Machine Learning Basics

The idea of making intelligent, sentient, and self-aware machines is not something that suddenly came into existence in the last few years. In fact a lot of lore from Greek mythology talks about intelligent machines and inventions having self-awareness and intelligence of their own. The origins and the evolution of the computer have been really revolutionary over a period of several centuries, starting from the basic Abacus and its descendant the slide rule in the 17th Century to the first general purpose computer designed by Charles Babbage in the 1800s. In fact, once computers started evolving with the invention of the Analytical Engine by Babbage and the first computer program, which was written by Ada Lovelace in 1842, people started wondering and contemplating that could there be a time when computers or machines truly become intelligent and start thinking for themselves. In fact, the renowned computer scientist, Alan Turing, was highly influential in the development of theoretical computer science, algorithms, and formal language and addressed concepts like artificial intelligence and Machine Learning as early as the 1950s. This brief insight into the evolution of making machines learn is just to give you an idea of something that has been out there since centuries but has recently started gaining a lot of attention and focus.
With faster computers, better processing, better computation power, and more storage, we have been living in what I like to call, the "age of information" or the "age of data". Day in and day out, we deal with managing *Big Data* and building intelligent systems by using concepts and methodologies from *Data Science*, *Artificial Intelligence*, *Data Mining*, and *Machine Learning*. Of course, most of you must have heard many of the terms I just mentioned and come across sayings like *"data is the new oil"*. The main challenge that businesses and organizations have embarked on in the last decade is to use approaches to try to make sense of all the data that they have and use valuable information and insights from it in order to make better decisions. Indeed with great advancements in technology, including availability of cheap and massive computing, hardware (including GPUs) and storage, we have seen a thriving ecosystem built around domains like Artificial Intelligence, Machine Learning, and most recently Deep Learning. Researchers, developers, data scientists, and engineers are working continuously round the clock to research and build tools, frameworks, algorithms, techniques, and methodologies to build intelligent models and systems that can predict events, automate tasks, perform complex analyses, detect anomalies, self-heal failures, and even understand and respond to human inputs.

The Need for Machine Learning

Human beings are perhaps the most advanced and intelligent lifeform on this planet at the moment. We can think, reason, build, evaluate, and solve complex problems. The human brain is still something we ourselves haven't figured out completely and hence artificial intelligence is still something that's not surpassed human intelligence in several aspects. Thus you might get a pressing question in mind as to why do we really need Machine Learning? What is the need to go out of our way to spend time and effort to make machines learn and be intelligent? The answer can be summed up in a simple sentence, "To make data-driven decisions at scale". We will dive into details to explain this sentence in the following sections.

Making Data-Driven Decisions

Getting key information or insights from data is the key reason businesses and organizations invest heavily in a good workforce as well as newer paradigms and domains like Machine Learning and artificial intelligence. The idea of data-driven decisions is not new. Fields like operations research, statistics, and management information systems have existed for decades and attempt to bring efficiency to any business or organization by using data and analytics to make data-driven decisions. The art and science of leveraging your data to get actionable insights and make better decisions is known as making data-driven decisions. Of course, this is easier said than done because rarely can we directly use raw data to make any insightful decisions. Another important aspect of this problem is that often we use the power of reasoning or intuition to try to make decisions based on what we have learned over a period of time and on the job. Our brain is an extremely powerful device that helps us do so. Consider problems like understanding what your fellow colleagues or friends are speaking, recognizing people in images, deciding whether to approve or reject a business transaction, and so on. While we can solve these problems almost involuntary, can you explain someone the process of how you solved each of these problems? Maybe to some extent, but after a while, it would be like, "Hey! My brain did most of the thinking for me!" This is exactly why it is difficult to make machines learn to solve these problems like regular computational programs like computing loan interest or tax rebates. Solutions to problems that cannot be programmed inherently need a different approach where we use the data itself to drive decisions instead of using programmable logic, rules, or code to make these decisions. We discuss this further in future sections.

Efficiency and Scale

While getting insights and making decisions driven by data are of paramount importance, it also needs to be done with efficiency and at scale. The key idea of using techniques from Machine Learning or artificial intelligence is to automate processes or tasks by learning specific patterns from the data. We all want computers or machines to tell us when a stock might rise or fall, whether an image is of a computer or a television, whether our product placement and offers are the best, determine shopping price trends, detect failures or outages before they occur, and the list just goes on! While human intelligence and expertise is something that we definitely can't do without, we need to solve real-world problems at huge scale with efficiency.

A REAL-WORLD PROBLEM AT SCALE

Consider the following real-world problem. You are the manager of a world-class infrastructure team for the DSS Company that provides Data Science services in the form of cloud based infrastructure and analytical platforms for other businesses and consumers. Being a provider of services and infrastructure, you want your infrastructure to be top-notch and robust to failures and outages. Considering you are starting out of St. Louis in a small office, you have a good grasp over monitoring all your network devices including routers, switches, firewalls, and load balancers regularly with your team of 10 experienced employees. Soon you make a breakthrough with providing cloud based Deep Learning services and GPUs for development and earn huge profits. However, now you keep getting more and more customers. The time has come for expanding your base to offices in San Francisco, New York, and Boston. You have a huge connected infrastructure now with hundreds of network devices in each building! How will you manage your infrastructure at scale now? Do you hire more manpower for each office or do you try to leverage Machine Learning to deal with tasks like outage prediction, auto-recovery, and device monitoring? Think about this for some time from both an engineer as well as a manager's point of view.

