep dive into the Spark programming model, we should first arrive at an acceptable definition of what Spark is. We believe that it is important to understand what Spark is, and having a clear definition will help you to choose appropriate use cases where Spark is going to be useful as a technological choice.

There is no one silver bullet for all your enterprise problems. You must pick and choose the right technology from a plethora of options presented to you. With that, Spark can be defined as:

> ***Spark*** *is a distributed in-memory processing engine and framework that provides you with abstract APIs to process big volumes of data using an immutable distributed collection of objects called* ***Resilient Distributed Datasets****. It comes with a rich set of libraries, components, and tools, which let you write-in memory-processed distributed code in an efficient and fault-tolerant manner.*

Now that you are clear on what Spark is, let's understand how the Spark programming model works. The following diagram represents a high-level component of the Spark programming model:

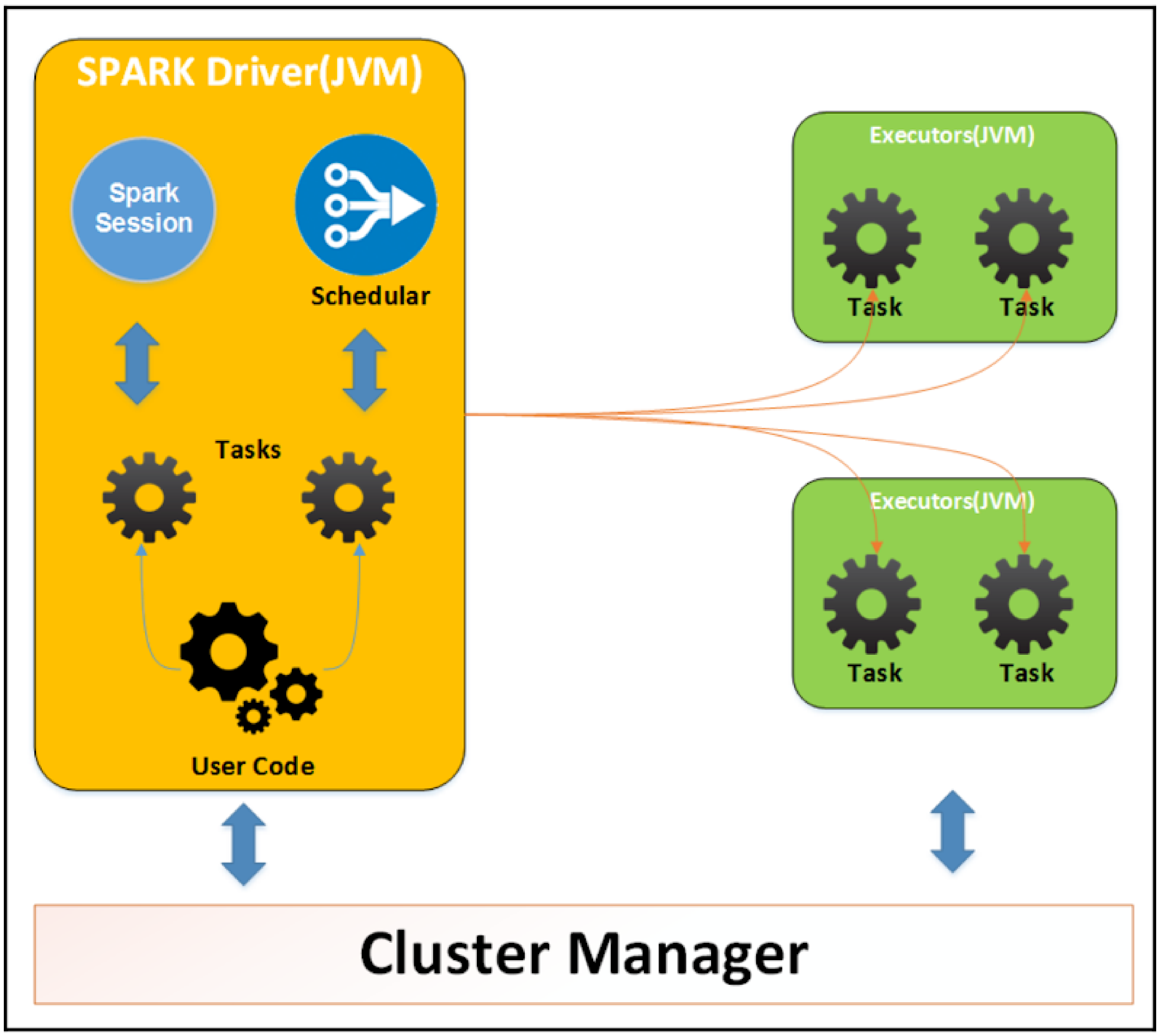


Figure 3.7 Spark programming model

As shown, all Spark applications are **Java Virtual Machine** (**JVM**)-based components comprising three processes: **driver**, **executor**, and **cluster manager**. The driver program runs as a separate process on a logically- or physically-segregated node and is responsible for launching the Spark application, maintaining all relevant information and configurations about launched Spark applications, executing application DAG as per user code and schedules, and distributing tasks across different available executors. Programmatically, the `main()` method of your Spark code runs as a driver. The driver program uses a `SparkContext` or `SparkSession` object created by user code to coordinate all Spark cluster activity. SparkContext or SparkSession is an entry point for executing any code using a Spark-distributed engine. To schedule any task, the driver program converts logical DAG to a physical plan and divides user code into a set of tasks. Each of those tasks are then scheduled by schedulers, running in Spark driver code, to run on executors. The driver is a central piece of any Spark application and it runs throughout the lifetime of the Spark application. If the driver fails, the entire application will fail. In that way, the driver becomes a single point of failure for the Spark application.

Spark executor processes are responsible for running the tasks assigned to it by the driver processes, storing data in in-memory data structures called RDDs, and reporting its code-execution state back to the driver processes. The key point to remember here is that, by default, executor processes are not terminated by the driver even if they are not being used or executing any tasks. This behavior can be explained with the fact that the RDDs follow a lazy evaluation design pattern. However, even if executors are killed accidentally, the Spark application does not stop and those executors can be relaunched by driver processes.

Cluster managers are processes that are responsible for physical machines and resource allocation to any Spark application. Even driver code is launched by the cluster manager processes. The cluster manager is a pluggable component and is cynical to the Spark user code, which is responsible for data processing. There are three types of cluster managers supported by the Spark processing engine: standalone, YARN, and Mesos.

