Machine Learning

by ambedkar@IISc

- Probabilistic view of linear regression
- Logistic regression
- Hyperplane based classifiers and perceptron

Topics

Probabilistic View of Linear Regression

Logistic Regression

Hyperplane based classifiers and Perceptron

Probabilistic View of Linear Regression

Maximum Likelihood Estimation

- Let $X = x_1, x_2, \ldots, x_N$, where $x_n \in \mathbb{R}^d$ be some data that is generated from $x_n \sim P(x|\theta)$
  - **Recall**: In the statistical approach to machine learning, we assume that there is an underlying probability distribution from which the data is sampled.
  - Hence θ denotes the parameters of the distribution.
  - For example $x_n \sim \mathcal{N}(x|\mu, \sigma)$. That is $\theta = (\mu, \sigma)$.
- **Assumption**: The data in X is generated i.i.d. (independent and identically distributed). This is very important assumption and we see this very often.
- **Aim**: Learn θ given the data $X = x_1, x_2, \ldots, x_N$.

Diversion: Some Probability

- We say two random variables X, Y are identical that means that their probability distributions are the same
  - If two Gaussian random variables are same only if their means and variance (covariance matrices) are same.
- We say two random variables X, Y are independent if

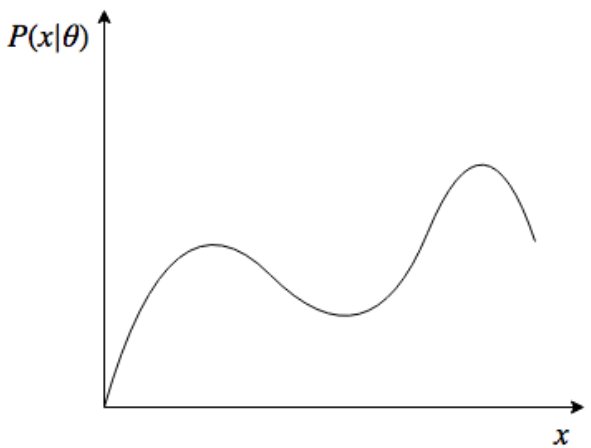
$$P(X, Y) = P(X)P(Y)$$

Maximum Likelihood Estimation (contd. . .)

- Given $X = x_1, x_2, \ldots, x_N$, and $x_n \sim P(x|\theta)$
  - Learn P so that likelihood of $x_1, x_2, \ldots, x_N$ are sampled from P is maximum.
  - Equivalently learn or estimate θ so that likelihood of $x_1, x_2, \ldots, x_N$ are sampled from P is maximum.
- By the iid assumption

$$P(X|\theta) = P(x_1, x_2, \ldots, x_N|\theta)$$
$$= \prod_{n=1}^{N} P(x_n|\theta)$$

- $P(X|\theta)$ is the likelihood.



Maximum Likelihood Estimation (contd. . .)

How do we estimate θ given the data X.

$$\Downarrow$$

Find value of θ that makes observed data most probable.

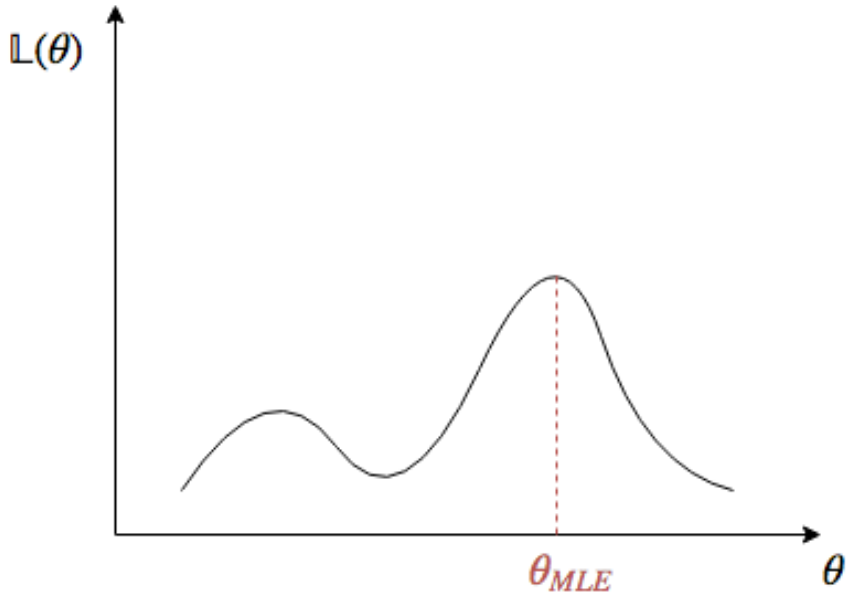
$$\Downarrow$$

Find θ that maximizes likelihood function

$$\mathcal{L} = \log P(X|\theta) = \sum_{n=1}^{N} \log P(x_n|\theta)$$

Maximum Likelihood Estimation (contd...)

$$\theta^*_{\text{MLE}} = \arg\max_{\theta} \mathcal{L}(\theta) = \arg\max_{\theta} \sum_{n=1}^{N} \log P(x_n|\theta)$$



Maximum Likelihood Estimation (contd. . .)

Example:

Suppose X_n is a binary random variable. Suppose it follows Bernoulli distribution

i.e. $P(x|\theta) = \theta^x(1-\theta)^{1-x}$

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} \log P(x_n|\theta) = \sum_{n=1}^{N} x_n \log\theta + (1-x_n)\log(1-\theta)$$

$$\frac{\partial\mathcal{L}(\theta)}{\partial\theta} = \frac{1}{\theta}\sum_{n=1}^{N} x_n + \frac{1}{1-\theta}\sum_{n=1}^{N}(1-x_n)$$

$$= \frac{1}{\theta}\sum_{n=1}^{N} x_n + \frac{1}{1-\theta}(N - \sum_{n=1}^{N} x_n)$$

Maximum Likelihood Estimation (contd. . .)

$$\implies \theta^*_{MLE} = \frac{\sum_{n=1}^{N} x_n}{N}$$

[In a coin tossing experiment, it is just a fraction of heads]

Maximum a Posteriori Estimate

- We will have a prior on parameter θ i.e. $P(\theta)$
  Yes θ is no more a mere number, it is a Random Variable.
  - One can have knowledge on θ
  - It acts as a regularizer (We will see later)
- **Bayes Rule:**

