Perceptron and Intro to SVM

Support Vector Machines

Online Learning via Stochastic Gradient Descent for Logistic Regression

Recall the gradient descent (GD) update rule for (unreg.) logistic regression

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \sum_{n=1}^{N} (\mu_n^{(t)} - y_n)\mathbf{x}_n$$

where the *predicted* probability of y_n being 1 is $\mu_n^{(t)} = \frac{1}{1+\exp(-\mathbf{w}^{(t)\top}\mathbf{x}_n)}$

Stochastic GD (SGD): Approx. the gradient using a randomly chosen $(\mathbf{x}_n, y_n)$

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

where η_t is the learning rate at update t (typically decreasing with t)

Let's replace the predicted label prob. $\mu_n^{(t)}$ by the predicted binary label $\hat{y}_n^{(t)}$

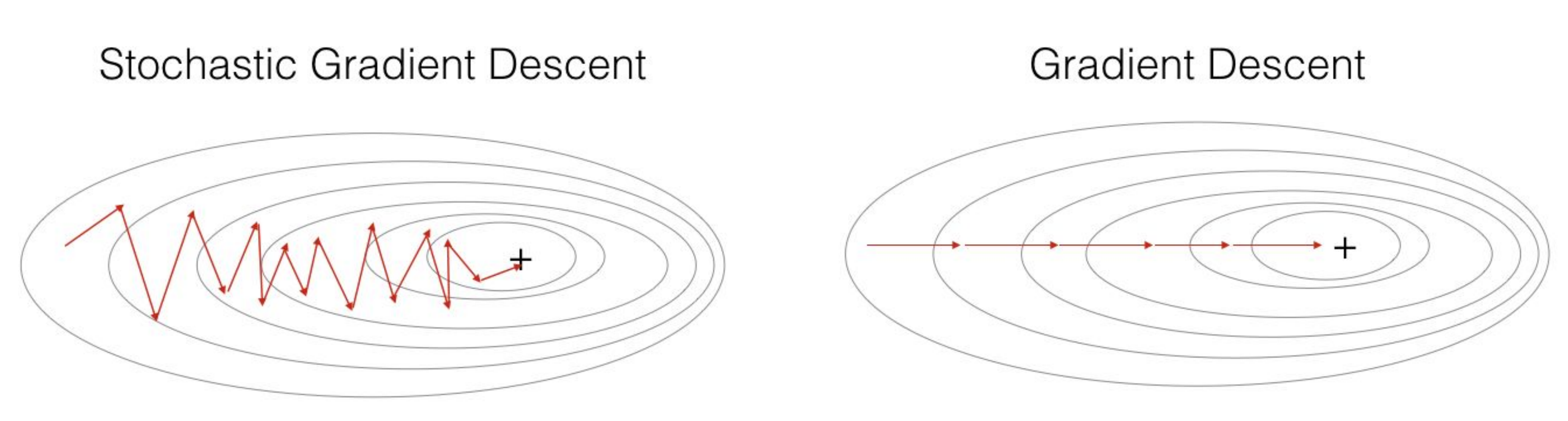
$$\boxed{\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t(\hat{y}_n^{(t)} - y_n)\mathbf{x}_n}$$

where

$$\hat{y}_n^{(t)} = \begin{cases} 1 & \text{if } \mu_n^{(t)} \geq 0.5 \quad \text{or} \quad \mathbf{w}^{(t)\top}\mathbf{x}_n \geq 0 \\ 0 & \text{if } \mu_n^{(t)} < 0.5 \quad \text{or} \quad \mathbf{w}^{(t)\top}\mathbf{x}_n < 0 \end{cases}$$

Thus $\mathbf{w}^{(t)}$ gets updated only when $\hat{y}_n^{(t)} \neq y_n$ (i.e., when $\mathbf{w}^{(t)}$ mispredicts)

SGD vs BGD



Mistake-Driven Learning

Consider the "mistake-driven" SGD update rule

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

Let's, from now on, assume the binary labels to be {-1,+1}, not {0,1}. Then

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

Thus whenever the model mispredicts (i.e., $\hat{y}_n^{(t)} \neq y_n$), we update $\mathbf{w}^{(t)}$ as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + 2\eta_t y_n \mathbf{x}_n$$

