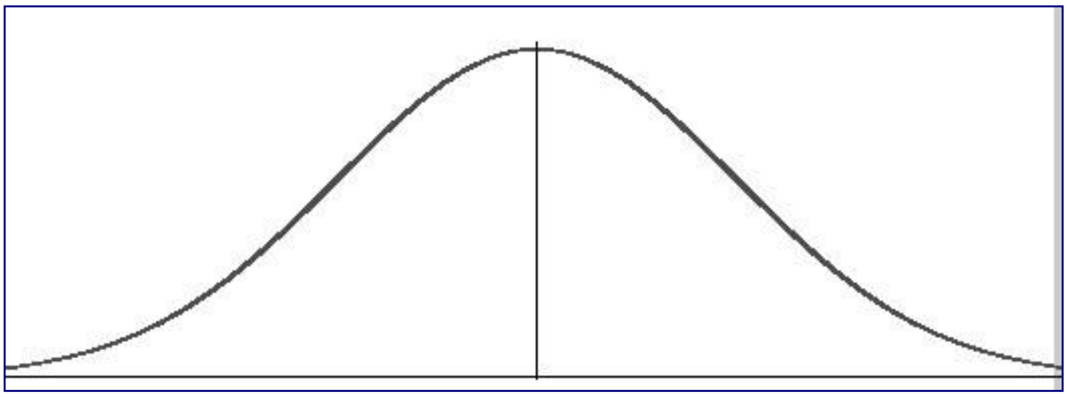
What is a Probabilistic Model?

Probabailistic models incorporate random variables and probability distributions into the model of an event or phenomenon. While a deterministic model gives a single possible outcome for an event, **a probabilistic model gives a probability distribution as a solution.** These models take into account the fact that we can rarely know everything about a situation. There's nearly always an element of randomness to take into account. For example, life insurance is based on the fact we know with certainty that we *will* die, but we don't know *when*. These models can be part deterministic and part random or wholly random.

Random variables from the normal distribution, binomial distribution and Bernoulli distribution form the foundation for this type of modeling.



A normal distribution curve, sometimes called a bell curve, is one of the building blocks of a probabilistic model.

What is the Probabilistic Method?

The probabilistic method, first introduced by Paul Erdős, is a way to prove the existence of a structure with certain properties in combinatorics. The idea is that you create a probability space, and — choosing elements at random — prove than any random element from the space has both a positive probability and the properties sought after. The method is widely used in a variety of disciplines, including: statistical physics, quantum mechanics, and theoretical computer science.

What is normal distribution:

A sample of data will form a **distribution**, and by far the most well-known **distribution** is the **Gaussian distribution**, often called the **Normal distribution**. The **distribution** provides a parameterized mathematical function that can be used to calculate the probability for any individual observation from the sample space.

Normal Distribution

The normal distribution is also called the Gaussian distribution (named for Carl Friedrich Gauss) or the bell curve distribution.

The distribution covers the probability of real-valued events from many different problem domains, making it a common and well-known distribution, hence the name "*normal.*" A continuous random variable that has a normal distribution is said to be "*normal*" or "*normally distributed.*"

Some examples of domains that have normally distributed events include:

- The heights of people.
- The weights of babies.
- The scores on a test.

The distribution can be defined using two parameters:

- **Mean** (*mu*): The expected value.
- **Variance** (*sigma^2*): The spread from the mean.

Often, the standard deviation is used instead of the variance, which is calculated as the square root of the variance, e.g. normalized.

- **Standard Deviation** (*sigma*): The average spread from the mean.

A distribution with a mean of zero and a standard deviation of 1 is called a standard normal distribution, and often data is reduced or "*standardized*" to this for analysis for ease of interpretation and comparison.

We can define a distribution with a mean of 50 and a standard deviation of 5 and sample random numbers from this distribution. We can achieve this using the normal() NumPy function.

The example below samples and prints 10 numbers from this distribution.

# sample a normal distribution from numpy.random import normal # define the distribution mu = 50 sigma = 5 n = 10 # generate the sample sample = normal(mu, sigma, n) print(sample)

```
# sample a normal distribution
from numpy.random import
normal
# define the distribution
mu = 50
sigma = 5
n = 10
# generate the sample
sample = normal(mu, sigma, n)
print(sample)
```

Running the example prints 10 numbers randomly sampled from the defined normal distribution

```
[48.71009029 49.36970461 45.58247748 51.96846616 46.05793544 40.3903483
48.39189421 50.08693721 46.85896352 44.83757824]
```

A sample of data can be checked to see if it is random by plotting it and checking for the familiar normal shape, or by using statistical tests. If the samples of observations of a random variable are normally distributed, then they can be summarized by just the mean and variance, calculated directly on the samples.

We can calculate the probability of each observation using the probability density function. A plot of these values would give us the tell-tale bell shape.

We can define a normal distribution using the norm() SciPy function and then calculate properties such as the moments, PDF, CDF, and more.

The example below calculates the probability for integer values between 30 and 70 in our distribution and plots the result, then does the same for the cumulative probability.

# pdf and cdf for a normal distribution from scipy.stats import norm from matplotlib import pyplot # define distribution parameters mu = 50 sigma = 5 # create distribution dist = norm(mu, sigma) # plot pdf values = [value for value in range(30, 70)] probabilities = [dist.pdf(value) for value in values] pyplot.plot(values, probabilities) pyplot.show() # plot cdf cprobs = [dist.cdf(value) for value in values] pyplot.plot(values, cprobs) pyplot.show()

```
# pdf and cdf for a normal distribution
from scipy.stats import norm
from matplotlib import pyplot
# define distribution parameters
mu = 50
sigma = 5
# create distribution
dist = norm(mu, sigma)
# plot pdf
values = [value for value in range(30,
70)]
probabilities = [dist.pdf(value) for
value in values]
pyplot.plot(values, probabilities)
pyplot.show()
# plot cdf
cprobs = [dist.cdf(value) for value in
values]
pyplot.plot(values, cprobs)
pyplot.show()
```

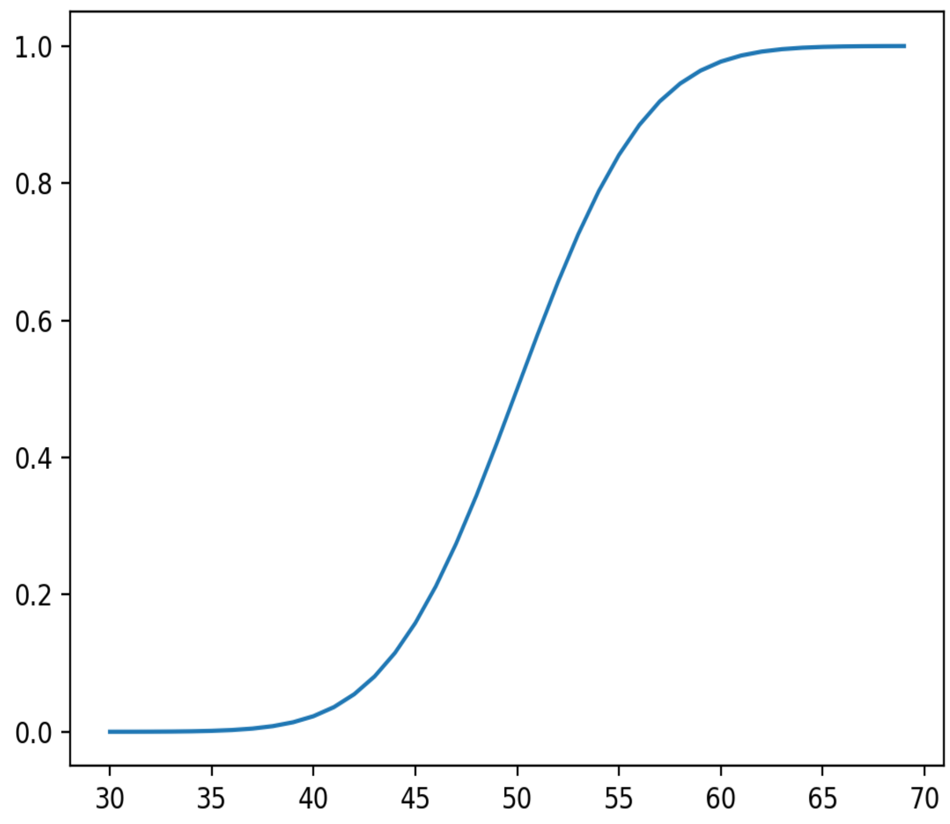
Running the example first calculates the probability for integers in the range [30, 70] and creates a line plot of values and probabilities.

The plot shows the Gaussian or bell-shape with the peak of highest probability around the expected value or mean of 50 with a probability of about 8%.

The linked image cannot be displayed. The file may have been moved, renamed, or deleted. Verify that the link points to the correct file and location.

Line Plot of Events vs Probability or the Probability Density Function for the Normal Distribution

The cumulative probabilities are then calculated for observations over the same range, showing that at the mean, we have covered about 50% of the expected values and very close to 100% after the value of about 65 or 3 standard deviations from the mean (50 + (3 * 5)).