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

$$P(\theta|X) : Posterior$$

$$P(X|\theta) : Likelihood$$

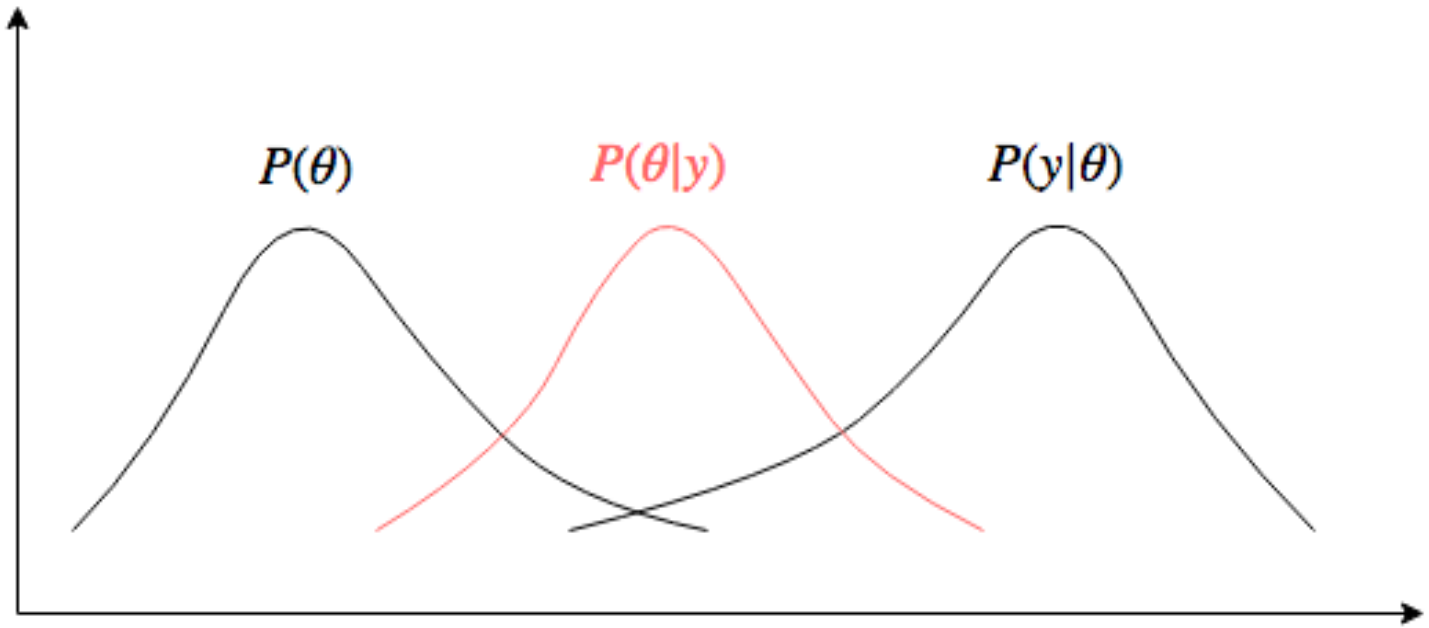
$$P(\theta) : Prior$$

$$P(X) : Evidence$$

Maximum a Posteriori Estimate (contd. . .)

Bayes Rule:

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$



Maximum a Posteriori Estimate (contd. . .)

MAP Estimate

$$\begin{aligned}\theta^*_{MAP} &= \arg\max_{\theta} P(\theta|x) \\ &= \arg\max_{\theta} \log P(x|\theta) + \log P(\theta) \\ &= \arg\max_{\theta} \sum_{n=1}^{N} \log P(x_n|\theta) + \log P(\theta)\end{aligned}$$

Note: When $P(\theta)$ is a uniform distribution, it reduces to MLE.

Linear Regression : Probabilistic Setting

- Each response is generated by a linear model + Gaussian noise

$$Y = W^T X + E$$

- That is, given N training samples $\{(x_n, y_n)_{n=1}^N\}$ i.i.d. $x_n \in \mathbb{R}^d$ and $y_n \in \mathbb{R}$
  - $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$
  - $y_n \sim \mathcal{N}(w^T x_n, \sigma^2)$

$$\implies P(Y|X, W) = \mathcal{N}(y|w^T x, \sigma^2)$$
$$= \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(-\frac{(y - w^T x)^2}{2\sigma^2}\Big)$$

Log Likelihood

$$\begin{aligned}
\log \mathcal{L}(w) &= \log P(\mathcal{D}|w) = \log P(y|X, W) \\
&= \log \prod_{n=1}^{N} P(y_n|x_n, w) \\
&= \sum_{n=1}^{N} \log P(y_n|x_n, w) \\
&= \sum_{n=1}^{N} \left[-\frac{1}{2} \log(2\pi\sigma^2) - \frac{(y_n - w^T x_n)^2}{2\sigma^2} \right]
\end{aligned}$$

Linear Regression : ML Estimation (contd. . .)

$$W^*_{MLE} = \arg\max_w -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - w^T x_n)^2$$

$$= \arg\min_w \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - w^T x_n)^2$$

i.e. ML Estimation in the case of Gaussian environment $\equiv$ Least square objective for regression

Linear Regression : MAP Estimate

- Here we introduce prior on the parameter w.
  $\Rightarrow$ This will lead to regularization of model.
  - Remember we treat parameters as Random Variables in MAP.
- $\mathrm{P(w)} = \mathcal{N}(\mathrm{w}|\underbrace{0}_{\text{Mean}}, \underbrace{\lambda^{-1} I}_{\text{Variance}})$
- We have multivariate Gaussian

$$\mathcal{N}(x : \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\Big(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(X-\mu)\Big)$$

$$= \frac{1}{\sqrt{(2\pi)^{\frac{D}{2}} (\frac{1}{\lambda})^{\frac{D}{2}}}} \exp\Big(-\frac{\lambda}{2} w^T w\Big)$$

Linear Regression : MAP Estimate (contd. . .)

- log posterior probability

$$\begin{aligned}\log(w|\mathcal{D}) &= \log \frac{P(\mathcal{D}|w)P(w)}{P(\mathcal{D})} \\ &= \log P(w) + \log P(w|\mathcal{D}) - \log P(\mathcal{D})\end{aligned}$$

