Logistic Regression

Background: Generative and Discriminative Classifiers

Logistic Regression

Important analytic tool in natural and social sciences

Baseline supervised machine learning tool for classification

Is also the foundation of neural networks

Generative and Discriminative Classifiers

Naïve Bayes is a **generative** classifier

by contrast:

Logistic regression is a **discriminative** classifier

Generative and Discriminative Classifiers

Suppose we're distinguishing cat from dog images

imagenet

imagenet

Generative Classifier:

- Build a model of what's in a cat image
  - Knows about whiskers, ears, eyes
  - Assigns a probability to any image:
    - how cat-y is this image?

Also build a model for dog images

Now given a new image:

Run both models and see which one fits better

Discriminative Classifier

Just try to distinguish dogs from cats

Oh look, dogs have collars!

Let's ignore everything else

Finding the correct class c from a document d in Generative vs Discriminative Classifiers

Naive Bayes

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \frac{\overbrace{P(d|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}}}{P(d)}$$

Logistic Regression

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \overset{\text{posterior}}{P(c|d)}$$

Components of a probabilistic machine learning classifier

Given *m* input/output pairs $(x^{(i)}, y^{(i)})$:

1. A **feature representation** of the input. For each input observation $x^{(j)}$, a vector of features $[x_1, x_2, \ldots, x_n]$. Feature i for input $x^{(j)}$ is x_i, more completely $x_i^{(j)}$, or sometimes $f_i(x)$.
2. A **classification function** that computes $\hat{y}$, the estimated class, via $p(y|x)$, like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

The two phases of Logistic Regression

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

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

$$P(y = 1|x) + P(y = 0|x) = 1.0$$
$$P(y = 1|x) > P(y = 0|x)$$

<=> $P(y=1|x) > 0.5$

Logistic Regression

Background: Generative and Discriminative Classifiers

Logistic Regression

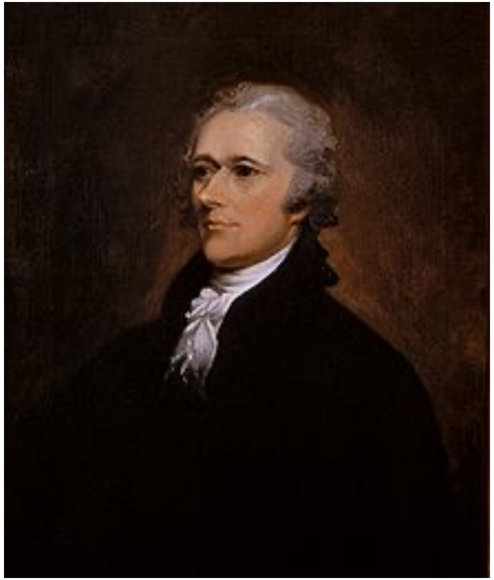
Classification with Logistic Regression

Classification Reminder

Positive/negative sentiment

Spam/not spam

Authorship attribution (Hamilton or Madison?)



Alexander Hamilton

Text Classification: definition

Input:

- a document x
- a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

Output: a predicted class $\hat{y} \in C$

Binary Classification in Logistic Regression

Given a series of input/output pairs:

- $(x^{(i)}, y^{(i)})$

For each observation $x^{(i)}$

- We represent $x^{(i)}$ by a **feature vector** $[x_1, x_2, \ldots, x_n]$
- We compute an output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

Features in logistic regression

- For feature x_i, weight w_i tells how important is x_i
  - x_i ="review contains 'awesome'": $w_i = +10$
  - x_j ="review contains 'abysmal'": $w_j = -10$
  - x_k ="review contains 'mediocre'": $w_k = -2$

Logistic Regression for one observation x

Input observation: vector $x = [x_1, x_2, \ldots, x_n]$

Weights: one per feature: $\mathbf{w} = [w_1, w_2, \ldots, w_n]$

- Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$

Output: a predicted class $\hat{y} \in \{0,1\}$

(multinomial logistic regression: $y \in \{0, 1, 2, 3, 4\}$)

How to do classification

For each feature x_i, weight w_i tells us importance of x_i

- (Plus we'll have a bias b)

We'll sum up all the weighted features and the bias

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

$$z = w \cdot x + b$$

If this sum is high, we say y=1; if low, then y=0

But we want a probabilistic classifier

We need to formalize “sum is high”.

We’d like a principled classifier that gives us a probability, just like Naive Bayes did

We want a model that can tell us:

$p(y=1|x; \theta)$

$p(y=0|x; \theta)$

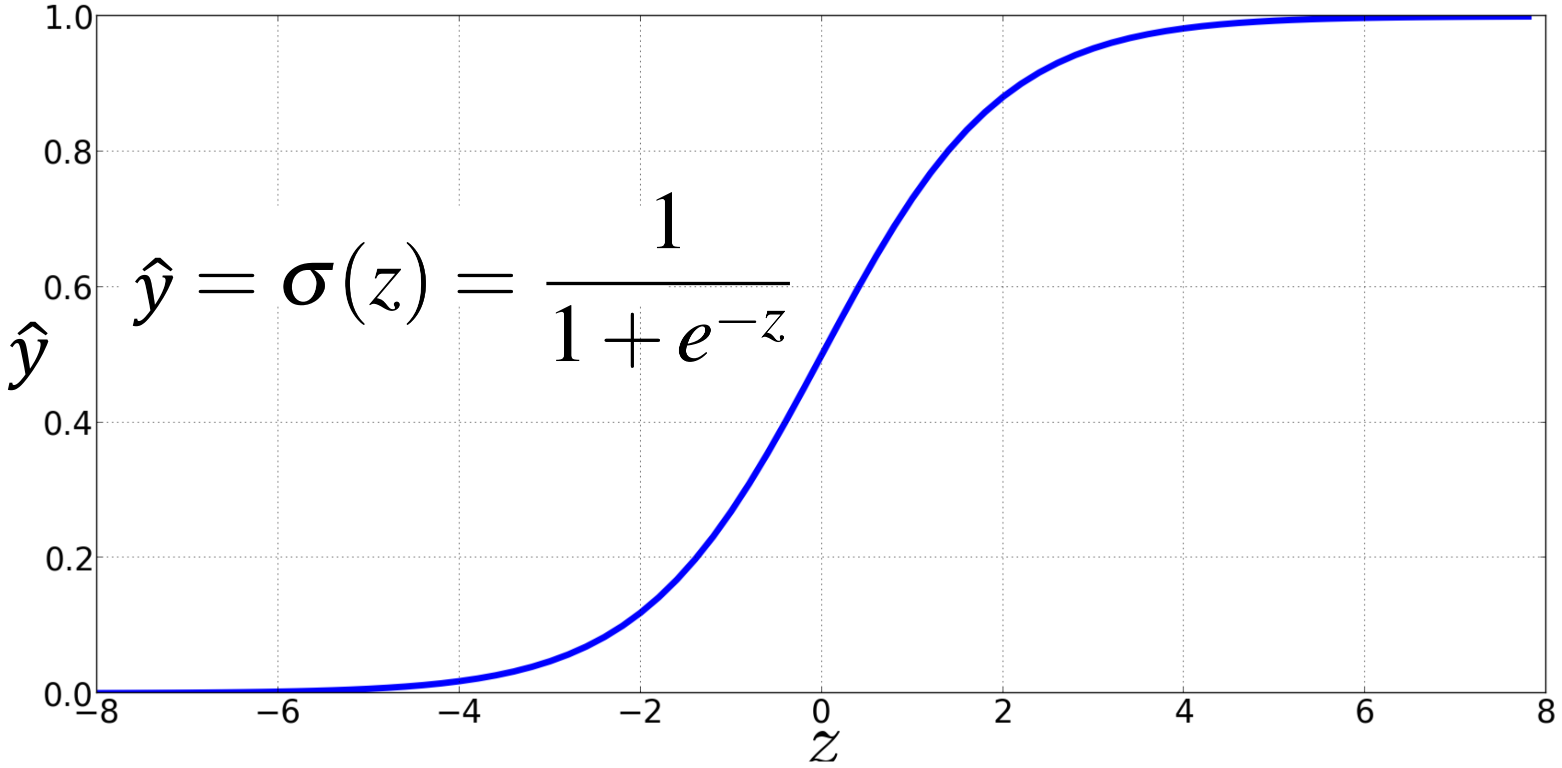
The problem: z isn't a probability, it's just a number!

$$z = w \cdot x + b$$

Solution: use a function of z that goes from 0 to 1

$$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$$

The very useful sigmoid or logistic function



Idea of logistic regression

We'll compute $w \cdot x + b$

And then we'll pass it through the sigmoid function:

$$\sigma(w \cdot x + b)$$

And we'll just treat it as a probability

Making probabilities with sigmoids

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

$$
\begin{aligned}
P(y=0) &= 1-\sigma(w\cdot x+b)\\
&= 1-\frac{1}{1+\exp(-(w\cdot x+b))}\\
&= \frac{\exp(-(w\cdot x+b))}{1+\exp(-(w\cdot x+b))} = \sigma(-(w\cdot x+b))
\end{aligned}
$$