Line Plot of Events vs. Cumulative Probability or the Cumulative Density Function for the Normal Distribution

In fact, the normal distribution has a heuristic or rule of thumb that defines the percentage of data covered by a given range by the number of standard deviations from the mean. It is called the 68-95-99.7 rule, which is the approximate percentage of the data covered by ranges defined by 1, 2, and 3 standard deviations from the mean.

For example, in our distribution with a mean of 50 and a standard deviation of 5, we would expect 95% of the data to be covered by values that are 2 standard deviations from the mean, or 50 – (2 * 5) and 50 + (2 * 5) or between 40 and 60.

We can confirm this by calculating the exact values using the percentage-point function.

The middle 95% would be defined by the percentage point function value for 2.5% at the low end and 97.5% at the high end, where 97.5 – 2.5 gives the middle 95%.

The complete example is listed below

```
# calculate the values that define the middle 95%
from scipy.stats import norm
# define distribution parameters
mu = 50
sigma = 5
# create distribution
dist = norm(mu, sigma)
```

```
low_end = dist.ppf(0.025)
high_end = dist.ppf(0.975)
print('Middle 95%% between %.1f and %.1f' %
(low_end, high_end))
```

Running the example gives the exact outcomes that define the middle 95% of expected outcomes that are very close to our standard-deviation-based heuristics of 40 and 60.

```
Middle 95% between 40.2 and 59.8
```

An important related distribution is the Log-Normal probability distribution.

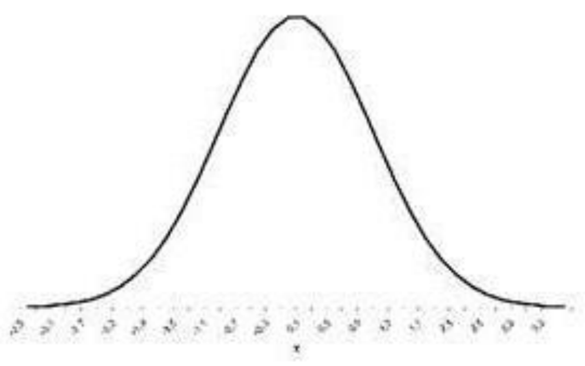
The theory behind normal distribution
1. if u r comfortable with coding example go with it.
Else
2. go for theory which is explained below

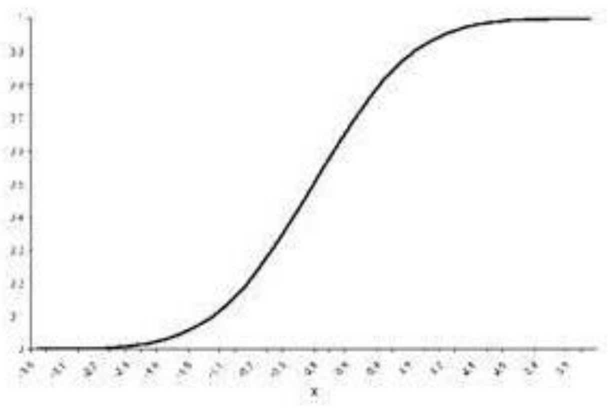
In data science and even in machine learning you will see one distribution so many times that dealing with that distribution will become so normal to you that you will call it "**Normal Distribution**".

What is Normal distribution?

Following formula gives the pdf(probability distribution function) of the normal distribution:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(\mu - x)^2}{2\sigma^2}}$$

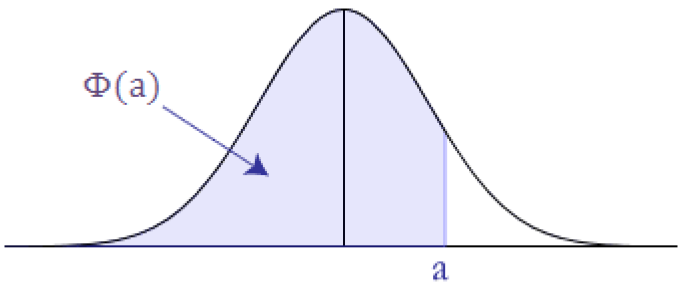

a) Normal Probability Density Function


b) Normal Cumulative Distribution Function

The mean of the normal distribution is μ (mu) and a standard deviation is σ sigma.

If you don't know pdf, in simple terms pdf gives you the "relative likelihood of a continuous random variable taking that value". In Normal distribution, it is like the bell-shaped curve.

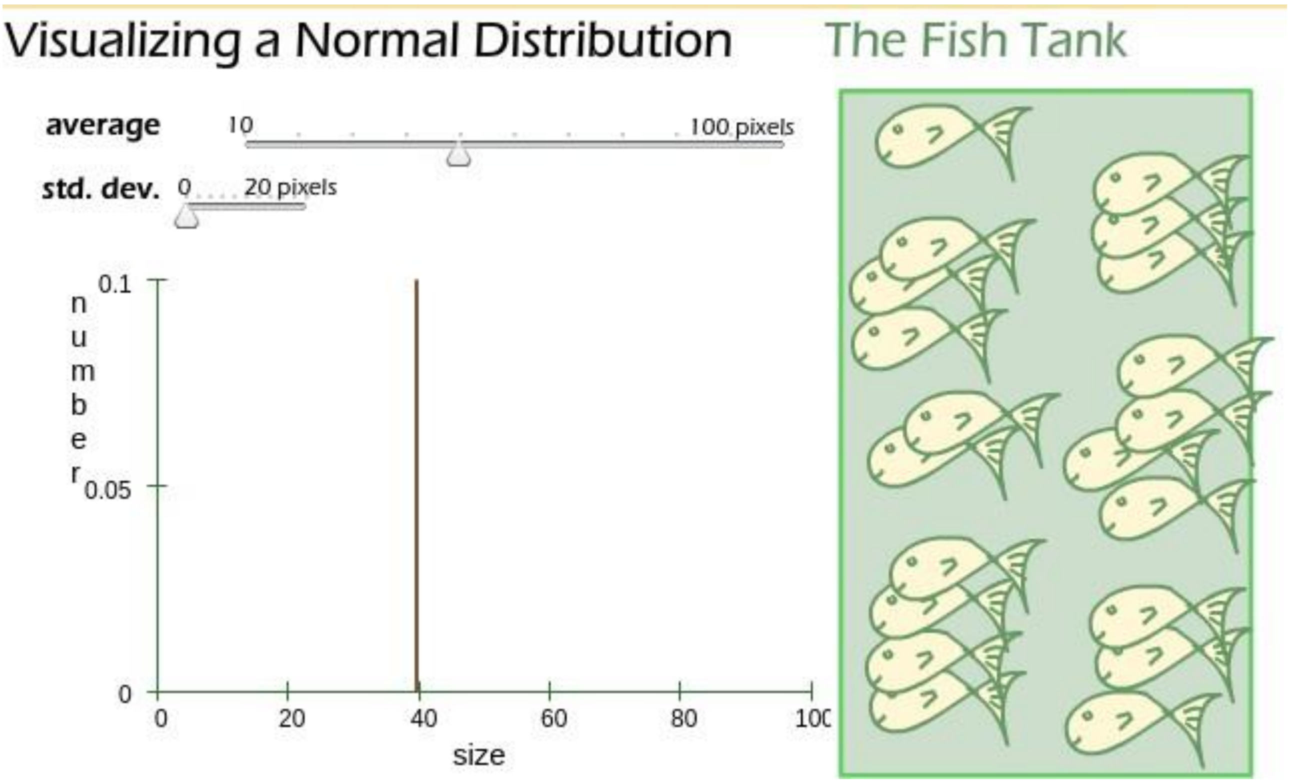
CDF (Cumulative Distribution Function) is nothing but the integration of pdf. In the normal distribution, it is shown as Φ(z). **Which is nothing but the probability that normally distributed random variable is less than value z.**