-

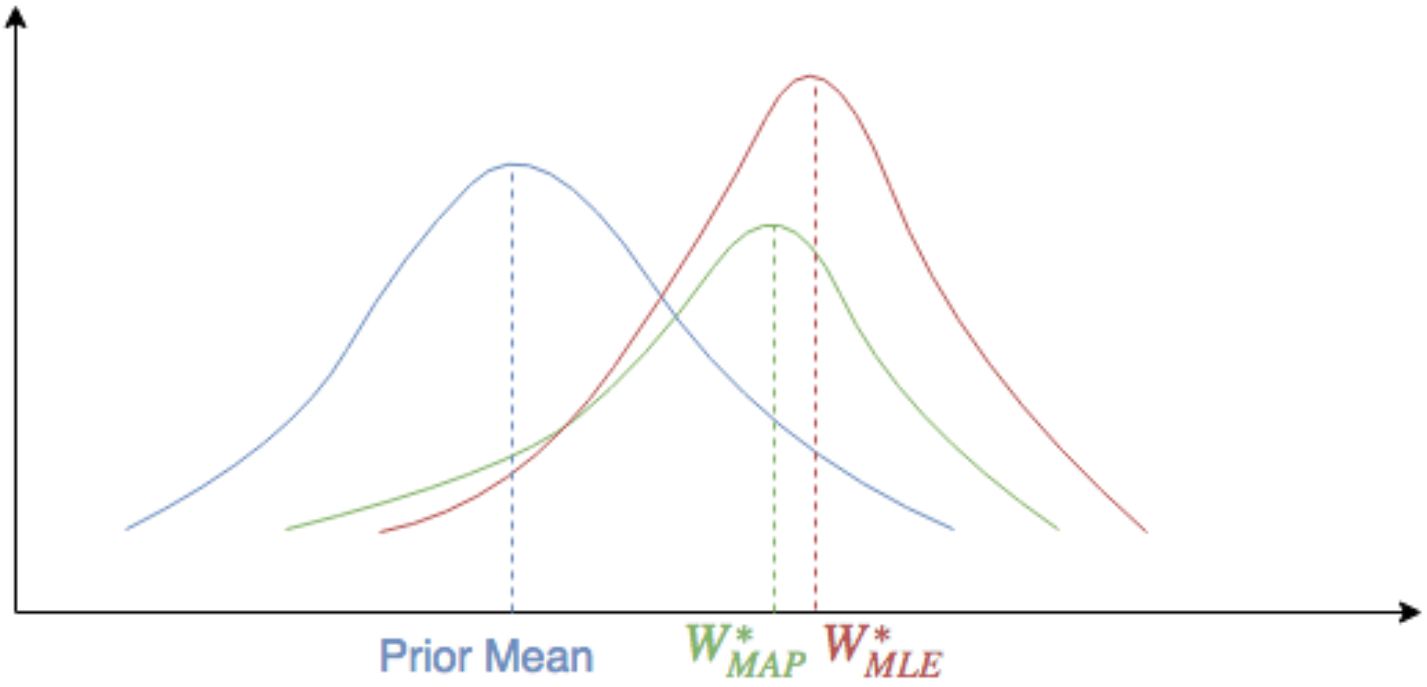
$$\begin{aligned}W^*_{MAP} &= \arg\max_w \log P(w|\mathcal{D}) \\ &= \arg\max_w \{\log P(w) + \log P(\mathcal{D}|w) + \log P(\mathcal{D})\} \\ &= \arg\max_w \{\log P(w) + \log P(\mathcal{D}|w)\}\end{aligned}$$

Linear Regression : MAP Estimate (contd. . .)

$$
\begin{aligned}
W^*_{MAP} &= \arg\max_w \log P(w|\mathcal{D}) \\
&= \arg\max_w \Big[-\frac{D}{2}\log 2\pi - \frac{\lambda}{2} w^T w \\
&\qquad + \sum_{n=1}^{N} \Big(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{(x_n - w^T x_n)^2}{2\sigma^2} \Big) \Big] \\
&= \arg\max_w \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - w^T x_n)^2 + \frac{\lambda}{2} w^T w
\end{aligned}
$$

MAP estimate in the case of Gaussian environment $\equiv$ Least square objective with L_2 regularization.

MLE vs MAP



MAP estimate shrinks the estimate of w towards the prior.

Logistic Regression

Problem Set Up

- Two class classification
- Instead of the exact labels estimate the probabilities of the labels.
- i.e predict

$$P(y_n = 1|x_n, w) = \mu_n$$
$$P(y_n = 0|x_n, w) = 1 - \mu_n$$

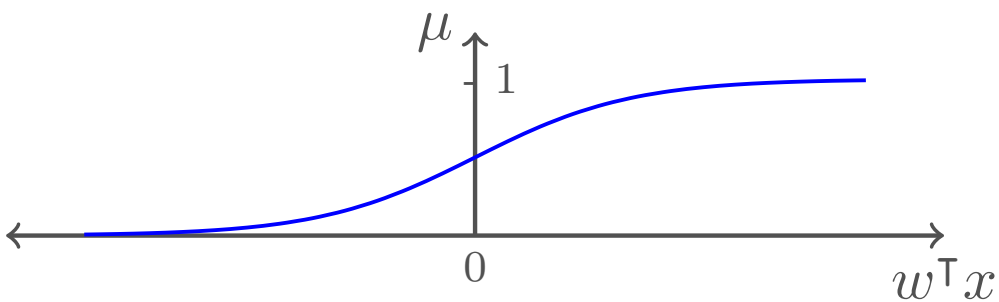
The Logistic Regression Model

$$\mu_n = f(x_n) = \sigma(w^\intercal x_n) = \frac{1}{1+\exp(-w^\intercal x_n)} = \frac{\exp(w^\intercal x_n)}{1+\exp(w^\intercal x_n)}$$

- Here σ is the sigmoid or logistic function.
- The model first computes a real-values score.

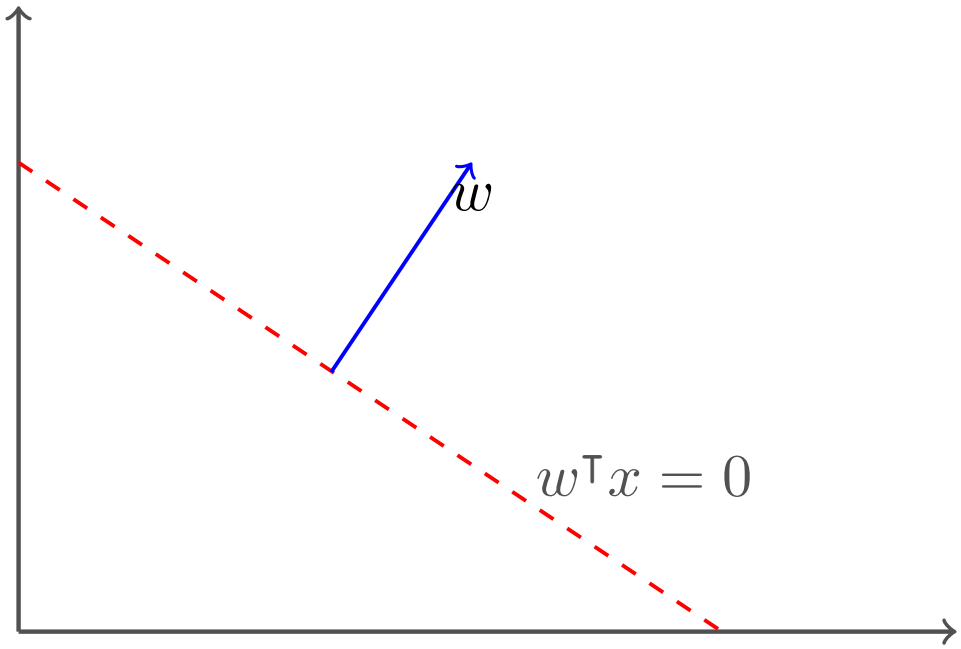
$$w^\intercal x = \sum_{i=1}^{d} w_i x_i$$

and **non-linearly** squashes it between $(0, 1)$ to turn this into a probability.



