1. **Definitions**:
   a. **Occam’m Razor**: This is a logical principle which which states that one should not increase, beyond what is necessary, the number of entities required to explain anything. For example, given 2 data points, we may draw a straight line through them to predict future data points. However, we could also draw an infinite variety of the most complicated curves passing through those same two points, and these curves would fit the empirical data just as well. Only Occam's razor would in this case guide us in choosing the "straight" (i.e. linear) relation as best candidate model.

   b. **MLE and MAP**:
      i. MLE: Let X_1, X_2,..., X_n be a random sample from a distribution that depends on one or more unknown parameters θ_1, θ_2,..., θ_m with probability density (or mass) function $f(x_i|\ \theta_1, \theta_2,..., \theta_m)$. Suppose that ($\theta_1$, θ_2,..., θ_m) is restricted to a given parameter space Ω. The likelihood function L is the function obtained by the following equation:

      $$L_x(\theta) = f_\theta(x); \quad \theta \in \Omega, \qquad x \in X1, X2, \dots, Xn$$

      In the method of maximum likelihood, we try to find a value of u(x) of the parameter θ that maximizes $L_x(\theta)$for each x. If we can do this, then u(x) is called a maximum likelihood estimator (MLE) of *θ.*

      ii. **MAP** : In the light of MLE, if we want to calculate probability of *θ given X, then from bayes rule*

      prob(*θ/X) = (prob(X/ θ)*prob(θ)) / prob(X)*

      We now seek that value for θ which maximizes the posterior probability prob(θ|X). We denote such a value of θ by $\hat{\theta}$MAP.

   c. **Single perceptron and its learning**: A perceptron is a mathematical function that maps its input x to an output value f (x) such that:

      f(x) = 1 if w0 + w1*x > 0; 0 otherwise

In machine learning, this function used as a binary classi_er to decide the instance (x) is either positive/class (1) or negative/class (0).

d. **Multilayer perceptron and its learning**: Multilayer perceptron (MLP) is a fully connected network of multiple nodes with two or more layer (one or more hidden layer and output layer). Each node in one layer connects with a certain weight w_ij to every node of the following layer. Each node is called neuron with a nonlinear activation function. MLP is a supervised learning technique and learns through a algorithm called back-propagation. MLP is able to learn from non-linearly separable dataset also.

2. MLE for Bernoulli/Binomial Distribution

Given

$$
\begin{aligned}
P(\mathcal{D}|\theta) &= \theta^{N_1}(1-\theta)^{N_0} \\
\text{Log Likelihood } \mathcal{L}(\mathcal{D}|\theta) &= N_1 \log\theta + N_0 \log(1-\theta) \\
\mathcal{L}'(\mathcal{D}|\theta) &= \frac{N_1}{\theta} - \frac{N_0}{1-\theta}
\end{aligned}
$$

Now if we set $\mathcal{L}'(\mathcal{D}|\theta) = 0$ then we will get the optimized value for θ.

$$
\begin{aligned}
&\Rightarrow \frac{N_1}{\theta} - \frac{N_0}{1-\theta} = 0 \\
&\Rightarrow \frac{N_1}{\theta} = \frac{N_0}{1-\theta} \\
&\Rightarrow N_1(1-\theta) = (N - N_1)\theta \\
&\Rightarrow N_1 - N_1\theta = N\theta - N_1\theta \\
&\Rightarrow \theta = \frac{N_1}{N} \\
\text{So, } &\hat{\theta}_{MLE} = \frac{N_1}{N}
\end{aligned}
$$

3. MLE for Poisson Distribution

Log likelihood of Poisson distribution is

$$\begin{aligned}
\mathcal{L}(x|\lambda) &= \log\Big(\prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\Big) \\
&= \sum_{i=1}^{n} \log\Big(\frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\Big) \\
&= \sum_{i=1}^{n} \big(x_i \log \lambda - \lambda - \log(x_i!)\big) \\
&= (\sum_{i=1}^{n} x_i) \log \lambda - n\lambda - \sum_{i=1}^{n} \log x_i!
\end{aligned}$$

