Natural Language Processing

Logistic Regression

Yulia Tsvetkov

yuliats@cs.washington.edu

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

Readings

- J&M Chapter 4 https://web.stanford.edu/~jurafsky/slp3/4.pdf
-

Next class: Logistic regression

- Supervised text classification
  - Rule-based
  - Probabilistic
    - Generative models
      - Naïve Bayes
    - Discriminative models
      - Linear models
        - Logistic regression
      - Non-linear models
        - Multilayer perceptron

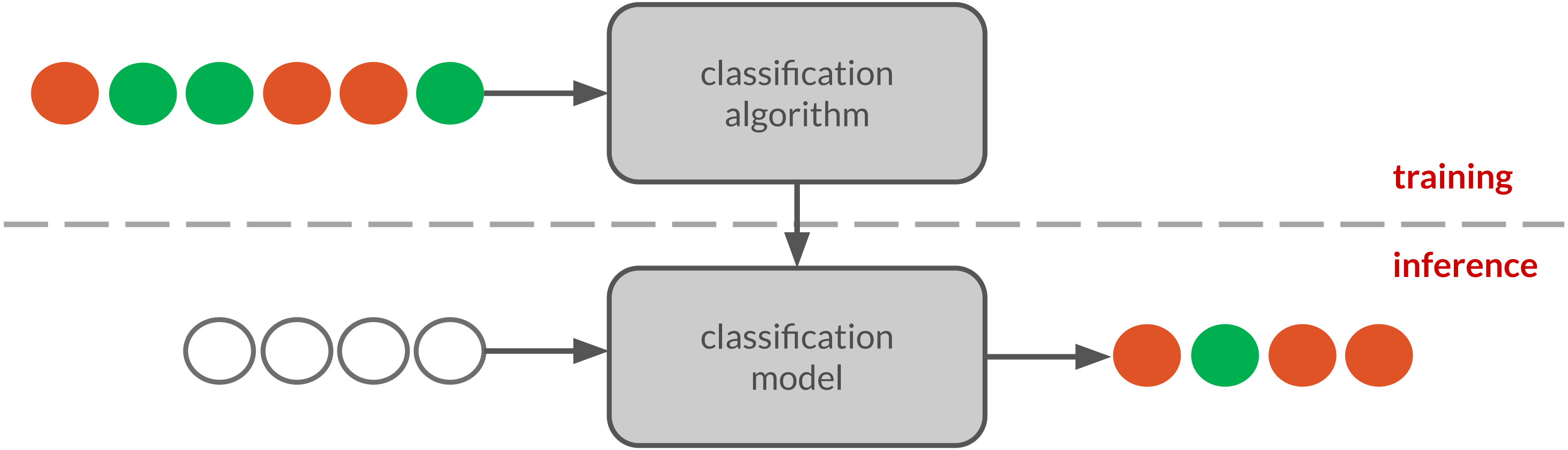
Classification: learning from data

- Supervised
  - labeled examples
    - Binary (true, false)
    - Multi-class classification (politics, sports, gossip)
    - Multi-label classification (#party #FRIDAY #fail)
- Unsupervised
  - no labeled examples
- Semi-supervised
  - labeled examples + non-labeled examples
- Weakly supervised
  - heuristically-labeled examples

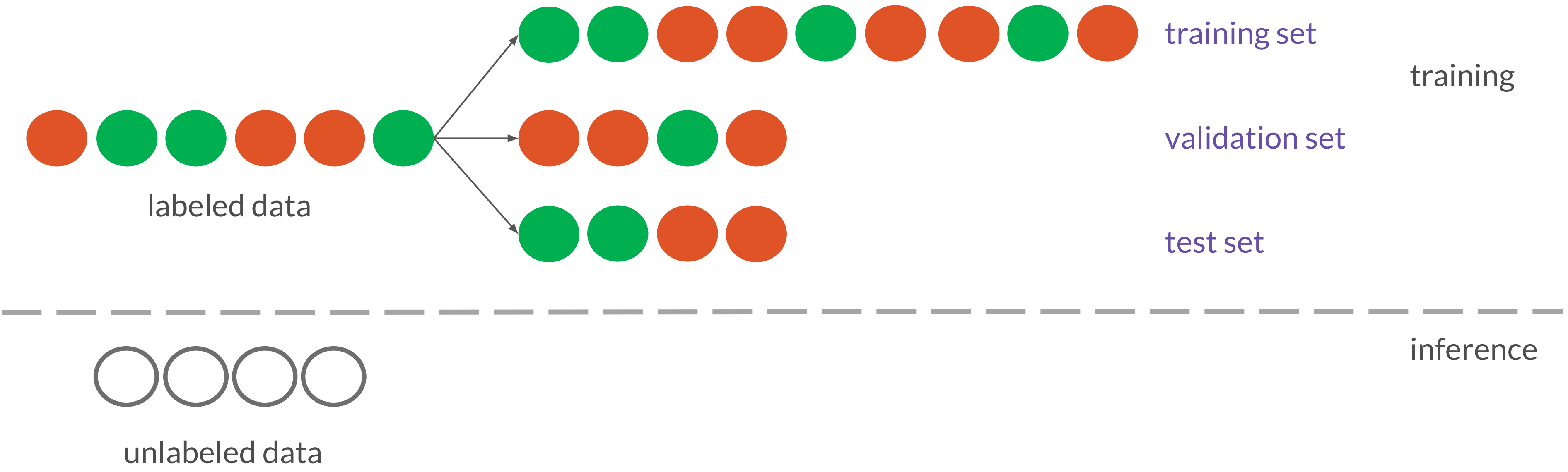
Where do datasets come from?