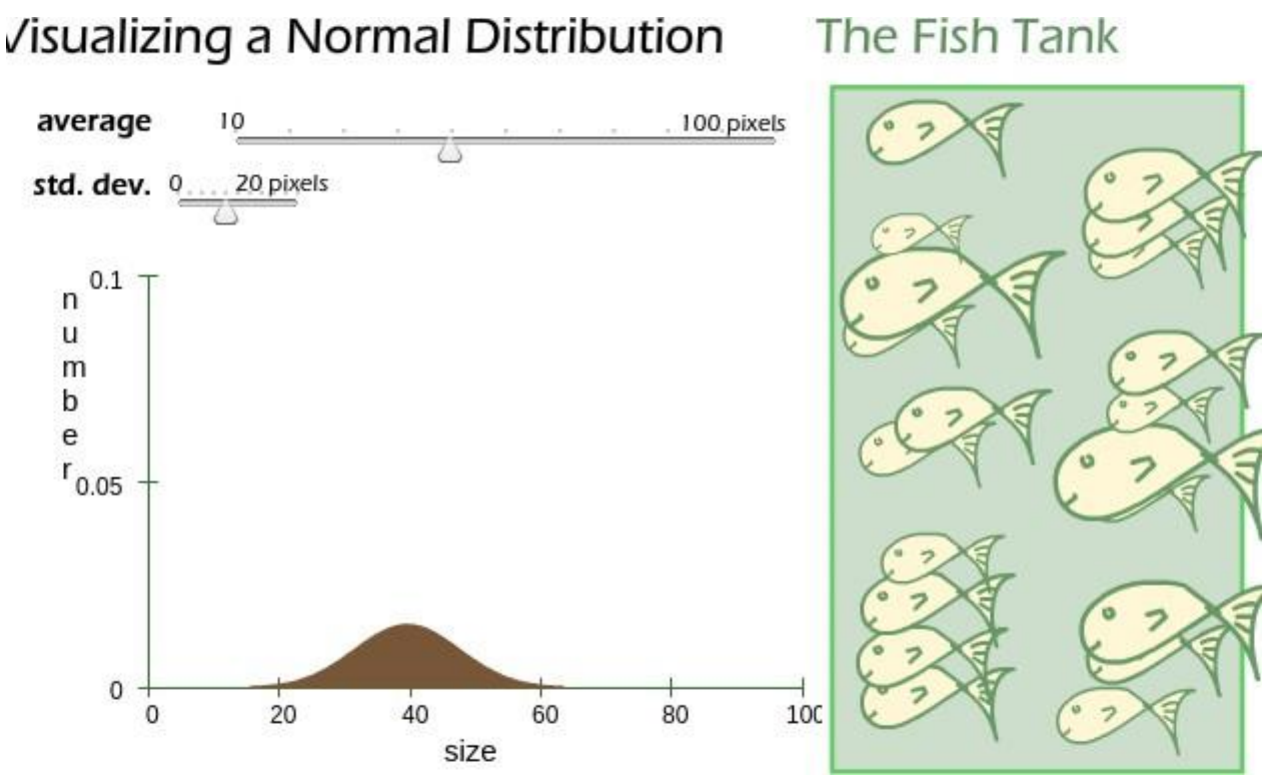
Φ(a)

In general, mean μ gives you the central value (where normal distribution pdf is at peak) while standard deviation (from here on I will refer it as std or σ) gives you the spread around mean. If the std is large, then our sample has large spread while if std is small then the sample is distributed very closely around the mean.

To have a clear idea of μ (mean) and a σ (standard deviation) follow following diagram.

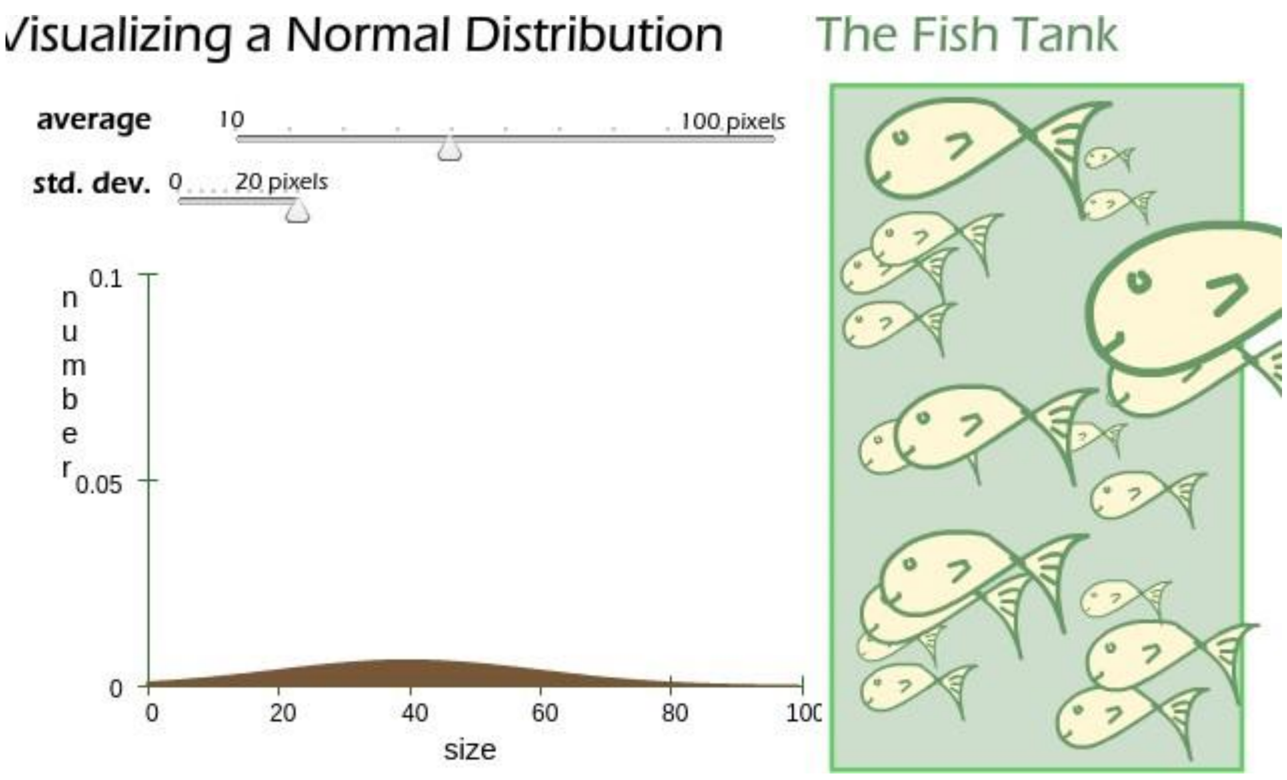


If the standard deviation is zero all sizes of fish is exactly the same and that of μ. You can see the spread of distribution is nothing but a straight verticle line.



When we increase standard deviation we can see that the size of fish varies around mean. Few of them have got bigger, while few of them have shrunk.

Now when we further increase the size of std you can notice the variance in fish size also increase. You can also notice the spread of the distribution is almost flat.



Standard Normal:

What will be the value of a normal distribution with mean zero and std one,

$$f_Z(z) = \frac{1}{\sqrt{2\pi}} \cdot e^{-0.5z^2}$$

Now as we have calculated z-scores that is nothing but the *probability that a normally distributed random variable Z, with mean equal to 0 and variance equal to 1, is less than or equal to z.*

These values are numerically computed and we can refer them using the z-score table.

Linear Transformation Property:

> **If X is a Normal random variable and if you did the linear transformation on X, the new random variable will also be distributed as a Normal distribution.**

X is a normal distribution $X \sim \text{Normal}(\mu, \sigma^2)$ and $Y = aX + b$ then, Y is also a normal distribution with parameters ($a\mu + b$, $a2\ \sigma^2$).

Now generally in problems with standard distribution, we have to find out the probability of random variable less than specific value, or if you are dealing with interval problems such as confidence interval (during hypothesis testing), you have to find out are in normal distribution lying between two points, in such case integrating above pdf equation is challenging task.

Now using the above Linear Transformation Property, using the following formula:

$$z = \frac{(x - \mu)}{\sigma}$$

For example, consider the following problem:

Consider wait period for a reply from a bank manager is an average of 200 hours with a standard deviation of 75 hours. How many of these wait periods can be expected to **last for longer** than 300 hours?

Using the above formula we will calculate z as:

$z = (300–200)/75$

$z = 1.33$

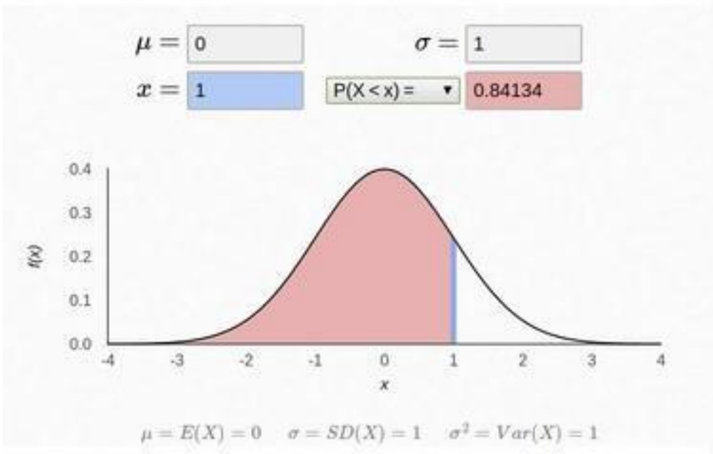
$\Phi(1.33) = 0.9082 = 90.82\%$ (Calculated using z-score table.)

answer = $1–0.9082 = 9.12\ \%$

One important thing about using z-scores is they are only provided for positive values. So to calculate z-score for the negative value you will use the following property.

