Machine Learning: Chenhao Tan

University of Colorado Boulder

LECTURE 6

Slides adapted from Chris Ketelsen, Noah Smith

Logistics

- HW1 available on Github, due in 7 days
- Wednesday is a hands-on day, so bring your laptops

Learning objectives

- Understand parametric models and linear classifiers
- Understand the perceptron algorithm

Outline

Parametric models
- Neuron-Inspired Classifier
- Linear classifiers

Perceptron algorithm
- Vanilla perceptron algorithm
- Interpretation of weight values
- Convergence of the perceptron algorithm
- Bonus: average perceptron

Outline

Complexity of KNN

Last time we learned about curse of dimensionality in the context of KNN

How does the algorithm scale? (memory and efficiency of the naive implementation)

Complexity of KNN

Last time we learned about curse of dimensionality in the context of KNN

How does the algorithm scale? (memory and efficiency of the naive implementation)
Training: N/A

Complexity of KNN

Last time we learned about curse of dimensionality in the context of KNN

How does the algorithm scale? (memory and efficiency of the naive implementation)
Training: N/A
Testing

- memory: $O(Nd)$
- time: $O(Nd)$

In comparison, decision tree does not need to literally remember all data, and only need to use features

Happy medium

Decision trees (that aren't too deep): use relatively few features to classify.

K-nearest neighbors: all features weighted equally.

Today: use all features, but weight them.

Happy medium

Decision trees (that aren't too deep): 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 $\boldsymbol{x} \in \mathbb{R}^d$.

Happy medium

Decision trees (that aren't too deep): use relatively few features to classify.

K-nearest neighbors: all features weighted equally.

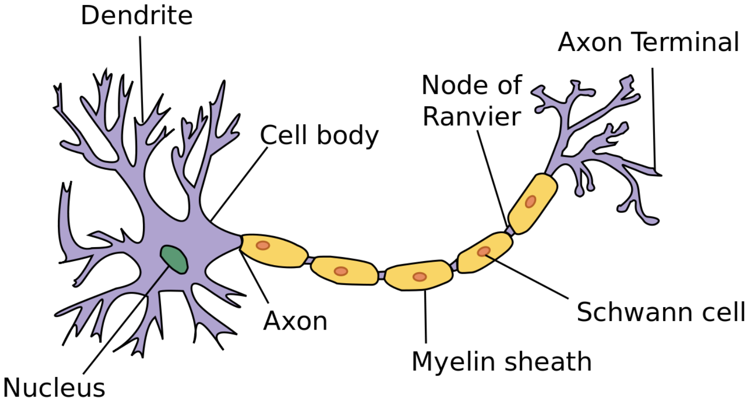
Today: use all features, but weight them.

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

Remember that $\boldsymbol{x}_j = \phi_j(x)$. (Features have already been applied to the data.)

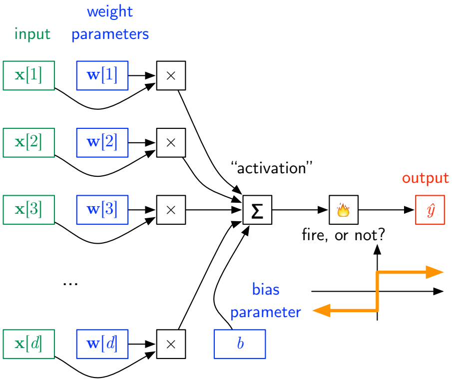
Inspiration from Neurons

Image from Wikimedia Commons.



Input signals come in through dendrites, output signal passes out through the axon.

Neuron-inspired classifier



Neuron-inspired classifier

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

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

Neuron-inspired classifier

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

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.