Human institutions	Noisy labels	Expert annotation	Crowd workers
Government proceedings	Domain names	Treebanks	Question answering
Product reviews	Link text	Biomedical corpora	Image captions

Supervised classification



Training, validation, and test sets



Supervised classification: formal setting

- Learn a classification model from labeled data on
  - properties ("features") and their importance ("weights")
- X: set of attributes or features $\{x_1, x_2, \ldots, x_n\}$
  - e.g. fruit measurements, or word counts extracted from an input documents
- y: a "class" label from the label set $Y = \{y_1, y_2, \ldots, y_k\}$
  - e.g., fruit type, or spam/not spam, positive/negative/neutral

Supervised classification: training

- Learn a classification model from labeled data on
  - properties ("features") and their importance ("weights")
- X: set of attributes or features $\{x_1, x_2, \ldots, x_n\}$
  - e.g. fruit measurements, or word counts extracted from an input documents
- y: a "class" label from the label set $Y = \{y_1, y_2, \ldots, y_k\}$
  - e.g., fruit type, or spam/not spam, positive/negative/neutral

- Given data samples $\{x_1, x_2, \ldots, x_n\}$ and corresponding labels $Y = \{y_1, y_2, \ldots, y_k\}$
- We train a function $f: x \in X \rightarrow y \in Y$ (the model)

Supervised classification: inference

- Learn a classification model from labeled data on
  - properties ("features") and their importance ("weights")
- X: set of attributes or features $\{x_1, x_2, \ldots, x_n\}$
  - e.g. fruit measurements, or word counts extracted from an input documents
- y: a "class" label from the label set $Y = \{y_1, y_2, \ldots, y_k\}$
  - e.g., fruit type, or spam/not spam, positive/negative/neutral

- At inference time, apply the model on new instances to predict the label $\hat{y}_i$

Generative and discriminative models

- Generative text classification: Learn a model of the joint P(X, y), and find

$$\hat{y} = \underset{\tilde{y}}{\operatorname{argmax}}\ P(X, \tilde{y})$$

- Discriminative text classification: Learn a model of the conditional P(y | X), and find

$$\hat{y} = \underset{\tilde{y}}{\operatorname{argmax}}\ P(\tilde{y}|X)$$

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

- Naive Bayes

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

- Logistic Regression

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

Next class: Logistic regression

- Supervised text classification
  - Rule-based
  - Probabilistic
    - Generative models
      - Naïve Bayes
    - Discriminative models
      - Linear models
        - Logistic regression
      - Non-linear models
        - Multilayer perceptron

Logistic regression classifier

- Important analytic tool in natural and social sciences
- Baseline supervised machine learning tool for classification
- Is also the foundation of neural networks

Text classification

Input:

- a document d (e.g., a movie review)
- a fixed set of classes $C = \{c_1, c_2, \ldots c_j\}$ (e.g., positive, negative, neutral)

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 $x_i \in \{x_1, x_2, \ldots, x_n\}$, weight $w_i \in \{w_1, w_2, \ldots, w_n\}$ tells us how important is x_i
  - x_i = "review contains 'awesome'": w_i = +10
  - x_j = "review contains horrible": w_j = -10
  - x_k = "review contains 'mediocre'": w_k = -2

Logistic Regression for one observation x

- Input observation: vector $x^{(i)} = \{x_1, x_2, \ldots, x_n\}$
- Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$
  - Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$
- Output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

