Lecture 15: Kernel perceptron, Neural Networks, SVMs etc

Instructor: Prof. Ganesh Ramakrishnan

Binary Classification using Perceptron

Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Assuming linearly separability, is there a learning rule that converges in finite time?

In general for multiclass $f(x) \in \{c_1, c_2 .. c_K\}$

2 approaches to multiclass

Wrapper around binary.
- 1-vs-rest for each c_i
- 1-vs-1 for each pair (c_i, c_j)

Wrapping based on single vs multilabeled

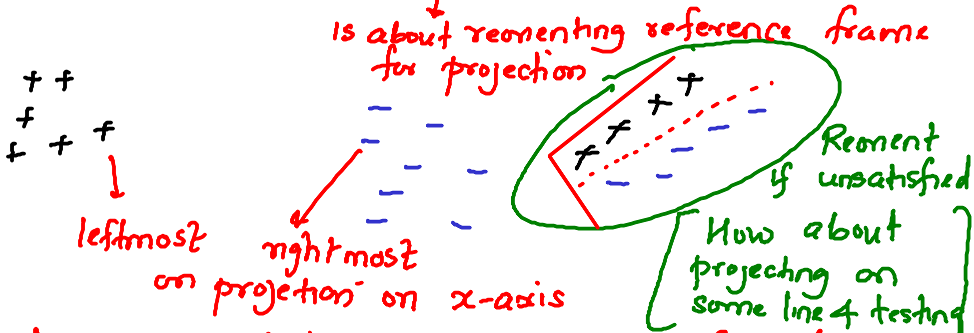
Principally extend the binary classifier

Effectiveness → measured using "real world" performance measures

Efficiency → measured in terms of how easy it is to train models

Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Assuming linearly separability, is there a learning rule that converges in finite time?



In a single dimension: Find leftmost & rightmost of + & − respectively after projection

Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Assuming linearly separability, is there a learning rule that converges in finite time?
- Naive Idea: Perform linear regression by constraining $y \in \{+1, -1\}$.

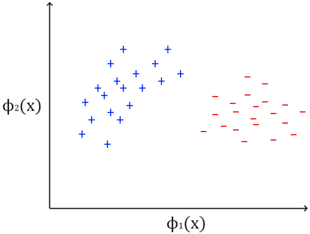
$$w_{ridge} = (\phi^T\phi + \lambda I)^{-1}\phi^T y$$

Does not respect that $y = -500$ & $y = -1$ should be treated similarly & so should $y = 500$ & $y = +1$

Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Assuming linearly separability, is there a learning rule that converges in finite time?
- Naive Idea: Perform linear regression by constraining $y \in \{+1, -1\}$.
- Can we do better? What is ideal?
- Desirable: Any new (unseen) input pattern similar to a seen pattern is **classified** correctly

$\phi(x_u)$ $\phi(x_s)$ $sgn(\phi^T(x)w+b)$

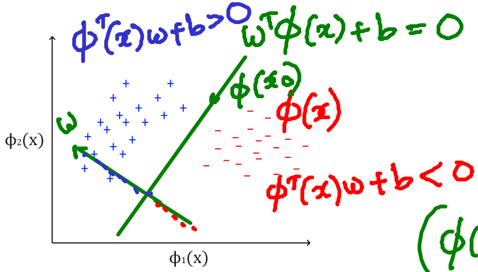


Won't happen if $y=+1$ & $y-1$ are treated to be more similar than $y=+1$ & $y=+500$!

Perceptron Classifier

Note: $\phi^T(x_0)w + b = 0 \Rightarrow \phi^T(x_0)w = -b$ for any $\phi(x_0)$ on hyperplane

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Assuming linearly separability, is there a learning rule that converges in finite time?
- Naive Idea: Perform linear regression by constraining $y \in \{+1, -1\}$.
- Can we do better? What is ideal?
- Desirable: Any new (unseen) input pattern similar to a seen pattern is **classified** correctly



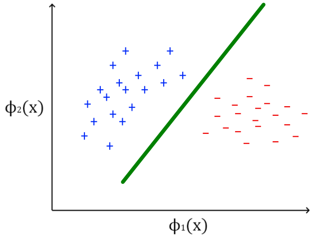
sgn(distance of point from hyperplane computed along w) $= y$

Linear Classification?

