Online Learning via Stochastic Optimization, Perceptron, and Intro to SVMs

Piyush Rai

Machine Learning (CS771A)

Aug 20, 2016

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 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)

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 $\hat{y}_n^{(t)}$

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

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

$$\hat{y}_n^{(t)} =$$

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

$$\hat{y}_n^{(t)} = \begin{cases} 1 & \\ & \end{cases}$$

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

$$\hat{y}_n^{(t)} = \begin{cases} 1 & \text{if } \mu_n^{(t)} \geq 0.5 \\ & \end{cases}$$

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

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

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

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

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

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

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

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

Stochastic Gradient Descent for Logistic Regression

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

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

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

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

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

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 $\hat{y}_n^{(t)}$

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

where

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

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

Mistake-Driven Learning

- Consider the "mistake-driven" SGD update rule

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

Mistake-Driven Learning

- Consider the "mistake-driven" SGD update rule

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

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

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

Mistake-Driven Learning

- Consider the "mistake-driven" SGD update rule

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

- 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 $\boldsymbol{w}^{(t)}$ as

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

Mistake-Driven Learning

- Consider the "mistake-driven" SGD update rule

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

- 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 $\boldsymbol{w}^{(t)}$ as

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

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

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

Mistake-Driven Learning

- Consider the "mistake-driven" SGD update rule

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

- 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 $\boldsymbol{w}^{(t)}$ as

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

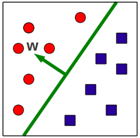
- For $\eta_t = 0.5$, this is akin to the Perceptron (a hyperplane based learning algo)

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

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

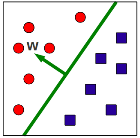
The Perceptron Algorithm

- One of the earliest algorithms for binary classification (Rosenblatt, 1958)
- Learns a linear hyperplane to separate the two classes



The Perceptron Algorithm

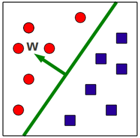
- One of the earliest algorithms for binary classification (Rosenblatt, 1958)
- Learns a linear hyperplane to separate the two classes



- A very simple, mistake-driven Online Learning algorithm. Learns using one example at a time. Also highly scalable for large training data sets (like SGD based logistic regression and other SGD based learning algorithms)

The Perceptron Algorithm

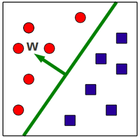
- One of the earliest algorithms for binary classification (Rosenblatt, 1958)
- Learns a linear hyperplane to separate the two classes



- A very simple, mistake-driven Online Learning algorithm. Learns using one example at a time. Also highly scalable for large training data sets (like SGD based logistic regression and other SGD based learning algorithms)
- Guaranteed to find a separating hyperplane if the data is *linearly separable*

The Perceptron Algorithm

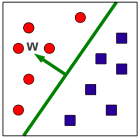
- One of the earliest algorithms for binary classification (Rosenblatt, 1958)
- Learns a linear hyperplane to separate the two classes



- A very simple, mistake-driven Online Learning algorithm. Learns using one example at a time. Also highly scalable for large training data sets (like SGD based logistic regression and other SGD based learning algorithms)
- Guaranteed to find a separating hyperplane if the data is *linearly separable*
- If data not linearly separable
  - First make it linearly separable (more when we discuss Kernel Methods)

The Perceptron Algorithm

- One of the earliest algorithms for binary classification (Rosenblatt, 1958)
- Learns a linear hyperplane to separate the two classes

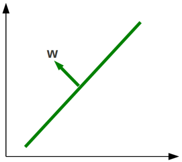


- A very simple, mistake-driven Online Learning algorithm. Learns using one example at a time. Also highly scalable for large training data sets (like SGD based logistic regression and other SGD based learning algorithms)
- Guaranteed to find a separating hyperplane if the data is *linearly separable*
- If data not linearly separable
  - First make it linearly separable (more when we discuss Kernel Methods)
  - .. or use multi-layer Perceptrons (more when we discuss Deep Learning)

Hyperplanes and Margins

Hyperplanes

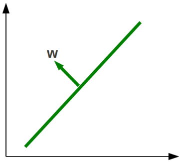
- Separates a D-dimensional space into two **half-spaces** (positive and negative)



