Hyperplane based Classification: Perceptron and (Intro to) Support Vector Machines

Piyush Rai

CS5350/6350: Machine Learning

September 8, 2011
Hyperplane

- Separates a $D$-dimensional space into two half-spaces

- Defined by an outward pointing normal vector $\mathbf{w} \in \mathbb{R}^D$
Hyperplane

- Separates a $D$-dimensional space into two half-spaces

- Defined by an outward pointing normal vector $\mathbf{w} \in \mathbb{R}^D$

- $\mathbf{w}$ is orthogonal to any vector lying on the hyperplane
Hyperplane

- Separates a $D$-dimensional space into two half-spaces

- Defined by an outward pointing normal vector $\mathbf{w} \in \mathbb{R}^D$

- $\mathbf{w}$ is orthogonal to any vector lying on the hyperplane

- Assumption: The hyperplane passes through origin.
Separates a $D$-dimensional space into two half-spaces

Defined by an outward pointing normal vector $\mathbf{w} \in \mathbb{R}^D$

$\mathbf{w}$ is orthogonal to any vector lying on the hyperplane

Assumption: The hyperplane passes through origin. If not,
  - have a *bias* term $b$; we will then need both $\mathbf{w}$ and $b$ to define it
  - $b > 0$ means moving it parallelly along $\mathbf{w}$ ($b < 0$ means in opposite direction)
Linear Classifiers: Represent the decision boundary by a hyperplane $\mathbf{w}$

For binary classification, $\mathbf{w}$ is assumed to point *towards* the positive class.
Linear Classifiers: Represent the decision boundary by a hyperplane $\mathbf{w}$

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

Classification rule:

$$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) = \text{sign}(\sum_{j=1}^{D} w_j x_j + b)$$
Linear Classification via Hyperplanes

- Linear Classifiers: Represent the decision boundary by a hyperplane $\mathbf{w}$

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

- Classification rule:
  
  $$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) = \text{sign}\left(\sum_{j=1}^{D} w_j x_j + b\right)$$

  - $\mathbf{w}^T \mathbf{x} + b > 0 \Rightarrow y = +1$
  - $\mathbf{w}^T \mathbf{x} + b < 0 \Rightarrow y = -1$
Linear Classification via Hyperplanes

- Linear Classifiers: Represent the decision boundary by a hyperplane \( \mathbf{w} \)

- For binary classification, \( \mathbf{w} \) is assumed to point *towards* the positive class

- Classification rule:
  \[
  y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) = \text{sign}(\sum_{j=1}^{D} w_j x_j + b)
  \]
  - \( \mathbf{w}^T \mathbf{x} + b > 0 \Rightarrow y = +1 
  
  - \( \mathbf{w}^T \mathbf{x} + b < 0 \Rightarrow y = -1 
  
(Hyperplane based Classification)
Linear Classification via Hyperplanes

- Linear Classifiers: Represent the decision boundary by a hyperplane $\mathbf{w}$

  ![Hyperplane diagram]

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

- Classification rule:
  $$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) = \text{sign}(\sum_{j=1}^{D} w_j x_j + b)$$

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

- **Question:** What about the points $\mathbf{x}$ for which $\mathbf{w}^T \mathbf{x} + b = 0$?
Linear Classification via Hyperplanes

- Linear Classifiers: Represent the decision boundary by a hyperplane $\mathbf{w}$

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

- Classification rule:
  $$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) = \text{sign}(\sum_{j=1}^{D} w_j x_j + b)$$
  - $\mathbf{w}^T \mathbf{x} + b > 0 \Rightarrow y = +1$
  - $\mathbf{w}^T \mathbf{x} + b < 0 \Rightarrow y = -1$

- **Question:** What about the points $\mathbf{x}$ for which $\mathbf{w}^T \mathbf{x} + b = 0$?
- **Goal:** To learn the hyperplane $(\mathbf{w}, b)$ using the training data
Concept of Margins

- **Geometric margin** $\gamma_n$ of an example $x_n$ is its distance from the hyperplane

$$
\gamma_n = \frac{w^T x_n + b}{||w||}
$$
Concept of Margins

- **Geometric margin** $\gamma_n$ of an example $x_n$ is its distance from the hyperplane

  $$
  \gamma_n = \frac{w^T x_n + b}{||w||}
  $$

- Geometric margin may be positive (if $y_n = +1$) or negative (if $y_n = -1$)

- **Margin** of a set $\{x_1, \ldots, x_N\}$ is the *minimum absolute geometric margin*

  $$
  \gamma = \min_{1 \leq n \leq N} |\gamma_n| = \min_{1 \leq n \leq N} \frac{|w^T x_n + b|}{||w||}
  $$
Concept of Margins

- **Geometric margin** $\gamma_n$ of an example $x_n$ is its distance from the hyperplane
  \[ \gamma_n = \frac{w^T x_n + b}{||w||} \]

- Geometric margin may be positive (if $y_n = +1$) or negative (if $y_n = -1$)

- **Margin** of a set $\{x_1, \ldots, x_N\}$ is the *minimum absolute geometric margin*
  \[
  \gamma = \min_{1 \leq n \leq N} |\gamma_n| = \min_{1 \leq n \leq N} \left| \frac{(w^T x_n + b)}{||w||} \right|
  \]

- **Functional margin** of a training example: $y_n(w^T x_n + b)$
Concept of Margins

- **Geometric margin** $\gamma_n$ of an example $x_n$ is its distance from the hyperplane
  \[
  \gamma_n = \frac{w^T x_n + b}{||w||}
  \]

- Geometric margin may be positive (if $y_n = +1$) or negative (if $y_n = -1$)

