ML Supervised Learning 6

by ambedkar@IISc

- Logistic regression
- Hyperplane based classifiers and perceptron

What we learning so far?

- Bayes Decision Theory
- Some foundational aspects of Machine learning and Generalizing capacity
- Linear Regression
- Regularization (very important)
- Gradient Descent

Topics

Logistic Regression

Problem Set Up

- Two class classification
- Instead of the exact labels estimate the probabilities of the labels.
- that is predict

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

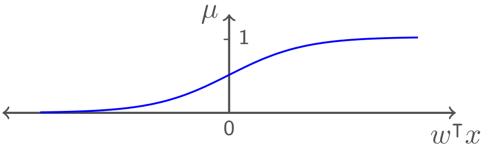
The Logistic Regression Model

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

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

$$w^\intercal x = \sum_{d=1}^{D} w_d x_d$$

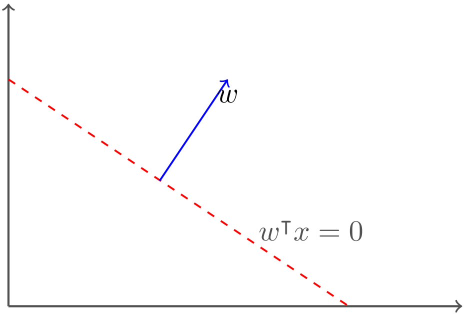
and **non-linearly** squashes it between $(0, 1)$ to turn this into a probability.



Logistic Regression: Sigmoid function

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

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

Logistic Regression: MLE formulation(contd...)

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

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

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

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

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

- 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.

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

- 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

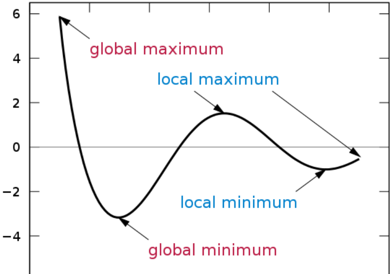
- Almost all problems in machine learning leads to optimization problems
- The following two factors decides the fate of any method:
  - What kind of optimization problem that we are led to
  - What are all optimization methods that are available to us
- There are several methods that are available for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

Gradient Descent methods are used in ....

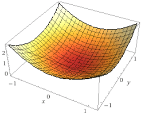
- Linear Regression
- Logistic Regression
  - It is just classification, but instead of labels it gives us class probability
- Support Vector Machines
- Neural Networks
  - The backbone of neural networks is Back-propagation algorithm

Example of an objective

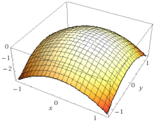
- Most often, we do not even have functional form of the objective.
  - Given x, we can only compute $f(x)$
  - Sometime this may involve a simulating a system
  - Computing each $f(x)$ can be time consuming



