Linear models for classification: Perceptron

Problem setting

- Given data points $\mathbf{x}_i, i = 1, 2, \ldots, m$
- Possible class choices: $c_1, c_2, \ldots, c_k$
- Wish to estimate a mapping/classifier,

$$f : \mathbf{x} \to \{c_1, c_2, \ldots, c_k\}$$

that predicts class labels $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m$

Problem setting

- Given data points $\mathbf{x}_i, i = 1, 2, \ldots, m$
- Possible class choices: $c_1, c_2, \ldots, c_k$
- Wish to estimate a mapping/classifier,

$$f : \mathbf{x} \to \{c_1, c_2, \ldots, c_k\}$$

  that predicts class labels $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m$
- In general, series of mappings

$$\mathbf{x} \xrightarrow{f(\cdot)} \mathbf{y} \xrightarrow{g(\cdot)} \mathbf{z} \xrightarrow{h(\cdot)} \{c_1, c_2, \ldots, c_k\}$$

  where y, z are in some latent space.

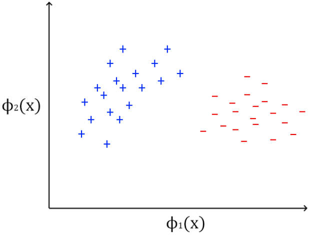
Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$

- Objective: Learn a linear classifier

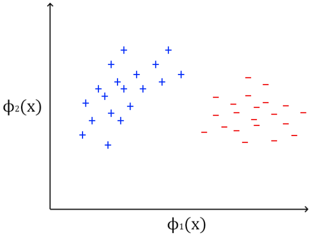
Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Desirable: Any new input pattern similar to a seen pattern is **classified** correctly



Perceptron Classifier

- Consider a binary classification problem: $f(\mathbf{x}) \in \{-1, +1\}$
- Desirable: Any new input pattern similar to a seen pattern is **classified** correctly



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)*

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

Perceptron Classifier

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

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

- $\mathbf{w}^\top \phi(\mathbf{x}) = 0$ 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,

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

Perceptron Intuition

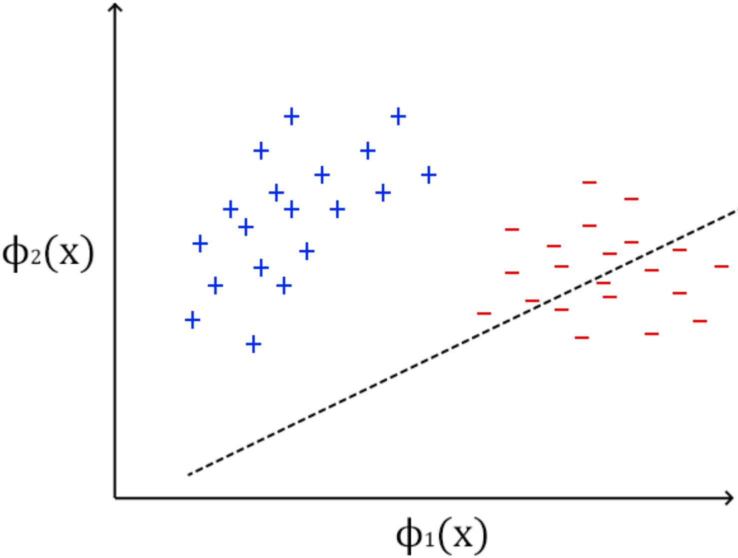
1. Go over all the existing examples, whose class is known, and check their classification with the current weight vector
2. If correct,continue
3. If not, marginally correct the weights ....
   - ....by adding to the weights a quantity that is proportional to the product of the input pattern with the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

