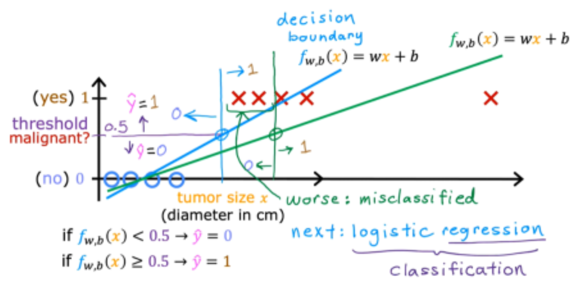
Module 3

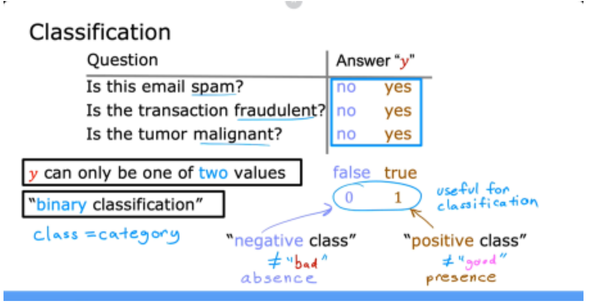
05 August 2025 23:49

In each of these problems the variable that you want to predict can
only be one of two possible values.
No or yes.
This type of classification problem where there are only two possible outputs is
called binary classification.
Where the word binary refers to there being only
two possible classes or two possible categories.
In these problems I will use the terms class and
category relatively interchangeably.
They mean basically the same thing.
By convention we can refer to these two classes or
categories in a few common ways.

From <https://www.coursera.org/learn/machine-learning/lecture/aoMt6/motivations>
Linear regression and try to fit a straight line to the data.
If you do that, maybe the straight line looks like this, right?
And that's your F effects.
Linear regression predicts not just the values zero and one.
But all numbers between zero and one or even less than zero or greater than one.
But here we want to predict categories.
One thing you could try is to pick a threshold of say 0.5.
So that if the model outputs a value below 0.5,
then you predict why equal zero or not malignant.
And if the model outputs a number equal to or
greater than 0.5, then predict Y equals one or malignant.
Notice that this threshold value of 0.5 intersects
the best fit straight line at this point.
So if you draw this vertical line here,
everything to the left ends up with a prediction of y equals zero.
And everything on the right ends up with the prediction of y equals one.
Now, for this particular data set it looks like linear
regression could do something reasonable.
But now let's see what happens if your dataset has one more training example.
This one way over here on the right.
Let's also extend the horizontal axis.
Notice that this training example shouldn't really change how you classify
the data points.
This vertical dividing line that we drew just now still makes sense as the cut off
where tumors smaller than this should be classified as zero.
And tumors greater than this should be classified as one.
But once you've added this extra training example on the right.
The best fit line for linear regression will shift over like this.
And if you continue using the threshold of 0.5, you now notice
that everything to the left of this point is predicted at zero non malignant.
And everything to the right of this point is predicted to be one or malignant.
This isn't what we want because adding that example way to the right shouldn't
change any of our conclusions about how to classify malignant versus benign tumors.
But if you try to do this with linear regression,
adding this one example which feels like it shouldn't be changing anything.
It ends up with us learning a much worse function for this classification problem.
Clearly, when the tumor is large, we want the algorithm to classify it as malignant.
So what we just saw was linear regression causes the best fit line.
When we added one more example to the right to shift over.
And does the dividing line also called the decision
boundary to shift over to the right

From <https://www.coursera.org/learn/machine-learning/lecture/aoMt6/motivations>





Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is the transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes

y can only be one of two values
"binary classification"
class = category
false true
0 1
useful for classification
"negative class" ≠ "bad" absence
"positive class" ≠ "good" presence

Let's talk about logistic regression,
which is probably the single most
widely used classification algorithm in the world.
This is something that I use all the time in my work.
Let's continue with the example of
classifying whether a tumor is malignant.
Whereas before we're going to use the label 1 or
yes to the positive class to represent malignant tumors,
and zero or no and negative examples
to represent benign tumors.
Here's a graph of the dataset where
the horizontal axis is
the tumor size and
the vertical axis takes on only values of 0 and 1,
because is a classification problem.
You saw in the last video that
linear regression is not
a good algorithm for this problem.
In contrast, what logistic regression we end
up doing is fit a curve that looks like this,
S-shaped curve to this dataset.

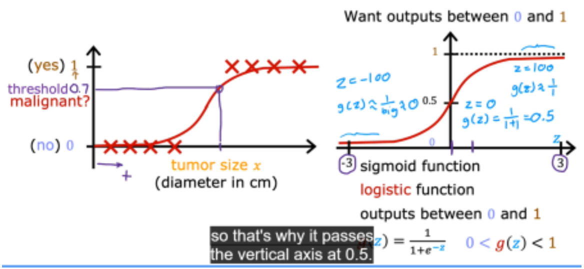
From <https://www.coursera.org/learn/machine-learning/lecture/zNxaw/logistic-regression>
To build out to the logistic regression algorithm,
there's an important mathematical function I like to
describe which is called the Sigmoid function,
sometimes also referred to as the logistic function.
The Sigmoid function looks like this.
Notice that the x-axis of
the graph on the left and right are different.
In the graph to the left on the x-axis is the tumor size,
so is all positive numbers.