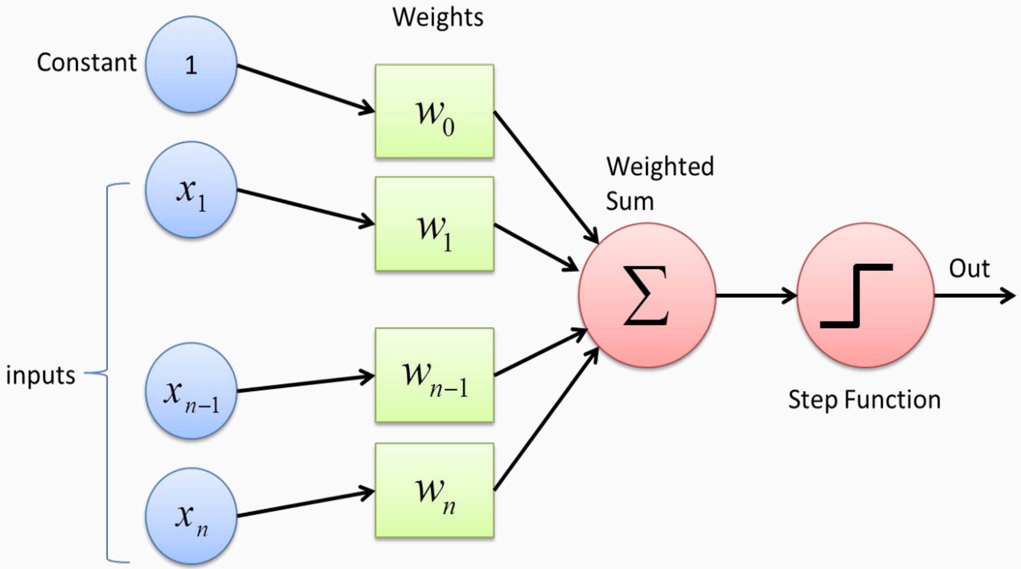
For $\eta_t = 0.5$, this is akin to the Perceptron (a hyperplane based learning algo)

$$\boxed{\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_n \mathbf{x}_n}$$

Note: There are other ways of deriving the Perceptron rule (will see shortly)

Perceptron (Sounds like a wise Transformer)

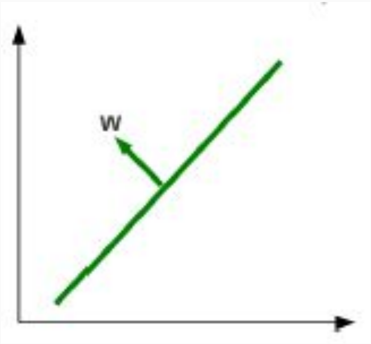
- **Perceptron is a single layer neural network** and a multi-layer perceptron is called Neural Networks.
- One of the earliest algorithms for binary classification (Rosenblatt, 1958)
- Learns a **linear hyperplane** to separate the two classes.
- The perceptron consists of 4 parts .
    - Input values or One input layer
    - Weights and Bias
    - Net sum
    - Activation Function
- Guaranteed to find a separating hyperplane if the data is **linearly separable.**



Hyperplanes and Margins

Hyperplanes

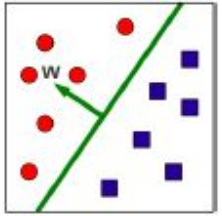
- Separates a D-dimensional space into two half-spaces (positive and negative)



- Defined by normal vector **w** ϵ R^D (pointing towards positive half-space)
- **w** is orthogonal to any vector **x** lying on the hyperplane
    - $\mathbf{w}^T x = 0$ (equation of the hyperplane)
- Assumption: The hyperplane passes through origin. If not, add a bias b 2 R
    - $\mathbf{w}^T x + b = 0$
- b > 0 means moving it parallely along **w** (b < 0 means in opposite direction)

Hyperplane based Linear Classification

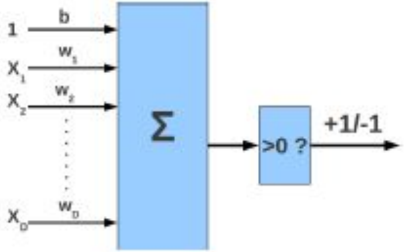
Represents the decision boundary by a hyperplane $\boldsymbol{w} \in \mathbb{R}^D$



For binary classification, $\boldsymbol{w}$ is assumed to point *towards* the positive class

Classification rule: $y = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$

- $\boldsymbol{w}^T\boldsymbol{x} + b > 0 \Rightarrow y = +1$
- $\boldsymbol{w}^T\boldsymbol{x} + b < 0 \Rightarrow y = -1$