The Decision Boundary

$$\text{If } w^\intercal x > 0 \implies P(y_n = 1|x_n, w) > P(y_n = 0|x_n, w)$$
$$w^\intercal x < 0 \implies P(y_n = 1|x_n, w) < P(y_n = 0|x_n, w)$$



Logistic Regression: Decision Boundary

Loss Function Optimization

- Squared Loss

$$\begin{aligned}\ell(y_n, f(x_n) &= (y_n - f(x_n))^2 \\ &= (y_n - \sigma(w^\intercal x_n))^2\end{aligned}$$

  - This is non-convex and not easy to optimize.

- Cross Entropy loss

$$\begin{aligned}\ell(y_n, f(x_n)) &= \begin{cases} -\log(\mu_n) & \text{when } y_n = 1 \\ -\log(1-\mu_n) & \text{when } y_n = 0 \end{cases} \\ &= \begin{cases} -P(y_n = 1 | x_n, w_n) & \text{when } y_n = 1 \\ -P(y_n = 0 | x_n, w_n) & \text{when } y_n = 0 \end{cases}\end{aligned}$$

Cross Entropy loss

$$
\begin{aligned}
l(y_n, f(x_n)) &= -y_n \log(\mu_n) - (1 - y_n) \log(1 - \mu_n) \\
&= -y_n \log(\sigma(w^\intercal x_n)) - (1 - y_n) \log(1 - \sigma(w^\intercal x_n))
\end{aligned}
$$

- Cross Entropy Loss over entire data.

$$
\begin{aligned}
L(w) &= \sum_{n=1}^{N} l(y_n, f(x_n)) \\
&= \sum_{n=1}^{N} [-y_n \log(\mu_n) - (1 - y_n) \log(1 - \mu_n)] \\
&= -\sum_{n=1}^{N} [y_n w^\intercal x_n - \log(1 + \exp(w^\intercal x_n)]
\end{aligned}
$$

Cross Entropy loss

- By adding L_2 regularizer.

$$L(w) = -\sum_{n=1}^{N}[y_n w^\intercal x_n - \log(1 + \exp(w^\intercal x_n)) + \lambda||w||^2$$

Logistic Regression: MLE formulation

- **AIM** Learn w from the data that can predict the probability of x_n belong to 0 or 1.
- **Log Likelihood**: Given $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$

$$\begin{aligned}
\log L(w) &= \log P(\mathcal{D}|w) \\
&= \log P(Y|X, w) \\
&= \log \prod_{n=1}^{N} P(y_n|x_n, w) \\
&= \log \prod_{n=1}^{N} \mu_n^{y_n} (1 - \mu_n)^{1-y_n}
\end{aligned}$$

- $\because Y$ is a Bernoulli random variable

$$P(y_n = 1|x_n, w) = \mu_n$$
$$P(y_n = 0|x_n, w) = 1 - \mu_n$$

Logistic Regression: MLE formulation(contd...)

$$P(Y|X,w) = \sum_{n=1}^{N} [y_n \log \mu_n + (1-y_n)\log(1-\mu_n)]$$

$$\text{We have } \mu_n = \frac{\exp(w^\intercal x_n)}{1+\exp(w^\intercal x_n)}$$

$$\implies L(w) = \sum_{n=1}^{N} [y_n w^\intercal x_n - \log(1+\exp(w^\intercal x_n)]$$

Which is same as the cross entropy loss minimization.

Logistic Regression: MAP estimate

- MAP $\rightarrow$ We assume a prior distribution on the weight vector w. That is

$$P(w) = N(w|0), \lambda^{-1}\mathtt{I})$$
$$= \frac{1}{(2\pi)^{\frac{D}{2}}} \exp\left(-\frac{\lambda}{2} w^{\intercal} w \right)$$

- Note: Multivariate Gaussian is defined as

$$P(w) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left[-\frac{1}{2}(X-\mu)^{\intercal}\Sigma^{-1}(X-\mu) \right]$$

- Then MAP estimate is

$$W^*_{MAP} = \arg\max_w \log P(W|\mathcal{D})$$

Logistic Regression: MAP estimate (cont...)

- We have

$$
\begin{aligned}
W^*_{MAP} &= \arg\max_w \log P(W|\theta) \\
&= \arg\max_w \log P(\mathcal{D}|w) + \log P(w) \\
&= \arg\max_w \Bigg[-\frac{D}{2}\log 2\pi - \frac{\lambda}{2} w^\intercal w \\
&\qquad\qquad - \sum_{n=1}^{N} \log(1 + \exp(-y_n w^\intercal x_n)) \Bigg] \\
&= \arg\max_w \sum_{n=1}^{N} \log \Big[1 + \exp(-y_n w^\intercal x_n) \Big] + \frac{\lambda}{2} w^\intercal w
\end{aligned}
$$

Which is same as the minimizing regularized cross entropy loss.

Logistic Regression: Some Comments

- Objective function of Logistic Regression is same as SVMs except for the loss function.

$$\begin{aligned} \text{Logistic Regression} &\rightarrow \text{log loss} \\ \text{SVM} &\rightarrow \text{hinge loss} \end{aligned}$$

- Logistic regression can be extended to multiclass case: just use softmax function.

$$P(Y = k|w, x) = \frac{\exp(w_k^\intercal x)}{\sum_k \exp(w_k^\intercal x)} \quad k = 1, 2, \ldots, K \text{ classes}$$

Optimization is the Key

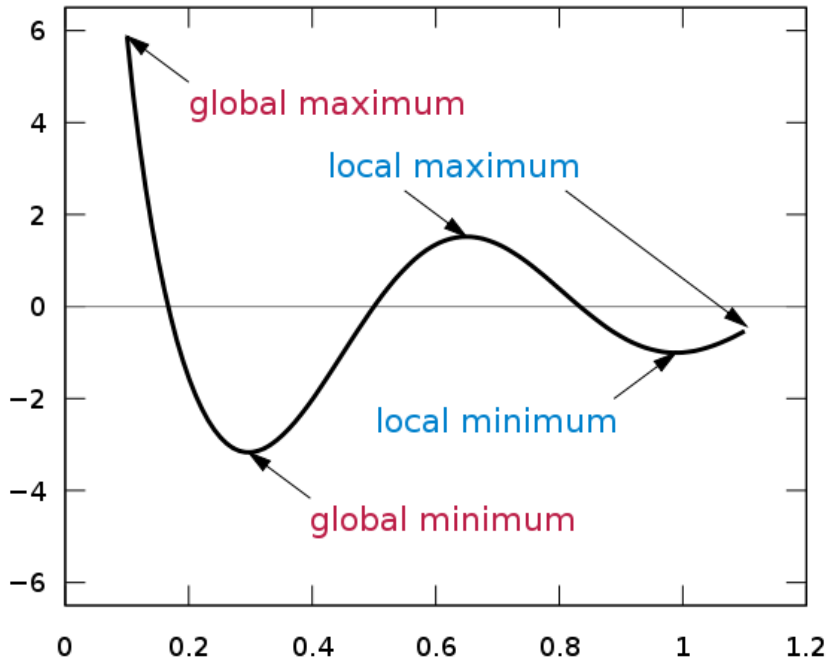
- Almost all problems in machine learning leads to optimization problems
- The following two factors decides the fate of any method:
  - What kind of optimization problem that we are led to
  - What are all optimization methods that are available to us
- There are several methods that are available for optimization, among these gradient descent methods are most popular

Gradient Descent methods are Used in ....

