LOGISTIC REGRESSION IN PYTHON

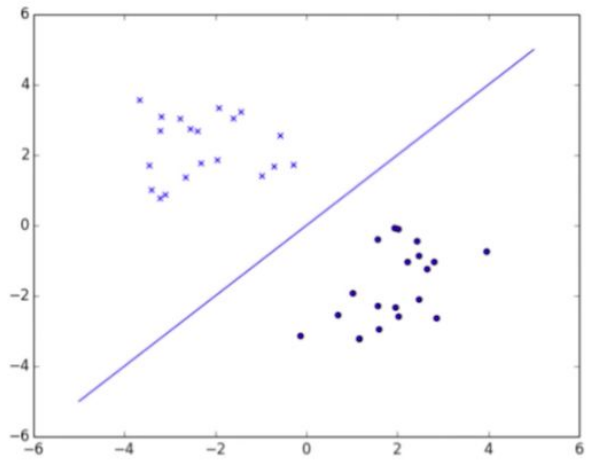
2-dimensional classification

$y = mx + b$

or

$0 = ax + by + c$

$a = 1, b = -1, c = 0$



Machine learning lingo

We call $(x,y) \rightarrow (x_1, x_2) = \mathbf{x}$

We rename the constants to w_i

We call the bias term / intercept w_0

$h(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2$

We say h() is a **linear combination** of the components of **x**

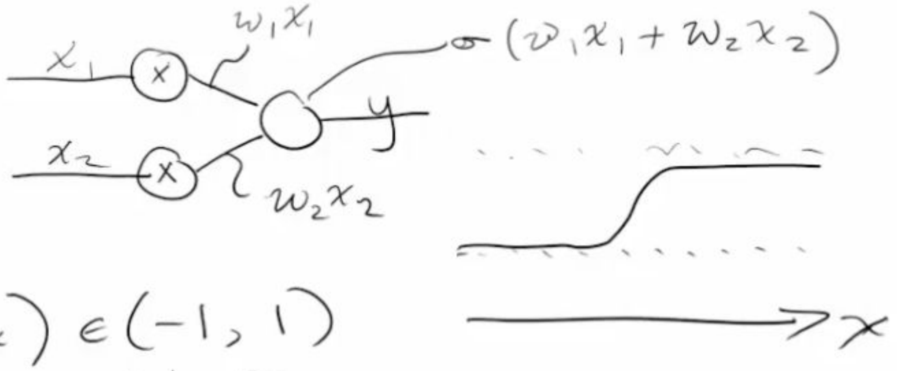
In vector form: $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

In 3-dimensions: line → plane, In > 3 dimensions: hyperplane

Neuron Analogies

1. Many inputs → One output
2. Spike or no spike → 0/1 output
3. Synapse strengths → linear weights

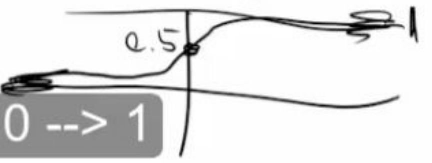
mber 6, 2015 11:24 PM



$\tanh(x) \in (-1, 1)$
y-int 0

$\sigma(z) = \frac{1}{1+e^{-z}} \in (0, 1)$ y-int 0.5

$\sigma(w^T x)$

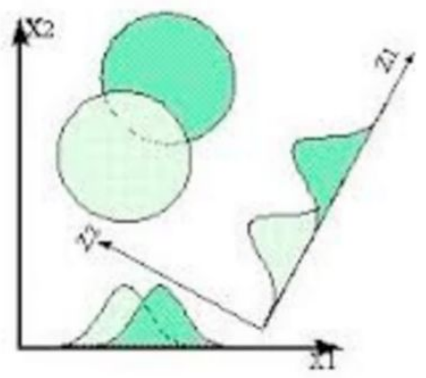


Before we had w*x > 0 --> 1

Problem Description

- 2 Gaussian Clouds
- Both have the same covariance
- Multivariate Gaussian PDF:

$$p(x) = \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$



Bayes' Rule

p(Y | X) = p(X | Y)p(Y) / p(X)

I.e.:

p(Y=1 | X) = p(X | Y=1)p(Y=1) / p(X)
p(Y=0 | X) = p(X | Y=0)p(Y=0) / p(X)

- p(X | Y) is the Gaussian - we calculate it over all the data that belongs to class Y
- p(Y) is just the frequency estimate of Y →
  e.g. p(Y=1) = #times class 1 appeared / # total

Put it into the logistic regression framework

Manipulate Bayes' Rule

$$p(y = 1 \mid x) = \frac{p(x \mid y{=}1)p(y{=}1)}{p(x)} = \frac{p(x \mid y{=}1)p(y{=}1)}{p(x \mid y{=}1)p(y{=}1) + p(x \mid y{=}0)p(y{=}0)}$$

Divide top and bottom by p(x | y=1)p(y=1)

$$p(y = 1 \mid x) = \frac{1}{1 + \frac{p(x \mid y{=}0)p(y{=}0)}{p(x \mid y{=}1)p(y{=}1)}}$$

Looks a lot like Logistic Regression!

Put it into the logistic regression framework

Manipulate Bayes' Rule

$$p(y = 1 \mid x) = \frac{p(x \mid y{=}1)p(y{=}1)}{p(x)} = \frac{p(x \mid y{=}1)p(y{=}1)}{p(x \mid y{=}1)p(y{=}1) + p(x \mid y{=}0)p(y{=}0)}$$

Divide top and bottom by p(x | y=1)p(y=1)

$$p(y = 1 \mid x) = \frac{1}{1 + \frac{p(x \mid y{=}0)p(y{=}0)}{p(x \mid y{=}1)p(y{=}1)}}$$

Looks a lot like Logistic Regression!

Logistic Regression

$$p(y=1 \mid x) = \frac{1}{1+\frac{p(x \mid y=0)p(y=0)}{p(x \mid y=1)p(y=1)}} = \frac{1}{1+exp(-w^T x)}$$

$$-w^T x = ln\left(\frac{p(x \mid y=0)p(y=0)}{p(x \mid y=1)p(y=1)}\right)$$

Let’s just manipulate the right side and remember that it equals to the left

Logistic Regression

Let: $p(y=1) = 1-\alpha,\ p(y=0) = \alpha$

$$ln\left(\frac{p(x \mid y=0)p(y=0)}{p(x \mid y=1)p(y=1)}\right) = ln\, p(x \mid y=0) + ln\alpha - ln\, p(x \mid y=1) - ln(1-\alpha)$$

$$= ln\frac{1}{\sqrt{(2\pi)^D|\Sigma|}}e^{-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)} - ln\frac{1}{\sqrt{(2\pi)^D|\Sigma|}}e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)} + ln\frac{\alpha}{1-\alpha}$$

Logistic Regression

$$= ln \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} e^{-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)} - ln \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)} + ln \frac{\alpha}{1-\alpha}$$

1 / sqrt(...) cancels out

$$\begin{aligned}
=& -\tfrac{1}{2}(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_0 - \mu_0{}^T\Sigma^{-1}x + \mu_0{}^T\Sigma^{-1}\mu_0) \\
&+ \tfrac{1}{2}(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_1 - \mu_1{}^T\Sigma^{-1}x + \mu_1{}^T\Sigma^{-1}\mu_1) \\
&+ ln \tfrac{\alpha}{1-\alpha}
\end{aligned}$$

Logistic Regression

$$\begin{aligned}
=& -\tfrac{1}{2}(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_0 - \mu_0{}^T\Sigma^{-1}x + \mu_0{}^T\Sigma^{-1}\mu_0) \\
&+ \tfrac{1}{2}(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_1 - \mu_1{}^T\Sigma^{-1}x + \mu_1{}^T\Sigma^{-1}\mu_1) \\
&+ ln \tfrac{\alpha}{1-\alpha}
\end{aligned}$$

- Quadratic term cancels out
- Is $x^T\Sigma^{-1}\mu = \mu^T\Sigma^{-1}x$? Yes! Try it on paper to prove to yourself. Remember the covariance, and hence its inverse, is symmetric

Logistic Regression

$$= \mu_0{}^T\Sigma^{-1}x - \mu_1{}^T\Sigma^{-1}x - \tfrac{1}{2}\mu_0{}^T\Sigma^{-1}\mu_0 + \tfrac{1}{2}\mu_1{}^T\Sigma^{-1}\mu_1 + ln\tfrac{\alpha}{1-\alpha}$$

$$= (\mu_0{}^T - \mu_1{}^T)\Sigma^{-1}x - \tfrac{1}{2}\mu_0{}^T\Sigma^{-1}\mu_0 + \tfrac{1}{2}\mu_1{}^T\Sigma^{-1}\mu_1 + ln\tfrac{\alpha}{1-\alpha}$$

$$= \quad -(w^Tx+b)$$

Finally!

$$w^T = (\mu_1{}^T - \mu_0{}^T)\Sigma^{-1}$$
$$b = \tfrac{1}{2}\mu_0{}^T\Sigma^{-1}\mu_0 - \tfrac{1}{2}\mu_1{}^T\Sigma^{-1}\mu_1 - ln\tfrac{\alpha}{1-\alpha}$$

Our data

$\mu_0 = (-2, -2)^T$, $\mu_1 = (+2, +2)^T$, $\sigma^2 = 1$

w = (4, 4), b = 0 (assume equal number of samples of each class)

- Exercise: Solve it yourself on paper!
- Exercise: Make sure you can do the derivation yourself without watching the lectures.
- Also called: Linear Discriminant Analysis (LDA)
- If covariance is a diagonal matrix: Naive Bayes (why?)
- If we have different covariances: Quadratic Discriminant Analysis (QDA)
  - Exercise: Write your own code and compare the performance vs. LDA
- This solution is optimal provided assumptions about distributions are true

Input Size

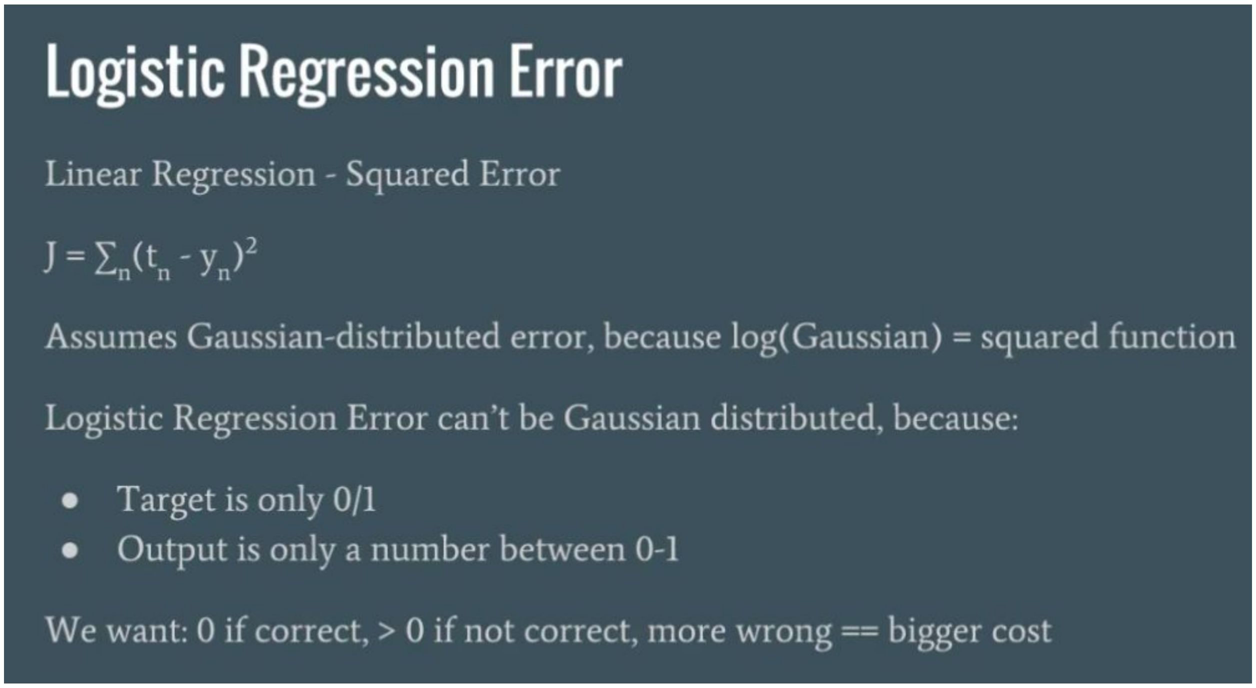
- N = number of samples we've collected
  - i.e. I measure the height of 10 students to find the average height. N=10
- D = number of dimensions or features per sample
  - i.e I measure height, weight, and girth, to try and predict body fat %. D=3 (height, weight, girth)
- A matrix of data called X would be an NxD matrix
  - Each row is a sample
  - Each column is the value of one feature in each sample

Data and Targets

- Sometimes we say we train a model on inputs X and targets Y
- If we consider all Ys at the same time, then it is an Nx1 matrix (for binary classification, an Nx1 matrix of 0s and 1s)
- Sometimes we use T for the target instead
  - i.e. The cross-entropy error function: $-[\, t \log(y) + (1 - t) \log(1 - y) \,]$
- Now Y is used for something else: output of logistic regression.
- Before the output was more precise, $P(Y=1 \mid X)$
- But that takes longer to write...