- **Margin** of a set $\{x_1, \ldots, x_N\}$ is the minimum absolute geometric margin
  \[
  \gamma = \min_{1 \leq n \leq N} |\gamma_n| = \min_{1 \leq n \leq N} \frac{|(w^T x_n + b)|}{||w||}
  \]

- **Functional margin** of a training example: $y_n(w^T x_n + b)$
  - Positive if prediction is correct; Negative if prediction is incorrect
Concept of Margins

- **Geometric margin** $\gamma_n$ of an example $x_n$ is its distance from the hyperplane
  \[ \gamma_n = \frac{w^T x_n + b}{||w||} \]

- Geometric margin may be positive (if $y_n = +1$) or negative (if $y_n = -1$)

- **Margin** of a set $\{x_1, \ldots, x_N\}$ is the minimum absolute geometric margin
  \[ \gamma = \min_{1 \leq n \leq N} |\gamma_n| = \min_{1 \leq n \leq N} \frac{|(w^T x_n + b)|}{||w||} \]

- **Functional margin** of a training example: $y_n(w^T x_n + b)$
  - Positive if prediction is correct; Negative if prediction is incorrect

- **Absolute** value of the functional margin = confidence in the predicted label
  - ..or “mis-confidence” if prediction is wrong
Concept of Margins

- **Geometric margin** $\gamma_n$ of an example $x_n$ is its distance from the hyperplane
  \[
  \gamma_n = \frac{w^T x_n + b}{||w||}
  \]

- Geometric margin may be positive (if $y_n = +1$) or negative (if $y_n = -1$)

- **Margin** of a set $\{x_1, \ldots, x_N\}$ is the *minimum absolute geometric margin*
  \[
  \gamma = \min_{1 \leq n \leq N} |\gamma_n| = \min_{1 \leq n \leq N} \frac{|(w^T x_n + b)|}{||w||}
  \]

- **Functional margin** of a training example: $y_n(w^T x_n + b)$
  - Positive if prediction is correct; **Negative** if prediction is incorrect

- **Absolute** value of the functional margin = **confidence** in the predicted label
  - ..or “mis-confidence” if prediction is wrong
  - large margin $\Rightarrow$ high confidence
The Perceptron Algorithm

- One of the earliest algorithms for linear classification (Rosenblatt, 1958)
- Based on finding a separating hyperplane of the data
The Perceptron Algorithm

- One of the earliest algorithms for linear classification (Rosenblatt, 1958)
- Based on finding a separating hyperplane of the data
- Guaranteed to find a separating hyperplane if the data is \textit{linearly separable}
The Perceptron Algorithm

- One of the earliest algorithms for linear classification (Rosenblatt, 1958)
- Based on finding a separating hyperplane of the data
- Guaranteed to find a separating hyperplane if the data is *linearly separable*

![Diagram of Perceptron Algorithm]

- If data not linear separable
  - Make it linearly separable (more on this when we cover **Kernel Methods**)

(CS5350/6350) Hyperplane based Classification September 8, 2011
The Perceptron Algorithm

- One of the earliest algorithms for linear classification (Rosenblatt, 1958)
- Based on finding a separating hyperplane of the data
- Guaranteed to find a separating hyperplane if the data is *linearly separable*

If data not linear separable

- Make it linearly separable (more on this when we cover **Kernel Methods**)
- .. or use a *combination* of multiple perceptrons (Neural Networks)
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an online algorithm)
The Perceptron Algorithm

- Cycles through the training data by
  - processing training examples *one at a time* (an *online* algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]; b = 0\))
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an online algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]; \ b = 0\))

- An iterative **mistake-driven** learning algorithm for updating \((w, b)\)
The Perceptron Algorithm

- Cycles through the training data by
  - processing training examples *one at a time* (an online algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]; \ b = 0\))

- An iterative mistake-driven learning algorithm for updating \((w, b)\)
  - Don’t update if \(w\) correctly predicts the label of the current training example
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an online algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]; b = 0\))

- An iterative mistake-driven learning algorithm for updating \((w, b)\)
  - Don’t update if \(w\) correctly predicts the label of the current training example
  - Update \(w\) when it mispredicts the label of the current training example
The Perceptron Algorithm

- Cycles through the training data by
  - processing training examples *one at a time* (an *online* algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]\); \(b = 0\))

- An iterative *mistake-driven* learning algorithm for updating \((w, b)\)
  - *Don’t update* if \(w\) correctly predicts the label of the current training example
  - *Update* \(w\) when it *mispredicts* the label of the current training example
    - *True* label is +1, but \(\text{sign}(w^T x + b) = -1\) (or vice-versa)
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an online algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]; b = 0\))

- An iterative mistake-driven learning algorithm for updating \((w, b)\)
  - Don’t update if \(w\) correctly predicts the label of the current training example
  - Update \(w\) when it mispredicts the label of the current training example
    - *True* label is +1, but \(\text{sign}(w^T x + b) = -1\) (or vice-versa)

- Repeat until convergence
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an *online* algorithm)

- Starts with some initialization for \((\mathbf{w}, b)\) (e.g., \(\mathbf{w} = [0, \ldots, 0]; \ b = 0\))

- An iterative *mistake-driven* learning algorithm for updating \((\mathbf{w}, b)\)
  - Don’t update if \(\mathbf{w}\) correctly predicts the label of the current training example
  - Update \(\mathbf{w}\) when it mispredicts the label of the current training example
    - *True* label is +1, but \(\text{sign}(\mathbf{w}^T \mathbf{x} + b) = -1\) (or vice-versa)

- Repeat until convergence

- **Batch vs Online learning algorithms:**
The Perceptron Algorithm

- Cycles through the training data by processing training examples one at a time (an online algorithm)

- Starts with some initialization for $(w, b)$ (e.g., $w = [0, \ldots, 0]; b = 0$)