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

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 ranges from $-\infty$ to ∞

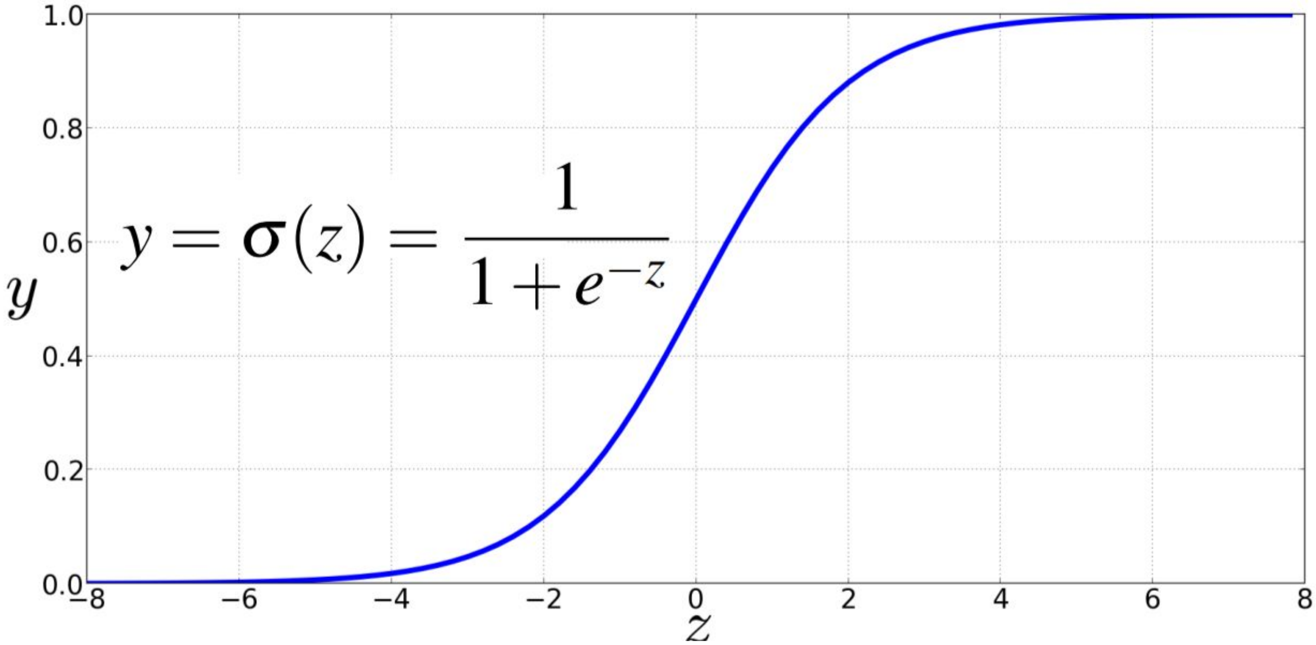
$$z = w \cdot x + b$$

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

"sigmoid" or "logistic" function

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

The very useful sigmoid or logistic function



Idea of logistic regression

- We'll compute w·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

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

Making probabilities with sigmoids

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

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

By the way:

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

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

Because

$$1-\sigma(x)=\sigma(-x)$$

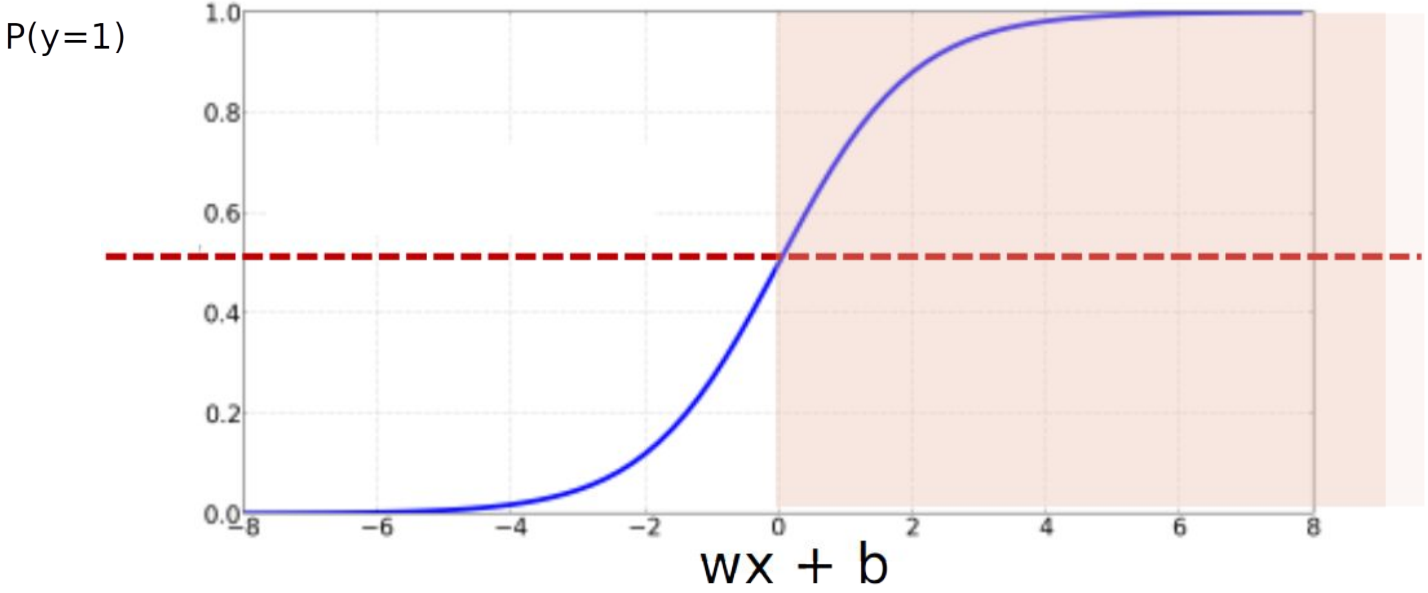
Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