Cost function

- Cost function = error function = objective function
- We want to minimize cost and error - easy to see semantically
- Objective function can mean something to maximize OR minimize
- Trivial sign flipping - minimizing x^2 = maximizing $-x^2$
- We will first maximize likelihood - P(data | model)
- Same as maximizing log-likelihood L = log P(data | model)
  - i.e. If $A > B$, then $\log(A) > \log(B)$
- Sometimes, we use the letter J
  - May be in the form $J = L \rightarrow$ Maximize it
  - Or $J = -L \rightarrow$ Minimize it



Logistic Regression Error

Linear Regression - Squared Error

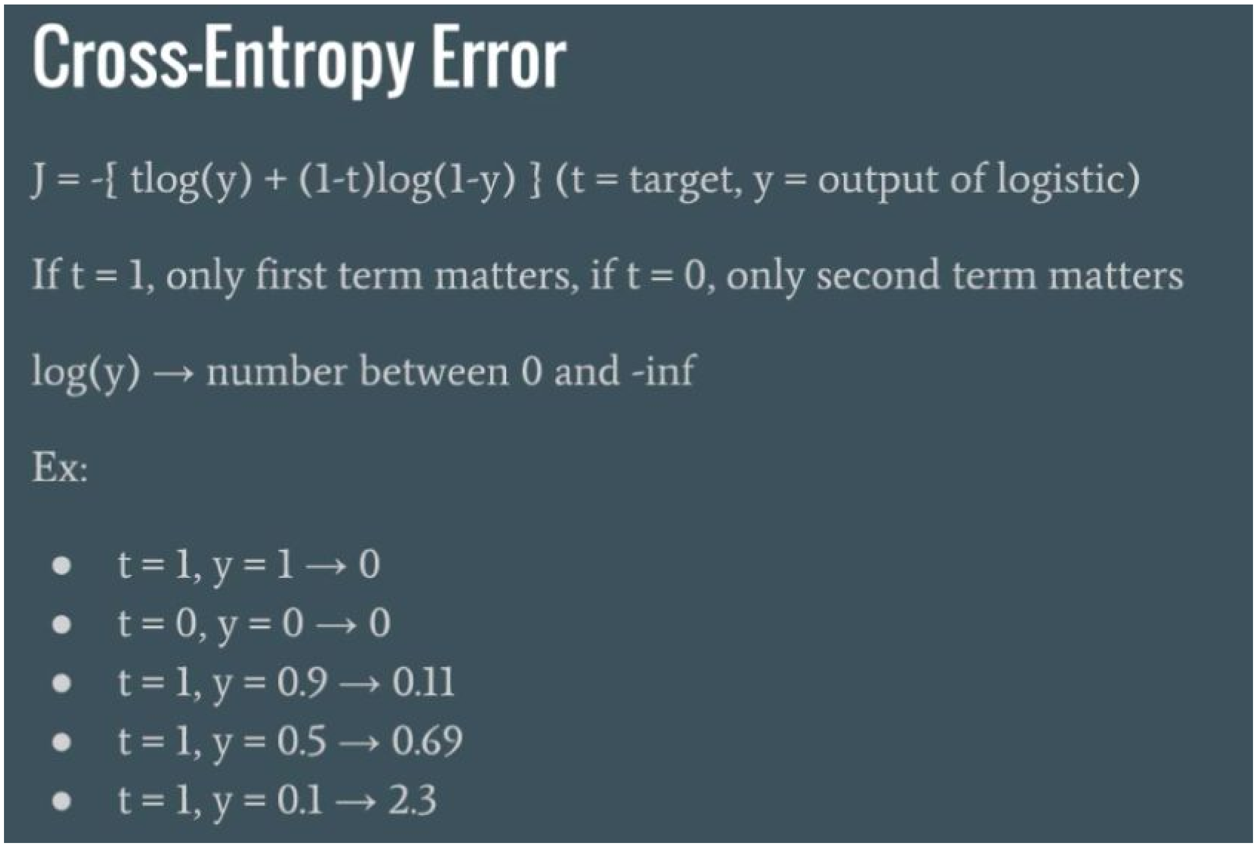
$J = \sum_n (t_n - y_n)^2$

Assumes Gaussian-distributed error, because log(Gaussian) = squared function

Logistic Regression Error can't be Gaussian distributed, because:

- Target is only 0/1
- Output is only a number between 0-1

We want: 0 if correct, > 0 if not correct, more wrong == bigger cost



Cross-Entropy Error

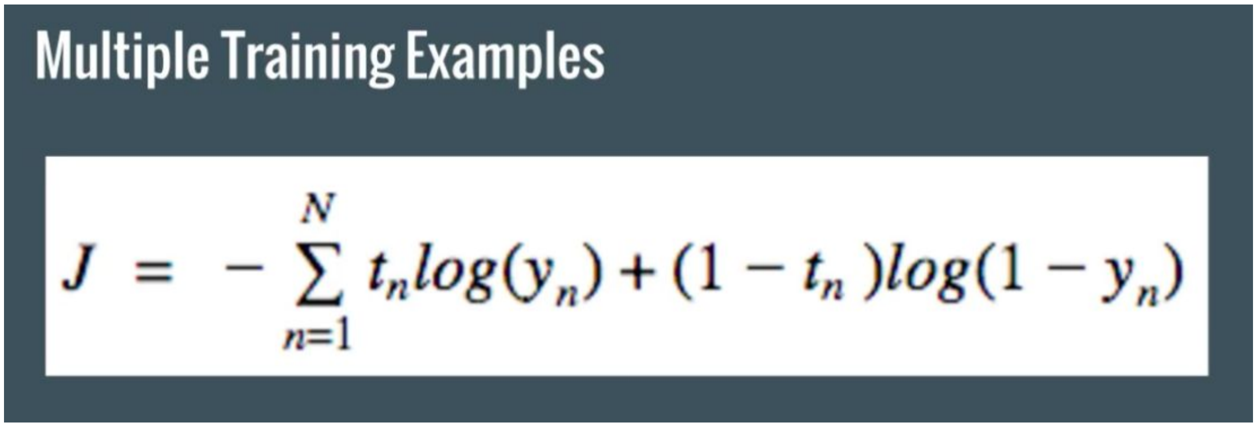
$J = -\{ t\log(y) + (1-t)\log(1-y) \}$ (t = target, y = output of logistic)

If t = 1, only first term matters, if t = 0, only second term matters

$\log(y) \rightarrow$ number between 0 and -inf

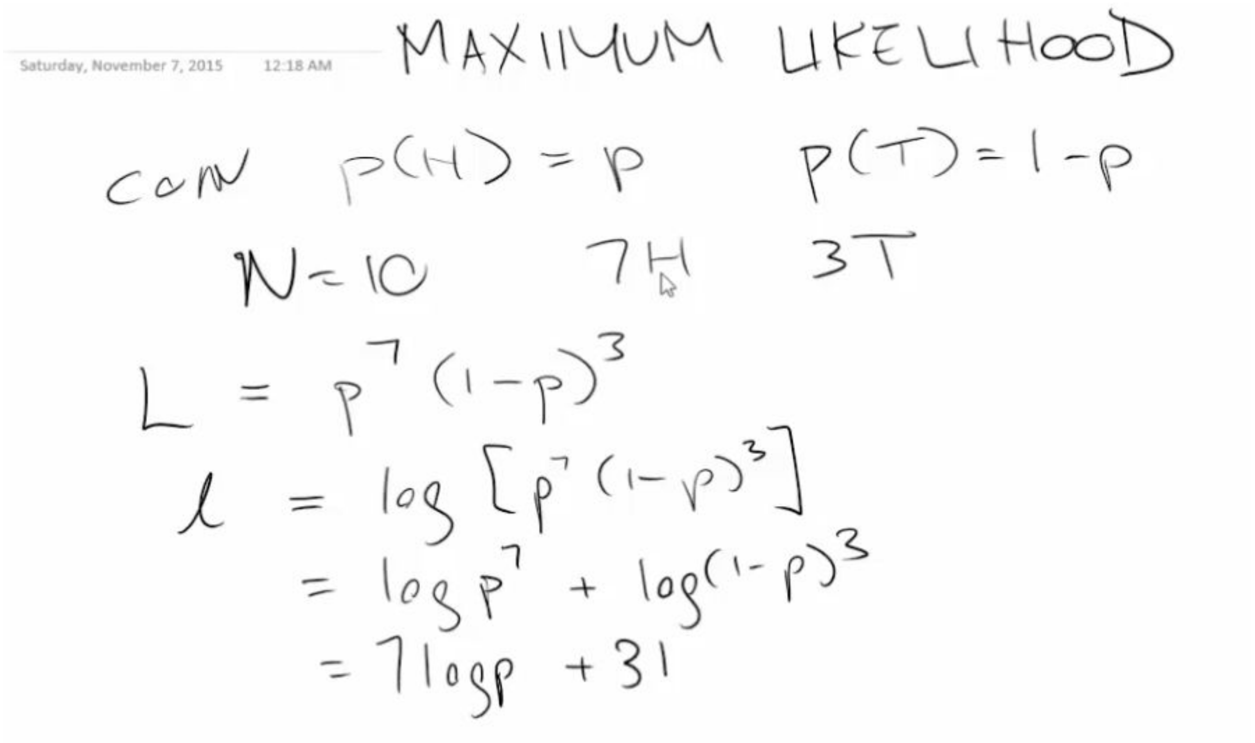
Ex:

- $t = 1, y = 1 \rightarrow 0$
- $t = 0, y = 0 \rightarrow 0$
- $t = 1, y = 0.9 \rightarrow 0.11$
- $t = 1, y = 0.5 \rightarrow 0.69$
- $t = 1, y = 0.1 \rightarrow 2.3$



Multiple Training Examples

$$J = -\sum_{n=1}^{N} t_n log(y_n) + (1 - t_n) log(1 - y_n)$$



Saturday, November 7, 2015 12:18 AM

MAXIIMUM LIKELIHOOD

coin $p(H) = p$ $p(T) = 1-p$

$N = 10$ $7H$ $3T$

$$L = p^7 (1-p)^3$$

$$\begin{aligned} \ell &= \log \left[p^7 (1-p)^3 \right] \\ &= \log p^7 + \log(1-p)^3 \\ &= 7 \log p + 3 l \end{aligned}$$

$L = p \; (\; 1 \;)$

$$\ell = \log\left[p^7 (1-p)^3\right]$$

$$= \log p^7 + \log(1-p)^3$$

$$= 7\log p + 3\log(1-p)$$

$$\frac{\partial \ell}{\partial p} = \frac{7}{p} + \frac{3}{1-p}(-1) = 0$$

$$\frac{7}{p} = \frac{3}{1-p}$$

$$\frac{1-p}{p} = \frac{3}{7}$$

$$\ell = \log\left[p^7 (1-p)^3\right]$$
$$= \log p^7 + \log(1-p)^3$$
$$= 7\log p + 3\log(1-p)$$
$$\frac{\partial \ell}{\partial p} = \frac{7}{p} + \frac{3}{1-p}(-1) = 0$$
$$\frac{7}{p} = \frac{3}{1-p}$$
$$\frac{1-p}{p} = \frac{3}{7}$$
$$\frac{1}{p} - 1 = \frac{3}{7}$$

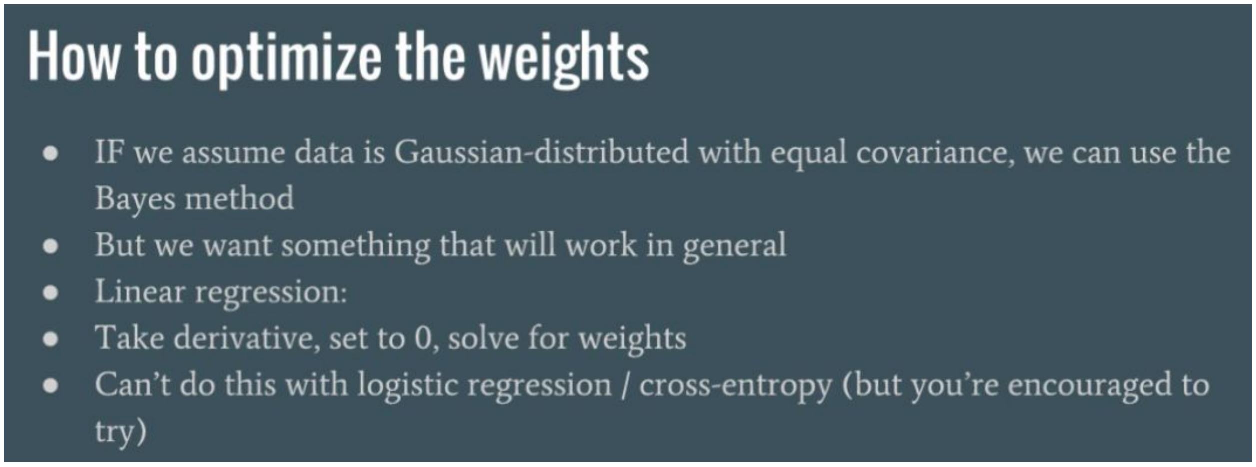
$$\frac{1}{p} = \frac{10}{7}$$

$$p = \frac{7}{10} = p(H)$$

$$P(y = 1 \mid x) = \sigma(w^T x) = y$$

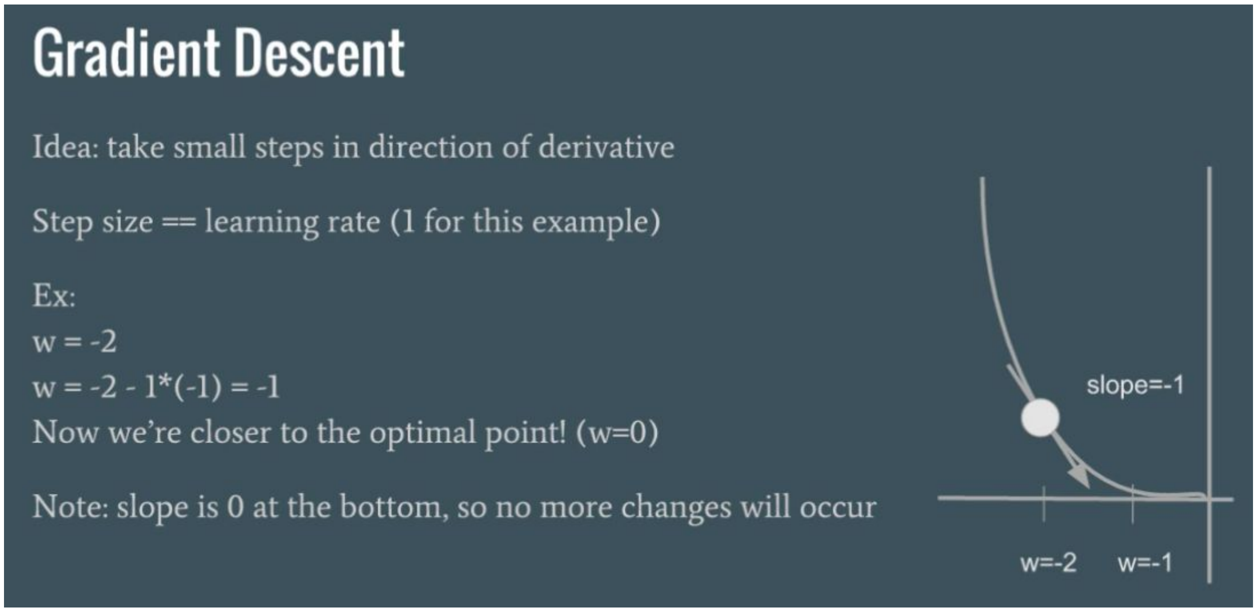
$$L = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1 - t_n}$$

$$l = \sum_n t_n \log y_n + (1 - t_n) \log (1 - y_n)$$



How to optimize the weights

- IF we assume data is Gaussian-distributed with equal covariance, we can use the Bayes method
- But we want something that will work in general
- Linear regression:
- Take derivative, set to 0, solve for weights
- Can't do this with logistic regression / cross-entropy (but you're encouraged to try)