Linear classifiers

- Binary Classification: Only two classes
- 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

$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

Perceptron classifiers are linear classifiers

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

Outline

Parametric models
- Neuron-Inspired Classifier
- Linear classifiers

Perceptron algorithm
- Vanilla perceptron algorithm
- Interpretation of weight values
- Convergence of the perceptron algorithm
- Bonus: average perceptron

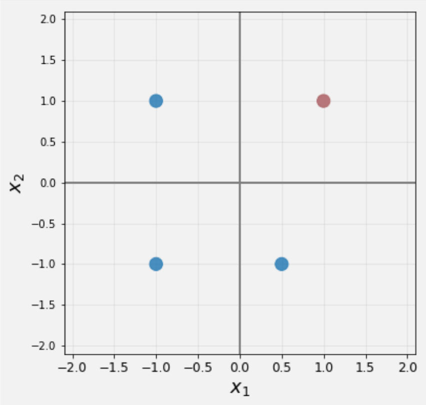
Perceptron learning algorithm

Data: $D = \langle(\boldsymbol{x}_n, y_n)\rangle_{n=1}^{N}$, number of epochs E
Result: weights $\boldsymbol{w}$ and bias b
initialize: $\boldsymbol{w} = \mathbf{0}$ 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

Example

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

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

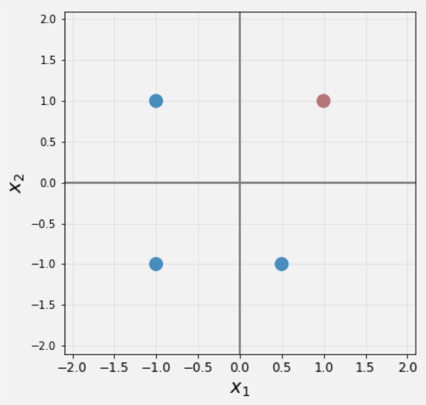


Example

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

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

- $(1,1) : a = 1, y = 1$: no update

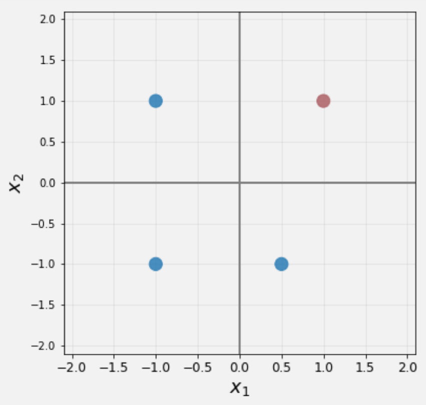


Example

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

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

- $(1, 1) : a = 1, y = 1$: no update
- $(0.5, -1) : a = 0.5, y = -1$: $w = [0.5, 1], b = -1$

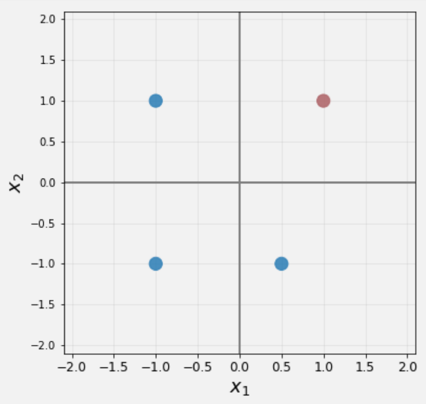


Example

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

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

- $(1,1): a = 1, y = 1$: no update
- $(0.5,-1): a = 0.5, y = -1$: $w = [0.5, 1], b = -1$
- $(-1,-1): a = -2.5, y = -1$: no update

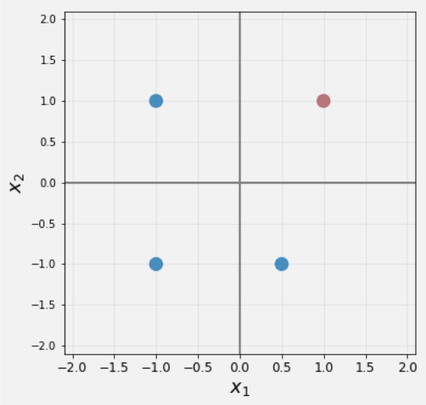


Example

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

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

- $(1,1) : a = 1, y = 1$: no update
- $(0.5, -1) : a = 0.5, y = -1$: $\boldsymbol{w} = [0.5, 1], b = -1$
- $(-1, -1) : a = -2.5, y = -1$: no update
- $(-1, 1) : a = -0.5, y = -1$: no update



Why does this algorithm work?

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

$$a = \boldsymbol{w} \cdot \boldsymbol{x} + b, ay \leq 0$$

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

$$a = \boldsymbol{w} \cdot \boldsymbol{x} + b, ay \leq 0$$

After the update: $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x}, b' = b + y$

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

Why does this algorithm work?

