FADML: Scribe

Weekly Summary - 8

10th March, 2025 to 14th March 2025

Prashant Bisht (24BM6JP42)

Bayesian Learning

Bayesian Learning is a fundamental approach in probabilistic modelling that allows us to determine the probability of an outcome given a set of observed variables. This method is widely used in machine learning, decision theory, and artificial intelligence due to its ability to incorporate prior knowledge and update beliefs as new data become available. The core objective of Bayesian Learning is to compute the posterior probability:

$$P(Y|X_1, X_2, \dots, X_n)$$

where Y is the target variable, and X_1, X_2 .... Xn are the input features.

Why Bayesian Learning?

- Captures dependencies between features and outcomes.
- Challenges include **data sparsity** and **exponential growth of parameters**, which make direct probability estimation infeasible.

Issues with the Direct Estimation

• Data Sparsity: Many feature combinations may not appear in the training data, making the probability estimation unreliable.

• Exponential Growth of Parameters: Without simplifying assumptions, estimating 2n − 1 probabilities become infeasible for large n. For example, with 10 binary features, we need to have 1023 probability values. To overcome these challenges, we turn to Bayesian parameter estimation techniques that use prior knowledge to make probability estimation feasible.

Bayesian Inference and Conditional Independence

Bayes' Theorem: Now, let us see if we can reduce the values needed for an n-binary feature dataset using Bayes' rule.

$$P(Y \mid X1, \dots, Xn) = \frac{P(Y)P(X_1, \dots, X_n|Y)}{P(X_1, \dots, X_n)}$$

$$P(Y \mid X1, \dots, Xn) = \frac{P(Y)P(X_1, \dots, X_n \mid Y)}{P(Y=1)P(X_1, \dots, X_n \mid Y=1) + P(Y=0)P(X_1, \dots, X_n \mid Y=0)}$$

So, now we need $2^n - 1$ estimations for each for $P(Y = 1)$, $P(X_1, \ldots, X_n | Y = 1)$, and $P(Y = 0)$, $P(X_1, \ldots, X_n | Y = 0)$. Then, two more estimations are combined, leading to a total estimation of $2(2^n - 1) + 2$ estimations.

Interestingly, applying Bayes' rule initially increases the number of required estimations rather than reducing them.

To simplify computations, we assume conditional independence between features given Y:

$$P(X_1, \ldots, X_n \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y)$$

This assumption forms the basis of the Naive Bayes classifier, reducing the number of required estimates from $2(2^n - 1) + 2$ to $2n + 2$.

Conditional Independence Breakdown

$$P(X1 \mid X2, Y) = P(X1 \mid Y)$$

This implies that X_1 and X_2 are conditionally independent given Y, which can be represented as:

$$X1 \perp X2 \mid Y$$

We factorize the probabilities as below:

$$P(X1, X2 \mid Y) = P(X1 \mid Y)P(X2 \mid Y)$$

Using the definition of conditional probability,

$$P(X1, X2, Y) = P(X1|Y)P(X2|Y)P(Y)$$

Dividing both sides by $P(Y)$:

$$P(X_1, X_2) = \frac{P(X_1 \mid Y)P(X_2 \mid Y)P(Y)}{P(Y)}$$

Since $P(Y)$ cancels out in numerator and denominator:

$$P(X1, X2) = P(X1|Y)P(X2|Y)$$

Naive Bayes Classification - Handling Data Sparsity

Let us better understand this concept with an example.

Consider a classification problem where we want to predict whether a student will pass or fail based on the following 3 attributes - Attendance (Poor(P)/Average(A)/High(H)), Reads (Y/N), and Assignment Solving (Low(L)/Medium(M)/High(H)).

$$P(+| A, N, M) = \frac{P(+)}{P(M \mid +)} \times P(A \mid +) \times P(N \mid +)$$

Expanding the denominator:

$$P(+)\big(P(A \mid +)P(N \mid +)P(M \mid +)\big) + P(-)P(A \mid -)P(N \mid -)P(M \mid -)$$

Decision Boundaries and Log-Linear Models

- **Decision boundaries** separate different classification regions.
- Taking the **logarithm** of Naive Bayes probabilities transforms them into a **log-linear model**:

Decision Boundary in Naive Bayes: To classify an observation, we compare the posterior probabilities.

$$P(Y \mid X_1, X_2, \ldots, X_n) \geq P(\bar{Y} \mid X_1, \ldots, X_n)$$

Taking the natural logarithm:

$$\ln\left(\frac{P(Y)\prod_{i=1}^{n} P(X_i \mid Y)}{P(\bar{Y})\prod_{i=1}^{n} P(X_i \mid \bar{Y})}\right) \geq 0$$

Rewriting:

$$\ln P(Y) - \ln P(\bar{Y}) + \sum_{i=1}^{n} \ln\left(\frac{P(X_i \mid Y)}{P(X_i \mid \bar{Y})}\right) \geq 0$$

Since ln P(Y) − ln P($\bar{Y}$) is a constant, we define:

$$C = \ln P(Y) - \ln P(\bar{Y})$$

Thus, the decision boundary is given by:

$$C + \sum_{i=1}^{n} \ln\left(\frac{P(X_i \mid Y)}{P(X_i \mid \bar{Y})}\right) \geq 0$$

Log-Linear Model Interpretation

This shows that the Naive Bayes classifier produces a log-linear model, meaning:

- Probabilities are transformed into log-space to form a linear decision boundary.
- When mapped back to the original probability space, the boundary can become non-linear.