- Defined by normal vector $\boldsymbol{w} \in \mathbb{R}^D$ (pointing towards positive half-space)

Hyperplanes

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

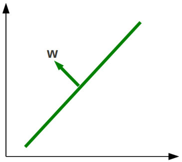


- Defined by normal vector $\boldsymbol{w} \in \mathbb{R}^D$ (pointing towards positive half-space)
- $\boldsymbol{w}$ is orthogonal to any vector $\boldsymbol{x}$ lying on the hyperplane

$$\boldsymbol{w}^\top \boldsymbol{x} = 0 \quad \text{(equation of the hyperplane)}$$

Hyperplanes

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



- Defined by normal vector $\boldsymbol{w} \in \mathbb{R}^D$ (pointing towards positive half-space)
- $\boldsymbol{w}$ is orthogonal to any vector $\boldsymbol{x}$ lying on the hyperplane

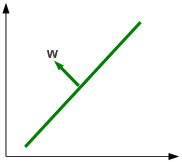
$$\boldsymbol{w}^\top \boldsymbol{x} = 0 \quad \text{(equation of the hyperplane)}$$

- Assumption: The hyperplane passes through origin. If not, add a bias $b \in \mathbb{R}$

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$$

Hyperplanes

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



- Defined by normal vector $\boldsymbol{w} \in \mathbb{R}^D$ (pointing towards positive half-space)
- $\boldsymbol{w}$ is orthogonal to any vector $\boldsymbol{x}$ lying on the hyperplane

$$\boldsymbol{w}^\top \boldsymbol{x} = 0 \quad \text{(equation of the hyperplane)}$$

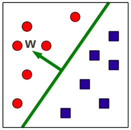
- Assumption: The hyperplane passes through origin. If not, add a bias $b \in \mathbb{R}$

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$$

- $b > 0$ means moving it parallely along $\boldsymbol{w}$ ($b < 0$ means in opposite direction)

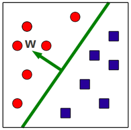
Hyperplane based Linear Classification

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



Hyperplane based Linear Classification

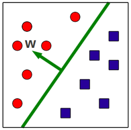
- Represents the decision boundary by a hyperplane $\boldsymbol{w} \in \mathbb{R}^D$



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

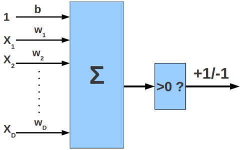
Hyperplane based Linear Classification

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



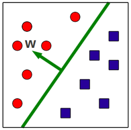
- For binary classification, $\boldsymbol{w}$ is assumed to point *towards* the positive class

- Classification rule: $y = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$
  - $\boldsymbol{w}^T\boldsymbol{x} + b > 0 \Rightarrow y = +1$
  - $\boldsymbol{w}^T\boldsymbol{x} + b < 0 \Rightarrow y = -1$

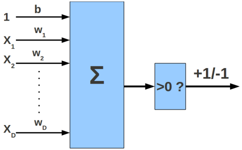


Hyperplane based Linear Classification

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



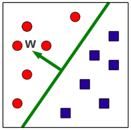
- For binary classification, $\boldsymbol{w}$ is assumed to point *towards* the positive class
- Classification rule: $y = \text{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$
  - $\boldsymbol{w}^T \boldsymbol{x} + b > 0 \Rightarrow y = +1$
  - $\boldsymbol{w}^T \boldsymbol{x} + b < 0 \Rightarrow y = -1$



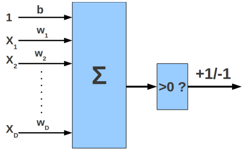
- **Note:** $y(\boldsymbol{w}^T \boldsymbol{x} + b) < 0$ mean a mistake on training example $(\boldsymbol{x}, y)$

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: $y = \text{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$
  - $\boldsymbol{w}^T \boldsymbol{x} + b > 0 \Rightarrow y = +1$
  - $\boldsymbol{w}^T \boldsymbol{x} + b < 0 \Rightarrow y = -1$

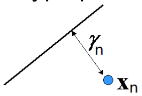


- **Note:** $y(\boldsymbol{w}^T \boldsymbol{x} + b) < 0$ mean a mistake on training example $(\boldsymbol{x}, y)$
- **Note:** Some algorithms that we have already seen (e.g., "distance from means", logistic regression, etc.) can also be viewed as learning hyperplanes

