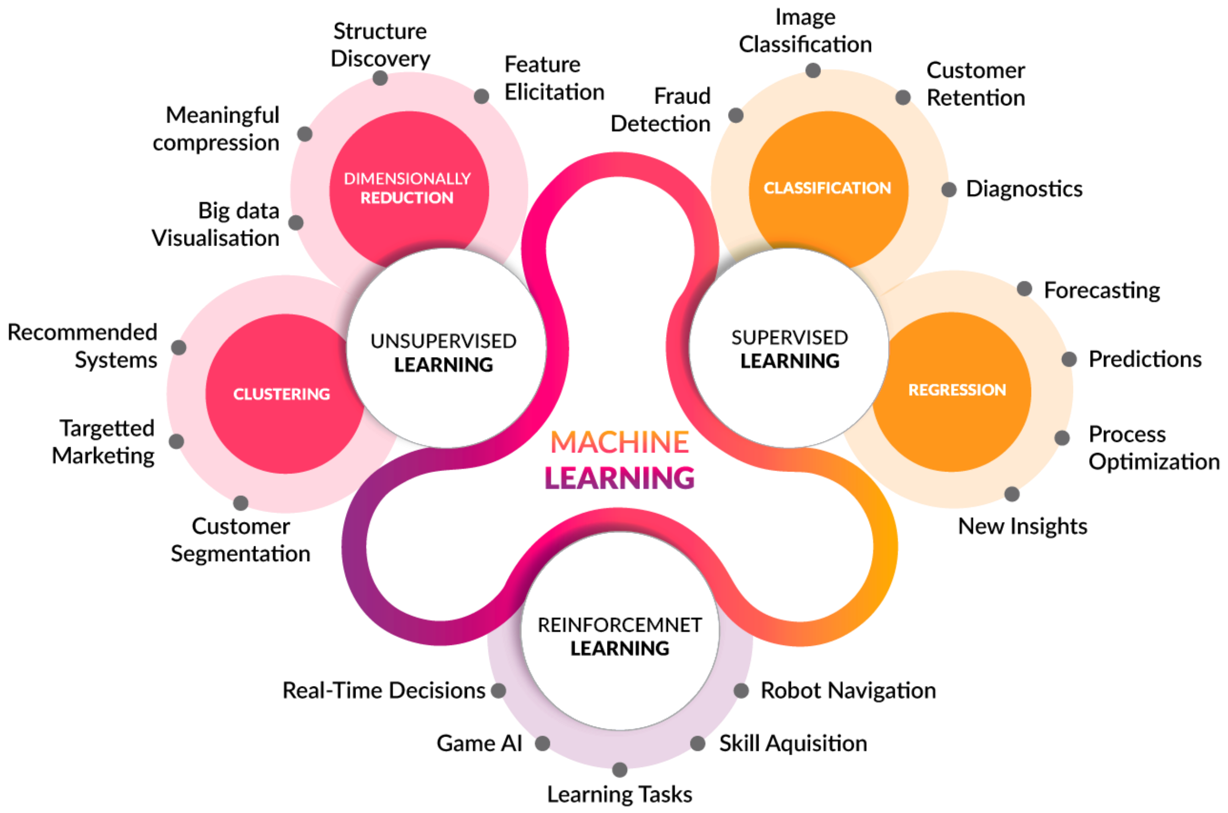
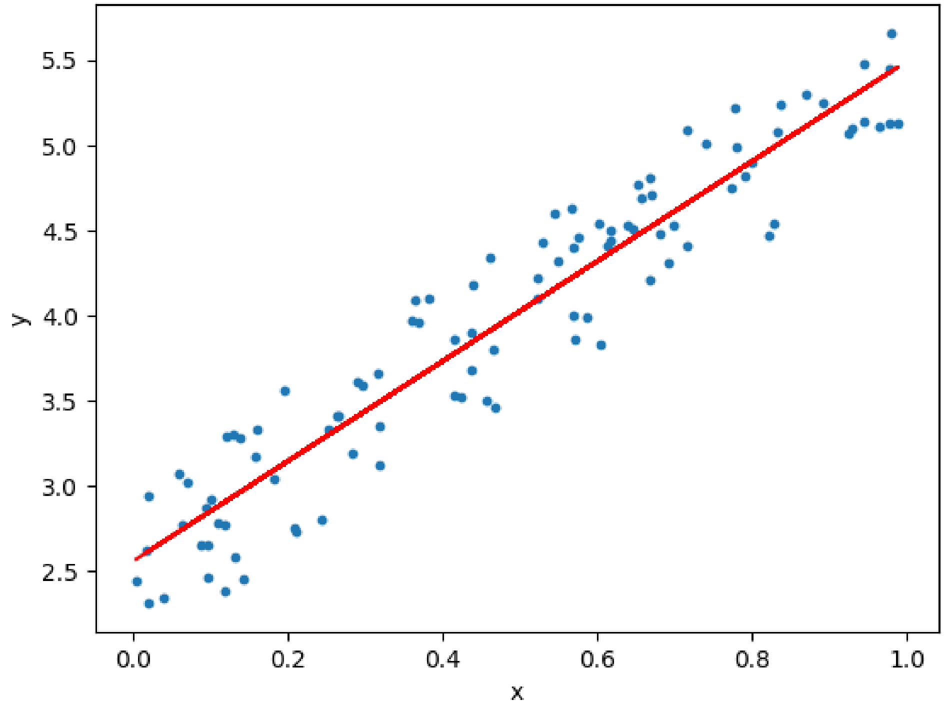


Supervised Learning

Unit 2

Linear Regression

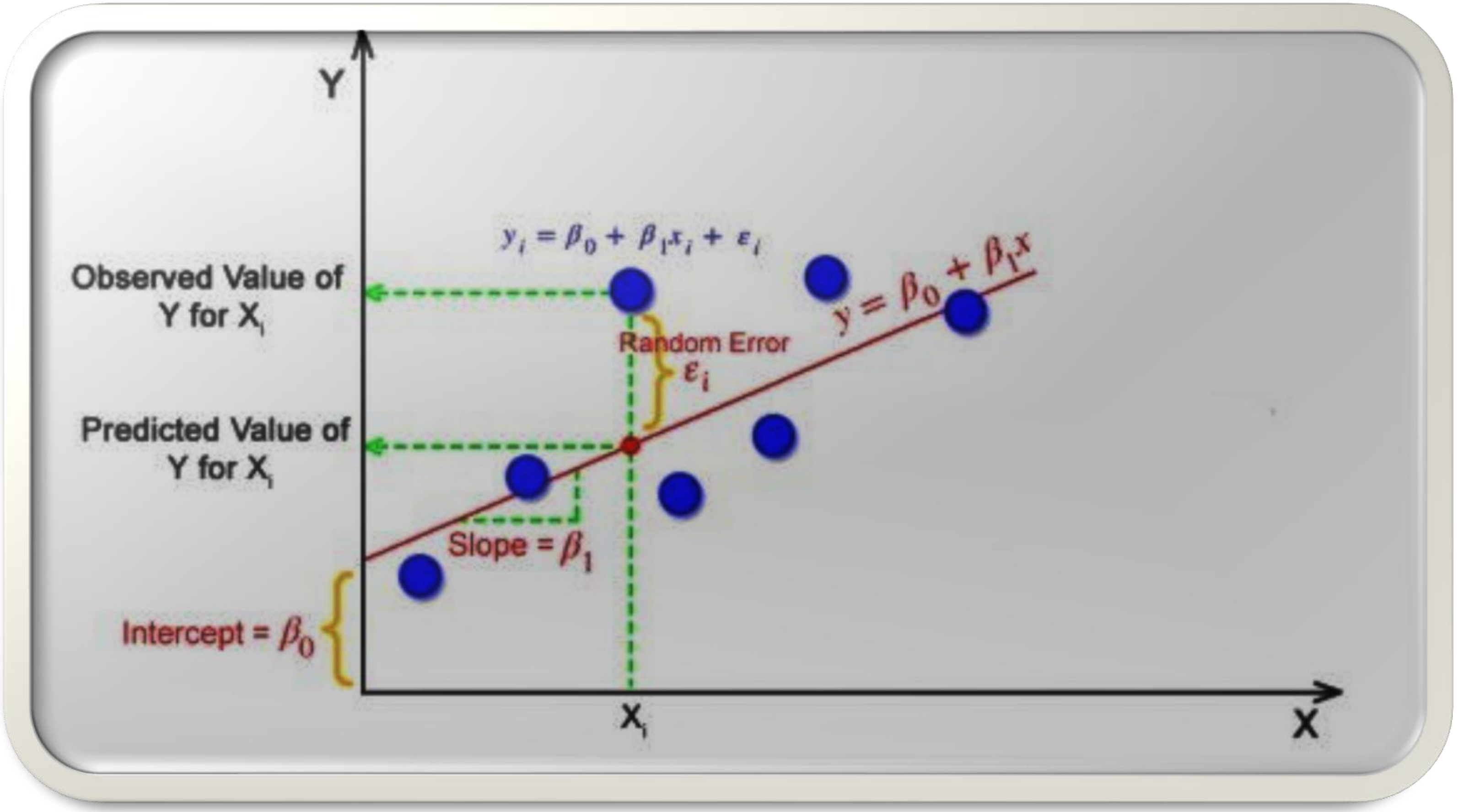
- Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data.
- One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable.
- For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.



Linear Regression (contd.)

- Linear regression is a simple yet powerful and mostly used algorithm in data science.
- Linear regression shows the linear relationship between the independent(predictor) variable i.e. X-axis and the dependent(output) variable i.e. Y-axis, called linear regression.
- If there is a single input variable X(independent variable), such linear regression is called simple linear regression.
- The above graph presents the linear relationship between the output(y) variable and predictor(X) variables.
- The red line is referred to as the best fit straight line.

Linear Regression (contd.)


Y
$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$
Observed Value of Y for X_i
Random Error ε_i
$y = \beta_0 + \beta_1 x$
Predicted Value of Y for X_i
Slope = β_1
Intercept = β_0
X_i
X

Linear Regression (contd.)

- The goal of the linear regression algorithm is to get the best values for β_0 and β_1 to find the best fit line.
- In simple terms, the best fit line is a line that fits the given scatter plot in the best way.
- The best fit line is a line that has the least error which means the error between predicted values and actual values should be minimum.

Linear Regression (contd.)

Random Error(Residuals)

In regression, the difference between the observed value of the dependent variable(**y**$_i$) and the predicted value is called the **residuals**.

$$\boldsymbol{e} = \boldsymbol{y}_{predicted} - \boldsymbol{y}_{actual}$$

Then we can define the RSS (Residual Sum of Squares) as

$$RSS = e_1^2 + e_2^2 + \dots\dots\dots\dots\dots\dots\dots\dots\dots + e_m^2$$

In Linear Regression, generally **Mean Squared Error (MSE)** cost function is used, which is the average of squared error that occurred between the $y_{predicted}$ and y_i.

$$MSE = 1/m \sum_{i=1}^{m} \left(y_i - (\beta_0 + B_1 x)\right)^2$$

We use different optimization techniques to minimize the error or RSS. Some of them are ordinary least square method, gradient descent etc.

Gradient Descent

We use linear regression to predict the dependent continuous variable y on the basis of independent X. It assumes the relationship between independent and dependent variables to be linear as such:

$$h(x) = \hat{y}_i = w_0 + w_1 x_1 + \ldots + w_n x_n = \sum_{j=1}^{n} w_j x_j$$

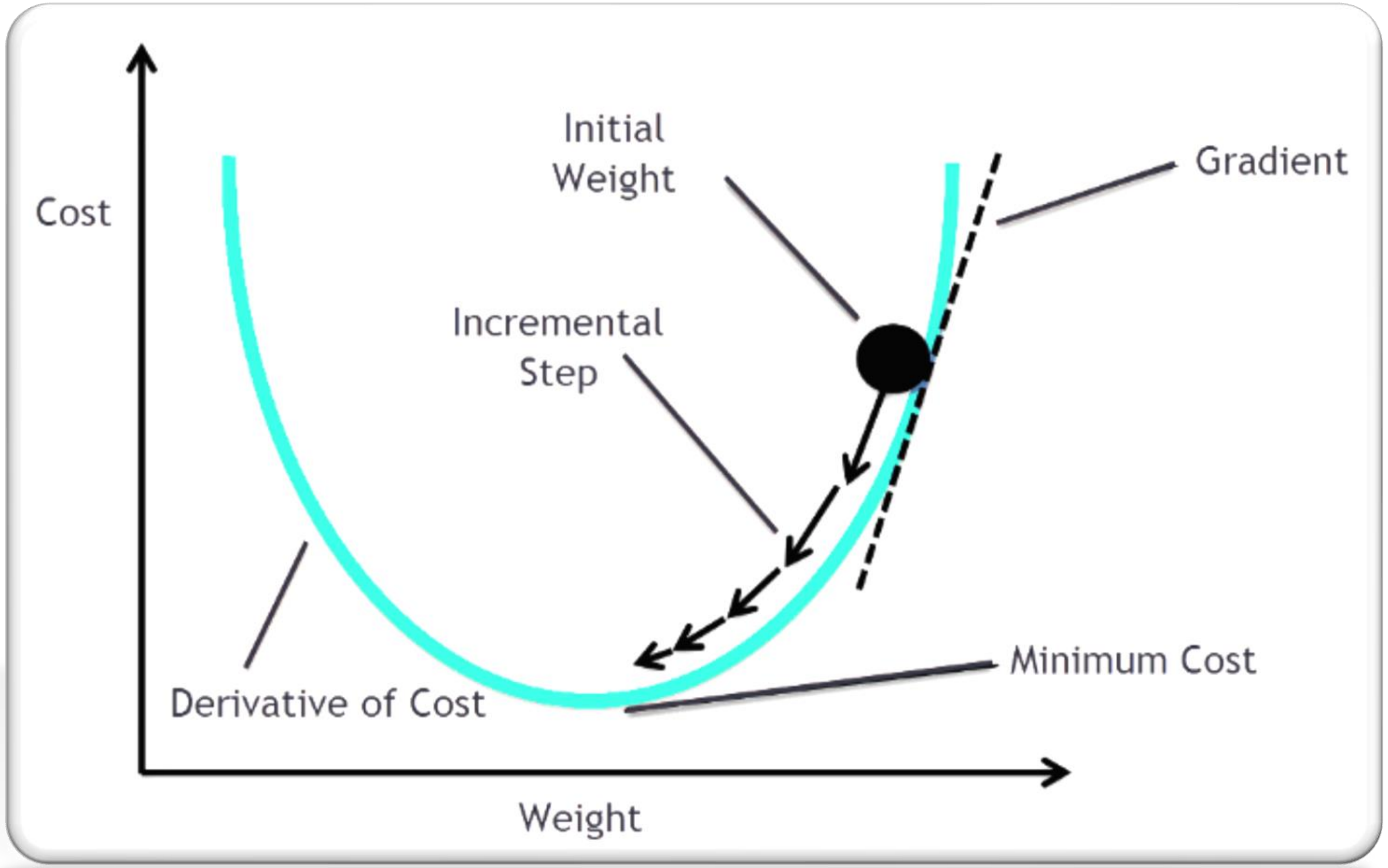
Gradient Descent is the optimization algorithms that optimize the cost function(objective function) to reach the optimal minimal solution.

To find the optimum solution we need to reduce the cost function(MSE) for all data points.

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)} \right)^2$$

This is done by updating the values of $w_0 \ldots w_1$ iteratively until we get an optimal solution.

Gradient Descent



Gradient Descent (contd.)

We used the update rule as:

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w)$$

where α is the learning rate.

And, considering only one training example (x, y), we compute the derivate as:

$$\frac{\partial}{\partial w_j} J(w)$$
$$= \frac{\partial}{\partial w_j} \frac{1}{2} (h(x) - y)^2$$
$$= \frac{1}{2} \frac{\partial}{\partial w_j} (h(x) - y)^2$$
$$= \frac{1}{2} \frac{\partial (h(x) - y)^2}{\partial (h(x) - y)} \frac{\partial (h(x) - y)}{\partial w_j}$$

$$\frac{\partial}{\partial w_j} J(w)$$
$$= \frac{1}{2} \frac{\partial (h(x) - y)^2}{\partial (h(x) - y)} \frac{\partial (h(x) - y)}{\partial w_j}$$
$$= \frac{1}{2} 2\, (\mathrm{h(x)} - y). \frac{\partial}{\partial w_j} \sum_{i=0}^{n} (w_j x_j - y)$$
$$= (h(x) - y). x_j$$

Gradient Descent (contd.)