- Linear Regression
- Logistic Regression
  - It is just classification, but instead of labels it gives us class probability
- Support Vector Machines
- Neural Networks
  - The backbone of neural networks is Back-propagation algorithm

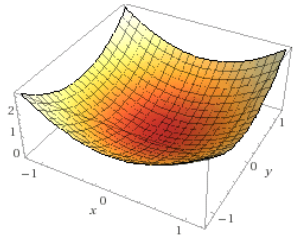
Example of an objective

- Most often, we do not even have functional form of the objective.
  - Given x, we can only compute $f(x)$
  - Sometime this may involve a simulating a system
  - Computing each $f(x)$ can be time consuming

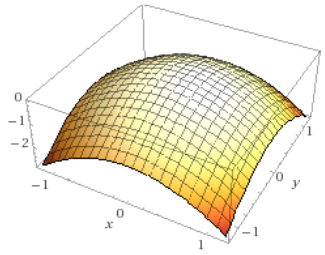


- This becomes even more difficult as x is a D-dimensional vector and D is very large

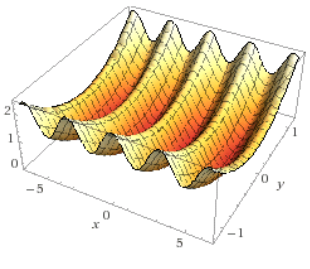
Multivariate Functions



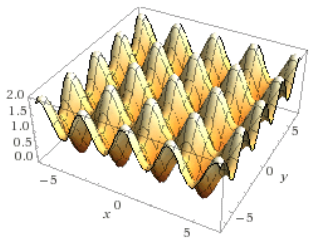
(a) $f(x, y) = x^2 + y^2$



(b) $f(x, y) = -x^2 + y^2$

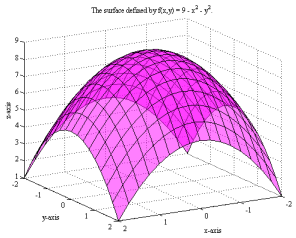


(c) $f(x, y) = \cos^2(x) + y^2$

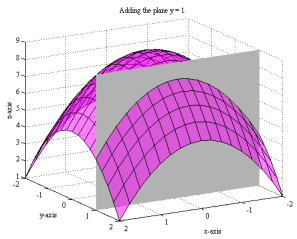


(d) $f(x, y) = \cos^2(x) + \cos^2(y)$

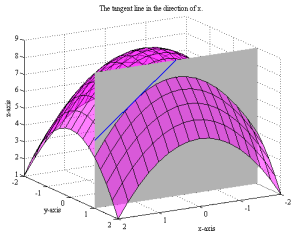
Partial Derivatives



(a) *Surface given by* $f(x, y) = 9 - \frac{x}{2} - \frac{y}{2}$

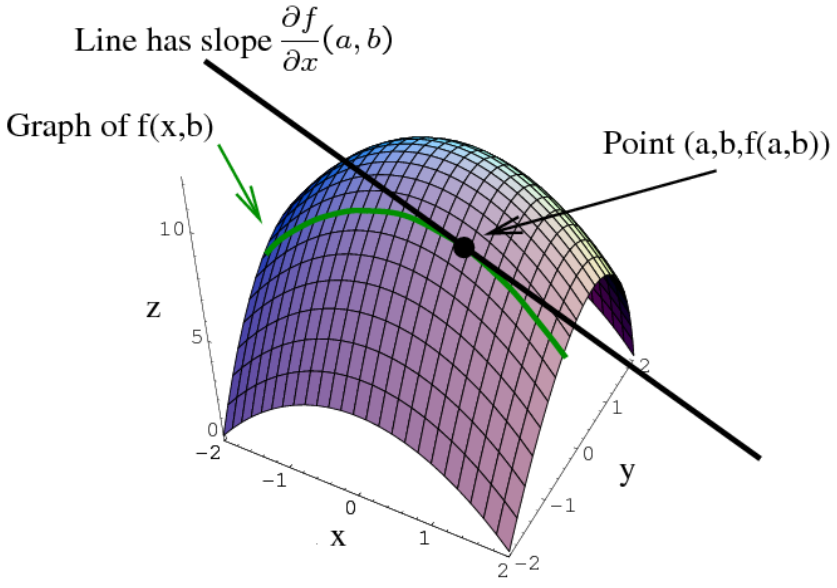


(b) *Plane* $y = 1$



(c) $f(x, 1) = 8 - \frac{x}{2}$ *denotes a curve, and* $f'(x) = -2x$ *denotes derivative (or slope) of that curve*

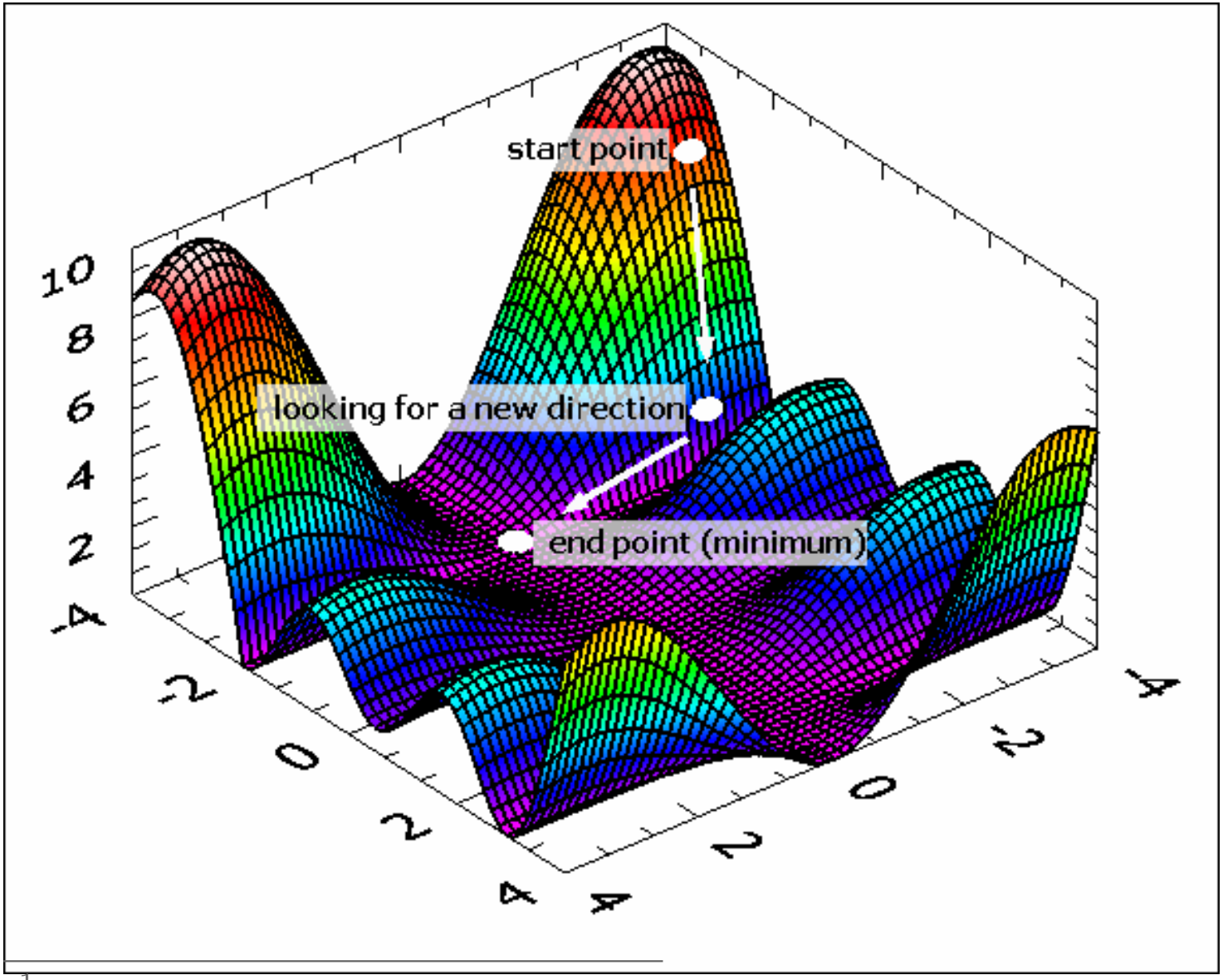
Partial Derivatives (contd. . .)