Real-World Applications

- **Spam Filtering**: Classifying emails as spam or not.
- **Medical Diagnosis**: Identifying diseases like cancer using MRI scans.

Gaussian Naive Bayes for Continuous Features

• Up to this point, we have primarily dealt with discrete-valued features. However, many real-world applications involve features that vary continuously over a range.

• To extend Naive Bayes classification to such cases, we assume that each feature follows a specific probability distribution.

• The most common type of distribution is Gaussian (Normal) Distribution:

$$P(X_i \mid Y = k) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(X-\mu)^2}{2\sigma^2}}$$

where:

– μ is the mean of the feature for class k

– $\sigma^2$is the variance of the feature for class k

• For predicting for a new data point, we just substitute the values of X for each of the features and find the class which has highest probability

Example: Basketball Player Classification

To demonstrate Gaussian Naive Bayes classification, consider a dataset where students are described using two continuous features:

1. Height (H)

2. Marks (M)

• We want to classify whether a student is a basketball player (BB) or not

Using Bayes' Theorem:

$$P(BB \mid H, M) = \frac{P(H \mid BB)P(M \mid BB)P(BB)}{P(H, M)}$$

Expanding the denominator:

$$P(H, M) = P(H, M \mid BB)P(BB) + P(H, M \mid \overline{BB})P(\overline{BB})$$

$$P(BB \mid H, M) = \frac{P(H \mid BB)P(M \mid BB)P(BB)}{P(H, M \mid BB)P(BB) + P(H, M \mid \overline{BB})P(\overline{BB})}$$

Alternatively, we can use the ratio:

$$\frac{P(BB)P(H|BB)P(M|BB)}{P(\overline{BB})P(H|\overline{BB})P(M|\overline{BB})}$$

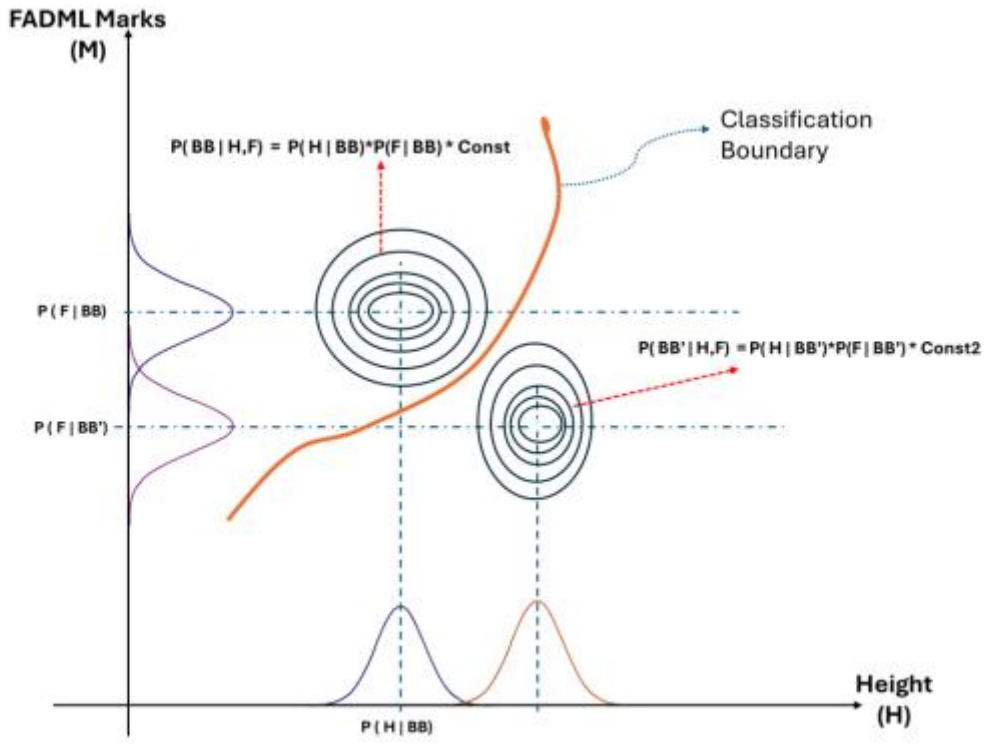
If the ratio is greater than 1, the student is classified as a basketball player (BB).