Now, we can rewrite the update rule as:

$$w_j = w_j - \alpha(h(x) - y).x_j$$

Which gives,

$$w_j = w_j + \alpha(y - h(x)).x_j$$

And this is the required form for the weight update. This rule is called the **LMS (*Least Mean Square*) update rule** and is also known as the **Windrow-Hoff learning rule.**

Gradient Descent (contd.)

This update has several properties:

1. The update has direction in opposite of gradient i.e. we always move towards the opposite of the gradient.
2. The magnitude of the update is proportional to the error term $\left(y - h(x)\right)$ i.e. the lesser the error is smaller the update on parameters and larger change to parameters occur if our prediction has larger error.

Types of Gradient Descent

1. Batch Gradient Descent

- In this type of gradient descent, we compute the gradient term for all the training examples and then we update the parameters with respect to the gradient of all training observations.
- It computes the error of each training example within training observations but are summed up once before update is performed.
- This gradient descent has to scan through the entire training set before taking a single step.
- When training set is very large, it is costly operation to take a single step for convergence.
- This is computationally efficient algorithm and it produces stable gradient and stable convergence.
- Downside, it also requires entire training observations be in memory and available throughout the model training.

Types of Gradient Descent (contd.)

1. Batch Gradient Descent (contd.)

Its implementation is given as:

```
repeat until convergence {
        $w_j = w_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h\left(x^{(i)}\right)\right) . x_j^{(i)}$
}
```

Types of Gradient Descent (contd.)

2. Stochastic Gradient Descent

- In this type of gradient descent, we repeatedly run through the training set and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example.
- Often, Stochastic Gradient Descent gets β close to minimum much faster compared to batch gradient descent.
- It is preferred when the training set is large.
- Due to the frequent update, the Gradient descend will oscillates more resulting noisy gradients and each update is computationally expensive compared to batch gradient descent.

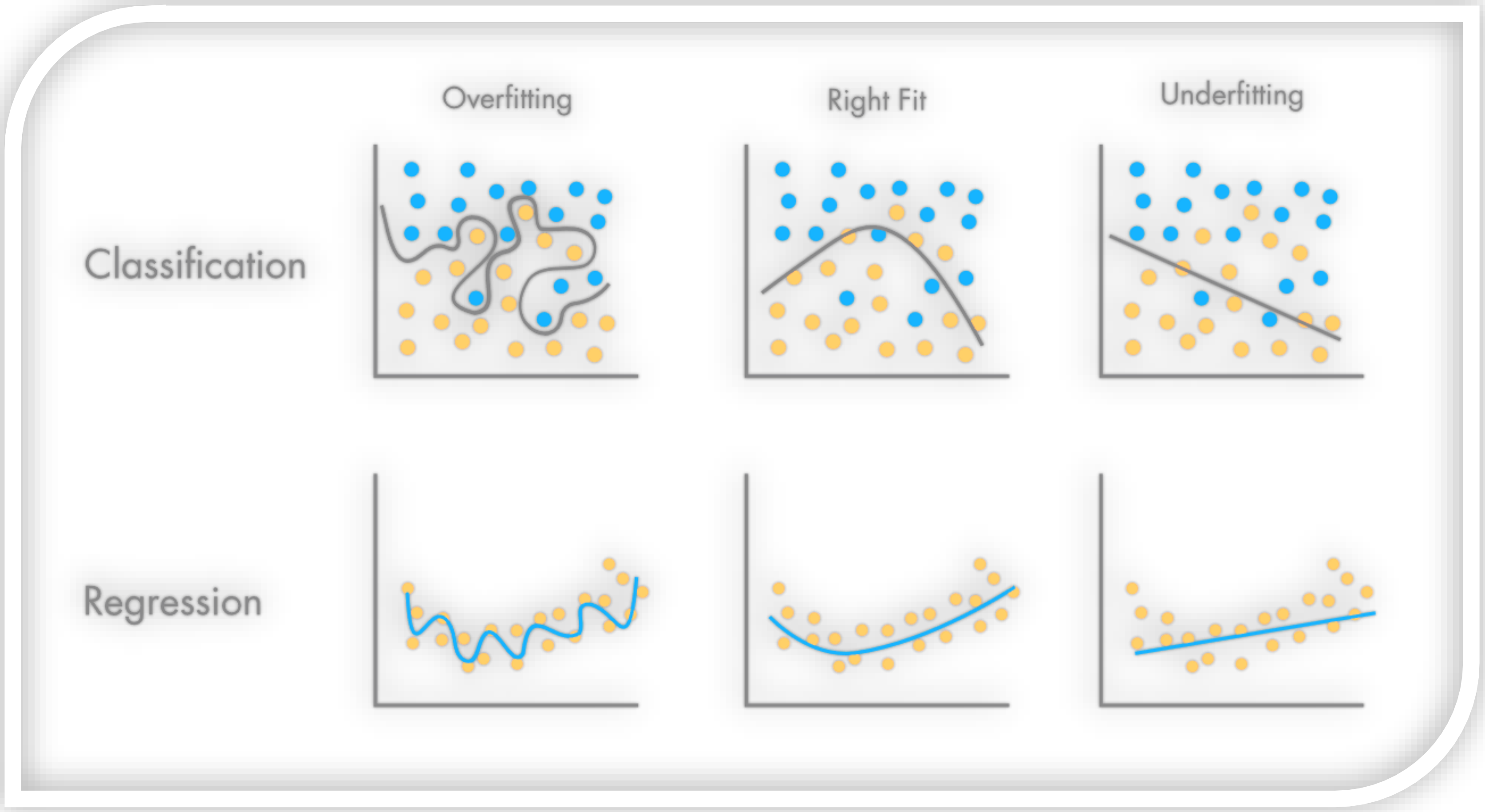
Types of Gradient Descent (contd.)

2. Stochastic Gradient Descent (contd.)

Its implementation is given as:

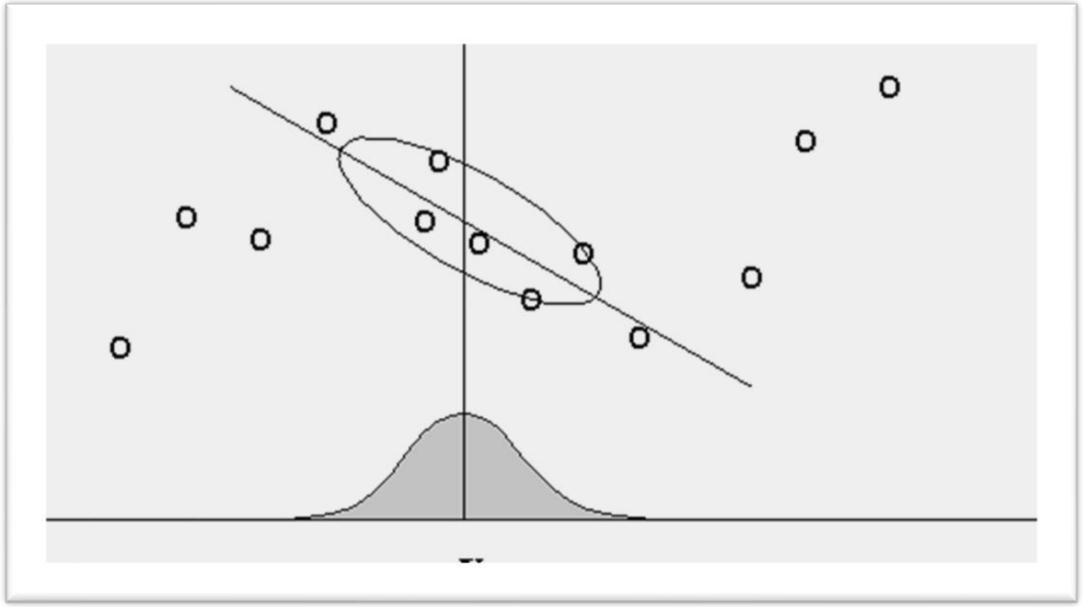
```
Loop for every j-th feature {

   For every training example i from 1 to m {
             $\beta_j = \beta_j + \alpha\left(y^{(i)} - h_\beta\left(x^{(i)}\right)\right).x_j^{(i)}$
   }
}
```

Overfitting and Underfitting



Locally Weighted Regression

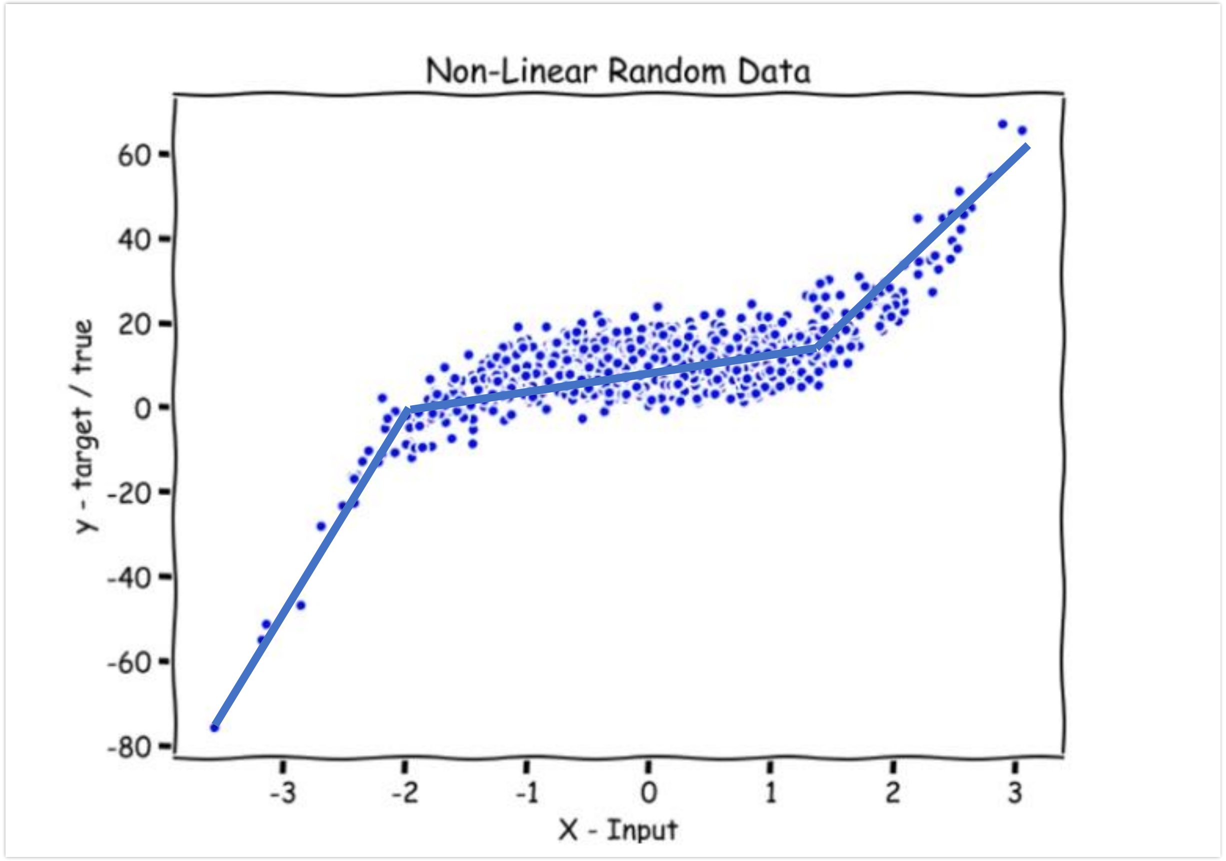
- Locally weighted regression is a non-parametric algorithm i.e. non-parametric algorithm make any assumptions about underlying data.
- Also, non-parametric algorithm doesn't learn a fixed set of parameters as in linear regression algorithm.
- It is also known as memory based method that performs regression around a point of interest and uses a training data that are local to that point only.
- In locally weighted regression, points are weighted by proximity to the current x in question using a kernel. A regression is then computed using the weighted points.



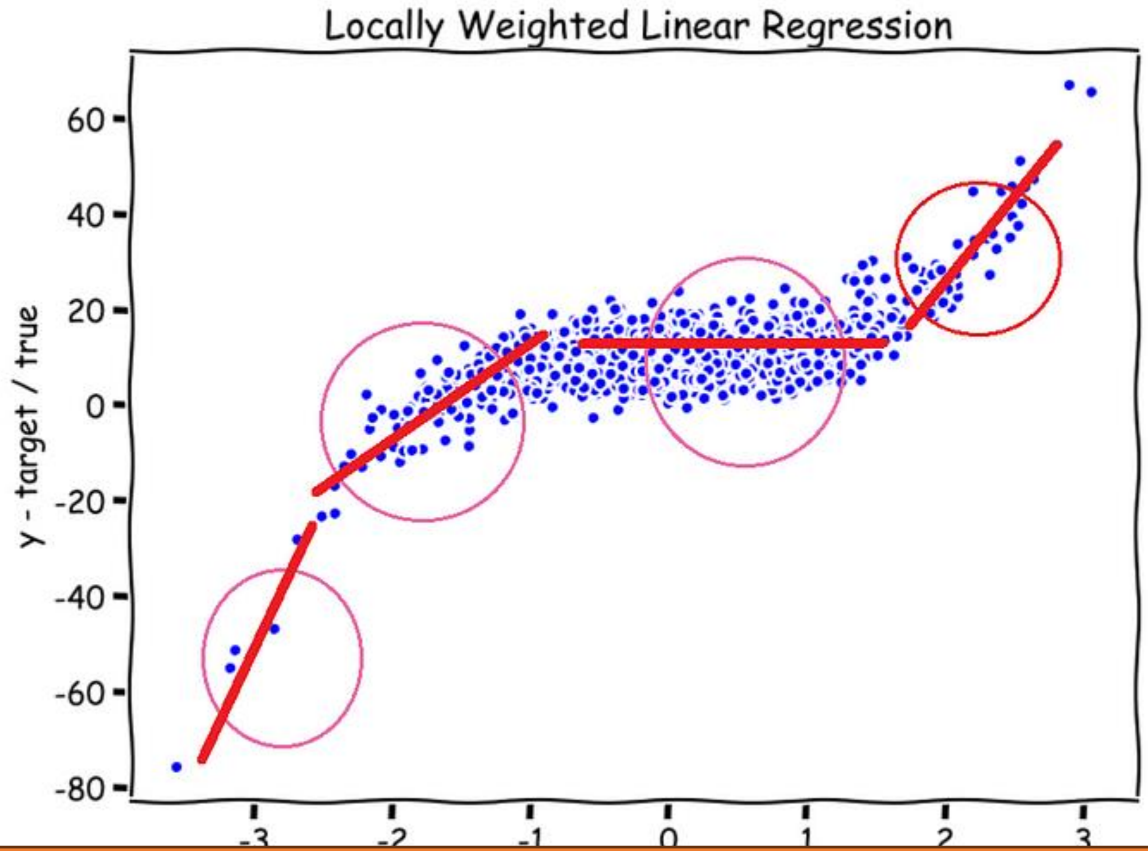
Locally Weighted Regression (contd.)

- How does it works?
  - LWLR is a non-parametric regression technique that fits a linear regression model to a dataset by giving more weight to nearby data points.
  - The weights assigned to each training data point are inversely proportional to their distance from the query point.
  - Training data points that are closer to the query point will have a higher weight and contribute more to the linear regression model.
  - LWLR is useful when a global linear model does not well-capture the relationship between the input and output variables. The goal is to capture local patterns in the data.

Locally Weighted Regression (contd.)



Locally Weighted Regression (contd.)



In locally weighted regression, points are weighted by proximity to the current **x** in question using a kernel. A regression is then computed using the weighted points.

Locally Weighted Regression (contd.)

In linear regression we train model that learns w_j from the given training data such that during the training we tend to minimize $\sum_i \left(y^{(i)} - h(x^{(i)})\right)^2$.

