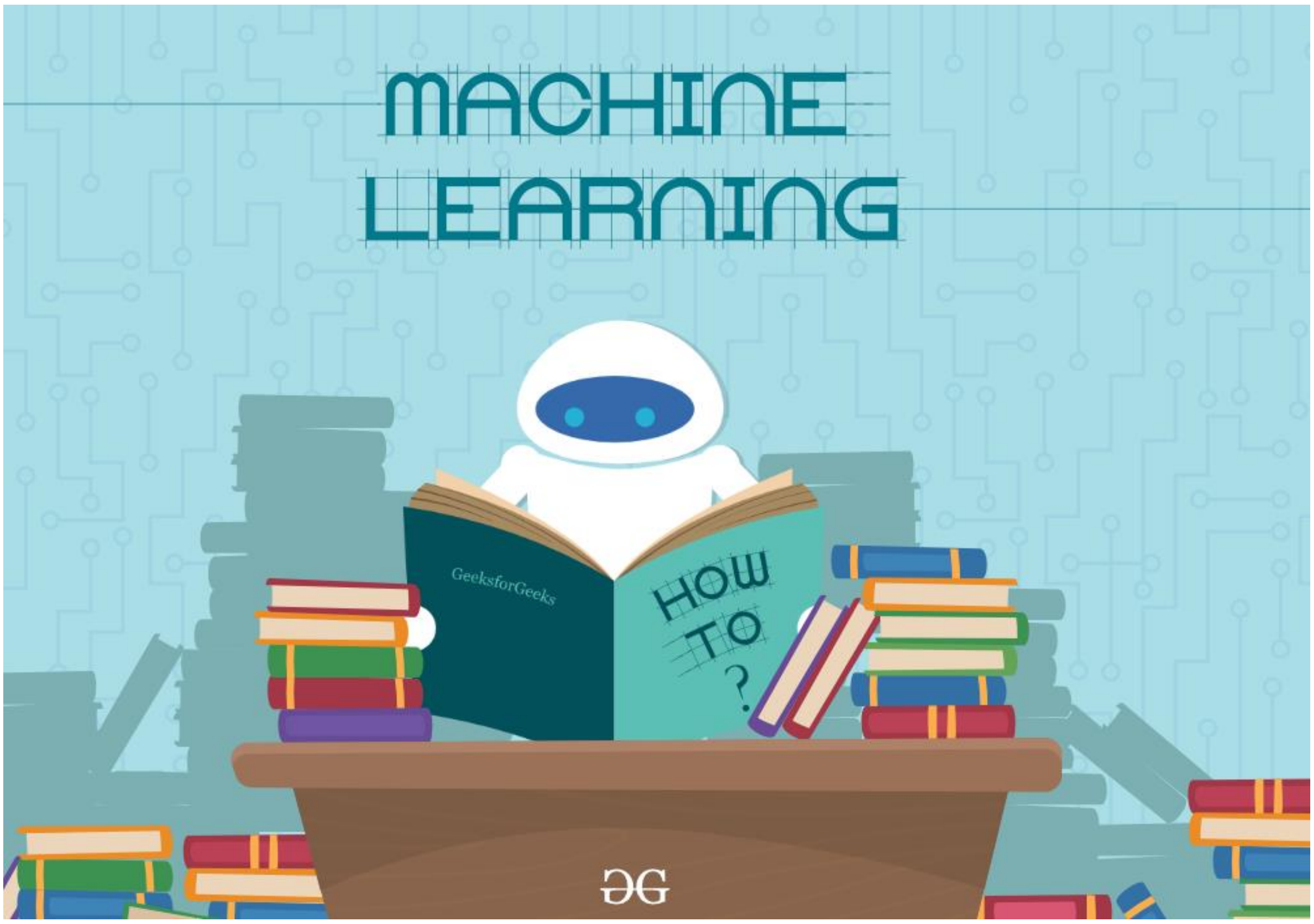


Machine Learning – Hand Book

Part: 1

Edited By: Susmoy Barman

Source: GeeksForGeeks

Machine Learning

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that which makes it more similar to humans: ***The ability to learn***. Machine learning is actively being used today, perhaps in many more places than one would expect

Introduction :

Introduction:

Getting started with Machine Learning

This article discusses the categories of machine learning problems, and terminologies used in the field of machine learning.

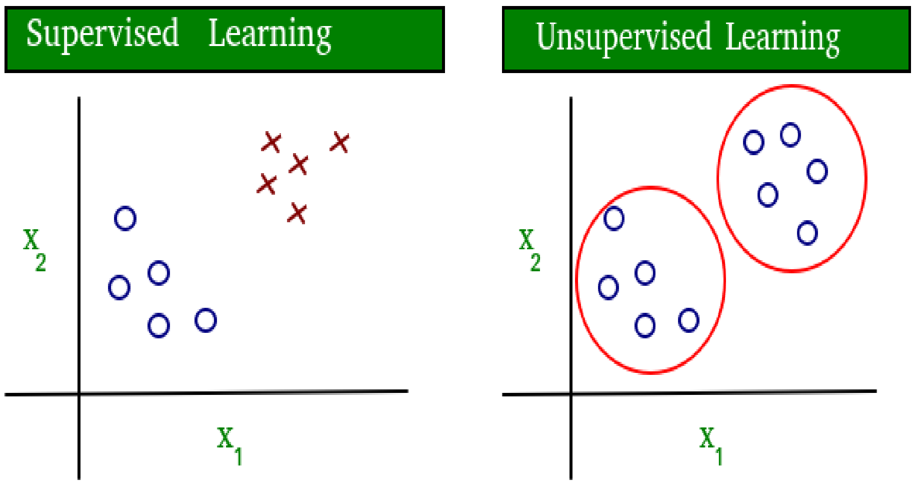
Types of machine learning problems

There are various ways to classify machine learning problems. Here, we discuss the most obvious ones.
1. On basis of the nature of the learning "signal" or "feedback" available to a learning system

- **Supervised learning**: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Some real life examples are:
    - **Image Classification:** You train with images/labels. Then in the future you give a new image expecting that the computer will recognize the new object.
    - **Market Prediction/Regression:** You train the computer with historical market data and ask the computer to predict the new price in the future.

- **Unsupervised learning**: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. It is used for clustering population in different groups. Unsupervised learning can be a goal in itself (discovering hidden patterns in data).
    - **Clustering:** You ask the computer to separate similar data into clusters, this is essential in research and science.
    - **High Dimension Visualization:** Use the computer to help us visualize high dimension data.
    - **Generative Models:** After a model captures the probability distribution of your input data, it will be able to generate more data. This can be very useful to make your classifier more robust.

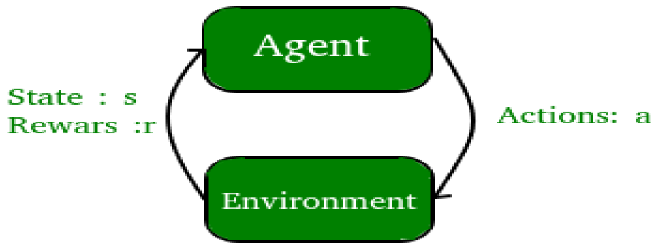
A simple diagram which clears the concept of supervised and unsupervised learning is shown below:



As you can see clearly, the data in supervised learning is labelled, where as data in unsupervised learning is unlabelled.

- **Semi-supervised learning**: Problems where you have a large amount of input data and only some of the data is labeled, are called semi-supervised learning problems. These problems sit in between both supervised and unsupervised learning. For example, a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.

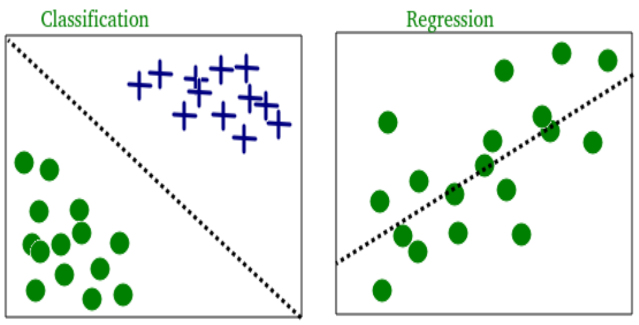
- **Reinforcement learning**: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.



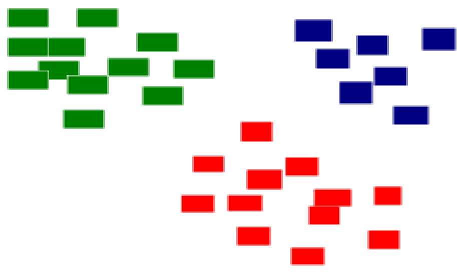
2. On the basis of "output" desired from a machine learned system

- **Classification**: Inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
- **Regression**: It is also a supervised learning problem, but the outputs are continuous rather than discrete. For example, predicting the stock prices using historical data.

An example of classification and regression on two different datasets is shown below:



- **Clustering**: Here, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
  As you can see in the example below, the given dataset points have been divided into groups identifiable by the colors red, green and blue.



- **Density estimation**: The task is to find the distribution of inputs in some space.
- **Dimensionality reduction**: It simplifies inputs by mapping them into a lower-dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

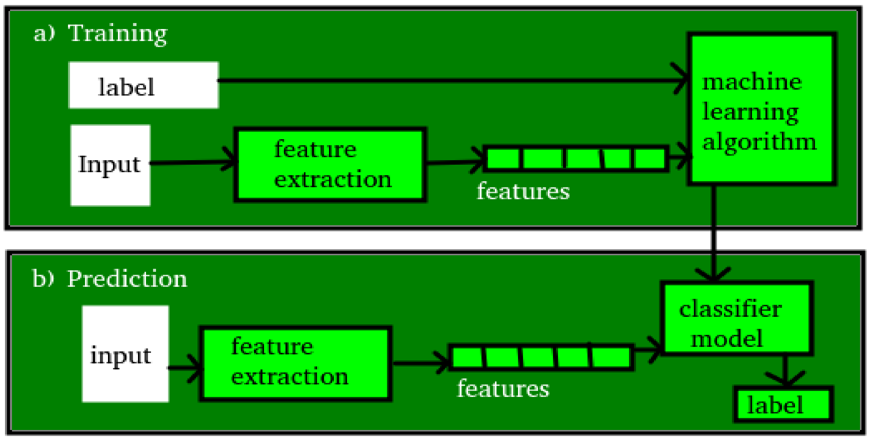
On the basis of these machine learning tasks/problems, we have a number of algorithms which are used to accomplish these tasks. Some commonly used machine learning algorithms are Linear Regression, Logistic Regression, Decision Tree, SVM(Support vector machines), Naive Bayes, KNN(K nearest neighbors), K-Means ,Random Forest, etc.

Note: All these algorithms will be covered in upcoming articles.

Terminologies of Machine Learning

- **Model**
  A model is a **specific representation** learned from data by applying some machine learning algorithm. A model is also called **hypothesis**.
- **Feature**
  A feature is an individual measurable property of our data. A set of numeric features can be conveniently described by a **feature vector**. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, **etc.**
  Note: Choosing informative, discriminating and independent features is a crucial step for effective algorithms. We generally employ a **feature extractor** to extract the relevant features from the raw data.
- **Target (Label)**
  A target variable or label is the value to be predicted by our model. For the fruit example discussed in features section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
- **Training**
  The idea is to give a set of inputs(features) and it's expected outputs(labels), so after training we will have a model (hypothesis) that will then map new data to one of the categories trained on.
- **Prediction**
  Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output(label).

The figure shown below clears the above concepts:



Artificial Intelligence | An Introduction

The most common answer that one expects is **"to make computers intelligent so that they can act intelligently!"**, but the question is how much intelligent? How can one judge the intelligence?

…as intelligent as humans. If the computers can, somehow, solve real-world problems, by improving on their own from the past experiences, they would be called "intelligent".
Thus, the AI systems are more generic(rather than specific), have the ability to "think" and are more flexible.

Intelligence, as we know, is the ability to acquire and apply the knowledge. Knowledge is the information acquired through experience. Experience is the knowledge gained through exposure(training). Summing the terms up, we get **artificial intelligence** as the "copy of something natural(i.e., human beings) 'WHO' is capable of acquiring and applying the information it has gained through exposure."

Intelligence is composed of:

- Reasoning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence

Many tools are used in AI, including versions of search and mathematical optimization, logic, methods based on probability and economics. The AI field draws upon computer science, mathematics, psychology, linguistics, philosophy, neuro-science, artificial psychology and many others.

Need for Artificial Intelligence

1. To create expert systems which exhibit intelligent behavior with the capability to learn, demonstrate, explain and advice its users.

2. Helping machines find solutions to complex problems like humans do and applying them as algorithms in a computer-friendly manner.

Applications of AI include **Natural Language Processing, Gaming, Speech Recognition, Vision Systems, Healthcare, Automotive** etc.

An AI system is composed of an agent and its environment. An agent(e.g., human or robot) is anything that can perceive its environment through sensors and acts upon that environment through effectors. Intelligent agents must be able to set goals and achieve them. In classical planning problems, the agent can assume that it is the only system acting in the world, allowing the agent to be certain of the consequences of its actions. However, if the agent is not the only actor, then it requires that the agent can reason under uncertainty. This calls for an agent that cannot only assess its environment and make predictions but also evaluate its predictions and adapt based on its assessment. Natural language processing gives machines the ability to read and understand human language. Some straightforward applications of natural language processing include information retrieval, text mining, question answering and machine translation. Machine perception is the ability to use input from sensors (such as cameras, microphones, sensors etc.) to deduce aspects of the world. e.g., Computer Vision. Concepts such as game theory, decision theory, necessitate that an agent be able to detect and model human emotions.

Many times, students get confused between Machine Learning and Artificial Intelligence, but Machine learning, a fundamental concept of AI research since the field's inception, is the study of computer algorithms that improve automatically through experience. The mathematical analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as a computational learning theory.

Stuart Shapiro divides AI research into three approaches, which he calls computational psychology, computational philosophy, and computer science. Computational psychology is used to make computer programs that mimic human behavior. Computational philosophy is used to develop an adaptive, free-flowing computer mind. Implementing computer science serves the goal of creating computers that can perform tasks that only people could previously accomplish.

AI has developed a large number of tools to solve the most difficult problems in computer science, like:

- Search and optimization
- Logic
- Probabilistic methods for uncertain reasoning
- Classifiers and statistical learning methods
- Neural networks
- Control theory
- Languages

High-profile examples of AI include autonomous vehicles (such as drones and self-driving cars), medical diagnosis, creating art (such as poetry), proving mathematical theorems, playing games (such as Chess or Go), search engines (such as Google search), online assistants (such as Siri), image recognition in photographs, spam filtering, prediction of judicial decisions[204] and targeting online advertisements. Other applications include Healthcare, Automotive
Finance, Video games etc

Are there limits to how intelligent machines – or human-machine hybrids – can be? A superintelligence, hyperintelligence, or superhuman intelligence is a hypothetical agent that would possess intelligence far surpassing that of the brightest and most gifted human mind. ‘’Superintelligence’’ may also refer to the form or degree of intelligence possessed by such an agent.

References: https://en.wikipedia.org/wiki/Artificial_intelligence

This article is contributed by **Palak Jain**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

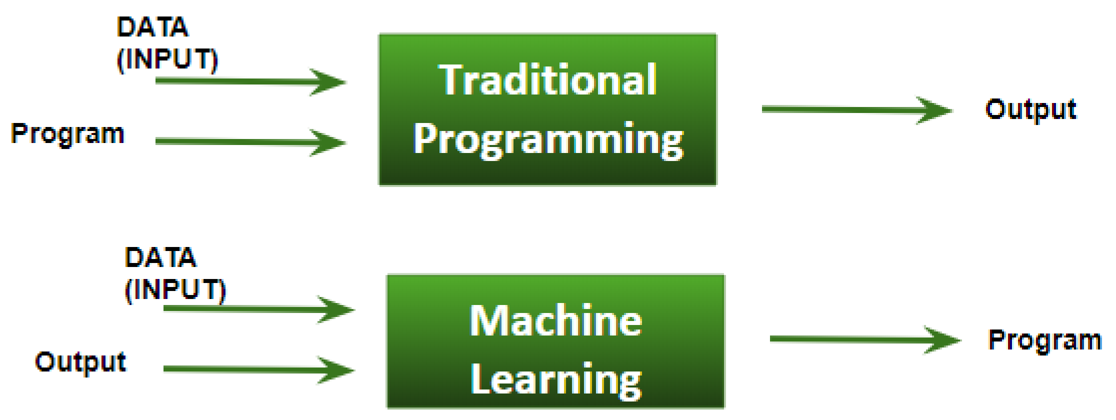
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

ML | What is Machine Learning ?

Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term **"Machine Learning"**. He defined machine learning as – **"Field of study that gives computers the capability to learn without being explicitly programmed"**.
In a very layman manner, Machine Learning(ML) can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance. The process starts with feeding a good quality data and then training our machines(computers) by building machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data do we have and what kind of task we are trying to automate.

Example: Training of students during exam.
While preparing for the exams students don't actually cram the subject but try to learn it with complete understanding. Before examination, they feed their machine(brain) with good amount of high quality data (questions and answers from different books or teachers notes or online video lectures). Actually, they are training their brain with input as well as output i.e. what kind of approach or logic do they have to solve different kind of questions. Each time they solve practice test papers, and find the performance (accuracy /score) by comparing answers with answer key given, Gradually, the performance keeps on increasing, gaining more confidence with the adopted approach. That's how actually models are built, train machine with data (both inputs and outputs are given to model) and when the time comes test on data (with input only) and achieve our model scores by comparing its answer with actual output which have not been fed while training. Researchers are working with assiduous efforts to improve algorithms, techniques so that these models perform even much better

.