Perceptron Update Rule

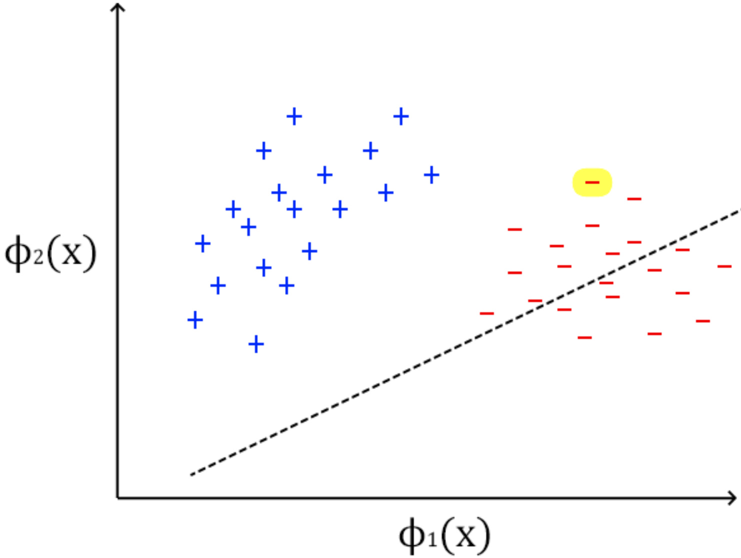
- Start with some weight vector $\mathbf{w}^{(0)}$, and
  for $k = 0, 1, 2, 3, \ldots, n$ (for every example denoted by ($\mathbf{x}$',y')), do:

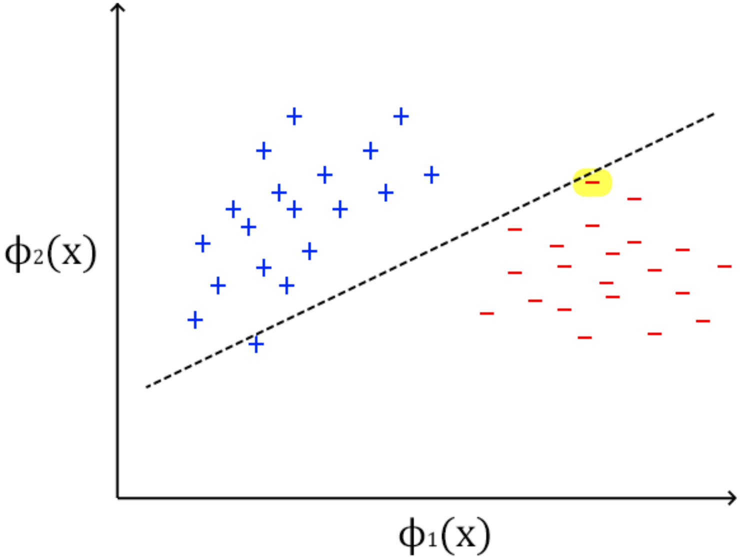
$$\mathbf{w}^{k+1} = \mathbf{w}^{k} + y'\phi(\mathbf{x}')$$

- only when $\mathbf{x}'$ is misclassified by $(\mathbf{w}^k)^\top \phi(\mathbf{x})$ i.e. $y'(\mathbf{w}^k)^\top \phi(\mathbf{x}') < 0$

