Logistic Regression

INFO-4604, Applied Machine Learning
University of Colorado Boulder

September 14, 2017

Prof. Michael Paul

Linear Classification

$\mathbf{w}^T\mathbf{x_i}$ is the classifier **score** for the instance $\mathbf{x_i}$

The score can be used in different ways to make a classification.

- Perceptron: output positive class if score is at least 0, otherwise output negative class
- Today: output the *probability* that the instance belongs to a class

Activation Function

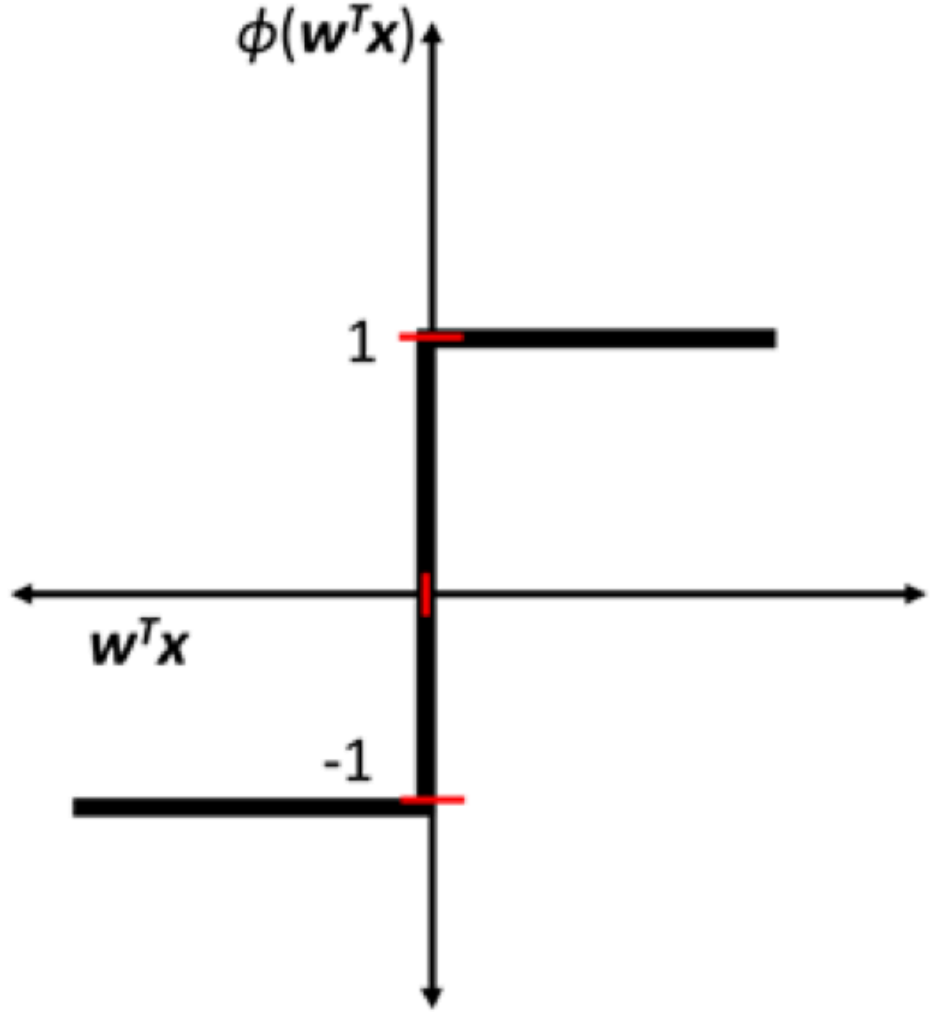
An **activation function** for a linear classifier converts the score to an output.

Denoted $\phi(z)$, where z refers to the score, $\mathbf{w}^T\mathbf{x_i}$

Activation Function

Perceptron uses a threshold function:

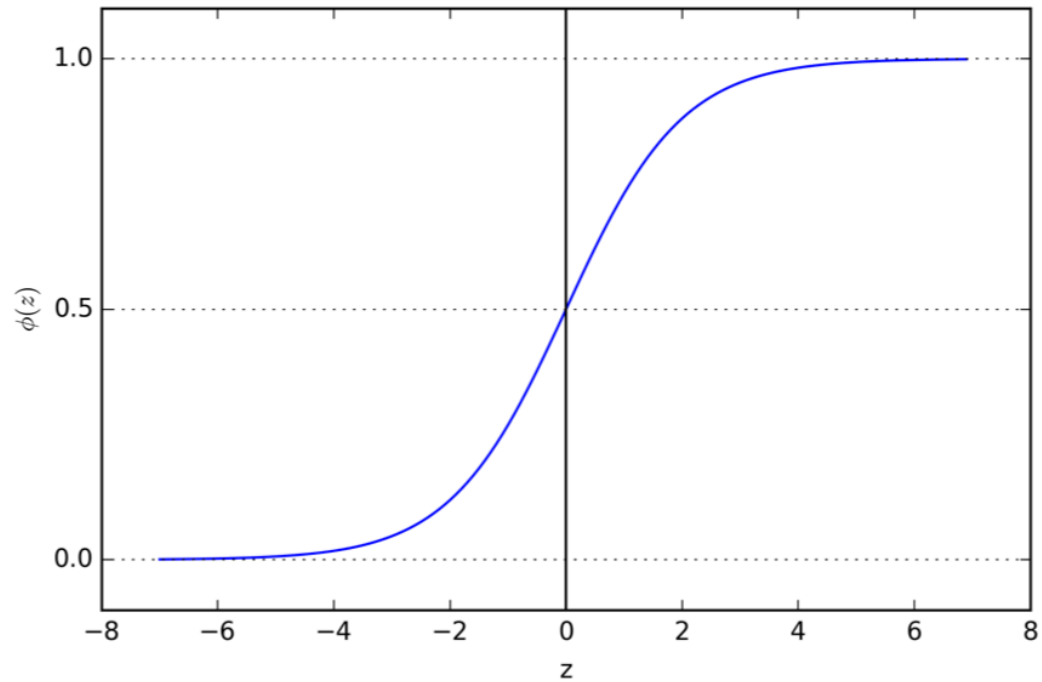
$$\phi(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$



Activation Function

Logistic function:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

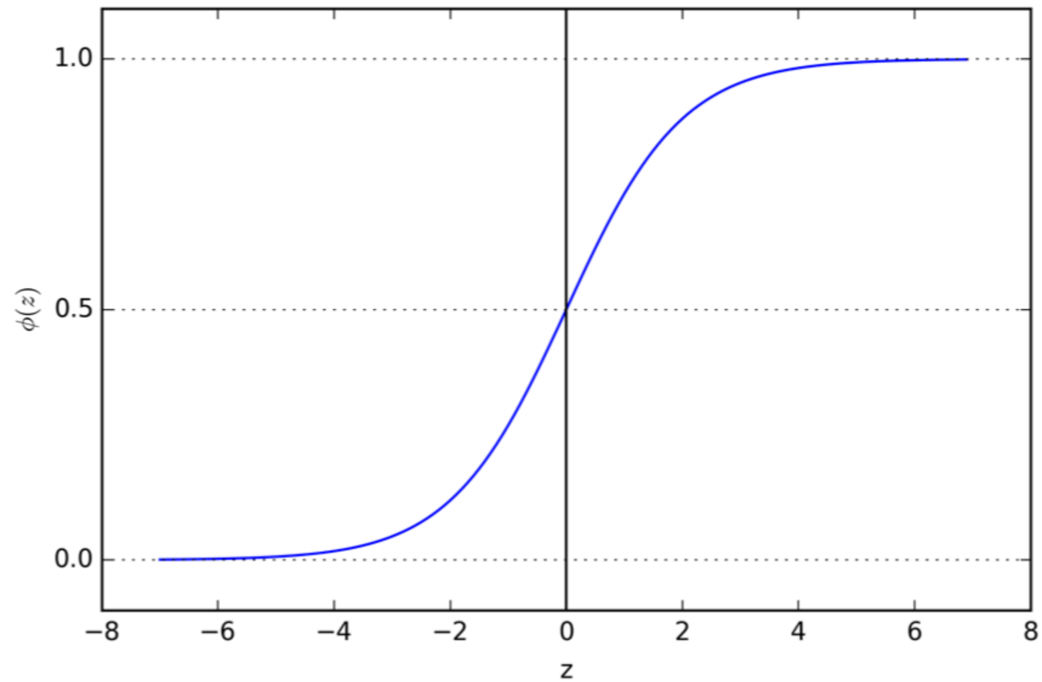


The logistic function is a type of *sigmoid* function (an S-shaped function)

Activation Function

Logistic function:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Outputs a real number between 0 and 1

Outputs 0.5 when z=0

Output goes to 1 as z goes to infinity

Output goes to 0 as z goes to negative infinity

Quick note on notation: $\exp(z) = e^z$

Logistic Regression

A linear classifier like perceptron that defines…

- Score: $\mathbf{w}^T\mathbf{x_i}$ (same as perceptron)
- Activation: logistic function (instead of threshold)

This classifier gives you a value between 0 and 1, usually interpreted as the probability that the instance belongs to the positive class.

- Final classification usually defined to be the positive class if the probability ≥ 0.5.

Logistic Regression

Confusingly:
This is a method for *classification*, not regression.

It is regression in that it is learning a function that outputs continuous values (the logistic function), BUT you are using those values to predict discrete classes.

Logistic Regression

Considered a linear classifier, even though the logistic function is not linear.

