

Lecture 3: logistic regression, SGD, and mini-batching

Jikael Gagnon (Based on slides by Sheheryar Parvaz and Shidan Javaheri)

McGill Artificial Intelligence Society

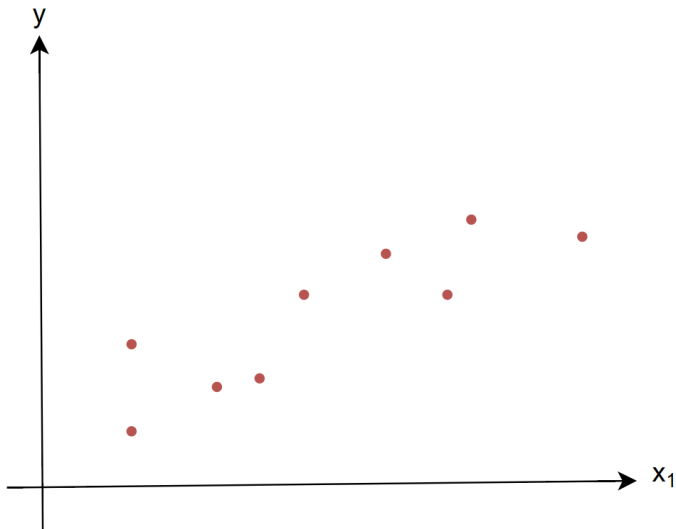
Winter 2026



admin

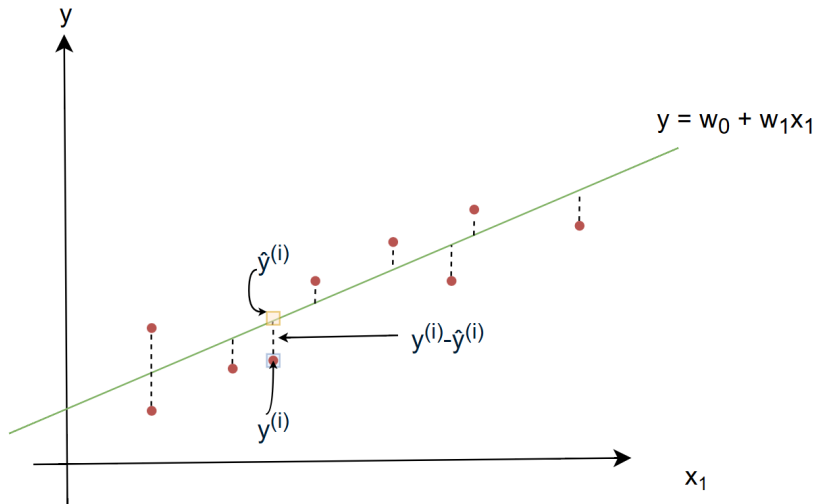
- Assignment 1 released last week - due March 1st
- Deliverable 2 released today - due March 1st

review: linear regression



review: linear regression

Our goal is to find the line of best fit of the data:



review: linear regression

We assume that the relationship between input and output is linear:

- **1-dimensional input:** $\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)}$

review: linear regression

We assume that the relationship between input and output is linear:

- **1-dimensional input:** $\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)}$
- **D -dimensional input:** $\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_D x_D^{(i)}$

where $\hat{y}^{(i)}$ is our prediction for input $x^{(i)}$

review: linear regression

This can be rewritten as a dot product:

$$\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_D x_D^{(i)} = w_0 + \vec{w}^\top \cdot \vec{x}$$

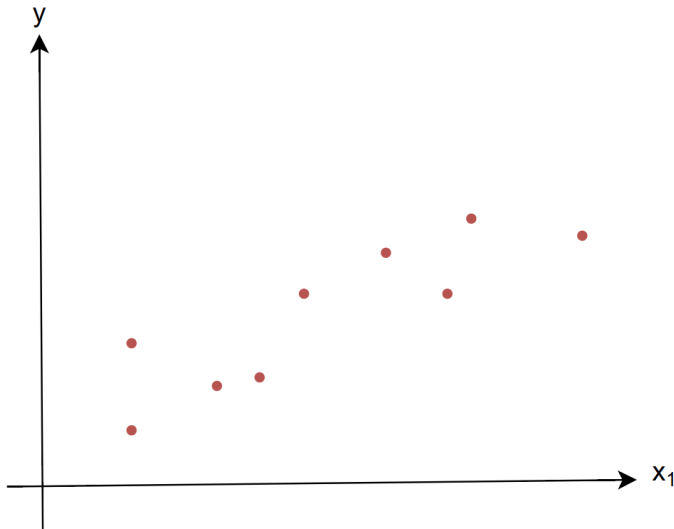
for $\vec{x}^{(i)}, \vec{w} \in \mathbb{R}^D$

review: linear regression

Aside: I'm being explicit to write $\vec{w}^\top$ because many textbooks/resources assume that every vector is a column vector. For a dot product to give you a scalar, you need to take the dot of a row vector with a column vector.

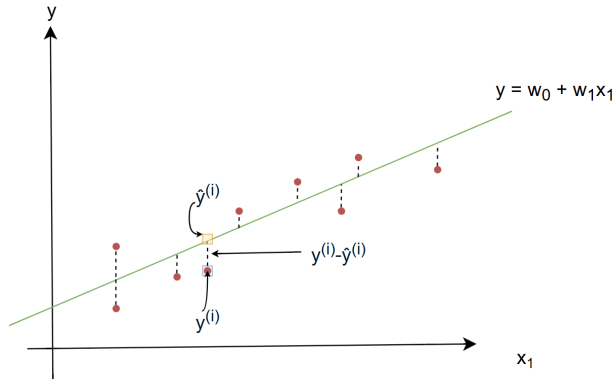
review: linear regression

Our goal is to find the line that minimizes the *sum of squared error* (SSE) of our data:



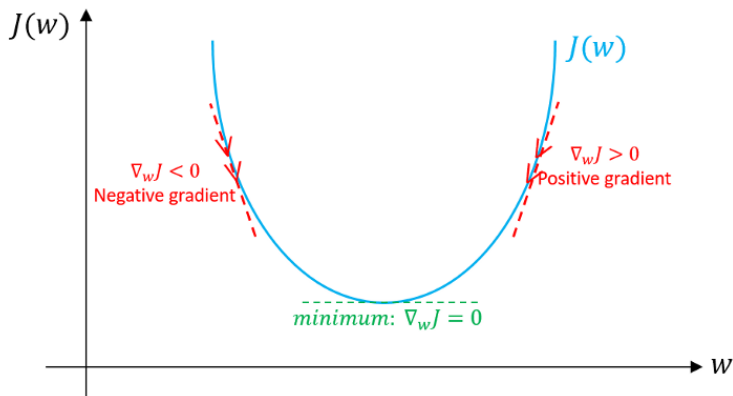
review: linear regression

$$J(\vec{w}) = \text{SSE}(\vec{w}) = \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)})^2 = \sum_{i=1}^N (y^{(i)} - (w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^2$$



review: gradient descent

Gradient descent iteratively finds a minimum by taking steps proportional to the gradient:



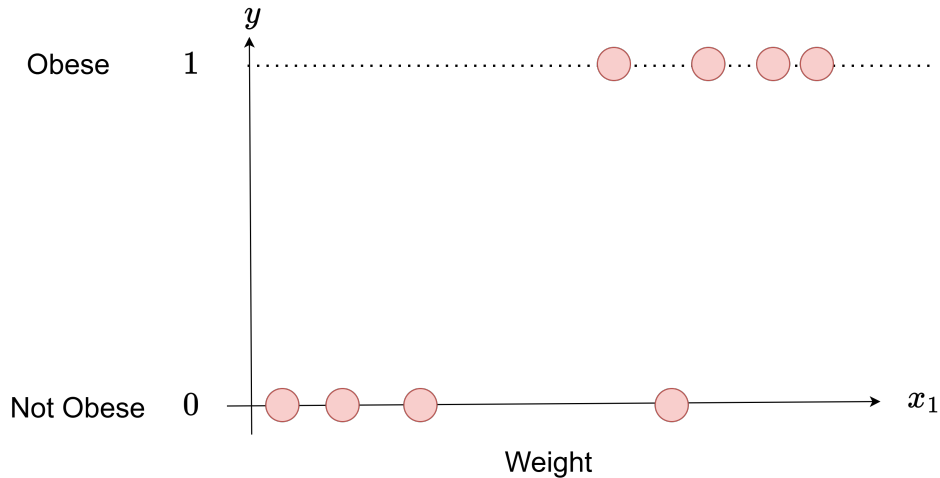
logistic regression

our goal

Our dataset consists of weights of mice (x_1) and binary labels (obese or not obese):

#	weight(x_1)	label (y)
1	5	<i>Obese (1)</i>
2	2	<i>Not Obese (0)</i>
$\vdots$	$\vdots$	$\vdots$
9	8	<i>Obese (1)</i>

our goal

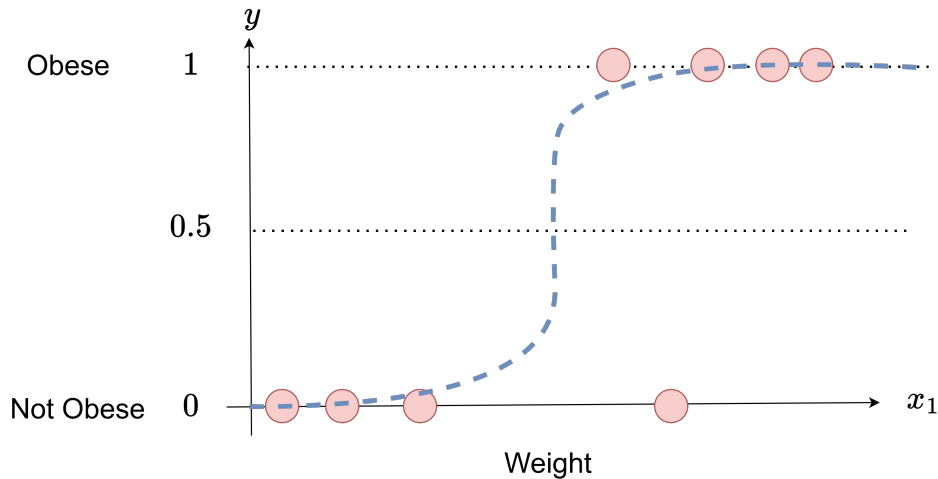


our goal

Instead of predicting the class directly, we will find a curve that predicts the probability our input is in class 1 (ie. probability the mouse is obese):

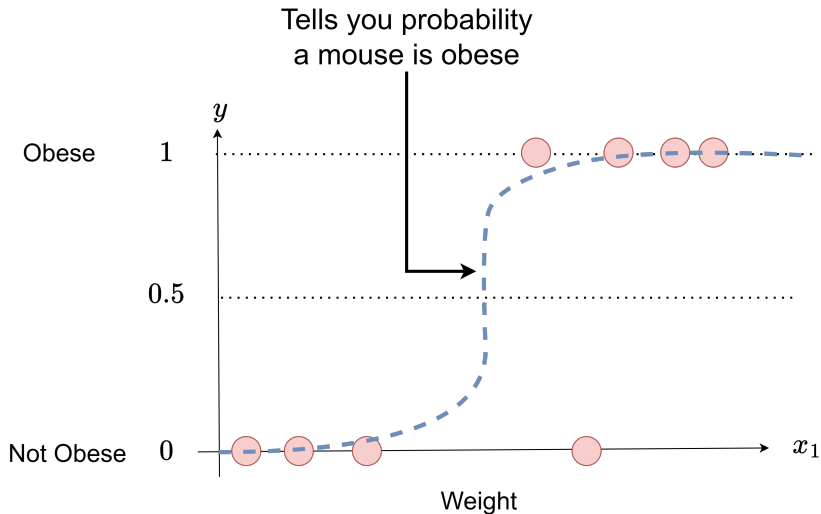
our goal

Instead of predicting the class directly, we will find a curve that predicts the probability our input is in class 1 (ie. probability the mouse is obese):

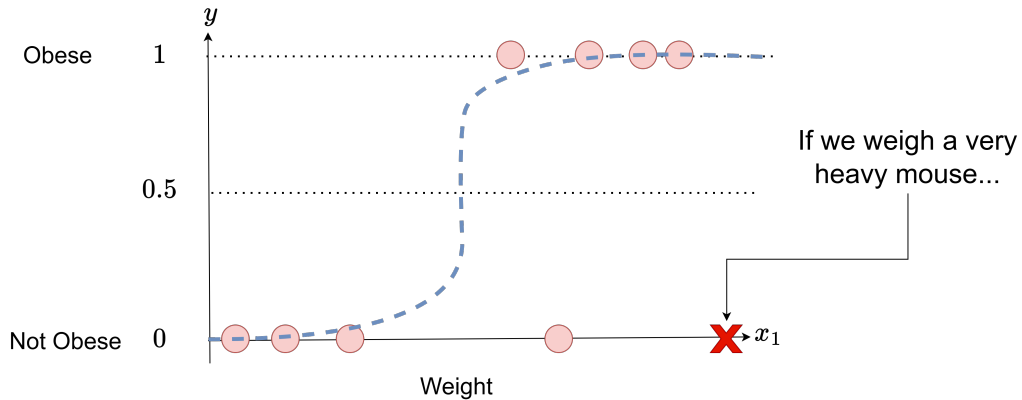


our goal

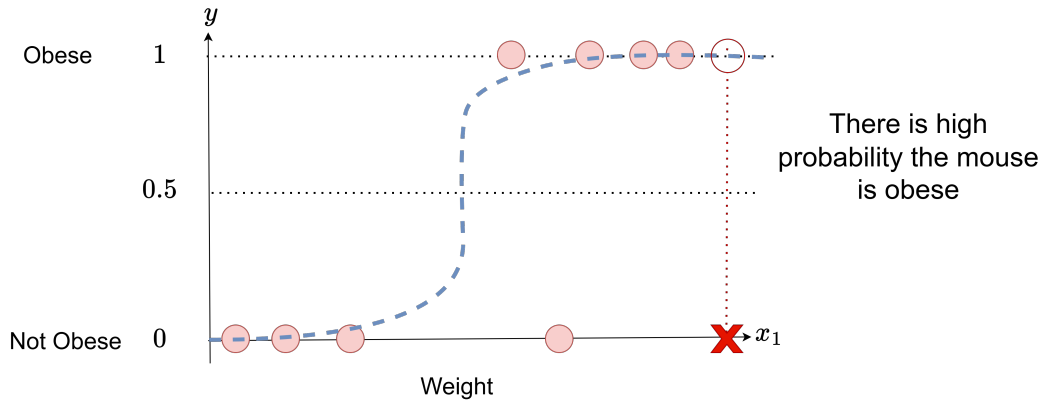
Note that the y -axis is being used to represent the class **and** the probability!