- 0.5 here is called the **decision boundary**

The probabilistic classifier

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



Turning a probability into a classifier

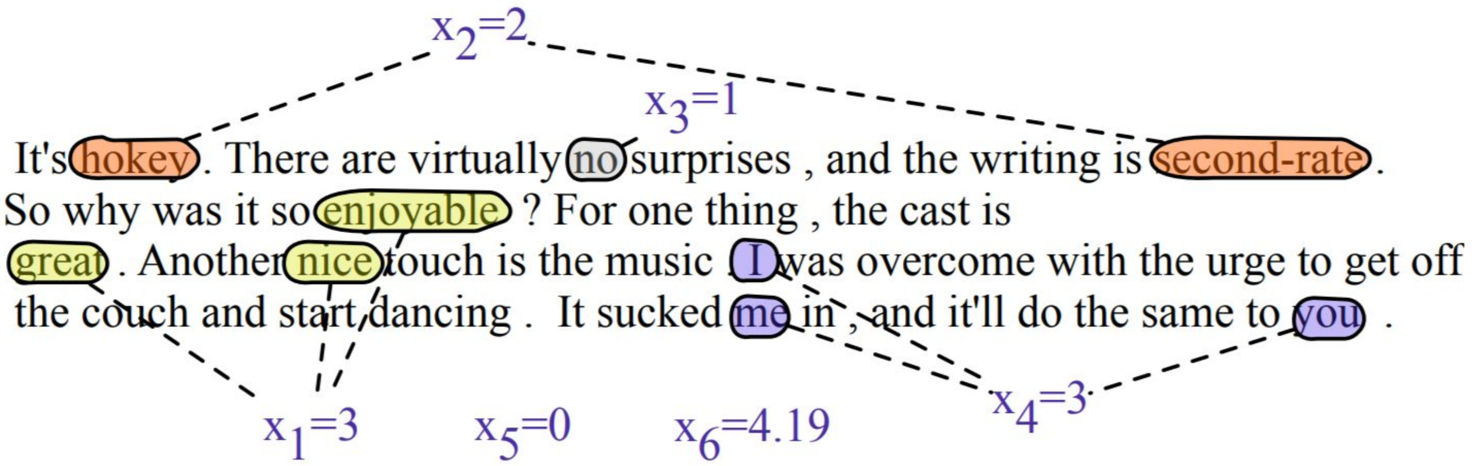
$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

if $w \cdot x + b > 0$

if $w \cdot x + b \leq 0$

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
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	Value
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 $\mathbf{w}$ = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]

b = 0.1

Classifying sentiment for input x

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

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

Scaling input features

- z-score

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_i^{(j)} \qquad \sigma_i = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(\mathbf{x}_i^{(j)} - \mu_i\right)}$$

$$\mathbf{x}_i' = \frac{\mathbf{x}_i - \mu_i}{\sigma_i}$$

- normalize

$$\mathbf{x}_i' = \frac{\mathbf{x}_i - \min(\mathbf{x}_i)}{\max(\mathbf{x}_i) - \min(\mathbf{x}_i)}$$

Wait, where did the W's come from?

- Supervised classification:
  - At training time we know the correct label y (either 0 or 1) for each x.
  - But what the system produces at inference time is an estimate $\hat{y}$

Wait, where did the W's come from?

