

## 1. Introduction

### 1.1 What Is Machine Learning?

Machine learning is programming computers to optimize a performance criterion using example data or past experience. In this process, a model is defined by a set of parameters, and learning involves executing a computer program to adjust these parameters based on training data or prior experience. The main objective is to enable the model to improve its performance over time as it encounters more data. Depending on the problem, the model may be predictive, where it is used to forecast future outcomes or make predictions, or descriptive, where it is used to uncover patterns and insights from existing data. In some cases, a model may serve both purposes.

Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term “Machine Learning” in 1959 while at IBM. He defined machine learning as “the field of study that gives computers the ability to learn without being explicitly programmed.” Samuel’s work involved creating programs that could improve through experience, notably in games like checkers, where the program would learn strategies by playing against itself. Although Samuel’s definition has become foundational, there is no single universally accepted definition for machine learning. Different authors and researchers may define the term in various ways based on their areas of focus and application. Some definitions may emphasize the statistical foundations of machine learning, while others highlight the algorithmic or computational aspects, making the field diverse and continuously evolving. As machine learning techniques and applications continue to advance, new definitions and frameworks are likely to emerge.

#### Definition of learning

##### Definition

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  $T$ , as measured by  $P$ , improves with experience  $E$ .

#### Examples

##### i) Handwriting recognition learning problem

- Task  $T$ : The task is to recognize and classify handwritten words within images. This can be a challenging problem due to the variability in human handwriting and the need to identify words correctly regardless of how they are written.
- Performance  $P$ : The performance measure could be the percentage of words correctly classified, or more formally, the accuracy rate. The goal is to minimize errors and improve the classification accuracy of handwritten words.
- Training Experience  $E$ : The training experience consists of a dataset containing labeled examples of handwritten words, along with their corresponding classifications (e.g., the word "apple" written in various styles). The more diverse and representative the dataset, the better the model can learn to generalize to unseen data.

ii) A robot driving learning problem

- Task T: Driving on highways using vision sensors
- Performance measure P: Average distance traveled before an error
- training experience: A sequence of images and steering commands recorded while observing a human driver

iii) A chess learning problem

- Task T: Playing chess
- Performance measure P: Percent of games won against opponents
- Training experience E: Playing practice games against itself

A computer program which learns from experience is called a machine learning program or simply a learning program. Such a program is sometimes also referred to as a learner.

### **Components of Learning**

Basic components of learning process

The learning process, whether by a human or a machine, can be divided into four components, namely, data storage, abstraction, generalization and evaluation. Figure 1.1 illustrates the various components and the steps involved in the learning process.

#### **1. Data storage**

Facilities for storing and retrieving huge amounts of data are an important component of the learning process. Humans and computers alike utilize data storage as a foundation for advanced reasoning.

- In a human being, the data is stored in the brain and data is retrieved using electrochemical signals.
- Computers use hard disk drives, flash memory, random access memory and similar devices to store data and use cables and other technology to retrieve data.

#### **2. Abstraction**

The second component of the learning process is known as abstraction. Abstraction is the process of extracting knowledge about stored data. This involves creating general concepts about the data as a whole. The creation of knowledge involves application of known models and creation of new models. The process of fitting a model to a dataset is known as training. When the model has been trained, the data is transformed into an abstract form that summarizes the original information.

#### **3. Generalization**

The third component of the learning process is known as generalisation. The term generalization describes the process of turning the knowledge about stored data into a form that can be utilized for

future action. These actions are to be carried out on tasks that are similar, but not identical, to those what have been seen before. In generalization, the goal is to discover those properties of the data that will be most relevant to future tasks.

#### 4.Evaluation

Evaluation is the last component of the learning process. It is the process of giving feedback to the user to measure the utility of the learned knowledge. This feedback is then utilised to effect improvements in the whole learning process.

#### Why Use Machine Learning?

Consider how you would write a spam filter using traditional programming techniques (Figure 1-1):

1. First you would look at what spam typically looks like. You might notice that some words or phrases (such as “4U,” “credit card,” “free,” and “amazing”) tend to come up a lot in the subject. Perhaps you would also notice a few other patterns in the sender’s name, the email’s body, and so on.
2. You would write a detection algorithm for each of the patterns that you noticed, and your program would flag emails as spam if a number of these patterns are detected.
3. You would test your program, and repeat steps 1 and 2 until it is good enough.

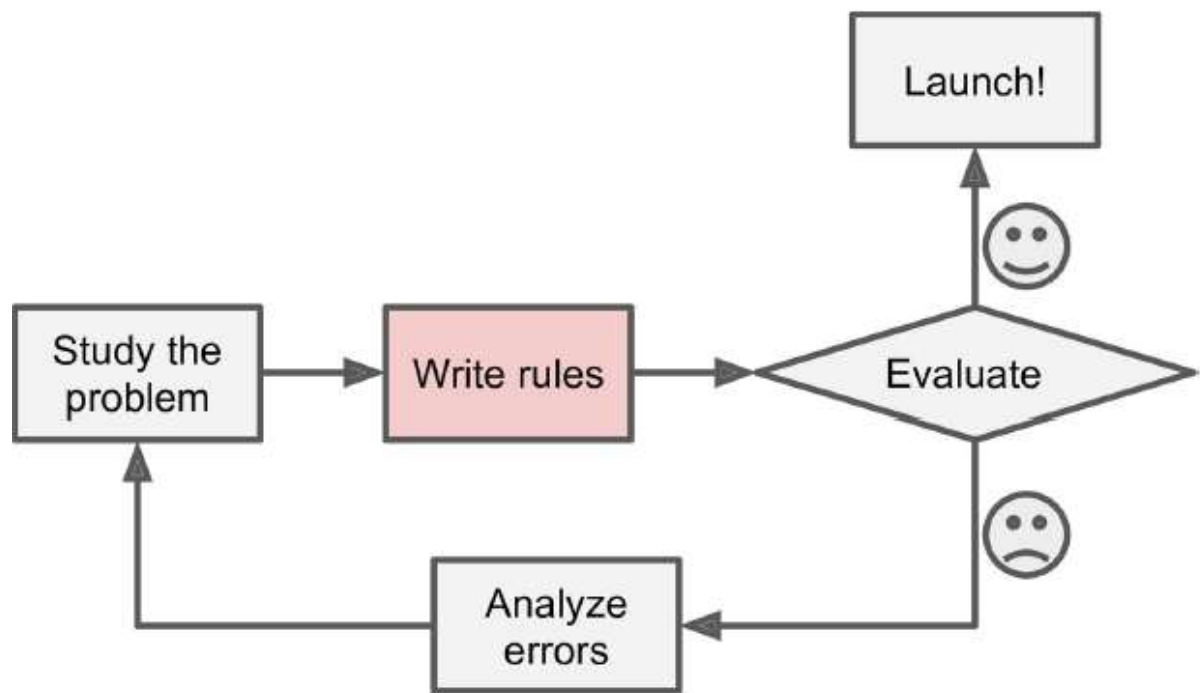


Figure 1-1. The traditional approach

Since the problem is not trivial, your program will likely become a long list of complex rules — pretty hard to maintain. In contrast, a spam filter based on Machine Learning techniques automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples (Figure 1-2). The program is much shorter, easier to maintain, and most likely more accurate.

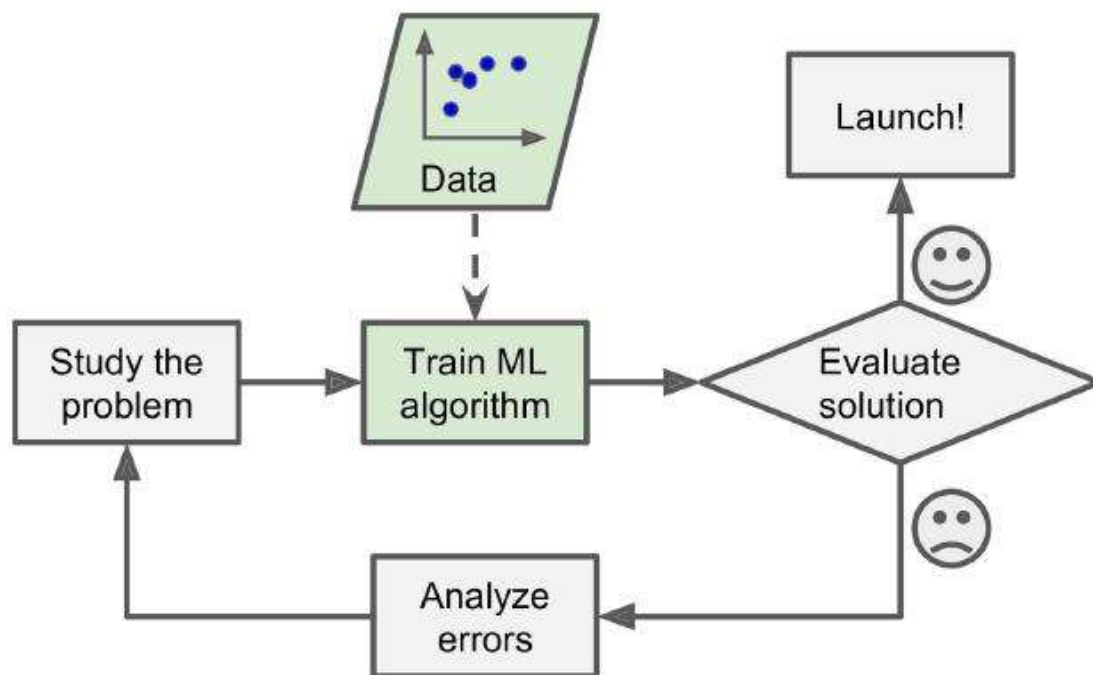


Figure 1-2. Machine Learning approach

Moreover, if spammers notice that all their emails containing “4U” are blocked, they might start writing “For U” instead. A spam filter using traditional programming techniques would need to be updated to flag “For U” emails. If spammers keep working around your spam filter, you will need to keep writing new rules forever.

In contrast, a spam filter based on Machine Learning techniques automatically notices that “For U” has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention (Figure 1-3).



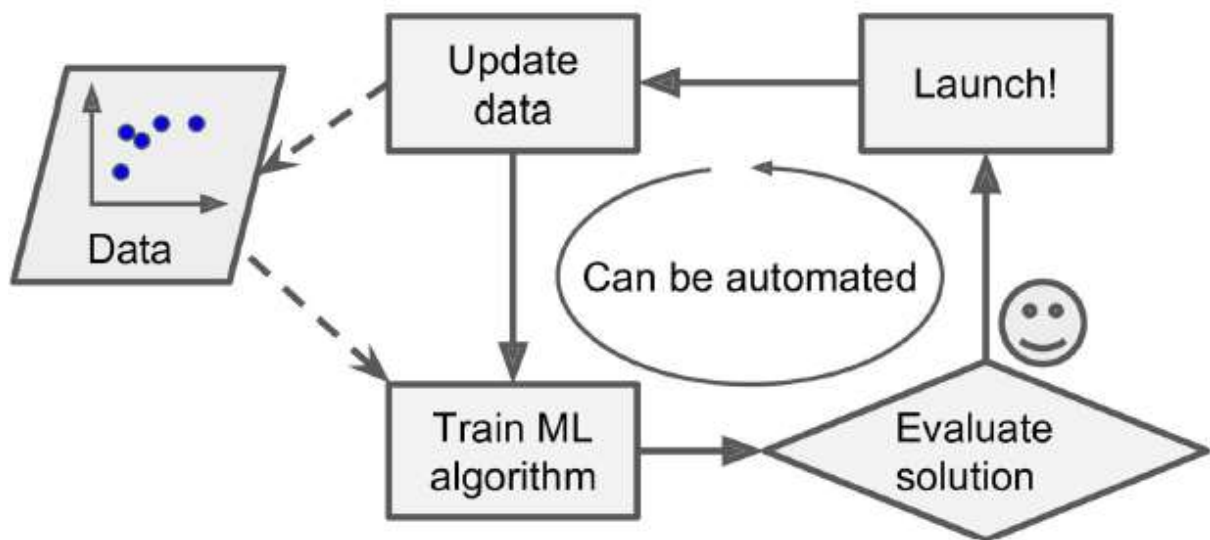


Figure 1-3. Automatically adapting to change

Another area where Machine Learning shines is for problems that either are too complex for traditional approaches or have no known algorithm. For example, consider speech recognition: say you want to start simple and write a program capable of distinguishing the words “one” and “two.” You might notice that the word “two” starts with a high-pitch sound (“T”), so you could hardcode an algorithm that measures high-pitch sound intensity and use that to distinguish ones and twos. Obviously this technique will not scale to thousands of words spoken by millions of very different people in noisy environments and in dozens of languages. The best solution (at least today) is to write an algorithm that learns by itself, given many example recordings for each word.

Finally, Machine Learning can help humans learn (Figure 1-4): ML algorithms can be inspected to see what they have learned (although for some algorithms this can be tricky). For instance, once the spam filter has been trained on enough spam, it can easily be inspected to reveal the list of words and combinations of words that it believes are the best predictors of spam. Sometimes this will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem. Applying ML techniques to dig into large amounts of data can help discover patterns that were not immediately apparent. This is called data mining.

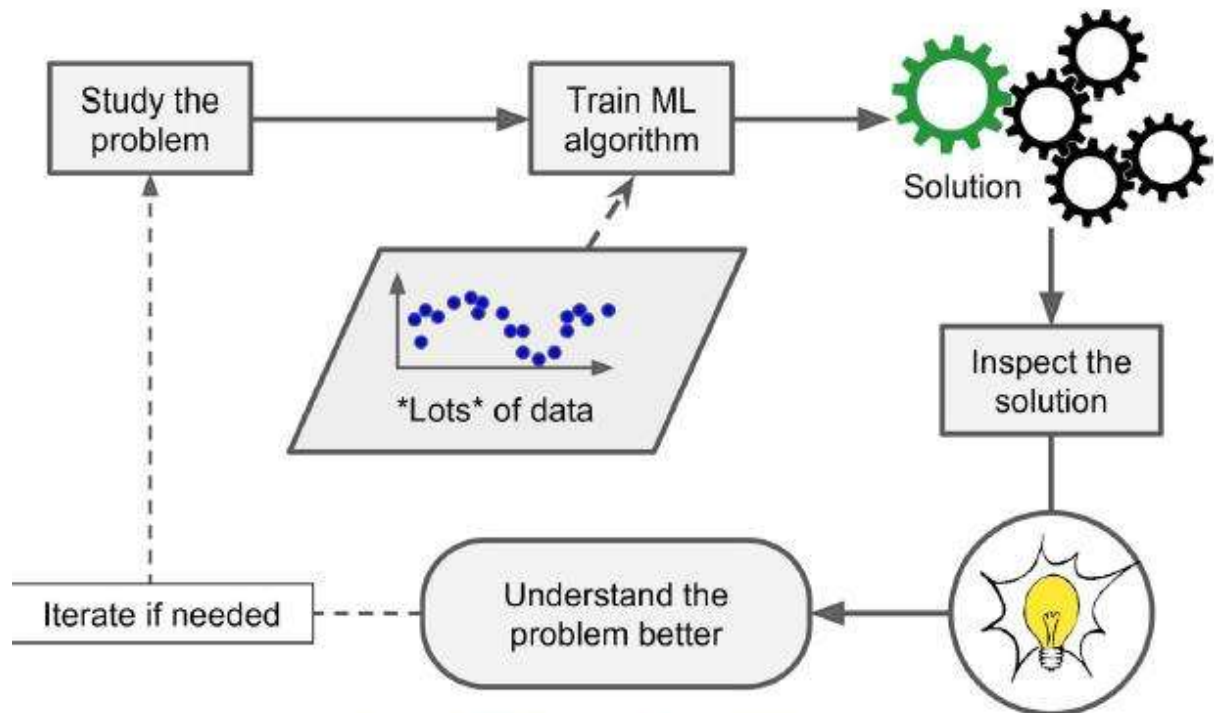


Figure 1-4. Machine Learning can help humans learn

To summarize, Machine Learning is great for:

1. Problems for which existing solutions require a lot of hand-tuning or long lists of rules: one Machine

Learning algorithm can often simplify code and perform better.

2. Complex problems for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.

3. Fluctuating environments: a Machine Learning system can adapt to new data.

4. Getting insights about complex problems and large amounts of data.

### Types of Machine Learning Systems

There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:

Whether or not they are trained with human supervision (supervised, unsupervised, semisupervised, and Reinforcement Learning)

Whether or not they can learn incrementally on the fly (online versus batch learning) Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do (instance-based versus model-based learning)

These criteria are not exclusive; you can combine them in any way you like. For example, a state-of-the-art spam filter may learn on the fly using a deep neural network model trained using examples of spam and ham; this makes it an online, model-based, supervised learning system.

Let's look at each of these criteria a bit more closely.

## Supervised/Unsupervised Learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning.

### Supervised learning

In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels (Figure 1-5).

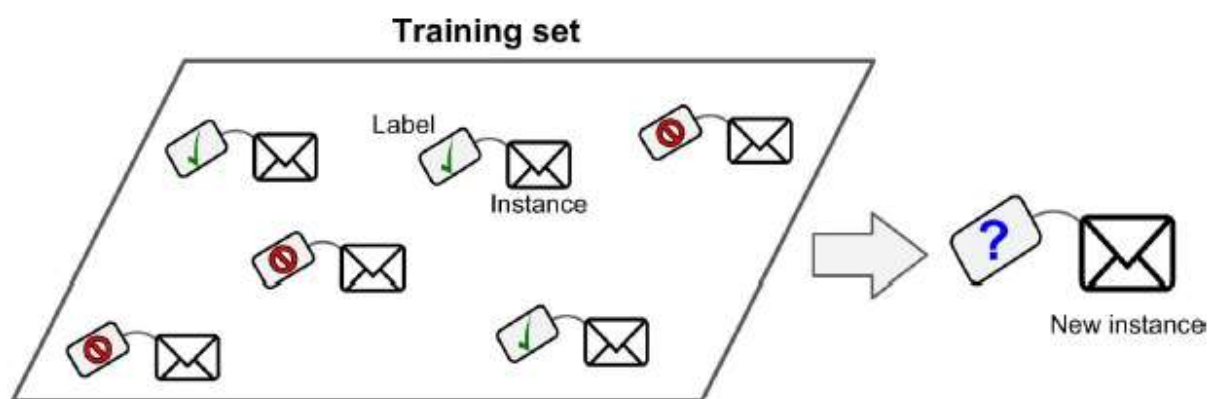
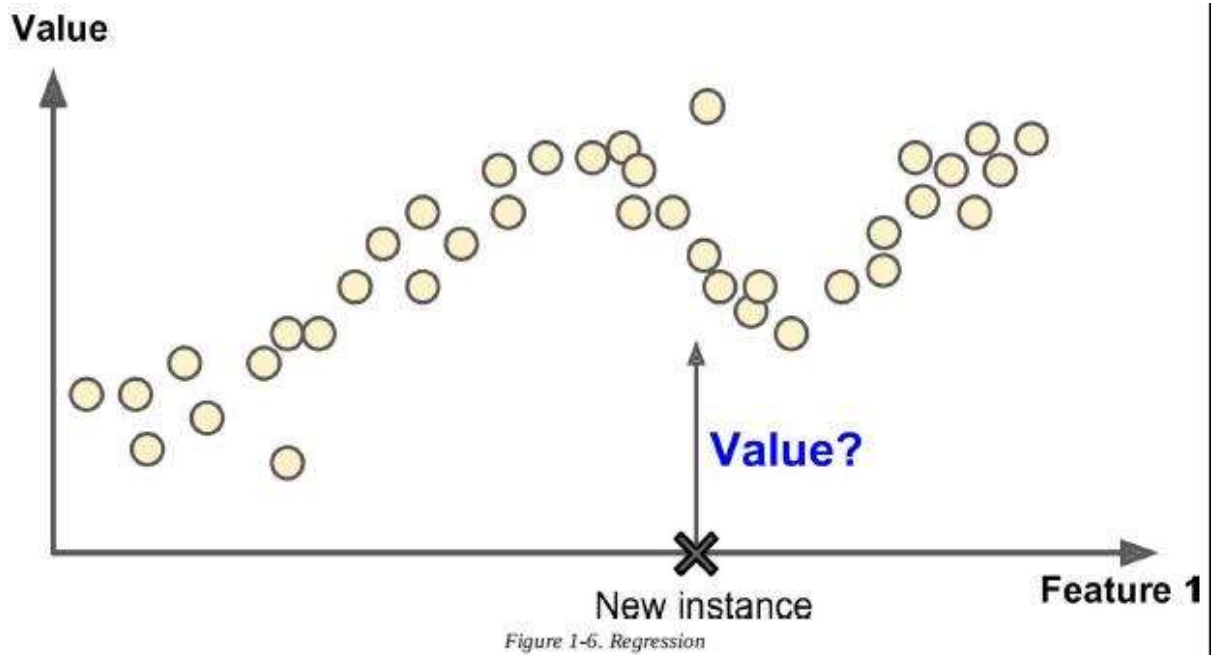


Figure 1-5. A labeled training set for supervised learning (e.g., spam classification)

A typical supervised learning task is classification. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails.

Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.) called predictors. This sort of task is called regression (Figure 1-6). To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).



Note that some regression algorithms can be used for classification as well, and vice versa. For example, Logistic Regression is commonly used for classification, as it can output a value that corresponds to the probability of belonging to a given class (e.g., 20% chance of being spam).

Here are some of the most important supervised learning algorithms

k-Nearest Neighbors

Linear Regression

Logistic Regression

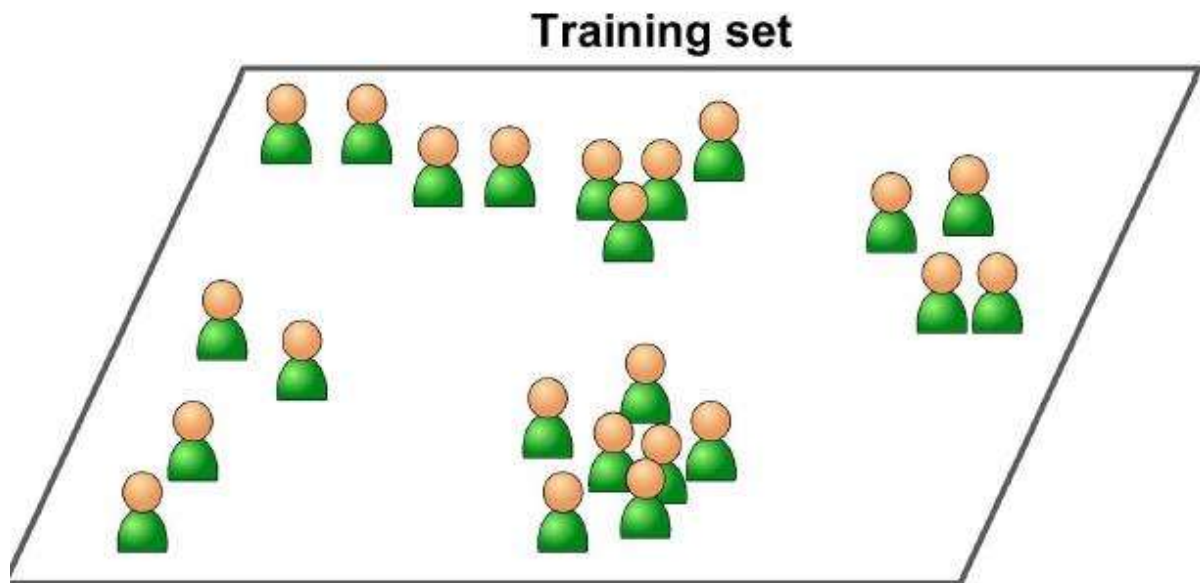
Support Vector Machines (SVMs)

Decision Trees and Random Forests

Neural networks

### **Unsupervised learning**

In unsupervised learning, as you might guess, the training data is unlabeled (Figure 1-7). The system tries to learn without a teacher.



*Figure 1-7. An unlabeled training set for unsupervised learning*

Here are some of the most important unsupervised learning algorithms

Clustering

k-Means

Hierarchical Cluster Analysis (HCA)

Expectation Maximization

Visualization and dimensionality reduction

Principal Component Analysis (PCA)

Kernel PCA

Locally-Linear Embedding (LLE)

t-distributed Stochastic Neighbor Embedding (t-SNE)

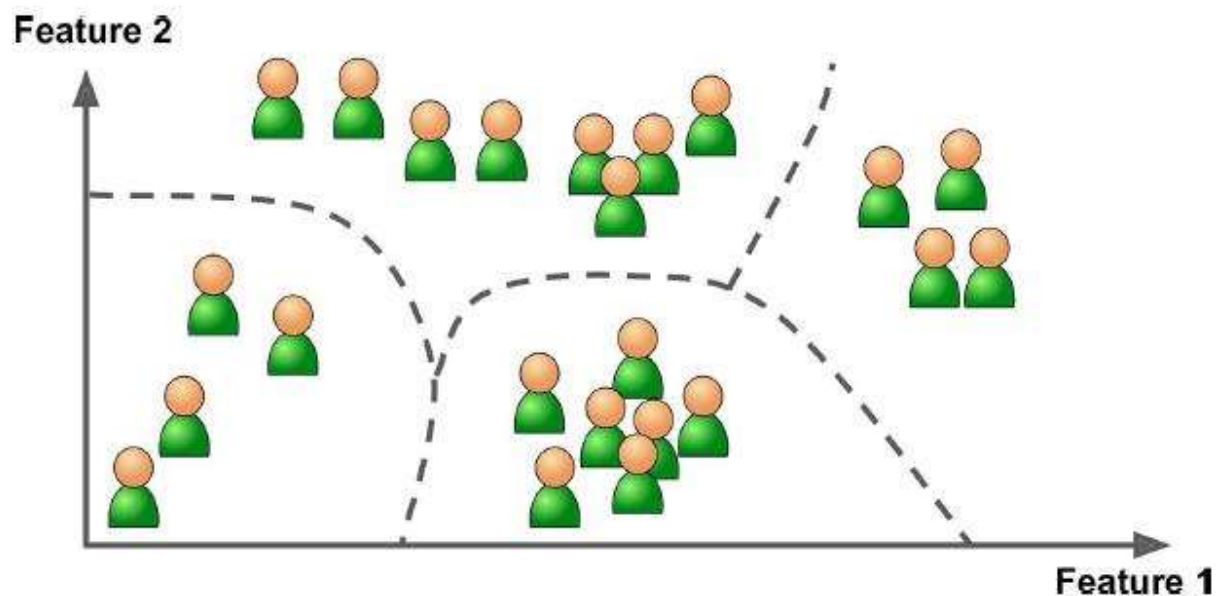
Association rule learning

Apriori

Eclat

For example, say you have a lot of data about your blog's visitors. You may want to run a clustering algorithm to try to detect groups of similar visitors (Figure 1-8). At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help. For example, it might notice that 40% of your visitors are males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends, and so on. If you use a

hierarchical clustering algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.



*Figure 1-8. Clustering*

Visualization algorithms are also good examples of unsupervised learning algorithms: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted (Figure 1-9). These algorithms try to preserve as much structure as they can (e.g., trying to keep separate clusters in the input space from overlapping in the visualization), so you can understand how the data is organized and perhaps identify unsuspected patterns.



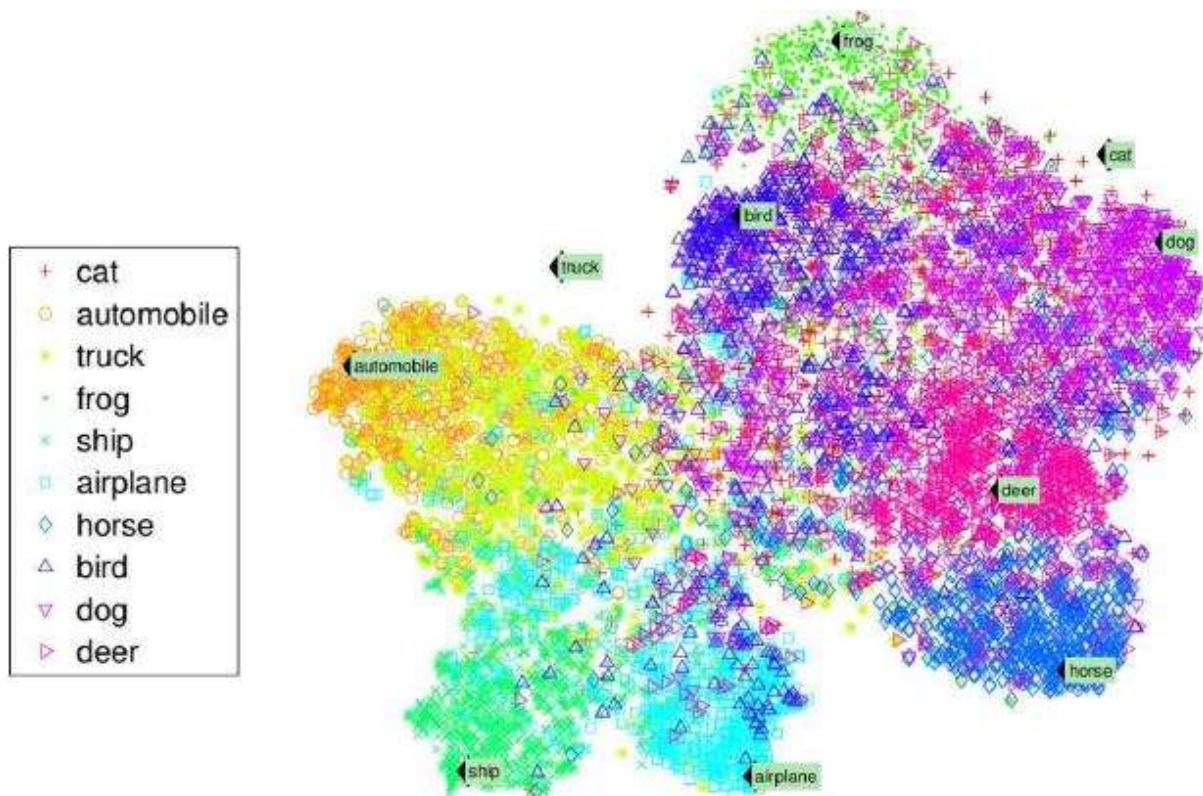


Figure 1-9. Example of a t-SNE visualization highlighting semantic clusters<sup>3</sup>

A related task is dimensionality reduction, in which the goal is to simplify the data without losing too much information. One way to do this is to merge several correlated features into one. For example, a car's mileage may be very correlated with its age, so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear. This is called feature extraction.

Yet another important unsupervised task is anomaly detection — for example, detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. The system is trained with normal instances, and when it sees a new instance it can tell whether it looks like a normal one or whether it is likely an anomaly (see Figure 1-10).

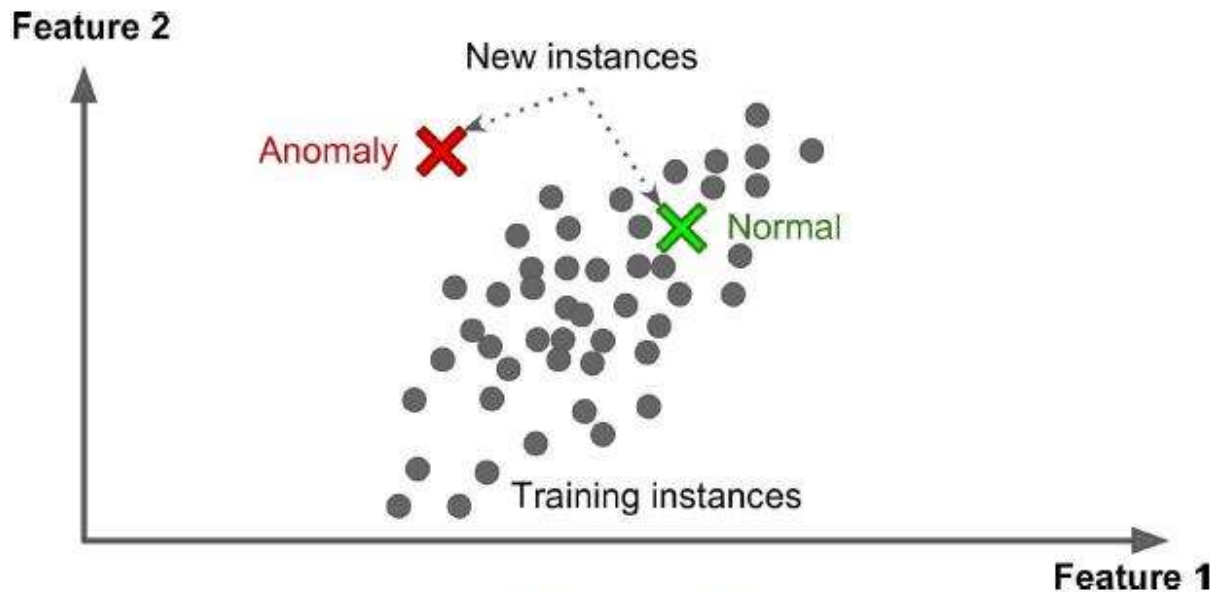


Figure 1-10. Anomaly detection

Finally, another common unsupervised task is association rule learning, in which the goal is to dig into large amounts of data and discover interesting relations between attributes. For example, suppose you own a supermarket. Running an association rule on your sales logs may reveal that people who purchase barbecue sauce and potato chips also tend to buy steak. Thus, you may want to place these items close to each other.

#### Semisupervised learning

Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called semisupervised learning (Figure 1-11). Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7. This is the unsupervised part of the algorithm (clustering). Now all the system needs is for you to tell it who these people are. Just one label per person,<sup>4</sup> and it is able to name everyone in every photo, which is useful for searching photos.



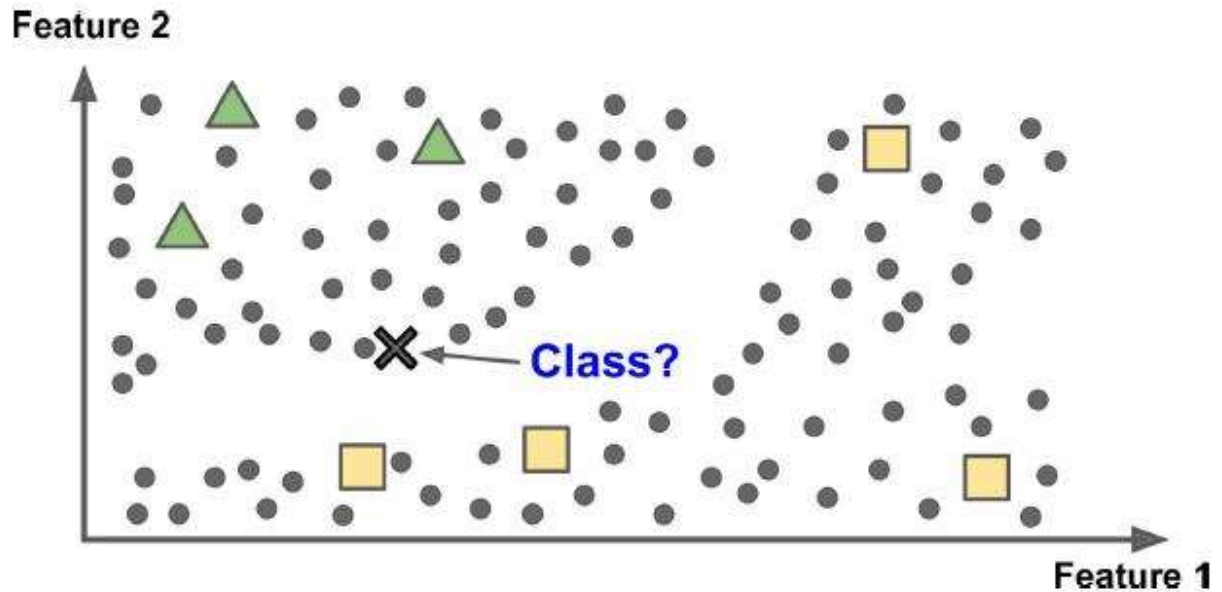


Figure 1-11. Semisupervised learning

Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms. For example, deep belief networks (DBNs) are based on unsupervised components called restricted Boltzmann machines (RBMs) stacked on top of one another. RBMs are trained sequentially in an unsupervised manner, and then the whole system is fine-tuned using supervised learning techniques.

### Reinforcement Learning

Reinforcement Learning is a very different beast. The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards, as in Figure 1-12). It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

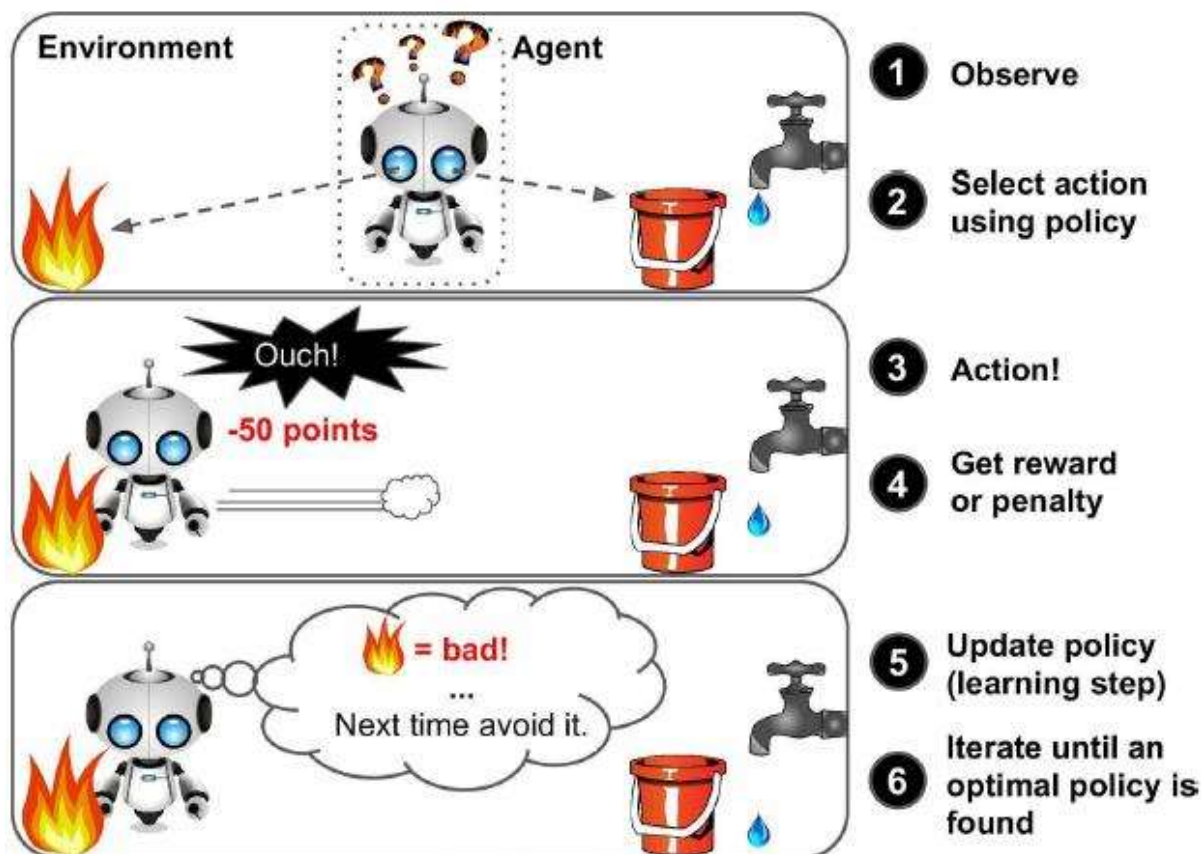


Figure 1-12. Reinforcement Learning

For example, many robots implement Reinforcement Learning algorithms to learn how to walk. DeepMind's AlphaGo program is also a good example of Reinforcement Learning: it made the headlines in March 2016 when it beat the world champion Lee Sedol at the game of Go. It learned its winning policy by analyzing millions of games, and then playing many games against itself. Note that learning was turned off during the games against the champion; AlphaGo was just applying the policy it had learned.

### Batch and Online Learning

Another criterion used to classify Machine Learning systems is whether or not the system can learn incrementally from a stream of incoming data.

#### Batch learning

In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning.

If you want a batch learning system to know about new data (such as a new type of spam), you need to train a new version of the system from scratch on the full dataset (not just the new data, but also the old data), then stop the old system and replace it with the new one.

Fortunately, the whole process of training, evaluating, and launching a Machine Learning system can be automated fairly easily (as shown in Figure 1-3), so even a batch learning system can adapt to change. Simply update the data and train a new version of the system from scratch as often as needed. This solution is simple and often works fine, but training using the full set of data can take many hours, so you would typically train a new system only every 24 hours or even just weekly. If your system needs to adapt to rapidly changing data (e.g., to predict stock prices), then you need a more reactive solution.

Also, training on the full set of data requires a lot of computing resources (CPU, memory space, disk space, disk I/O, network I/O, etc.). If you have a lot of data and you automate your system to train from scratch every day, it will end up costing you a lot of money. If the amount of data is huge, it may even be impossible to use a batch learning algorithm.

Finally, if your system needs to be able to learn autonomously and it has limited resources (e.g., a smartphone application or a rover on Mars), then carrying around large amounts of training data and taking up a lot of resources to train for hours every day is a showstopper.

Fortunately, a better option in all these cases is to use algorithms that are capable of learning incrementally.

### Online learning

In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini-batches. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives (see Figure 1-13).

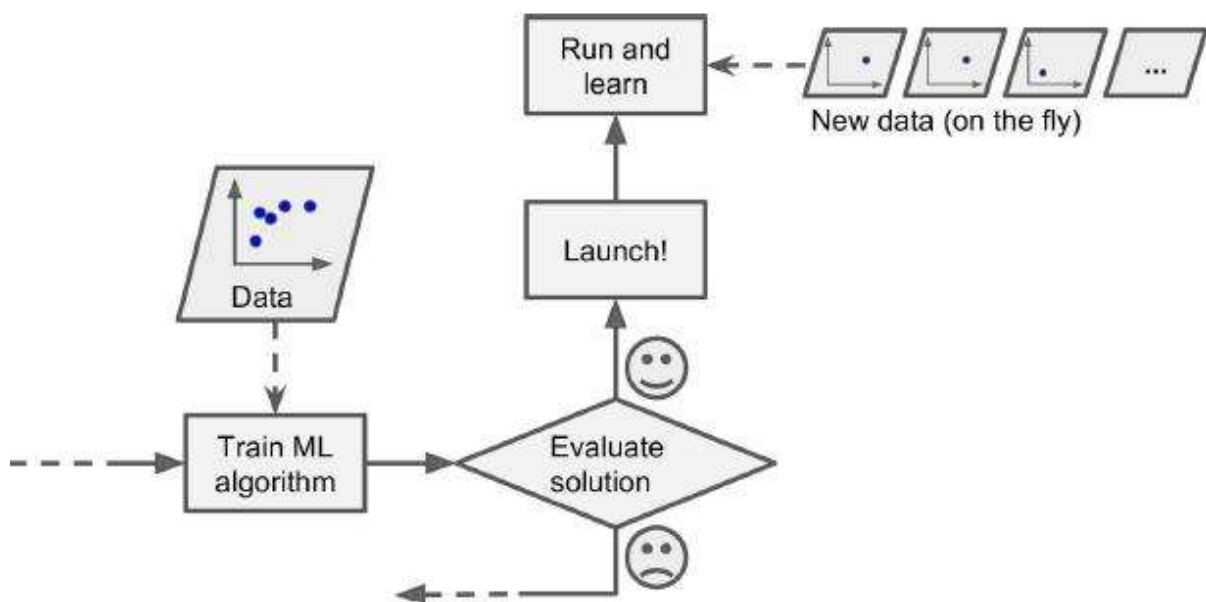


Figure 1-13. Online learning

Online learning is great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously. It is also a good option if you have limited computing resources:

once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them (unless you want to be able to roll back to a previous state and “replay” the data). This can save a huge amount of space.

Online learning algorithms can also be used to train systems on huge datasets that cannot fit in one machine’s main memory (this is called out-of-core learning). The algorithm loads part of the data, runs a training step on that data, and repeats the process until it has run on all of the data (see Figure 1-14).

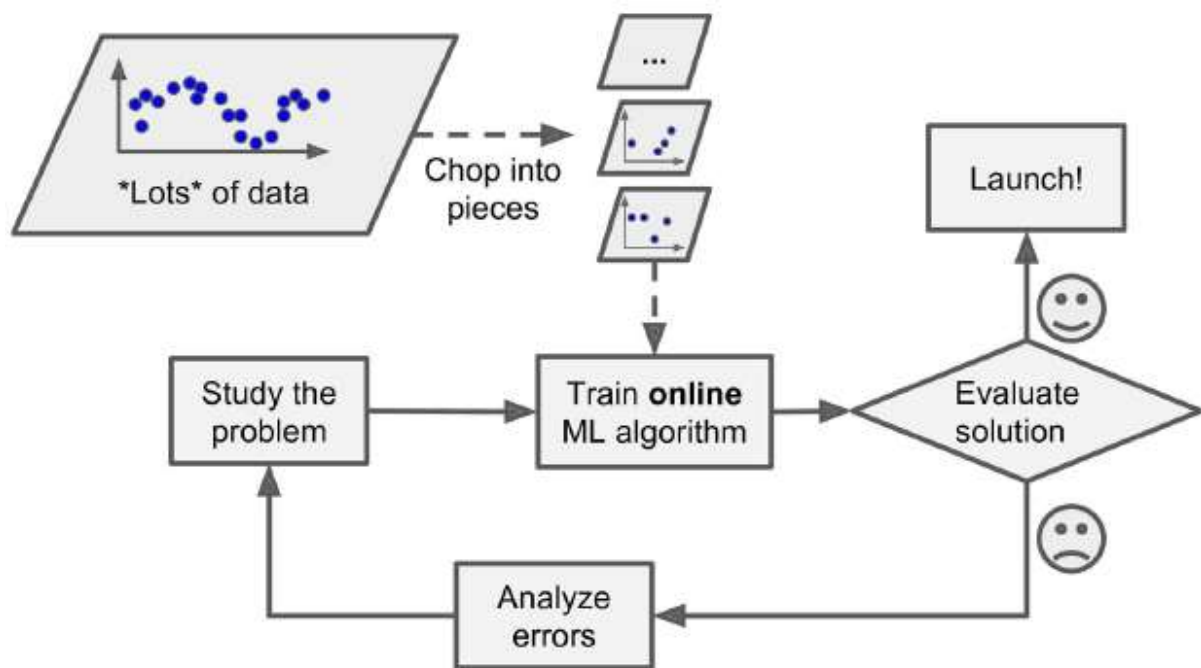


Figure 1-14. Using online learning to handle huge datasets

One important parameter of online learning systems is how fast they should adapt to changing data: this is called the learning rate. If you set a high learning rate, then your system will rapidly adapt to new data, but it will also tend to quickly forget the old data (you don’t want a spam filter to flag only the latest kinds of spam it was shown). Conversely, if you set a low learning rate, the system will have more inertia; that is, it will learn more slowly, but it will also be less sensitive to noise in the new data or to sequences of nonrepresentative data points.

A big challenge with online learning is that if bad data is fed to the system, the system’s performance will gradually decline. If we are talking about a live system, your clients will notice. For example, bad data could come from a malfunctioning sensor on a robot, or from someone spamming a search engine to try to rank high in search results. To reduce this risk, you need to monitor your system closely and promptly switch learning off (and possibly revert to a previously working state) if you detect a drop in performance. You may also want to monitor the input data and react to abnormal data (e.g., using an anomaly detection algorithm).

#### Instance-Based Versus Model-Based Learning

One more way to categorize Machine Learning systems is by how they generalize. Most Machine

Learning tasks are about making predictions. This means that given a number of training examples, the system needs to be able to generalize to examples it has never seen before. Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances. There are two main approaches to generalization: instance-based learning and model-based learning.

### Instance-based learning

Possibly the most trivial form of learning is simply to learn by heart. If you were to create a spam filter this way, it would just flag all emails that are identical to emails that have already been flagged by users — not the worst solution, but certainly not the best.

Instead of just flagging emails that are identical to known spam emails, your spam filter could be programmed to also flag emails that are very similar to known spam emails. This requires a measure of similarity between two emails. A (very basic) similarity measure between two emails could be to count the number of words they have in common. The system would flag an email as spam if it has many words in common with a known spam email.

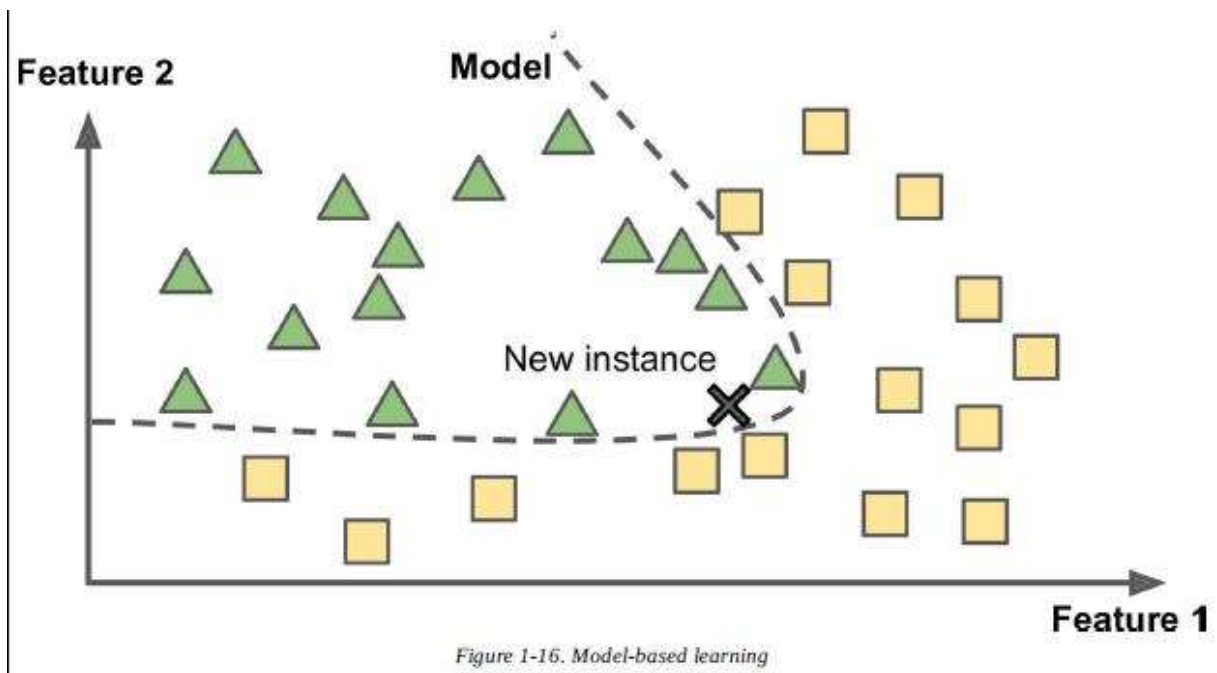
This is called instance-based learning: the system learns the examples by heart, then generalizes to new cases using a similarity measure (Figure 1-15).



Figure 1-15. Instance-based learning

### Model-based learning

Another way to generalize from a set of examples is to build a model of these examples, then use that model to make predictions. This is called model-based learning (Figure 1-16).



For example, suppose you want to know if money makes people happy, so you download the Better Life Index data from the OECD’s website as well as stats about GDP per capita from the IMF’s website. Then you join the tables and sort by GDP per capita. Table 1-1 shows an excerpt of what you get.

Table 1-1. Does money make people happier?

Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2

Let’s plot the data for a few random countries (Figure 1-17).

Equation 1-1. A simple linear model

$$life\_satisfaction = \theta_0 + \theta_1 \times GDP\_per\_capita$$

This model has two *model parameters*,  $\theta_0$  and  $\theta_1$ .<sup>5</sup> By tweaking these parameters, you can make your model represent any linear function, as shown in Figure 1-18.



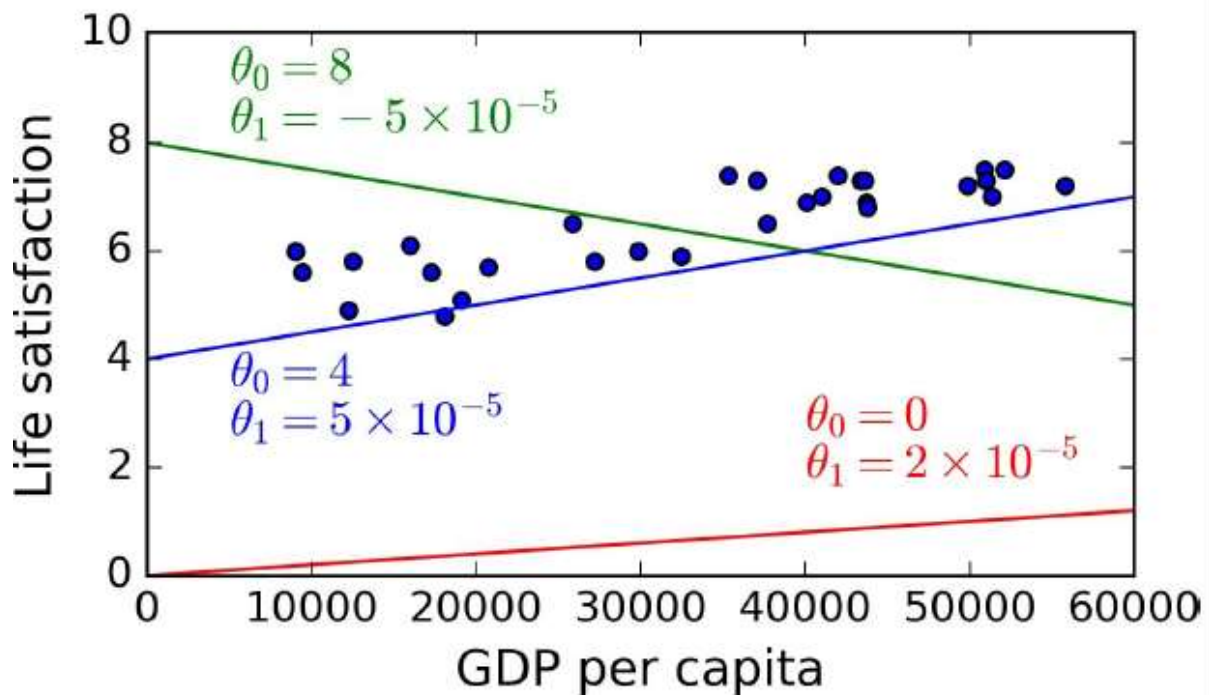


Figure 1-18. A few possible linear models

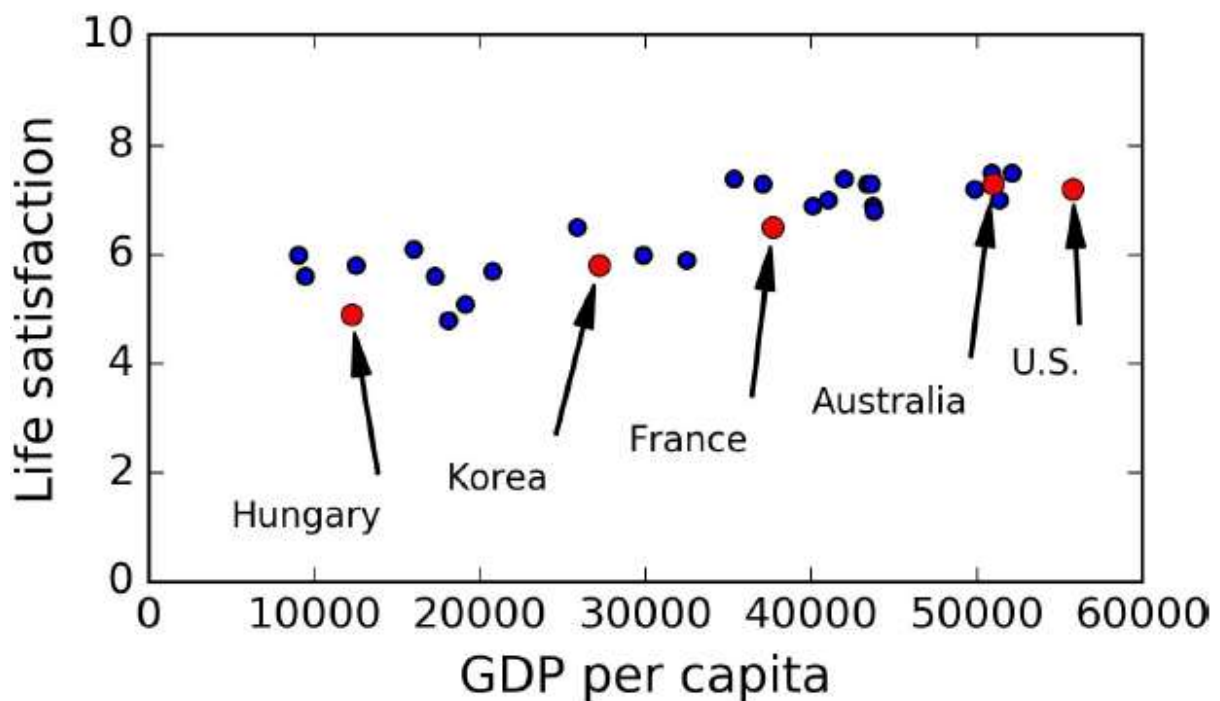


Figure 1-17. Do you see a trend here?