Notion of Margins

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

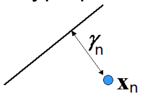
$$\gamma_n = \frac{\boldsymbol{w}^T \boldsymbol{x}_n + b}{||\boldsymbol{w}||}$$



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

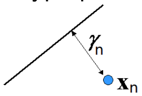
- Margin of a set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ w.r.t. $\boldsymbol{w}$ is the *min. abs. geometric margin*

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

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*

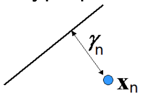
$$\gamma = \min_{1 \leq n \leq N} |\gamma_n| = \min_{1 \leq n \leq N} \frac{|(\boldsymbol{w}^T \boldsymbol{x}_n + b)|}{||\boldsymbol{w}||}$$

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

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*

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

- 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

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

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

A Loss Function for Hyperplane based Classification

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

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

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

- Let's perform stochastic optimization on this loss. Stochastic (sub-)gradients are

$$\frac{\partial \ell_n(\boldsymbol{w}, b)}{\partial \boldsymbol{w}} = -y_n \boldsymbol{x}_n$$

$$\frac{\partial \ell_n(\boldsymbol{w}, b)}{\partial b} = -y_n$$

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

A Loss Function for Hyperplane based Classification

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

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

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

- Let's perform stochastic optimization on this loss. Stochastic (sub-)gradients are

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

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

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

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

A Loss Function for Hyperplane based Classification

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

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

- Seems natural: if $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 0$, then the loss on $(\boldsymbol{x}_n, y_n)$ will be 0; otherwise the model will incur some positive loss when $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) < 0$
- Let's perform stochastic optimization on this loss. Stochastic (sub-)gradients are

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

  (when $\boldsymbol{w}$ makes a mistake, and are zero otherwise)
- Upon every mistake, update rule for $\boldsymbol{w}$ and b (assuming learning rate $= 1$)

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

- 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)

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

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

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

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

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)

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

Why Perceptron Updates Work?

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} < 0$

Why Perceptron Updates Work?

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} < 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} + \boldsymbol{x}_n$ (since $y_n = +1$)

Why Perceptron Updates Work?

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} < 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} + \boldsymbol{x}_n$ (since $y_n = +1$)
  - $b_{new} = b_{old} + y_n = b_{old} + 1$

Why Perceptron Updates Work?

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} < 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} + \boldsymbol{x}_n$ (since $y_n = +1$)
  - $b_{new} = b_{old} + y_n = b_{old} + 1$

$$\boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} = (\boldsymbol{w}_{old} + \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} + 1$$

Why Perceptron Updates Work?

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} < 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} + \boldsymbol{x}_n$ (since $y_n = +1$)
  - $b_{new} = b_{old} + y_n = b_{old} + 1$

$$\begin{aligned} \boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} &= (\boldsymbol{w}_{old} + \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} + 1 \\ &= (\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}) + \boldsymbol{x}_n^T \boldsymbol{x}_n + 1 \end{aligned}$$

Why Perceptron Updates Work?

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} < 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} + \boldsymbol{x}_n$ (since $y_n = +1$)
  - $b_{new} = b_{old} + y_n = b_{old} + 1$

$$\begin{aligned} \boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} &= (\boldsymbol{w}_{old} + \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} + 1 \\ &= (\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}) + \boldsymbol{x}_n^T \boldsymbol{x}_n + 1 \end{aligned}$$

- Thus $\boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new}$ is less negative than $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}$

Why Perceptron Updates Work?

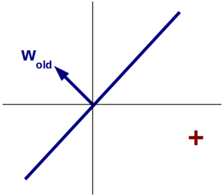
- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} < 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} + \boldsymbol{x}_n$ (since $y_n = +1$)
  - $b_{new} = b_{old} + y_n = b_{old} + 1$

$$
\begin{aligned}
\boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} &= (\boldsymbol{w}_{old} + \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} + 1 \\
&= (\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}) + \boldsymbol{x}_n^T \boldsymbol{x}_n + 1
\end{aligned}
$$

- Thus $\boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new}$ is less negative than $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}$
  - So we are making ourselves more correct on this example!