This is because the score is a linear function, which is really what determines the output.

Learning

How do we learn the parameters **w** for logistic regression?

Last time: need to define a **loss function** and find parameters that minimize it.

Probability

Because logistic regression's output is interpreted as a probability, we are going to define the loss function using probability.

For help with probability, review *OpenIntro Stats*, Ch 2.

Probability

A **conditional probability** is the probability of a random variable given that some variables are known.

$P(Y \mid X)$ is read as "the probability of Y given X" or "the probability of Y conditioned on X"

The variable on the left hand side is what you want to know the probability of.

The variable on the right-hand side is what you know.

Probability

$P(y_i = 1 \mid \mathbf{x_i}) = \phi(\mathbf{w}^T\mathbf{x_i})$
$P(y_i = 0 \mid \mathbf{x_i}) = 1 - \phi(\mathbf{w}^T\mathbf{x_i})$

Goal for learning: learn **w** that makes the labels in your training data more likely

- The probability of something you know to be true is 1, so that's what the probability should be of the labels in your training data.

Note: the convention for logistic regression is that the classes are 1 and 0 (instead of 1 and -1)

Learning

$$P(y_i \mid \mathbf{x_i}) = \phi(\mathbf{w}^T\mathbf{x_i})^{y_i} * (1 - \phi(\mathbf{w}^T\mathbf{x_i}))^{1-y_i}$$

Learning

$$P(y_i \mid \mathbf{x_i}) = \phi(\mathbf{w}^T\mathbf{x_i})^{y_i} * (1 - \phi(\mathbf{w}^T\mathbf{x_i}))^{1-y_i}$$

if $y_i = 1$

Learning

$$P(y_i \mid \mathbf{x_i}) = \phi(\mathbf{w}^T\mathbf{x_i})^{y_i} * (1 - \phi(\mathbf{w}^T\mathbf{x_i}))^{1-y_i}$$

if $y_i = 0$

Learning

$$P(y_i \mid \mathbf{x_i}) = \phi(\mathbf{w}^T\mathbf{x_i})^{y_i} * (1 - \phi(\mathbf{w}^T\mathbf{x_i}))^{1-y_i}$$

or

$$\log P(y_i \mid \mathbf{x_i}) = y_i \log(\phi(\mathbf{w}^T\mathbf{x_i})) + (1-y_i) \log(1-\phi(\mathbf{w}^T\mathbf{x_i}))$$

Taking the logarithm (base *e*) of the probability makes the math work out easier.

Learning

$\log P(y_i \mid \mathbf{x_i}) = y_i \log(\phi(\mathbf{w}^T\mathbf{x_i})) + (1-y_i) \log(1-\phi(\mathbf{w}^T\mathbf{x_i}))$

This is the log of the probability of an instance's label y_i given the instance's feature vector $\mathbf{x_i}$

What about the probability of all the instances?

$$\sum_{i=1}^{N} \log P(y_i \mid \mathbf{x_i})$$

This is called the **log-likelihood** of the dataset.

Learning

Our goal was to define a loss function for logistic regression. Let's use log-likelihood… almost.

A loss function refers specifically to something you want to minimize (that's why it's called "loss"), but we want to *maximize* probability!

So let's minimize the *negative* log-likelihood:

$$L(\mathbf{w}) = \sum_{i=1}^{N} -\log P(y_i \mid \mathbf{x_i}) = \sum_{i=1}^{N} -y_i \log(\phi(\mathbf{w}^T\mathbf{x_i})) - (1-y_i)\log(1-\phi(\mathbf{w}^T\mathbf{x_i}))$$

Learning

We can use gradient descent to minimize the negative log-likelihood, $L(\mathbf{w})$

The partial derivative of L with respect to w_j is:

$$dL/dw_j = \sum_{i=1}^{N} x_{ij} \left(y_i - \phi(\mathbf{w}^T \mathbf{x_i})\right)$$

Learning

We can use gradient descent to minimize the negative log-likelihood, L(**w**)

The partial derivative of L with respect to w_j is:

$$dL/dw_j = \sum_{i=1}^{N} x_{ij} \, (y_i - \phi(\mathbf{w}^T \mathbf{x_i}))$$

if $y_i = 1$…

The derivative will be 0 if $\phi(\mathbf{w}^T\mathbf{x_i})=1$

(that is, the probability that y_i=1 is 1, according to the classifier)

Learning

We can use gradient descent to minimize the negative log-likelihood, L(**w**)

The partial derivative of L with respect to w_j is:

$$dL/dw_j = \sum_{i=1}^{N} x_{ij} \left(y_i - \phi(\mathbf{w}^T \mathbf{x_i})\right)$$

if $y_i = 1$…

The derivative will be positive if $\phi(\mathbf{w}^T \mathbf{x_i}) < 1$

(the probability was an underestimate)

