Regression

Classification in logistic regression: summary

Given:

- a set of classes: (+ sentiment,- sentiment)
- a vector **x** of features `[x1, x2, …, xn]`
  - x1= count("awesome")
  - x2 = log(number of words in review)
- A vector **w** of weights `[w1, w2, …, wn]`
  - w_i for each feature f_i

1 per class if more than 2 classes

softmax if more than 2 classes

$$P(y=1) = \sigma(w \cdot x + b)$$
$$= \frac{1}{1+\exp(-(w \cdot x + b))}$$

The two phases of logistic regression

Training: we learn weights *w* and *b* using **stochastic gradient descent** and **cross-entropy loss**.

learning algorithm metric

Test: Given a test example *x* we compute $p(y|x)$ using learned weights *w* and *b*, and return whichever label ($y = 1$ or $y = 0$) is higher probability

How Does Learning Work?

Learning in Supervised Classification

Supervised classification:

- We know the correct label y (either 0 or 1) for each x.
- But what the system produces is an estimate, $\hat{y}$

We want to set w and b to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.

- We need a distance estimator: a **loss function** or a **cost function**
- We need an optimization algorithm to update w and b to minimize the loss.

Learning components

A loss function: cross-entropy loss
negative log likelihood loss } same

An optimization algorithm:
stochastic gradient descent

w / θ : used interchangeably to denote
weights (learned parameters)

The distance between $\hat{y}$ and y

$\hat{y}$: our classifier's guess (between 0 & 1)

y : the datapoint's actual label
the class the doc actually belongs to
(either 0 or 1)

$L(\hat{y}, y)$: how far off is our model
loss
from the true label?

Intuition of negative log likelihood loss = cross-entropy loss

A case of conditional maximum likelihood estimation

We choose the parameters *w*,*b* that maximize

- the log probability
- of the true *y* labels in the training data
- given the observations *x*

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize: $p(y|x) = \underbrace{\hat{y}^{y}}_{\text{matters when } y=1} \underbrace{(1-\hat{y})^{1-y}}_{\text{matters when } y=0}$

Since there are only 2 discrete outcomes (0 or 1) we can express the probability $p(y|x)$ from our classifier as:

Note: $y=1$, this simplifies $\hat{y}$

$y=0$, this simplifies to $1-\hat{y}$

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize: $$p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

$$\log p(y|x) = \log\left[\hat{y}^{y}(1-\hat{y})^{1-y}\right]$$

$$= y\log\hat{y} + (1-y)\log(1-\hat{y})$$

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Minimize the cross-entropy loss

Minimize: $L_{CE}(\hat{y}, y) = -\log p(y|x)$

$= -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$

$L_{CE}(\hat{y}, y) = -[y \log \sigma(wx+b) + (1-y)\log(1-\sigma(wx+b))]$

Does this work for our sentiment example?

We want loss to be:

- smaller if the model estimate is close to correct
- bigger if model is confused

Let's first suppose the true label of this is y=1 (positive)

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Let's see if this works for our sentiment example

$w = [2.5, -5, -1.2, 0.5, 2, 0.7]$ $b = 0.1$

$x = [3, 2, 1, 3, 0, 4.19]$

True value is y=1. How well is our model doing?

$$p(+|x) = P(y=1|x) = \sigma(wx+b)$$

$$= \sigma(w \cdot x + 0.1)$$

$$= \sigma(0.833)$$

$$= 0.70$$

Let's see if this works for our sentiment example

True value is y=1. How well is our model doing?

$$p(+|x) = P(y=1|x) \;=\; 0.7$$

Pretty well! What's the loss?

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \\ &= -[y \log(0.7) + (1-y) \log(1-0.7)] \\ &= -[1 \cdot \log(0.7) + \cancel{(1-1) \log(1-0.7)}] \\ &= -\log(0.7) = 0.36 \end{aligned}$$

What if the true label was 0?

$$p(+|x) = P(y = 1|x) \quad =$$

What if the true label was 0?

$$p(+|x) = P(y=1|x) = 0.7$$

$$p(-|x) = 1 - p(+|x) = 0.3$$

$$L_{CE}(\hat{y}, y) = -[y \log \sigma(wx+b) + (1-y)\log(1-\sigma(wx+b))]$$

$$= -[0 \cdot \log \hat{y} + (1-0)\log(1-0.7)]$$

$$= -\log(0.3)$$

$$= 1.3$$

The loss when model was right (if true y=1)

