Machine Learning: Yoshinari Fujinuma

University of Colorado Boulder

LECTURE 6

Slides adapted from Chenhao Tan, Chris Ketelsen, Noah Smith

Logistics

- Next Monday is a hands-on day, be prepared to use a python notebook
- HW1 Part 3 math notations

Learning objectives

- Understand the perceptron algorithm

Outline

Perceptron

Perceptron algorithm in depth

Interpretation of weight values

Convergence of the perceptron algorithm

Bonus: average perceptron

Decision Trees, K-NN, and Today's Model

Decision trees (shallow): use relatively few features to classify.

K-nearest neighbors: all features weighted equally.

Today: use all features, but weight them.

Decision Trees, K-NN, and Today's Model

Decision trees (shallow): use relatively few features to classify.

K-nearest neighbors: all features weighted equally.

Today: use all features, but weight them.

For today's lecture, assume that $y \in \{-1, +1\}$ instead of $\{0, 1\}$, and that $\boldsymbol{x} \in \mathbb{R}^d$.

Outline

Perceptron

Perceptron algorithm in depth

Interpretation of weight values

Convergence of the perceptron algorithm

Bonus: average perceptron

Linear classifiers

- Binary Classification
- A linear classifier draws a line through space separating the two classes.
- For two-features, a linear classifier takes form on the right

Perceptron classifiers are linear classifiers

Let's think about having two features i.e., $\boldsymbol{x} = (x_1, x_2)$

$$\begin{aligned} y &= \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b) \\ &= \text{sign}(w_1 x_1 + w_2 x_2 + b) \end{aligned}$$

Perceptron classifiers are linear classifiers

Let's think about having two features i.e., $\boldsymbol{x} = (x_1, x_2)$

$$
\begin{aligned}
y &= \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b) \\
&= \text{sign}(w_1 x_1 + w_2 x_2 + b) \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{cases}
\end{aligned}
$$

Perceptron

Let $\boldsymbol{x} \in \mathbb{R}^d$ be the input data with d features
Let $\boldsymbol{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$
We denote trainable parameters as blue

$$f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right)$$

$$\text{remembering that: } \boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{d} \boldsymbol{w}_j \cdot \boldsymbol{x}_j$$

Perceptron

Let $\boldsymbol{x} \in \mathbb{R}^d$ be the input data with d features
Let $\boldsymbol{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$
We denote trainable parameters as blue

$$f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right)$$

$$\text{remembering that: } \boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{d} \boldsymbol{w}_j \cdot \boldsymbol{x}_j$$

Learning requires us to set the weights $\boldsymbol{w}$ and the bias b.

Outline

Perceptron

Perceptron algorithm in depth

Interpretation of weight values

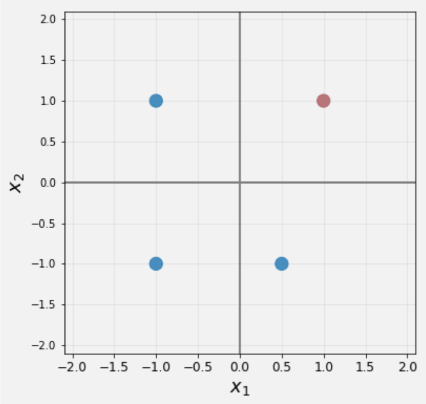
Convergence of the perceptron algorithm

Bonus: average perceptron

Example

- Start with $\boldsymbol{w} = [1, 0], b = 0$
- Process points in order (red: $+1$, blue: -1):

$$(1, 1), (0.5, -1), (-1. - 1), (-1, 1)$$

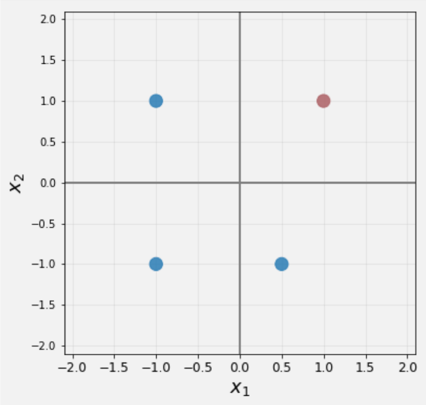


Example

- Start with $\boldsymbol{w} = [1, 0], b = 0$
- Process points in order (red: $+1$, blue: -1):

$$(1,1), (0.5, -1), (-1. -1), (-1, 1)$$

- $(1,1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 1, y = 1$: no update