From <https://www.coursera.org/learn/machine-learning/lecture/zNxaw/logistic-regression>

So the Sigmoid function outputs value is between 0 and 1.
If I use g of z to denote this function,
then the formula of g of z is equal
to 1 over 1 plus e to the negative z.
Where here e is a mathematical
constant that takes on a value of about 2.7,
and so e to the negative z is that
mathematical constant to the power of negative z.
Notice if z where really be, say a 100,
e to the negative z is e to the
negative 100 which is a tiny number.
So this ends up being 1
over 1 plus a tiny little number,
and so the denominator will be basically very close to 1.
Play video starting at :3:8 and

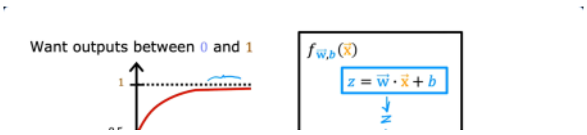
From <https://www.coursera.org/learn/machine-learning/lecture/zNxaw/logistic-regression>

Which is why when z is large,
g of z that is a Sigmoid function
of z is going to be very close to 1.
Conversely, you can also check for yourself
that when z is a very large negative number,
then g of z becomes 1 over a giant number,
which is why g of z is very close to 0.



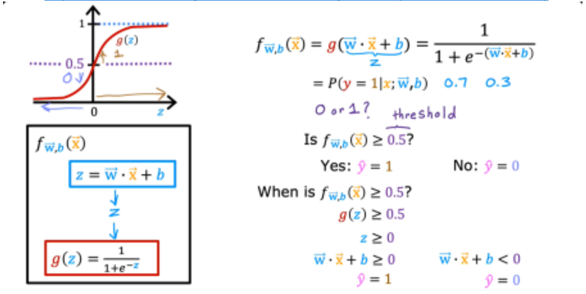
From <https://www.coursera.org/learn/machine-learning/lecture/zNxaw/logistic-regression>
When you take these two equations and put them together,
they then give you the logistic regression model f of x,
which is equal to g of wx plus b.
Or equivalently g of z,
which is equal to this formula over here.
This is the logistic regression model,
and what it does is it inputs feature or set
of features X and outputs a number between 0 and 1.

From <https://www.coursera.org/learn/machine-learning/lecture/zNxaw/logistic-regression>



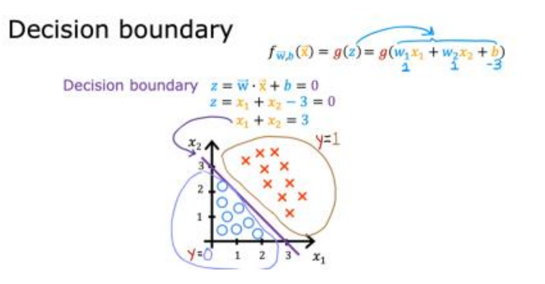
I'm going to take an example of
a classification problem where you have two features,
x1 and x2 instead of just one feature.
Here's a training set where the little red crosses denote
the positive examples and
the little blue circles denote negative examples.
The red crosses corresponds to y equals 1,
and the blue circles correspond to y equals 0.
The logistic regression model will make predictions using
this function f of x equals g of z,
where z is now this expression over here,
w1x1 plus w2x2 plus b,
because we have two features x1 and x2.
Let's just say for this example that
the value of the parameters are w1 equals 1,
w2 equals 1, and b equals negative 3.
Let's now take a look at how
logistic regression makes predictions.
In particular, let's figure out when
wx plus b is greater than equal to
0 and when wx plus b is less than 0.
Play video starting at :5:4 and follow

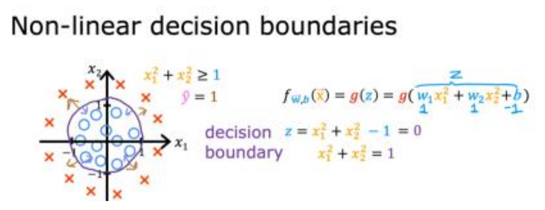
From <https://www.coursera.org/learn/machine-learning/lecture/qrxwU/decision-boundary>

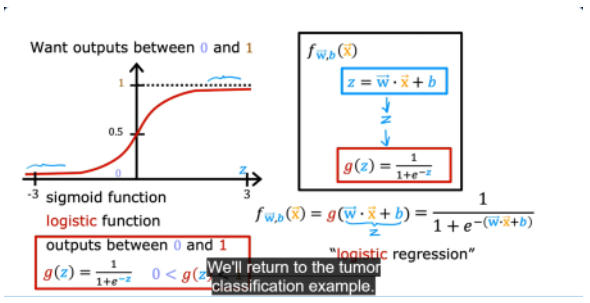

$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w}\cdot\vec{x}+b}_{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$
$= P(y=1|\vec{x};\vec{w},b)$ 0.7 0.3
0 or 1? threshold
Is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?
Yes: $\hat{y}=1$
No: $\hat{y}=0$
When is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?
$g(z) \geq 0.5$
$z \geq 0$
$\vec{w}\cdot\vec{x}+b \geq 0$
$\hat{y}=1$
$\vec{w}\cdot\vec{x}+b < 0$
$\hat{y}=0$
$z = \vec{w}\cdot\vec{x}+b$
$g(z) = \frac{1}{1+e^{-z}}$

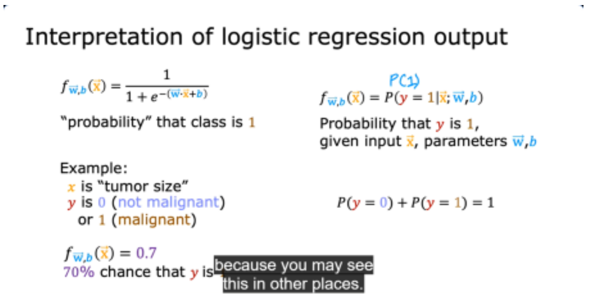
To figure that out,
there's a very interesting line to look at,
which is when wx plus b is exactly equal to 0.
It turns out that this line is also called
the decision boundary because that's the line where
you're just almost neutral about
whether y is 0 or y is 1.
Now, for the values of the parameters w_1, w_2,
and b that we had written down above,
this decision boundary is just x_1 plus x_2 minus 3.
When is x_1 plus x_2 minus 3 equal to 0?
Well, that will correspond to the line
x_1 plus x_2 equals 3,
and that is this line shown over here.
This line turns out to be the decision boundary,
where if the features x are to the right of this line,
logistic regression would predict
1 and to the left of this line,
logistic regression with predicts 0.
In other words, what we have just visualize is
the decision boundary for
logistic regression when the parameters w_1,
w_2, and b are 1,1 and negative 3.