Basic Difference in ML and Traditional Programming?

- **Traditional Programming :** We feed in DATA (Input) + PROGRAM (logic), run it on machine and get output.
- **Machine Learning :** We feed in DATA(Input) + Output, run it on machine during training and the machine creates its own program(logic), which can be evaluated while testing.

What does exactly learning means for a computer?

A computer is said to be learning from **Experiences** with respect to some class of **Tasks**, if its performance in a given Task improves with the Experience.

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

Example: playing checkers.
E = the experience of playing many games of checkers
T = the task of playing checkers.
P = the probability that the program will win the next game

In general, any machine learning problem can be assigned to one of two broad classifications:
Supervised learning and Unsupervised learning.

How things work in reality:-

- Talking about online shopping, there are millions of users with unlimited range of interests with respect to brands, colors, price range and many more. While online shopping, buyers tend to search for a number of products. Now, searching a product frequently will make buyer's Facebook, web pages, search engine or that online store start recommending or showing offers on that particular

product. There is no one sitting over there to code such task for each and every user, all this task is completely automatic. Here, ML plays its role. Researchers, data scientists, machine learners build models on machine using good quality and huge amount of data and now their machine is automatically performing and even improving with more and more experience and time.
Traditionally, advertisement was only done using newspapers, magazines and radio but now technology has made us smart enough to do **Targeted advertisement** (online ad system) which is a way more efficient method to target most receptive audience.

- Even in health care also, ML is doing a fabulous job. Researchers and scientists have prepared models to train machines for **detecting cancer** just by looking at slide – cell images. For humans to perform this task it would have taken a lot of time. But now, no more delay, machines predict the chances of having or not having cancer with some accuracy and doctors just have to give a assurance call, that's it. The answer to – how is this possible is very simple -all that is required, is, high computation machine, large amount of good quality image data, ML model with good algorithms to achieve state-of-the-art results.
  Doctors are using ML even to **diagnose patients** based on different parameters under consideration.

- You all might have use **IMDB ratings**, **Google Photos** where it recognizes faces, **Google Lens** where the ML image-text recognition model can extract text from the images you feed in, **Gmail** which categories E-mail as social, promotion, updates or forum using text classification,which is a part of ML.

How ML works?

- Gathering past data in the form of text file, excel file, images or audio data. The more better the quality of data, the better will be the model learning
- Data Processing – Sometimes, the data collected is in the raw form and it needs to be rectified. Example: if data has some missing values, then it has to be rectified. If data is in the form of text or images then converting it to numerical form will be required, be it list or array or matrix. Simply, Data is to be made relevant and understandable by the machine
- Building up models with suitable algorithms and techniques and then training it.
- Testing our prepared model with data which was not feed in at the time of training and so evaluating the performance – score, accuracy with high level of precision.

Pre-requisites to learn ML:

- Linear Algebra
- Statistics and Probability
- Calculus
- Graph theory
- Programming Skills – Language such as Python, R, MATLAB, C++ or Octave

An introduction to Machine Learning

The term Machine Learning was coined by Arthur Samuel in 1959, an American pioneer in the field of computer gaming and artificial intelligence and stated that "it gives computers the ability to learn without being explicitly programmed".
And in 1997, Tom Mitchell gave a "well-posed" mathematical and relational definition that "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Machine Learning is a latest buzzword floating around. It deserves to, as it is one of the most interesting subfield of Computer Science. So what does Machine Learning really mean?

Let's try to understand Machine Learning in layman terms. Consider you are trying to toss a paper to a dustbin.

After first attempt, you realize that you have put too much force in it. After second attempt, you realize you are closer to target but you need to increase your throw angle. What is happening here is basically after every throw we are learning something and improving the end result. We are programmed to learn from our experience.

This implies that the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?"
Within the field of data analytics, machine learning is used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data set(input).

Suppose that you decide to check out that offer for a vacation . You browse through the travel agency website and search for a hotel. When you look at a specific hotel, just below the hotel description there is a section titled "You might also like these hotels". This is a common use case of Machine Learning called "Recommendation Engine". Again, many data points were used to train a model in order to predict what will be the best hotels to show you under that section, based on a lot of information they already know about you.

So if you want your program to predict, for example, traffic patterns at a busy intersection (task T), you can run it through a machine learning algorithm with data about past traffic patterns (experience E) and, if it has successfully "learned", it will then do better at predicting future traffic patterns (performance measure P).
The highly complex nature of many real-world problems, though, often means that inventing specialized algorithms that will solve them perfectly every time is impractical, if not impossible. Examples of machine learning problems include, "Is this cancer?", "Which of these people are good friends with each other?", "Will this person like this movie?" such problems are excellent targets for Machine Learning, and in fact machine learning has been applied such problems with great success.

Classification of Machine Learning

Machine learning implementations are classified into three major categories, depending on the nature of the learning "signal" or "response" available to a learning system which are as follows:-

1. **Supervised learning :** When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of Supervised learning. This approach is indeed similar to human learning under the supervision of a teacher. The teacher provides good examples for the student to memorize, and the student then derives general rules from these specific examples.
2. **Unsupervised learning :**Whereas when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of un-correlated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.
   As a kind of learning, it resembles the methods humans use to figure out that certain objects or events are from the same class, such as by observing the degree of similarity between objects. Some recommendation systems that you find on the web in the form of marketing automation are based on this type of learning.
3. **Reinforcement learning :** When you present the algorithm with examples that lack labels, as in unsu-pervised learning. However, you can accompany an example with positive or negative feedback according to the solution the algorithm proposes comes under the category of Reinforcement learning, which is connected to applications for which the algorithm must make decisions (so the product is prescriptive, not just descriptive, as in unsupervised learning), and the decisions bear consequences. In the human world, it is just like learning by trial and error.
   Errors help you learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching you that a certain course of action is less likely to succeed than others. An interesting example of reinforcement learning occurs when computers learn to play video games by themselves. In this case, an application presents the algorithm with examples of specific situations, such as having the gamer stuck in a maze while avoiding an enemy. The application lets the algorithm know the outcome of actions it takes, and learning occurs while trying to avoid what it discovers to be dan-gerous and to pursue survival. You can have a look at how the company Google DeepMind has created a reinforcement learning program that plays old Atari's videogames. When watching the video, notice how the program is initially clumsy and unskilled but steadily improves with training until it becomes a champion.
4. **Semi-supervised learning :** where an incomplete training signal is given: a training set with some (often many) of the target outputs missing. There is a special case of this principle known as Transduction where the entire set of problem instances is known at learning time, except that part of the targets are missing.

Categorizing on the basis of required Output

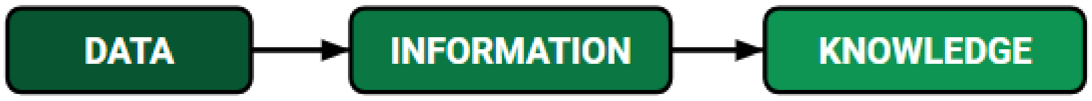
Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system:

1. **Classification :** When inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
2. **Regression :** Which is also a supervised problem, A case when the outputs are continuous rather than discrete.
3. **Clustering :** When a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

Machine Learning comes into the picture when problems cannot be solved by means of typical approaches.

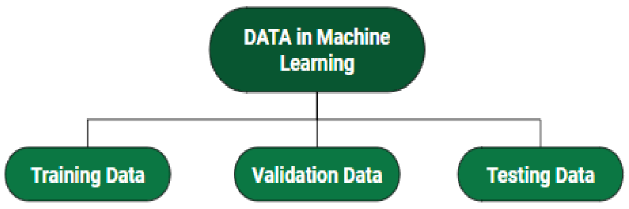
ML | Introduction to Data in Machine Learning

DATA : It can be any unprocessed fact, value, text, sound or picture that is not being interpreted and analyzed. Data is the most important part of all Data Analytics, Machine Learning, Artificial Intelligence. Without data, we can’t train any model and all modern research and automation will go vain. Big Enterprises are spending loads of money just to gather as much certain data as possible.
Example: Why did Facebook acquire WhatsApp by paying a huge price of $19 billion?
The answer is very simple and logical – it is to have access to the users’ information that Facebook may not have but WhatsApp will have. This information of their users is of paramount importance to Facebook as it will facilitate the task of improvement in their services.
INFORMATION : Data that has been interpreted and manipulated and has now some meaningful inference for the users.
KNOWLEDGE : Combination of inferred information, experiences, learning and insights. Results in awareness or concept building for an individual or organization.



How we split data in Machine Learning?

- **Training Data:** The part of data we use to train our model. This is the data which your model actually sees(both input and output) and learn from.
- **Validation Data:** The part of data which is used to do a frequent evaluation of model, fit on training dataset along with improving involved hyperparameters (initially set parameters before the model begins learning). This data plays it’s part when the model is actually training.
- **Testing Data:** Once our model is completely trained, testing data provides the unbiased evaluation. When we feed in the inputs of Testing data, our model will predict some values(without seeing actual output). After prediction, we evaluate our model by comparing it with actual output present in the testing data. This is how we evaluate and see how much our model has learned from the experiences feed in as training data, set at the time of training.



Consider an example:
There’s a Shopping Mart Owner who conducted a survey for which he has a long list of questions and answers that he had asked from the customers, this list of questions and answers is **DATA**. Now every time when he want to infer anything and can’t just go through each and every question of thousands of customers to find something relevant as it would be time-consuming and not helpful. In order to reduce this overhead and time wastage and to make work easier, data is manipulated through software, calculations, graphs etc. as per own convenience, this inference from manipulated data is **Information**. So, Data is must for Information. Now **Knowledge** has its role in differentiating between two individuals having same information. Knowledge is actually not a technical content but is linked to human thought process.

Properties of Data –

1. **Volume :** Scale of Data. With growing world population and technology at exposure, huge data is being generated each and every millisecond.
2. **Variety :** Different forms of data – healthcare, images, videos, audio clippings.
3. **Velocity :** Rate of data streaming and generation.
4. **Value :** Meaningfulness of data in terms of information which researchers can infer from it.
5. **Veracity :** Certainty and correctness in data we are working on.

Some facts about Data:

- As compared to 2005, 300 times i.e. 40 Zettabytes (1ZB=10^21 bytes) of data will be generated by 2020.
- By 2011, healthcare sector has a data of 161 Billion Gigabytes
- 400 Million tweets are sent by about 200 million active users per day
- Each month, more than 4 Billion hours of video streaming is done by the users.
- 30 Billion different types of contents are shared every month by the user.
- It is reported that about 27% of data is inaccurate and so 1 in 3 business idealists or leaders don’t trust the information on which they are making decisions.

The above-mentioned facts are just a glimpse of the actually existing huge data statistics. When we talk in terms of real world scenario, the size of data currently present and is getting generated each and every moment is beyond our mental horizons to imagine.

Demystifying Machine Learning

Machine Learning”. Now that’s a word that packs a punch! Machine learning is hot stuff these days! And why won’t it be? Almost every “enticing” new development in the field of Computer Science and Software Development in general has something related to machine learning behind the veils. Microsoft’s Cortana – Machine Learning. Object and Face Recognition – Machine Learning and Computer Vision. Advanced UX improvement programs – Machine Learning (yes!. The Amazon product recommendation you just got was the number crunching effort of some Machine Learning Algorithm).

And not even just that. Machine Learning and Data Science in general is EVERYWHERE. It is as omnipotent as God himself, had he been into Computers! Why? Because Data is everywhere!

So it is natural, that anyone who has above average brains and can differentiate between Programming Paradigms by taking a sneak-peek at Code, is intrigued by Machine Learning. But what is Machine Learning? And how big is Machine Learning? Let’s demystify Machine Learning, once and for all. And to do that, rather than presenting technical specifications, we’ll follow a “Understand by Example” approach.

Machine Learning : What is it really?

Well, Machine Learning is a subfield of Artificial Intelligence which evolved from Pattern Recognition and Computational Learning theory. Arthur Lee Samuel defines Machine Learning as: Field of study that gives computers the ability to learn without being explicitly programmed.

So, basically, the field of Computer Science and Artificial intelligence that "learns" from data without human intervention.

But this view has a flaw. As a result of this perception, whenever the word Machine Learning is thrown around, people usually think of "A.I." and "Neural Networks that can mimic Human brains (as of now, that is not possible)", Self Driving Cars and what not. But Machine Learning is far beyond that. Below we uncover some expected and some generally not expected facets of Modern Computing where Machine Learning is in action.

Machine Learning: The Expected

We'll start with some places where you might expect Machine Learning to play a part.

1. **Speech Recognition (Natural Language Processing in more technical terms) :** You talk to Cortana on Windows Devices. But how does it understand what you say? Along comes the field of Natural Language Processing, or N.L.P. It deals with the study of interactions between Machines and Humans, via Linguistics. Guess what is at the heart of NLP: Machine Learning Algorithms and Systems (Hidden Markov Models being one).

2. **Computer Vision :** Computer Vision is a subfield of AI which deals with a Machine's (probable) interpretation of the Real World. In other words, all Facial Recognition, Pattern Recognition, Character Recognition Techniques belong to Computer Vision. And Machine Learning once again, with it wide range of Algorithms, is at the heart of Computer Vision.

3. **Google's Self Driving Car** : Well. You can imagine what drives it actually. More Machine Learning goodness.

But these were expected applications. Even a naysayer would have a good insight about these feats of technology being brought to life by some "mystical (and extremely hard) mind crunching Computer wizardry".

Machine Learning : The Unexpected

Let's visit some places normal folks would not really associate easily with Machine Learning:

1. **Amazon's Product Recommendations:** Ever wondered how Amazon always has a recommendation that just tempts you to lighten your wallet. Well, that's a Machine Learning Algorithm(s) called "Recommender Systems" working in the backdrop. It learns every user's personal preferences and makes recommendations according to that.

2. **Youtube/Netflix :** They work just as above!

3. **Data Mining / Big Data :** This might not be so much of a shock to many. But Data Mining and Big Data are just manifestations of studying and learning from data at a larger scale. And wherever there's the objective of extracting information from data, you'll find Machine Learning lurking nearby.

4. **Stock Market/Housing Finance/Real Estate :** All of these fields, incorporate a lot of Machine Learning systems in order to better assess the market, namely "Regression Techniques", for things as mediocre as predicting the price of a House, to predicting and analyzing stock market trends.

So as you might have seen now. Machine Learning actually is everywhere. From Research and Development to improving business of Small Companies. It is everywhere. And hence it makes up for quite a career option, as the industry is on the rise and is the boon is not stopping any time soon.

So, this is it for now. This wraps up our Machine Learning 101. We'll hopefully meet again, and when we do, we'll dive into some technical details of Machine Learning, what tools are used in the industry, and how to start your journey to Machine Learning prowess. Till then, Code Away!

Machine Learning – Applications

Introduction

Machine learning is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that which makes it more similar to humans: The ability to learn. Machine learning is actively being used today, perhaps in many more places than one would expect. We probably use a learning algorithm dozens of time without even knowing it. Applications of Machine Learning include:

- **Web Search Engine:** One of the reasons why search engines like google, bing etc work so well is because the system has learnt how to rank pages through a complex learning algorithm.

- **Photo tagging Applications:** Be it facebook or any other photo tagging application, the ability to tag friends makes it even more happening. It is all possible because of a face recognition algorithm that runs behind the application.

- **Spam Detector:** Our mail agent like Gmail or Hotmail does a lot of hard work for us in classifying the mails and moving the spam mails to spam folder. This is again achieved by a spam classifier running in the back end of mail application.

Today, companies are using Machine Learning to improve business decisions,increase productivity, detect disease, forecast weather, and do many more things. With the exponential growth of technology, we not only need better tools to understand the data we currently have, but we also need to prepare ourselves for the data we will have. To achieve this goal we need to build intelligent machines. We can write a program to do simple things. But for most of times Hardwiring Intelligence in it is difficult. Best way to do it is to have some way for machines to learn things themselves. A mechanism for learning – if a machine can learn from input then it does the hard work for us. This is where Machine Learning comes in action. Some examples of machine learning are:

- **Database Mining for growth of automation:** Typical applications include Web-click data for better UX(User eXperience), Medical records for better automation in healthcare, biological data and many more.

- **Applications that cannot be programmed:** There are some tasks that cannot be programmed as the computers we use are not modelled that way. Examples include Autonomous Driving, Recognition tasks from unordered data (Face Recognition/ Handwriting Recognition), Natural language Processing, computer Vision etc.
- **Understanding Human Learning:** This is the closest we have understood and mimicked the human brain. It is the start of a new revolution, The real AI. Now, After a brief insight lets come to a more formal definition of Machine Learning
- **Arthur Samuel(1959):** "Machine Learning is a field of study that gives computers, the ability to learn without explicitly being programmed."Samuel wrote a Checker playing program which could learn over time. At first it could be easily won. But over time, it learnt all the board position that would eventually lead him to victory or loss and thus became a better chess player than Samuel itself. This was one of the most early attempts of defining Machine Learning and is somewhat less formal.

- **Tom Michel(1999):** "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." This is a more formal and mathematical definition. For the previous Chess program
  - E is number of games.
  - T is playing chess against computer.
  - P is win/loss by computer.