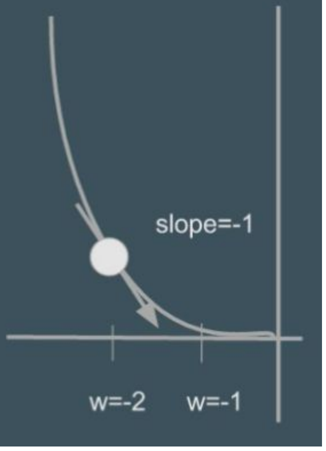
Gradient Descent

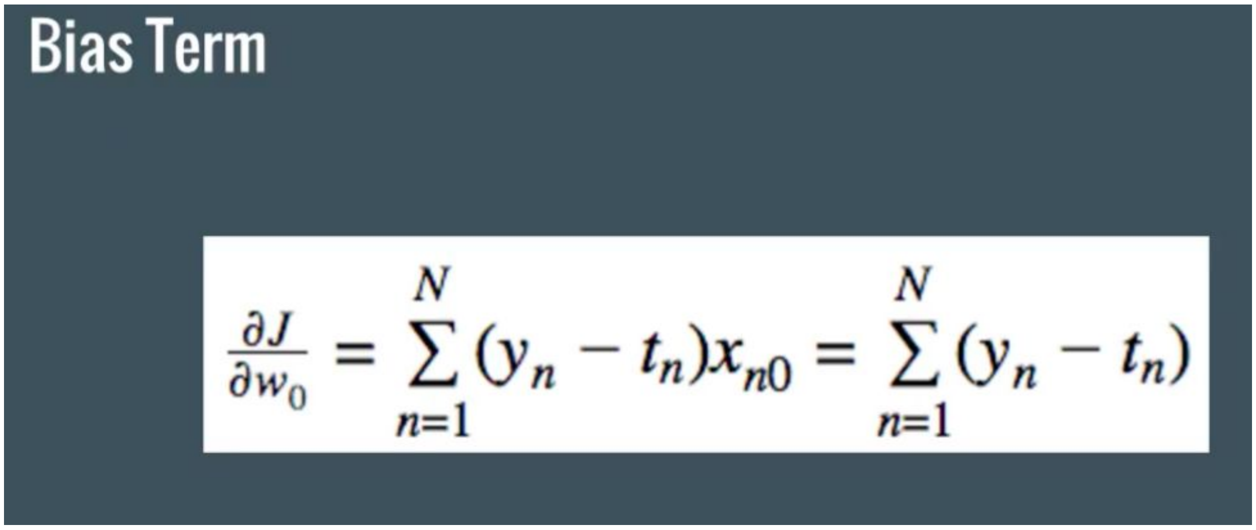
Idea: take small steps in direction of derivative

Step size == learning rate (1 for this example)

Ex:
w = -2
w = -2 - 1*(-1) = -1
Now we're closer to the optimal point! (w=0)

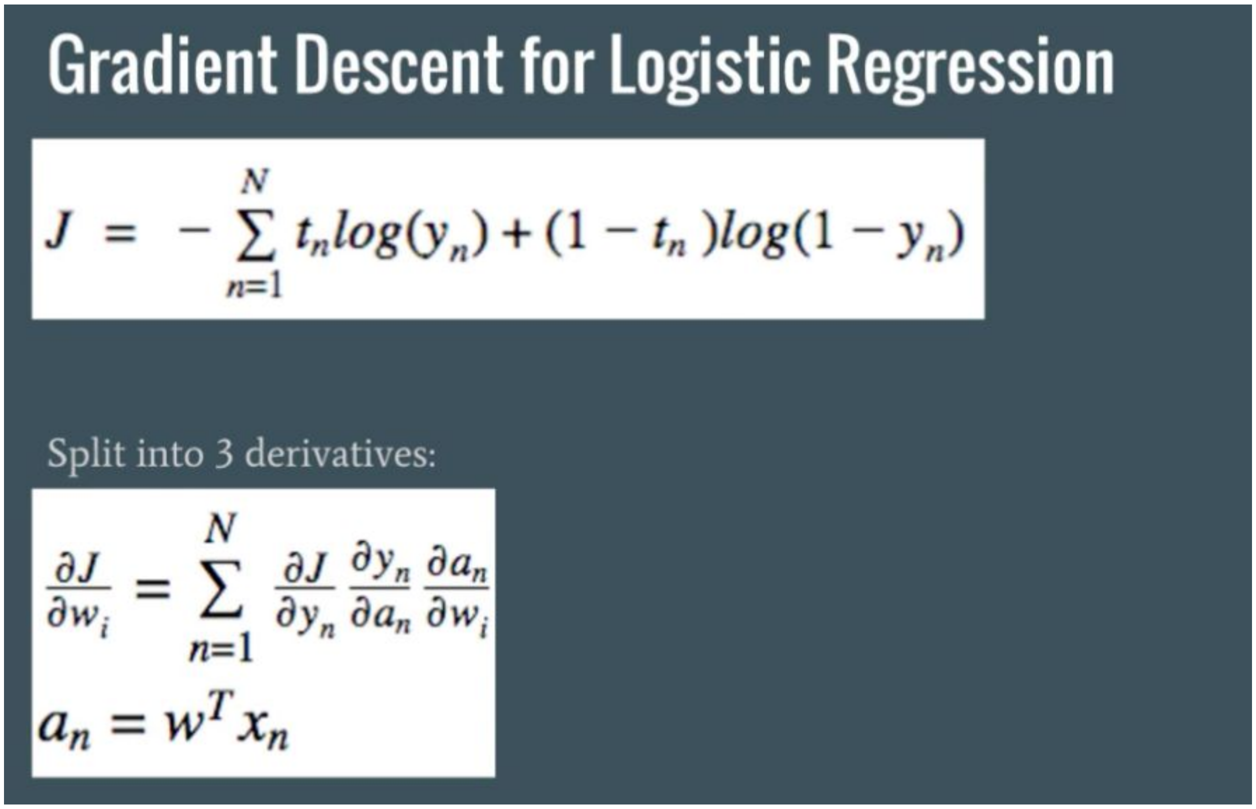
Note: slope is 0 at the bottom, so no more changes will occur





Bias Term

$$\frac{\partial J}{\partial w_0} = \sum_{n=1}^{N}(y_n - t_n)x_{n0} = \sum_{n=1}^{N}(y_n - t_n)$$



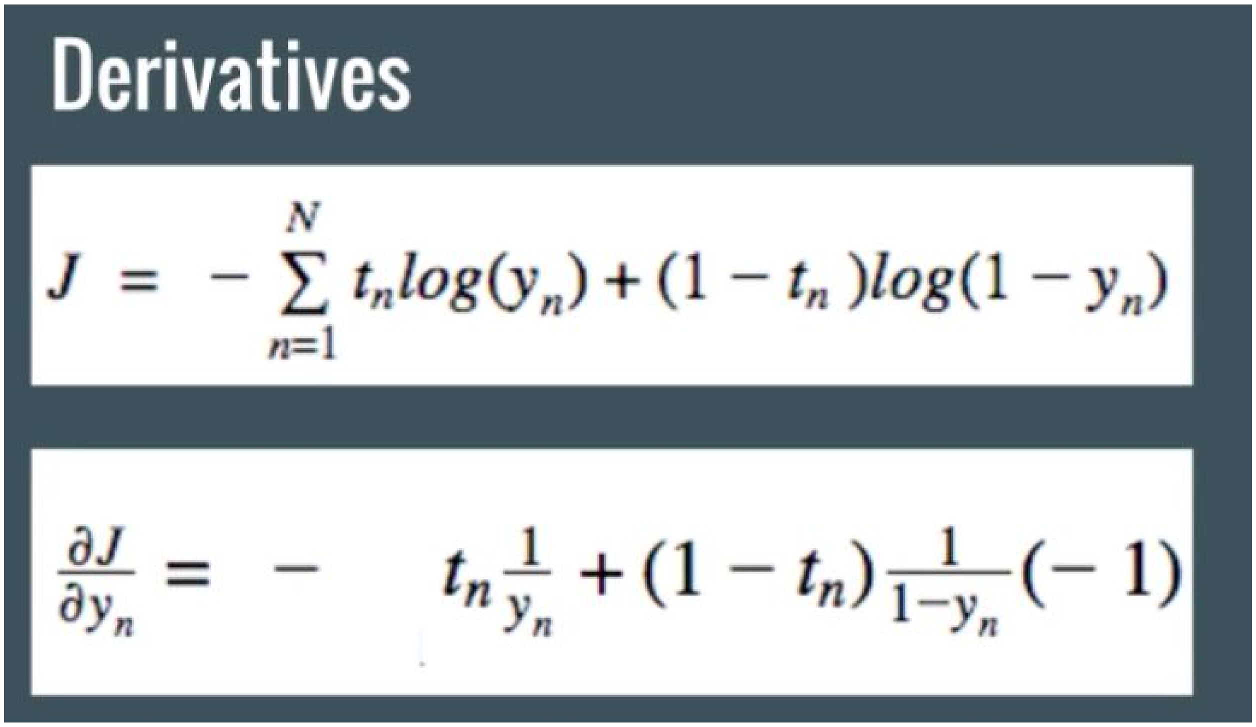
Gradient Descent for Logistic Regression

$$J = -\sum_{n=1}^{N} t_n log(y_n) + (1 - t_n) log(1 - y_n)$$

Split into 3 derivatives:

$$\frac{\partial J}{\partial w_i} = \sum_{n=1}^{N} \frac{\partial J}{\partial y_n} \frac{\partial y_n}{\partial a_n} \frac{\partial a_n}{\partial w_i}$$

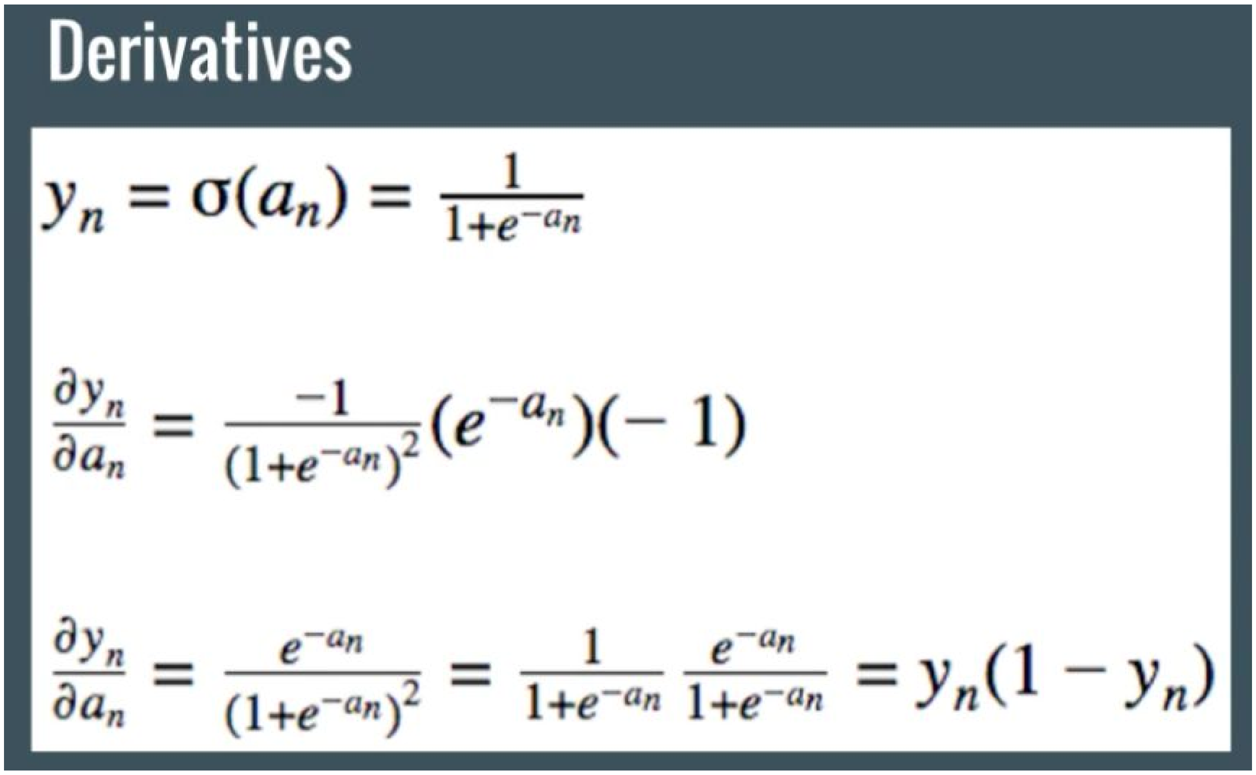
$$a_n = w^T x_n$$



Derivatives

$$J = -\sum_{n=1}^{N} t_n log(y_n) + (1 - t_n) log(1 - y_n)$$

$$\frac{\partial J}{\partial y_n} = - \quad t_n \frac{1}{y_n} + (1 - t_n) \frac{1}{1 - y_n} (-1)$$