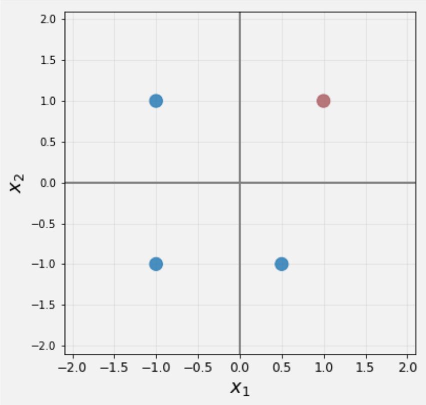


Example

- Start with $\boldsymbol{w} = [1, 0], b = 0$
- Process points in order (red: $+1$, blue: -1):

$$(1, 1), (0.5, -1), (-1. - 1), (-1, 1)$$

- $(1, 1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 1, y = 1$: no update
- $(0.5, -1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 0.5, y = -1$: misclassification!

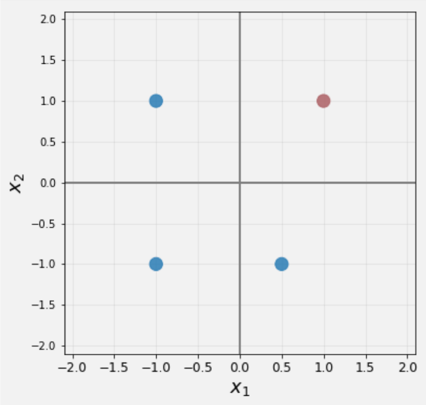


Example

- Start with $\boldsymbol{w} = [1, 0], b = 0$
- Process points in order (red: $+1$, blue: -1):

$$(1, 1), (0.5, -1), (-1. - 1), (-1, 1)$$

- $(1, 1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 1, y = 1$: no update
- $(0.5, -1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 0.5, y = -1$: misclassification!
  So we update the parameters
  $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x} = [0.5, 1], b' = b + y = -1$

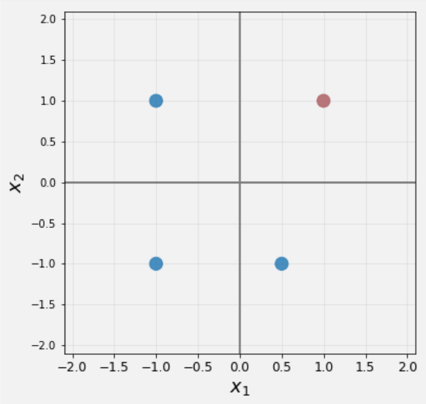


Example

- Start with $\boldsymbol{w} = [1, 0], b = 0$
- Process points in order (red: $+1$, blue: -1):

$$(1, 1), (0.5, -1), (-1. - 1), (-1, 1)$$

- $(1, 1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 1, y = 1$: no update
- $(0.5, -1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 0.5, y = -1$: misclassification!
  So we update the parameters
  $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x} = [0.5, 1], b' = b + y = -1$
- $(-1, -1) : \boldsymbol{w}' \cdot \boldsymbol{x} + b' = -2.5, y = -1$: no update