Idea of Gradient Descent Algorithm

- Start at some random point (of course final results will depend on this)
- Take steps based on the gradient vector of the current position till convergence
  - Gradient vector give us direction and rate of fastest increase any any point
  - Any point x if the gradient is nonzero, then the direction of gradient is the direction in which the function most quickly from x
  - The magnitude of gradient is the rate of increase in that direction

Idea of Gradient Descent Algorithm[1]



[1] Credits for all the images in this sections goes to Michailidis and Maiden

Gradient Descent

- **AIM**: To minimize the function

$$L(w) = \sum_{n=1}^{N} \left[y_n w^\intercal x_n - \log(1 + \exp(w^\intercal x_n) \right]$$

- We do this by calculating the derivative of L w.r.t w.

- **Note**: Since log function is concave in w, this has a unique minimum.

Gradient Descent

- **AIM**: To minimize the function

$$L(w) = \sum_{n=1}^{N} \left[y_n w^\intercal x_n - \log(1 + \exp(w^\intercal x_n) \right]$$

- **Gradient**:

$$\frac{\partial L}{\partial w} = -\sum_{n=1}^{N} \left[y_n x_n - \frac{\exp(w^\intercal x_n)}{1 + \exp(w^\intercal x_n)} x_n \right]$$

$$= -\sum_{n=1}^{N} (y_n - \mu_n) x_n = X^{-1}(\mu - y)$$

$$\text{where } \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_N \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \text{and} \quad X = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{N \times D}$$

Gradient Descent (contd...)

- Since there is no closed form solution, we take a recourse to iterative methods like gradient descent.
- Gradient Descent:
  1. Initialize $w^{(1)} \in \mathbb{R}^D$ randomly.
  2. Iterate until the convergence.

$$w^{(t+1)} = w^{(t)} - \eta \underbrace{\sum_{n=1}^{N} \left(\mu_n^{(t)} - y_n\right) x_n}_{\text{Gradient at previous value}}$$

  - $\mu_n^{(t)} = \sigma(w^{(t)\intercal} x_n)$
  - η is the learning rate.

Gradient Descent (contd...)

- We have the following update

$$w^{(t+1)} = w^{(t)} - \eta \underbrace{\sum_{n=1}^{N} (\mu_n^{(t)} - y_n) x_n}_{\text{Gradient at previous value}}$$

- Note: Calculating gradient in each iteration requires all the data. When N is large this may not be feasible.

- Stochastic Gradient Descent: Use mini-batches to compute the gradient.

Gradient Descent: Some Remarks

Note on Learning Rate:

- Sometimes choosing the learning rate is difficult
  - Larger learning rate $\rightarrow$ Too much fluctuation.
  - Smaller learning rate $\rightarrow$ Slow convergence

To deal with this problem:

- Choose optimal step size at each iteration η_t using line search.
- Add momentum to the update.

$$w^{(t+1)} = w^{(t)} - \eta_{(t)} g^{(t)} + \alpha_t \left(w^{(t)} - w^{(t-1)} \right)$$

- Use second order methods like Newton method to exploit the curvature of the loss function. (But we need to compute Hessian matrix.)

Multiclass Logistic or Softmax Regression

- Logistic regression can be extend for the multiclass case.
- Let $y_n \in \{0, 1, \ldots, k-1\}$
- Define

$$P(y_n = k | x_n, W) = \frac{\exp(w_k^\intercal x_n)}{\sum_{l=1}^{K} \exp(w_k^\intercal x_{n_l})}$$
$$= \mu_{n_k}$$

  * μ_{n_K}: Probability that n^{th} sample belongs to k^{th} class and $\sum_{l=1}^{k} \mu_{n_l} = 1$

- Softmax: Class k with largest $w_k^\intercal x_n$ dominates the probability.

Multiclass Logistic or Softmax Regression

- $P(y_n = k|x_n, W) = \frac{\exp(w_k^\intercal x_n)}{\sum_{l=1}^{K} \exp(w_k^\intercal x_n)}$
- $W = [w_1 w_2 \dots w_k]_{D \times K}$
  - We can think of y_n are drawn from multimodal distribution

$$P(y|X, W) = \prod_{n=1}^{N} \prod_{l=1}^{K} \mu_{n_l}^{y_{n_l}} : \text{ Likelihood function}$$

    - where $y_{n_l} = 1$ if true class of example n is l and $y_{n_l} = 0$ for all other l.

Hyperplane based classifiers and Perceptron

Supervised Learning Problem

- Given data $\{(x_n, y_n)\}_{n=1}^{N}$ find $f : \mathcal{X} \to \mathcal{Y}$ that best approximates the relation between X and Y.
- Determine f such a way that loss $l(y, f(x))$ is minimum.
- f and l are specific to the problem and the method that we choose.

Linear Regression

- Data: $\{(x_n, y_n)\}_{n=1}^{N}$
  - $x_n \in \mathbb{R}^D$ is a D dimensional input
  - $y_n \in \mathbb{R}$ is the output

  Aim is to find a **hyperplane** that fits **best** these points.
- Here hyperplane is a model of choice i.e.,

$$f(x) = \sum_{j=1}^{D} x_j w_j + b = w^\intercal x + b$$

  - Here $w_1, \ldots, w_d$ and b are model parameters
- Best is determined by some loss function

$$Loss(w) = \sum_{n=1}^{N} [y_n - f(x_n)]^2$$

- **Aim** : Determine the model parameters that minimize the loss.