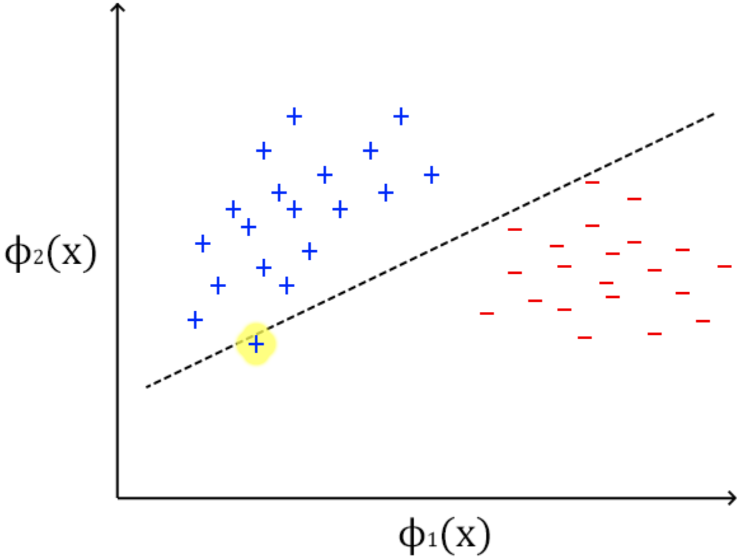


$\phi_2(x)$
$\phi_1(x)$

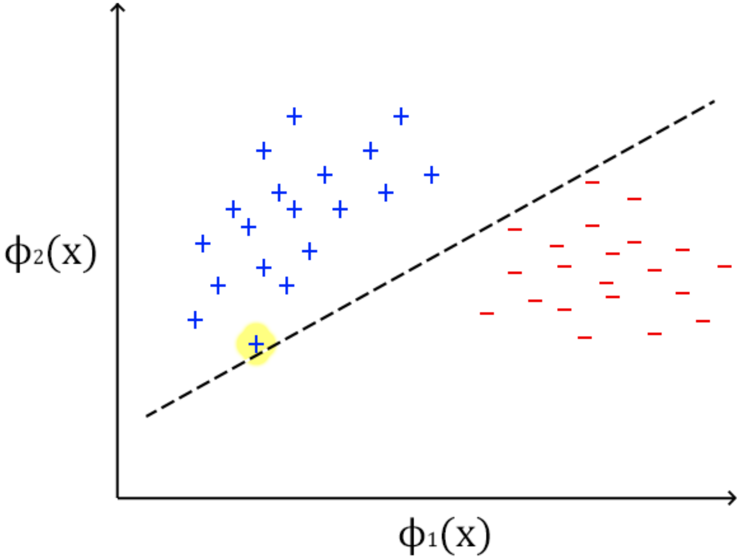

ϕ2(x)
ϕ1(x)



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



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



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

Perceptron: Separating Hyperplane

- Perceptron does not find the *best* separating hyperplane, it finds *any* seperating hyperplane.

Perceptron: Separating Hyperplane

- Perceptron does not find the *best* separating hyperplane, it finds *any* seperating hyperplane.
- The separating hyperplane does not provide enough breathing space – this is what SVMs address!

Perceptron Update Rule

- 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=zeros()
  - REPEAT: for each $< \mathbf{x}, y >$
    - If $y\mathbf{w}^T\Phi(\mathbf{x}) < 0$

Perceptron Update Rule

- 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=zeros()
  - REPEAT: for each $< \mathbf{x}, y >$
    - If $y\mathbf{w}^T\Phi(\mathbf{x}) < 0$
    - then, $\mathbf{w} = \mathbf{w} + \eta\phi(\mathbf{x}).y$

Perceptron Update Rule

- 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=zeros()
  - REPEAT: for each $< \mathbf{x}, y >$
    - If $y\mathbf{w}^T\Phi(\mathbf{x}) < 0$
    - then, $\mathbf{w} = \mathbf{w} + \eta\phi(\mathbf{x}).y$
    - endif

Perceptron Update Rule

- **Intuition:**

$$y(\mathbf{w}^{(k+1)})^T\phi(\mathbf{x}) \quad =$$

Perceptron Update Rule

- **Intuition:**

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

Since $y(\mathbf{w}^k)^T\phi(\mathbf{x}) \leq 0$, we have $y(\mathbf{w}^{(k+1)})^T\phi(\mathbf{x}) \; > \; y(\mathbf{w}^k)^T\phi(\mathbf{x}) \Rightarrow$

Perceptron Update Rule

- **Intuition:**

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

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.

Summarily: Perceptron Update Rule

- Start with zero-weights vector, $\mathbf{w} \leftarrow \vec{0}$
- For each training instance, $\{\mathbf{x}_i, y_i\} \in \mathcal{D}$

  Predict its label using the current weight vector

$$\hat{y}_i = \begin{cases} +1 \text{ if } \mathbf{w}^T\phi(\mathbf{x_i}) \geq 0 \\ -1 \text{ otherwise} \end{cases}$$

  If $y_i \neq \hat{y}_i$ (or equivalently, $y_i\mathbf{w}^T\phi(\mathbf{x}_i) < 0$)

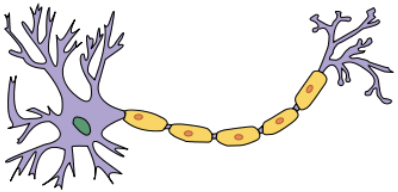
  Adjust the weight vector using the following update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + y_i\phi(\mathbf{x}_i)$$

- Repeat until $\text{sign}(\mathbf{w}^T\mathbf{x}) = y$ for all $(\mathbf{x}, y) \in \mathcal{D}$

Why is it called a perceptron?