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{y},y) &= -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\
&= -[\log\sigma(w\cdot x+b)] \\
&= -\log(.70) \\
&= .36
\end{aligned}
$$

Is lower than the loss when model was wrong (if true y=0):

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{y},y) &= -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\
&= -[\log(1-\sigma(w\cdot x+b))] \\
&= -\log(.30) \\
&= 1.2
\end{aligned}
$$

Stochastic Gradient Descent

Slides borrowed from Jurafsky & Martin Edition 3

Our goal: minimize the loss

Let's make explicit that the loss function is parameterized by weights θ=(w,b)

We'll represent $\hat{y}$ as $f\,(x;\,\theta\,)$ to make the dependence on θ more obvious

input

θ = Model's parameters

We want the weights that minimize the loss, averaged over all examples:

model's guess

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}\left(f(x^{(i)};\theta), y^{(i)}\right)$$

input

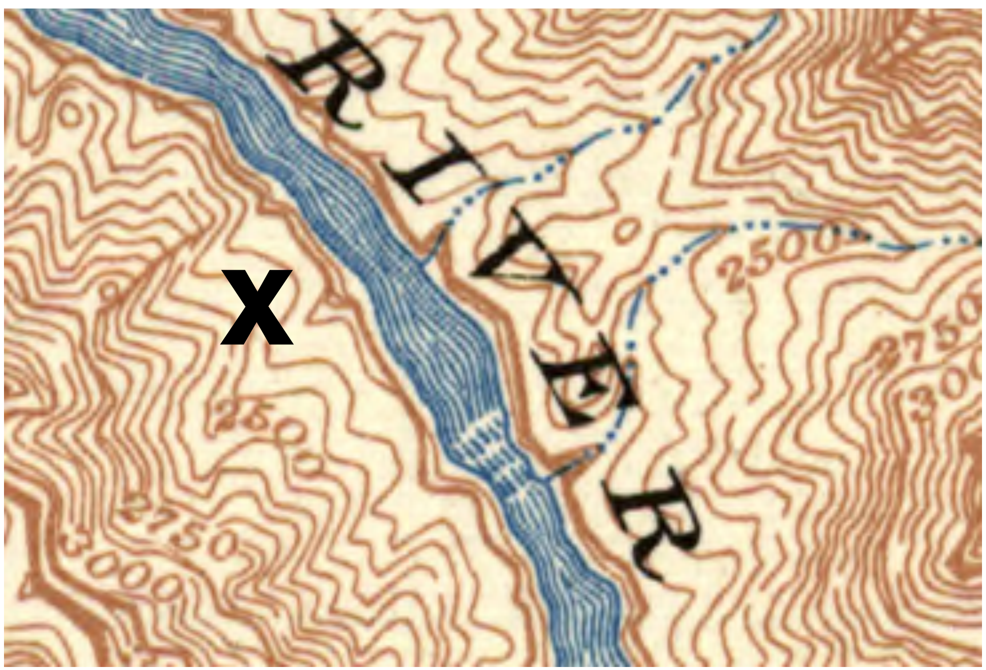
current weights

actual label

m = length of dataset

Intuition of gradient descent

How do I get to the bottom of this river canyon?