In the Next tutorial we shall classify the types of Machine Learning problems and shall also discuss about useful packages and setting environment for Machine Learning and how can we use it to design new projects.

Machine Learning and Artificial Intelligence

Machine Learning and **Artificial Intelligence** are creating a huge buzz worldwide. The plethora of applications in Artificial Intelligence have changed the face of technology. These terms Machine Learning and Artificial Intelligence are often used interchangeably. However, there is a stark difference between the two that is still unknown to the industry professionals.
Let's start by taking an example of Virtual Personal Assistants which have been familiar to most of us from quite some time now.

Working of Virtual Personal Assistants –
Siri(part of Apple Inc.'s iOS, watchOS, macOS, and tvOS operating systems), **Google Now** (a feature of Google Search offering predictive cards with information and daily updates in the Google app for Android and iOS.), **Cortana** (Cortana is a virtual assistant created by Microsoft for Windows 10) are intelligent digital personal assistants on the platforms like iOS, Android and Windows respectively. To put it plainly, they help to find relevant information when requested using voice. For instance, for answering queries like 'What's the temperature today?' or 'What is the way to the nearest supermarket' etc. and the assistant will react by searching information, transferring that information from the phone or sending commands to various other applications.

AI is critical in these applications, as they gather data on the user's request and utilize that data to perceive speech in a better manner and serve the user with answers that are customized to his inclination. **Microsoft says that Cortana "consistently finds out about its user" and that it will in the end build up the capacity to anticipate users' needs and cater to them.** Virtual assistants process a tremendous measure of information from an assortment of sources to find out about users and be more compelling in helping them arrange and track their data. Machine learning is a vital part of these personal assistants as they gather and refine the data based on user's past participation with them. Thereon, this arrangement of information is used to render results that are custom-made to user's inclinations.

Roughly speaking, Artificial Intelligence (AI) is when a computer algorithm does intelligent work. On the other hand, Machine Learning is a part of AI that learns from the data that also involves the information gathered from the previous experiences and allows the computer program to change its behavior accordingly. **Artificial Intelligence is the superset of Machine Learning** i.e. all the Machine Learning is Artificial Intelligence but not all the AI is Machine Learning.

Artificial Intelligence	Machine Learning
AI manages more comprehensive issues of automating a system. This computerization	Machine Learning (ML) manages influencing user's machine to gain from the external environment. This external

Artificial Intelligence	Machine Learning
should be possible by utilizing any field such as image processing, cognitive science, neural systems, machine learning etc.	environment can be sensors, electronic segments, external storage gadgets and numerous other devices.
AI manages the making of machines, frameworks and different gadgets savvy by enabling them to think and do errands as all people generally do.	What ML does, depends on the user input or a query requested by the client, the framework checks whether it is available in the knowledge base or not. If it is available, it will restore the outcome to the user related with that query, however if it isn't stored initially, the machine will take in the user input and will enhance its knowledge base, to give a better value to the end user

Future Scope –

- Artificial Intelligence is here to stay and is going nowhere. It digs out the facts from algorithms for a meaningful execution of various decisions and goals predetermined by a firm.
- Artificial Intelligence and Machine Learning are likely to replace the current mode of technology that we see these days, for example, traditional programming packages like **ERP** and **CRM** are certainly losing their charm.
- Firms like Facebook, Google are investing a hefty amount in AI to get the desired outcome at a relatively lower computational time.
- Artificial Intelligence is something that is going to redefine the world of software and IT in the near future.

Difference between Machine learning and Artificial Intelligence

Artificial Intelligence and Machine Learning are the terms of computer science. This article discusses some points on the basis of which we can differentiate between these two terms.

Overview

Artificial Intelligence : The word Artificial Intelligence comprises of two words "Artificial" and "Intelligence". Artificial refers to something which is made by human or non natural thing and Intelligence means ability to understand or think. There is a misconception that Artificial Intelligence is a system, but it is not a system .AI is implemented in the system. There can be so many definition of AI, one definition can be ***"It is the study of how to train the computers so that computers can do things which at present human can do better."***Therefore It is a intelligence where we want to add all the capabilities to machine that human contain.

Machine Learning : Machine Learning is the learning in which machine can learn by its own without being explicitly programmed. It is an application of AI that provide system the ability to automatically learn and improve from experience. Here we can generate a program by integrating input and output of that program. One of the simple definition of the Machine Learning is ***"Machine Learning is said to learn from experience E w.r.t some class of task T and a performance measure P if learners performance at the task in the class as measured by P improves with experiences."***

The key difference between AI and ML are:

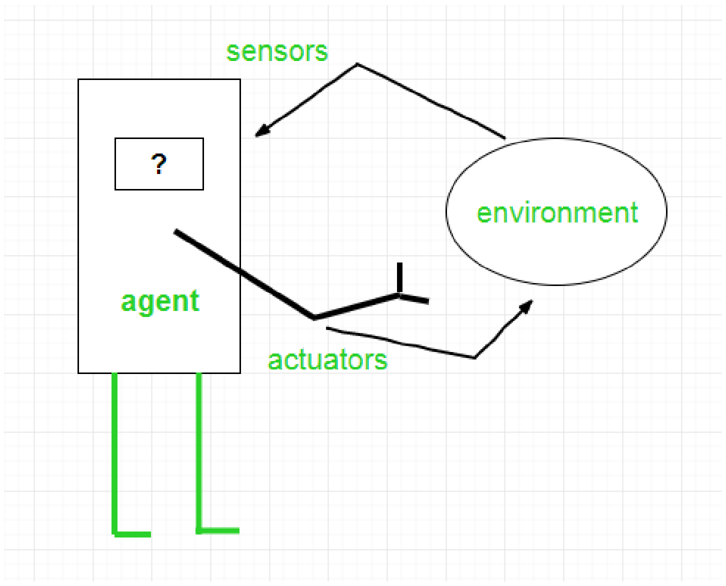
ARTIFICIAL INTELLIGENCE	MACHINE LEARNING
AI stands for Artificial intelligence, where intelligence is defined acquisition of knowledge intelligence is defined as a ability to acquire and apply knowledge.	ML stands for Machine Learning which is defined as the acquisition of knowledge or skill
The aim is to increase chance of success and not accuracy.	The aim is to increase accuracy, but it does not care about success
It work as a computer program that does smart work	It is a simple concept machine takes data and learn from data.
The goal is to simulate natural intelligence to solve complex problem	The goal is to learn from data on certain task to maximize the performance of machine on this task.
AI is decision making.	ML allows system to learn new things from data.
It leads to develop a system to mimic human to respond behave in a circumstances.	It involves in creating self learning algorithms.
AI will go for finding the optimal solution.	ML will go for only solution for that whether it is optimal or not.
AI leads to intelligence or wisdom.	ML leads to knowledge.

Agents in Artificial Intelligence

Artificial intelligence is defined as study of rational agents. A rational agent could be anything which makes decisions, like a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts(agent's perceptual inputs at a given instance).
An AI system is composed of an **agent and its environment**. The agents act in their environment. The environment may contain other agents. An agent is anything that can be viewed as :

- perceiving its environment through **sensors** and
- acting upon that environment through **actuators**

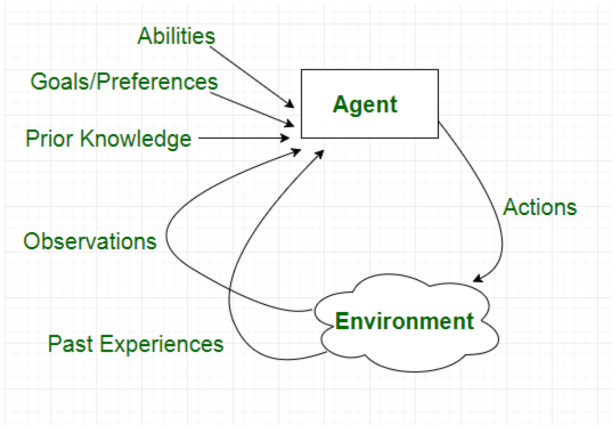
Note : Every agent can perceive its own actions (but not always the effects)



To understand the structure of Intelligent Agents, we should be familiar with *Architecture* and *Agent Program*. **Architecture** is the machinery that the agent executes on. It is a device with sensors and actuators, for example : a robotic car, a camera, a PC. **Agent program** is an implementation of an agent function. An **agent function** is a map from the percept sequence(history of all that an agent has perceived till date) to an action.

Agent = Architecture + Agent Program

Examples of Agent:-
A **software agent** has Keystrokes, file contents, received network packages which act as sensors and displays on the screen, files, sent network packets acting as actuators.
A **Human agent** has eyes, ears, and other organs which act as sensors and hands, legs, mouth, and other body parts acting as actuators.
A **Robotic agent** has Cameras and infrared range finders which act as sensors and various motors acting as actuators.



Types of Agents

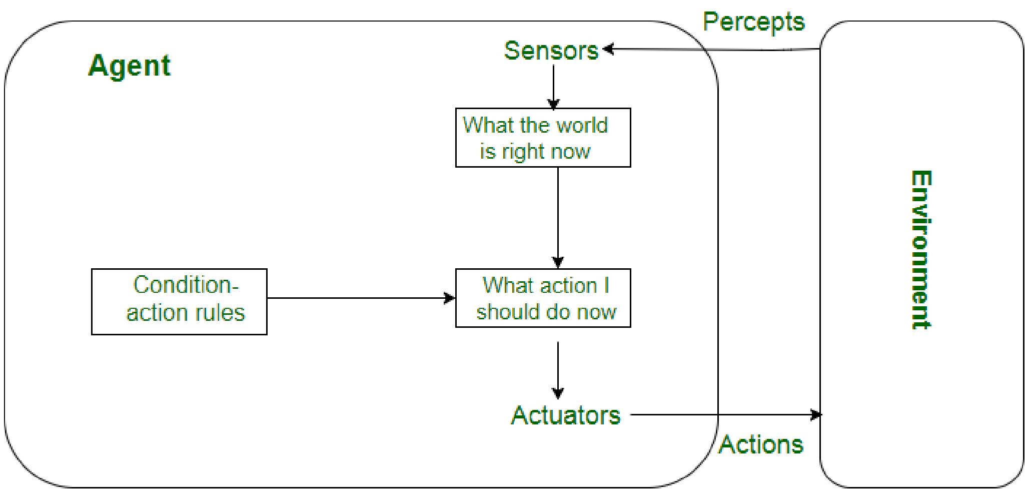
Agents can be grouped into four classes based on their degree of perceived intelligence and capability :

- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents

Simple reflex agents

Simple reflex agents ignore the rest of the percept history and act only on the basis of the **current percept**. Percept history is the history of all that an agent has perceived till date. The agent function is based on the **condition-action rule**. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions. Problems with Simple reflex agents are :

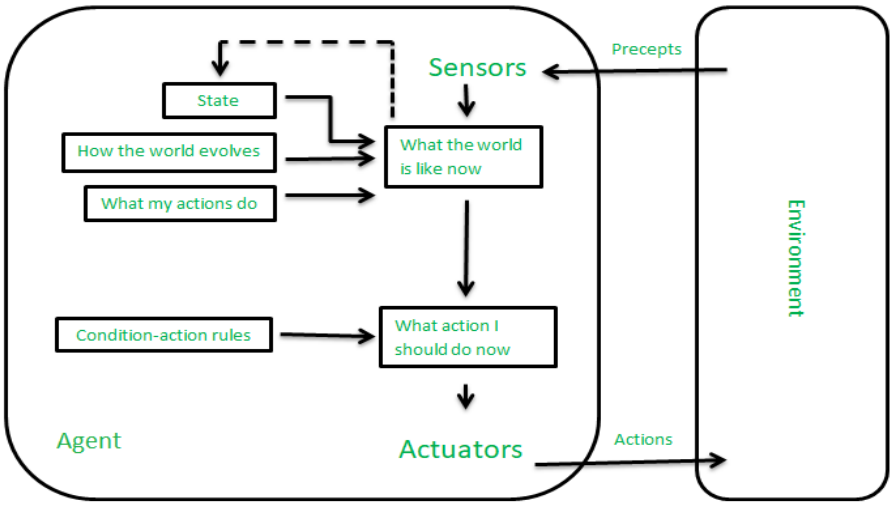
- Very limited intelligence.
- No knowledge of non-perceptual parts of state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collection of rules need to be updated.



Model-based reflex agents

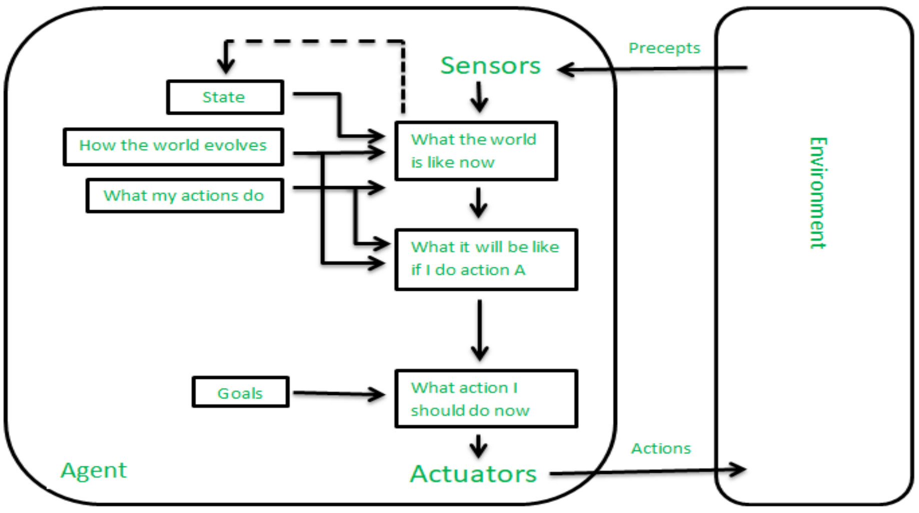
It works by finding a rule whose condition matches the current situation. A model-based agent can handle **partially observable environments** by use of model about the world. The agent has to keep track of **internal state** which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen. Updating the state requires the information about :

- how the world evolves in-dependently from the agent, and
- how the agent actions affects the world.



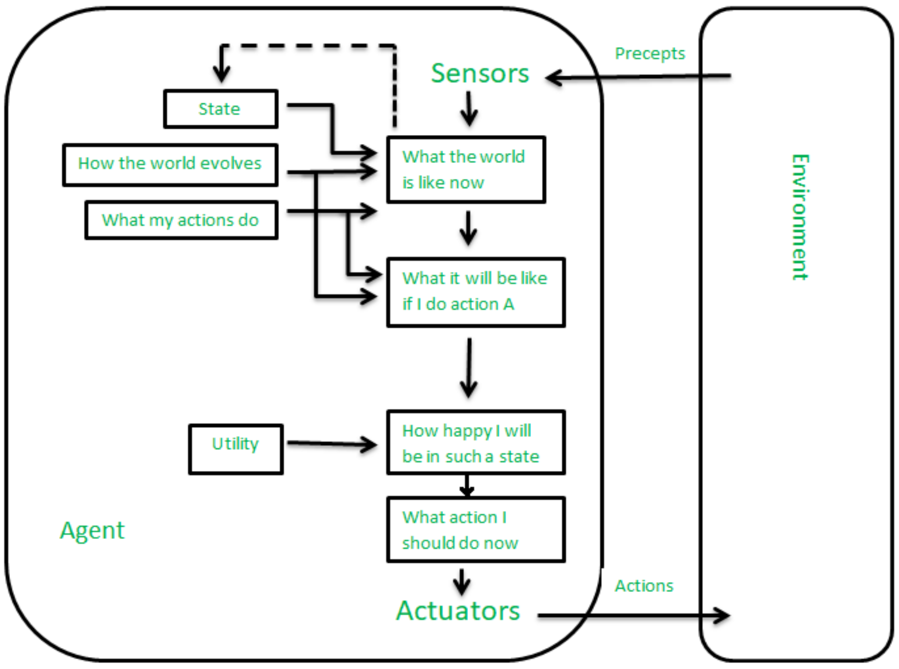
Goal-based agents

These kind of agents take decision based on how far they are currently from their **goal**(description of desirable situations). Their every action is intended to reduce its distance from goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They usually require search and planning. The goal based agent's behavior can easily be changed.



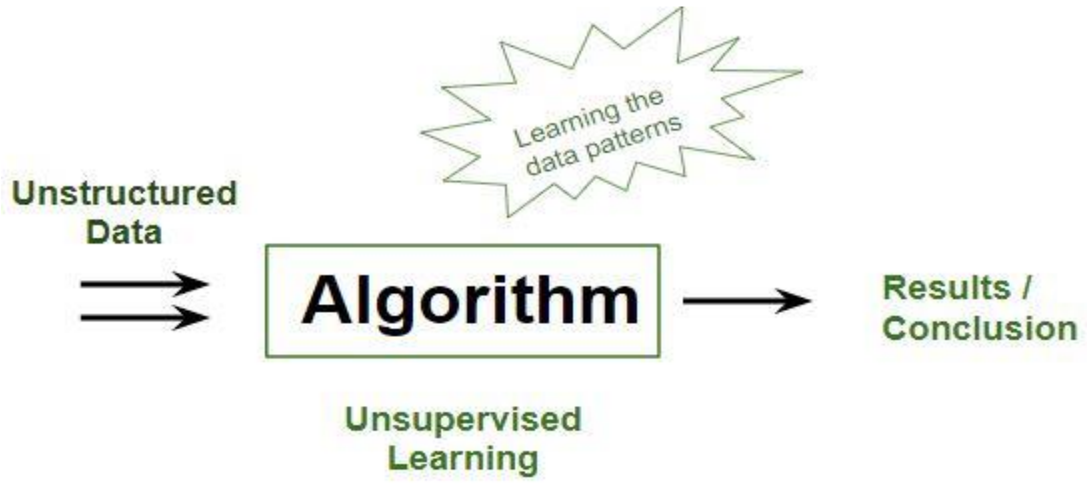
Utility-based agents

The agents which are developed having their end uses as building blocks are called utility based agents. When there are multiple possible alternatives, then to decide which one is best, utility based agents are used.They choose actions based on a **preference (utility)** for each state. Sometimes achieving the desired goal is not enough. We may look for quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration. Utility describes how **"happy"** the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.