Traditional Programming Paradigm

Computers, while being extremely sophisticated and complex devices, are just another version of our well known idiot box, the television! "How can that be?" is a very valid question at this point. Let's consider a television or even one of the so-called smart TVs, which are available these days. In theory as well as in practice, the TV will do whatever you program it to do. It will show you the channels you want to see, record the shows you want to view later on, and play the applications you want to play! The computer has been doing the exact same thing but in a different way. Traditional programming paradigms basically involve the user or programmer to write a set of instructions or operations using code that makes the computer perform specific computations on data to give the desired results. Figure 1-1 depicts a typical workflow for traditional programming paradigms.

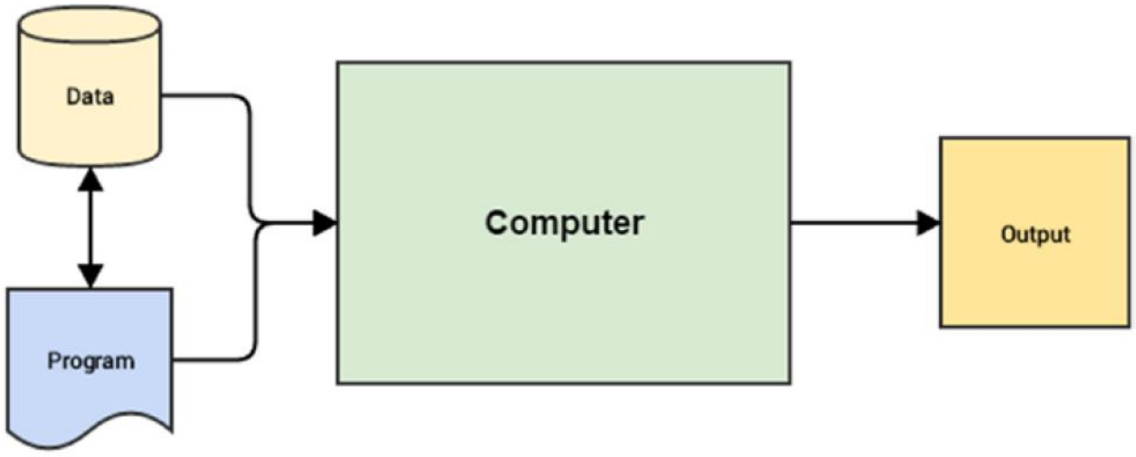

***Figure 1-1.** Traditional programming paradigm*

From Figure 1-1, you can get the idea that the core inputs that are given to the computer are data and one or more programs that are basically code written with the help of a programming language, such as high-level languages like Java, Python, or low-level like C or even Assembly. Programs enable computers to work on data, perform computations, and generate output. A task that can be performed really well with traditional programming paradigms is computing your annual tax.

Now, let's think about the real-world infrastructure problem we discussed in the previous section for DSS Company. Do you think a traditional programming approach might be able to solve this problem? Well, it could to some extent. We might be able to tap in to the device data and event streams and logs and access various device attributes like usage levels, signal strength, incoming and outgoing connections, memory and processor usage levels, error logs and events, and so on. We could then use the domain knowledge of our network and infrastructure experts in our teams and set up some event monitoring systems based on specific decisions and rules based on these data attributes. This would give us what we could call as a rule-based reactive analytical solution where we can monitor devices, observe if any specific anomalies or outages occur, and then take necessary action to quickly resolve any potential issues. We might also have to hire some support and operations staff to continuously monitor and resolve issues as needed. However, there is still a pressing problem of trying to prevent as many outages or issues as possible before they actually take place. Can Machine Learning help us in some way?

Why Machine Learning?

We will now address the question that started this discussion of why we need Machine Learning. Considering what you have learned so far, while the traditional programming paradigm is quite good and human intelligence and domain expertise is definitely an important factor in making data-driven decisions, we need Machine Learning to make faster and better decisions. The Machine Learning paradigm tries to take into account data and expected outputs or results if any and uses the computer to build the program, which is also known as a model. This program or model can then be used in the future to make necessary decisions and give expected outputs from new inputs. Figure 1-2 shows how the Machine Learning paradigm is similar yet different from traditional programming paradigms.