There does seem to be a trend here! Although the data is noisy (i.e., partly random), it looks like life satisfaction goes up more or less linearly as the country's GDP per capita increases. So you decide to model life satisfaction as a linear function of GDP per capita. This step is called model selection: you selected a linear model of life satisfaction with just one attribute, GDP per capita (Equation 1-1).

### Applications of machine learning

Application of machine learning methods to large databases is called data mining. In data mining, a large volume of data is processed to construct a simple model with valuable use, for example, having high predictive accuracy.

The following is a list of some of the typical applications of machine learning.

1. In retail business, machine learning is used to study consumer behaviour.
2. In finance, banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market.
3. In manufacturing, learning models are used for optimization, control, and troubleshooting.
4. In medicine, learning programs are used for medical diagnosis.
5. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service.
6. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing and searching for relevant information cannot be done manually.
7. In artificial intelligence, it is used to teach a system to learn and adapt to changes so that the system designer need not foresee and provide solutions for all possible situations.
8. It is used to find solutions to many problems in vision, speech recognition, and robotics.
9. Machine learning methods are applied in the design of computer-controlled vehicles to steer correctly when driving on a variety of roads.
10. Machine learning methods have been used to develop programmes for playing games such as chess, backgammon and Go.

## **PERSPECTIVES AND ISSUES IN MACHINE LEARNING**

### **Perspectives in Machine Learning**

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner. For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights  $w_0$  through  $w_6$ . The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples. The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.



In this notes, we present algorithms that search a hypothesis space defined by some underlying representation (e.g., linear functions, logical descriptions, decision trees, artificial neural networks). These different hypothesis representations are appropriate for learning different kinds of target functions. For each of these hypothesis representations, the corresponding learning algorithm takes advantage of a different underlying structure to organize the search through the hypothesis space.

Throughout this book we will return to this perspective of learning as a search problem in order to characterize learning methods by their search strategies and by the underlying structure of the search spaces they explore. We will also find this viewpoint useful in formally analyzing the relationship between the size of the hypothesis space to be searched, the number of training examples available, and the confidence we can have that a hypothesis consistent with the training data will correctly generalize to unseen examples.

### Issues in Machine Learning

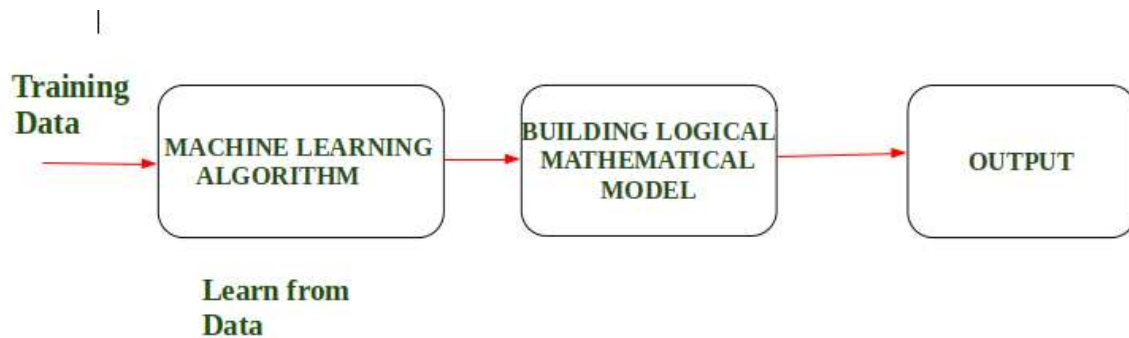
Our checker example raises a number of generic questions about machine learning. The field of machine learning, and much of this book, is concerned with answering questions such as the following:

- 1 What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- 2 How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- 3 When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- 4 What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- 5 What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- 6 How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

**Aspects of developing a learning system: Training data, concept representation, function approximation.**

According to Arthur Samuel “Machine Learning enables a Machine to Automatically learn from Data, Improve performance from an Experience and predict things without explicitly programmed.”

In Simple Words, When we fed the Training Data to Machine Learning Algorithm, this algorithm will produce a mathematical model and with the help of the mathematical model, the machine will make a prediction and take a decision without being explicitly programmed. Also, during training data, the more machine will work with it the more it will get experience and the more efficient result is produced.



**Example :** In Driverless Car, the training data is fed to Algorithm like how to Drive Car in Highway, Busy and Narrow Street with factors like speed limit, parking, stop at signal etc. After that, a Logical and Mathematical model is created on the basis of that and after that, the car will work according to the logical model. Also, the more data the data is fed the more efficient output is produced.

### **Designing a Learning System in Machine Learning :**

According to Tom Mitchell, “A computer program is said to be learning from experience (E), with respect to some task (T). Thus, the performance measure (P) is the performance at task T, which is measured by P, and it improves with experience E.”

**Example:** In Spam E-Mail detection,

- **Task, T:** To classify mails into Spam or Not Spam.
  - **Performance measure, P:** Total percent of mails being correctly classified as being “Spam” or “Not Spam”.
  - **Experience, E:** Set of Mails with label “Spam”
- 
- **Steps for Designing Learning System are:**



•

**Step 1) Choosing the Training Experience:** The very important and first task is to choose the training data or training experience which will be fed to the Machine Learning Algorithm. It is important to note that the data or experience that we fed to the algorithm must have a significant impact on the Success or Failure of the Model. So Training data or experience should be chosen wisely.

Below are the attributes which will impact on Success and Failure of Data:

- The training experience will be able to provide direct or indirect feedback regarding choices. For example: While Playing chess the training data will provide feedback to itself like instead of this move if this is chosen the chances of success increases.
- Second important attribute is the degree to which the learner will control the sequences of training examples. For example: when training data is fed to the machine then at that time accuracy is very less but when it gains experience while playing again and again with itself or opponent the machine algorithm will get feedback and control the chess game accordingly.
- Third important attribute is how it will represent the distribution of examples over which performance will be measured. For example, a Machine learning algorithm will get experience while going through a number of different cases and different examples. Thus, Machine Learning Algorithm will get more and more experience by passing through more and more examples and hence its performance will increase.

**Step 2- Choosing target function:** The next important step is choosing the target function. It means according to the knowledge fed to the algorithm the machine learning will choose NextMove function which will describe what type of legal moves should be taken. For example : While playing chess with the opponent, when opponent will play then the machine learning algorithm will decide what be the number of possible legal moves taken in order to get success.

**Step 3- Choosing Representation for Target function:** When the machine algorithm will know all the possible legal moves the next step is to choose the optimized move using any representation i.e. using linear Equations, Hierarchical Graph Representation, Tabular form etc. The NextMove function will move the Target move like out of these move which will provide more success rate. For Example : while playing chess machine have 4 possible moves, so the machine will choose that optimized move which will provide success to it.

**Step 4- Choosing Function Approximation Algorithm:** An optimized move cannot be chosen just with the training data. The training data had to go through with set of example and through these examples the training data will approximates which steps are chosen and after that machine will provide feedback on it. For Example : When a training data of Playing chess is fed to algorithm so at that time it is not machine algorithm will fail or get success and again from that failure or success it will measure while next move what step should be chosen and what is its success rate.

**Step 5- Final Design:** The final design is created at last when system goes from number of examples , failures and success , correct and incorrect decision and what will be the next step etc. Example: DeepBlue is an intelligent computer which is ML-based won chess game against the chess expert Garry Kasparov, and it became the first computer which had beaten a human chess expert.

## Machine Learning in Practice

### Data preprocessing, Model selection, Evaluation, Deployment, Ethics of machine learning

#### Data preprocessing

Data preprocessing is a crucial step in the machine learning pipeline that involves cleaning, organizing, and transforming raw data into a format suitable for training and evaluating machine learning models. The goal is to enhance the quality of the data, address potential issues, and prepare it for effective model training. Here are key aspects of data preprocessing in the context of machine learning projects:

1. **Handling Missing Data:**
  - Identify and handle missing values in the dataset. This may involve imputing missing values using statistical measures (mean, median, mode) or advanced techniques such as interpolation.
2. **Dealing with Outliers:**
  - Detect and address outliers that could significantly impact model performance. This may involve removing outliers or transforming them to bring them within an acceptable range.
3. **Data Cleaning:**
  - Clean the dataset by addressing inconsistencies, errors, or discrepancies. This includes fixing typos, standardizing formats, and ensuring data consistency.
4. **Feature Scaling:**
  - Normalize or scale features to ensure that all features contribute equally to model training. Common methods include Min-Max scaling or standardization (Z-score normalization).
5. **Encoding Categorical Variables:**
  - Convert categorical variables into a format suitable for machine learning algorithms. This often involves one-hot encoding, where categorical variables are transformed into binary vectors.
6. **Handling Imbalanced Data:**
  - Address class imbalances in classification tasks to prevent models from being biased towards the majority class. Techniques include oversampling, undersampling, or using specialized algorithms for imbalanced data.
7. **Feature Engineering:**
  - Create new features or transform existing ones to capture more relevant information for the model. Feature engineering can improve a model's ability to understand complex relationships within the data.
8. **Data Splitting:**
  - Split the dataset into training, validation, and test sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the test set evaluates the model's performance on unseen data.
9. **Handling Time-Series Data:**
  - For time-series data, handle temporal aspects such as time gaps, missing values, and seasonality. This may involve resampling, lag features, or other time-specific preprocessing steps.

## 10. Documentation and Logging:

- Document all steps taken during data preprocessing and maintain a record of transformations applied. Logging helps in reproducing results, debugging, and ensuring transparency in the machine learning pipeline.

## Model Selection in Machine Learning Projects

### Model Selection in Machine Learning Projects: Picking the Right Tool for the Job

Choosing the right model for your ML project is like picking the perfect tool for a DIY project. You wouldn't use a screwdriver for hammering, and you wouldn't use a chainsaw for trimming nails! Similarly, in the vast toolbox of ML algorithms, selecting the best one for your specific problem is crucial for success. Let's break down the practical side of model selection:

#### Why is it important?

Imagine throwing every tool you own at a problem! It's probably going to be messy and inefficient. Likewise, using the wrong model in your project can lead to:

- Poor performance: Inaccurate predictions, weak generalizability, and wasted resources.
- Overfitting: Memorizing the training data but failing on unseen examples.
- Underfitting: Failing to capture the underlying patterns in the data.

#### Factors to Consider:

- Problem type: Regression, classification, clustering, etc. Each requires different types of models.
- Data characteristics: Size, dimensionality, complexity, distribution. Some models work better with specific data types.
- Project constraints: Training time, computational resources, model interpretability. Different models have different demands.
- Business needs: Accuracy, explainability, real-time predictions. Prioritize metrics relevant to your goals.

#### Common Model Selection Techniques:

- Hold-out validation: Split your data into training and testing sets, train models on the training set, and evaluate them on the unseen testing set.
- K-fold cross-validation: Randomly split your data into k folds, train models on k-1 folds, and test on the remaining fold, repeat for all k folds, and average the results.
- Grid search and hyperparameter tuning: Explore different combinations of model parameters (hyperparameters) to find the optimal configuration for your data.
- Model comparison metrics: Accuracy, precision, recall, F1-score, AUC-ROC. Choose metrics that align with your project goals.

### **Real-world Example:**

Building a model to predict house prices. You might consider:

- Regression models: Linear regression, decision trees, or gradient boosting for continuous price prediction.
- Feature engineering: Create features like "distance to amenities" or "average price in the neighborhood".
- Model comparison: Use k-fold cross-validation to compare performance of different models on the same data.
- Choose the model: Select the one with the best average score on the chosen metric (e.g., mean squared error) while considering additional factors like interpretability for explaining price variations.

### **Tips for Effective Selection:**

- Start simple: Don't jump into complex models too quickly. Consider baseline models for initial comparisons.
- Iterate and refine: Keep trying different models, hyper parameters, and feature engineering techniques.
- Understand the trade-offs: No model is perfect. Balance accuracy with factors like training time, interpretability, and resource constraints.
- Visualize your results: Use plots and charts to understand model behavior and compare performance.

Remember, model selection is an art, not a science. There's no one-size-fits-all solution. By carefully considering your specific project context and applying these practical tips, you can choose the right tool for the job and build successful ML models that deliver real value.

### **Model evaluation**

Model evaluation is a critical aspect of machine learning projects that involves assessing the performance and generalization ability of a trained model. It helps determine how well the model is likely to perform on new, unseen data. Here are key considerations and techniques for model evaluation in the context of machine learning projects:

#### **1. Metrics Selection:**

- Choose appropriate evaluation metrics based on the nature of the problem. Common metrics include accuracy, precision, recall, F1 score for classification tasks, and mean squared error, R-squared for regression tasks. The choice of metric depends on the project goals and the specific characteristics of the data.

#### **2. Confusion Matrix:**

- For classification problems, construct a confusion matrix to visualize the model's performance in terms of true positives, true negatives, false positives, and false negatives. This aids in understanding the types of errors the model makes.

#### **3. Cross-Validation:**

- Implement cross-validation techniques, such as k-fold cross-validation or leave-one-out cross-validation, to robustly assess model performance. This helps mitigate the impact of variability in the training and test datasets.

#### **4. Learning Curves:**

- Analyze learning curves to understand how the model's performance changes with respect to the amount of training data. This helps identify issues like overfitting or underfitting.

5. **ROC Curve and AUC-ROC:**

- For binary classification problems, plot Receiver Operating Characteristic (ROC) curves and calculate the Area Under the Curve (AUC-ROC). This provides insights into the trade-off between sensitivity and specificity.

6. **Precision-Recall Curve:**

- For imbalanced classification problems, examine precision-recall curves, which illustrate the precision-recall trade-off at different decision thresholds.

7. **Feature Importance:**

- Assess the importance of features in the model to understand which features contribute most to the predictions. This can be done using techniques such as permutation importance or feature importance plots.

8. **Model Comparison:**

- Compare the performance of multiple models to select the best-performing one. This can involve statistical tests or visualizations to highlight differences in performance.

9. **Bias and Fairness Evaluation:**

- Evaluate the model for bias and fairness, especially in sensitive applications. Analyze the model's behavior across different demographic groups to ensure equitable outcomes.

10. **Deployment Considerations:**

- Consider the practical implications of deploying the model, including the potential impact on end-users and the broader environment. Evaluate the model's robustness to changes in the input data distribution.

## **Model deployment**

Model deployment in the context of machine learning projects refers to the process of taking a trained machine learning model and integrating it into a production environment where it can be used to make predictions on new, unseen data. It involves several steps to ensure that the model functions effectively, reliably, and securely in real-world applications. Here's an overview of the key considerations in model deployment:

1. **Model Serialization:**

- Serialize the trained model to a format that can be easily loaded and utilized by the deployment environment. Common serialization formats include pickle, joblib, or formats compatible with specific deployment frameworks.

2. **Scalability:**

- Ensure that the deployed model can scale to handle varying workloads and data volumes. This may involve using scalable infrastructure, containerization (e.g., Docker), or cloud-based solutions.

3. **API Development:**

- Create an API (Application Programming Interface) to expose the model's functionality, allowing other software applications to interact with and make predictions using the model. RESTful APIs are commonly used for this purpose.

4. **Input Validation:**

- Implement robust input validation to ensure that the incoming data meets the model's expectations. This includes checking data types, ranges, and handling missing values appropriately.



**5. Security Measures:**

- Implement security measures to protect the deployed model from potential vulnerabilities. This may involve encryption of data in transit, access controls, and regular security audits.

**6. Monitoring and Logging:**

- Set up monitoring systems to track the model's performance in real-time. Logging should capture relevant information, including input data, predictions, and any issues that may arise during deployment.

**7. Versioning:**

- Establish version control for the deployed models to track changes and updates. This ensures that different versions of the model can be easily managed, rolled back, or upgraded without disrupting the system.

**8. Model A/B Testing:**

- Implement A/B testing methodologies to evaluate the performance of different model versions in a live environment. This allows for data-driven decisions regarding model improvements or changes.

**9. Downtime Mitigation:**

- Plan for and mitigate downtime during updates or maintenance. This may involve deploying redundant instances, load balancing, or using strategies like canary releases to minimize service interruptions.

**10. Documentation:**

- Create comprehensive documentation for the deployed model, including information on the API, input requirements, output format, and any specific considerations for users or developers interacting with the model.

Model deployment is a critical phase that bridges the gap between the development of machine learning models and their practical application in real-world scenarios. Effective deployment ensures that the benefits of the trained model are realized in production while maintaining performance, reliability, and security. It requires collaboration between data scientists, software engineers, and DevOps teams to create a seamless and efficient deployment pipeline.

### **Ethics in machine learning**

Ethics in machine learning refers to the responsible and fair development, deployment, and use of machine learning models and algorithms. In the context of machine learning projects, addressing ethical considerations is crucial to ensure that the technology benefits society without causing harm or reinforcing biases. Here are key aspects of ethics in machine learning projects:

**1. Fairness and Bias:**

- Assess and mitigate biases in the training data and model predictions. Pay attention to potential disparities across different demographic groups to avoid reinforcing or exacerbating existing societal biases.

**2. Transparency:**

- Strive for transparency in model development and decision-making processes. Clearly communicate how models work, the data used, and the potential impact of model predictions. Transparent models enhance trust and accountability.

**3. Accountability:**

- Establish accountability for the impact of machine learning models. Clearly define roles and responsibilities, and hold individuals and organizations

accountable for the ethical implications of their models.

4. **Data Privacy:**

- Prioritize data privacy by implementing measures to protect sensitive information. Comply with data protection regulations, and consider anonymization or encryption techniques to safeguard user privacy.

5. **Informed Consent:**

- Obtain informed consent from individuals whose data is used for training models. Clearly communicate the purposes of data collection and how the data will be used, giving users the option to opt in or opt out.

6. **Robustness and Reliability:**

- Ensure that machine learning models are robust and reliable across different contexts. Consider potential adversarial attacks and implement measures to enhance model robustness in real-world scenarios.

7. **Interpretability:**

- Design models that are interpretable and understandable by humans. Interpretability aids in understanding model decisions, allowing stakeholders to identify and rectify potential biases or ethical concerns.

8. **Continual Monitoring:**

- Implement ongoing monitoring of deployed models to identify and address ethical issues that may arise as the model interacts with new data in a live environment.

9. **Stakeholder Engagement:**

- Involve diverse stakeholders, including affected communities, in the decision-making process. Gather input from individuals who may be impacted by the use of machine learning models to ensure a more comprehensive understanding of potential ethical implications.

10. **Societal Impact Assessment:**

- Conduct a societal impact assessment to anticipate and evaluate the broader consequences of deploying machine learning models. This includes considering potential economic, social, and environmental impacts.

Ethics in machine learning is an evolving field, and it requires a proactive and collaborative effort from data scientists, developers, policymakers, and other stakeholders. By incorporating ethical considerations into the entire machine learning lifecycle, practitioners can help build and deploy models that contribute positively to society while minimizing risks and negative consequences. Regularly reassessing ethical practices and staying informed about evolving ethical guidelines is essential in the rapidly changing landscape of machine learning technology.

## UNIT-II

**Regression:** Introduction to Regression analysis, measure of linear relationship, Regression with stats models, Determining coefficient, meaning and significance of coefficients, coefficient calculation with least square method, Types of regression, Simple linear regression, Using simple features, Polynomial Regression, Metrics for Regression: MSE, RMSE, MAE.

### Introduction to Regression analysis:

Regression Analysis is a statistical process for estimating the relationships between the dependent variables or criterion variables and one or more independent variables or predictors. Regression analysis explains the changes in criteria in relation to changes in select predictors. The conditional expectation of the criteria is based on predictors where the average value of the dependent variables is given when the independent variables are changed. Three major uses for regression analysis are determining the strength of predictors, forecasting an effect, and trend forecasting.

### Measure of linear Relationship:

1. **Linear regression** is used for predictive analysis. Linear regression is a linear approach for modelling the relationship between the criterion or the scalar response and the multiple predictors or explanatory variables. Linear regression focuses on the conditional probability distribution of the response given the values of the predictors. For linear regression, there is a danger of overfitting. The formula for linear regression is:  $Y' = bX + A$ .
2. **Polynomial regression** is used for curvilinear data. Polynomial regression is fit with the method of least squares. The goal of regression analysis is to model the expected value of a dependent variable  $y$  in regards to the independent variable  $x$ . The equation for polynomial regression is:

The diagram shows the linear regression equation  $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$ . Labels with arrows point to each term: 'Dependent Variable' points to  $Y_i$ , 'Population Y intercept' points to  $\beta_0$ , 'Population Slope Coefficient' points to  $\beta_1$ , 'Independent Variable' points to  $X_i$ , and 'Random Error term' points to  $\epsilon_i$ . A bracket under  $\beta_0 + \beta_1 X_i$  is labeled 'Linear component', and a bracket under  $\epsilon_i$  is labeled 'Random Error component'.

3. **Stepwise regression** is used for fitting regression models with predictive models. It is carried out automatically. With each step, the variable is added or subtracted from the set of explanatory variables. The approaches for stepwise regression are forward selection, backward elimination, and bidirectional elimination.

$$b_{j.std} = b_j (s_x * s_y^{-1}).$$

4. **Ridge regression** is a technique for analyzing multiple regression data. When multicollinearity occurs, least squares estimates are unbiased. A degree of bias is added to the regression estimates, and as a result, ridge regression reduces the standard errors. The formula for ridge

$$\beta = (X^T X + \lambda * I)^{-1} X^T y.$$

**Lasso regression** is a regression analysis method that performs both variable selection and regularization. Lasso regression uses soft thresholding. Lasso regression selects only a subset of the provide

covariates for use in the final model. Lasso regression is

$$N^{-1} \sum_{i=1}^N f(x_i, y_i, \alpha, \beta).$$

5. **ElasticNet regression** is a regularized regression method that linearly combines the penalties of the lasso and ridge methods. ElasticNet regression is used for support vector machines, metric learning, and portfolio optimization.

The penalty function is given by  $\|\beta\|_1 = \sum_{j=1}^p |\beta_j|$ .

Regression with stats models:

Linear models with independently and identically distributed errors, and for errors with heteroscedasticity or autocorrelation. This module allows estimation by ordinary least squares (OLS), weighted least squares (WLS), generalized least squares (GLS), and feasible generalized least squares with autocorrelated AR(p) errors.

## Ordinary Least Square (OLS) Method for Linear Regression

This post is about the ordinary least square method (OLS) for simple linear regression. If you are new to linear regression, read this article for getting a clear idea about the implementation of simple linear regression. This post will help you to understand how simple linear regression works step-by-step.

The simple linear regression is a model with a single regressor (independent variable)  $x$  that has a relationship with a response (dependent or target)  $y$  that is a

$$y = \beta_0 + \beta_1 x + \varepsilon \text{ — — — — — (1)}$$

Where  $\beta_0$ : intercept

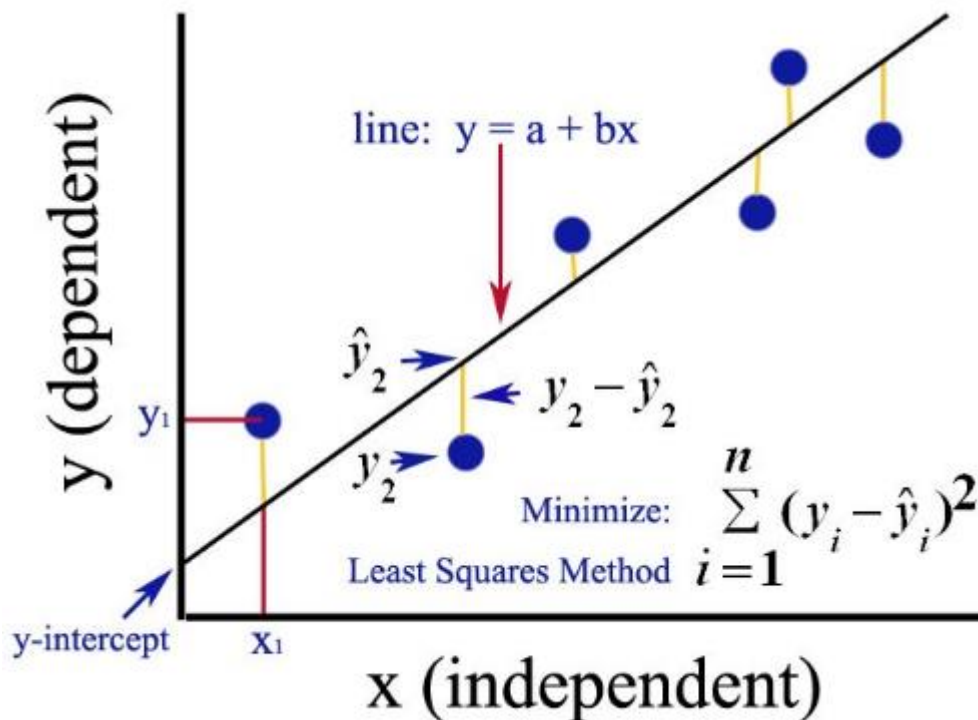
$\beta_1$ : slope (unknown constant)

$\varepsilon$ : random error component

This is a line where  $y$  is the dependent variable we want to predict,  $x$  is the independent variable, and  $\beta_0$  and  $\beta_1$  are the coefficients that we need to estimate.

### Estimation of $\beta_0$ and $\beta_1$ :

The OLS method is used to estimate  $\beta_0$  and  $\beta_1$ . The OLS method seeks to minimize the sum of the squared residuals. This means from the given data we calculate the distance from each data point to the regression line, square it, and the sum of all of the squared errors together.



From equation (1) we may write

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, i = 1, 2, \dots, n \text{ --- (2)}$$

The equation (2) is a sample regression model, written in terms of the  $n$  pairs of data  $(y_i, x_i)$  ( $i = 1, 2, \dots, n$ ). Thus, the least-squares criteria are

The least square criteria is,

$$S(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$$

The least square estimators of  $\beta_0$  &  $\beta_1$ , say  $\hat{\beta}_0$  &  $\hat{\beta}_1$  must satisfy

$$\left. \frac{\partial S}{\partial \beta_0} \right|_{\hat{\beta}_0, \hat{\beta}_1} = -2 \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0 \text{ --- (1)}$$

$$\left. \frac{\partial S}{\partial \beta_1} \right|_{\hat{\beta}_0, \hat{\beta}_1} = -2 \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0 \text{ --- (2)}$$

simplifying these two equations

Eq<sup>^</sup> — (1)

$$-2 \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\sum_{i=1}^n y_i = n \hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^n x_i$$

$$\hat{\beta}_0 = \frac{\sum_{i=1}^n y_i - \hat{\beta}_1 \sum_{i=1}^n x_i}{n}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$



$$\text{Eq}^n \text{ --- (2)}$$

$$-2 \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0$$

$$\sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) x_i = 0$$

$$\sum_{i=1}^n x_i y_i - \hat{\beta}_0 \sum_{i=1}^n x_i - \hat{\beta}_1 \sum_{i=1}^n x_i^2 = 0$$

$$\sum_{i=1}^n (x_i y_i - (\bar{y} - \hat{\beta}_1 \bar{x}) x_i - \hat{\beta}_1 x_i^2) = 0$$

$$\sum_{i=1}^n (x_i y_i - \bar{y} x_i + \hat{\beta}_1 \bar{x} x_i - \hat{\beta}_1 x_i^2) = 0$$



$$\sum_{i=1}^n (y_i - \bar{y} + \hat{\beta}_1 \bar{x} - \hat{\beta}_1 x_i) x_i = 0$$

$$\sum_{i=1}^n (y_i - \bar{y}) + \hat{\beta}_1 (\bar{x} - x_i) = 0$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad \bar{y} = \frac{\sum_{i=1}^n y_i}{n}, \quad \bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

Ordinary Least Square Method

Year	Sales (Million Euro)	Advertising (Million Euro)
1	651	25
2	762	28
3	853	35
4	1062	40
5	1190	46
6	1293	53

Let's take a simple example. This table shows some data from the manufacturing company. Each row in the table shows the sales for a year and the amount spent on advertising that year. Here our target variable is sales-which we want to predict.

Linear Regression estimates that  $\text{Sales} = \beta_0 + \beta_1 * (\text{Advertising})$

### Estimating the Slope ( $\beta_1$ ):

1. Calculate the mean value of x and y

Year	Sales (Million Euro)	Advertising (Million Euro)
	Y	X
1	651	25
2	762	28
3	853	35
4	1062	40
5	1190	46
6	1293	53
Mean of Y = 968.5		Mean of X = 37.83333333

2. Calculate the error of each variable from the mean

Year	Sales (Million Euro)	Advertising (Million Euro)	$Y - \bar{Y}$	$X - \bar{X}$
	Y	X		
1	651	25	-317.5	-12.83
2	762	28	-186.5	-9.83
3	853	35	-95.5	-2.83
4	1062	40	90.5	2.17
5	1190	46	221.5	8.17
6	1293	53	324.5	15.17
Mean of Y = 968.5		Mean of X = 37.83333333		

3. Multiply the error for each x with the error for each y and calculate the sum of these multiplications

Year	Sales (Million Euro) Y	Advertising (Million Euro) X	$Y - \bar{Y}$	$X - \bar{X}$	$(Y - \bar{Y}) * (X - \bar{X})$
1	651	25	-317.5	-12.83	4074.58333
2	762	28	762	28	21336
3	853	35	853	35	29855
4	1062	40	1062	40	42480
5	1190	46	1190	46	54740
6	1293	53	1293	53	68529
<b>Mean of Y =</b> <b>968.5</b>		<b>Mean of X =</b> <b>37.83333333</b>		<b>SUM=</b> <b>221014.583</b>	

4. Square the residual of each x value from the mean and sum of these squared values

Year	Sales (Million Euro) Y	Advertising (Million Euro) X	$Y - \bar{Y}$	$X - \bar{X}$	$(Y - \bar{Y}) * (X - \bar{X})$	$(X - \bar{X})^2$
1	651	25	-317.5	-12.83	4074.583333	164.69444
2	762	28	762	28	21336	784
3	853	35	853	35	29855	1225
4	1062	40	1062	40	42480	1600
5	1190	46	1190	46	54740	2116
6	1293	53	1293	53	68529	2809
<b>Mean of Y =</b> <b>968.5</b>		<b>Mean of X =</b> <b>37.83333333</b>		<b>SUM=</b> <b>221014.5833</b>	<b>SUM=</b> <b>8698.694</b>	

Now we have all the values to calculate the slope ( $\beta_1$ ) =  
 $221014.5833 / 8698.694 = 25.41$

### Estimating the Intercept ( $\beta_0$ ):

$$\beta_0 = \text{mean}(y) - (\beta_1 * \text{mean}(x))$$

we already know all the values to calculate  $\beta_0$ .

$$\beta_0 = 968.5 - (37.83333333 * 25.41) = 7.239$$

## Making Predictions:

Now, we have the coefficients for our simple linear regression.

$$y = 7.239 + 25.41 * x$$

Let's make predictions for our data.

Year	Sales (Million Euro) Y	Advertising (Million Euro) X	Y-ȳ	X- $\bar{x}$	(Y-ȳ)*(X- $\bar{x}$ )	(X- $\bar{x}$ )^2	Predictions
1	651	25	-317.5	-12.83	4074.583333	164.694444	642.433389
2	762	28	762	28	21336	784	718.656752
3	853	35	853	35	29855	1225	896.511268
4	1062	40	1062	40	42480	1600	1023.55021
5	1190	46	1190	46	54740	2116	1175.99693
6	1293	53	1293	53	68529	2809	1353.85145
Mean of Y = 968.5		Mean of X = 37.83333333		SUM= 221014.5833		SUM= 8698.6944	
				beta1= 25.40778789		beta0= 7.2386916	

**Estimating Error:** Calculate the error for our predictions (Root Mean Squared Error or RMSE)

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (p_i - y_i)^2}{n}}$$

Root Mean Squared Error

Year	Sales (Million Euro) Yi	Advertising (Million Euro) X	Y-ȳ	X- $\bar{x}$	(Y-ȳ)*(X- $\bar{x}$ )	(X- $\bar{x}$ )^2	Predictions (Pi)	(Pi-Yi)	(Pi-Yi)^2
1	651	25	-317.5	-12.83	4074.583333	164.694444	642.433388	-8.566611209	73.386828
2	762	28	762	28	21336	784	718.656752	-43.34324755	1878.6371
3	853	35	853	35	29855	1225	896.511267	43.51126766	1893.2304
4	1062	40	1062	40	42480	1600	1023.550207	-38.44979291	1478.3866
5	1190	46	1190	46	54740	2116	1175.996934	-14.00306559	196.08585
6	1293	53	1293	53	68529	2809	1353.85145	60.85144961	3702.8989
Mean of Y = 968.5		Mean of X = 37.83333333		SUM= 221014.5833		SUM= 8698.6944		SUM= 9222.6257	
				beta1= 25.40778789		beta0= 7.2386916		[[ (Pi-Yi)^2 ]/n = 1537.1043	
								RMSE = 39.205922	

From the calculated RMSE we can say that each prediction is on average wrong by about 39.2059 units.

## Weighted Least Squares:

The method of ordinary least squares assumes that there is constant variance in the errors (which is called homoscedasticity). The method of weighted least squares can be used when the ordinary least squares assumption of constant variance in the errors is violated (which is called heteroscedasticity). The model under consideration is

$$Y = X\beta + \epsilon^*,$$

where  $\epsilon^*$  is assumed to be (multivariate) normally distributed with mean vector  $\mathbf{0}$  and nonconstant variance-covariance matrix

$$(\sigma_1^2 \ 0 \dots 0 \ 0 \ \sigma_2^2 \dots 0 \ \vdots \ \vdots \ 0 \dots 0 \ \sigma_n^2)$$

If we define the reciprocal of each variance,  $\sigma_i^2$ , as the weight,  $w_i = 1/\sigma_i^2$ , then let matrix  $\mathbf{W}$  be a diagonal matrix containing these weights:

$$\mathbf{W} = (w_1 \ 0 \dots 0 \ 0 \ w_2 \dots 0 \ \vdots \ \vdots \ 0 \dots 0 \ w_n)$$

The weighted least squares estimate is then

$$\hat{\beta}^{WLS} = \arg \min_{\beta} \sum_{i=1}^n w_i (y_i - x_i \beta)^2 = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{Y}$$

With this setting, we can make a few observations:

- Since each weight is inversely proportional to the error variance, it reflects the information in that observation. So, an observation with small error variance has a large weight since it contains relatively more information than an observation with large error variance (small weight).
- The weights have to be known (or more usually estimated) up to a proportionality constant.

To illustrate, consider the famous 1877 [Galton data set](#), consisting of 7 measurements each of  $X = \text{Parent}$  (pea diameter in inches of parent plant) and  $Y = \text{Progeny}$  (average pea diameter in inches of up to 10 plants grown from seeds of the parent plant). Also included in the dataset are standard deviations, SD, of the offspring peas grown from each parent. These standard deviations reflect the information in the response  $Y$  values (remember these are averages) and so in estimating a regression model we should downweight the observations with a large standard deviation and upweight the observations with a small

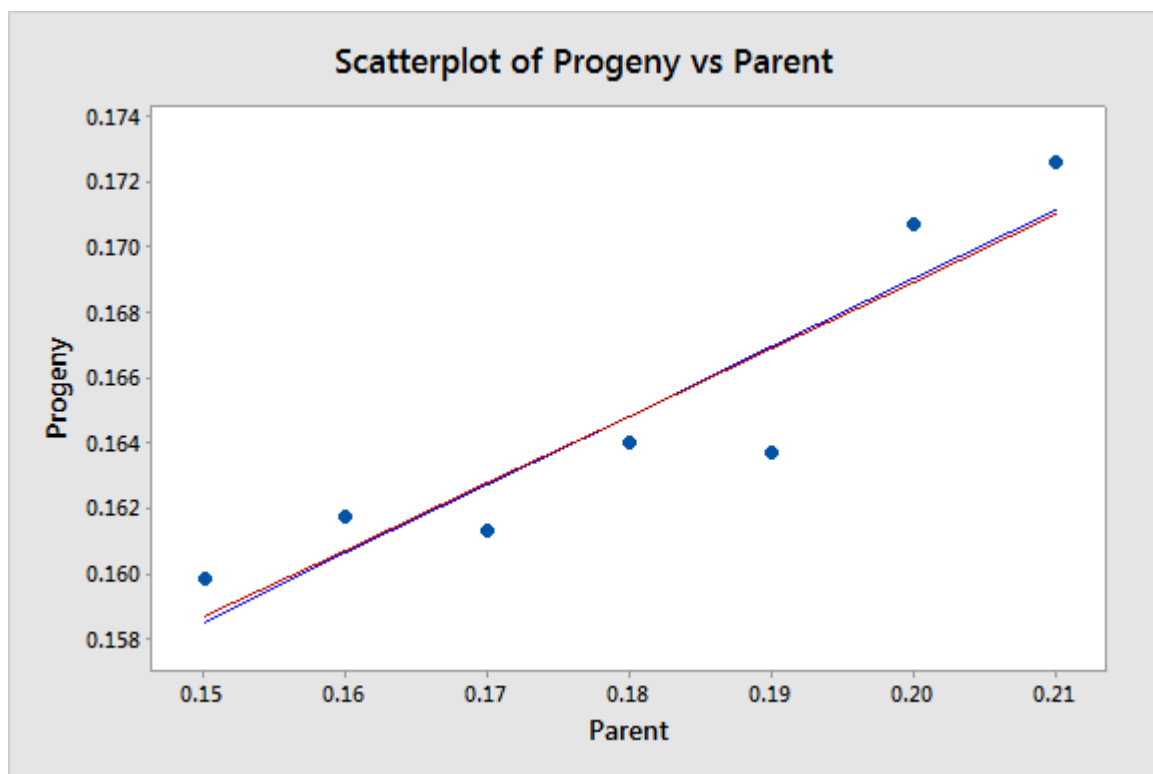
standard deviation. In other words we should use weighted least squares with weights equal to  $1/SD^2$ . The resulting fitted equation from Minitab for this model is:

$$\text{Progeny} = 0.12796 + 0.2048 \text{ Parent}$$

Compare this with the fitted equation for the ordinary least squares model:

$$\text{Progeny} = 0.12703 + 0.2100 \text{ Parent}$$

The equations aren't very different but we can gain some intuition into the effects of using weighted least squares by looking at a scatterplot of the data with the two regression lines superimposed:



The black line represents the OLS fit, while the red line represents the WLS fit. The standard deviations tend to increase as the value of Parent increases, so the weights tend to decrease as the value of Parent increases. Thus, on the left of the graph where the observations are upweighted the red fitted line is pulled slightly closer to the data points, whereas on the right of the graph where the observations are downweighted the red fitted line is slightly further from the data points.

For this example the weights were known. There are other circumstances where the weights are known:



- If the  $i$ -th response is an average of  $n_i$  equally variable observations, then  $\text{Var}(y_i) = \sigma^2/n_i$  and  $w_i = n_i$ .
- If the  $i$ -th response is a total of  $n_i$  observations, then  $\text{Var}(y_i) = n_i\sigma^2$  and  $w_i = 1/n_i$ .
- If variance is proportional to some predictor  $x_i$ , then  $\text{Var}(y_i) = x_i\sigma^2$  and  $w_i = 1/x_i$ .

In practice, for other types of dataset, the structure of  $\mathbf{W}$  is usually unknown, so we have to perform an ordinary least squares (OLS) regression first. Provided the regression function is appropriate, the  $i$ -th squared residual from the OLS fit is an estimate of  $\sigma_i^2$  and the  $i$ -th absolute residual is an estimate of  $\sigma_i$  (which tends to be a more useful estimator in the presence of outliers). The residuals are much too variable to be used directly in estimating the weights,  $w_i$ , so instead we use either the squared residuals to estimate a variance function or the absolute residuals to estimate a standard deviation function. We then use this variance or standard deviation function to estimate the weights.

---

**Some possible variance and standard deviation function estimates include:**

- If a residual plot against a predictor exhibits a megaphone shape, then regress the absolute values of the residuals against that predictor. The resulting fitted values of this regression are estimates of  $\sigma_i$ . (And remember  $w_i = 1/\sigma_i^2$ ).
- If a residual plot against the fitted values exhibits a megaphone shape, then regress the absolute values of the residuals against the fitted values. The resulting fitted values of this regression are estimates of  $\sigma_i$ .
- If a residual plot of the squared residuals against a predictor exhibits an upward trend, then regress the squared residuals against that predictor. The resulting fitted values of this regression are estimates of  $\sigma_i^2$ .
- If a residual plot of the squared residuals against the fitted values exhibits an upward trend, then regress the squared residuals against the fitted values. The resulting fitted values of this regression are estimates of  $\sigma_i^2$ .

After using one of these methods to estimate the weights,  $w_i$ , we then use these weights in estimating a weighted least squares regression model. We consider some examples of this approach in the next section.

---

**Some key points regarding weighted least squares are:**

1. The difficulty, in practice, is determining estimates of the error variances (or standard deviations).



2. Weighted least squares estimates of the coefficients will usually be nearly the same as the "ordinary" unweighted estimates. In cases where they differ substantially, the procedure can be iterated until estimated coefficients stabilize (often in no more than one or two iterations); this is called *iteratively reweighted least squares*.
3. In some cases, the values of the weights may be based on theory or prior research.
4. In designed experiments with large numbers of replicates, weights can be estimated directly from sample variances of the response variable at each combination of predictor variables.
5. Use of weights will (legitimately) impact the widths of statistical intervals.

## Generalized least squares:

The generalized least squares (GLS) estimator of the coefficients of a [linear regression](#) is a generalization of the ordinary least squares (OLS) estimator. It is used to deal with situations in which the OLS estimator is not BLUE (best linear unbiased estimator) because one of the main assumptions of the [Gauss-Markov theorem](#), namely that of homoskedasticity and absence of serial correlation, is violated. In such situations, provided that the other assumptions of the Gauss-Markov theorem are satisfied, the GLS estimator is BLUE.

## Setting

The linear regression is  $y = X\beta + \epsilon$  where:

- $y$  is an  $n \times 1$  vector of outputs (  $n$  is the [sample size](#));
- $X$  is an  $n \times k$  matrix of regressors (  $k$  is the number of regressors);
- $\beta$  is the  $k \times 1$  vector of regression coefficients to be estimated;
- $\epsilon$  is an  $n \times 1$  vector of error terms.

We assume that:

1.  $X$  has full rank;
2.  $\epsilon$  is normally distributed with mean zero and constant variance  $\sigma^2$ ;
3.  $\epsilon$  is uncorrelated, where  $\Sigma$  is a symmetric positive definite matrix.

These assumptions are the same made in the Gauss-Markov theorem in order to prove that OLS is BLUE, except for assumption 3.

In the Gauss-Markov theorem, we make the more restrictive assumption that

where  $I_n$  is the  $n \times n$  identity matrix. The latter assumption means that the errors of the regression are homoskedastic (they all have the same variance) and uncorrelated (their covariances are all equal to zero).

Instead, we now allow for heteroskedasticity (the errors can have different variances) and correlation (the covariances between errors can be different from zero).

### The GLS estimator

Since  $V$  is symmetric and positive definite, there is an invertible matrix  $P$  such that

If we pre-multiply the regression equation by  $P$ , we obtain

Define  $\tilde{y} = Py$  so that the transformed regression equation can be written as

The following proposition holds.

**Proposition** The OLS estimator of the coefficients of the transformed regression equation,

$$\begin{aligned}\hat{\beta}_{GLS} &= (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \tilde{y} \\ &= (X^T V^{-1} X)^{-1} X^T V^{-1} y\end{aligned}$$

called generalized least squares estimator, is

Furthermore,  $\hat{\beta}_{GLS}$  is BLUE (best linear unbiased).

[Proof](#)

### The generalized least squares problem

Remember that the OLS estimator of a linear regression solves the problem

$$\hat{\beta}_{OLS} = \arg \min_b (y - Xb)^T (y - Xb)$$

that is, it minimizes the sum of squared residuals.

The GLS estimator can be shown to solve the problem

$$\hat{\beta}_{GLS} = \arg \min_b (y - Xb)^T V^{-1} (y - Xb)$$

which is called generalized least squares problem.

[Proof](#)

$$(y - Xb)^T V^{-1} (y - Xb) = [\Sigma^{-1} (y - Xb)]^T [\Sigma^{-1} (y - Xb)]$$

The function to be minimized can be written as

It is also a sum of squared residuals, but the original residuals are rescaled by before being squared and summed.

### Weighted least squares

When the covariance matrix is diagonal (i.e., the error terms are uncorrelated), the GLS estimator is called weighted least squares estimator (WLS). In this case the function to be

$$(y - Xb)^T V^{-1} (y - Xb) = \sum_{i=1}^N V_{ii}^{-1} (y_i - X_i b)^2$$

minimized becomes where is the -th entry of , is the -th row of , and is the -th diagonal element of . Thus, we are minimizing a weighted sum of the squared residuals, in which each squared residual is weighted by the reciprocal of its variance. In other words, while estimating , we are giving less weight to the observations for which the linear relationship to be estimated is more noisy, and more weight to those for which it is less noisy.

### Feasible generalized least squares

Note that we need to know the [covariance matrix](#) in order to actually compute . In practice, we seldom know and we replace it with an estimate . The estimator thus

$$\hat{\beta}_{FGLS} = \left( X^T \hat{V}^{-1} X \right)^{-1} X^T \hat{V}^{-1} y$$

obtained, that is, is called **feasible generalized least squares** estimator.

There is no general method for estimating , although the residuals of a first-step OLS regression are typically used to compute . How the problem is approached depends on the specific application and on additional assumptions that may be made about the process generating the errors of the regression.

**Example** A typical situation in which is estimated by running a first-step OLS regression is when the observations are indexed by time. For example, we could assume that is diagonal and estimate its diagonal elements with an exponential moving average

$$\hat{V}_{i,i} = \alpha \hat{V}_{i-1,i-1} + (1 - \alpha) \hat{\varepsilon}_i^2 \quad \text{where}$$

## Determining coefficient, meaning and significance of coefficients:

### Regression Coefficients:

Regression coefficients are the quantities by which the variables in a regression equation are multiplied. The most commonly used type of regression is linear regression. The aim of linear regression is to find the regression coefficients that produce the best-fitted line.

The regression coefficients in linear regression help in predicting the value of an unknown variable using a known variable. In this article, we will learn more about regression coefficients, their formulas as well as see certain associated examples so as to find the best-fitted regression line.

### What are Regression Coefficients:

Regression coefficients can be defined as estimates of some unknown parameters to describe the relationship between a predictor variable and the corresponding response. In other words, regression coefficients are used to predict the value of an unknown [variable](#) using a known variable. Linear regression is used to quantify how a unit change in an independent variable causes an effect in the dependent variable by determining the equation of the best-fitted [straight line](#). This process is known as regression analysis.

### Formula for Regression Coefficients:

The goal of linear regression is to find the equation of the straight line that best describes the relationship between two or more variables. For example, suppose a simple regression equation is given by  $y = 7x - 3$ , then 7 is the [coefficient](#),  $x$  is the predictor and -3 is the constant term. Suppose the equation of the best-fitted line is given by  $Y = aX + b$  then, the regression coefficients formula is given as follows:

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

$$b = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$$

here,  $n$  refers to the number of data points in the given data sets.

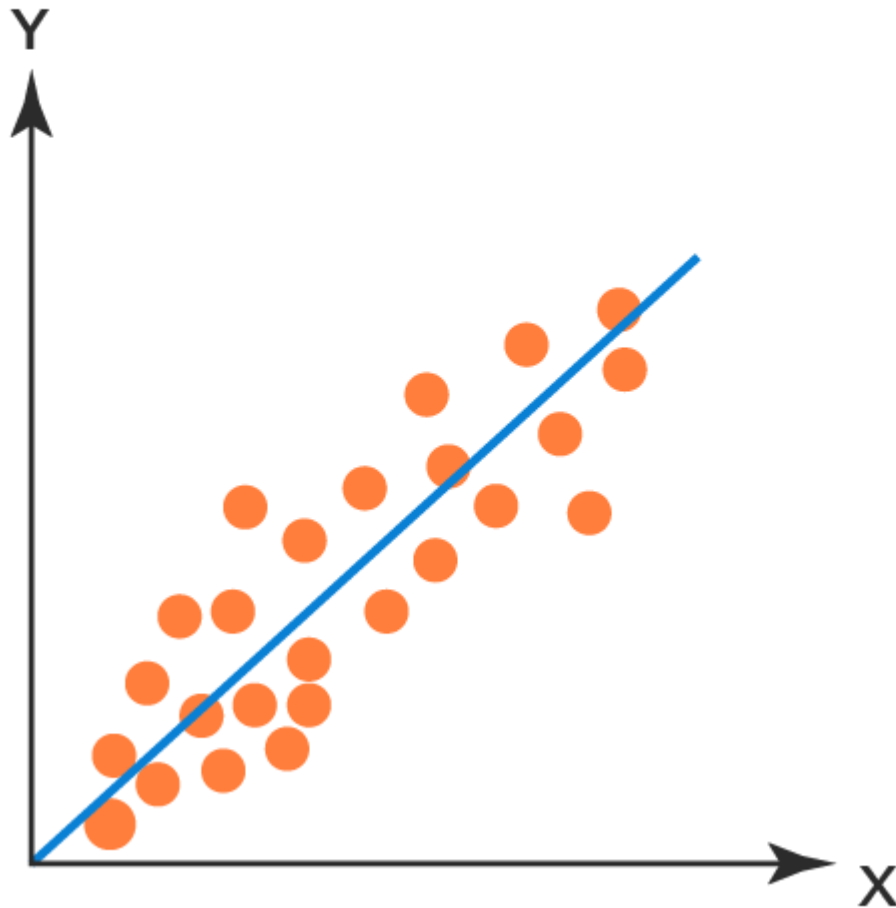
### How to Find Regression Coefficients:

Before determining the regression coefficients to find the best-fitted line, it is necessary to check whether the variables follow a linear relationship or not. This can be done by using the [correlation coefficient](#) and interpreting the corresponding value. Given below are the steps to find the regression coefficients for regression analysis.

- To find the coefficient of  $X$  use the formula  $a$   
$$= \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$
- To find the constant term the formula is  $b$   
$$= \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$$
- Now input the regression coefficients in the equation  $Y = aX + b$ .

- A scatter plot can also be made so as to visually depict the regression line as shown below.

## Regression Analysis Graph



### Regression Coefficients Interpretation:

It is necessary to understand the nature of the regression coefficient as this helps to make certain predictions about the unknown variable. It helps to check to what extent a dependent variable will change with a unit change in the independent variable. Given below are the regression coefficients interpretation.

- If the sign of the coefficients is positive it implies that there is a direct relationship between the variables. This means that if the independent variable increases (or decreases) then the dependent variable also increases (or decreases).
- If the sign of the coefficients is negative it means that if the independent variable increases then the dependent variable decreases and vice versa. This means it is an indirect relationship.

### Important Notes on Regression Coefficients

- Regression coefficients are values that are used in a regression equation to estimate the predictor variable and its response.

- The most commonly used type of regression is linear regression. The equation of the best-fitted line is given by  $Y = aX + b$ .
- By using formulas, the values of the regression coefficient can be determined so as to get the regression line for the given variables.

### Examples on Regression Coefficients

- **Example 1:** Find the regression coefficients for the following data:

Age	Glucose Level
43	99
21	65
25	79
42	75
57	87
59	81

- **Solution:**

Age (x)	Glucose Level (y)	xy	$x^2$	$y^2$
43	99	4257	1849	9801
21	65	1365	441	4225

Age (x)	Glucose Level (y)	xy	x <sup>2</sup>	y <sup>2</sup>
25	79	1975	625	6241
42	75	3150	1764	5625
57	87	4959	3249	7569
59	81	4779	3481	6561
Total = 247	486	20485	11409	40022

- The formula for finding the regression coefficients are as follows:
- $a = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2} = \frac{247(20485) - (247)(486)}{247(11409) - (247)^2} = 0.39$
- $b = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2} = \frac{(486)(11409) - (247)(20485)}{247(11409) - (247)^2} = 65.14$
- The regression equation is  $Y = 0.39X + 65.14$
- **Answer:** a = 0.39 and b = 65.14
- **Example 2:** Find the regression line for the following data.