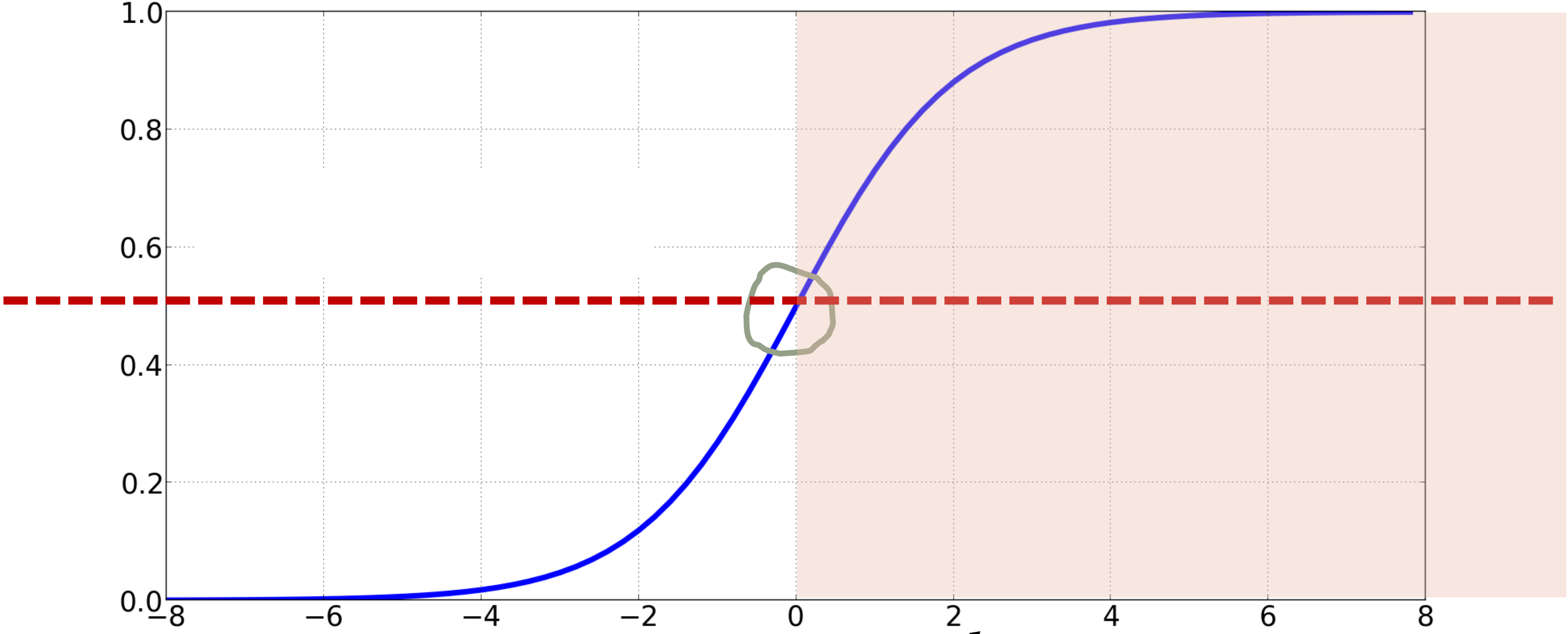
Turning a probability into a classifier

$$\text{Prediction} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \quad \text{if } w\cdot x+b > 0 \\ 0 & \text{otherwise} \quad \text{if } w\cdot x+b \leq 0 \end{cases}$$

0.5 here is called the **decision threshold**

The LR classifier

$$\hat{y} = P(y=1) \;=\; \sigma(w \cdot x + b)$$



$$z = w \cdot x + b$$

Logistic Regression

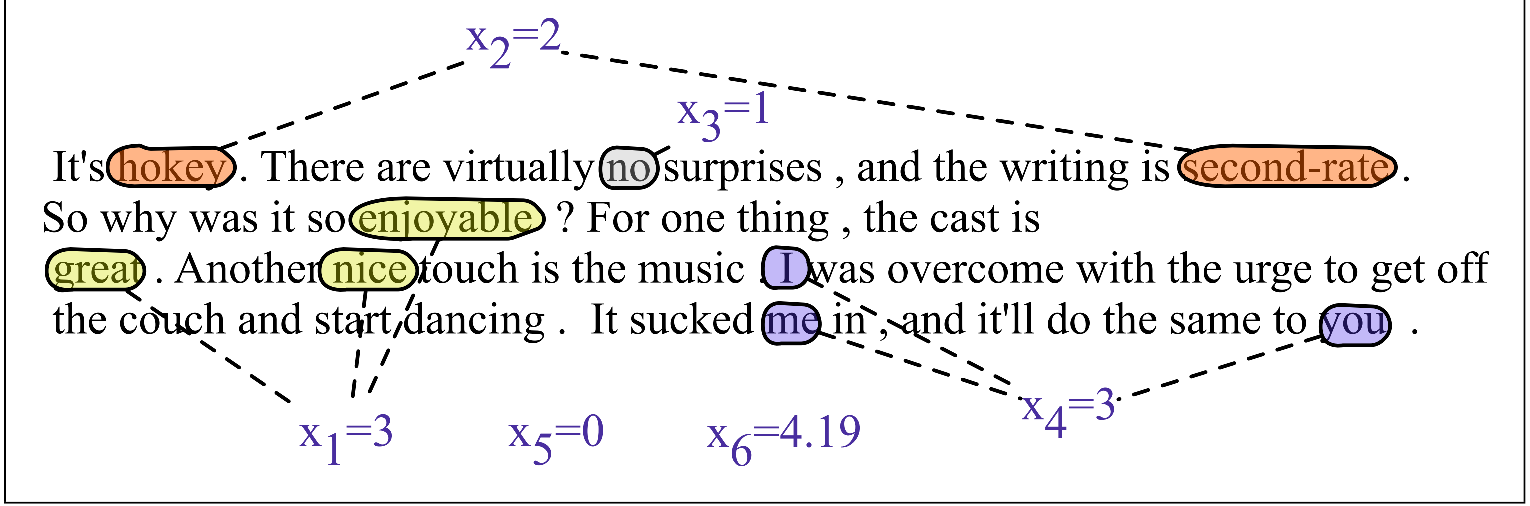
Classification in Logistic Regression

Logistic Regression

Logistic Regression: Sentiment Classification

Sentiment example: does y=1 or y=0?

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 .



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Classifying sentiment for input x

Var	Definition	Val
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Suppose $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

$b = 0.1$

Classifying sentiment for input x

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70
\end{aligned}
$$

$$
\begin{aligned}
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.30
\end{aligned}
$$

We can build features for logistic regression for any classification task: period disambiguation

End of sentence

This ends in a period.
The house at 465 Main St. is new.

Not end

$$x_1 = \begin{cases} 1 & \text{if } \text{“}Case(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if } \text{“}w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if } \text{“}w_i = \text{St. \& } Case(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases}$$

Classification in (**binary**) logistic regression: summary

Given:

- a set of classes: (+ sentiment,- sentiment)
- a vector **x** of features `[`x_1`, `x_2`, …, `x_n`]`
  - x_1= count("awesome")
  - x_2 = log(number of words in review)
- A vector **w** of weights `[`w_1`, `w_2`, …, `w_n`]`
  - w_i for each feature x_i

$$P(y=1) = \sigma(w \cdot x + b)$$

$$= \frac{1}{1+e^{-(w \cdot x+b)}}$$

Logistic Regression

Logistic Regression:
Sentiment Classification

Text Classification: Feature Engineering

More on Sentiment Classification

Sentiment Classification: Dealing with Negation

I really like this movie

I really **don't** like this movie

Negation changes the meaning of "like" to negative.