Gaussian distributions exist for Marks, representing probabilities conditioned on whether a student is a basketball player or not. – Similarly, two Gaussian distributions exist for Height, representing conditional probabilities given the class.

The combination of Height and Marks distributions results in a region of concentric circles in the graph. – These circles represent contours of equal probability for classifying a student as BB or BB.

The classification boundary is depicted in orange. – This boundary separates basketball players from non-basketball players based on height and marks.

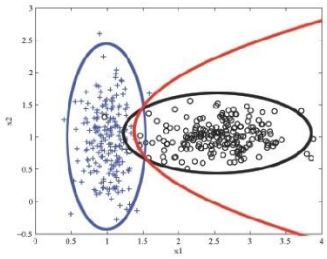
Depending on the mean and variance of Height and Marks distributions, the concentric circles can deform into ellipses. – This is because Gaussian distributions in two dimensions often lead to elliptical decision boundaries rather than perfect circles.



Gaussian Distributions and Decision Boundary for Basketball Player Classification

Otherwise, the student is classified as not a basketball player BB.

If the σ's are same, then the boundary will be linear. Otherwise, if the σ's are not same we will have non-linear boundary.



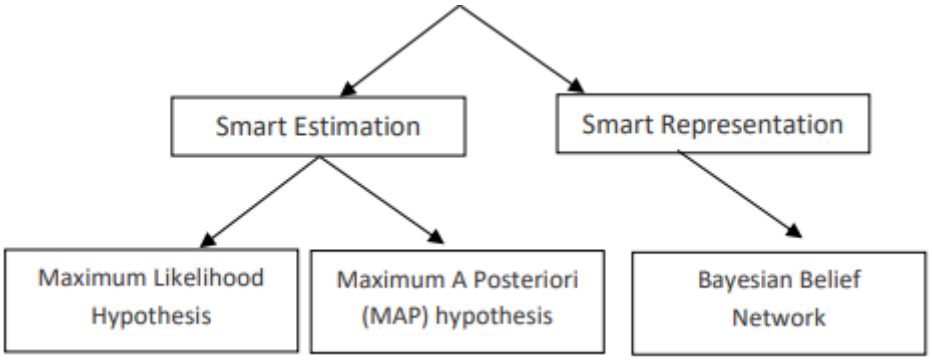
Key Takeaways

- Naive Bayes is computationally efficient and reduces complexity.
- Decision boundaries are log-linear, making classification straightforward.
- Gaussian Naive Bayes extends the model to continuous data for better real-world applications.
- Widely used in spam detection, medical diagnostics, and text classification.

Question:

How does the Naive Bayes model help in text classification problems like spam detection?

Data sparsity solution



BAYESIAN BELIEF NETWORK:

Smart representation of joint probability distribution is called Bayesian Network in simple terms.

A graph with nodes as attributes and edges as dependencies is called Bayesian Network. It is DAG (Directed Acyclic Graph)

The naive Bayes classifier makes significant use of the assumption that the values of the attributes ($a_1 \ldots a_n$) are conditionally independent given the target value y. This assumption dramatically reduces the complexity of learning the target function. When it is met, the naive Bayes classifier outputs the optimal Bayes classification. However, in many cases this conditional independence assumption is clearly overly restrictive. In contrast to the naive Bayes classifier, which assumes that all the variables are conditionally independent given the value of the target variable, Bayesian belief networks allow conditional independence assumptions that apply to subsets of the variables. Thus, Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier, but more tractable than avoiding conditional independence assumptions altogether. In general, a Bayesian belief network describes the probability distribution over a set of variables.

Concept of Conditional Independence

$$[\, X \perp Y \mid Z \qquad \forall i,j,k \text{ "}\{ (X \text{ is independent of } Y, \text{given } Z)\}\,]$$

$$[\, P(X = x\_i \mid Y = y\_j, Z = z\_k) = P(X = x\_i \mid Z = z\_k)\,]$$

Alternatively,

$$P(X = x, y = y_i | Z = z_k) = P(X = x_i | Z = z_k) \cdot P(y = y_i | Z = z_k)$$