- An iterative mistake-driven learning algorithm for updating $(w, b)$
  - Don’t update if $w$ correctly predicts the label of the current training example
  - Update $w$ when it mispredicts the label of the current training example
    - True label is $+1$, but $\text{sign}(w^T x + b) = -1$ (or vice-versa)

- Repeat until convergence

- Batch vs Online learning algorithms:
  - Batch algorithms operate on the entire training data
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an online algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]; b = 0\))

- An iterative mistake-driven learning algorithm for updating \((w, b)\)
  - Don’t update if \(w\) correctly predicts the label of the current training example
  - Update \(w\) when it mispredicts the label of the current training example
    - *True* label is +1, but \(\text{sign}(w^T x + b) = -1\) (or vice-versa)

- Repeat until convergence

**Batch vs Online learning algorithms:**
- Batch algorithms operate on the entire training data
- Online algorithms can process one example at a time
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an online algorithm)

- Starts with some initialization for \((w, b)\) (e.g., \(w = [0, \ldots, 0]; b = 0\))

- An iterative mistake-driven learning algorithm for updating \((w, b)\)
  - Don’t update if \(w\) correctly predicts the label of the current training example
  - Update \(w\) when it mispredicts the label of the current training example
    - True label is +1, but \(\text{sign}(w^T x + b) = -1\) (or vice-versa)

- Repeat until convergence

**Batch vs Online learning algorithms:**
- Batch algorithms operate on the entire training data
- Online algorithms can process one example at a time
  - Usually *more efficient* (computationally, memory-footprint-wise) than batch
The Perceptron Algorithm

- Cycles through the training data by processing training examples *one at a time* (an online algorithm).

- Starts with some initialization for \((\mathbf{w}, b)\) (e.g., \(\mathbf{w} = [0, \ldots, 0]; \ b = 0\)).

- An iterative mistake-driven learning algorithm for updating \((\mathbf{w}, b)\):
  - Don’t update if \(\mathbf{w}\) correctly predicts the label of the current training example.
  - Update \(\mathbf{w}\) when it mispredicts the label of the current training example.
    - *True* label is +1, but \(\text{sign}(\mathbf{w}^T \mathbf{x} + b) = -1\) (or vice-versa).

- Repeat until convergence.

**Batch vs Online learning algorithms:**
- Batch algorithms operate on the entire training data.
- Online algorithms can process one example at a time.
  - Usually *more efficient* (computationally, memory-footprint-wise) than batch.
  - Often batch problems can be solved using online learning!
The Perceptron Algorithm: Formally

- **Given:** Sequence of $N$ training examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- **Initialize:** $w = [0, \ldots, 0], b = 0$
- **Repeat until convergence:**
  - For $n = 1, \ldots, N$
    - if $\text{sign}(w^T x_n + b) \neq y_n$ (i.e., mistake is made)
      
        $$w = w + y_n x_n$$

        $$b = b + y_n$$
The Perceptron Algorithm: Formally

- Given: Sequence of $N$ training examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Initialize: $\mathbf{w} = [0, \ldots, 0]$, $b = 0$
- Repeat until convergence:
  - For $n = 1, \ldots, N$
    - if $\text{sign}(\mathbf{w}^T x_n + b) \neq y_n$ (i.e., mistake is made)
      - $\mathbf{w} = \mathbf{w} + y_n x_n$
      - $b = b + y_n$

- Stopping condition: stop when either
  - All training examples are classified correctly
The Perceptron Algorithm: Formally

- Given: Sequence of $N$ training examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Initialize: $w = [0, \ldots, 0], b = 0$
- Repeat until convergence:
  - For $n = 1, \ldots, N$
    - if $\text{sign}(w^T x_n + b) \neq y_n$ (i.e., mistake is made)
      - $w = w + y_n x_n$
      - $b = b + y_n$

- Stopping condition: stop when either
  - All training examples are classified correctly
    - May overfit, so less common in practice
The Perceptron Algorithm: Formally

- Given: Sequence of $N$ training examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Initialize: $\mathbf{w} = [0, \ldots, 0]$, $b = 0$
- Repeat until convergence:
  - For $n = 1, \ldots, N$
    - if $\text{sign}(\mathbf{w}^T x_n + b) \neq y_n$ (i.e., mistake is made)
      - $\mathbf{w} = \mathbf{w} + y_n x_n$
      - $b = b + y_n$

- 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: Formally

- Given: Sequence of $N$ training examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Initialize: $w = [0, \ldots, 0], b = 0$
- Repeat until convergence:
  - For $n = 1, \ldots, N$
    - if $\text{sign}(w^T x_n + b) \neq y_n$ (i.e., mistake is made)
      $$w = w + y_n x_n$$
      $$b = b + y_n$$

- 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: Formally

- Given: Sequence of \( N \) training examples \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
- Initialize: \( w = [0, \ldots, 0] \), \( b = 0 \)
- Repeat until convergence:
  - For \( n = 1, \ldots, N \)
    - if \( \text{sign}(w^T x_n + b) \neq y_n \) (i.e., mistake is made)
      \[
      w = w + y_n x_n \\
      b = b + y_n
      \]
- 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 (more passes just not possible)
The Perceptron Algorithm: Formally

- Given: Sequence of $N$ training examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Initialize: $w = [0, \ldots, 0]$, $b = 0$
- Repeat until convergence:
  - For $n = 1, \ldots, N$
    - if $\text{sign}(w^T x_n + b) \neq y_n$ (i.e., mistake is made)
      
      $\begin{align*}
      w &= w + y_n x_n \\
      b &= b + y_n
      \end{align*}$

- 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 (more passes just not possible)

- Note: $\text{sign}(w^T x_n + b) \neq y_n$ is equivalent to $y_n(w^T x_n + b) < 0$
Let's look at a misclassified positive example \((y_n = +1)\)

- Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} < 0\)
Let's look at a misclassified positive example \((y_n = +1)\)

- Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} < 0\)

Updates would be

- \(w_{new} = w_{old} + y_n x_n = w_{old} + x_n \) (since \(y_n = +1\))
Why Perceptron Updates Work?