From <https://www.coursera.org/learn/machine-learning/lecture/qrxwU/decision-boundary>


Decision boundary
$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + b)$ 1 1 -3
Decision boundary $z = \vec{w}\cdot\vec{x}+b = 0$
$z = x_1 + x_2 - 3 = 0$
$x_1 + x_2 = 3$
y=1
y=0


Non-linear decision boundaries
$x_1^2 + x_2^2 \geq 1$
$\hat{y}=1$
$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1^2 + w_2x_2^2 + b)$ 1 1 -1
decision boundary $z = x_1^2 + x_2^2 - 1 = 0$
$x_1^2 + x_2^2 = 1$


Want outputs between 0 and 1
sigmoid function
logistic function
outputs between 0 and 1
$g(z) = \frac{1}{1+e^{-z}}$ $0 < g(z)$
$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w}\cdot\vec{x}+b}_{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$
"logistic regression"
$z = \vec{w}\cdot\vec{x}+b$
$g(z) = \frac{1}{1+e^{-z}}$
We'll return to the tumor classification example.

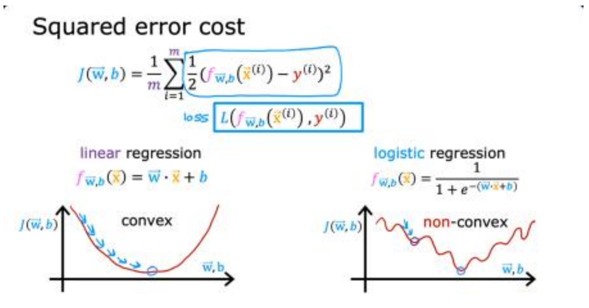

Interpretation of logistic regression output
$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$
"probability" that class is 1
$f_{\vec{w},b}(\vec{x}) = P(y=1|\vec{x};\vec{w},b)$
Probability that y is 1, given input $\vec{x}$, parameters $\vec{w}, b$
Example:
x is "tumor size"
y is 0 (not malignant) or 1 (malignant)
$P(y=0) + P(y=1) = 1$
$f_{\vec{w},b}(\vec{x}) = 0.7$
70% chance that y is
because you may see this in other places.

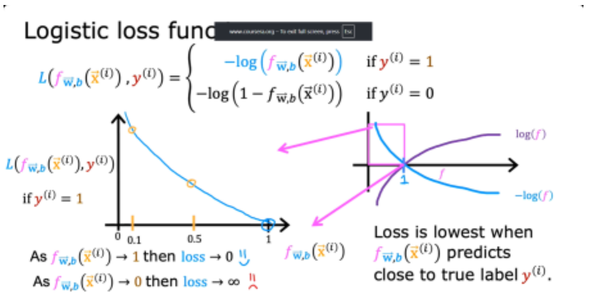
Now you could try to use
the same cost function for logistic regression.
But it turns out that if I were to write f of
x equals 1 over 1 plus e to
the negative wx plus b and
plot the cost function using this value of f of x,
then the cost will look like this.
This becomes what's called a
non-convex cost function is not convex.
What this means is that if
you were to try to use gradient descent.

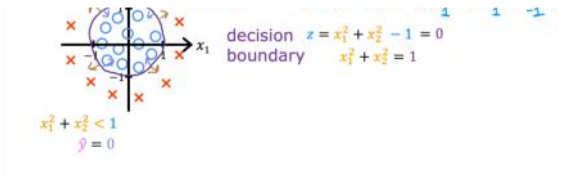
From <https://www.coursera.org/learn/machine-learning/lecture/0hpr8/cost-function-for-logistic-regression>

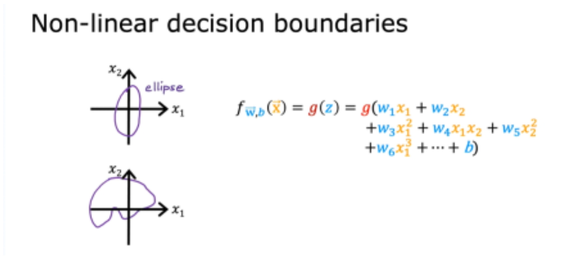
In particular, if you look inside this summation,
let's call this term inside
the loss on a single training example.
I'm going to denote the loss via this capital
L and as a function
of the prediction of the learning algorithm,
f of x as well as of the true label y.
The loss given the predictor f of x and the true label
y is equal in this case to 1.5 of the squared difference.
We'll see shortly that by choosing
a different form for this loss function,
will be able to keep the overall cost function,
which is 1 over n times the sum of
these loss functions to be a convex function.

From <https://www.coursera.org/learn/machine-learning/lecture/0hpr8/cost-function-for-logistic-regression>


Squared error cost
$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$
loss $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$
linear regression
$f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x}+b$
convex
logistic regression
$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$
non-convex


Logistic loss function
$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)}=1 \\ -\log(1-f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)}=0 \end{cases}$
if $y^{(i)}=1$
As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to 0$
As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to \infty$
log(f)
-log(f)
Loss is lowest when $f_{\vec{w},b}(\vec{x}^{(i)})$ predicts close to true label $y^{(i)}$.