Whereas, with the locally weighted linear regression algorithm, we tend to minimize $\sum_i \omega^{(i)} \left(y^{(i)} - h(x^{(i)})\right)^2$ during the training. That is cost function is given as:

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} \omega^{(i)} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Here, $\omega^{(i)'}s$ are non-negative valued called weights (*Don't get confused with w_j which is actually a coefficient and defines the weight of features*) which refers to the value of particular data points in dataset.

Locally Weighted Regression (contd.)

Intuitively, if $\omega^{(i)}$ is large for a particular value of i, then in picking ω, we will train our model to make $\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2$ small. Similarly, if $\omega^{(i)}$ is small, then the $\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2$ error term will be pretty much ignored in the fit.

A common choice for the weights is

$$\omega^{(i)} = \exp\left(-\frac{\left(x^{(i)} - x\right)^2}{2\tau^2}\right)$$

where τ is the bandwidth parameter which depends on the particular point x at which we're trying to evaluate x.

Locally Weighted Regression (contd.)

The bandwidth parameter τ decides the width of the neighborhood you should look into to fit the local straight line.

In other words, τ controls how quickly the weight of a training example $x^{(i)}$ falls off with distance from the query point x.

Depending on the choice of τ, we make a choice among fatter or thinner bell-shaped curve, which cause us to look in a bigger or narrower window respectively, which in turn decides the number of nearby examples to use in order to fit the straight line.

τ has an effect on overfitting and underfitting:

- If τ is too broad, you end up over-smoothing the data leading to underfitting.
- If τ is too thin, you end up fitting a very jagged fit to the data leading to overfitting. T is a hyper-parameter of the locally weighted regression algorithm.

Locally Weighted Regression (contd.)

There's a huge literature on choosing τ . For example, instead of the bell-shaped curve, a triangle shaped function can be used so your weights actually go to zero outside some small range. So there are many versions of this algorithm.

Moreover, if $|x^{(i)} - x|$ is small, then $\omega^{(i)}$ is close to 1; and if $|x^{(i)} - x|$ is large, then $\omega^{(i)}$ is small.

Hence, ω is chosen giving much higher weight to the data points close to the query point x than the data points farther to the query point.

*Note: Although the formula of ω takes the form that is cosmetically similar to pdf of Gaussian distribution, they do not directly have anything to do with Gaussians, and in particular they are not random variables normally distributed as well.

Logistic Regression

Logistic Regression is a binary classification problem where we use linear line to separate between two known categories of interest called classes.

This is similar to the regression problem, except that the values of y we wan tot predict takes discrete values.

As logistic regression is binary classification, we have two classes where all the data points belongs, i.e. $y \epsilon \{0, 1\}$.

For instance, we are trying to classify email to spam and ham classes through spam classifier. $x^{(i)}$ refers to the i-th training dataset that either belongs to ham class or spam class . Thus, $y^{(i)}$ has value 1 if it is spam, 0 otherwise.

Note that for given $x^{(i)}$, corresponding $y^{(i)}$ is given in data and called as **label**.

Logistic Regression (contd.)

Using linear regression, we predict h(x) as:

$$h(x) = \hat{y} = \sum w_j x_j = w^T x$$

where the value of $\hat{y}$ is continuous.

Let's say instead of predicting continuous values for $\hat{y}$, we are taking probabilities P. Thus the value of P must also lie within 0 and 1. But h(x) consists of value ranging from – Inf to + Inf.

So instead of just taking probabilities we take odds:

i.e. $\text{odds} = \frac{p}{1-p}$

Logistic Regression (contd.)

The **odds** are defined as the probability that the event will occur divided by the probability that the event will not occur. Thus, Odds are nothing but the ratio of the probability of success and probability of failure.

The value of odds lies between 0 to +ve infinity.

And, If the odds are high (million to one), the probability is almost 1.00. If the odds are tiny (one to a million), the probability is tiny, almost zero.

To convert from a probability to odds, divide the probability by one minus that probability. So if the probability is 10% or 0.10 , then the odds are 0.1/0.9 or ‘1 to 9’ or 0.111.

To convert from odds to a probability, divide the odds by one plus the odds. So to convert odds of 1/9 to a probability, divide 1/9 by 10/9 to obtain the probability of 0.10.

Logistic Regression (contd.)

But, the problem here with the ***odds*** is that the range is restricted as values of odds lies between to Infinity and we don’t want a restricted range because if we do so then our correlation will decrease.

By restricting the range it is difficult to model a variable. Hence, in order to control this we take the log of odds which has a range from (-∞,+∞).

So, we get,

$$\log\left(\frac{p}{1-p}\right) = \sum_{j=1}^{n} w_j x_j = w^T x$$

Now, taking exponentiation on both side

$$\exp\left(\log\left(\frac{p}{1-p}\right)\right) = \exp(w^T x)$$

Logistic Regression (contd.)

$$\frac{p}{1-p} = \exp(w^T x)$$
$$\Rightarrow \; p = \exp(w^T x) - p.\exp(w^T x)$$
$$\Rightarrow 1 = \frac{\exp(w^T x)}{p} - \exp(w^T x)$$
$$\Rightarrow 1 + \; \exp(w^T x) = \frac{\exp(w^T x)}{p}$$
$$\Rightarrow p(1 + \exp(w^T x)) = \exp(w^T x)$$
$$\Rightarrow \; p = \frac{\exp(w^T x)}{(1+\exp(w^T x))}$$

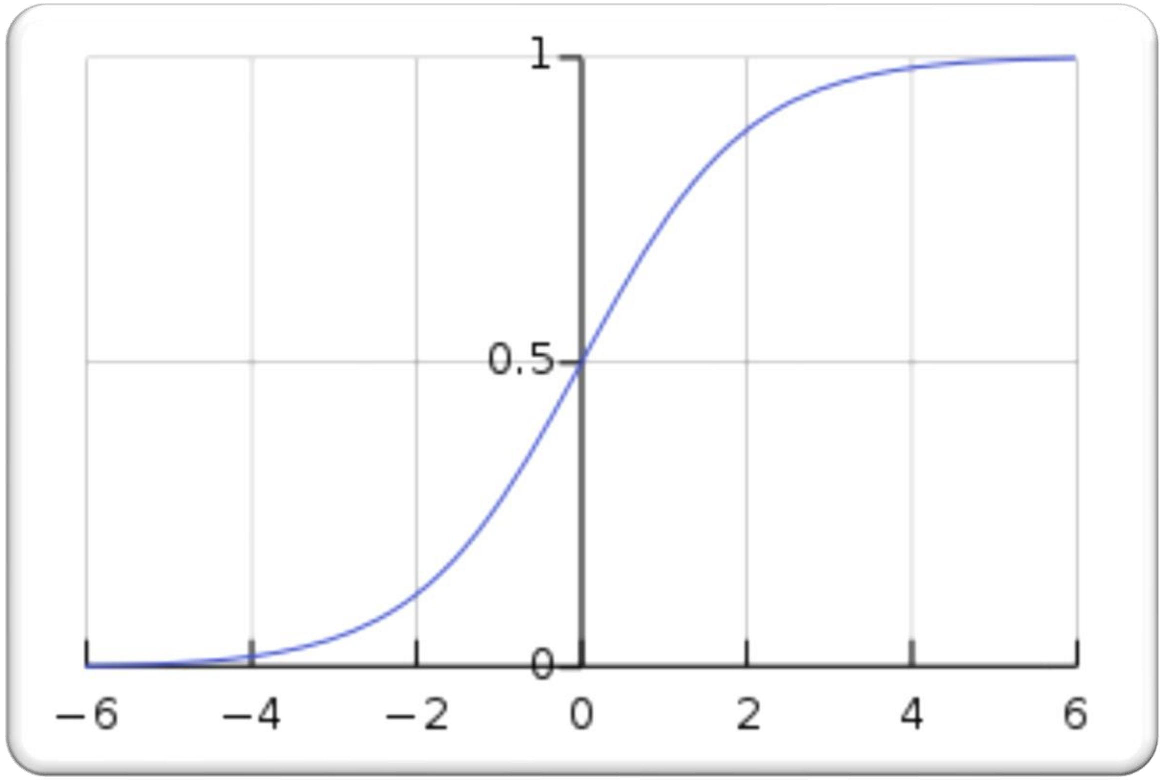
Now, dividing both side by $\exp(w^T x)$

$$\Rightarrow p = \frac{\exp(w^T x)/\exp(w^T x)}{(1+\exp(w^T x))/\exp(w^T x)}$$
$$\Rightarrow p = \frac{1}{1/\exp(w^T x)+1}$$
$$\Rightarrow p = \frac{1}{\exp(-w^T x)+1}$$
$$\Rightarrow p = \frac{1}{1+\exp(-z)} \quad \text{where } z = w^T x$$

This gives us the **sigmoid function** or the logistic function which is the function of z or $w^T x$.

Logistic Regression (contd.)

- Graphically, the sigmoid function is



Logistic Regression (contd.)

From above, we have our hypothesis for logistic function as:

$$h(x) = g(z) = \frac{1}{1 + \exp(-z)}$$

Here, as $g(z)$ tends towards 1 as $z \to \infty$, and $g(z)$ tends towards 0 as $z \to -\infty$. This now shows the value of $h(x)$ is bounded within 0 and 1.

We now get the equation for logistic function which we can use for logistic regression. But how do we fit the model? How do we determine the weight coefficient of our model?

Logistic Regression (contd.)

Given x, let the probability that the given data point x belongs to class 1

$$P(y = 1|\, x; w) = h(x)$$
$$P(y = 0|\, x; w) = 1 - h(x)$$

Or we can write as:

$$p(y|x; w) = (h(x))^{y}\big(1\, - h(x)\big)^{1-y}$$

Lets assume that the m training examples were generated independently, we can the write the likelihood of the parameters as:

$$L(w) = p(y|X; w) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; w)$$
$$= \prod_{i=1}^{m} (h\big(x^{(i)}\big))^{y^{(i)}}\big(1\, - h(x)\big)^{1-y^{(i)}}$$

Logistic Regression (contd.)

Instead of maximizing $L(w)$, we can also maximize any strictly increasing function of $L(w)$. In particular, to make our derivation simple we maximize the likelihood:

$$l(w) = \log L(w) = \log\left(\prod_{i=1}^{m} (h(x^{(i)}))^{y^{(i)}} (1 - h(x))^{1-y^{(i)}}\right)$$

$$\Rightarrow l(w) = \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})$$

This is also the form of loss function commonly known as Cross Entropy Loss function, commonly used across classification problem.

But how do we maximize the likelihood?

Answer to this is, similar to gradient descent used in linear regression, we can use gradient ascent since we need to maximize the likelihood. And this is given as:

$$w_j = w_j + \alpha \frac{\partial \mathrm{L}}{\partial w_j}$$

Logistic Regression (contd.)

Then, let's determine the derivate of log likelihood and as before we consider only one training example to ease our derivation:

$$\frac{\partial\, l(w)}{\partial w_j} = \frac{\partial}{\partial w_j}\left[y \log h(x) + (1-y)\log(1-h(x))\right]$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \frac{\partial}{\partial w_j}\left[y \log\left(g(\boldsymbol{w}^T\boldsymbol{x})\right) + (1-y)\log(1-g(\boldsymbol{w}^T\boldsymbol{x}))\right]$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = y\frac{\partial}{\partial w_j}\log\left(g(\boldsymbol{w}^T\boldsymbol{x})\right) + (1-y)\frac{\partial}{\partial w_j}\log\left(1 - g(\boldsymbol{w}^T\boldsymbol{x})\right)$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = y\frac{\partial}{\partial g(\boldsymbol{w}^T\boldsymbol{x})}\log\left(g(\boldsymbol{w}^T\boldsymbol{x})\right)\frac{\partial\, g(\boldsymbol{w}^T\boldsymbol{x})}{\partial w_j} + (1 - y)\frac{\partial \log\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)}{\partial\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)}\frac{\partial\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)}{\partial g(\boldsymbol{w}^T\boldsymbol{x})}\frac{\partial\, g(\boldsymbol{w}^T\boldsymbol{x})}{\partial w_j}$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[y\frac{1}{g(\boldsymbol{w}^T\boldsymbol{x})} + (1-y)\frac{1}{1-g(\boldsymbol{w}^T\boldsymbol{x})}(0-1)\right]\frac{\partial\, g(\boldsymbol{w}^T\boldsymbol{x})}{\partial w_j}$$

* $\boldsymbol{w}^T\boldsymbol{x}$ *is a vector dot product form of* $\boldsymbol{w}$ *and* $\boldsymbol{x}$ *vector and equals to* $\sum w_j x_j$

Logistic Regression (contd.)

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[\frac{y}{g(\boldsymbol{w}^T\boldsymbol{x})} - \frac{(1-y)}{1-g(\boldsymbol{w}^T\boldsymbol{x})}\right]\frac{\partial}{\partial w_j} g(\boldsymbol{w}^T\boldsymbol{x})$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[\frac{y\big(1-g(\boldsymbol{w}^T\boldsymbol{x})\big)-(1-y)g(\boldsymbol{w}^T\boldsymbol{x})}{g(\boldsymbol{w}^T\boldsymbol{x})\big(1-g(\boldsymbol{w}^T\boldsymbol{x})\big)}\right] g(\boldsymbol{w}^T\boldsymbol{x})\big(1-g(\boldsymbol{w}^T\boldsymbol{x})\big)\, x_j \; \textit{(*How?)}$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \big[y\big(1-g(\boldsymbol{w}^T\boldsymbol{x})\big) - (1-y)g(\boldsymbol{w}^T\boldsymbol{x})\big]x_j$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \big[y\,\big(1 \;- h(x)\big) \;-(1-y)h(x)\big]x_j$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = [y \;- yh(x) - h(x) + yh(x)]x_j$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = [y \;- h(x)]x_j$$

Which is the required form.

$$\boxed{\begin{aligned} g'(z) &= \frac{\partial}{\partial z}\Big(\frac{1}{1+e^{-z}}\Big) = \frac{\partial\big(\frac{1}{1+e^{-z}}\big)}{\partial(1+e^{-z})}\,\frac{\partial(1+e^{-z})}{\partial\, e^{-z}}\,\frac{\partial\, e^{-z}}{\partial z} \\ &= -\frac{1}{(1+e^{-z})^2}\,(0+1)(-e^{-z}) \\ &= \frac{e^{-z}}{(1+e^{-z})^2} \\ &= \frac{e^{-z}}{1+e^{-z}}\,\frac{1}{1+e^{-z}} \\ &= \frac{-1+1+e^{-z}}{1+e^{-z}}\,\frac{1}{1+e^{-z}} \\ &= \Big(\frac{-1}{1+e^{-z}} + \frac{1+e^{-z}}{1+e^{-z}}\Big)\frac{1}{1+e^{-z}} \\ &= (-g(z)+1)\,g(z) = g(z)(\,1-g(z) \end{aligned}}$$

Logistic Regression (contd.)

Then, we can rewrite our update rule given above as:

$$w_j = w_j + \alpha(y^{(i)} - h(x^{(i)})x_j^{(i)}$$

Which is the stochastic form of the gradient ascent update rule.