Look around me 360°

Find the direction of steepest slope down

Go that way

Our goal: minimize the loss

For logistic regression, loss function is **convex**

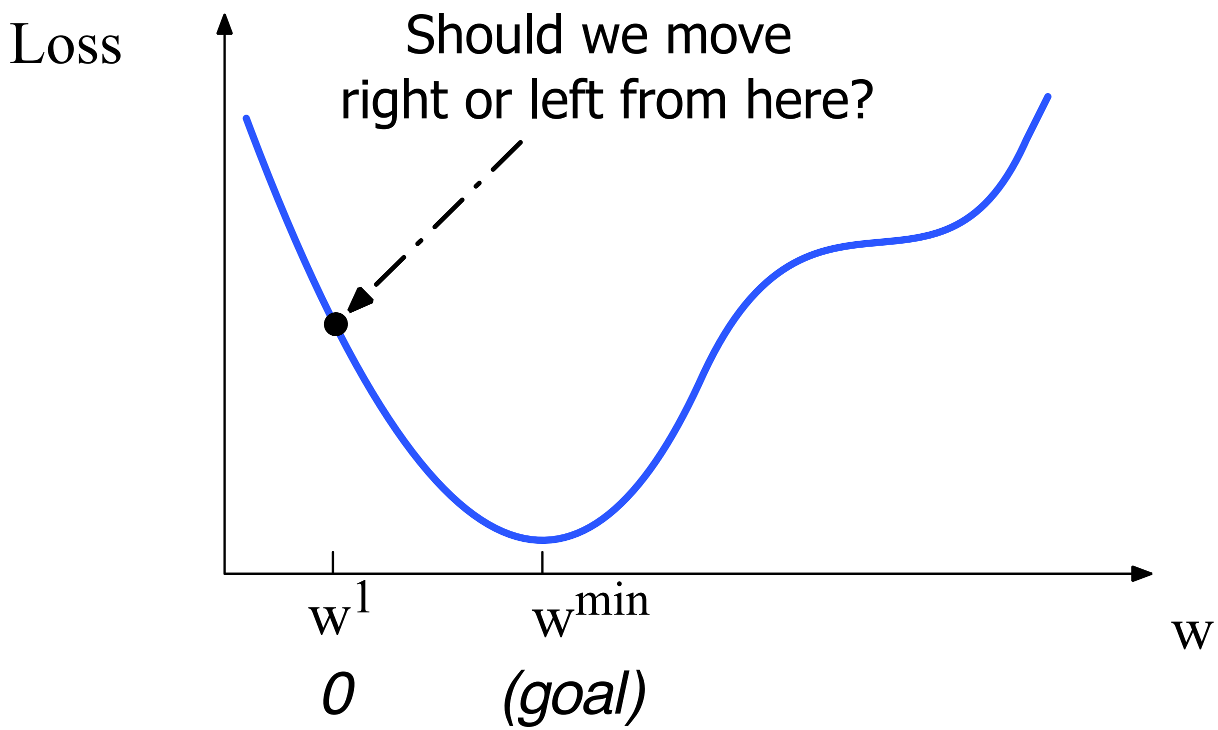
- A convex function has just one minimum
- Gradient descent starting from any point is guaranteed to find the minimum
  - (Loss for neural networks is non-convex)

Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function

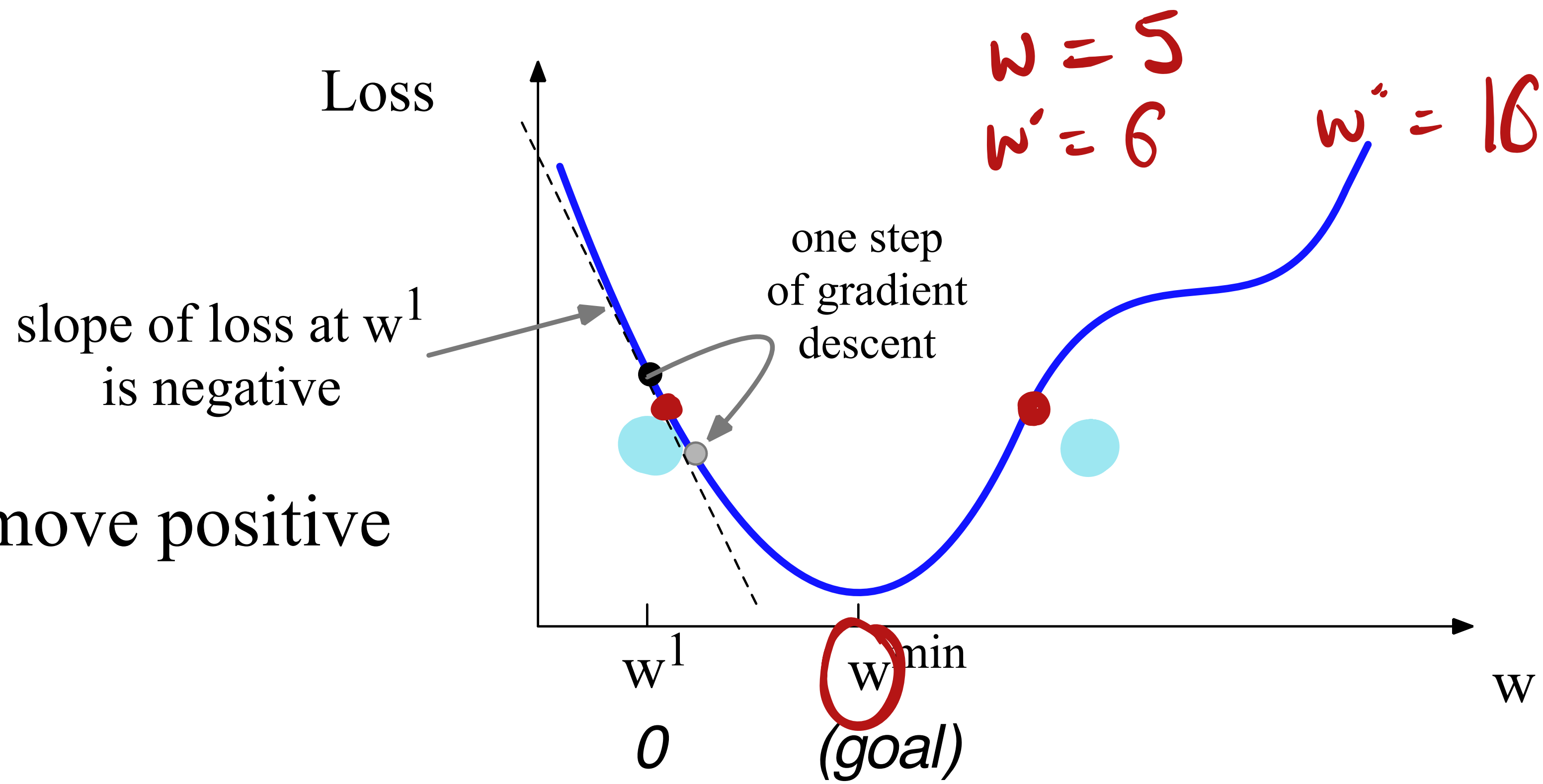
w = cont (positive)



Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function



So we'll move positive

Gradients

The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

Gradient Descent: Find the gradient of the loss function at the current point and move in the **opposite** direction.

How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η hyperparameter
- Higher learning rate means that we make bigger adjustments to the weights

$$w^{t+1} = w^{t} - \eta \frac{d}{dw}L(f(x;w),y)$$

if only we knew this!

particular weight

Now let's consider N dimensions

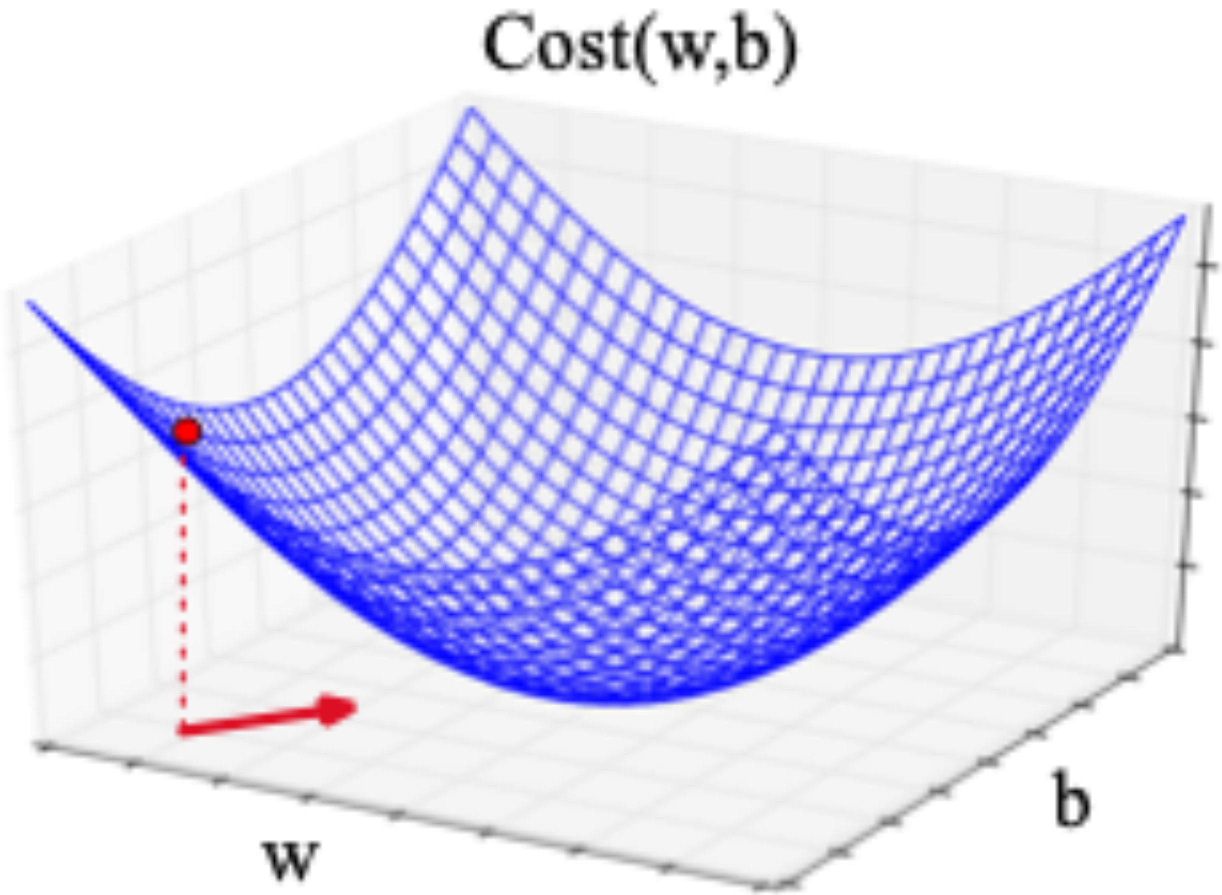
We want to know where in the N-dimensional space (of the *N* parameters that make up θ) we should move.