Now to get the maximum value of λ, we have to set zero for $\mathcal{L}'(x|\lambda)$ with respect to λ

$$\frac{\sum_{i=1}^{n} x_i}{\hat{\lambda}} - n = 0$$

$$\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\text{Hence, } \hat{\lambda}_{MLE} = \frac{\sum_{i=1}^{n} x_i}{n}$$

4. **Fitting a Naïve Bayes spam filter by hand**

Features	Appearances in spam	Appearances in regular
secret	3	1
offer	2	0
low	0	2
price	0	2
valued	0	1
customer	0	1
today	1	1
dollar	1	0
million	1	0
sports	0	2
is	1	1
for	0	1
play	0	1
healthy	0	1
pizza	0	1

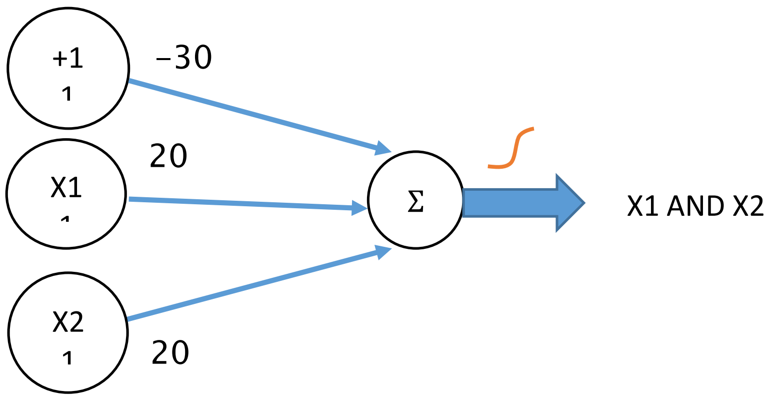
$$\theta_{spam} = \frac{no.of\ spam\ message}{total\ message} = 3/7$$

$$\theta_{secret|spam} = \frac{count(secret,spam)}{count(w,spam)+|V|} = \frac{3}{9+15} = \frac{3}{24} = \frac{1}{8}$$

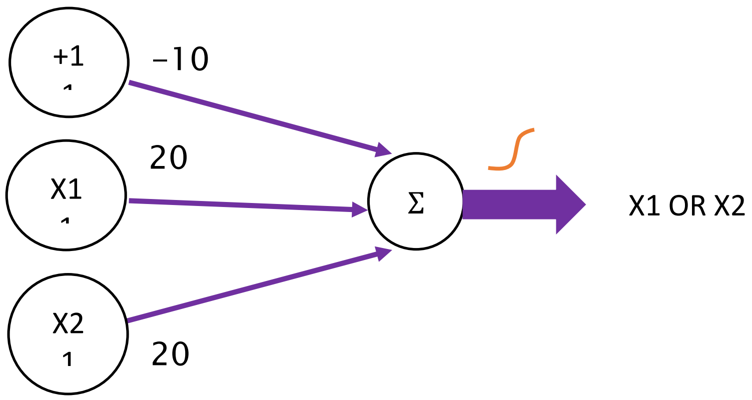
$$\theta_{secret|non-spam} = \frac{count(secret,non-spam)}{count(w,non-spam)+|V|} = \frac{1}{15+15} = \frac{1}{30}$$

$$\theta_{dollar|spam} = \frac{count(dollar,spam)}{count(w,spam)+|V|} = \frac{1}{9+15} = \frac{1}{24}$$