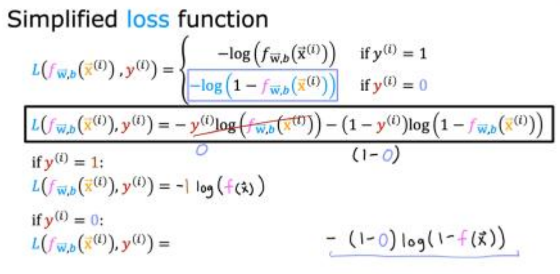


Non-linear decision boundaries

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_1x_2 + w_5x_2^2 + w_6x_1^3 + \dots + b)$$

Given a prediction f of x and the target label y,
the loss equals negative y times log of f minus
1 minus y times log of
1 minus f. It turns out this equation,
which we just wrote in one line,
is completely equivalent to
this more complex formula up here.

From <https://www.coursera.org/learn/machine-learning/lecture/Zjj2j/simplified-cost-function-for-logistic-regression>



Simplified loss function

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ \boxed{-\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)} & \text{if } y^{(i)} = 0 \end{cases}$$

$$\boxed{L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -\underset{0}{\cancel{y^{(i)}}}\log\left(\cancel{f_{\vec{w},b}(\vec{x}^{(i)})}\right) - \underset{(1-0)}{(1 - y^{(i)})}\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)}$$

if $y^{(i)} = 1$:

$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -1 \log(f(\vec{x}))$

if $y^{(i)} = 0$:

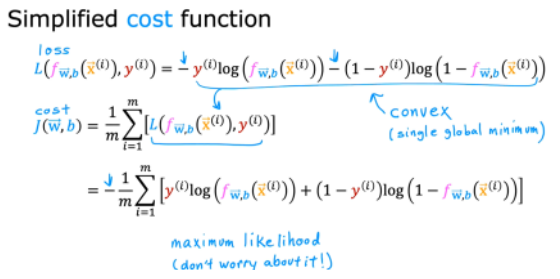
$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \quad - (1-0)\log(1-f(\vec{x}))$

Using this simplified loss function,
let's go back and write out
the cost function for logistic regression.
Here again is the simplified loss function.
Recall that the cost J is just the average loss,
average across the entire training set of m examples.
So it's 1 over
n times the sum of the loss from i equals 1
to m. If you
plug in the definition for
the simplified loss from above,
then it looks like this,
1 over m times the sum of this term above.
If you bring the negative signs and move them outside,
then you end up with this expression over here,
and this is the cost function.

From <https://www.coursera.org/learn/machine-learning/lecture/Zjj2j/simplified-cost-function-for-logistic-regression>

I'd just like to mention that
this particular cost function is derived from
statistics using a statistical principle
called maximum likelihood estimation,
which is an idea from statistics on how to
efficiently find parameters for different models.
This cost function has
the nice property that it is convex.
But don't worry about learning
the details of maximum likelihood.
It's just a deeper rationale and
justification behind this particular cost function.

From <https://www.coursera.org/learn/machine-learning/lecture/Zjj2j/simplified-cost-function-for-logistic-regression>



Simplified cost function

loss

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)$$

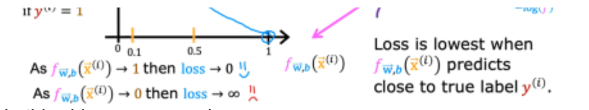
cost

$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left[L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)\right]$$

convex
(single global minimum)

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)\right]$$

maximum likelihood
(don't worry about it!)



As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to 0$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to \infty$

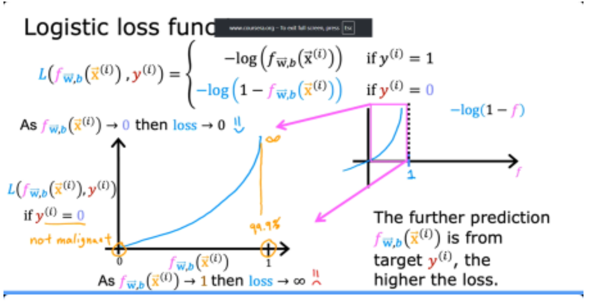
Loss is lowest when $f_{\vec{w},b}(\vec{x}^{(i)})$ predicts close to true label $y^{(i)}$.

In this video, you saw why
the squared error cost function
doesn't work well for logistic regression.
We also defined the loss for
a single training example and
came up with a new definition for
the loss function for logistic regression.
It turns out that with this choice of loss function,
the overall cost function will be convex and thus you can
reliably use gradient descent
to take you to the global minimu

From <https://www.coursera.org/learn/machine-learning/lecture/0hpr8/cost-function-for-logistic-regression>

Proving that this function is convex,
it's beyond the scope of this cost.
You may remember that the cost function is
a function of the entire training set and is,
therefore, the average or 1 over
m times the sum of
the loss function on the individual training examples.
The cost on a certain set of parameters, w and b,
is equal to 1 over
m times the sum of
all the training examples of
the loss on the training examples.
If you can find the value of the parameters, w and b,
that minimizes this, then you'd have
a pretty good set of values for the parameters
w and b for logistic regression.

From <https://www.coursera.org/learn/machine-learning/lecture/0hpr8/cost-function-for-logistic-regression>



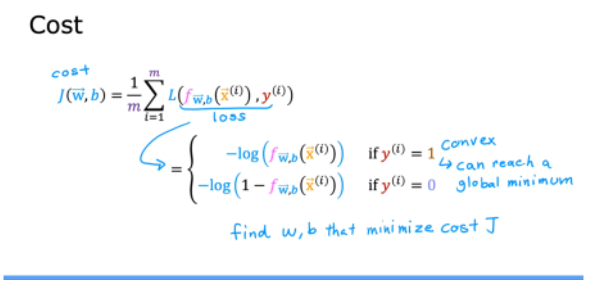
Logistic loss func

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to 0$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to \infty$

The further prediction $f_{\vec{w},b}(\vec{x}^{(i)})$ is from target $y^{(i)}$, the higher the loss.