Let’s look at a misclassified positive example \( (y_n = +1) \)

- Perceptron (wrongly) thinks \( w_{old}^T x_n + b_{old} < 0 \)

- Updates would be
  
  \[
  \begin{align*}
  w_{new} &= w_{old} + y_n x_n = w_{old} + x_n \quad \text{(since } y_n = +1) \\
  b_{new} &= b_{old} + y_n = b_{old} + 1
  \end{align*}
  \]
Why Perceptron Updates Work?

- Let's look at a misclassified positive example ($y_n = +1$)
  - Perceptron (wrongly) thinks $w_{old}^T x_n + b_{old} < 0$
  
- Updates would be
  - $w_{new} = w_{old} + y_n x_n = w_{old} + x_n$ (since $y_n = +1$)
  - $b_{new} = b_{old} + y_n = b_{old} + 1$
  
  $$w_{new}^T x_n + b_{new} = (w_{old} + x_n)^T x_n + b_{old} + 1$$
Why Perceptron Updates Work?

- Let's look at a misclassified positive example \((y_n = +1)\)
  - Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} < 0\)
  - Updates would be
    - \(w_{new} = w_{old} + y_n x_n = w_{old} + x_n\) (since \(y_n = +1\))
    - \(b_{new} = b_{old} + y_n = b_{old} + 1\)

\[
\begin{align*}
w_{new}^T x_n + b_{new} &= (w_{old} + x_n)^T x_n + b_{old} + 1 \\
&= (w_{old}^T x_n + b_{old}) + x_n^T x_n + 1
\end{align*}
\]
Why Perceptron Updates Work?

- Let's look at a misclassified positive example \((y_n = +1)\)
  - Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} < 0\)

- Updates would be
  - \(w_{new} = w_{old} + y_n x_n = w_{old} + x_n\) (since \(y_n = +1\))
  - \(b_{new} = b_{old} + y_n = b_{old} + 1\)

\[
\begin{align*}
w_{new}^T x_n + b_{new} &= (w_{old} + x_n)^T x_n + b_{old} + 1 \\
&= (w_{old}^T x_n + b_{old}) + x_n^T x_n + 1
\end{align*}
\]

- Thus \(w_{new}^T x_n + b_{new}\) is less negative than \(w_{old}^T x_n + b_{old}\)
Why Perceptron Updates Work?

- Let's look at a misclassified positive example \( (y_n = +1) \)
  - Perceptron (wrongly) thinks \( w_{old}^T x_n + b_{old} < 0 \)

- Updates would be
  - \( w_{new} = w_{old} + y_n x_n = w_{old} + x_n \) (since \( y_n = +1 \))
  - \( b_{new} = b_{old} + y_n = b_{old} + 1 \)

\[
\begin{align*}
  w_{new}^T x_n + b_{new} & = (w_{old} + x_n)^T x_n + b_{old} + 1 \\
  & = (w_{old}^T x_n + b_{old}) + x_n^T x_n + 1
\end{align*}
\]

- Thus \( w_{new}^T x_n + b_{new} \) is less negative than \( w_{old}^T x_n + b_{old} \)

- 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)?

\[ \mathbf{w}_{new} = \mathbf{w}_{old} + \mathbf{x} \]
Why Perceptron Updates Work (Pictorially)?

\[ \mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + \mathbf{x} \]
Now let’s look at a misclassified negative example \((y_n = -1)\)

- Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} > 0\)
Why Perceptron Updates Work?

- Now let's look at a misclassified negative example \((y_n = -1)\)
  - Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} > 0\)
  - Updates would be
    - \(\mathbf{w}_{new} = \mathbf{w}_{old} + y_n x_n = \mathbf{w}_{old} - x_n\) (since \(y_n = -1\))
Why Perceptron Updates Work?

Now let’s look at a misclassified negative example \((y_n = -1)\)

- Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} > 0\)

Updates would be

- \(w_{new} = w_{old} + y_n x_n = w_{old} - x_n\) (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) thinks \( w_{old}^T x_n + b_{old} > 0 \)
  - Updates would be
    - \( w_{new} = w_{old} + y_n x_n = w_{old} - x_n \) (since \( y_n = -1 \))
    - \( b_{new} = b_{old} + y_n = b_{old} - 1 \)
    \[
    w_{new}^T x_n + b_{new} = (w_{old} - x_n)^T x_n + b_{old} - 1
    \]
Now let's look at a misclassified negative example \((y_n = -1)\)

- Perceptron (wrongly) thinks \(w_{old}^T x_n + b_{old} > 0\)

Updates would be

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

\[
\begin{align*}
w_{new}^T x_n + b_{new} &= (w_{old} - x_n)^T x_n + b_{old} - 1 \\
&= (w_{old}^T x_n + b_{old}) - x_n^T x_n - 1
\end{align*}
\]
Now let’s look at a misclassified negative example ($y_n = -1$)

- Perceptron (wrongly) thinks $w_{old}^T x_n + b_{old} > 0$

Updates would be

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

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

Thus $w_{new}^T x_n + b_{new}$ is less positive than $w_{old}^T x_n + b_{old}$
Now let's look at a misclassified negative example \( (y_n = -1) \)

- Perceptron (wrongly) thinks \( w_{old}^T x_n + b_{old} > 0 \)