Negation can also change negative to positive-ish

- **Don't** dismiss this film
- **Doesn't** let us get bored

Sentiment Classification: Dealing with Negation

Das, Sanjiv and Mike Chen. 2001. Yahoo! for Amazon: Extracting market sentiment from stock message boards. In Proceedings of the Asia Pacific Finance Association Annual Conference (APFA).
Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. EMNLP-2002, 79—86.

Simple baseline method:

Add NOT_ to every word between negation and following punctuation:

```
didn't like this movie , but I
```

```
didn't NOT_like NOT_this NOT_movie but I
```

Sentiment Classification: Lexicons

Sometimes we don't have enough labeled training data

In that case, we can make use of pre-built word lists

Called **lexicons**

There are various publically available lexicons

MPQA Subjectivity Cues Lexicon

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann (2005). Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. Proc. of HLT-EMNLP-2005.

Riloff and Wiebe (2003). Learning extraction patterns for subjective expressions. EMNLP-2003.

Home page: http://www.cs.pitt.edu/mpqa/subj_lexicon.html

6885 words from 8221 lemmas, annotated for intensity (strong/weak)

- 2718 positive
- 4912 negative

+ : *admirable, beautiful, confident, dazzling, ecstatic, favor, glee, great*

− : *awful, bad, bias, catastrophe, cheat, deny, envious, foul, harsh, hate*

The General Inquirer

Philip J. Stone, Dexter C Dunphy, Marshall S. Smith, Daniel M. Ogilvie. 1966. The General Inquirer: A Computer Approach to Content Analysis. MIT Press

- Home page: http://www.wjh.harvard.edu/~inquirer
- List of Categories: http://www.wjh.harvard.edu/~inquirer/homecat.htm
- Spreadsheet: http://www.wjh.harvard.edu/~inquirer/inquirerbasic.xls

Categories:

- Positiv (1915 words) and Negativ (2291 words)
- Strong vs Weak, Active vs Passive, Overstated versus Understated
- Pleasure, Pain, Virtue, Vice, Motivation, Cognitive Orientation, etc

Free for Research Use

Bing Liu Opinion Lexicon

Minqing Hu and Bing Liu. Mining and Summarizing Customer Reviews. ACM SIGKDD-2004.

Bing Liu's Page on Opinion Mining

http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar

6786 words

- 2006 positive
- 4783 negative

Using Lexicons in Sentiment Classification

Add a feature that gets a count whenever a word from the lexicon occurs

- **E.g., a feature called "this word occurs in the positive lexicon" or "this word occurs in the negative lexicon"**

Now all positive words (*good, great, beautiful, wonderful*) or negative words count for that feature.

Using 1-2 features isn't as good as using all the words.

- But when training data is sparse or not representative of the test set, dense lexicon features can help

Other tasks: Spam Filtering

SpamAssassin Features:

- Mentions millions of (dollar) ((dollar) NN,NNN,NNN.NN)
- From: starts with many numbers
- Subject is all capitals
- HTML has a low ratio of text to image area
- "One hundred percent guaranteed"
- Claims you can be removed from the list
- http://spamassassin.apache.org/tests_3_3_x.html

Language ID

Determining what language a piece of text is written in.

Features based on character n-grams do very well

Important to train on lots of varieties of each language (world English, etc)

Text Classification: Feature Engineering

More on Sentiment Classification

Logistic Regression

Logistic Regression: Learning

Wait, where did the W's come from?

Supervised classification:

- we know the correct label y (either 0 or 1) for each x.

What the system produces is an estimate $\hat{y} = p(y = 1|x)$

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 (to minimize):

- **Cross-Entropy loss = Negative Log-Likelihood**

An optimization algorithm (to minimize loss):

- **Stochastic Gradient Descent**

Logistic Regression

Learning in Logistic Regression

Logistic Regression

Cross-Entropy Loss

= Negative Log-Likelihood

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

We want to know how far is the classifier output:

$$\hat{y} = \sigma(w \cdot x + b) = p_w(y=1|x)$$

from the true output:

$$y \qquad [= \text{either } 0 \text{ or } 1]$$

We'll call this difference:

$L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

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 probability (likelihood) of the true *y* labels in the training data given the observations *x*.
- w, b, that *maximize* the likelihood <=>
  - w, b that *maximize* the log-likelihood <=>
    - w, b that *minimize* the negative log-likelihood

the cross-entropy

Deriving cross-entropy loss for a single observation x

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

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

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

noting:

if y=1, this simplifies to $\hat{y}$ $\quad\quad$ $\hat{y} = \sigma(w \cdot x + b) = p_w(y=1|x)$

if y=0, this simplifies to $1-\hat{y}$ $\quad\quad$ $1-\hat{y} = 1 - p_w(y=1|x) = p_w(y=0|x)$

Deriving cross-entropy loss for a single observation x

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

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

Now take the log of both sides (mathematically handy)

Maximize: $$\begin{aligned} \log p(y|x) &= \log \left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y \log \hat{y} + (1-y) \log (1-\hat{y}) \end{aligned}$$

Whatever values maximize log p(y|x) will also maximize p(y|x)

Deriving cross-entropy loss for a single observation x

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

Maximize: $$\begin{aligned} \log p(y|x) &= \log \left[\hat{y}^{y}\,(1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y}+(1-y)\log(1-\hat{y}) \end{aligned}$$

Now flip sign to turn this into a loss: something to minimize

Cross-entropy loss (because is formula for cross-entropy(y, $\hat{y}$))

Minimize: $$L_{\mathrm{CE}}(\hat{y},y) = -\log p(y|x) \;=\; -\left[y\log\hat{y}+(1-y)\log(1-\hat{y})\right]$$

Or, plugging in definition of $\hat{y}$:

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

Let's see if this works 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

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

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w\cdot x+b) \\
&= \sigma([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.19]+0.1) \\
&= \sigma(.833) \\
&= 0.70 \qquad (5.6)
\end{aligned}
$$

Pretty well! What's the loss?