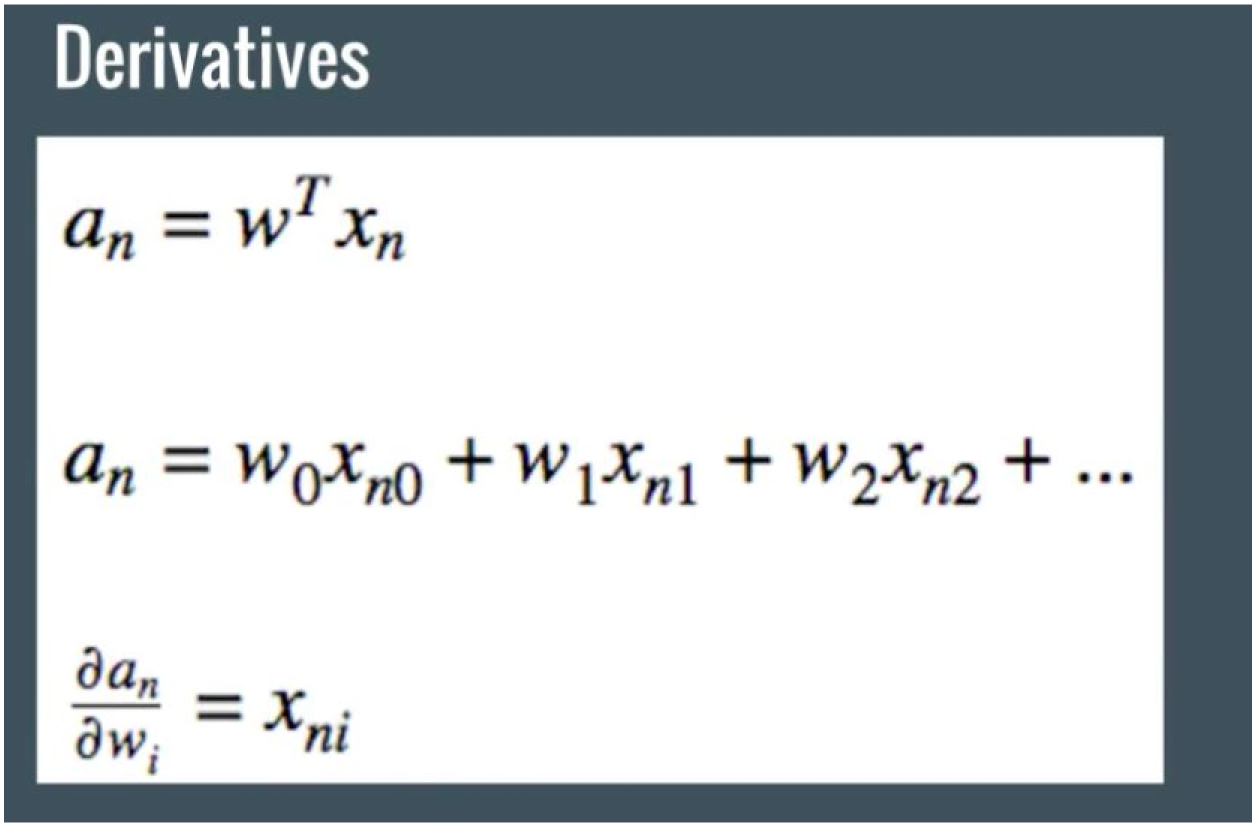


Derivatives

$$y_n = \sigma(a_n) = \frac{1}{1+e^{-a_n}}$$

$$\frac{\partial y_n}{\partial a_n} = \frac{-1}{(1+e^{-a_n})^2}(e^{-a_n})(-1)$$

$$\frac{\partial y_n}{\partial a_n} = \frac{e^{-a_n}}{(1+e^{-a_n})^2} = \frac{1}{1+e^{-a_n}}\frac{e^{-a_n}}{1+e^{-a_n}} = y_n(1-y_n)$$

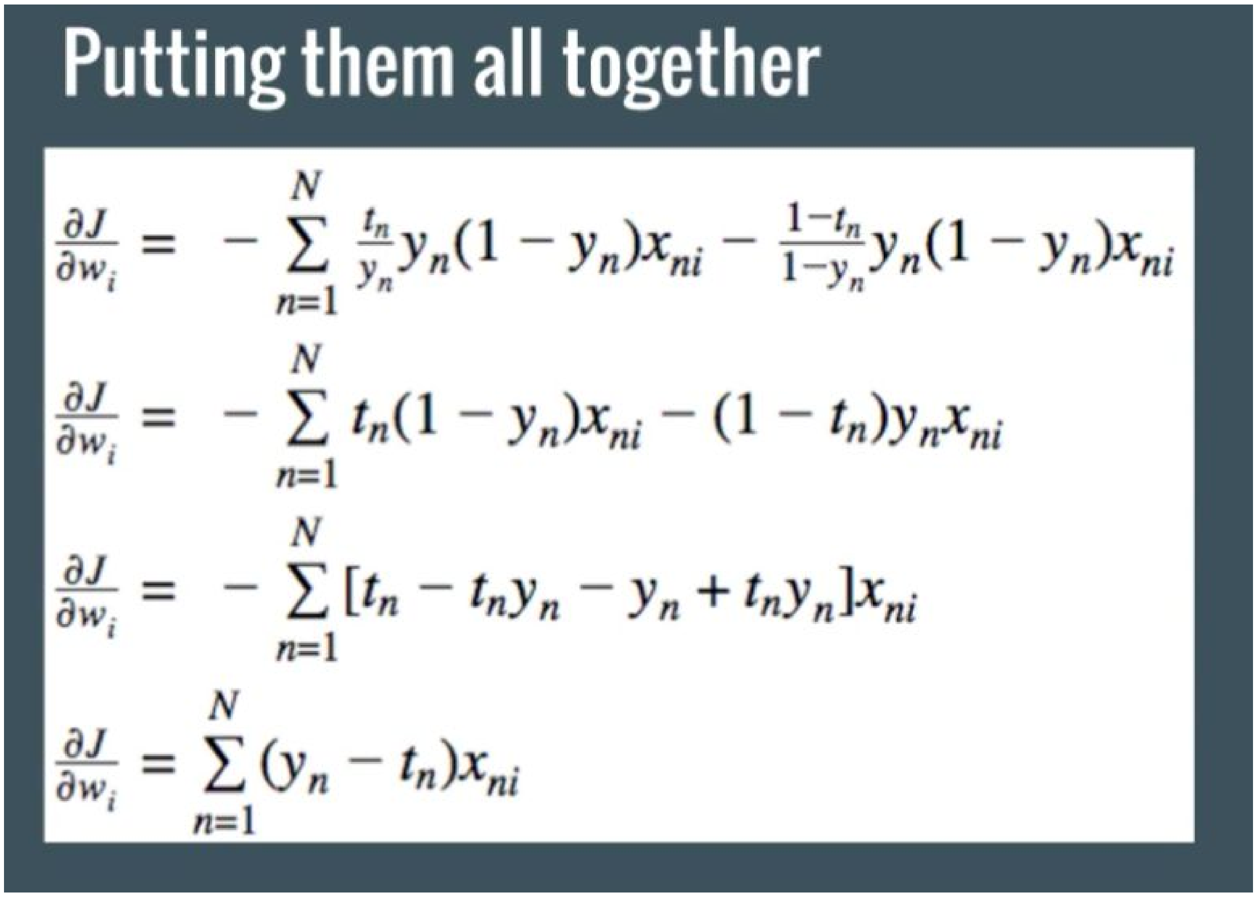


Derivatives

$$a_n = w^T x_n$$

$$a_n = w_0 x_{n0} + w_1 x_{n1} + w_2 x_{n2} + \ldots$$

$$\frac{\partial a_n}{\partial w_i} = x_{ni}$$



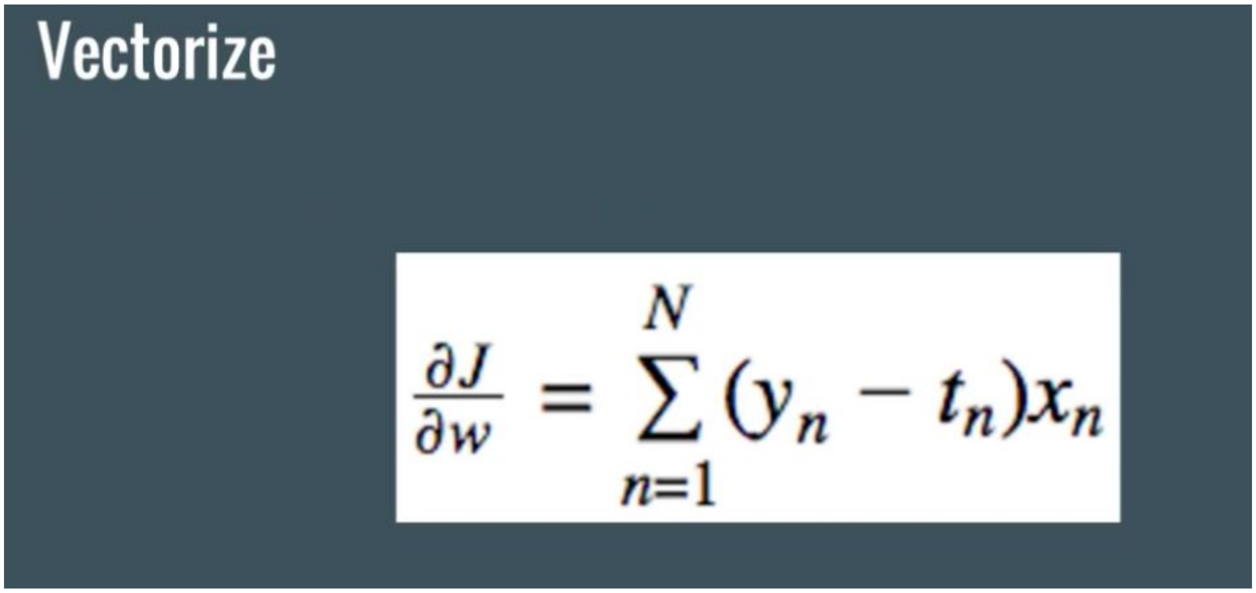
Putting them all together

$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} \frac{t_n}{y_n} y_n(1 - y_n)x_{ni} - \frac{1-t_n}{1-y_n} y_n(1 - y_n)x_{ni}$$

$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} t_n(1 - y_n)x_{ni} - (1 - t_n)y_n x_{ni}$$

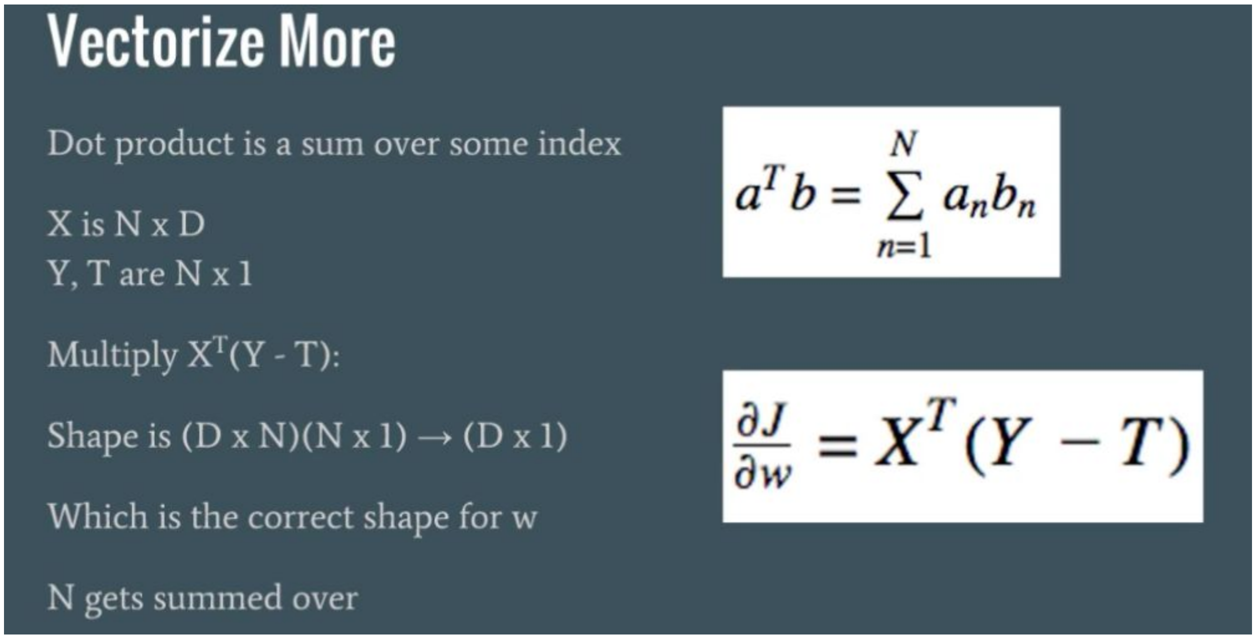
$$\frac{\partial J}{\partial w_i} = -\sum_{n=1}^{N} [t_n - t_n y_n - y_n + t_n y_n]x_{ni}$$

$$\frac{\partial J}{\partial w_i} = \sum_{n=1}^{N} (y_n - t_n)x_{ni}$$



Vectorize

$$\frac{\partial J}{\partial w} = \sum_{n=1}^{N} (y_n - t_n)x_n$$



Vectorize More

Dot product is a sum over some index

$$a^T b = \sum_{n=1}^{N} a_n b_n$$

X is N x D
Y, T are N x 1

Multiply $X^T(Y - T)$:

Shape is (D x N)(N x 1) $\rightarrow$ (D x 1)

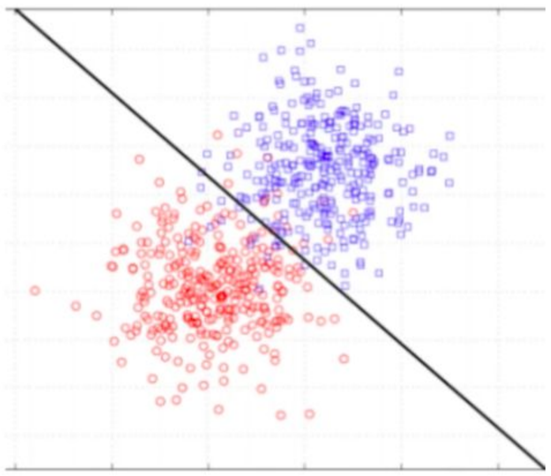
$$\frac{\partial J}{\partial w} = X^T(Y - T)$$

Which is the correct shape for w

N gets summed over

Generalization and Overfitting

- Recall our “Gaussian cloud” problem
- 2 clouds
- One centered at (2,2)
- Other centered at (-2,-2)
- Exact Bayesian solution was:
- w = [0, 4, 4]
- Represent as y = mx + b
- High school math!
- 0 + 4x + 4y = 0
- y = -x



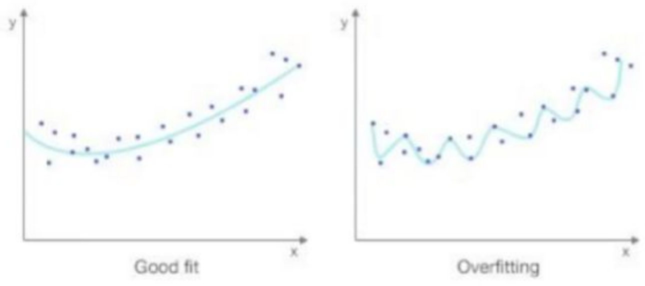
y = -x

- Slope: -1, y-intercept: 0
- Why is the Bayesian solution (4, 4)?
- Why not (1, 1)? Why not (10, 10)?
- These all would represent the same line
- First hint at why we might need regularization

Objective function

- $J = t\log(y) + (1-t)\log(1-y)$
- This y is the output of logistic regression
- Take a test point $(x_1, x_2) = (1, 1)$ and existing weights (0, 4, 4)
- Should be classified as 1
- $\sigma(0 + 4 + 4) = \sigma(8) = 0.99966$
- What would be better? Exactly 1!
- With $y = \sigma(8)$, $J = -0.00033540637289566265$
- What if my weights are (0, 1, 1)? → $J = -0.12692801104297263$
- Not as good
- What if my weights are (0, 10, 10)? → $J = -2.0611536942919273\text{e-}09$
- The "best" weights are thus (0, infinity, infinity)
- In the computer, that's an error