Updates would be

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

\[
\begin{align*}
w^T_{new} x_n + b_{new} & = (w_{old} - x_n)^T x_n + b_{old} - 1 \\
& = (w^T_{old} x_n + b_{old}) - x_n^T x_n - 1
\end{align*}
\]

Thus \( w^T_{new} x_n + b_{new} \) is less positive than \( w^T_{old} x_n + b_{old} \)

- 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)?

\[ \mathbf{w}_{new} = \mathbf{w}_{old} - \mathbf{x} \]
Why Perceptron Updates Work (Pictorially)?

\[ \mathbf{w}_{new} = \mathbf{w}_{old} - \mathbf{x} \]
Convergence of Perceptron

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

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

**Proof:**

- Margin of $w_*$ on any arbitrary example $(x_n, y_n)$: $\frac{y_n w_*^T x_n}{||w_*||} = y_n w_*^T x_n \geq \gamma$
Convergence of Perceptron

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

**Proof:**
- Margin of $w_*$ on any arbitrary example $(x_n, y_n)$: $y_n w_*^T x_n / ||w_*|| = y_n w_*^T x_n \geq \gamma$
- Consider the $(k + 1)^{th}$ mistake: $y_n w_k^T x_n \leq 0$, and update $w_{k+1} = w_k + y_n x_n$
Theorem (Block & Novikoff): If the training data is linearly separable with margin $\gamma$ by a unit norm hyperplane $\mathbf{w}_*$ ($||\mathbf{w}_*|| = 1$) with $b = 0$, then perceptron converges after $R^2/\gamma^2$ mistakes during training (assuming $||x|| < R$ for all $x$).

Proof:

- Margin of $\mathbf{w}_*$ on any arbitrary example $(x_n, y_n)$:
  \[ y_n \mathbf{w}_*^T x_n = y_n \mathbf{w}_*^T x_n \geq \gamma \]

- Consider the $(k + 1)^{th}$ mistake: $y_n \mathbf{w}_k^T x_n \leq 0$, and update $\mathbf{w}_{k+1} = \mathbf{w}_k + y_n \mathbf{x}_n$

- $\mathbf{w}_{k+1}^T \mathbf{w}_* = \mathbf{w}_k^T \mathbf{w}_* + y_n \mathbf{w}_*^T x_n \geq \mathbf{w}_k^T \mathbf{w}_* + \gamma$
Convergence of Perceptron

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

**Proof:**

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

- Consider the \((k + 1)^{th}\) mistake: \( y_n \mathbf{w}_k^T \mathbf{x}_n \leq 0 \), and update \( \mathbf{w}_{k+1} = \mathbf{w}_k + y_n \mathbf{x}_n \)

- \( \mathbf{w}_{k+1}^T \mathbf{w}_* = \mathbf{w}_k^T \mathbf{w}_* + y_n \mathbf{w}_*^T \mathbf{x}_n \geq \mathbf{w}_k^T \mathbf{w}_* + \gamma \) (**why is this nice?**)

Hyperplane based Classification

September 8, 2011 12 / 20
Convergence of Perceptron

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

**Proof:**

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

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

**Proof:**

- Margin of $w_*$ on any arbitrary example $(x_n, y_n)$: $y_n w_*^T x_n/\|w_*\| = y_n w_*^T x_n \geq \gamma$
- Consider the $(k + 1)^{th}$ mistake: $y_n w_k^T x_n \leq 0$, and update $w_{k+1} = w_k + y_n x_n$
- $w_{k+1}^T w_* = w_k^T w_* + y_n w_*^T x_n \geq w_k^T w_* + \gamma$ (why is this nice?)
- Repeating iteratively $k$ times, we get $w_{k+1}^T w_* > k\gamma$ (1)
- $\|w_{k+1}\|^2$
Theorem (Block & Novikoff): If the training data is linearly separable with margin $\gamma$ by a unit norm hyperplane $\mathbf{w}_*$ ($\|\mathbf{w}_*\| = 1$) with $b = 0$, then perceptron converges after $R^2/\gamma^2$ mistakes during training (assuming $\|\mathbf{x}\| < R$ for all $\mathbf{x}$).

Proof:

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

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

**Proof:**

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

- Consider the $(k + 1)^{th}$ mistake: $y_n\mathbf{w}_k^T \mathbf{x}_n \leq 0$, and update $\mathbf{w}_{k+1} = \mathbf{w}_k + y_n \mathbf{x}_n$

- $\mathbf{w}_{k+1}^T \mathbf{w}_* = \mathbf{w}_k^T \mathbf{w}_* + y_n \mathbf{w}_*^T \mathbf{x}_n \geq \mathbf{w}_k^T \mathbf{w}_* + \gamma$ (why is this nice?)

- Repeating iteratively $k$ times, we get $\mathbf{w}_{k+1}^T \mathbf{w}_* > k\gamma$ \hspace{1cm} (1)

- $||\mathbf{w}_{k+1}||^2 = ||\mathbf{w}_k||^2 + 2y_n\mathbf{w}_k^T \mathbf{x}_n + ||\mathbf{x}||^2 \leq ||\mathbf{w}_k||^2 + R^2$ (since $y_n\mathbf{w}_k^T \mathbf{x}_n \leq 0$)
Convergence of Perceptron

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

Proof:

- Margin of $w_*$ on any arbitrary example $(x_n, y_n)$: $y_n w_*^T x_n = y_n w_*^T x_n \geq \gamma$

- Consider the $(k + 1)^{th}$ mistake: $y_n w_k^T x_n \leq 0$, and update $w_{k+1} = w_k + y_n x_n$

- $w_{k+1}^T w_* = w_k^T w_* + y_n w_*^T x_n \geq w_k^T w_* + \gamma$ (why is this nice?)