- Supervised classification:
  - A training time we know the correct label y (either 0 or 1) for each x.
  - But what the system produces at inference time 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 in LR

A loss function:

- **cross-entropy loss**

An optimization algorithm:

- **stochastic gradient descent**

Loss function: the distance between ŷ and y

We want to know how far is the classifier output $\hat{y} = \sigma(w \cdot x + b)$

from the true output: y [= either 0 or 1]

We'll call this difference: $L(\hat{y},y)$ = how much ŷ differs from the true y

Intuition of negative log likelihood loss (NLL) = 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

Todo – combine prev and next parts

Components of a probabilistic machine learning classifier

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

1. A **feature representation** for the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \ldots, x_n]$. Feature j for input $x^{(i)}$ is x_j, more completely $x_1^{(i)}$, or sometimes $f_j(x)$.

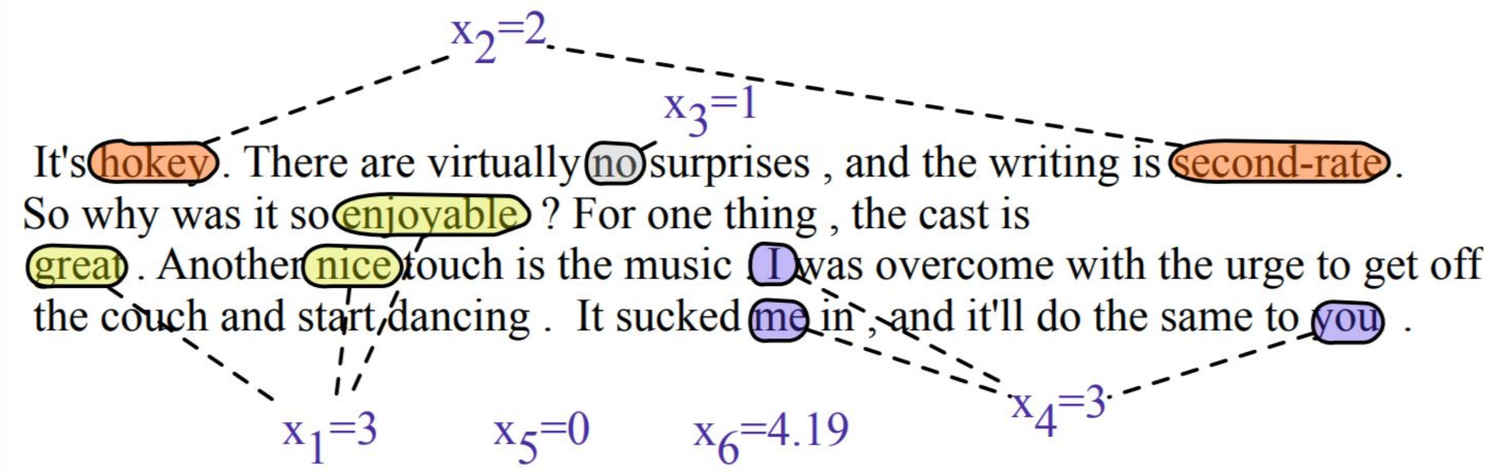
2. A **classification function** that computes $\hat{y}$ the estimated class, via $p(y|x)$, like the **sigmoid** functions

3. An **objective function** for learning [today]

4. An algorithm for **optimizing** the objective function [Friday]

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
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	Value
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

Logistic Regression for one observation x

- Input observation: vector $x^{(i)} = \{x_1, x_2, \ldots, x_n\}$
- Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$
  - Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$
- Output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