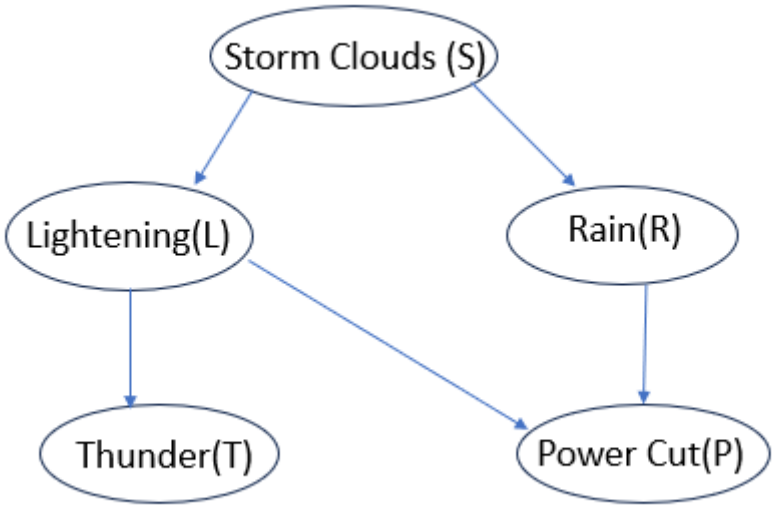
Marginal Independence:

$$X \perp Y \quad \forall (i,j)$$

$$P\big(X = x_i, y = y_j\big) = P(X = x_i) \cdot P\big(y = y_j\big)$$

$$\equiv P\big(X = x_i \big| y = y_j\big) = P(X = x_i)$$

Example: A **Bayesian Network** with variables:



Each node will have its local table of joint probability distribution rather than the global table.

For the node P, L and R are its parent.

Parent		Power cut(P)	
		P=1	P=0
L(Lightening)	R(Rain)	Probabilities	Probabilities
0	0	0.01	0.99
0	1	..	..
1	0	..	..
1	1	..	..

Local Joint Probability table for Power Cut depending upon parents L and R

But using the chain rule of probability we can estimate the overall probability of any entry of Joint Distribution. Only dependent entries are present, so Bayesian Network cuts the no. o entries per node.

So, we have a joint probability distribution table (JPDT) for each of the attribute. JPDT (Xi, Parent (Xi))

Question: Why fragmented probability distribution tables for each attribute rather than complete JPDT?

Answer: Because now we have a smarter casual dependency, we only need that information to compute probability.

Probability of (S, L, ~R, P, ~T) = ?

If given a JPDT, just look up, but not given.

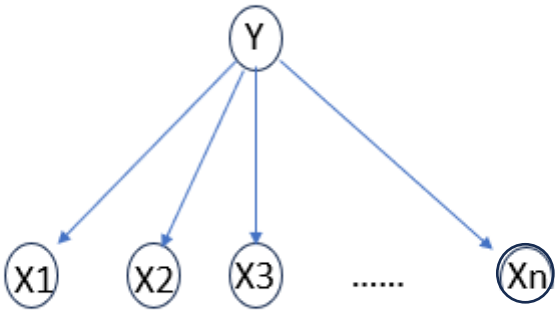
P(S, L, ~R, P, ~T) = P(S) * P(L | S) * P(~R | S) * P(P | L, ~R) * P(~T | L)

Each of them is a direct look-up into the individual attribute Joint probability table.

Bayes Network for Naïve Bayes:

$$P(X_1, \ldots, X_n \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y)$$

Naïve Bayes has a restricted view of the Bayesian Network.



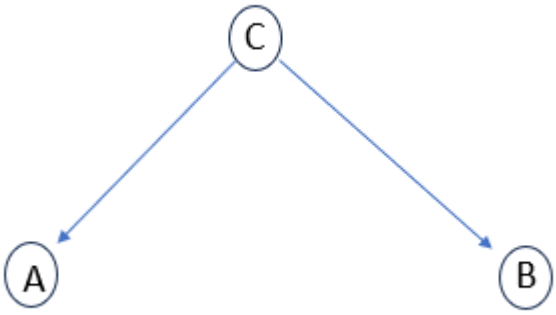
We would be better of with having partial conditional dependence than total independence (like in Naïve Bayes)

Demerit: Store Graph and Traverse.

D-separation Algorithm

Let's take an example: Consider A = Shoe Size, B = Reading Ability, C = Age

Case 1: Given Child Age (C), A and B are independent. [Tail to Tail]



Fork

$$P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z) \text{ , if } \boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z}$$

$$P(A\,B|C) = \frac{P(A\,B\,C)}{P(C)}$$

$$P(A\,B|C) = \frac{P(A|C) \cdot P(B|C) \cdot P(C)}{P(C)}$$

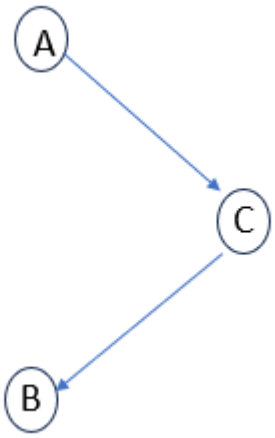
$$P(A\,B|C) = P(A|C) \cdot P(B|C)$$

Hence, A is conditionally independent of B given C

$$\boldsymbol{A} \perp \boldsymbol{B} \mid \text{C and } \boldsymbol{A} \not\perp \boldsymbol{B} \mid \Phi$$