Perceptron Learning Algorithm

Consider modifying the logistic regression to "force" it to produce either 0 or 1 exactly as an output. Then, we can express it as a threshold function as:

$$g(z) = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; if \; z < 0 \end{cases}$$

And using this definition along with the update rule as:

$$w_j = w_j + \alpha \left(y^{(i)} - h\left(x^{(i)}\right) \right) x_j^{(i)}$$

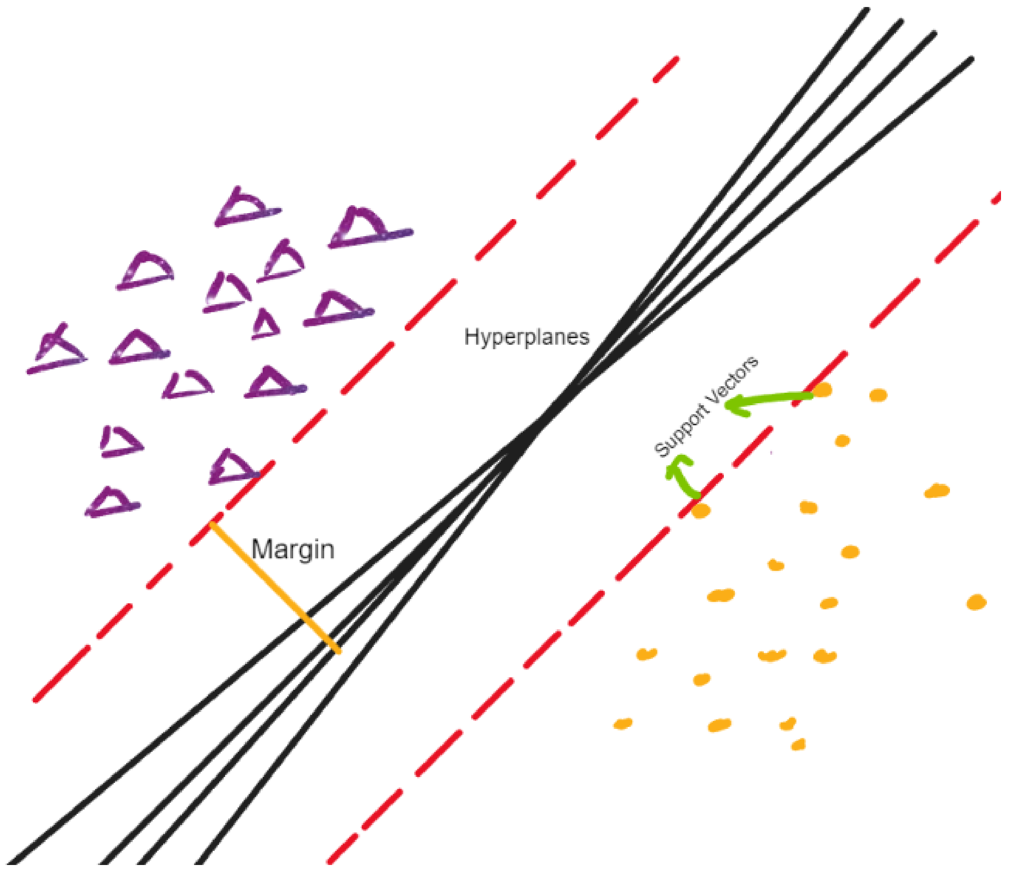
We get the perceptron learning algorithm which used to be considered as the model for human's brain neuron in early days.

Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about – basically, logistic regression, it is actually a very different type of algorithm; in particular, it is difficult to endow the perceptron's predictions with meaningful probabilistic interpretations, or derive the perceptron as a maximum likelihood estimation algorithm.

Support Vector Machines (SVM)

- Support Vector Machines (SVM) is a powerful supervised machine learning algorithm used for classification and regression analysis.
- Support Vector Machine (SVM) is non parametric supervised machine learning algorithm that fits best for classification problem. But it also works well with regression problem.
- In N-dimensional input space, our goal using SVM is to learn a hyperplane that best separates the input space into classes.
    - Hyperplane is a line in 2D space and just a point in 1D space.
- Multiple hyperplane can exits in given input space that separates classes. And, the optimal hyperplane is the one among all plausible hyperplanes that separates classes with largest margin.

SVM (contd.)



- Here, the distance between hyperplane and the closest data point is called **margin**.
- The margin is computed only with the closest points called `**support vectors**` as the perpendicular distance from these support vectors with the hyperplane.

SVM (contd.)

- They are called `support vectors` because they support the construction of hyperplane determining their orientation.
- Thus in SVM, the optimal hyperplane is obtained from the learning process using training data through the maximization of margin.
- SVM is not just limited to linear classification, SVM's can efficiently perform a non-linear classification, implicitly mapping their inputs into high dimensional feature space.

SVM (contd.)

- The distance of any line, ax + by + c = 0 from a given point say, (x0 , y0) is given by d.

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{(a^2 + b^2)}}$$

- Similarly, the distance of a hyperplane equation $w^T X + b = 0$ from a given point x_0 can be written as:

$$d = \frac{|w^{TX} + b|}{||w||_2}$$

Hard Margin vs. Soft Margin

- We have already defined margin as the perpendicular distance between support vectors and the hyperplanes. Also, we choose the hyperplane with maximal margin as the optimal one.
- But, in SVM, we have notion of two different margin viz. Hard Margin and Soft Margin.
- In SVM, the hyperplane is expected to separate the classes clearly such that the points belonging to one class is on one side of the hyperplane only and the points belonging to the another class is on other side of the hyperplane only. This is called **Hard Margin** which is default state of SVM.
- On the other hand, if we allow some data points to cross the boundary set by hyperplane i.e. we allow some misclassification of points by relaxing the hard constraints set by hyperplane. This is called **Soft Margin**.

Hard Margin vs. Soft Margin (contd.)

- We incorporate soft margin by introducing slack variables C. This is a regularization parameter that defines how much misclassification are we allowing for i.e. C determines the no. of observations allowed to cross the boundary set by hyperplane.
- The smaller the value of C the more violations of boundary or misclassification is allowed and is more sensitive to the training data. This is the case of higher variance and lower bias.
- Conversely, the larger the value of the algorithm is more sensitive towards training data causing lower variance and higher bias.
- Conclusion is, hard margin implementation has larger value of C and Soft margin implementation has lesser value of C.

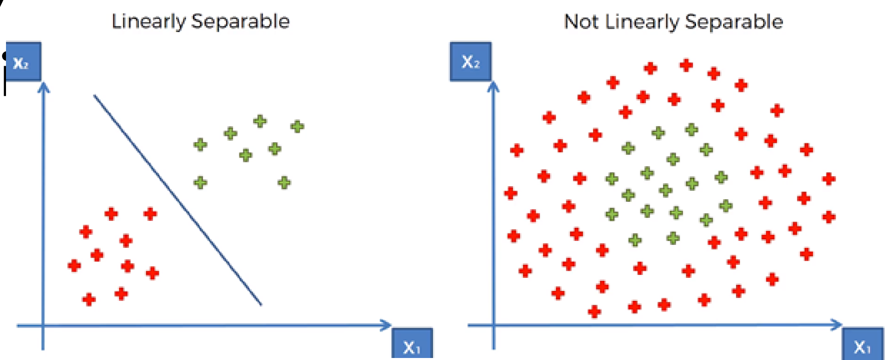
Types of SVM

Linear SVM

- Linear SVM is used for linearly separable data.
- It finds the optimal hyperplane that separates the two classes using a linear function.
- The hyperplane w maximum margin the best classifier.

Non Linear SMV

- Nonlinear SVM is used for non-linearly separable data.
- It uses a kernel function to transform the data into a higher-dimensional feature space, where a linear boundary can be used to separate the classes.
- function can be linear, l, Gaussian, or sigmoid.



Kernelized SVM

- We have already mentioned above that the SVM not only works with linear data but also with non linear data. It does so by transforming lower dimension data into higher dimension such that classes are linearly separable. And this is achieved through the use of so called `Kernel`.
- In SVM, a kernel is a function that performs mapping of the input data from the original feature space to a higher-dimensional space.
- The kernel trick enables SVM to find a nonlinear decision boundary in the feature space by transforming the data into a higher-dimensional space where a linear boundary can be used.
- The learning of hyperplane explained above is the result of linear kernel.

Types of Kernel

1. Linear Kernel

The linear kernel is the dot product between new data point and the support vectors.

For each data point s_j in X, compute

$$K(s_i, s_j) = s_i . s_j$$

where, s_i is the support vector.

Types of Kernel (contd.)

2. Polynomial Kernel

The polynomial kernel is given as:

foreach data point s_j in X, compute

$$K(s_i, s_j) = (s_i . s_j + 1)^d$$

where d is the degree of polynomial and is a hyperparameter which can't be learned through training.

Types of Kernel (contd.)

3. Gaussian Kernel

It is general purpose Kernel and used when there is no prior knowledge about data. It is given as:

$$K(s_i, s_j) = \exp\left(-\frac{\left|\left|s_i - s_j\right|\right|^2}{2\sigma^2}\right)$$

Where,

σ is the standard deviation.

Types of Kernel (contd.)

4. Gaussian Radial Basis Function (RBF) Kernel

It is also general purpose Kernel and used when there is no prior knowledge about data. It is given as:

$$K(s_i, s_j) = \exp(\gamma \ * \sum(s_j \ - s_i^2))$$

Where, gamma is the hyperparameter which defines how far the single training example influence is. Thus, higher value of gamma consider points closer to the plausible hyperparameter and conversely, lower value of gamma consider farther points.

Types of Kernel (contd.)

5. Laplace RBF Kernel

It is also general purpose Kernel and used when there is no prior knowledge about data. It is given as:

$$K(s_i, s_j) = \exp\left(-\frac{||s_i - s_j||}{\sigma}\right)$$

SVM Numerical

Consider the following data points $(3, 1)$, $(3, -1)$, $(6, 1)$, $(6, -1)$ as positively labelled data and (1, 0), (0, 1), (0, -1), (-1, 0) as negatively labelled data. Determine the equation of hyperplane that divides the above data points into two classes using SVM and predict in which class (5, 2) belongs.

Solution:

We use Linear Kernel.

We identify support vectors are (1,0), (3, 1) and (3, -1). (**Hint:** *Draw the graph and obtained from the graph for your ease*)

Let s1 =(1, 0), s2 = (3, 1) and s3 = (3, -1).

SVM Numerical (contd.)

Since there are three support vectors, we need three variables, so our equation becomes,

$$\alpha s1.s1 + \beta s2.s1 + \lambda s3.s1 = -1$$
$$\alpha s1.s2 + \beta s2.s2 + \lambda s3.s2 = 1$$
$$\alpha s1.s3 + \beta s2.s3 + \lambda s3.s3 = 1$$

*Note:

1. *We are using linear kernel thus we are using dot products.*
2. *In first equation RHS has -1 since the support vector s1 is negatively labelled class and other two equation has +1 on RHS as s2 and s3 belongs to positively labeled class.*

SVM Numerical (contd.)

Adding bias coefficient 1 to the points we get support vectors as:

$$s1 = (1, 0, 1), \qquad s2 = (3, 1, 1), \qquad s3 = (3, -1, 1)$$

From the dot product we get,

$$s1.s1 = 2, \qquad s1.s2 = 4, \qquad s1.s3 = 4, \qquad s2.s2 = 11,$$
$$s3.s3 = 11$$

So our equation becomes,

$$2\alpha + 4\beta + 4\lambda = -1$$
$$4\alpha + 11\beta + 9\lambda = 1$$
$$4\alpha + 9\beta + 11\lambda = 1$$

On solving, we get $\alpha = -3.5, \beta = 0.75 \;\&\; \lambda = 0.75$.

SVM Numerical (contd.)

Now, weight vector of hyperplane is

$w = \alpha . s1 + \beta . s2 + \lambda . s3 = -3.5\ * (1, 0, 1) + 0.75\ * (3, 1, 1) + 0.75\ * (3, -1, 1)$

$w = \ (1, 0, -2)$

Where $w_0\ =\ -2$ (bias), $w_1 =\ 1$ and $w_2 =\ 0.$

So the equation becomes

$y = w_1 x_1 + w_2 x_2 + w_0 = 1\ * x1 + 0\ * x2\ - 2$

$y = x_1 - 2$

SVM Numerical (contd.)

Let's now make prediction for (5, 2):

We have,

$$y = x_1 - 2 = 5 - 2 = 3$$

Which means it belongs to positive class.

KNN (K – Nearest Neighbor) Algorithm

- K-Nearest Neighbor (KNN) is a non-parametric machine learning algorithm used for classification and regression tasks.
- It is a supervised learning technique.
- It is based on the idea of finding the K nearest neighbors to a new observation in the training set and assigning the observation to the class that has the majority of neighbors.
- But the new point into the class that is most similar to one of the available classes.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset into memory and when it gets new data, then it classifies that data into a class that is much similar to the new data.

How KNN works?

- The KNN algorithm works by finding the distance between the new observation and all the other observations in the training set.
- The distance is usually calculated using Euclidean distance.
- The algorithm then selects the K nearest neighbors to the new observation based on their distances.
- Finally, the new observation is assigned to the class that has the majority of neighbors among the K nearest ones.

Working of KNN Algorithm

- Given training data is:

BRIGHTNESS	SATURATION	CLASS
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue

The new data point to classify is:

BRIGHTNESS	SATURATION	CLASS
20	35	?

Working of KNN Algorithm (contd.)

- Next, we compute a distance between new point and the existing points using Euclidean distance.

$$dist = \sqrt{\{(x_2 - x_1)^2 + (y_2 - y_1)^2\}}$$

BRIGHTNESS	SATURATION	CLASS	DISTANCE
40	20	Red	25
50	50	Blue	33.54
60	90	Blue	68.01
10	25	Red	10
70	70	Blue	61.03
60	10	Red	47.17
25	80	Blue	45

Working of KNN Algorithm (contd.)

- Now, we rearrange the distances in ascending order:

BRIGHTNESS	SATURATION	CLASS	DISTANCE
10	25	Red	10
40	20	Red	25
50	50	Blue	33.54
25	80	Blue	45
60	10	Red	47.17
70	70	Blue	61.03
60	90	Blue	68.01

Since we choose 5 as the value of K, we'll only consider the first five rows.

As you can see above, the majority class within the 5 nearest neighbors to the new entry is Red.

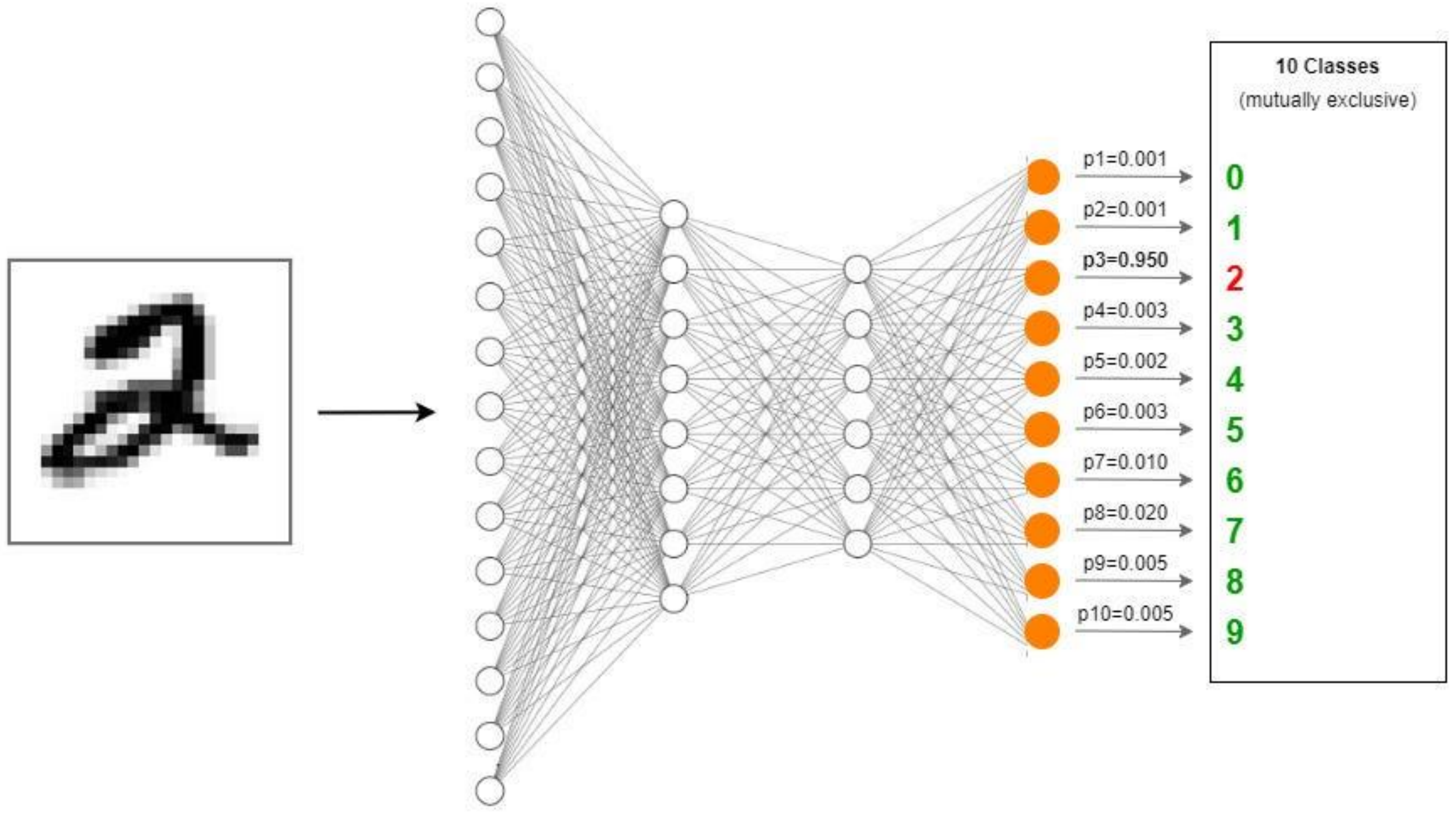
How to choose the value of K in KNN?