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

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 ranges from $-\infty$ to ∞

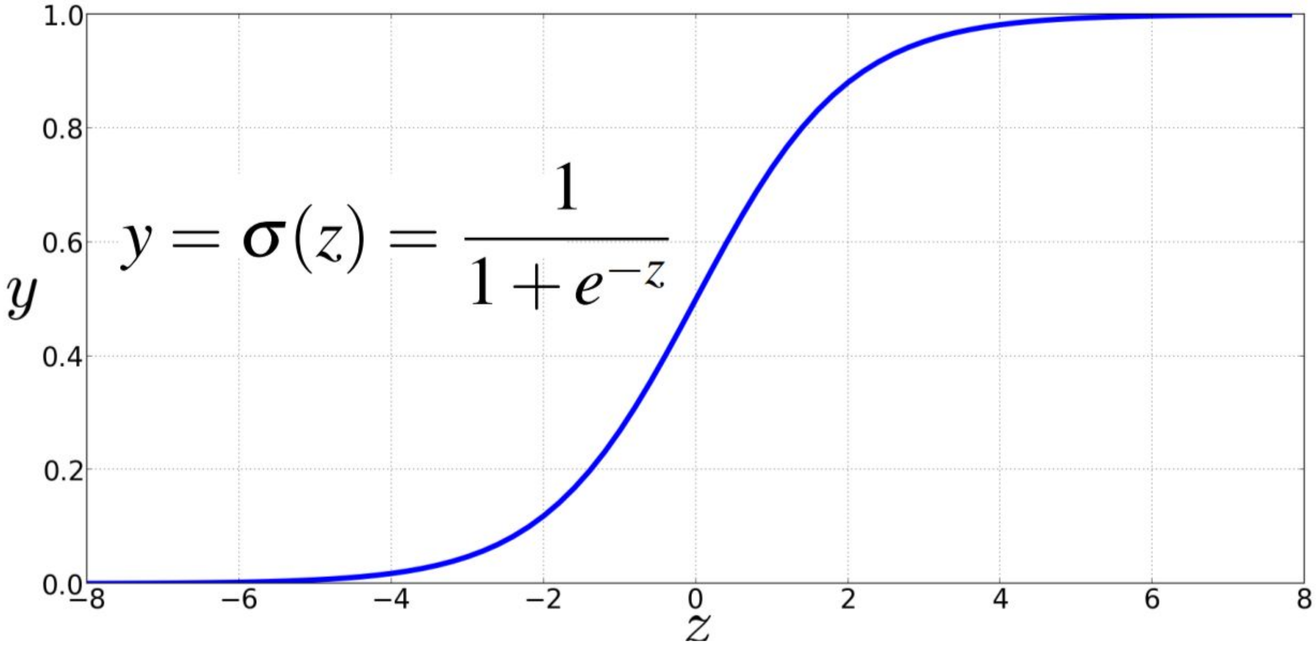
$$z = w \cdot x + b$$

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

"sigmoid" or "logistic" function

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

The very useful sigmoid or logistic function



Idea of logistic regression

- We'll compute w·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

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

Making probabilities with sigmoids

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

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

By the way:

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

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

Because

$$1-\sigma(x)=\sigma(-x)$$

Turning a probability into a classifier

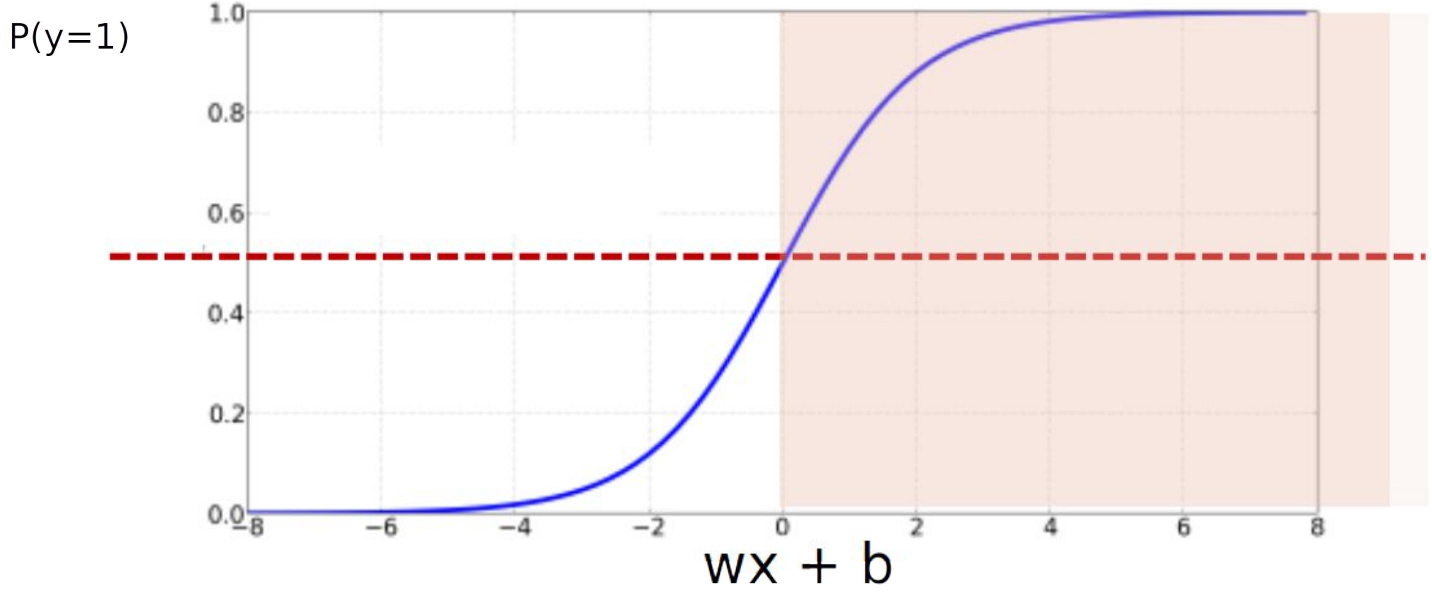
$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