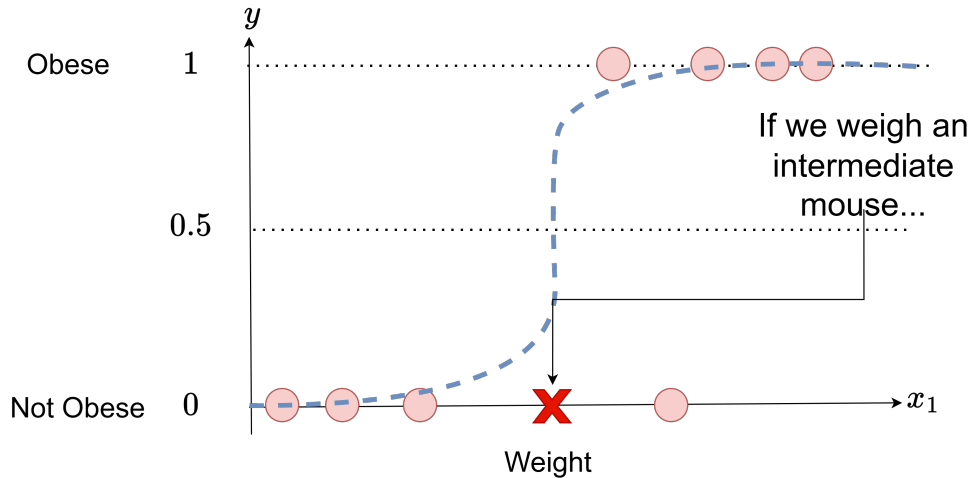
our goal



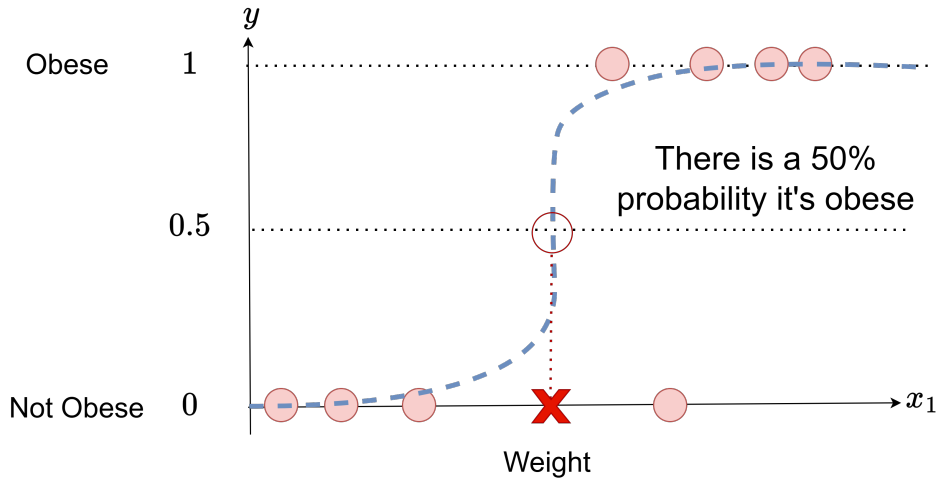
our goal



our goal



our goal



notation alert!!!

Since we're talking about probabilities, let's introduce some notation.
Read the following as “the probability $Y = 1$ given that $X = \vec{x}$ ”:

$$P(Y = 1 | X = \vec{x})$$

notation alert!!!

Since we're talking about probabilities, let's introduce some notation.
Read the following as “the probability $Y = 1$ given that $X = \vec{x}$ ”:

$$P(Y = 1|X = \vec{x})$$

Following our mouse example,

$$P(Y = 1|X = 5) = 0.9$$

means “the probability that our mouse is obese, given that it weighs 5 pounds is 90%”

notation alert!!!

Our squiggly line is modelling

$$\tilde{P}(Y = 1|X = \vec{x})$$

notation alert!!!

Our squiggly line is modelling

$$\tilde{P}(Y = 1|X = \vec{x})$$

$\tilde{P}$ will be used to denote that it's an *estimate* of the true probability distribution P .

notation alert!!!

Our squiggly line is modelling

$$\tilde{P}(Y = 1|X = \vec{x})$$

$\tilde{P}$ will be used to denote that it's an *estimate* of the true probability distribution P .

Our squiggly line will be learned using our data, but as usual, our dataset might not capture the true distribution of the entire population.

notation alert!!!

Our squiggly line is modelling

$$\tilde{P}(Y = 1|X = \vec{x})$$

$\tilde{P}$ will be used to denote that it's an *estimate* of the true probability distribution P .

Our squiggly line will be learned using our data, but as usual, our dataset might not capture the true distribution of the entire population. In the real world we almost never have access to the true distribution of the data, so we're always estimating it.

Hopefully, if our data is a representative sample of the population, we can learn probabilities that are close to the true probability distribution.

how should our model work?

Question: Given that our model predicts probabilities, how should we use it for classification?

how should our model work?

Question: Given that our model predicts probabilities, how should we use it for classification?

Answer: Pick the class with the highest probability

how should our model work?

Question: Given that our model predicts probabilities, how should we use it for classification?

Answer: Pick the class with the highest probability

- If $\tilde{P}(Y = 1|X = \vec{x}^{(i)}) > 0.5$, predict class 1

how should our model work?

Question: Given that our model predicts probabilities, how should we use it for classification?

Answer: Pick the class with the highest probability

- If $\tilde{P}(Y = 1|X = \vec{x}^{(i)}) > 0.5$, predict class 1
- If $\tilde{P}(Y = 1|X = \vec{x}^{(i)}) < 0.5$, predict class 0

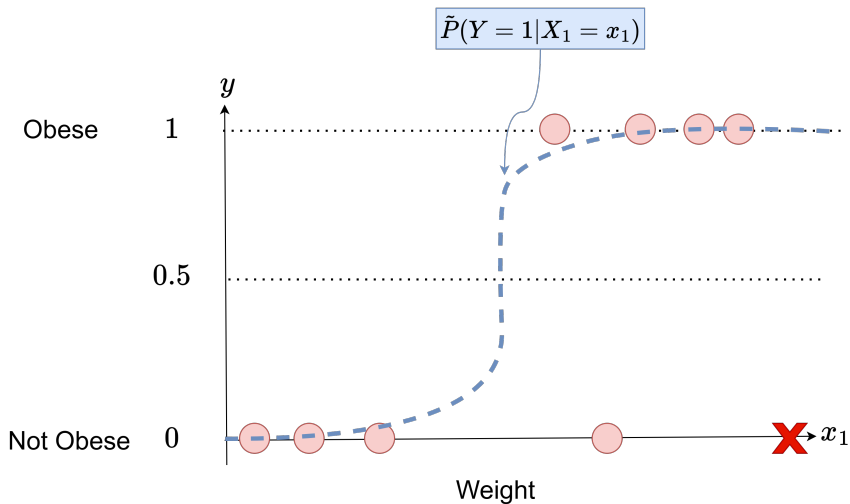
how should our model work?

Question: Given that our model predicts probabilities, how should we use it for classification?

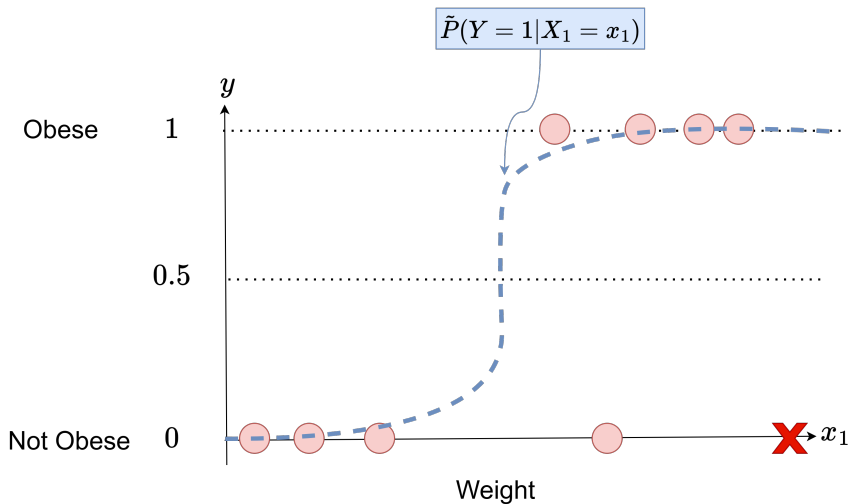
Answer: Pick the class with the highest probability

- If $\tilde{P}(Y = 1|X = \vec{x}^{(i)}) > 0.5$, predict class 1
- If $\tilde{P}(Y = 1|X = \vec{x}^{(i)}) < 0.5$, predict class 0
- If $\tilde{P}(Y = 1|X = \vec{x}^{(i)}) = 0.5$, it's up to you (eg. default to class 1, coin flip, etc...)

How would we classify this point?

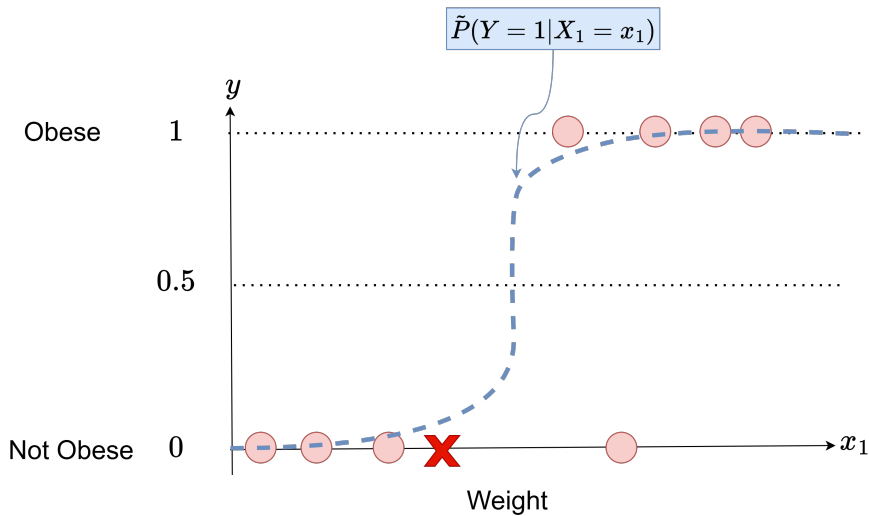


How would we classify this point?

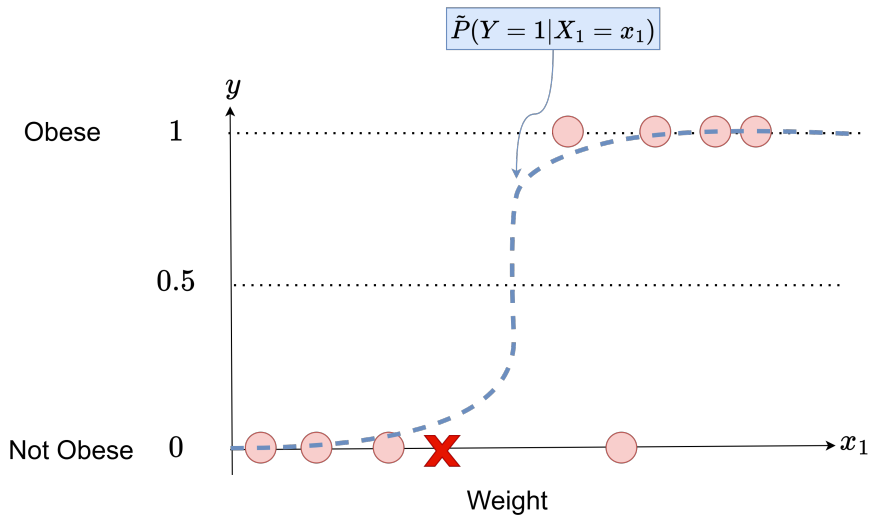


Answer: Obese, since $P(Y = 1 | X_1 = x_1) > 0.5$

And this one?

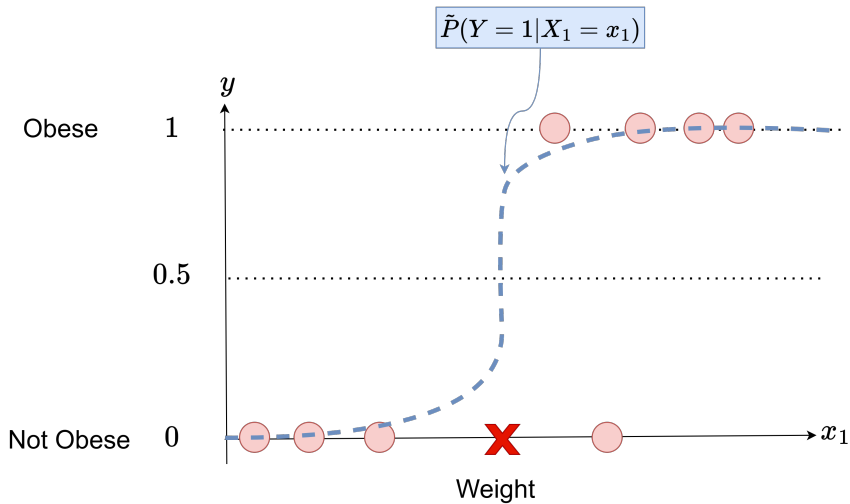


And this one?