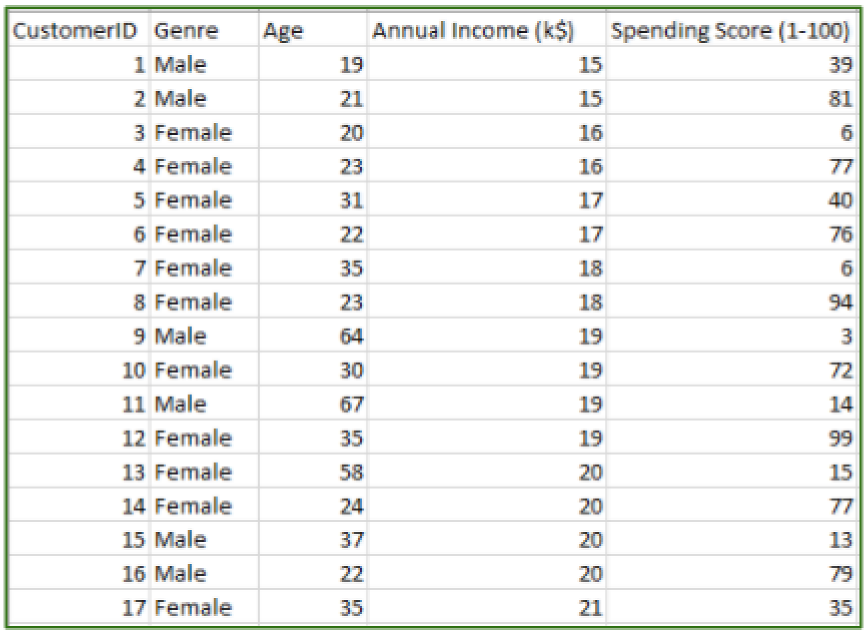

Sensors
Precepts
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Actions
Environment

Supervised and Unsupervised learning

ML | Types of Learning :



Unsupervised Learning :
It's a type of learning where we don't give target to our model while training i.e. training model has only input parameter values. The model by itself has to find which way it can learn. Data-set in Figure A is mall data that contains information of its clients that subscribe to them. Once subscribed they are provided a membership card and so the mall has complete information about customer and his/her every purchase. Now using this data and unsupervised learning techniques, mall can easily group clients based on the parameters we are feeding in.

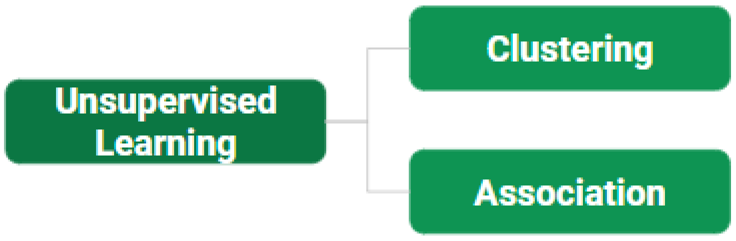


CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35

Figure A

Training data we are feeding is –

- **Unstructured data**: May contain noisy(meaningless) data, missing values or unknown data
- **Unlabeled data** : Data only contains value for input parameters, there is no targeted value(output). It is easy to collect as compared to labelled one in Supervised approach.



Types of Unsupervised Learning :-

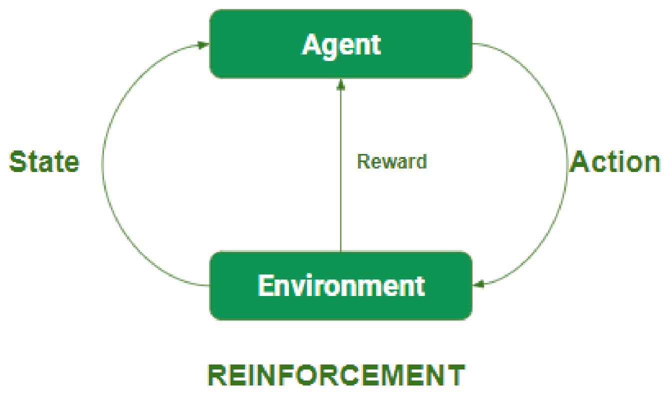
- **Clustering:** Broadly this technique is applied to group data based on different patterns, our machine model finds. For example in above figure we are not given output parameter value, so this technique will be used to group clients based on the input parameters provided by our data.
- **Association:** This technique is a rule based ML technique which finds out some very useful relations between parameters of a large data set. For e.g. shopping stores use algorithms based on this technique to find out relationship between sale of one product w.r.t to others sale based on customer behavior. Once trained well, such models can be used to increase their sales by planning different offers.

Some algorithms:

- K-Means Clustering
- DBSCAN – Density-Based Spatial Clustering of Applications with Noise
- BIRCH – Balanced Iterative Reducing and Clustering using Hierarchies
- Hierarchical Clustering

Semi-supervised Learning:
As the name suggests, its working lies between Supervised and Unsupervised techniques. We use these

techniques when we are dealing with a data which is a little bit labelled and rest large portion of it is unlabeled. We can use unsupervised technique to predict labels and then feed these labels to supervised techniques. This technique is mostly applicable in case of image data-sets where usually all images are not labelled.



Reinforcement Learning:
In this technique, model keeps on increasing its performance using a Reward Feedback to learn the behavior or pattern. These algorithms are specific to a particular problem e.g. Google Self Driving car, AlphaGo where a bot competes with human and even itself to getting better and better performer of Go Game. Each time we feed in data, they learn and add the data to its knowledge that is training data. So, more it learns the better it get trained and hence experienced.

- Agents observe input.
- Agent performs an action by making some decisions.
- After its performance, agent receives reward and accordingly reinforce and the model stores in state-action pair of information.

Some algorithms:

- Temporal Difference (TD)
- Q-Learning
- Deep Adversarial Networks

Supervised and Unsupervised learning

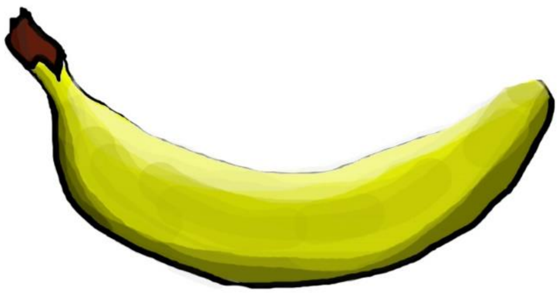
<u>Supervised learning</u>

Supervised learning as the name indicates a presence of supervisor as teacher. Basically supervised learning is a learning in which we teach or train the machine using data which is well labeled that means some data is already tagged with correct answer. After that, machine is provided with new set of examples(data) so that supervised learning algorithm analyses the training data(set of training examples) and produces an correct outcome from labeled data.

For instance, suppose you are given an basket filled with different kinds of fruits. Now the first step is to train the machine with all different fruits one by one like this:

- If shape of object is rounded and depression at top having color Red then it will be labelled as – **Apple**.
- If shape of object is long curving cylinder having color Green-Yellow then it will be labelled as – **Banana**.

Now suppose after training the data, you have given a new separate fruit say Banana from basket and asked to identify it.



Since machine has already learnt the things from previous data and this time have to use it wisely. It will first classify the fruit with its shape and color, and would confirm the fruit name as BANANA and put it in Banana category. Thus machine learns the things from training data(basket containing fruits) and then apply the knowledge to test data(new fruit).

Supervised learning classified into two categories of algorithms:

- **Classification**: A classification problem is when the output variable is a category, such as "Red" or "blue" or "disease" and "no disease".
- **Regression**: A regression problem is when the output variable is a real value, such as "dollars" or "weight".

Unsupervised learning

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabeled data by our-self.

For instance, suppose it is given an image having both dogs and cats which have not seen ever.

Thus machine has no any idea about the features of dogs and cat so we can't categorize it in dogs and cats. But it can categorize them according to their similarities, patterns and differences i.e., we can easily categorize the above picture into two parts. First first may contain all pics having **dogs** in it and second part may contain all pics having **cats** in it. Here you didn't learn anything before, means no training data or examples.

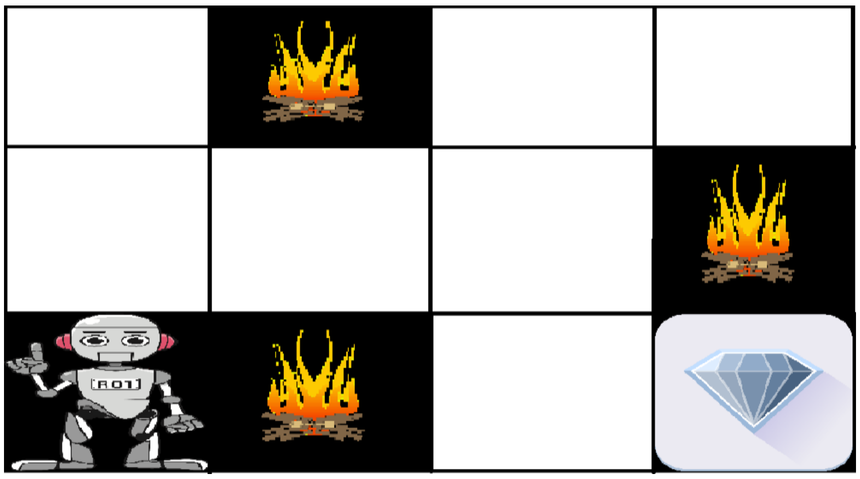
Unsupervised learning classified into two categories of algorithms:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Reinforcement learning

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience.

Example : The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.



The above image shows robot, diamond and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that is fire. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

- Input: The input should be an initial state from which the model will start
- Output: There are many possible output as there are variety of solution to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

<u>Difference between Reinforcement learning and Supervised learning:</u>

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words we can say that the out depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	Supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game	Example: Object recognition

Types of Reinforcement: There are two types of Reinforcement:

1. **Positive –**
   Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words it has a positive effect on the behavior.

   Advantages of reinforcement learning are:

   - Maximizes Performance
   - Sustain Change for a long period of time

   Disadvantages of reinforcement learning:

   - Too much Reinforcement can lead to overload of states which can diminish the results
2. **Negative –**
   Negative Reinforcement is defined as strengthening of a behavior because a negative condition is stopped or avoided.

   Advantages of reinforcement learning:

   - Increases Behavior
   - Provide defiance to minimum standard of performance

Disadvantages of reinforcement learning:

- It Only provides enough to meet up the minimum behavior

Various Practical applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

RL can be used in large environments in the following situations:

1. A model of the environment is known, but an analytic solution is not available;
2. Only a simulation model of the environment is given (the subject of simulation-based optimization);[6]
3. The only way to collect information about the environment is to interact with it.

Parametric Methods :

Regression and Classification | Supervised Machine Learning:

What is Regression and Classification in Machine Learning?

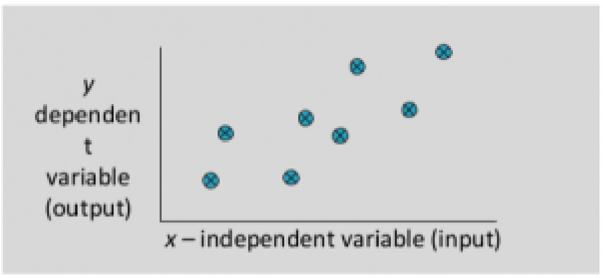
Data scientists use many different kinds of machine learning algorithms to discover patterns in big data that lead to actionable insights. At a high level, these different algorithms can be classified into two groups based on the way they "learn" about data to make predictions: supervised and unsupervised learning.

Supervised Machine Learning: The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output **Y = f(X)** . The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

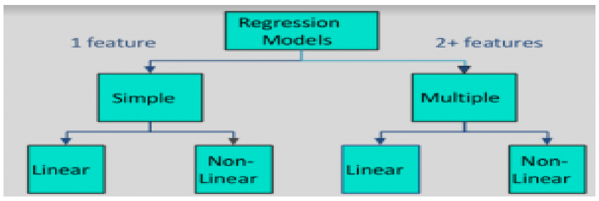
Techniques of Supervised Machine Learning algorithms include **linear** and **logistic regression**, **multi-class classification**, **Decision Trees** and **support vector machines**. Supervised learning requires that the data used to train the algorithm is already labeled with correct answers. For example, a classification algorithm will learn to identify animals after being trained on a dataset of images that are properly labeled with the species of the animal and some identifying characteristics.
Supervised learning problems can be further grouped into **Regression** and **Classification** problems. Both problems have as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the fact that the dependent attribute is numerical for regression and categorical for classification.

Regression

A regression problem is when the output variable is a real or continuous value, such as “salary” or “weight”. Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points.



Types of Regression Models:



For Examples:
Which of the following is a regression task?

- Predicting age of a person
- Predicting nationality of a person
- Predicting whether stock price of a company will increase tomorrow
- Predicting whether a document is related to sighting of UFOs?

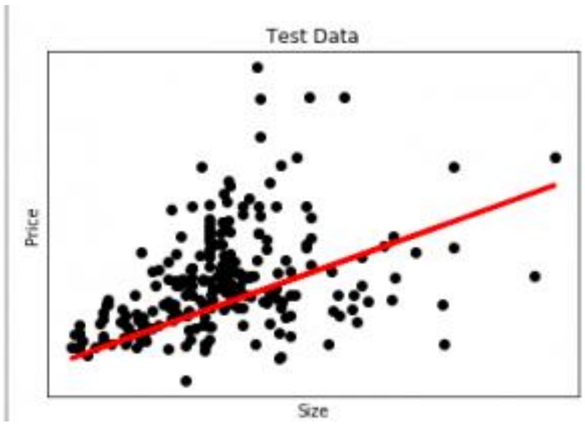
Solution : Predicting age of a person (because it is a real value, predicting nationality is categorical, whether stock price will increase is discreet-yes/no answer, predicting whether a document is related to UFO is again discreet- a yes/no answer).

Let’s take an example of linear regression. We have a **Housing data set** and we want to predict the price of the house. Following is the python code for it.

```
# Python code to illustrate

# regression using data set

import matplotlib

matplotlib.use('GTKAgg')


import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear_model

import pandas as pd
```

```
# Load CSV and columns
df = pd.read_csv("Housing.csv")

Y = df['price']
X = df['lotsize']

X=X.reshape(len(X),1)
Y=Y.reshape(len(Y),1)

# Split the data into training/testing sets
X_train = X[:-250]
X_test = X[-250:]

# Split the targets into training/testing sets
Y_train = Y[:-250]
Y_test = Y[-250:]

# Plot outputs
plt.scatter(X_test, Y_test,  color='black')
plt.title('Test Data')
plt.xlabel('Size')
plt.ylabel('Price')
plt.xticks(())
plt.yticks(())



# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, Y_train)

# Plot outputs
plt.plot(X_test, regr.predict(X_test), color='red',linewidth=3)
plt.show()
```

The output of the above code will be:



Here in this graph, we plot the test data. The red line indicates the best fit line for predicting the price. To make an individual prediction using the linear regression model:

```
print( str(round(regr.predict(5000))) )
```

Classification

A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”. A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes.
For example, when filtering emails “spam” or “not spam”, when looking at transaction data, “fraudulent”, or “authorized”. In short Classification either predicts categorical class labels or classifies data (construct a model) based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data. There are a number of classification models. Classification models include logistic regression, decision tree, random forest, gradient-boosted tree, multilayer perceptron, one-vs-rest, and Naive Bayes.

For example :
Which of the following is/are classification problem(s)?