A	B
6.25	4.03
6.5	4.02
6.5	4.02
6	4.04
6.25	4.03



A	B
6.25	4.03

• **Solution:**

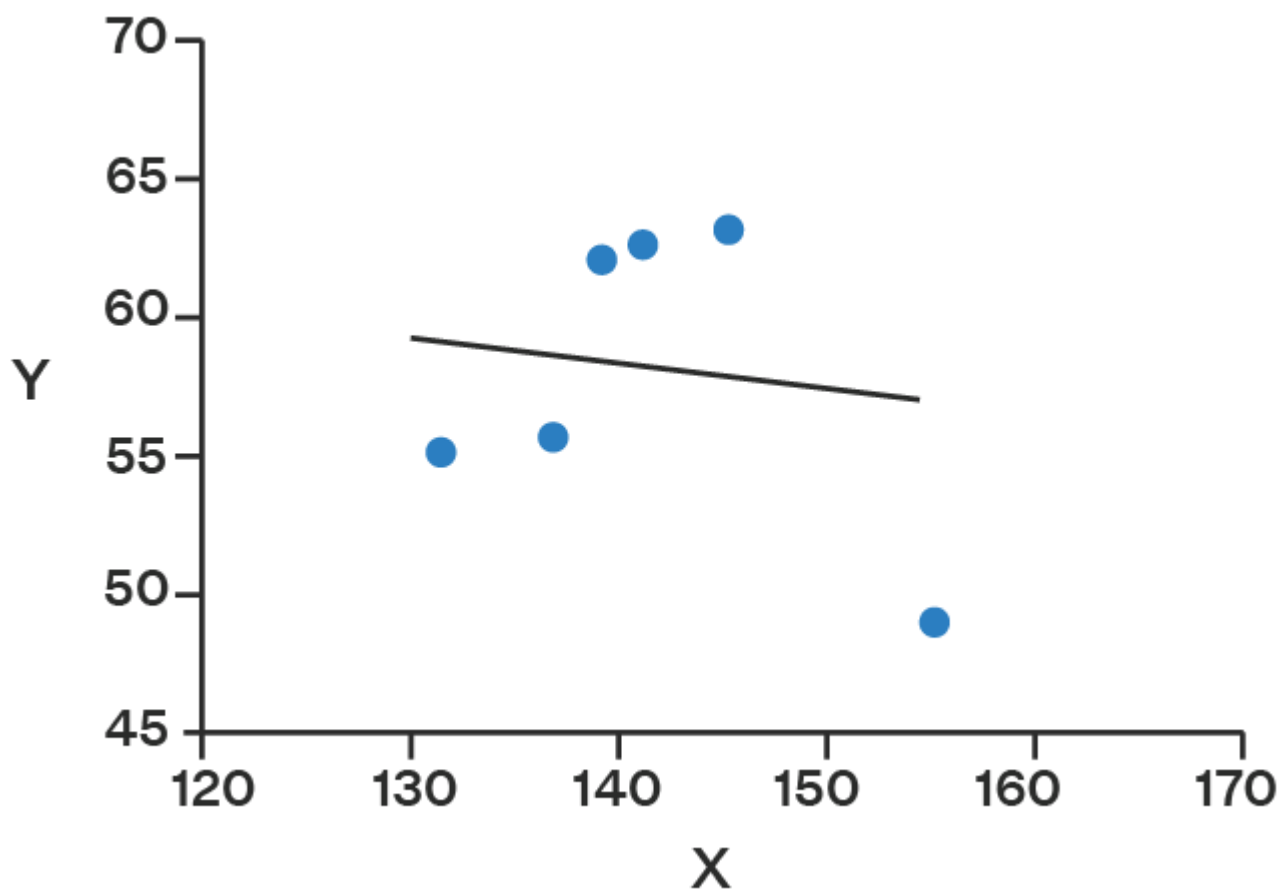
X	Y	XY	X <sup>2</sup>	Y <sup>2</sup>
6.25	4.03	25.19	39.06	16.24
6.5	4.02	26.13	42.25	16.16
6.5	4.02	26.13	42.25	16.16
6	4.04	24.24	36	16.32
6.25	4.03	25.19	39.06	16.24
6.25	4.03	25.19	39.06	16.24
Total = 37.75	24.17	152.06	237.69	97.37

- The formula for finding the regression coefficients are as follows:
- $a = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$
- $= -0.04$
- $b = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$
- $= 4.28$
- The regression equation is  $Y = -0.04X + 4.28$
- **Answer:** Regression equation is  $Y = -0.04X + 4.28$
- **Example 3:** Plot the graph for the following data if the regression coefficients are given as  $a = -0.07$  and  $b = 68.63$

X	Y
---	---

X	Y
130	55
135	56
140	62
142	63
147	63
156	51

- **Solution:** The regression coefficients are given as  $a = -0.07$  and  $b = 68.63$
- Thus, the regression line is  $Y = -0.07X + 68.63$



### least square method :

The **least square method** is the process of finding the best-fitting curve or line of best fit for a set of data points by reducing the sum of the squares of the offsets (residual part) of the points from the curve. During the process of finding the relation between two variables, the trend of outcomes are estimated quantitatively. This process is termed as **regression analysis**. The method of curve fitting is an approach to regression analysis. This method of fitting equations which approximates the curves to given raw data is the least squares.

It is quite obvious that the fitting of curves for a particular data set are not always unique. Thus, it is required to find a curve having a minimal deviation from all the measured data points. This is known as the best-fitting curve and is found by using the least-squares method.

### Least Square Method Definition:

The least-squares method is a crucial statistical method that is practised to find a regression line or a best-fit line for the given pattern. This method is described by an equation with specific parameters. The method of least squares is generously used in evaluation and regression. In regression analysis, this method is said to be a standard approach for the approximation of sets of equations having more equations than the number of unknowns.

The method of least squares actually defines the solution for the minimization of the sum of squares of deviations or the errors in the result of each equation. Find the [formula for sum of squares of errors](#), which help to find the variation in observed data.

The least-squares method is often applied in data fitting. The best fit result is assumed to reduce the sum of squared errors or residuals which are stated to be the differences between the observed or experimental value and corresponding fitted value given in the model.

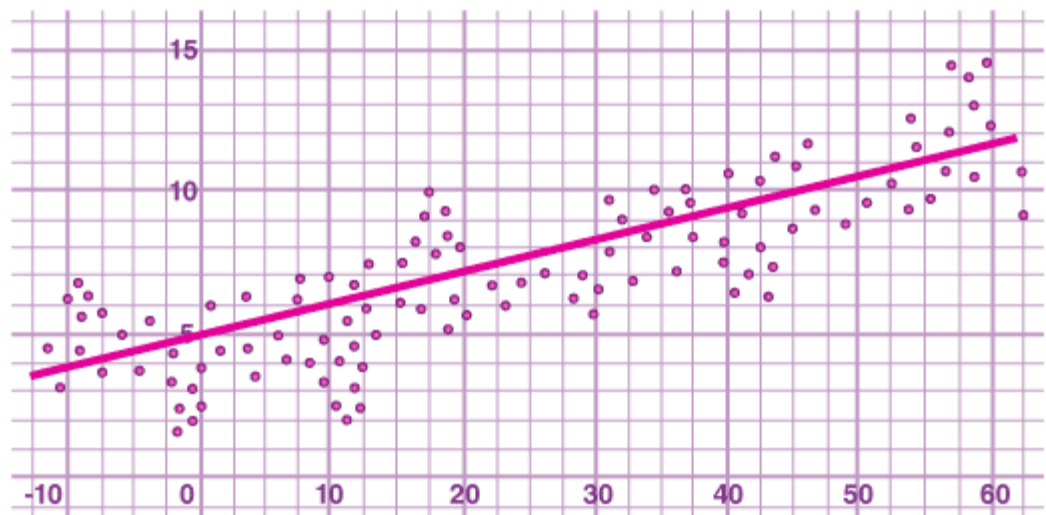
There are two basic categories of least-squares problems:

- Ordinary or linear least squares
- Nonlinear least squares

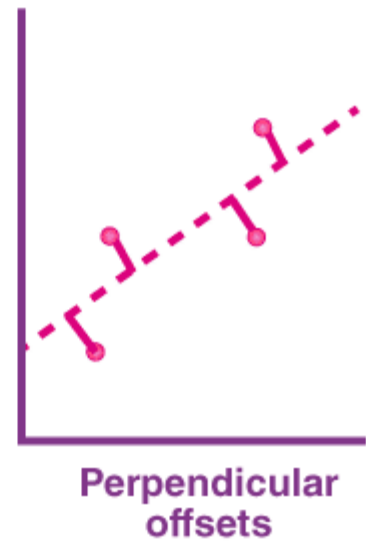
These depend upon linearity or nonlinearity of the residuals. The linear problems are often seen in regression analysis in statistics. On the other hand, the non-linear problems are generally used in the iterative method of refinement in which the model is approximated to the linear one with each iteration.

## Least Square Method Graph

In linear regression, the line of best fit is a straight line as shown in the following diagram:



The given data points are to be minimized by the method of reducing residuals or offsets of each point from the line. The vertical offsets are generally used in surface, polynomial and hyperplane problems, while perpendicular offsets are utilized in common practice.



## Least Square Method Formula

The least-square method states that the curve that best fits a given set of observations, is said to be a curve having a minimum sum of the squared residuals (or deviations or errors) from the given data points. Let us assume that the given points of data are  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$ , ...,  $(x_n, y_n)$  in which all  $x$ 's are independent variables, while all  $y$ 's are dependent ones. Also, suppose that  $f(x)$  is the fitting curve and  $d$  represents error or deviation from each given point.

Now, we can write:

$$d_1 = y_1 - f(x_1)$$

$$d_2 = y_2 - f(x_2)$$

$$d_3 = y_3 - f(x_3)$$

.....

$$d_n = y_n - f(x_n)$$

The least-squares explain that the curve that best fits is represented by the property that the sum of squares of all the deviations from given values must be minimum, i.e:

$$S = \sum_{i=1}^n d_i^2$$

$$S = \sum_{i=1}^n [y_i - f_{x_i}]^2$$

$$S = d_1^2 + d_2^2 + d_3^2 + \dots + d_n^2$$

Sum = Minimum Quantity

Suppose when we have to determine the equation of line of best fit for the given data, then we first use the following formula.

The equation of least square line is given by  $Y = a + bX$

Normal equation for 'a':

$$\sum Y = na + b\sum X$$

Normal equation for 'b':

$$\sum XY = a\sum X + b\sum X^2$$

Solving these two normal equations we can get the required trend line equation.

Thus, we can get the line of best fit with formula  $y = ax + b$

## Solved Example

The Least Squares Model for a set of data  $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)$  passes through the point  $(x_a, y_a)$  where  $x_a$  is the average of the  $x_i$ 's and  $y_a$  is the average of the  $y_i$ 's. The below example explains how to find the equation of a straight line or a least square line using the least square method.

### Question:

Consider the time series data given below:

$x_i$	8	3	2	10	11	3	6	5	6	8
$y_i$	4	12	1	12	9	4	9	6	1	14

Use the least square method to determine the equation of line of best fit for the data. Then plot the line.

### Solution:

Mean of  $x_i$  values =  $(8 + 3 + 2 + 10 + 11 + 3 + 6 + 5 + 6 + 8)/10 = 62/10 = 6.2$

Mean of  $y_i$  values =  $(4 + 12 + 1 + 12 + 9 + 4 + 9 + 6 + 1 + 14)/10 = 72/10 = 7.2$

Straight line equation is  $y = a + bx$ .

The normal equations are

$$\sum y = an + b\sum x$$

$$\sum xy = a\sum x + b\sum x^2$$

x	y	$x^2$	xy
8	4	64	32
3	12	9	36

2	1	4	2
10	12	100	120
11	9	121	99
3	4	9	12
6	9	36	54
5	6	25	30
6	1	36	6
8	14	64	112
$\sum x = 62$	$\sum y = 72$	$\sum x^2 = 468$	$\sum xy = 503$

Substituting these values in the normal equations,

$$10a + 62b = 72 \dots (1)$$

$$62a + 468b = 503 \dots (2)$$

$$(1) \times 62 - (2) \times 10,$$

$$620a + 3844b - (620a + 4680b) = 4464 - 5030$$

$$-836b = -566$$

$$b = 566/836$$

$$b = 283/418$$

$$b = 0.677$$

Substituting  $b = 0.677$  in equation (1),

$$10a + 62(0.677) = 72$$

$$10a + 41.974 = 72$$

$$10a = 72 - 41.974$$

$$10a = 30.026$$

$$a = 30.026/10$$

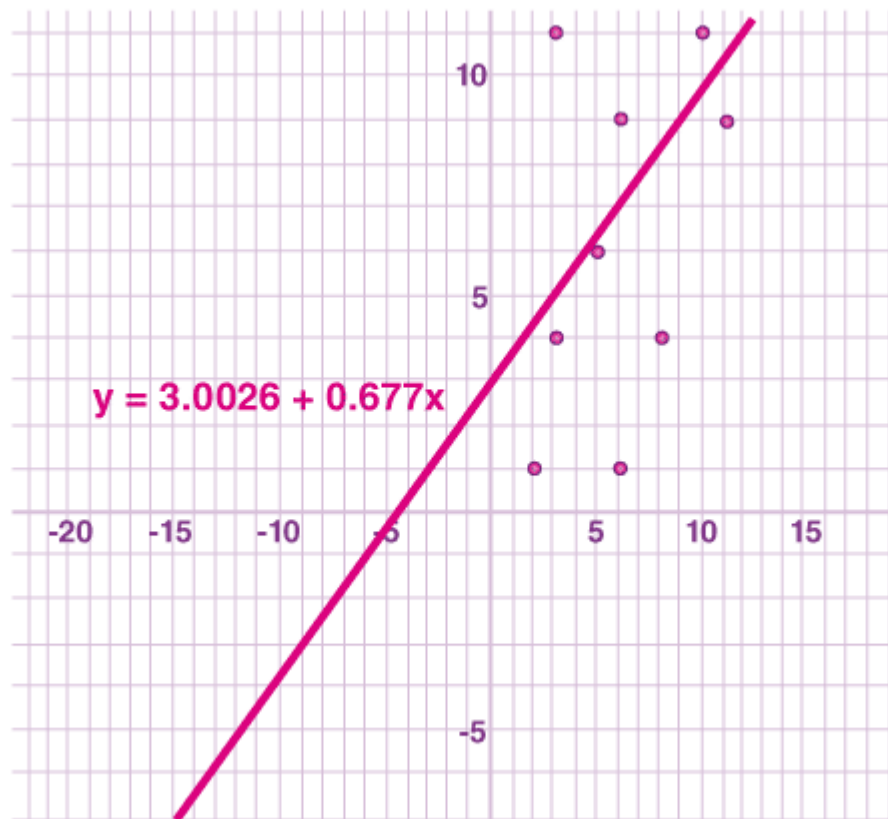
$$a = 3.0026$$

Therefore, the equation becomes,



$$y = a + bx$$

$$y = 3.0026 + 0.677x$$



This is the required trend line equation.

Now, we can find the sum of squares of deviations from the obtained values as:

$$d_1 = [4 - (3.0026 + 0.677 \cdot 8)] = (-4.4186)$$

$$d_2 = [12 - (3.0026 + 0.677 \cdot 3)] = (6.9664)$$

$$d_3 = [1 - (3.0026 + 0.677 \cdot 2)] = (-3.3566)$$

$$d_4 = [12 - (3.0026 + 0.677 \cdot 10)] = (2.2274)$$

$$d_5 = [9 - (3.0026 + 0.677 \cdot 11)] = (-1.4496)$$

$$d_6 = [4 - (3.0026 + 0.677 \cdot 3)] = (-1.0336)$$

$$d_7 = [9 - (3.0026 + 0.677 \cdot 6)] = (1.9354)$$

$$d_8 = [6 - (3.0026 + 0.677 \cdot 5)] = (-0.3876)$$

$$d_9 = [1 - (3.0026 + 0.677 \cdot 6)] = (-6.0646)$$

$$d_{10} = [14 - (3.0026 + 0.677 \cdot 8)] = (5.5814)$$

$$\sum d^2 = (-4.4186)^2 + (6.9664)^2 + (-3.3566)^2 + (2.2274)^2 + (-1.4496)^2 + (-1.0336)^2 + (1.9354)^2 + (-0.3876)^2 + (-6.0646)^2 + (5.5814)^2 = 159.27990$$

## Limitations for Least-Square Method

The least-squares method is a very beneficial method of curve fitting. Despite many benefits, it has a few shortcomings too. One of the main limitations is discussed here.

In the process of regression analysis, which utilizes the least-square method for curve fitting, it is inevitably assumed that the errors in the independent variable are negligible or zero. In such cases, when independent variable errors are non-negligible, the models are subjected to measurement errors. Therefore, here, the least square method may even lead to hypothesis testing, where parameter estimates and confidence intervals are taken into consideration due to the presence of errors occurring in the independent variables.

## Types of Regressions:

Linear regression and logistic regression are two types of regression analysis techniques that are used to solve the regression problem using machine learning. They are the most prominent techniques of regression. But, there are many types of regression analysis techniques in machine learning, and their usage varies according to the nature of the data involved.

This article will explain the different types of regression in machine learning, and under what condition each of them can be used. If you are new to machine learning, this article will surely help you in understanding the regression modeling concept.

## What is Regression Analysis?

Regression analysis is a predictive modelling technique that analyzes the relation between the target or dependent variable and independent variable in a dataset. The different types of regression analysis techniques get used when the target and independent variables show a linear or non-linear relationship between each other, and the target variable contains continuous values. The regression technique gets used mainly to determine the predictor strength, forecast trend, time series, and in case of cause & effect relation.

Regression analysis is the primary technique to solve the regression problems in machine learning using data modelling. It involves determining the best fit line, which is a line that passes through all the data points in such a way that distance of the line from each data point is minimized.

*Learn [AI & ML Courses](#) online from the World's top Universities – Masters, Executive Post Graduate Programs, and Advanced Certificate Program in ML & AI to fast-track your career.*

## Types of Regression Analysis Techniques

There are many types of regression analysis techniques, and the use of each method depends upon the number of factors. These factors include the type of target variable, shape of the regression line, and the number of independent variables.

**Below are the different regression techniques:**

1. Linear Regression
2. Logistic Regression
3. Ridge Regression
4. Lasso Regression
5. Polynomial Regression
6. Bayesian Linear Regression

**The different types of regression in machine learning techniques are explained below in detail:**

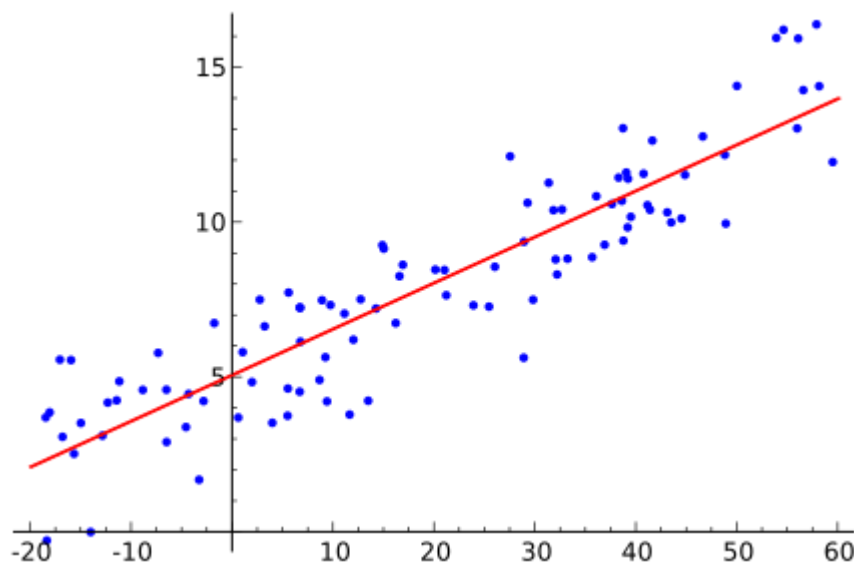
### **1. Linear Regression**

Linear regression is one of the most basic types of regression in machine learning. The linear regression model consists of a predictor variable and a dependent variable related linearly to each other. In case the data involves more than one independent variable, then linear regression is called multiple linear regression models.

*The below-given equation is used to denote the linear regression model:*

$$y=mx+c+e$$

where  $m$  is the slope of the line,  $c$  is an intercept, and  $e$  represents the error in the model.



#### Source

The best fit line is determined by varying the values of  $m$  and  $c$ . The predictor error is the difference between the observed values and the predicted value. The values of  $m$  and  $c$  get selected in such a way that it gives the minimum predictor error. It is important to note that a simple linear regression model is susceptible to outliers. Therefore, it should not be used in case of big size data.

There are different types of linear regression. The two major types of linear regression are simple linear regression and multiple linear regression. Below is the formula for simple linear regression.

- Here,  $\hat{y}$  is the predicted value of the dependent variable ( $y$ ) for any value of the independent variable ( $x$ )
- $\beta_0$  is the intercepted, aka the value of  $y$  when  $x$  is zero
- $\beta_1$  is the regression coefficient, meaning the expected change in  $y$  when  $x$  increases
- $x$  is the independent variable
- $\epsilon$  is the estimated error in the regression

Simple linear regression can be used:

- To find the intensity of dependency between two variables. Such as the rate of carbon emission and global warming.
  - To find the value of the dependent variable on an explicit value of the independent variable. For example, finding the amount of increase in atmospheric temperature with a certain amount of carbon dioxide emission.
- In multiple linear regression, a relationship is established between two or more independent variables and the corresponding dependent variables. Below is the equation for multiple linear regression.

- Here,  $\hat{y}$  is the predicted value of the dependent variable
- $\beta_0$  = Value of  $y$  when other parameters are zero
- $\beta_1 X_1$  = The regression coefficient of the first variable
- ... = Repeating the same no matter how many variables you test
- $\beta_n X_n$  = Regression coefficient of the last independent variable
- $\epsilon$  = Estimated error in the regression

Multiple linear regression can be used:

- To estimate how strongly two or more independent variables influence the single dependent variable. Such as how location, time, condition, and area can influence the price of a property.
- To find the value of the dependent variables at a definite condition of all the independent variables. For example, finding the price of a property located at a certain place, with a specific area and its condition.

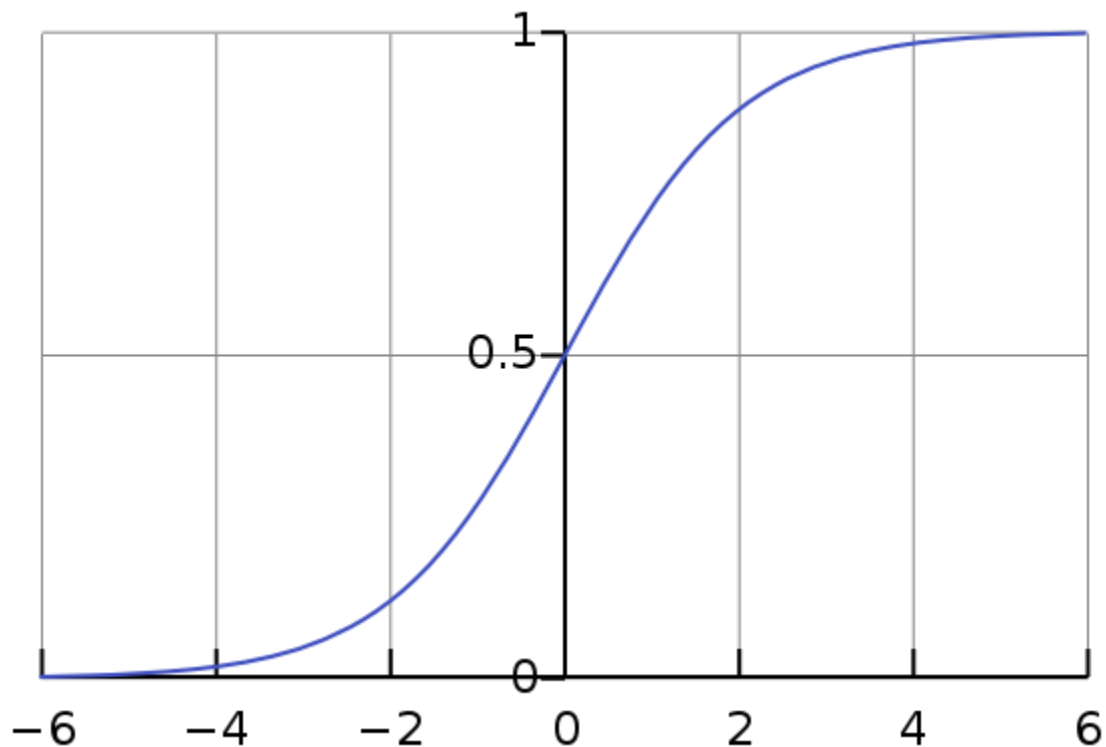
Also visit [upGrad's Degree Counselling](#) page for all undergraduate and postgraduate programs.

## 2. Logistic Regression

Logistic regression is one of the types of regression analysis technique, which gets used when the dependent variable is discrete. Example: 0 or 1, true or false, etc. This means the target variable can have only two values, and a sigmoid curve denotes the relation between the target variable and the independent variable.

Logit function is used in Logistic Regression to measure the relationship between the target variable and independent variables. Below is the equation that denotes the logistic regression.

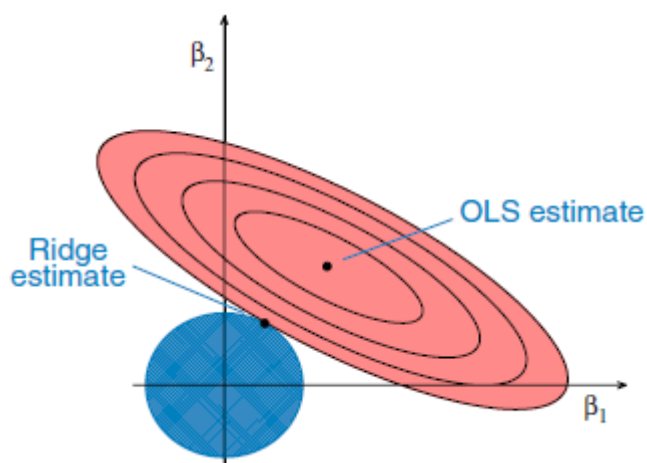
$\text{logit}(p) = \ln(p/(1-p)) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \dots + b_kX_k$   
where  $p$  is the probability of occurrence of the feature.



Source

For selecting logistic regression, as the regression analyst technique, it should be noted, the size of data is large with the almost equal occurrence of values to come in target variables. Also, there should be no multicollinearity, which means that there should be no correlation between independent variables in the dataset.

### 3. Ridge Regression



Source

This is another one of the types of regression in machine learning which is usually used when there is a high correlation between the independent variables.

This is because, in the case of multi collinear data, the least square estimates give unbiased values. But, in case the collinearity is very high, there can be some bias value. Therefore, a bias matrix is introduced in the equation of Ridge Regression. This is a powerful regression method where the model is less susceptible to overfitting.

*Below is the equation used to denote the Ridge Regression, where the introduction of  $\lambda$  (lambda) solves the problem of multicollinearity:*

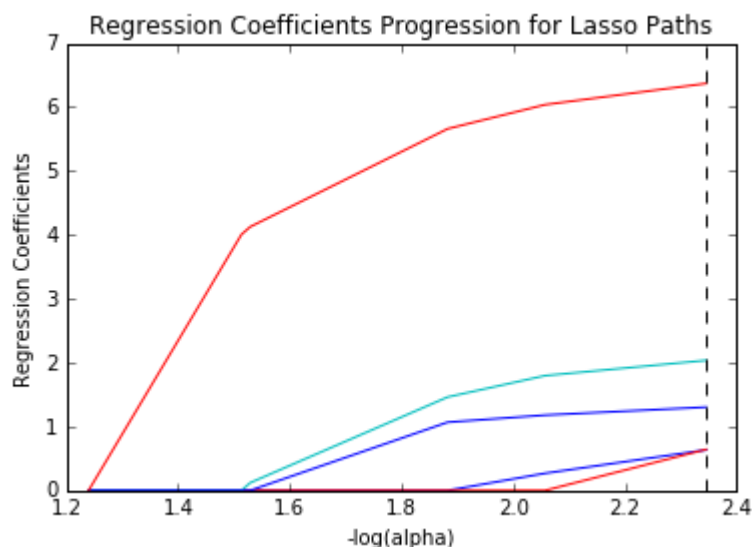
$$\beta = (X^T X + \lambda I)^{-1} X^T y$$

**Check out:** [5 Breakthrough Applications of Machine Learning](#)

#### 4. Lasso Regression

Lasso Regression is one of the types of regression in machine learning that performs regularization along with feature selection. It prohibits the absolute size of the regression coefficient. As a result, the coefficient value gets nearer to zero, which does not happen in the case of Ridge Regression.

Due to this, feature selection gets used in Lasso Regression, which allows selecting a set of features from the dataset to build the model. In the case of Lasso Regression, only the required features are used, and the other ones are made zero. This helps in avoiding the overfitting in the model. In case the independent variables are highly collinear, then Lasso regression picks only one variable and makes other variables to shrink to zero.



Source

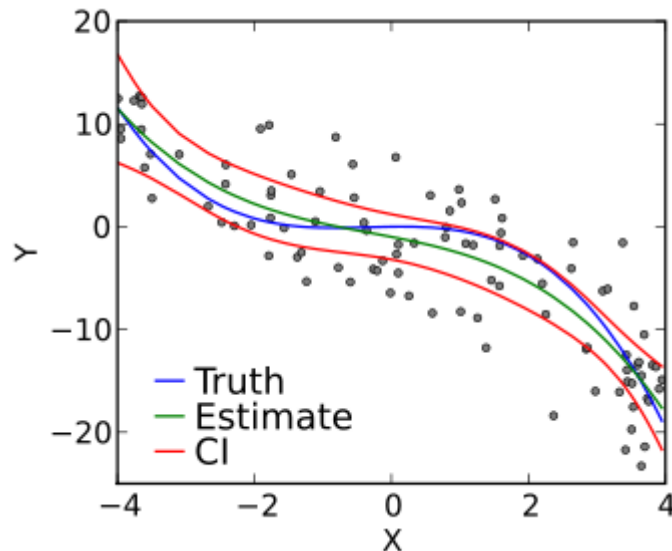
*Below is the equation that represents the Lasso Regression method:*

$$N^{-1} \sum_{i=1}^N f(x_i, y_i, \alpha, \beta)$$

#### 5. Polynomial Regression

Polynomial Regression is another one of the types of regression analysis techniques in machine learning, which is the same as Multiple Linear Regression with a little modification. In Polynomial Regression, the relationship between independent and dependent variables, that is X and Y, is denoted by the n-th degree.

It is a linear model as an estimator. Least Mean Squared Method is used in Polynomial Regression also. The best fit line in Polynomial Regression that passes through all the data points is not a straight line, but a curved line, which depends upon the power of X or value of n.



Source

While trying to reduce the Mean Squared Error to a minimum and to get the best fit line, the model can be prone to overfitting. It is recommended to analyze the curve towards the end as the higher Polynomials can give strange results on extrapolation.

*Below equation represents the Polynomial Regression:*

$$l = \beta_0 + \beta_0 x_1 + \epsilon$$

Read: Machine Learning Project Ideas

## 6. Bayesian Linear Regression

Bayesian Regression is one of the types of regression in machine learning that uses the Bayes theorem to find out the value of regression coefficients. In this method of regression, the posterior distribution of the features is determined instead of finding the least-squares. Bayesian Linear Regression is like both Linear Regression and Ridge Regression but is more stable than the simple Linear Regression.

## Simple Linear Regression in Machine Learning:

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the ***dependent variable must be a continuous/real value***. However, the independent variable can be measured on continuous or categorical values.



Simple Linear regression algorithm has mainly two objectives:

- **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.
- **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

Simple Linear Regression Model:

The Simple Linear Regression model can be represented using the below equation:

$$y = a_0 + a_1x + \epsilon$$

Machine Learning - Data Description - Measures of Central Tendency: Mean, Median and Mode

$$y = a_0 + a_1x + \epsilon$$

Where,

**$a_0$** = It is the intercept of the Regression line (can be obtained putting  $x=0$ )

**$a_1$** = It is the slope of the regression line, which tells whether the line is increasing or decreasing.

**$\epsilon$**  = The error term. (For a good model it will be negligible)

Implementation of Simple Linear Regression Algorithm using Python

**Problem Statement example for Simple Linear Regression:**

Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable). The goals of this problem is:

- **We want to find out if there is any correlation between these two variables**
- **We will find the best fit line for the dataset.**
- **How the dependent variable is changing by changing the independent variable.**

In this section, we will create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.

To implement the Simple Linear regression model in machine learning using Python, we need to follow the below steps:

**Step-1: Data Pre-processing**

The first step for creating the Simple Linear Regression model is data pre-processing

. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

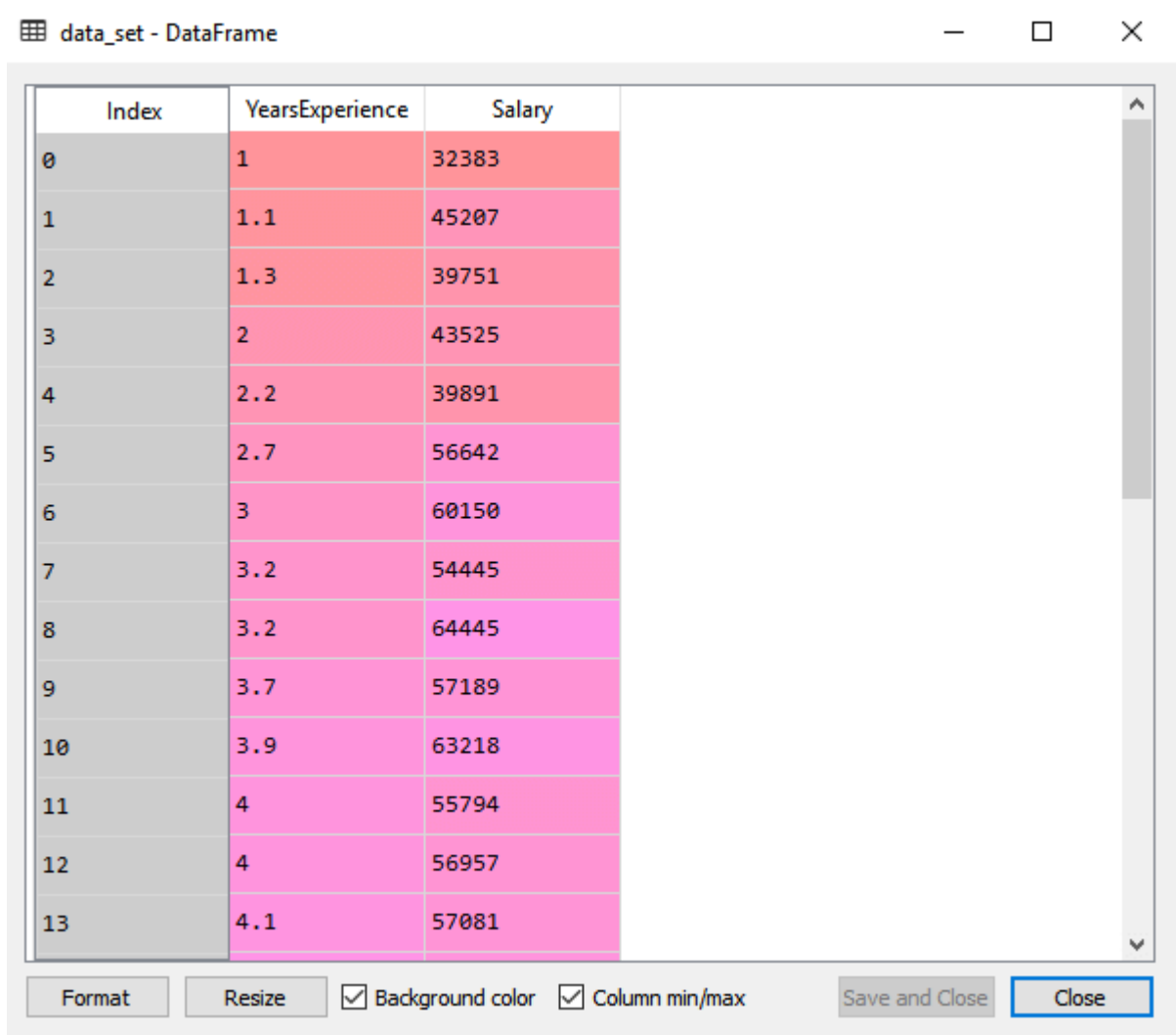
- First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

1. **import** numpy as nm
2. **import** matplotlib.pyplot as mtp
3. **import** pandas as pd

- Next, we will load the dataset into our code:

1. `data_set= pd.read_csv('Salary_Data.csv')`

By executing the above line of code (ctrl+ENTER), we can read the dataset on our Spyder IDE screen by clicking on the variable explorer option.



Index	YearsExperience	Salary
0	1	32383
1	1.1	45207
2	1.3	39751
3	2	43525
4	2.2	39891
5	2.7	56642
6	3	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4	55794
12	4	56957
13	4.1	57081

The above output shows the dataset, which has two variables: Salary and Experience.

**Note:** In Spyder IDE, the folder containing the code file must be saved as a working directory, and the dataset or csv file should be in the same folder.

- After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

1. `x= data_set.iloc[:, :-1].values`
2. `y= data_set.iloc[:, 1].values`

In the above lines of code, for x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

By executing the above line of code, we will get the output for X and Y variable as:

The image shows two side-by-side NumPy array viewer windows. The left window, titled 'x - NumPy array', displays a 1D array of 12 values: 0, 1, 1.1, 1.3, 2, 2.2, 2.7, 3, 3.2, 3.2, 3.7, 3.9, 4. The right window, titled 'y - NumPy array', displays a 1D array of 12 values: 32383, 45207, 39751, 43525, 39891, 56642, 60150, 54445, 64445, 57189, 63218, 55794, 56957. Both windows have a 'Format' button, a 'Resize' button, a checked 'Background color' checkbox, and 'Save and Close' and 'Close' buttons at the bottom.

	0
0	1
1	1.1
2	1.3
3	2
4	2.2
5	2.7
6	3
7	3.2
8	3.2
9	3.7
10	3.9
11	4
12	4

	0
0	32383
1	45207
2	39751
3	43525
4	39891
5	56642
6	60150
7	54445
8	64445
9	57189
10	63218
11	55794
12	56957

In the above output image, we can see the X (independent) variable and Y (dependent) variable has been extracted from the given dataset.

- Next, we will split both variables into the test set and training set. We have 30 observations, so we will take 20 observations for the training set and 10 observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

1. # Splitting the dataset into training and test set.
2. from sklearn.model\_selection **import** train\_test\_split
3. x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 1/3, random\_state=0)

By executing the above code, we will get x-test, x-train and y-test, y-train dataset. Consider the below images:

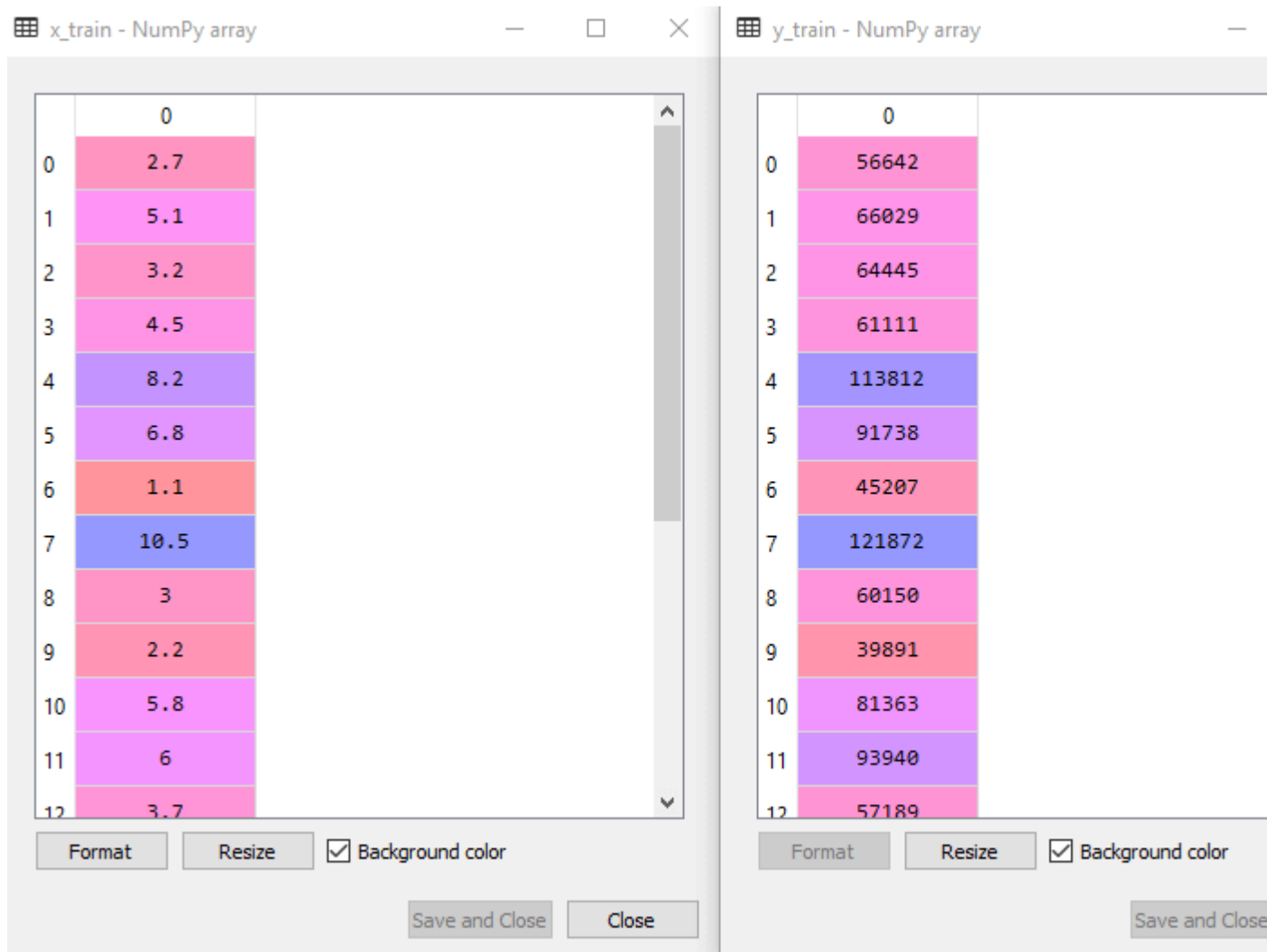
### Test-dataset:

	0
0	1.3
1	10.3
2	4.1
3	3.9
4	9.5
5	8.7
6	9.6
7	4
8	5.3
9	7.9

	0
0	39751
1	122391
2	57081
3	63218
4	116969
5	109431
6	112635
7	55794
8	83088
9	101302

### Training Dataset:



- For simple linear Regression, we will not use Feature Scaling. Because Python libraries take care of it for some cases, so we don't need to perform it here. Now, our dataset is well prepared to work on it and we are going to start building a Simple Linear Regression model for the given problem.

## Step-2: Fitting the Simple Linear Regression to the Training Set:

Now the second step is to fit our model to the training dataset. To do so, we will import the **LinearRegression** class of the **linear\_model** library from the **scikit learn**. After importing the class, we are going to create an object of the class named as a **regressor**. The code for this is given below:

1. #Fitting the Simple Linear Regression model to the training dataset
2. from sklearn.linear\_model **import** LinearRegression
3. regressor= LinearRegression()
4. regressor.fit(x\_train, y\_train)

In the above code, we have used a **fit()** method to fit our Simple Linear Regression object to the training set. In the fit() function, we have passed the x\_train and y\_train, which is our training dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

### **Output:**

```
Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

### **Step: 3. Prediction of test set result:**

dependent (salary) and an independent variable (Experience). So, now, our model is ready to predict the output for the new observations. In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector **y\_pred**, and **x\_pred**, which will contain predictions of test dataset, and prediction of training set respectively.

1. #Prediction of Test and Training set result
2. y\_pred= regressor.predict(x\_test)
3. x\_pred= regressor.predict(x\_train)

On executing the above lines of code, two variables named y\_pred and x\_pred will generate in the variable explorer options that contain salary predictions for the training set and test set.

### **Output:**

You can check the variable by clicking on the variable explorer option in the IDE, and also compare the result by comparing values from y\_pred and y\_test. By comparing these values, we can check how good our model is performing.

### **Step: 4. visualizing the Training set results:**

Now in this step, we will visualize the training set result. To do so, we will use the `scatter()` function of the `pyplot` library, which we have already imported in the pre-processing step. The **`scatter()` function** will create a scatter plot of observations.

In the x-axis, we will plot the Years of Experience of employees and on the y-axis, salary of employees. In the function, we will pass the real values of training set, which means a year of experience `x_train`, training set of Salaries `y_train`, and color of the observations. Here we are taking a green color for the observation, but it can be any color as per the choice.

Now, we need to plot the regression line, so for this, we will use the **`plot()` function** of the `pyplot` library. In this function, we will pass the years of experience for training set, predicted salary for training set `x_pred`, and color of the line.

Next, we will give the title for the plot. So here, we will use the **`title()` function** of the **`pyplot`** library and pass the name ("Salary vs Experience (Training Dataset)").

After that, we will assign labels for x-axis and y-axis using **`xlabel()` and `ylabel()` function**.

Finally, we will represent all above things in a graph using `show()`. The code is given below:

1. `mtp.scatter(x_train, y_train, color="green")`
2. `mtp.plot(x_train, x_pred, color="red")`
3. `mtp.title("Salary vs Experience (Training Dataset)")`
4. `mtp.xlabel("Years of Experience")`
5. `mtp.ylabel("Salary(In Rupees)")`
6. `mtp.show()`

### **Output:**

By executing the above lines of code, we will get the below graph plot as an output.





In the above plot, we can see the real values observations in green dots and predicted values are covered by the red regression line. The regression line shows a correlation between the dependent and independent variable.

The good fit of the line can be observed by calculating the difference between actual values and predicted values. But as we can see in the above **plot, most of the observations are close to the regression line, hence our model is good for the training set.**

#### **Step: 5. visualizing the Test set results:**

In the previous step, we have visualized the performance of our model on the training set. Now, we will do the same for the Test set. The complete code will remain the same as the above code, except in this, we will use `x_test`, and `y_test` instead of `x_train` and `y_train`.

Here we are also changing the color of observations and regression line to differentiate between the two plots, but it is optional.

1. `#visualizing the Test set results`
2. `mtp.scatter(x_test, y_test, color="blue")`
3. `mtp.plot(x_train, x_pred, color="red")`

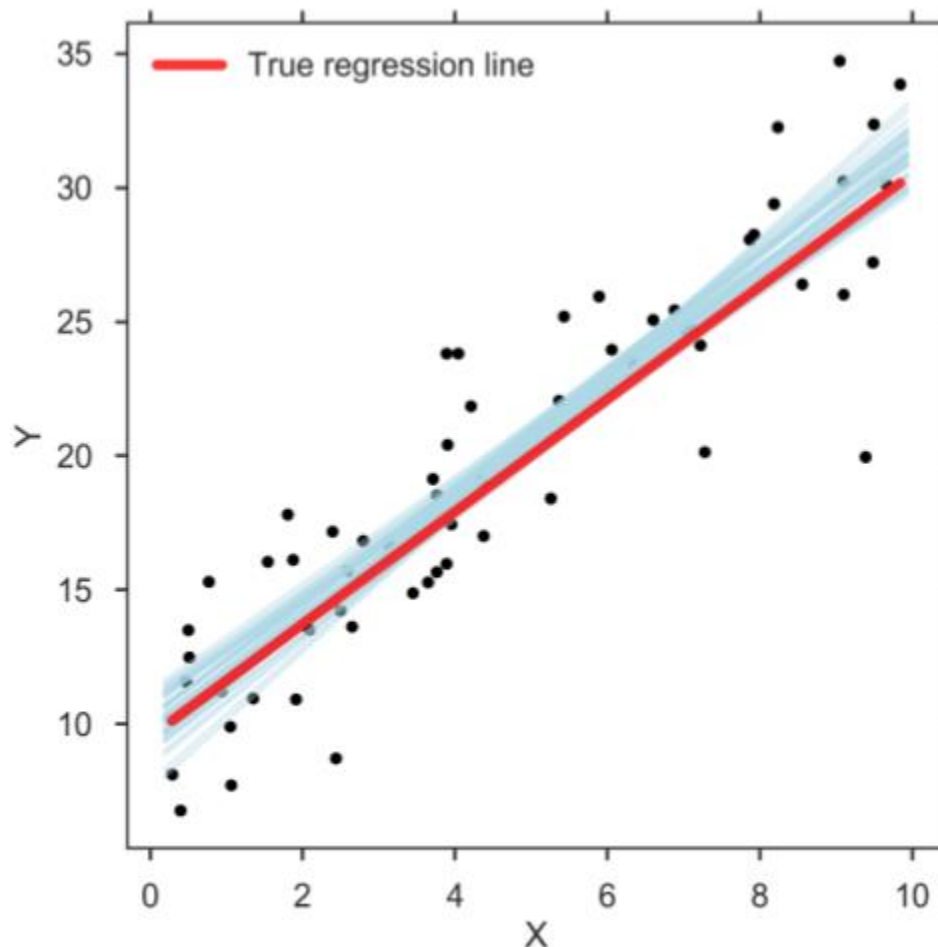
4. `mtp.title("Salary vs Experience (Test Dataset)")`
5. `mtp.xlabel("Years of Experience")`
6. `mtp.ylabel("Salary(In Rupees)")`
7. `mtp.show()`

### Output:

By executing the above line of code, we will get the output as:



In the above plot, there are observations given by the blue color, and prediction is given by the red regression line. As we can see, most of the observations are close to the regression line, hence we can say our Simple Linear Regression is a good model and able to make good predictions.



#### Source

People often wonder “what is regression in AI” or “what is regression in machine learning”. Machine learning is a subset of AI; hence, both questions have the same answer.

In the case of regression in AI, different algorithms are used to make a machine learn the relationship between the provided data sets and make predictions accordingly.

Hence, regression in AI is mainly used to add a level of automation to the machines.

Regression AI is often used in sectors like finance and investment, where establishing a relationship between a single dependent variable and multiple independent variables is a common case. A common example of regression AI will be factors that estimate a house’s price based on its location, size, ROI, etc.

Regression plays a vital role in predictive modelling and is found in many machine learning applications. Algorithms from the regressions provide different perspectives regarding the relationship between the variables and their outcomes. These set models could then be used as a guideline for fresh input data or to find missing data.

As the models are trained to understand a variety of relationships between different variables, they are often extremely helpful in predicting the portfolio performance or stocks and trends.

These implementations fall under machine learning in finance.

The very common use of regression in AI includes:

- Predicting a company’s sales or marketing success
- Generating continuous outcomes like stock prices
- Forecasting different trends or customer’s purchase behaviour

Hope this helped to understand what is regression in AI or what is regression in machine learning.

### Polynomial Regression

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

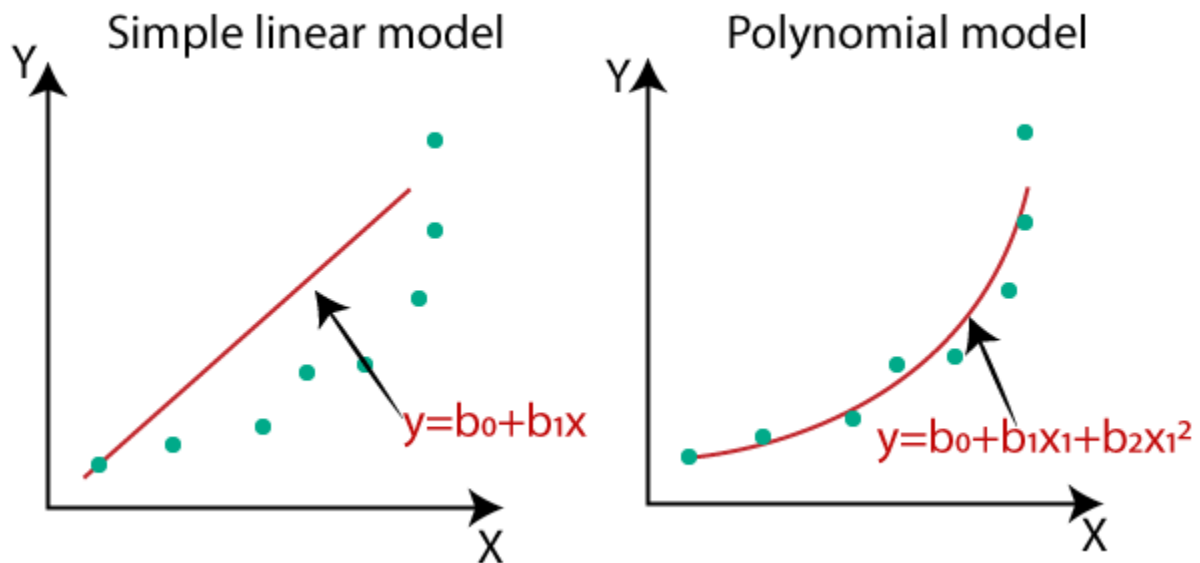
$$y = b_0 + b_1x_1 + b_2x_1^2 + b_3x_1^3 + \dots + b_nx_1^n$$

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.
- **Hence, "In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,...,n) and then modeled using a linear model."**

Need for Polynomial Regression:

The need of Polynomial Regression in ML can be understood in the below points:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.*

**Note:** A Polynomial Regression algorithm is also called Polynomial Linear Regression because it does not depend on the variables, instead, it depends on the coefficients, which are arranged in a linear fashion.

Equation of the Polynomial Regression Model:

**Simple Linear Regression equation:**  $y = b_0 + b_1x$  .....(a)

**Multiple Linear Regression equation:**  $y = b_0 + b_1x + b_2x_2 + b_3x_3 + \dots + b_nx_n$  .....(b)

**Polynomial Regression equation:**  $y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots + b_nx^n$  .....(c)

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

***Note:** To better understand Polynomial Regression, you must have knowledge of Simple Linear Regression.*

Implementation of Polynomial Regression using Python:

Here we will implement the Polynomial Regression using Python. We will understand it by comparing Polynomial Regression model with the Simple Linear Regression model. So first, let's understand the problem for which we are going to build the model.

**Problem Description:** There is a Human Resource company, which is going to hire a new candidate. The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff. So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels. By checking the dataset available, we have found that there is a **non-linear relationship between the Position levels and the salaries**. Our goal is to build a **Bluffing detector regression** model, so HR can hire an honest candidate. Below are the steps to build such a model.

Position	Level(X-variable)	Salary(Y-Variable)
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Steps for Polynomial Regression:

The main steps involved in Polynomial Regression are given below:

- Data Pre-processing
- Build a Linear Regression model and fit it to the dataset
- Build a Polynomial Regression model and fit it to the dataset
- Visualize the result for Linear Regression and Polynomial Regression model.
- Predicting the output.

**Note:** Here, we will build the Linear regression model as well as Polynomial Regression to see the results between the predictions. And Linear regression model is for reference.

### Data Pre-processing Step:

The data pre-processing step will remain the same as in previous regression models, except for some changes. In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons:

- The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.
- In this model, we want very accurate predictions for salary, so the model should have enough information.

The code for pre-processing step is given below:

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('Position_Salaries.csv')
8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, 1:2].values
11. y= data_set.iloc[:, 2].values
```

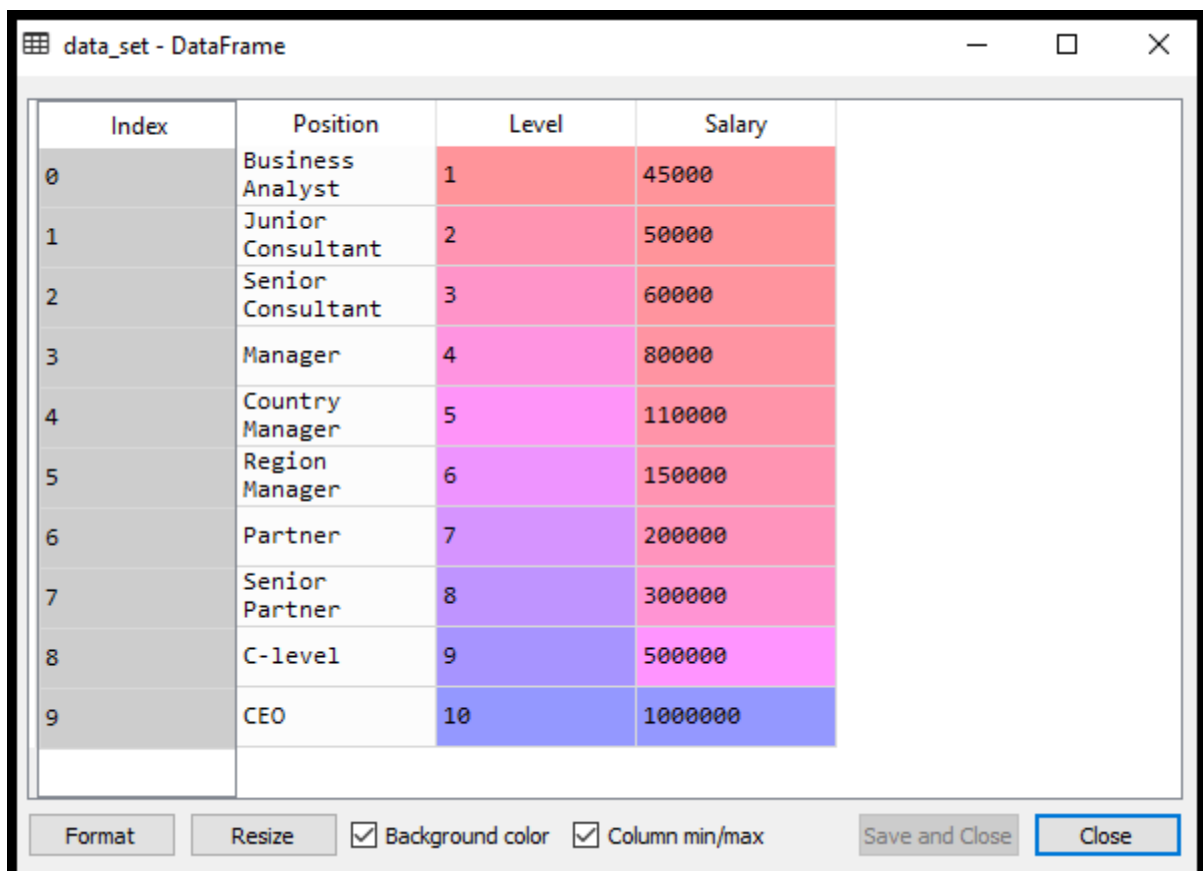
### Explanation:

- In the above lines of code, we have imported the important Python libraries to import dataset and operate on it.