$(\phi(x)-\phi(x_0))^T w = \phi^T(x)w - \phi^T(x_0)w = \phi^T(x)w + b$

Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Assuming linearly separability, is there a learning rule that converges in finite time?
- Naive Idea: Perform linear regression by constraining $y \in \{+1, -1\}$.
- Can we do better? What is ideal?
- Desirable: Any new (unseen) input pattern similar to a seen pattern is **classified** correctly



signed distance of pt from separating hyperplane

Linear Classification?

$\mathbf{w}^\top \phi(\mathbf{x}) + b \geq 0$ *for +ve points* $(y = +1)$

$\mathbf{w}^\top \phi(\mathbf{x}) + b < 0$ *for -ve points* $(y = -1)$

$\mathbf{w}, \phi \in \mathbb{R}^m$

$y(w^T\phi(x) + b) \geq 0$

Perceptron Classifier: Setting up Notation

- Often, b is indirectly captured by including it in $\mathbf{w}$, and using a ϕ as: $\phi_{aug} = [\phi, 1]$

hide b within w_{aug}

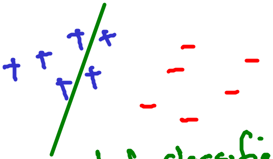
- Thus, $\mathbf{w}^\top \phi(\mathbf{w})$

$$= \begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_m & b \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_m \\ 1 \end{bmatrix}$$

- $\mathbf{w}^\top \phi(\mathbf{x}) = 0$ is the separating hyperplane.

Perceptron Intuition

1. Go over all the existing examples, whose class is known, and check their classification with the current weight vector
2. If correct, continue
3. If not, marginally correct the weights
    - By adding to the weights a quantity that is proportional to the product of the input pattern with the desired output $y = \pm 1$ such that unsigned distance of that misclassified point increases



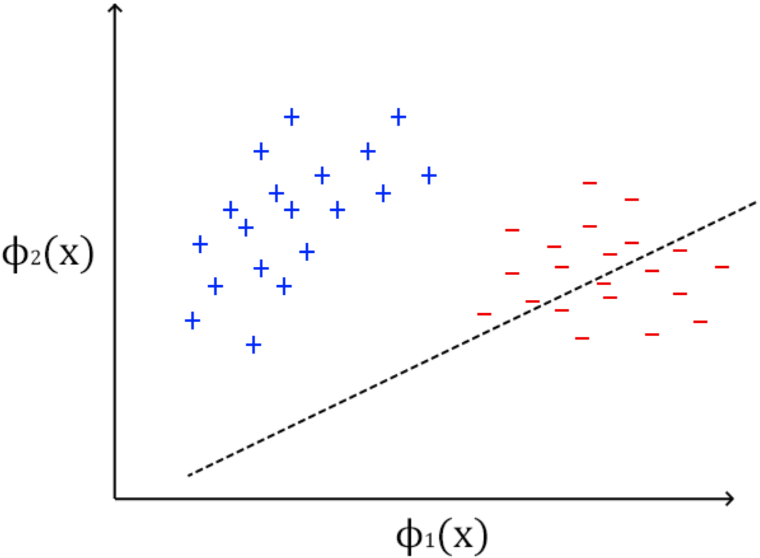
$y_i(w^T\phi(x_i) + b)$ = unsigned distance