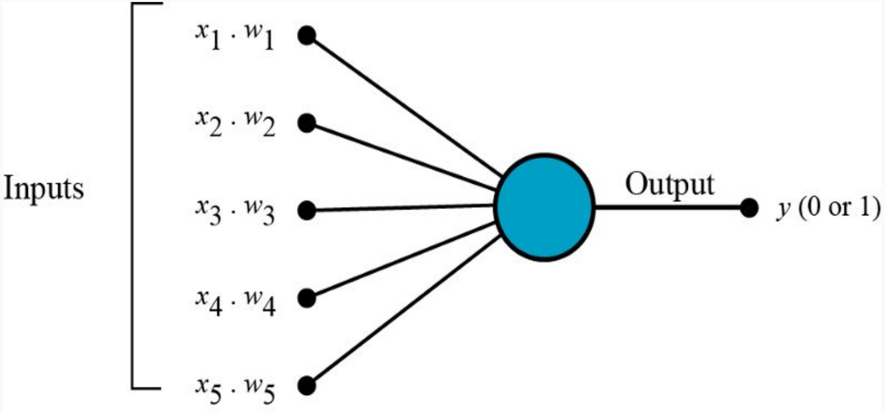


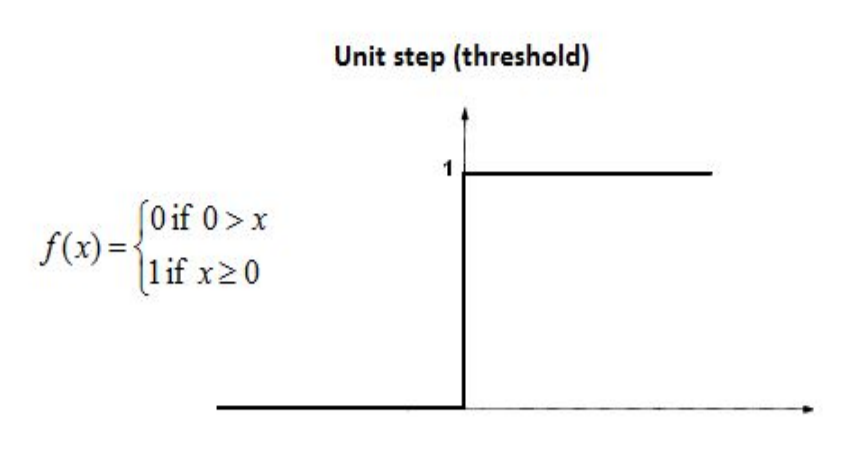
Note: $y(\boldsymbol{w}^T\boldsymbol{x} + b) < 0$ mean a mistake on training example $(\boldsymbol{x}, y)$

Note: Some algorithms that we have already seen (e.g., "distance from means", logistic regression, etc.) can also be viewed as learning hyperplanes

Example

- The perceptron works on these simple steps.
    - All the inputs ***x*** are multiplied with their weights ***w***. Let's call it ***k*.**





    - ***Add*** all the multiplied values and call them ***Weighted Sum.***
    - ***Apply*** that weighted sum to the correct ***Activation Function***.

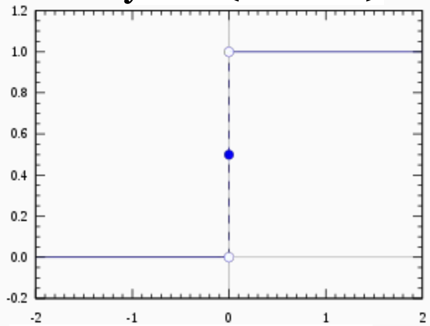
Example : Logical Function

Let’s start with a very simple problem:

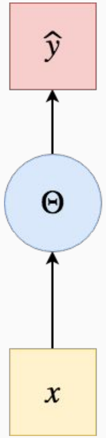
- *Can a perceptron implement the NOT logical function?*
    - NOT(x) is a 1-variable function, that means that we will have one input at a time: N=1.
    - Also, it is a **logical function**, and so both the input and the output have only two possible states: 0 and 1 (i.e., False and True):
    - the Heaviside step function seems to fit our case since it produces a binary output.

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

$$\hat{y} = \Theta(wx + b)$$



w = -1 and *b = 0.5*.