- Next, we have imported the dataset '**Position\_Salaries.csv**', which contains three columns (Position, Levels, and Salary), but we will consider only two columns (Salary and Levels).
- After that, we have extracted the dependent(Y) and independent variable(X) from the dataset. For x-variable, we have taken parameters as `[:,1:2]`, because we want 1 index(levels), and included `:2` to make it as a matrix.

### Output:

By executing the above code, we can read our dataset as:



Index	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

As we can see in the above output, there are three columns present (Positions, Levels, and Salaries). But we are only considering two columns because Positions are equivalent to the levels or may be seen as the encoded form of Positions.

Here we will predict the output for level **6.5** because the candidate has 4+ years' experience as a regional manager, so he must be somewhere between levels 7 and 6.

### Building the Linear regression model:

Now, we will build and fit the Linear regression model to the dataset. In building polynomial regression, we will take the Linear regression model as reference and compare both the results. The code is given below:



1. #Fitting the Linear Regression to the dataset
2. from sklearn.linear\_model **import** LinearRegression
3. lin\_regs= LinearRegression()
4. lin\_regs.fit(x,y)

In the above code, we have created the Simple Linear model using **lin\_regs** object of **LinearRegression** class and fitted it to the dataset variables (x and y).

### Output:

```
Out[5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

### Building the Polynomial regression model:

Now we will build the Polynomial Regression model, but it will be a little different from the Simple Linear model. Because here we will use **PolynomialFeatures** class of **preprocessing** library. We are using this class to add some extra features to our dataset.

1. #Fitting the Polynomial regression to the dataset
2. from sklearn.preprocessing **import** PolynomialFeatures
3. poly\_regs= PolynomialFeatures(degree= 2)
4. x\_poly= poly\_regs.fit\_transform(x)
5. lin\_reg\_2 =LinearRegression()
6. lin\_reg\_2.fit(x\_poly, y)

In the above lines of code, we have used **poly\_regs.fit\_transform(x)**, because first we are converting our feature matrix into polynomial feature matrix, and then fitting it to the Polynomial regression model. The parameter value(degree= 2) depends on our choice. We can choose it according to our Polynomial features.

After executing the code, we will get another matrix **x\_poly**, which can be seen under the variable explorer option:

	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25
5	1	6	36
6	1	7	49
7	1	8	64
8	1	9	81
9	1	10	100

Next, we have used another `LinearRegression` object, namely **lin\_reg\_2**, to fit our **x\_poly** vector to the linear model.

### Output:

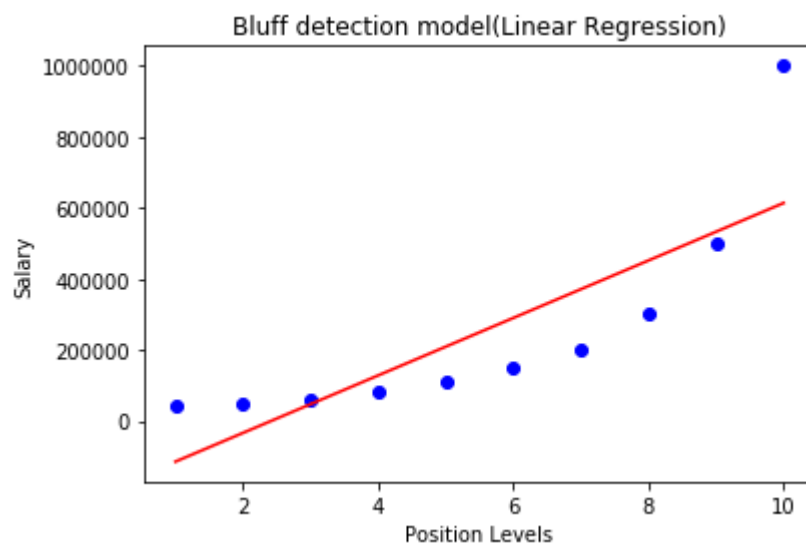
```
Out[11]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

### Visualizing the result for Linear regression:

Now we will visualize the result for Linear regression model as we did in Simple Linear Regression. Below is the code for it:

1. #Visulaizing the result **for** Linear Regression model
2. `mtp.scatter(x,y,color="blue")`
3. `mtp.plot(x,lin_regs.predict(x), color="red")`
4. `mtp.title("Bluff detection model(Linear Regression)")`
5. `mtp.xlabel("Position Levels")`
6. `mtp.ylabel("Salary")`
7. `mtp.show()`

### Output:



In the above output image, we can clearly see that the regression line is so far from the datasets. Predictions are in a red straight line, and blue points are actual values. If we consider this output to predict the value of CEO, it will give a salary of approx. 600000\$, which is far away from the real value.

So we need a curved model to fit the dataset other than a straight line.

### Visualizing the result for Polynomial Regression

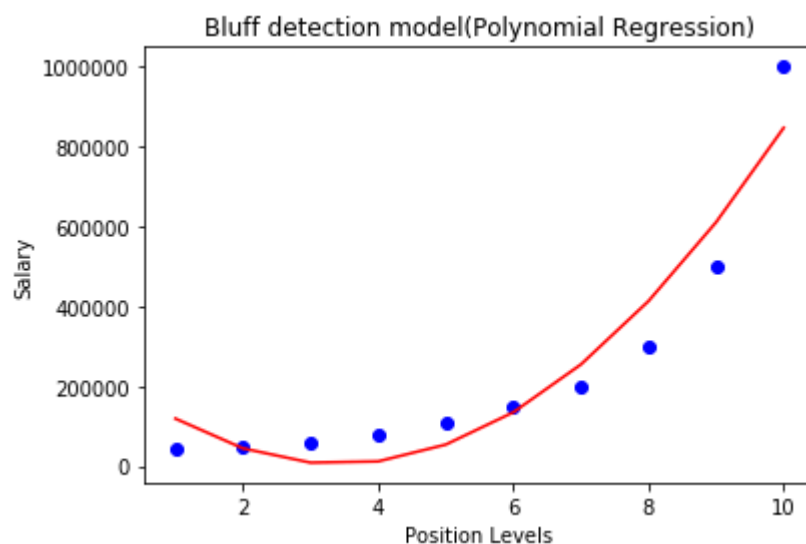
Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.

Code for this is given below:

1. #Visulaizing the result **for** Polynomial Regression
2. `mtp.scatter(x,y,color="blue")`
3. `mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")`
4. `mtp.title("Bluff detection model(Polynomial Regression)")`
5. `mtp.xlabel("Position Levels")`
6. `mtp.ylabel("Salary")`
7. `mtp.show()`

In the above code, we have taken `lin_reg_2.predict(poly_regs.fit_transform(x))`, instead of `x_poly`, because we want a Linear regressor object to predict the polynomial features matrix.

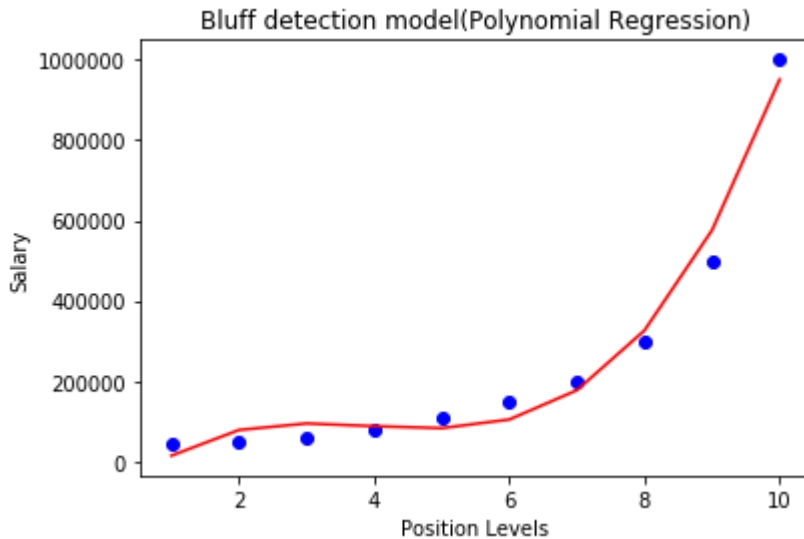
### Output:



As we can see in the above output image, the predictions are close to the real values. The above plot will vary as we will change the degree.

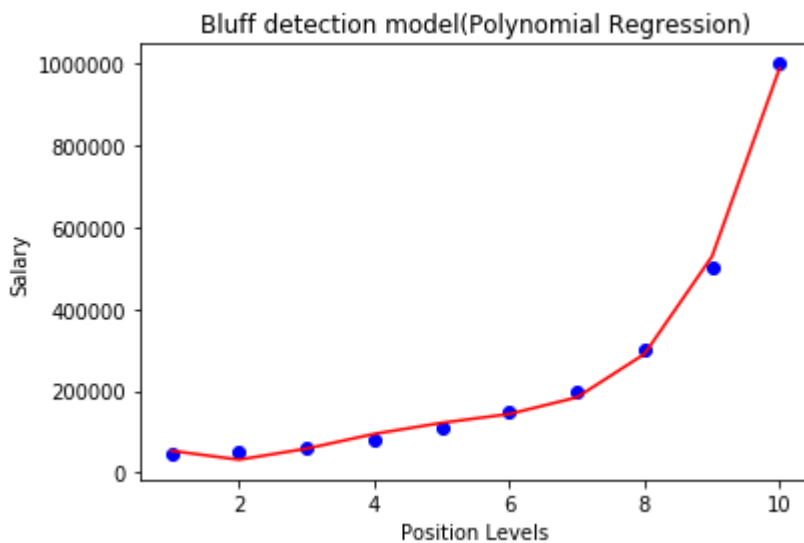
### For degree= 3:

If we change the degree=3, then we will give a more accurate plot, as shown in the below image.



SO as we can see here in the above output image, the predicted salary for level 6.5 is near to 170K\$-190k\$, which seems that future employee is saying the truth about his salary.

**Degree= 4:** Let's again change the degree to 4, and now will get the most accurate plot. Hence we can get more accurate results by increasing the degree of Polynomial.



### Predicting the final result with the Linear Regression model:

Now, we will predict the final output using the Linear regression model to see whether an employee is saying truth or bluff. So, for this, we will use the **predict()** method and will pass the value 6.5. Below is the code for it:

1. `lin_pred = lin_regs.predict([[6.5]])`
2. `print(lin_pred)`

**Output:**

```
[330378.78787879]
```

**Predicting the final result with the Polynomial Regression model:**

Now, we will predict the final output using the Polynomial Regression model to compare with Linear model. Below is the code for it:

1. `poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))`
2. `print(poly_pred)`

**Output:**

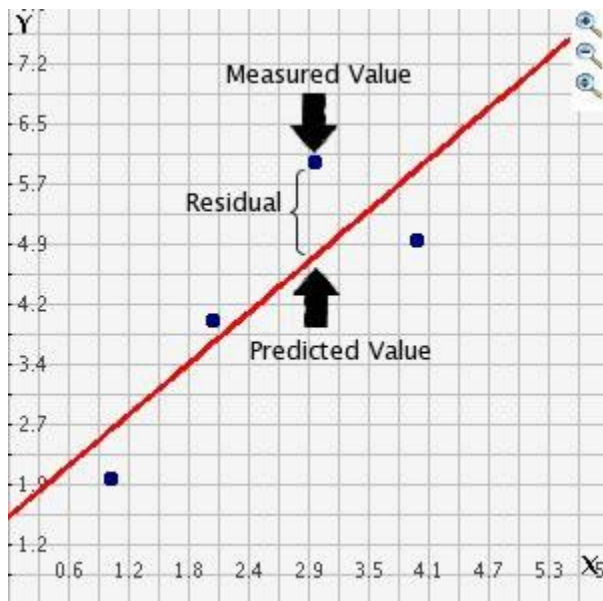
```
[158862.45265153]
```

As we can see, the predicted output for the Polynomial Regression is [158862.45265153], which is much closer to real value hence, we can say that future employee is saying true.

## **Metrics for Regression:**

Regression is a problem where we try to predict a continuous dependent variable using a set of independent variables. For example, weather forecasting, market trends, etc. These problems are used to answer “How much?” or “How many?”

In regression problems, the prediction error is used to define the model performance. The prediction error is also referred to as residuals and it is defined as the difference between the actual and predicted values.



[Source](#)

The regression model tries to fit a line that produces the smallest difference between predicted and actual(measured) values.

Residuals are important when determining the quality of a model. You can examine residuals in terms of their magnitude and/or whether they form a pattern.

- Where the residuals are all 0, the model predicts perfectly. The further residuals are from 0, the less accurate the model is.
- Where the average residual is not 0, it implies that the model is systematically biased (i.e., consistently over-or under-predicting).
- Where residuals contain patterns, it implies that the model is qualitatively wrong, as it is failing to explain some properties of the data.

**Residual = actual value — predicted value**

$$error(e) = y - \hat{y}$$

*So, the question is when you have residuals then why do we need different metrics? Let's find out...*

We can calculate the residual for every point in our data set, and each of these residuals will be of use in assessment.

Date	Month	Inflation (%)	Predicted (%)	Residual (%)
Jan-17	1	0.5	1.8	-1.3
Feb-17	2	2.7	1.8	0.9
Mar-17	3	2.4	1.8	0.6
Apr-17	4	2.2	1.9	0.3
May-17	5	1.9	1.9	0.0
Jun-17	6	1.6	1.9	-0.3
Jul-17	7	1.7	2.0	-0.3
Aug-17	8	1.9	2.0	-0.1
Sep-17	9	2.2	2.0	0.2
Oct-17	10	2.0	2.1	-0.1
Nov-17	11	2.2	2.1	0.1
Dec-17	12	2.1	2.2	0.0

Source: [Displayr](#)

Residual = Inflation — Predicted

We can technically inspect all residuals to judge the model's accuracy, but this does not scale if we have thousands or millions of data points. That's why we have summary measurements that take our collection of residuals and condense them into a *single* value representing our model's predictive ability.



Now we'll turn our focus to metrics of our model.

### **Regression Evaluation Metrics:**

In this section, we will take a closer look at the popular metrics for regression models.

#### **Mean Absolute Error (MAE):**

It is the average of the absolute differences between the actual value and the model's predicted value.

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

#### **Mean Absolute Error**

where,

$N$  = total number of data points

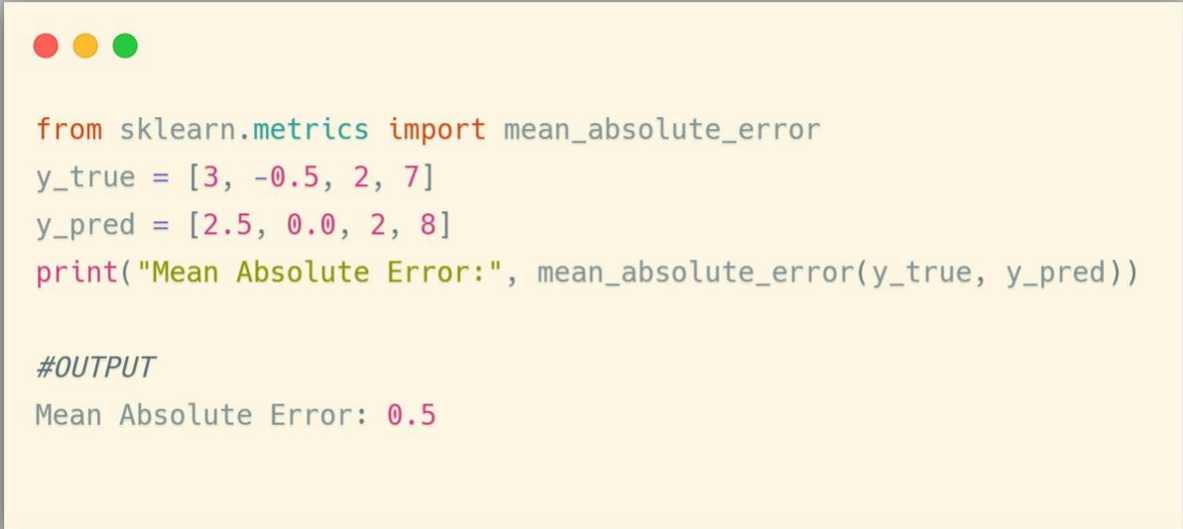
$y_i$  = actual value

$\hat{y}_i$  = predicted value

If we don't take the absolute values, then the negative difference will cancel out the positive difference and we will be left with a zero upon summation.

*A small MAE suggests the model is great at prediction, while a large MAE suggests that your model may have trouble in certain areas. MAE of 0 means that your model is a perfect predictor of the outputs.*

Here's a Scikit-learn implementation of MAE:



```
from sklearn.metrics import mean_absolute_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
print("Mean Absolute Error:", mean_absolute_error(y_true, y_pred))

#OUTPUT
Mean Absolute Error: 0.5
```

## Mean Absolute Error

The mean absolute error (MAE) has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units.

The bigger the MAE, the more critical the error is. It is robust to outliers. Therefore, by taking the absolute values, MAE can deal with the outliers

Here, a big error doesn't overpower a lot of small errors and thus the output provides us with a relatively unbiased understanding of how the model is performing. Hence, it fails to punish the bigger error terms.

MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

### **Mean Squared Error (MSE):**

It is the average of the squared differences between the actual and the predicted values.

Lower the value, the better the regression model.

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

### **Mean Squared Error**

where,

**n** = total number of data points

**y<sub>i</sub>** = actual value

**ŷ<sub>i</sub>** = predicted value

Its unit is the square of the variable's unit.

Here's a Scikit-learn implementation of MSE:



```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
print("Mean Squared Error:", mean_squared_error(y_true, y_pred))
```

*#OUTPUT*

Mean Squared Error: 0.375

## Mean Squared Error

*If you have outliers in the dataset then it penalizes the outliers most and the calculated MSE is bigger. So, in short, It is not Robust to outliers which were an advantage in MAE.*

MSE uses the square operation to remove the sign of each error value and to punish large errors.

As we take the square of the error, the effect of larger errors become more pronounced than smaller error, hence the model can now focus more on the larger errors.

The main reason this is not that useful is that if we make a single very bad prediction, the squaring will make the error even worse and it may skew the metric towards overestimating the model's badness.

On the other hand, if all the errors are small, or rather, smaller than 1, then we may underestimate the model's badness.

### **Root Mean Squared Error (RMSE):**

It is the average root-squared difference between the real value and the predicted value. By taking a square root of MSE, we get the Root Mean Square Error.

We want the value of RMSE to be as low as possible, as lower the RMSE value is, the better the model is with its predictions. A Higher RMSE indicates that there are large deviations between the predicted and actual value.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

### **Root Mean Squared Error**


where,

**n** = total number of data points

**y<sub>j</sub>** = actual value

**ŷ<sub>j</sub>** = predicted value

Here's a Scikit-learn implementation of RMSE:



```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
print("Root Mean Squared Error:", mean_squared_error(y_true, y_pred, squared=False))

#OUTPUT
Root Mean Squared Error: 0.6123724356957945
```

## Root Mean Squared Error

### Max Error:

While RMSE is the most common metric, it can be hard to interpret. One alternative is to look at quantiles of the distribution of the absolute percentage errors. The Max-Error metric is the **worst-case** error between the predicted value and the true value.

Here's a Scikit-learn implementation of Max Error:



```
from sklearn.metrics import max_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
print("Max Error:", max_error(y_true, y_pred))
```

*#OUTPUT*

Max Error: 0.1

## Max Error

### R<sup>2</sup> score, the coefficient of determination:

R-squared explains to what extent the variance of one variable explains the variance of the second variable. In other words, it measures the proportion of variance of the dependent variable explained by the independent variable.

R squared is a popular metric for identifying model accuracy. It tells how close are the data points to the fitted line generated by a regression algorithm. A larger R squared value indicates a better fit. This helps us to find the relationship between the independent variable towards the dependent variable.

*R<sup>2</sup> score ranges from 0 to 1. The closer to 1 the R<sup>2</sup>, the better the regression model is. If R<sup>2</sup> is equal to 0, the model is not performing better than a random model. If R<sup>2</sup> is negative, the regression model is erroneous.*

It is the ratio of the sum of squares and the total sum of squares

$$R^2 = 1 - \frac{SSE}{SST}$$

## R2 Score

where **SSE** is the sum of the square of the difference between the actual value and the predicted value

$$SSE = \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

## Sum of Squared Errors

and, **SST** is the total sum of the square of the difference between the actual value and the mean of the actual value.

$$SST = \sum_{i=1}^m (y_i - \bar{y})^2$$

## Total sum of squares

Here,  $y_i$  is the observed target value,  $\hat{y}_i$  is the predicted value, and  $\bar{y}$  is the mean value,  $m$  represents the total number of observations.

When we add new features in our data, R2 score starts increasing or constant but never decreases because It assumes that while adding more data variance of data increases.

But the problem is when we add an irrelevant feature in the dataset then at that time R2 sometimes starts increasing which is incorrect.

Here's a Scikit-learn implementation of R2 Score:





```
from sklearn.metrics import r2_score  
y_true = [3, -0.5, 2, 7]  
y_pred = [2.5, 0.0, 2, 8]  
print("R2 Score:", r2_score(y_true, y_pred))
```

*#OUTPUT*

R2 score: 0.9486081370449679

## R2 Score

R2 describes the proportion of variance of the dependent variable explained by the regression model. If the regression model is “perfect”, SSE is zero, and R2 is 1. If the regression model is a total failure, SSE is equal to SST, no variance is explained by the regression, and R2 is zero.

## Adjusted R-Square:

**Adjusted R<sup>2</sup>** is the same as standard R<sup>2</sup> except that it penalizes models when additional features are added.

To counter the problem which is faced by R-square, Adjusted r-square penalizes adding more independent variables which don't increase the explanatory power of the regression model.

The value of adjusted r-square is always less than or equal to the value of r-square.

It ranges from 0 to 1, the closer the value is to 1, the better it is.

It measures the variation explained by only the independent variables that actually affect the dependent variable.

$$R^2_{adjusted} = \left[ \frac{(1-R^2)(n-1)}{n-k-1} \right]$$

### **Adjusted R-Squared**

where

**n** is the number of data points

**k** is the number of independent variables in your model

## UNIT-III

**Classification:** Classification problem, Probability based approach, Logistic Regression-log-odd, sigmoid transformation. MNIST, Training a binary classifier, Performance measures: Measuring accuracy using cross validation, Confusion matrix, Precision/Recall and Tradeoffs, ROC curve, Multiclass classification, Error analysis, Multilabel classification, Multioutput classification Introduction to gradient descent

**Classification** is a central topic in machine learning that has to do with teaching machines how to group together data by particular criteria. Classification is the process where computers group data together based on predetermined characteristics — this is called supervised learning. There is an unsupervised version of classification, called clustering where computers find shared characteristics by which to group data when categories are not specified.

A common example of classification comes with detecting spam emails. To write a program to filter out spam emails, a computer programmer can train a machine learning algorithm with a set of spam-like emails labelled as spam and regular emails labelled as not-spam. The idea is to make an algorithm that can learn characteristics of spam emails from this training set so that it can filter out spam emails when it encounters new emails.

Classification is an important tool in today's world, where big data is used to make all kinds of decisions in government, economics, medicine, and more. Researchers have access to huge amounts of data, and classification is one tool that helps them to make sense of the data and find patterns.

While classification in machine learning requires the use of (sometimes) complex algorithms, classification is something that humans do naturally everyday. Classification is simply grouping things together according to similar features and attributes. When you go to a grocery store, you can fairly accurately group the foods by food group (grains, fruit, vegetables, meat, etc.) In machine learning, classification is all about teaching computers to do the same.

Here are a few examples of situations where classification is useful:

### **Classifying Images**

**Speech**

**Tagging**

**Music**

**Identification**

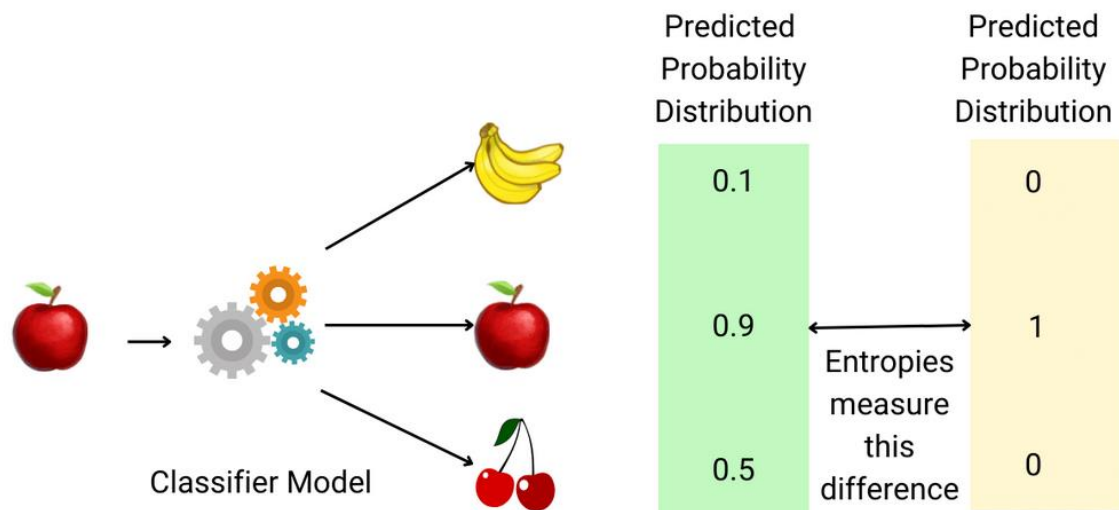
## Classification Problems in Machine Learning

In regression problems, the mapping function that algorithms want to learn is discrete. The objective is to find the decision boundary/boundaries, dividing the dataset into different categories. But the Classification is a type of problem that requires the use of machine learning algorithms that learn how to assign a class label to the input data.

For example, suppose there are three class labels, **[Apple, Banana, Cherry]**. But the problem is that machines don't have the sense to understand these labels. That's why we need to convert these labels into a machine-readable format. For the above example, we can define **Apple = [1,0,0]**, **Banana = [0,1,0]**, **Cherry = [0,0,1]**

Once the machine learns from these labeled training datasets, it will give probabilities of different classes on the test dataset like this: **[P(Apple), P(Banana), P(Cherry)]**

These predicted probabilities can be from one type of probability distribution function (PDF), and the actual (true) labeled dataset can be from another probability distribution function (PDF). If the predicted distribution function follows the actual distribution function, the model is learning accurately. **Note:** These PDF functions are continuous. As a similarity between classification and regression, if the predicted PDF follows the actual PDF, we can say the model learns the trends.



**Some of the standard cost functions for the classification problems**

### **Categorical Cross-Entropy**

Suppose there are M class labels, and the predicted distribution for the  $i$ -th datasample is :

$$P(Y) = [Y_{i1}', Y_{i2}', \dots, Y_{iM}']$$

And, actual distribution for that sample would be,

$$A(Y) = [Y_{i1}, Y_{i2}, \dots, Y_{iM}]$$

$$\text{Cross Entropy (CE}_i\text{)} = -(Y_{i1} \cdot \log(Y_{i1}') + Y_{i2} \cdot \log(Y_{i2}') + \dots + Y_{iM} \cdot \log(Y_{iM}'))$$

$$\text{Categorical Cross Entropy} = - \sum_i^N \frac{CE_i}{N}$$

## Binary Cross-Entropy

This is a special case of categorical cross-entropy, where there is only one output that can have two values, either 0 or 1. For example, if we want to predict whether a cat is present in any image or not.

Here, the cross-entropy function varies with the true value of  $Y$ ,

$$CE_i = -Y_i \log(Y_i'), \text{ if } Y_i = 1$$

$$CE_i = -(1-Y_i) \log(1-Y_i'), \text{ if } Y_i = 0$$

And similarly, Binary-Cross-Entropy would be averaged over all the datasets.

---

Now, the primary question that we should ask ourselves is: **If PDFs (probability distribution functions) are continuous in the range of [0,1], why can't MAE/MSE be chosen here? Take a pause and think!**

**Reason:** MAE and MSE do well when the probability of an event occurring is close to the predicted value or when the wrong prediction's confidence is not that high. To understand the term confidence of prediction, let's take one example:

Suppose our ML model predicted that the patient-lady in the figure below is pregnant, and our model predicted it with the probability of 0.9. We can say that our model is very much confident. Now let's consider one scenario when the ML model says the patient-man in the below figure is pregnant with a probability of 0.9. This is a case where the model predicts something wrong and is confident about the prediction. To address these cases, the model needs to be penalized more for these predictions. Right?

---

Let's calculate the cross-entropy (CE), MAE, and MSE of the case where the ML model is predicting that a man is pregnant with high confidence (Probability ( $Y'$ ) = 0.8). Obviously, the actual output  $Y$  will be 0 here.

$$CE = -(1-Y) \log(1-Y') = -(1-0) \log(1-0.8) = 1.64$$

$$MAE = |(Y-Y')| = |0-0.8| = 0.8$$

$$MSE = |(Y-Y')^2| = (0-0.8)^2 = 0.64$$

As you can see, MAE and MSE have lower values than CE, which means the Cost function /Error function produces more value. Hence the model should be penalized more.

That's why we needed different cost functions for the classification problem.

The most common evaluation metric for classification models would be,

- Accuracy
- Confusion Matrix
- F1-Score
- Precision
- Recall etc. (Definitions can be found [here](#) ). We will learn about these terms in greater detail in our later blogs.

#### Examples of classification problems

- Classifying if a mail is spam or not, based on its content, and how others have classified similar types of mails.
- Classifying a dog breed based on its physical features such as height, width, and skin color.
- Classifying whether today's weather is hot or cold.

#### Algorithms for Classification

- Logistic Regression
- Support Vector Classification
- Decision Tree

## Introduction to Probabilistic Classification:

### Classifying cats and dogs

Imagine creating a model with the sole purpose of classifying cats and dogs. The classification model will not be perfect and therefore wrongly classify certain observations. Some cats will be classified as dogs and vice versa. That's life. In this example, the model classifies 100 cats and dogs. The [confusion matrix](#) is a commonly used visualization tool to show prediction accuracy and Figure 1 shows the confusion matrix for this example.

	Actual: Cat	Actual: Dog
Predicted: Cat	45	12
Predicted: Dog	8	35

Figure 1: Confusion matrix for classification of 100 cats and dogs. Source: Author.

Let's focus on the 12 observations where the model predicts a cat while in reality it is a dog. If the model predicts 51% probability of cat and it turns out to be a dog, for sure that's possible. However, if the model predicts 95% probability of cat and it turns out to be a dog? This seems highly unlikely.

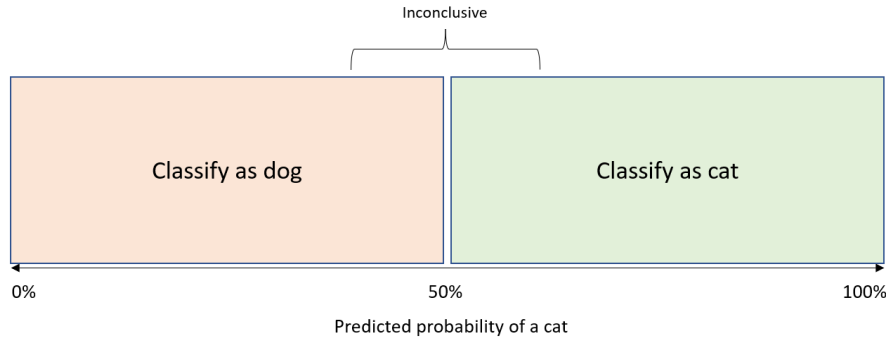


Figure 2: Predicted probability of cat and the classification threshold. Source: Author.

Classifiers use a predicted probability and a threshold to classify the observations. Figure 2 visualizes the classification for a threshold of 50%. It seems intuitive to use a threshold of 50% but there is no restriction on adjusting the threshold. So, in the end the only thing that matters is the ordering of the observations. Changing the objective to predict probabilities instead of labels requires a different approach. For this, we enter the field of *probabilistic classification*.

## Evaluation metric 1: Logloss

Let us generalize from cats and dogs to class labels of 0 and 1. Class probabilities are any real number between 0 and 1. The model objective is to match predicted probabilities with class labels, i.e. to maximize the [likelihood](#), given in Eq. 1, of observing class labels given the predicted probabilities.

$$\mathbb{P}(\vec{x}, \vec{y}) = \prod_{i=1}^N \hat{p}(x_i) \cdot y_i$$

Equation 1: Likelihood for class labels  $\mathbf{y}$  and predicted probabilities based on features  $\mathbf{x}$ .

A major drawback of the likelihood is that if the number of observations grow, the product of the individual probabilities becomes increasingly small. So, with enough data, the likelihood will underflow the numerical precision of any computer. Next to that, a product of parameters is difficult to differentiate. That's the reason the logarithm of the likelihood is preferred, commonly referred to as the loglikelihood. A logarithm is a monotonically increasing function of its argument. Therefore, *maximization of the log of a function is equivalent to maximization of the function itself*.

$$\text{Logloss}(\vec{x}, \vec{y}) = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \ln(\hat{p}(x_i)) + (1 - y_i) \cdot \ln(1 - \hat{p}(x_i))]$$

Equation 2: Logloss for class labels  $\mathbf{y}$  and predicted probabilities based on features  $\mathbf{x}$ .

Nonetheless, the loglikelihood still scales with the number of observations so an average loglikelihood is better metric to explain the observed variation. However, in practice, most people minimize the negative average loglikelihood instead maximizing the average



loglikelihood because optimizers normally minimize functions. Data scientists commonly refer to this metric as Logloss, as given in Eq. 2. For a more elaborate discussion of the Logloss and its relation to the evaluation metrics normally used in classification model evaluation, I refer you to [this article](#).

## Evaluation metric 2: Brier Score

Next to the Logloss, the [Brier Score](#), as given in Eq. 3, is commonly used as an evaluation metric for predicted probabilities. In essence, it is a quadratic loss on the predicted probabilities and the class labels. Note the similarity between the [Mean Squared Error \(MSE\)](#) used in regression model evaluation.

$$\text{Brier Score}(\vec{x}, \vec{y}) = \frac{1}{N} \sum_{i=1}^N (\hat{p}(x_i) - y_i)^2$$

Equation 3: Brier Score for class labels  $\mathbf{y}$  and predicted probabilities based on features  $\mathbf{x}$ .

However, a notable difference with the MSE is that the minimum Brier Score is not 0. The Brier Score is the squared loss on the **labels and probabilities**, and therefore by definition is not 0. Simply said, the minimum is not 0 if the underlying process is non-deterministic which is the reason to use probabilistic classification in the first place. In order to cope with this problem, the probabilities are commonly evaluated on a relative basis with other probabilistic classifiers using for instance the Brier Skill Score.

There are perhaps four main types of classification tasks that you may encounter; they are:

- Binary Classification
- Multi-Class Classification
- Multi-Label Classification
- Imbalanced Classification

### Binary Classification

[Binary classification](#) refers to those classification tasks that have two class labels.

Examples include:

- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not).

Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.

For example “*not spam*” is the normal state and “*spam*” is the abnormal state. Another example is “*cancer not detected*” is the normal state of a task that involves a medical test and “*cancer detected*” is the abnormal state.

The class for the normal state is assigned the class label 0 and the class with the abnormal state is assigned the class label 1.

It is common to model a binary classification task with a model that predicts a [Bernoulli probability distribution](#) for each example.

The Bernoulli distribution is a discrete probability distribution that covers a case where an event will have a binary outcome as either a 0 or 1. For classification, this means that the model predicts a probability of an example belonging to class 1, or the abnormal state.

Popular algorithms that can be used for binary classification include:

- Logistic Regression
- k-Nearest Neighbors
- Decision Trees
- Support Vector Machine
- Naive Bayes

Some algorithms are specifically designed for binary classification and do not natively support more than two classes; examples include Logistic Regression and Support Vector Machines.

## Logistic Regression

Logistic Regression is one of the supervised machine learning algorithms used for classification. In logistic regression, the dependent variable is categorical.

The objective of the model is, given the independent variables, what is the class likely to be? [For binary classification, 0 or 1]

## Why not Linear Regression?

In Logistic Regression-binary classification, we will predict the output as 0 or 1.

**Example:**

1. Diabetic (1) or not (0)
2. Spam (1) or Ham (0)
3. Malignant(1) or not (0)

In Linear Regression, output prediction will be continuous. So, if we fit a linear model, it won't predict the output between 0 and 1.

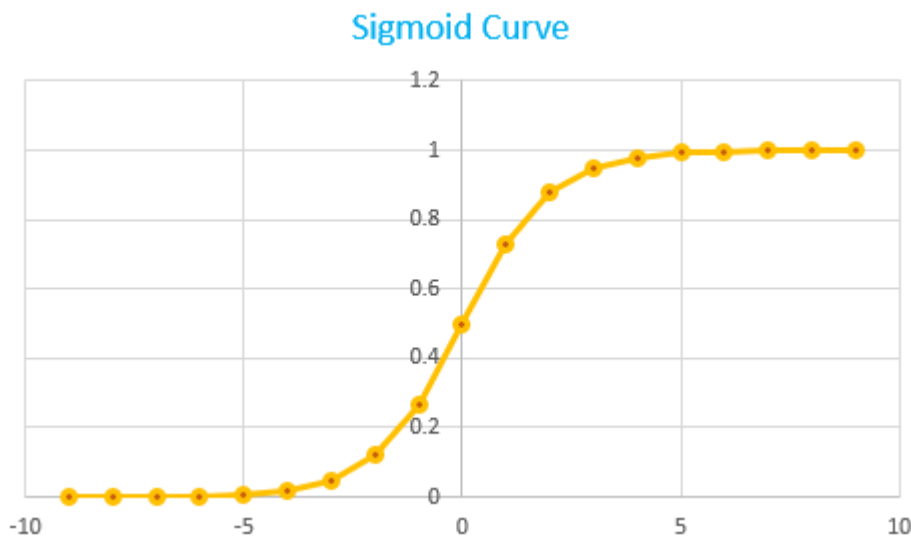
So, we have to transform the linear model to the S -curve using the sigmoid function, which will convert the input between 0 and 1.

## Sigmoid Function

The sigmoid function is used to convert the input into range 0 and 1.

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

1. If  $z \rightarrow -\infty$ ,  $\text{sigmoid}(z) \rightarrow 0$
2. If  $z \rightarrow \infty$ ,  $\text{sigmoid}(z) \rightarrow 1$
3. If  $z=0$ ,  $\text{sigmoid}(z)=0.5$



Sigmoid Curve [Image by Author]

So, if we input the linear model to the sigmoid function, it will convert the input between range 0 and 1

In Linear regression, the predicted value of y is calculated by using the below equation.

$$\hat{y} = \beta_0 + \beta_1 x$$

For logistic regression , predicted value of y is calculated by

$$\hat{y} = \text{sigmoid}(\beta_0 + \beta_1 x) \rightarrow \hat{y} \in [0,1]$$

$$\hat{y} = \frac{1}{1+e^{-(\beta_0 + \beta_1 x)}}$$

In logistic regression,  $\hat{y}$  is  $p(y=1|x)$ . This means  $\hat{y}$  provides an estimate of the probability that  $y=1$ , given a particular set of values for independent variables(x)

→ If the predicted value is close to 1 means we can be more certain that the data point

*belongs to class 1.*

*→ If the predicted value is close to 0 means we can be more certain that the data point belongs to class 0.*

# How to determine the best fit sigmoid curve?

## Cost Function

### Why not least squares as cost function?

In logistic regression, the actual  $y$  value will be 0 or 1. The predicted  $y$  value  $\hat{y}$  will be between 0 and 1.

In the least-squares method, the error is calculated by subtracting actual  $y$  and predicted  $y$  value and squaring them

$$\text{Error} = (y - \hat{y})^2$$

If we calculate least squares for the misclassified data point say  $y=0$  and  $\hat{y}$  is close to 1, the error will be very less only.

The cost incurred is very less even for misclassified data points. This is one of the reasons, least squares is not used as a cost function for logistic regression.

### Cost Function — Log loss (Binary Cross Entropy)

Log loss or Binary Cross Entropy is used as a cost function for logistic regression

$$\text{Error} = -\{y \ln \hat{y} + (1-y) \ln (1-\hat{y})\}$$

Let's check some properties of the classification cost function

1. If  $y=\hat{y}$ , Error should be zero
2. The error should be very high for misclassification
3. The error should be greater than or equal to zero.

Let's check whether these properties hold good for the log loss or binary cross-entropy function.

1. **If  $y=\hat{y}$ , the error should be zero.**

Case 1:  $y=0$  and  $\hat{y}=0$  or close to 0

$$\text{Error} = -\{y \ln \hat{y} + (1-y) \ln (1-\hat{y})\}$$

$$= -\{0 + (1-0) \ln (1-0)\}$$

$$= -\{0 + 1 \ln 1\}$$

$$\text{Error} = 0$$

Case 2:  $y=1$  and  $\hat{y}=1$  or close to 1.

$$\text{Error} = -\{y \ln \hat{y} + (1-y) \ln (1-\hat{y})\}$$

$$= -\{1 \ln 1 + 0\}$$

$$\text{Error} = 0$$

$\ln 1 = 0$  and  $\ln 0 = -\infty$

2. The error should be very high for misclassification

Case 1:  $y=1$  and  $\hat{y}=0$  or close to 0

$$\text{Error} = -\{y \ln \hat{y} + (1-y) \ln (1-\hat{y})\}$$

$$= -\{1 \ln 0 + 0\}$$

$$\text{Error} \rightarrow \text{Infinity}$$

Case 2:  $y=0$  and  $\hat{y}=1$  or close to 1

$$\text{Error} = -\{y \ln \hat{y} + (1-y) \ln (1-\hat{y})\}$$

$$= -\{0 + 1 \ln 0\}$$

Error → Infinity

The error tends to be very high for misclassified data points.

3. The error should be greater than or equal to zero.

$$\text{Error} = -\{y \ln \hat{y} + (1-y) \ln (1-\hat{y})\}$$

→  $y$  is either 0 or 1

→  $\hat{y}$  is always between 0 and 1

→  $\ln \hat{y}$  is negative and  $\ln (1-\hat{y})$  is negative

→ negative sign before the expression is included to make the error positive [In linear regression least-squares method, we will be squaring the error]

So, the error will be always greater than or equal to zero.

## Interpreting Model coefficient

To interpret the model coefficient, we need to know the terms odds, log odds, odds ratio.

### Odds, Log Odds, Odds Ratio

#### Odds

Odds is defined as the probability of an event occurring divided by the probability of the event not occurring.

$$\text{Odds} = \frac{P(\text{Occurring})}{P(\text{Not Occurring})}$$

$$\text{Odds} = \frac{p}{1-p}$$

**Example:** Odds of getting 1 while rolling a fair die

$$\text{Odds (getting 1)} = \frac{1/6}{5/6} = \frac{1}{5} = 1:5$$

### Log odds (Logit Function)

$$\text{Log odds} = \ln(p/1-p)$$

After applying the sigmoid function, we know that

$$\hat{y} = p(y=1|x) = \frac{1}{1+e^{-(\beta_0 + \beta_1 x)}}$$

From this equation, odds can be written as,

$$\text{Odds} = \frac{p}{1-p} = e^{(\beta_0 + \beta_1 x)}$$

$$\text{Log Odds} = \ln(p/1-p) = \beta_0 + \beta_1 x$$

So, we can convert the logistic regression as a linear function by using log odds.

### Odds Ratio

Odds Ratio is the ratio of two odds

$$\text{Odds Ratio} = \frac{\text{Odds}_0}{\text{Odds}_1}$$

$$\text{Odds Ratio} = \frac{p_1 / 1-p_1}{p_0 / 1-p_0}$$

### Interpreting Logistic Regression Coefficient

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1$$

Logistic Regression model

$\beta_0 \rightarrow$  Log odds is  $\beta_0$  when  $X$  is zero.

$\beta_1 \rightarrow$  Change in log-odds associated with variable  $X_1$ .

If  $X_1$  is numerical variables,  $\beta_1$  indicates, for every one-unit increase in  $X_1$ , log odds is increased by  $\beta_1$ .

If  $X_1$  is a binary categorical variable,  $\beta_1$  indicates, change in log odds for  $x_1=1$  relative to  $X_1=0$ .

## How to get Odds Ratio from the model coefficient?

### Odds Ratio in Logistic Regression

The odds ratio of an independent variable in logistic regression depends on how that odds change with one unit increase in that particular variable by keeping all the other independent variables constant.

$\beta_1 \rightarrow$  Change in log-odds associated with variable  $X_1$ .  
The odds Ratio for variable  $X_1$  is the exponential of  $\beta_1$

Odds Ratio of variable  $x_1 = e^{\beta_1}$



### Derivation of Odds Ratio from Model Coefficient

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 \rightarrow \ln(\text{odds}) = \beta_0 + \beta_1 X_1$$

### Calculating odds ratio for change in $x_1$ by 1 unit

$$\text{odds ratio} = \frac{\left(\frac{p}{1-p} \mid x_1+1\right)}{\left(\frac{p}{1-p} \mid x_1\right)}$$

$$\ln(\text{odds ratio}) = \frac{\ln\left(\frac{p}{1-p} \mid x_1+1\right)}{\ln\left(\frac{p}{1-p} \mid x_1\right)}$$

$$= \ln\left(\frac{p}{1-p} \mid x_1 + 1\right) - \ln\left(\frac{p}{1-p} \mid x_1\right)$$

$$= \beta_0 + \beta_1 (X_1+1) - \beta_0 + \beta_1 X_1$$

$$= \beta_1 X_1 + \beta_1 - \beta_1 X_1$$

$$\ln(\text{odds ratio}) = \beta_1$$

$$\text{Odds Ratio} = e^{\beta_1}$$

## Evaluation Metrics for Classification

1. Accuracy
2. TPR
3. TNR
4. FPR
5. PPR
6. F1 Score

## Accuracy

Accuracy measures the proportion of actual predictions out of total predictions. Here we didn't know the exact distribution of error. [Distribution between False-positive and False negative]

Actual/Predicted	True	False
True	True Positive	False Negative
False	False Positive	True Negative

Confusion Matrix [Image by Author]

In the accuracy metric, we didn't know the exact distribution of error [Distribution between False-positive and False negative]. So, we go for other metrics.

## Sensitivity or Recall or True Positive rate(TPR)

True Positive Rate measures the proportion of actual positives that are correctly classified.

## Specificity or True Negative Rate (TNR)

True Negative Rate measure the proportion of actual negatives that are correctly classified.

## False Positive rate or (1-Specificity)

False Positive Rate measure the proportion of actual negatives that are misclassified.

## Positive Predicted Rate (PPR) or Precision

Positive Predicted Rate measures the proportion of the actual positives out of total positive predictions.

## F1 Score

F1 Score is the harmonic mean of precision and recall

How to calculate the harmonic mean?

1. Take the inverse of precision and recall ( $1/\text{precision}$ ,  $1/\text{recall}$ )
2. Find the average of inverse of precision and recall

$$\frac{1/Precision + 1/Recall}{2}$$

3. Then, inverse the result.

$$\frac{2}{1/Precision + 1/Recall}$$

$$\frac{2}{Recall + Precision} / Precision * Recall$$

$$F1 \text{ score} = \frac{2 * Precision * Recall}{Precision + Recall}$$

### Which metric to chose?

It depends on the problem statement.

1. **Accuracy** → When we need to know the prediction accuracy like how many 1's classified as 1 and how many 0's classified as 0 but not concerned with the distribution of errors(FP and FN).
2. **Sensitivity** → When we want all our positive records to be identified correctly. More importance on False Negative[In cancer dataset, cancer patients should be predicted correctly]
3. **Specificity** → When we don't want any negative labels to be misclassified. More importance on False Positive[In spam detection, we want all our genuine emails to be predicted properly]
4. **F1 score** metric is used for an imbalanced dataset.

## Metrics Formula

TPR	True Positive Rate	Sensitivity or Recall	$\frac{\text{True Positive}}{\text{Total number of Actual Positives}}$	$\frac{TP}{TP + FN}$
TNR	True Negative Rate	Specificity	$\frac{\text{True Negative}}{\text{Total number of Actual Negatives}}$	$\frac{TN}{TN + FP}$
FPR	False Positive Rate	1-Specificity	$\frac{\text{False Positive}}{\text{Total number of Actual Negatives}}$	$\frac{FP}{TN + FP}$
PPR	Positive Predictive Rate	Precision	$\frac{\text{True Positive}}{\text{Total number of Predicted Positives}}$	$\frac{TP}{TP + FP}$

Accuracy	$\frac{TP + TN}{\text{Total number of points}}$
F1 Score	$\frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$

Metrics [Image by Author]

## What is the threshold level? How to set the threshold level?

### Threshold Level

The logistic regression model predicts the outcome in terms of probability. But, we want to make a prediction of 0 or 1. This can be done by setting a threshold value.

If the threshold value is set as 0.5 means, the predicted probability greater than 0.5 will be converted to 1 and the remaining values as 0.

### ROC Curve

ROC curve is the trade-off between **FPR and TPR** or (**1-Specificity vs Sensitivity**) at all classification threshold levels.

FPR → Fraction of negative labels misclassified.

TPR → Fraction of positive labels correctly classified

If FPR=TPR=0 means the model predicts all instances as the negative class.

If FPR=TPR=1 means model predicts all instances as the positive class

If TPR=1, FPR=0 means the model predicts all data points correctly. (ideal model)

**Example:** I have calculated all metrics from actual y and predicted probability at all threshold levels.

Actual Y	Predicted Probability
0	0.25
0	0.33
0	0.45
0	0.67
0	0.75
0	0.74
1	0.78
1	0.8
1	0.89
1	0.92

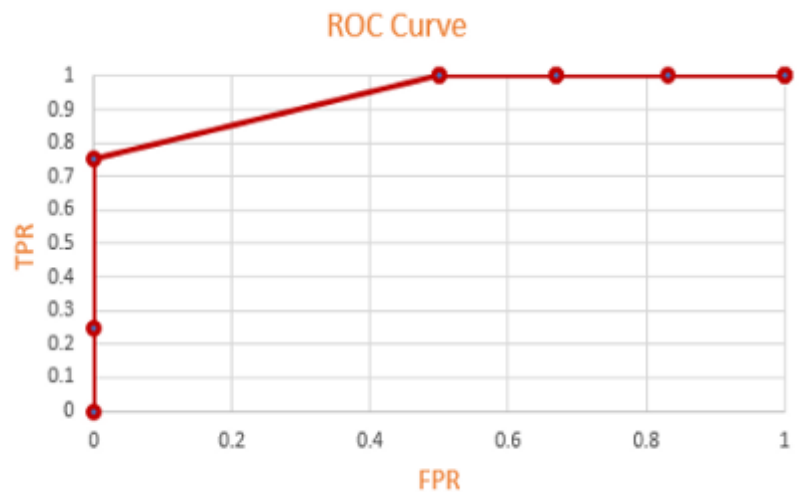
Actual y and predicted probability

Threshold	TP	TN	FP	FN	TPR(Recall)	FPR	Specificity	Precision	Accuracy
0	4	0	6	0	1	1	0	0.4	0.4
0.1	4	0	6	0	1	1	0	0.4	0.4
0.2	4	0	6	0	1	1	0	0.4	0.4
0.3	4	1	5	0	1	0.833333	0.166666667	0.444444	0.5
0.4	4	2	4	0	1	0.666667	0.333333333	0.5	0.6
0.5	4	2	4	0	1	0.666667	0.333333333	0.5	0.6
0.6	4	3	3	0	1	0.5	0.5	0.571429	0.7
0.7	4	3	3	0	1	0.5	0.5	0.571429	0.7
0.8	3	6	0	1	0.75	0	1	1	0.9
0.9	1	6	0	3	0.25	0	1	1	0.7
1	0	6	0	4	0	0	1	0	0.6

All Metrics calculated for all threshold levels (0–1)

Let's plot ROC-Curve [FPR vs TPR] at all threshold levels.