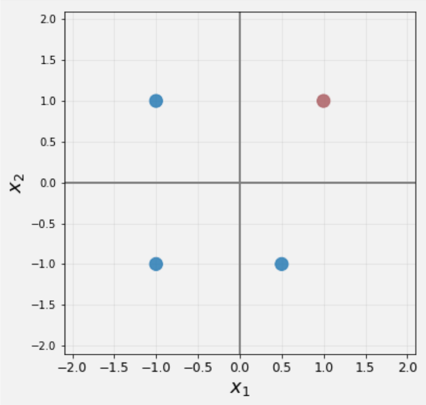


Example

- Start with $\boldsymbol{w} = [1, 0], b = 0$
- Process points in order (red: $+1$, blue: -1):

$$(1, 1), (0.5, -1), (-1. - 1), (-1, 1)$$

- $(1, 1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 1, y = 1$: no update
- $(0.5, -1) : \boldsymbol{w} \cdot \boldsymbol{x} + b = 0.5, y = -1$:
  misclassification!
  So we update the parameters
  $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x} = [0.5, 1], b' = b + y = -1$
- $(-1, -1) : \boldsymbol{w}' \cdot \boldsymbol{x} + b' = -2.5, y = -1$: no update
- $(-1, 1) : \boldsymbol{w}' \cdot \boldsymbol{x} + b' = -0.5, y = -1$: no update



Why does this algorithm work?

Assume that we have just misclassified a point $(\boldsymbol{x}, y)$, it means

$$ay \leq 0$$

since $a = \boldsymbol{w} \cdot \boldsymbol{x} + b = \{+1, -1\}$
After the update: $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x}, b' = b + y$

Why does this algorithm work?

Assume that we have just misclassified a point $(\boldsymbol{x}, y)$, it means

$$ay \leq 0$$

since $a = \boldsymbol{w} \cdot \boldsymbol{x} + b = \{+1, -1\}$
After the update: $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x}, b' = b + y$

$$\begin{aligned} a' &= \boldsymbol{w}' \cdot \boldsymbol{x} + b' \\ &= \boldsymbol{w} \cdot \boldsymbol{x} + y||\boldsymbol{x}||^2 + b + y \\ &= a + y||\boldsymbol{x}||^2 + y \end{aligned}$$

Why does this algorithm work?

Assume that we have just misclassified a point $(\boldsymbol{x}, y)$, it means

$$ay \leq 0$$

since $a = \boldsymbol{w} \cdot \boldsymbol{x} + b = \{+1, -1\}$
After the update: $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x}, b' = b + y$

$$\begin{aligned}
a' &= \boldsymbol{w}' \cdot \boldsymbol{x} + b' \\
&= \boldsymbol{w} \cdot \boldsymbol{x} + y||\boldsymbol{x}||^2 + b + y \\
&= a + y||\boldsymbol{x}||^2 + y \\
a'y &= ay + ||\boldsymbol{x}||^2 + 1 > ay
\end{aligned}$$

Perceptron learning algorithm

Data: $D = \langle(\boldsymbol{x}_n, y_n)\rangle_{n=1}^{N}$, number of epochs E
Result: weights $\boldsymbol{w}$ and bias b
initialize: $\boldsymbol{w} = \boldsymbol{0}$ and $b = 0$;
for $e \in \{1, \ldots, E\}$ **do**
    for $n \in \{1, \ldots, N\}$, *in random order* **do**
        # predict
        $a = (\boldsymbol{w} \cdot \boldsymbol{x}_n + b)$;
        if $ay_n \leq 0$ **then**
            # update
            $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n \cdot \boldsymbol{x}_n$;
            $b \leftarrow b + y_n$;
        end
    end
end

Perceptron learning algorithm

Data: $D = \langle (\boldsymbol{x}_n, y_n) \rangle_{n=1}^{N}$, number of epochs E
Result: weights $\boldsymbol{w}$ and bias b
initialize: $\boldsymbol{w} = \boldsymbol{0}$ and $b = 0$;
for $e \in \{1, \ldots, E\}$ **do**
    for $n \in \{1, \ldots, N\}$, *in random order* **do**
        # predict
        $a = (\boldsymbol{w} \cdot \boldsymbol{x}_n + b)$;
        if $a y_n \leq 0$ **then**
            # update
            $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n \cdot \boldsymbol{x}_n$;
            $b \leftarrow b + y_n$;
        end
    end
end

- why $a y_n \leq 0$ rather than $a y_n < 0$?