Cost

cost

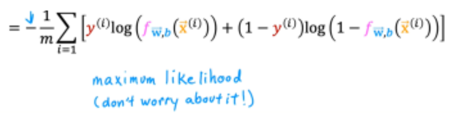
$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\underbrace{L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)}_{\text{loss}}$$

$$= \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

convex
↳ can reach a global minimum

find w, b that minimize cost J

If you want to minimize
the cost j as a function of w and b,
well, here's the usual gradient descent algorithm,
where you repeatedly update
each parameter as the 0 value minus Alpha,
the learning rate times this derivative term.
Let's take a look at the derivative
of j with respect to w_j.
This term up on top here, where as usual,
j goes from one through n,
where n is the number of features.
If someone were to apply the rules of calculus,
you can show that the derivative with respect to w_j of
the cost function capital J is
equal to this expression over here,
is 1 over m times the sum
from 1 through m of this error term.
That is f minus the label y times x_j.
Here are just x I j is
the j feature of training example i.
Now let's also look at the derivative of
j with respect to the parameter b.

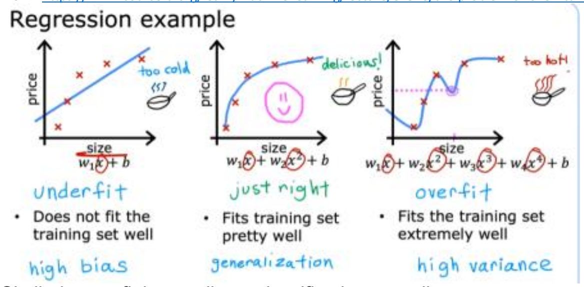


$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right]$$

maximum likelihood
(don't worry about it!)

To recap, if you have
too many features like
the fourth-order polynomial on the right,
then the model may fit the training set well,
but almost too well or overfit and have high variance.
On the flip side if you have too few features,
then in this example, like the one on the left,
it underfits and has high bias.
In this example, using
quadratic features x and x squared,
that seems to be just right.
So far we've looked at underfitting and
overfitting for linear regression model.

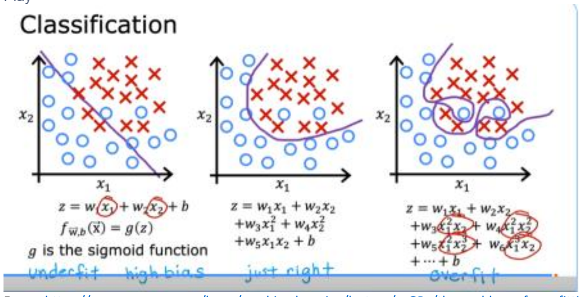
From <https://www.coursera.org/learn/machine-learning/lecture/erGPe/the-problem-of-overfitting>



Similarly, overfitting applies a classification as well.
Here's a classification example
with two features, x_1 and x_2,
where x_1 is maybe
the tumor size and x_2 is the age of patient.
We're trying to classify if
a tumor is malignant or benign,
as denoted by these crosses and circles,
one thing you could do is fit
a logistic regression model.
Just a simple model like this, where as usual,
g is the sigmoid function and this term here inside is z.

From <https://www.coursera.org/learn/machine-learning/lecture/erGPe/the-problem-of-overfitting>

Play



Classification
x_2
x_1
$z = w_1x_1 + w_2x_2 + b$
$f_{\vec{w},b}(\vec{x}) = g(z)$
g is the sigmoid function
$z = w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2 + b$
$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + w_6x_1^3x_2 + \cdots + b$
underfit
high bias
just right
overfit

From <https://www.coursera.org/learn/machine-learning/lecture/erGPe/the-problem-of-overfitting>

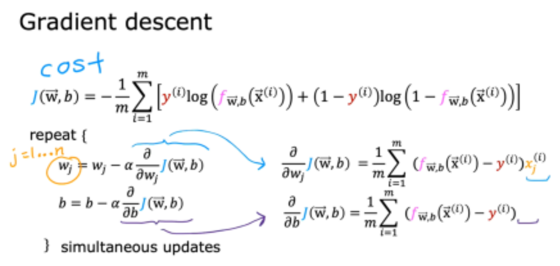
Let's say instead of minimizing this objective function,
this is a cost function for linear regression.
Let's say you were to modify the cost function and
add to it 1000 times W3 squared plus 1000 times W4 squared.
And here I'm just choosing 1000 because it's a big number but
any other really large number would be okay.
So with this modified cost function,
you could in fact be penalizing the model if W3 and W4 are large.
Because if you want to minimize this function, the only way to make this
new cost function small is if W3 and W4 are both small, right?
Because otherwise this 1000 times W3 squared and
1000 times W4 square terms are going to be really, really big.
So when you minimize this function,
you're going to end up with W3 close to 0 and W4 close to 0.

From <https://www.coursera.org/learn/machine-learning/lecture/UZTPk/cost-function-with-regularization>

And if we do that, then we end up with a fit to the data that's much closer to
the quadratic function,
including maybe just tiny contributions from the features x cubed and extra 4.
And this is good because it's a much better fit to the data compared to if all
the parameters could be large and you end up with this weekly quadratic function
more generally, here's the idea behind regularization.
The idea is that if there are smaller values for the parameters,

That is I minus the label y times x_J.
Here are just x I j is
the j feature of training example i.
Now let's also look at the derivative of
j with respect to the parameter b.