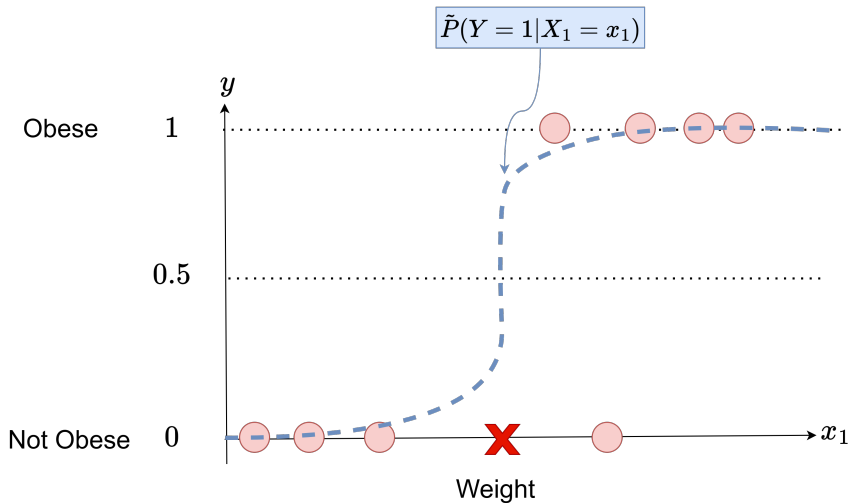


Answer: Not obese, since $P(Y = 1|X_1 = x_1) < 0.5$

How about this one?



How about this one?



Answer: Up to you, since $P(Y = 1 | X_1 = x_1) = 0.5$

our prediction function

How should we predict probabilities?

our prediction function

How should we predict probabilities?

We need a function that maps our inputs to probabilities. Probabilities are between 0 and 1.

our prediction function

How should we predict probabilities?

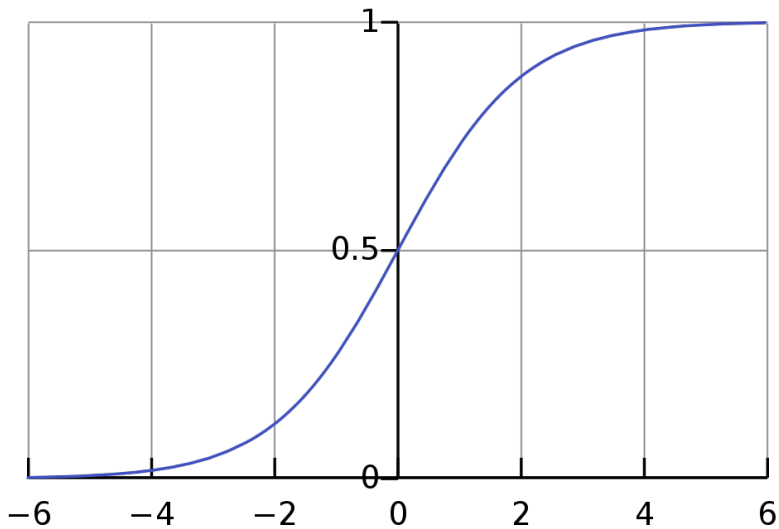
We need a function that maps our inputs to probabilities. Probabilities are between 0 and 1.

In logistic regression we use the “logistic” or “sigmoid” function:

$$\sigma: \mathbb{R} \rightarrow (0, 1)$$

defined as

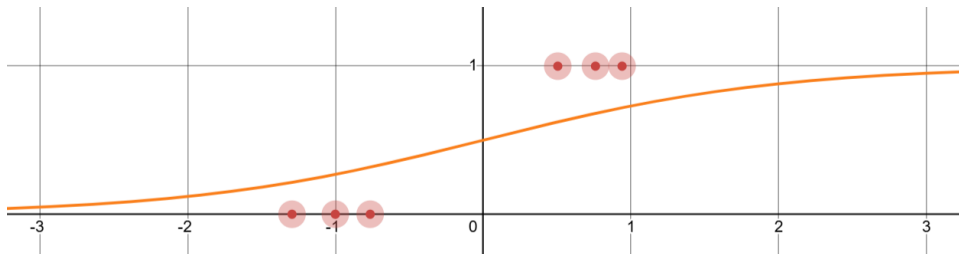
$$\sigma(x) = \frac{1}{1 + e^{-x}} = \tilde{P}(Y = 1|X = x)$$



The function is centered at 0 (ie. $\tilde{P}(Y = 1|X = 0) = 0.5$)

The function is centered at 0 (ie. $\tilde{P}(Y = 1|X = 0) = 0.5$)

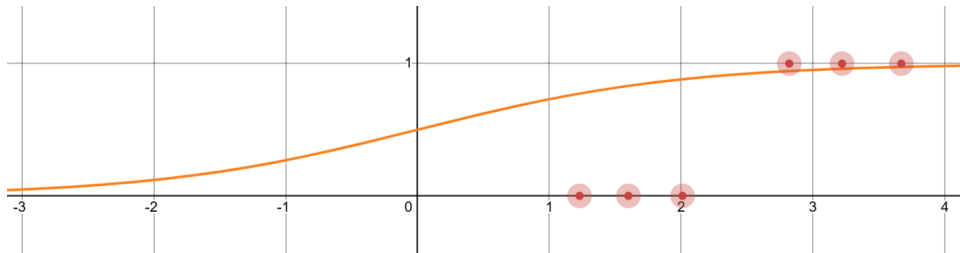
Thus, it works well if class 0 has values of $x < 0$ and class 1 has values of $x > 0$:



The function is centered at 0 (ie. $\tilde{P}(Y = 1|X = 0) = 0.5$)

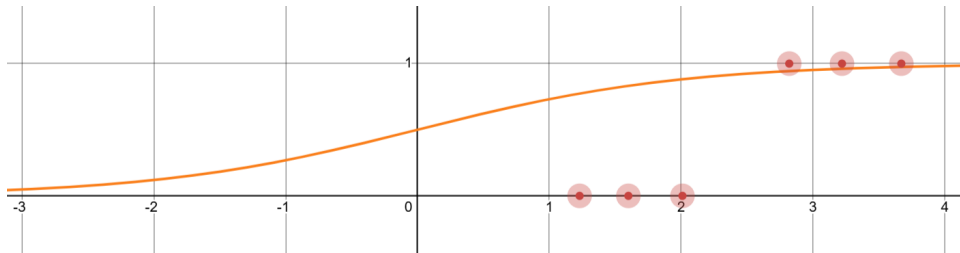
The function is centered at 0 (ie. $\tilde{P}(Y = 1|X = 0) = 0.5$)

This won't always be a good fit for our data:



The function is centered at 0 (ie. $\tilde{P}(Y = 1|X = 0) = 0.5$)

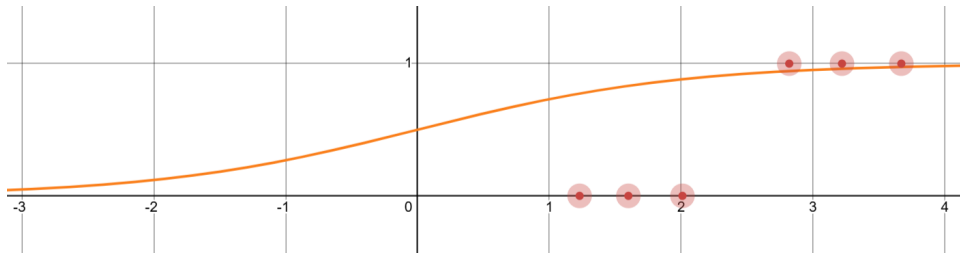
This won't always be a good fit for our data:



In this case, our model classifies everything as class 1 (bad)!

The function is centered at 0 (ie. $\tilde{P}(Y = 1|X = 0) = 0.5$)

This won't always be a good fit for our data:



In this case, our model classifies everything as class 1 (bad)!

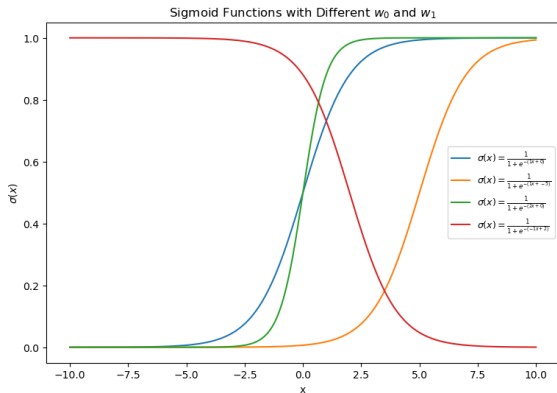
What should we do?

We can transform our curve by applying a linear transformation to our data before feeding it as input:

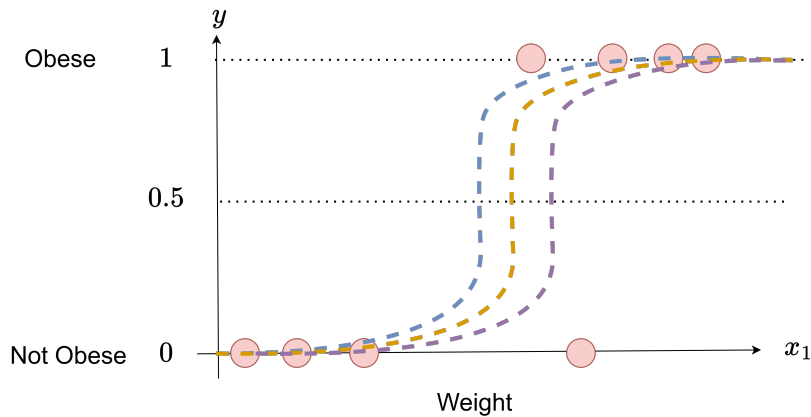
$$\sigma(w_1 x_1 + w_0) = \frac{1}{1 + e^{-(w_1 x + w_0)}}$$

We can transform our curve by applying a linear transformation to our data before feeding it as input:

$$\sigma(w_1x_1 + w_0) = \frac{1}{1 + e^{-(w_1x + w_0)}}$$



We can transform our curve until we find the version that models our data the best:



More generally, we can do this with a D dimensional input, just like in linear regression:

$$\sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

Example: 3D-Input

Let's predict whether the mouse is obese based on weight (x_1), height (x_2), and body fat percentage (x_3):

Example: 3D-Input

Let's predict whether the mouse is obese based on weight (x_1), height (x_2), and body fat percentage (x_3):

Given:

- $\vec{x}^{(i)} = [x_1, x_2, x_3] = [5, 3, 0.2]$
- $\vec{w} = [w_0, w_1, w_2, w_3] = [0.1, 2, 1, 3]$

Example: 3D-Input

Let's predict whether the mouse is obese based on weight (x_1), height (x_2), and body fat percentage (x_3):

Given:

- $\vec{x}^{(i)} = [x_1, x_2, x_3] = [5, 3, 0.2]$
- $\vec{w} = [w_0, w_1, w_2, w_3] = [0.1, 2, 1, 3]$

$$w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + w_3 x_3^{(i)} = 0.1 + 5(2) + 3(1) + 0.2(3) = 13.7$$

Example: 3D-Input

Let's predict whether the mouse is obese based on weight (x_1), height (x_2), and body fat percentage (x_3):

Given:

- $\vec{x}^{(i)} = [x_1, x_2, x_3] = [5, 3, 0.2]$
- $\vec{w} = [w_0, w_1, w_2, w_3] = [0.1, 2, 1, 3]$

$$w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + w_3 x_3^{(i)} = 0.1 + 5(2) + 3(1) + 0.2(3) = 13.7$$

$$\sigma(13.7) = \frac{1}{1 + e^{-13.7}} = 0.99$$

Example: 3D-Input

Let's predict whether the mouse is obese based on weight (x_1), height (x_2), and body fat percentage (x_3):

Given:

- $\vec{x}^{(i)} = [x_1, x_2, x_3] = [5, 3, 0.2]$
- $\vec{w} = [w_0, w_1, w_2, w_3] = [0.1, 2, 1, 3]$

$$w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + w_3 x_3^{(i)} = 0.1 + 5(2) + 3(1) + 0.2(3) = 13.7$$

$$\sigma(13.7) = \frac{1}{1 + e^{-13.7}} = 0.99$$

Thus, we predict that there is a 99% chance this mouse is obese.

Given our weights $w_0, w_1, \dots, w_D$ we can define our prediction function f as:

$$f(x) = \begin{cases} 1 & \text{if } \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) > 0.5 \\ 0 & \text{if } \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) < 0.5 \end{cases}$$

and do whatever we want if $\sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = 0.5$

aside: why do we call it logistic *regression*?

At this point, there may be some confusion about the name of this model. Why do we call it logistic *regression*, if we're using it for classification? Shouldn't it be called logistic classification?

aside: why do we call it logistic *regression*?

At this point, there may be some confusion about the name of this model. Why do we call it logistic *regression*, if we're using it for classification? Shouldn't it be called logistic classification?

The logistic regression model is simply the following:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

aside: why do we call it logistic *regression*?

At this point, there may be some confusion about the name of this model. Why do we call it logistic *regression*, if we're using it for classification? Shouldn't it be called logistic classification?

The logistic regression model is simply the following:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

This model predicts *probabilities*, not classes. Probabilities are continuous, so this is indeed regression.

aside: why do we call it logistic *regression*?

At this point, there may be some confusion about the name of this model. Why do we call it logistic *regression*, if we're using it for classification? Shouldn't it be called logistic classification?

The logistic regression model is simply the following:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

This model predicts *probabilities*, not classes. Probabilities are continuous, so this is indeed regression.