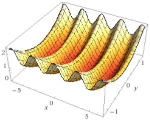
Multivariate Functions



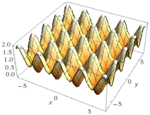
(a) $f(x, y) = x^2 + y^2$



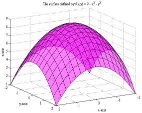
(b) $f(x, y) = -x^2 + y^2$



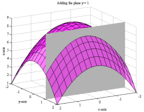
(c) $f(x, y) = \cos^2(x) + y^2$



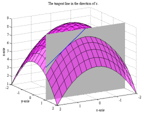
(d) $f(x, y) = \cos^2(x) + \cos^2(y)$



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

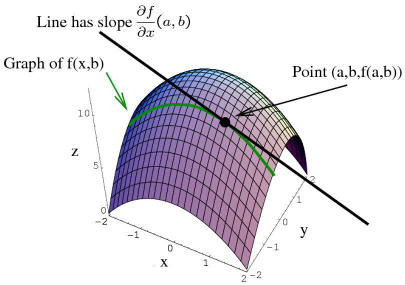


***(b)** Plane $y = 1$*



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

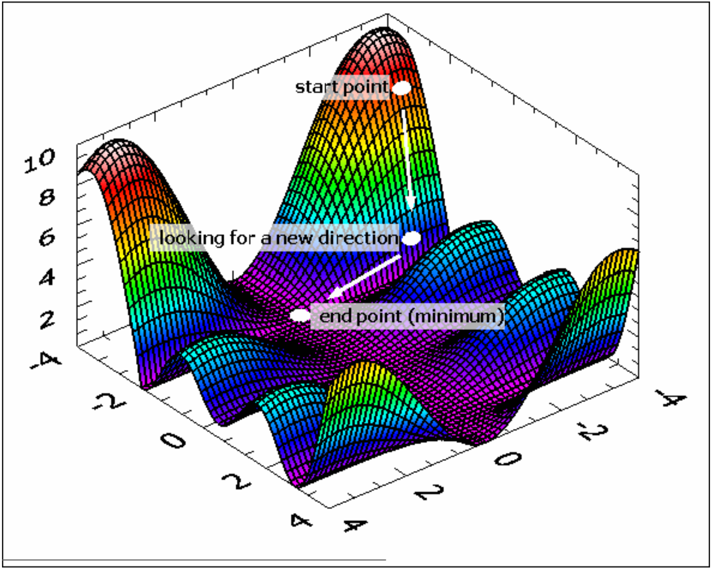
Partial Derivatives (contd. . .)



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**:

$$\begin{aligned} \frac{\partial L}{\partial w} &= -\sum_{n=1}^{N} \left[y_n x_n - \frac{\exp(w^\intercal x_n)}{1 + \exp(w^\intercal x_n)} x_n \right] \\ &= -\sum_{n=1}^{N} (y_n - \mu_n) x_n = X^{-1}(\mu - y) \end{aligned}$$

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

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

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

  * 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.

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

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

Problem

Find a function f such that,

$$\mu = f(x_n)$$

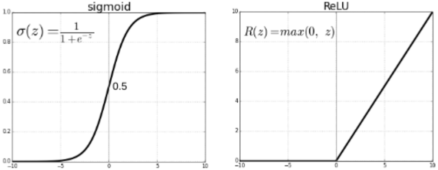
Model

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

Logistic Regression(Contd...)

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.

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

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

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

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

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

- **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.

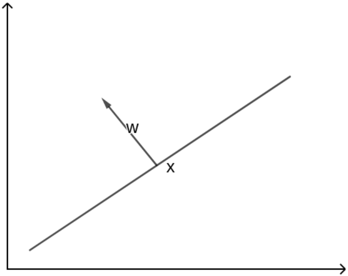
$$w^{(t+1)} = w^{(t)} - 2\eta_{(t)} y_n x_n$$

- $\implies$ 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 $b \in \mathbb{R}$

$$w^{\intercal}x + b = 0 \quad \begin{array}{ll} b > 0 & \text{moving the hyperplane parallely along } w \\ b < 0 & \text{opposite direction} \end{array}$$

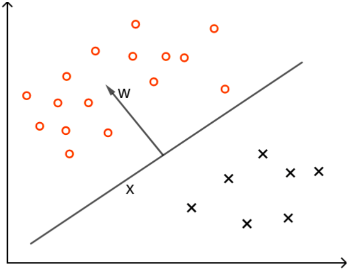
Hyperplane based classification

- Classification rule

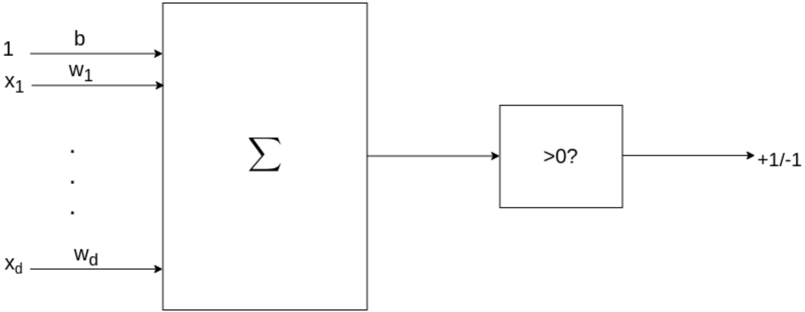
$$y = \mathsf{sign}(w^\intercal x + b)$$