Regularization

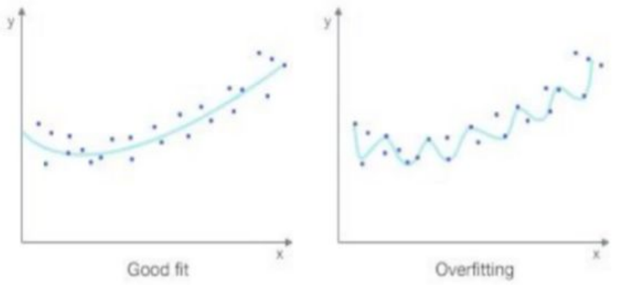


- People usually explain regularization in terms of overfitting (image on the right)
- But this is regression so it's not exactly applicable
- If your data fills up the space of all possible inputs, you shouldn't overfit even if your model is very complex
- That's why we want to have lots of data
- Your model overfits when it has to "guess" what the output should be in a space it's never seen
- But if the train set entirely covers all future test sets, then it's fine

Regularization

- This scenario is different - we could have a perfectly split up dataset that covers the entire possible input space
- Logistic regression would still try to go to w = (0, infinity, infinity)
- Solution: regularization
- Penalizes very large weights
- Existing cost function: J = -[tlog(y) + (1 - t)log(1 - y)]
- Add a penalty for big weights:
- $J_{reg} = J + (\lambda/2)||w||^2 = J + (\lambda/2)w^Tw$
- Now we won't try to go to (0, 10, 10)
- λ = smoothing parameter (usually ~0.1, 1) but depends on your data
- No universal formula to choose (maybe, but too advanced for this class)

Regularization



- People usually explain regularization in terms of overfitting (image on the right)
- But this is regression so it's not exactly applicable
- If your data fills up the space of all possible inputs, you shouldn't overfit even if your model is very complex
- That's why we want to have lots of data
- Your model overfits when it has to "guess" what the output should be in a space it's never seen
- But if the train set entirely covers all future test sets, then it's fine

Regularization

- This scenario is different - we could have a perfectly split up dataset that covers the entire possible input space
- Logistic regression would still try to go to w = (0, infinity, infinity)
- Solution: regularization
- Penalizes very large weights
- Existing cost function: $J = -[\, t\log(y) + (1 - t)\log(1 - y) \,]$
- Add a penalty for big weights:
- $J_{reg} = J + (\lambda/2)||w||^2 = J + (\lambda/2)w^Tw$
- Now we won't try to go to (0, 10, 10)
- λ = smoothing parameter (usually ~0.1, 1) but depends on your data
- No universal formula to choose (maybe, but too advanced for this class)

Solving for w

- Still do gradient descent
- Now we want dJ_{reg} / dw instead of dJ / dw
- Addition doesn't affect other terms in derivative
- reg_cost = $(\lambda/2)(\, w_0^2 + w_1^2 + w_2^2 + \dots \,)$
- d(reg_cost)/$dw_i = \lambda w_i$
- In vector form: d(reg_cost)/dw = λw
- $dJ_{reg} / dw = X^T(Y - T) + \lambda w$

Probabilistic Perspective

- Cross-entropy maximizes the likelihood, since J = -log(likelihood)
- Let's make J something we want to maximize:
- $J = +[t \log y + (1 - t) \log(1 - y)] - (\lambda/2)||w||^2$
- $\exp(J) = y^t(1- y)^{(1 - t)} \exp(-\lambda||w||^2 / 2)$
- Exponentiate the first part: Bernoulli (the likelihood)
- Exponentiate -ve squared term: Gaussian (the prior)
- $t \sim \text{Bernoulli}(y)$
- $w \sim N(0, 1/\lambda)$

Probabilistic Perspective

- We have one probability distribution x another probability distribution
- Your first baby steps toward a Bayesian perspective of ML
- posterior $\propto$ likelihood x prior
- $t \sim \text{Bernoulli}(y) \rightarrow$ (likelihood)
- $w \sim N(0, 1/\lambda) \rightarrow$ (prior)
- Our "prior belief" about w is that it's Gaussian distributed with variance $1/\lambda$

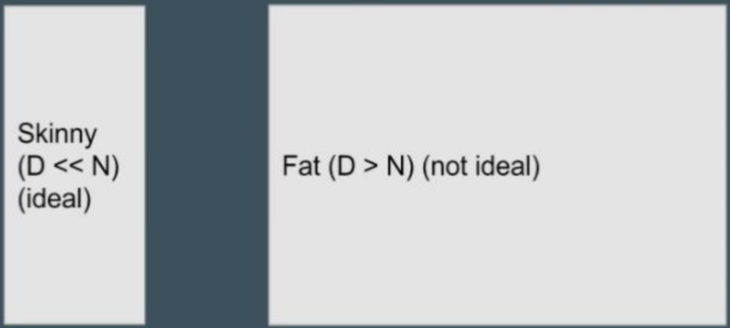
Bayes rule

- P(w|X, Y) ∝ P(X, Y|w) P(w)
- Just a consequence of:
- P(B|A) = P(A|B)P(B) / P(A) = P(A,B) / P(A)
- Bottom part is just Σ_B P(A,B) = Σ_B P(A|B)P(B)

- Without regularization, we maximize the likelihood
- With regularization, we maximize the posterior
- This is called “maximum a posteriori” or MAP estimation

L1 Regularization

- Previously: you saw that even adding a column of completely random noise can improve R^2
- In general, we want D << N (# features << # samples)



L1 Regularization

- L2 regularization used L2 norm for penalty term
- As you might guess, L1 regularization uses L1 norm for penalty term

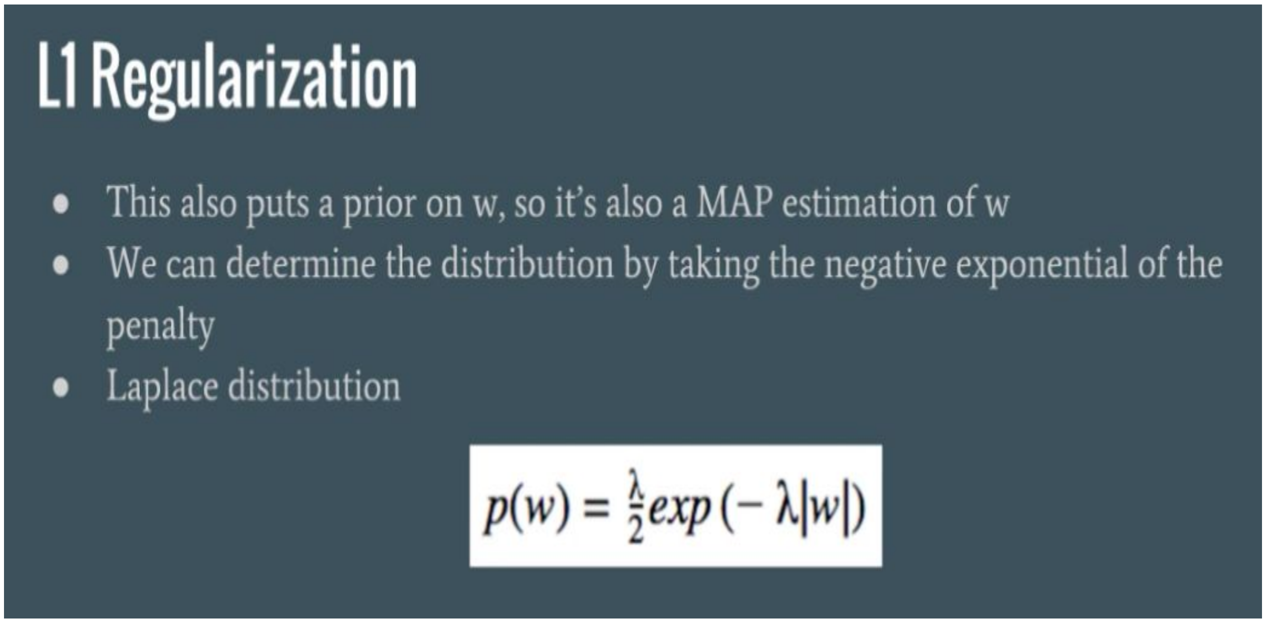
$$J_{RIDGE} = -\sum_{n=1}^{N}(t_n log y_n + (1 - t_n) log(1 - y_n)) + \lambda \|w\|_2^{\,2}$$

$$J_{LASSO} = -\sum_{n=1}^{N}(t_n log y_n + (1 - t_n) log(1 - y_n)) + \lambda \|w\|_1$$

L1 Regularization

- This also puts a prior on w, so it's also a MAP estimation of w
- We can determine the distribution by taking the negative exponential of the penalty
- Laplace distribution

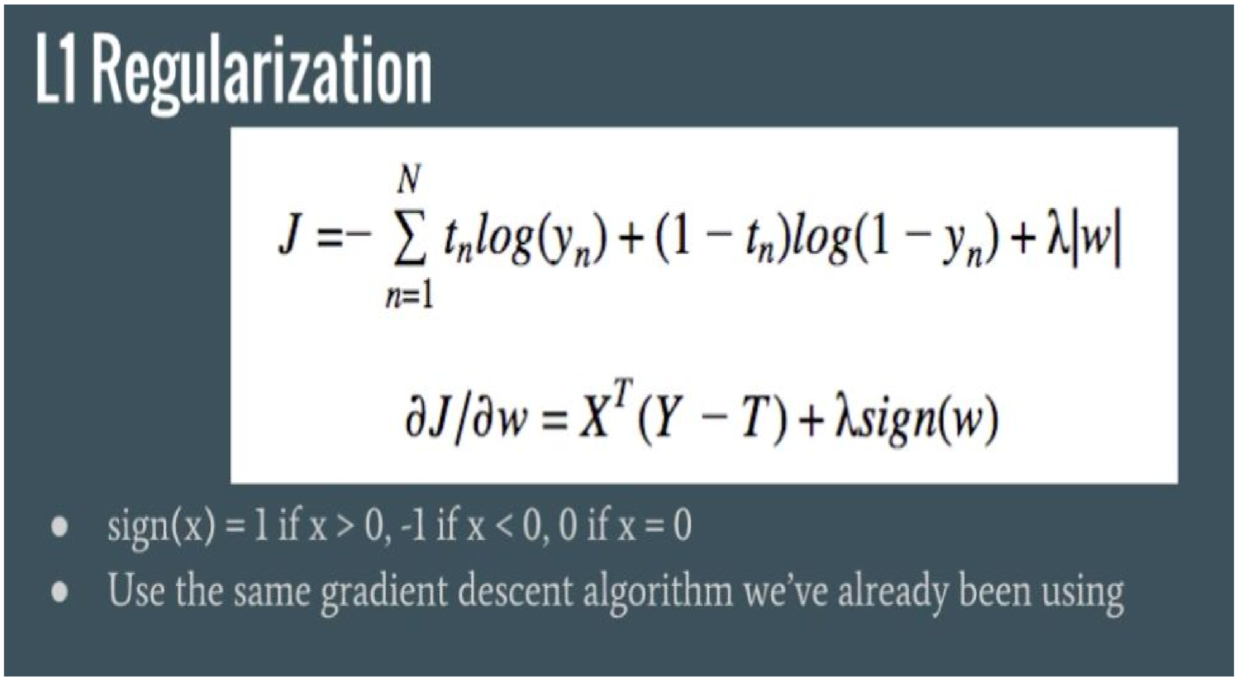
$$p(w) = \frac{\lambda}{2} exp\left(-\lambda|w|\right)$$



L1 Regularization

- This also puts a prior on w, so it's also a MAP estimation of w
- We can determine the distribution by taking the negative exponential of the penalty
- Laplace distribution