Parameters and Hyperparameters

Parameters are numerical values ($\boldsymbol{w}$ and b).

Parameters and Hyperparameters

Parameters are numerical values ($\boldsymbol{w}$ and b).

Only hyperparameter is E, the number of epochs (passes over the training data).

Outline

Perceptron

Perceptron algorithm in depth

Interpretation of weight values

Convergence of the perceptron algorithm

Bonus: average perceptron

Interpretation of Weight Values

What does it mean when ...

- $w_1 = 100$?
- $w_2 = -1$?
- $w_3 = 0$?

Interpretation of Weight Values

In other words, how sensitive is the final classification to changes in individual features?

$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{cases}$$

Interpretation of Weight Values

In other words, how sensitive is the final classification to changes in individual features?

$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{cases}$$

To measure a small change with respect to jth feature x_j,

$$\frac{\partial \boldsymbol{w} \cdot \boldsymbol{x} + b}{\partial \boldsymbol{x}_j} = \boldsymbol{w}_j$$

Interpretation of Weight Values

In other words, how sensitive is the final classification to changes in individual features?

$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{cases}$$

To measure a small change with respect to jth feature x_j,

$$\frac{\partial \boldsymbol{w} \cdot \boldsymbol{x} + b}{\partial \boldsymbol{x}_j} = \boldsymbol{w}_j$$

If features are similar scale (i.e., standardized) then large weights indicate important features

Outline

Perceptron

Perceptron algorithm in depth

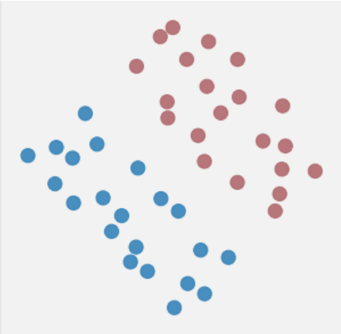
Interpretation of weight values

Convergence of the perceptron algorithm

Bonus: average perceptron

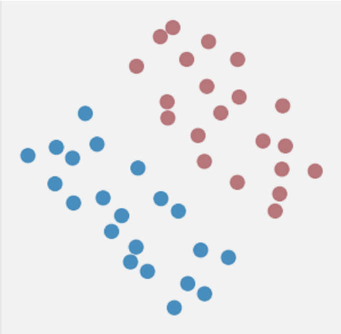
Convergence of the perceptron algorithm

- If a linear classifier can separate the training set, Perceptron will find it



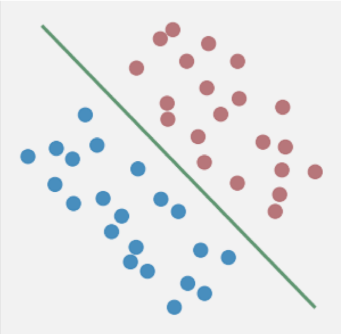
Convergence of the perceptron algorithm

- If a linear classifier can separate the training set, Perceptron will find it
- Such training sets are called **linearly separable**



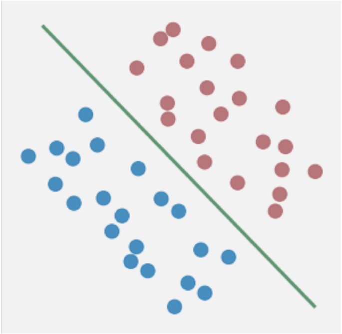
Convergence of the perceptron algorithm

- If a linear classifier can separate the training set, Perceptron will find it
- Such training sets are called **linearly separable**