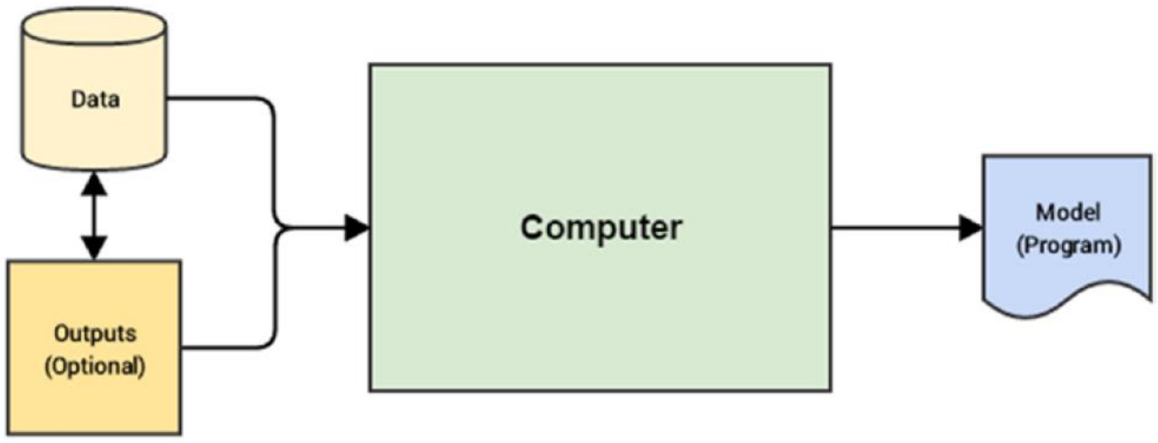

***Figure 1-2.** Machine Learning paradigm*

Figure 1-2 reinforces the fact that in the Machine Learning paradigm, the machine, in this context the computer, tries to use input data and expected outputs to try to learn inherent patterns in the data that would ultimately help in building a model analogous to a computer program, which would help in making data-driven decisions in the future (predict or tell us the output) for new input data points by using the learned knowledge from previous data points (its knowledge or experience). You might start to see the benefit in this. We would not need hand-coded rules, complex flowcharts, case and if-then conditions, and other criteria that are typically used to build any decision making system or a decision support system. The basic idea is to use Machine Learning to make insightful decisions.

This will be clearer once we discuss our real-world problem of managing infrastructure for DSS Company. In the traditional programming approach, we talked about hiring new staff, setting up rule-based monitoring systems, and so on. If we were to use a Machine Learning paradigm shift here, we could go about solving the problem using the following steps.

• Leverage device data and logs and make sure we have enough historical data in some data store (database, logs, or flat files)

• Decide key data attributes that could be useful for building a model. This could be device usage, logs, memory, processor, connections, line strength, links, and so on.

• Observe and capture device attributes and their behavior over various time periods that would include normal device behavior and anomalous device behavior or outages. These outcomes would be your outputs and device data would be your inputs

• Feed these input and output pairs to any specific Machine Learning algorithm in your computer and build a model that learns inherent device patterns and observes the corresponding output or outcome

• Deploy this model such that for newer values of device attributes it can predict if a specific device is behaving normally or it might cause a potential outage

Thus once you are able to build a Machine Learning model, you can easily deploy it and build an intelligent system around it such that you can not only monitor devices reactively but you would be able to proactively identify potential problems and even fix them before any issues crop up. Imagine building self-heal or auto-heal systems coupled with round the clock device monitoring. The possibilities are indeed endless and you will not have to keep on hiring new staff every time you expand your office or buy new infrastructure.

Of course, the workflow discussed earlier with the series of steps needed for building a Machine Learning model is much more complex than how it has been portrayed, but again this is just to emphasize and make you think more conceptually rather than technically of how the paradigm has shifted in case of Machine Learning processes and you need to change your thinking too from the traditional based approaches toward being more data-driven. The beauty of Machine Learning is that it is never domain constrained and you can use techniques to solve problems spanning multiple domains, businesses, and industries. Also, as depicted in Figure 1-2, you always do not need output data points to build a model; sometimes input data is sufficient (or rather output data might not be present) for techniques more suited toward unsupervised learning .A simple example is trying to determine customer shopping patterns by looking at the grocery items they typically buy together in a store based on past transactional data. In the next section, we take a deeper dive toward understanding

Understanding Machine Learning

By now, you have seen how a typical real-world problem suitable to solve using Machine Learning might look like. Besides this, you have also got a good grasp over the basics of traditional programming and Machine Learning paradigms. In this section, we discuss Machine Learning in more detail. To be more specific, we will look at Machine Learning from a conceptual as well as a domain-specific standpoint. Machine Learning came into prominence perhaps in the 1990s when researchers and scientists started giving it more prominence as a sub-field of Artificial Intelligence (AI) such that techniques borrow concepts from AI, probability, and statistics, which perform far better compared to using fixed rule-based models requiring a lot of manual time and effort. Of course, as we have pointed out earlier, Machine Learning didn't just come out of nowhere in the 1990s. It is a multi-disciplinary field that has gradually evolved over time and is still evolving as we speak.
A brief mention of history of evolution would be really helpful to get an idea of the various concepts and techniques that have been involved in the development of Machine Learning and AI. You could say that it started off in the late 1700s and the early 1800s when the first works of research were published which basically talked about the Bayes' Theorem. In fact Thomas Bayes' major work, "An Essay Towards Solving a Problem in the Doctrine of Chances," was published in 1763. Besides this, a lot of research and discovery was done during this time in the field of probability and mathematics. This paved the way for more ground breaking research and inventions in the 20th Century, which included Markov Chains by Andrey Markov in the early 1900s, proposition of a learning system by Alan Turing, and the invention of the very famous perceptron by Frank Rosenblatt in the 1950s. Many of you might know that neural networks had several highs and lows since the 1950s and they finally came back to prominence in the 1980s with the discovery of backpropagation (thanks to Rumelhart, Hinton, and Williams!) and several other inventions, including Hopfield networks, neocognition, convolutional and recurrent neural networks, and Q-learning. Of course, rapid strides of evolution started taking place in Machine Learning too since the 1990s with the discovery of random forests, support vector machines, long short-term memory networks (LSTMs), and development and release of frameworks in both machine and Deep Learning including torch, theano, tensorflow, scikit-learn, and so on. We also saw the rise of intelligent systems including IBM Watson, DeepFace, and AlphaGo. Indeed the journey has been quite a roller coaster ride and there's still miles to go in this journey. Take a moment and reflect on this evolutional journey and let's talk about the purpose of this journey. Why and when should we really make machines learn?

Why Make Machines Learn?

We have discussed a fair bit about why we need Machine Learning in a previous section when we address the issue of trying to leverage data to make data-driven decisions at scale using learning algorithms without focusing too much on manual efforts and fixed rule-based systems. In this section, we discuss in more detail why and when should we make machines learn. There are several real-world tasks and problems that humans, businesses, and organizations try to solve day in and day out for our benefit. There are several scenarios when it might be beneficial to make machines learn and some of them are mentioned as follows.