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

Let's see if this works for our sentiment example

Suppose true value instead was y=0.

$$
\begin{aligned}
p(-|x) = P(Y=0|x) &= 1-\sigma(w\cdot x+b) \\
&= 0.30
\end{aligned}
$$

What's the loss?

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

Let's see if this works for our sentiment example

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):

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

Sure enough, loss was bigger when model was wrong!

Logistic Regression

Cross-Entropy Loss

Logistic Regression

Stochastic Gradient Descent

Our goal: minimize the loss

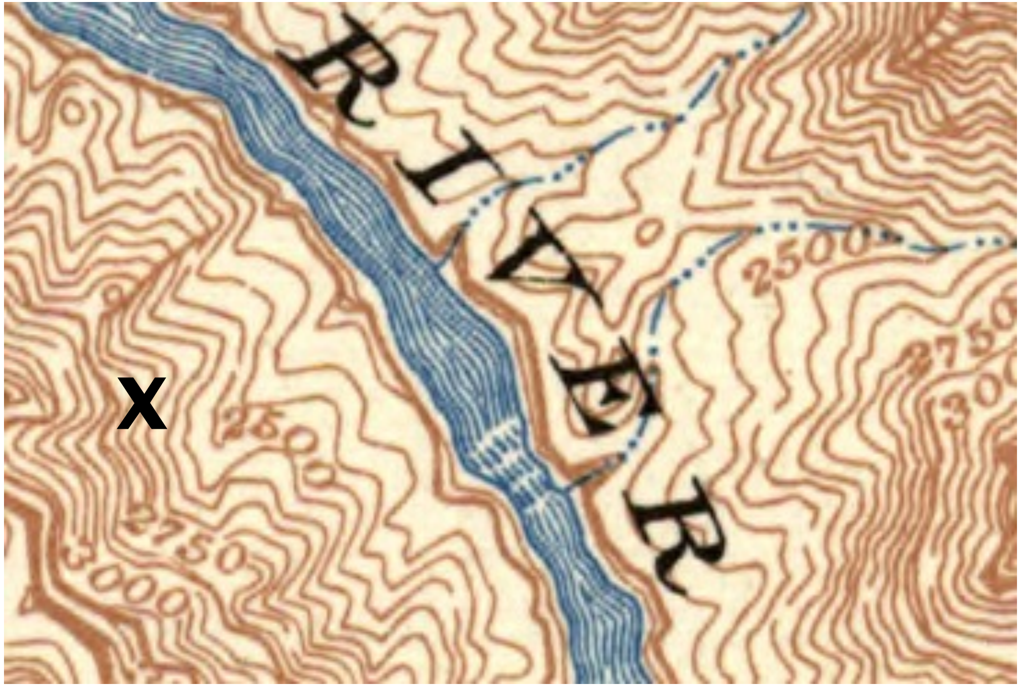
Let's make explicit that the loss function in parameterized by weights θ=(w,b)