This model can be (and almost always is) used *in* classification, but that only works after adding the additional rule that we classify based on the probabilities. Thus, this lecture is really about using logistic regression *in* classification.

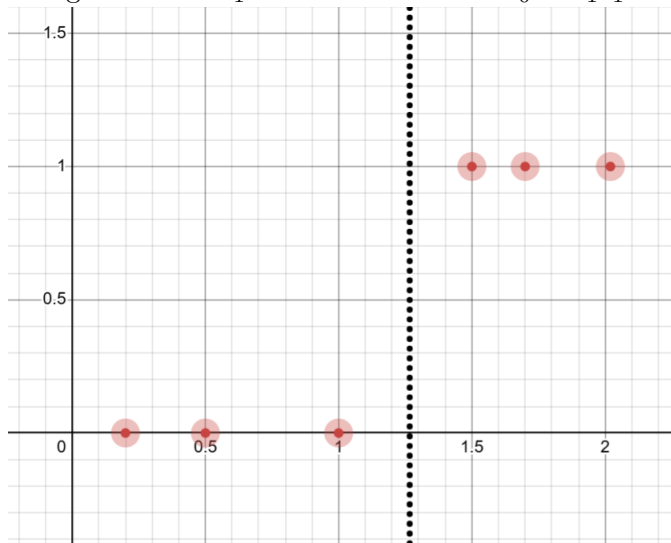
Demo in 1D: <https://www.desmos.com/calculator/ltwxit0m1t>

geometric interpretation of logistic regression

In 1D, there is a single value of x_1 such that $\vec{w}^\top \cdot \vec{x} = w_0 + w_1 x_1 = 0$ (dotted line):

geometric interpretation of logistic regression

In 1D, there is a single value of x_1 such that $\vec{w}^\top \cdot \vec{x} = w_0 + w_1 x_1 = 0$ (dotted line):



geometric interpretation of logistic regression

Similarly, the set of values of x_1 satisfying

- $\vec{w}^\top \cdot \vec{x} = w_0 + w_1 x_1 > 0$ (green)
- $\vec{w}^\top \cdot \vec{x} = w_0 + w_1 x_1 < 0$ (purple)

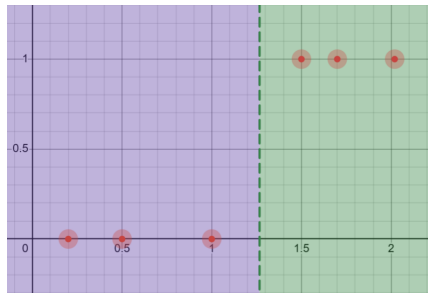
form *regions*

geometric interpretation of logistic regression

Similarly, the set of values of x_1 satisfying

- $\vec{w}^\top \cdot \vec{x} = w_0 + w_1 x_1 > 0$ (green)
- $\vec{w}^\top \cdot \vec{x} = w_0 + w_1 x_1 < 0$ (purple)

form *regions*

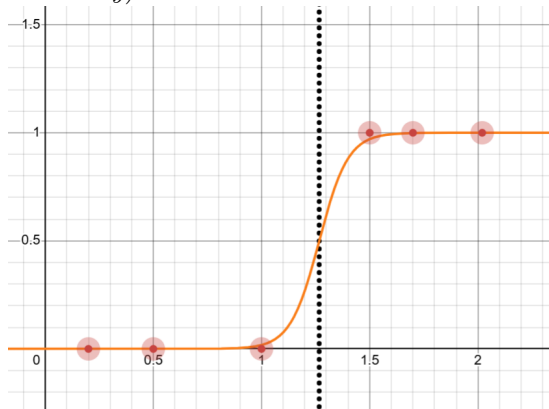


geometric interpretation of logistic regression

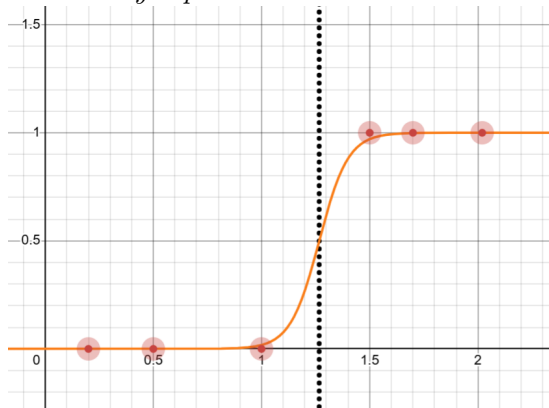
We refer to the line $\vec{w}^\top \cdot \vec{x} = 0$ as a *decision boundary* (ie. we *decide* how to classify points based on the *boundary*).

geometric interpretation of logistic regression

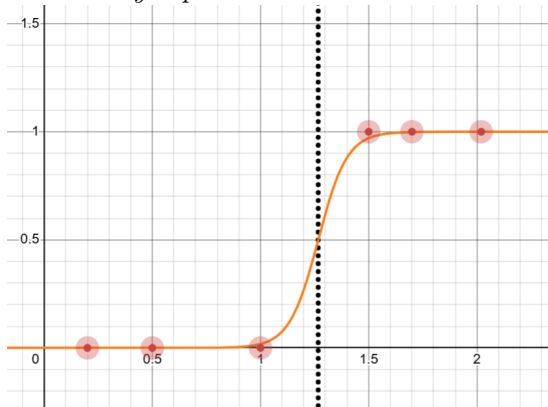
We refer to the line $\vec{w}^\top \cdot \vec{x} = 0$ as a *decision boundary* (ie. we *decide* how to classify points based on the *boundary*).



If there exists a decision boundary that correctly splits our data into two classes, we say that our data is *linearly seperable*.

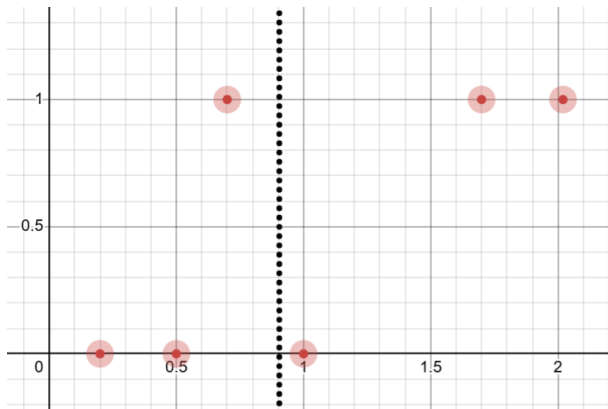


If there exists a decision boundary that correctly splits our data into two classes, we say that our data is *linearly seperable*.

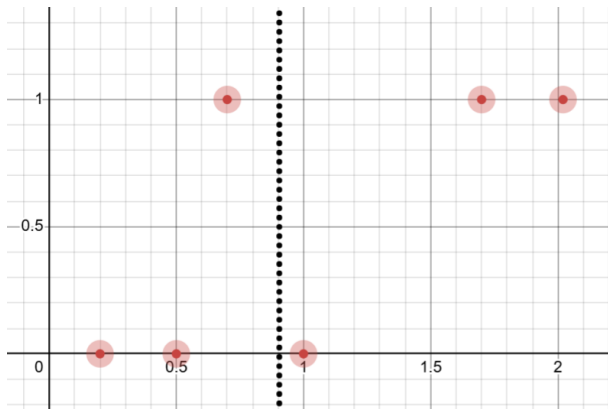


If this is the case, our model can achieve 100% accuracy!

Sometimes the data is not linearly seperable:



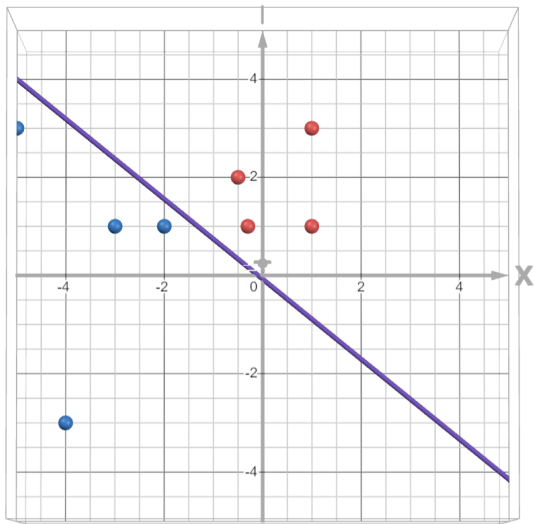
Sometimes the data is not linearly seperable:



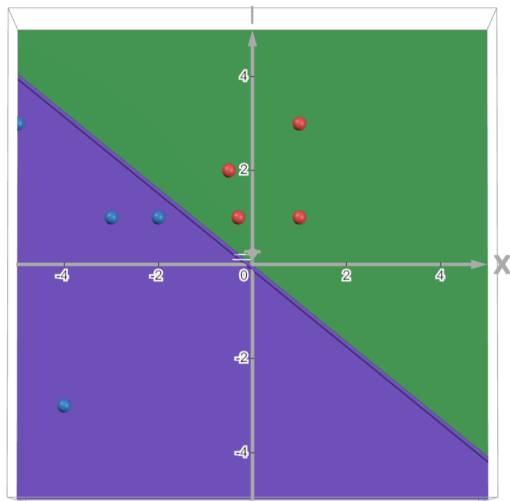
In this case, we cannot achieve 100% accuracy.

This generalizes to higher dimensions. Let's look at the 2D case:
<https://www.desmos.com/3d/vvcu7juj0h>

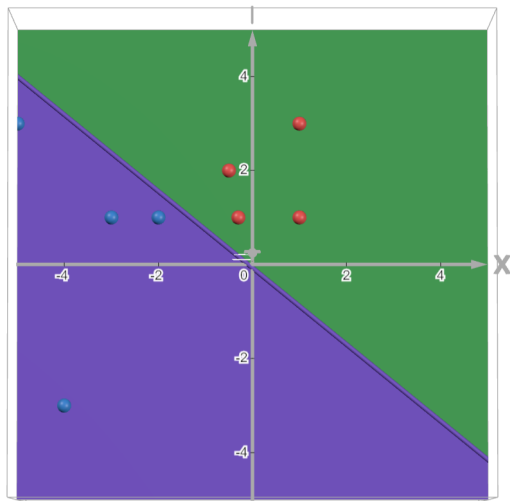
Now we have three weights: $\vec{w} = [w_0, w_1, w_2]$. The set of $\vec{x} = [x_1, x_2]$ satisfying $\vec{w}^\top \cdot \vec{x} = w_0 + w_1 x_1 + w_2 x_2 = 0$ define a line with a slope and intercept:



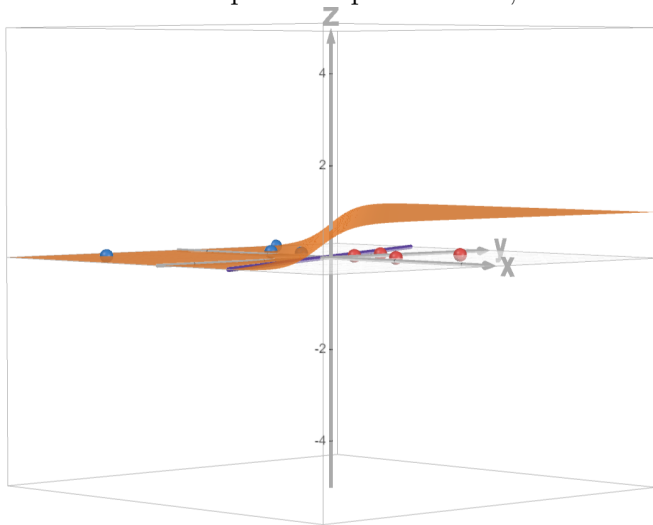
Again, $\vec{w} \cdot \vec{x} > 0$ (green) and $\vec{w} \cdot \vec{x} < 0$ (purple) define regions:



Again, $\vec{w} \cdot \vec{x} > 0$ (green) and $\vec{w} \cdot \vec{x} < 0$ (purple) define regions:



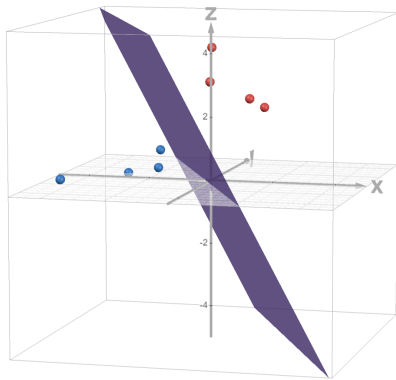
Since our sigmoid now takes as input multiple variables, it now forms a surface:



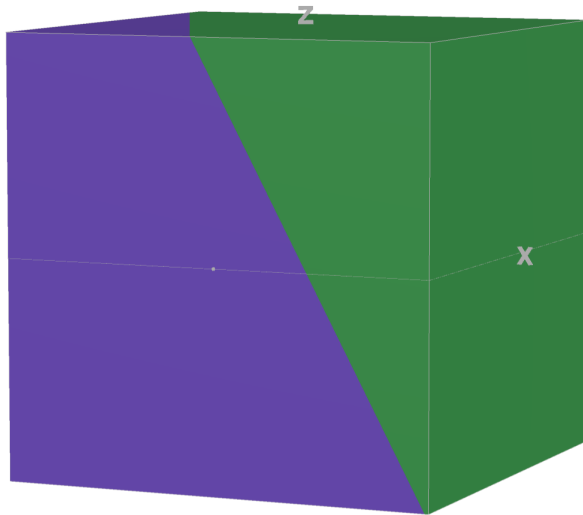
Finally, we will look at the 3D case: <https://www.desmos.com/3d/reb8x2dxed>