1: $\Phi(a) + \Phi(–a) = 1$

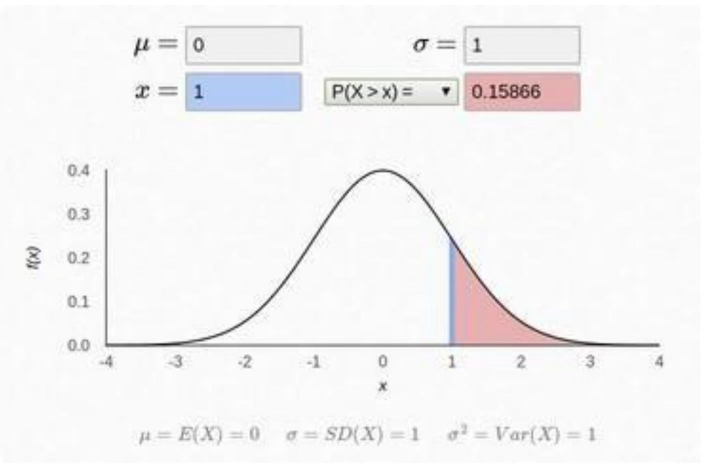
See the following diagrams and you will understand:



$\mu = 0$ $\sigma = 1$

$x = 1$ P(X < x) = 0.84134

0.4
0.3
0.2
0.1
0.0
f(x)
-4 -3 -2 -1 0 1 2 3 4
x
$\mu = E(X) = 0$ $\sigma = SD(X) = 1$ $\sigma^2 = Var(X) = 1$



$\mu = 0$ $\sigma = 1$

$x = 1$ P(X > x) = 0.15866

0.4
0.3
0.2
0.1
0.0
f(x)
-4 -3 -2 -1 0 1 2 3 4
x
$\mu = E(X) = 0$ $\sigma = SD(X) = 1$ $\sigma^2 = Var(X) = 1$

$\Phi(a) + \Phi(-a) = -1$

Central Limit Theorem:

In very loose terms Central Limit Theorem says that the sum of a large number of sample is approximately normal. (There are few conditions on it.)

Infamous "All of statistics" Central Limit Theorem has stated as "For large sample size sample average has approximately normal distribution"

> In more simple terms, when you add independent random variables then their properly normalized sum tends toward a normal distribution (informally a "*bell curve*") even if the original variables themselves are not normally distributed.

This is the reason, normal distributions occur so many numbers of times. So many phenomena such as the height, weight, living age of individuals all resemble bell-shaped curves. If you give this concept a more thought you will understand the logic.

Probabilistic models for categorical data

Categorical Data

Categorical variables represent types of data which may be divided into groups. Examples of categorical variables are race, sex, age group, and educational level. While the latter two variables may also be considered in a numerical manner by using exact values for age and highest grade completed, it is often more informative to categorize such variables into a relatively small number of groups.

Analysis of categorical data generally involves the use of data tables. A ***two-way table*** presents categorical data by counting the number of observations that fall into each group for two variables, one divided into rows and the other divided into columns. For example, suppose a survey was conducted of a group of 20 individuals, who were asked to identify their hair and eye color. A two-way table presenting the results might appear as follows:

		Eye Color			
Hair Color	Blue	Green	Brown	Black	Total
Blonde	2	1	2	1	6

Red	1	1	2	0	4
Brown	1	0	4	2	7
Black	1	0	2	0	3
Total	5	2	10	3	20

The totals for each category, also known as ***marginal distributions***, provide the number of individuals in each row or column without accounting for the effect of the other variable (in the example above, the total number of individuals with blue eyes, regardless of hair color, is 5).

Since simple counts are often difficult to analyze, two-way tables are often converted into percentages. In the above example, there are 4 individuals with red hair. Since there were a total of 20 observations, this means that 20% of the individuals survered are redheads. One also might want to investigate the percentages within a given category -- of the 4 redheads, 2 (50%) have brown eyes, 1 (25%) has blue eyes, and 1 (25%) has green eyes.

Example

Categorical Data is the data that generally takes a limited number of possible values. Also, the data in the category need not be numerical, it can be textual in nature. All machine learning models are some kind of mathematical model that need numbers to work with. This is one of the primary reasons we need to pre-process the categorical data before we can feed it to machine learning models.

Let's consider following data set:

```
import pandas as pd
df = pd.read_csv('household_data.txt')
print(df)
```

Output

```
Item_Category Gender Age Salary Purchased
0 Fitness Male 20 30000 Yes
1 Fitness Female 50 70000 No
2 Food Male 35 50000 Yes
3 Kitchen Male 22 40000 No
4 Kitchen Female 30 35000 Yes
```

Intuitively, you can see that Item_Category (Fitness, Food, Kitchen), Gender (Male, Female), and Purchased (Yes, No) are the categorical variables since there is only a limited set of values that these can take.

In the rest of this guide, we will see how we can use the python scikit-learn library to handle the categorical data. Scikit-learn is a machine learning toolkit that provides various tools to cater to different aspects of machine learning e.g. Classification, Regression, Clustering, Dimensionality reduction, Model selection, Preprocessing.

There is a subtle difference in how the categorical data for the dependent and independent variables are handled. We will learn more about this later in the guide. That said, we need to break our data set into the dependent matrix (X) and independent vector (y).

Fine, we'll create the dependent matrix (X) from the data set:2

```
X = df.iloc[:, :-1].values
print(X)
```

Output

```
[['Fitness' 'Male' 20 30000]
['Fitness' 'Female' 50 70000]
['Food' 'Male' 35 50000]
['Kitchen' 'Male' 22 40000]
['Kitchen' 'Female' 30 35000]]
```

Then, we'll extract the dataset to get the dependent vector:

```
y = df.iloc[:, -1].values
print(Y)
```

Output

```
['Yes', 'No', 'Yes', 'No', 'Yes']
```

pd

Encoding the Categorical Data for Independent Features Matrix X

Next, we're going to encode the categorical data for Item_Category and Gender so that they're changed to numbers which can then be fed to the machine learning models.

Consider the following code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('household_data.txt')
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
labelencoder_X = LabelEncoder()
X[:,0] = labelencoder_X.fit_transform(X[:,0])
X[:,1] = labelencoder_X.fit_transform(X[:,1])
print(X)
```

python

Output

```
[[0 1 20 30000]
[0 0 50 70000]
[1 1 35 50000]
[2 1 22 40000]
[2 0 30 35000]]
```

In the above code, we have used the LabelEncoder class from sklearn preprocessing to transform the labels for Item_Category and Gender to numbers. So for Item_Category, 'Fitness' is assigned as 0, 'Food' is assigned as 1, and 'Kitchen' as 2. Similarly for Gender, 'Male' is assigned as 1 and 'Female' as 0. This leads us to another challenge when working with the machine learning models. Since all mathematical models deal with numbers and some numbers are greater than others, this can skew the models leading to inaccurate results.

Consider Gender, in the above output 'Male' is assigned the value as 1 while 'Female' is 0. In all the calculations that are going to take place, the weight of Male is going to be more than that of Female. Thisdoes not make sense because Gender is a category of data and both variables need to be treated equally by the model to predict accurate results.

One-Hot Encoding

The solution to this problem is achieved by incorporating the concept of dummy columns. For each of the values of a certain category, a new column is introduced. So, if the row value of Item_Category is Fitness then that row will get the value as 1 and Food and Kitchen will get the value as 0.

Item_Category
Fitness
Food
Kitchen

is converted to

Fitness	Food	Kitchen
1	0	0
0	1	0
0	0	1

OneHotEncoder is the class in the scikit-learn preprocessing that helps us achive this with ease. Consider the following code block:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
df = pd.read_csv('household_data.txt')
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
labelencoder_X = LabelEncoder()
X[:,0] = labelencoder_X.fit_transform(X[:,0])
X[:,1] = labelencoder_X.fit_transform(X[:,1])
onehotencoder = OneHotEncoder(categorical_features=[0,1])
```

```
X = onehotencoder.fit_transform(X).toarray()
print(X)
```

python

Output

```
[[1.0e+00 0.0e+00 0.0e+00 0.0e+00 1.0e+00 2.0e+01 3.0e+04]
[1.0e+00 0.0e+00 0.0e+00 1.0e+00 0.0e+00 5.0e+01 7.0e+04]
[0.0e+00 1.0e+00 0.0e+00 0.0e+00 1.0e+00 3.5e+01 5.0e+04]
[0.0e+00 0.0e+00 1.0e+00 0.0e+00 1.0e+00 2.2e+01 4.0e+04]
[0.0e+00 0.0e+00 1.0e+00 1.0e+00 0.0e+00 3.0e+01 3.5e+04]]
```

python

You may notice that the columns have increased in the data set. The column 'Item_Category' is broken into three columns and column Gender is broken into two columns. Thus, the resulting number of columns in X vector is increased from four to seven. Also, notice that after applying the OneHotEncoding function, the values in the Panda Dataframe are changed to scientific notation.

Encoding the Dependent Vector Y

Encoding the dependent vector is much simpler than that of independent variables. For the dependent variables, we don't have to apply the One-Hot encoding and the only encoding that will be utilized is Lable Encoding. In the below code we are going to apply label encoding to the dependent variable, which is 'Purchased' in our case.

```
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
print(y)
```

python

Output

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

python

Understanding the categorical data is one of the most important aspects of dealing with Data Science. The human mind is designed in a way so that it is easy to understand the representations of the data when presented in the categorical forms. On the other hand, it is not easy for the computers to work with this kind of data, as mathematical equations don't like the input in this form. So firm understanding of concepts required to handle categorical data is a requirement when starting to design your machine learning solutions. It is worth mentioning that not just the input but the ultimate output of your model is also important. If the output of your model is an input to some other data engine than it is best to leave it in the numeric form. However, if the ultimate user of the solution is a human than probably you may want to change the numeric data to categories to help them make easy sense of it.