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

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

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

Go that way

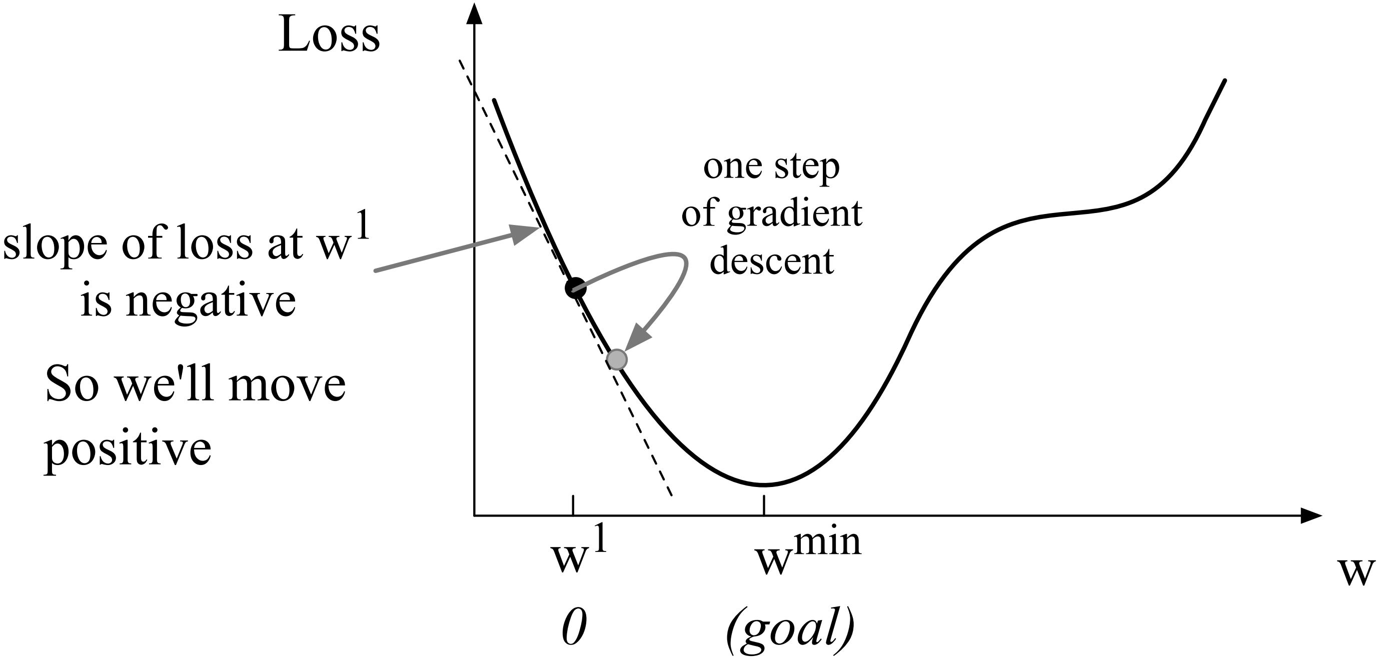
Our goal: minimize the loss

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



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) $\frac{d}{dw} f(x; w)$ weighted by a **learning rate** η
- Higher learning rate means move *w* faster

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

Now let's consider N dimensions

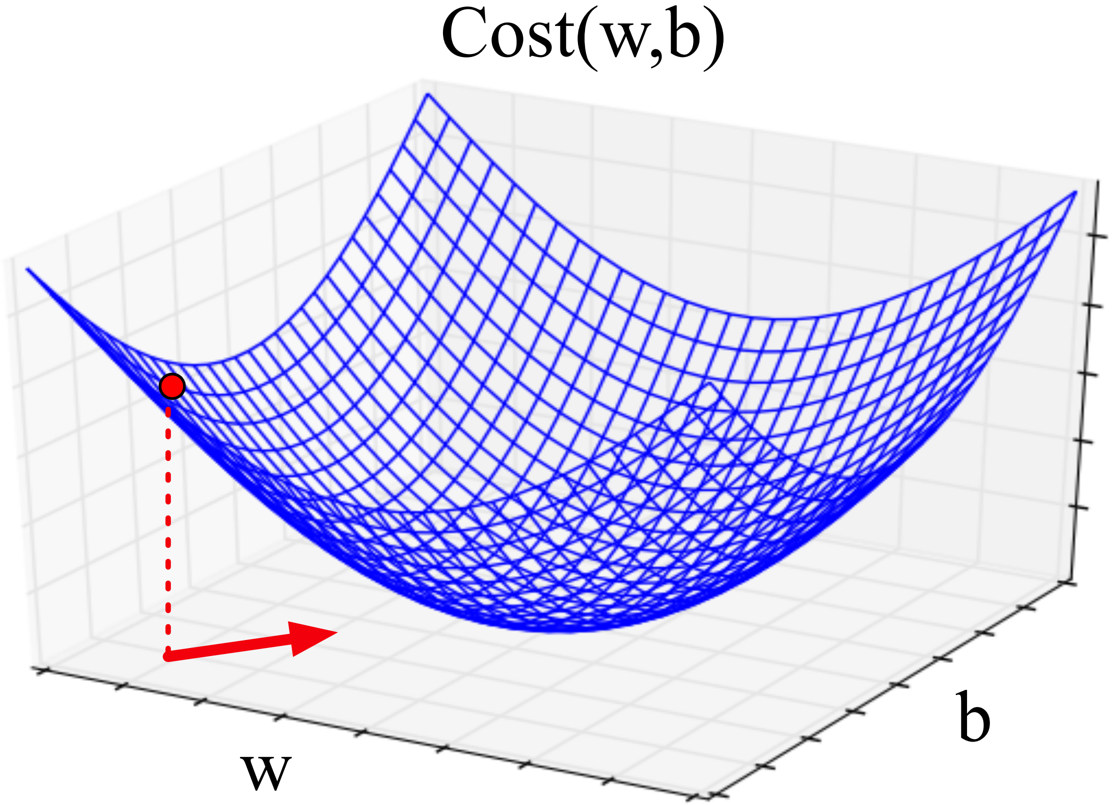
We want to know where in the N-dimensional space (of the *N* parameters that make up θ) we should move.

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



Real gradients

Are much longer; lots and lots of weights

For each dimension w_i the gradient component i tells us the slope with respect to that variable.

- "How much would a small change in w_i influence the total loss function L?"
- We express the slope as a partial derivative ∂ of the loss ∂w_i

The gradient is then defined as a vector of these partials.

The gradient

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

$$\nabla_\theta L(f(x;\theta),y)) \;=\; \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} \;=\; \theta_t - \eta \nabla L(f(x;\theta),y)$$

What are these partial derivatives for logistic regression?

The loss function

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

The elegant derivative of this function (see textbook 5.8 for derivation)

$$\frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_j} \;=\; [\sigma(w\cdot x+b)-y]x_j$$

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?

   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

More on hyperparameters in Chapter 7

- 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.

Logistic Regression

Stochastic Gradient Descent

Logistic Regression

Stochastic Gradient Descent: An example and more details

Working through an example

One step of gradient descent

A mini-sentiment example, where the true y=1 (positive)

Two features:

$x_1 = 3$ (count of positive lexicon words)

$x_2 = 2$ (count of negative lexicon words)

Assume 3 parameters (2 weights and 1 bias) in Θ^0 are zero:

$w_1 = w_2 = b = 0$

$\eta = 0.1$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

where $$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta_{t+1} \; = \; \theta_t - \eta \nabla L(f(x;\theta), y)$$

$\eta = 0.1$;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make w_2 negative

Mini-batch training

Stochastic gradient descent chooses a single random example at a time.