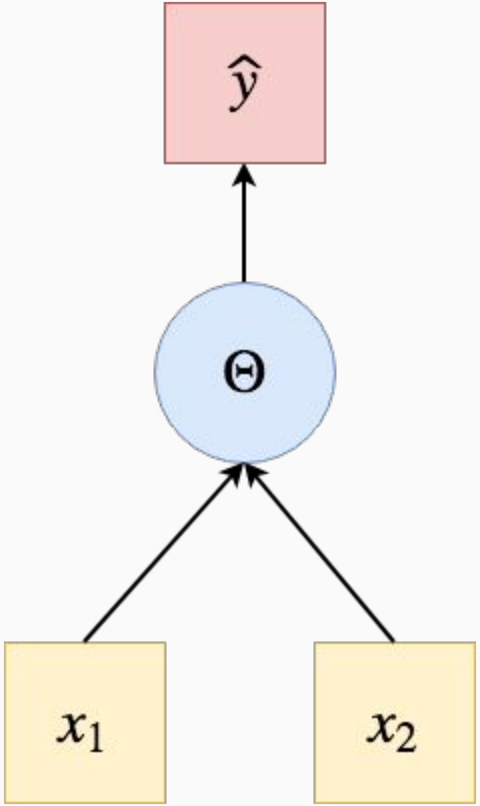


AND logical function

- This graph is associated with the following computation:

$$\hat{y} = \Theta(w1*x1 + w2*x2 + b)$$

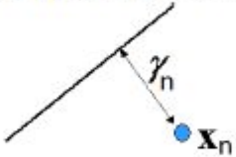
- Can you guess which are three values for these parameters which would allow the perceptron to solve the **AND ?**



Notion of Margins

Geometric margin γ_n of an example $\boldsymbol{x}_n$ is its signed distance from hyperplane

$$\gamma_n = \frac{\boldsymbol{w}^T \boldsymbol{x}_n + b}{||\boldsymbol{w}||}$$



Geometric margin may be +ve/-ve based on which side of the plane $\boldsymbol{x}_n$ is

Margin of a set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ w.r.t. $\boldsymbol{w}$ is the *min. abs. geometric margin*

$$\gamma = \min_{1 \le n \le N} |\gamma_n| = \min_{1 \le n \le N} \frac{|(\boldsymbol{w}^T \boldsymbol{x}_n + b)|}{||\boldsymbol{w}||}$$

Functional margin of $\boldsymbol{w}$ on a training example $(\boldsymbol{x}_n, y_n)$: $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$

- Positive if $\boldsymbol{w}$ predicts y_n correctly
- Negative if $\boldsymbol{w}$ predicts y_n incorrectly

A Loss Function for Hyperplane based Classification

For a hyperplane based model, let's consider the following loss function

$$\ell(\mathbf{w}, b) = \sum_{n=1}^{N} \ell_n(\mathbf{w}, b) = \sum_{n=1}^{N} \max\{0, -y_n(\mathbf{w}^T \mathbf{x}_n + b)\}$$

Seems natural: if $y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 0$, then the loss on $(\mathbf{x}_n, y_n)$ will be 0; otherwise the model will incur some positive loss when $y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0$

Let's perform stochastic optimization on this loss. Stochastic gradients are

$$\begin{aligned} \frac{\partial \ell_n(\mathbf{w}, b)}{\partial \mathbf{w}} &= -y_n \mathbf{x}_n \\ \frac{\partial \ell_n(\mathbf{w}, b)}{\partial b} &= -y_n \end{aligned}$$

(when $\mathbf{w}$ makes a mistake, and are zero otherwise)

Upon every mistake, update rule for $\mathbf{w}$ and b (assuming learning rate $= 1$)

$$\begin{aligned} \mathbf{w} &= \mathbf{w} + y_n \mathbf{x}_n \\ b &= b + y_n \end{aligned}$$

These updates define the Perceptron algorithm

The Perceptron Algorithm

- Given: Training data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
- Initialize: $\boldsymbol{w}_{old} = [0, \ldots, 0], b_{old} = 0$
- Repeat until convergence:
  - For a random $(\boldsymbol{x}_n, y_n) \in \mathcal{D}$
    - if $sign(\boldsymbol{w}^T\boldsymbol{x}_n + b) \neq y_n$ or $y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \leq 0$ (i.e., mistake is made)

$$
\begin{aligned}
\boldsymbol{w}_{new} &= \boldsymbol{w}_{old} + y_n\boldsymbol{x}_n \\
b_{new} &= b_{old} + y_n
\end{aligned}
$$

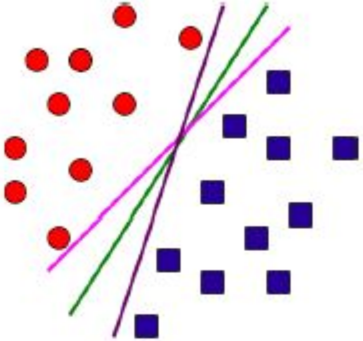
- Stopping condition: stop when either
  - All training examples are classified correctly
    - May overfit, so less common in practice
  - A fixed number of iterations completed, or some convergence criteria met
  - Completed one pass over the data (each example seen once)
    - E.g., examples arriving in a streaming fashion and can't be stored in memory

The Best Hyperplane Separator?