Perceptron Update Rule

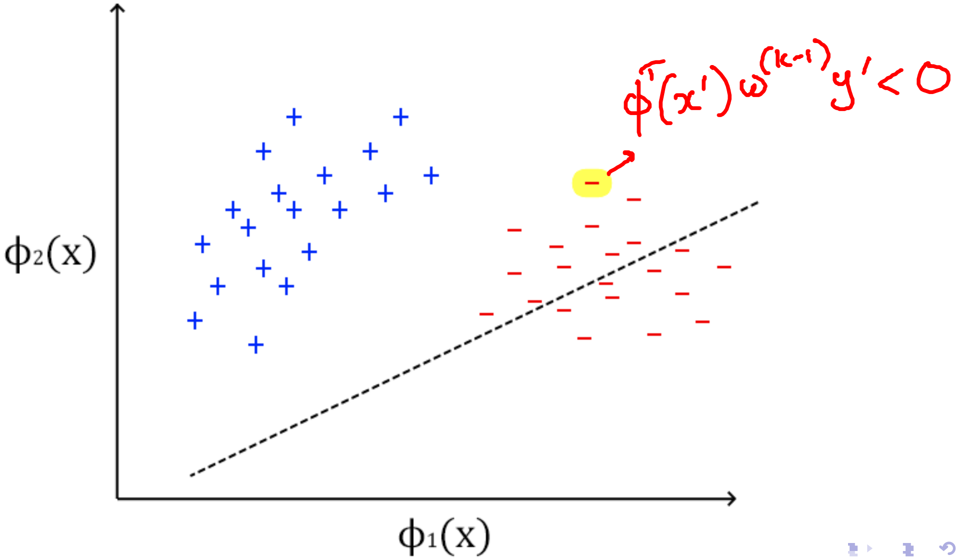
- Start with some weight vector $\mathbf{w}^{(0)}$, and for $k = 0, 1, 2, 3, \ldots, n$ (for every example), do: $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Gamma\phi(x')$
- where $\mathbf{x}'$ s.t. $\mathbf{x}'$ is misclassified by $(\mathbf{w}^{(k)})^\top \phi(\mathbf{x})$ i.e. $y'(\mathbf{w}^{(k)})^\top \phi(\mathbf{x}') < 0$

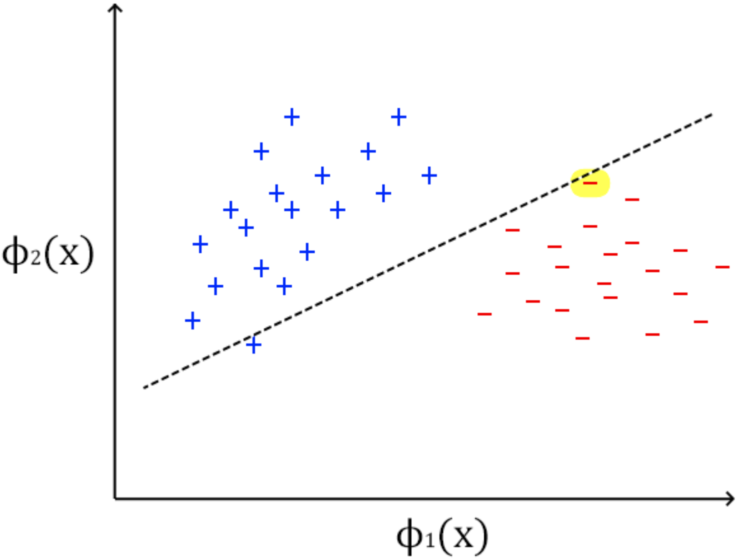
Find: x' s.t $(w^{(k-1)})^T \phi(x')y < 0$
$w^{(k)} = w^{(k-1)} + \Gamma\phi(x')$

What happens to the unsigned distance of x' after this update?