That can result in choppy movements

More common to compute gradient over batches of training instances.

Batch training: entire dataset

Mini-batch training: *m* examples (512, or 1024)

Logistic Regression

Stochastic Gradient Descent: An example and more details

Logistic Regression

Regularization

Overfitting

A model that perfectly matches the training data may have a problem.

It may 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

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

+

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

-

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

4gram features that just "memorize" training set and might cause problems

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

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?
    - L2 and L1 Regularization in logistic regression
    - Dropout in neural networks

Regularization

A solution for overfitting

Add a regularization term *R*(θ) to the loss function
(for now written as maximizing logprob rather than minimizing loss)

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(\theta)$$

Idea: choose an *R*(θ) that penalizes large weights

- fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

L2 Regularization (= ridge regression)

The sum of the squares of the weights

The name is because this is the (square of the) **L2 norm** $||\theta||_2$, = **Euclidean distance** of θ to the origin.

$$R(\theta) \;=\; ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

L2 regularized objective function:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} \theta_j^2$$

L1 Regularization (= lasso regression)

The sum of the (absolute value of the) weights

Named after the **L1 norm** $||W||_1$, = sum of the absolute values of the weights, = **Manhattan distance**

$$R(\theta) \;=\; ||\theta||_1 = \sum_{i=1}^{n} |\theta_i|$$

L1 regularized objective function:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |\theta_j|$$

Logistic Regression

Regularization

Logistic Regression

Multinomial Logistic Regression

Multinomial Logistic Regression

Often we need more than 2 classes

- Positive/negative/neutral
- Parts of speech (noun, verb, adjective, adverb, preposition, etc.)
- Classify emergency SMSs into different actionable classes

If >2 classes we use **multinomial logistic regression**

= Softmax regression

= Multinomial logit

= (defunct names : Maximum entropy modeling or MaxEnt)

So "logistic regression" will just mean binary (2 output classes)

Multinomial Logistic Regression

The probability of everything must still sum to 1

```
P(positive|doc) + P(negative|doc) + P(neutral|doc) = 1
```

Need a generalization of the sigmoid called the **softmax**

- Takes a vector $z = [z1, z2, ..., zk]$ of k arbitrary values
- Outputs a probability distribution
  - each value in the range [0,1]
  - all the values summing to 1

The **softmax** function

Turns a vector $z = [z_1, z_2, \ldots, z_k]$ of k arbitrary values into probabilities

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \le i \le k$$

The denominator $\sum_{i=1}^{k} e^{z_i}$ is used to normalize all the values into probabilities.

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

The **softmax** function

- Turns a vector $z = [z_1, z_2, \ldots, z_k]$ of k arbitrary values into probabilities

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\mathrm{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Softmax in multinomial logistic regression

$$p(y=c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} \exp(w_j \cdot x + b_j)}$$

Input is still the dot product between weight vector *w* and input vector *x*
But now we'll need separate weight vectors for each of the *K* classes.

Features in binary versus multinomial logistic regression

Binary: pos weight => y=1, neg weight => y=0

$$x_5 = \begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \qquad \mathrm{w}_5 = 3.0$$

Multinominal: separate weights for each class:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	-5.3

Logistic Regression

Multinomial Logistic Regression

Text Classification: Performance Measures

Precision, Recall, and F measure

Evaluation

Let's consider just binary text classification tasks

Imagine you're the CEO of Delicious Pie Company

You want to know what people are saying about your pies

So you build a "Delicious Pie" tweet detector

- Positive class: tweets about Delicious Pie Co
- Negative class: all other tweets

The 2-by-2 confusion matrix

gold standard labels

system output labels		gold positive	gold negative
	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

$$\textbf{precision} = \frac{\text{tp}}{\text{tp+fp}}$$

$$\textbf{recall} = \frac{\text{tp}}{\text{tp+fn}}$$

$$\textbf{accuracy} = \frac{\text{tp+tn}}{\text{tp+fp+tn+fn}}$$

Evaluation: Accuracy

Why don't we use **accuracy** as our metric?

Imagine we saw 1 million tweets

- 100 of them talked about Delicious Pie Co.
- 999,900 talked about something else

We could build a simple classifier that just labels every tweet "not about pie"

- It would get 99.99% accuracy!!! Wow!!!!
- But useless! Doesn't return the comments we are looking for!
- That's why we use **precision** and **recall** instead

100 gold positive.
999,900 gold negative

0 system positive
1,000,000 system negative

tp = 0 fp = 0

fn = 100 tn = 999,900

=> P = 0 / (0 + 0)
=> R = 0 / (0 + 100) = 0%

gold standard labels

system output labels		gold positive	gold negative	
	system positive	**true positive**	**false positive**	$\textbf{precision} = \frac{tp}{tp+fp}$
	system negative	**false negative**	**true negative**	
		$\textbf{recall} = \frac{tp}{tp+fn}$		$\textbf{accuracy} = \frac{tp+tn}{tp+fp+tn+fn}$

Evaluation: Precision

% of items the system detected (i.e., items the system labeled as positive) that are in fact positive (according to the human gold labels)

$$\textbf{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Evaluation: Recall

% of items actually present in the input that were correctly identified by the system.

$$\textbf{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Why Precision and recall

Our simple pie-classifier

- Just label nothing as "about pie"

Accuracy=99.99%

but

Recall = 0

- (it doesn't get any of the 100 Pie tweets)

Precision and recall, unlike accuracy, emphasize true positives:

- finding the things that we are supposed to be looking for.

