Lecture 14: Perceptrons Again

COMP 411, Fall 2021
Victoria Manfredi

WESLEYAN UNIVERSITY

Acknowledgements: These slides are based primarily on content from slides by Michael Paul (U of Colorado Boulder)

Today's Topics

Perceptron

- Overview
- Example
- Learning the weights
- Linear separability

Perceptron

OVERVIEW

Recall: linear classifiers

Learn a linear function that separates instances of different classes

A linear function divides the coordinate space into two parts

- Every point is either on one side of the line (or plane or hyperplane) or the other (unless it is exactly on the line and need to break ties)

This means it can only separate two classes

- Classification with two classes is called **binary classification**
- Conventionally, one class is called the positive class and the other is the negative class

Perceptron

Perceptron is an algorithm for binary classification that uses a linear prediction function

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

This is called a step function, which reads:

- The output is 1 if $\mathbf{w}^T\mathbf{x} + b \geq 0$ is true
- The output is -1 $\mathbf{w}^T\mathbf{x} + b < 0$ is true

Perceptron

Perceptron is an algorithm for binary classification that uses a linear prediction function

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

By convention, the 2 classes are +1 or -1

Perceptron

Perceptron is an algorithm for binary classification that uses a linear prediction function

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

By convention, the slope parameters are denoted $\mathbf{w}$. Often these parameters are called weights.

Perceptron

Perceptron is an algorithm for binary classification that uses a linear prediction function

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

By convention, ties are broken in favor of the positive class: if $\mathbf{w}^T\mathbf{x} + b$ is exactly 0, output +1 instead of -1

Perceptron

The $\mathbf{w}$ parameters are unknown. This is what we have to learn

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

In the same way that linear regression learns the slope parameters to best fit the data points, perceptron learns the parameters to best separate the instances.

Perceptron

EXAMPLE

Example

Suppose we want to predict whether a web user will click on an ad for a refrigerator

Four features:

- Recently searched “refrigerator repair”
- Recently searched “refrigerator reviews”
- Recently bought a refrigerator
- Has clicked on any ad in the recent past

These are all **binary features** (values can be either 0 or 1)

Example

Suppose these are the weights

- Recently searched “refrigerator repair”: 2.0
- Recently searched “refrigerator reviews”: 8.0
- Recently bought a refrigerator: -15.0
- Has clicked on any ad in the recent past: 5.0
- b (intercept): -9.0

Prediction function

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

Example

Suppose these are the weights

- Recently searched "refrigerator repair": 2.0
- Recently searched "refrigerator reviews": 8.0
- Recently bought a refrigerator: -15.0
- Has clicked on any ad in the recent past: 5.0
- b (intercept): -9.0

$$\mathbf{w}^T\mathbf{x} + b$$
$$= 2 \cdot 0 + 8 \cdot 1 - 15 \cdot 0 + 5 \cdot 0 + -9$$
$$= 8 - 9 = -1$$

Prediction: No

Example

Suppose these are the weights

- Recently searched “refrigerator repair”: 2.0
- Recently searched “refrigerator reviews”: 8.0
- Recently bought a refrigerator: -15.0
- Has clicked on any ad in the recent past: 5.0
- b (intercept): -9.0

$$\mathbf{w}^T\mathbf{x} + b$$
$$= 2 \cdot 1 + 8 \cdot 1 - 15 \cdot 0 + 5 \cdot 0 + -9$$
$$= 2 + 8 - 9 = 1$$

Prediction: Yes

Example

Suppose these are the weights

- Recently searched "refrigerator repair": 2.0
- Recently searched "refrigerator reviews": 8.0
- Recently bought a refrigerator: -15.0
- Has clicked on any ad in the recent past: 5.0
- b (intercept): -9.0

$$\mathbf{w}^T\mathbf{x} + b$$
$$= 2 \cdot 0 + 8 \cdot 1 - 15 \cdot 0 + 5 \cdot 1 + -9$$
$$= 8 + 5 - 9 = 1$$

Prediction: Yes

Example

Suppose these are the weights

- Recently searched "refrigerator repair": 2.0
- Recently searched "refrigerator reviews": 8.0
- Recently bought a refrigerator: -15.0
- Has clicked on any ad in the recent past: 5.0
- b (intercept): -9.0

$$\mathbf{w}^T\mathbf{x} + b$$

$$= 2 \cdot 0 + 8 \cdot 1 - 15 \cdot 1 + 5 \cdot 1 + -9$$

$$= 8 - 15 + 5 - 9$$

Prediction: No

If someone bought a refrigerator recently, they probably aren't interested in shopping for another one anything soon

Perceptron

LEARNING THE WEIGHTS

Learning the weights

The perceptron algorithm learns the weights by

1. Initialize all weights $\mathbf{w}$ to 0
2. Iterate through the training data. For each training instance, classify the instance
   - If the prediction (the output of the classifier) was correct, don't do anything (it means the classifier is working, so leave it alone!)
   - If the prediction was wrong, modify the weights by using the **update rule**
3. Repeat step 2 some number of times (more on this later)

Learning the weights

What does an **update rule** do?