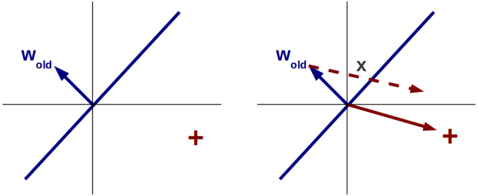
Why Perceptron Updates Work (Pictorially)?

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + \boldsymbol{x}$$



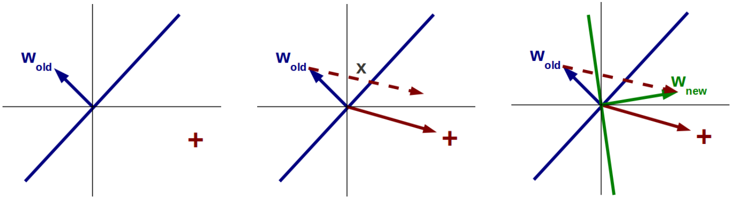
Why Perceptron Updates Work (Pictorially)?

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + \boldsymbol{x}$$



Why Perceptron Updates Work (Pictorially)?

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + \boldsymbol{x}$$



Why Perceptron Updates Work?

- Now let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} > 0$

Why Perceptron Updates Work?

- Now let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} > 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} - \boldsymbol{x}_n$ (since $y_n = -1$)

Why Perceptron Updates Work?

- Now let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} > 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} - \boldsymbol{x}_n$ (since $y_n = -1$)
  - $b_{new} = b_{old} + y_n = b_{old} - 1$

Why Perceptron Updates Work?

- Now let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} > 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} - \boldsymbol{x}_n$ (since $y_n = -1$)
  - $b_{new} = b_{old} + y_n = b_{old} - 1$

$$\boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} = (\boldsymbol{w}_{old} - \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} - 1$$

Why Perceptron Updates Work?

- Now let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} > 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} - \boldsymbol{x}_n$ (since $y_n = -1$)
  - $b_{new} = b_{old} + y_n = b_{old} - 1$

$$\begin{aligned} \boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} &= (\boldsymbol{w}_{old} - \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} - 1 \\ &= (\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}) - \boldsymbol{x}_n^T \boldsymbol{x}_n - 1 \end{aligned}$$

Why Perceptron Updates Work?

- Now let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} > 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} - \boldsymbol{x}_n$ (since $y_n = -1$)
  - $b_{new} = b_{old} + y_n = b_{old} - 1$

$$\begin{aligned} \boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} &= (\boldsymbol{w}_{old} - \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} - 1 \\ &= (\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}) - \boldsymbol{x}_n^T \boldsymbol{x}_n - 1 \end{aligned}$$

- Thus $\boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new}$ is less positive than $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}$

Why Perceptron Updates Work?

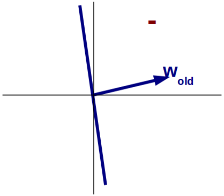
- Now let's look at a misclassified negative example ($y_n = -1$)
  - Perceptron (wrongly) predicts that $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old} > 0$
- Updates would be
  - $\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + y_n \boldsymbol{x}_n = \boldsymbol{w}_{old} - \boldsymbol{x}_n$ (since $y_n = -1$)
  - $b_{new} = b_{old} + y_n = b_{old} - 1$

$$\begin{aligned} \boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new} &= (\boldsymbol{w}_{old} - \boldsymbol{x}_n)^T \boldsymbol{x}_n + b_{old} - 1 \\ &= (\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}) - \boldsymbol{x}_n^T \boldsymbol{x}_n - 1 \end{aligned}$$

- Thus $\boldsymbol{w}_{new}^T \boldsymbol{x}_n + b_{new}$ is less positive than $\boldsymbol{w}_{old}^T \boldsymbol{x}_n + b_{old}$
  - So we are making ourselves more correct on this example!