$$p(w) = \frac{\lambda}{2} exp\left(-\lambda|w|\right)$$

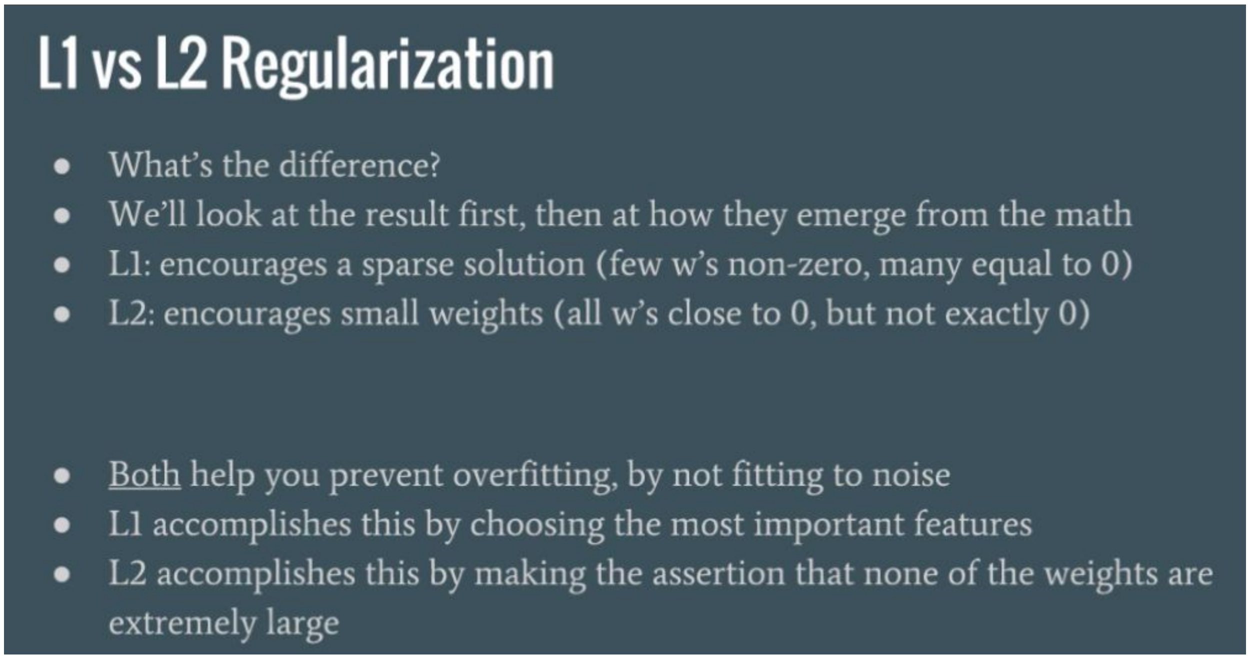


L1 Regularization

$$J = -\sum_{n=1}^{N} t_n log(y_n) + (1 - t_n) log(1 - y_n) + \lambda|w|$$

$$\partial J / \partial w = X^T (Y - T) + \lambda sign(w)$$

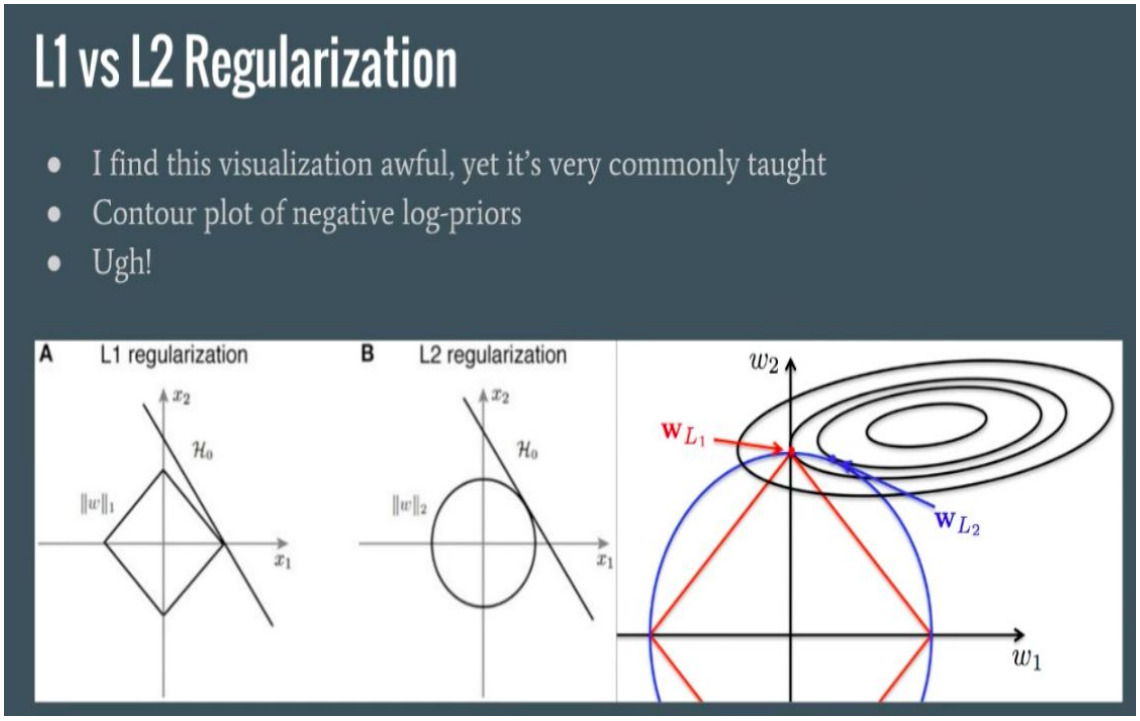
- sign(x) = 1 if x > 0, -1 if x < 0, 0 if x = 0
- Use the same gradient descent algorithm we've already been using



L1 vs L2 Regularization

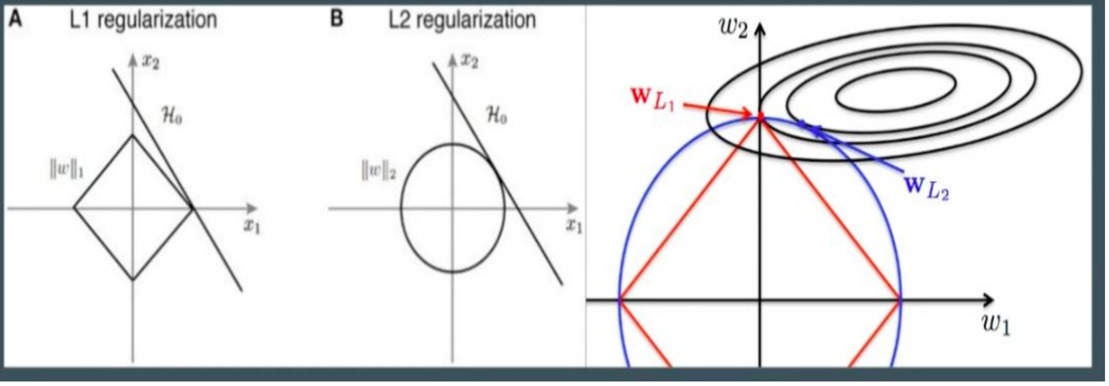
- What's the difference?
- We'll look at the result first, then at how they emerge from the math
- L1: encourages a sparse solution (few w's non-zero, many equal to 0)
- L2: encourages small weights (all w's close to 0, but not exactly 0)

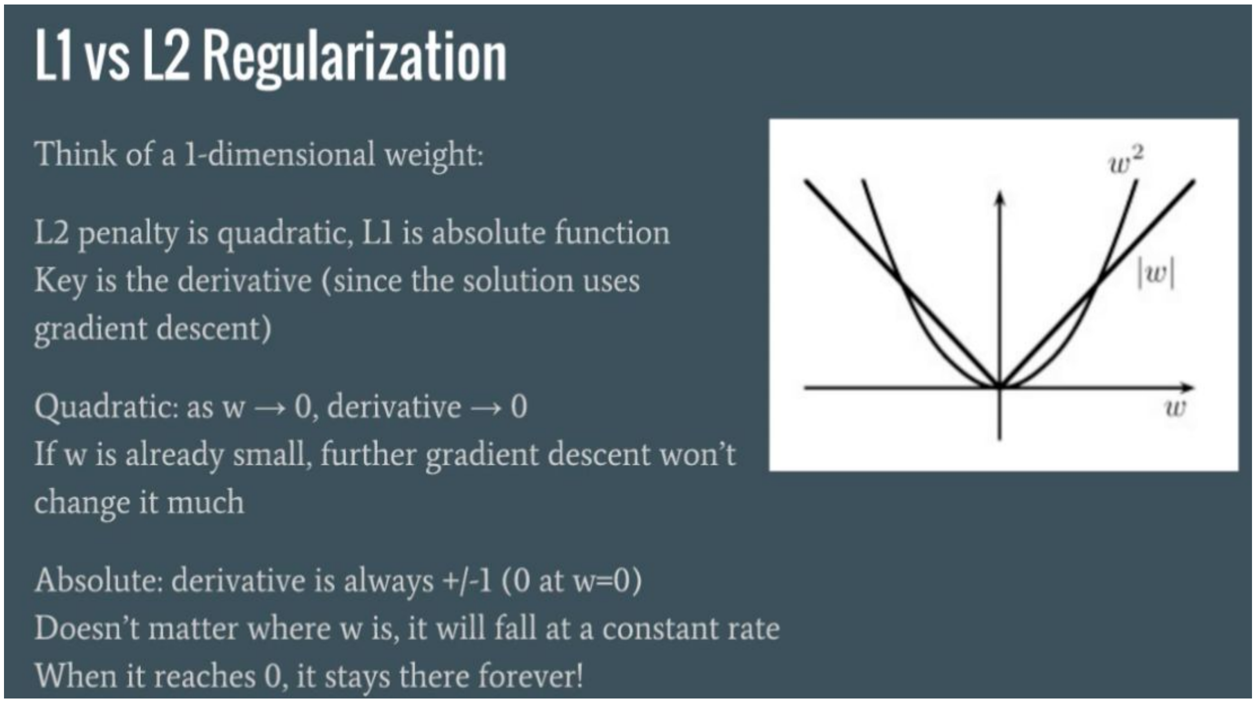
- Both help you prevent overfitting, by not fitting to noise
- L1 accomplishes this by choosing the most important features
- L2 accomplishes this by making the assertion that none of the weights are extremely large



L1 vs L2 Regularization

- I find this visualization awful, yet it's very commonly taught
- Contour plot of negative log-priors
- Ugh!





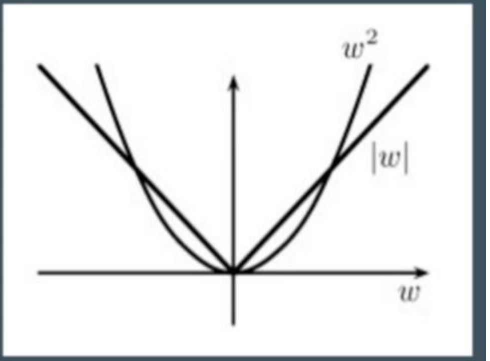
L1 vs L2 Regularization

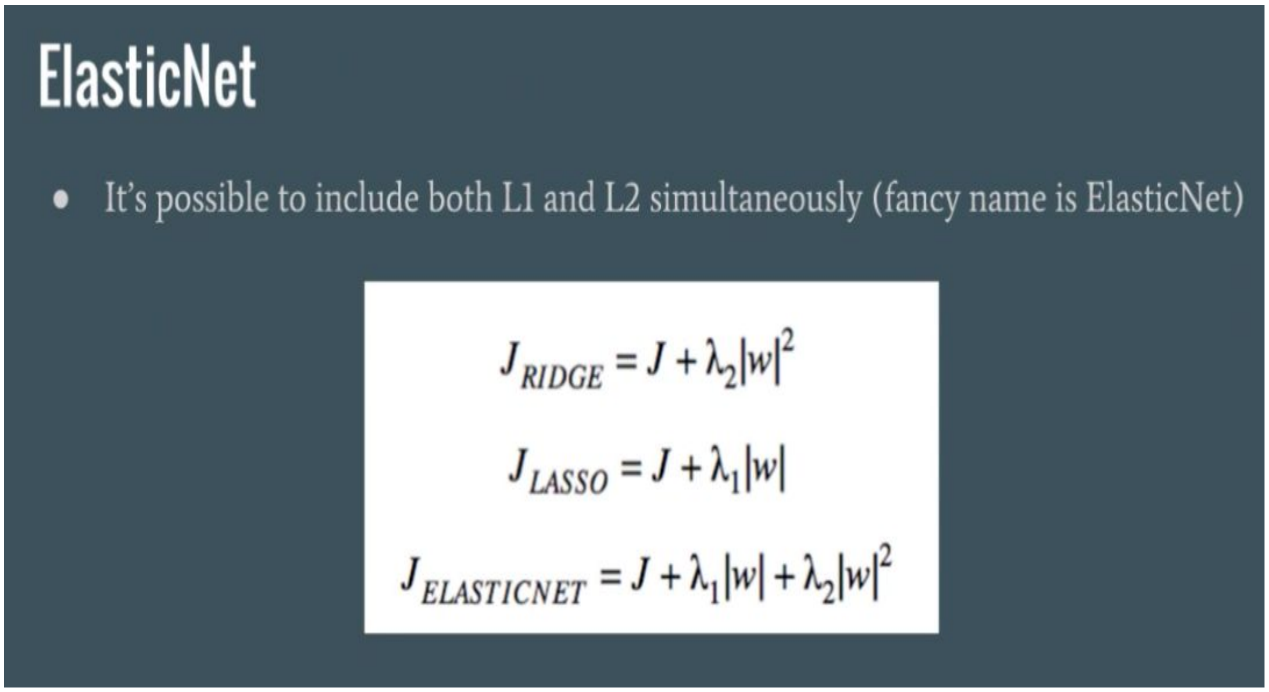
Think of a 1-dimensional weight:

L2 penalty is quadratic, L1 is absolute function
Key is the derivative (since the solution uses gradient descent)

Quadratic: as w → 0, derivative → 0
If w is already small, further gradient descent won't change it much

Absolute: derivative is always +/-1 (0 at w=0)
Doesn't matter where w is, it will fall at a constant rate
When it reaches 0, it stays there forever!





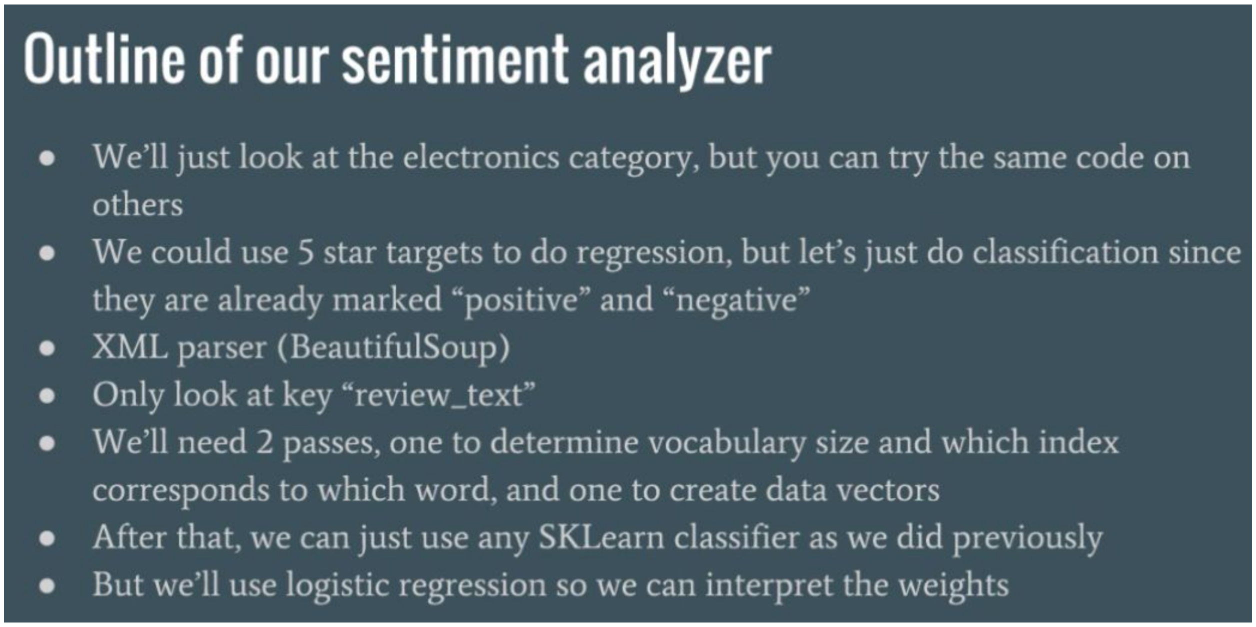
ElasticNet

- It's possible to include both L1 and L2 simultaneously (fancy name is ElasticNet)