If the classifier predicted an instance was negative but it should have been positive …

- Currently: $\mathbf{w}^T\mathbf{x}_i + b < 0$
- Want: $\mathbf{w}^T\mathbf{x}_i + b \geq 0$
- Adjust the weights $\mathbf{w}$ so that this function value moves toward positive

If the classifier predicted positive but it should have been negative, shift the weights so that the value moves toward negative.

Learning the weights

The perceptron update rule:

$$w_j += (y_i - f(x_i))x_{ij}$$

w_j: The weight of feature j
y_i: The true label of instance i
x_i: The feature vector of instance i
$f(x_i)$: The class prediction for instance i
x_{ij}: The value of feature j in instance i

Learning the weights

The perceptron update rule:

$$w_j += (y_i - f(x_i))x_{ij}$$

w_j: The weight of feature j
y_i: The true label of instance i
x_i: The feature vector of instance i
$f(x_i)$: The class prediction for instance i
x_{ij}: The value of feature j in instance i

Let's assume x_{ij} is 1 in this example for now.

Learning the weights

The perceptron update rule:

$$w_j += (y_i - f(x_i))x_{ij}$$

w_j: The weight of feature j
y_i: The true label of instance i
x_i: The feature vector of instance i
$f(x_i)$: The class prediction for instance i
x_{ij}: The value of feature j in instance i

This term is 0 if the prediction was correct ($y_i = f(\mathbf{x}_i)$).
Then the entire update rule is 0, so no change is made

Learning the weights

The perceptron update rule:

$$w_j += (y_i - f(x_i))x_{ij}$$

w_j: The weight of feature j
y_i: The true label of instance i
x_i: The feature vector of instance i
$f(x_i)$: The class prediction for instance i
x_{ij}: The value of feature j in instance i

If the prediction is wrong:

This term is +2 if $y_i = +1$ and $f(\mathbf{x}_i) = -1$

This term is -2 if $y_i = -1$ and $f(\mathbf{x}_i) = +1$

The sign of this term indicates the direction of the mistake

Learning the weights

The perceptron update rule:

$$w_j += (y_i - f(x_i))x_{ij}$$

w_j: The weight of feature j
y_i: The true label of instance i
x_i: The feature vector of instance i
$f(x_i)$: The class prediction for instance i
x_{ij}: The value of feature j in instance i

If the prediction is wrong:

This term is +2 if $y_i = +1$ and $f(\mathbf{x}_i) = -1$

This will increase w_j (still assuming x_{ij} is 1), which will increase $\mathbf{w}^T\mathbf{x} + b$, which will make it more likely $\mathbf{w}^T\mathbf{x_i} + b \geq 0$ next time (which is what we need for the classifier to be correct)

Learning the weights

The perceptron update rule:

$$w_j += (y_i - f(x_i))x_{ij}$$

w_j: The weight of feature j
y_i: The true label of instance i
x_i: The feature vector of instance i
$f(x_i)$: The class prediction for instance i
x_{ij}: The value of feature j in instance i

If the prediction is wrong:

This term is -2 if $y_i = -1$ and $f(\mathbf{x}_i) = +1$

This will decrease w_j (still assuming x_{ij} is 1), which will decrease $\mathbf{w}^T\mathbf{x} + b$, which will make it more likely $\mathbf{w}^T\mathbf{x_i} + b < 0$ next time (which is what we need for the classifier to be correct)

Learning the weights

The perceptron update rule:

$$w_j += (y_i - f(x_i))x_{ij}$$

> w_j: The weight of feature j
> y_i: The true label of instance i
> x_i: The feature vector of instance i
> $f(x_i)$: The class prediction for instance i
> x_{ij}: The value of feature j in instance i

If x_{ij} is 0, there will be no update

The feature does not affect the prediction for this instance, so it won't affect the weight updates.

If x_{ij} is negative, the sign of the update flips

Learning the weights

What about b?

This is the intercept of the linear function, also called the **bias**

Common implementation:

Realize that $\mathbf{w}^T\mathbf{x} + b = \mathbf{w}^T\mathbf{x} + b \cdot 1$

- If we add an extra feature to every instance whose value is always 1, then we can simply write this as $\mathbf{w}^T\mathbf{x}$ where the final feature weight is the value of the bias
- Then we can update this parameter the same way as all of the other weights

Learning the weights

The vector of $\mathbf{w}$ is called the **weight vector**

Is the bias b counted when we use this phrase?