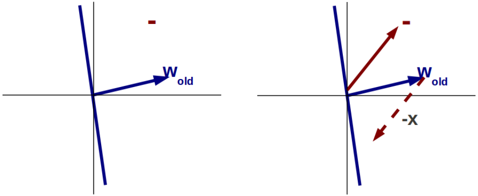
Why Perceptron Updates Work (Pictorially)?

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \mathbf{x}$$



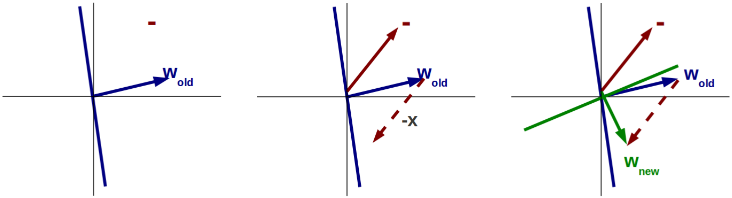
Why Perceptron Updates Work (Pictorially)?

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \boldsymbol{x}$$



Why Perceptron Updates Work (Pictorially)?

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \boldsymbol{x}$$



Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)
- $||\boldsymbol{w}_{k+1}||^2$

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)
- $||\boldsymbol{w}_{k+1}||^2 = ||\boldsymbol{w}_k||^2 + 2y_n \boldsymbol{w}_k^T \boldsymbol{x}_n + ||\boldsymbol{x}||^2$

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)
- $||\boldsymbol{w}_{k+1}||^2 = ||\boldsymbol{w}_k||^2 + 2y_n \boldsymbol{w}_k^T \boldsymbol{x}_n + ||\boldsymbol{x}||^2 \leq ||\boldsymbol{w}_k||^2 + R^2$ (since $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$)

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)
- $||\boldsymbol{w}_{k+1}||^2 = ||\boldsymbol{w}_k||^2 + 2y_n \boldsymbol{w}_k^T \boldsymbol{x}_n + ||\boldsymbol{x}||^2 \leq ||\boldsymbol{w}_k||^2 + R^2$ (since $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$)
- Repeating iteratively k times, we get $||\boldsymbol{w}_{k+1}||^2 \leq kR^2$ (2)

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)
- $||\boldsymbol{w}_{k+1}||^2 = ||\boldsymbol{w}_k||^2 + 2y_n \boldsymbol{w}_k^T \boldsymbol{x}_n + ||\boldsymbol{x}||^2 \leq ||\boldsymbol{w}_k||^2 + R^2$ (since $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$)
- Repeating iteratively k times, we get $||\boldsymbol{w}_{k+1}||^2 \leq kR^2$ (2)
- Using (1), (2), and $||\boldsymbol{w}_*|| = 1$, we get $k\gamma < \boldsymbol{w}_{k+1}^T \boldsymbol{w}_* \leq ||\boldsymbol{w}_{k+1}|| \leq R\sqrt{k}$

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)
- $||\boldsymbol{w}_{k+1}||^2 = ||\boldsymbol{w}_k||^2 + 2y_n \boldsymbol{w}_k^T \boldsymbol{x}_n + ||\boldsymbol{x}||^2 \leq ||\boldsymbol{w}_k||^2 + R^2$ (since $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$)
- Repeating iteratively k times, we get $||\boldsymbol{w}_{k+1}||^2 \leq kR^2$ (2)
- Using (1), (2), and $||\boldsymbol{w}_*|| = 1$, we get $k\gamma < \boldsymbol{w}_{k+1}^T \boldsymbol{w}_* \leq ||\boldsymbol{w}_{k+1}|| \leq R\sqrt{k}$

$$\boxed{k \leq R^2/\gamma^2}$$

Convergence of Perceptron

Theorem (Block & Novikoff): If the training data is linearly separable with margin γ by a unit norm hyperplane $\boldsymbol{w}_*$ ($||\boldsymbol{w}_*|| = 1$) with $b = 0$, then perceptron converges after R^2/γ^2 mistakes during training (assuming $||\boldsymbol{x}|| < R$ for all $\boldsymbol{x}$).