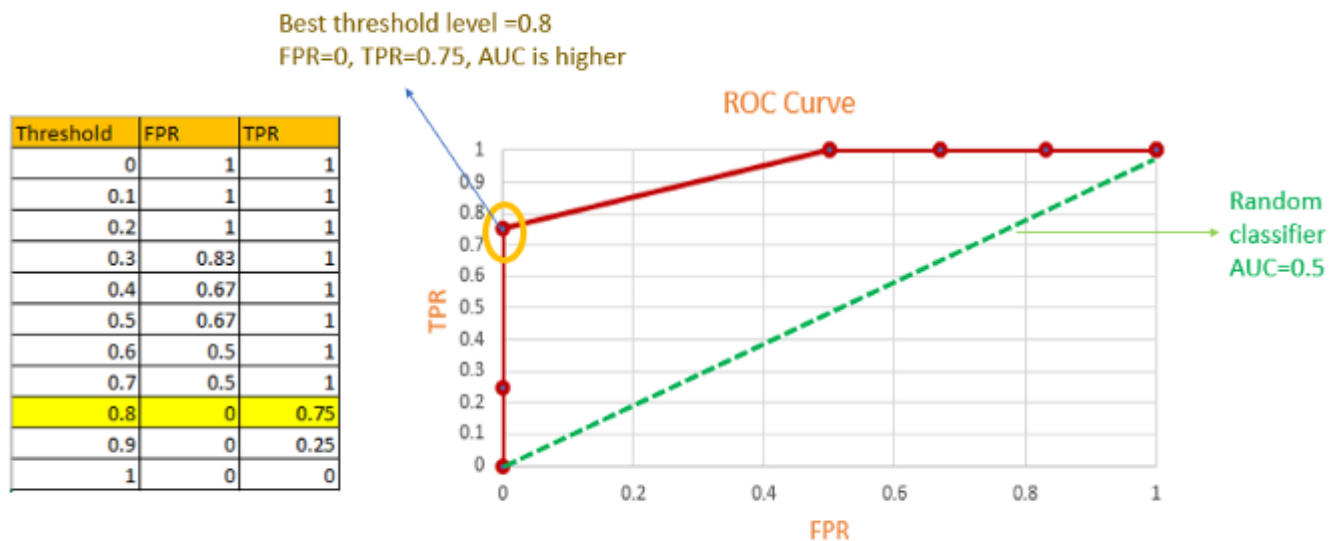
Threshold	FPR	TPR
0	1	1
0.1	1	1
0.2	1	1
0.3	0.83	1
0.4	0.67	1
0.5	0.67	1
0.6	0.5	1
0.7	0.5	1
0.8	0	0.75
0.9	0	0.25
1	0	0



ROC Curve

## Interpretation of ROC curve

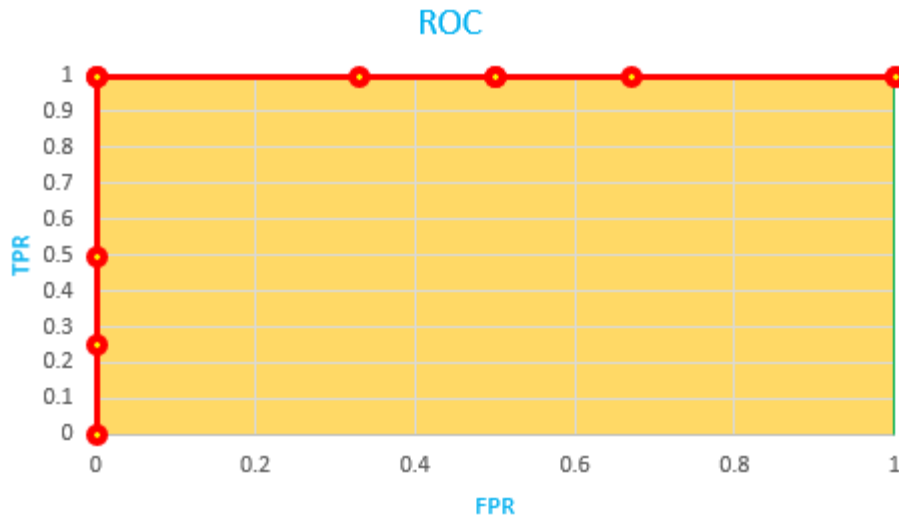
From the above ROC curve, we can choose the best threshold level. At FPR = 0 and TPR = 0.75, is the best threshold level for the above ROC curve. AUC tends to be higher for that point and also FPR is zero.



AUC — Area under the curve is the area covered by the ROC curve. AUC range from 0–1.

For any random classifier, AUC = 0.5. So AUC score for good models should be between 0.5 and 1.

If TPR=1, FPR=0 means the model predicts all data points correctly. (ideal model). In this case, AUC is 1.



AUC=1 [Image by Author]

### Why ROC curve is used?

1. To compare the performance of different models. AUC is calculated from the ROC curve and the model which has a higher AUC performs better.
2. To select the best threshold for the model.

## Key Takeaways

1. To interpret the model coefficient we use the equation

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1$$

2. To calculate  $\hat{y}$ , we use the equation

$$\hat{y} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

3. Exponential model coefficient gives the odds ratio

$$\text{Odds Ratio} = e^{\beta_1}$$

## ***MNIST***

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint.

## **What is Binary Classification?**

In machine learning, binary classification is a supervised learning algorithm that categorizes new observations into one of **two** classes.

The following are a few binary classification applications, where the 0 and 1 columns are two possible classes for each observation:

<b>Application</b>	<b>Observation</b>	<b>0</b>	<b>1</b>
Medical Diagnosis	Patient	Healthy	Diseased
Email Analysis	Email	Not Spam	Spam
Financial Data Analysis	Transaction	Not Fraud	Fraud
Marketing	Website visitor	Won't Buy	Will Buy
Image Classification	Image	Hotdog	Not Hotdog



## Quick example

In a medical diagnosis, a binary classifier for a specific disease could take a patient's symptoms as input features and predict whether the patient is healthy or has the disease. The possible outcomes of the diagnosis are **positive** and **negative**.

## Evaluation of binary classifiers

If the model successfully predicts the patients as positive, this case is called *True Positive (TP)*. If the model successfully predicts patients as negative, this is called *True Negative (TN)*. The binary classifier may misdiagnose some patients as well. If a diseased patient is classified as healthy by a negative test result, this error is called *False Negative (FN)*. Similarly, If a healthy patient is classified as diseased by a positive test result, this error is called *False Positive (FP)*.

We can evaluate a binary classifier based on the following parameters:

- True Positive (TP): The patient is diseased and the model predicts "diseased"
- False Positive (FP): The patient is healthy but the model predicts "diseased"
- True Negative (TN): The patient is healthy and the model predicts "healthy"
- False Negative (FN): The patient is diseased and the model predicts "healthy"

After obtaining these values, we can compute the **accuracy score** of the binary classifier as follows:

The following is a *confusion matrix*, which represents the above parameters:

		PREDICTED	
		Positive	Negative
ACTUAL	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

In machine learning, many methods utilize binary classification. The most common are:

- Support Vector Machines
- Naive Bayes
- Nearest Neighbor
- Decision Trees
- Logistic Regression
- Neural Networks

The following Python example will demonstrate using binary classification in a logistic regression problem.

## A Python example for binary classification

For our data, we will use the breast cancer dataset from scikit-learn. This dataset contains tumor observations and corresponding labels for whether the tumor was malignant or benign.

First, we'll import a few libraries and then load the data. When loading the data, we'll specify `as_frame=True` so we can work with pandas objects (see our [pandas tutorial](#) for an introduction).

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

dataset = load_breast_cancer(as_frame=True)
```

### Learn Data Science with

The dataset contains a DataFrame for the observation data and a Series for the target data.

Let's see what the first few rows of observations look like:

```
dataset['data'].head()
```

Out:

	mean radius	mean texture	mean perimeter	mean area	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	worst radius	worst texture	worst perimeter	worst area	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	
17	10.9	122.38	101.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	0.2538	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
20	17.57	132.90	132.26	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.249	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
19	21.69	130.00	120.03	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.2357	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	11.20	77.538	101.0	0.14840	0.28360	0.2001	0.14710	0.2519	0.09871	0.1438	26.33	98.60	56.0	0.2022	0.8666	0.6119	0.2654	0.6601	0.17890

malignancy	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	63.59	1018.1	0.1623	0.6637	0.1189	0.2712	0.3001	0.4686	21.98	16.66	101.6	1533.0	0.1753	0.2763	0.3001	0.4686	0.3001	0.4686
0	20.57	17.31	132.6	1961.0	0.1753	0.8169	0.1809	0.2743	0.3399	0.5804	25.04	19.51	158.3	2141.0	0.1809	0.2743	0.3399	0.5804	0.5804	0.5804
0	19.76	14.71	135.8	1959.0	0.1471	0.8110	0.2098	0.3403	0.3397	0.6381	23.51	18.10	142.5	2019.0	0.1471	0.2098	0.3403	0.3397	0.6381	0.6381
0	17.35	15.33	149.4	2149.0	0.1735	0.7571	0.1976	0.2738	0.3114	0.5101	20.38	16.99	143.6	2044.7	0.1735	0.2738	0.3114	0.5101	0.5101	0.5101
0	16.97	11.42	104.0	1634.0	0.1471	0.9046	0.2098	0.3403	0.3397	0.6381	23.51	18.10	142.5	2019.0	0.1471	0.2098	0.3403	0.3397	0.6381	0.6381

5 rows × 30 columns

The output shows five observations with a column for each feature we'll use to predict malignancy.

Now, for the targets:

```
dataset['target'].head()
Out:
0      0
1      0
2      0
3      0
4      0
Name: target, dtype: int32
```

### Learn Data Science with

The targets for the first five observations are all zero, meaning the tumors are benign. Here's how many malignant and benign tumors are in our dataset:

```
dataset['target'].value_counts()
Out:
1      357
0      212
Name: target, dtype: int64
```

## Learn Data Science with

So we have 357 malignant tumors, denoted as 1, and 212 benign, denoted as 0. So, we have a binary classification problem.

To perform binary classification using logistic regression with sklearn, we must accomplish the following steps.

### **Step 1: Define explanatory and target variables**

We'll store the rows of observations in a variable `x` and the corresponding class of those observations (0 or 1) in a variable `y`.

```
X = dataset['data']  
y = dataset['target']
```

## Learn Data Science with

### **Step 2: Split the dataset into training and testing sets**

We use 75% of data for training and 25% for testing. Setting `random_state=0` will ensure your results are the same as ours.

```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size=0.25,  
random_state=0)
```

## Learn Data Science with

### **Step 3: Normalize the data for numerical stability**

Note that we normalize *after* splitting the data. It's good practice to apply any data transformations to training and testing data separately to prevent [data leakage](#).

```
from sklearn.preprocessing import StandardScaler  
  
ss_train = StandardScaler()  
X_train = ss_train.fit_transform(X_train)
```

```
ss_test = StandardScaler()  
X_test = ss_test.fit_transform(X_test)
```

[Learn Data Science with](#)

#### **Step 4: Fit a logistic regression model to the training data**

This step effectively trains the model to predict the targets from the data.

#### **Step 5: Make predictions on the testing data**

With the model trained, we now ask the model to predict targets based on the test data.

```
predictions = model.predict(X_test)
```

[Learn Data Science with](#)

#### **Step 6: Calculate the accuracy score by comparing the actual values and predicted values.**

We can now calculate how well the model performed by comparing the model's predictions to the true target values, which we reserved in the `y_test` variable.

First, we'll calculate the confusion matrix to get the necessary parameters:

```
from sklearn.metrics import confusion_matrix  
  
cm = confusion_matrix(y_test, predictions)  
  
TN, FP, FN, TP = confusion_matrix(y_test, predictions).ravel()  
  
print('True Positive(TP)   = ', TP)  
print('False Positive(FP)  = ', FP)  
print('True Negative(TN)   = ', TN)  
print('False Negative(FN)  = ', FN)
```

Out:

```
True Positive(TP)   = 86  
False Positive(FP)  = 2  
True Negative(TN)   = 51  
False Negative(FN)  = 4
```

[Learn Data Science with](#)

With these values, we can now calculate an accuracy score:

```
accuracy = (TP + TN) / (TP + FP + TN + FN)

print('Accuracy of the binary classifier = {:.3f}'.format(accuracy))
Out:

Accuracy of the binary classifier = 0.958
```

## Cross Validation and Performance Measures

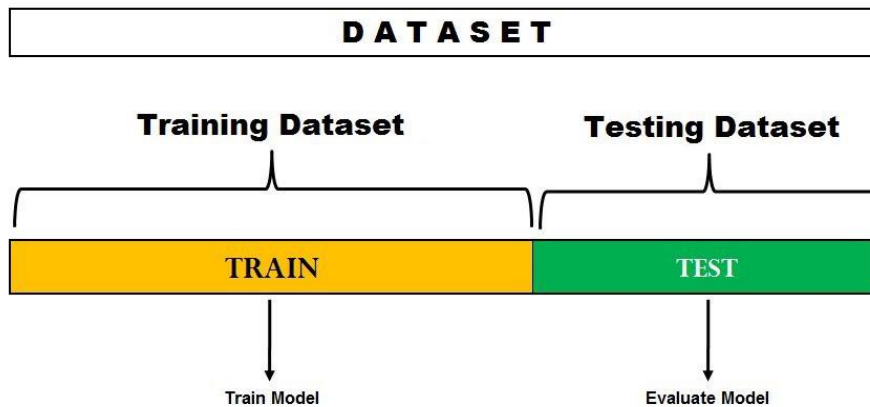
Deciding what cross validation and performance measures should be used while using a particular machine learning technique is very important. After training our model on the dataset, we can't say for sure that the model will perform well on the data which it hasn't seen before. The process of deciding whether the numerical results quantifying hypothesised relationships between variables, are acceptable as descriptions of the data, is known as validation. Based on the performance on unseen data, we can say whether model is overfitted, underfitted or well generalized.

### Cross Validation

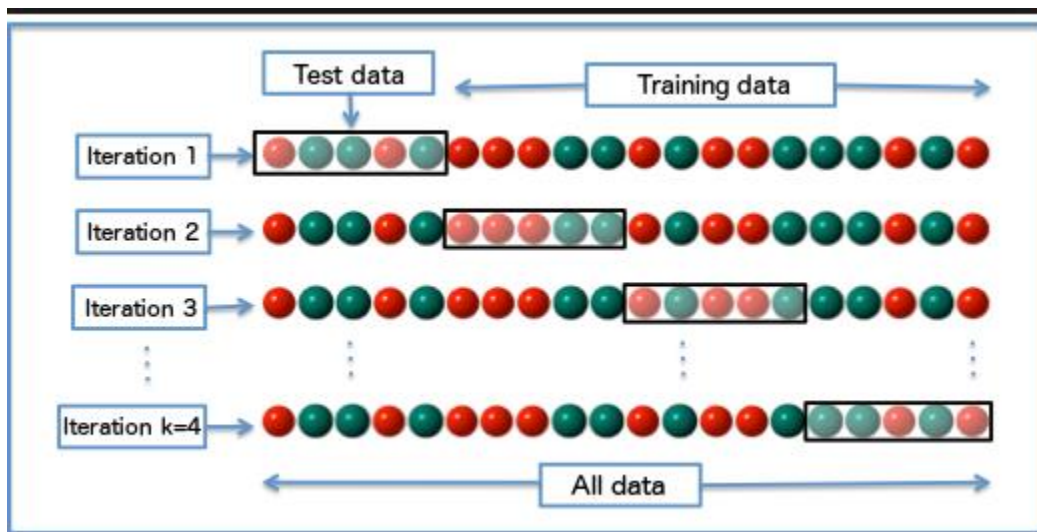
Cross validation is a technique which is used to evaluate the machine learning model by training it on the subset of the available data and then evaluating them on the remaining input data. On a simple note, we keep a portion of data aside and then train the model on the remaining data. And then we test and evaluate the performance of model on portion of data that was kept aside.

### Types of Cross Validation Techniques

1. **Holdout Method:** The holdout method is the simple type of cross validation where the data set is divided into two sets, called the training set and the testing set. The model is fitted and trained using the training set only. Then the model is asked to predict the output values for the data in the testing set and it has never seen this data before. The model is evaluated using the appropriate performance measure such as mean absolute test set error.  
**Advantage** — It is preferable to the residual method and takes less time to compute. However, its evaluation can have a high variance. The evaluation depends entirely on which data points are in the training set and the test set, and thus the evaluation will be different depending on the division made.



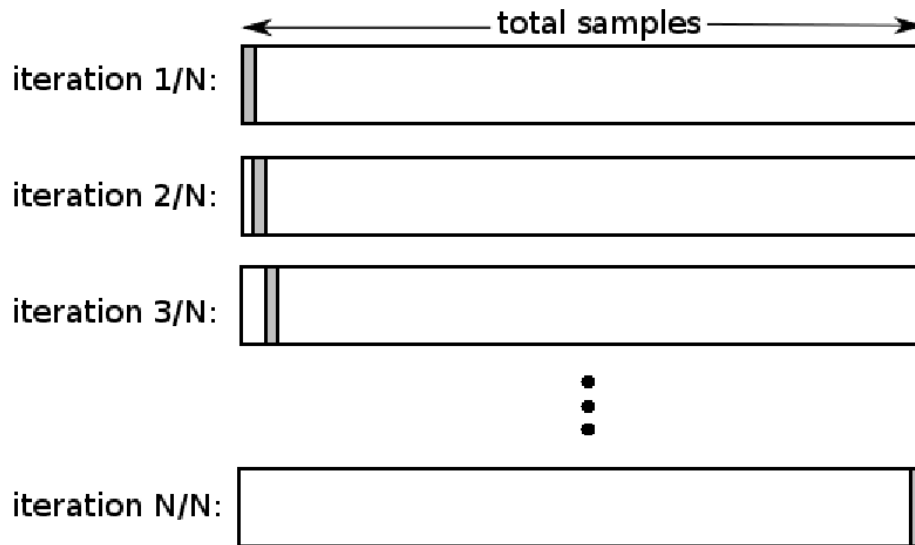
**2. K-Fold Cross Validation Method:** It is a modification in the holdout method. The dataset is divided into  $k$  subsets and the value of  $k$  shouldn't be too small or too large, ideally we choose 5 to 10 depending on the data size. The higher value of  $k$  leads to less biased model whereas the lower value of  $K$  is similar to the holdout approach. Then we train the model using the  $k-1$  folds and validate and test the model on the remaining  $k$ th fold. Note down the errors. This process is repeated until every  $K$ -fold serve as the test set. Then the average of the recorded scores is taken which is the performance metric for the model.



**Advantage** — It doesn't matter how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set  $k-1$  times. The variance of the resulting estimate is reduced as  $k$  is increased.

**Disadvantage** — The training algorithm has to be rerun from scratch  $k$  times, which means it takes  $k$  times as much computation to make an evaluation.

**3. Leave-one-out cross validation** is K-fold cross validation taken to its logical extreme, with  $K$  equal to  $N$ , the number of data points in the set. That means that  $N$  separate times, the model is trained on all the data except for one point and a prediction is made for that point. As before, the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error (LOO-XVE) is good, but at first pass it seems very expensive to compute.



# Performance Measures

## Classification Accuracy

It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{\text{Number of Correct predictions}}{\text{Total number of predictions made}}$$

It works well only if there are equal number of samples belonging to each class. For example, if there are 95% samples of class A and 5% samples of class B in our training set. Then the model can easily get 95% training accuracy by simply predicting every training sample belonging to class A. When the same model is tested on a test set with 55% samples of class A and 45% samples of class B, then the test accuracy would drop down to 55%.

## Logarithmic Loss

Logarithmic Loss penalises the false classifications and it works well for multi-class classification. The classifier must assign probability to each class for all the samples. If there are N samples belonging to M classes, then the Log Loss is calculated as below :

$$LogarithmicLoss = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} * \log(p_{ij})$$

where  $y_{ij}$  indicates whether sample  $i$  belongs to class  $j$  or not and  $p_{ij}$  indicates the probability of sample  $i$  belonging to class  $j$

Log Loss has no upper bound and it exists on the range  $[0, \infty)$ . Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy.



## Confusion Matrix

Confusion Matrix gives us a matrix as output and describes the complete performance of the model.

There are 4 important terms :

- **True Positives** : The cases in which we predicted YES and the actual output was also YES.
- **True Negatives** : The cases in which we predicted NO and the actual output was NO.
- **False Positives** : The cases in which we predicted YES and the actual output was NO.
- **False Negatives** : The cases in which we predicted NO and the actual output was YES.

Accuracy for the matrix can be calculated by taking average of the values lying across the main diagonal i.e

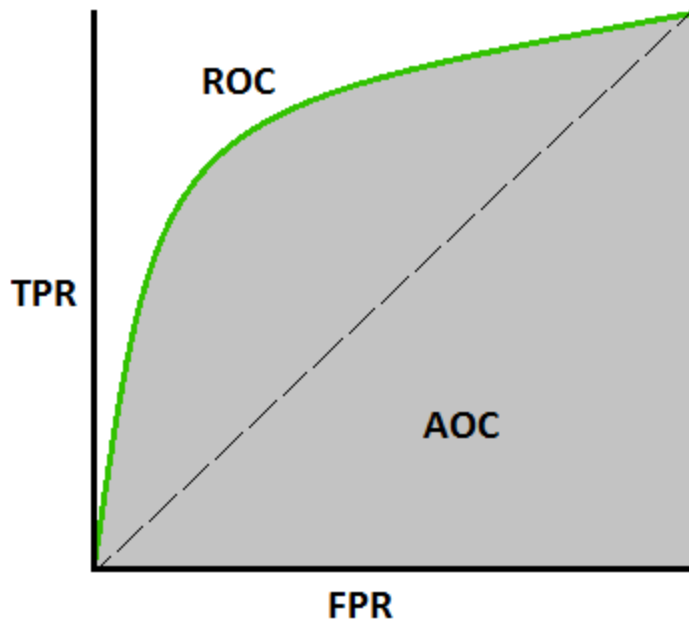
$$Accuracy = \frac{TruePositives + FalseNegatives}{TotalNumberofSamples}$$

## Area Under Curve

Area Under Curve(AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example. Before defining AUC, let us understand two basic terms :

- **True Positive Rate (Sensitivity)** : True Positive Rate is calculated by  $TP / (FN+TP)$ . True Positive Rate is the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. It has values in the range [0, 1].
- **False Positive Rate (Specificity)** : False Positive Rate is calculate by  $FP / (FP+TN)$  which means that it is the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points. It has values in the range [0, 1].

AUC is the area under the curve of plot False Positive Rate vs True Positive Rate at different points in [0, 1].



AUC also has a range of [0, 1] and greater the value, the better is the performance of our model.

## F1 Score

F1 Score is the harmonic mean(H.M.) between precision and recall. The range is [0, 1]. It depicts how precise the classifier is i.e. how many instances it classifies correctly and that it didn't miss a significant number of instances. The greater the F1 Score, the better is the performance of the model.

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

- **Precision** : It is the number of correct positive results divided by the number of positive results predicted by the classifier.
- **Recall** : It is the number of correct positive results divided by the number of all samples that should have been identified as positive.

## Mean Absolute Error

It is the average of the difference between the original values and the predicted values. It doesn't give us any idea of the direction of the error i.e. whether the model is under predicting or over predicting the data.

## Mean Squared Error

Mean Squared Error(MSE) is quite similar to Mean Absolute Error with the difference that MSE takes average of the square of the difference between the original values and the predicted values.

$$MeanSquaredError = \frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j)^2$$

**Advantage** — It is easier to compute the gradient, whereas MAE needs complicated linear programming tools to compute the gradient.

## Predicting Probabilities

In a classification problem, we may decide to predict the class values directly.

Alternately, it can be more flexible to predict the probabilities for each class instead. The reason for this is to provide the capability to choose and even calibrate the threshold for how to interpret the predicted probabilities.

For example, a default might be to use a threshold of 0.5, meaning that a probability in [0.0, 0.49] is a negative outcome (0) and a probability in [0.5, 1.0] is a positive outcome (1).

This threshold can be adjusted to tune the behavior of the model for a specific problem. An example would be to reduce more of one or another type of error.

When making a prediction for a binary or two-class classification problem, there are two types of errors that we could make.

- **False Positive.** Predict an event when there was no event.
- **False Negative.** Predict no event when in fact there was an event.

By predicting probabilities and calibrating a threshold, a balance of these two concerns can be chosen by the operator of the model.

For example, in a smog prediction system, we may be far more concerned with having low false negatives than low false positives. A false negative would mean not warning about a smog day when in fact it is a high smog day, leading to health issues in the public that are unable to take precautions. A false positive means the public would take precautionary measures when they didn't need to.

A common way to compare models that predict probabilities for two-class problems is to use a ROC curve.

## What Are ROC Curves?

A useful tool when predicting the probability of a binary outcome is the [Receiver Operating Characteristic curve](#), or ROC curve.

It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0. Put another way, it plots the false alarm rate versus the hit rate.

The true positive rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive.

$$\text{True Positive Rate} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$

The true positive rate is also referred to as sensitivity.

$$\text{Sensitivity} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$

The false positive rate is calculated as the number of false positives divided by the sum of the number of false positives and the number of true negatives.

It is also called the false alarm rate as it summarizes how often a positive class is predicted when the actual outcome is negative.

$$\text{False Positive Rate} = \text{False Positives} / (\text{False Positives} + \text{True Negatives})$$

The false positive rate is also referred to as the inverted specificity where specificity is the total number of true negatives divided by the sum of the number of true negatives and false positives.

$$\text{Specificity} = \text{True Negatives} / (\text{True Negatives} + \text{False Positives})$$

Where:

$$\text{False Positive Rate} = 1 - \text{Specificity}$$

The ROC curve is a useful tool for a few reasons:

- The curves of different models can be compared directly in general or for different thresholds.
- The area under the curve (AUC) can be used as a summary of the model skill.

The shape of the curve contains a lot of information, including what we might care about most for a problem, the expected false positive rate, and the false negative rate.

To make this clear:

- Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives.
- Larger values on the y-axis of the plot indicate higher true positives and lower false negatives.

If you are confused, remember, when we predict a binary outcome, it is either a correct prediction (true positive) or not (false positive). There is a tension between these options, the same with true negative and false negative.

A skilful model will assign a higher probability to a randomly chosen real positive occurrence than a negative occurrence on average. This is what we mean when we say that the model has skill. Generally, skilful models are represented by curves that bow up to the top left of the plot.

A no-skill classifier is one that cannot discriminate between the classes and would predict a random class or a constant class in all cases. A model with no skill is represented at the point (0.5, 0.5). A model with no skill at each threshold is represented by a diagonal line from the bottom left of the plot to the top right and has an AUC of 0.5.

A model with perfect skill is represented at a point (0,1). A model with perfect skill is represented by a line that travels from the bottom left of the plot to the top left and then across the top to the top right.

An operator may plot the ROC curve for the final model and choose a threshold that gives a desirable balance between the false positives and false negatives.

## ROC Curves and AUC in Python

We can plot a ROC curve for a model in Python using the [`roc\_curve\(\)`](#) scikit-learn function.

The function takes both the true outcomes (0,1) from the test set and the predicted probabilities for the 1 class. The function returns the false positive rates for each threshold, true positive rates for each threshold and thresholds.

```
...
```

```
# calculate roc curve
```

```
fpr, tpr, thresholds = roc_curve(y, probs)
```

The AUC for the ROC can be calculated using the [`roc\_auc\_score\(\)`](#) function.

Like the `roc_curve()` function, the AUC function takes both the true outcomes (0,1) from the test set and the predicted probabilities for the 1 class. It returns the AUC score between 0.0 and 1.0 for no skill and perfect skill respectively.

```
...
```

```
# calculate AUC
```

```
auc = roc_auc_score(y, probs)
```

```
print('AUC: %.3f' % auc)
```

A complete example of calculating the ROC curve and ROC AUC for a Logistic Regression model on a small test problem is listed below.

```
# roc curve and auc

from sklearn.datasets import make_classification

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import roc_curve

from sklearn.metrics import roc_auc_score

from matplotlib import pyplot

# generate 2 class dataset

X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)

# split into train/test sets

trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)

# generate a no skill prediction (majority class)

ns_probs = [0 for _ in range(len(testy))]

# fit a model

model = LogisticRegression(solver='lbfgs')

model.fit(trainX, trainy)

# predict probabilities

lr_probs = model.predict_proba(testX)

# keep probabilities for the positive outcome only

lr_probs = lr_probs[:, 1]

# calculate scores

ns_auc = roc_auc_score(testy, ns_probs)

lr_auc = roc_auc_score(testy, lr_probs)

# summarize scores

print('No Skill: ROC AUC=%.3f' % (ns_auc))
```

```
print('Logistic: ROC AUC=%.3f' % (lr_auc))

# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)

# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')

# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')

# show the legend
pyplot.legend()

# show the plot
pyplot.show()
```

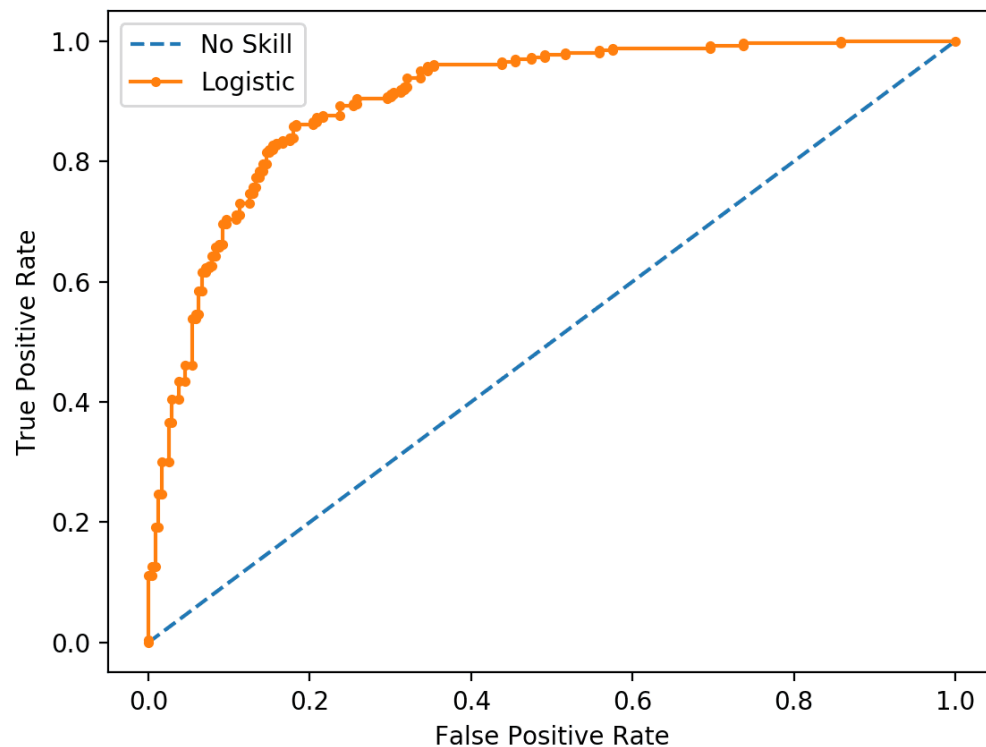
Running the example prints the ROC AUC for the logistic regression model and the no skill classifier that only predicts 0 for all examples.

No Skill: ROC AUC=0.500

Logistic: ROC AUC=0.903

A plot of the ROC curve for the model is also created showing that the model has skill.

**Note:** Your [results may vary](#) given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.



ROC Curve Plot for a No Skill Classifier and a Logistic Regression Model

## What Are Precision-Recall Curves?

There are many ways to evaluate the skill of a prediction model.

An approach in the related field of [information retrieval](#) (finding documents based on queries) measures [precision and recall](#).

These measures are also useful in applied machine learning for evaluating binary classification models.

Precision is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class. Precision is referred to as the positive predictive value.

$$\text{Positive Predictive Power} = \text{True Positives} / (\text{True Positives} + \text{False Positives})$$

or

$$\text{Precision} = \text{True Positives} / (\text{True Positives} + \text{False Positives})$$



Recall is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as sensitivity.

$$\text{Recall} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$

or

$$\text{Sensitivity} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$

Recall == Sensitivity

Reviewing both precision and recall is useful in cases where there is an imbalance in the observations between the two classes. Specifically, there are many examples of no event (class 0) and only a few examples of an event (class 1).

The reason for this is that typically the large number of class 0 examples means we are less interested in the skill of the model at predicting class 0 correctly, e.g. high true negatives.

Key to the calculation of precision and recall is that the calculations do not make use of the true negatives. It is only concerned with the correct prediction of the minority class, class 1.

A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve.

A no-skill classifier is one that cannot discriminate between the classes and would predict a random class or a constant class in all cases. The no-skill line changes based on the distribution of the positive to negative classes. It is a horizontal line with the value of the ratio of positive cases in the dataset. For a balanced dataset, this is 0.5.

While the baseline is fixed with ROC, the baseline of [precision-recall curve] is determined by the ratio of positives (P) and negatives (N) as  $y = P / (P + N)$ . For instance, we have  $y = 0.5$  for a balanced class distribution ...

— [The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets](#), 2015.

A model with perfect skill is depicted as a point at (1,1). A skilful model is represented by a curve that bows towards (1,1) above the flat line of no skill.

There are also composite scores that attempt to summarize the precision and recall; two examples include:

- **F-Measure** or [F1 score](#): that calculates the harmonic mean of the precision and recall ([harmonic mean](#) because the precision and recall are rates).
- **Area Under Curve**: like the AUC, summarizes the integral or an approximation of the area under the precision-recall curve.

In terms of model selection, F-Measure summarizes model skill for a specific probability threshold (e.g. 0.5), whereas the area under curve summarize the skill of a model across thresholds, like ROC AUC.

This makes precision-recall and a plot of precision vs. recall and summary measures useful tools for binary classification problems that have an imbalance in the observations for each class.

## Precision-Recall Curves in Python

Precision and recall can be calculated in scikit-learn.

The precision and recall can be calculated for thresholds using the [precision\\_recall\\_curve\(\)](#) function that takes the true output values and the probabilities for the positive class as input and returns the precision, recall and threshold values.

...

```
# calculate precision-recall curve
```

```
precision, recall, thresholds = precision_recall_curve(testy, probs)
```

The F-Measure can be calculated by calling the [f1\\_score\(\)](#) function that takes the true class values and the predicted class values as arguments.

...

```
# calculate F1 score
```

```
f1 = f1_score(testy, yhat)
```

The area under the precision-recall curve can be approximated by calling the [auc\(\)](#) function and passing it the recall (x) and precision (y) values calculated for each threshold.

...

```
# calculate precision-recall AUC
```

```
auc = auc(recall, precision)
```

When plotting precision and recall for each threshold as a curve, it is important that recall is provided as the x-axis and precision is provided as the y-axis.

The complete example of calculating precision-recall curves for a Logistic Regression model is listed below.

```
# precision-recall curve and f1

from sklearn.datasets import make_classification

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import precision_recall_curve

from sklearn.metrics import f1_score

from sklearn.metrics import auc

from matplotlib import pyplot

# generate 2 class dataset

X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)

# split into train/test sets

trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)

# fit a model

model = LogisticRegression(solver='lbfgs')

model.fit(trainX, trainy)

# predict probabilities

lr_probs = model.predict_proba(testX)

# keep probabilities for the positive outcome only

lr_probs = lr_probs[:, 1]

# predict class values

yhat = model.predict(testX)

lr_precision, lr_recall, _ = precision_recall_curve(testy, lr_probs)

lr_f1, lr_auc = f1_score(testy, yhat), auc(lr_recall, lr_precision)

# summarize scores

print('Logistic: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))

# plot the precision-recall curves

no_skill = len(testy[testy==1]) / len(testy)
```

```
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')

pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic')

# axis labels

pyplot.xlabel('Recall')

pyplot.ylabel('Precision')

# show the legend

pyplot.legend()

# show the plot

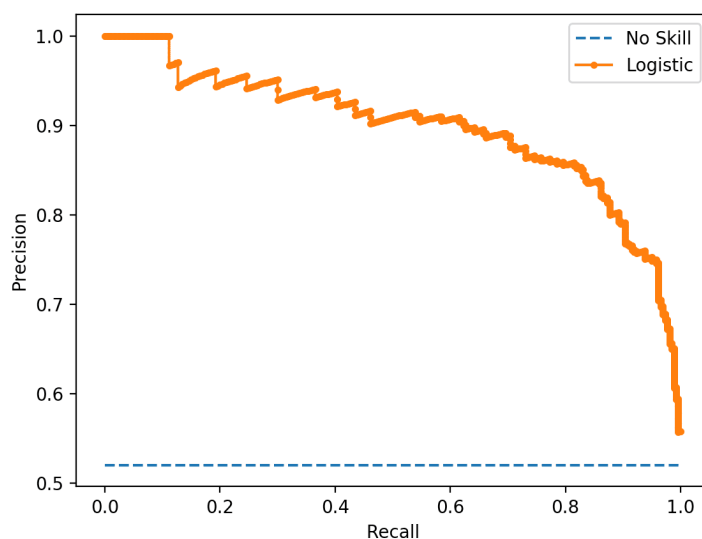
pyplot.show()
```

Running the example first prints the F1, area under curve (AUC) for the logistic regression model.

**Note:** Your [results may vary](#) given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Logistic: f1=0.841 auc=0.898

The precision-recall curve plot is then created showing the precision/recall for each threshold for a logistic regression model (orange) compared to a no skill model (blue).



## Precision-Recall Plot for a No Skill Classifier and a Logistic Regression Model

### When to Use ROC vs. Precision-Recall Curves?

Generally, the use of ROC curves and precision-recall curves are as follows:

- ROC curves should be used when there are roughly equal numbers of observations for each class.
- Precision-Recall curves should be used when there is a moderate to large class imbalance.

The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance.

### Multi-Class Classification

[Multi-class classification](#) refers to those classification tasks that have more than two class labels.

Examples include:

- Face classification.
- Plant species classification.
- Optical character recognition.

Unlike binary classification, multi-class classification does not have the notion of normal and abnormal outcomes. Instead, examples are classified as belonging to one among a range of known classes.

The number of class labels may be very large on some problems. For example, a model may predict a photo as belonging to one among thousands or tens of thousands of faces in a face recognition system.

Problems that involve predicting a sequence of words, such as text translation models, may also be considered a special type of multi-class classification. Each word in the sequence of words to be predicted involves a multi-class classification where the size of the vocabulary defines the number of possible classes that may be predicted and could be tens or hundreds of thousands of words in size.

It is common to model a multi-class classification task with a model that predicts a [Multinoulli probability distribution](#) for each example.

The Multinoulli distribution is a discrete probability distribution that covers a case where an event will have a categorical outcome, e.g.  $K$  in  $\{1, 2, 3, \dots, K\}$ . For classification, this means that the model predicts the probability of an example belonging to each class label.

Many algorithms used for binary classification can be used for multi-class classification.

Popular algorithms that can be used for multi-class classification include:

- k-Nearest Neighbors.
- Decision Trees.
- Naive Bayes.
- Random Forest.
- Gradient Boosting.

Algorithms that are designed for binary classification can be adapted for use for multi-class problems.

This involves using a strategy of fitting multiple binary classification models for each class vs. all other classes (called one-vs-rest) or one model for each pair of classes (called one-vs-one).

- **One-vs-Rest:** Fit one binary classification model for each class vs. all other classes.
- **One-vs-One:** Fit one binary classification model for each pair of classes.

Binary classification algorithms that can use these strategies for multi-class classification include:

- Logistic Regression.
- Support Vector Machine.

Next, let's take a closer look at a dataset to develop an intuition for multi-class classification problems.

## Multi-Label Classification

[Multi-label classification](#) refers to those classification tasks that have two or more class labels, where one or more class labels may be predicted for each example.

Consider the example of [photo classification](#), where a given photo may have multiple objects in the scene and a model may predict the presence of multiple known objects in the photo, such as “*bicycle*,” “*apple*,” “*person*,” etc.

This is unlike binary classification and multi-class classification, where a single class label is predicted for each example.

It is common to model multi-label classification tasks with a model that predicts multiple outputs, with each output taking predicted as a Bernoulli probability distribution. This is essentially a model that makes multiple binary classification predictions for each example.

Classification algorithms used for binary or multi-class classification cannot be used directly for multi-label classification. Specialized versions of standard classification algorithms can be used, so-called multi-label versions of the algorithms, including:

- Multi-label Decision Trees
- Multi-label Random Forests
- Multi-label Gradient Boosting

Another approach is to use a separate classification algorithm to predict the labels for each class.

Next, let's take a closer look at a dataset to develop an intuition for multi-label classification problems.

We can use the [make\\_multilabel\\_classification\(\) function](#) to generate a synthetic multi-label classification dataset.

In multi-output classification, the goal is to learn a classification rule whose output is a set, or vector, of labels i.e.  $y_1$  belong to  $Y_1$ ,  $y_2$  belonging to  $Y_2$ ,  $y_n$  belonging to  $Y_n$  and  $v$ , the vector is composed of  $y_1, y_2, y_3, \dots, y_n$ .

Multi-label classification involves classifying instances into several labels that share semantics, for example, the problem of classifying songs according to their genre — it is possible to classify a dance as either ballet or traditional, but also possible to classify a dance as both (i.e. "ballet traditional"). Here, pop and rock share semantics: they both relate to the songs' genre, and are thus two different values of the same label. There is also no a priori knowledge regarding the size of the output — it is very possible that a song cannot be classified as any previously known genre, or that it is best classified as several different genres.

The problem of multi-output classification is effectively the opposite of multi-label classification — the output values do not share semantics, but the number of outputs is known a priori.

In a classification problem where the goal is to simultaneously predict temperature (low, medium, or high) and pressure (low, medium, or high) inside a pressure cooker — in this case, the model is expected to output exactly two values, one value for the temperature label and another for the pressure label.

The machine learning task of solving a multi-output problem thus involves building a predictive model that simultaneously outputs a set of (two or more) labels that measure different concepts — essentially two or more separate (although related) classification problems are solved concurrently within the same model.

A multi-output classification is multitask-classification — which illustrates the fact that a multi-output classification problem is effectively equivalent to multiple simultaneous (multi-tasked) single-label classification problems.

Multi-label problems can be transformed into multi-output problems, the opposite is not necessarily true.

We all know how to predict one target column given multiple feature columns, let's see how to predict two columns at once.

**Multi- output :** Yes, there will be multiple outputs (2 or more) for a single feature set( a set of independent values)

Many times beginners get confused between MultiClass and MultiLabel.

**Multi-Class :** Each data point can only belong to one label. For example: A fraud detection model can only classify one feature set into either “fraud” or “non fraud”. It can’t be both or there’s no middle ground.

**Multi-Label:** One data point can belong to one or more labels. For example: when building movie genre prediction model, the model can classify one movie into more than one label, since a movie can be action, thriller ,can be both action and thriller.

	Number of targets	Target cardinality	Valid type_of_target
Multiclass classification	1	>2	‘multiclass’
Multilabel classification	>1	2 (0 or 1)	‘multilabel-indicator’
Multiclass-multioutput classification	>1	>2	‘multiclass-multioutput’
Multioutput regression	>1	Continuous	‘continuous-multioutput’

## GRADIENT DESCENT INTRODUCTION:

Gradient descent is, with no doubt, the heart and soul of most Machine Learning (ML) algorithms. I definitely believe that you should take the time to understanding it. Because once you do, for starters, you will better comprehend how most ML algorithms work. Besides, understanding basic concepts is key for developing intuition about more complicated subjects.

To understand Gradient Descent at its heart, let’s have a running example. The task is an old one in the field — predict house prices using some historical data as prior knowledge.

But our goal here is to talk about Gradient Descent. To do that, let’s make the example simple enough so we can concentrate on the good parts.

But, Before we go ahead, you can get the code [here](#).

## Basic Concepts

Suppose you want to climb a very tall hill. Your goal is to get to the top of the hill the fastest. You look around and you realize you have more than one path to start off. Since you are on the bottom, all of these options seem to take you somewhat closer to the summit.

But you want to get to the top in the fastest way possible. So, how can you do that? How can you take a step that takes you as close as possible to the summit?

Up to this point, it is not clear how to take this step. That is where the Gradient can help you.

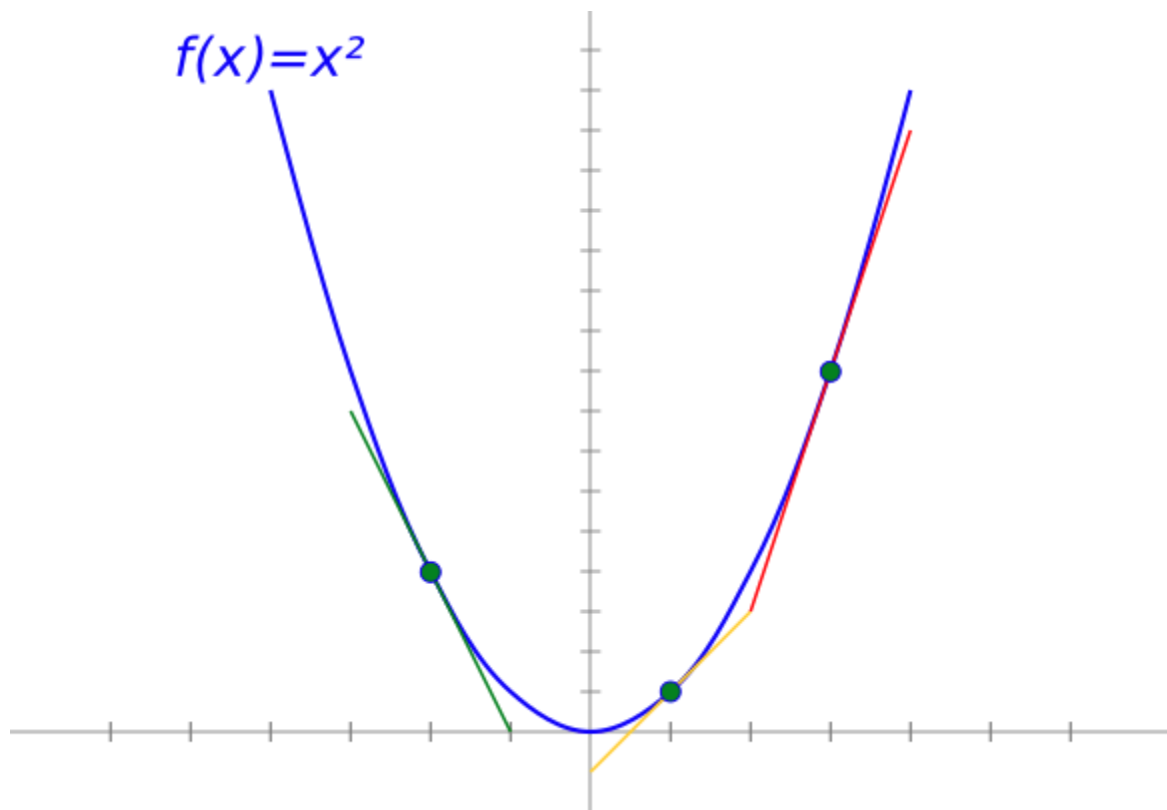


As stated in this Khan Academy [video](#), the gradient captures all the partial derivatives of a multi-variable function.

Let's go step by step and see how it works.

In simpler terms, the derivative is the rate of change or the slope of a function at a given point.

Take the  $f(x) = x^2$  function as an example. The derivative of  $f(x)$ , is another function  $f'(x)$  that computes the slope of  $f(x)$  at a given point  $x$ . In this situation, for  $x = 2$ , the slope of  $f(x) = x^2$  is  $2x$  or  $2 \cdot 2 = 4$ .



The slope of  $f(x) = x^2$  at different points.

Simply putting, the derivative points to the direction of **steepest ascent**. And the good thing is, the gradient is exactly the same thing. With one exception, the Gradient is a vector-valued function that stores partial derivatives. In other words, the gradient is a vector, and each of its components is a partial derivative with respect to one specific variable.

Take the function,  $f(x, y) = 2x^2 + y^2$  as another example.

Here,  $f(x, y)$  is a multi-variable function. Its gradient is a vector, containing the partial derivatives of  $f(x, y)$ . The first with respect to  $x$ , and the second with respect to  $y$ .

If we calculate the partials of  $f(x, y)$  we get.

$$\frac{\partial f}{\partial x} = 4x$$

$$\frac{\partial f}{\partial y} = 2y$$

So the gradient is the following vector:

$$\nabla f(x, y) = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]^T$$

$$\nabla f(x, y) = [4x, 2y]^T$$

Note that each component indicates what is the direction of steepest ascent for each of the function' variables. Put it differently, the gradient points to the direction where the function increases the most.

Back to the hill climbing example, the gradient points you to the direction that takes you to the peak of the mountain the fastest. In other words, the gradient points to the higher altitudes of a surface.

In the same way, if we get a function with 4 variables, we would get a gradient vector with 4 partial derivatives. Generally, an  $n$ -variable function results in an  $n$ -dimensional gradient vector.

$$f(x_1, x_2, \dots, x_n) = \nabla f = \left[ \frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \dots, \frac{\partial}{\partial x_n} \right]$$

For Gradient descent, however, we do not want to maximize  $f$  as fast as we can, we want to **minimize** it.

But let's define our task first and things will look much cleaner.

## Predicting House Prices

We are going to solve the problem of predicting house prices based on historical data. To build a Machine Learning model, we often need at least 3 things. A problem  $T$ , a performance measure  $P$ , and an experience  $E$ , from where our model will learn patterns from.

To solve task  $T$ , we are going to use a simple Linear Regression Model. This model will learn from experience  $E$ , and after training, it will be able to generalize its knowledge to unseen data.

The Linear Model is an excellent model to learn. It's the foundation of many other ML algorithms like Neural Networks and Support Vector Machines.

For this example, the experience  $E$ , is the HOUSES Dataset. The HOUSES dataset contains a collection of recent real estate listings in San Luis Obispo County and around it.

The collection contains 781 data records and it is available for download in CSV format [here](#). Among the 8 available features, for simplicity, we are going to focus on only two of them: the Size, and Price. For each of the 781 records, the Size, in square feet, will be our input features, and the Price our target values.

Besides, to check if our model is properly learning from experience  $E$ , we need a mechanism to measure its performance. To do that, we take the mean of squared errors (MSE) as our performance measure.

$$Err = \frac{1}{N} \sum_{\mu} (\overset{\text{Ground truth}}{y^{\mu}} - \underbrace{(W_1 X^{\mu} + W_0)}_{\text{Predicted values}})^2$$

---

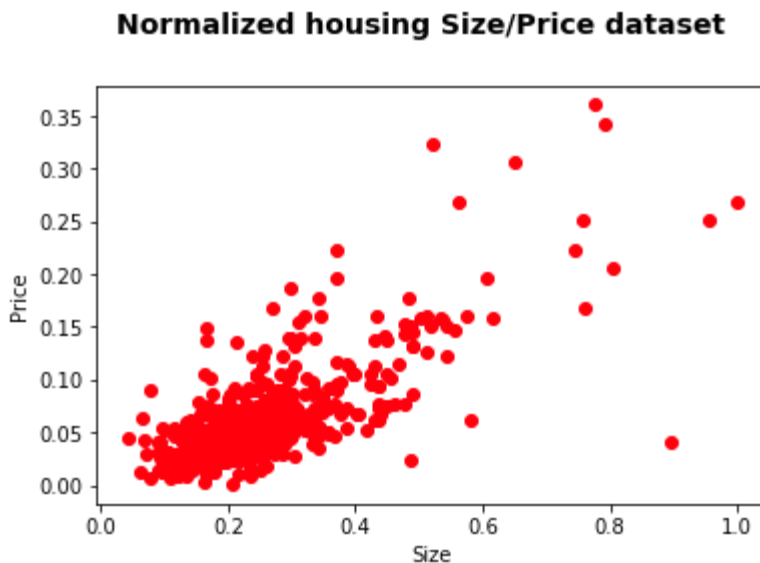
**MSE**

MSE has been a standard for linear regression for many years. But in theory, any other error measure like the Absolute Error would work. Some of the benefits of MSE is that it penalizes larger errors more than the Absolute error.

Now that we have formalized our learning algorithm, let's dive into the code.

First we load the data in python using Pandas, and separate the Size and Prices features. After, we normalize the data to prevent some of the features to out value some of the others. Also, Gradient Descent is known for converging much faster with normalized data than otherwise.

Bellow, you can see the distribution of house prices by its size in square meters.



Distribution of house prices by Size. The data is normalized to the [0,1] interval.

A Linear Regression model works by drawing a line on the data. As such, our model is represented by a simple line equation.

$$y = mx + b$$

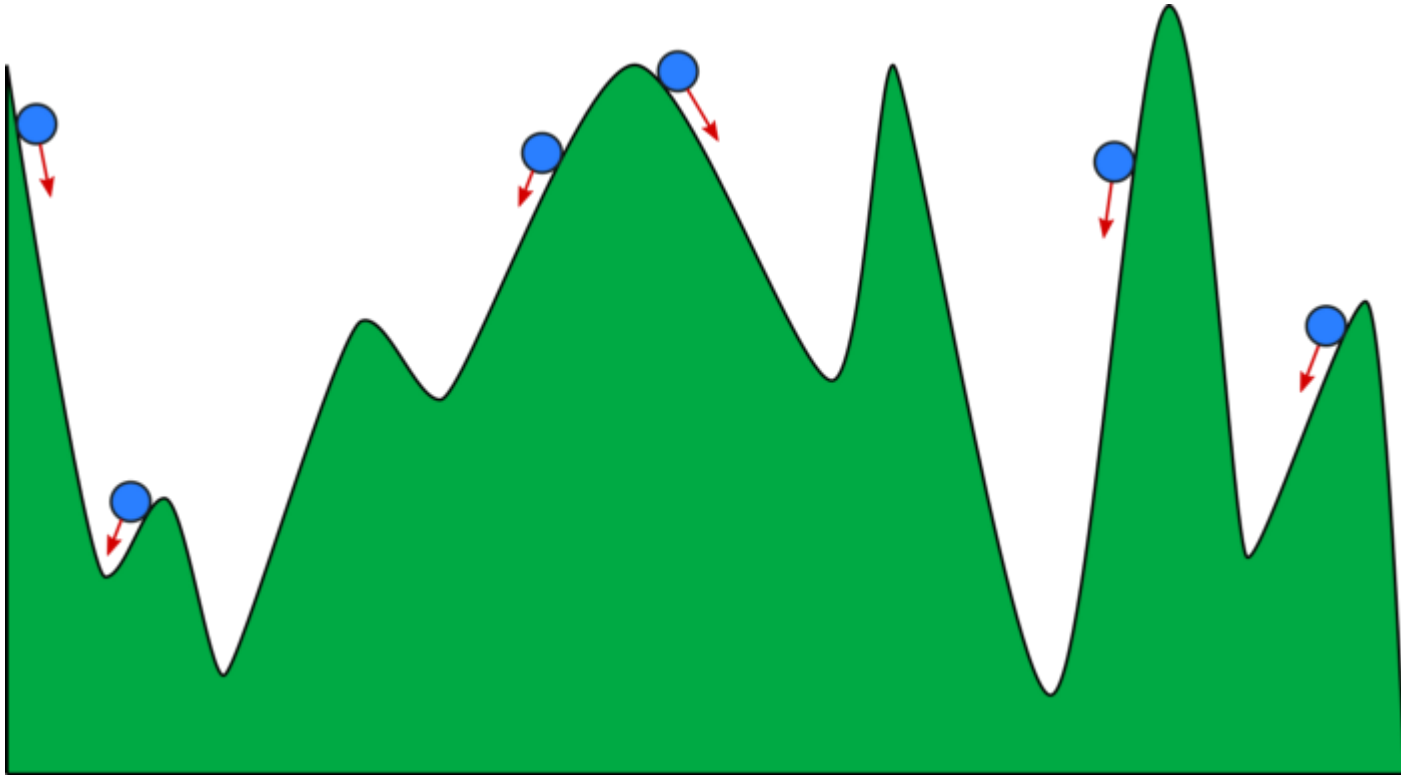
Line equation.  $m$  and  $b$  are the slope and the y-intercept respectively. The  $x$  variable is the placeholder for the input values.

For a Linear Model, the two free parameters are the slope  $m$  and the y-intercept  $y$ . These two variables are the knobs that we are going to change in order to find the best line equation.

Iteratively, we are going to perform slight changes to them, so it can follow the direction of steepest descent on the error surface. After each iteration, these weight changes will refine our model so that it can represent the trends of the dataset.

Before going further, remember that for Gradient Descent, we want to take the direction opposite to the gradient.

You can think of Gradient Descent as a ball rolling down on a valley. we want it to sit in the deepest place of the mountains, however, it is easy to see that things can go wrong.



In analogy, we can think of Gradient Descent as being a ball rolling down on a valley. The deepest valley is the optimal global minimum and that is the place we aim for.

Depending on where the ball starts rolling, it may rest in the bottom of a valley. But not in the lowest one. This is called a local minimum and in the context of our model, the valley is the error surface.

Note that, in the analogy, not all local minima are bad. Some of them are actually almost as low (good) as the lowest (global) one. In fact, for high dimensional error surfaces, it is most common to settle in one of these (not so bad) local minima.

Similarly, the way we initialize our model weights may lead it to rest in a local minimum. To avoid that, we initialize the two weight vectors with values from a random normal distribution with zero mean and low variance.

At each iteration, we are going to take a random subset of our dataset and linearly combine it with our weights. This subset is called a *mini-batch*. After the linear combination, we feed the resulting vector in the MSE function to calculate the current error.

With this error signal, we can calculate the partial derivatives of the error and get the Gradient.

First, we get the partial derivative with respect to  $W_0$ .

$$\frac{\partial}{\partial W_0} = -\frac{2}{N} \sum_{\mu} (y^{\mu} - (W_1 X^{\mu} + W_0))$$

Partial with respect to  $W_0$

Second, we do the same, but taking  $W_1$  as the actor.

$$\frac{\partial}{\partial W_1} = -\frac{2}{N} \sum_{\mu} X \times (y^{\mu} - (W_1 \times X^{\mu} + W_0))$$

Partial with respect to  $W_1$

With the two partials, we have the gradient vector:

$$\nabla Err = \left[ \frac{\partial}{\partial W_0}, \frac{\partial}{\partial W_1} \right]^T$$

The Gradient

Where  $Err$  is the MSE error function.

Having that, our next step is to update the weight vectors  $W_0$  and  $W_1$ , using the gradients, to minimize the error.

We want to update the weights so they can push the error down in the next iteration. We need to make them follow the opposite direction of each respective gradient signal. To do that, we are going to take small steps of size  $\eta$  in that direction.

The step size  $\eta$  is the learning rate and it controls the rate of learning. Empirically, a good starting point is 0.1. In the end, the update step rule is set as:

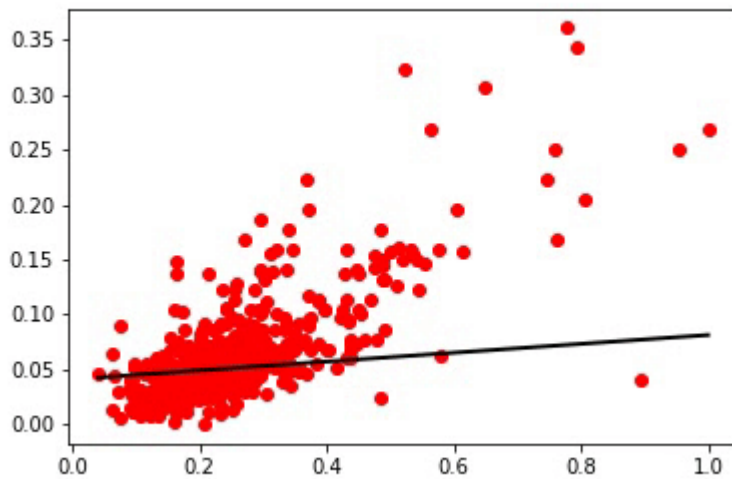
$$W_0 = W_0 - \eta \left( \frac{\partial}{\partial W_0} \right)$$

$$W_1 = W_1 - \eta \left( \frac{\partial}{\partial W_1} \right)$$

In code, the complete model looks like this. Look at the **minus sign** in front of both gradients  $DW_0$  and  $DW_1$ . This guarantees that we will take steps in the opposite direction to the gradient.

After updating the weights, we repeat the process with another random mini-batch. And that is it.

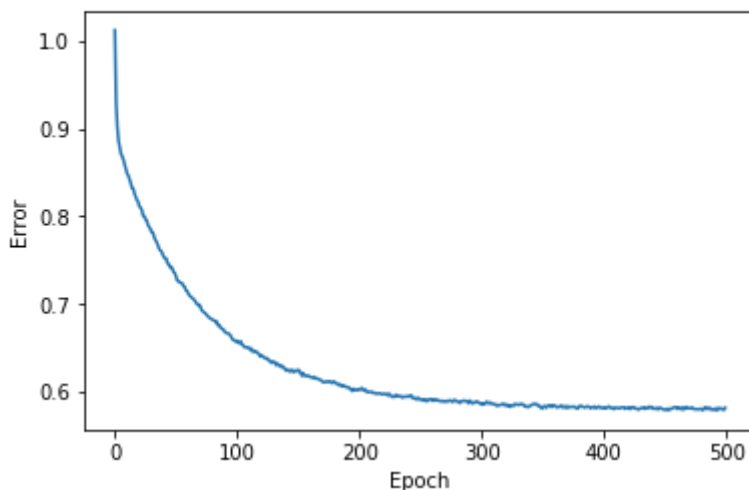
Step by step, each weight update causes a small shift in the line towards its best representation. In the end, when the error variance is small enough we can stop learning.



Linear model conversion over time. The first weight updates cause the line to rapidly reach an ideal representation.

This version of Gradient Descent is called Mini-Batch Stochastic Gradient Descent. In this version, we use a small subset of the training data to calculate the gradient. Each mini-batch gradient offers an approximation to the optimal direction. Even though the gradients do not point to the exact direction, in practice it converges to very good solutions.

**Epoch/Error graph**



Error signal by epoch. Note that after decreasing the error signal very fast the model slows down and converges.

If you look close at the error/episode graph, you notice that in the beginning, learning occurs at a faster pace.

After some epochs, however, it slows down and plateaus. This happens because, in the beginning, the gradient vectors that point to the steepest descent are long in magnitude. As result, the two weight variables  $W_0$  and  $W_1$  suffer more drastic changes.

However, as they get closer to the summit of the error surface, the gradient slowly gets smaller and smaller, which causes very small changes to the weights.



## UNIT-IV

Unsupervised Learning: clustering algorithms, k-means/k-medoid, Fuzzy K-means, hierarchical clustering, top- down, bottom-up: single-linkage, multiple linkage, dimensionality reduction, principal component analysis.

### Unsupervised learning

**Unsupervised learning** is a type of machine learning where the algorithm is given data without explicit instructions on what to do with it. The system tries to learn the patterns and relationships within the data on its own. There are two main types of unsupervised learning: clustering and dimensionality reduction.

#### 1. **Clustering:**

- **K-Means Clustering:** This algorithm partitions data into 'k' clusters based on similarity. For example, in customer segmentation, you can use K-means to group customers with similar purchasing behavior.
- **Hierarchical Clustering:** It creates a tree of clusters, where the root is a single cluster containing all data points, and the leaves are individual data points. This can be used in taxonomy or gene expression analysis.

#### 2. **Dimensionality Reduction:**

- **Principal Component Analysis (PCA):** PCA is used to reduce the number of features in a dataset while retaining its essential information. It's often applied in image compression or feature extraction for machine learning models.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE):** t-SNE is used for visualizing high-dimensional data in two or three dimensions. It's useful for exploring the structure of data and finding patterns. For instance, visualizing similarities between words in natural language processing.

#### 3. **Association Rule Learning:**

- **Apriori Algorithm:** This algorithm is used to discover associations between different items in a dataset. For example, in retail, it can identify relationships like "Customers who buy product A are likely to buy product B."

#### 4. **Generative Models:**

- **Generative Adversarial Networks (GANs):** GANs consist of a generator and a discriminator that are trained together. GANs can generate new data instances that resemble the training data. They are used in image and video generation tasks.

#### 5. **Autoencoders:**

- **Variational Autoencoders (VAE):** VAEs are a type of autoencoder that learns a probabilistic mapping between the data space and a latent space. They are used for generating new data points and can be applied to image and text generation.

Unsupervised learning is particularly valuable when you have a large amount of unlabeled data and want to explore the underlying structure or relationships within it. It is widely used in various domains, including pattern recognition, anomaly detection, and feature learning.

## Clustering algorithms

**Clustering algorithms** in unsupervised learning aim to group similar data points together into clusters or segments based on certain criteria. The goal is to discover hidden patterns or structures within the data. There are various clustering algorithms, each with its own approach and characteristics. Here are a few commonly used clustering algorithms:

#### 1. **K-Means Clustering:**

- **Objective:** Partition the data into 'k' clusters based on similarity.
- **Process:** It starts by randomly selecting 'k' centroids (cluster centers) and assigns each data point to the nearest centroid. Then, it recalculates the centroids based on the mean of the data points in each cluster. This process iterates until convergence.
- **Example:** Customer segmentation in marketing based on purchasing behavior.

#### 2. **Hierarchical Clustering:**

- **Objective:** Create a tree-like structure (dendrogram) of clusters.
- **Process:** It begins with each data point as a separate cluster and merges the closest clusters iteratively until all points belong to a single cluster. The resulting dendrogram can be cut at different levels to obtain clusters of varying sizes.
- **Example:** Taxonomy in biology or organizational hierarchy.

#### 3. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):**

- **Objective:** Identify clusters based on the density of data points.
- **Process:** It defines clusters as dense regions separated by areas of lower point density. It classifies points as core points, border points, or outliers (noise) based on their density and proximity to other points.
- **Example:** Anomaly detection in network traffic where unusual patterns represent potential security threats.

#### 4. **Mean Shift:**

- **Objective:** Discover modes or peaks of high-density regions.
- **Process:** It iteratively shifts the center of a kernel until it converges to a high-density region. The algorithm is adaptive and does not require specifying the number of clusters beforehand.
- **Example:** Image segmentation where pixels with similar colors form clusters.

### 5. Agglomerative Clustering:

- **Objective:** Build clusters by successively merging or agglomerating data points.
- **Process:** It starts with each data point as a singleton cluster and merges the closest pairs iteratively until a stopping criterion is met. The result is a dendrogram that can be cut to form clusters.
- **Example:** Social network analysis to identify communities within a network.

### 6. Gaussian Mixture Model (GMM):

- **Objective:** Model the data as a mixture of several Gaussian distributions.
- **Process:** It assumes that the data is generated by a mixture of several Gaussian distributions. The algorithm estimates the parameters of these distributions, including means and covariances, to identify clusters.
- **Example:** Speech and handwriting recognition where multiple patterns contribute to the observed data.

## K-Means algorithm

The **K-Means algorithm** is a popular unsupervised machine learning algorithm used for clustering data. The goal of K-Means is to partition a dataset into 'k' clusters, where each data point belongs to the cluster with the nearest mean. Here's a detailed explanation of the K-Means algorithm:

### Key Concepts:

- **Unsupervised Learning:** It works with unlabeled data, finding patterns on its own.
- **Clustering:** It groups similar data points together into distinct clusters.
- **Centroids:** Each cluster has a central point (centroid), representing the "average" of data points within it.
- **Distance Metric:** It measures how close data points are to each other, usually using Euclidean distance.

### Algorithm Steps:

#### Step 1: Initialization

- **Input:** Dataset with 'n' data points and the desired number of clusters 'k'.
- **Process:**
  - Randomly select 'k' data points from the dataset as initial centroids.

#### Step 2: Assignment

- **Input:** Initial centroids.
- **Process:**
  - For each data point in the dataset, calculate the Euclidean distance to each centroid.
  - Assign the data point to the cluster associated with the nearest centroid.

$$j = \arg \min_k \|x_i - c_k\|^2$$

**Step 3: Update**

- **Input:** Assigned clusters.
- **Process:**
  - Recalculate the centroids of each cluster by computing the mean of all data points in that cluster.

$$c_k = \frac{1}{\text{number of points in cluster } k} \sum_{i \text{ in cluster } k} x_i$$

**Step 4: Convergence Check**

- **Input:** Updated centroids.
- **Process:**
  - Repeat the assignment and update steps iteratively until convergence.
  - Convergence occurs when the centroids no longer change significantly or after a fixed number of iterations.

**Step 5: Result**

- **Output:** Final clusters.
- **Process:**
  - Once convergence is reached, the algorithm stops, and each data point is assigned to a specific cluster.

**Pseudo code:**

1. Randomly initialize k centroids:  $c_1, c_2, \dots, c_k$
2. Repeat until convergence:
  - a. For each data point  $x_i$ , assign it to the cluster with the closest centroid:  
 $j = \operatorname{argmin}_k \|x_i - c_k\|^2$
  - b. Update centroids:  $c_k = (1/|\text{cluster } k|) * \sum x_i$  for all  $i$  in cluster  $k$

## Program 1: Implement a program to cluster a dataset using K-means clustering.

### Example 1: iris flowers

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
data = iris.data # Features only, not using target labels

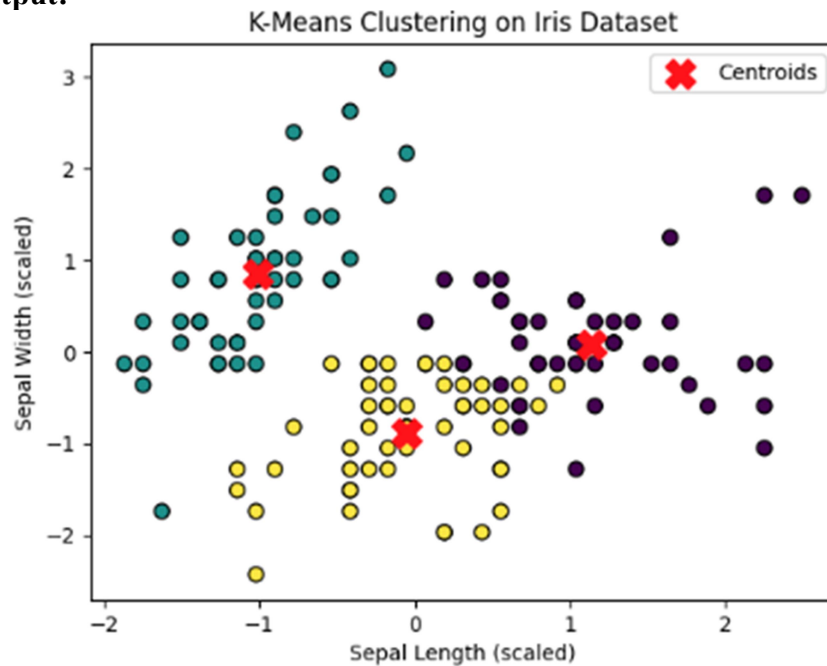
# Standardize the data (important for K-Means)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(data_scaled)

# Get cluster labels and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Visualize the clustering result
plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis', edgecolors='k', s=50)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200, color='red', label='Centroids')
plt.title('K-Means Clustering on Iris Dataset')
plt.xlabel('Sepal Length (scaled)')
plt.ylabel('Sepal Width (scaled)')
plt.legend()
plt.show()
```