Discriminative learning by optimising conditional likelihood

Discriminative models, also referred to as conditional **models**, are a class of **models** used in statistical classification, especially in supervised **machine learning**. A **discriminative** classifier tries to **model** by just depending on the observed data while **learning** how to do the classification from the given statistics.

Conditional likelihood probability

Conditional probability is the probability of one event occurring with some relationship to one or more other events. For example:

- Event A is that it is raining outside, and it has a 0.3 (30%) chance of raining today.
- Event B is that you will need to go outside, and that has a probability of 0.5 (50%).

A conditional probability would look at these two events in relationship with one another, such as **the probability that it is both raining** *and* **you will need to go outside.**

The formula for conditional probability is:
P(B|A) = P(A and B) / P(A)
which you can also rewrite as:
P(B|A) = P(A∩B) / P(A)

Example 1

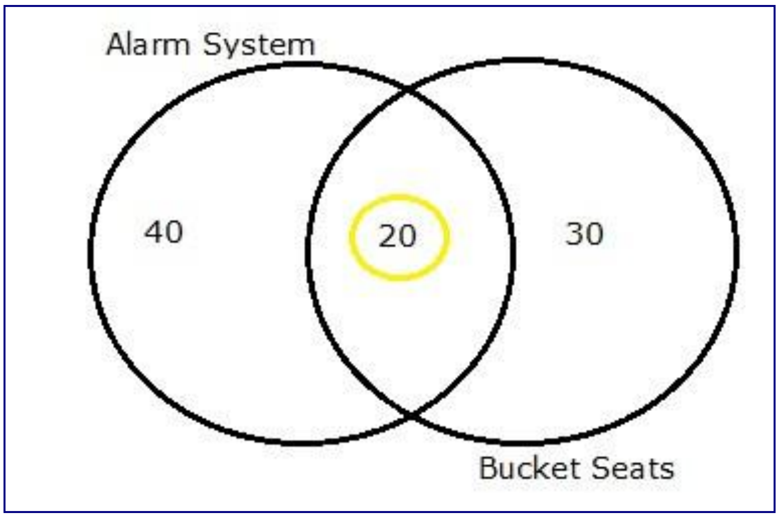
In a group of 100 sports car buyers, 40 bought alarm systems, 30 purchased bucket seats, and 20 purchased an alarm system and bucket seats. If a car buyer chosen at random bought an alarm system, what is the probability they also bought bucket seats?

Step 1: Figure out P(A). It’s given in the question as 40%, or 0.4.

Step 2: Figure out P(A∩B). This is the intersection of A and B: both happening together. It’s given in the question 20 out of 100 buyers, or 0.2.

Step 3: Insert your answers into the formula:
P(B|A) = P(A∩B) / P(A) = 0.2 / 0.4 = 0.5.

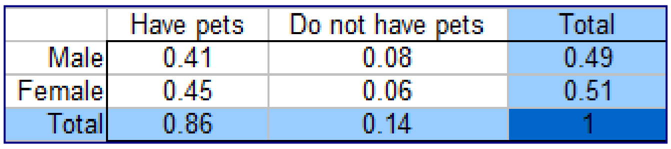
The probability that a buyer bought bucket seats, given that they purchased an alarm system, is 50%.



Venn diagram showing that 20 out of 40 alarm buyers purchased bucket seats.

Example 2:

This question uses the following contingency table:

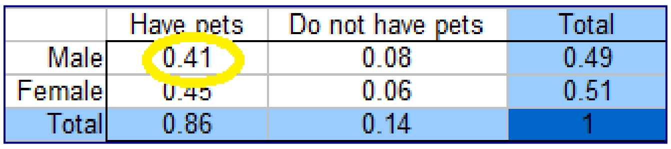


	Have pets	Do not have pets	Total
Male	0.41	0.08	0.49
Female	0.45	0.06	0.51
Total	0.86	0.14	1

What is the probability a randomly selected person is male, given that they own a pet?

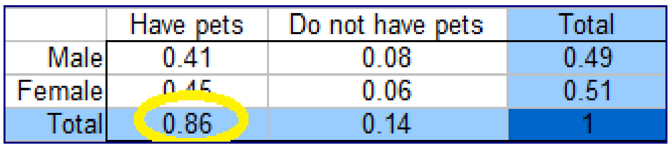
Step 1: Repopulate the formula with new variables so that it makes sense for the question (optional, but it helps to clarify what you're looking for). I'm going to say M is for male and PO stands for pet owner, so the formula becomes:
P(M|PO) = P(M∩PO) / P(PO)

Step 2: Figure out P(M∩PO) from the table. The intersection of male/pets (the intersection on the table of these two factors) is 0.41.



	Have pets	Do not have pets	Total
Male	0.41	0.08	0.49
Female	0.45	0.06	0.51
Total	0.86	0.14	1

Step 3: Figure out P(PO) from the table. From the total column, 86% (0.86) of respondents had a pet.



	Have pets	Do not have pets	Total
Male	0.41	0.08	0.49
Female	0.45	0.06	0.51
Total	0.86	0.14	1

Step 4: Insert your values into the formula:
P(M|PO) = P(M∩PO) / P(M) = 0.41 / 0.86 = 0.477, or 47.7%.

Why do we care about conditional probability? Events in life rarely have simple probability. Think about the probability of getting rainfall.

Conditional Probability in Real Life

Conditional probability is used in many areas, including finance, insurance and politics. For example, the re-election of a president depends upon the voting preference of voters and perhaps the success of television advertising — even the probability of the opponent making gaffes during debates!

The weatherman might state that your area has a probability of rain of 40 percent. However, this fact is *conditional* on many things, such as the probability of…

- …a cold front coming to your area.
- …rain clouds forming.
- …another front pushing the rain clouds away.

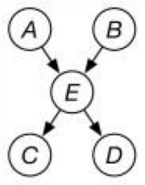
We say that the **conditional probability** of rain occurring depends on all the above events.

Probabilistic Models with hidden variables

10.3.2 Hidden Variables

The next simplest case is where the model is given, but not all variables are observed. A **hidden variable** or a **latent variable** is a variable in a belief network whose value is not observed for any of the examples. That is, there is no column in the data corresponding to that variable.

Model	Data	➪ Probabilities



A	B	C	D
t	f	t	t
f	t	t	t
t	t	f	t
…			

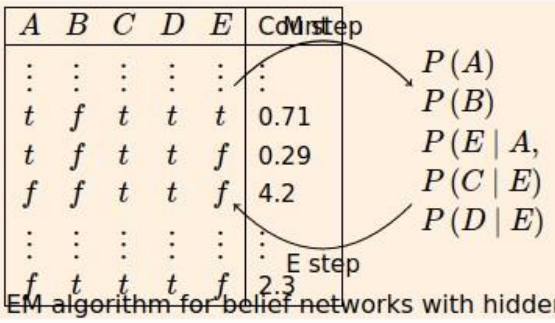
$P(A)$
$P(B)$
$P(E \mid A, B)$
$P(C \mid E)$
$P(D \mid E)$

Figure 10.8: Deriving probabilities with missing data

Example 10.12. *Figure 10.8 shows a typical case. Assume that all the variables are binary. The model contains a hidden variable E that is in the model but not the data set. The aim is to learn the parameters of the model that includes the hidden variable E. There are 10 parameters to learn.*

Note that, if E was not part of the model, the algorithm would have to learn $P(A)$, $P(B)$, $P(C \mid AB)$, $P(D \mid ABC)$, which has 14 parameters. The reason for introducing hidden variables is, paradoxically, to make the model simpler and, therefore, less prone to overfitting.

The **expectation maximization** or **EM** algorithm for learning belief networks with hidden variables is essentially the same as the EM algorithm for clustering. The E step, shown in Figure 10.9, involves probabilistic inference for each example to infer the probability distribution of the hidden variable(s) given the observed variables for that example. The M step of inferring the probabilities of the model from the augmented data is the same as in the fully observable case discussed in the previous section, but, in the augmented data, the counts are not necessarily integers.



A	B	C	D	E	Count
⋮	⋮	⋮	⋮	⋮	⋮
t	f	t	t	t	0.71
t	f	t	t	f	0.29
f	f	t	t	f	4.2
⋮	⋮	⋮	⋮	⋮	⋮
f	t	t	t	f	2.3

Figure 10.9: EM algorithm for belief networks with hidden variables

10.3.1 Learning the Probabilities Bibliography Index 10.3.3 Missing Data

Generated on Sat Nov 3 11:48:18 2018 by LaTeXML

If u have time pls study about EM algorithem in detail

Features

Features are simply **variables**, observable phenomenon that can be quantified and recorded. You want to select independent features and sometimes derive new features from existing ones.
For example, take an ML application trying to determine the probability of heart disease in patients. What are some possible features?

- Gender
- Age
- Height
- Weight
- Blood pressure
- Resting heart rate
- Past medical history

Let's look at some of the features in detail. One thing we want to do sometimes is categorize features with many values and place them into a smaller number of categories. Age, blood pressure, and resting heart rate all have a valid range of values from 0 to some integer with an upper bound. But do we want 120 different possible values for age? To make it easier, why not categorize the ages as 0 to 18, 19 to 29, 30 to 39, and so on? Likewise, place the vital signs into with three to seven categories. As for deriving values, height and weight can be combined into a standard feature: BMI. So calculate BMI and break it up into a small number of categories.