Proof:

- Margin of $\boldsymbol{w}_*$ on any *arbitrary example* $(\boldsymbol{x}_n, y_n)$: $\frac{y_n \boldsymbol{w}_*^T \boldsymbol{x}_n}{||\boldsymbol{w}_*||} = y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$, and update $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + y_n \boldsymbol{x}_n$
- $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* = \boldsymbol{w}_k^T \boldsymbol{w}_* + y_n \boldsymbol{w}_*^T \boldsymbol{x}_n \geq \boldsymbol{w}_k^T \boldsymbol{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\boldsymbol{w}_{k+1}^T \boldsymbol{w}_* > k\gamma$ (1)
- $||\boldsymbol{w}_{k+1}||^2 = ||\boldsymbol{w}_k||^2 + 2y_n \boldsymbol{w}_k^T \boldsymbol{x}_n + ||\boldsymbol{x}||^2 \leq ||\boldsymbol{w}_k||^2 + R^2$ (since $y_n \boldsymbol{w}_k^T \boldsymbol{x}_n \leq 0$)
- Repeating iteratively k times, we get $||\boldsymbol{w}_{k+1}||^2 \leq kR^2$ (2)
- Using (1), (2), and $||\boldsymbol{w}_*|| = 1$, we get $k\gamma < \boldsymbol{w}_{k+1}^T \boldsymbol{w}_* \leq ||\boldsymbol{w}_{k+1}|| \leq R\sqrt{k}$

$$\boxed{k \leq R^2/\gamma^2}$$

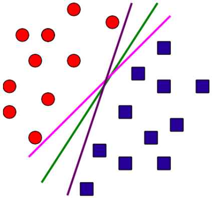
Nice Thing: Convergence rate does not depend on the number of training examples N or the data dimensionality D. Depends only on the margin!!!

The Best Hyperplane Separator?

- Perceptron finds one of the many possible hyperplanes separating the data
  - .. if one exists

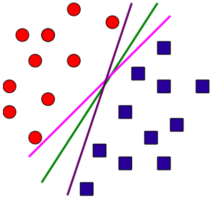
The Best Hyperplane Separator?

- Perceptron finds one of the many possible hyperplanes separating the data
  - .. if one exists
- Of the many possible choices, which one is the best?



The Best Hyperplane Separator?

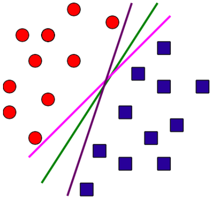
- Perceptron finds one of the many possible hyperplanes separating the data
  - .. if one exists
- Of the many possible choices, which one is the best?



- Intuitively, we want the hyperplane having the maximum margin

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)

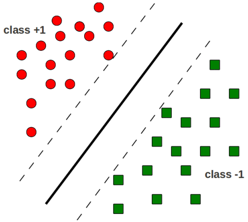
- Probably the most popular/influential classification algorithm
- Backed by solid theoretical groundings (Vapnik and Cortes, 1995)

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)

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

Support Vector Machine

- A hyperplane based linear classifier defined by $\boldsymbol{w}$ and b
- Prediction rule: $y = sign(\boldsymbol{w}^T\boldsymbol{x} + b)$

Support Vector Machine

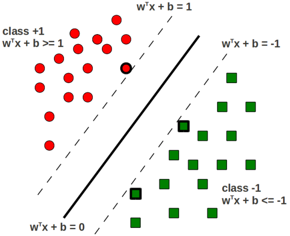
- A hyperplane based linear classifier defined by $\boldsymbol{w}$ and b
- Prediction rule: $y = sign(\boldsymbol{w}^T \boldsymbol{x} + b)$
- **Given:** Training data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
- **Goal:** Learn $\boldsymbol{w}$ and b that achieve the maximum margin

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)

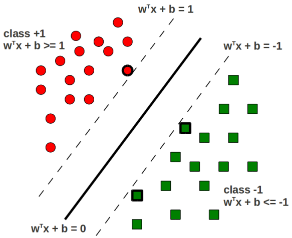
Support Vector Machine

- 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)



Support Vector Machine

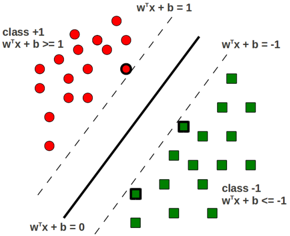
- A hyperplane based linear classifier defined by $\boldsymbol{w}$ and b
- Prediction rule: $y = sign(\boldsymbol{w}^T\boldsymbol{x} + b)$
- **Given:** Training data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
- **Goal:** Learn $\boldsymbol{w}$ and b that achieve the maximum margin
- 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 $y_n = +1$