A combined measure: F

F measure: a single number that combines P and R:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

We almost always use balanced F_1 (i.e., $\beta = 1$)

$$\mathrm{F}_1 = \frac{2PR}{P + R}$$

Why not use Avg = (P + R)/2?
1. P = 70%   R = 70%
2. P = 90%   R = 50%
Same Avg = 70%, but different F_1

Evaluation with more than two classes:
Confusion Matrix for 3-class classification

gold labels

system output	urgent	normal	spam	
urgent	8	10	1	**precision**u= $\frac{8}{8+10+1}$
normal	5	60	50	**precision**n= $\frac{60}{5+60+50}$
spam	3	30	200	**precision**s= $\frac{200}{3+30+200}$
	recallu = $\frac{8}{8+5+3}$	**recall**n = $\frac{60}{10+60+30}$	**recall**s = $\frac{200}{1+50+200}$	

How to combine P/R from 3 classes to get one metric

Macro-averaging:

- compute the performance for each class, and then average over classes

Micro-averaging:

- collect decisions for all classes into one confusion matrix
- compute precision and recall from that table.

Macro-averaging and Micro-averaging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Development Test Sets ("Devsets") and Cross-validation

Training set | Development Test Set | Test Set

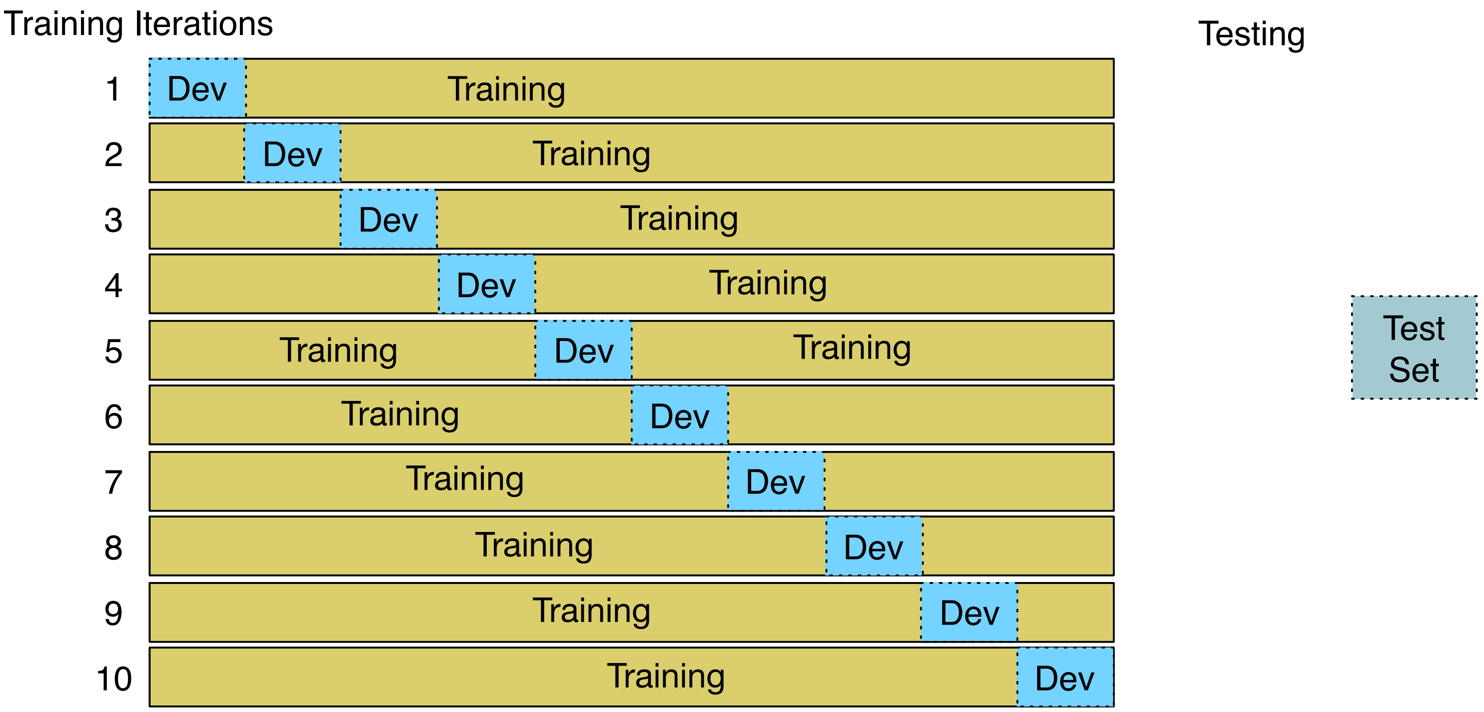
Train on training set, tune on devset, report on testset

- This avoids overfitting ('tuning to the test set')
- More conservative estimate of performance
- But paradox: want as much data as possible for training, and as much for dev; how to split?

See more details here:
https://scikit-learn.org/stable/modules/cross_validation.html

Cross-validation: multiple splits

Pool results over splits, Compute pooled dev performance



Text Classification: Performance Measures

Precision, Recall, and F measure

Recommended readings

- Chapter 5 (LR) in J&M.
- Section 2.5 (LR), section 4.1 (sentiment analysis), and section 4.4 (evaluation measures) in Eisenstein.