Further reference to about Spark RDDs and cluster managers can be found at the following links:

- `https://spark.apache.org/docs/latest/cluster-overview.html`
- `https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#resilient-distributed-datasets-rdds`

The Spark MLlib library

The **Spark MLlib** is a library of machine learning algorithms and utilities designed to make machine learning easy and run in parallel. This includes regression, collaborative filtering, classification, and clustering. Spark MLlib provides two types of API included in the packages, namely `spark.mllib` and `spark.ml`, where `spark.mllib` is built on top of RDDs and spark.ml is built on top of the DataFrame. The primary machine learning API for Spark is now the DataFrame-based API in the `spark.ml` package. Using `spark.ml` with the DataFrame API is more versatile and flexible, and we can have the benefits provided by DataFrame, such as catalyst optimizer and `spark.mllib`, which is an RDD-based API that is expected to be removed in the future.

Machine learning is applicable to various data types, including text, images, structured data, and vectors. To support these data types under a unified dataset concept, Spark ML includes the Spark SQL DataFrame. It is easy to combine various algorithms in a single workflow or pipeline.

The following sections will give you a detailed view of a few key concepts in the Spark ML API.

The transformer function

This is something that can transform one DataFrame into another. For instance, an ML model can transform a DataFrame with features into a DataFrame with predictions. A transformer contains feature transformer and learned model. This uses the `transform()` method to transform one DataFrame into another. The code for this is given for your reference:

```
import org.apache.spark.ml.feature.Tokenizer

val df = spark.createDataFrame(Seq(  ("This is the Transformer", 1.0),
("Transformer is pipeline component", 0.0))).toDF( "text", "label") val
tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words") val
tokenizedDF = tokenizer.transform(df)
```

The estimator algorithm

An **estimator** is another algorithm that can produce a transformer by fitting on a DataFrame. For instance, a learning algorithm can train on a dataset and produce a model. This produces a transformer by learning an algorithm. It uses the `fit()` method to produce a transformer. For instance, the **Naïve Bayes** learning algorithm is an estimator that calls the `fit()` method and trains a Naïve Bayes model, which is a transformer. We will use the following code to train the model:

```
import org.apache.spark.ml.classification.NaiveBayes

val nb = new NaiveBayes().setModelType("multinomial")

val model = nb.fit(Training_DataDF)
```

Pipeline

`Pipeline` represents a sequence of stages, where every stage is a transformer or an estimator. All these stages run in an order and the dataset that is input is altered as it passes through every stage. For the stages of transformers, the `transform ()` method is used, while for the stages of estimators, the `fit()` method is used to create a transformer.