- Repeating iteratively $k$ times, we get $w_{k+1}^T w_* > k\gamma$ (1)

- $||w_{k+1}||^2 = ||w_k||^2 + 2y_n w_k^T x_n + ||x||^2 \leq ||w_k||^2 + R^2$ (since $y_n w_k^T x_n \leq 0$)

- Repeating iteratively $k$ times, we get $||w_{k+1}||^2 \leq kR^2$ (2)
Convergence of Perceptron

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

**Proof:**

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

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

**Proof:**

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

- Consider the $(k+1)^{th}$ mistake: $y_n \mathbf{w}_k^T x_n \leq 0$, and update $\mathbf{w}_{k+1} = \mathbf{w}_k + y_n x_n$

- $\mathbf{w}_{k+1}^T \mathbf{w}_* = \mathbf{w}_k^T \mathbf{w}_* + y_n \mathbf{w}_*^T x_n \geq \mathbf{w}_k^T \mathbf{w}_* + \gamma$ (why is this nice?)

- Repeating iteratively $k$ times, we get $\mathbf{w}_{k+1}^T \mathbf{w}_* > k\gamma$ \hspace{1cm} (1)

- $\|\mathbf{w}_{k+1}\|^2 = \|\mathbf{w}_k\|^2 + 2y_n \mathbf{w}_k^T x_n + \|x\|^2 \leq \|\mathbf{w}_k\|^2 + R^2$ (since $y_n \mathbf{w}_k^T x_n \leq 0$)

- Repeating iteratively $k$ times, we get $\|\mathbf{w}_{k+1}\|^2 \leq kR^2$ \hspace{1cm} (2)

- Using (1), (2), and $\|\mathbf{w}_*\| = 1$, we get $k\gamma < \mathbf{w}_{k+1}^T \mathbf{w}_* \leq \|\mathbf{w}_{k+1}\| \leq R\sqrt{k}$

$$k \leq R^2/\gamma^2$$
Convergence of Perceptron

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

**Proof:**

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

$k \leq R^2/\gamma^2$

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

- The **Perceptron loss function** (without any regularization on \( \mathbf{w} \)):

\[
E(\mathbf{w}, b) = \sum_{n=1}^{N} \max\{0, -y_n(\mathbf{w}^T \mathbf{x}_n + b)\}
\]
The **Perceptron loss function** (without any regularization on $\mathbf{w}$):

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

- Loss $= 0$ on examples where Perceptron is correct, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) > 0$
The **Perceptron loss function** (without any regularization on $w$):

$$ E(w, b) = \sum_{n=1}^{N} \max\{0, -y_n(w^T x_n + b)\} $$

- **Loss = 0** on examples where Perceptron is **correct**, i.e., $y_n(w^T x_n + b) > 0$
- **Loss > 0** on examples where Perceptron **misclassifies**, i.e., $y_n(w^T x_n + b) < 0$
The **Perceptron loss function** (without any regularization on $w$):

$$E(w, b) = \sum_{n=1}^{N} \max\{0, -y_n(w^T x_n + b)\}$$

- **Loss** = 0 on examples where Perceptron is **correct**, i.e., $y_n(w^T x_n + b) > 0$
- **Loss** > 0 on examples where Perceptron **misclassifies**, i.e., $y_n(w^T x_n + b) < 0$

**Stochastic gradient descent** on $E(w, b)$ gives the Perceptron updates
The Perceptron loss function (without any regularization on $w$):

$$E(w, b) = \sum_{n=1}^{N} \max\{0, -y_n(w^T x_n + b)\}$$

- Loss $= 0$ on examples where Perceptron is correct, i.e., $y_n(w^T x_n + b) > 0$
- Loss $> 0$ on examples where Perceptron misclassifies, i.e., $y_n(w^T x_n + b) < 0$
- Stochastic gradient descent on $E(w, b)$ gives the Perceptron updates

Variants/Improvements of the basic Perceptron algorithm:
The **Perceptron loss function** (without any regularization on $\mathbf{w}$):

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

- Loss $= 0$ on examples where Perceptron is correct, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) > 0$
- Loss $> 0$ on examples where Perceptron misclassifies, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0$
- **Stochastic gradient descent** on $E(\mathbf{w}, b)$ gives the Perceptron updates

**Variants/Improvements of the basic Perceptron algorithm:**
- The Perceptron produces a set of weight vectors $\mathbf{w}^k$ during training
Perceptron: some additional notes

- The **Perceptron loss function** (without any regularization on $\mathbf{w}$):

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

  - Loss = 0 on examples where Perceptron is correct, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) > 0$
  - Loss > 0 on examples where Perceptron misclassifies, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0$

- **Stochastic gradient descent** on $E(\mathbf{w}, b)$ gives the Perceptron updates

- **Variants/Improvements of the basic Perceptron algorithm:**
  - The Perceptron produces a set of weight vectors $\mathbf{w}^k$ during training
  - The *standard Perceptron* simply uses the final weight vector at test time
The Perceptron loss function (without any regularization on $\mathbf{w}$):

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

- Loss $= 0$ on examples where Perceptron is correct, i.e., $y_n(\mathbf{w}^T\mathbf{x}_n + b) > 0$
- Loss $> 0$ on examples where Perceptron misclassifies, i.e., $y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0$

Stochastic gradient descent on $E(\mathbf{w}, b)$ gives the Perceptron updates

**Variants/Improvements of the basic Perceptron algorithm:**
- The Perceptron produces a set of weight vectors $\mathbf{w}^k$ during training
- The *standard Perceptron* simply uses the final weight vector at test time
  - This may sometimes not be a good idea!
Perceptron: some additional notes