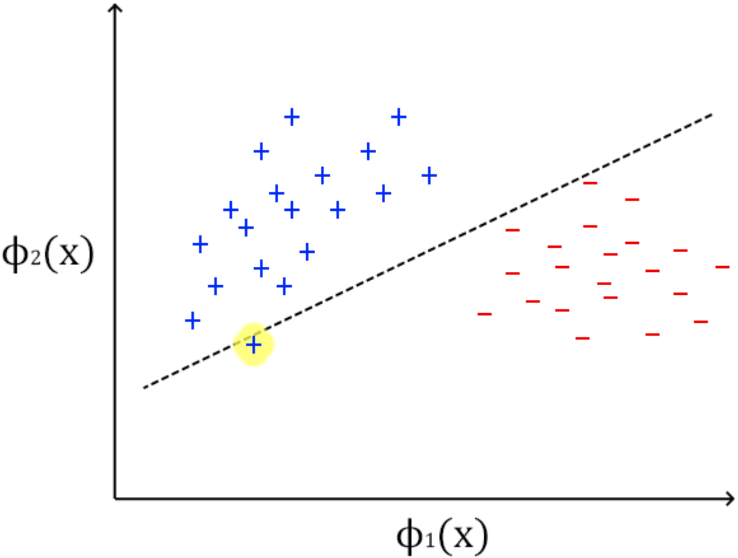


$\phi_2(x)$
$\phi_1(x)$

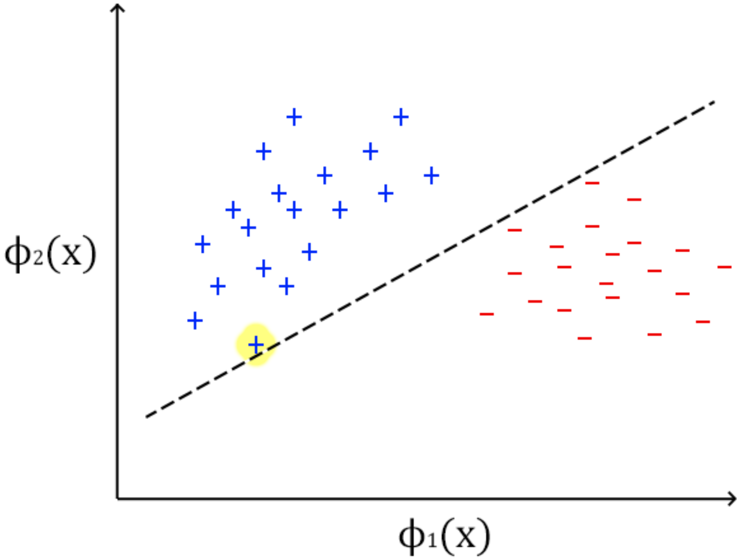

$\phi^T(x')\, w^{(k-1)} y' < 0$
$\phi_2(x)$
$\phi_1(x)$



$\phi_2(x)$
$\phi_1(x)$



$\phi_2(x)$
$\phi_1(x)$



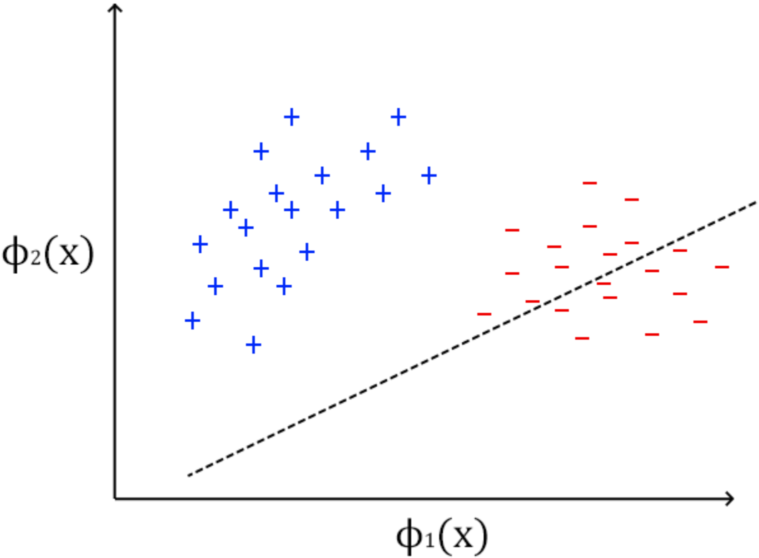
ϕ2(x)
ϕ1(x)