Let's take another example: Consider A = Fire, B = Alarm, C = Smoke

Case 2: Given C, A and B are independent. [Head to Tail]



Chain

$$P(A\,B|C) = \frac{P(A\,B\,C)}{P(C)}$$

$$P(A\,B|C) = \frac{P(A|C) \cdot P(B|C) \cdot P(C)}{P(C)}$$

$$P(A\,B|C) = P(A|C) \cdot P(B|C)$$

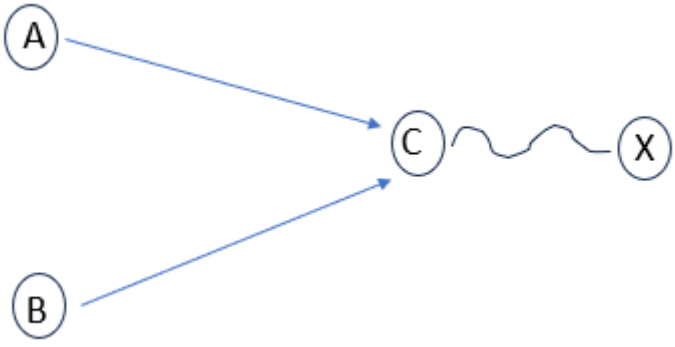
Hence, A is conditionally independent of B given C

$$\boldsymbol{A} \perp \boldsymbol{B} \mid \mathbf{C}$$

$$\boldsymbol{A} \not\perp \boldsymbol{B} \mid \Phi$$

Let's take another example: Consider A = Talent, B = Beauty, C = Celebrity

Case 3: Given C, A and B are not independent. [Head to Head]



Collide

$$P(A\,B|C) = \frac{P(A\,B\,C)}{P(C)}$$

$$P(A\,B|C) = \frac{P(C|A) \cdot P(C|B) \cdot P(A) \cdot P(B)}{P(C)}$$

$$P(A\,B|C) \neq P(A|C) \cdot P(B|C)$$

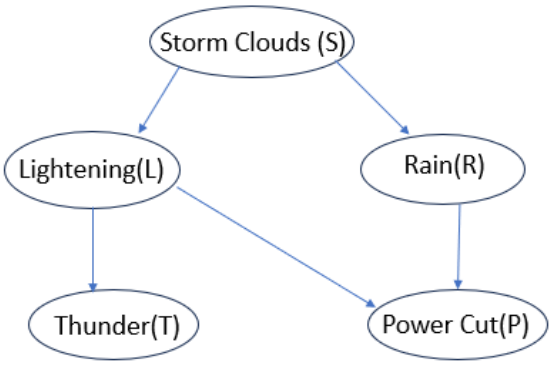
Hence, A is not conditionally independent of B given C if both are the causes of C.

$$\boldsymbol{A} \not\perp \boldsymbol{B} \mid \mathbf{C}$$

$$\boldsymbol{A} \perp \boldsymbol{B} \mid \Phi$$

Question for practice:

Given the below graph answer the following questions based on the D-separation algorithm



Is $L \perp R \mid S$? : Yes

Is $L \perp R \mid P$? : No

Key Concept Summary of D-separation Algorithm:

Two variables X and Y are d-separated (conditionally independent) given a set of variables Z if every path between X and Y is blocked by Z in the Bayesian network.

A path is blocked if at least one of the following holds:

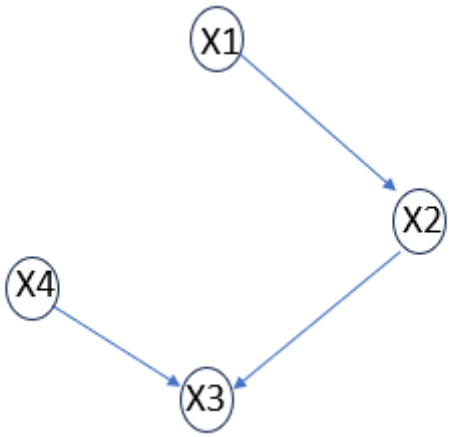
1. Chain Structure: X→M→Y
    - Blocked if M is in Z.
2. Fork Structure: X←M→Y
    - Blocked if M is in Z.
3. Collider Structure (V-Structure): X→M←Y
    - Blocked if M is not in Z and no descendant of M is in Z.

d-Separation Algorithm

Given a Bayesian network and variables X, Y, and Z, follow these steps:

1. Identify all paths between X and Y in the DAG.
2. Check for blocking conditions:
    - If any path is blocked due to a chain or fork node being in Z, the path is blocked.
    - If a path has a collider node that is not in Z (and none of its descendants are in Z), the path is blocked.
3. If all paths are blocked, then X and Y are conditionally independent given Z (i.e., X⊥Y|Z).
4. If at least one path is open, then X and Y are conditionally dependent given Z.