- Predicting the gender of a person by his/her handwriting style
- Predicting house price based on area
- Predicting whether monsoon will be normal next year
- Predict the number of copies a music album will be sold next month

Solution : Predicting the gender of a person Predicting whether monsoon will be normal next year. The other two are regression.
As we discussed classification with some examples. Now there is an example of classification in which we are performing classification on the iris dataset using *RandomForestClassifier* in python. You can download the dataset from Here
Dataset Description

```
Title: Iris Plants Database
Attribute Information:
      1. sepal length in cm
      2. sepal width in cm
      3. petal length in cm
      4. petal width in cm
      5. class:
       -- Iris Setosa
       -- Iris Versicolour
       -- Iris Virginica
 Missing Attribute Values: None
Class Distribution: 33.3% for each of 3 classes
```

```
# Python code to illustrate
# classification using data set
#Importing the required library
import pandas as pd
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

#Importing the dataset

dataset = pd.read_csv(
          'https://archive.ics.uci.edu/ml/machine-learning-'+
          'databases/iris/iris.data',sep= ',', header= None)
data = dataset.iloc[:, :]

#checking for null values
print("Sum of NULL values in each column. ")
print(data.isnull().sum())

#seperating the predicting column from the whole dataset
X = data.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

#Encoding the predicting variable
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

#Spliting the data into test and train dataset
X_train, X_test, y_train, y_test = train_test_split(
              X, y, test_size = 0.3, random_state = 0)

#Using the random forest classifier for the prediction
```

```
classifier=RandomForestClassifier()

classifier=classifier.fit(X_train,y_train)

predicted=classifier.predict(X_test)


#printing the results

print ('Confusion Matrix :')

print(confusion_matrix(y_test, predicted))

print ('Accuracy Score :',accuracy_score(y_test, predicted))

print ('Report : ')

print (classification_report(y_test, predicted))
```

Output:

```
Sum of NULL values in each column.
        0    0
        1    0
        2    0
        3    0
        4    0

Confusion Matrix :
                 [[16  0  0]
                  [ 0 17  1]
                  [ 0  0 11]]

Accuracy Score : 97.7

Report :
           precision   recall  f1-score   support
     0       1.00      1.00      1.00       16
     1       1.00      0.94      0.97       18
     2       0.92      1.00      0.96       11
avg/total    0.98      0.98      0.98       45
```

Understanding Logistic Regression

Pre-requisite: Linear Regression
This article discusses the basics of Logistic Regression and its implementation in Python. Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for given set of features(or inputs), X.
We can also say that the target variable is **categorical**. Based on the number of categories, Logistic regression can be classified as:

1. **binomial:** target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc.
2. **multinomial:** target variable can have 3 or more possible types which are not ordered(i.e. types have no quantitative significance) like “disease A” vs “disease B” vs “disease C”.

3. **ordinal:** it deals with target variables with ordered categories. For example, a test score can be categorized as:"very poor", "poor", "good", "very good". Here, each category can be given a score like 0, 1, 2, 3.

First of all, we explore the simplest form of Logistic Regression, i.e **Binomial Logistic Regression**.

Binomial Logistic Regression

Consider an example dataset which maps the number of hours of study with the result of an exam. The result can take only two values, namely passed(1) or failed(0):

Hours(x)	0.50	0.75	1.00	1.25	1.50	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	3.75	4.00	4.25	4.50	4.75	5.00	5.50
Pass(y)	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

So, we have

$$y = \begin{cases} 0, if fail \\ 1, if pass \end{cases}$$

i.e. y is a categorical target variable which can take only two possible type:"0" or "1".
In order to generalize our model, we assume that:

- The dataset has 'p' feature variables and 'n' observations.
- The feature matrix is represented as:

Here, x_{ij} denotes the values of j^{th} feature for i^{th} observation.
Here, we are keeping the convention of letting x_{i0} = 1. (Keep reading, you will understand the logic in a few moments).

- The i^{th} observation, x_i, can be represented as:

$$x_i = \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ . \\ . \\ x_{ip} \end{bmatrix}$$

- $h(x_i)$ represents the predicted response for i^{th} observation, i.e. x_i. The formula we use for calculating $h(x_i)$ is called **hypothesis**.

If you have gone though Linear Regression, you should recall that in Linear Regression, the hypothesis we used for prediction was:

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots.. + \beta_p x_{ip}$$

where, $\beta_0, \beta_1, , \beta_p$ are the regression coefficients.

Let regression coefficient matrix/vector, β be:

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ . \\ . \\ \beta_p \end{bmatrix}$$

Then, in a more compact form,

$$h(x_i) = \beta^T x_i$$

> *The reason for taking x_0 = 1 is pretty clear now.*
>
> *We needed to do a matrix product, but there was no*
>
> *actual x_0 multiplied to β_0 in original hypothesis formula. So, we defined x_0 = 1.*

Now, if we try to apply Linear Regression on above problem, we are likely to get continuous values using the hypothesis we discussed above. Also, it does not make sense for $h(x_i)$ to take values larger that 1 or smaller than 0.

So, some modifications are made to the hypothesis for classification:

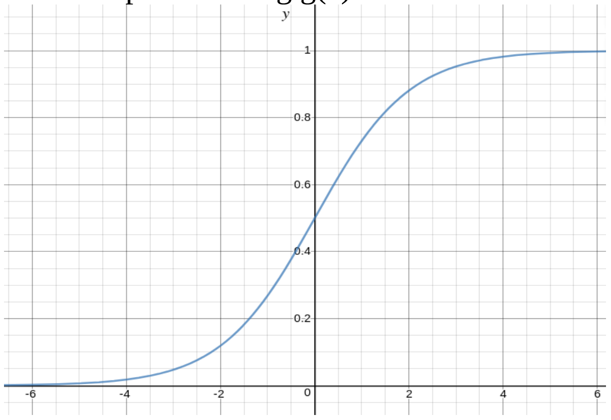
$$h(x_i) = g(\beta^T x_i) = \frac{1}{1 + e^{-\beta^T x_i}}$$

where,

$$g(z) = \frac{1}{1 + e^{-z}}$$

is called **logistic function** or the **sigmoid function**.

Here is a plot showing g(z):



We can infer from above graph that:

- g(z) tends towards 1 as
- g(z) tends towards 0 as
- g(z) is always bounded between 0 and 1

So, now, we can define conditional probabilities for 2 labels(0 and 1) for ith observation as:

$P(y_i = 1|x_i; \beta) = h(x_i)$

$P(y_i = 0|x_i; \beta) = 1 - h(x_i)$

We can write it more compactly as:

$P(y_i|x_i; \beta) = (h(x_i))^{y_i}(1 - h(x_i))^{1-y_i}$

Now, we define another term, **likelihood of parameters** as:

> *Likelihood is nothing but the probability of data(training examples), given a model and specific parameter values(here, β). It measures the support provided by the data for each possible value of the β. We obtain it by multiplying all $P(y_i|x_i)$ for given β.*

And for easier calculations, we take **log likelihood**:

$l(\beta) = log(L(\beta))$

or

$$l(\beta) = \sum_{i=1}^{n} y_i log(h(x_i)) + (1 - y_i) log(1 - h(x_i))$$

The **cost function** for logistic regression is proportional to inverse of likelihood of parameters. Hence, we can obtain an expression for cost function, J using log likelihood equation as:

$$J(\beta) = \sum_{i=1}^{n} -y_i log(h(x_i)) - (1 - y_i) log(1 - h(x_i))$$

and our aim is to estimate β so that cost function is minimized !!

Using Gradient descent algorithm

Firstly, we take partial derivatives of $J(\beta)$ w.r.t each $\beta_j \in \beta$ to derive the stochastic gradient descent rule(we present only the final derived value here):

$$\frac{\partial J(\beta)}{\partial \beta_j} = (h(x) - y)x_j$$

Here, y and h(x) represent the response vector and predicted response vector(respectively). Also, x_j is the vector representing the observation values for j^{th} feature.

Now, in order to get min $J(\beta)$,

$Repeat\{$

$$\beta_j := \beta_j - \alpha \sum_{i=1}^{n} (h(x_i) - y_i) x_{ij}$$

$(Simultaneously\ update\ all\ \beta_j)$

$\}$

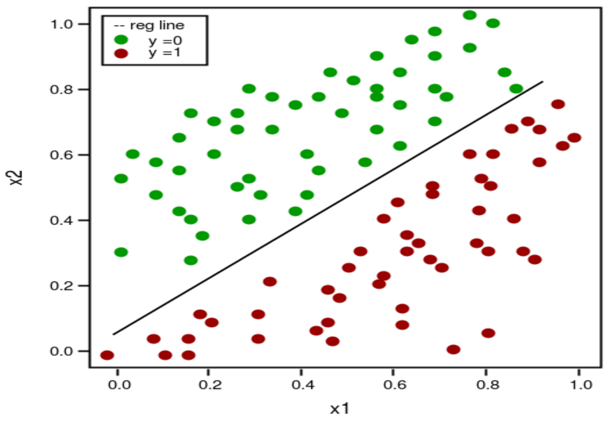
where α is called **learning rate** and needs to be set explicitly.

Let us see the python implementation of above technique on a sample dataset (download it from here):

2.25 2.50 2.75 3.00 3.25 3.50 3.75 4.00 4.25 4.50 4.75 5.00 5.50

```
import csv
import numpy as np
import matplotlib.pyplot as plt



def loadCSV(filename):
    '''
    function to load dataset
    '''
    with open(filename,"r") as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for i in range(len(dataset)):
            dataset[i] = [float(x) for x in dataset[i]]
    return np.array(dataset)
def normalize(X):
    '''
    function to normalize feature matrix, X
    '''
    mins = np.min(X, axis = 0)
    maxs = np.max(X, axis = 0)
    rng = maxs - mins
    norm_X = 1 - ((maxs - X)/rng)
    return norm_X

def logistic_func(beta, X):
    '''
    logistic(sigmoid) function
    '''
    return 1.0/(1 + np.exp(-np.dot(X, beta.T)))

def log_gradient(beta, X, y):
    '''
    logistic gradient function
    '''
    first_calc = logistic_func(beta, X) - y.reshape(X.shape[0], -1)
    final_calc = np.dot(first_calc.T, X)
    return final_calc


def cost_func(beta, X, y):
    '''
    cost function, J
    '''
    log_func_v = logistic_func(beta, X)
    y = np.squeeze(y)
    step1 = y * np.log(log_func_v)
    step2 = (1 - y) * np.log(1 - log_func_v)
    final = -step1 - step2
    return np.mean(final)


def grad_desc(X, y, beta, lr=.01, converge_change=.001):
    '''
    gradient descent function
    '''
    cost = cost_func(beta, X, y)
    change_cost = 1
    num_iter = 1

    while(change_cost > converge_change):
        old_cost = cost
        beta = beta - (lr * log_gradient(beta, X, y))
        cost = cost_func(beta, X, y)
        change_cost = old_cost - cost
        num_iter += 1

    return beta, num_iter



def pred_values(beta, X):
    '''
    function to predict labels
    '''
    pred_prob = logistic_func(beta, X)
    pred_value = np.where(pred_prob >= .5, 1, 0)
    return np.squeeze(pred_value)

def plot_reg(X, y, beta):
    '''
    function to plot decision boundary
    '''
    # labelled observations
    x_0 = X[np.where(y == 0.0)]
    x_1 = X[np.where(y == 1.0)]

    # plotting points with diff color for diff label
    plt.scatter([x_0[:, 1]], [x_0[:, 2]], c='b', label='y = 0')
    plt.scatter([x_1[:, 1]], [x_1[:, 2]], c='r', label='y = 1')

    # plotting decision boundary
    x1 = np.arange(0, 1, 0.1)
    x2 = -(beta[0,0] + beta[0,1]*x1)/beta[0,2]
    plt.plot(x1, x2, c='k', label='reg line')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.show()

if __name__ == "__main__":
    # load the dataset
    dataset = loadCSV('dataset1.csv')

    # normalizing feature matrix
    X = normalize(dataset[:, :-1])

    # stacking columns wth all ones in feature matrix
    X = np.hstack((np.matrix(np.ones(X.shape[0])).T, X))
    # response vector
    y = dataset[:, -1]

    # initial beta values
    beta = np.matrix(np.zeros(X.shape[1]))

    # beta values after running gradient descent
    beta, num_iter = grad_desc(X, y, beta)

    # estimated beta values and number of iterations
    print("Estimated regression coefficients:", beta)
    print("No. of iterations:", num_iter)

    # predicted labels
    y_pred = pred_values(beta, X)

    # number of correctly predicted labels
    print("Correctly predicted labels:", np.sum(y == y_pred))
    # plotting regression line
    plot_reg(X, y, beta)
```

```
Estimated regression coefficients: [[  1.70474504  15.04062212 -20.47216021]]
No. of iterations: 2612
Correctly predicted labels: 100
```



Note: Gradient descent is one of the many way to estimate .
Basically, these are more advanced algorithms which can be easily run in Python once you have defined your cost function and your gradients. These algorithms are:

- BFGS(Broyden–Fletcher–Goldfarb–Shanno algorithm)
- L-BFGS(Like BFGS but uses limited memory)
- Conjugate Gradient

Advantages/disadvantages of using any one of these algorithms over Gradient descent:

- Advantages
  - Don’t need to pick learning rate
  - Often run faster (not always the case)
  - Can numerically approximate gradient for you (doesn’t always work out well)
- Disadvantages
  - More complex
  - More of a black box unless you learn the specifics

Multinomial Logistic Regression

In Multinomial Logistic Regression, the output variable can have **more than two possible discrete outputs**. Consider the Digit Dataset. Here, the output variable is the digit value which can take values out of (0, 12, 3, 4, 5, 6, 7, 8, 9).
Given below is the implementation of Multinomial Logisitc Regression using scikit-learn to make predictions on digit dataset.

```
from sklearn import datasets, linear_model, metrics

# load the digit dataset
digits = datasets.load_digits()

# defining feature matrix(X) and response vector(y)
X = digits.data
y = digits.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=1)

# create logistic regression object
reg = linear_model.LogisticRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# making predictions on the testing set
y_pred = reg.predict(X_test)

# comparing actual response values (y_test) with predicted response values (y_pred)
print("Logistic Regression model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
```

```
Logistic Regression model accuracy(in %): 95.6884561892
```

At last, here are some points about Logistic regression to ponder upon:

- Does NOT assume a linear relationship between the dependent variable and the independent variables, but it does assume linear relationship between the **logit of the explanatory variables** and the **response**.
- Independent variables can be even the power terms or some other nonlinear transformations of the original independent variables.
- The dependent variable does NOT need to be normally distributed, but it typically assumes a distribution from an exponential family (e.g. binomial, Poisson, multinomial, normal,…); binary logistic regression assume binomial distribution of the response.
- The homogeneity of variance does NOT need to be satisfied.
- Errors need to be independent but NOT normally distributed.
- It uses maximum likelihood estimation (MLE) rather than ordinary least squares (OLS) to estimate the parameters, and thus relies on **large-sample approximations**.

Multivariate Regression

Prerequisite Article-Machine Learning

The goal in any data analysis is to extract from raw information the accurate estimation. One of the most important and common question concerning if there is statistical relationship between a response variable (Y) and explanatory variables (Xi). An option to answer this question is to employ regression analysis in order to *model* its relationship. Further it can be used to *predict* the response variable for any arbitrary set of explanatory variables.

The Problem:

Multivariate Regression is one of the simplest Machine Learning Algorithm. It comes under the class of Supervised Learning Algorithms i.e, when we are provided with training dataset. Some of the problems that can be solved using this model are:

- A researcher has collected data on three psychological variables, four academic variables (standardized test scores), and the type of educational program the student is in for 600 high school students. She is interested in how the set of psychological variables is related to the academic variables and the type of program the student is in.

- A doctor has collected data on cholesterol, blood pressure, and weight. She also collected data on the eating habits of the subjects (e.g., how many ounces of red meat, fish, dairy products, and chocolate consumed per week). She wants to investigate the relationship between the three measures of health and eating habits.

- A property dealer wants to set housing prices which are based various factors like Size of house, No of bedrooms, Age of house, etc. We shall discuss the algorithm further using this example.

The Solution:

The solution is divided into various parts.

- Selecting the features: Finding the features on which a response variable depends (or not) is one of the most important steps in Multivariate Regression. To make our analysis simple, we assume that the features on which the response variable is dependent are already selected.

- Normalizing the features: The features are then scaled in order to bring them in range of (0,1) to make better analysis. This can be done by changing the value of each feature by:

$$Xi = \frac{x_i - \mu_i}{\delta_i}, Where, x_i = Training\ examples\ for\ ith\ feature,$$
$$\mu_i = mean\ of\ ith\ feature.$$
$$\delta_i = range\ of\ ith\ feature.$$

- Selecting Hypothesis and Cost function: A hypothesis is a predicted value of the response variable represented by h(x). Cost function defines the cost for wrongly predicting hypothesis. It should be as small as possible. We choose hypothesis function as linear combination of features X.

$$h(x^i) = \theta_0 + \theta_1 x_1^i + ... + \theta_n x_n^i$$
$$where \Theta = [\theta_0 + \theta_1 + ... + \theta_n]^T is\ the\ parameter\ vector,$$
$$and\ x_i^j = value\ of\ ith\ feature\ in\ jth\ training\ example.$$
$$And\ the\ cost\ function\ as\ sum\ of\ squared\ error\ over\ all\ training\ examples.$$
$$J(\theta) = \frac{1}{2m * \sum(h_\theta(x^i) - y^i)^2}$$

- Minimizing the Cost function: Next some Cost minimization algorithm runs over the datasets which adjusts the parameters of the hypothesis. Once the cost function is minimized for the training dataset, it should also be minimized for an arbitrary dataset if the relation is universal. Gradient descent algorithm is a good choice for minimizing the cost function in case of multivariate regression.

- Testing the hypothesis: The hypothesis function is then tested over the test set to check its correctness and efficiency.

Implementation:

Multivariate regression technique can be implemented efficiently with the help of matrix operations. With python, it can be implemented using “numpy” library which contains definitions and operations for matrix object.