Loose inspiration drawn from neurons



Features are inputs; each feature associated with a weight. Activation quantified by the sum of weighted features.

Image from Wikimedia commons

Perceptron can model linearly separable functions

Convergence of Perceptron Update Rule: Formally

Convergence of the perceptron algorithm

Consider the case when data is linearly separable i.e. there exists a weight vector $\mathbf{u}$ s.t. $y = \text{sign}(\mathbf{u}^T\mathbf{x}) \quad \forall \{\mathbf{x}, y\} \in \mathcal{D}$. Without loss of generality, let us assume that $\mathbf{u}$ is a unit-length vector. Let us also assume that the data is scaled to lie in the Euclidean ball of radius 1, i.e., $||\mathbf{x}|| \leq 1 \quad \forall \mathbf{x} \in \mathcal{D}$.

Convergence of the perceptron algorithm

Consider the case when data is linearly separable i.e. there exists a weight vector $\mathbf{u}$ s.t. $y = \text{sign}(\mathbf{u}^T\mathbf{x}) \quad \forall \{\mathbf{x}, y\} \in \mathcal{D}$. Without loss of generality, let us assume that $\mathbf{u}$ is a unit-length vector. Let us also assume that the data is scaled to lie in the Euclidean ball of radius 1, i.e., $||\mathbf{x}|| \leq 1 \quad \forall \mathbf{x} \in \mathcal{D}$.

Define a new quantity, margin of separation $\gamma = \min_{\mathbf{x} \in \mathcal{D}} |\mathbf{u}^T\mathbf{x}|$

Convergence of the perceptron algorithm

Consider the case when data is linearly separable i.e. there exists a weight vector $\mathbf{u}$ s.t. $y = \text{sign}(\mathbf{u}^T\mathbf{x}) \quad \forall\{\mathbf{x}, y\} \in \mathcal{D}$. Without loss of generality, let us assume that $\mathbf{u}$ is a unit-length vector. Let us also assume that the data is scaled to lie in the Euclidean ball of radius 1, i.e., $||\mathbf{x}|| \leq 1 \quad \forall \mathbf{x} \in \mathcal{D}$.

Define a new quantity, margin of separation $\gamma = \min_{\mathbf{x}\in\mathcal{D}} |\mathbf{u}^T\mathbf{x}|$

Theorem: If there exists a unit vector $\mathbf{u}$ such that $y\mathbf{u}^T\mathbf{x} \geq \gamma$ for all $\mathbf{x}$, then the number of weight updates made by the perceptron algorithm is at most $\frac{1}{\gamma^2}$.

Proof of the mistake bound for the perceptron algorithm

Let us track the following two quantities: $\mathbf{w}^T\mathbf{u}$ and $||\mathbf{w}||^2$.
Quantity 1: $\mathbf{w}^T\mathbf{u}$ on every update increases by at least γ.
For a positive example:

$$\begin{aligned}(\mathbf{w}+\mathbf{x})^T\mathbf{u} &= \mathbf{w}^T\mathbf{u} + \mathbf{x}^T\mathbf{u} \\ &\geq \mathbf{w}^T\mathbf{u} + \gamma\end{aligned}$$

Proof continued

Quantity 2: $||\mathbf{w}||^2$ after every update increases by at most 1.
For a positive example:

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

Proof continued

Putting these two inequalities together i.e. $\mathbf{w}_{i+1}^T\mathbf{u} \geq \mathbf{w}_i^T\mathbf{u} + \gamma$ and $||\mathbf{w}_{i+1}||^2 \leq ||\mathbf{w}_i||^2 + 1$, after k updates, we have

$$\mathbf{w}_k^T\mathbf{u} \geq k\gamma \text{ and } ||\mathbf{w}_k||^2 \leq k$$

Proof continued

Putting these two inequalities together i.e. $\mathbf{w}_{i+1}^T\mathbf{u} \geq \mathbf{w}_i^T\mathbf{u} + \gamma$ and $||\mathbf{w}_{i+1}||^2 \leq ||\mathbf{w}_i||^2 + 1$, after k updates, we have

$$\mathbf{w}_k^T\mathbf{u} \geq k\gamma \text{ and } ||\mathbf{w}_k||^2 \leq k$$

$$\Rightarrow \sqrt{k} \geq ||\mathbf{w}_k|| \geq \mathbf{w}_k^T\mathbf{u} \geq k\gamma$$