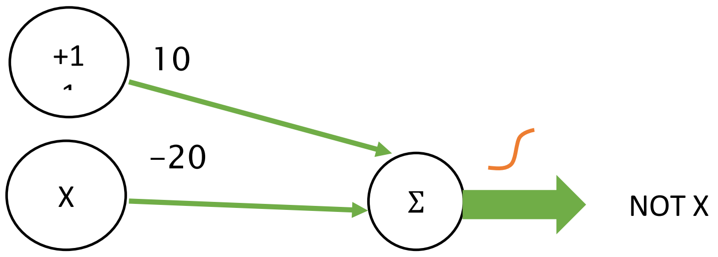
5. **Perceptron warm-up.**



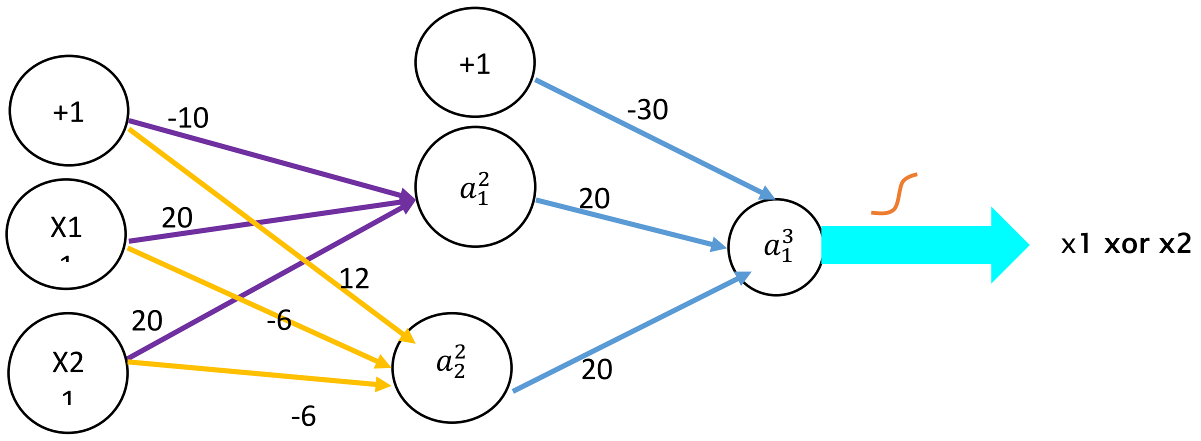
X1	X2	g(-30 + 20*X1 + 20*X2)
0	0	g(-30)≈ 0
0	1	g(-10) ≈ 0
1	0	g(-10) ≈ 0
1	1	g(10) ≈ 1



X1	X2	g(-10 + 20*X1 + 20*X2)
0	0	g(-10)≈ 0
0	1	g(10) ≈ 1
1	0	g(10) ≈ 1
1	1	g(30) ≈ 1



X	g(10 – 20*X)
0	g(10)≈ 1
1	g(−10) ≈ 0



Legend :

🟪	Implements X1 OR X2
🟨	Implements (NOT X1) OR (NOT X2)
🟦	Implements (X1 OR X2) AND [(NOT X1) OR (NOT X2)] = X1 XOR X2

∫ Sigmoid function g(z) = 1/ (1+e^−z)

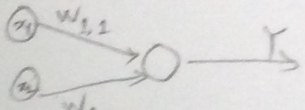
#6 Perceptron learning algorithm:

Desired output $y = f(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$

Let

$a = W^T x + \theta$

$= w_{1,1}x_1 + w_{1,2}x_2 + \theta$

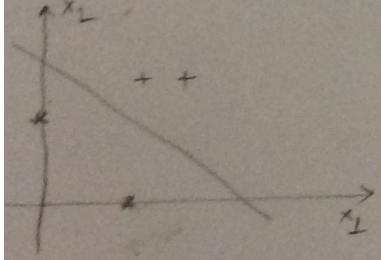


Let's assume, initially,

weight vector, $w = [1\ 1]$ & $\theta = -1.1$

Given training set

x_1	x_2	actual value, t
0	1	0
1	0	0
1	-1	0
2	1	1



1st iteration, $a = [1\ 1]\begin{bmatrix}0\\1\end{bmatrix} - 1.1$

$= -0.1 \Rightarrow y = 0$

error, $e = t - y = 0$

So, no update is required for weight vector & θ.

2nd iteration, $a = [1\ 1]\begin{bmatrix}1\\-0\end{bmatrix} - 1.1 = -0.1$

$\Downarrow$

$y = 0$

error, $e = t - y = 0$

3rd iteration,

$a = [1\ 1]\begin{bmatrix}1\\-1\end{bmatrix} - 1.1 = -2.1$

$\Downarrow$

$y = 0$

$e = 0$

4th iteration

$a = [1\ 1]\begin{bmatrix}2\\1\end{bmatrix} - 1.1 = 1.9 \Rightarrow y =$

So, w gets converged.

In case of non-linearly separable function outputs, weight vector w will not get converged.