From <https://www.coursera.org/learn/machine-learning/lecture/Ha1RP/gradient-descent-implementation>



Gradient descent

cost

$$J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right]$$

repeat {

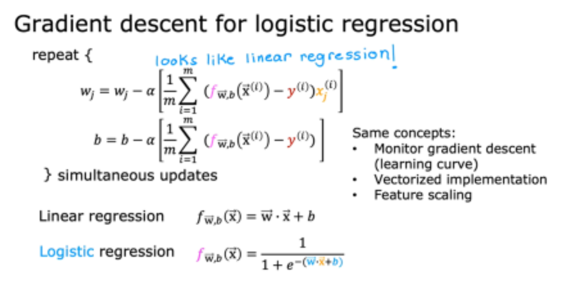
$j=1\ldots n$ $\quad w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\vec{w},b)$ $\qquad \frac{\partial}{\partial w_j}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}$

$b = b - \alpha\frac{\partial}{\partial b}J(\vec{w},b)$ $\qquad \frac{\partial}{\partial b}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$

} simultaneous updates

It turns out to be this expression over here.
It's quite similar to the expression above,
except that it is not multiplied by
this x superscript i subscript j at the end.
Just as a reminder,
similar to what you saw for linear regression,
the way to carry out these updates is
to use simultaneous updates,
meaning that you first
compute the right-hand side for all of
these updates and then simultaneously
overwrite all the values on the left at the same time.
Let me take these derivative expressions
here and plug them into these terms here.
This gives you gradient descent for logistic regression.

From <https://www.coursera.org/learn/machine-learning/lecture/Ha1RP/gradient-descent-implementation>



Gradient descent for logistic regression

repeat { $\qquad$ looks like linear regression!

$$w_j = w_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}\right]$$

$$b = b - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)\right]$$

} simultaneous updates

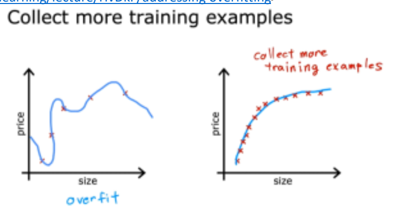
Same concepts:
- Monitor gradient descent (learning curve)
- Vectorized implementation
- Feature scaling

Linear regression $\quad f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$

Logistic regression $\quad f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$

You can continue to fit
a high order polynomial
or some of the function with a lot of features,
and if you have enough training examples,
it will still do okay.
To summarize, the number one tool you can
use against overfitting is to get more training
data.
Now, getting more data isn't always an option.
Maybe only so many houses have
been sold in this location,
so maybe there just isn't more data to be add.
But when the data is available,
this can work really well.

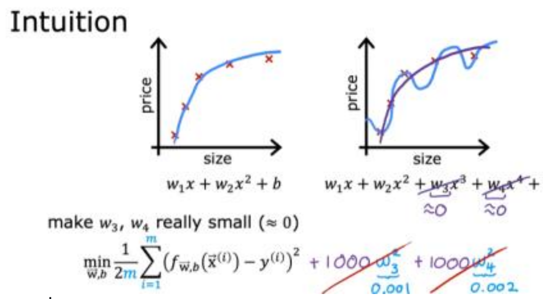
From <https://www.coursera.org/learn/machine-learning/lecture/HvDkF/addressing-overfitting>



It turns out that if you have a lot of features like
these but don't have enough training data,
then your learning algorithm may
also overfit to your training set.
Now instead of using all 100 features,
if we were to pick just a subset of the most useful
ones,
maybe size, bedrooms,
and the age of the house.
If you think those are the most relevant features,
then using just that smallest subset of features,

including maybe just tiny contributions from the features x cubed and extra 4.
And this is good because it's a much better fit to the data compared to if all
the parameters could be large and you end up with this weekly quadratic function
more generally, here's the idea behind regularization.
The idea is that if there are smaller values for the parameters,
then that's a bit like having a simpler model.
Maybe one with fewer features, which is therefore less prone to overfitting.
On the last slide we penalize or
we say we regularized only W3 and W4.
But more generally, the way that regularization tends to be implemented is
if you have a lot of features, say a 100 features, you may not know which
are the most important features and which ones to penalize

From <https://www.coursera.org/learn/machine-learning/lecture/UZTPk/cost-function-with-regularization>



$w_1x + w_2x^2 + b$

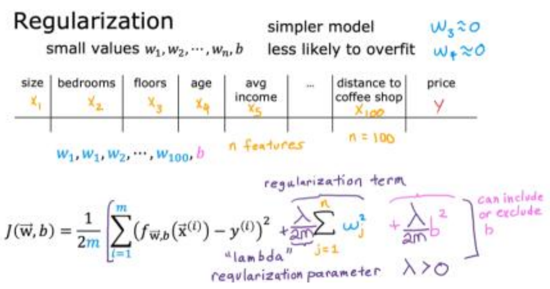
$w_1x + w_2x^2 + \underbrace{w_3x^3}_{\approx 0} + \underbrace{w_4x^4}_{\approx 0} + b$

$$\min_{\vec{w},b} \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \cancel{1000\underbrace{w_3^2}_{0.001}} + \cancel{1000\underbrace{w_4^2}_{0.002}}$$

t may be
hard to pick an advance which features to include and which ones to exclude.
So let's build a model that uses all 100 features.
So you have these 100 parameters W1 through W100,
as well as 100 and first parameter B.
Because we don't know which of these parameters are going to be
the important ones.
Let's penalize all of them a bit and shrink all of them by adding
this new term lambda times the sum from J equals 1 through n where n is 100.
The number of features of wj squared.