Past medical history can be made into multiple binary features. Has the patient suffered from a stroke? Yes or no. Has the patient been diagnosed with high blood pressure?
That's just the start, but it's a good basic example of identifying, classifying, and deriving useful features for machine learning applications.

Kinds of features:

There are three distinct types of features: quantitative, ordinal, and categorical. We can also consider a fourth type of feature—the Boolean—as this type does have a few distinct qualities, although it is actually a type of categorical feature. These feature types can be ordered in terms of how much information they convey. Quantitative features have the highest information capacity followed by ordinal, categorical, and Boolean.

Let's take a look at the tabular analysis:

Feature type	Order	Scale	Tendency	Dispersion	Shape
Quantitative	Yes	Yes	Mean	Range, variance, and standard deviation	Skewness, kurtosis
Ordinal	Yes	No	Median	Quantiles	NA
Categorical	No	No	Mode	NA	NA

The preceding table shows the three types of features, their statistics, and properties. Each feature inherits the statistics from the features from the next row it in the table. For example, the measurement of central tendency for quantitative features includes the median and mode.

<u>What is Feature Selection</u>

Operations and statistics
Features can be defined by the allowable operations that can be performed on them. Consider two

features: a person's age and their phone number. Although both these features can be described by integers, they actually represent two very different types of information. This is clear when we see which operations we can usefully perform on them. For example, calculating the average age of a group of people will give us a meaningful result; calculating the average phone number will not. We can call the range of possible calculations that can be performed on a feature as its statistics. These statistics describe three separate aspects of data. These are—its **central tendency**, its **dispersion**, and its **shape**.
To calculate the central tendency of data, we usually use one or more of the following statistics:
the mean (or average),
the median (or the middle value in an ordered list),and the mode (or the majority of all values).
The mode is the only statistic that can beapplied… Structured features

We assume that each instance can be represented as a vector of feature values and that all relevant aspects are represented by this vector. This is sometimes called an **abstraction** because we filter out unnecessary information and represent a real-world phenomena with a vector. For example, representing the entire works of Leo Tolstoy as a vector of word frequencies is an abstraction. We make no pretense that this abstraction will serve any more than a very particular limited application. We may learn something about Tolstoy's use of language and perhaps elicit some information regarding the sentiment and subject of Tolstoy's writing. However, we are unlikely to gain any significant insights into the broad canvas of the 19th century Russia portrayed in these works. A human reader, or a more sophisticated algorithm, will gain these insights not from the counting of each word but by the structure that these words are part of.

We can think of structured features in a similar...

Transforming features

When we transform features, our aim, obviously, is to make them more useful to our models. This can be done by adding, removing, or changing information represented by the feature. A common feature transformation is that of changing the feature type. A typical example is **binarization**, that is, transforming a categorical feature into a set of binary ones. Another example is changing an ordinal feature into a categorical feature. In both these cases, we lose information. In the first instance, the value of a single categorical feature is mutually exclusive, and this is not conveyed by the binary representation. In the second instance, we lose the ordering information. These types of transformations can be considered inductive because they consist of a well-defined logical procedure that does not involve an objective choice apart from the decision to carry out these transformations in the first place.

What is Feature Selection

Feature selection is also called variable selection or attribute selection.
It is the automatic selection of attributes in your data (such as columns in tabular data) that are most relevant to the predictive modeling problem you are working on.
"feature selection… is the process of selecting a subset of relevant features for use in model construction"

Feature selection is different from dimensionality reduction. Both methods seek to reduce the number of attributes in the dataset, but a dimensionality reduction method do so by creating new combinations of attributes, where as feature selection methods include and exclude attributes present in the data without changing them.

The Problem The Feature Selection Solves

Feature selection methods aid you in your mission to create an accurate predictive model. They help you by choosing features that will give you as good or better accuracy whilst requiring less data.

Feature selection methods can be used to identify and remove unneeded, irrelevant and redundant attributes from data that do not contribute to the accuracy of a predictive model or may in fact decrease the accuracy of the model.

Fewer attributes is desirable because it reduces the complexity of the model, and a simpler model is simpler to understand and explain.

"The objective of variable selection is three-fold: improving the prediction performance of the predictors, providing faster and more cost-effective predictors, and providing a better understanding of the underlying process that generated the data."

Feature Selection Algorithms<u>hybrid</u>

There are three general classes of feature selection algorithms: filter methods, wrapper methods and embedded methods.

Filter Methods

Filter feature selection methods apply a statistical measure to assign a scoring to each feature. The features are ranked by the score and either selected to be kept or removed from the dataset. The methods are often univariate and consider the feature independently, or with regard to the dependent variable.
Some examples of some filter methods include the Chi squared test, information gain and correlation coefficient scores.

Wrapper Methods

Wrapper methods consider the selection of a set of features as a search problem, where different combinations are prepared, evaluated and compared to other combinations. A predictive model is used to evaluate a combination of features and assign a score based on model accuracy.
The search process may be methodical such as a best-first search, it may stochastic such as a random hill-climbing algorithm, or it may use heuristics, like forward and backward passes to add and remove features.
"Feature subset generation One intuitive way is to generate subsets of features sequentially If we start with an empty subset and gradually add one feature at a time we adopt a scheme called sequential forward selection. if we start with a full set and remove one feature at a time we have a scheme called sequential backward selection We can also randomly generate a subset so that each possible subset in total N whereN is the number of features has an approximately equal chance to be generated One extreme way is to exhaustively enumerate N possible subsets.
An example if a wrapper method is the recursive feature elimination algorithm.

Embedded Methods

Embedded methods learn which features best contribute to the accuracy of the model while the model is being created. The most common type of embedded feature selection methods are regularization methods.
Regularization methods are also called penalization methods that introduce additional constraints into the features.

Feature construction

Feature construction is a process that discovers missing information about the relationships between features and augments the space of features by inferring or creating additional features Assuming there are n features A1,A2.... A_n after feature construction we may have additional m features A_{n+1}A,$_{n+2}$,A_{n+m} All new constructed features are defined in terms of original features as su,h no inherently new informed is added through feature construction. Feature construction attempts to increase the expressive power of the original features Usually the dimensionality of the new feature set is expanded and is bigger than that of the original feature setI ntuitively there could be exponentially many combinations of original features, and not all combinations are necessary and useful. feature construction aims to automatically transform the original representation space to a new one that can help better achieve machine learning problems.

How to construct new features:
The various approaches can be categorized in to four groups.
Data driven: this approach is to construct new features based on analysis of the available data by applying various operators.
Hypothesis driven: here to construct new features based on the hypothesis generated previously.
Knowledge based: Here construct new features applying existing knowledge and domain knowledge.

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. That is why ensemble methods placed first in many prestigious machine learning competitions, such as the Netflix Competition, KDD 2009, and Kaggle.

Ensemble methods are meta algorithms that combine sevaral machine learning techniques into one predictive model in order to decrease variance (**bagging**), bias(**boosting**), or improve predictionss(stacking).

Ensemble methods can be divided into two groups:

- *sequential* ensemble methods where the base learners are generated sequentially (e.g. **AdaBoost**).

  The basic motivation of sequential methods is to **exploit the dependence between the base learners.** The overall performance can be boosted by weighing previously mislabeled examples with higher weight.

- *parallel* ensemble methods where the base learners are generated in parallel (**e.g. RandomForest**).

  The basic motivation of parallel methods is to **exploit independence between the base learners** since the error can be reduced dramatically by averaging.

Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type, leading to *homogeneous ensembles*.

There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to *heterogeneous ensembles*. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible.

Simple Ensemble Techniques

In this section, we will look at a few simple but powerful techniques, namely:

1. Max Voting
2. Averaging
3. Weighted Averaging

2.1 Max Voting

The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a 'vote'. The predictions which we get from the majority of the models are used as the final prediction.

For example, when you asked 5 of your colleagues to rate your movie (out of 5); we'll assume three of them rated it as 4 while two of them gave it a 5. Since the majority gave a rating of 4, the final rating will be taken as 4. **You can consider this as taking the mode of all the predictions.**

The result of max voting would be something like this:

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4

2.2 Averaging

Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

For example, in the below case, the averaging method would take the average of all the values.

i.e. (5+4+5+4+4)/5 = 4.4

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4.4

2.3 Weighted Average

This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction. For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.

The result is calculated as [(5*0.23) + (4*0.23) + (5*0.18) + (4*0.18) + (4*0.18)] = 4.41.

	Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
weight	0.23	0.23	0.18	0.18	0.18	
rating	5	4	5	4	4	4.41

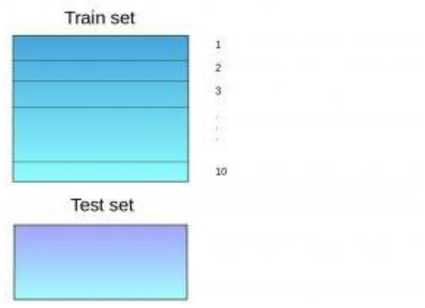
Advanced Ensemble techniques

Now that we have covered the basic ensemble techniques, let’s move on to understanding the advanced techniques.