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to inaccurate predictions due to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties. However, always use an odd number as the value of K.
- Use Hierarchical clustering.

Multi-Class Classification

- Multiclass classification is a classification task with more than two classes. Each sample can only be labeled as one class.
- For example, classification using features extracted from a set of images of fruit, where each image may either be of an orange, an apple, or a pear.
- Each image is one sample and is labeled as one of the 3 possible classes.
- Multiclass classification makes the assumption that each sample is assigned to one and only one label - one sample cannot, for example, be both a pear and an apple.

Multi-Class Classification



Multi-Class Classification (contd.)

- Multi-class classification is obtained through softmax regression which is given as:

$$g\left(z_j\right) = \frac{\exp(z_j)}{\sum_j \exp(z_j)}$$

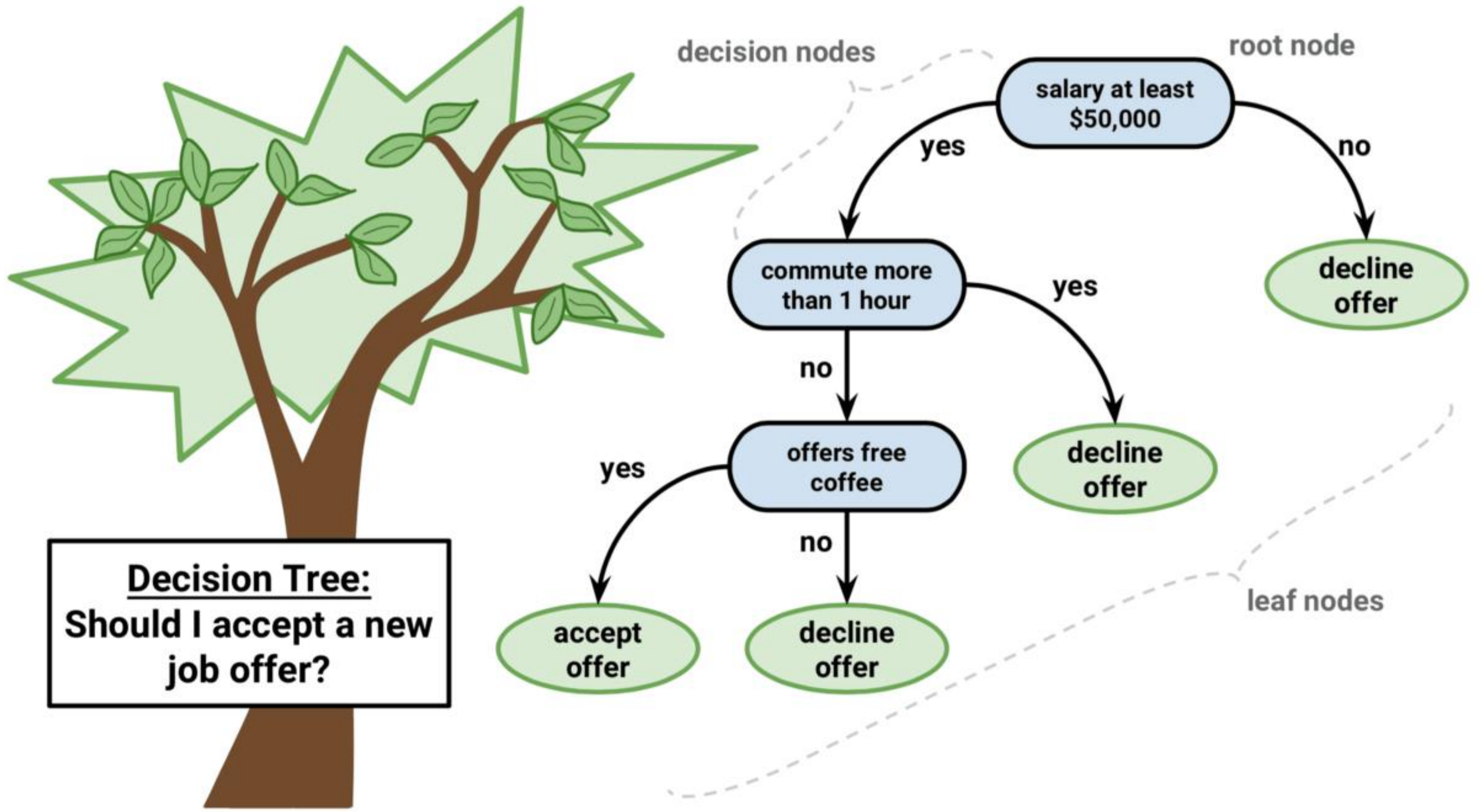
Where, j = 1…|C| and $z = \boldsymbol{w}^T\boldsymbol{x}$

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

Decision Tree

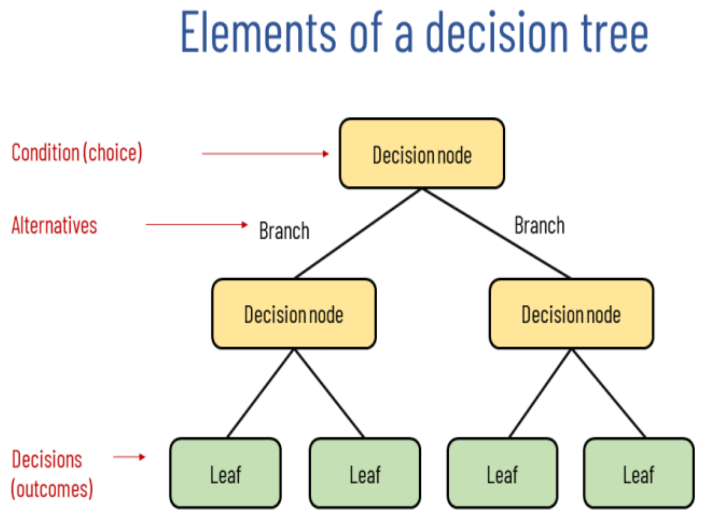
- Decision Trees are also a non-parametric model used for both regression and classification.
- Depiction of inverted tree like structure gave it a name tree and since it is used to make decisions, thus it is called decision tree.
- Each internal node denotes an attribute with some branching condition, if it has, extending to child nodes.
- Decision tree is a graphical representation of all possible solutions to a decision.
- In ML, It is constructed by recursively splitting the training data into subsets based on the values of the attributes until a stopping criterion is met, such as the maximum depth of the tree or the minimum number of samples required to split a node.

Decision Tree



Decision Tree

- During training, the Decision Tree algorithm selects the best attribute to split the data based on a metric such as entropy or Gini impurity, which measures the level of impurity or randomness in the subsets.
- The goal is to find the attribute that maximizes the information gain or the reduction in impurity after the split.
- Decision trees also provide the foundation for more advanced ensemble methods such as bagging, random forests and gradient boosting.



Construction of Decision Tree

- There are various techniques to construct decision tree:
  - CART (Classification and Regression Trees)
  - ID3 (Iterative Dichotomizer 3)
  - C4.5
  - CHAID (Chi-Squared Automatic Interaction Detection)

Classification and Regression Tree (CART)

- CART is simple to understand, interpret, visualize and requires little or no effort for data preparation.
- Moreover, it performs feature selection.
- Regression trees are mainly used when the target variable is numerical.
- Here value obtained by a terminal node is always the mean or average of the responses falling in that region. As a result, if any unseen data or observation will predict with the mean value.
- Classification is used when the target variable is categorical.
- Here value obtained by a terminal node is the mode of response falling in that region and any unseen data or observation in this region will make a prediction based on the mode value.

CART (contd.)

- The decision to make a strategic split heavily affects the accuracy of the tree and the decision criteria for regression and classification trees will be different.
- Entropy/Information gain or Gini Index can be used for choosing the best split.
- For a given dataset with different features, to decide which feature to be considered as the root node and which feature should be the next decision node and so on, information gain of each feature should be known.
- The feature which has maximum information gain will be considered as the root node.

CART (contd.)

- Entropy: Entropy is a measure of disorder or impurity in the given dataset.
- In the decision tree, messy data are split based on values of the feature vector associated with each data point.
- With each split, the data becomes more homogenous which will decrease the entropy. However, some data in some nodes will not be homogenous, where the entropy value will not be small.
- The higher the entropy, the harder it is to draw any conclusion. When the tree finally reaches the terminal or leaf node maximum purity is added.

Decision Tree for Classification, Entropy, and Information Gain | by Sandhya Krishnan | CodeX | Medium | CodeX

CART (contd.)

- For a dataset that has C classes and the probability of randomly choosing data from class i is p_i. Then entropy E(S) can be mathematically represented as

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

Then information gain is given as:

$$Gain(A, S) = E(S) - E(A, S)$$

Where,

$$, S) = \sum_{j=1}^{v} \frac{|S_j|}{|S|} . E(S_j)$$

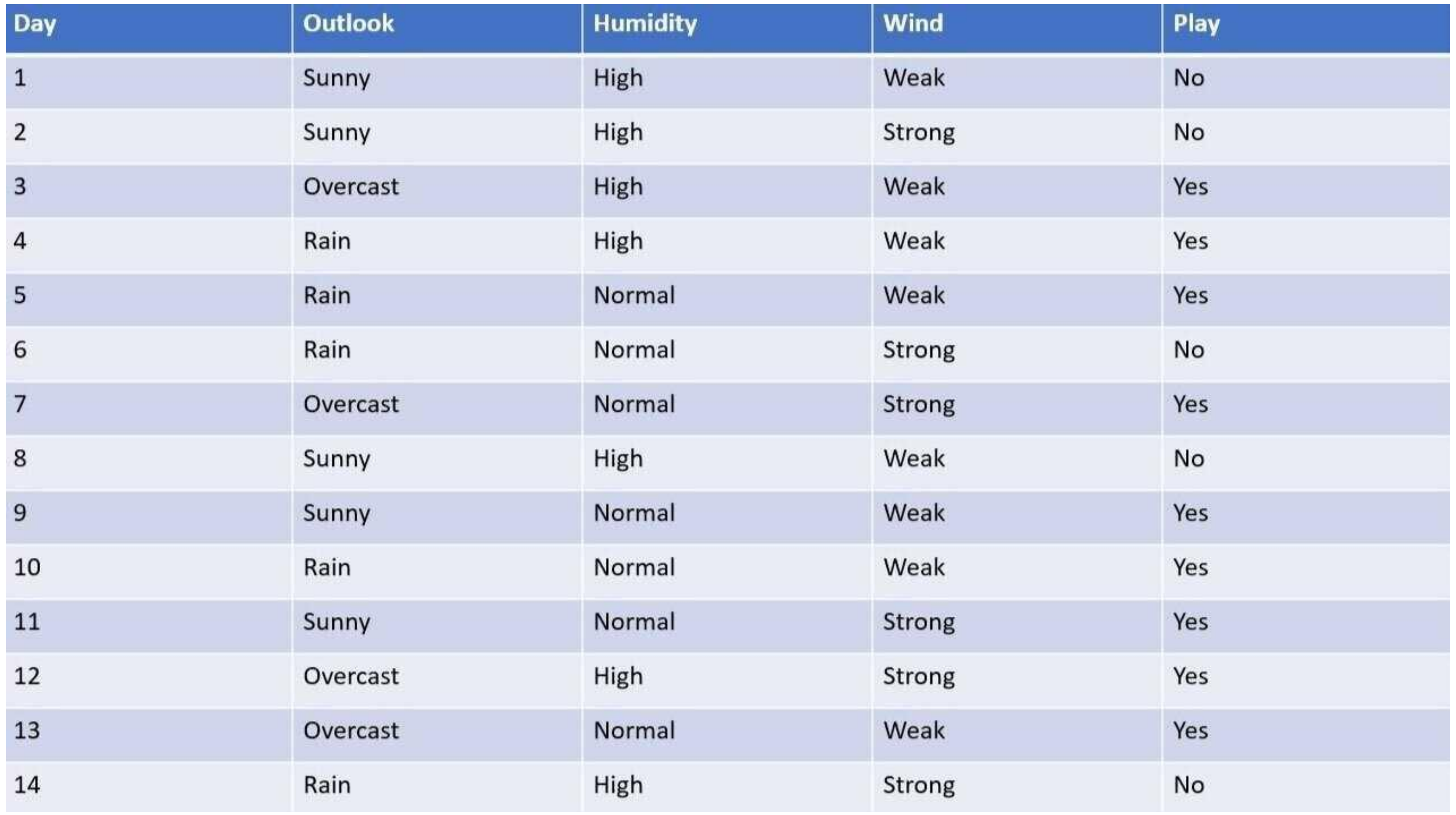
Information gain indicates how much information a particular variable or feature gives us about the final outcome. It can be found out by subtracting the entropy of a particular attribute inside the data set from the entropy of the whole data set.

<u>Notations</u>

$H(S)$: entropy of whole data set S
$|S_j|$: number of instances with j value of an attribute A
$|S|$: total number of instances in the dataset
v - set of distinct values of an attribute A
$E(S_j)$ - entropy of subset of instances for attribute A
$E(A, S)$ - entropy of an attribute A

CART (contd.)

- Let’s take an example:



Day	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No

CART (contd.)

First in order to find the root node, we calculate the entropy of our dataset as

$$E(S) = -\left(\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right) = 0.94$$

How?

Check on the decision node, there are two events as outcomes. One is yes and the other is no. Out of 14 records, 9 are yes and 5 are no. Thus,

$$P(yes) = 9/14$$

$$P(no) = 5/14$$

CART (contd.)

Next, we compute the information gain for each feature. For that compute,

$$E(Outlook, S)$$
$$= -\frac{5}{14}\left(-\left(\frac{2}{5}\log_2\frac{2}{5}+\frac{3}{5}\log_2\frac{3}{5}\right)\right)+\frac{4}{14}\left(-\left(\frac{4}{4}\log_2\frac{4}{4}\right)\right)$$
$$+ -\frac{5}{14}\left(-\left(\frac{2}{5}\log_2\frac{2}{5}+\frac{3}{5}\log_2\frac{3}{5}\right)\right)=0.693$$

CART (contd.)