Lack of sufficient human expertise in a domain (e.g., simulating navigations in unknown territories or even spatial planets).
• Scenarios and behavior can keep changing over time (e.g., availability of infrastructure in an organization, network connectivity, and so on).
• Humans have sufficient expertise in the domain but it is extremely difficult to formally explain or translate this expertise into computational tasks (e.g., speech recognition, translation, scene recognition, cognitive tasks, and so on).
• Addressing domain specific problems at scale with huge volumes of data with too many complex conditions and constraints.
The previously mentioned scenarios are just several examples where making machines learn would be more effective than investing time, effort, and money in trying to build sub-par intelligent systems that might be limited in scope, coverage, performance, and intelligence. We as humans and domain experts already have enough knowledge about the world and our respective domains, which can be objective, subjective,

and sometimes even intuitive. With the availability of large volumes of historical data, we can leverage the Machine Learning paradigm to make machines perform specific tasks by gaining enough experience by observing patterns in data over a period of time and then use this experience in solving tasks in the future with minimal manual intervention. The core idea remains to make machines solve tasks that can be easily defined intuitively and almost involuntarily but extremely hard to define formally.

Formal Definition

We are now ready to define Machine Learning formally. You may have come across multiple definitions of Machine Learning by now which include, techniques to make machines intelligent, automation on steroids, automating the task of automation itself, the sexiest job of the 21st century, making computers learn by themselves and countless others! While all of them are good quotes and true to certain extents, the best way to define Machine Learning would be to start from the basics of Machine Learning as defined by renowned professor Tom Mitchell in 1997.

The idea of Machine Learning is that there will be some learning algorithm that will help the machine learn from data. Professor Mitchell defined it as follows.

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

While this definition might seem daunting at first, I ask you go read through it a couple of times slowly focusing on the three parameters—T, P, and E—which are the main components of any learning algorithm, as depicted in Figure 1-3.

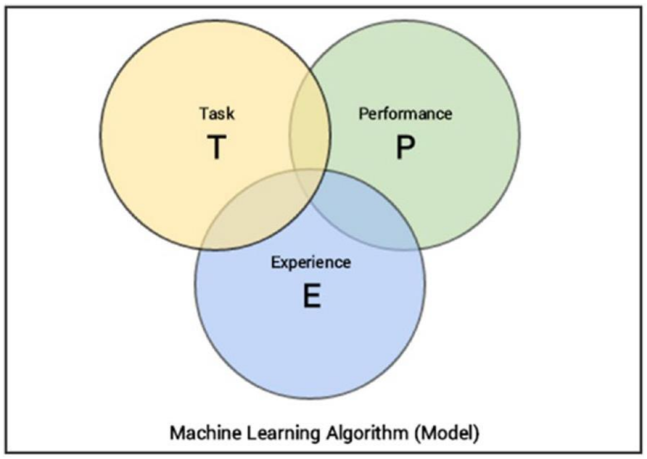

Figure 1-3. *Defining the components of a learning algorithm*

We can simplify the definition as follows. Machine Learning is a field that consists of learning algorithms that:

• Improve their performance *P*

• At executing some task *T*

• Over time with experience *E*

While we discuss at length each of these entities in the following sections, we will not spend time in formally or mathematically defining each of these entities since the scope of the book is more toward applied or practical Machine Learning. If you consider our real-world problem from earlier, one of the tasks *T* could be predicting outages for our infrastructure; experience *E* would be what our Machine Learning model would gain over time by observing patterns from various device data attributes; and the performance of the model *P* could be measured in various ways like how accurately the model predicts outages.

Defining the Task, T

We had discussed briefly in the previous section about the task, T, which can be defined in a two-fold approach. From a problem standpoint, the task, T, is basically the real-world problem to be solved at hand, which could be anything from finding the best marketing or product mix to predicting infrastructure failures. In the Machine Learning world, it is best if you can define the task as concretely as possible such that you talk about what the exact problem is which you are planning to solve and how you could define or formulate the problem into a specific Machine Learning task.

Machine Learning based tasks are difficult to solve by conventional and traditional programming approaches. A task, *T,* can usually be defined as a Machine Learning task based on the process or workflow that the system should follow to operate on data points or samples. Typically a data sample or point will consist of multiple data attributes (also called features in Machine Learning lingo) just like the various device parameters we mentioned in our problem for DSS Company earlier. A typical data point can be denoted by a vector (Python list) such that each element in the vector is for a specific data feature or attribute.

"Feature Engineering and Selection".

Coming back to the typical tasks that could be classified as Machine Learning tasks, the following list describes some popular tasks.

• **Classification or categorization**: This typically encompasses the list of problems or tasks where the machine has to take in data points or samples and assign a specific class or category to each sample. A simple example would be classifying animal images into dogs, cats, and zebras.

• **Regression**: These types of tasks usually involve performing a prediction such that a real numerical value is the output instead of a class or category for an input data point. The best way to understand a regression task would be to take the case of a real-world problem of predicting housing prices considering the plot area, number of floors, bathrooms, bedrooms, and kitchen as input attributes for each data point.

• **Anomaly detection**: These tasks involve the machine going over event logs, transaction logs, and other data points such that it can find anomalous or unusual patterns or events that are different from the normal behavior. Examples for this include trying to find denial of service attacks from logs, indications of fraud, and so on.