Output:



## Example 2: Pima Indians Diabetes Database

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load the diabetes dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
column_names = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
diabetes_data = pd.read_csv(url, names=column_names)

# Select features for clustering (for example, Glucose and BMI)
selected_features = ["Glucose", "BMI"]
data_subset = diabetes_data[selected_features]

# Standardize the data (important for K-Means)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_subset)

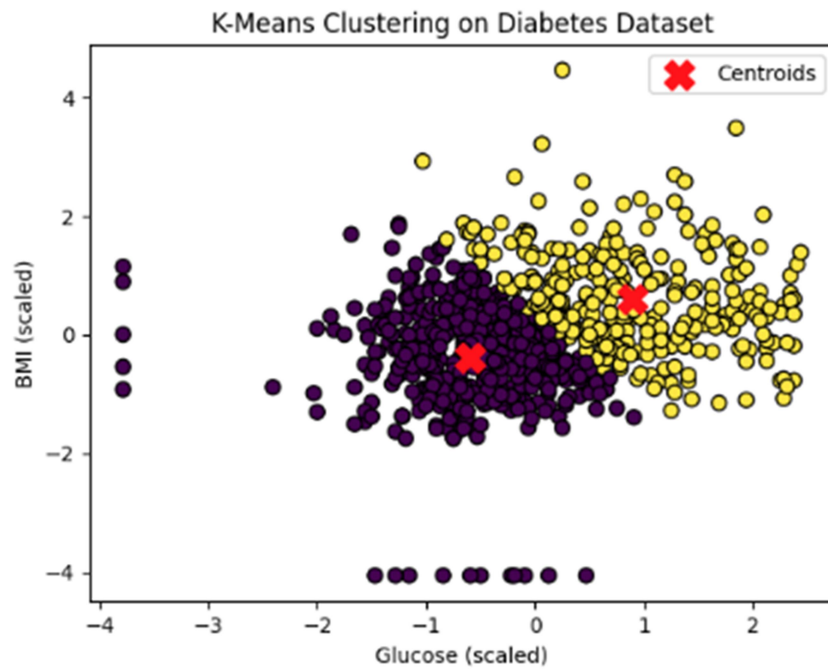
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(data_scaled)

# Get cluster labels and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Visualize the clustering result
plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis', edgecolors='k', s=50)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=200, color='red', label='Centroids')
plt.title('K-Means Clustering on Diabetes Dataset')
plt.xlabel('Glucose (scaled)')
plt.ylabel('BMI (scaled)')
plt.legend()
plt.show()
```

## Output:

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: warn()
```



### Example 3: Breast Cancer Wisconsin (Diagnostic) dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

# Load the breast cancer dataset
cancer = load_breast_cancer()
column_names = cancer.feature_names
cancer_data = pd.DataFrame(cancer.data, columns=column_names)

# Select features for clustering (for example, mean radius and mean texture)
selected_features = ["mean radius", "mean texture"]
data_subset = cancer_data[selected_features]

# Standardize the data (important for K-Means)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_subset)

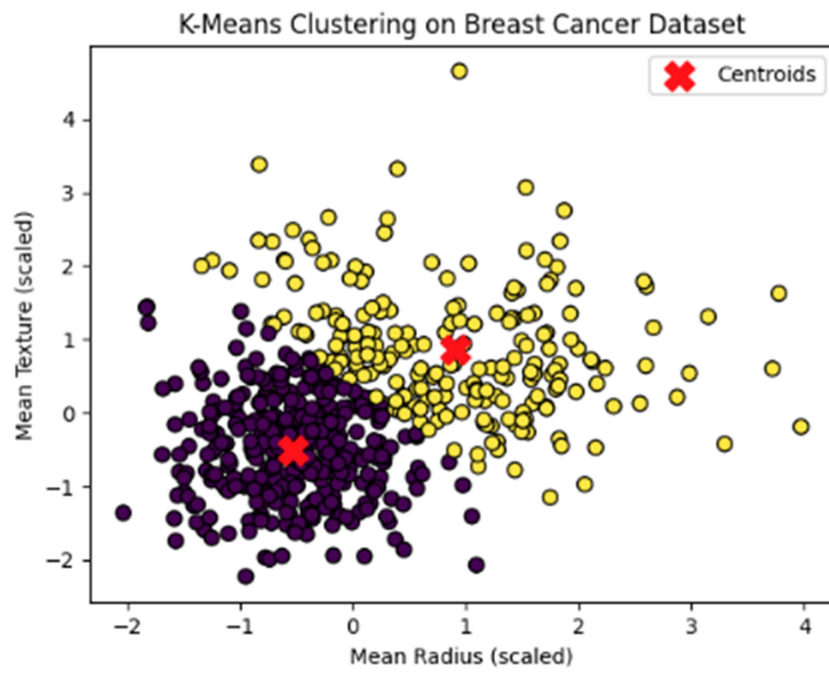
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(data_scaled)

# Get cluster labels and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Visualize the clustering result
plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=labels, cmap='viridis', edgecolors='k', s=50)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200, color='red', label='Centroids')
plt.title('K-Means Clustering on Breast Cancer Dataset')
plt.xlabel('Mean Radius (scaled)')
plt.ylabel('Mean Texture (scaled)')
plt.legend()
plt.show()
```



**Output:**



## Program 2: Implement a program to calculate the elbow method for K-means clustering.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

# Generate example data
np.random.seed(42)
X, _ = make_blobs(n_samples=300, centers=4, random_state=42, cluster_std=1.0)

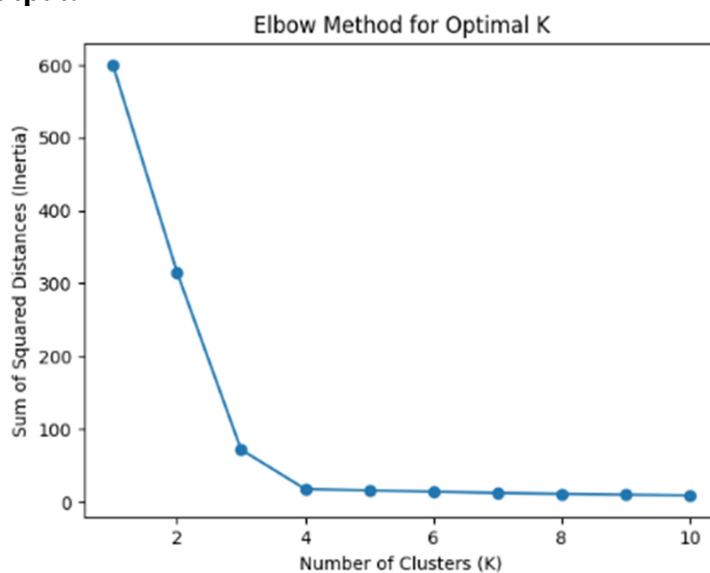
# Standardize the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Implement the elbow method
distortions = []
max_k = 10

for k in range(1, max_k + 1):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    distortions.append(kmeans.inertia_) # Inertia: Sum of squared distances to the closest centroid

# Plot the elbow method graph
plt.plot(range(1, max_k + 1), distortions, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Sum of Squared Distances (Inertia)')
plt.show()
```

### Output:



**Program 3: Implement a program to calculate the silhouette coefficient for K-means clustering.**

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Generate example data
np.random.seed(42)
X, _ = make_blobs(n_samples=300, centers=4, random_state=42, cluster_std=1.0)

# Standardize the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Calculate silhouette scores for different values of 'k'
max_k = 10
silhouette_scores = []

for k in range(2, max_k + 1):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    labels = kmeans.labels_
    silhouette_avg = silhouette_score(X_scaled, labels)
    silhouette_scores.append(silhouette_avg)

# Find the optimal 'k' with the highest silhouette score
optimal_k = np.argmax(silhouette_scores) + 2 # Add 2 because we started from k=2
optimal_score = silhouette_scores[optimal_k - 2] # Subtract 2 to get the correct index

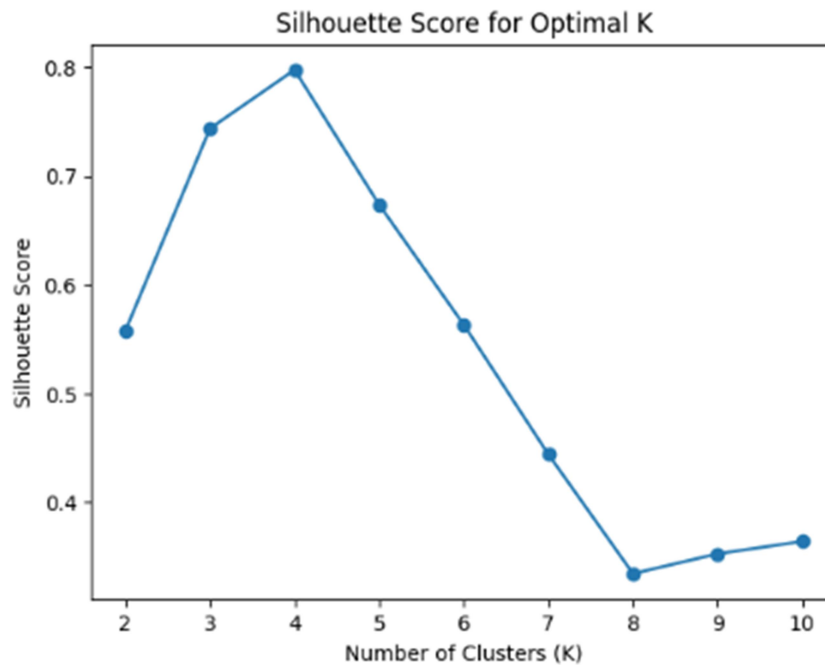
# Print the results
print(f"Optimal number of clusters (k): {optimal_k}")
print(f"Silhouette score for optimal k: {optimal_score}")

# Plot the silhouette scores
import matplotlib.pyplot as plt

plt.plot(range(2, max_k + 1), silhouette_scores, marker='o')
plt.title('Silhouette Score for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.show()
```

**Output:**

```
Optimal number of clusters (k): 4  
Silhouette score for optimal k: 0.7974739889632732
```

**Example:**

Imagine a dataset of customer spending habits. K-means could group customers into clusters based on similar spending patterns, helping businesses tailor marketing strategies.

**Key Points:**

- Simple and Efficient: K-means is popular due to its simplicity and efficiency.
- Sensitivity to Initialization: It can get stuck in local optima, so multiple runs with different initializations are often recommended.
- Non-Spherical Clusters and Outliers: It struggles with non-spherical cluster shapes or outliers.
- Choosing k: Selecting the optimal number of clusters is often challenging.

**Applications:**

- Customer Segmentation: Identifying customer groups with similar characteristics.
- Image Compression: Grouping similar pixels for compression.
- Anomaly Detection: Finding unusual data points that don't fit into clusters.
- Document Clustering: Grouping text documents based on content similarity.
- Gene Expression Analysis: Identifying patterns in gene expression data.

## Hierarchical clustering

Hierarchical clustering is a method used in unsupervised machine learning to group similar data points into clusters in a hierarchical manner. It builds a tree-like structure, called a dendrogram, to represent the relationships between clusters. The two main approaches to hierarchical clustering are agglomerative (bottom-up) and divisive (top-down).

### Agglomerative Hierarchical Clustering:

1. **Start:** Treat each data point as a singleton cluster, and compute the pairwise distances between all clusters.
2. **Merge:** Find the two closest clusters and merge them into a new cluster. Update the distance matrix.
3. **Repeat:** Repeat step 2 until only a single cluster remains, forming a dendrogram.
4. **Dendrogram Interpretation:** The dendrogram can be cut at different heights to obtain clusters at different levels of granularity.

### Divisive Hierarchical Clustering:

1. **Start:** Treat the entire dataset as a single cluster.
2. **Split:** Identify the cluster with the maximum dissimilarity and split it into two clusters.
3. **Repeat:** Repeat step 2 until each data point forms a singleton cluster, creating a dendrogram.
4. **Dendrogram Interpretation:** The dendrogram can be cut at different heights to obtain clusters at different levels of granularity.

### Example:

Let's consider a small dataset for illustrative purposes:

Data Points: A, B, C, D, E

Distances:

- $\text{Dist}(A, B) = 2$
- $\text{Dist}(A, C) = 3$
- $\text{Dist}(A, D) = 4$
- $\text{Dist}(A, E) = 5$
- $\text{Dist}(B, C) = 1$
- $\text{Dist}(B, D) = 6$
- $\text{Dist}(B, E) = 7$
- $\text{Dist}(C, D) = 8$
- $\text{Dist}(C, E) = 9$
- $\text{Dist}(D, E) = 10$

### Agglomerative Hierarchical Clustering:

1. **Step 1:** Treat each data point as a singleton cluster:  $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}$ .
2. **Step 2:** Merge the closest clusters:  $\{A, B\}, \{C\}, \{D\}, \{E\}$  ( $\text{Dist}(A, B) = 2$ ).
3. **Step 3:** Merge the closest clusters:  $\{A, B\}, \{C, D\}, \{E\}$  ( $\text{Dist}(B, C) = 1$ ).
4. **Step 4:** Merge the closest clusters:  $\{A, B, C, D\}, \{E\}$  ( $\text{Dist}(C, D) = 8$ ).
5. **Step 5:** Merge the closest clusters:  $\{A, B, C, D, E\}$ .

#### Dendrogram Interpretation:

The dendrogram would show the hierarchy of merging clusters at different heights, and we can choose the height at which we want to cut the dendrogram to obtain clusters.

#### Divisive Hierarchical Clustering:

1. **Step 1:** Treat the entire dataset as a single cluster:  $\{A, B, C, D, E\}$ .
2. **Step 2:** Split the cluster into two:  $\{A, B, C\}, \{D, E\}$ .
3. **Step 3:** Split the cluster further:  $\{A, B\}, \{C\}, \{D, E\}$ .
4. **Step 4:** Split the cluster further:  $\{A\}, \{B\}, \{C\}, \{D, E\}$ .
5. **Step 5:** Split the cluster further:  $\{A\}, \{B\}, \{C\}, \{D\}, \{E\}$ .

#### Dendrogram Interpretation:

Similar to agglomerative clustering, the dendrogram in divisive clustering represents the hierarchy of splitting clusters.

In practice, the choice of distance metric and linkage criteria (how to measure the distance between clusters) influences the results of hierarchical clustering.

```

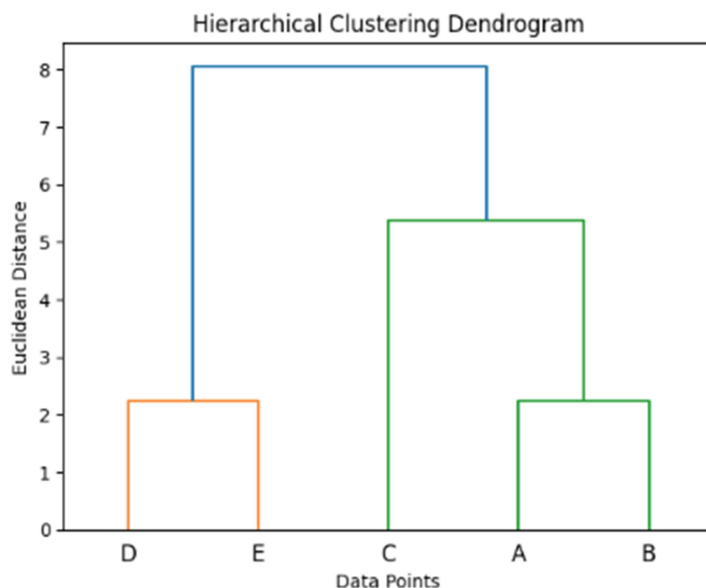
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram

# Generate example data
np.random.seed(42)
X = np.array([[2, 5], [3, 3], [5, 8], [8, 5], [10, 6]])

# Perform hierarchical clustering using linkage function
linkage_matrix = linkage(X, method='complete', metric='euclidean')

# Plot the dendrogram
dendrogram(linkage_matrix, labels=['A', 'B', 'C', 'D', 'E'])
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()

```



## Top-Down (Divisive) and Bottom-Up (Agglomerative) Hierarchical Clustering:

In the context of unsupervised learning algorithms, specifically hierarchical clustering, the terms "top-down" and "bottom-up" refer to two different approaches for building the hierarchical structure, and "single-linkage" and "multiple linkage" refer to different strategies for measuring the distance between clusters.

### Top-Down (Divisive) and Bottom-Up (Agglomerative) Hierarchical Clustering:

#### 1. Top-Down (Divisive) Hierarchical Clustering:

- This approach starts with the entire dataset as one cluster and recursively splits it into smaller clusters until each data point forms a singleton cluster.
- At each step, the algorithm selects a cluster and divides it into two.

#### 2. Bottom-Up (Agglomerative) Hierarchical Clustering:

- This approach starts with each data point as a singleton cluster and merges the closest clusters until only one cluster remains.
- At each step, the algorithm merges the two closest clusters.

### **Single-Linkage and Multiple-Linkage:**

#### **1. Single-Linkage (Nearest-Neighbor Linkage):**

- In single-linkage hierarchical clustering, the distance between two clusters is defined as the shortest distance between any two points in the two clusters.
- The linkage criterion is based on the minimum distance between points in different clusters.

#### **2. Complete-Linkage (Farthest-Neighbor Linkage):**

- In complete-linkage hierarchical clustering, the distance between two clusters is defined as the longest distance between any two points in the two clusters.
- The linkage criterion is based on the maximum distance between points in different clusters.

#### **3. Average-Linkage:**

- In average-linkage hierarchical clustering, the distance between two clusters is defined as the average distance between all pairs of points in the two clusters.
- The linkage criterion is based on the average distance between points in different clusters.

#### **4. Ward's Method:**

- Ward's method is another linkage criterion that minimizes the variance within clusters when merging them.
- It tends to produce more balanced and compact clusters.



```

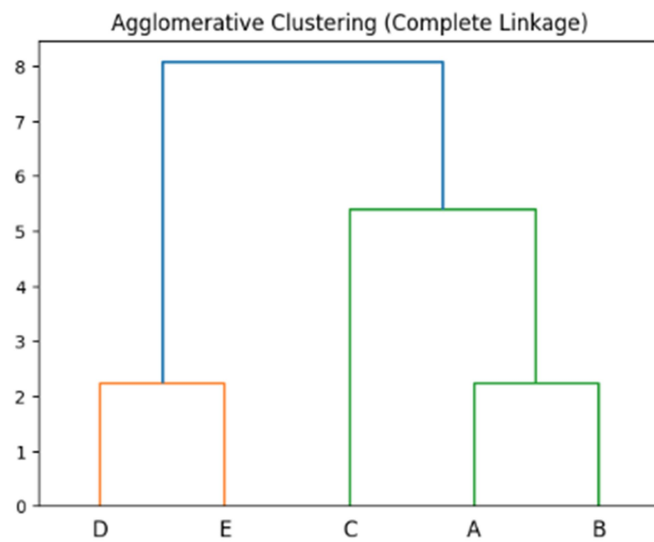
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Generate example data
np.random.seed(42)
X = np.array([[2, 5], [3, 3], [5, 8], [8, 5], [10, 6]])

# Agglomerative clustering with complete linkage
agg_cluster_complete = AgglomerativeClustering(n_clusters=None, linkage='complete', distance_threshold=0)
agg_labels_complete = agg_cluster_complete.fit_predict(X)

# Plot dendrogram for complete linkage
linkage_matrix_complete = linkage(X, method='complete')
dendrogram(linkage_matrix_complete, labels=['A', 'B', 'C', 'D', 'E'])
plt.title('Agglomerative Clustering (Complete Linkage)')
plt.show()

```



## Dimensionality reduction

Dimensionality reduction is a technique used in machine learning and data analysis to reduce the number of input features or variables while preserving the important information in the data. This is often done to mitigate the curse of dimensionality, improve computational efficiency, and avoid over fitting. Here are some popular algorithms for dimensionality reduction:

Key Algorithms:

### 1. Principal Component Analysis (PCA) (1901):

- Identifies orthogonal directions of maximum variance in the data.
- Projects data onto these principal components, creating lower-dimensional representations.
- Widely used due to simplicity and effectiveness.

### 2. Linear Discriminant Analysis (LDA) (1936):

- Supervised method that considers class labels during projection.
- Maximizes class separability in the reduced space.
- Often used for classification tasks.

### 3. Factor Analysis (1904):

- Assumes underlying latent variables explain observed correlations between features.
- Models these latent factors to reduce dimensionality.
- Commonly used in social sciences and psychometrics.

### 4. t-Distributed Stochastic Neighbor Embedding (t-SNE) (2008):

- Non-linear technique for visualizing high-dimensional data in 2D or 3D.
- Preserves local structure while revealing global patterns.
- Popular for visual exploration of complex datasets.

### 5. Singular Value Decomposition (SVD) (1873):

- Matrix factorization technique with various applications, including dimensionality reduction.
- Decomposes a matrix into three matrices:  $U$ ,  $\Sigma$ , and  $V^*$ .
- Truncated SVD approximates the original matrix with fewer dimensions.

### 6. Auto encoders (1980s - 1990s):

- Neural networks trained to reconstruct their input data.
- Learn compressed representations in the hidden layers.
- Flexible for non-linear dimensionality reduction.

### 7. Independent Component Analysis (ICA) (1990s):

- Finds independent components within the data.
- Useful for signal separation and blind source separation.

### 8. Random Projection (2006):

- Projects data onto a lower-dimensional subspace using random matrices.
- Often computationally efficient and preserves pairwise distances well.

### 9. Uniform Manifold Approximation and Projection (UMAP) (2018):

- Non-linear technique preserving global structure while revealing local patterns.
- Often used for visualization and clustering, often outperforming t-SNE.

### Choosing the Right Algorithm:

- Data characteristics: linear vs. non-linear relationships, noise levels, etc.
- Purpose: visualization, compression, classification, feature selection, etc.
- Interpretability requirements: some algorithms yield more interpretable results.
- Computational efficiency: consider algorithm speed and scalability.

## **Principal Component Analysis (PCA)**

Principal Component Analysis (PCA) is a dimensionality reduction technique widely used in machine learning and data analysis. Its primary goal is to transform high-dimensional data into a lower-dimensional representation while retaining as much of the original variability as possible. PCA achieves this by identifying the directions (principal components) in the data along which the variance is maximized.

### **Key Concepts:**

#### **1. Covariance Matrix:**

- PCA starts by computing the covariance matrix of the input data. The covariance matrix captures the relationships between different features, indicating how they vary together.

#### **2. Eigenvalues and Eigenvectors:**

- PCA then calculates the eigenvalues and corresponding eigenvectors of the covariance matrix. Eigenvectors represent the directions of maximum variance, and eigenvalues indicate the magnitude of variance along those directions.

#### **3. Principal Components:**

- The eigenvectors become the principal components of the data. These are the new coordinate axes in the transformed space. The first principal component corresponds to the direction of maximum variance, the second to the second-highest variance, and so on.

#### **4. Explained Variance:**

- Each eigenvalue represents the amount of variance captured by its corresponding principal component. The total variance of the data remains constant, but PCA allows for prioritizing the most important dimensions.

#### **5. Dimensionality Reduction:**

- The principal components are ranked by their corresponding eigenvalues. By selecting the top-k principal components (where k is the desired reduced dimensionality), you can create a lower-dimensional representation of the data.

### **Steps in PCA:**

#### **1. Standardization:**

- Standardize the input features (subtract mean and divide by the standard deviation) to ensure that each feature contributes equally to the analysis.

#### **2. Covariance Matrix Calculation:**

- Calculate the covariance matrix of the standardized data.

#### **3. Eigen decomposition:**

- Perform Eigen decomposition on the covariance matrix to obtain eigenvalues and eigenvectors.

#### **4. Select Principal Components:**

- Rank the eigenvalues in descending order and choose the top-k eigenvectors to form the principal components matrix.

#### **5. Data Transformation:**

- Project the original data onto the selected principal components to obtain the lower-dimensional representation.

```

In [1]: import numpy as np

# Step 1: Data standardization
def standardize(X):
    return (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Step 2: Covariance matrix calculation
def compute_covariance_matrix(X):
    return np.cov(X.T)

# Step 3: Eigenvalue and eigenvector calculation
def find_eigenvectors_and_eigenvalues(X):
    cov_matrix = compute_covariance_matrix(X)
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
    return eigenvalues, eigenvectors

# Step 4: Principal component calculation
def project_data(X, eigenvectors, k):
    sorted_eigenvectors = eigenvectors[:, np.argsort(-eigenvalues)[:k]]
    return np.dot(X, sorted_eigenvectors)

# Step 5: Dimensionality reduction
def get_variance_explained(eigenvalues, k):
    return sum(eigenvalues[:k]) / sum(eigenvalues)

# Example usage
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
X_std = standardize(X)
eigenvalues, eigenvectors = find_eigenvectors_and_eigenvalues(X_std)
projected_data = project_data(X_std, eigenvectors, 2)
variance_explained = get_variance_explained(eigenvalues, 2)

print("Standardized data:")
print(X_std)

print("Covariance matrix:")
print(compute_covariance_matrix(X_std))

print("Eigenvalues:")
print(eigenvalues)

print("Eigenvectors:")
print(eigenvectors)

print("Projected data:")
print(projected_data)

print("Variance explained:")
print(variance_explained)

```

## Output:

```
Standardized data:
[[-1.22474487 -1.22474487 -1.22474487]
 [ 0.         0.         0.         ]
 [ 1.22474487  1.22474487  1.22474487]]
Covariance matrix:
[[1.5 1.5 1.5]
 [1.5 1.5 1.5]
 [1.5 1.5 1.5]]
Eigenvalues:
[0.  4.5 0. ]
Eigenvectors:
[[-0.81649658  0.57735027  0.         ]
 [ 0.40824829  0.57735027 -0.70710678]
 [ 0.40824829  0.57735027  0.70710678]]
Projected data:
[[[-1.41421356  0.29289322]
  [-0.54818816 -0.34108138]
  [-2.28023897 -2.07313218]]

 [[ 0.         0.         ]
 [ 0.         0.         ]
 [ 0.         0.         ]]

 [[ 1.41421356 -0.29289322]
 [ 0.54818816  0.34108138]
 [ 2.28023897  2.07313218]]]
Variance explained:
1.0
```

### Program 1: Implement a program to perform principal component analysis on a dataset.

```
✓ 2s ▶ import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Standardize the data (important for PCA)
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

# Perform PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_standardized)

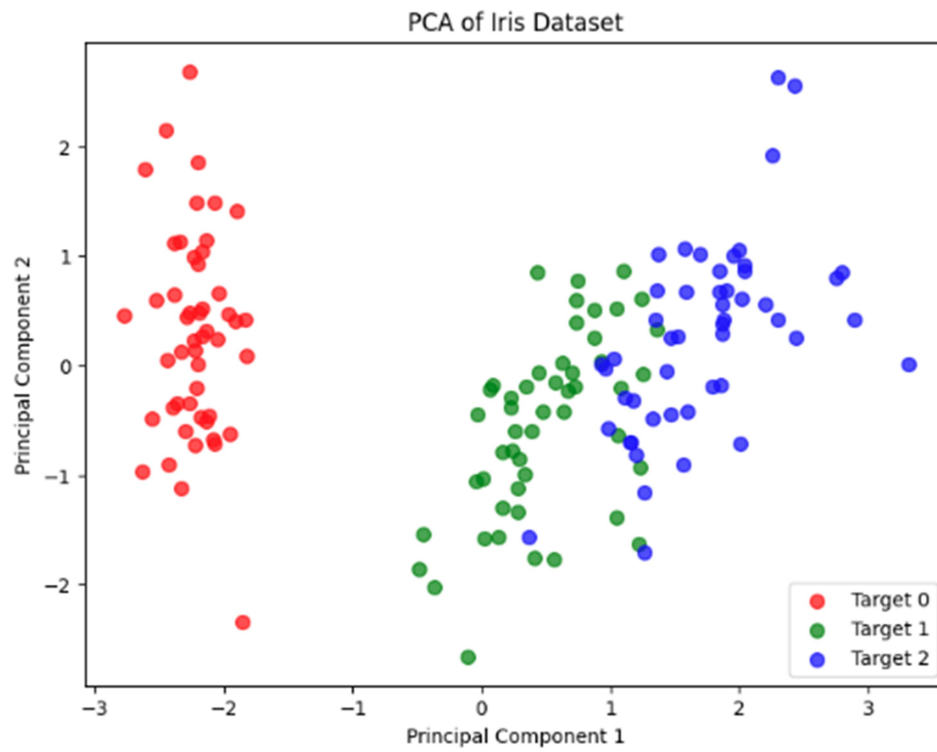
# Create a DataFrame for visualization
df_pca = pd.DataFrame(data=X_pca, columns=['Principal Component 1', 'Principal Component 2'])
df_pca['Target'] = y

# Plot the PCA result
plt.figure(figsize=(8, 6))
targets = [0, 1, 2]
colors = ['r', 'g', 'b']

for target, color in zip(targets, colors):
    indices_to_keep = df_pca['Target'] == target
    plt.scatter(df_pca.loc[indices_to_keep, 'Principal Component 1'],
                df_pca.loc[indices_to_keep, 'Principal Component 2'],
                c=color, s=50, alpha=0.7, label=f'Target {target}')

plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

Output:



```

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the Breast Cancer Wisconsin (Diagnostic) dataset
data = load_breast_cancer()
X = data.data
y = data.target
feature_names = data.feature_names

# Print the total number of features
total_features = X.shape[1]
print(f"Total number of features: {total_features}")

# Standardize the data (subtract mean and divide by standard deviation)
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

# Perform PCA with scikit-learn
pca = PCA()
principal_components = pca.fit_transform(X_standardized)

# Number of Components vs. Explained Variance Plot
num_components = np.arange(1, total_features + 1)
explained_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# Plot the Number of Components vs. Explained Variance
plt.figure(figsize=(10, 6))
plt.plot(num_components, explained_variance_ratio, marker='o', linestyle='--')
plt.title('Number of Components vs. Explained Variance')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid(True)
plt.show()

# Specify a threshold for cumulative explained variance (e.g., 95%)
threshold_variance = 0.95

# Find the number of components that achieve the specified threshold
num_components_threshold = np.argmax(explained_variance_ratio >= threshold_variance) + 1

# Print the number of components that achieve the specified threshold
print(f"Number of components to achieve {threshold_variance * 100}% cumulative explained variance: {num_components_threshold}")

# Select the best features based on the specified number of components
best_features = pca.transform(X_standardized)[:num_components_threshold]

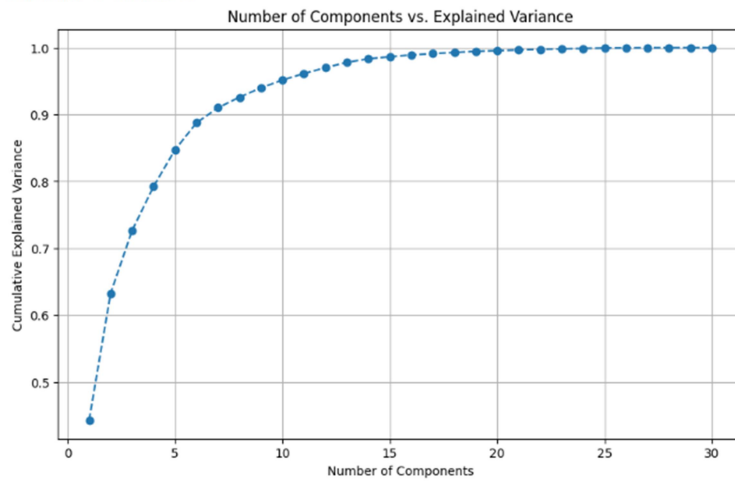
# Print the names of the best features
print(f"Best features after PCA:")
best_feature_names = [feature_names[i] for i in range(num_components_threshold)]
print(best_feature_names)

```



## Output:

Total number of features: 30



Number of components to achieve 95.0% cumulative explained variance: 10

Best features after PCA:

['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothness', 'mean compactness', 'mean concavity', 'mean concave points', 'mean symmetry', 'mean fractal dimension']

## Program 2: Implement a program to calculate the covariance matrix for a dataset.

```
import numpy as np

# Predefined dataset
dataset = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
])

# Calculate the covariance matrix
covariance_matrix = np.cov(dataset, rowvar=False)

# Print the covariance matrix
print("Covariance Matrix:")
print(covariance_matrix)
```

Covariance Matrix:

```
[[15. 15. 15.]
 [15. 15. 15.]
 [15. 15. 15.]
```

**Program 3: Implement a program to calculate the singular value decomposition for a dataset.**

```
✓ 0s ▶ import numpy as np

# Predefined dataset
dataset = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
])

# Perform Singular Value Decomposition
U, S, Vt = np.linalg.svd(dataset)

# U, S, and Vt are matrices such that dataset = U * S * Vt

# Print the results
print("U matrix:")
print(U)
print("\nS diagonal matrix (singular values):")
print(np.diag(S))
print("\nVt matrix:")
print(Vt)
```

```
U matrix:
[[-0.14087668 -0.82471435  0.53999635 -0.09167299]
 [-0.34394629 -0.42626394 -0.65166613  0.52472017]
 [-0.54701591 -0.02781353 -0.31665681 -0.77442137]
 [-0.75008553  0.37063688  0.42832658  0.34137419]]

S diagonal matrix (singular values):
[[2.54624074e+01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.29066168e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 2.40694596e-15]]

Vt matrix:
[[-0.50453315 -0.5745157 -0.64449826]
 [ 0.76077568  0.05714052 -0.64649464]
 [-0.40824829  0.81649658 -0.40824829]]
```

## Hierarchical clustering:

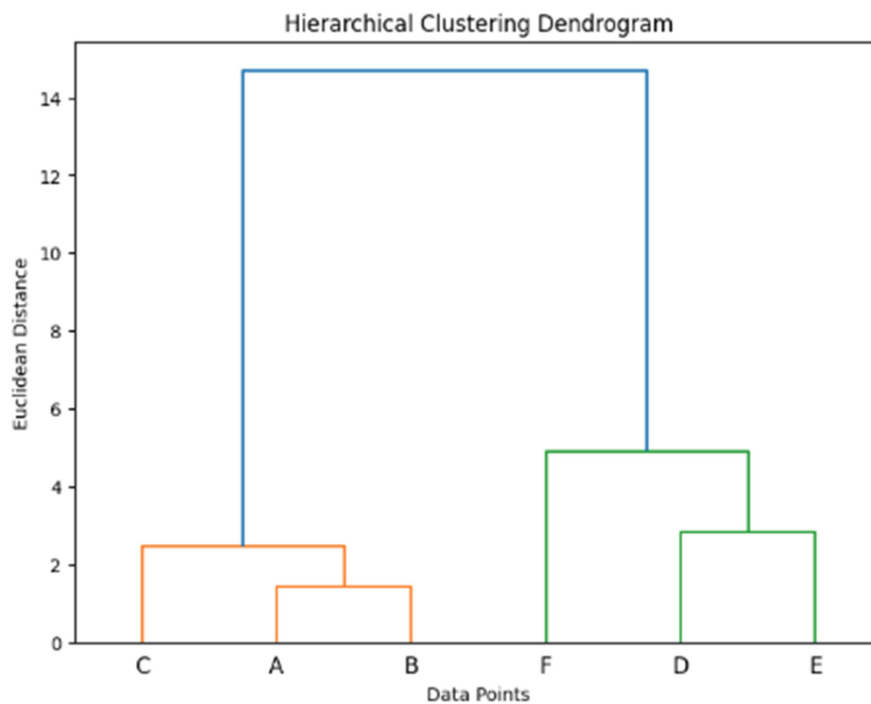
Program 1: Implement a program to perform hierarchical clustering on a dataset.

```
13 import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Predefined dataset
dataset = np.array([
    [1, 2],
    [2, 3],
    [3, 4],
    [6, 7],
    [8, 9],
    [10, 11]
])

# Perform hierarchical clustering
linkage_matrix = linkage(dataset, method='ward') # You can choose different linkage methods

# Plot the dendrogram
plt.figure(figsize=(8, 6))
dendrogram(linkage_matrix, labels=['A', 'B', 'C', 'D', 'E', 'F'])
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()
```



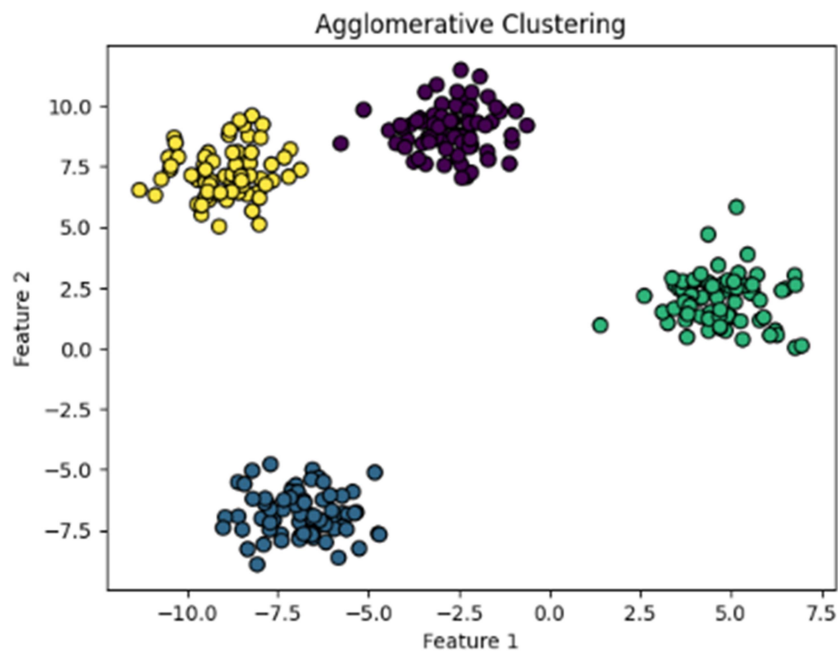
Program 2: Implement a program to calculate the agglomerative clustering algorithm for a hierarchical clustering.

```
✓ 2s ▶ import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs

# Predefined dataset
X, _ = make_blobs(n_samples=300, centers=4, random_state=42, cluster_std=1.0)

# Perform agglomerative clustering
n_clusters = 4 # You can choose the number of clusters
agglomerative_cluster = AgglomerativeClustering(n_clusters=n_clusters)
labels = agglomerative_cluster.fit_predict(X)

# Plot the clustered data
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', edgecolors='k', s=50)
plt.title('Agglomerative Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



Program 3: Implement a program to calculate the divisive clustering algorithm for a hierarchical clustering.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.datasets import make_blobs

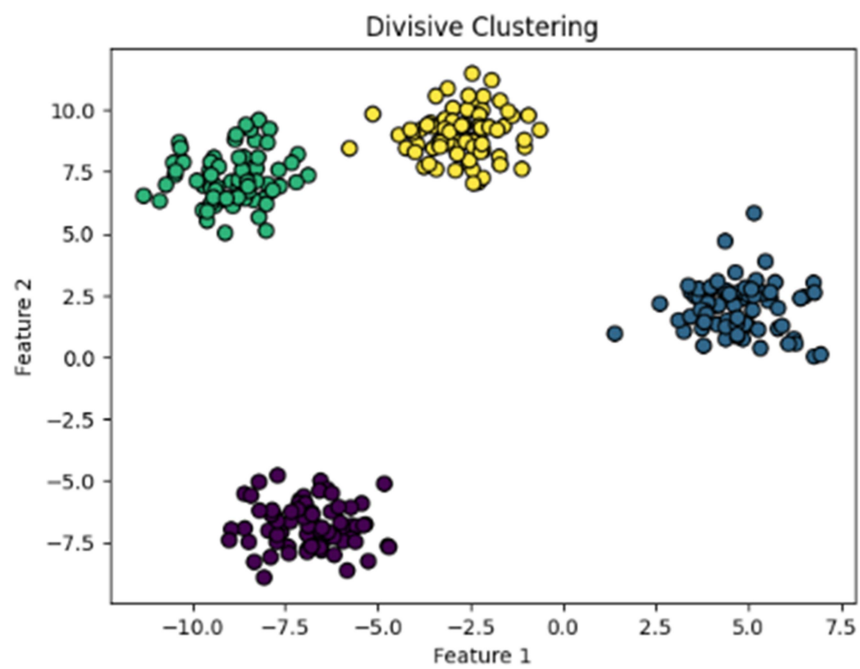
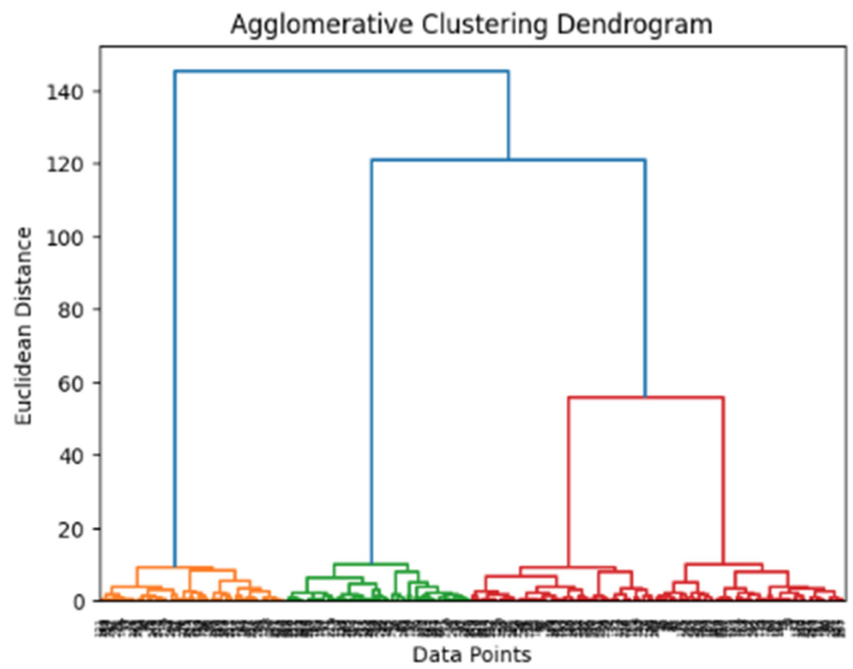
# Predefined dataset
X, _ = make_blobs(n_samples=300, centers=4, random_state=42, cluster_std=1.0)

# Perform agglomerative clustering
linkage_matrix = linkage(X, method='ward') # You can choose different linkage methods
dendrogram(linkage_matrix)
plt.title('Agglomerative Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()

# Set the threshold to cut the dendrogram and get divisive clustering
threshold = 25 # You can adjust this threshold
divisive_labels = fcluster(linkage_matrix, t=threshold, criterion='distance')

# Plot the divisive clustered data
plt.scatter(X[:, 0], X[:, 1], c=divisive_labels, cmap='viridis', edgecolors='k', s=50)
plt.title('Divisive Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Output:



## Anomaly detection:

Program 1: Implement a program to detect anomalies in a dataset using the isolation forest algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.datasets import make_blobs

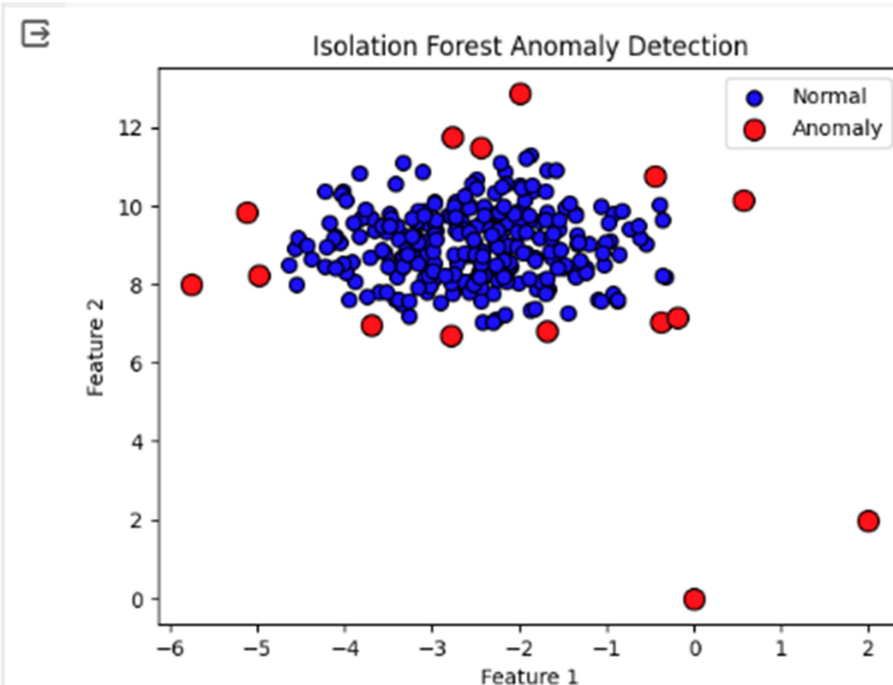
# Predefined dataset
X, _ = make_blobs(n_samples=300, centers=1, random_state=42, cluster_std=1.0)
# Introduce some anomalies
X[-1] = [0, 0]
X[-2] = [2, 2]

# Create and fit the Isolation Forest model
isolation_forest = IsolationForest(contamination=0.05) # Adjust contamination based on your data
isolation_forest.fit(X)

# Predict anomaly scores
anomaly_scores = isolation_forest.decision_function(X)

# Plot the dataset and highlight anomalies
plt.scatter(X[:, 0], X[:, 1], c='b', edgecolors='k', s=50, label='Normal')
plt.scatter(X[anomaly_scores < 0, 0], X[anomaly_scores < 0, 1], c='r', edgecolors='k', s=100, label='Anomaly')
plt.title('Isolation Forest Anomaly Detection')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output:



Program 2: Implement a program to detect anomalies in a dataset using the one-class support vector machine.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import OneClassSVM
from sklearn.datasets import make_blobs

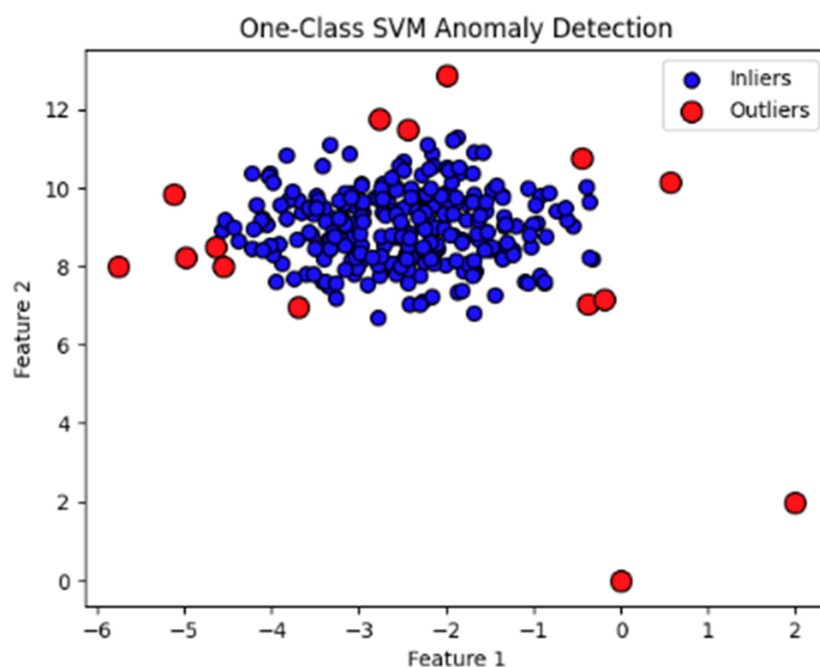
# Predefined dataset
X, _ = make_blobs(n_samples=300, centers=1, random_state=42, cluster_std=1.0)
# Introduce some anomalies
X[-1] = [0, 0]
X[-2] = [2, 2]

# Create and fit the One-Class SVM model
ocsvm = OneClassSVM(nu=0.05) # Adjust nu based on your data
ocsvm.fit(X)

# Predict inliers and outliers
inliers = ocsvm.predict(X) == 1
outliers = ocsvm.predict(X) == -1

# Plot the dataset and highlight anomalies
plt.scatter(X[inliers, 0], X[inliers, 1], c='b', edgecolors='k', s=50, label='Inliers')
plt.scatter(X[outliers, 0], X[outliers, 1], c='r', edgecolors='k', s=100, label='Outliers')
plt.title('One-Class SVM Anomaly Detection')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output:





Program 3: Implement a program to evaluate the performance of an anomaly detection algorithm.