Now, Information Gain

$$IG(Outlook) = E(S) - E(Outlook, S) = 0.94 \ - 0.693 = 0.247$$

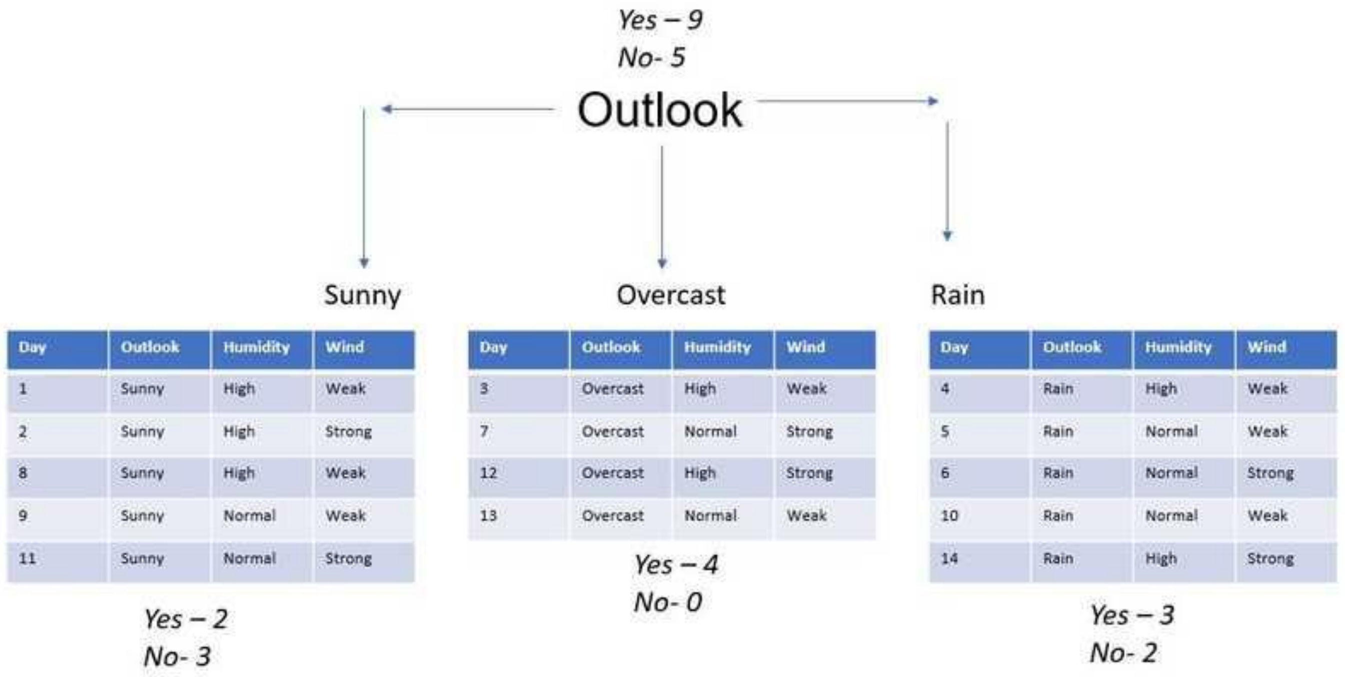
Repeat same for other attributes as well. So,

$IG(Wind) = 0.94 \ - \frac{8}{14}\left(-\left(\frac{6}{8}\log_2\frac{6}{8} + \frac{2}{8}\log_2\frac{2}{8}\right)\right) + \frac{6}{14}\left(-\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right)\right) = 0.048$

$IG(Humidity) = 0.94 \ - \frac{7}{14}\left(-\left(\frac{3}{7}\log_2\frac{3}{7} + \frac{4}{7}\log_2\frac{4}{7}\right)\right) + \frac{7}{14}\left(-\left(\frac{6}{7}\log_2\frac{6}{7} + \frac{1}{7}\log_2\frac{1}{7}\right)\right) =$
0.151

CART (contd.)

Since, the information gain of the Outlook is highest, Outlook will be the root of the tree and we get tree as



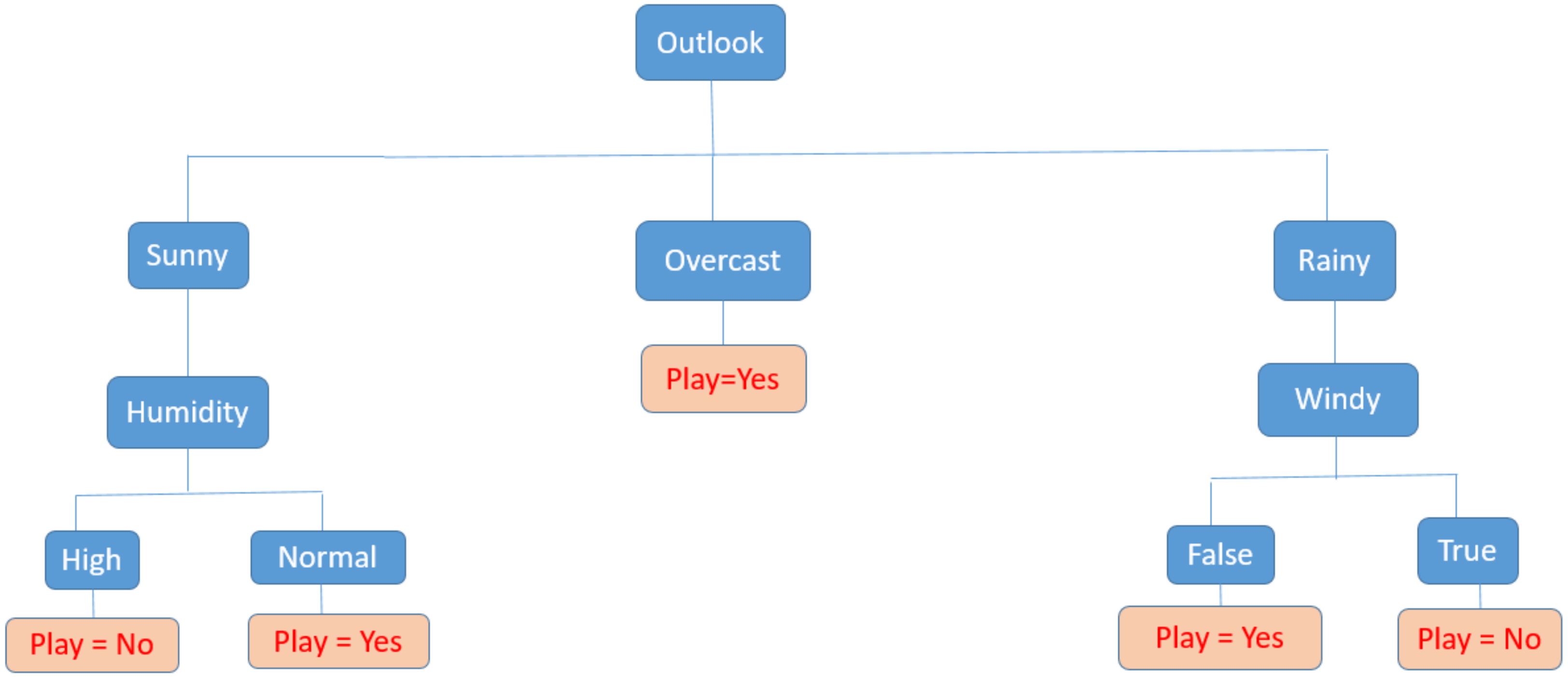
Day	Outlook	Humidity	Wind
1	Sunny	High	Weak
2	Sunny	High	Strong
8	Sunny	High	Weak
9	Sunny	Normal	Weak
11	Sunny	Normal	Strong

Day	Outlook	Humidity	Wind
3	Overcast	High	Weak
7	Overcast	Normal	Strong
12	Overcast	High	Strong
13	Overcast	Normal	Weak

Day	Outlook	Humidity	Wind
4	Rain	High	Weak
5	Rain	Normal	Weak
6	Rain	Normal	Strong
10	Rain	Normal	Weak
14	Rain	High	Strong

CART (contd.)

- Repeat the process until you get decision on leaf node as below:



Random Forest

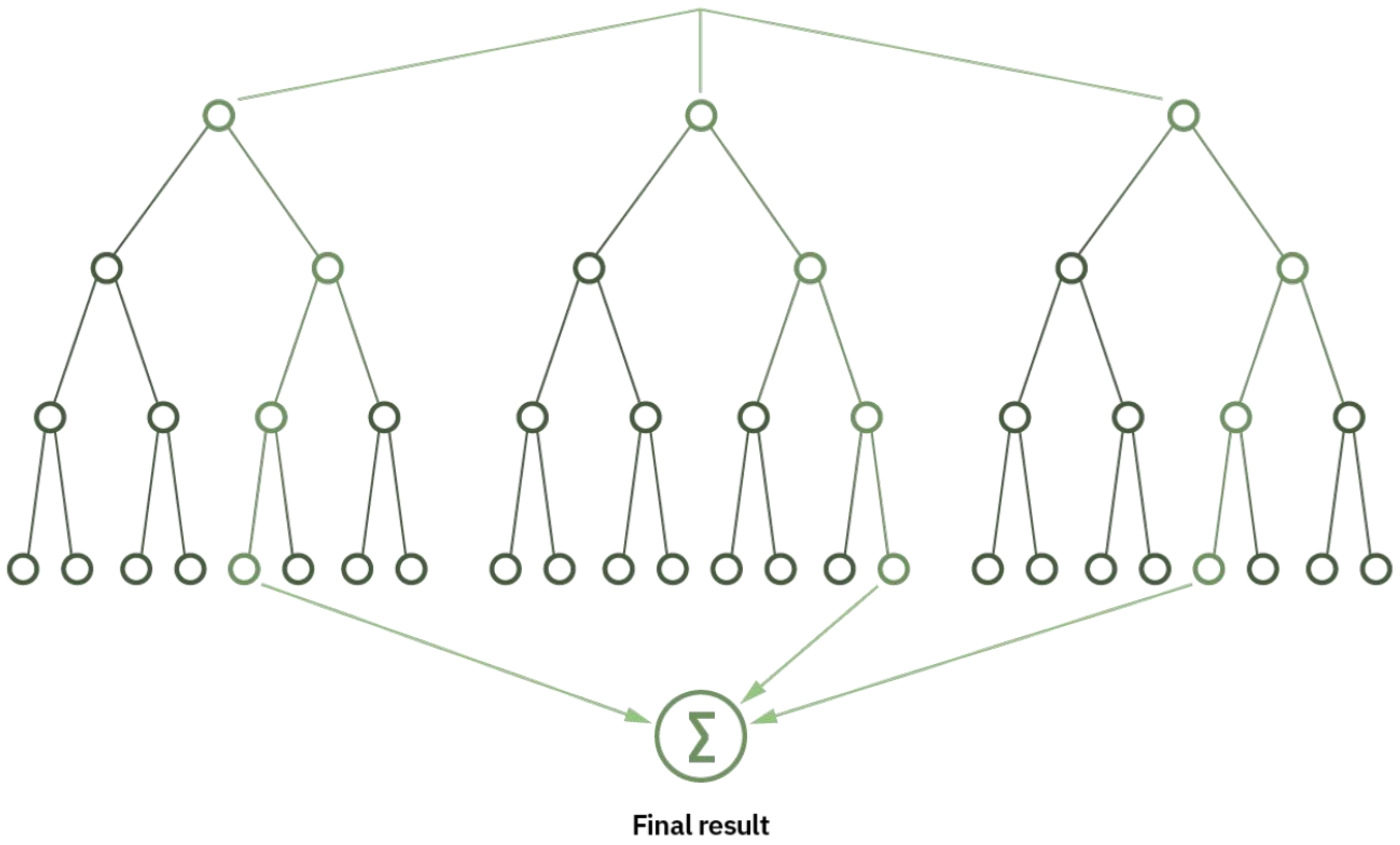
- Random Forest is a commonly-used ML algorithm which gives the foundation to the rise of **ensemble method**.
- Forest basically means the multiple trees.
- Thus, Random Forest combines the output of multiple decision tree to determine single outcome with feature randomness to create an uncorrelated forest of decision trees.
- Feature randomness generates a random subset of features, which ensures low correlation among decision trees. *(This marks the key difference between decision trees and random forests.)*

What is Random Forest? | IBM

Random Forest (contd.)

- One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables, as in the case of regression, and categorical variables, as in the case of classification.
- It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

Random Forest (contd.)



Essential Features of Random Forest

- Miscellany: Each tree has a unique attribute, variety and features concerning other trees. Not all trees are the same.
- Immune to the curse of dimensionality: Since a tree is a conceptual idea, it requires no features to be considered. Hence, the feature space is reduced.
- Parallelization: We can fully use the CPU to build random forests since each tree is created autonomously from different data and features.
- Train-Test split: In a Random Forest, we don’t have to differentiate the data for train and test because the decision tree never sees 30% of the data.
- Stability: The final result is based on Bagging, meaning the result is based on majority voting or average.

Steps Involved in Random Forest Algorithm

1. In the Random forest model, a subset of data points and a subset of features is selected for constructing each decision tree. Simply put, n random records and m features are taken from the data set having k number of records.
2. Individual decision trees are constructed for each sample.
3. Each decision tree will generate an output.
4. Final output is considered based on ***Majority Voting or Averaging*** for Classification and regression, respectively.

When to Avoid Using Random Forests?

Random Forests Algorithms are not ideal in the following situations:

- **Extrapolation**: Random Forest regression is not ideal in the extrapolation of data. Unlike linear regression, which uses existing observations to estimate values beyond the observation range.
- **Sparse Data**: Random Forest does not produce good results when the data is sparse. In this case, the subject of features and bootstrapped sample will have an invariant space. This will lead to unproductive spills, which will affect the outcome.

Pros and Cons of Random Forest

Pros

- Reduced risk of overfitting: Decision trees run the risk of overfitting as they tend to tightly fit all the samples within training data. However, when there's a robust number of decision trees in a random forest, the classifier won't overfit the model since the averaging of uncorrelated trees lowers the overall variance and prediction error.
- Provides flexibility: Since random forest can handle both regression and classification tasks with a high degree of accuracy, it is a popular method among data scientists. Feature bagging also makes the random forest classifier an effective tool for estimating missing values as it maintains accuracy when a portion of the data is missing.
- Easy to determine feature importance: Random forest makes it easy to evaluate variable importance, or contribution, to the model. There are a few ways to evaluate feature importance. Gini importance and mean decrease in impurity (MDI) are usually used to measure how much the model's accuracy decreases when a given variable is excluded. However, permutation importance, also known as **mean decrease accuracy (MDA)**, is another importance measure. MDA identifies the average decrease in accuracy by randomly permutating the feature values in oob samples.

Pros and Cons of Random Forest

Cons

- Time-consuming process: Since random forest algorithms can handle large data sets, they can be provide more accurate predictions, but can be slow to process data as they are computing data for each individual decision tree.
- Requires more resources: Since random forests process larger data sets, they'll require more resources to store that data.
- More complex: The prediction of a single decision tree is easier to interpret when compared to a forest of them.

Naïve Bayes

- Naive Bayes is a probabilistic machine learning algorithm used for classification tasks.
- It is based on Bayes' theorem and assumes feature independence, where Bayes theorem is given as:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

Where

$P(H|D)$ is the posterior probability distribution – which means probability of hypothesis being true/false given data such that $P(H)$ is the prior probability distribution – which means prior measure/belief of probability of hypothesis being true or false and $P(D|H)$ is the likelihood - which means probability of data coming from hypothesis when hypothesis is true/false.

Naïve Bayes (contd.)

Consider the task of classifying email to HAM (non-SPAM) and SPAM based on the content of mail. With this in place, our model should be able to classify any new inbox email as SPAM or non-SPAM email. This is one of the typical application of machine learning for text classification.

Let's say we have a training dataset which contains email labelled as SPAM and HAM. We will represent email using feature vector whose length is equal to the number of words in the dictionary.

Dictionary means here the complete collection of words from email in training data.

Now, if a sample email contains following words (say):

$$Million, \qquad Dollors, \qquad Congratulations\ etc.$$

Naïve Bayes (contd.)

Then the feature vector for the above email looks like:

$$\left[\begin{array}{c} 0 \\ 0 \\ . \\ 1 \\ . \\ 1 \\ . \\ . \\ 1 \\ . \\ . \\ 0 \end{array}\right] \quad \begin{array}{c} a \\ ant \\ . \\ Congratulation \\ . \\ Dollars \\ . \\ . \\ Million \\ . \\ . \\ Zebra \end{array}$$

Dictionary contains all words from email and it is commonly known as **Vocabulary**. If a word is present in email, than its corresponding feature vector contains 1 for its presence otherwise 0.

Naïve Bayes (contd.)

We now have to model $p(x|y)$ such that if y = 1, then email is SPAM and if y = 0 then email is HAM.

To do so, we make a very strong assumption about the features independence such that $x_i's$ are conditionally independent given y. This is known as **Naïve Assumption**.

Consider a vocabulary of size $50{,}000$, For instance, if $y = 1$ means spam email; "congratulation" is word 185 and "*dollar*" is word 789; then we are assuming that if we know $y = 1$ (*that a particular piece of email is spam*), then knowledge of x_{185} (knowledge of whether "*congratulation*" appears in the message) will have no effect on your beliefs about the value of x_{789} (whether "*dollar*" appears).

More formally, this can be written $p(x_{123}|y) = p(x_{185}|y, x_{789})$.

(Note that this is not the same as saying that x_{123} and x_{789} are independent, which would have been written "$p(x_{123}) = p(x_{123}|x_{789})$"; rather, we are only assuming that x_{123} and x_{789} are conditionally independent given y.)

Naïve Bayes (contd.)