• **Structured annotation**: This usually involves performing some analysis on input data points and adding structured metadata as annotations to the original data that depict extra information and relationships among the data elements. Simple examples would be annotating text with their parts of speech, named entities, grammar, and sentiment. Annotations can also be done for images like assigning specific categories to image pixels, annotate specific areas of images based on their type, location, and so on.

• **Translation**: Automated machine translation tasks are typically of the nature such that if you have input data samples belonging to a specific language, you translate it into output having another desired language. Natural language based translation is definitely a huge area dealing with a lot of text data.

• **Clustering or grouping**: Clusters or groups are usually formed from input data samples by making the machine learn or observe inherent latent patterns, relationships and similarities among the input data points themselves. Usually there is a lack of pre-labeled or pre-annotated data for these tasks hence they form a part of unsupervised Machine Learning (which we will discuss later on). Examples would be grouping similar products, events and entities.

• **Transcriptions**: These tasks usually entail various representations of data that are usually continuous and unstructured and converting them into more structured and discrete data elements. Examples include speech to text, optical character recognition, images to text, and so on.

This should give you a good idea of typical tasks that are often solved using Machine Learning, but this list is definitely not an exhaustive one as the limits of tasks are indeed endless and more are being discovered with extensive research over time.

Defining the Experience, E

At this point, you know that any learning algorithm typically needs data to learn over time and perform a specific task, which we named as T. The process of consuming a dataset that consists of data samples or data points such that a learning algorithm or model learns inherent patterns is defined as the experience, *E* which is gained by the learning algorithm. Any experience that the algorithm gains is from data samples or data points and this can be at any point of time. You can feed it data samples in one go using historical data or even supply fresh data samples whenever they are acquired.

Thus, the idea of a model or algorithm gaining experience usually occurs as an iterative process, also known as training the model. You could think of the model to be an entity just like a human being which gains knowledge or experience through data points by observing and learning more and more about various attributes, relationships and patterns present in the data. Of course, there are various forms and ways of learning and gaining experience including supervised, unsupervised, and reinforcement learning but we will discuss learning methods in a future section. For now, take a step back and remember the analogy we drew that when a machine truly learns, it is based on data which is fed to it from time to time thus allowing it to gain experience and knowledge about the task to be solved, such that it can used this experience, E, to predict or solve the same task, T, in the future for previously unseen data points.

Defining the Performance, P

Let's say we have a Machine Learning algorithm that is supposed to perform a task, T, and is gaining experience, E, with data points over a period of time. But how do we know if it's performing well or behaving the way it is supposed to behave? This is where the performance, *P,* of the model comes into the picture.

The performance, P, is usually a quantitative measure or metric that's used to see how well the algorithm or model is performing the task, T, with experience, E. While performance metrics are usually standard metrics that have been established after years of research and development, each metric is usually computed specific to the task, T, which we are trying to solve at any given point of time.

Typical performance measures include accuracy, precision, recall, F1 score, sensitivity, specificity, error rate, misclassification rate, and many more. Performance measures are usually evaluated on training data samples (used by the algorithm to gain experience, E) as well as data samples which it has not seen or learned from before, which are usually known as validation and test data samples. The idea behind this is to generalize the algorithm so that it doesn't become too biased only on the training data points and performs well in the future on newer data points. More on training, validation, and test data will be discussed when we talk about model building and validation.

While solving any Machine Learning problem, most of the times, the choice of performance measure, P, is either accuracy, F1 score, precision, and recall. While this is true in most scenarios, you should always remember that sometimes it is difficult to choose performance measures that will accurately be able to give us an idea of how well the algorithm is performing based on the actual behavior or outcome which is expected from it. A simple example would be that sometimes we would want to penalize misclassification or false positives more than correct hits or predictions. In such a scenario, we might need to use a modified cost function or priors such that we give a scope to sacrifice hit rate or overall accuracy for more accurate predictions with lesser false positives. A real-world example would be an intelligent system that predicts if we should give a loan to a customer. It's better to build the system in such a way that it is more cautious against giving a loan than denying one. The simple reason is because one big mistake of giving a loan to a potential defaulter can lead to huge losses as compared to denying several smaller loans to potential customers. To conclude, you need to take into account all parameters and attributes involved in task, T, such that you can decide on the right performance measures, P, for your system.

A Multi-Disciplinary Field

We have formally introduced and defined Machine Learning in the previous section, which should give you a good idea about the main components involved with any learning algorithm. Let's now shift our perspective to Machine Learning as a domain and field. You might already know that Machine Learning is mostly considered to be a sub-field of artificial intelligence and even computer science from some perspectives. Machine Learning has concepts that have been derived and borrowed from multiple fields over a period of time since its inception, making it a true multi-disciplinary or inter-disciplinary field. Figure 1-4 should give you a good idea with regard to the major fields that overlap with Machine Learning based on concepts, methodologies, ideas, and techniques. An important point to remember here is that this is definitely not an exhaustive list of domains or fields but pretty much depicts the major fields associated in tandem with Machine Learning.