The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the *N* dimensions.

Imagine 2 dimensions, w and b

Visualizing the gradient vector at the red point

It has two dimensions shown in the x-y plane



function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

    # where: L is the loss function

    #   f is a function parameterized by θ

    #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$

$\theta \leftarrow 0$

repeat til done   # see caption

  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

    1. Optional (for reporting):   # How are we doing on this tuple?

prediction   Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?

      Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?

    2. $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?

    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead

return θ

Hyperparameters

The learning rate η is a **hyperparameter**

- too high: the learner will take big steps and overshoot
- too low: the learner will take too long

Hyperparameters:

- Briefly, a special kind of parameter for an ML model
- Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.

How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η
- Higher learning rate means that we make bigger adjustments to the weights

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w),y)$$

Partial Derivative for Logistic Regression

Cross-Entropy Loss $L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$

Chain Rule: $\frac{\partial f}{\partial x} = \frac{du}{dv} \cdot \frac{dv}{dx}$

Weight Update $w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$

$f(x) = u(v(x))$

Derivative of Loss

$$\frac{\partial L(f(x;\theta), y)}{\partial w} = \frac{\partial - [y \log \sigma(wx+b) + (1-y)\log(1-\sigma(wx+b))]}{\partial w}$$

$$= \frac{\partial y \log \sigma(wx+b)}{\partial w} + \frac{\partial (1-y)(\log(1-\sigma(wx+b)))}{\partial w}$$

$$= -\frac{y}{\sigma(wx+b)} \cdot \frac{\partial \sigma(wx+b)}{\partial w} - \frac{1-y}{1-\sigma(wx+b)} \cdot \frac{\partial (1-\sigma(wx+b))}{\partial w}$$

Derivative of $\sigma(x)$: $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))$

Derivative of $\ln(x)$: $\frac{\partial \ln(x)}{\partial x} = \frac{1}{x}$

$$= [\sigma(wx+b) - y]x$$

Partial Derivative for Logistic Regression

Cross-Entropy Loss

$$L_{\mathrm{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

Weight Update

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

Derivative of Cross-Entropy Loss

$$\frac{d}{dw} L(f(x; w), y) = [\overbrace{\sigma(w \cdot x + b)}^{\hat{y}} - y] x_j$$

Deriving cross-entropy loss for multi-label classification

Goal: maximize probability of the correct label $p(y|x)$

$$L_{\mathrm{CE}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} \mathbf{y}_k \log \hat{\mathbf{y}}_k$$

Overfitting

A model that perfectly match the training data has a problem.

It will also **overfit** to the data, modeling noise

- A random word that perfectly predicts *y* (it happens to only occur in one class) will get a very high weight.
- Failing to generalize to a test set without this word.

A good model should be able to **generalize**

Overfitting

Useful or harmless features

This movie drew me in, and it'll do the same to you.

+

X1 = "this"
X2 = "movie
X3 = "hated"
X4 = "drew me in"

I can't tell you how much I hated this movie. It sucked.

-

X5 = "the same to you"
X7 = "tell you how much"

"Memorizing" the training data can cause problems

Overfitting

4-gram model on tiny data will just memorize the data

- 100% accuracy on the training set

But it will be surprised by the novel 4-grams in the test data

- Low accuracy on test set

Models that are too powerful can **overfit** the data

- Fitting the details of the training data so exactly that the model doesn't generalize well to the test set
- How to avoid overfitting?
  - Regularization in logistic regression
  - Dropout in neural networks

Game

www.i-am.ai/gradient-descent.html