Now our points exist in 3D space and are separated by a *plane* defined by the set of points satisfying:

$$w_0 + w_1x_1 + w_2x_2 + w_3x_3 = 0$$

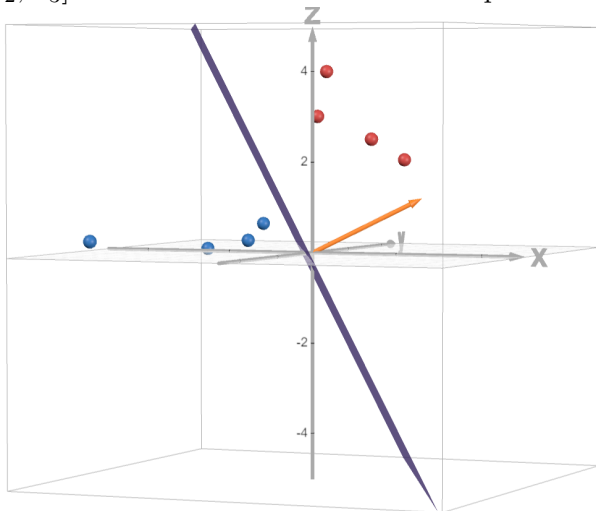


3D space gets split into two regions:



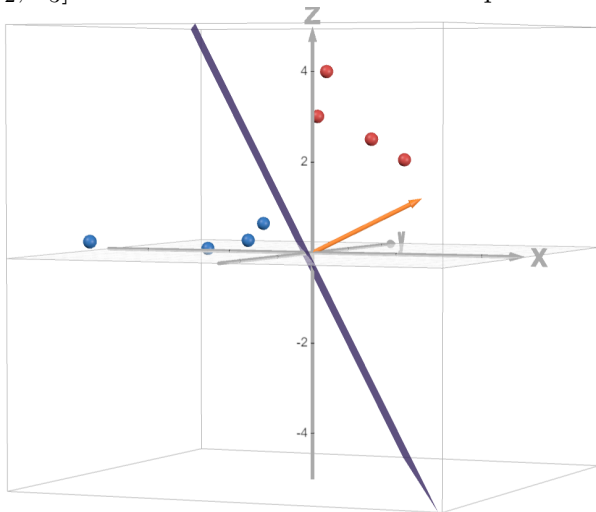
$$w_0 + w_1x_1 + w_2x_2 + w_3x_3 = 0$$

The weights $[w_1, w_2, w_3]$ define the *normal* vector of the plane and w_0 translates it:



$$w_0 + w_1x_1 + w_2x_2 + w_3x_3 = 0$$

The weights $[w_1, w_2, w_3]$ define the *normal* vector of the plane and w_0 translates it:



This concept generalizes to any number of dimensions. The equation $w_0 + \vec{w}^\top \cdot \vec{x} = 0$ defines a *hyperplane*.

This concept generalizes to any number of dimensions. The equation $w_0 + \vec{w}^\top \cdot \vec{x} = 0$ defines a *hyperplane*.

Points on one side of the hyperplane are in one class, and points on the other side are in the other, where the weights $[w_1, \dots, w_D]$ define the normal vector and w_0 is the translation term.

This concept generalizes to any number of dimensions. The equation $w_0 + \vec{w}^\top \cdot \vec{x} = 0$ defines a *hyperplane*.

Points on one side of the hyperplane are in one class, and points on the other side are in the other, where the weights $[w_1, \dots, w_D]$ define the normal vector and w_0 is the translation term.

Thus, we can now think of our goal as finding the weights that form the best decision boundary!

This concept generalizes to any number of dimensions. The equation $w_0 + \vec{w}^\top \cdot \vec{x} = 0$ defines a *hyperplane*.

Points on one side of the hyperplane are in one class, and points on the other side are in the other, where the weights $[w_1, \dots, w_D]$ define the normal vector and w_0 is the translation term.

Thus, we can now think of our goal as finding the weights that form the best decision boundary!

Key result: We have a new way of interpreting the weights as parameters of a hyperplane.

Recap (so far)

We are classifying points by estimating the probability that they are in class 1 using the sigmoid function:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

Recap (so far)

We are classifying points by estimating the probability that they are in class 1 using the sigmoid function:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

Recap (so far)

We are classifying points by estimating the probability that they are in class 1 using the sigmoid function:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

The weights in $w_0 + \vec{w}^\top \cdot \vec{x}$ can be thought of as the parameters of a *hyperplane* which acts as a decision boundary.

Recap (so far)

We are classifying points by estimating the probability that they are in class 1 using the sigmoid function:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

The weights in $w_0 + \vec{w}^\top \cdot \vec{x}$ can be thought of as the parameters of a *hyperplane* which acts as a decision boundary. The best weights correspond to the decision boundary that does the best job at classifying our data.

Recap (so far)

We are classifying points by estimating the probability that they are in class 1 using the sigmoid function:

$$\tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

The weights in $w_0 + \vec{w}^\top \cdot \vec{x}$ can be thought of as the parameters of a *hyperplane* which acts as a decision boundary. The best weights correspond to the decision boundary that does the best job at classifying our data.

So how do we find the weights?

defining our loss function

Let $p(\vec{x}^{(i)}; w_0, \vec{w}) = \tilde{P}(Y = 1 | X = \vec{x}^{(i)}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)})$.

In the following (optional) slides, we prove that the optimal weights are those that minimize

$$J(w_0, w_1, \dots, w_D) = - \sum_{i=1}^N [y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))]$$

defining our loss function

Let $p(\vec{x}^{(i)}; w_0, \vec{w}) = \tilde{P}(Y = 1 | X = \vec{x}^{(i)}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)})$.

In the following (optional) slides, we prove that the optimal weights are those that minimize

$$J(w_0, w_1, \dots, w_D) = - \sum_{i=1}^N [y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))]$$

This is known as “**cross-entropy loss**”.

defining our loss function (optional)

To find our weights, we first need to define what makes a model good. In the case of logistic regression our goal is to find the best decision boundary.

defining our loss function (optional)

To find our weights, we first need to define what makes a model good. In the case of logistic regression our goal is to find the best decision boundary. This goal can be reformulated as finding the model that maximizes the *likelihood* of our dataset.

maximum likelihood (optional)

Remember that

$$\tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

means “the probability, according to our model, of the *correct* label, $y^{(i)}$, when the input is $x^{(i)}$ ”.

maximum likelihood (optional)

Remember that

$$\tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

means “the probability, according to our model, of the *correct* label, $y^{(i)}$, when the input is $x^{(i)}$ ”.

To compute the likelihood of our dataset we multiply the probabilities of correctly predicting each datapoint in the dataset:

maximum likelihood (optional)

Remember that

$$\tilde{P}(Y = y^{(i)}|X = x^{(i)})$$

means “the probability, according to our model, of the *correct* label, $y^{(i)}$, when the input is $x^{(i)}$ ”.

To compute the likelihood of our dataset we multiply the probabilities of correctly predicting each datapoint in the dataset:

$$\tilde{P}(Y = y^{(1)}|X = x^{(1)}) \times \tilde{P}(Y = y^{(2)}|X = x^{(2)}) \times \dots \times \tilde{P}(Y = y^{(N)}|X = x^{(N)})$$

maximum likelihood (optional)

Remember that

$$\tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

means “the probability, according to our model, of the *correct* label, $y^{(i)}$, when the input is $x^{(i)}$ ”.

To compute the likelihood of our dataset we multiply the probabilities of correctly predicting each datapoint in the dataset:

$$\tilde{P}(Y = y^{(1)} | X = x^{(1)}) \times \tilde{P}(Y = y^{(2)} | X = x^{(2)}) \times \dots \times \tilde{P}(Y = y^{(N)} | X = x^{(N)})$$

This can also be written as

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

(Π is similar to Σ except it multiplies instead of adding)

maximum likelihood (optional)

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

This equation tells us the likelihood that our model would predict the right class for each instance of our dataset.

maximum likelihood (optional)

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

This equation tells us the likelihood that our model would predict the right class for each instance of our dataset.

The best model is the one that maximizes this likelihood.

maximum likelihood (optional)

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

This equation tells us the likelihood that our model would predict the right class for each instance of our dataset.

The best model is the one that maximizes this likelihood. This is equivalent to finding the best decision boundary.

maximum likelihood (optional)

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

This equation tells us the likelihood that our model would predict the right class for each instance of our dataset.

The best model is the one that maximizes this likelihood. This is equivalent to finding the best decision boundary. The model that has the highest likelihood *is* the model that has the best decision boundary (ie. it classifies the most points correctly).

maximum likelihood (optional)

Let's rewrite the equation from the previous slide in terms of σ .

maximum likelihood (optional)

Let's rewrite the equation from the previous slide in terms of σ .

We have that:

$$\sigma(\vec{w}^\top \cdot \vec{x}) = \tilde{P}(Y = 1|X = x)$$

maximum likelihood (optional)

Let's rewrite the equation from the previous slide in terms of σ .

We have that:

$$\sigma(\vec{w}^\top \cdot \vec{x}) = \tilde{P}(Y = 1|X = x)$$

Note that

$$\tilde{P}(Y = 0|X = \vec{x}) = 1 - \tilde{P}(Y = 1|X = x) = 1 - \sigma(\vec{w}^\top \cdot \vec{x})$$

maximum likelihood (optional)

Let's rewrite the equation from the previous slide in terms of σ .

We have that:

$$\sigma(\vec{w}^\top \cdot \vec{x}) = \tilde{P}(Y = 1|X = x)$$

Note that

$$\tilde{P}(Y = 0|X = \vec{x}) = 1 - \tilde{P}(Y = 1|X = x) = 1 - \sigma(\vec{w}^\top \cdot \vec{x})$$

We can use this fact to write an equation that tells us the probability of $y^{(i)}$, regardless of whether it's 0 or 1:

$$\begin{aligned}\tilde{P}(Y = y|X = \vec{x}) &= (\tilde{P}(Y = 1|X = x))^y (1 - \tilde{P}(Y = 1|X = x))^{1-y} \\ &= (\sigma(w_0 + \vec{w}^\top \cdot \vec{x}))^y (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}))^{1-y}\end{aligned}$$

maximum likelihood (optional)

Let's pause and see a quick example of how / why this works:

$$\tilde{P}(Y = y|X = \vec{x}) = (\tilde{P}(Y = 1|X = x))^y(1 - \tilde{P}(Y = 1|X = x))^{1-y}$$

maximum likelihood (optional)

Let's pause and see a quick example of how / why this works:

$$\tilde{P}(Y = y|X = \vec{x}) = (\tilde{P}(Y = 1|X = x))^y(1 - \tilde{P}(Y = 1|X = x))^{1-y}$$

If we want to find $\tilde{P}(Y = 1|X = x)$ we simply plug $y = 1$ into this equation:

$$\tilde{P}(Y = 1|X = \vec{x}) = (\tilde{P}(Y = 1|X = x))^{(1)}(1 - \tilde{P}(Y = 1|X = x))^{1-1} = \tilde{P}(Y = 1|X = x)$$

maximum likelihood (optional)

Let's pause and see a quick example of how / why this works:

$$\tilde{P}(Y = y|X = \vec{x}) = (\tilde{P}(Y = 1|X = x))^y(1 - \tilde{P}(Y = 1|X = x))^{1-y}$$

If we want to find $\tilde{P}(Y = 1|X = x)$ we simply plug $y = 1$ into this equation:

$$\tilde{P}(Y = 1|X = \vec{x}) = (\tilde{P}(Y = 1|X = x))^{(1)}(1 - \tilde{P}(Y = 1|X = x))^{1-1} = \tilde{P}(Y = 1|X = x)$$

Similarly, if we want to find $\tilde{P}(Y = 0|X = x)$ we simply plug $y = 0$ into this equation:

$$\tilde{P}(Y = 0|X = \vec{x}) = (\tilde{P}(Y = 1|X = x))^{(0)}(1 - \tilde{P}(Y = 1|X = x))^{1-0} = 1 - \tilde{P}(Y = 1|X = x)$$

maximum likelihood (optional)

Substituting

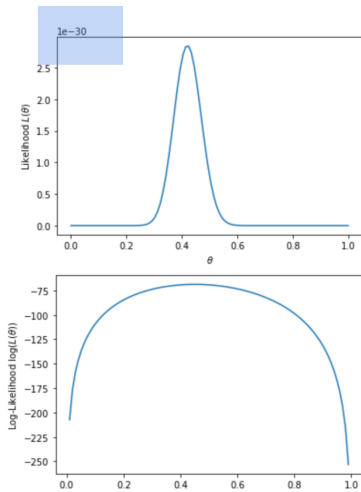
$$\tilde{P}(Y = y|X = \vec{x}) = (\sigma(w_0 + \vec{w}^\top \cdot \vec{x}))^y (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}))^{1-y}$$

into our likelihood equation we get:

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)}|X = x^{(i)}) = \prod_{i=1}^N (\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}}$$

log property (optional)

Fact: If x^* maximizes $f(x)$, then it also maximizes $\log(f(x))$



applying log (optional)

We can make the likelihood easier to read (and more stable numerically) by taking the log:

$$\log \left[\prod_{i=1}^N (\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}} \right]$$

applying log (optional)

We can make the likelihood easier to read (and more stable numerically) by taking the log:

$$\log \left[\prod_{i=1}^N (\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}} \right]$$

Using the logarithm property $\log(ab) = \log a + \log b$, we can split this into:

$$\sum_{i=1}^N \log \left[(\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}} \right]$$

applying log (optional)

We can make the likelihood easier to read (and more stable numerically) by taking the log:

$$\log \left[\prod_{i=1}^N (\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}} \right]$$

Using the logarithm property $\log(ab) = \log a + \log b$, we can split this into:

$$\sum_{i=1}^N \log \left[(\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}} \right]$$

Applying the logarithm property $\log(a^b) = b \log a$, we get:

$$\sum_{i=1}^N \left[y^{(i)} \log \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)})) \right]$$

applying log (optional)

We can make the likelihood easier to read (and more stable numerically) by taking the log:

$$\log \left[\prod_{i=1}^N (\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}} \right]$$

Using the logarithm property $\log(ab) = \log a + \log b$, we can split this into:

$$\sum_{i=1}^N \log \left[(\sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{y^{(i)}} (1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))^{1-y^{(i)}} \right]$$

