Linear Classification and Perceptron

INFO-4604, Applied Machine Learning

University of Colorado Boulder

September 6, 2018

Prof. Michael Paul

Prediction Functions

Remember: a **prediction function** is the function that predicts what the output should be, given the input

Last time we looked at **linear functions**, which are commonly used as prediction functions.

Linear Functions

General form with *k* variables (arguments):

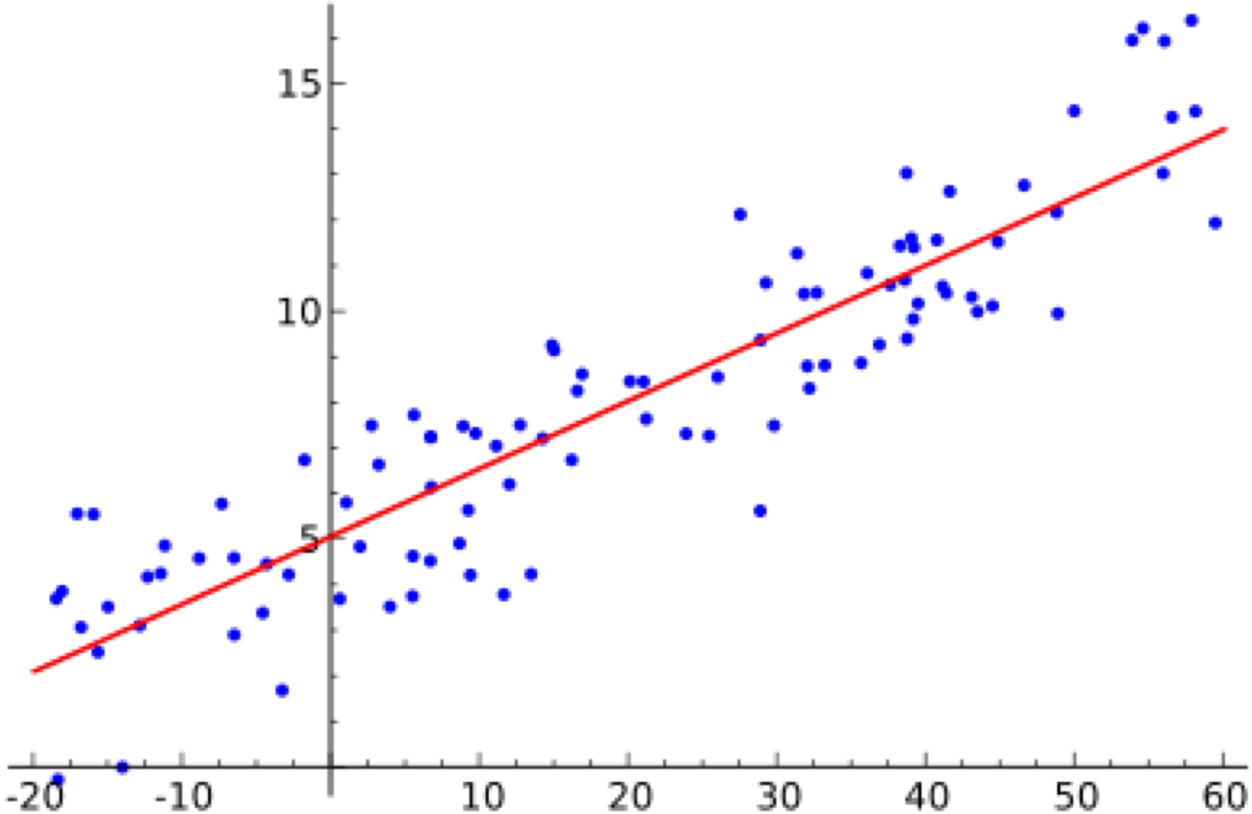
$$f(x_1,\ldots,x_k) = \sum_{i=1}^{k} m_i x_i \quad + b$$

or equivalently:

$$f(\mathbf{x}) = \mathbf{m}^T\mathbf{x} + b$$

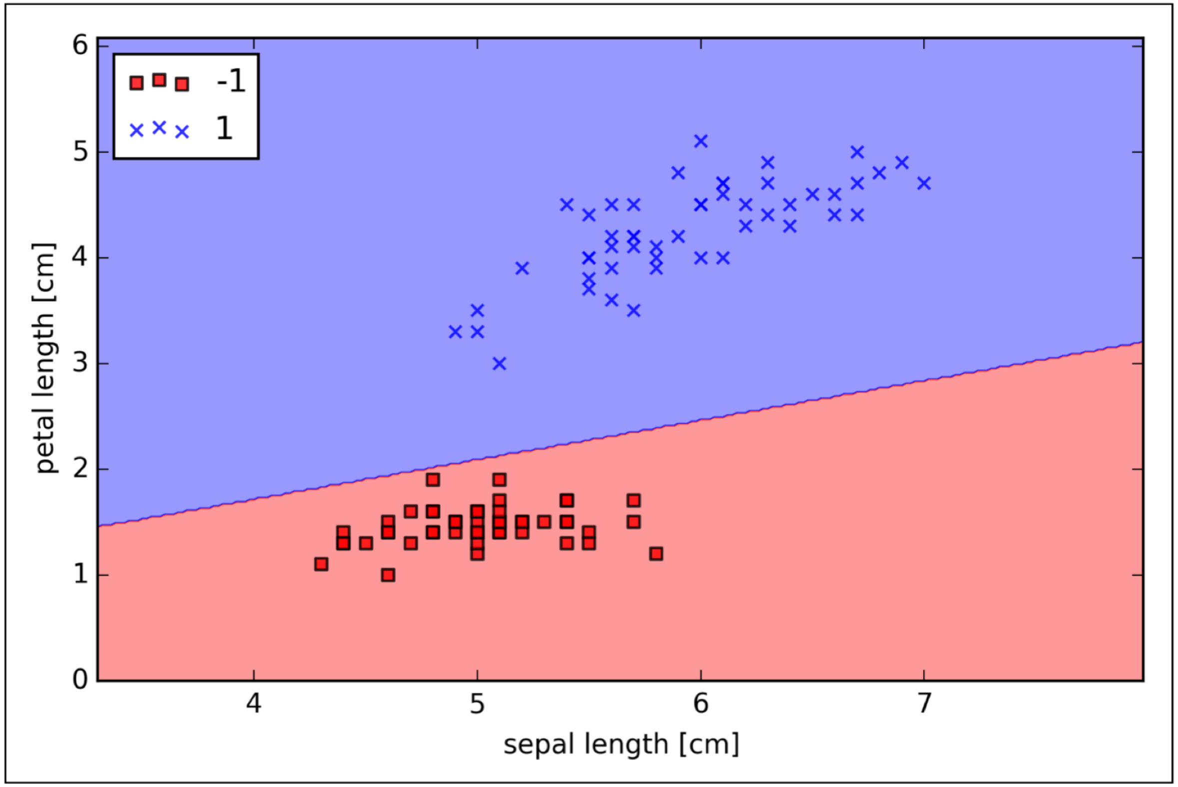
Linear Predictions

Regression:



Linear Predictions

Classification:



Learn a linear function that **separates** instances of different classes

Linear Classification

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 (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.
  - We'll discuss classification with >2 classes later on.

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, and the output is -1 if instead "$\mathbf{w}^T\mathbf{x} + b < 0$" 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 two 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 **w** (instead of m as we used last time).

- 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 **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.

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:

Searched "repair"	2.0
Searched "reviews"	8.0
Recent purchase	-15.0
Clicked ads before	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:

Searched “repair”	2.0
Searched “reviews”	8.0
Recent purchase	-15.0
Clicked ads before	5.0
b (intercept)	-9.0

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

$2*0 + 8*1 + -15*0 + 5*0 + -9 =$

$8 - 9 = -1$

Prediction:

No

Example

Suppose these are the weights:

Searched "repair"	2.0
Searched "reviews"	8.0
Recent purchase	-15.0
Clicked ads before	5.0
b (intercept)	-9.0

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

$2 + 8 - 9 = 1$

Prediction:

Yes

Example

Suppose these are the weights:

Searched “repair”	2.0
Searched “reviews”	8.0
Recent purchase	-15.0
Clicked ads before	5.0
b (intercept)	-9.0

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

$8 + 5 - 9 = 4$

Prediction:

Yes

Example

Suppose these are the weights:

Searched "repair"	2.0
Searched "reviews"	8.0
Recent purchase	-15.0
Clicked ads before	5.0
b (intercept)	-9.0

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

$8 - 15 + 5 - 9 = -11$

Prediction:

No

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

Example

Suppose these are the weights:

Searched "repair"	2.0
Searched "reviews"	8.0
Recent purchase	-15.0
Clicked ads before	5.0
b (intercept)	-9.0

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

-9

Prediction:

No

Since most people don't click ads, the "default" prediction is that they will not click (the intercept pushes it negative)

Learning the Weights

The perceptron algorithm learns the weights by:

1. Initialize all weights **w** to 0
2. Iterate through the training data. For each training instance, classify the instance.
   a) If the prediction (the output of the classifier) was correct, don't do anything. (It means the classifier is working, so leave it alone!)
   b) 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 **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:

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

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:

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

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 ($y_i = f(x_i)$).

- Then the entire update rule is 0, so no change is made.

Learning the Weights

The perceptron update rule:

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

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 $f(x_i) = -1$.
- This term is -2 if $y_i = -1$ and $f(x_i) = +1$.

The *sign* of this term indicates the direction of the mistake.