The code requires “numpy” library for python (www.numpy.org/) which is not installed on GfG servers and thus the code is unable to run on gfg IDE. However link to the code is:

```
import sys

import numpy


def GradDesc(X, y, theta, alpha, num_iters):


    for i in range(num_iters):

        h_theta = X * theta.T;

        temp = h_theta - y;
        delta = (alpha / len(y))*(temp.T*X);

        theta = theta - delta;


    return theta;


inp = sys.stdin.readline().split();

n = int(inp[0]);

m = int(inp[1]);

x=numpy.zeros(m*(n+1)).reshape((m,n+1));

y = numpy.zeros(m).reshape((m,1));


for i in range(m):

    inp = sys.stdin.readline().split();

    x[i][0] = 1.0;

    for j in range(n):

        x[i][j+1] = float(inp[j]);

    y[i][0] = float(inp[n]);

x = numpy.matrix(x);

y = numpy.matrix(y);


alpha = 0.1;

num_iters = 1500;


theta = numpy.matrix(numpy.zeros(n+1));


theta = GradDesc(x, y, theta, alpha, num_iters);

t = int(sys.stdin.readline());

t_matrix = numpy.zeros(t*(n+1)).reshape(t,n+1);

for i in range(t):

    inp = sys.stdin.readline().split();

    t_matrix[i][0] = 1;

    for j in range(n):
            t_matrix[i][j+1] = float(inp[j]);
t_matrix = numpy.matrix(t_matrix);
final = t_matrix * theta.T;
for i in range(t):
      print(final[i]);
```

Output:

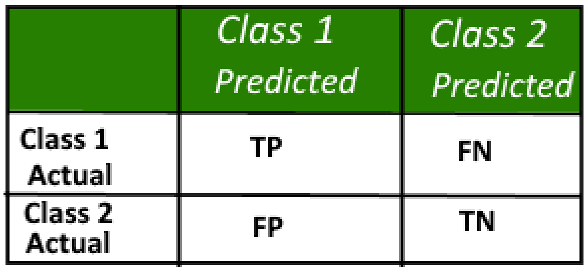
```
2 7
0.18 0.89 109.85
1.0 0.26 155.72
0.92 0.11 137.66
0.07 0.37 76.17
0.85 0.16 139.75
0.99 0.41 162.6
0.87 0.47 151.77
4
0.49 0.18
0.57 0.83
0.56 0.64
0.76 0.18
```

Confusion Matrix in Machine Learning

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix.
A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.
It allows easy identification of confusion between classes e.g. one class is commonly mislabeled as the other. Most performance measures are computed from the confusion matrix.

This article aims at:
1. What the confusion matrix is and why you need to use it.
2. How to calculate a confusion matrix for a 2-class classification problem from scratch.
3. How to create a confusion matrix in Python.

Confusion Matrix:
A confusion matrix is a summary of prediction results on a classification problem.
The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.
The confusion matrix shows the ways in which your classification model is confused when it makes predictions.
It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.



	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

Here,
• Class 1 : Positive
• Class 2 : Negative

Definition of the Terms:
• Positive (P) : Observation is positive (for example: is an apple).
• Negative (N) : Observation is not positive (for example: is not an apple).
• True Positive (TP) : Observation is positive, and is predicted to be positive.
• False Negative (FN) : Observation is positive, but is predicted negative.
• True Negative (TN) : Observation is negative, and is predicted to be negative.
• False Positive (FP) : Observation is negative, but is predicted positive.

Classification Rate/Accuracy:
Classification Rate or Accuracy is given by the relation:

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

Recall:
Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN).

Recall is given by the relation:

$$\textit{Recall} = \frac{TP}{TP + FN}$$

Precision:
To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (small number of FP).
Precision is given by the relation:

$$\textit{Precision} = \frac{TP}{TP + FP}$$

High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)

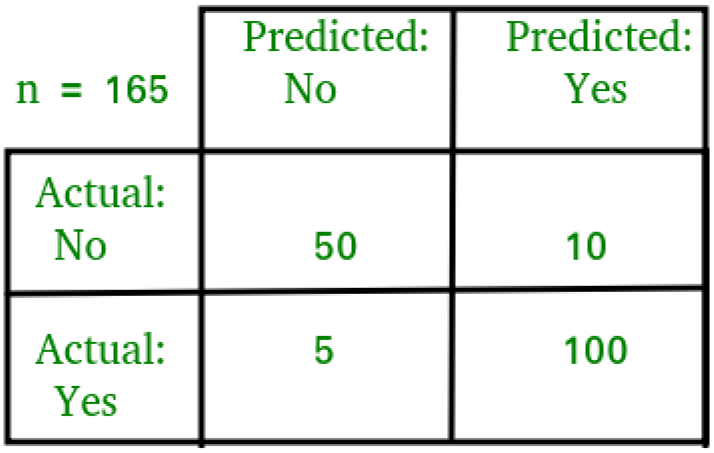
F-measure:
Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.
The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$\textit{F - measure} = \frac{2*\text{Recall}*\text{Precision}}{\text{Recall} + \text{Precision}}$$

Let's consider an example now, in which we have infinite data elements of class B and a single element of class A and the model is predicting class A against all the instances in the test data.
Here,
Precision : 0.0
Recall : 1.0

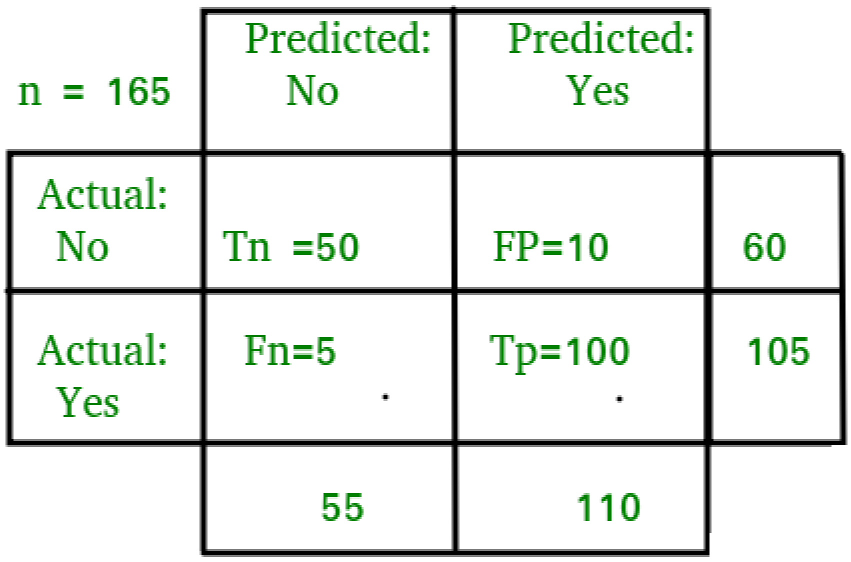
Now:
Arithmetic mean: 0.5
Harmonic mean: 0.0
When taking the arithmetic mean, it would have 50% correct. Despite being the worst possible outcome! While taking the harmonic mean, the F-measure is 0.

Example to interpret confusion matrix:



n = 165	Predicted: No	Predicted: Yes
Actual: No	50	10
Actual: Yes	5	100

For the simplification of the above confusion matrix i have added all the terms like TP,FP,etc and the row and column totals in the following image:



n = 165	Predicted: No	Predicted: Yes	
Actual: No	Tn =50	FP=10	60
Actual: Yes	Fn=5	Tp=100	105
	55	110	

Now,
Classification Rate/Accuracy:
Accuracy = (TP + TN) / (TP + TN + FP + FN)= (100+50) /(100+5+10+50)= 0.90

Recall: Recall gives us an idea about when it's actually yes, how often does it predict yes.
Recall=TP / (TP + FN)=100/(100+5)=0.95

Precision: Precsion tells us about when it predicts yes, how often is it correct.
Precision = TP / (TP + FP)=100/ (100+10)=0.91

F-measure:
Fmeasure=(2*Recall*Precision)/(Recall+Presision)=(2*0.95*0.91)/(0.91+0.95)=0.92

Here is a python script which demonstrates how to create a confusion matrix on a predicted model.For this, we have to import confusion matrix module from sklearn library which helps us to generate the confusion matrix.

Below is the Python implementation of above explanation :
Note that this program might not run on Geeksforgeeks IDE, but it can run easily on your local python interpreter, provided, you have installed the required libraries.

```
# Python script for confusion matrix creation.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
actual = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
predicted = [1, 0, 0, 1, 0, 0, 1, 1, 1, 0]
results = confusion_matrix(actual, predicted)
print 'Confusion Matrix :'
print(results)
print 'Accuracy Score :',accuracy_score(actual, predicted)
print 'Report : '
print classification_report(actual, predicted)
```

OUTPUT ->

```
Confusion Matrix :

[[4 2]

 [1 3]]

Accuracy Score : 0.7

Report :

              precision    recall  f1-score   support

          0       0.80      0.67      0.73         6

          1       0.60      0.75      0.67         4

avg / total       0.72      0.70      0.70        10
```

Linear Regression (Python Implementation)

This article discusses the basics of linear regression and its implementation in Python programming language.

Linear regression is a statistical approach for modelling relationship between a dependent variable with a given set of independent variables.

Note: In this article, we refer dependent variables as **response** and independent variables as **features** for simplicity.

In order to provide a basic understanding of linear regression, we start with the most basic version of linear regression, i.e. **Simple linear regression**.

Simple Linear Regression

Simple linear regression is an approach for predicting a **response** using a **single feature**.

It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

Let us consider a dataset where we have a value of response y for every feature x:

x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

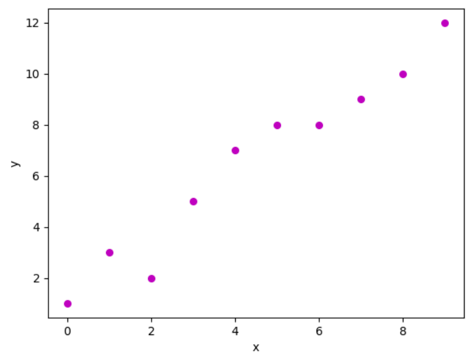
For generality, we define:

x as **feature vector**, i.e x = [x_1, x_2, …., x_n],

y as **response vector**, i.e y = [y_1, y_2, …., y_n]

for **n** observations (in above example, n=10).

A scatter plot of above dataset looks like:-



Now, the task is to find a **line which fits best** in above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in dataset)

This line is called **regression line**.

The equation of regression line is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_i$$

Here,

- h(x_i) represents the **predicted response value** for ith observation.
- b_0 and b_1 are regression coefficients and represent **y-intercept** and **slope** of regression line respectively.

To create our model, we must “learn” or estimate the values of regression coefficients b_0 and b_1. And once we’ve estimated these coefficients, we can use the model to predict responses!

In this article, we are going to use the **Least Squares technique**.

Now consider:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

Here, e_i is **residual error** in ith observation.
So, our aim is to minimize the total residual error.

We define the squared error or cost function, J as:

$$J(\beta_0, \beta_1) = \frac{1}{2n} \sum_{i=1}^{n} \varepsilon_i^2$$

and our task is to find the value of b_0 and b_1 for which J(b_0,b_1) is minimum!

Without going into the mathematical details, we present the result here:

$$\beta_1 = \frac{SS_{xy}}{SS_{xx}}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

where SS_xy is the sum of cross-deviations of y and x:

$$SS_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} y_i x_i - n\bar{x}\bar{y}$$

and SS_xx is the sum of squared deviations of x:

$$SS_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - n(\bar{x})^2$$

Note: The complete derivation for finding least squares estimates in simple linear regression can be found here.

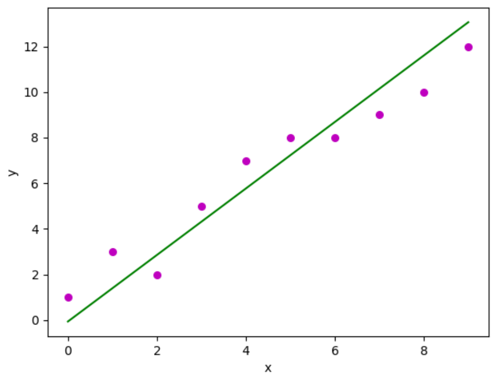
Given below is the python implementation of above technique on our small dataset:

```
import numpy as np

import matplotlib.pyplot as plt



def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return(b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

Output of above piece of code is:

```
Estimated coefficients:
b_0 = -0.0586206896552
b_1 = 1.45747126437
```

And graph obtained looks like this:



Multiple linear regression

Multiple linear regression attempts to model the relationship between **two or more features** and a response by fitting a linear equation to observed data.

Clearly, it is nothing but an extension of Simple linear regression.

Consider a dataset with **p** features(or independent variables) and one response(or dependent variable). Also, the dataset contains **n** rows/observations.

We define:

X (**feature matrix**) = a matrix of size **n X p** where x_{ij} denotes the values of jth feature for ith observation.

So,

$$\begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \vdots & x_{np} \end{pmatrix}$$

and

y (**response vector**) = a vector of size **n** where y_{i} denotes the value of response for ith observation.

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix}$$

The **regression line** for **p** features is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}$$

where h(x_i) is **predicted response value** for ith observation and b_0, b_1, …, b_p are the **regression coefficients**.

Also, we can write:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i$$
$$or$$
$$y_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

where e_i represents **residual error** in ith observation.

We can generalize our linear model a little bit more by representing feature matrix **X** as:

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

So now, the linear model can be expressed in terms of matrices as:

$$y = X\beta + \varepsilon$$

where,

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ \beta_p \end{bmatrix}$$

and

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_n \end{bmatrix}$$

Now, we determine **estimate of b**, i.e. b' using **Least Squares method**.

As already explained, Least Squares method tends to determine b' for which total residual error is minimized.

We present the result directly here:

$$\hat{\beta} = (X'X)^{-1}X'y$$

where ' represents the transpose of the matrix while -1 represents the matrix inverse.

Knowing the least square estimates, b', the multiple linear regression model can now be estimated as:

$$\hat{y} = X\hat{\beta}$$

where y' is **estimated response vector**.

Note: The complete derivation for obtaining least square estimates in multiple linear regression can be found here.

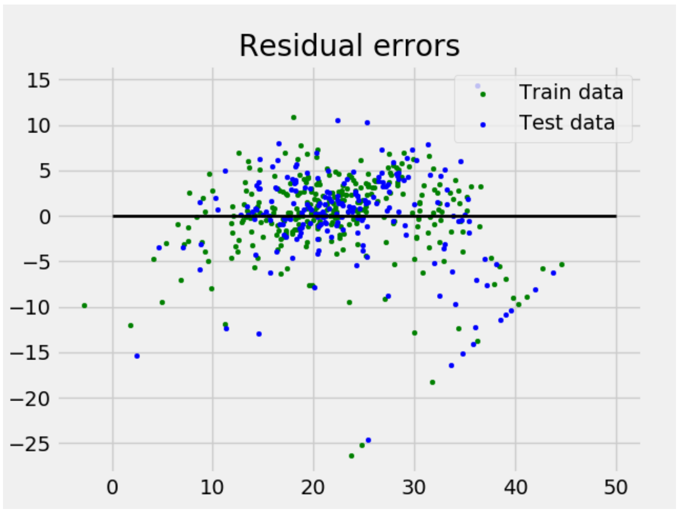
Given below is the implementation of multiple linear regression technique on the Boston house pricing dataset using Scikit-learn.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics


# load the boston dataset
boston = datasets.load_boston(return_X_y=False)


# defining feature matrix(X) and response vector(y)
X = boston.data
y = boston.target


# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=1)


# create linear regression object
reg = linear_model.LinearRegression()


# train the model using the training sets
reg.fit(X_train, y_train)


# regression coefficients
print('Coefficients: \n', reg.coef_)


# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))


# plot for residual error


## setting plot style
plt.style.use('fivethirtyeight')


## plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
            color = "green", s = 10, label = 'Train data')
```

```
## plotting residual errors in test data
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,
            color = "blue", s = 10, label = 'Test data')


## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)


## plotting legend
plt.legend(loc = 'upper right')


## plot title
plt.title("Residual errors")


## function to show plot
plt.show()
```

Output of above program looks like this:

```
Coefficients:
[ -8.80740828e-02   6.72507352e-02   5.10280463e-02   2.18879172e+00
-1.72283734e+01   3.62985243e+00   2.13933641e-03  -1.36531300e+00
2.88788067e-01  -1.22618657e-02  -8.36014969e-01   9.53058061e-03
-5.05036163e-01]
Variance score: 0.720898784611
```



and **Residual Error plot** looks like this:

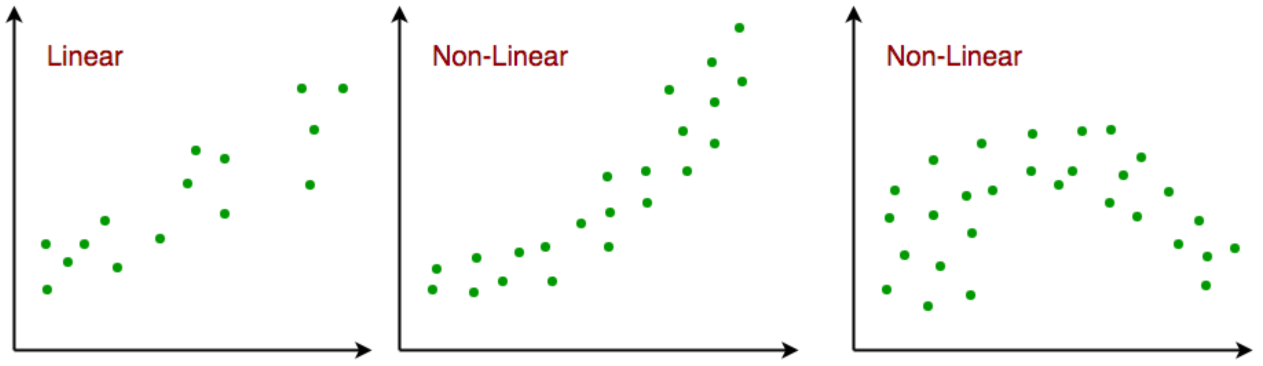
In above example, we determine accuracy score using **Explained Variance Score**.
We define:
explained_variance_score = 1 – Var{y – y'}/Var{y}
where y' is the estimated target output, y the corresponding (correct) target output, and Var is Variance, the

square of the standard deviation.
The best possible score is 1.0, lower values are worse.