Learning

We can use gradient descent to minimize the negative log-likelihood, L(**w**)

The partial derivative of L with respect to w_j is:

$$dL/dw_j = \sum_{i=1}^{N} x_{ij} \left(y_i - \phi(\mathbf{w}^T \mathbf{x_i})\right)$$

if $y_i = 0$…

The derivative will be 0 if $\phi(\mathbf{w}^T\mathbf{x_i})=0$

(that is, the probability that $y_i=0$ is 1, according to the classifier)

Learning

We can use gradient descent to minimize the negative log-likelihood, L(**w**)

The partial derivative of L with respect to w_j is:

$$dL/dw_j = \sum_{i=1}^{N} x_{ij} \, (y_i - \phi(\mathbf{w}^T \mathbf{x_i}))$$

if $y_i = 0$…

The derivative will be negative if $\phi(\mathbf{w}^T \mathbf{x_i}) > 0$

(the probability was an overestimate)

Learning

We can use gradient descent to minimize the negative log-likelihood, $L(\mathbf{w})$

The partial derivative of L with respect to w_j is:

$$dL/dw_j = \sum_{i=1}^{N} x_{ij} \left(y_i - \phi(\mathbf{w}^T \mathbf{x_i})\right)$$

So the gradient descent update for each w_j is:

$$w_j \mathrel{+}= \eta \sum_{i=1}^{N} x_{ij} \left(y_i - \phi(\mathbf{w}^T \mathbf{x_i})\right)$$

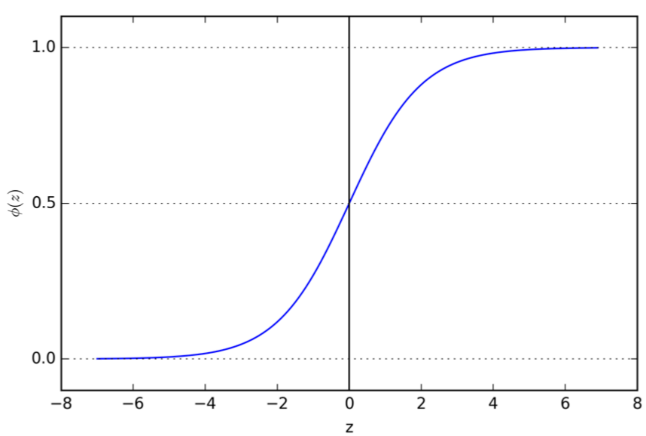
Learning

So gradient descent is trying to…

- make $\phi(\mathbf{w}^T\mathbf{x_i}) = 1$ if $y_i = 1$
- make $\phi(\mathbf{w}^T\mathbf{x_i}) = 0$ if $y_i = 0$

But there's a problem…

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



z would have to be ∞ (or -∞) in order to make $\phi(z)$ equal to 1 (or 0)

Learning

So gradient descent is trying to…

- make $\phi(\mathbf{w}^T\mathbf{x_i}) \cancel{=} 1$ if $y_i = 1$
- make $\phi(\mathbf{w}^T\mathbf{x_i}) \cancel{=} 0$ if $y_i = 0$

Instead, make
"close" to 1 or 0

Don't want to optimize "too" much while running gradient descent

Learning

So gradient descent is trying to…

- make $\phi(\mathbf{w}^T\mathbf{x_i}) \cancel{=} 1$ if $y_i = 1$
- make $\phi(\mathbf{w}^T\mathbf{x_i}) \cancel{=} 0$ if $y_i = 0$

Instead, make
"close" to 1 or 0

We can modify the loss function that basically means, get as close to 1 or 0 as possible but without making the **w** parameters too extreme.

- How? That's for next time.

Learning

Remember from last time:

- Gradient descent
  - Uses the full gradient
- Stochastic gradient descent (SGD)
  - Uses an approximate of the gradient based on a single instance
  - Iteratively update the weights one instance at a time

Logistic regression can use either, but SGD more common, and is usually faster.

Prediction

The probabilities give you an estimate of the confidence of the classification.

Typically you classify something positive if $\phi(\mathbf{w}^T\mathbf{x_i}) \geq 0.5$, but you could create other rules.

- If you don't want to classify something as positive unless you're really confident, use $\phi(\mathbf{w}^T\mathbf{x_i}) \geq 0.99$ as your rule.

    Example: spam classification

    - Maybe worse to put a legitimate email in the spam box than to put a spam email in the inbox
    - Want high confidence before calling something spam

Other Disciplines

Logistic regression is used in other ways.

- Machine learning is focused on prediction (outputting something you don’t know).
- Many disciplines is it as a tool to understand relationships between variables.

  What demographics are correlated with smoking?

  Build a model that “predicts” if someone is a smoker based on some variables
  (e.g., age, education, income).

  The parameters can tell you which variables increase or decrease the likelihood of smoking.