- The **Perceptron loss function** (without any regularization on $w$):

$$E(w, b) = \sum_{n=1}^{N} \max\{0, -y_n(w^T x_n + b)\}$$

- Loss = 0 on examples where Perceptron is correct, i.e., $y_n(w^T x_n + b) > 0$
- Loss > 0 on examples where Perceptron misclassifies, i.e., $y_n(w^T x_n + b) < 0$
- **Stochastic gradient descent** on $E(w, b)$ gives the Perceptron updates

**Variants/Improvements of the basic Perceptron algorithm:**

- The Perceptron produces a set of weight vectors $w^k$ during training
- The *standard Perceptron* simply uses the final weight vector at test time
  - This may sometimes not be a good idea!
  - Some $w^k$ may be correct on 1000 consecutive examples but one mistake ruins it!
The Perceptron loss function (without any regularization on $\mathbf{w}$):

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

- Loss = 0 on examples where Perceptron is correct, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) > 0$
- Loss > 0 on examples where Perceptron misclassifies, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0$
- Stochastic gradient descent on $E(\mathbf{w}, b)$ gives the Perceptron updates

**Variants/Improvements of the basic Perceptron algorithm:**

- The Perceptron produces a set of weight vectors $\mathbf{w}^k$ during training
- The *standard Perceptron* simply uses the final weight vector at test time
  - This may sometimes not be a good idea!
  - Some $\mathbf{w}^k$ may be correct on 1000 consecutive examples but one mistake ruins it!
- We can actually do better using also the intermediate weight vectors
The Perceptron loss function (without any regularization on $\mathbf{w}$):

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

- Loss $= 0$ on examples where Perceptron is correct, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) > 0$
- Loss $> 0$ on examples where Perceptron misclassifies, i.e., $y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0$
- *Stochastic gradient descent* on $E(\mathbf{w}, b)$ gives the Perceptron updates

**Variants/Improvements of the basic Perceptron algorithm:**
- The Perceptron produces a set of weight vectors $\mathbf{w}^k$ during training
- The *standard Perceptron* simply uses the final weight vector at test time
  - This may sometimes not be a good idea!
  - Some $\mathbf{w}^k$ may be correct on 1000 consecutive examples but one mistake ruins it!
- We can actually do better using also the intermediate weight vectors
  - *Voted Perceptron* (vote on the predictions of the intermediate weight vectors)
The **Perceptron loss function** (without any regularization on \( \mathbf{w} \)):

\[
E(\mathbf{w}, b) = \sum_{n=1}^{N} \max\{0, -y_n(\mathbf{w}^T \mathbf{x}_n + b)\}
\]

- Loss = 0 on examples where Perceptron is correct, i.e., \( y_n(\mathbf{w}^T \mathbf{x}_n + b) > 0 \)
- Loss > 0 on examples where Perceptron misclassifies, i.e., \( y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0 \)
- **Stochastic gradient descent** on \( E(\mathbf{w}, b) \) gives the Perceptron updates

**Variants/Improvements of the basic Perceptron algorithm:**
- The Perceptron produces a set of weight vectors \( \mathbf{w}^k \) during training
- The *standard Perceptron* simply uses the final weight vector at test time
  - This may sometimes not be a good idea!
  - Some \( \mathbf{w}^k \) may be correct on 1000 consecutive examples but one mistake ruins it!
- We can actually do better using **also** the intermediate weight vectors
  - **Voted Perceptron** (vote on the predictions of the intermediate weight vectors)
  - **Averaged Perceptron** (average the intermediate weight vectors and then predict)
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
  - We will see this formally when we cover Learning Theory
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
A hyperplane based linear classifier defined by $\mathbf{w}$ and $b$
Support Vector Machine

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

- A hyperplane based linear classifier defined by $w$ and $b$
- Prediction rule: $y = \text{sign}(w^T x + b)$
- **Given:** Training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- **Goal:** Learn $w$ and $b$ that achieve the maximum margin
A hyperplane based linear classifier defined by \( w \) and \( b \)

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

**Given:** Training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)

**Goal:** Learn \( w \) and \( b \) that achieve the maximum margin

For now, assume the entire training data is correctly classified by \((w, b)\)
- Zero loss on the training examples (non-zero loss case later)
Support Vector Machine

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

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

**Given:** Training data $\{ (x_1, y_1), \ldots, (x_N, y_N) \}$

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

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

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

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

For now, assume the entire training data is correctly classified by \((\mathbf{w}, b)\)
- Zero loss on the training examples (non-zero loss case later)

- Assume the hyperplane is such that
  \[ \mathbf{w}^T \mathbf{x}_n + b \geq 1 \text{ for } y_n = +1 \]
Support Vector Machine

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

For now, assume the entire training data is correctly classified by $(\mathbf{w}, b)$
- Zero loss on the training examples (non-zero loss case later)

Assume the hyperplane is such that
- $\mathbf{w}^T \mathbf{x}_n + b \geq 1$ for $y_n = +1$
- $\mathbf{w}^T \mathbf{x}_n + b \leq -1$ for $y_n = -1$
Support Vector Machine

- A hyperplane based linear classifier defined by $\mathbf{w}$ and $b$
- Prediction rule: $y = sign(\mathbf{w}^T \mathbf{x} + b)$
- **Given:** Training data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
- **Goal:** Learn $\mathbf{w}$ and $b$ that achieve the maximum margin
- For now, assume the entire training data is correctly classified by $(\mathbf{w}, b)$
  - Zero loss on the training examples (non-zero loss case later)

  Assume the hyperplane is such that
  - $\mathbf{w}^T \mathbf{x}_n + b \geq 1$ for $y_n = +1$
  - $\mathbf{w}^T \mathbf{x}_n + b \leq -1$ for $y_n = -1$
  - Equivalently, $y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1$
Support Vector Machine