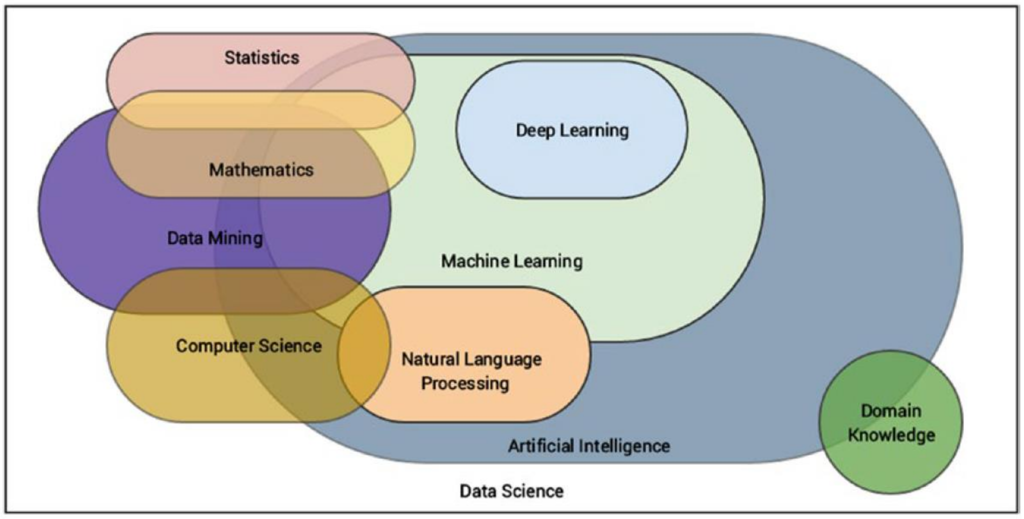

Figure 1-4. *Machine Learning: a true multi-disciplinary field*

The major domains or fields associated with Machine Learning include the following, as depicted in Figure 1-4. We will discuss each of these fields in upcoming sections.

- Artificial intelligence
- Natural language processing
- Data mining
- Mathematics
- Statistics
- Computer science
- Deep Learning
- Data Science

You could say that *Data Science* is like a broad inter-disciplinary field spanning across all the other fields which are sub-fields inside it. Of course this is just a simple generalization and doesn't strictly indicate that it is inclusive of all other other fields as a superset, but rather borrows important concepts and methodologies from them. The basic idea of Data Science is once again processes, methodologies, and techniques to extract information from data and domain knowledge. This is a big part of what we discuss in an upcoming section when we talk about Data Science in further details.

Coming back to Machine Learning, ideas of pattern recognition and basic *data mining* methodologies like *knowledge discovery of databases* (KDD) came into existence when relational databases were very prominent. These areas focus more on the ability and technique to mine for information from large datasets, such that you can get patterns, knowledge, and insights of interest. Of course, KDD is a whole process by itself that includes data acquisition, storage, warehousing, processing, and analysis. Machine Learning borrows concepts that are more concerned with the analysis phase, although you do need to go through the other steps to reach to the final stage. *Data mining* is again a interdisciplinary or multi-disciplinary field and borrows concepts from computer science, mathematics, and statistics. The consequence of this is the fact that computational statistics form an important part of most Machine Learning algorithms and techniques.

Artificial intelligence (AI) is the superset consisting of Machine Learning as one of its specialized areas. The basic idea of AI is the study and development of intelligence as exhibited by machines based on their perception of their environment, input parameters and attributes and their response such that they can perform desired tasks based on expectations. AI itself is a truly massive field which is itself inter-disciplinary. It draws on concepts from mathematics, statistics, computer science, cognitive sciences, linguistics, neuroscience, and many more. Machine Learning is more concerned with algorithms and techniques that can be used to understand data, build representations, and perform tasks such as predictions. Another major sub-field under AI related to Machine Learning is *natural language processing* (NLP) which borrows concepts heavily from computational linguistics and computer science. *Text Analytics* is a prominent field

today among analysts and data scientists to extract, process and understand natural human language. Combine NLP with AI and Machine Learning and you get chatbots, machine translators, and virtual personal assistants, which are indeed the future of innovation and technology!
Coming to *Deep Learning*, it is a subfield of Machine Learning itself which deals more with techniques related to representational learning such that it improves with more and more data by gaining more experience. It follows a layered and hierarchical approach such that it tries to represent the given input attributes and its current surroundings, using a nested layered hierarchy of concept representations such that, each complex layer is built from another layer of simpler concepts. Neural networks are something which is heavily utilized by Deep Learning and we will look into Deep Learning in a bit more detail in a future section and solve some real-world problems later on in this book.
Computer science is pretty much the foundation for most of these domains dealing with study, development, engineering, and programming of computers. Hence we won't be expanding too much on this but you should definitely remember the importance of computer science for Machine Learning to exist and be easily applied to solve real-world problems. This should give you a good idea about the broad landscape of the multi-disciplinary field of Machine Learning and how it is connected across multiple related and overlapping fields. We will discuss some of these fields in more detail in upcoming sections and cover some basic concepts in each of these fields wherever necessary.
Let's look at some core fundamentals of Computer Science in the following section.

Computer Science

The field of computer science (CS) can be defined as the study of the science of understanding computers. This involves study, research, development, engineering, and experimentation of areas dealing with understanding, designing, building, and using computers. This also involves extensive design and development of algorithms and programs that can be used to make the computer perform computations and tasks as desired. There are mainly two major areas or fields under computer science, as follows.

- Theoretical computer science
- Applied or practical computer science

The two major areas under computer science span across multiple fields and domains wherein each field forms a part or a sub-field of computer science. The main essence of computer science includes formal languages, automata and theory of computation, algorithms, data structures, computer design and architecture, programming languages, and software engineering principles.