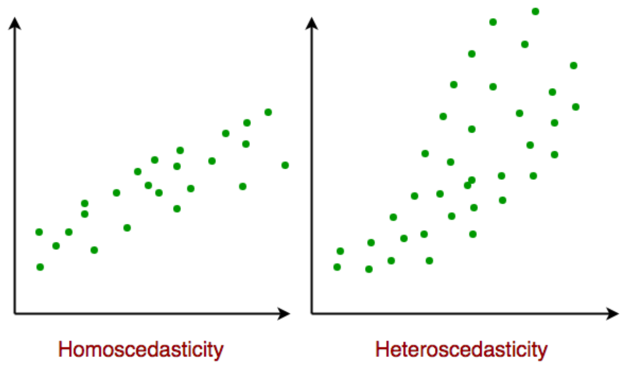
Assumptions

Given below are the basic assumptions that a linear regression model makes regarding a dataset on which it is applied:

- **Linear relationship**: Relationship between response and feature variables should be linear. The linearity assumption can be tested using scatter plots. As shown below, 1st figure represents linearly related variables where as variables in 2nd and 3rd figure are most likely non-linear. So, 1st figure will give better predictions using linear regression.



- **Little or no multi-collinearity**: It is assumed that there is little or no multicollinearity in the data. Multicollinearity occurs when the features (or independent variables) are not independent from each other.
- **Little or no auto-correlation**: Another assumption is that there is little or no autocorrelation in the data. Autocorrelation occurs when the residual errors are not independent from each other. You can refer here for more insight into this topic.
- **Homoscedasticity**: Homoscedasticity describes a situation in which the error term (that is, the "noise" or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables. As shown below, figure 1 has homoscedasticity while figure 2 has heteroscedasticity.
-



As we reach to the end of this article, we discuss some applications of linear regression below.

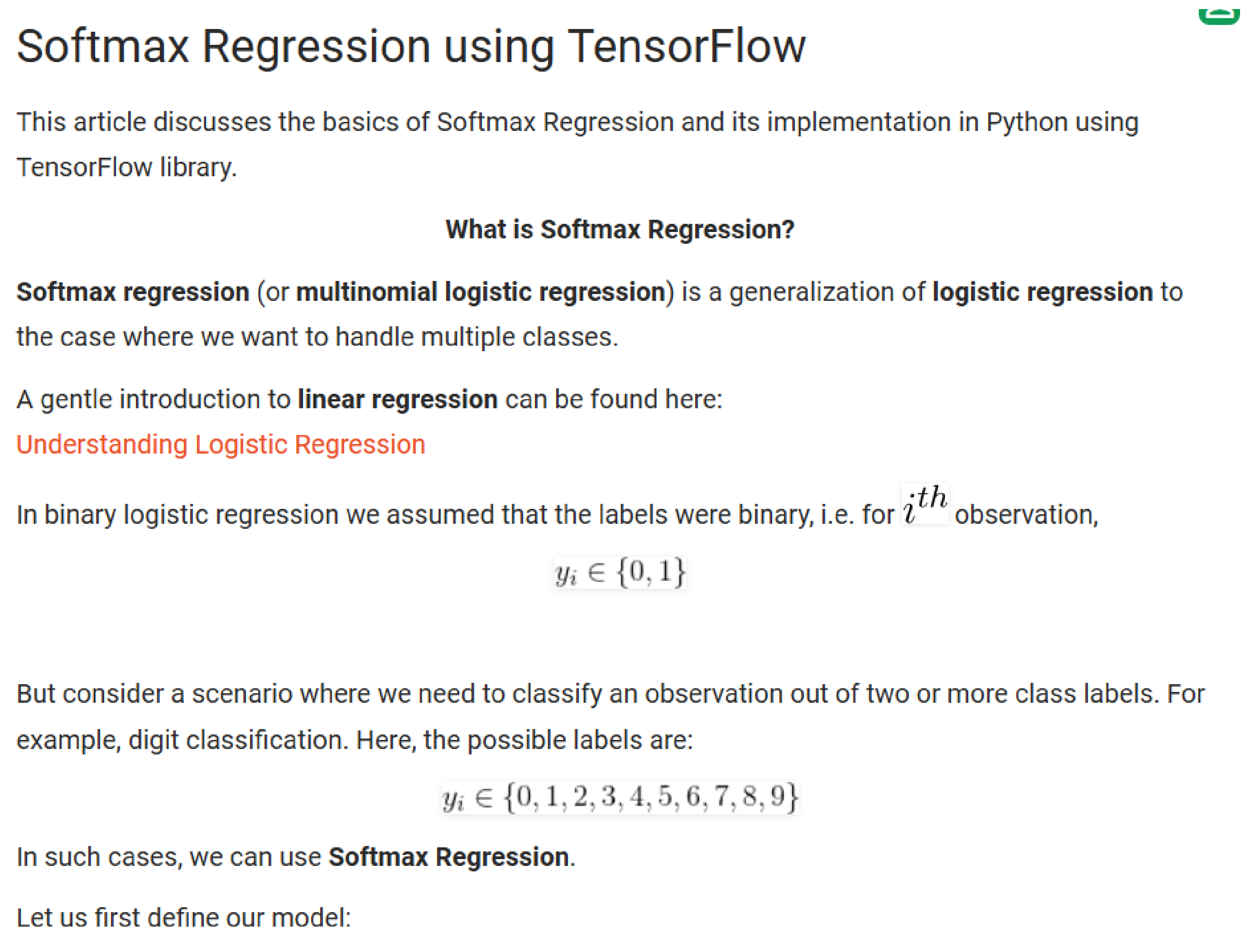
Applications:

1. Trend lines: A trend line represents the variation in some quantitative data with passage of time (like GDP, oil prices, etc.). These trends usually follow a linear relationship. Hence, linear regression can be applied to predict future values. However, this method suffers from a lack of scientific validity in cases where other potential changes can affect the data.

2. Economics: Linear regression is the predominant empirical tool in economics. For example, it is used to predict consumption spending, fixed investment spending, inventory investment, purchases of a country's exports, spending on imports, the demand to hold liquid assets, labor demand, and labor supply.

3. Finance: Capital price asset model uses linear regression to analyze and quantify the systematic risks of an investment.

4. Biology: Linear regression is used to model causal relationships between parameters in biological systems.



Softmax Regression using TensorFlow

This article discusses the basics of Softmax Regression and its implementation in Python using TensorFlow library.

What is Softmax Regression?

Softmax regression (or **multinomial logistic regression**) is a generalization of **logistic regression** to the case where we want to handle multiple classes.

A gentle introduction to **linear regression** can be found here:
Understanding Logistic Regression

In binary logistic regression we assumed that the labels were binary, i.e. for i^{th} observation,

$$y_i \in \{0, 1\}$$

But consider a scenario where we need to classify an observation out of two or more class labels. For example, digit classification. Here, the possible labels are:

$$y_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

In such cases, we can use **Softmax Regression**.

Let us first define our model:

- Let the dataset have 'm' features and 'n' observations. Also, there are 'k' class labels, i.e every observation can be classified as one of the 'k' possible target values. For example, if we have a dataset of 100 handwritten digit images of vector size 28×28 for digit classification, we have, n = 100, m = 28×28 = 784 and k = 10.
- **Feature matrix**

  The feature matrix, X, is represented as:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1m} \\ 1 & x_{21} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

  Here, x_{ij} denotes the values of j^{th} feature for i^{th} observation. The matrix has dimensions: $nX(m+1)$

- **Weight matrix**

  We define a weight matrix, W as:

$$\mathbf{W} = \begin{pmatrix} w_{01} & w_{02} & \cdots & w_{0k} \\ w_{11} & w_{12} & \cdots & w_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mk} \end{pmatrix}$$

  Here, w_{ij} represents the weight assigned to i^{th} feature for j^{th} class label. The matrix has dimensions: $(m+1)Xk$. Initially, weight matrix is filled using some normal distribution.

- **Logit score matrix**

  Then, we define our net input matrix(also called **logit score matrix**), Z, as:

$$Z = XW$$

  The matrix has dimensions: nXk.

Currently, we are taking an extra column in feature matrix, X and an extra row in weight matrix, W. These extra columns and rows correspond to the bias terms associated with each prediction. This could be simplified by defining an extra matrix for bias, b of size nXk where $b_{ij} = w_{0j}$. (In practice, all we need is a vector of size k and some broadcasting techniques for the bias terms!)

So, the final score matrix, Z is:

$$Z = XW + b$$

where X matrix has dimensions nXm while W has dimensions mXk. But Z matrix still has same value and dimensions!

But what does matrix Z signify? Actually, Z_{ij} is the likelihood of label j for i^{th} observation. It is not a proper probability value but can be considered as a score given to each class label for each observation!

Let us define Z_i as the **logit score vector** for i^{th} observation.

For example, let the vector $Z_5 = [1.1, 2.0, 3.1, 5.2, 1.0, 1.5, 0.2, 0.1, 1.2, 0.4]$ represents the score for each of the class labels $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ in handwritten digit classification problem for 5^{th} observation. Here, the max score is 5.2 which corresponds to class label '3'. Hence, our model currently predicts the 5^{th} observation/image as '3'.

- **Softmax layer**
  It is harder to train the model using score values since it is hard to differentiate them while implementing **Gradient Descent algorithm** for minimizing the cost function. So, we need some function which normalizes the logit scores as well as makes them easily differentiable!In order to convert the score matrix Z to probabilities, we use **Softmax function**.
  For a vector y, softmax function $S(y)$ is defined as:

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=0} e^{y_j}}$$

So, softmax function will do 2 things:

```
1. convert all scores to probabilities.
2. sum of all probabilities is 1.
```

Recall that in Binary Logistic classifier, we used **sigmoid function** for the same task. Softmax function is nothing but a generalization of sigmoid function! Now, this softmax function computes the probability that the i^{th} training sample belongs to class j given the logits vector Z_i as:

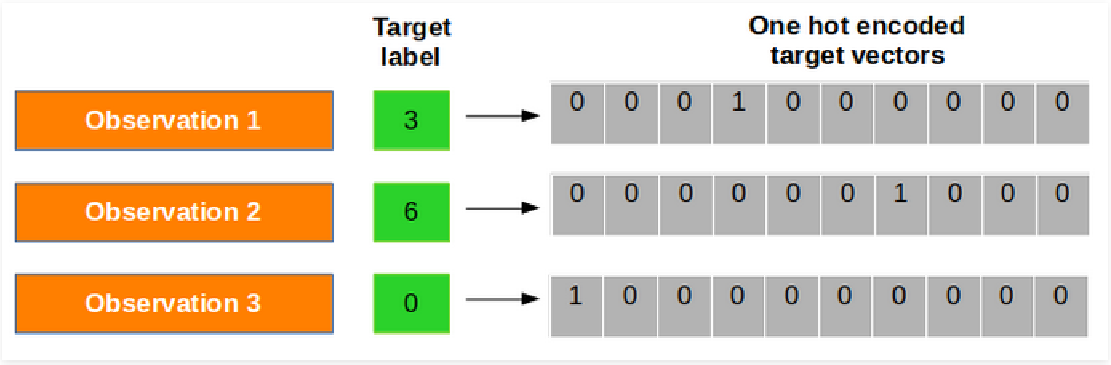
$$P(y=j|Z_i) = [S(Z_i)]_j = \frac{e^{Z_{ij}}}{\sum_{p=0}^{k} e^{Z_{ip}}}$$

In vector form, we can simply write:

$$P(y|Z_i) = S(Z_i)$$

For simplicity, let S_i denote the **softmax probability vector** for i^{th} observation.

- **One-hot encoded target matrix**
  Since softmax function provides us with a vector of probabilities of each class label for a given observation, we need to convert target vector in the same format to calculate the cost function! Corresponding to each observation, there is a target vector (instead of a target value!) composed of only zeros and ones where only correct label is set as 1. This technique is called **one-hot encoding**.See the diagram given below for a better understanding:



Now, we denote one-hot encoding vector for i^{th} observation as T_i

- **Cost function**

  Now, we need to define a cost function for which, we have to compare the softmax probabilities and one-hot encoded target vector for similarity. We use the concept of **Cross-Entropy** for the same. The **Cross-entropy** is a **distance calculation function** which takes the calculated probabilities from softmax function and the created one-hot-encoding matrix to calculate the distance. For the right target classes, the distance values will be lesser, and the distance values will be larger for the wrong target classes.We define cross entropy, $D(S_i, T_i)$ for i^{th} observation with softmax probability vector, S_i and one-hot target vector, T_i as:

  $$D(S_i, T_i) = -\sum_{j=1}^{k} T_{ij} log S_{ij}$$

  And now, cost function, J can be defined as the average cross entropy, i.e:

  $$J(W, b) = \frac{1}{n}\sum_{i=1}^{n} D(S_i, T_i)$$

  and the task is to minimize this cost function!

- **Gradient Descent algorithm**

  In order to learn our softmax model via gradient descent, we need to compute the derivative:

  $$\Delta_W J(W, b)$$

  and

  $$\Delta_b J(W, b)$$

which we then use to update the weights and biases in opposite direction of the gradient:

$$w_j := w_j - \alpha \Delta_W J(W, b)$$

and

$$b_j := b_j - \alpha \Delta_b J(W, b)$$

for each class j where $j \in 1, 2, .., k$ and α is learning rate.Using this cost gradient, we iteratively update the weight matrix until we reach a specified number of epochs (passes over the training set) or reach the desired cost threshold.

Implementation

Let us now implement **Softmax Regression** on the MNIST handwritten digit dataset using **TensorFlow** library.

For a gentle introduction to **TensorFlow**, follow this tutorial:
Introduction to TensorFlow

Step 1: Import the dependencies

First of all, we import the dependencies.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

Step 2: Download the data

TensorFlow allows you to download and read in the MNIST data automatically. Consider the code given below. It will download and save data to the folder, **MNIST_data**, in your current project directory and load it in current program.

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/",one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

Step 3: Understanding data

Now, we try to understand the structure of the dataset.

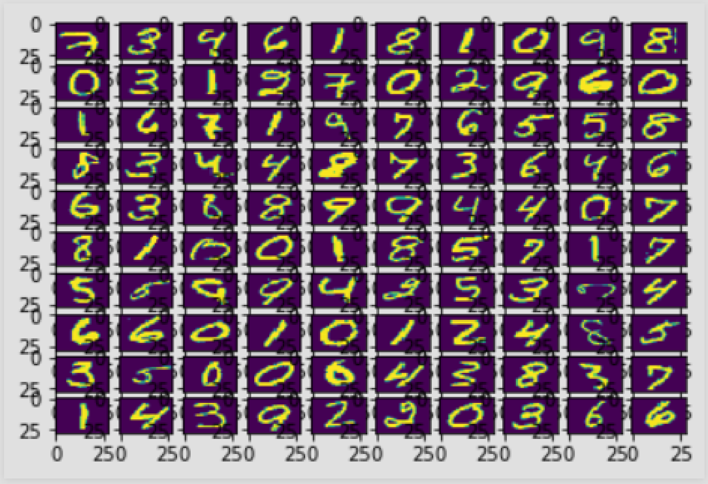
The MNIST data is split into three parts: 55,000 data points of training data (**mnist.train**), 10,000 points of test data (**mnist.test**), and 5,000 points of validation data (**mnist.validation**).

Each image is 28 pixels by 28 pixels which has been flattened into 1-D numpy array of size 784. Number of class labels is 10. Each target label is already provided in one-hot encoded form.

```
print("Shape of feature matrix:", mnist.train.images.shape)
print("Shape of target matrix:", mnist.train.labels.shape)
print("One-hot encoding for 1st observation:\n",
mnist.train.labels[0])

# visualize data by plotting images
fig,ax = plt.subplots(10,10)
k = 0
for i in range(10):
    for j in range(10):
        ax[i][j].imshow(mnist.train.images[k].reshape(28,28),
aspect='auto')
        k += 1
plt.show()
```

Output:

```
Shape of feature matrix: (55000, 784)
Shape of target matrix: (55000, 10)
One-hot encoding for 1st observation:
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
```



Step 4: Defining computation graph

Now, we create a computation graph.

```
# number of features
num_features = 784
# number of target labels
num_labels = 10
# learning rate (alpha)
learning_rate = 0.05
# batch size
batch_size = 128
# number of epochs
num_steps = 5001

# input data
train_dataset = mnist.train.images
train_labels = mnist.train.labels
test_dataset = mnist.test.images
test_labels = mnist.test.labels
valid_dataset = mnist.validation.images
valid_labels = mnist.validation.labels

# initialize a tensorflow graph
graph = tf.Graph()

with graph.as_default():
    """
    defining all the nodes
    """

    # Inputs
    tf_train_dataset = tf.placeholder(tf.float32, shape=(batch_size, num_features))
    tf_train_labels = tf.placeholder(tf.float32, shape=(batch_size, num_labels))
    tf_valid_dataset = tf.constant(valid_dataset)
    tf_test_dataset = tf.constant(test_dataset)

    # Variables.
    weights = tf.Variable(tf.truncated_normal([num_features, num_labels]))
    biases = tf.Variable(tf.zeros([num_labels]))

    # Training computation.
    logits = tf.matmul(tf_train_dataset, weights) + biases
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                        labels=tf_train_labels, logits=logits))

    # Optimizer.
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

    # Predictions for the training, validation, and test data.
    train_prediction = tf.nn.softmax(logits)
    valid_prediction = tf.nn.softmax(tf.matmul(tf_valid_dataset, weights) + biases)
    test_prediction = tf.nn.softmax(tf.matmul(tf_test_dataset, weights) + biases)
```

Some important points to note:

- For the training data, we use a placeholder that will be fed at run time with a training minibatch. The technique of using minibatches for training model using gradient descent is termed as **Stochastic Gradient Descent**.