Every DataFrame that is output from one stage is input for the next stage. The pipeline is also an estimator. Therefore, it produces `PipelineModel` once the `fit()` method is run. `PipelineModel` is a transformer. `PipelineModel` contains the same number of stages as in the original pipeline. `PipelineModel` and pipelines make sure that the test and training data pass through similar feature-processing steps. For instance, consider a pipeline with three stages: Tokenizer, which will tokenize the sentence and convert it into a word with the use of `Tokenizer.transform()`; `HashingTF`, which is used to represent a string in a vector form as all ML algorithms understand only vectors and not strings and this uses the `HashingTF.transform()` method; and `NaiveBayes`, an estimator that is used for prediction.

We can save the model at `HDFSlocation` using the `save()` method, so in future we can load it using the `load` method and use it for prediction on the new dataset. This loaded model will work on the feature column of `newDataset`, and return the predicted column with this `newDataset` will also pass through all the stages of the pipeline:

```
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.feature.{HashingTF, Tokenizer}
import org.apache.spark.ml.classification.NaiveBayes
```

```
val df = spark.createDataFrame(Seq(
  ("This is the Transformer", 1.0),
  ("Transformer is pipeline component", 0.0)
)).toDF( "text", "label")

val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")

val
HashingTF=newHashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutp
utCol).setOutputCol(“features”)

val nb = new NaiveBayes().setModelType("multinomial")

val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, nb))
val model = pipeline.fit(df)
model.save("/HDFSlocation/Path/")
val loadModel = PipelineModel.load(("/HDFSlocation/Path/")

val PredictedData = loadModel.transform(newDataset)
```

Regression analysis

Regression analysis is a statistical modeling technique that is used for predicting or forecasting the occurrence of an event or the value of a continuous variable (dependent variable), based on the value of one or many independent variables. For example, when we want to drive from one place to another, there are numerous factors that affect the amount of time it will take to reach the destination, for example, the start time, distance, real-time traffic conditions, construction activities on the road, and weather conditions. All these factors impact the actual time it will take to reach the destination. As you can imagine, some factors have more impact than the others on the value of the dependent variable. In regression analysis, we mathematically sort out which variables impact the outcome, leading us to understand which factors matter most, which ones do not impact the outcome in a meaningful way, how these factors relate to each other, and mathematically, the quantified impact of variable factors on the outcome.

Various regression techniques that are used depend on the number and distribution of values of independent variables. These variables also derive the shape of the curve that represents predictor function. There are various regression techniques, and we will learn about them in detail in the following sections.

Linear regression

With linear regression, we model the relationship between the dependent variable, y, and an explanatory variable or independent variable, x. When there is one independent variable, it is called **simple linear regression**, and in the case of multiple independent variables, the regression is called **multiple linear regression**. The predictor function in the case of linear regression is a straight line (refer to figure 4 for an illustration). The regression line defines the relationship between x and y. When the value of y increases when x increases, there is a positive relationship between x and y. Similarly, when x and y are inversely proportional, there is a negative relationship between x and y. The line should be plotted on x and y dimensions to minimize the difference between the predicted value and the actual value, called prediction error.

In its simplest form, the linear regression equation is:

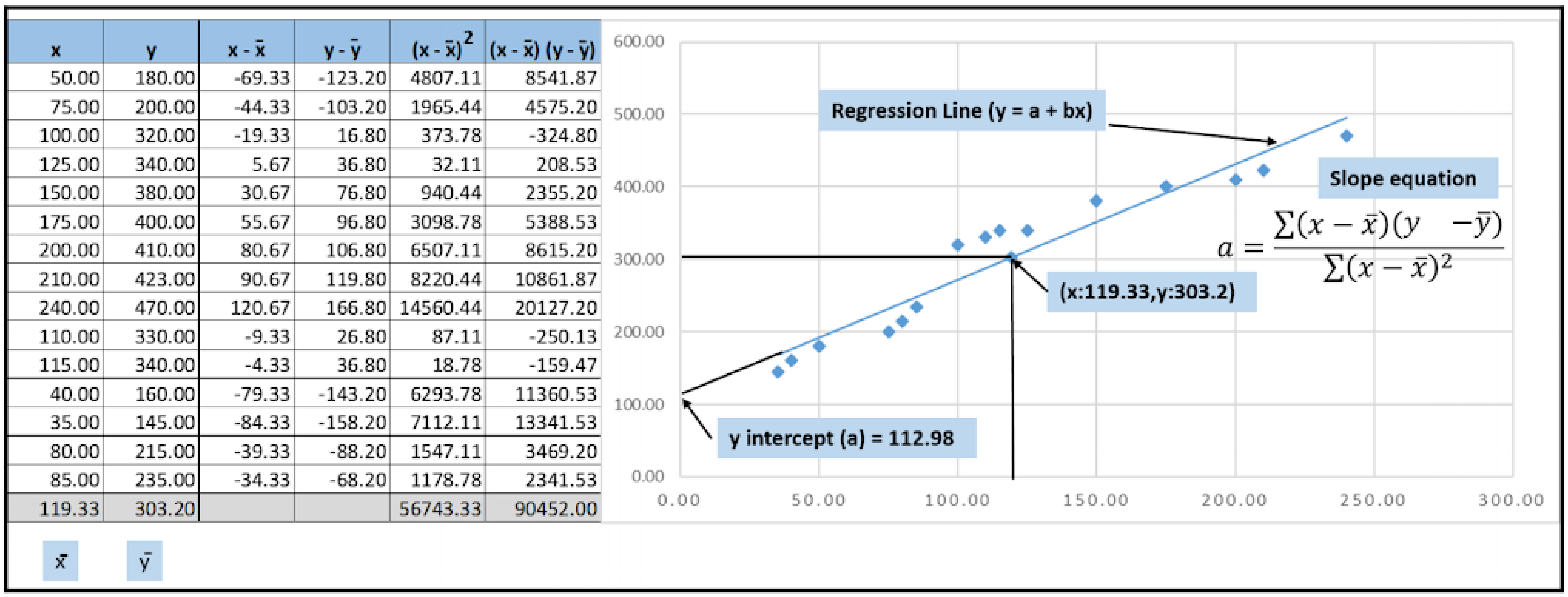
$$y = a + bx$$

This is the equation of a straight line, where y is the value of dependent variable, a is the y intercept (the value of y where the regression line meets the y axis), and b is the slope of the line. Let's consider the least square method in which we can derive the regression line with minimum prediction error.

Least square method

Let's consider the same training data we referred to earlier in this chapter. We have values for the independent variable, x, and corresponding values for the dependent variable, y. These values are plotted on a two-dimensional scatter plot. The goal is to draw a regression line through the training data so as to minimize the error of our predictions. The linear regression line with minimum error always passes the mean intercept for x and y values.

The following figure shows the least square method:



x	y	x - x̄	y - ȳ	(x - x̄)^2	(x - x̄) (y - ȳ)
50.00	180.00	-69.33	-123.20	4807.11	8541.87
75.00	200.00	-44.33	-103.20	1965.44	4575.20
100.00	320.00	-19.33	16.80	373.78	-324.80
125.00	340.00	5.67	36.80	32.11	208.53
150.00	380.00	30.67	76.80	940.44	2355.20
175.00	400.00	55.67	96.80	3098.78	5388.53
200.00	410.00	80.67	106.80	6507.11	8615.20
210.00	423.00	90.67	119.80	8220.44	10861.87
240.00	470.00	120.67	166.80	14560.44	20127.20
110.00	330.00	-9.33	26.80	87.11	-250.13
115.00	340.00	-4.33	36.80	18.78	-159.47
40.00	160.00	-79.33	-143.20	6293.78	11360.53
35.00	145.00	-84.33	-158.20	7112.11	13341.53
80.00	215.00	-39.33	-88.20	1547.11	3469.20
85.00	235.00	-34.33	-68.20	1178.78	2341.53
119.33	303.20			56743.33	90452.00
x̄	ȳ				

Figure 3.8 Least square method

The formula for calculating the *y* intercept is as follows:

$$a = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sum(x-\bar{x})^2}$$

The least square method calculates the *y* intercept and the slope of the line with the following steps:

1. Calculate the mean of all the *x* values (119.33).
2. Calculate the mean of all the *y* values (303.20).
3. Calculate difference from the mean for all the *x* and *y* values.
4. Calculate the square of mean difference for all the *x* values.
5. Multiply the mean difference of *x* by the mean difference of *y* for all the combinations of *x* and *y*.
6. Calculate the sum squares of all the mean differences of the *x* values *(*56743.33*)*.
7. Calculate the sum of mean difference products of the *x* and *y* values (90452.00).
8. The slope of the regression line is obtained by dividing the sum of the mean difference products of *x* and *y* by the sum of the squares of all the mean differences of the *x* values *(90452.00 / 56743.33 = 1.594)*. In this training data, since there is direct proportion between the *x* and *y* values, the slope is positive. This is the value for *b* in our equation.