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

For now, assume the entire training data is correctly classified by $(\mathbf{w}, b)$
- Zero loss on the training examples (non-zero loss case later)

Assume the hyperplane is such that
- $\mathbf{w}^T \mathbf{x}_n + b \geq 1$ for $y_n = +1$
- $\mathbf{w}^T \mathbf{x}_n + b \leq -1$ for $y_n = -1$
- Equivalently, $y_n (\mathbf{w}^T \mathbf{x}_n + b) \geq 1$
  $\Rightarrow \min_{1 \leq n \leq N} |\mathbf{w}^T \mathbf{x}_n + b| = 1$
- The hyperplane’s margin:
  $$\gamma = \min_{1 \leq n \leq N} \frac{|\mathbf{w}^T \mathbf{x}_n + b|}{||\mathbf{w}||}$$
A hyperplane based linear classifier defined by $w$ and $b$

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

**Given:** Training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$

**Goal:** Learn $w$ and $b$ that achieve the maximum margin

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

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

Assume the hyperplane is such that

- $w^T x_n + b \geq 1$ for $y_n = +1$
- $w^T x_n + b \leq -1$ for $y_n = -1$

Equivalently, $y_n(w^T x_n + b) \geq 1$

$\Rightarrow \min_{1 \leq n \leq N} |w^T x_n + b| = 1$

The hyperplane's margin:

$$\gamma = \min_{1 \leq n \leq N} \frac{|w^T x_n + b|}{||w||} = \frac{1}{||w||}$$
We want to maximize the margin \( \gamma = \frac{1}{||w||} \)
We want to maximize the margin \( \gamma = \frac{1}{||w||} \)

Maximizing the margin \( \gamma = \text{minimizing} \ ||w|| \) (the norm)
We want to maximize the margin \( \gamma = \frac{1}{||w||} \)

Maximizing the margin \( \gamma = \text{minimizing} \ ||w|| \) (the norm)

Our optimization problem would be:

Minimize \( f(w, b) = \frac{||w||^2}{2} \)

subject to \( y_n(w^Tx_n + b) \geq 1, \quad n = 1, \ldots, N \)
Support Vector Machine: The Optimization Problem

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

- Maximizing the margin \( \gamma = \text{minimizing} \ ||w|| \) (the norm)

- Our optimization problem would be:

\[
\text{Minimize} \quad f(w, b) = \frac{||w||^2}{2} \\
\text{subject to} \quad y_n(w^T x_n + b) \geq 1, \quad n = 1, \ldots, N
\]

- 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 $\gamma = \frac{1}{||w||}$
- Large margin $\Rightarrow$ small $||w||$
Large Margin = Good Generalization

- Large margins intuitively mean good generalization
- We can give a slightly more formal justification to this
  - Recall: Margin $\gamma = \frac{1}{||w||}$
  - Large margin $\Rightarrow$ small $||w||$
  - Small $||w||$ $\Rightarrow$ regularized/simple solutions ($w_i$’s don’t become too large)
  - Simple solutions $\Rightarrow$ good generalization on test data
Large Margin = Good Generalization

- Large margins intuitively mean good generalization

- We can give a slightly more formal justification to this

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

- Large margin $\Rightarrow$ small $||w||$

- Small $||w|| \Rightarrow$ regularized/simple solutions ($w_i$’s don’t become too large)

- Simple solutions $\Rightarrow$ good generalization on test data

- Want to see an even more formal justification? :-)

(CS5350/6350) Hyperplane based Classification  September 8, 2011  18 / 20
Large Margin = Good Generalization

- Large margins intuitively mean good generalization

- We can give a slightly more formal justification to this

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

- Large margin $\Rightarrow$ small $||w||$

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

- Simple solutions $\Rightarrow$ good generalization on test data

- Want to see an even more formal justification? :-)
  - Wait until we cover Learning Theory!
Our optimization problem is:

\[
\text{Minimize } f(w, b) = \frac{||w||^2}{2}
\]

subject to \( 1 \leq y_n(w^T x_n + b), \quad n = 1, \ldots, N \)
Solving the SVM Optimization Problem

Our optimization problem is:

\[
\begin{align*}
\text{Minimize} & \quad f(w, b) = \frac{||w||^2}{2} \\
\text{subject to} & \quad 1 \leq y_n(w^T x_n + b), \quad n = 1, \ldots, N
\end{align*}
\]

Introducing **Lagrange Multipliers** \( \alpha_n \) \((n = \{1, \ldots, N\})\), one for each constraint, leads to the **Lagrangian**:

\[
\begin{align*}
\text{Minimize} & \quad L(w, b, \alpha) = \frac{||w||^2}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b)\} \\
\text{subject to} & \quad \alpha_n \geq 0; \quad n = 1, \ldots, N
\end{align*}
\]
Solving the SVM Optimization Problem

Our optimization problem is:

\[
\text{Minimize } f(w, b) = \frac{||w||^2}{2}
\]
subject to \(1 \leq y_n(w^T x_n + b), \quad n = 1, \ldots, N\)

Introducing Lagrange Multipliers \(\alpha_n (n = \{1, \ldots, N\})\), one for each constraint, leads to the Lagrangian:

\[
\text{Minimize } L(w, b, \alpha) = \frac{||w||^2}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b)\}
\]
subject to \(\alpha_n \geq 0; \quad n = 1, \ldots, N\)

We can now solve this Lagrangian

- i.e., optimize \(L(w, b, \alpha)\) w.r.t. \(w, b,\) and \(\alpha\)
- .. making use of the Lagrangian Duality theory.
Next class.

- Solving the SVM optimization problem
- Allowing misclassified training examples (non-zero loss)
- Introduction to kernel methods (nonlinear SVMs)