Question for practice:



- $X1 \perp X3 \mid X2 = ?$
- $X1 \perp X4 \mid X2 = ?$
- $X1 \perp X4 \mid \{X2, X3\} = ?$
- $X1 \perp X4 \mid \Phi = ?$

Bayesian Network Known JPDT Full Observability	Bayesian Network Known CPDT Partial Observability
Bayesian Network Unknown CPDT Full Observability	Bayesian Network Unknown CPDT: Partial Unsolved

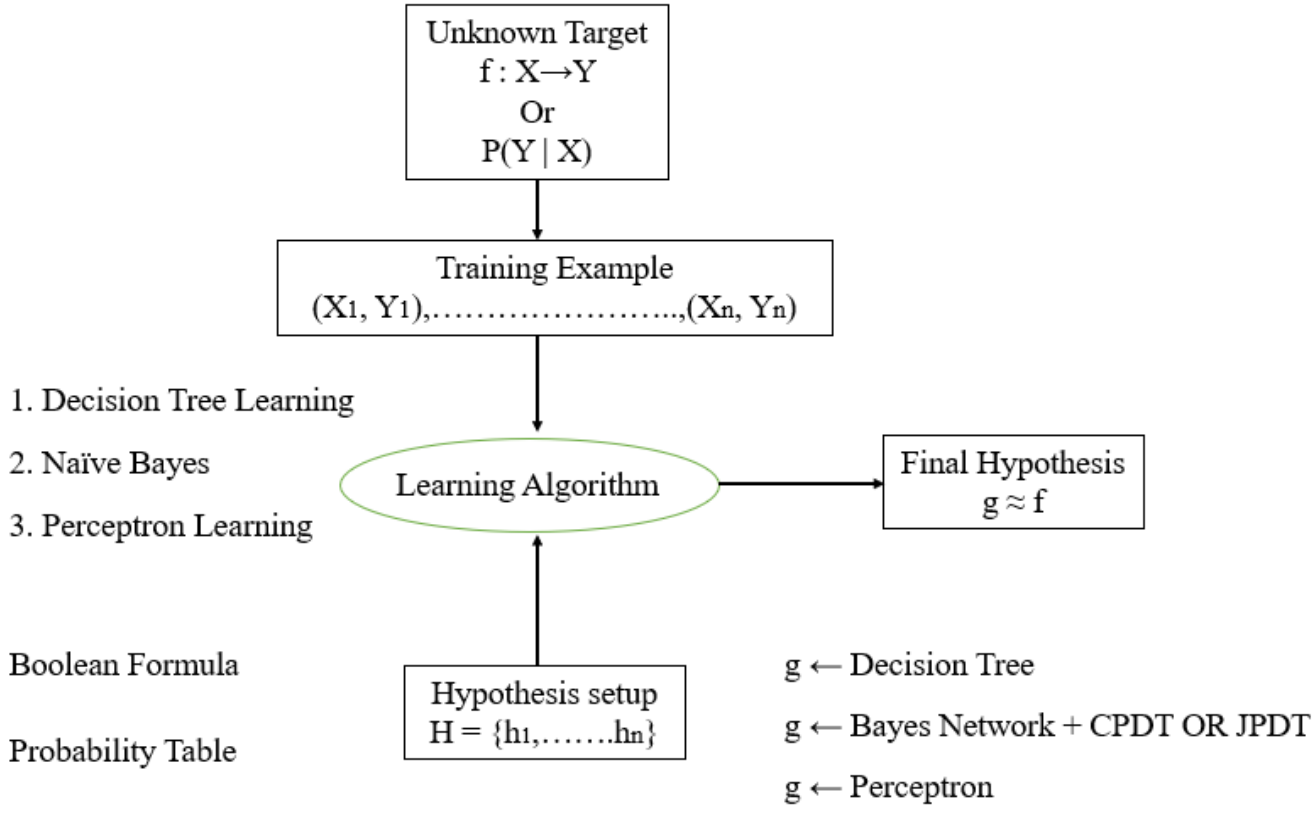
The **Expectation-Maximization (EM) algorithm** is an **iterative method** used to estimate unknown parameters in statistical models. It helps find the best values for unknown parameters, especially when some data is missing or hidden.

It works in two steps:

- **E-step (Expectation Step):** Estimates missing or hidden values using current parameter estimates.
- **M-step (Maximization Step):** Updates model parameters to maximize the likelihood based on the estimated values from the E-step.

By iteratively repeating these steps, the EM algorithm seeks to maximize the likelihood of the observed data. It is commonly used for clustering, where latent variables are inferred and has applications in various fields, including machine learning, computer vision, and natural language processing.