From <https://www.coursera.org/learn/machine-learning/lecture/UZTPk/cost-function-with-regularization>
This value lambda here is the Greek alphabet lambda and
it's also called a regularization parameter.
So similar to picking a learning rate alpha,
you now also have to choose a number for lambda.
A couple of things I would like to point out by convention,
instead of using lambda times the sum of wj squared.
We also divide lambda by 2m so that both the 1st and
2nd terms here are scaled by 1 over 2m.
It turns out that by scaling both terms the same way
it becomes a little bit easier to choose a good value for lambda.
And in particular you find that even if your training set size growth,
say you find more training examples.
So m the training set size is now bigger
Also by the way,
by convention we're not going to penalize the parameter b for being large.
In practice, it makes very little difference whether you do or not.
And some machine learning engineers and actually some learning algorithm
implementations will also include lambda over 2m times the b squared term.
But this makes very little difference in practice and
the more common convention which was used in this course is to regularize
only the parameters w rather than the parameter b.

From <https://www.coursera.org/learn/machine-learning/lecture/UZTPk/cost-function-with-regularization>

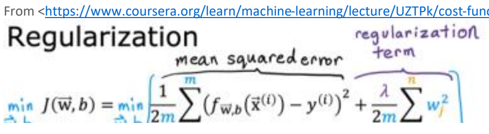


size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

$$J(\vec{w},b) = \frac{1}{2m}\left[\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2 + \frac{\lambda}{2m}b^2\right]$$

From <https://www.coursera.org/learn/machine-learning/lecture/UZTPk/cost-function-with-regularization>
Trying to minimize this first term encourages the algorithm to fit
the training data well by minimizing the squared differences of the predictions and
the actual values.
And try to minimize the second term.
The algorithm also tries to keep the parameters wj small,
which will tend to reduce overfitting.
The value of lambda that you choose, specifies the relative importance or
the relative trade off or how you balance between these two goals.
Let's take a look at what different values of lambda will cause you're
learning algorithm to do.
Le

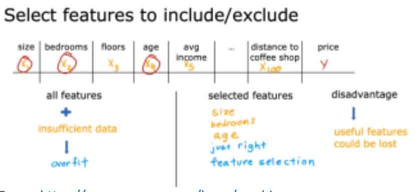
From <https://www.coursera.org/learn/machine-learning/lecture/UZTPk/cost-function-with-regularization>



$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2\right]$$

maybe size, bedrooms,
and the age of the house.
If you think those are the most relevant features,
then using just that smallest subset of features,
you may find that your model no longer overfits
as badly.

From <https://www.coursera.org/learn/machine-learning/lecture/HvDkF/addressing-overfitting>
Choosing the most appropriate set of features to
use is sometimes also called feature selection.
One way you could do so is to use
your intuition to choose what you
think is the best set of features,
what's most relevant for predicting the price.
Now, one disadvantage of feature selection
is that by using only a subset of the features,
the algorithm is throwing away some of
the information that you have about the houses.
For example, maybe all of these features,
all 100 of them are actually
useful for predicting the price of a house.
Maybe you don't want to throw away some of
the information by throwing away some of the
features

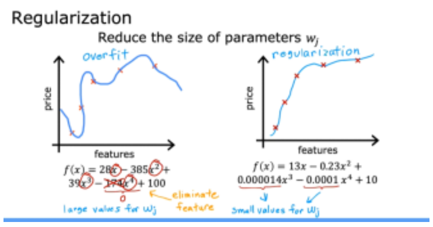


size	bedrooms	floors	age	avg income	...	distance to coffee shop	price
x_1	x_2	x_3	x_4	x_5		x_{100}	y

From <https://www.coursera.org/learn/machine-learning/lecture/HvDkF/addressing-overfitting>
This technique, which we'll look at in even greater
depth
in the next video is called regularization.
If you look at an overfit model,
here's a model using polynomial features: x,
x squared, x cubed, and so on.
You find that the parameters are often relatively
large.
Now if you were to
eliminate some of these features, say,
if you were to eliminate the feature x4,
that corresponds to setting this parameter to 0.
So setting a parameter to 0
is equivalent to eliminating a feature,
which is what we saw on the previous slide.
It turns out that regularization
is a way to more gently reduce
the impacts of some of the features without
doing something as harsh as eliminating it
outright.
What regularization does is encourage
the learning algorithm to shrink the values of
the parameters without necessarily
demanding that the parameter is set to exactly 0.
Play video
By the way, by convention,
we normally just reduce the size of the wj
parameters,
that is w1 through wn.
It doesn't make a huge difference whether you
regularize the parameter b as well,
you could do so if you want or not if you don't.
I usually don't and it's just
fine to regularize w1, w2,
all the way to wn,
but not really encourage b to become smaller.
In practice, it should make very little difference
whether you also regularize b or not.

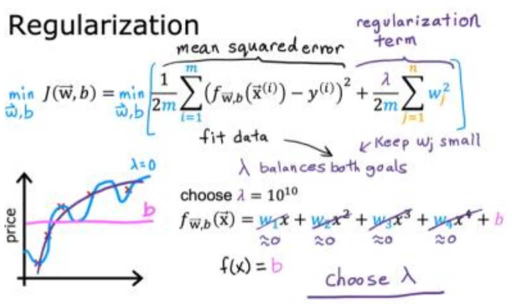
From <https://www.coursera.org/learn/machine-learning/lecture/HvDkF/addressing-overfitting>

From <https://www.coursera.org/learn/machine-learning/lecture/HvDkF/addressing-overfitting>