- 0.5 here is called the **decision boundary**

The probabilistic classifier

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

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



Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

if $w \cdot x + b > 0$

if $w \cdot x + b \leq 0$

Scaling input features

- z-score

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_i^{(j)} \qquad \sigma_i = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(\mathbf{x}_i^{(j)} - \mu_i\right)}$$

$$\mathbf{x}_i' = \frac{\mathbf{x}_i - \mu_i}{\sigma_i}$$

- normalize

$$\mathbf{x}_i' = \frac{\mathbf{x}_i - \min(\mathbf{x}_i)}{\max(\mathbf{x}_i) - \min(\mathbf{x}_i)}$$

Wait, where did the W's come from?

- Supervised classification:
  - A training time we know the correct label y (either 0 or 1) for each x.
  - But what the system produces at inference time is an estimate $\hat{y}$

Wait, where did the W's come from?

- Supervised classification:
  - A training time we know the correct label y (either 0 or 1) for each x.
  - But what the system produces at inference time 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

Components of a probabilistic machine learning classifier

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

1. A **feature representation** for the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \ldots, x_n]$. Feature j for input $x^{(i)}$ is x_j, more completely $x_1^{(i)}$, or sometimes $f_j(x)$.

2. A **classification function** that computes $\hat{y}$ the estimated class, via $p(y|x)$, like the **sigmoid** functions

3. An **objective function** for learning, like **cross-entropy loss**

4. An algorithm for **optimizing** the objective function: **stochastic gradient descent** [next class]

Learning components in LR

A loss function:

- **cross-entropy loss**

An optimization algorithm:

- **stochastic gradient descent**

Loss function: the distance between ŷ and y

We want to know how far is the classifier output $\hat{y} = \sigma(w \cdot x + b)$

from the true output: y [= either 0 or 1]

We'll call this difference: $L(\hat{y}, y)$ = how much ŷ 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 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)

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

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

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 $\hat{y}$

if $y=0$, 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: $p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y}$

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

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

Deriving cross-entropy loss for a single observation x

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

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

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

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

Deriving cross-entropy loss for a single observation x

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

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

Now flip sign to turn this into a **cross-entropy loss**: something to minimize

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

Deriving cross-entropy loss for a single observation x

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

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

Now flip sign to turn this into a **cross-entropy loss**: something to minimize

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

Or, plug in definition of $\hat{y} = \sigma(w \cdot x+b)$

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

Let's see if this works for our sentiment example

We want loss to be:

- smaller if the model estimate $\hat{y}$ 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 (positive). How well is our model doing?

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

Pretty well!

Let's see if this works for our sentiment example

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

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

Pretty well! What's the loss?

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

Let's see if this works for our sentiment example

Suppose the true value instead was y=0 (negative).

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

Let's see if this works for our sentiment example

Suppose the true value instead was y=0 (negative).

$$p(-|x) = P(Y=0|x) = 1 - \sigma(w \cdot x + b)$$
$$= 0.30$$

What's the loss?

$$L_{\text{CE}}(\hat{y}, y) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1-y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$$
$$= -[\log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$$
$$= -\log(.30)$$
$$= 1.2$$

Let's see if this works for our sentiment example

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

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

The loss when the model was wrong (if true y=0)

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

Sure enough, loss was bigger when model was wrong!

Learning components

A loss function:

- **cross-entropy loss**

An optimization algorithm:

- **stochastic gradient descent**

Stochastic Gradient Descent

- Stochastic Gradient Descent algorithm
  - is used to optimize the weights
  - for logistic regression
  - also for neural networks