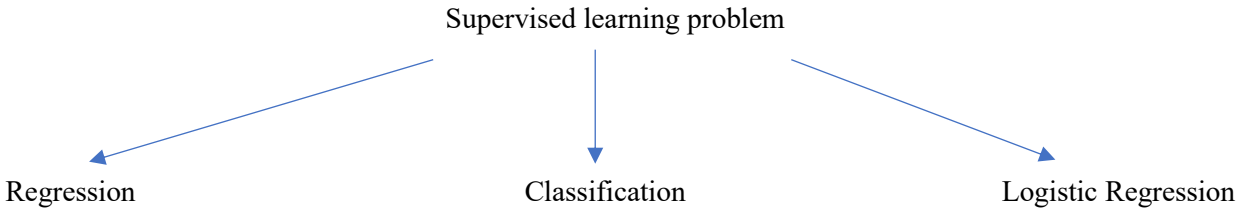
Supervised learning Framework



Objectives

- Algorithm: $\rightarrow E_{in}(g) \approx 0$
- Theory: (Hoeffding's Inequality)

$$P(|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon) \leq 2M \exp(-2\epsilon^2 N)$$

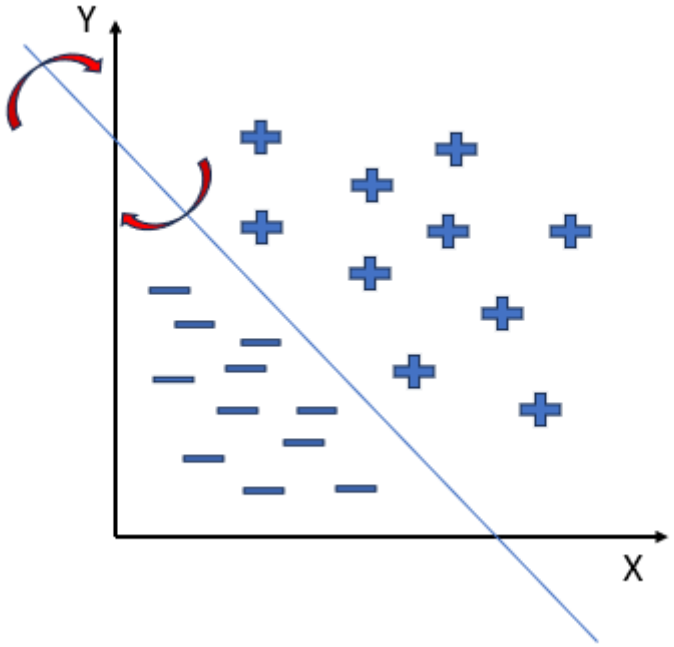


If we want probabilities as outcomes, we use Naïve Bayes and Gaussian Naïve Bayes etc.

If we reduce the in-sample error, we can track the out sample well. This ensures that what we have learned in sample is a good hypothesis.

Perceptron Learning Algorithm

The **Perceptron Learning Algorithm (PLA)** is a fundamental algorithm in machine learning used for binary classification. It is an iterative method that adjusts the weights of a linear classifier to correctly classify training examples.



The **decision boundary** of a perceptron is a straight line (or hyperplane in higher dimensions) that separates the two classes. When a **misclassified point** is encountered, the perceptron updates its weight vector **to move the decision boundary in the correct direction**.

$$w_1 x_1 + w_2 x_2 \geq \text{Threshold}$$

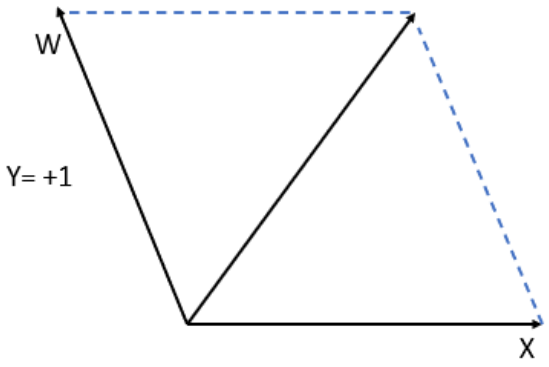
$$\Rightarrow \quad w_1 x_1 + w_2 x_2 + w_0 \cdot 1 \geq 0$$

$$\Rightarrow \quad \sum_{i=0}^{d} w_i x_i \geq 0 \quad \text{where} \quad x_0 = 1$$

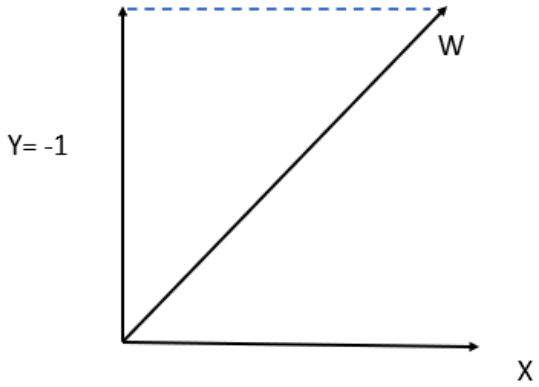
$$\text{sign}\left(\sum_{i=0}^{d} w_i x_i\right) = \text{sign}(w^T x)$$

where, w and X are vectors of dimensions *(d+1) X 1*.