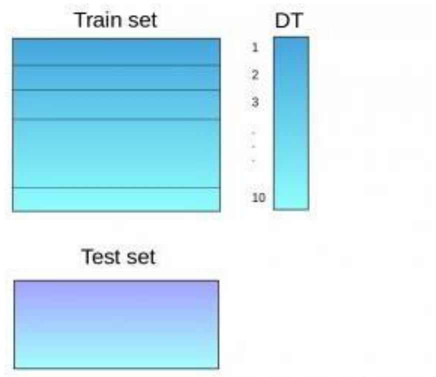
Stacking

Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set. Below is a step-wise explanation for a simple stacked ensemble:

1. The train set is split into 10 parts.

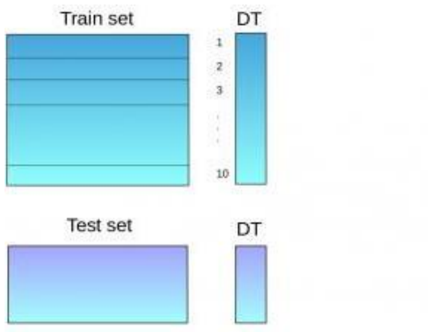


2. A base model (suppose a decision tree) is fitted on 9 parts and predictions are made for the 10th part. This is done for each part of the train set.

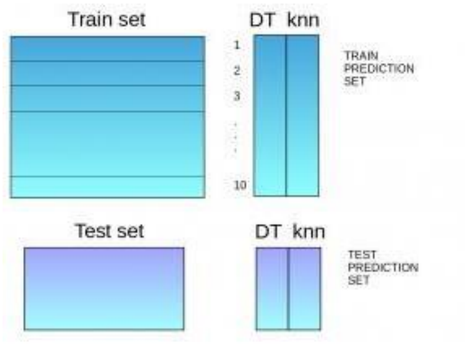


3. The base model (in this case, decision tree) is then fitted on the whole train dataset.

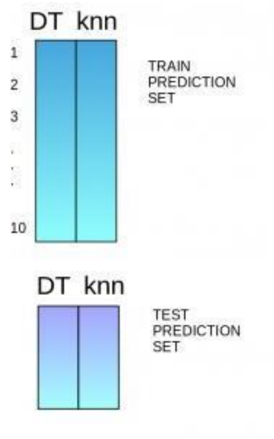
4. Using this model, predictions are made on the test set.



5. Steps 2 to 4 are repeated for another base model (say knn) resulting in another set of predictions for the train set and test set.



6. The predictions from the train set are used as features to build a new model.

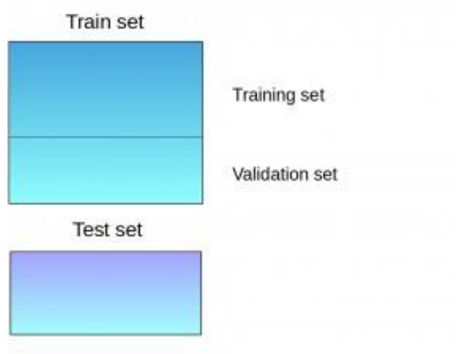


7. This model is used to make final predictions on the test prediction set.

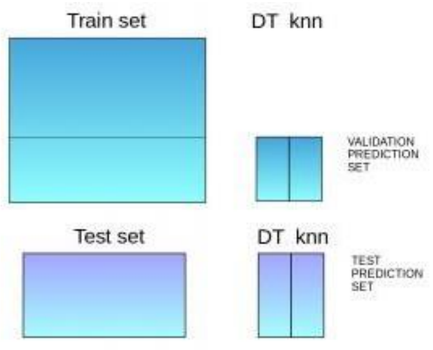
Blending

Blending follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions. In other words, unlike stacking, the predictions are made on the holdout set only. The holdout set and the predictions are used to build a model which is run on the test set. Here is a detailed explanation of the blending process:

8. The train set is split into training and validation sets.



9. Model(s) are fitted on the training set.
10. The predictions are made on the validation set and the test set.



11. The validation set and its predictions are used as features to build a new model.
12. This model is used to make final predictions on the test and meta-features.

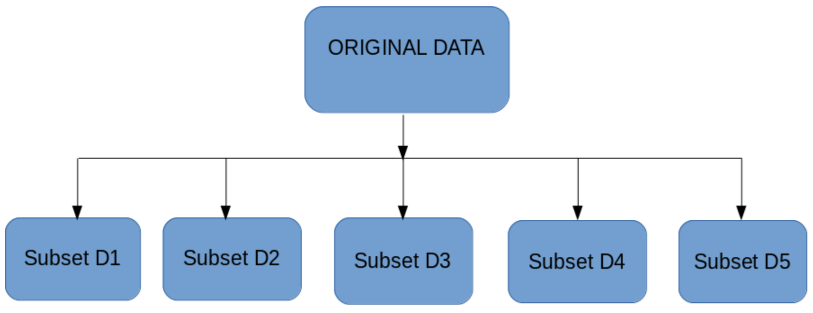
Bagging

The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Here's a question: If you create all the models on the same set of data and combine it, will it be useful? There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem? One of the techniques is bootstrapping.

Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**. The size of the subsets is the same as the size of the original set.

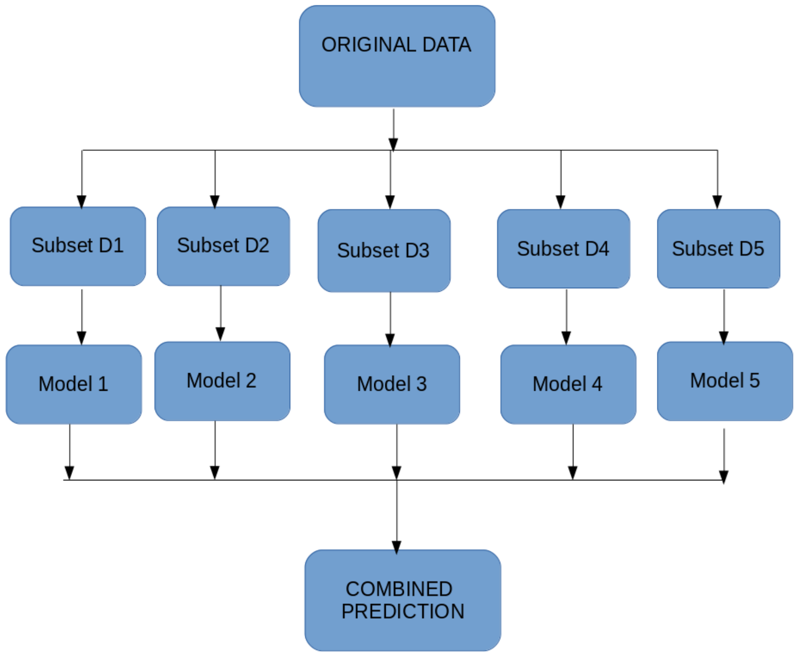
Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.

$$\begin{tikzcd}
 & & \text{ORIGINAL DATA} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\text{Subset D1} & \text{Subset D2} & \text{Subset D3} & \text{Subset D4} & \text{Subset D5}
\end{tikzcd}$$



13. Multiple subsets are created from the original dataset, selecting observations with replacement.

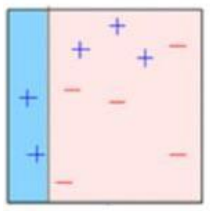
14. A base model (weak model) is created on each of these subsets.
15. The models run in parallel and are independent of each other.
16. The final predictions are determined by combining the predictions from all the models.



Boosting

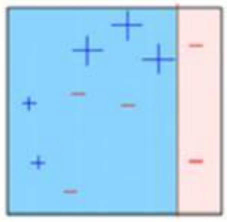
Before we go further, here's another question for you: If a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.
Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model. Let's understand the way boosting works in the below steps.

17. A subset is created from the original dataset.
18. Initially, all data points are given equal weights.
19. A base model is created on this subset.
20. This model is used to make predictions on the whole dataset.

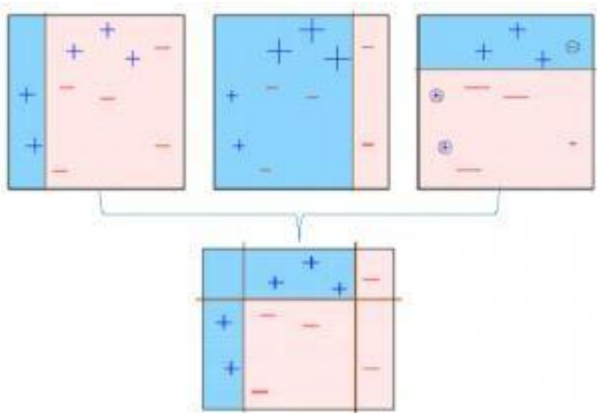


21. Errors are calculated using the actual values and predicted values.
22. The observations which are incorrectly predicted, are given higher weights.
    (Here, the three misclassified blue-plus points will be given higher weights)

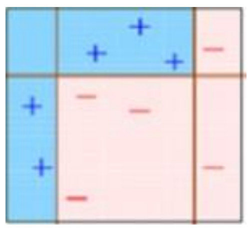
23. Another model is created and predictions are made on the dataset.
(This model tries to correct the errors from the previous model)



24. Similarly, multiple models are created, each correcting the errors of the previous model.
25. The final model (strong learner) is the weighted mean of all the models (weak learners).