Learning the Weights

The perceptron update rule:

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

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:

- The $(y_i - f(x_i))$ term is +2 if $y_i = +1$ and $f(x_i) = -1$.
  - This will increase w_j (still assuming x_{ij} is 1)…
  - …which will increase $\mathbf{w}^T\mathbf{x_i} + b$…
  - …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:

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

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:

- The $(y_i - f(x_i))$ term is -2 if $y_i = -1$ and $f(x_i) = +1$.
  - This will decrease w_j (still assuming x_{ij} is 1)…
  - …which will decrease $\mathbf{w}^T\mathbf{x_i} + b$…
  - …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:

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

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: $\mathbf{w}^T\mathbf{x} + b = \mathbf{w}^T\mathbf{x} + b*1$.

- 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 the other weights.

Learning the Weights

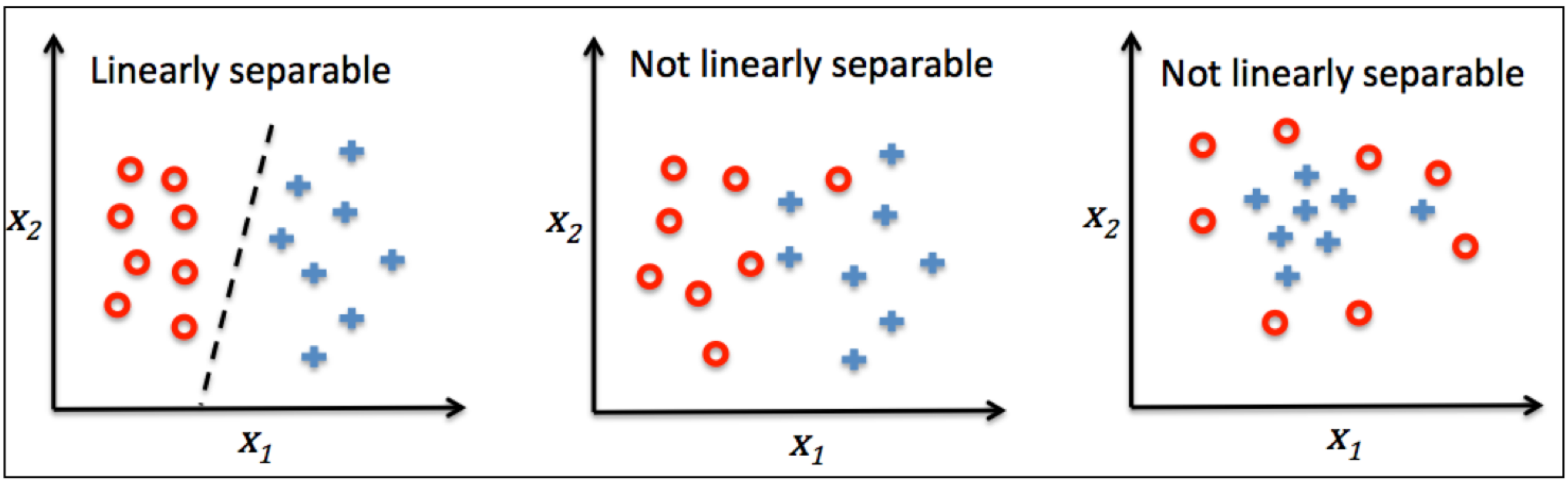
The vector of **w** values is called the **weight vector**.

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.

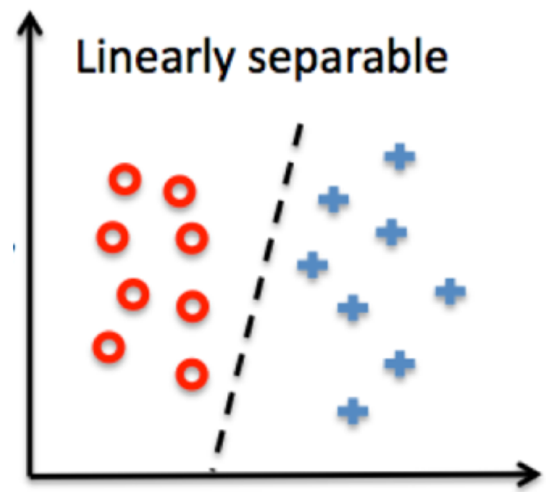
Linear Separability

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 algorithm will find weights **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:

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

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 η=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 previously correct classifications to become incorrect.

We'll learn about step sizes more next time.

Summary

Perceptron: Prediction

Prediction function:

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

Perceptron: Learning

1. Initialize all weights **w** to 0.
2. Iterate through the training data. For each training instance, classify the instance.
   a) If the prediction (the output of the classifier) was correct, don't do anything.
   b) If the prediction was wrong, modify the weights by using the **update rule**: $w_j \mathrel{+}= \eta \, (y_i - f(x_i)) \, x_{ij}$
3. Repeat step 2 until the perceptron correctly classifiers every instance or the maximum number of iterations has been reached.