Theoretical Computer Science

Theoretical computer science is the study of theory and logic that tries to explain the principles and processes behind computation. This involves understanding the theory of computation which talks about how computation can be used efficiently to solve problems. Theory of computation includes the study of formal languages, automata, and understanding complexities involved in computations and algorithms. Information and coding theory is another major field under theoretical CS that has given us domains like signal processing, cryptography, and data compression. Principles of programming languages and their analysis is another important aspect that talks about features, design, analysis, and implementations of various programming languages and how compilers and interpreters work in understanding these languages. Last but never the least, data structures and algorithms are the two fundamental pillars of theoretical CS used extensively in computational programs and functions.

Practical Computer Science

Practical computer science also known as applied computer science is more about tools, methodologies, and processes that deal with applying concepts and principles from computer science in the real world to solve practical day-to-day problems. This includes emerging sub-fields like artificial intelligence, Machine Learning, computer vision, Deep Learning, natural language processing, data mining, and robotics and they try to solve complex real-world problems based on multiple constraints and parameters and try to emulate tasks that require considerable human intelligence and experience. Besides these, we also have well established fields, including computer architecture, operating systems, digital logic and design, distributed computing, computer networks, security, databases, and software engineering.

Important Concepts

These are several concepts from computer science that you should know and remember since they would be useful as foundational concepts to understand. It's not an
exhaustive list but should pretty much cover enough to get started.

Algorithms

An *algorithm* can be described as a sequence of steps, operations, computations, or functions that can be executed to carry out a specific task. They are basically methods to describe and represent a computer program formally through a series of operations, which are often described using plain natural language, mathematical symbols, and diagrams. Typically, flowcharts, pseudocode, and natural language are used extensively to represent algorithms. An algorithm can be as simple as adding two numbers and as complex as computing the inverse of a matrix.

Programming Languages

A *programming language* is a language that has its own set of symbols, words, tokens, and operators having their own significance and meaning. Thus syntax and semantics combine to form a formal language in itself. This language can be used to write computer programs, which are basically real-world implementations of algorithms that can be used to specify specific instructions to the computer such that it carries our necessary computation and operations. Programming languages can be low level like C and Assembly or high level languages like Java and Python.

Code

This is basically source code that forms the foundation of computer programs. Code is written using programming languages and consists of a collection of computer statements and instructions to make the computer perform specific desired tasks. Code helps convert algorithms into programs using programming languages. We will be using Python to implement most of our real-world Machine Learning solutions.

Data Structures

Data structures are specialized structures that are used to manage data. Basically they are real-world implementations for abstract data type specifications that can be used to store, retrieve, manage, and operate on data efficiently. There is a whole suite of data structures like arrays, lists, tuples, records, structures, unions, classes, and many more. We will be using Python data structures like lists, arrays, dataframes, and dictionaries extensively to operate on real-world data!

Data Science

The field of *Data Science* is a very diverse, inter-disciplinary field which encompasses multiple fields that we depicted in Figure 1-4. Data Science basically deals with principles, methodologies, processes, tools, and techniques to gather knowledge or information from data (structured as well as unstructured). Data Science is more of a compilation of processes, techniques, and methodologies to foster a data-driven decision based culture. In fact Drew Conway's "Data Science Venn Diagram," depicted in Figure 1-5, shows the core components and essence of Data Science, which in fact went viral and became insanely popular!

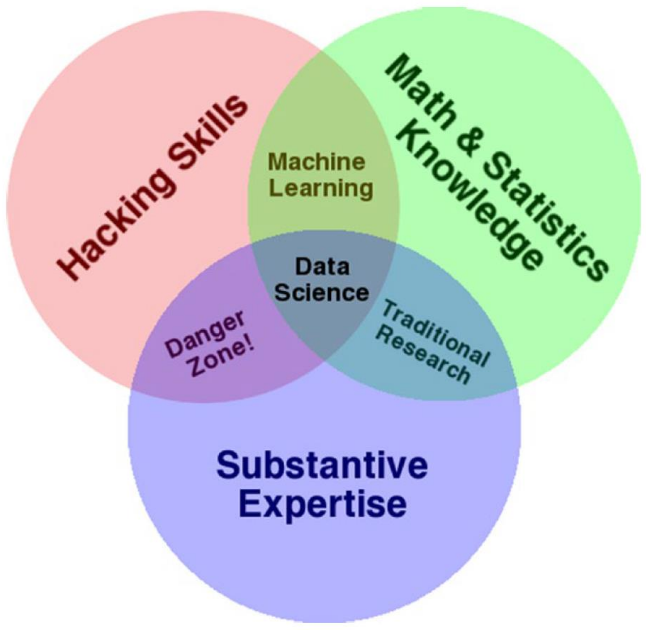

Figure 1-5. *Drew Conway's Data Science Venn diagram*

Figure 1-5 is quite intuitive and easy to interpret. Basically there are three major components and Data Science sits at the intersection of them. *Math and statistics knowledge* is all about applying various computational and quantitative math and statistical based techniques to extract insights from data. *Hacking skills* basically indicate the capability of handling, processing, manipulating and wrangling data into easy to understand and analyzable formats. *Substantive expertise* is basically the actual real-world domain expertise which is extremely important when you are solving a problem because you need to know about various factors, attributes, constraints, and knowledge related to the domain besides your expertise in data and algorithms.

Thus Drew rightly points out that Machine Learning is a combination of expertise on data hacking skills, math, and statistical learning methods and for Data Science, you need some level of domain expertise and knowledge along with Machine Learning. You can check out Drew's personal insights in his article at http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram, where talks all about the Data Science Venn diagram. Besides this, we also have Brendan Tierney, who talks about the true nature of Data Science being a multi-disciplinary field with his own depiction, as shown in Figure 1-6.