Problem Set-Up

- Two class classification
- Instead of the exact labels estimate the probabilities of the labels i.e.

$$\begin{aligned} \text{Predict} \qquad & P(y_n = 1 | x_n, w) = \mu_n \\ & P(y_n = 0 | x_n, w) = 1 - \mu_n \end{aligned}$$

- Here (x_n, y_n) is the input output pair.

Logistic Regression(Contd...)

Problem

Find a function f such that,

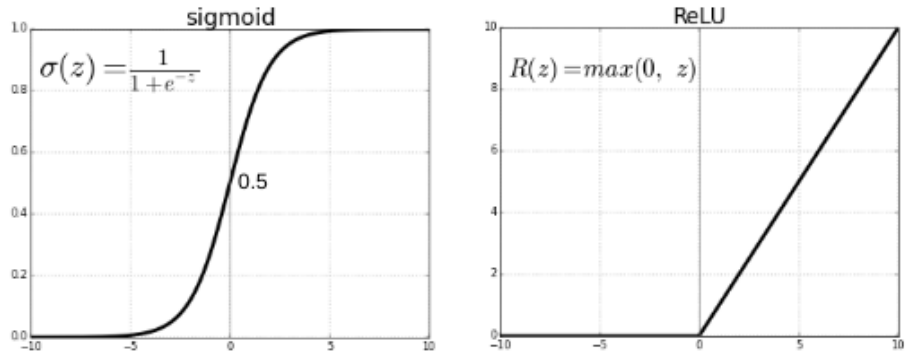
$$\mu = f(x_n)$$

Model

$$\mu_n = f(x_n) = \sigma(w^\intercal x_n) = \frac{1}{1 + \exp(-w^\intercal x_n)}$$
$$= \frac{\exp(w^\intercal x_n)}{1 + \exp(w^\intercal x_n)}$$

Sigmoid Function

- Here $\sigma(.)$ is the sigmoid function.



- The model first computes a real valued score $w^\intercal x = \sum_{i=1}^{D} w_i x_i$ and then nonlinearly "squashes" it between (0,1) to turn into a probability.

Logistic Regression(contd...)

Loss Function: Here we use cross entropy loss instead of squared loss.

Cross entropy loss is defined as:

$$
\begin{aligned}
L(y_n, f(x_n)) &= \begin{cases} -\log(\mu_n) & \text{when} \quad y_n = 1 \\ -\log(1-\mu_n) & \text{when} \quad y_n = 0 \end{cases} \\
&= -y_n \log(\mu_n) - (1-y_n)\log(1-\mu_n) \\
&= -y_n \log(\sigma(w^\intercal x_n)) - (1-y_n)\log(1-\sigma(w^\intercal x_n))
\end{aligned}
$$

And now empirical risk is

$$
L(w) = -\sum_{n=1}^{N} [y_n w^\intercal x_n - \log(1+\exp(w^\intercal x_n))]
$$

Logistic Regression(contd...)

By taking the derivative w.r.t w

$$\frac{\partial L}{\partial w} = \sum_{n=1}^{N} (\mu_n - y_n) x_n$$

- Here the Gradient Descent Algorithm is
  1. Initialize $w^{(1)} \in \mathbb{R}^D$ randomly
  2. Iterate until the convergence

$$\underbrace{w^{(t+1)}}_{\substack{\text{New learned} \\ \text{parameter or} \\ \text{weights}}} = \underbrace{w^{(t)}}_{\substack{\text{previous} \\ \text{value}}} - \underbrace{\eta}_{\substack{\text{Learning} \\ \text{rate}}} \sum_{n=1}^{N} \underbrace{(\mu_n^{(t)} - y_n) x_n}_{\substack{\text{Gradient at} \\ \text{previous value}}}$$

  - Note: Here $\mu^{(t)} = \sigma(w^{(t)\intercal} x_n)$

Logistic Regression (contd. . .)

Let us take a look at the update equation again

$$\underbrace{w^{(t+1)}}_{\substack{\text{New learned}\\ \text{parameter or}\\ \text{weights}}} = \underbrace{w^{(t)}}_{\substack{\text{previous}\\ \text{value}}} - \underbrace{\eta}_{\substack{\text{Learning}\\ \text{rate}}} \sum_{n=1}^{N} \underbrace{(\mu_n^{(t)} - y_n)x_n}_{\substack{\text{Gradient at}\\ \text{previous value}}}$$

What do we notice here?

Problem: Calculating gradient in each iteration requires all the data. When N is large this may not be feasible.

Stochastic Gradient Descent

- **Strategy:** Approximate gradient using randomly chosen data point (x_n, y_n)

$$w^{(t+1)} = w^{(t)} - \eta_t(\mu_n^{(t)} - y_n)x_n$$

.

- **Also**: Replace predicted label probability $\mu_n^{(t)}$ by predicted binary label $\hat{y}_n^{(t)}$, where

$$\hat{y}_n^{(t)} = \begin{cases} 1 & \text{if } \mu_n^{(t)} \geqslant 0.5 \text{ or } w^{(t)\intercal} x_n \geqslant 0 \\ 0 & \text{if } \mu_n^{(t)} < 0.5 \text{ or } w^{(t)\intercal} x_n < 0 \end{cases}$$

Stochastic Gradient Descent (contd. . .)

- **Hence**: Update rule becomes

$$w^{(t+1)} = w^{(t)} - \eta_t(\hat{y}_n^{(t)} - y_n)x_n$$

  - This is mistake driven update rule
  - $w^{(t)}$ gets updated only when there is a misclassification i.e. $\hat{y}_n^{(t)} \neq y_n$

Stochastic Gradient Descent (contd. . .)

We will do one more simple change:

- **Change**: the class labels to {-1,+1}

$$\implies \hat{y}_n^{(t)} - y_n = \begin{cases} -2y_n & if \;\; \hat{y}_n^{(t)} \neq y_n^{(t)} \\ 0 & if \;\; \hat{y}_n^{(t)} = y_n^{(t)} \end{cases}$$

- **Hence**: Whenever there is a misclassification.

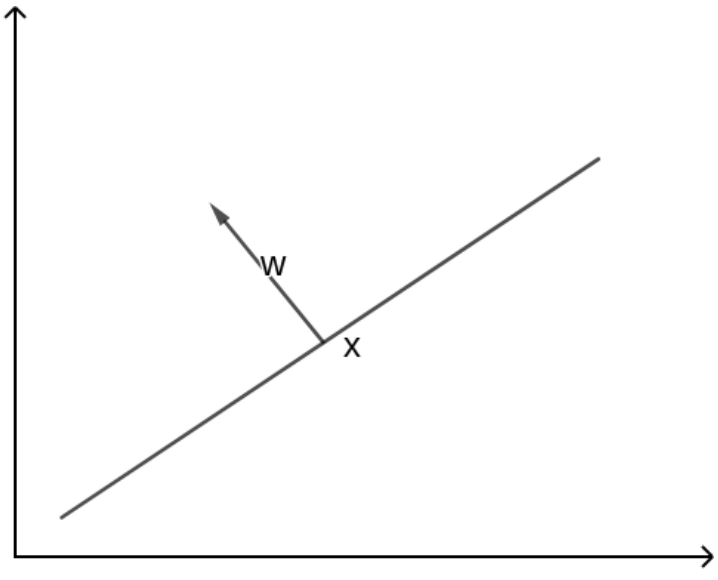
$$w^{(t+1)} = w^{(t)} - 2\eta_{(t)} y_n x_n$$

- $\implies$ This is a perceptron learning algorithm which is a hyperplane based learning algorithm.