$$J_{RIDGE} = J + \lambda_2 |w|^2$$

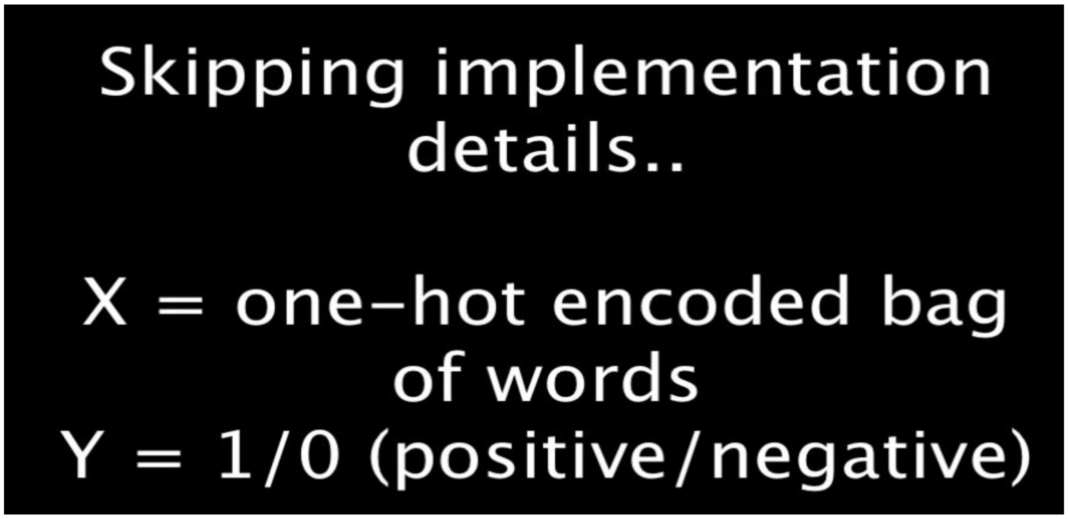
$$J_{LASSO} = J + \lambda_1 |w|$$

$$J_{ELASTICNET} = J + \lambda_1 |w| + \lambda_2 |w|^2$$



Outline of our sentiment analyzer

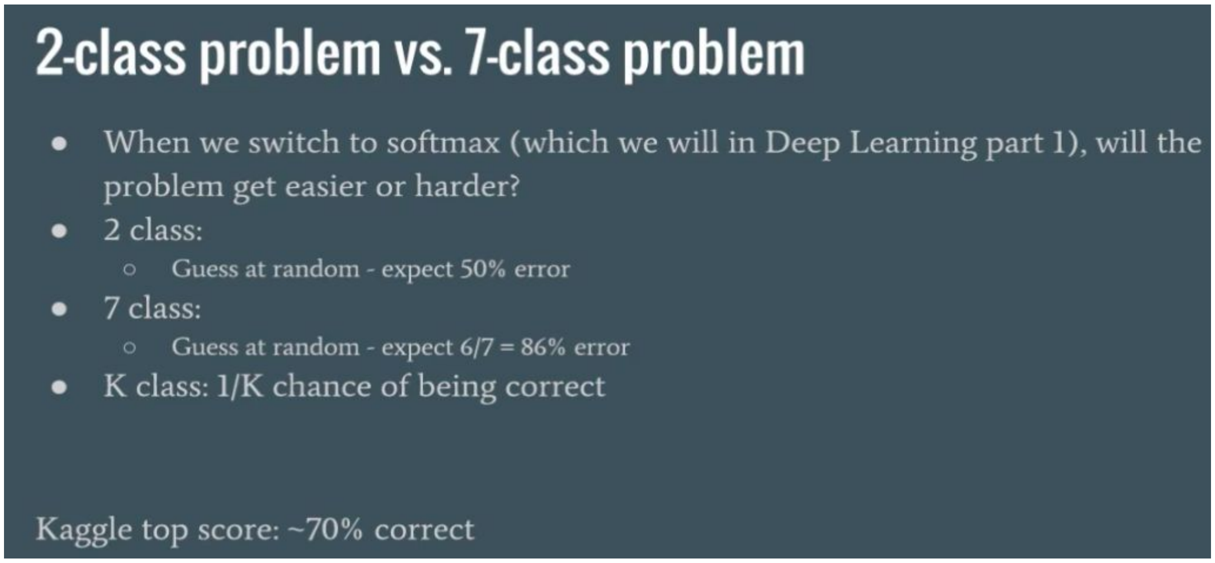
- We'll just look at the electronics category, but you can try the same code on others
- We could use 5 star targets to do regression, but let's just do classification since they are already marked "positive" and "negative"
- XML parser (BeautifulSoup)
- Only look at key "review_text"
- We'll need 2 passes, one to determine vocabulary size and which index corresponds to which word, and one to create data vectors
- After that, we can just use any SKLearn classifier as we did previously
- But we'll use logistic regression so we can interpret the weights



Skipping implementation details..

X = one-hot encoded bag of words

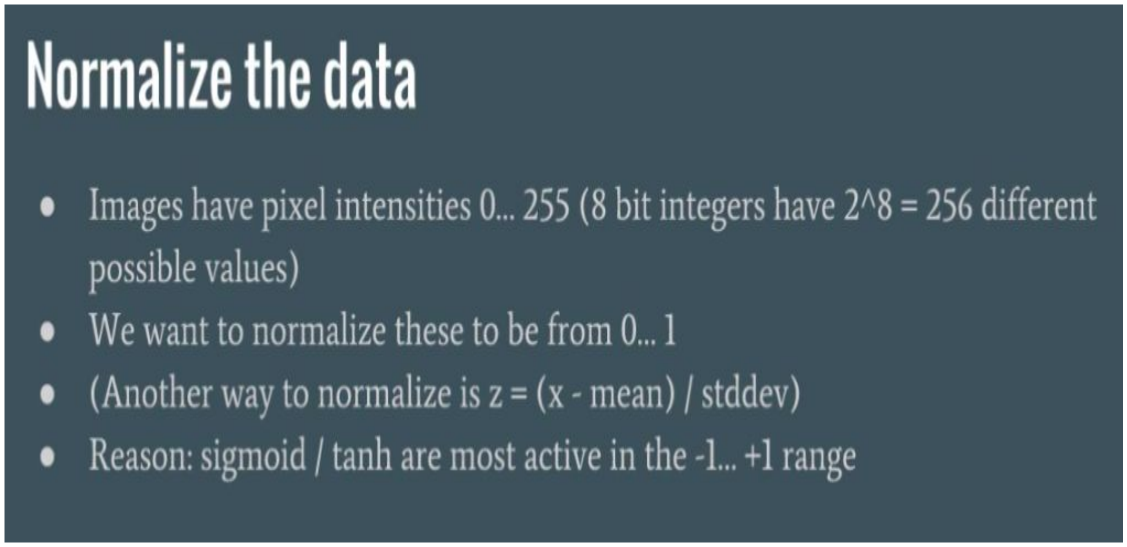
Y = 1/0 (positive/negative)



2-class problem vs. 7-class problem

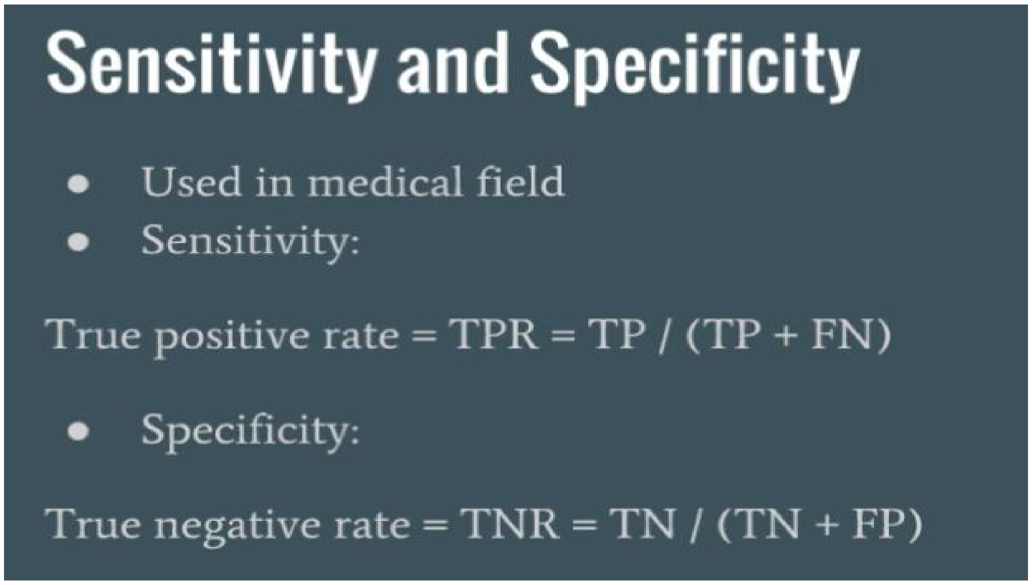
- When we switch to softmax (which we will in Deep Learning part 1), will the problem get easier or harder?
- 2 class:
    - Guess at random - expect 50% error
- 7 class:
    - Guess at random - expect 6/7 = 86% error
- K class: 1/K chance of being correct

Kaggle top score: ~70% correct



Normalize the data

- Images have pixel intensities 0... 255 (8 bit integers have 2^8 = 256 different possible values)
- We want to normalize these to be from 0... 1
- (Another way to normalize is z = (x - mean) / stddev)
- Reason: sigmoid / tanh are most active in the -1... +1 range



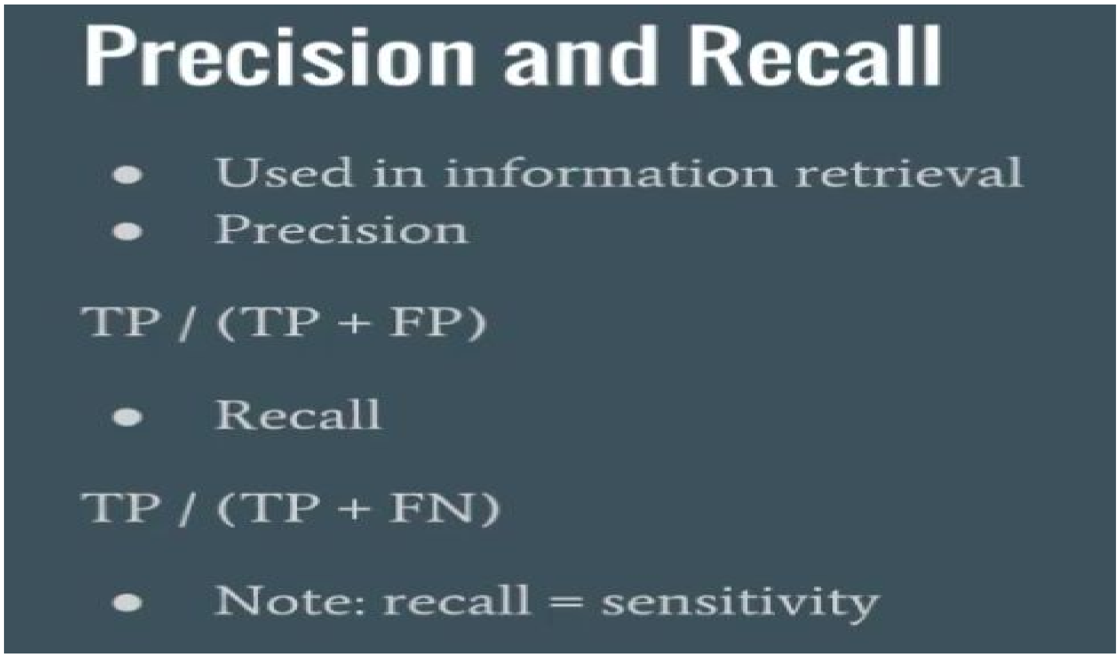
Sensitivity and Specificity

- Used in medical field
- Sensitivity:

True positive rate = TPR = TP / (TP + FN)

- Specificity:

True negative rate = TNR = TN / (TN + FP)



Precision and Recall

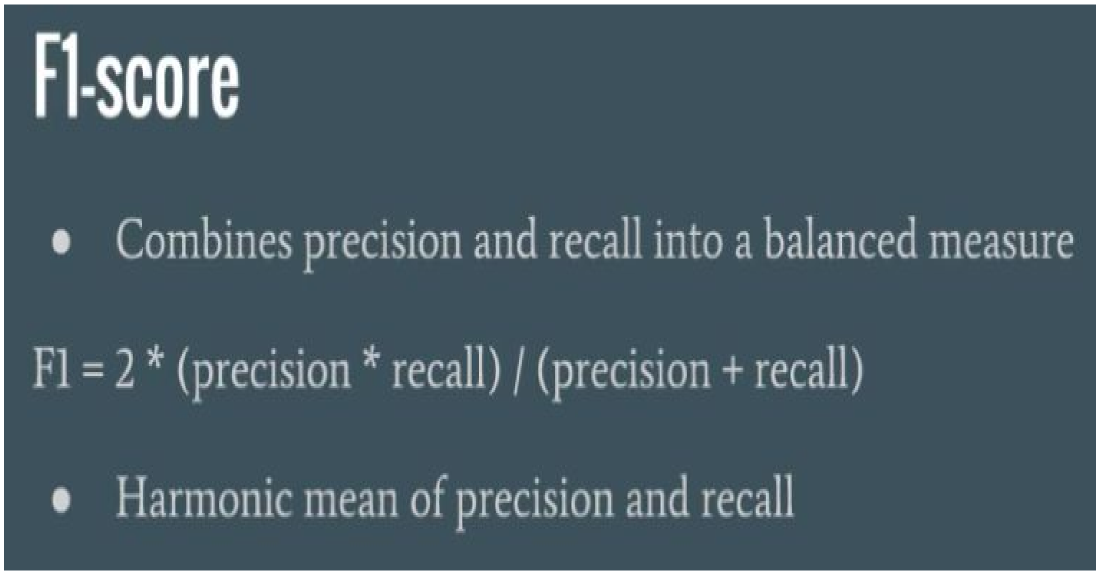
- Used in information retrieval
- Precision

TP / (TP + FP)

- Recall

TP / (TP + FN)

- Note: recall = sensitivity

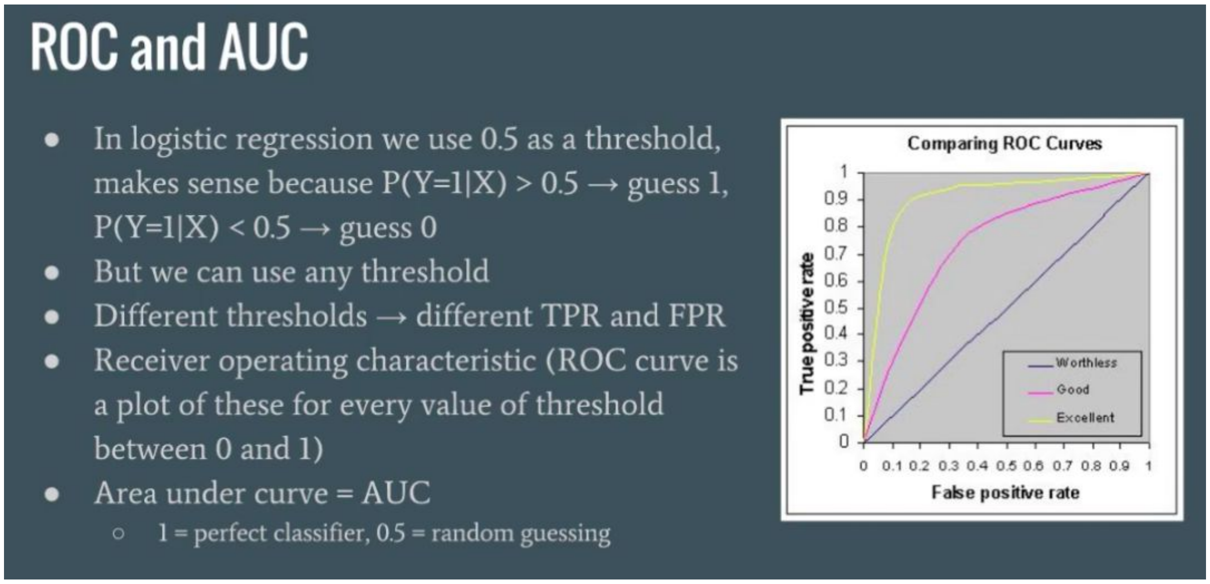


F1-score

- Combines precision and recall into a balanced measure

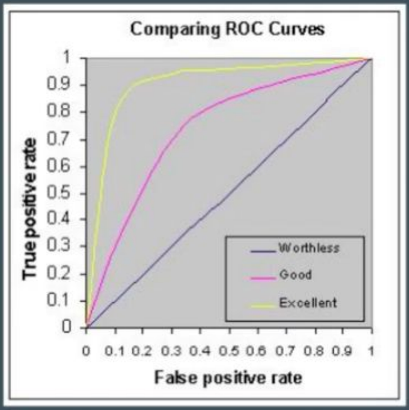
F1 = 2 * (precision * recall) / (precision + recall)