-

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



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 sign$(w^\intercal x + b) \neq y_n$ i.e mistake mode]
  - $w_{new} = w_{old} + y_n x_n$
  - $b_{new} = b_{old} + y_n$

Case 1: Misclassified positive example ($y_n = +1$)

- That is we are in a mistake mode and the perceptron wrongly predicts that

$$\begin{aligned} w_{old}^T x_n + b_{old} &< 0 \\ \implies y_n(w_{old}^T x_n + b_{old}) &< 0 \end{aligned}$$

- Update

$$\begin{aligned} w_{new} &= w_{old} + y_n x_n = w_{old} + x_n \text{ (since } y_n = +1) \\ b_{new} &= b_{old} + y_n \quad\;\; = b_{old} + 1 \end{aligned}$$

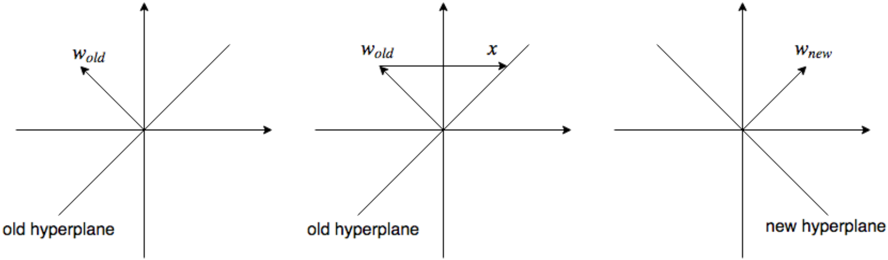
- Then

$$\begin{aligned} w_{new}^T x_n + b_{new} &= (w_{old} + x_n)^T x_n + b_{old} + 1 \\ &= (w_{old}^T x_n + b_{old}) + x_n^T x_n + 1 \end{aligned}$$

Case 1 (contd...) : Misclassified positive example ($y_n = +1$)

$\implies$ $w_{new}^T x_n + b_{new}$ is less negative than $w_{old}^T x_n + b_{old}$

$\implies$ Hence, hyperplane gets adjusted in a right direction.



Perceptron Algorithm : In Working (contd. . .)

Case 2: Misclassified negative example ($y_n = -1$)

- Again we are in a mistake mode and perceptron wrongly predicts that

$$w_{old}^T x_n + b_{old} > 0$$
$$i.e. y_n(w_{old}^T x_n + b_{old}) < 0$$

- Update

$$w_{new} = w_{old} + y_n x_n = w_{old} - x_n \text{ (since } y_n = -1)$$
$$b_{new} = b_{old} + y_n = b_{old} - 1$$

- Then

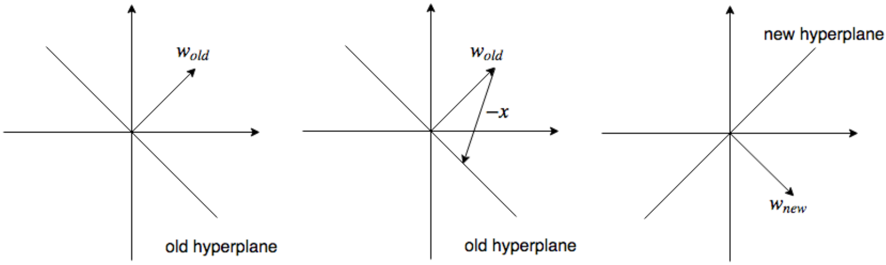
$$w_{new}^T x_n + b_{new} = (w_{old} - x_n)^T x_n + b_{old} - 1$$
$$= (w_{old} x_n + b_{old}) - (x_n^T x_n + 1)$$

Perceptron Algorithm : In Working (contd. . .)

Case 2 (contd. . .) : Misclassified negative example ($y_n = -1$)

$\implies$ $w_{new}^T x_n + b_{new}$ is less positive than $w_{old}^T x_n + b_{old}$

$\implies$ Hence, hyperplane gets adjusted in a right direction.



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!