We now compute, from the law of probabilities

$P(x_1, \dots, x_n | y) = \ p(x_1|y)p(x_2|y;\ x_1)p(x_3|y;\ x_1, x_2) \dots p(x_n|y;\ x_1, x_2, \dots, x_{n-1})$

$P(x_1, \dots, x_n | y) = p(x_1|y)p(x_2|y)p(x_3|y) \dots p(x_{n-1}|y)$ [*using naïve assumptions*]

$P(x_1, \dots, x_n | y) = \ \prod_{j=1}^{n} p(x_j|y)$

Now, fitting into bayes theorem,

$$P(y|x_1, \dots, x_n) = \frac{P(x_1, \dots, x_n|y)P(y)}{P(x_1, \dots, x_n)}$$

As explained already, denominator can be omitted for now. So we get,

Naïve Bayes (contd.)

$$P(y|x_1,\ldots,x_n) = P(x_1,\ldots,x_n|y)P(y) = \prod_{j=1}^{n} p(x_j, y)p(y) = p(\mathrm{y}) \prod_{j=1}^{n} p(x_j|y)$$

The class prediction is the y with maximum value of $P(y|x_1,\ldots,x_n)$ i.e.

$$class = argmax_y \left(p(\mathrm{y}) \prod_{j=1}^{n} p(x_j|y) \right)$$

In case of multiclass,

$$class = argmax_{c_k} \left(p(c_k) \prod_{j=1}^{n} p(x_j|c_k) \right)$$

Numerical Example - Naïve Bayes (contd.)

Type of family structure	Age group	Income status	Will they buy a car?
Nuclear	Young	Low	Yes
Extended	Old	Low	No
Childless	Middle-aged	Low	No
Childless	Young	Medium	Yes
Single Parent	Middle-aged	Medium	Yes
Childless	Young	Low	No
Nuclear	Old	High	Yes
Nuclear	Middle-aged	Medium	Yes
Extended	Middle-aged	High	Yes
Single Parent	Old	Low	No

Will a person buy a car when Single Parent, Young and Low are the inputs?

Naïve Bayes (contd.)

- Firstly, let's compute the probability of the output labels $P(C_i)$ given the data.

$$P(No) = 4/10$$
$$P(Yes) = 6/10$$

- Now let's calculate the probability of the likelihood of the evidence. Given the inputs Childless, Young, and Low, we'll calculate the probability with respect to both class labels as follows:

$$P(\text{Single Parent}|Yes) = \frac{1}{6}, \qquad P(\text{Single Parent}|No) = \frac{1}{4}$$
$$P(Young|Yes) = \frac{2}{6}, \qquad P(Young|No) = \frac{1}{4}$$
$$P(Low|Yes) = \frac{1}{6}, \qquad P(Low|No) = 4/4$$

Naïve Bayes (contd.)

- Now the posterior probability is calculated as:

$$P(Yes|X) = P(\text{"}Single\ Parent\text{"}|Yes) * P(Young|Yes) * P(Low|Yes) = 1/6 * 2/6 * 1/6 = 0.0093$$

$$P(No|X) = P(\text{"}Single\ Parent\text{"}|No) * P(Young|No) * P(Low|No) = 1/4 * 1/4 * 4/4 = 0.0625$$

Naïve Bayes (contd.)

The final probabilities are:

$$P(Yes|X) = 0.0093/(0.0625 + 0.0093) = 0.13$$

$$P(No|X) = 0.0625/(0.0093 + 0.0625) = 0.87$$

Thus, the results clearly show that the car probably will not be purchased.

Bias Variance Tradeoff

- The prediction error for any machine learning algorithm can be broadly divided into:

1. Bias Error
2. Variance Error
3. Irreducible Error

- The Irreducible error cannot be reduced regardless of what algorithm we use. These sorts of error occurred due to influence of some unknown factor.
- However, we can take control of bias and variance to some level.

Bias Variance Tradeoff (contd.)

Bias

- Bias refers to the difference between average prediction of our model and the actual value or label in data.
- Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training data and test data.
- Bias is caused by underfitting and is the result of simplify taking the problem.
- Bias occurs when we use simpler model for complex tasks i.e. for example we using linear regression to model polynomial regression of d-th degree.

- **Low Bias**: Low bias value means fewer assumptions are taken to build the target function. In this case, the model will closely match the training dataset.
- **High Bias**: High bias value means more assumptions are taken to build the target function. In this case, the model will not match the training dataset closely.

Bias Variance Tradeoff (contd.)

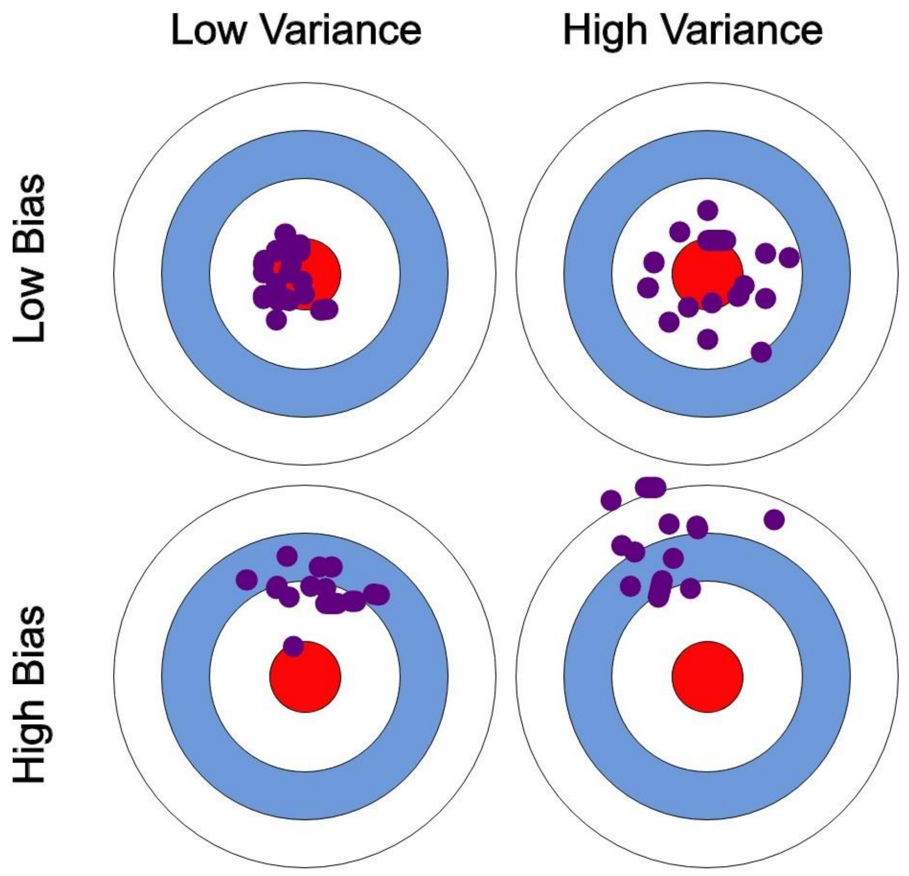
Variance

- Variance is taken as the variability of a model prediction for a given data point.
- Model with high variance pays a lot of attention to the training data and does not generalize on the data which it hasn't seen before.
- As a result, such models perform very well on training data but has high error rates on test data.
- High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs.
- **Low variance:** Low variance means that the model is less sensitive to changes in the training data and can produce consistent estimates of the target function with different subsets of data from the same distribution. This is the case of underfitting when the model fails to generalize on both training and test data.
- **High variance:** High variance means that the model is very sensitive to changes in the training data and can result in significant changes in the estimate of the target function when trained on different subsets of data from the same distribution. This is the case of overfitting when the model performs well on the training data but poorly on new, unseen test data. It fits the training data too closely that it fails on the new training dataset.

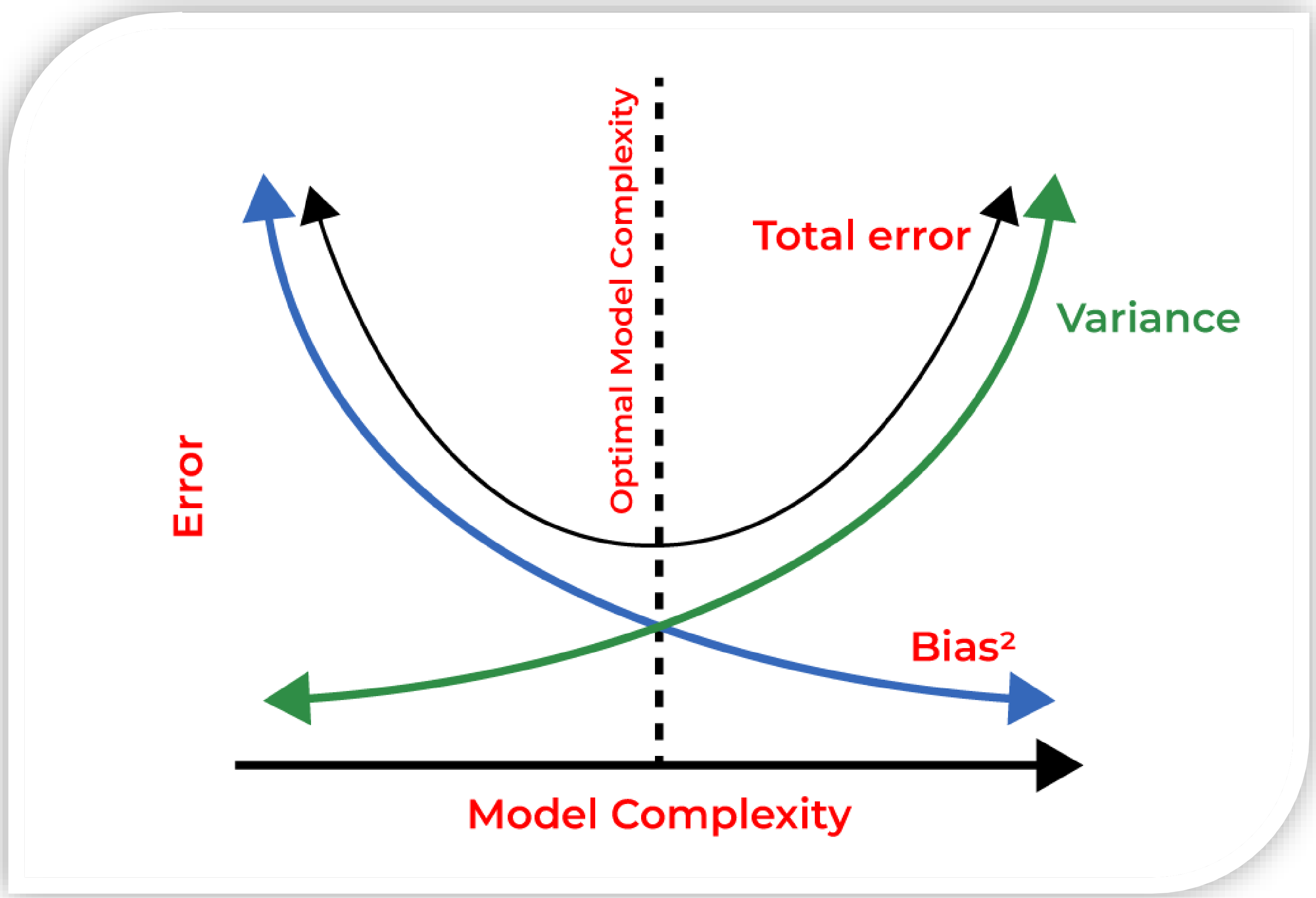
Bias Variance Tradeoff (contd.)

- If our model is too simple and has very few parameters then it may have high bias and low variance.
- On the other hand, if our model has large number of parameters then its going to have high variance and low bias.
- So we need to find the right balance without overfitting and underfitting the data. This keeps the MSE minimal.

$$MSE = Bias^2 + Variance + Irreducible\ Error$$



Bias Variance Tradeoff (contd.)



Ensemble Learning

- Ensemble in literal term refers to the group of artists performing together. Machine learning use similar notion of ensemble where group of machine learning models learn together to perform better i.e. predict better results.
- For example, if you have to buy a laptop, you don't directly go randomly to any shop and buy a laptop that shopkeeper shows to you. You visit multiple shops, based on your requirement, specifications available in the market and budget you have you make a choice. But you investigate the laptop models of different brands of different costs and varieties of spec. You may also consult with your friends and family before you make a purchase. This is the process of accumulating information from all possible sources and make a best decision.

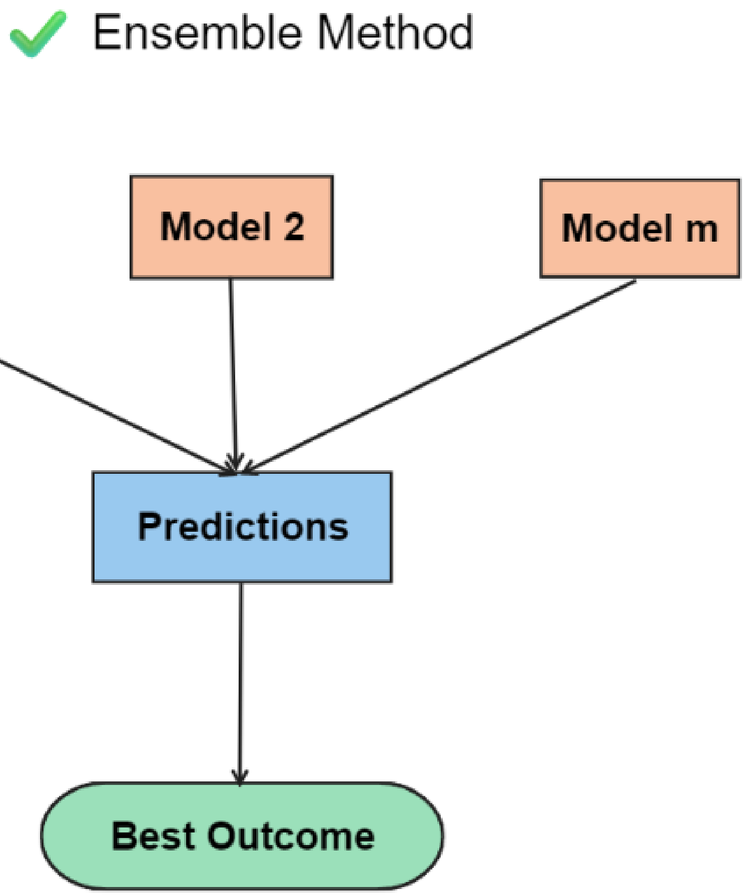
Ensemble Learning (contd.)

- Same applies to ensemble learning in machine learning.
- Ensemble learning is a technique to combine the prediction power of two or more models for better prediction on machine learning tasks.
- That is, ensemble learning combines the decisions of multiple models in order to improve the overall performance of model to make better decision.
- In ensemble learning technique, several learners are combined to obtain a better performance than any learners individually.
- From the above definition, we have already known that the ensemble technique is more powerful then individual models.

Ensemble Learning (contd.)

- Thus, **Ensemble Learning** is an approach of combining the multiple machine learning models with an objective to maximize predictability accurateness on the same problem individual models were employed.
- It mainly aims to reduce biases that may exists in individual models through collective intelligence.
- It utilizes the strengths of multiple models to compensate each model weakness.
- The underlying concept behind ensemble of multiple models is to predict more precise outcome than what individual model is capable of thus, by utilizing the strengths of multiple models.
- This way, we not only maximize preciseness of outcome but also provide resilience against uncertainties in data providing more robust and reliable prediction.