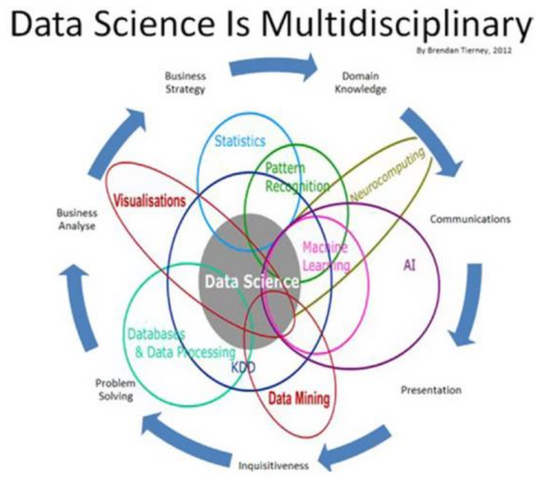

Figure 1-6. *Brendan Tierney's depiction of Data Science as a true multi-disciplinary field*

Mathematics

The field of mathematics deals with numbers, logic, and formal systems. The best definition of mathematics was coined by Aristotle as “The science of quantity”. The scope of mathematics as a scientific field is huge spanning across areas including algebra, trigonometry, calculus, geometry, and number theory just to name a few major fields. Linear algebra and probability are two major sub-fields under mathematics that are used extensively in Machine Learning and we will be covering a few important concepts from them in this section. Our major focus will always be on practical Machine Learning, and applied mathematics is an important aspect for the same. Linear algebra deals with mathematical objects and structures like vectors, matrices, lines, planes, hyperplanes, and vector spaces. The theory of probability is a mathematical field and framework used for studying and quantifying events of chance and uncertainty and deriving theorems and axioms from the same. These laws and axioms help us in reasoning, understanding, and quantifying uncertainty and its effects in any real-world system or scenario, which helps us in building our Machine Learning models by leveraging this framework.

Important Concepts

In this section, we discuss some key terms and concepts from applied mathematics, namely linear algebra and probability theory. These concepts are widely used across Machine Learning and form some of the foundational structures and principles across Machine Learning algorithms, models, and processes.

Scalar

A *scalar* usually denotes a single number as opposed to a collection of numbers. A simple example might be $x = 5$ or $x \in R$, where x is the scalar element pointing to a single number or a real-valued single number.

Vector

A *vector* is defined as a structure that holds an array of numbers which are arranged in order. This basically means the order or sequence of numbers in the collection is important. Vectors can be mathematically denoted as $x = [x_1, x_2, \ldots, x_n]$, which basically tells us that x is a one-dimensional vector having n elements in the array. Each element can be referred to using an array index determining its position in the vector. The following snippet shows us how we can represent simple vectors in Python.

```
In [1]: x = [1, 2, 3, 4, 5]
   ...: x
Out[1]: [1, 2, 3, 4, 5]
In [2]: import numpy as np
   ...: x = np.array([1, 2, 3, 4, 5])
   ...:
   ...: print(x)
   ...: print(type(x))
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Thus you can see that Python lists as well as numpy based arrays can be used to represent vectors. Each row in a dataset can act as a one-dimensional vector of n attributes, which can serve as inputs to learning algorithms.

Matrix

A *matrix* is a two-dimensional structure that basically holds numbers. It's also often referred to as a 2D array. Each element can be referred to using a row and column index as compared to a single vector index in case of vectors. Mathematically, you can depict a matrix as $M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$ such that M is a 3 x 3 matrix having three rows and three columns and each element is denoted by m_{rc} such that r denotes the row index and c denotes the column index. Matrices can be easily represented as list of lists in Python and we can leverage the numpy array structure as depicted in the following snippet.

```
In [3]: m = np.array([[1, 5, 2],
   ...:               [4, 7, 4],
   ...:               [2, 0, 9]])
```

```
In [4]: # view matrix
   ...: print(m)
[[1 5 2]
 [4 7 4]
 [2 0 9]]

In [5]: # view dimensions
   ...: print(m.shape)
(3, 3)
```

Thus you can see how we can easily leverage numpy arrays to represent matrices. You can think of a dataset with rows and columns as a matrix such that the data features or attributes are represented by columns and each row denotes a data sample. We will be using the same analogy later on in our analyses.
Of course, you can perform matrix operations like add, subtract, products, inverse, transpose, determinants, and many more. The following snippet shows some popular matrix operations.

```
In [9]: # matrix transpose
   ...: print('Matrix Transpose:\n', m.transpose(), '\n')
   ...:
   ...: # matrix determinant
   ...: print ('Matrix Determinant:', np.linalg.det(m), '\n')
   ...:
   ...: # matrix inverse
   ...: m_inv = np.linalg.inv(m)
   ...: print ('Matrix inverse:\n', m_inv, '\n')
   ...:
   ...: # identity matrix (result of matrix x matrix_inverse)
   ...: iden_m =  np.dot(m, m_inv)
   ...: iden_m = np.round(np.abs(iden_m), 0)
   ...: print ('Product of matrix and its inverse:\n', iden_m)
   ...:
Matrix Transpose:
 [[1 4 2]
  [5 7 0]
  [2 4 9]]

Matrix Determinant: -105.0

Matrix inverse:
 [[-0.6         0.42857143 -0.05714286]
  [ 0.26666667 -0.04761905 -0.03809524]
  [ 0.13333333 -0.0952381   0.12380952]]

Product of matrix and its inverse:
 [[ 1.  0.  0.]
  [ 0.  1.  0.]
  [ 0.  0.  1.]]
```