Applying the logarithm property $\log(a^b) = b \log a$, we get:

$$\sum_{i=1}^N \left[y^{(i)} \log \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)})) \right]$$

applying log (optional)

Thus, we want the model that maximizes:

$$L(w_0, w_1, \dots, w_D) = \sum_{i=1}^N [y^{(i)} \log \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))]$$

applying log (optional)

Thus, we want the model that maximizes:

$$L(w_0, w_1, \dots, w_D) = \sum_{i=1}^N [y^{(i)} \log \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))]$$

It's convention to minimize rather than maximize in machine learning, so we will flip the sign:

$$J(w_0, w_1, \dots, w_D) = - \sum_{i=1}^N [y^{(i)} \log \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))]$$

applying log (optional)

Thus, we want the model that maximizes:

$$L(w_0, w_1, \dots, w_D) = \sum_{i=1}^N [y^{(i)} \log \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))]$$

It's convention to minimize rather than maximize in machine learning, so we will flip the sign:

$$J(w_0, w_1, \dots, w_D) = - \sum_{i=1}^N [y^{(i)} \log \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(w_0 + \vec{w}^\top \cdot \vec{x}^{(i)}))]$$

Note: This is called the “**cross-entropy loss**” and it's a function of the *weights*!

cross-entropy loss: intuition

Our goal is to minimize

$$-\sum_{i=1}^N [y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))]$$

cross-entropy loss: intuition

Our goal is to minimize

$$-\sum_{i=1}^N [y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))]$$

This equation tells us to:

1. Iterate through each data point $(\vec{x}^{(i)}, y^{(i)})$
2. Compute $y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$
3. Add up all the results
4. Multiply by negative 1

cross-entropy loss: intuition

Since we multiply by -1 , our sum is *minimized* when the inner terms are *maximized*:

$$y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$$

- The inputs to log are probabilities, which fall in $(0, 1)$ $\implies$ log terms are largest when input is 1

cross-entropy loss: intuition

Since we multiply by -1 , our sum is *minimized* when the inner terms are *maximized*:

$$y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$$

- The inputs to log are probabilities, which fall in $(0, 1)$ $\implies$ log terms are largest when input is 1
- If $y^{(i)} = 1$, then the blue term disappears and we're left with $\log p(\vec{x}^{(i)}; w_0, \vec{w})$

cross-entropy loss: intuition

Since we multiply by -1 , our sum is *minimized* when the inner terms are *maximized*:

$$y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$$

- The inputs to log are probabilities, which fall in $(0, 1)$ $\implies$ log terms are largest when input is 1
- If $y^{(i)} = 1$, then the blue term disappears and we're left with $\log p(\vec{x}^{(i)}; w_0, \vec{w})$
 - Largest when $p(\vec{x}^{(i)}; w_0, \vec{w}) = \tilde{P}(Y = 1|x^{(i)}) \approx 1$

cross-entropy loss: intuition

Since we multiply by -1 , our sum is *minimized* when the inner terms are *maximized*:

$$y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$$

- The inputs to log are probabilities, which fall in $(0, 1)$ $\implies$ log terms are largest when input is 1
- If $y^{(i)} = 1$, then the blue term disappears and we're left with $\log p(\vec{x}^{(i)}; w_0, \vec{w})$
 - Largest when $p(\vec{x}^{(i)}; w_0, \vec{w}) = \tilde{P}(Y = 1|x^{(i)}) \approx 1$
- If $y^{(i)} = 0$, then the red term disappears and we're left with $\log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$

cross-entropy loss: intuition

Since we multiply by -1 , our sum is *minimized* when the inner terms are *maximized*:

$$y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$$

- The inputs to log are probabilities, which fall in $(0, 1) \implies$ log terms are largest when input is 1
- If $y^{(i)} = 1$, then the blue term disappears and we're left with $\log p(\vec{x}^{(i)}; w_0, \vec{w})$
 - ▶ Largest when $p(\vec{x}^{(i)}; w_0, \vec{w}) = \tilde{P}(Y = 1|x^{(i)}) \approx 1$
- If $y^{(i)} = 0$, then the red term disappears and we're left with $\log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$
 - ▶ Largest when $1 - p(\vec{x}^{(i)}; w_0, \vec{w}) = 1 - \tilde{P}(Y = 1|x^{(i)}) = \tilde{P}(Y = 0|x^{(i)}) \approx 1$

cross-entropy loss: intuition

Since we multiply by -1 , our sum is *minimized* when the inner terms are *maximized*:

$$y^{(i)} \log p(\vec{x}^{(i)}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$$

- The inputs to log are probabilities, which fall in $(0, 1) \implies$ log terms are largest when input is 1
- If $y^{(i)} = 1$, then the blue term disappears and we're left with $\log p(\vec{x}^{(i)}; w_0, \vec{w})$
 - ▶ Largest when $p(\vec{x}^{(i)}; w_0, \vec{w}) = \tilde{P}(Y = 1|x^{(i)}) \approx 1$
- If $y^{(i)} = 0$, then the red term disappears and we're left with $\log(1 - p(\vec{x}^{(i)}; w_0, \vec{w}))$
 - ▶ Largest when $1 - p(\vec{x}^{(i)}; w_0, \vec{w}) = 1 - \tilde{P}(Y = 1|x^{(i)}) = \tilde{P}(Y = 0|x^{(i)}) \approx 1$

Thus, the inner terms are maximized when we predict the *correct class* with high probability.

Recap

Question: How do we minimize the loss?

We can compute the gradient (given without proof):

$$\nabla_{\vec{w}} J(w_0, w_1, \dots, w_D) = \sum_{i=1}^N (\sigma(w_0 + \vec{w}^\top \vec{x}) - y) \vec{x}$$

We can compute the gradient (given without proof):

$$\nabla_{\vec{w}} J(w_0, w_1, \dots, w_D) = \sum_{i=1}^N (\sigma(w_0 + \vec{w}^\top \vec{x}) - y) \vec{x}$$

There is no closed form solution for

$$\nabla_{\vec{w}} J(w_0, w_1, \dots, w_D) = 0$$

We can compute the gradient (given without proof):

$$\nabla_{\vec{w}} J(w_0, w_1, \dots, w_D) = \sum_{i=1}^N (\sigma(w_0 + \vec{w}^\top \vec{x}) - y) \vec{x}$$

There is no closed form solution for

$$\nabla_{\vec{w}} J(w_0, w_1, \dots, w_D) = 0$$

In other words, we cannot find it directly by solving for $\vec{w}$ in this equation. Instead, we will search for the minimum via gradient descent (as seen last time).

recap

We are classifying points by estimating the probability that they are in class 1:

$$p(\vec{x}; w_0, \vec{w}) = \tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

recap

We are classifying points by estimating the probability that they are in class 1:

$$p(\vec{x}; w_0, \vec{w}) = \tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

recap

We are classifying points by estimating the probability that they are in class 1:

$$p(\vec{x}; w_0, \vec{w}) = \tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

The weights in $w_0 + \vec{w}^\top \cdot \vec{x}$ can be thought of as the parameters of a *hyperplane* which acts as a decision boundary.

recap

We are classifying points by estimating the probability that they are in class 1:

$$p(\vec{x}; w_0, \vec{w}) = \tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

The weights in $w_0 + \vec{w}^\top \cdot \vec{x}$ can be thought of as the parameters of a *hyperplane* which acts as a decision boundary. The best weights correspond to the decision boundary that does the best job at classifying our data.

recap

We are classifying points by estimating the probability that they are in class 1:

$$p(\vec{x}; w_0, \vec{w}) = \tilde{P}(Y = 1|X = \vec{x}) = \sigma(w_0 + \vec{w}^\top \cdot \vec{x}) = \frac{1}{1 + e^{-(w_0 + \vec{w}^\top \cdot \vec{x})}}$$

We classify based on the class with the higher estimated probability.

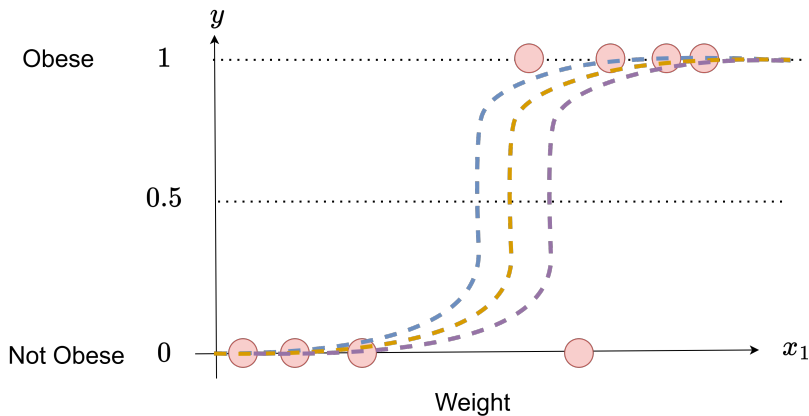
The weights in $w_0 + \vec{w}^\top \cdot \vec{x}$ can be thought of as the parameters of a *hyperplane* which acts as a decision boundary. The best weights correspond to the decision boundary that does the best job at classifying our data.

We find the optimal weights by minimizing the *cross-entropy loss* function using gradient descent:

$$J(w_0, w_1, \dots, w_D) = - \sum_{i=1}^N [y^{(i)} \log p(\vec{x}; w_0, \vec{w}) + (1 - y^{(i)}) \log(1 - p(\vec{x}; w_0, \vec{w}))]$$

logistic regression in action

In logistic regression, we use gradient descent to try different models (dotted lines) until we find the optimal model:



demo

<https://mlu-explain.github.io/logistic-regression/>

How can we generalize our algorithm to work for $C > 2$ classes? Let's take a look at the Iris dataset once again:

#	sepal length (x_0)	sepal width (x_1)	petal length (x_2)	petal width (x_3)	label (y)
1	5.1	3.5	1.4	0.2	<i>I. setosa</i>
2	4.9	3.0	1.4	0.2	<i>I. setosa</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
51	7.0	3.2	4.7	1.4	<i>I. versicolor</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
101	6.3	3.3	6.0	2.5	<i>I. virginica</i>
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
150	5.9	3.0	5.1	1.8	<i>I. virginica</i>

one-hot encoding

In general in machine learning, any categorical data should be in its **one-hot encoding notation**:

id	color
1	red
2	blue
3	green
4	blue



id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Let's see what our dataset looks like now:

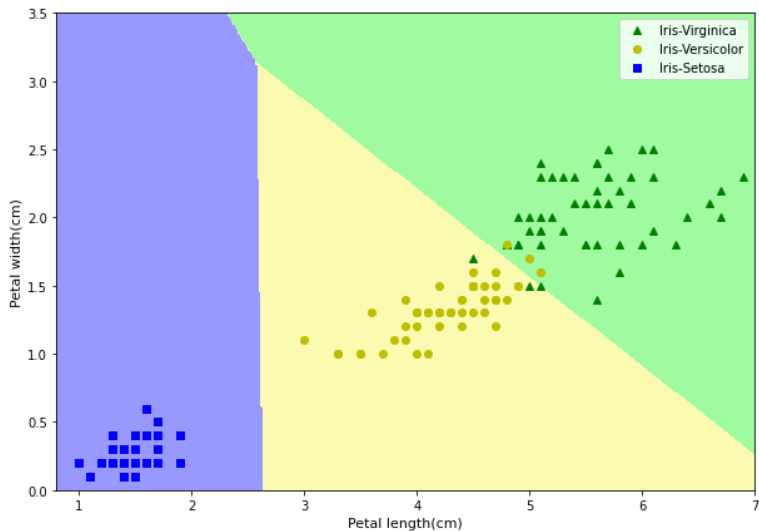
#	sepal length (x_0)	sepal width (x_1)	petal length (x_2)	petal width (x_3)	label ($\vec{y}$)
1	5.1	3.5	1.4	0.2	[1, 0, 0]
2	4.9	3.0	1.4	0.2	[1, 0, 0]
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
51	7.0	3.2	4.7	1.4	[0, 1, 0]
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
101	6.3	3.3	6.0	2.5	[0, 0, 1]
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
150	5.9	3.0	5.1	1.8	[0, 0, 1]

why one-hot encoding?

One-hot encoding is essential to turn categorical data into a numerical format (0s and 1s) without implying a false order or ranking between categories.

It prevents models from misinterpreting labels (e.g., Cat=1, Dog=2) as having mathematical relationships, allowing for more accurate, unbiased predictions.

Our model will learn decision boundaries to split our space into *three regions*:



multinomial logistic regression

Our previous algorithm only worked for *binary* classification.

multinomial logistic regression

Our previous algorithm only worked for *binary* classification.

In multinomial regression we will predict the probability of *every class* and pick the one with the highest probability.