```
import numpy as np
from sklearn.svm import OneClassSVM
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split

# Predefined dataset
X, _ = make_blobs(n_samples=1000, centers=1, random_state=42, cluster_std=1.0)
# Introduce some anomalies
X[-10:] = [0, 0]

# Split the data into training and testing sets
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Create and fit the One-Class SVM model
ocsvm = OneClassSVM(nu=0.05)
ocsvm.fit(X_train)

# Predict anomalies on the test set
predictions = ocsvm.predict(X_test)

# Evaluate performance
true_labels = np.ones(len(X_test)) # All instances are considered normal
true_labels[-10:] = -1 # Mark anomalies in the test set

precision = precision_score(true_labels, predictions, pos_label=-1)
recall = recall_score(true_labels, predictions, pos_label=-1)
f1 = f1_score(true_labels, predictions, pos_label=-1)
roc_auc = roc_auc_score(true_labels, predictions)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")

Precision: 0.0000
Recall: 0.0000
F1-Score: 0.0000
ROC AUC: 0.4868
```

## Case Study: Customer Segmentation for E-commerce using Unsupervised Learning

### Introduction:

In this case study, we will explore how unsupervised learning techniques, specifically clustering algorithms, can be applied to perform customer segmentation for an e-commerce business. Customer segmentation helps businesses understand the diverse needs and behaviors of their customers, enabling personalized marketing strategies, product recommendations, and improved customer experiences.

### Objective:

The goal is to identify distinct groups of customers based on their purchasing behavior, preferences, and engagement with the e-commerce platform. This segmentation can provide valuable insights for targeted marketing campaigns, product recommendations, and tailored services.

### Dataset:

Assume we have a dataset with the following features:

1. **Customer ID:** Unique identifier for each customer.
2. **Purchase History:** Information about the products purchased, including frequency, recency, and monetary value.
3. **Website Engagement:** Data related to customer engagement, such as time spent on the website, number of visits, etc.
4. **Demographic Information:** Age, gender, location, etc.

### Steps:

1. **Data Preprocessing:**
  - Handle missing values, if any.
  - Standardize or normalize numerical features.
  - Encode categorical variables.
2. **Exploratory Data Analysis (EDA):**
  - Understand the distribution of each feature.
  - Explore correlations between features.
  - Identify outliers and decide whether to handle or remove them.
3. **Feature Engineering:**
  - Create relevant features for analysis.
  - Combine or transform features if needed.
4. **Unsupervised Learning (Clustering):**
  - Apply clustering algorithms such as K-means, hierarchical clustering, or DBSCAN.
  - Choose the appropriate number of clusters based on the data and business understanding.
  - Analyze and interpret the results of clustering.
5. **Customer Segmentation:**
  - Identify and label the segments created by the clustering algorithm.
  - Analyze the characteristics of each segment.
  - Understand the differences and similarities between segments.
6. **Business Insights:**
  - Derive actionable insights for marketing, product development, and customer engagement strategies based on the identified segments.

- Tailor marketing campaigns to address the specific needs of each segment.
  - Optimize product recommendations and pricing strategies.
7. **Evaluation:**
- Evaluate the effectiveness of customer segmentation by monitoring key performance indicators (KPIs) over time.
  - Refine the segmentation approach if necessary.

**Tools and Technologies:**

- Python for data preprocessing, analysis, and visualization.
- Scikit-learn or other machine learning libraries for clustering algorithms.
- Matplotlib and Seaborn for data visualization.

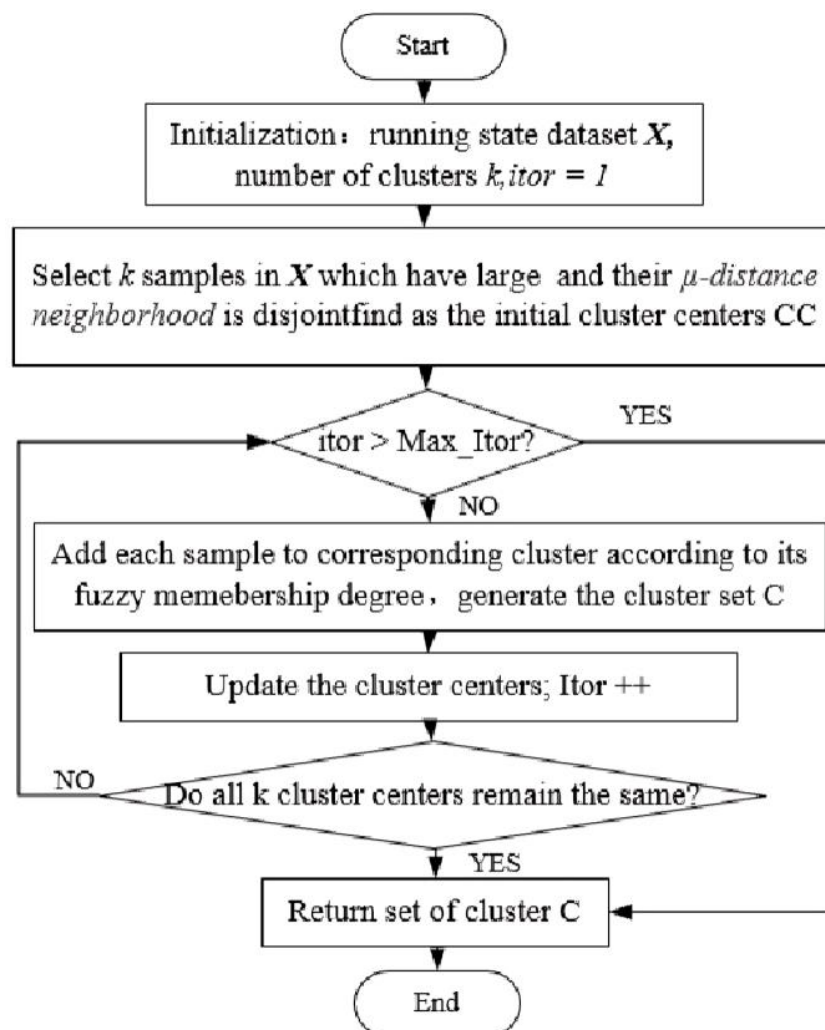
**Conclusion:**

Customer segmentation using unsupervised learning provides businesses with a powerful tool to enhance customer understanding and optimize marketing strategies. By leveraging the insights gained from segmentation, e-commerce businesses can foster customer loyalty, improve customer satisfaction, and drive overall business success.

## Fuzzy K-means clustering

Fuzzy K-means clustering is an extension of the traditional K-means algorithm that allows data points to belong to multiple clusters with varying degrees of membership. Unlike traditional K-means, where each data point is assigned exclusively to one cluster, Fuzzy K-means assigns a membership value between 0 and 1 to each data point for each cluster, indicating the degree to which it belongs to that cluster. A membership value of 1 signifies complete membership, while 0 indicates no membership. This "fuzziness" allows for a more nuanced representation of data, especially when clusters are overlapping or ill-defined.

The working of Fuzzy K-means involves an iterative process similar to traditional K-means. First, the algorithm randomly initializes 'K' cluster centers (centroids). Then, it calculates the membership values of each data point to each cluster based on its distance to the cluster centers. The membership value is inversely proportional to the distance; the closer a data point is to a cluster center, the higher its membership value to that cluster. Next, the cluster centers are updated based on the weighted average of all data points, where the weights are the membership values. This process of calculating membership values and updating cluster centers is repeated until a convergence criterion is met, such as a maximum number of iterations or a sufficiently small change in the cluster centers or membership values.



Flowchart of Fuzzy K-means clustering

The flowchart describes the Fuzzy K-means (or Fuzzy C-means) clustering algorithm. Let's break down each step:

1. **Start:** The beginning of the algorithm.
2. **Initialization:**
  - **Running state dataset X:** This represents the input data, a set of data points that you want to cluster.
  - **Number of clusters k:** This is a user-defined parameter specifying how many clusters the data should be divided into.
  - **itor = 1:** This initializes an iteration counter.
3. **Select  $k$  samples in X...as the initial cluster centers CC:** This is the initial selection of cluster centers (centroids). The text mentions selecting samples with "large  $\mu$ -distance" and "disjoint neighborhood." This suggests a method to choose initial centers that are well-separated, which can improve the algorithm's performance. " $\mu$ -distance" likely refers to a specific distance metric used in this implementation. A common simpler approach is to randomly select  $k$  data points as initial centers.
4. **itor > Max\_itor?:** This is a check for the maximum number of iterations. `Max_itor` is a predefined limit. If the current iteration count (`itor`) exceeds this limit, the algorithm terminates (YES branch). This prevents the algorithm from running indefinitely if it doesn't converge.
5. **NO (itor <= Max\_itor):** If the maximum number of iterations hasn't been reached, the algorithm proceeds with the clustering process.
6. **Add each sample to corresponding cluster according to its fuzzy membership degree, generate the cluster set C:** This is the core of Fuzzy K-means. Unlike K-means, where each data point is assigned to only one cluster, here, each data point is assigned a *membership degree* to each cluster. This degree is a value between 0 and 1, indicating the probability or degree to which the data point belongs to that cluster. The set of all these membership values for all data points and clusters forms the fuzzy partition. The cluster set `c` is generated based on these membership values.
7. **Update the cluster centers; Itor++:** The cluster centers (centroids) are recalculated based on the current membership degrees. Each cluster center is updated as the weighted average of all data points, where the weights are the membership degrees of those points to the cluster. The iteration counter is then incremented.
8. **Do all  $k$  cluster centers remain the same?:** This is a convergence check. If the cluster centers haven't changed significantly from the previous iteration, it means the algorithm has converged, and further iterations are unlikely to improve the clustering.
9. **YES (Cluster centers remain the same):** If the cluster centers haven't changed, the algorithm terminates and returns the set of clusters `c`.
10. **NO (Cluster centers have changed):** If the cluster centers have changed, the algorithm loops back to step 6 (adding samples based on fuzzy membership), continuing the iterative process.
11. **Return set of cluster C:** The algorithm outputs the final cluster assignments (represented by the membership degrees).
12. **End:** The end of the algorithm.

### Key Differences from K-means Highlighted in the Flowchart:

- **Fuzzy Membership:** The crucial difference is in step 6. K-means would assign each point to the *nearest* cluster. Fuzzy K-means assigns a *degree* of membership to *each* cluster.
- **Cluster Center Update:** The cluster center update (step 7) is also different. In K-means, it's the average of the points assigned to that cluster. In Fuzzy K-means, it's a *weighted* average based on the membership degrees.

This flowchart provides a good visual representation of the Fuzzy K-means algorithm's iterative process and highlights its key distinctions from the traditional K-means algorithm.

The key difference between Fuzzy K-means and traditional K-means lies in the cluster assignment and the objective function. In traditional K-means, cluster assignment is hard: each data point is assigned to the nearest cluster center, resulting in distinct and non-overlapping clusters. In contrast, Fuzzy K-means uses soft assignment: data points have membership values for all clusters, allowing for overlapping clusters and a more flexible representation of cluster boundaries. The objective function also differs. Traditional K-means aims to minimize the sum of squared distances between data points and their assigned cluster centers. Fuzzy K-means, on the other hand, minimizes the weighted sum of squared distances, where the weights are the membership values raised to a fuzziness parameter (usually denoted as 'm'). This fuzziness parameter controls the degree of fuzziness in the clustering; higher values of 'm' lead to more fuzzy clusters. This modified objective function allows Fuzzy K-means to handle situations where data points might reasonably belong to multiple clusters, a capability absent in traditional K-means.

## UNIT-V

**Tree based classification:** Decision tree, SVM, Random Forest, Accuracy measure and performance metrics.

**Ensemble methods:** XG Boost, ADA Boost, Bagging, Boosting.

### Tree based classification

Tree-based classification models are a type of supervised machine learning algorithm that uses a series of conditional statements to partition training data into subsets. Each successive split adds some complexity to the model, which can be used to make predictions.

Types of Tree based models are:

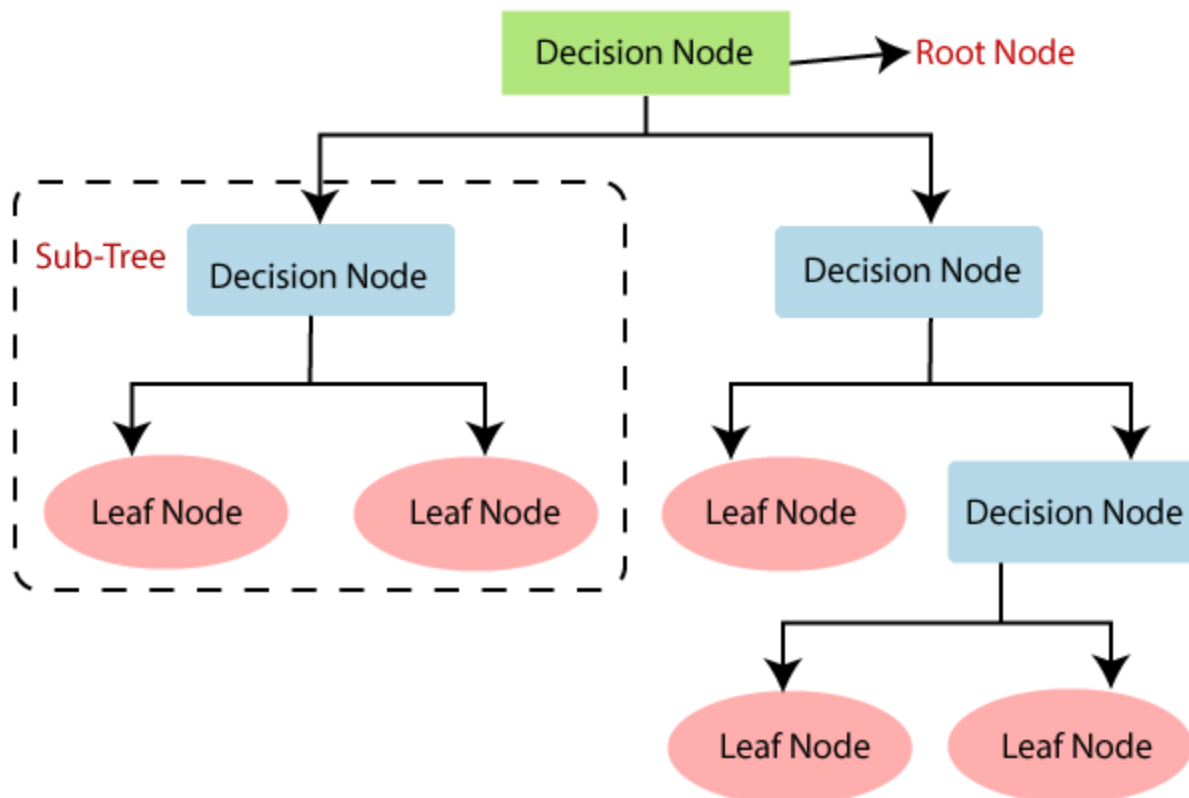
1. Decision Tree

2. SVM (Support Vector Machine)

3. Random Forest

#### 1. Decision Tree:

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome**.
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- *It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.*
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm**.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further splits the tree into subtrees.
- Below diagram explains the general structure of a decision tree:



## Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

## Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

## How does the Decision Tree algorithm Work?



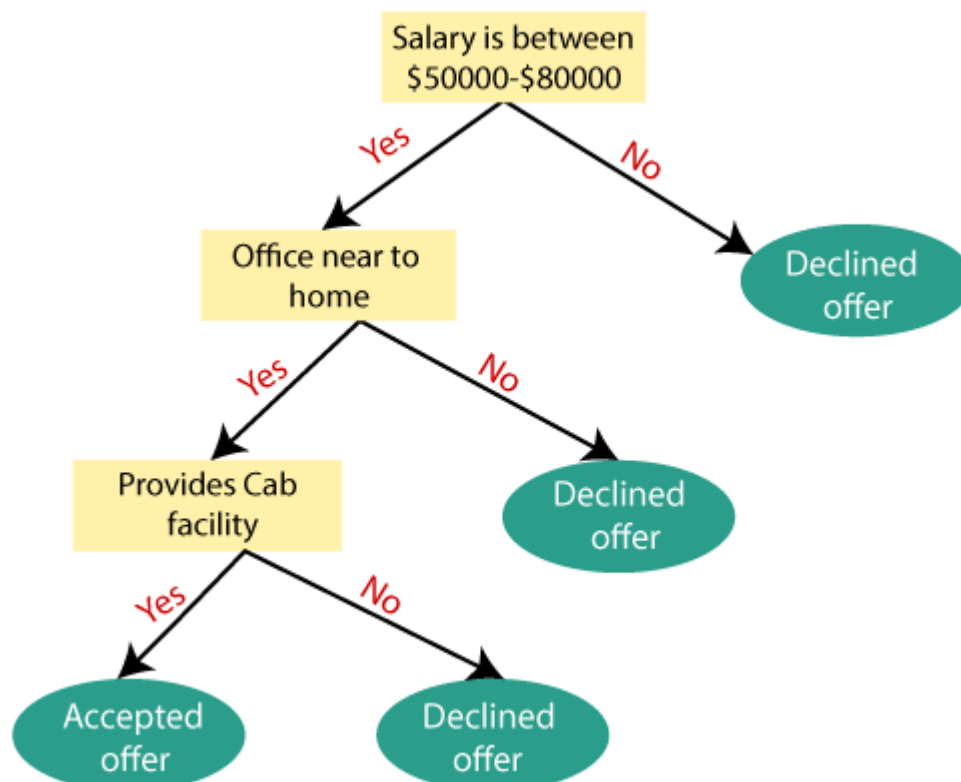
In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

X

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

**Example:** Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



## Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

### 1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

1.  $\text{Information Gain} = \text{Entropy}(S) - [(\text{Weighted Avg}) * \text{Entropy}(\text{each feature})]$

**Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy}(s) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

### 2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

## Pruning: Getting an Optimal Decision tree

*Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

## Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

## Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm**.
- For more class labels, the computational complexity of the decision tree may increase.

## Python Implementation of Decision Tree

Now we will implement the Decision tree using Python. For this, we will use the dataset "user\_data.csv," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as **KNN**

SVM,  
LogisticRegression,  
etc.

Steps will also remain the same, which are given below:

- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

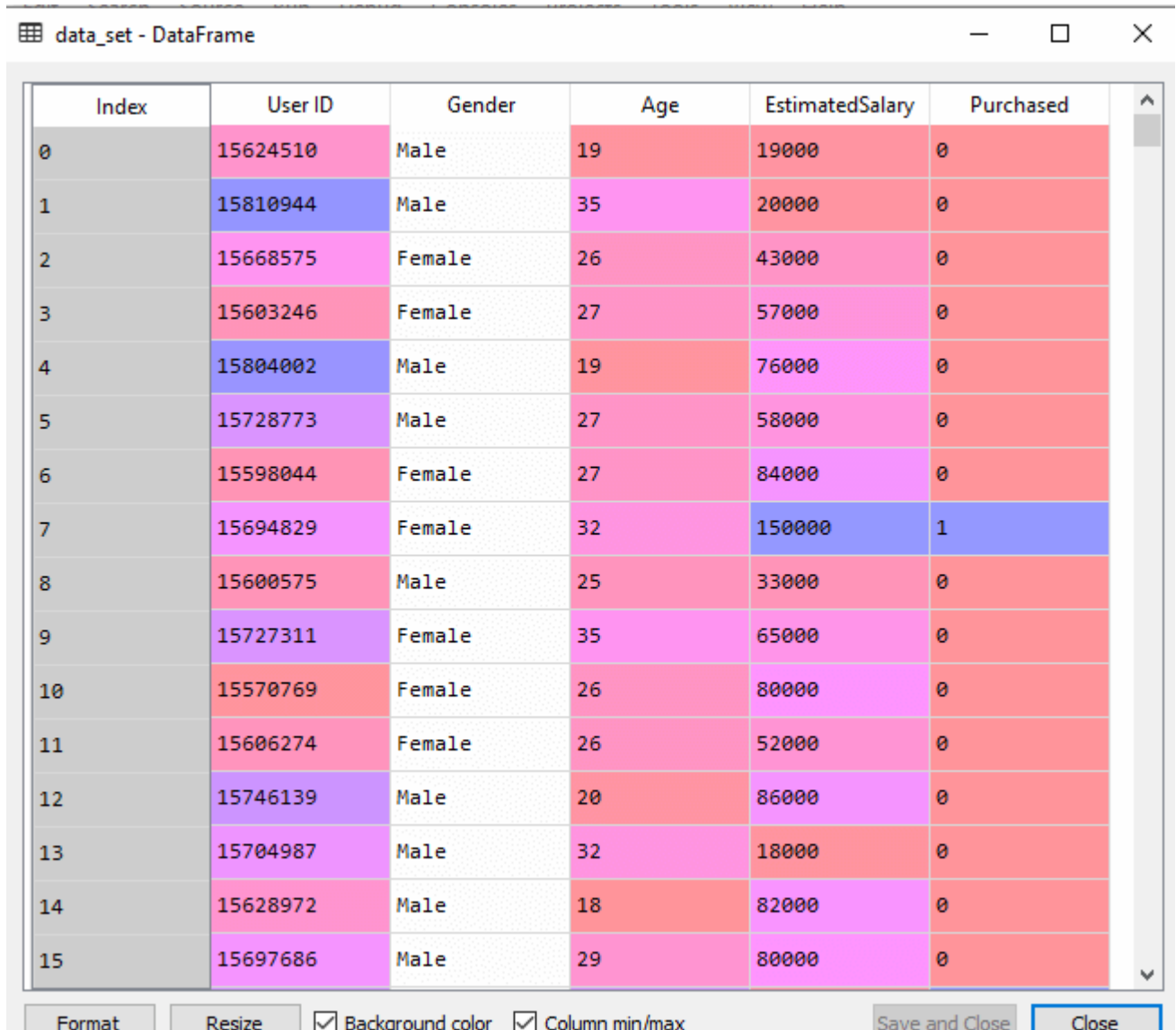
## 1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')
8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, [2,3]].values
11. y= data_set.iloc[:, 4].values
12.
13. # Splitting the dataset into training and test set.
14. from sklearn.model_selection import train_test_split
15. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
16.
17. #feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)
```

21. `x_test= st_x.transform(x_test)`

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

## 2. Fitting a Decision-Tree algorithm to the Training set:

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

1. #Fitting Decision Tree classifier to the training set
2. From sklearn.tree **import** DecisionTreeClassifier

3. `classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)`
4. `classifier.fit(x_train, y_train)`

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy'":** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random\_state=0":** For generating the random states.

Below is the output for this:

```
Out[8]:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
random_state=0, splitter='best')
```

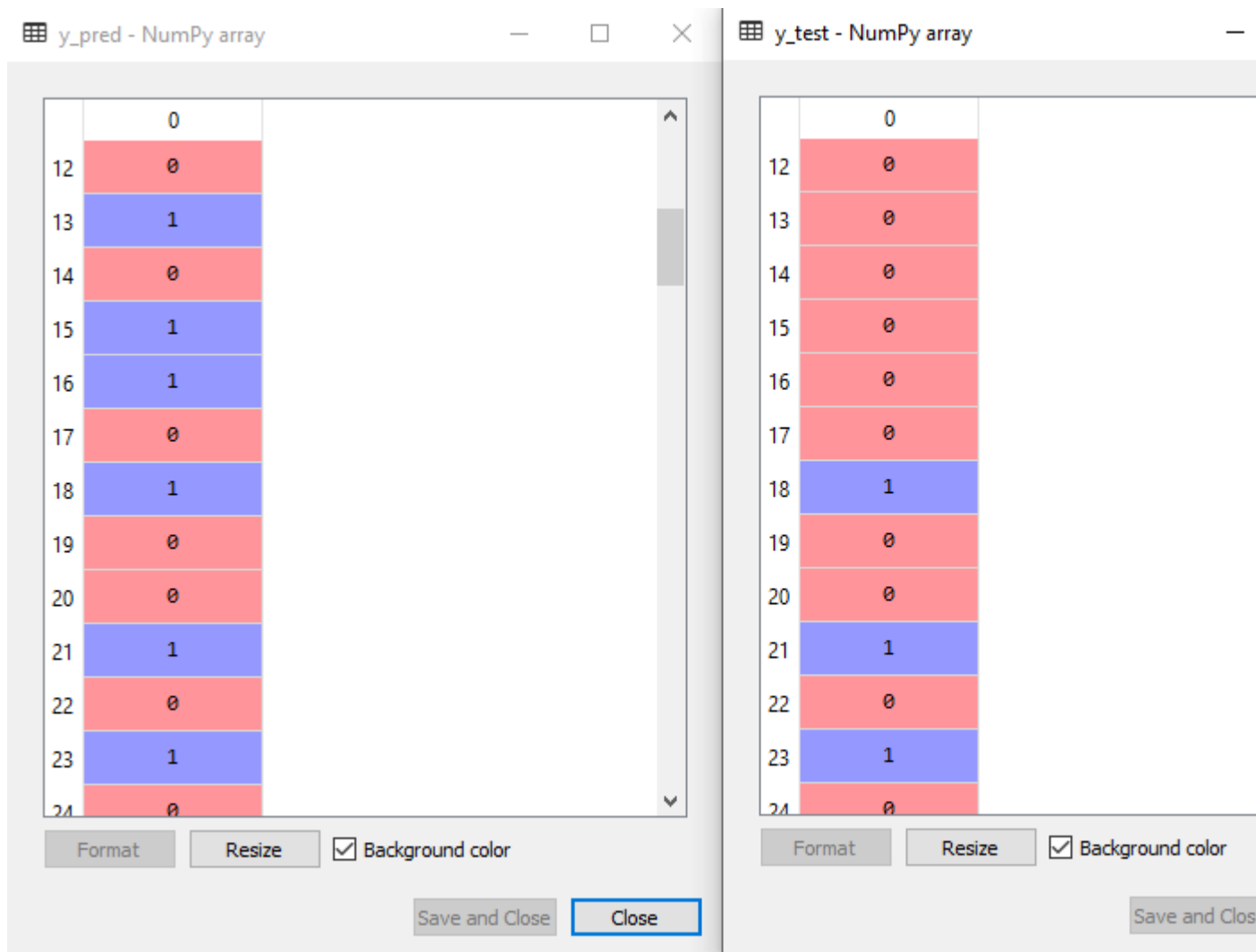
### 3. Predicting the test result:

Now we will predict the test set result. We will create a new prediction vector **y\_pred**. Below is the code for it:

1. `#Predicting the test set result`
2. `y_pred= classifier.predict(x_test)`

#### Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

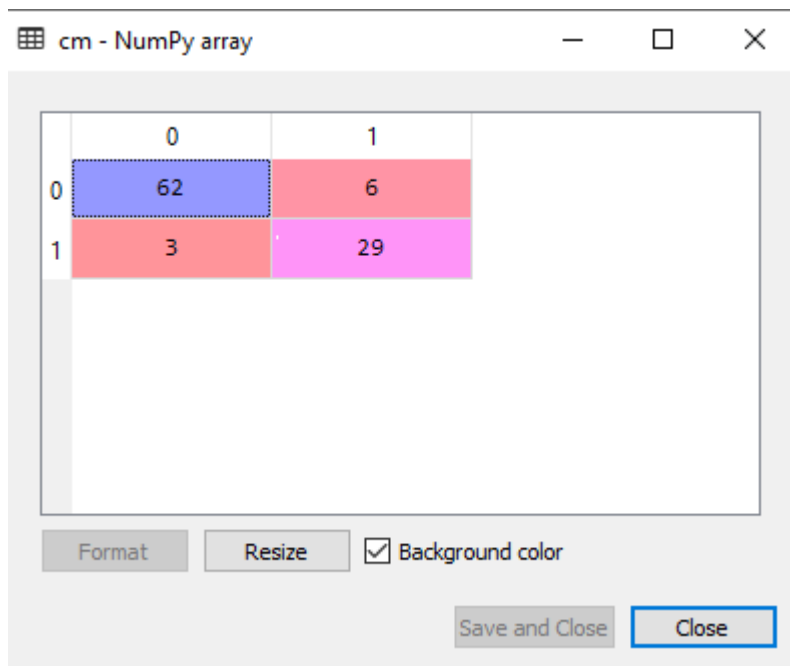


#### 4. Test accuracy of the result (Creation of Confusion matrix):

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix(y\_test, y\_pred)

**Output:**



In the above output image, we can see the confusion matrix, which has  $6+3=9$  incorrect predictions and  $62+29=91$  correct predictions. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.

## 5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in [Logistic Regression](#).

Below is the code for it:

1. #Visulaizing the trianing set result
2. from matplotlib.colors **import** ListedColormap
3. x\_set, y\_set = x\_train, y\_train
4. x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),  
1, stop = x\_set[:, 0].max() + 1, step = 0.01),
5. nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(('purple','green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y\_set)):
11. mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

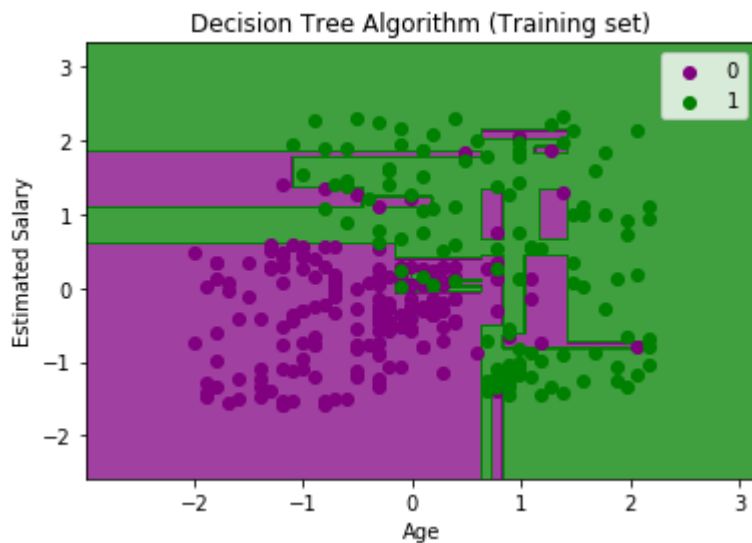


```

12.     c = ListedColormap(('purple', 'green'))(i), label = j)
13. mtp.title('Decision Tree Algorithm (Training set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

### Output:



The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

## 6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```

1. #Visulaizing the test set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_test, y_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
    1, stop = x_set[:, 0].max() + 1, step =0.01),

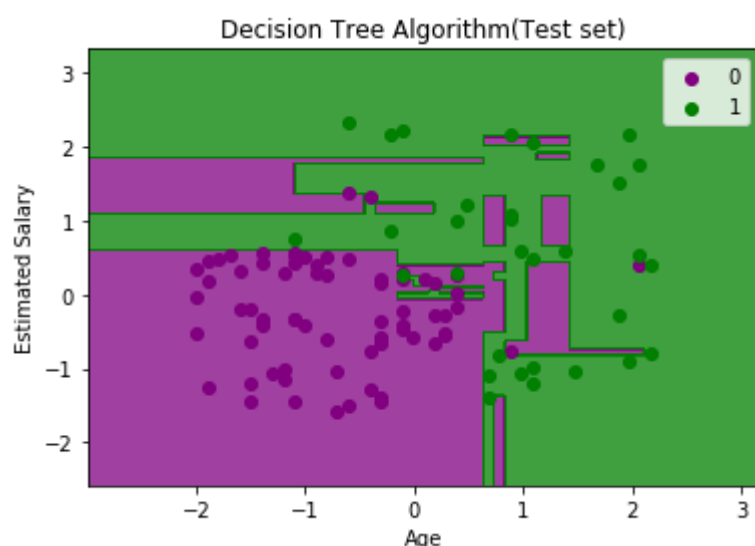
```

```

5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(('purple', 'green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.             c = ListedColormap(('purple', 'green'))(i), label = j)
13. mtp.title('Decision Tree Algorithm(Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

**Output:**



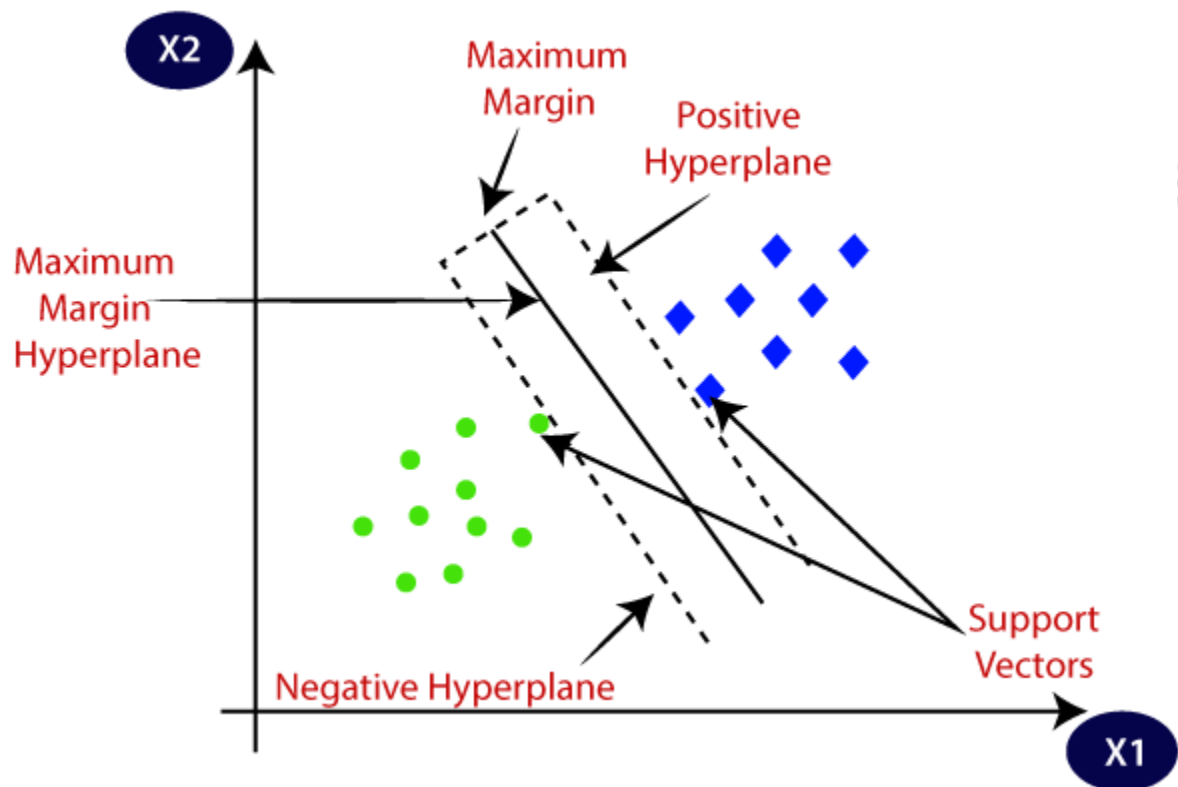
As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

## **2.SVM(Support Vector Machine);**

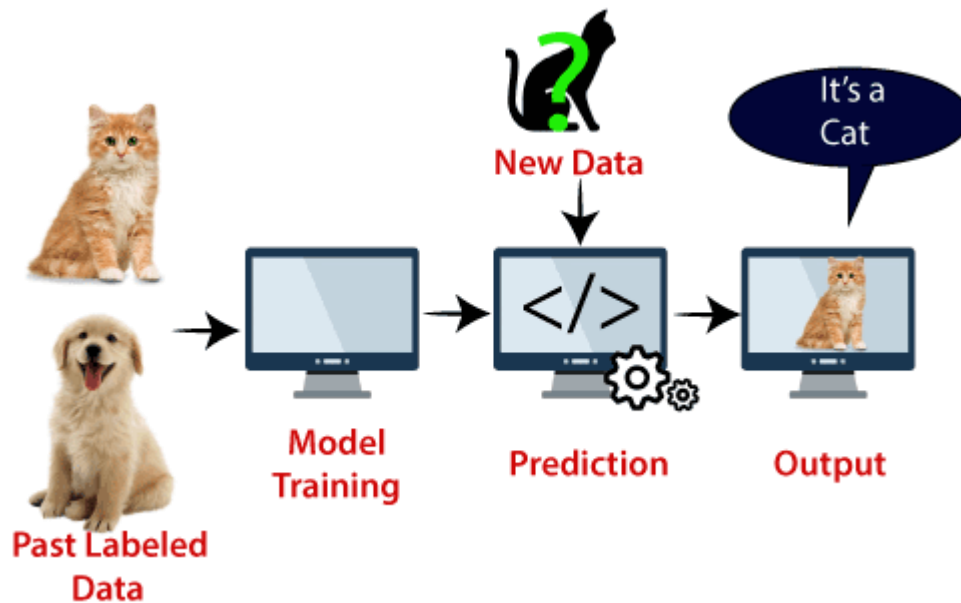
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



**Example:** SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization**, etc.

### Types of SVM

**SVM can be of two types:**

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

### Hyperplane and Support Vectors in the SVM algorithm:

**Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

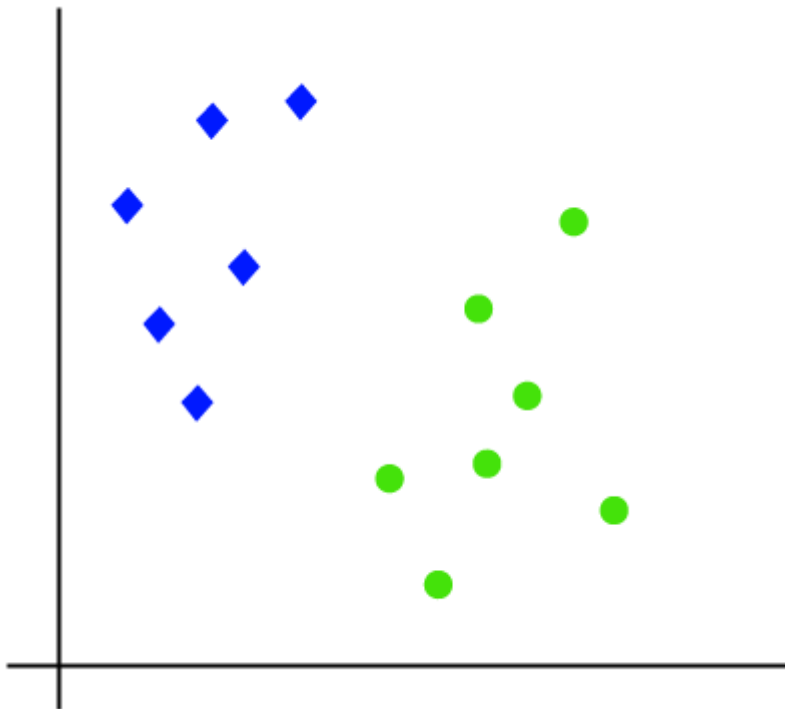
### Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

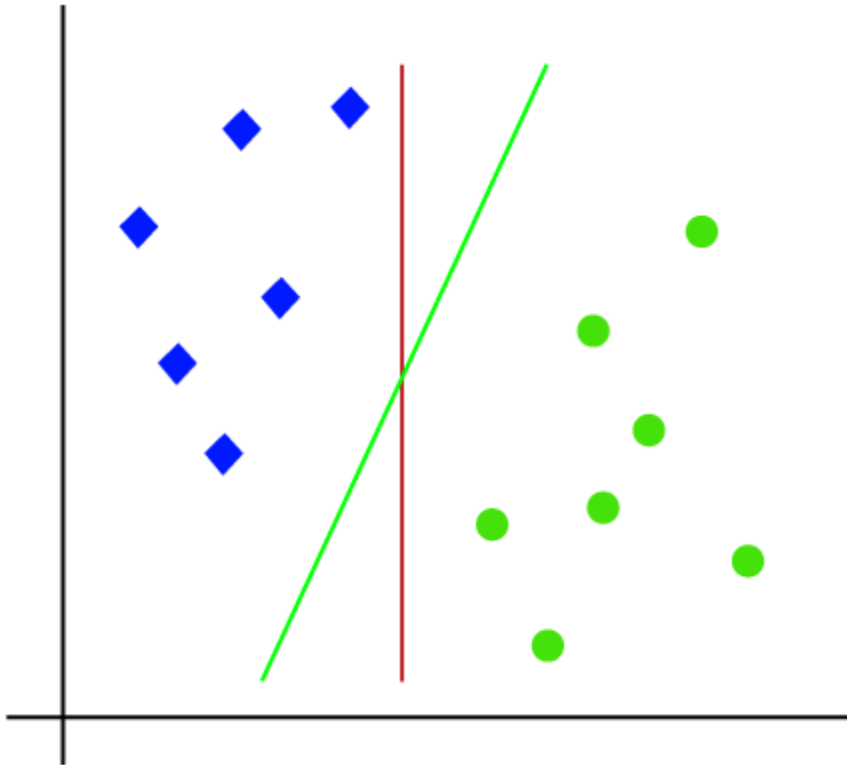
How does SVM works?

### Linear SVM:

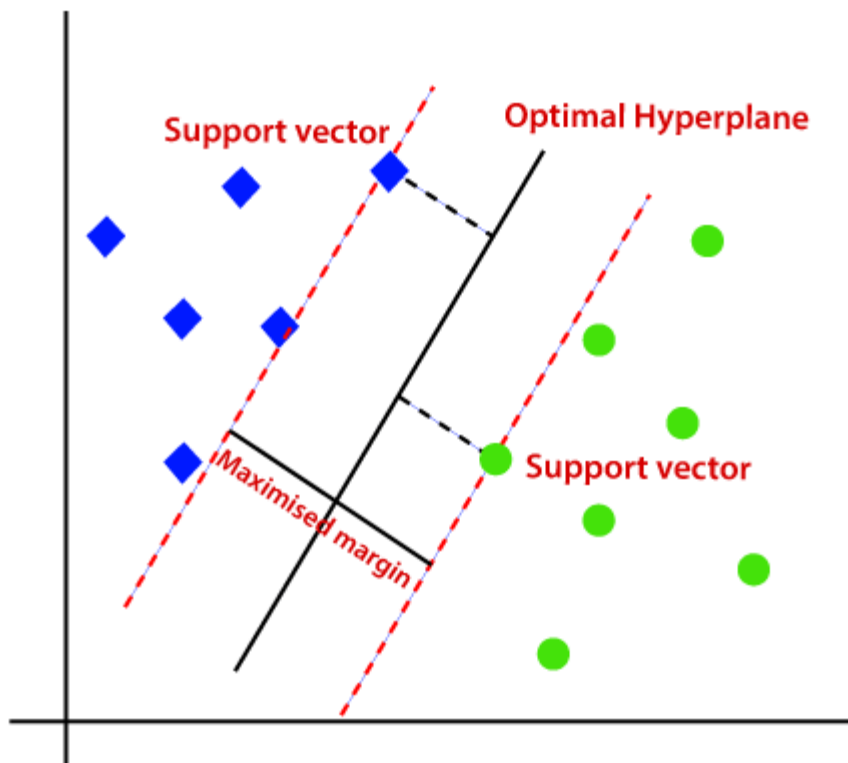
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features  $x_1$  and  $x_2$ . We want a classifier that can classify the pair( $x_1$ ,  $x_2$ ) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

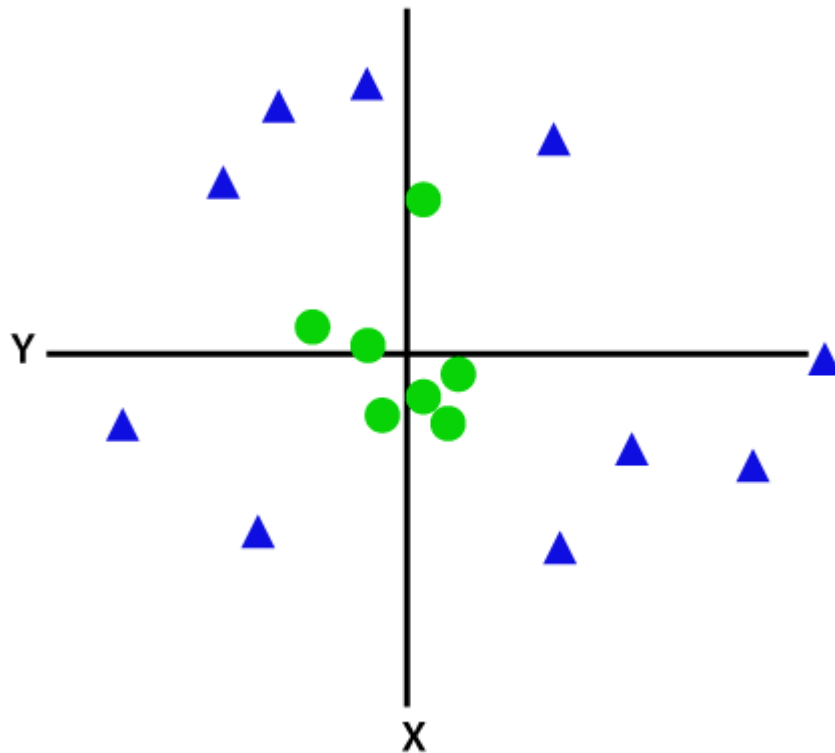


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



### Non-Linear SVM:

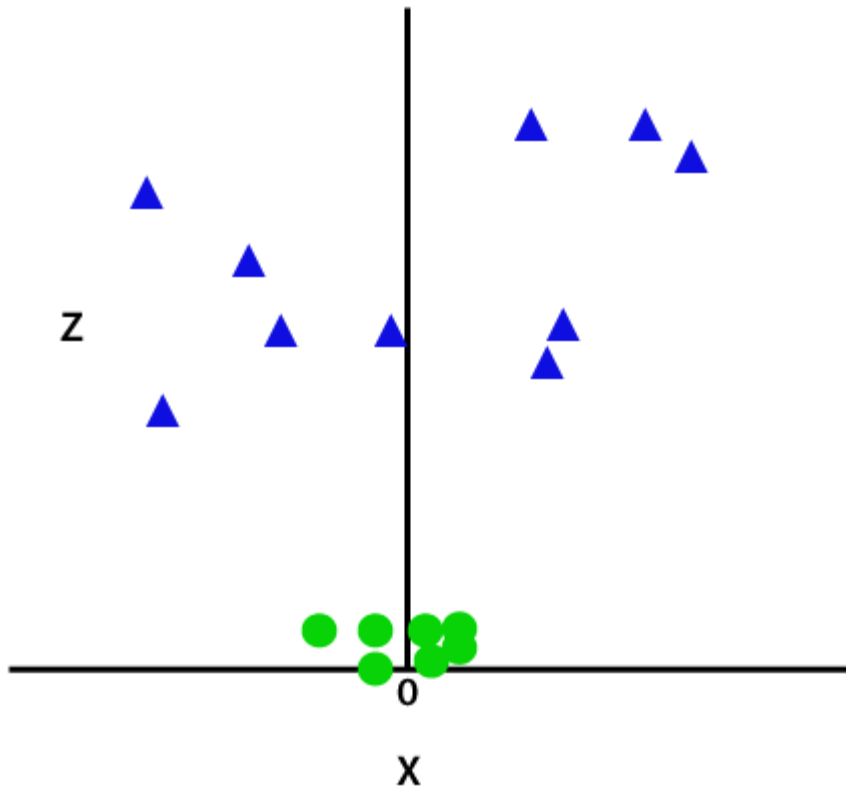
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



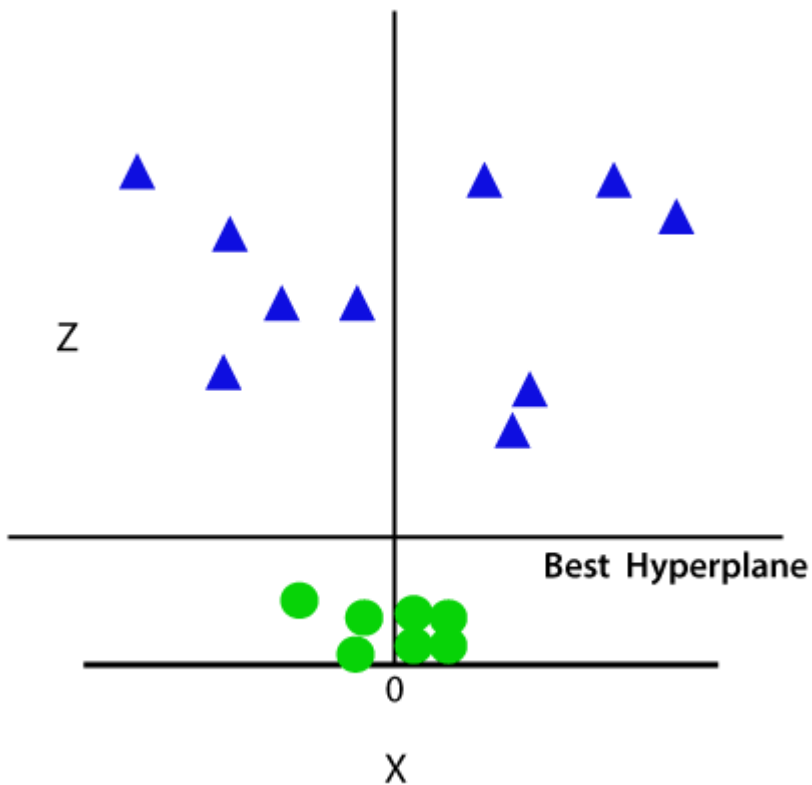
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z = x^2 + y^2$$

By adding the third dimension, the sample space will become as below image:

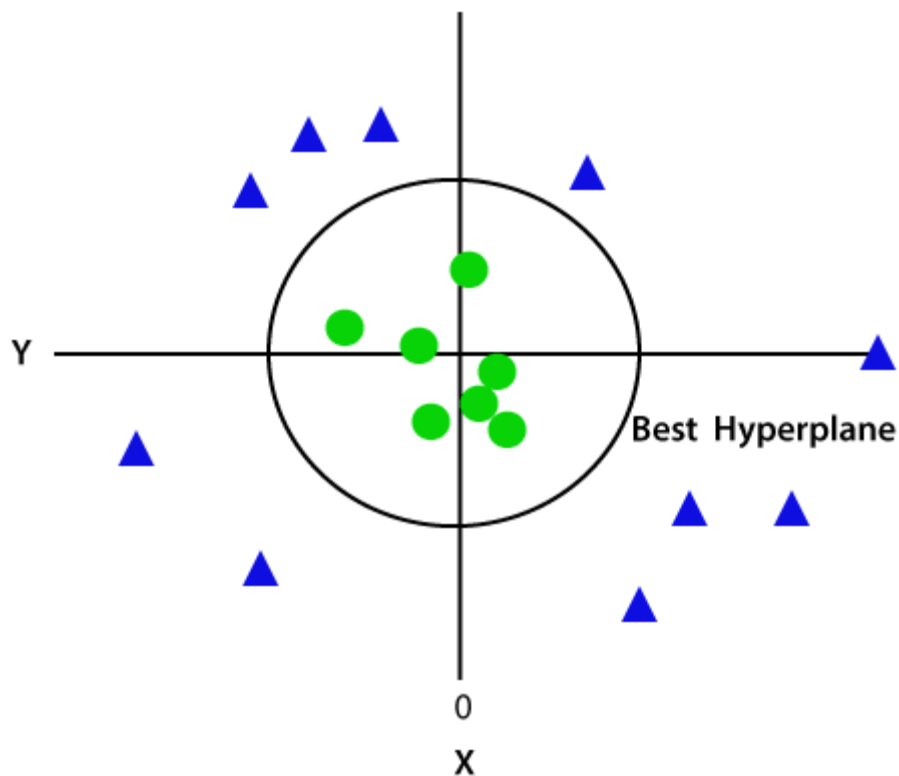


So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with  $z=1$ , then it will become as:





Hence we get a circumference of radius 1 in case of non-linear data.

### Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset **user\_data**, which we have used in Logistic regression and KNN classification.

#### ○ Data Pre-processing step

Till the Data pre-processing step, the code will remain the same. Below is the code:

1. #Data Pre-processing Step
2. # importing libraries
3. **import** numpy as nm
4. **import** matplotlib.pyplot as mtp
5. **import** pandas as pd
- 6.
7. #importing datasets
8. data\_set= pd.read\_csv('user\_data.csv')
- 9.