Proof continued

Putting these two inequalities together i.e. $\mathbf{w}_{i+1}^T \mathbf{u} \geq \mathbf{w}_i^T \mathbf{u} + \gamma$ and $||\mathbf{w}_{i+1}||^2 \leq ||\mathbf{w}_i||^2 + 1$, after k updates, we have

$$\mathbf{w}_k^T \mathbf{u} \geq k\gamma \text{ and } ||\mathbf{w}_k||^2 \leq k$$

$$\Rightarrow \sqrt{k} \geq ||\mathbf{w}_k|| \geq \mathbf{w}_k^T \mathbf{u} \geq k\gamma$$

$$\Rightarrow k \leq \frac{1}{\gamma^2}$$

What is the perceptron optimizing?

Hinge loss

Consider the following loss function for the perceptron algorithm:
$L_P(f_{\mathbf{w}}(\mathbf{x}), y) = \max(0, -y f_{\mathbf{w}}(\mathbf{x}))$

Hinge loss

Consider the following loss function for the perceptron algorithm:
$L_P(f_{\mathbf{w}}(\mathbf{x}), y) = \max(0, -yf_{\mathbf{w}}(\mathbf{x}))$

Apply the stochastic gradient descent (SGD) algorithm:

Consider a set of examples $\{\mathbf{x}_i, y_i\} \in \mathcal{D}$. Choose an example $\{\mathbf{x}_t, y_t\}$.

1. Compute the gradient of the loss of $f_{\mathbf{w}}(\mathbf{x}_t)$ with respect to $\mathbf{w}$, $\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x}_t), y)$
2. Update the weight vector: $\mathbf{w} \leftarrow \mathbf{w} - \nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x}_t), y)$

SGD and the perceptron algorithm

Consider a set of examples $\{\mathbf{x}_i, y_i\} \in \mathcal{D}$. Choose an example $\{\mathbf{x}, y\}$.

1. Compute the gradient of the loss of $f_{\mathbf{w}}(\mathbf{x})$ with respect to $\mathbf{w}$, $\nabla_{\mathbf{w}} L_P(f_{\mathbf{w}}(\mathbf{x}), y)$ where $L_P(f_{\mathbf{w}}(\mathbf{x}), y) = \max(0, -y f_{\mathbf{w}}(\mathbf{x}))$
2. Update the weight vector: $\mathbf{w} \leftarrow \mathbf{w} - \nabla_{\mathbf{w}} L_P(f_{\mathbf{w}}(\mathbf{x}), y)$

SGD and the perceptron algorithm

Consider a set of examples $\{\mathbf{x}_i, y_i\} \in \mathcal{D}$. Choose an example $\{\mathbf{x}, y\}$.

1. Compute the gradient of the loss of $f_{\mathbf{w}}(\mathbf{x})$ with respect to $\mathbf{w}$, $\nabla_{\mathbf{w}} L_P(f_{\mathbf{w}}(\mathbf{x}), y)$ where $L_P(f_{\mathbf{w}}(\mathbf{x}), y) = \max(0, -y f_{\mathbf{w}}(\mathbf{x}))$
2. Update the weight vector: $\mathbf{w} \leftarrow \mathbf{w} - \nabla_{\mathbf{w}} L_P(f_{\mathbf{w}}(\mathbf{x}), y)$

Step 2 of SGD gives $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$ which is exactly the perceptron update rule!

Which weight vector should we use for evaluation?

Option 1: Use the final $\mathbf{w}$ at test time. Not always a good idea!

Which weight vector should we use for evaluation?

Option 1: Use the final $\mathbf{w}$ at test time. Not always a good idea!

Option 2: Use an intermediate $\mathbf{w}$:

- **Voted Perceptron**: Count votes for weight vectors tallying how many examples they correctly classified.

Which weight vector should we use for evaluation?

Option 1: Use the final $\mathbf{w}$ at test time. Not always a good idea!

Option 2: Use an intermediate $\mathbf{w}$:

- **Voted Perceptron**: Count votes for weight vectors tallying how many examples they correctly classified.
- **Averaged Perceptron**: Use the averaged weight vector at the end of the algorithm rather than voting.

Beyond linearly separable data?