Delta Update Rule (Perceptron Learning Rule)



$W' \leftarrow W + X$



$W' \leftarrow W - X$

The **Delta Update Rule**, also known as the **Perceptron Learning Rule**, is a fundamental weight update mechanism used in perceptron learning and gradient-based optimization methods.

Explanation

- If a training example x_i is **correctly classified**, no update is made.
- If x_i is **misclassified**, the weight vector is adjusted in the direction of x_i to reduce classification error.
- The learning rate η controls how aggressively weights are updated.

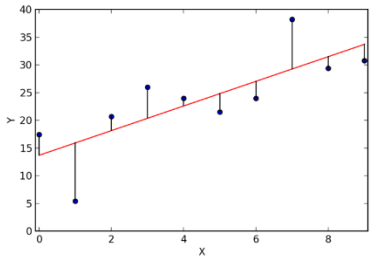
Pocket Learning Algorithm

The **Pocket Algorithm** is an improved version of the **Perceptron Learning Algorithm (PLA)** designed to handle cases where the data is **not linearly separable**. Unlike the standard perceptron, which continues updating weights indefinitely for non-separable data, the **Pocket Algorithm maintains the best weight vector found so far**.

Why Pocket Algorithm?

- The **Perceptron Algorithm** only works if the data is **linearly separable**.
- If data is **not linearly separable**, the perceptron **never stops updating** and does not converge to a stable solution.
- The **Pocket Algorithm stores (or "pockets") the best weight vector** encountered so far, ensuring a more stable and reliable solution.

Regression Problem



$$E_{\text{in}}(w) = \frac{1}{N}\sum_{i=1}^{N}(w^T x_i - y_i)^2$$

$$= \frac{1}{N}||Xw - Y||^2$$

Derivative and Solution:

$$\frac{dE_{\text{in}}}{dw} = 0$$

$$\Rightarrow \frac{2}{N}X^T(Xw - Y) = 0$$

$$\Rightarrow X^T Xw = X^T Y$$

$$\Rightarrow w = (X^T X)^{-1} X^T Y$$

Pseudo-Inverse Solution:

Since sometimes $X^T X$ is not invertible, we use the pseudo-inverse method:

$$w = X^+ Y$$

where, $X^+ = (X^T X)^{-1} X^T$

Dimensions:

- X is of shape $\big(N \times (d+1)\big)$
- $X^T X$ is of shape $\big((d+1) \times (d+1)\big)$
- $X^T Y$ is of shape $\big((d+1) \times 1\big)$
- w is of shape $\big((d+1) \times 1\big)$

Notes & Observations:

- **Issue:** If there is too much data and too many rows, inverse calculation becomes **computationally expensive**.
- **Alternative Approach:** Instead of direct inversion, we use **generalized inverse** or **pseudo-inverse** methods.

Advantages & Disadvantages of Closed-Form Solution

Demerit: It may tend to overfit **one side** and may not always provide the best-fit line.

Merit: It provides **initialization** for the **regression line**, which can be useful for iterative optimization methods.

Gradient Descent Methods for Linear Regression

Gradient Descent is an **iterative optimization algorithm** used to minimize the error function in machine learning models, especially when a closed-form solution is computationally expensive or infeasible. Below are the key types of Gradient Descent methods used in Linear Regression.

Batch Gradient Descent

Batch Gradient Descent updates the weights after computing the gradient using the entire dataset.

Update Rule:

$$w \coloneqq w - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla E_{\text{in}}(w)$$

Where:

- w *is the weight vector,*
- η *is the learning rate,*
- N *is the total number of samples,*
- $\nabla E_{\text{in}}(w) is the gradient of the error function.$

Stochastic Gradient Descent (SGD)

Instead of computing the gradient over the entire dataset, Stochastic Gradient Descent updates the weights using a **single random sample** at each iteration.

Update Rule:

$$w \coloneqq w - \eta \nabla E_{\text{in}}\left(w^{(i)}\right)$$

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent is a compromise between Batch and Stochastic Gradient Descent. It updates the weights using a **small subset (batch) of data** at each iteration.

Update Rule:

$$w \coloneqq w - \eta \frac{1}{B} \sum_{j=1}^{B} \nabla E_{\text{in}}\left(w^{(j)}\right)$$

where:

- B is the batch size.