$f(x) = 28x - 385x^2 + 39x^3 - \cancel{174}x^4 + 100$

$f(x) = 13x - 0.23x^2 + 0.000014x^3 - 0.0001x^4 + 10$



Regularization

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\overbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}^{\text{mean squared error}} + \overbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}^{\text{regularization term}}\right]$$

fit data → λ balances both goals ← Keep w_j small

choose $\lambda = 10^{10}$

$$f_{\vec{w},b}(\vec{x}) = \underset{\approx 0}{\cancel{w_1}}x + \underset{\approx 0}{\cancel{w_2}}x^2 + \underset{\approx 0}{\cancel{w_3}}x^3 + \underset{\approx 0}{\cancel{w_4}}x^4 + b$$

$f(x) = b$

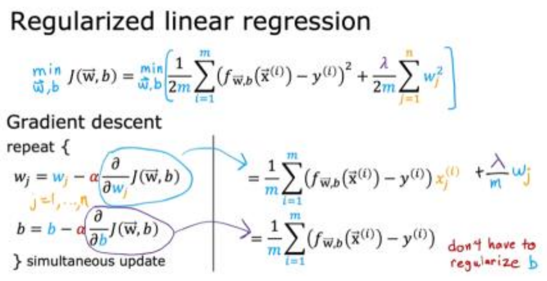
Choose λ

To recap if lambda is 0 this model will over fit If
lambda is enormous like 10 to the power of 10.
This model will under fit.
And so what you want is some value of lambda that is in between that more
appropriately balances these first and second terms of trading off,
minimizing the mean squared error and keeping the parameters small.
And when the value of lambda is not too small and not too large, but
just right, then hopefully you end up able to fit a 4th order polynomial,
keeping all of these features, but with a function that looks like this.
So that's how regularization works.

From <https://www.coursera.org/learn/machine-learning/lecture/UZTPk/cost-function-with-regularization>

Now that we've added this additional regularization term,
the only thing that changes is that the expression for
the derivative with respect to
w_j ends up with one additional term,
this plus Lambda over m times w_j.
And in particular for
the new definition of the cost function j,
these two expressions over here,
these are the new derivatives of J with respect
to w_j and the derivative of J with respect to b.
Recall that we don't regularize b,
so we're not trying to shrink B.
That's why the updated B remains the same as before,
whereas the updated w changes because
the regularization term causes us to try to shrink w_j.

From <https://www.coursera.org/learn/machine-learning/lecture/WRULa/regularized-linear-regression>

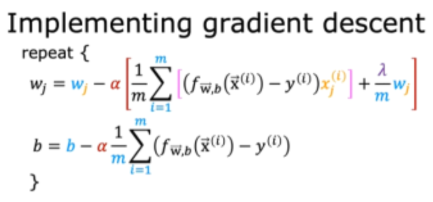


Regularized linear regression

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2\right]$$

Gradient descent

```
repeat {
    $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b)$        $= \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m} w_j$
        $j = 1, \dots, n$
    $b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b)$        $= \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$   don't have to regularize b
} simultaneous update
```

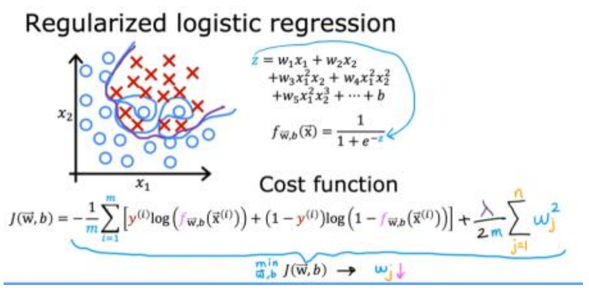


Implementing gradient descent

```
repeat {
    $w_j = w_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left[\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}\right] + \frac{\lambda}{m} w_j\right]$
    $b = b - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$
}
```

This was the cost function for logistic regression.
If you want to modify it to use regularization,
all you need to do is add to it the following term.
Let's add lambda to regularization parameter over
2m times the sum from j equals 1 through n,
where n is the number of features as usual of wj squared.
When you minimize this cost function
as a function of w and b,
it has the effect of penalizing parameters w_1,
w_2 through w_n,
and preventing them from being too large.
If you do this, then even though you're fitting
a high order polynomial with a lot of parameters,
you still get a decision boundary that looks like this.
Something that looks more reasonable
for separating positive and negative examples
while also generalizing hopefully
to new examples not in the training set.

From <https://www.coursera.org/learn/machine-learning/lecture/cAxpF/regularized-logistic-regression>



Regularized logistic regression

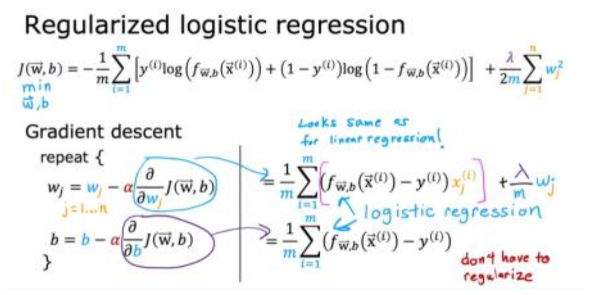
$$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + \cdots + b$$

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-z}}$$

Cost function

$$J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + \left(1-y^{(i)}\right)\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

$$\min_{\vec{w},b} J(\vec{w},b) \rightarrow w_j \downarrow$$



Regularized logistic regression

$$\underset{\vec{w},b}{\min} J(\vec{\mathrm{w}},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

Gradient descent

Looks same as for linear regression!

```
repeat {
    $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}},b)$    $j=1 \ldots n$
    $b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}},b)$
}
```

$$\frac{\partial}{\partial w_j} J(\vec{\mathrm{w}},b) = \frac{1}{m}\sum_{i=1}^{m}\left[\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)x_j^{(i)}\right] + \frac{\lambda}{m} w_j$$

logistic regression

$$\frac{\partial}{\partial b} J(\vec{\mathrm{w}},b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)$$

don't have to regularize