Perceptron finds one of the many possible hyperplanes separating the data

- .. if one exists

Of the many possible choices, which one is the best?



Intuitively, we want the hyperplane having the maximum margin

Large margin leads to good generalization on the test data

Support Vector Machine (SVM)

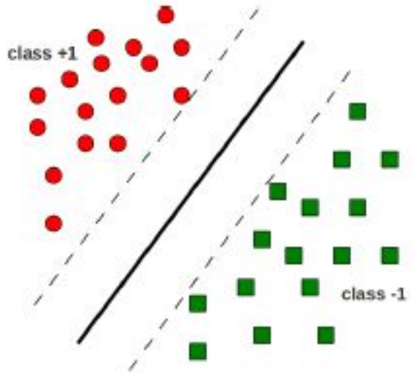
Probably the most popular/influential classification algorithm

Backed by solid theoretical groundings (Vapnik and Cortes, 1995)

A hyperplane based classifier (like the Perceptron)

Additionally uses the Maximum Margin Principle

- Finds the hyperplane with *maximum separation margin* on the training data



Support Vector Machine

A hyperplane based linear classifier defined by $\boldsymbol{w}$ and b

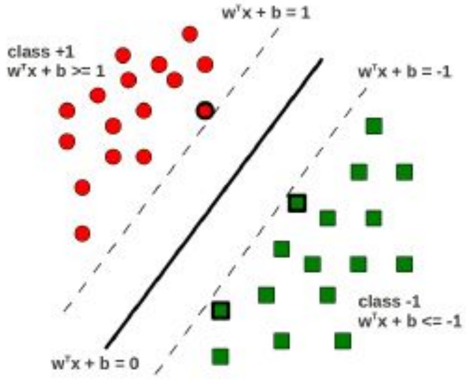
Prediction rule: $y = sign(\boldsymbol{w}^T\boldsymbol{x} + b)$

Given: Training data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$

Goal: Learn $\boldsymbol{w}$ and b that achieve the maximum margin

For now, assume the entire training data is correctly classified by $(\boldsymbol{w}, b)$

- Zero loss on the training examples (non-zero loss case later)

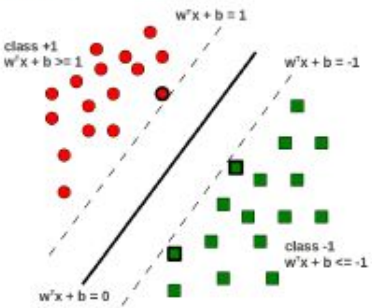


- Assume the hyperplane is such that
  - $\boldsymbol{w}^T\boldsymbol{x}_n + b \geq 1$ for $y_n = +1$
  - $\boldsymbol{w}^T\boldsymbol{x}_n + b \leq -1$ for $y_n = -1$
  - Equivalently, $y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$
    $\Rightarrow \min_{1\leq n\leq N} |\boldsymbol{w}^T\boldsymbol{x}_n + b| = 1$
  - The hyperplane's margin:

$$\boxed{\gamma = \min_{1\leq n\leq N} \frac{|\boldsymbol{w}^T\boldsymbol{x}_n + b|}{||\boldsymbol{w}||} = \frac{1}{||\boldsymbol{w}||}}$$

Support Vector Machine: The Optimization Problem

We want to maximize the margin $\gamma = \frac{1}{||\mathbf{w}||}$



Maximizing the margin γ = minimizing $||\mathbf{w}||$ (the norm)

Our optimization problem would be:

$$\boxed{\begin{array}{ll} \text{Minimize} & f(\mathbf{w}, b) = \dfrac{||\mathbf{w}||^2}{2} \\ \text{subject to} & y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \qquad n = 1, \ldots, N \end{array}}$$

This is a Quadratic Program (QP) with N linear inequality constraints

Large Margin = Good Generalization

Large margins intuitively mean good generalization

We can give a slightly more formal justification to this

Recall: Margin $\gamma = \frac{1}{||\boldsymbol{w}||}$

Large margin $\Rightarrow$ small $||\boldsymbol{w}||$

Small $||\boldsymbol{w}||$ $\Rightarrow$ regularized/simple solutions (w_i's don't become too large)

Simple solutions $\Rightarrow$ good generalization on test data