To predict a vector of probabilities, we will use the *softmax-function*:

$$\mathcal{S} : \mathbb{R}^C \rightarrow [0, 1]^C$$

as

$$\mathcal{S}(\vec{a}) = \left[\frac{e^{a[1]}}{\sum_{i=1}^C e^{a[i]}}, \quad \frac{e^{a[2]}}{\sum_{i=1}^C e^{a[i]}}, \quad \cdots, \quad \frac{e^{a[C]}}{\sum_{i=1}^C e^{a[i]}} \right] \in \mathbb{R}^C$$

To predict a vector of probabilities, we will use the *softmax-function*:

$$\mathcal{S} : \mathbb{R}^C \rightarrow [0, 1]^C$$

as

$$\mathcal{S}(\vec{a}) = \left[\frac{e^{a[1]}}{\sum_{i=1}^C e^{a[i]}}, \quad \frac{e^{a[2]}}{\sum_{i=1}^C e^{a[i]}}, \quad \cdots, \quad \frac{e^{a[C]}}{\sum_{i=1}^C e^{a[i]}} \right] \in \mathbb{R}^C$$

This will acts as our analogue to σ .

softmax: example

Let $\vec{a} = [1, 2, 3] \in \mathbb{R}^3$, then

$$\mathcal{S}(\vec{a}) = \left[\frac{e^1}{e^1 + e^2 + e^3}, \quad \frac{e^2}{e^1 + e^2 + e^3}, \quad \frac{e^3}{e^1 + e^2 + e^3} \right] = [0.09, 0.24, 0.67]$$

defining our model

Let C be the number of classes and $W \in \mathbb{R}^{C \times D}$. Notice that, for any $[x_1, x_2, \dots, x_D] = \vec{x} \in \mathbb{R}^D$,

defining our model

Let C be the number of classes and $W \in \mathbb{R}^{C \times D}$. Notice that, for any $[x_1, x_2, \dots, x_D] = \vec{x} \in \mathbb{R}^D$,

$$W\vec{x} + \vec{w}_0$$

defining our model

Let C be the number of classes and $W \in \mathbb{R}^{C \times D}$. Notice that, for any $[x_1, x_2, \dots, x_D] = \vec{x} \in \mathbb{R}^D$,

$$W\vec{x} + \vec{w}_0 = \begin{bmatrix} \text{---} & \vec{w}_1^\top & \text{---} \\ & \vdots & \\ \text{---} & \vec{w}_C^\top & \text{---} \end{bmatrix} \vec{x} + \begin{bmatrix} w_{1,0} \\ \vdots \\ w_{C,0} \end{bmatrix}$$

defining our model

Let C be the number of classes and $W \in \mathbb{R}^{C \times D}$. Notice that, for any $[x_1, x_2, \dots, x_D] = \vec{x} \in \mathbb{R}^D$,

$$W\vec{x} + \vec{w}_0 = \begin{bmatrix} \text{---} & \vec{w}_1^\top & \text{---} \\ & \vdots & \\ \text{---} & \vec{w}_C^\top & \text{---} \end{bmatrix} \vec{x} + \begin{bmatrix} w_{1,0} \\ \vdots \\ w_{C,0} \end{bmatrix} = \begin{bmatrix} \vec{w}_1^\top \cdot \vec{x} + w_{1,0} \\ \vdots \\ \vec{w}_C^\top \cdot \vec{x} + w_{C,0} \end{bmatrix}$$

defining our model

Let C be the number of classes and $W \in \mathbb{R}^{C \times D}$. Notice that, for any $[x_1, x_2, \dots, x_D] = \vec{x} \in \mathbb{R}^D$,

$$W\vec{x} + \vec{w}_0 = \begin{bmatrix} \text{---} & \vec{w}_1^\top & \text{---} \\ & \vdots & \\ \text{---} & \vec{w}_C^\top & \text{---} \end{bmatrix} \vec{x} + \begin{bmatrix} w_{1,0} \\ \vdots \\ w_{C,0} \end{bmatrix} = \begin{bmatrix} \vec{w}_1^\top \cdot \vec{x} + w_{1,0} \\ \vdots \\ \vec{w}_C^\top \cdot \vec{x} + w_{C,0} \end{bmatrix}$$

You can think of it as having C different decision boundaries, although the intuition is not as straightforward as in the binary case.

defining our model

In multinomial regression, we will learn a weight matrix W and bias vector $\vec{w}_0$ and predict probabilities using

$$p(\vec{x}; \vec{w}_0, W) = \tilde{P}(Y|X = \vec{x}) = \mathcal{S}(W\vec{x} + \vec{w}_0)$$

This works because the elements add up to zero (so we can treat them as probabilities).

softmax example

Let's compute an example of $\mathcal{S}(W\vec{x} + \vec{w}_0)$ for $\vec{x} = [x_1, x_2, x_3] \in \mathbb{R}^3$ with $C = 3$.

Suppose:

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 1 & -1 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \vec{w}_0 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

softmax example

Let's compute an example of $\mathcal{S}(W\vec{x} + \vec{w}_0)$ for $\vec{x} = [x_1, x_2, x_3] \in \mathbb{R}^3$ with $C = 3$.

Suppose:

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 1 & -1 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \vec{w}_0 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

We have that $W\vec{x} + \vec{w}_0 = [3, 4, 1]$

softmax example

Let's compute an example of $\mathcal{S}(W\vec{x} + \vec{w}_0)$ for $\vec{x} = [x_1, x_2, x_3] \in \mathbb{R}^3$ with $C = 3$.

Suppose:

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 1 & -1 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \vec{w}_0 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

We have that $W\vec{x} + \vec{w}_0 = [3, 4, 1]$

Next, apply the softmax function:

$$p(\vec{x}; \vec{w}_0, W) = \mathcal{S}(W\vec{x} + \vec{w}_0) = \left[\frac{e^3}{e^3 + e^4 + e^1}, \frac{e^4}{e^3 + e^4 + e^1}, \frac{e^1}{e^3 + e^4 + e^1} \right]$$

softmax example

Let's compute an example of $\mathcal{S}(W\vec{x} + \vec{w}_0)$ for $\vec{x} = [x_1, x_2, x_3] \in \mathbb{R}^3$ with $C = 3$.

Suppose:

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 1 & -1 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \vec{w}_0 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

We have that $W\vec{x} + \vec{w}_0 = [3, 4, 1]$

Next, apply the softmax function:

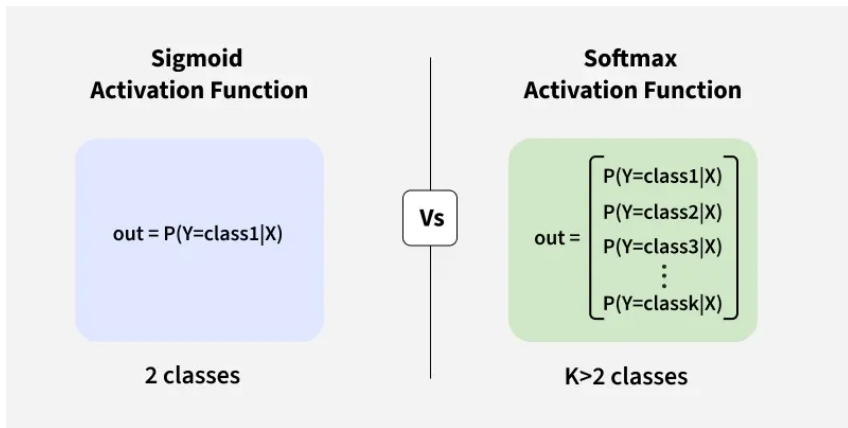
$$p(\vec{x}; \vec{w}_0, W) = \mathcal{S}(W\vec{x} + \vec{w}_0) = \left[\frac{e^3}{e^3 + e^4 + e^1}, \frac{e^4}{e^3 + e^4 + e^1}, \frac{e^1}{e^3 + e^4 + e^1} \right]$$

Our final result is approximately:

$$p(\vec{x}; \vec{w}_0, W) \approx [0.26, 0.71, 0.03]$$

In this case we would predict class 2, since the element at index 2 (using one-based indexing) has the highest probability.

sigmoid vs softmax



Optional: proving equivalence to sigmoid

Softmax on $\mathbb{R}^2$ is the same as sigmoid:

$$\mathcal{S}([a_1, a_2]) = \left[\frac{e^{a_1}}{e^{a_1} + e^{a_2}}, \frac{e^{a_2}}{e^{a_1} + e^{a_2}} \right]$$

Optional: proving equivalence to sigmoid

Softmax on $\mathbb{R}^2$ is the same as sigmoid:

$$\begin{aligned}\mathcal{S}([a_1, a_2]) &= \left[\frac{e^{a_1}}{e^{a_1} + e^{a_2}}, \frac{e^{a_2}}{e^{a_1} + e^{a_2}} \right] \\ &= \left[\frac{1}{1 + e^{a_2 - a_1}}, \frac{e^{a_2 - a_1}}{1 + e^{a_2 - a_1}} \right]\end{aligned}$$

Optional: proving equivalence to sigmoid

Softmax on $\mathbb{R}^2$ is the same as sigmoid:

$$\begin{aligned}\mathcal{S}([a_1, a_2]) &= \left[\frac{e^{a_1}}{e^{a_1} + e^{a_2}}, \frac{e^{a_2}}{e^{a_1} + e^{a_2}} \right] \\ &= \left[\frac{1}{1 + e^{a_2 - a_1}}, \frac{e^{a_2 - a_1}}{1 + e^{a_2 - a_1}} \right] \\ &= \left[\frac{1}{1 + e^{a_2 - a_1}}, 1 - \frac{1}{1 + e^{a_2 - a_1}} \right]\end{aligned}$$

Optional: proving equivalence to sigmoid

Softmax on $\mathbb{R}^2$ is the same as sigmoid:

$$\begin{aligned}\mathcal{S}([a_1, a_2]) &= \left[\frac{e^{a_1}}{e^{a_1} + e^{a_2}}, \frac{e^{a_2}}{e^{a_1} + e^{a_2}} \right] \\ &= \left[\frac{1}{1 + e^{a_2 - a_1}}, \frac{e^{a_2 - a_1}}{1 + e^{a_2 - a_1}} \right] \\ &= \left[\frac{1}{1 + e^{a_2 - a_1}}, 1 - \frac{1}{1 + e^{a_2 - a_1}} \right] \\ &= \left[\sigma(a_2 - a_1), 1 - \sigma(a_2 - a_1) \right]\end{aligned}$$

Optional: proving equivalence to sigmoid

Softmax on $\mathbb{R}^2$ is the same as sigmoid:

$$\begin{aligned}\mathcal{S}([a_1, a_2]) &= \left[\frac{e^{a_1}}{e^{a_1} + e^{a_2}}, \frac{e^{a_2}}{e^{a_1} + e^{a_2}} \right] \\ &= \left[\frac{1}{1 + e^{a_2 - a_1}}, \frac{e^{a_2 - a_1}}{1 + e^{a_2 - a_1}} \right] \\ &= \left[\frac{1}{1 + e^{a_2 - a_1}}, 1 - \frac{1}{1 + e^{a_2 - a_1}} \right] \\ &= \left[\sigma(a_2 - a_1), 1 - \sigma(a_2 - a_1) \right]\end{aligned}$$

Setting $a = a_2 - a_1$, computing multinomial logistic regression on two classes gives the same result as binary logistic regression.

our loss function

$$J(\vec{w}_0, W) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}^{(i)}[c] \log(p(\vec{x}; \vec{w}_0, W)[c])$$

where

- $p(\vec{x}; \vec{w}_0, W)[c]$ is the c -th element of the vector $p(\vec{x}; \vec{w}_0, W)$ (probability of class c)
- $\vec{y}^{(i)}[c]$ is the c -th element of the *one-hot encoding* of label $y^{(i)}$

our loss function

$$J(\vec{w}_0, W) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}^{(i)}[c] \log(p(\vec{x}; \vec{w}_0, W)[c])$$

where

- $p(\vec{x}; \vec{w}_0, W)[c]$ is the c -th element of the vector $p(\vec{x}; \vec{w}_0, W)$ (probability of class c)
- $\vec{y}^{(i)}[c]$ is the c -th element of the *one-hot encoding* of label $y^{(i)}$

optional: deriving the loss function

In the same way as before, the best model is the one that maximizes the probability of our data:

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

optional: deriving the loss function

In the same way as before, the best model is the one that maximizes the probability of our data:

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

Our model predicts the probability of class c as:

$$\tilde{P}(Y = c | X = \vec{x}) = \mathcal{S}(W\vec{x})_c$$

optional: deriving the loss function

In the same way as before, the best model is the one that maximizes the probability of our data:

$$\prod_{i=1}^N \tilde{P}(Y = y^{(i)} | X = x^{(i)})$$

Our model predicts the probability of class c as:

$$\tilde{P}(Y = c | X = \vec{x}) = \mathcal{S}(W\vec{x})_c$$

Plugging this in we get

$$\prod_{i=1}^N \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_{y^{(i)}}$$

optional: deriving the loss function

The following notation is a bit clunky:

$$\prod_{i=1}^N \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_{y^{(i)}}$$

optional: deriving the loss function

The following notation is a bit clunky:

$$\prod_{i=1}^N \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_{y^{(i)}}$$

We can make this cleaner by changing our labels to one-hot encodings.

optional: deriving the loss function

The following notation is a bit clunky:

$$\prod_{i=1}^N \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_{y^{(i)}}$$

We can make this cleaner by changing our labels to one-hot encodings.

One-hot encoding

If $y^{(i)} = c$, then our one-hot encoding is a vector with zeros everywhere except in the c -th position.

optional: deriving the loss function

The following notation is a bit clunky:

$$\prod_{i=1}^N \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_{y^{(i)}}$$

We can make this cleaner by changing our labels to one-hot encodings.

One-hot encoding

If $y^{(i)} = c$, then our one-hot encoding is a vector with zeros everywhere except in the c -th position.