In both gradient descent (GD) and stochastic gradient descent (SGD), you update a set of parameters in an iterative manner to minimize an error function. While in GD, you have to run through ALL the samples in your training set to do a single update for a parameter in a particular iteration, in SGD, on the other hand, you use ONLY ONE or SUBSET of training sample from your training set to do the update for a parameter in a particular iteration. If you use SUBSET, it is called Minibatch Stochastic gradient Descent. Thus, if the number of training samples are large, in fact very large, then using gradient descent may take too long because in every iteration when you are updating the values of the parameters, you are running through the complete training set. On the other hand, using SGD will be faster because you use only one training sample and it starts improving itself right away from the first sample. SGD often converges much faster compared to GD but the error function is not as well minimized as in the case of GD. Often in most cases, the close approximation that you get in SGD for the parameter values are enough because they reach the optimal values and keep oscillating there.

- The weight matrix is initialized using random values following a (truncated) normal distribution. This is achieved using **tf.truncated_normal** method. The biases get initialized to zero using **tf.zeros** method.
- Now, we multiply the inputs with the weight matrix, and add biases. We compute the softmax and cross-entropy using **tf.nn.softmax_cross_entropy_with_logits** (it's one operation in TensorFlow, because it's very common, and it can be optimized). We take the average of this cross-entropy across all training examples using **tf.reduce_mean** method.
- We are going to minimize the loss using gradient descent. For this, we use **tf.train.GradientDescentOptimizer**.
- **train_prediction**, **valid_prediction** and **test_prediction** are not part of training, but merely here so that we can report accuracy figures as we train.

Step 5: Running the computation graph

Since we have already built the computation graph, now it's time to run it through a session.

```
# utility function to calculate accuracy
def accuracy(predictions, labels):
    correctly_predicted = np.sum(np.argmax(predictions, 1) == np.argmax(labels, 
1))
    accu = (100.0 * correctly_predicted) / predictions.shape[0]
    return accu


with tf.Session(graph=graph) as session:
    # initialize weights and biases
    tf.global_variables_initializer().run()
    print("Initialized")

    for step in range(num_steps):
        # pick a randomized offset
        offset = np.random.randint(0, train_labels.shape[0] - batch_size - 1)

        # Generate a minibatch.
        batch_data = train_dataset[offset:(offset + batch_size), :]
        batch_labels = train_labels[offset:(offset + batch_size), :]

        # Prepare the feed dict
        feed_dict = {tf_train_dataset : batch_data,
                     tf_train_labels : batch_labels}
        # run one step of computation
        _, l, predictions = session.run([optimizer, loss, train_prediction],
                                        feed_dict=feed_dict)

        if (step % 500 == 0):
            print("Minibatch loss at step {0}: {1}".format(step, l))
            print("Minibatch accuracy: {:.1f}%".format(
                accuracy(predictions, batch_labels)))
            print("Validation accuracy: {:.1f}%".format(
                accuracy(valid_prediction.eval(), valid_labels)))

    print("\nTest accuracy: {:.1f}%".format(
        accuracy(test_prediction.eval(), test_labels)))
```

Output:

```
Initialized
Minibatch loss at step 0: 11.68728256225586
Minibatch accuracy: 10.2%
Validation accuracy: 14.3%
Minibatch loss at step 500: 2.239773750305176
Minibatch accuracy: 46.9%
Validation accuracy: 67.6%
Minibatch loss at step 1000: 1.0917563438415527
Minibatch accuracy: 78.1%
Validation accuracy: 75.0%
Minibatch loss at step 1500: 0.6598564386367798
Minibatch accuracy: 78.9%
Validation accuracy: 78.6%
Minibatch loss at step 2000: 0.24766433238983154
Minibatch accuracy: 91.4%
Validation accuracy: 81.0%
Minibatch loss at step 2500: 0.6181786060333252
Minibatch accuracy: 84.4%
Validation accuracy: 82.5%
Minibatch loss at step 3000: 0.9605385065078735
Minibatch accuracy: 85.2%
Validation accuracy: 83.9%
Minibatch loss at step 3500: 0.6315320730209351
Minibatch accuracy: 85.2%
Validation accuracy: 84.4%
Minibatch loss at step 4000: 0.812285840511322
Minibatch accuracy: 82.8%
Validation accuracy: 85.0%
Minibatch loss at step 4500: 0.5949224233627319
Minibatch accuracy: 80.5%
Validation accuracy: 85.6%
Minibatch loss at step 5000: 0.47554320096969604
Minibatch accuracy: 89.1%
Validation accuracy: 86.2%

Test accuracy: 86.5%
```

Some important points to note:

- In every iteration, a minibatch is selected by choosing a random offset value using **np.random.randint** method.
- To feed the placeholders **tf_train_dataset** and **tf_train_label**, we create a **feed_dict** like this:

```
feed_dict = {tf_train_dataset : batch_data, tf_train_labels : batch_labels}
```

➔ **A shortcut way of performing one step of computation is:**

```
_, l, predictions = session.run([optimizer, loss, train_prediction],
feed_dict=feed_dict)
```

- This node returns the new values of loss and predictions after performing optimization step.

This brings us to the end of the implementation. Complete code can be found here(https://gist.github.com/nikhilkumarsingh/c80a575b81b47739c0543b5fa52b349a).

Finally, here are some points to ponder upon:

- You can try to tweak with the parameters like learning rate, batch size, number of epochs, etc and achieve better results. You can also try a different optimizer like tf.train.AdamOptimizer.
- Accuracy of above model can be improved by using a neural network with one or more hidden layers. We will discuss its implementation using TensorFlow in some upcoming articles.
- **Softmax Regression vs. k Binary Classifiers**
  One should be aware of the scenarios where softmax regression works and where it doesn't. In many cases, you may need to use k different binary logistic classifiers for each of the k possible values of the class label.

  Suppose you are working on a computer vision problem where you're trying to classify images into three different classes:

  Case 1: Suppose that your classes are indoor_scene, outdoor_urban_scene, and outdoor_wilderness_scene.

  Case 2: Suppose your classes are indoor_scene, black_and_white_image, and image_has_people.

  In which case you would use **Softmax Regression** classifier and in which case you would use 3 **Binary Logistic Regression** classifiers?

  This will depend on whether the 3 classes are mutually exclusive.

  In **case 1**, a scene can be either indoor_scene, outdoor_urban_scene or outdoor_wilderness_scene. So, assuming that each training example is labeled with exactly one of the 3 classes, we should build a softmax classifier with k = 3.

  However, in **case 2**, the classes are not mutually exclusive since a scene can be both indoor and have people in it. So, in this case, it would be more appropriate to build 3 binary logistic regression classifiers. This way, for each new scene, your algorithm can separately decide whether it falls into each of the 3 categories.

Linear Regression using PyTorch

Linear Regression is a very commonly used statistical method that allows us to determine and study the relationship between two continuous variables. The various properties of linear regression and its Python implementation has been covered in this article previously. Now, we shall find out how to implement this in PyTorch, a very popular deep learning library that is being developed by Facebook.

Firstly, you will need to install PyTorch into your Python environment. The easiest way to do this is to use the `pip` or `conda` tool. Visit pytorch.org and install the version of your Python interpreter and the package manager that you would like to use.

```
# We can run this Python code on a Jupyter notebook

# to automatically install the correct version of

# PyTorch.


# http://pytorch.org / from os import path

from wheel.pep425tags import get_abbr_impl, get_impl_ver, get_abi_tag

platform = '{}{}-{}'.format(get_abbr_impl(), get_impl_ver(), get_abi_tag())


accelerator = 'cu80' if path.exists('/opt / bin / nvidia-smi') else 'cpu'


! pip install -q http://download.pytorch.org / whl/{accelerator}/torch-0.3.0.post4-
{platform}-linux_x86_64.whl torchvision
```

With PyTorch installed, let us now have a look at the code.
Write the two lines given below to import the necessary library functions and objects.

```
import torch
from torch.autograd import Variable
```

We also define some data and assign them to variables *x_data* and *y_data* as given below:

```
    x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0]]))
    y_data = Variable(torch.Tensor([[2.0], [4.0], [6.0]]))
```

Here, *x_data* is our independent variable and *y_data* is our dependent variable. This will be our dataset for now. Next, we need to define our model. There are two main steps associated with defining our model. They are:

1. Initialising our model.
2. Declaring the forward pass

We use the class given below:

```
class LinearRegressionModel(torch.nn.Module):

    def __init__(self):
        super(LinearRegressionModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred
```

As you can see, our *Model* class is a subclass of *torch.nn.module*. Also, since here we have only one input and one output, we use a Linear model with both the input and output dimension as 1.

Next, we create an object of this model.

```
# our model
our_model = LinearRegressionModel()
```

After this, we select the optimiser and the loss criteria. Here, we will use the mean squared error (MSE) as our loss function and stochastic gradient descent (SGD) as our optimiser. Also, we arbitrarily fix a learning rate of 0.01.

```
criterion = torch.nn.MSELoss(size_average = False)
optimizer = torch.optim.SGD(our_model.parameters(), lr = 0.01)
```

We now arrive at our training step. We perform the following tasks for 500 times during training:

1. Perform a forward pass by passing our data and finding out the predicted value of y.
2. Compute the loss using MSE.
3. Reset all the gradients to 0, peform a backpropagation and then, update the weights.

```
for epoch in range(500):

    # Forward pass: Compute predicted y by passing
    # x to the model
    pred_y = our_model(x_data)

    # Compute and print loss
    loss = criterion(pred_y, y_data)

    # Zero gradients, perform a backward pass,
    # and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    print('epoch {}, loss {}'.format(epoch, loss.data[0]))
```

Once the training is completed, we test if we are getting correct results using the model that we defined. So, we test it for an unknown value of *x_data*, in this case, 4.0.

```
new_var = Variable(torch.Tensor([[4.0]]))
pred_y = our_model(new_var)
print("predict (after training)", 4, our_model(new_var).data[0][0])
```

If you performed all steps correctly, you will see that for the input 4.0, you are getting a value that is very close to 8.0 as below. So, our model inherently learns the relationship between the input data and the output data without being programmed explicitly.

```
predict (after training) 4 7.966438293457031
```

For your reference, you can find the entire code of this article given below:

```
import torch
from torch.autograd import Variable

x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0]]))
y_data = Variable(torch.Tensor([[2.0], [4.0], [6.0]]))


class LinearRegressionModel(torch.nn.Module):

    def __init__(self):
        super(LinearRegressionModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred

# our model
our_model = LinearRegressionModel()

criterion = torch.nn.MSELoss(size_average = False)
optimizer = torch.optim.SGD(our_model.parameters(), lr = 0.01)

for epoch in range(500):

    # Forward pass: Compute predicted y by passing
    # x to the model
    pred_y = our_model(x_data)

    # Compute and print loss
    loss = criterion(pred_y, y_data)

    # Zero gradients, perform a backward pass,
    # and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    print('epoch {}, loss {}'.format(epoch, loss.data[0]))

new_var = Variable(torch.Tensor([[4.0]]))
pred_y = our_model(new_var)
print("predict (after training)", 4, our_model(new_var).data[0][0])
```

Identifying handwritten digits using Logistic Regression in PyTorch

Logistic Regression is a very commonly used statistical method that allows us to predict a binary output from a set of independent variables. The various properties of logistic regression and its Python implementation has been covered in this article previously. Now, we shall find out how to implement this in PyTorch, a very popular deep learning library that is being developed by Facebook.

Now, we shall see how to classify handwritten digits from the MNIST dataset using Logistic Regression in PyTorch. Firstly, you will need to install PyTorch into your Python environment. The easiest way to do this is to use the `pip` or `conda` tool. Visit pytorch.org and install the version of your Python interpreter and the package manager that you would like to use.

With PyTorch installed, let us now have a look at the code. Write the three lines given below to import the reqiored library functions and objects.

```
 import torch
import torch.nn as nn
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.autograd import Variable
```

Here, the *torch.nn* module contains the code required for the model, *torchvision.datasets* contains the MNIST dataset. It contains the dataset of handwritten digits that we shall be using here. The *torchvision.transforms* module contains various methods to transform objects into others. Here, we shall be using it to transform from images to PyTorch tensors. Also, the *torch.autograd* module contains the Variable class amongst others, which will be used by us while defining our tensors.

Next, we shall download and load the dataset to memory.

```
# MNIST Dataset (Images and Labels)
train_dataset = dsets.MNIST(root ='./data',
                            train = True,
                            transform = transforms.ToTensor(),
                            download = True)


test_dataset = dsets.MNIST(root ='./data',
                           train = False,
                           transform = transforms.ToTensor())


# Dataset Loader (Input Pipline)
train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
                                           batch_size = batch_size,
                                           shuffle = True)


test_loader = torch.utils.data.DataLoader(dataset = test_dataset,
                                          batch_size = batch_size,
                                          shuffle = False)
```

Now, we shall define our hyper parameters.

```
# Hyper Parameters
input_size = 784
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
```

In our dataset, the image size is 28*28. Thus, our input size is 784. Also, 10 digits are present in this and hence, we can have 10 different outputs. Thus, we set num_classes as 10. Also, we shall train for five times on the entire dataset. Finally, we will train in small batches of 100 images each so as to prevent the crashing of the program due to memory overflow.

After this, we shall be defining our model as below. Here, we shall initialise our model as a subclass of *torch.nn.Module* and then define the forward pass. In the code that we are writing, the softmax is internally calculated during each forward pass and hence we do not need to specify it inside the forward() function.

```
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.linear = nn.Linear(input_size, num_classes)

    def forward(self, x):
        out = self.linear(x)
        return out
```

Having defined our class, now we instantiate an object for the same.

```
   model = LogisticRegression(input_size, num_classes)
```

Next, we set our loss function and the optimiser. Here, we shall be using the cross entropy loss and for the optimiser, we shall be using the stochastic gradient descent algorithm with a learning rate of 0.001 as defined in the hyper parameter above.

```
   criterion = nn.CrossEntropyLoss()
   optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)
```

Now, we shall start the training. Here, we shall be performing the following tasks:

1. Reset all gradients to 0.
2. Make a forward pass.
3. Calculate the loss.
4. Perform backpropagation.
5. Update all weights.

```
# Training the Model
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = Variable(images.view(-1, 28 * 28))
        labels = Variable(labels)

        # Forward + Backward + Optimize
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i + 1) % 100 == 0:
            print('Epoch: [% d/% d], Step: [% d/% d], Loss: %.4f'
                  % (epoch + 1, num_epochs, i + 1,
                     len(train_dataset) // batch_size, loss.data[0]))
```

Finally, we shall be testing out model by using the following code.

```
# Test the Model
correct = 0
total = 0
for images, labels in test_loader:
    images = Variable(images.view(-1, 28 * 28))
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()

print('Accuracy of the model on the 10000 test images: % d %%' % (
            100 * correct / total))
```

Assuming that you performed all steps correctly, you will get an accuracy of 82%, which is far off from today's state of the art model, which uses a special type of neural network architecture. For your reference, you can find the entire code for this article below:

```
import torch
import torch.nn as nn
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.autograd import Variable


# MNIST Dataset (Images and Labels)
train_dataset = dsets.MNIST(root ='./data',
                            train = True,
                            transform = transforms.ToTensor(),
                            download = True)

test_dataset = dsets.MNIST(root ='./data',
                           train = False,
                           transform = transforms.ToTensor())
```

```
# Dataset Loader (Input Pipline)
train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
                                           batch_size = batch_size,
                                           shuffle = True)

test_loader = torch.utils.data.DataLoader(dataset = test_dataset,
                                          batch_size = batch_size,
                                          shuffle = False)

# Hyper Parameters
input_size = 784
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001

# Model
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.linear = nn.Linear(input_size, num_classes)

    def forward(self, x):
        out = self.linear(x)
        return out


model = LogisticRegression(input_size, num_classes)

# Loss and Optimizer
# Softmax is internally computed.
# Set parameters to be updated.
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)

# Training the Model
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = Variable(images.view(-1, 28 * 28))
        labels = Variable(labels)

        # Forward + Backward + Optimize
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i + 1) % 100 == 0:
            print('Epoch: [% d/% d], Step: [% d/% d], Loss: %.4f'
                  % (epoch + 1, num_epochs, i + 1,
                     len(train_dataset) // batch_size, loss.data[0]))
```

```
# Test the Model
correct = 0
total = 0
for images, labels in test_loader:
    images = Variable(images.view(-1, 28 * 28))
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()

print('Accuracy of the model on the 10000 test images: % d %%' % (
            100 * correct / total))
```