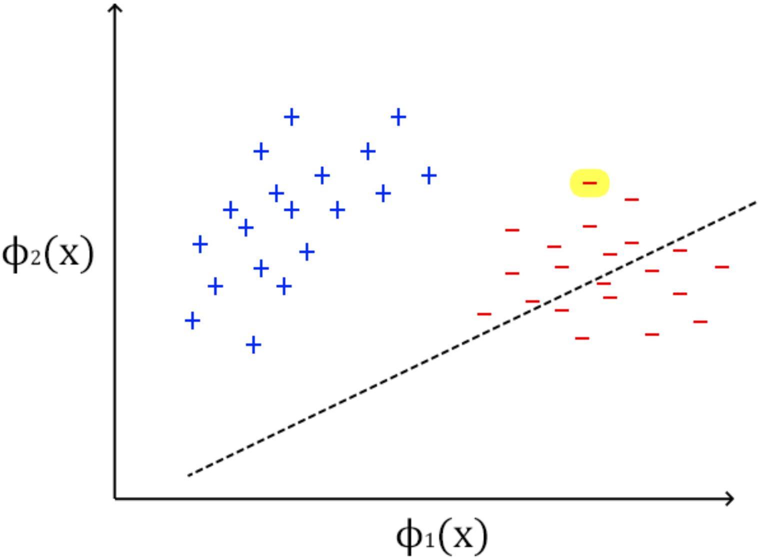
Some Notes about the Perceptron Algorithm

- Perceptron does not find the *best* seperating hyperplane, it finds *any* seperating hyperplane.
- In case the initial $\mathbf{w}$ does not classify all the examples, the seperating hyperplane corresponding to the final $\mathbf{w}^*$ will often *pass through an example*.
- The seperating hyperplane does not provide enough breathing space – this is what SVMs address!

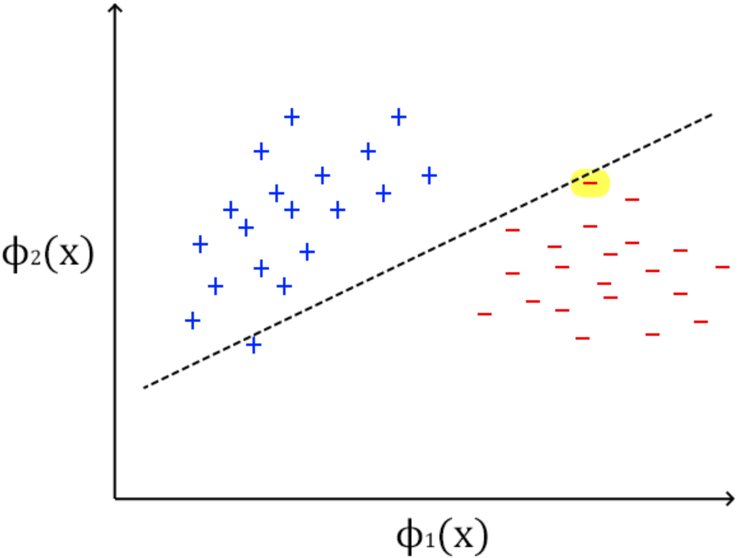
Perceptron Update Rule

- Start with some weight vector $w^{(0)}$, and for $k = 1, 2, 3, \ldots, n$ (for every example), do:
  $w^{(k)} = w^{(k-1)} + \Gamma\phi(x')$
- where x' s.t. x' is misclassified by $(w^{(k)})^\top \phi(x)$
  i.e. $y'(w^{(k)})^\top \phi(x') < 0$

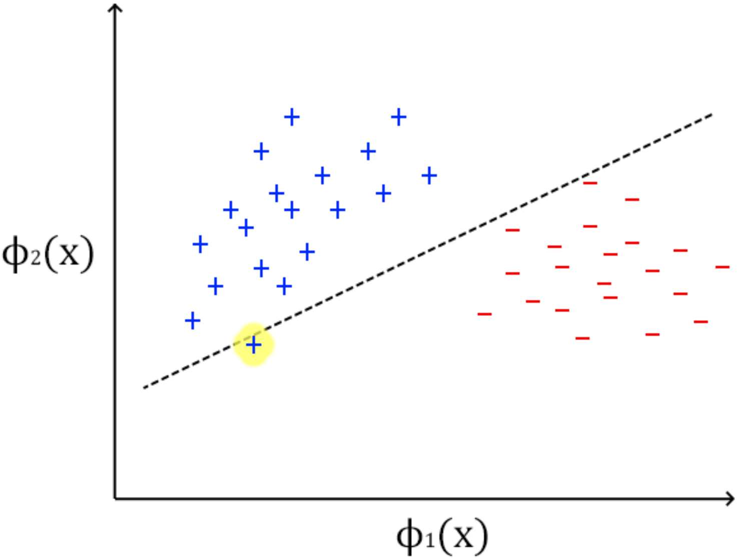

ϕ2(x)
ϕ1(x)