26. Thus, the boosting algorithm combines a number of weak learners to form a strong learner. The individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.



4. Algorithms based on Bagging and Boosting

Bagging and Boosting are two of the most commonly used techniques in machine learning. In this section, we will look at them in detail. Following are the algorithms we will be focusing on:

Bagging algorithms:

- Bagging meta-estimator
- Random

algorithms:

- AdaBoost
- GBM
- XGBM
- Light GBM
- CatBoost

4.1 Bagging meta-estimator

Bagging meta-estimator is an ensembling algorithm that can be used for both classification (BaggingClassifier) and regression (BaggingRegressor) problems. It follows the typical bagging technique to make predictions. Following are the steps for the bagging meta-estimator algorithm:

1. Random subsets are created from the original dataset (Bootstrapping).
2. The subset of the dataset includes all features.
3. A user-specified base estimator is fitted on each of these smaller sets.
4. Predictions from each model are combined to get the final result.

4.2 Random Forest

Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm. The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

Looking at it step-by-step, this is what a random forest model does:

1. Random subsets are created from the original dataset (bootstrapping).
2. At each node in the decision tree, only a random set of features are considered to decide the best split.
3. A decision tree model is fitted on each of the subsets.
4. The final prediction is calculated by averaging the predictions from all decision trees.

Note: The decision trees in random forest can be built on a subset of data and features. Particularly, the sklearn model of random forest uses all features for decision tree and a subset of features are randomly selected for splitting at each node.

To sum up, Random forest **randomly** selects data points and features, and builds **multiple trees (Forest) .**

4.3 AdaBoost

Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

Below are the steps for performing the AdaBoost algorithm:

1. Initially, all observations in the dataset are given equal weights.
2. A model is built on a subset of data.
3. Using this model, predictions are made on the whole dataset.
4. Errors are calculated by comparing the predictions and actual values.
5. While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6. Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
7. This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

4.4 Gradient Boosting (GBM)

Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems. GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Regression trees used as a base learner, each subsequent tree in series is built on the errors calculated by the previous tree.

We will use a simple example to understand the GBM algorithm. We have to predict the age of a group of people using the below data:

ID	Married	Gender	Current City	Monthly Income	Age (target)
1	Y	M	A	51,000	35
2	N	F	B	25,000	24
3	Y	M	A	74,000	38
4	N	F	A	29,000	30
5	N	F	B	37,000	33

1. The mean age is assumed to be the predicted value for all observations in the dataset.
2. The errors are calculated using this mean prediction and actual values of age.

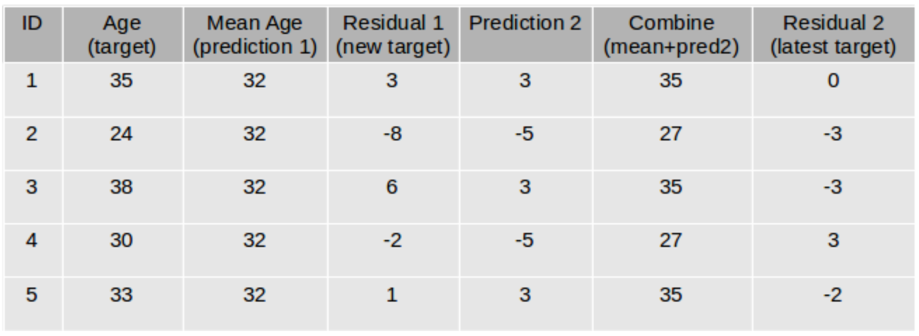
ID	Married	Gender	Current City	Monthly Income	Age (target)	Mean Age (prediction 1)	Residual 1
1	Y	M	A	51,000	35	32	3
2	N	F	B	25,000	24	32	-8
3	Y	M	A	74,000	38	32	6
4	N	F	A	29,000	30	32	-2
5	N	F	B	37,000	33	32	1

3. A tree model is created using the errors calculated above as target variable. Our objective is to find the best split to minimize the error.
4. The predictions by this model are combined with the predictions 1.

ID	Age (target)	Mean Age (prediction 1)	Residual 1 (new target)	Prediction 2	Combine (mean+pred2)
1	35	32	3	3	35
2	24	32	-8	-5	27
3	38	32	6	3	35
4	30	32	-2	-5	27
5	33	32	1	3	35

5. This value calculated above is the new prediction.

6. New errors are calculated using this predicted value and actual value.



ID	Age (target)	Mean Age (prediction 1)	Residual 1 (new target)	Prediction 2	Combine (mean+pred2)	Residual 2 (latest target)
1	35	32	3	3	35	0
2	24	32	-8	-5	27	-3
3	38	32	6	3	35	-3
4	30	32	-2	-5	27	3
5	33	32	1	3	35	-2

7. Steps 2 to 6 are repeated till the maximum number of iterations is reached (or error function does not change).

4.5 XGBoost

XGBoost (extreme Gradient Boosting) is an advanced implementation of the gradient boosting algorithm. XGBoost has proved to be a highly effective ML algorithm, extensively used in machine learning competitions and hackathons. XGBoost has high predictive power and is almost 10 times faster than the other gradient boosting techniques. It also includes a variety of regularization which reduces overfitting and improves overall performance. Hence it is also known as '**regularized boosting**' technique.

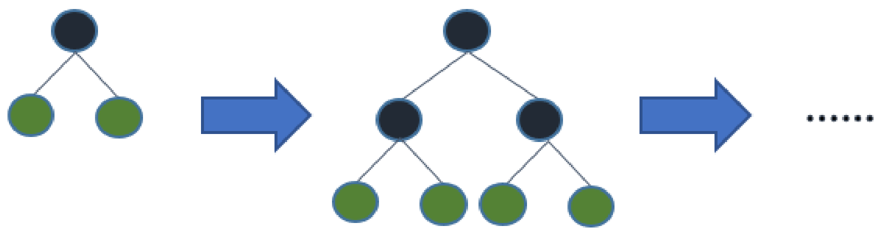
Let us see how XGBoost is comparatively better than other techniques:

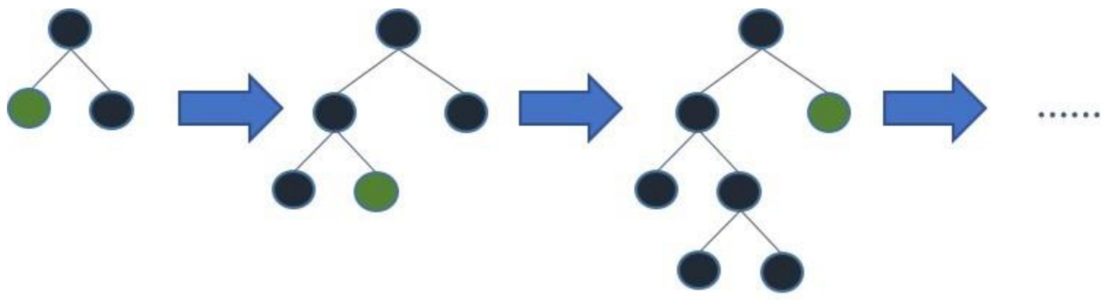
1. **Regularization:**
   - Standard GBM implementation has no regularisation like XGBoost.
   - Thus XGBoost also helps to reduce overfitting.
2. **Parallel Processing:**
   - XGBoost implements parallel processing and is faster than GBM .
   - XGBoost also supports implementation on Hadoop.
3. **High Flexibility:**
   - XGBoost allows users to define custom optimization objectives and evaluation criteria adding a whole new dimension to the model.
4. **Handling Missing Values:**
   - XGBoost has an in-built routine to handle missing values.
5. **Tree Pruning:**
   - XGBoost makes splits up to the max_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain.
6. **Built-in Cross-Validation:**
   - XGBoost allows a user to run a cross-validation at each iteration of the boosting

4.6 Light GBM

Before discussing how Light GBM works, let's first understand why we need this algorithm when we have so many others (like the ones we have seen above). **Light GBM beats all the other algorithms when the dataset is extremely large**. Compared to the other algorithms, Light GBM takes lesser time to run on a huge dataset.

LightGBM is a gradient boosting framework that uses tree-based algorithms and follows leaf-wise approach while other algorithms work in a level-wise approach pattern. The images below will help you understand the difference in a better way.


Level-wise tree growth


Leaf-wise tree growth

Leaf-wise growth may cause over-fitting on smaller datasets but that can be avoided by using the 'max_depth' parameter for learning.

4.7 CatBoost

Handling categorical variables is a tedious process, especially when you have a large number of such variables. When your categorical variables have too many labels (i.e. they are highly cardinal), performing one-hot-encoding on them exponentially increases the dimensionality and it becomes really difficult to work with the dataset.

CatBoost can automatically deal with categorical variables and does not require extensive data preprocessing like other machine learning algorithms