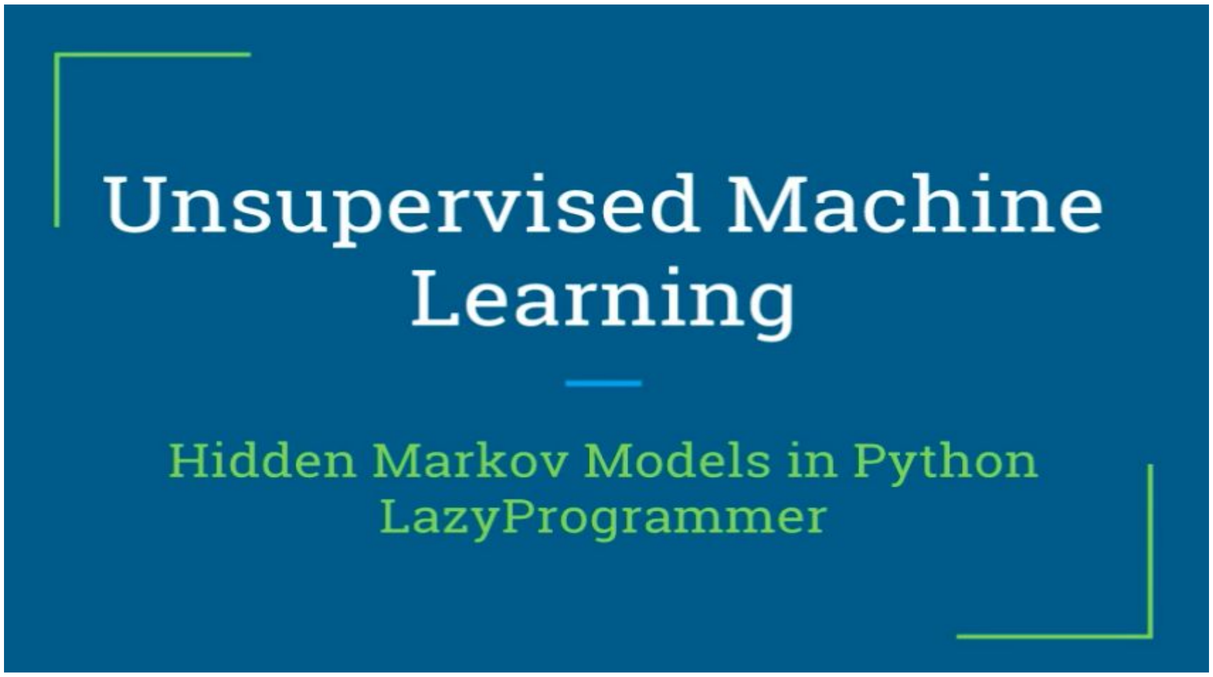
- Harmonic mean of precision and recall



ROC and AUC

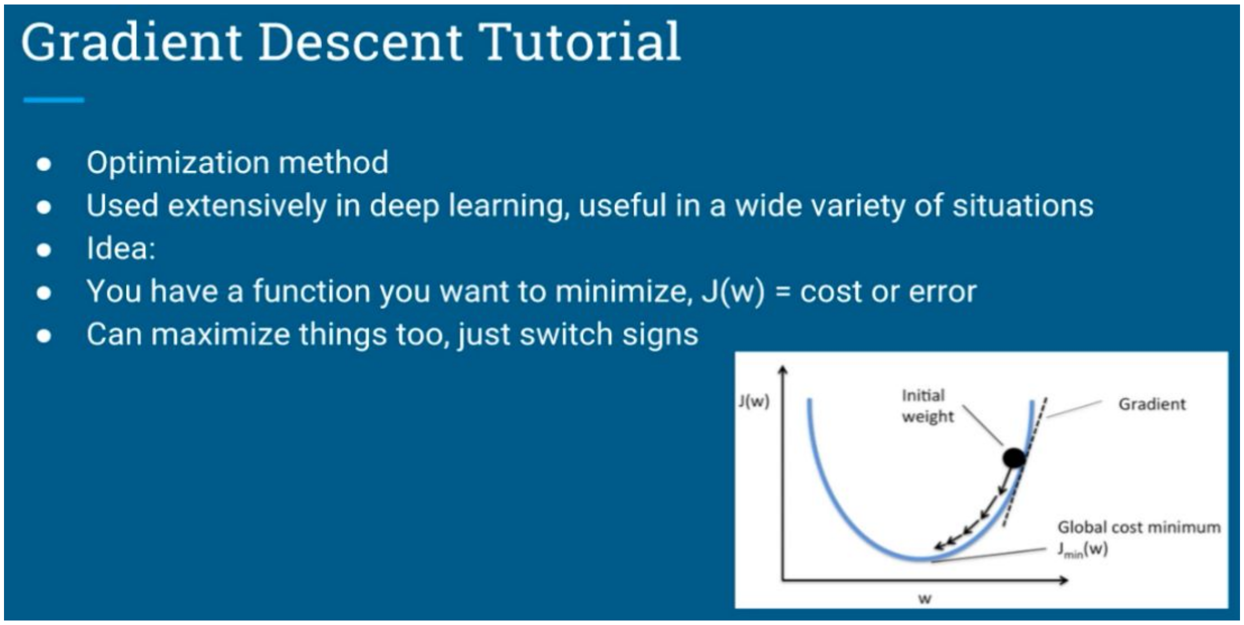
- In logistic regression we use 0.5 as a threshold, makes sense because $P(Y=1|X) > 0.5 \rightarrow$ guess 1, $P(Y=1|X) < 0.5 \rightarrow$ guess 0
- But we can use any threshold
- Different thresholds → different TPR and FPR
- Receiver operating characteristic (ROC curve is a plot of these for every value of threshold between 0 and 1)
- Area under curve = AUC
  - 1 = perfect classifier, 0.5 = random guessing





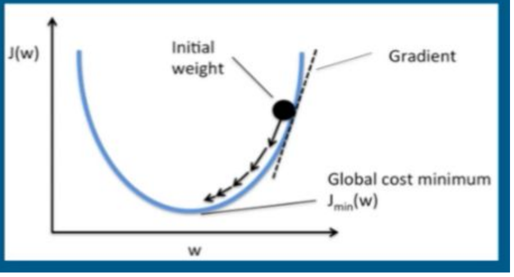
Unsupervised Machine Learning

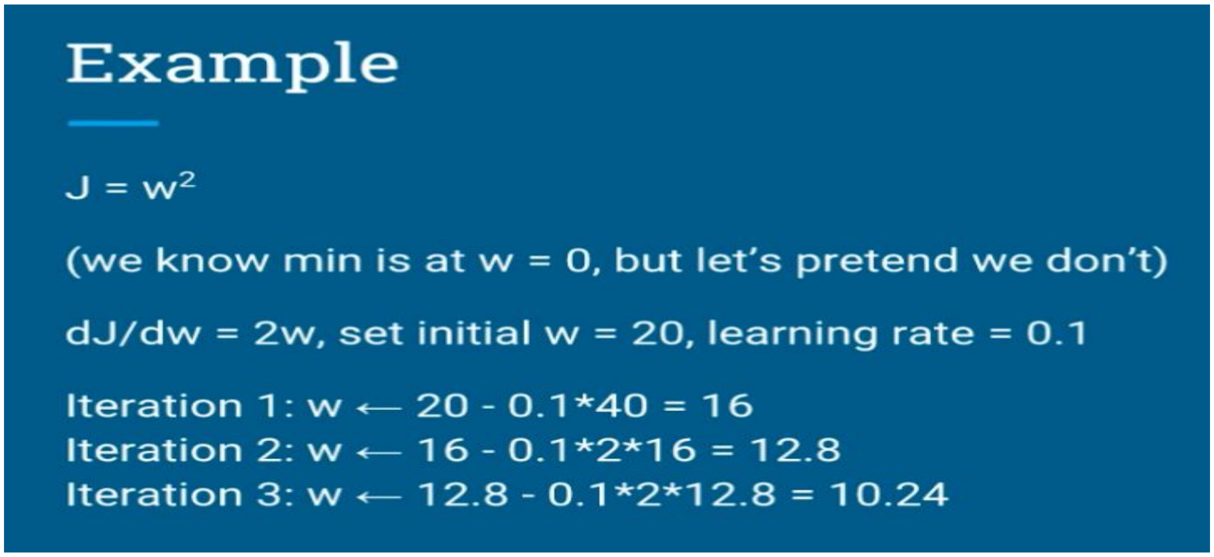
Hidden Markov Models in Python
LazyProgrammer



Gradient Descent Tutorial

- Optimization method
- Used extensively in deep learning, useful in a wide variety of situations
- Idea:
- You have a function you want to minimize, J(w) = cost or error
- Can maximize things too, just switch signs





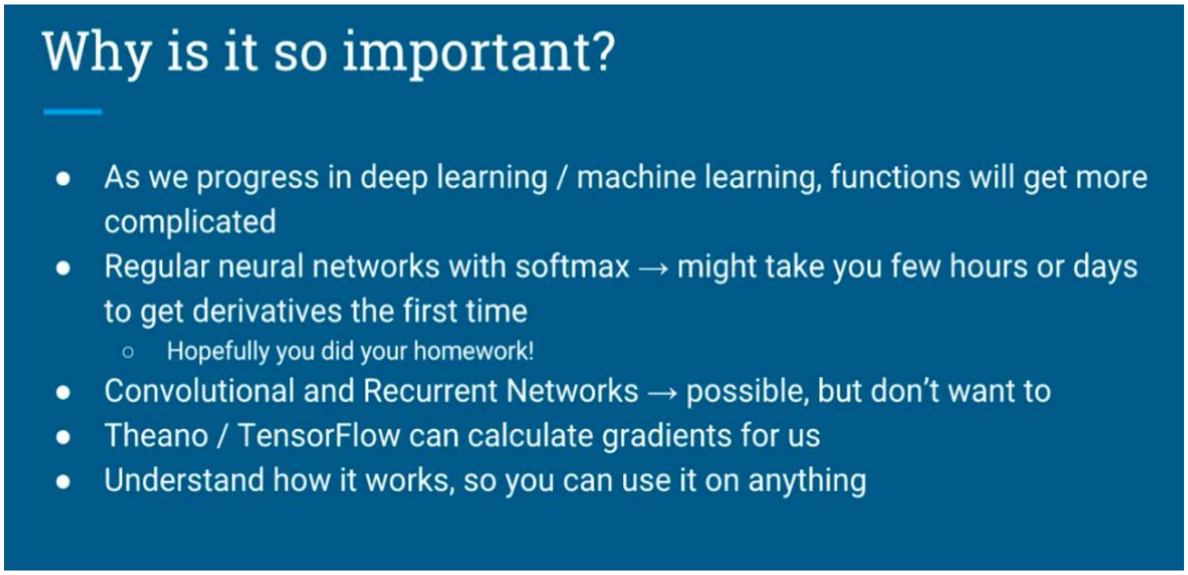
Example

$J = w^2$

(we know min is at w = 0, but let's pretend we don't)

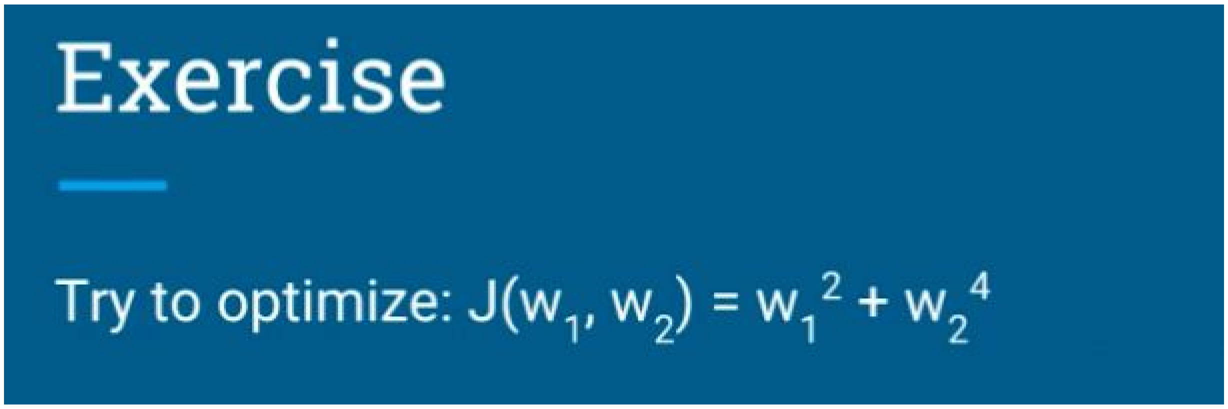
dJ/dw = 2w, set initial w = 20, learning rate = 0.1

Iteration 1: $w \longleftarrow 20 - 0.1*40 = 16$
Iteration 2: $w \longleftarrow 16 - 0.1*2*16 = 12.8$
Iteration 3: $w \longleftarrow 12.8 - 0.1*2*12.8 = 10.24$



Why is it so important?

- As we progress in deep learning / machine learning, functions will get more complicated
- Regular neural networks with softmax → might take you few hours or days to get derivatives the first time
  - Hopefully you did your homework!
- Convolutional and Recurrent Networks → possible, but don't want to
- Theano / TensorFlow can calculate gradients for us
- Understand how it works, so you can use it on anything



Exercise

Try to optimize: $J(w_1, w_2) = w_1^2 + w_2^4$

Maximizing the likelihood is the same as minimizing the cross-entropy function.