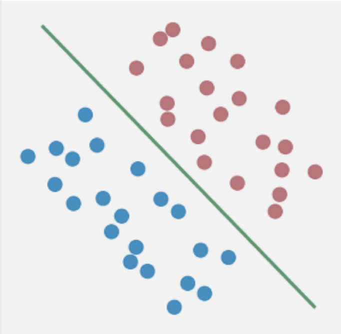
Convergence of the perceptron algorithm

- If a linear classifier can separate the training set, Perceptron will find it
- Such training sets are called **linearly separable**
- *Margin* characterizes how separable a dataset is



Convergence of the perceptron algorithm

- If a linear classifier can separate the training set, Perceptron will find it
- Such training sets are called **linearly separable**
- *Margin* characterizes how separable a dataset is
- How long it takes to converge depend on the margin



Convergence of the perceptron algorithm [Rosenblatt, 1958].

If D is linearly separable with margin $\gamma > 0$ and for all $n \in \{1, \dots, N\}$, $\|\boldsymbol{x}_n\|_2 \leq 1$ (note that n indexes instances here), then the perceptron algorithm will converge in at most $\frac{1}{\gamma^2}$ updates.

$$\gamma = \text{margin}(D, \boldsymbol{w}, b) = \begin{cases} \min_n y_n \cdot (\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{if } \boldsymbol{w} \text{ and } b \text{ separate } D \\ -\infty & \text{otherwise} \end{cases}$$

Convergence of the perceptron algorithm [Rosenblatt, 1958].

If D is linearly separable with margin $\gamma > 0$ and for all $n \in \{1, \ldots, N\}$, $\|\boldsymbol{x}_n\|_2 \leq 1$ (note that n indexes instances here), then the perceptron algorithm will converge in at most $\frac{1}{\gamma^2}$ updates.

$$\gamma = \text{margin}(D, \boldsymbol{w}, b) = \begin{cases} \min_n y_n \cdot (\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{if } \boldsymbol{w} \text{ and } b \text{ separate } D \\ -\infty & \text{otherwise} \end{cases}$$

- Proof can be found in Daume [2017], pp. 50–51.
- The theorem does not guarantee that the perceptron's classifier will achieve margin γ.

Perceptron Wrap-up

- The perceptron is a simple classifier that sometimes works very well
- Linear classifiers in general will pop up again and again, e.g., Logistic Regression
- The idea of margins will show up again when we talk about Support Vector Machines
- Neural Networks are essentially generalizations of the perceptron

Outline

Perceptron

Perceptron algorithm in depth

Interpretation of weight values

Convergence of the perceptron algorithm

Bonus: average perceptron

Voting perceptron

Suppose you have a data set with 10,000 training examples

Suppose that after 100 examples it's learned a really good set of weights

So good that for the next 9,899 examples it doesn't make any mistakes

And then on 10,000th example it misclassifies and totally changes the weights

Idea: Give more vote to weights that persist for a long time

Voting perceptron

Train as usual, save weights $(\boldsymbol{w}, b)^{(1)}, \ldots, (\boldsymbol{w}, b)^{(K)}$ and steps they persist $c^{(1)}, \ldots, c^{(K)}$

Then predict using a weighted activation:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}(\boldsymbol{w}^{(k)} \cdot \boldsymbol{x} + b^{(k)})\right)$$

Voting perceptron

Train as usual, save weights $(\boldsymbol{w}, b)^{(1)}, \ldots, (\boldsymbol{w}, b)^{(K)}$ and steps they persist $c^{(1)}, \ldots, c^{(K)}$

Then predict using a weighted activation:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}(\boldsymbol{w}^{(k)} \cdot \boldsymbol{x} + b^{(k)})\right)$$

A more efficient method is the Averaged Perceptron

$$\hat{y} = \text{sign}\left(\left(\sum_{k=1}^{K} c^{(k)} \boldsymbol{w}^{(k)}\right) \cdot \boldsymbol{x} + \sum_{k=1}^{K} b^{(k)}\right)$$

References

Hal Daume. *A Course in Machine Learning (v0.9)*. Self-published at `http://ciml.info/`, 2017.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.