```

10. #Extracting Independent and dependent Variable
11. x= data_set.iloc[:, [2,3]].values
12. y= data_set.iloc[:, 4].values
13.
14. # Splitting the dataset into training and test set.
15. from sklearn.model_selection import train_test_split
16. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
17. #feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)
21. x_test= st_x.transform(x_test)

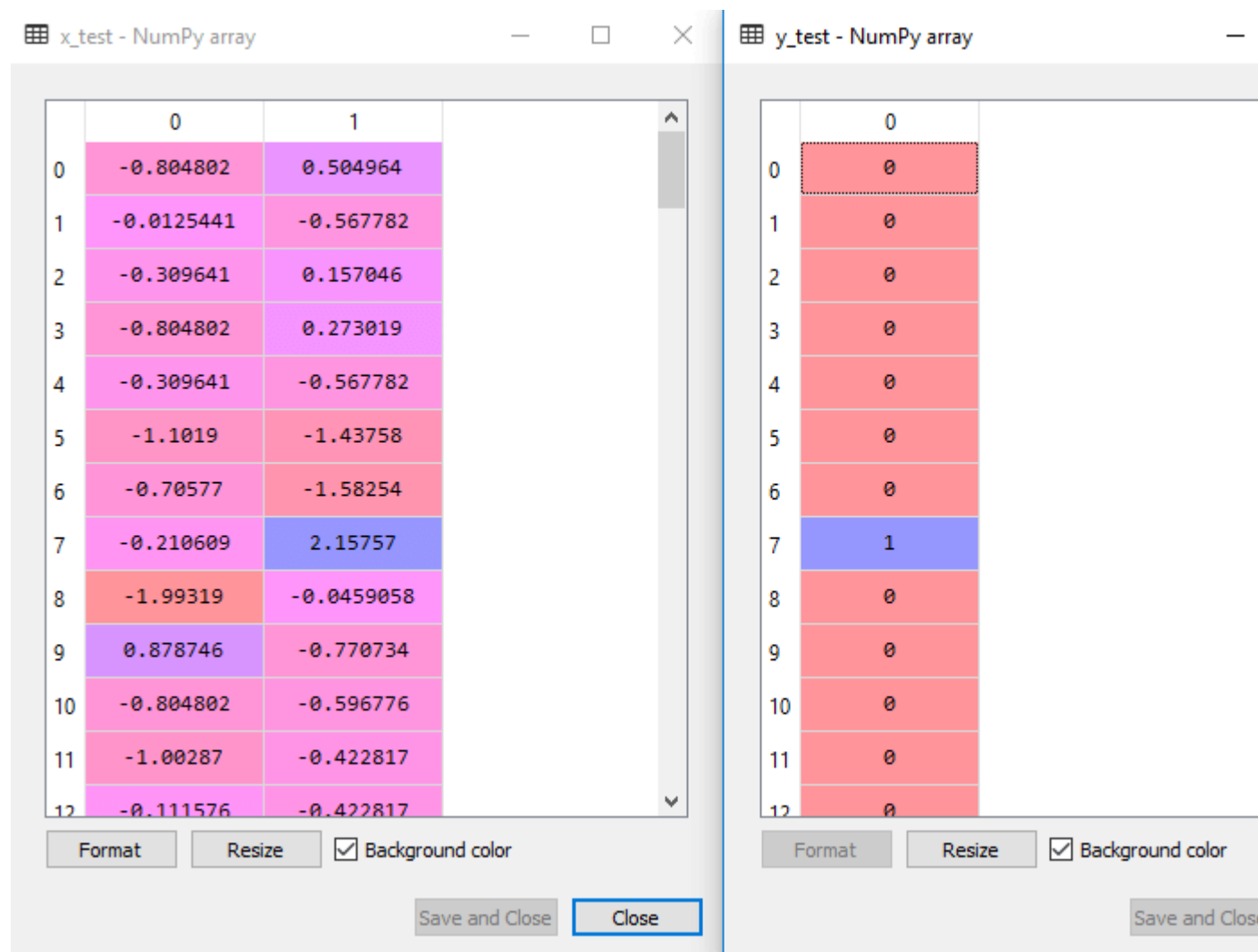
```

After executing the above code, we will pre-process the data. The code will give the dataset as:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0

Format    Resize    ☒ Background color    ☒ Column min/max    Save and Close    Close

The scaled output for the test set will be:



	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

### Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import **SVC** class from **Sklearn.svm** library. Below is the code for it:

1. from sklearn.svm **import** SVC # "Support vector classifier"
2. classifier = SVC(kernel='linear', random\_state=0)
3. classifier.fit(x\_train, y\_train)

In the above code, we have used **kernel='linear'**, as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x\_train, y\_train)

## Output:

```
Out[8]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

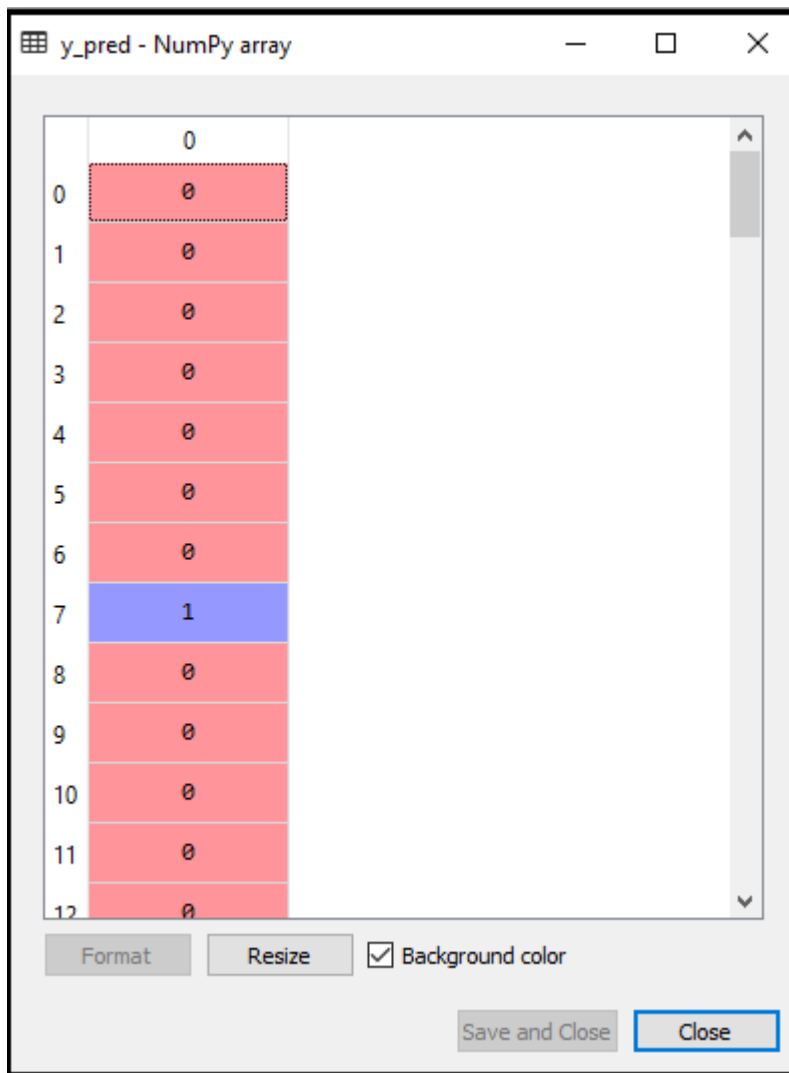
The model performance can be altered by changing the value of **C(Regularization factor)**, **gamma**, and **kernel**.

- **Predicting the test set result:**  
Now, we will predict the output for test set. For this, we will create a new vector **y\_pred**.  
Below is the code for it:

1. #Predicting the test set result
2. `y_pred= classifier.predict(x_test)`

After getting the **y\_pred** vector, we can compare the result of **y\_pred** and **y\_test** to check the difference between the actual value and predicted value.

**Output:** Below is the output for the prediction of the test set:

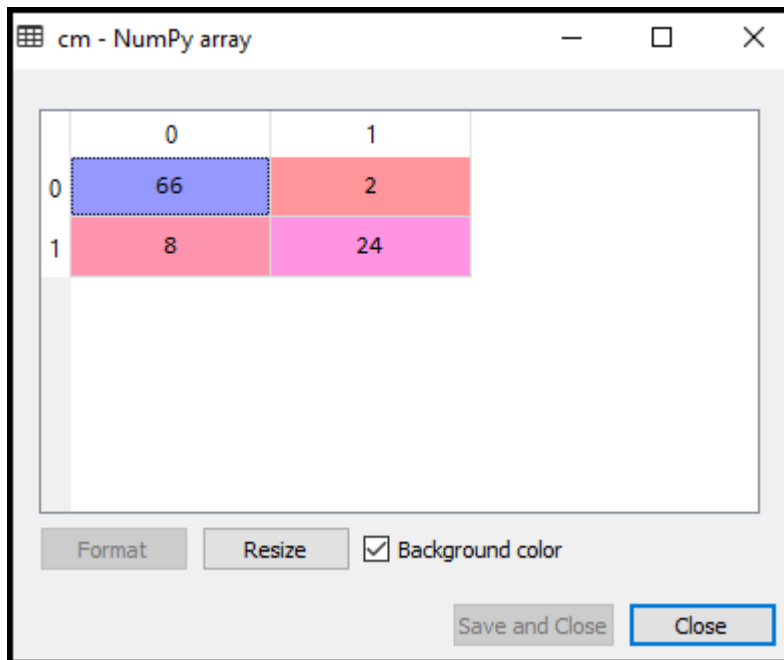


#### ○ Creating the confusion matrix:

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the **confusion\_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y\_true**( the actual values) and **y\_pred** (the targeted value return by the classifier). Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix(y\_test, y\_pred)

**Output:**



As we can see in the above output image, there are  $66+24=90$  correct predictions and  $8+2=10$  incorrect predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

#### ○ Visualizing the training set result:

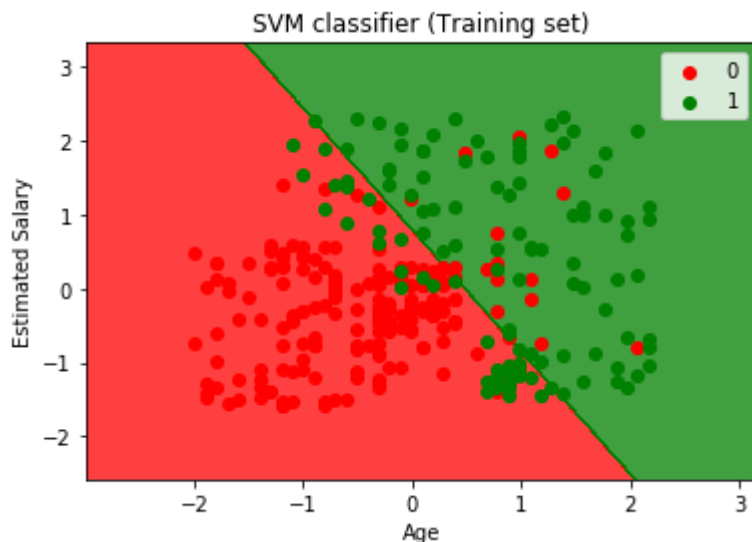
Now we will visualize the training set result, below is the code for it:

1. `from matplotlib.colors import ListedColormap`
2. `x_set, y_set = x_train, y_train`
3. `x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),`
4. `nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))`
5. `mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),`
6. `alpha = 0.75, cmap = ListedColormap(('red', 'green')))`
7. `mtp.xlim(x1.min(), x1.max())`
8. `mtp.ylim(x2.min(), x2.max())`
9. `for i, j in enumerate(nm.unique(y_set)):`
10. `mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],`
11. `c = ListedColormap(('red', 'green'))(i), label = j)`
12. `mtp.title('SVM classifier (Training set)')`
13. `mtp.xlabel('Age')`

14. mtp.ylabel('Estimated Salary')
15. mtp.legend()
16. mtp.show()

### Output:

By executing the above code, we will get the output as:



As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have **used a linear kernel in the classifier**. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

#### ○ Visualizing the test set result:

1. #Visulaizing the test set result
2. from matplotlib.colors **import** ListedColormap
3. x\_set, y\_set = x\_test, y\_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),  
1, stop = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))
5. nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),  
7. alpha = 0.75, cmap = ListedColormap(('red','green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())

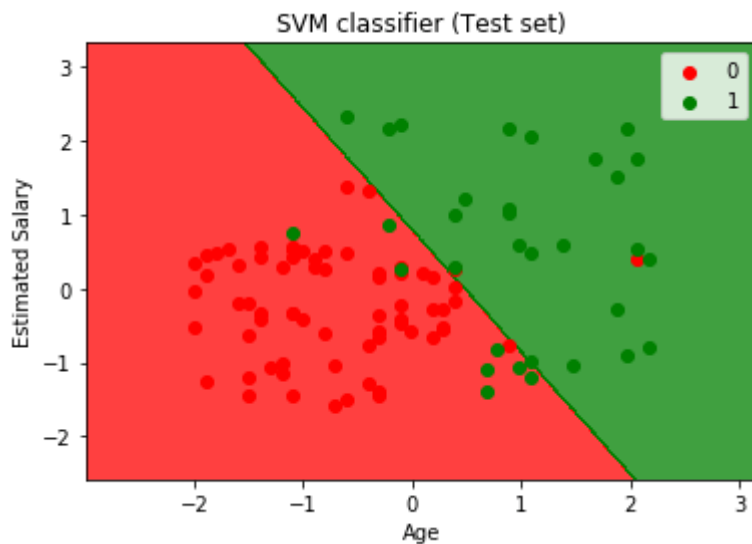
```

10. for i, j in enumerate(nm.unique(y_set)):
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.                 c = ListedColormap(('red', 'green'))(i), label = j)
13. mtp.title('SVM classifier (Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

### Output:

By executing the above code, we will get the output as:



As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased). Users who purchased the SUV are in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

### 3.Random Forest:

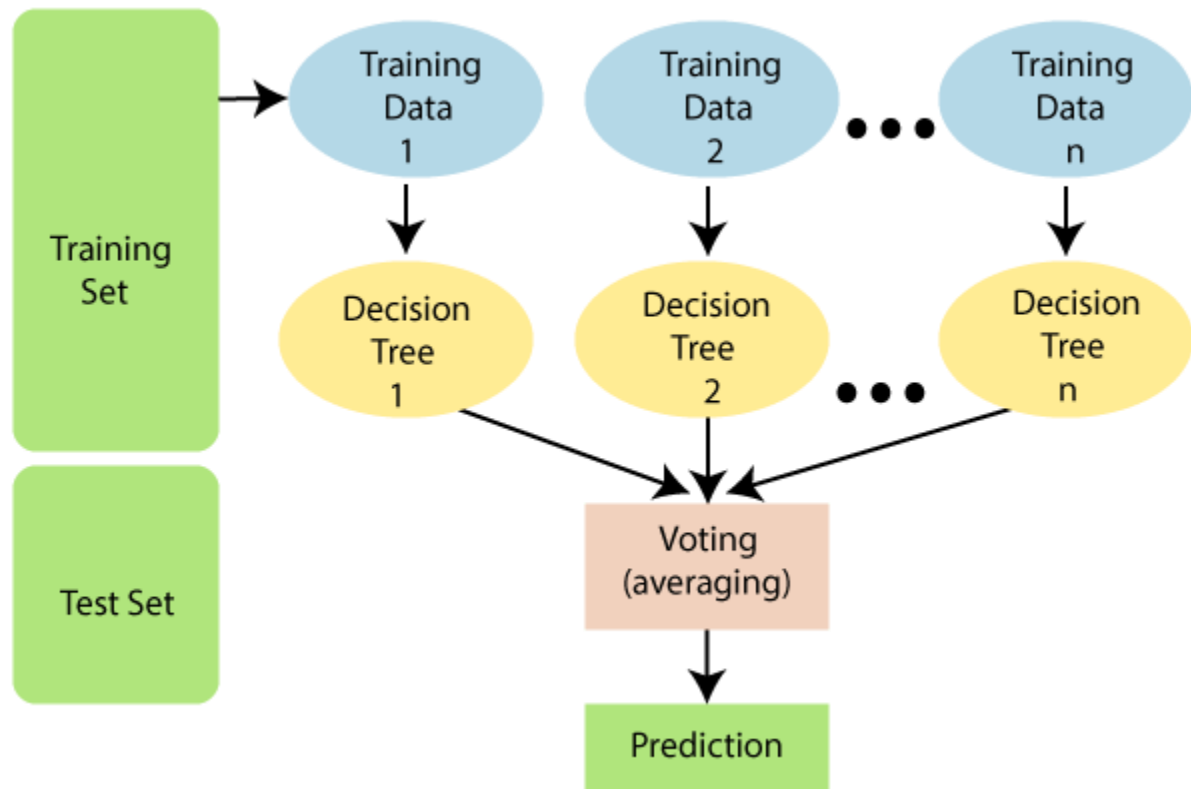
Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning**, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model*.

As the name suggests, *"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."* Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.



**The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.**

The below diagram explains the working of the Random Forest algorithm:



### Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

### Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

<="" li="">

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

### How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

**Step-1:** Select random K data points from the training set.

**Step-2:** Build the decision trees associated with the selected data points (Subsets).

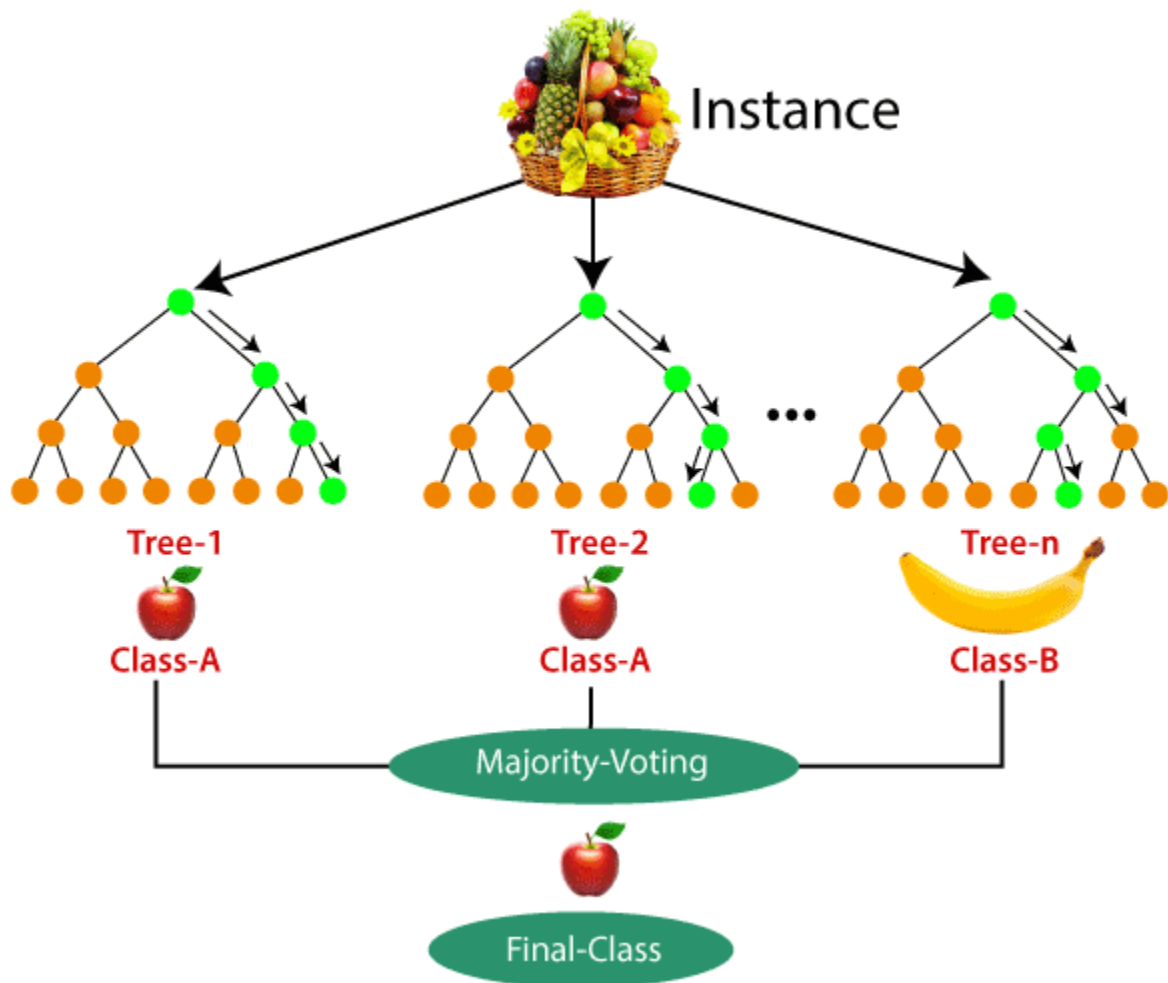
**Step-3:** Choose the number N for decision trees that you want to build.

**Step-4:** Repeat Step 1 & 2.

**Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

**Example:** Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



## Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

## Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

## Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

## Python Implementation of Random Forest Algorithm

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user\_data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as [Decision tree Classifier](#).

Implementation Steps are given below:

- Data Pre-processing step
- Fitting the Random forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

### 1.Data Pre-Processing Step:

Below is the code for the pre-processing step:

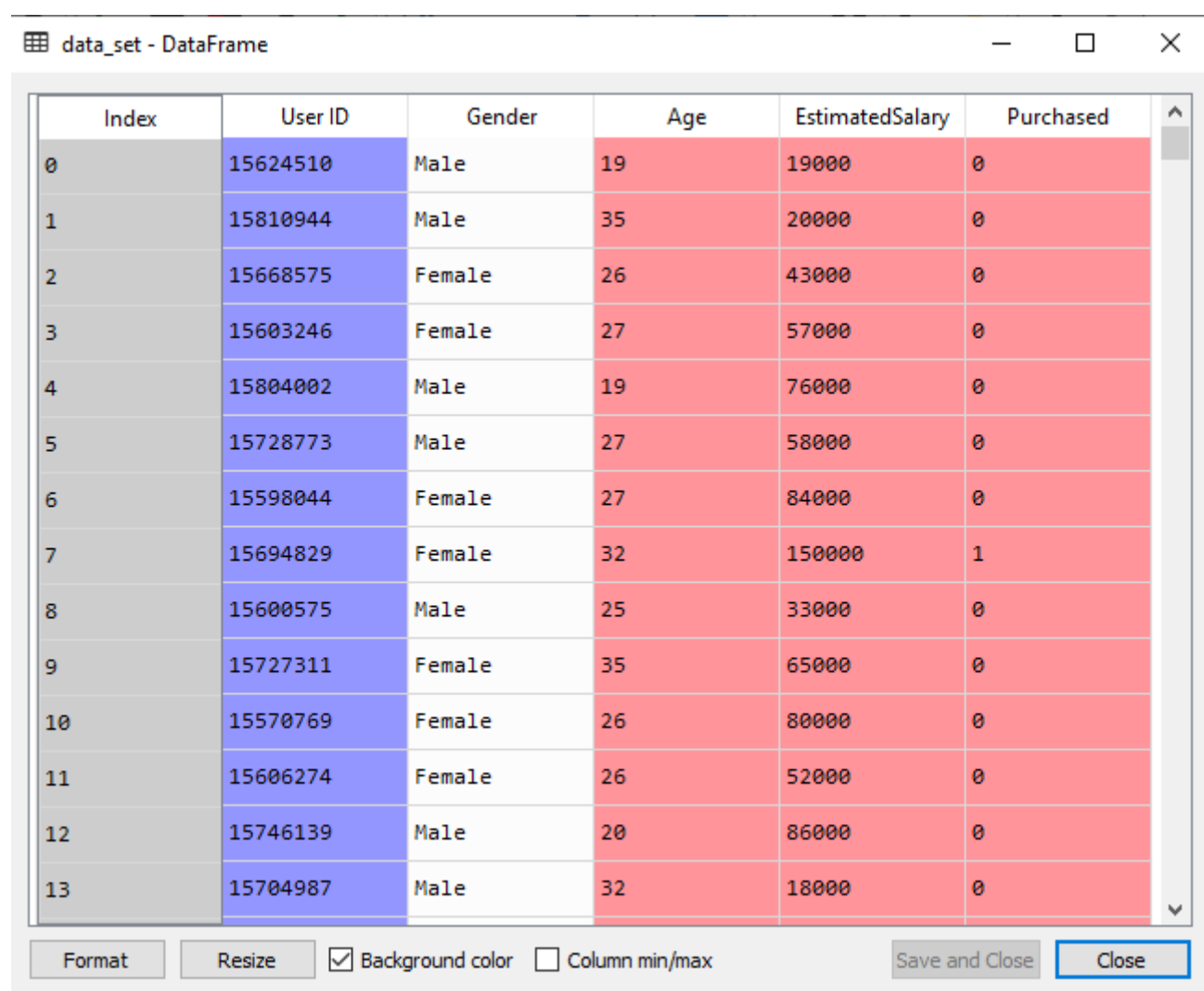
```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')
8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, [2,3]].values
11. y= data_set.iloc[:, 4].values
12.
13. # Splitting the dataset into training and test set.
```

```

14. from sklearn.model_selection import train_test_split
15. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
16.
17. #feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)
21. x_test= st_x.transform(x_test)

```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0

## 2. Fitting the Random Forest algorithm to the training set:

Now we will fit the Random forest algorithm to the training set. To fit it, we will import the **RandomForestClassifier** class from the **sklearn.ensemble** library. The code is given below:

1. #Fitting Decision Tree classifier to the training set
2. from sklearn.ensemble **import** RandomForestClassifier
3. classifier= RandomForestClassifier(n\_estimators= **10**, criterion="**entropy**")
4. classifier.fit(x\_train, y\_train)

In the above code, the classifier object takes below parameters:

- **n\_estimators**= The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- **criterion**= It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

#### Output:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
```

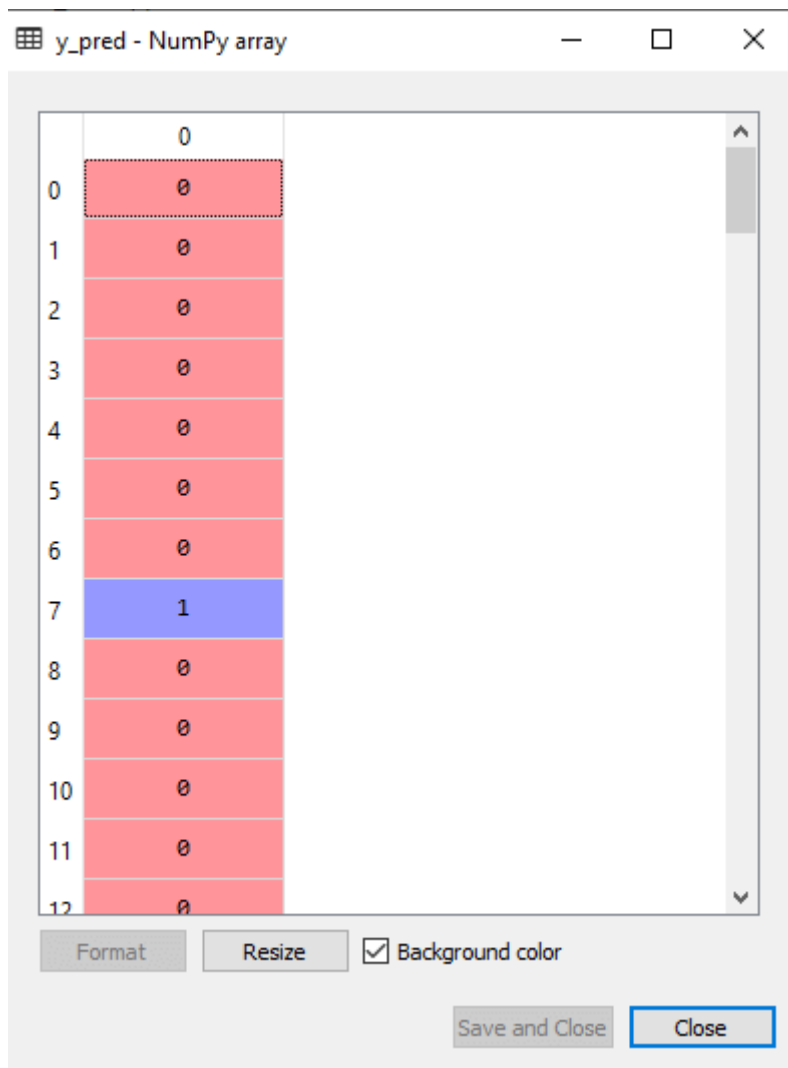
### 3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector y\_pred. Below is the code for it:

1. #Predicting the test set result
2. y\_pred= classifier.predict(x\_test)

#### Output:

The prediction vector is given as:



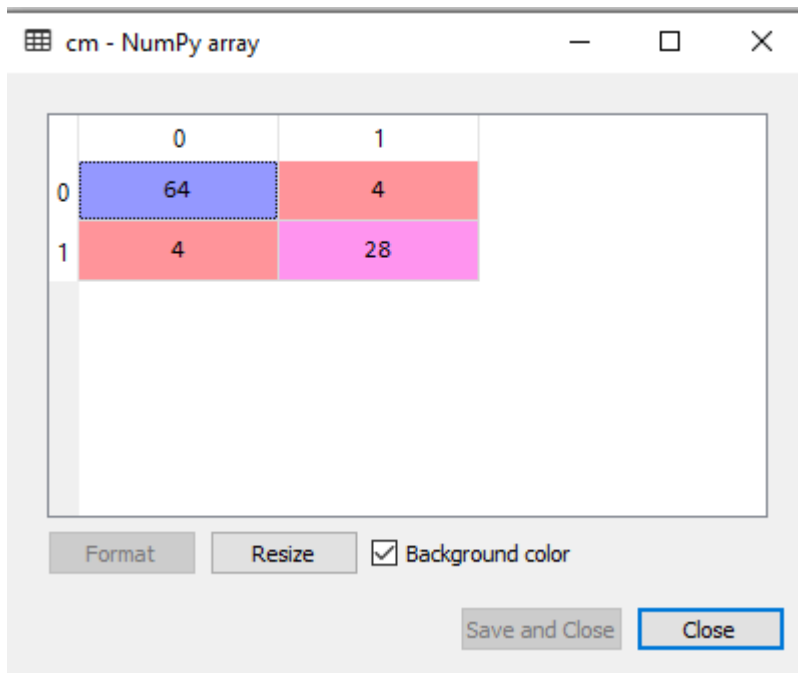
By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier.

#### 4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix(y\_test, y\_pred)

**Output:**



As we can see in the above matrix, there are  $4+4= 8$  incorrect predictions and  $64+28= 92$  correct predictions.

## 5. Visualizing the training Set result

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the Random forest classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in [Logistic Regression](#).

Below is the code for it:

1. from matplotlib.colors **import** ListedColormap
2. x\_set, y\_set = x\_train, y\_train
3. x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step = 0.01),  
nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))
4. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),  
alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
5. mtp.xlim(x1.min(), x1.max())
6. mtp.ylim(x2.min(), x2.max())
7. **for** i, j in enumerate(nm.unique(y\_set)):
8. mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],  
c = ListedColormap(('purple', 'green'))(i), label = j)

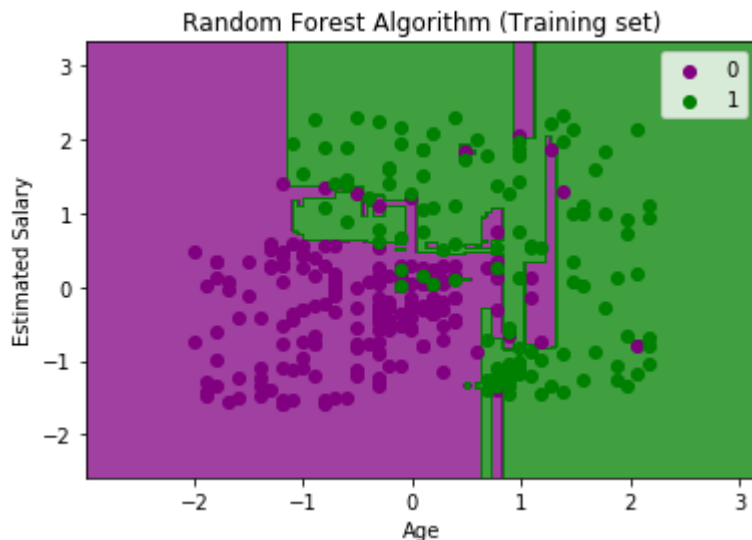


```

12. mtp.title('Random Forest Algorithm (Training set)')
13. mtp.xlabel('Age')
14. mtp.ylabel('Estimated Salary')
15. mtp.legend()
16. mtp.show()

```

### Output:



The above image is the visualization result for the Random Forest classifier working with the training set result. It is very much similar to the Decision tree classifier. Each data point corresponds to each user of the user\_data, and the purple and green regions are the prediction regions. The purple region is classified for the users who did not purchase the SUV car, and the green region is for the users who purchased the SUV.

So, in the Random Forest classifier, we have taken 10 trees that have predicted Yes or NO for the Purchased variable. The classifier took the majority of the predictions and provided the result.

## 6. Visualizing the test set result

Now we will visualize the test set result. Below is the code for it:

```

1. #Visulaizing the test set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_test, y_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
    1, stop = x_set[:, 0].max() + 1, step =0.01),

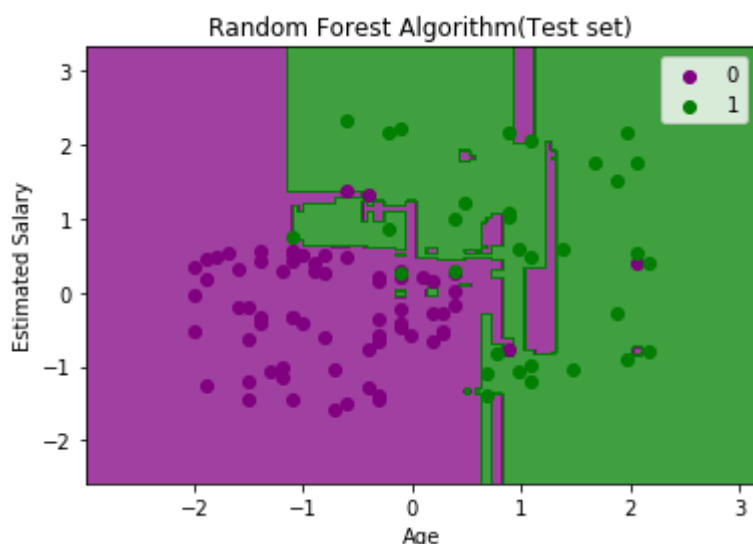
```

```

5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(['purple', 'green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.         c = ListedColormap(['purple', 'green'])(i), label = j)
13. mtp.title('Random Forest Algorithm(Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

**Output:**



The above image is the visualization result for the test set. We can check that there is a minimum number of incorrect predictions (8) without the Overfitting issue. We will get different results by changing the number of trees in the classifier.

## Performance Metrics for Classification Machine Learning Problems:

Many learning algorithms have been proposed. It is often valuable to assess the efficacy of an algorithm. In many cases, such assessment is relative, that is, evaluating which of several alternative algorithms is best suited to a specific application.

People even end up creating metrics that suit the application. In this article, we will see some of the most common metrics in a classification setting of a problem.

The most commonly used Performance metrics for classification problem are as follows,

- Accuracy
- Confusion Matrix
- Precision, Recall, and F1 score
- ROC AUC
- Log-loss

## Accuracy

Accuracy is the simple ratio between the number of correctly classified points to the total number of points.

To calculate accuracy, scikit-learn provides a utility function.

```
from sklearn.metrics import accuracy_score#predicted y values
y_pred = [0, 2, 1, 3]#actual y values
y_true = [0, 1, 2, 3]accuracy_score(y_true, y_pred)
0.5
```

Accuracy is simple to calculate but has its own disadvantages.

## Limitations of accuracy

- If the data set is highly imbalanced, and the model classifies all the data points as the majority class data points, the accuracy will be high. This makes accuracy not a reliable performance metric for imbalanced data.

- From accuracy, the probability of the predictions of the model can be derived. So from accuracy, we can not measure how good the predictions of the model are.

## Confusion Matrix

Confusion Matrix is a summary of predicted results in specific table layout that allows visualization of the performance measure of the machine learning model for a binary classification problem (2 classes) or multi-class classification problem (more than 2 classes)

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Confusion matrix of a binary classification

- TP means **True Positive**. It can be interpreted as the model predicted positive class and it is True.
- FP means **False Positive**. It can be interpreted as the model predicted positive class but it is False.

- FN means **False Negative**. It can be interpreted as the model predicted negative class but it is False.
- TN means **True Negative**. It can be interpreted as the model predicted negative class and it is True.

*For a sensible model, the principal diagonal element values will be high and the off-diagonal element values will be below i.e., TP, TN will be high.*

To get an appropriate example in a real-world problem, consider a diagnostic test that seeks to determine whether a person has a certain disease. A false positive in this case occurs when the person tests positive but does not actually have the disease. A false negative, on the other hand, occurs when the person tests negative, suggesting they are healthy when they actually do have the disease.

For a multi-class classification problem, with 'c' class labels, the confusion matrix will be a (c\*c) matrix.

To calculate confusion matrix, sklearn provides a utility function

```
from sklearn.metrics import confusion_matrix
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

### **Advantages of a confusion matrix:**

- The confusion matrix provides detailed results of the classification.
- Derivates of the confusion matrix are widely used.
- Visual inspection of results can be enhanced by using a heat map.

## Precision, Recall, and F-1 Score

**Precision** is the fraction of the correctly classified instances from the total classified instances. **Recall** is the fraction of the correctly classified instances from the total classified instances. Precision and recall are given as follows,

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

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

Mathematical formula of

Precision and Recall using the confusion matrix

For example, consider that a search query results in 30 pages, out of which 20 are relevant. And the results fail to display 40 other relevant results. So the precision is 20/30 and recall is 20/60.

Precision helps us understand how useful the results are. Recall helps us understand how complete the results are.

But to reduce the checking of pockets twice, the F1 score is used. F1 score is the harmonic mean of precision and recall. It is given as,

$$F1 \text{ score} = \frac{2 * Precision * Recall}{Precision + Recall}$$

## When to use the F1 Score?

- The F-score is often used in the field of information retrieval for measuring search, document classification, and query classification performance.

- The F-score has been widely used in the natural language processing literature, such as the evaluation of named entity recognition and word segmentation.

## Log Loss

Logarithmic loss (or log loss) measures the performance of a classification model where the prediction is a probability value between 0 and 1. Log loss increases as the predicted probability diverge from the actual label. Log loss is a widely used metric for Kaggle competitions.

$$\text{log-loss} = -\frac{1}{N} \sum_{i=1}^N y_i \log p_i + (1 - y_i) \log(1 - p_i).$$

Here 'N' is the total number of data points in the data set,  $y_i$  is the actual value of  $y$  and  $p_i$  is the probability of  $y$  belonging to the positive class.

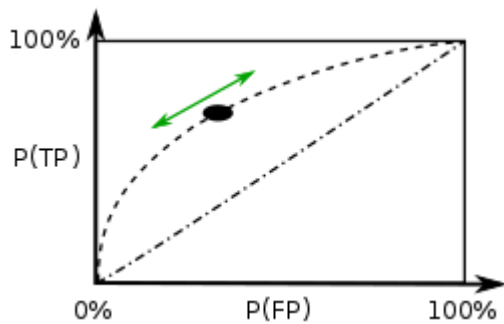
Lower the log-loss value, better are the predictions of the model.

To calculate log-loss, scikit-learn provides a utility function.

```
from sklearn.metrics import log_losslog_loss(y_true, y_pred)
```

## ROC AUC

A **Receiver Operating Characteristic curve** or **ROC curve** is created by plotting the True Positive (TP) against the False Positive (FP) at various threshold settings. The ROC curve is generated by plotting the [cumulative distribution function](#) of the True Positive in the y-axis versus the cumulative distribution function of the False Positive on the x-axis.



The dashed curved line is the ROC Curve

The area under the ROC curve (ROC AUC) is the single-valued metric used for evaluating the performance.

*The higher the AUC, the better the performance of the model at distinguishing between the classes.*

In general, an AUC of 0.5 suggests no discrimination, a value between 0.5–0.7 is acceptable and anything above 0.7 is good-to-go-model. However, medical diagnosis models, usually AUC of 0.95 or more is considered to be good-to-go-model.

### When to use ROC?

- ROC curves are widely used to compare and evaluate different classification algorithms.
- ROC curve is widely used when the dataset is imbalanced.
- ROC curves are also used in verification of forecasts in meteorology

## Ensemble methods

Ensemble methods are **techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model**. The combined models increase the accuracy of the results significantly. This has boosted the popularity of ensemble methods in machine learning.



The three main classes of ensemble learning methods are **bagging, stacking, and boosting**, and it is important to both have a detailed understanding of each method and to consider them on your predictive modeling project.

## **Bagging:**

Bagging, also known as Bootstrap aggregating, is **an ensemble learning technique that helps to improve the performance and accuracy of machine learning algorithms**. It is used to deal with bias-variance trade-offs and reduces the variance of a prediction model.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement,  $N$  examples (or data) from the original training dataset – *where  $N$  is the size of the original training set*. Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out.

Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

### **How Bagging works on training dataset ?**

How bagging works on an imaginary training dataset is shown below. Since Bagging resamples the original training dataset with replacement, some instance (or data) may be present multiple times while others are left out.

**Original training dataset:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

**Resampled training set 1:** 2, 3, 3, 5, 6, 1, 8, 10, 9, 1

**Resampled training set 2:** 1, 1, 5, 6, 3, 8, 9, 10, 2, 7

**Resampled training set 3:** 1, 5, 8, 9, 2, 10, 9, 7, 5, 4

Algorithm for the Bagging classifier:

#### **Classifier generation:**

Let  $N$  be the size of the training set.

for each of  $t$  iterations:

    sample  $N$  instances with replacement from the original training set.

    apply the learning algorithm to the sample.

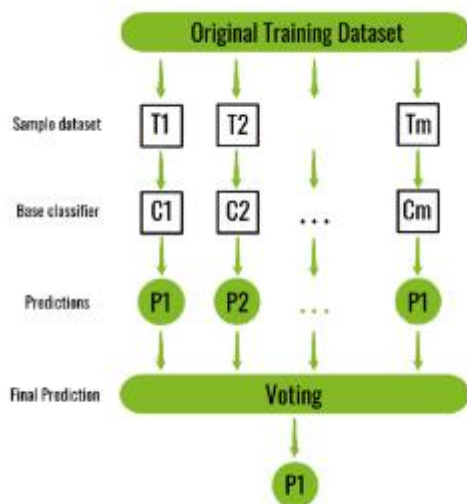
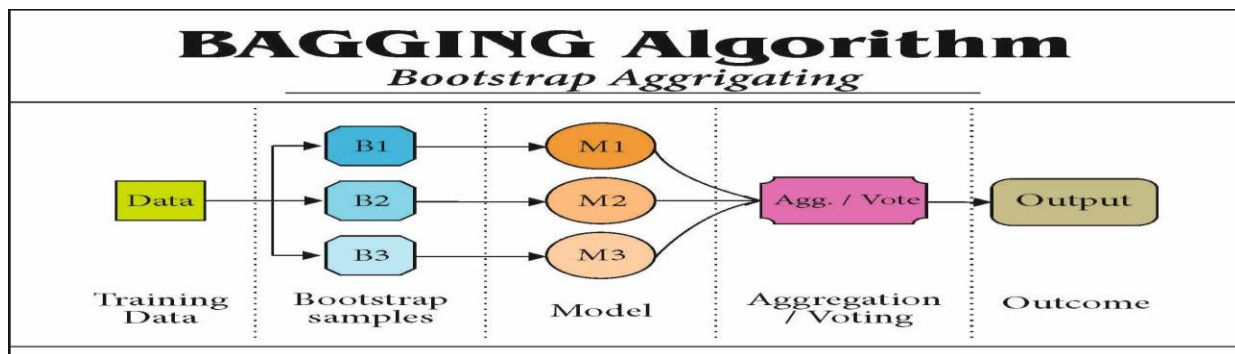
    store the resulting classifier.

#### **Classification:**

for each of the  $t$  classifiers:

    predict class of instance using classifier.

return class that was predicted most often.



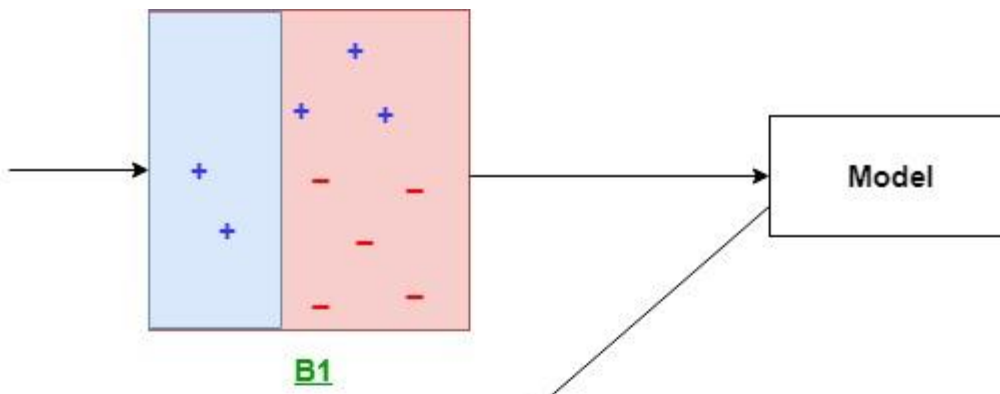
### BOOSTING:

**Boosting** is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

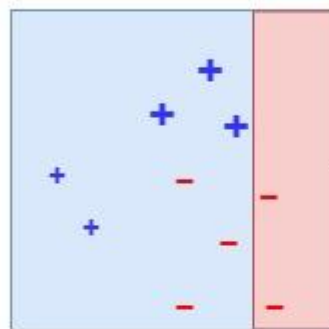
**AdaBoost** :It is the first really successful boosting algorithm developed for the purpose of binary classification. *AdaBoost* is short for *Adaptive Boosting* and is a very popular boosting technique that combines multiple “weak classifiers” into a single “strong classifier”. It was formulated by Yoav Freund and Robert Schapire. They also won the 2003 Gödel Prize for their work.

### Algorithm:

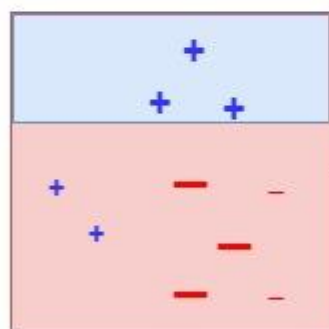
1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points.
4. if (got required results)
  - Goto step 5
  - else
  - Goto step 2
5. End



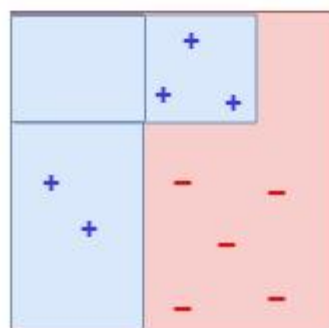
Model



Model



Model

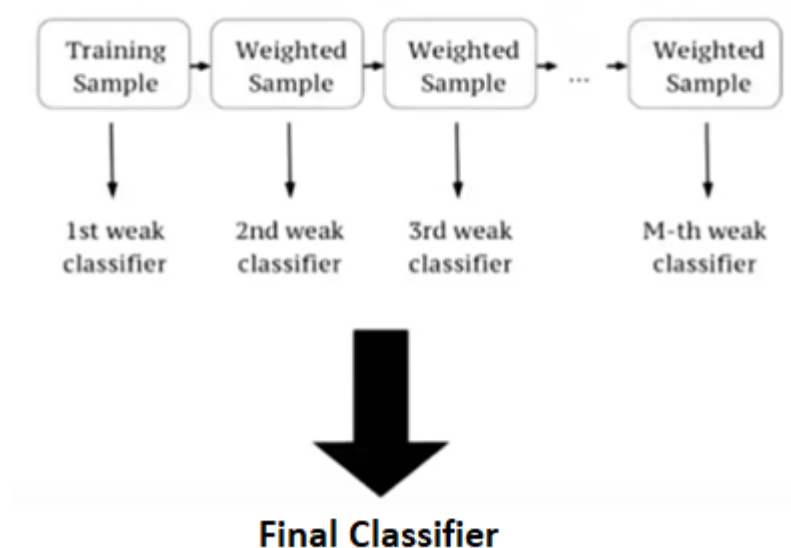


### Explanation:

The above diagram explains the AdaBoost algorithm in a very simple way. Let's try to understand it in a stepwise process:

- **B1** consists of 10 data points which consist of two types namely plus(+) and minus(-) and 5 of which are plus(+) and the other 5 are minus(-) and each one has been assigned equal weight initially. The first model tries to classify the data points and generates a vertical separator line but it wrongly classifies 3 plus(+) as minus(-).
- **B2** consists of the 10 data points from the previous model in which the 3 wrongly classified plus(+) are weighted more so that the current model tries more to classify these pluses(+) correctly. This model generates a vertical separator line that correctly classifies the previously wrongly classified pluses(+) but in this attempt, it wrongly classifies three minuses(-).
- **B3** consists of the 10 data points from the previous model in which the 3 wrongly classified minus(-) are weighted more so that the current model tries more to classify these minuses(-) correctly. This model generates a horizontal separator line that correctly classifies the previously wrongly classified minuses(-).
- **B4** combines together B1, B2, and B3 in order to build a strong prediction model which is much better than any individual model used.

**XG Boost:** is an implementation of Gradient Boosted decision trees. This library was written in C++. It is a type of Software library that was designed basically to improve speed and model performance. It has recently been dominating in applied machine learning. XGBoost models majorly dominate in many Kaggle Competitions. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and the variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.



**XGBoost Features** The library is laser-focused on computational speed and model performance, as such, there are few frills. **Model Features** Three main forms of gradient boosting are supported:

- Gradient Boosting
- Stochastic Gradient Boosting
- Regularized Gradient Boosting

### **System Features**

- For use of a range of computing environments this library provides-
- Parallelization of tree construction
- Distributed Computing for training very large models
- Cache Optimization of data structures and algorithm

### **XGBoost enhancements/optimizations**

XGBoost features various optimizations built-in to make the training faster when working with large datasets, in addition to its unique method of generating and pruning trees. Here is a handful of the most significant:

- **Approximate Greedy Algorithm:** instead of assessing every candidate split, this algorithm employs weighted quantiles to find the best node split.
- **Cash-Aware Access:** XGBoost stores data in the CPU's cache memory.
- **Sparsity:** Aware Split Finding calculates Gain by putting observations with missing values onto the left leaf when there is some missing data. It then repeats the process by placing them in the appropriate leaf and selecting the scenario with the highest Gain.

**Steps to Install Windows** XGBoost uses Git submodules to manage dependencies. So when you clone the repo, remember to specify `--recursive` option:

```
git clone --recursive https://github.com/dmlc/xgboost
```

For Windows users who use Github tools, you can open the git-shell and type the following command:

```
git submodule init
git submodule update
```

**OSX(Mac)** First, obtain gcc-8 with Homebrew (<https://brew.sh/>) to enable multi-threading (i.e. using multiple CPU threads for training). The default Apple Clang compiler does not support OpenMP, so using the default compiler would have disabled multi-threading.

```
brew install gcc@8
```

Then install XGBoost with pip:

```
pip3 install xgboost
```

You might need to run the command with `--user` flag if you run into permission errors.