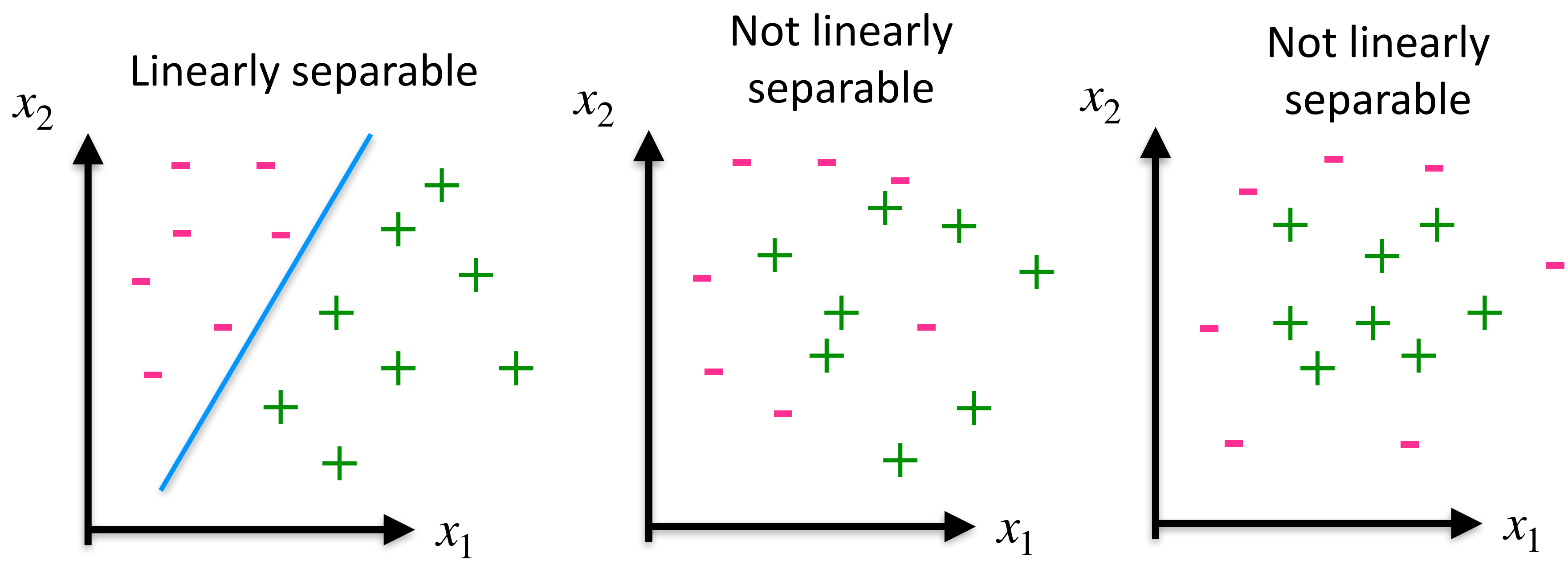
- Usually, especially if you include it by using the trick of adding an extra feature with value 1 rather than treating it separately
- Just be clear with your notation

Perceptron

LINEAR SEPARABILITY

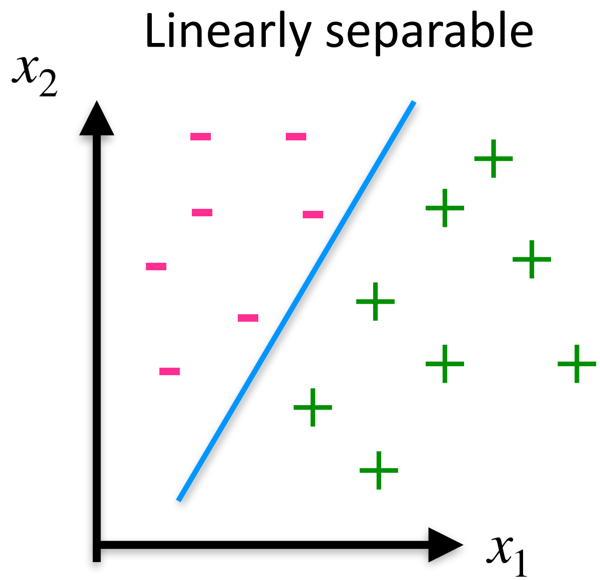
Linear separability

The training instances are linearly separable if there exists a hyperplane that will separate the two classes



Linear separability

If the training instances are linearly separable, eventually the perceptron will find $\mathbf{w}$ such that the classifier gets everything correct



Linear separability

If the training instances are not linearly separable, the classifier will always get some predictions wrong

- You need to implement some type of **stopping criteria** for when the algorithm will stop making updates, or it will run forever
- Usually this is specified by running the algorithm for a maximum number of **iterations** or **epochs**

Learning rate

Let's make a modification to the update rule:

$$w_j = w_j + \eta(y_i - f(\mathbf{x_i}))x_{ij}$$

where η is called the **learning rate** or **step size**

- When you update w_j to be more positive or negative, this controls the size of the change you make (or, how large a "step" you take.
- If $\eta = 1$ (a common value), then this is the same update rule from the earlier slide

Learning rate

How to choose the step size?

- If η is too small, the algorithm will be slow because the updates won't make much progress
- If η is too large, the algorithm will be slow because the updates will "overshoot" and may cause previous correct classification to become incorrect