Hyperplanes

- Separates a d-dimensional space into two half spaces(positive and negative)
- Equation of the hyperplane is

$$w^\intercal x = 0$$



- By adding bias $b \in \mathbb{R}$

$w^\intercal x + b = 0 \quad b > 0$ moving the hyperplane parallely along w

$b < 0$ opposite direction

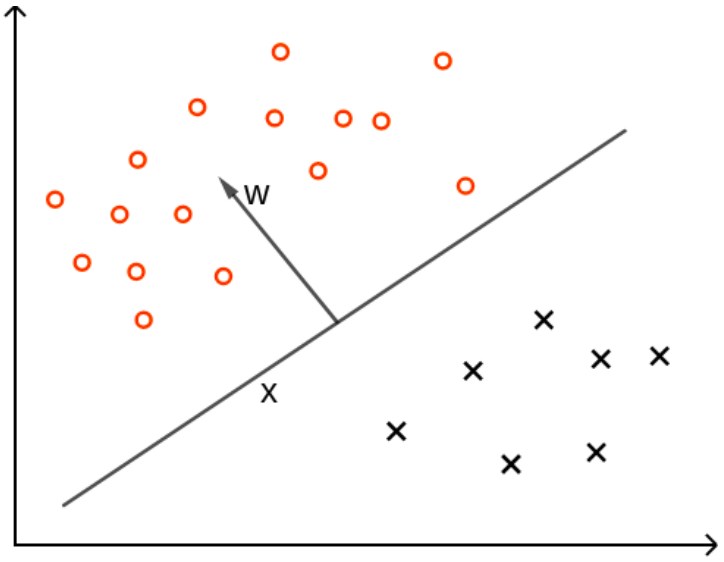
Hyperplane based classification

- Classification rule

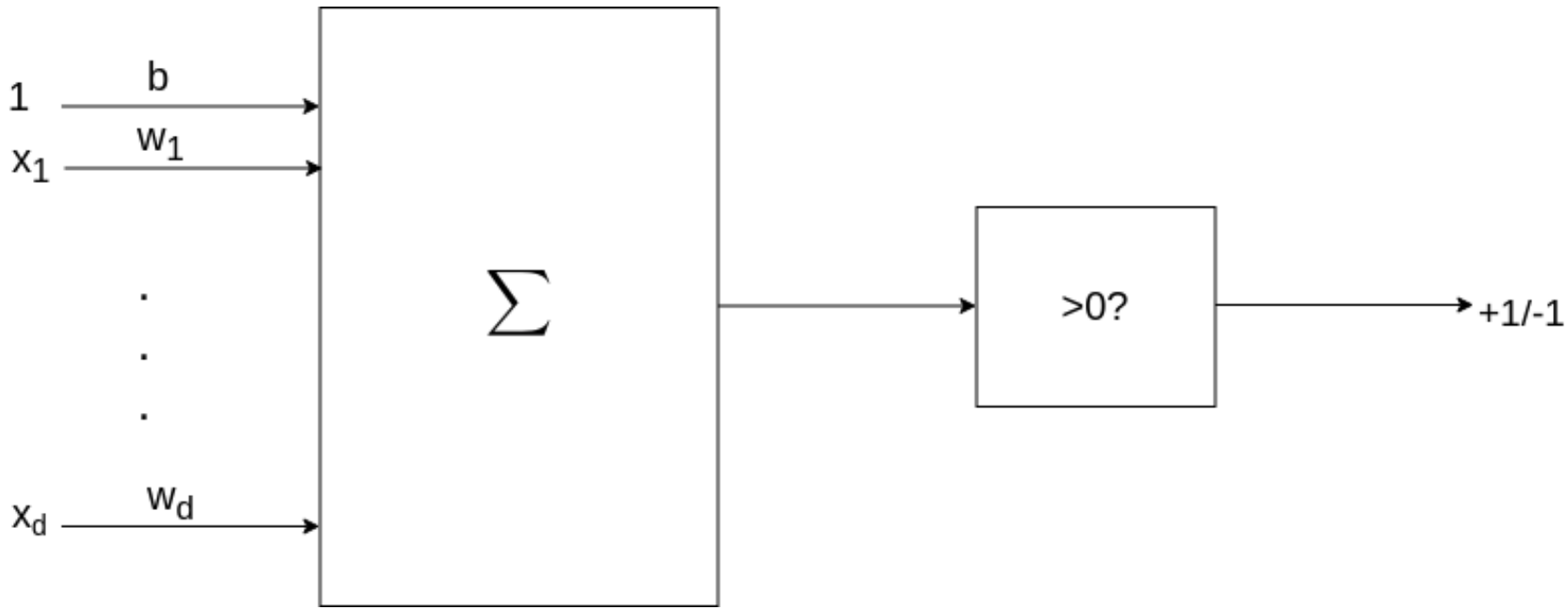
$$y = \text{sign}(w^\intercal x + b)$$

-

$$w^\intercal x + b > 0 \implies y = +1$$
$$w^\intercal x + b < 0 \implies y = -1$$



Hyperplane based classification



The Perceptron Algorithm (Rosenblatt, 1958)

- Aim is to learn a linear hyperplane to separate two classes.
- Mistake drives online learning algorithm
- Guaranteed to find a separating hyperplane if data is linearly separable.

Perceptron Algorithm

- Given training data $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$
- Initialize $w_{old} = [0, ..., 0], \;\; b_{old} = 0$
- Repeat until convergence.
  - For a random $(x_n, y_n) \in \mathcal{D}$
  - If $y_n(w^\intercal x_n + b) \leq 0$
    [Or $\text{sign}(w^\intercal x + b) \neq y_n$ i.e mistake mode]
  - $w_{new} = w_{old} + y_n x_n$
  - $b_{new} = b_{old} + y_n$

Perceptron Convergence Theorem (Block and Novikoff)

"Roughly" : If the data is linearly separable perceptron algorithm converges.

What if the data is not linearly separable?

Yes! In practice, most often the data is not linearly separable. Then

- Make linearly separable using kernel methods.
- (Or) Use multilayer perceptron.

What are all these?

- The first leads to Support Vector Machines, that rules machine learning for decades
- The second one leads to Deep Learning!

What we learned?

- Maximum Likelihood Estimates
- Bayes again! MAP
- Probabilistic view of Linear and Logistic Regression
- Hyperplanes and Perceptrons
- The two very big paradigms in ML