Ensemble Method

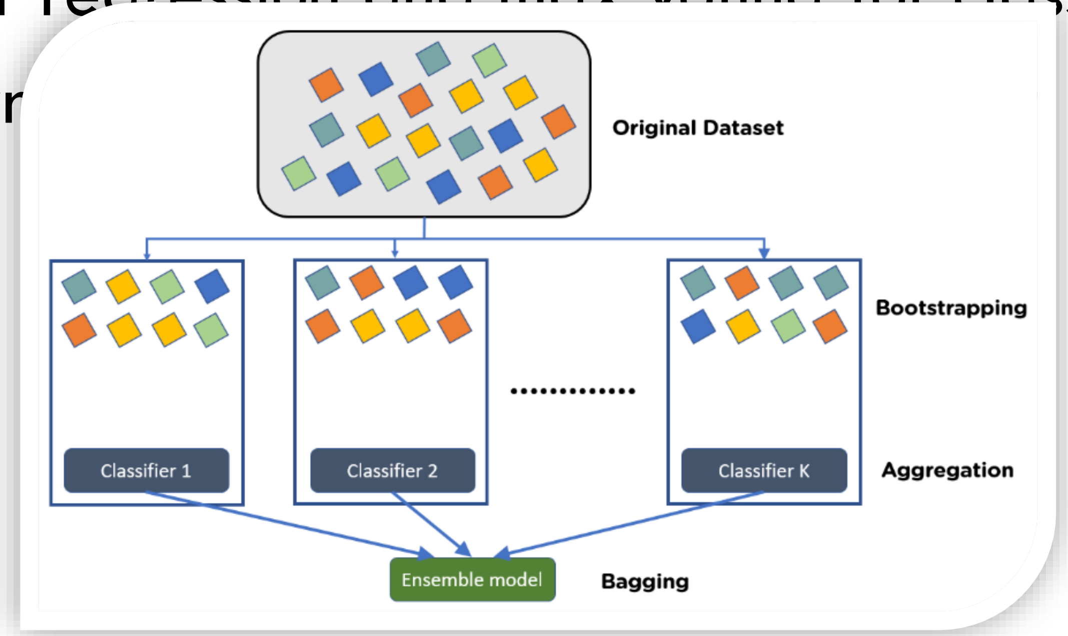


Ensemble Learning

- The primary behind ensemble is to combine weak learners to perform better than individual weak learner.
- Ensemble Learning models are broadly categorized into homogeneous and heterogeneous and common methods are:
    1. Bagging
    2. Boosting
    3. Stacking
- In homogeneous model, a single base machine learning model is used. On other hand, different machine learning models are used in heterogeneous model.
- Ensemble Learning algorithms are also called meta-learning algorithms.

1. Bagging

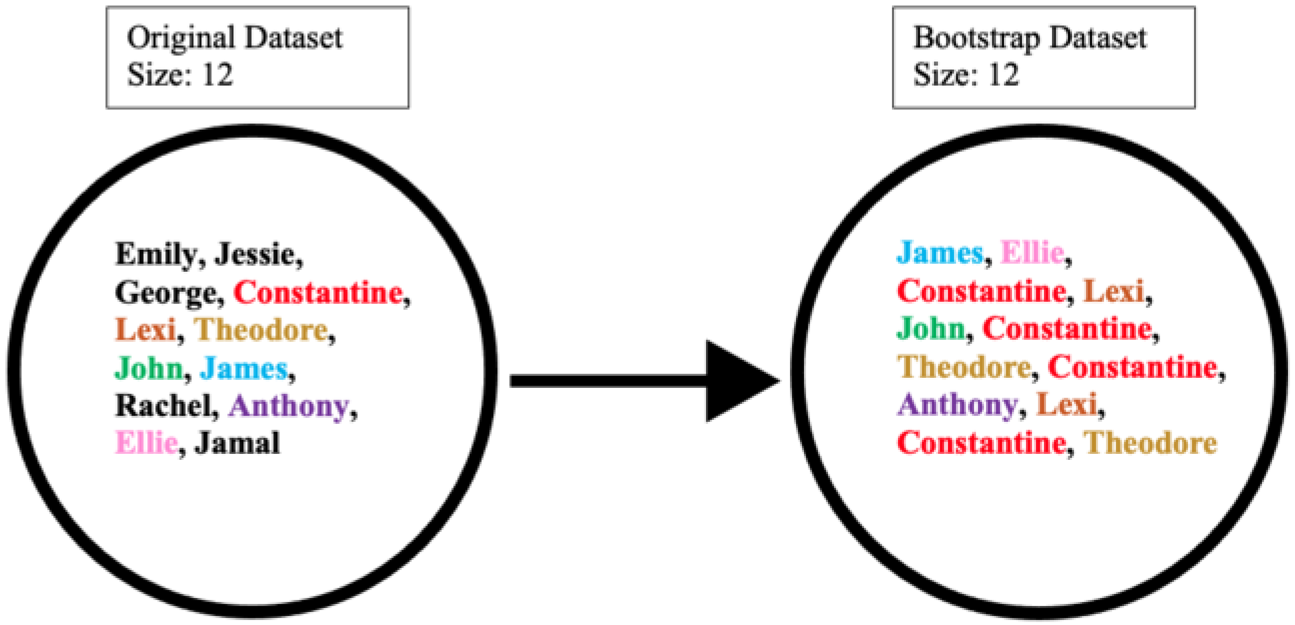
- Bagging (*Bootstrap Aggregation*) is one of the popular ensemble technique that combines several homogeneous weak learners and combines them.
- This ensemble technique use some sort of aggregation technique such as averaging for regression and max voting for classification.
- Bagging is known dataset.



1. Bagging (contd.)

What is Bootstrapping?

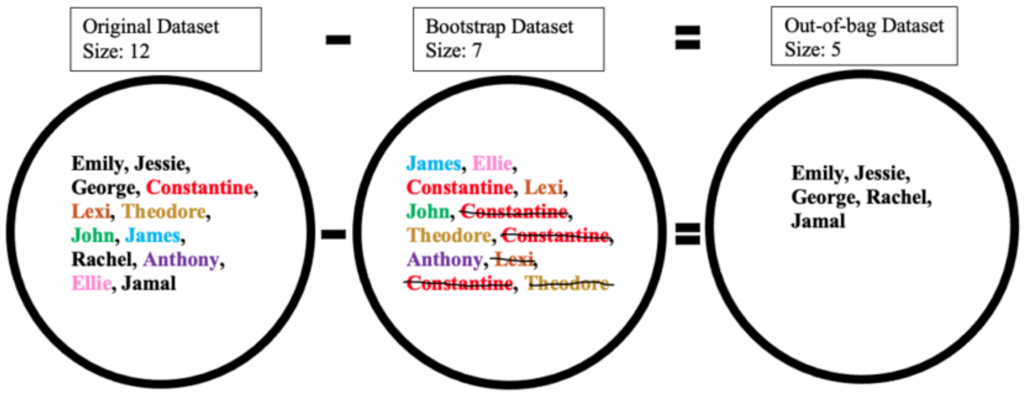
Bootstrapping is the method of randomly creating samples of data out of a population with replacement to estimate a population parameter.


Original Dataset
Size: 12
Emily, Jessie, George, Constantine, Lexi, Theodore, John, James, Rachel, Anthony, Ellie, Jamal
Bootstrap Dataset
Size: 12
James, Ellie, Constantine, Lexi, John, Constantine, Theodore, Constantine, Anthony, Lexi, Constantine, Theodore

The bootstrap dataset is made by randomly picking objects from the original dataset. Also, **it must be the same size as the original dataset.** However, the difference is that the bootstrap dataset can have duplicate objects.

1. Bagging (contd.)

- The **out-of-bag dataset** represents the remaining samples who were not in the bootstrap dataset.
- It can be calculated by taking the difference between the original and the bootstrap datasets.
- Keep in mind that since both datasets are sets, when taking the difference the duplicate names are ignored in the bootstrap dataset.

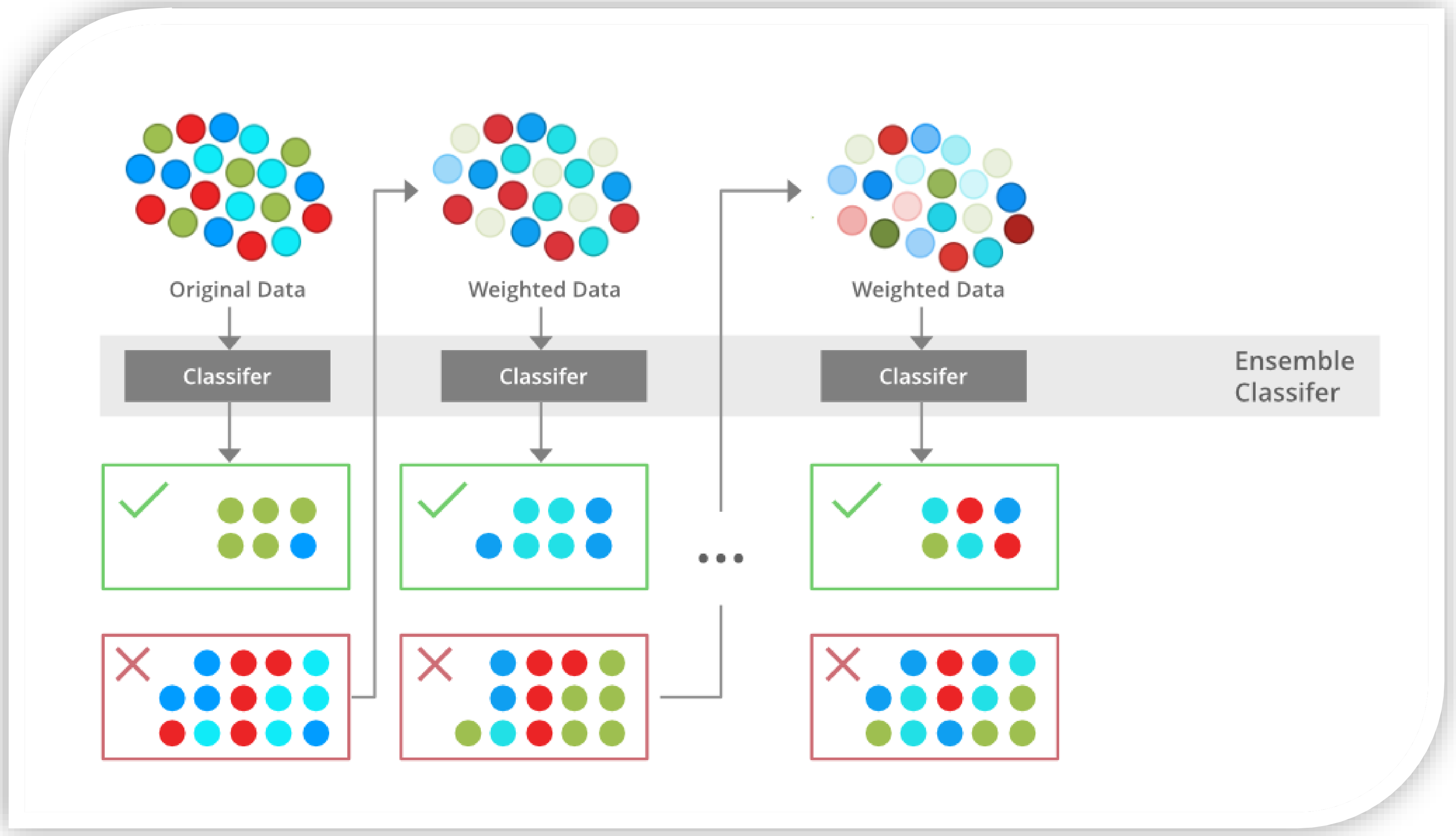

Original Dataset
Size: 12
-
Bootstrap Dataset
Size: 7
=
Out-of-bag Dataset
Size: 5
Emily, Jessie, George, Constantine, Lexi, Theodore, John, James, Rachel, Anthony, Ellie, Jamal
-
James, Ellie, Constantine, Lexi, John, ~~Constantine~~, Theodore, ~~Constantine~~, Anthony, ~~Lexi~~, ~~Constantine~~, ~~Theodore~~
=
Emily, Jessie, George, Rachel, Jamal

Individual ML algorithms are trained in bootstrap dataset. ***Since the algorithm generates multiple trees and therefore multiple datasets the chance that an object is left out of the bootstrap dataset is low***

2. Boosting

- Boosting is another ensemble technique that mainly focus on reducing bias.
- With Boosting, individual models are trained sequentially.
- Boosting combines individual homogeneous weak learners to obtain a strong learner such that individual models are trained sequentially where each subsequent model acts on the weaknesses of previous models.
- Each model in the sequence is trained giving more importance to samples in the dataset which were incorrectly predicted by the preceding models in the sequence.
- Example includes AdaBoost (Adaptive Boost), XGBoost (Gradient Boost) etc.

2. Boosting


Original Data
Weighted Data
Weighted Data
Classifer
Classifer
Classifer
Ensemble Classifer

2. Boosting

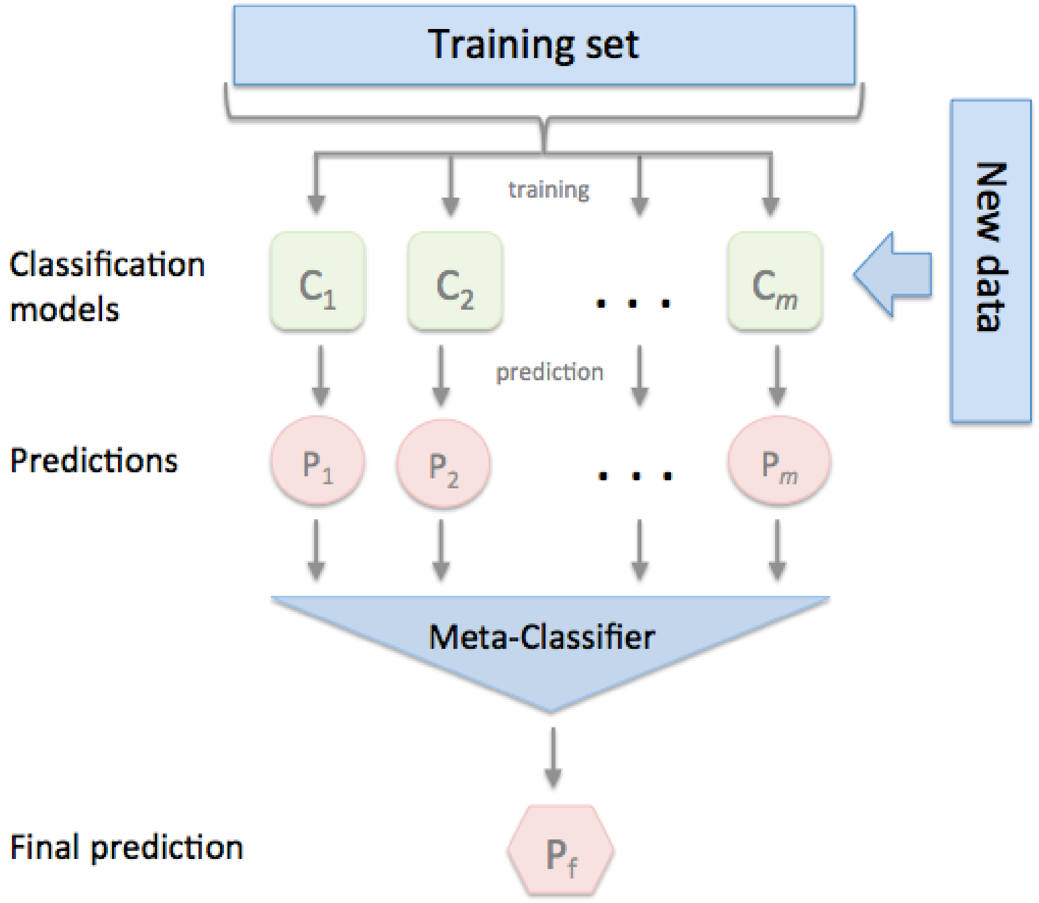
- AdaBoost initially gives the same weight to each dataset. Then, it automatically adjusts the weights of the data points after every decision tree. It gives more weight to incorrectly classified items to correct them for the next round. It repeats the process until the residual error, or the difference between actual and predicted values, falls below an acceptable threshold.
- Gradient Boosting (GB) is similar to AdaBoost in that it, too, is a sequential training technique. The difference between AdaBoost and GB is that GB does not give incorrectly classified items more weight. Instead, GB optimizes the loss function by generating base learners sequentially so that the present base learner is always more effective than the previous one. This method attempts to generate accurate results initially instead of correcting errors throughout the process, like AdaBoost.

What is Boosting? - Boosting in Machine Learning Explained - AWS (amazon.com)

3. Stacking

- Stacking is ensemble of heterogeneous weak learner.
- Stacking combines several heterogeneous base models aka. first level models to obtain final prediction.
- Stacking involves another machine learning model in second level aka meta-model that combines the output from the first level to make final prediction.
- The primary idea of stacking is to feed the predictions of numerous base models into a higher-level model known as the meta-model or blender, which then combines them to get the final forecast.

3. Stacking


Training set
training
New data
Classification models
C_1
C_2
. . .
C_m
prediction
Predictions
P_1
P_2
. . .
P_m
Meta-Classifier
Final prediction
P_f

End of the Chapter