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

$$a = \boldsymbol{w} \cdot \boldsymbol{x} + b, ay \leq 0$$

After the update: $\boldsymbol{w}' = \boldsymbol{w} + y\boldsymbol{x}, b' = b + y$

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

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 $e \in \{1, \dots, E\}$ **do**
    for $n \in \{1, \dots, N\}$, *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

- why $ay_n \leq 0$ rather than $ay_n < 0$?
- why random order?

Parameters and Hyperparameters

This is the first supervised algorithm we've seen that has **parameters** that are numerical values (w and b).

Parameters and Hyperparameters

This is the first supervised algorithm we've seen that has **parameters** that are numerical values (w and b).

The perceptron learning algorithm's sole hyperparameter is E, the number of epochs (passes over the training data).

Interpretation of Weight Values

What does it mean when . . .

Interpretation of Weight Values

What does it mean when . . .

- $w_{12} = 100$?

Interpretation of Weight Values

What does it mean when ...

- $w_{12} = 100$?
- $w_{12} = -1$?

Interpretation of Weight Values

What does it mean when ...

- $w_{12} = 100$?
- $w_{12} = -1$?
- $w_{12} = 0$?

Interpretation of Weight Values

What does it mean when . . .

- $w_{12} = 100$?
- $w_{12} = -1$?
- $w_{12} = 0$?

What if we multiply $\boldsymbol{w}$ by 2?

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?

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

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

Interpretation of Weight Values

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

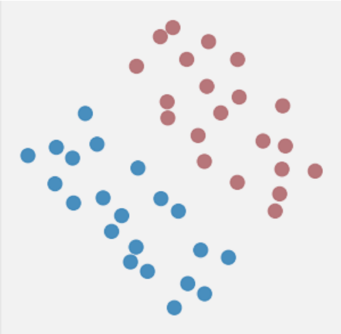
$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b) \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{cases}$$

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

If features are similar size then large weights indicate important features

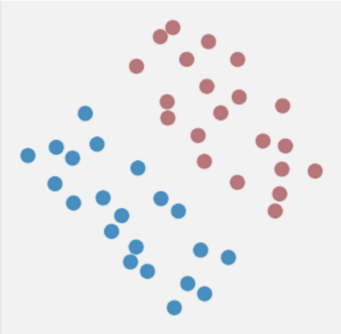
Convergence of the perceptron algorithm

- If possible for a linear classifier to separate data, Perceptron will find it



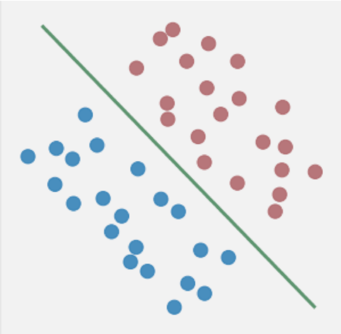
Convergence of the perceptron algorithm

- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called linearly separable



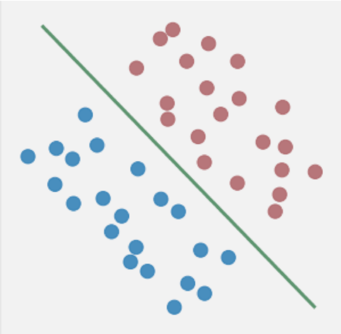
Convergence of the perceptron algorithm

- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called linearly separable



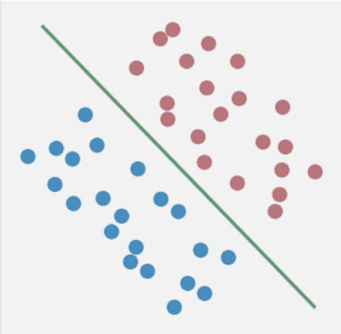
Convergence of the perceptron algorithm

- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called linearly separable
- *Margin* characterizes how separable a dataset is



Convergence of the perceptron algorithm

- If possible for a linear classifier to separate data, 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

Due to 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.

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

Convergence of the perceptron algorithm

Due to 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 will be pretty similar
- The idea of margins will show up again when we talk about Support Vector Machines
- Neural Networks are essentially generalizations of the 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.