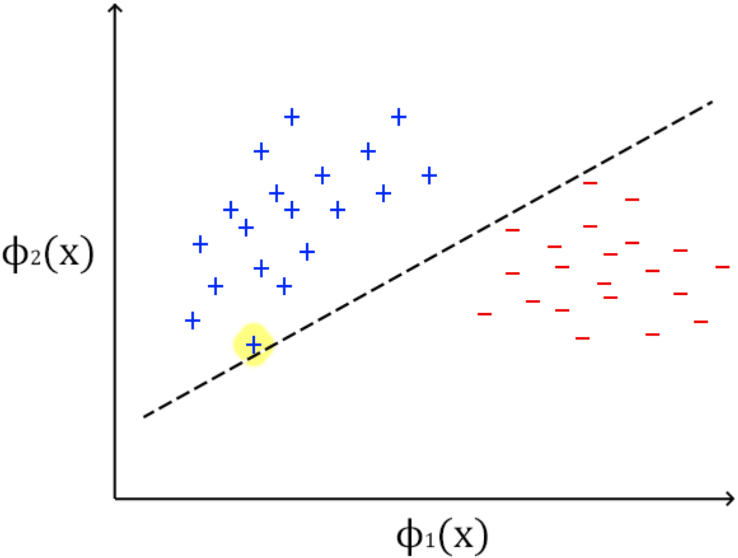
$\phi_2(x)$
$\phi_1(x)$



$\phi_2(x)$
$\phi_1(x)$

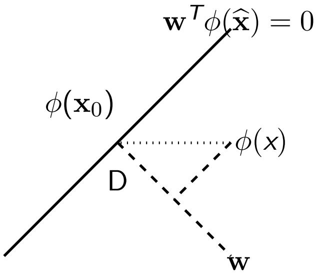


ϕ2(x)
ϕ1(x)


ϕ2(x)
ϕ1(x)

Perceptron Update Rule: Further analysis

- Explicitly account for signed distance of (misclassified) points from the hyperplane $\mathbf{w}^T\phi(\widehat{\mathbf{x}}) = 0$
  Distance from hyperplane can be calculated as follows:



  perpendicular distance $= y\mathbf{w}^T(\phi(\mathbf{x}_0) - \phi(\mathbf{x}))$
  Since $\mathbf{w}^T(\phi(\mathbf{x}_0)) = 0$ we get distance $= -y\mathbf{w}^T(\phi(\mathbf{x}))$

Perceptron Update Rule: Further analysis

- Perceptron works for two classes ($y = \pm 1$). A point is misclassified if $y\mathbf{w}^T(\phi(\mathbf{x})) < 0$
- Perceptron Algorithm:
  - INITIALIZE: w=ones()
  - REPEAT: for each $< \mathbf{x}, y >$
    - ★ If $y\mathbf{w}^T\Phi(\mathbf{x}) < 0$
    - ★ then, $\mathbf{w} = \mathbf{w} + \Phi(\mathbf{x}).y\lambda$
    - ★ endif
- **Intuition:**

$$\phi^T(x')\, w^{(k)} = \phi^T(x')\left(w^{(k-1)} + \lambda\,\phi(x')y'\right)$$

$$= \phi^T(x')w^{(k-1)} + \lambda\,\phi^T(x')\,\phi(x')y'$$

$$= \phi^T(x')w^{(k-1)} + \lambda\|\phi(x')\|^2 y'$$

If $y' = -1$ & $\phi^T(x')w^{(k-1)} > 0$ then $\phi^T(x')w^k = \phi^T(x')w^{(k-1)}$ + negative qty & vice versa

unsigned distance $= y'\phi^T(x')w^{(k)} = y'\phi^T(x')w^{(k-1)} + \lambda\|\phi(x')\|^2(y')^2 \geq 0$