Example: For $C = 3$, our one-hot encodings are

$$y = 1 \rightarrow [1, 0, 0]$$

$$y = 2 \rightarrow [0, 1, 0]$$

$$y = 3 \rightarrow [0, 0, 1]$$

optional: deriving the loss function

Converting each $y^{(i)}$ to its one-hot encoding we get:

$$\prod_{i=1}^N \prod_{c=1}^C \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c^{\vec{y}_c^{(i)}}$$

optional: deriving the loss function

Converting each $y^{(i)}$ to its one-hot encoding we get:

$$\prod_{i=1}^N \prod_{c=1}^C \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c^{\vec{y}_c^{(i)}}$$

Taking the negative log we get

$$J(W) = - \sum_{i=1}^N \sum_{c=1}^C \log(\mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c^{\vec{y}_c^{(i)}}) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}_c^{(i)} \log(\mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c)$$

optional: deriving the loss function

Converting each $y^{(i)}$ to its one-hot encoding we get:

$$\prod_{i=1}^N \prod_{c=1}^C \mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c^{\vec{y}_c^{(i)}}$$

Taking the negative log we get

$$J(W) = - \sum_{i=1}^N \sum_{c=1}^C \log(\mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c^{\vec{y}_c^{(i)}}) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}_c^{(i)} \log(\mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c)$$

Once again, we can compute the gradient (not shown) and minimize using gradient descent.

loss function: intuition

Again, we minimize by *maximizing* the inner terms (since we multiply by -1)

$$J(\vec{w}_0, W) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}^{(i)}[c] \log(p(\vec{x}; \vec{w}_0, W)[c])$$

- $\vec{y}^{(i)}[k]$ is 1 if $y^{(i)} = k$ and 0 for all other $c \neq k$

loss function: intuition

Again, we minimize by *maximizing* the inner terms (since we multiply by -1)

$$J(\vec{w}_0, W) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}^{(i)}[c] \log(p(\vec{x}; \vec{w}_0, W)[c])$$

- $\vec{y}^{(i)}[k]$ is 1 if $y^{(i)} = k$ and 0 for all other $c \neq k$
- $p(\vec{x}; \vec{w}_0, W)[c]$ is maximized when $\mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c = P(Y = c|X = x^{(i)}) \approx 1$

loss function: intuition

Again, we minimize by *maximizing* the inner terms (since we multiply by -1)

$$J(\vec{w}_0, W) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}^{(i)}[c] \log(p(\vec{x}; \vec{w}_0, W)[c])$$

- $\vec{y}^{(i)}[k]$ is 1 if $y^{(i)} = k$ and 0 for all other $c \neq k$
- $p(\vec{x}; \vec{w}_0, W)[c]$ is maximized when $\mathcal{S}(W\vec{x}^{(i)} + \vec{w}_0)_c = P(Y = c|X = x^{(i)}) \approx 1$

So, again, this is minimized when we predict the correct class with high probability.

recap

We are classifying points by estimating the probability that they are in each class:

$$p(\vec{x}; \vec{w}_0, W) = \left[\frac{e^{(W\vec{x} + \vec{w}_0)_1}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}}, \quad \frac{e^{(W\vec{x} + \vec{w}_0)_2}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}}, \quad \cdots, \quad \frac{e^{(W\vec{x} + \vec{w}_0)_C}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}} \right]$$

recap

We are classifying points by estimating the probability that they are in each class:

$$p(\vec{x}; \vec{w}_0, W) = \left[\frac{e^{(W\vec{x} + \vec{w}_0)_1}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}}, \quad \frac{e^{(W\vec{x} + \vec{w}_0)_2}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}}, \quad \cdots, \quad \frac{e^{(W\vec{x} + \vec{w}_0)_C}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}} \right]$$

We classify based on the class with the highest estimated probability.

recap

We are classifying points by estimating the probability that they are in each class:

$$p(\vec{x}; \vec{w}_0, W) = \left[\frac{e^{(W\vec{x} + \vec{w}_0)_1}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}}, \quad \frac{e^{(W\vec{x} + \vec{w}_0)_2}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}}, \quad \cdots, \quad \frac{e^{(W\vec{x} + \vec{w}_0)_C}}{\sum_{i=1}^C e^{(W\vec{x} + \vec{w}_0)_i}} \right]$$

We classify based on the class with the highest estimated probability.

We find the optimal weights by minimizing the *cross-entropy loss* function using gradient descent:

$$J(\vec{w}_0, W) = - \sum_{i=1}^N \sum_{c=1}^C \vec{y}_c^{(i)} \log(p(\vec{x}; \vec{w}_0, W))$$

stochastic gradient descent

Recall that loss functions typically take the form

$$J(W) = \sum_{i=1}^N \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

for some “error” function Error and prediction function f which is parametrized by weights W .

Recall that loss functions typically take the form

$$J(W) = \sum_{i=1}^N \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

for some “error” function Error and prediction function f which is parametrized by weights W .

Example: Sum of squared error for linear regression

In linear regression we make predictions $\hat{y}^{(i)} = f(\vec{x}; \vec{w}) = w_0 + \vec{w}^\top \cdot \vec{x}$ and we compute the error as the squared distance between our prediction and the target:

$$\text{Error}(y^{(i)}, f(\vec{x}; \vec{w})) = (y^{(i)} - f(\vec{x}; \vec{w}))^2 = (y^{(i)} - \hat{y}^{(i)})^2$$

Recall that loss functions typically take the form

$$J(W) = \sum_{i=1}^N \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

for some “error” function Error and prediction function f which is parametrized by weights W .

Example: Sum of squared error for linear regression

In linear regression we make predictions $\hat{y}^{(i)} = f(\vec{x}; \vec{w}) = w_0 + \vec{w}^\top \cdot \vec{x}$ and we compute the error as the squared distance between our prediction and the target:

$$\text{Error}(y^{(i)}, f(\vec{x}; \vec{w})) = (y^{(i)} - f(\vec{x}; \vec{w}))^2 = (y^{(i)} - \hat{y}^{(i)})^2$$

$$J(\vec{w}) = \sum_{i=1}^N \text{Error}(y^{(i)}, f(\vec{x}; \vec{w})) = \sum_{i=1}^N (y^{(i)} - f(\vec{x}; \vec{w}))^2$$

$$J(W) = \sum_{i=1}^N \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

for some “error” function Error and prediction function f which is parametrized by weights W .

$$J(W) = \sum_{i=1}^N \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

for some “error” function Error and prediction function f which is parametrized by weights W .

Recall the update rule for gradient descent:

$$W \leftarrow W - \alpha \nabla_W J(W)$$

$$J(W) = \sum_{i=1}^N \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

for some “error” function Error and prediction function f which is parametrized by weights W .

Recall the update rule for gradient descent:

$$W \leftarrow W - \alpha \nabla_W J(W) = W - \alpha \sum_{i=1}^N \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

$$J(W) = \sum_{i=1}^N \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

for some “error” function Error and prediction function f which is parametrized by weights W .

Recall the update rule for gradient descent:

$$W \leftarrow W - \alpha \nabla_W J(W) = W - \alpha \sum_{i=1}^N \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

Notice that gradient descent requires computing the gradient over the **entire training set**. This is called **batched gradient descent**

Since gradient descent requires computing the gradient over the **entire training set**, it can become very expensive!

Since gradient descent requires computing the gradient over the **entire training set**, it can become very expensive!

Example

Consider logistic regression where our inputs are $\vec{x}^{(i)} \in \mathbb{R}^{23,000}$ (eg. $\vec{x}$ is some gene data).

Since gradient descent requires computing the gradient over the **entire training set**, it can become very expensive!

Example

Consider logistic regression where our inputs are $\vec{x}^{(i)} \in \mathbb{R}^{23,000}$ (eg. $\vec{x}$ is some gene data). Since we have one weight per input dimension we also have $\vec{w} \in \mathbb{R}^{23,000}$.

Our gradient is thus a 23,000 dimensional vector! Thus, we will need to plug our data into 23,000 derivatives!

Since gradient descent requires computing the gradient over the **entire training set**, it can become very expensive!

Example

Consider logistic regression where our inputs are $\vec{x}^{(i)} \in \mathbb{R}^{23,000}$ (eg. $\vec{x}$ is some gene data). Since we have one weight per input dimension we also have $\vec{w} \in \mathbb{R}^{23,000}$.

Our gradient is thus a 23,000 dimensional vector! Thus, we will need to plug our data into 23,000 derivatives!

If we have 1,000,000 samples in our dataset, we will need to compute $1,000,000 \times 23,000 = 23,000,000,000$ derivatives *per weight update!!!*

Since gradient descent requires computing the gradient over the **entire training set**, it can become very expensive!

Example

Consider logistic regression where our inputs are $\vec{x}^{(i)} \in \mathbb{R}^{23,000}$ (eg. $\vec{x}$ is some gene data). Since we have one weight per input dimension we also have $\vec{w} \in \mathbb{R}^{23,000}$.

Our gradient is thus a 23,000 dimensional vector! Thus, we will need to plug our data into 23,000 derivatives!

If we have 1,000,000 samples in our dataset, we will need to compute $1,000,000 \times 23,000 = 23,000,000,000$ derivatives *per weight update!!!*

We often perform at least 1,000 weight updates, which means we would compute $1000 \times 23,000,000,000 = 2,300,000,000,000$ derivatives over the course of training!

Instead of computing the gradient over all of $\mathcal{D}$, on every iteration *sample* a single datapoint $(\vec{x}^{(i)}, \vec{y}^{(i)}) \in \mathcal{D}$ and compute the gradient only w.r.t. that point.

Instead of computing the gradient over all of $\mathcal{D}$, on every iteration *sample* a single datapoint $(\vec{x}^{(i)}, \vec{y}^{(i)}) \in \mathcal{D}$ and compute the gradient only w.r.t. that point.

$$W \leftarrow W - \alpha \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

Instead of computing the gradient over all of $\mathcal{D}$, on every iteration *sample* a single datapoint $(\vec{x}^{(i)}, \vec{y}^{(i)}) \in \mathcal{D}$ and compute the gradient only w.r.t. that point.

$$W \leftarrow W - \alpha \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

This is called **stochastic gradient descent**, and **online** gradient descent sometimes as well.

Instead of computing the gradient over all of $\mathcal{D}$, on every iteration *sample* a single datapoint $(\vec{x}^{(i)}, \vec{y}^{(i)}) \in \mathcal{D}$ and compute the gradient only w.r.t. that point.

$$W \leftarrow W - \alpha \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

This is called **stochastic gradient descent**, and **online** gradient descent sometimes as well. If we take a random sample K datapoints $(\vec{x}^{(1)}, \vec{y}^{(1)}), \dots, (\vec{x}^{(K)}, \vec{y}^{(K)}) \in \mathcal{D}$ and compute

$$W \leftarrow W - \alpha \sum_{i=1}^K \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

Instead of computing the gradient over all of $\mathcal{D}$, on every iteration *sample* a single datapoint $(\vec{x}^{(i)}, \vec{y}^{(i)}) \in \mathcal{D}$ and compute the gradient only w.r.t. that point.

$$W \leftarrow W - \alpha \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

This is called **stochastic gradient descent**, and **online** gradient descent sometimes as well. If we take a random sample K datapoints $(\vec{x}^{(1)}, \vec{y}^{(1)}), \dots, (\vec{x}^{(K)}, \vec{y}^{(K)}) \in \mathcal{D}$ and compute

$$W \leftarrow W - \alpha \sum_{i=1}^K \nabla_W \text{Error}(\vec{y}^{(i)}, f(\vec{x}^{(i)}; W))$$

then this is called **mini-batch stochastic gradient descent**. The batch size becomes a **hyperparameter**

comparison

- **Batch gradient descent:** compute gradient using the *whole dataset*
 - ▶ This gives you the *actual gradient*

comparison

- **Batch gradient descent:** compute gradient using the *whole dataset*
 - ▶ This gives you the *actual gradient*
- **Stochastic gradient descent:** compute an *estimate* of the gradient using a single point of the dataset
 - ▶ This gives you an *estimate* of the gradient

comparison

- **Batch gradient descent:** compute gradient using the *whole dataset*
 - ▶ This gives you the *actual gradient*
- **Stochastic gradient descent:** compute an *estimate* of the gradient using a single point of the dataset
 - ▶ This gives you an *estimate* of the gradient
- **Mini-batch gradient descent:** compute an *estimate* of the gradient using a *subset* of the dataset
 - ▶ This gives you a *better* estimate of the gradient (more stable / less sensitive)

comparison

Question: If batch gradient descent gives you the *actual* gradient, why would we use stochastic or mini-batch which only give estimates?

Answer:

1. They are much faster to compute, especially when you have a huge dataset (imagine 1,000,000 rows!)

comparison

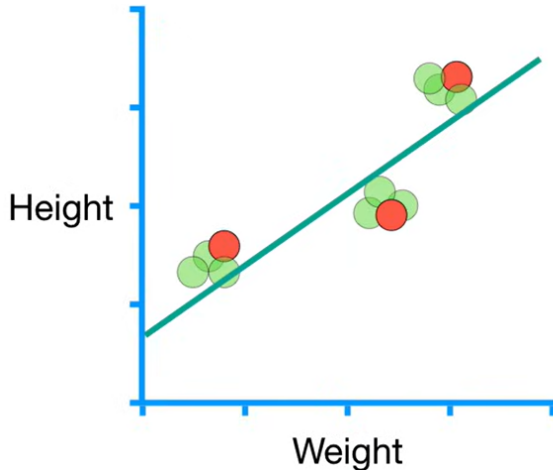
Question: If batch gradient descent gives you the *actual* gradient, why would we use stochastic or mini-batch which only give estimates?

Answer:

1. They are much faster to compute, especially when you have a huge dataset (imagine 1,000,000 rows!)
2. The gradients computed are *unbiased estimators* of the true gradient. In other words, if you average over enough random samples, the direction of these gradients is exactly the same as the true gradient.
3. They can help avoid redundant computations

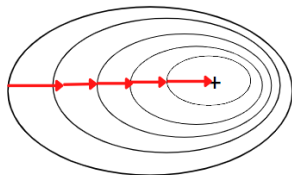
redundancy: visualized

Sometimes there might be natural clusters in our dataset. The gradient will be quite similar for each point in a cluster. Thus, including all of them in our gradient can be redundant. This is avoided in stochastic and mini-batch gradient descent:

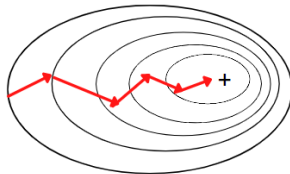


visualizing descent

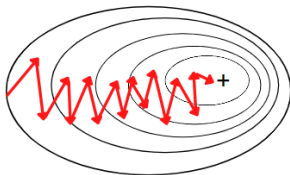
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent



bibliography & further reading

- StatQuest: Logistic Regression -
https://youtu.be/yIYKR4sgzI8?si=Tpa7M60Igk1Dkb_Q
- StatQuest: Stochastic Gradient Descent, Clearly Explained!!! -
https://youtu.be/vMh0zPT0tLI?si=Qe5_ZDLCph_EPRRN