Support Vector Machine

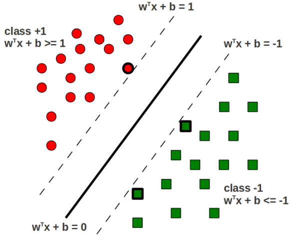
- A hyperplane based linear classifier defined by $\boldsymbol{w}$ and b
- Prediction rule: $y = sign(\boldsymbol{w}^T \boldsymbol{x} + b)$
- **Given:** Training data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
- **Goal:** Learn $\boldsymbol{w}$ and b that achieve the maximum margin
- 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 $y_n = -1$

Support Vector Machine

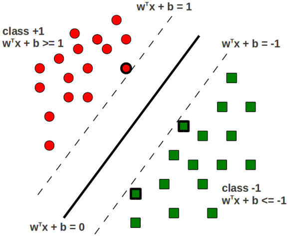
- A hyperplane based linear classifier defined by $\boldsymbol{w}$ and b
- Prediction rule: $y = sign(\boldsymbol{w}^T\boldsymbol{x} + b)$
- **Given:** Training data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
- **Goal:** Learn $\boldsymbol{w}$ and b that achieve the maximum margin
- 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, $y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1$

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)

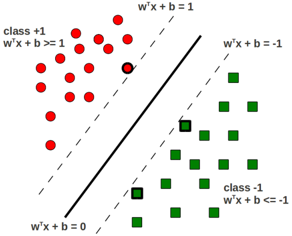


- Assume the hyperplane is such that
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b \geq 1$ for $y_n = +1$
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b \leq -1$ for $y_n = -1$
  - Equivalently, $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$
    $\Rightarrow \min_{1 \leq n \leq N} |\boldsymbol{w}^T \boldsymbol{x}_n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

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

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)

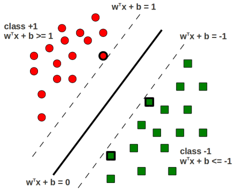


- Assume the hyperplane is such that
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b \geq 1$ for $y_n = +1$
  - $\boldsymbol{w}^T \boldsymbol{x}_n + b \leq -1$ for $y_n = -1$
  - Equivalently, $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$
    $\Rightarrow \min_{1 \leq n \leq N} |\boldsymbol{w}^T \boldsymbol{x}_n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

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

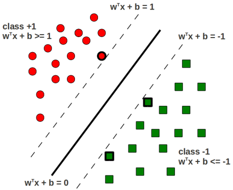
Support Vector Machine: The Optimization Problem

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



Support Vector Machine: The Optimization Problem

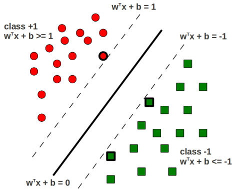
- We want to maximize the margin $\gamma = \frac{1}{||\boldsymbol{w}||}$



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

Support Vector Machine: The Optimization Problem

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

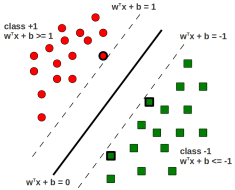


- Maximizing the margin γ = minimizing $||\boldsymbol{w}||$ (the norm)
- Our optimization problem would be:

$$\text{Minimize} \quad f(\boldsymbol{w}, b) = \frac{||\boldsymbol{w}||^2}{2}$$
$$\text{subject to} \quad y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1, \qquad n = 1, \dots, N$$

Support Vector Machine: The Optimization Problem

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



- Maximizing the margin γ = minimizing $||\boldsymbol{w}||$ (the norm)
- Our optimization problem would be:

$$\text{Minimize} \quad f(\boldsymbol{w}, b) = \frac{||\boldsymbol{w}||^2}{2}$$

$$\text{subject to} \quad y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

- 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

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 $||\boldsymbol{w}||$

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

Next class..

- Solving the SVM optimization problem
- Introduction to kernel methods (nonlinear SVMs)