Perceptron Update Rule: Further analysis

- Perceptron works for two classes ($y = \pm 1$). A point is misclassified if $y\mathbf{w}^T(\phi(\mathbf{x})) < 0$
- Perceptron Algorithm:
  - INITIALIZE: w=ones()
  - REPEAT: for each $< \mathbf{x}, y >$
    - If $y\mathbf{w}^T\Phi(\mathbf{x}) < 0$
    - then, $\mathbf{w} = \mathbf{w} + \Phi(\mathbf{x}).y$
    - endif
- **Intuition:**

$$\begin{aligned} y(\mathbf{w}^{(k+1)})^T\phi(\mathbf{x}) &= y(\mathbf{w}^k + y\phi^T(\mathbf{x})\phi(\mathbf{x}) \\ &= y(\mathbf{w}^k)^T\phi(\mathbf{x}) + y^2\|\phi(\mathbf{w})\|^2 \\ &> y(\mathbf{w}^k)^T\phi(\mathbf{x}) \end{aligned}$$

Since $y(\mathbf{w}^k)^T\phi(\mathbf{x}) \leq 0$, we have $y(\mathbf{w}^{(k+1)})^T\phi(\mathbf{x}) \; > \; y(\mathbf{w}^k)^T\phi(\mathbf{x}) \Rightarrow$ more hope that this point is classified correctly now.

Perceptron Update Rule: Further analysis

- Tries to minimize the error function E (sum of unsigned distances from hyperplane to misclassified points) over misclassified examples

$$E = - \sum_{(\mathbf{x},y)\in\mathcal{M}} y\mathbf{w}^T\phi(\mathbf{x})$$

where $\mathcal{M} \subseteq \mathcal{D}$ is the set of misclassified examples.

- **Gradient Descent (Batch Perceptron) Algorithm** $\nabla_{\mathbf{w}} E = - \sum_{(\mathbf{x},y)\in M} y\phi(\mathbf{x})$

$$\begin{aligned}\mathbf{w}^{(k+1)} &= \mathbf{w}^k - \eta \nabla_{\mathbf{w}} E \\ &= \mathbf{w}^k + \eta \sum_{(\mathbf{x},y)\in\mathcal{M}} y\phi(\mathbf{x})\end{aligned}$$

Less meaningful to take entire $\mathcal{M}$ since the fate of misclassified points is interdependent

Perceptron Update Rule: Further analysis

- Batch update considers all misclassified points simultaneously

$$\begin{aligned}\mathbf{w}^{(k+1)} &= \mathbf{w}^k - \eta \nabla_{\mathbf{w}} E \\ &= \mathbf{w}^k + \eta \sum_{(\mathbf{x},y)\in\mathcal{M}} y\phi(\mathbf{x})\end{aligned}$$

- Perceptron update $\Rightarrow$ *Stochastic Gradient Descent*:
$\nabla_{\mathbf{w}} E = -\sum_{(\mathbf{x},y)\in\mathcal{M}} y\phi(\mathbf{x}) = -\sum_{(\mathbf{x},y)\in\mathcal{M}} \nabla_{\mathbf{w}} E(\mathbf{x})$ s.t. $E(\mathbf{x}) = -y\mathbf{w}^T\phi(\mathbf{x})$

$$\begin{aligned}\mathbf{w}^{(k+1)} &= \mathbf{w}^k - \eta \nabla_w E(\mathbf{x}) \\ &= \mathbf{w}^k + \eta y\phi(\mathbf{x}) \qquad \text{(for any } (\mathbf{x}, y) \in \mathcal{M})\end{aligned}$$

Perceptron Update Rule: Further analysis

- **Formally,**:- If $\exists$ an optimal separating hyperplane with parameters $\mathbf{w}^*$ such that,

$$\forall\ (\mathbf{x}, y),\ \ y\phi^T(\mathbf{x})\mathbf{w}^* \geq 0$$

then the perceptron algorithm converges.
Proof:- We want to show that

$$\lim_{k\to\infty} \|\mathbf{w}^{(k+1)} - \rho\mathbf{w}^*\|^2 = 0 \tag{1}$$

(If this happens for some constant ρ, we are fine.)