Our goal: minimize the loss

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

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

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

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

$$L_{CE}(\hat{y}, y)$$

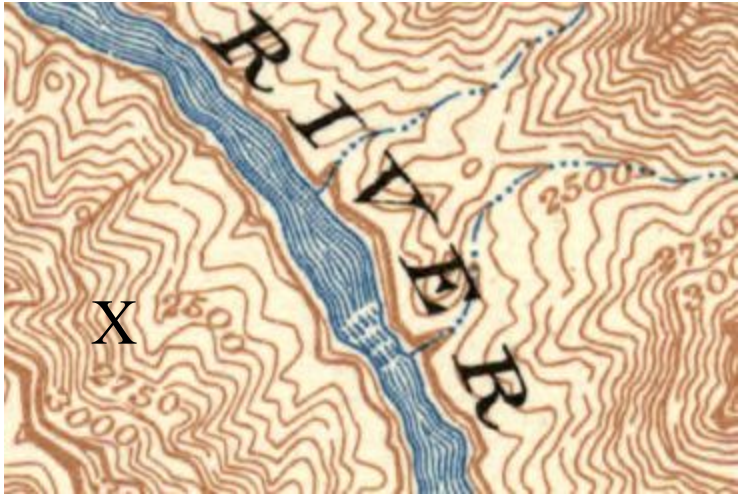
Intuition of gradient descent

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

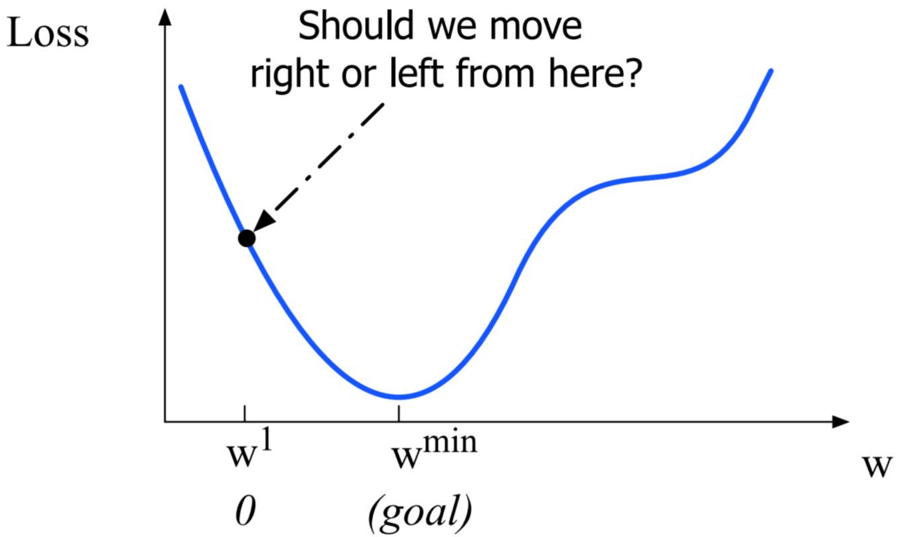
For logistic regression, loss function is **convex**

- 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?

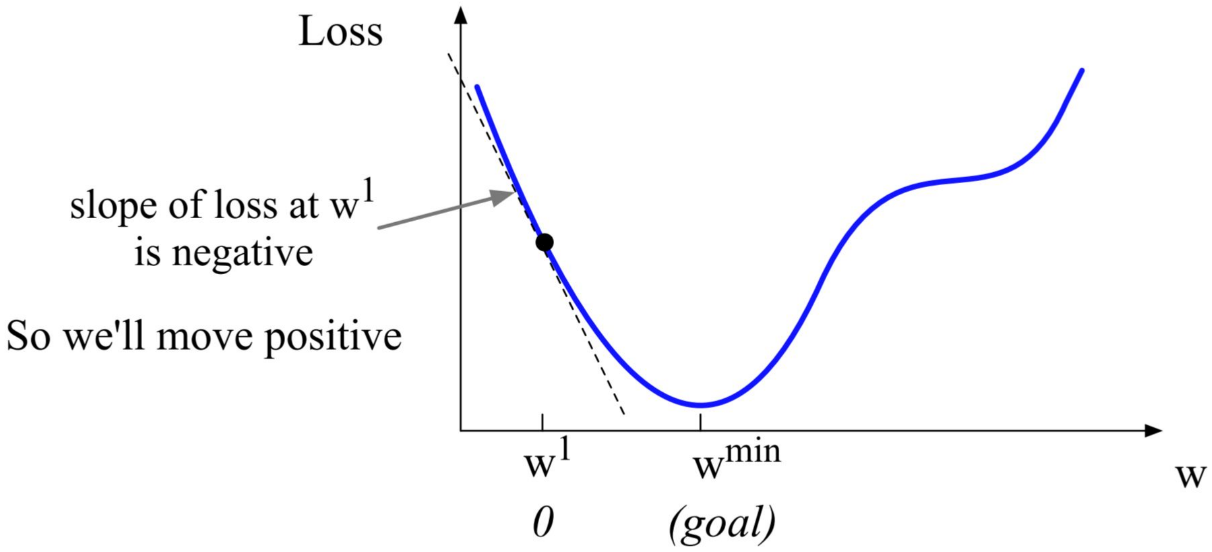
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

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

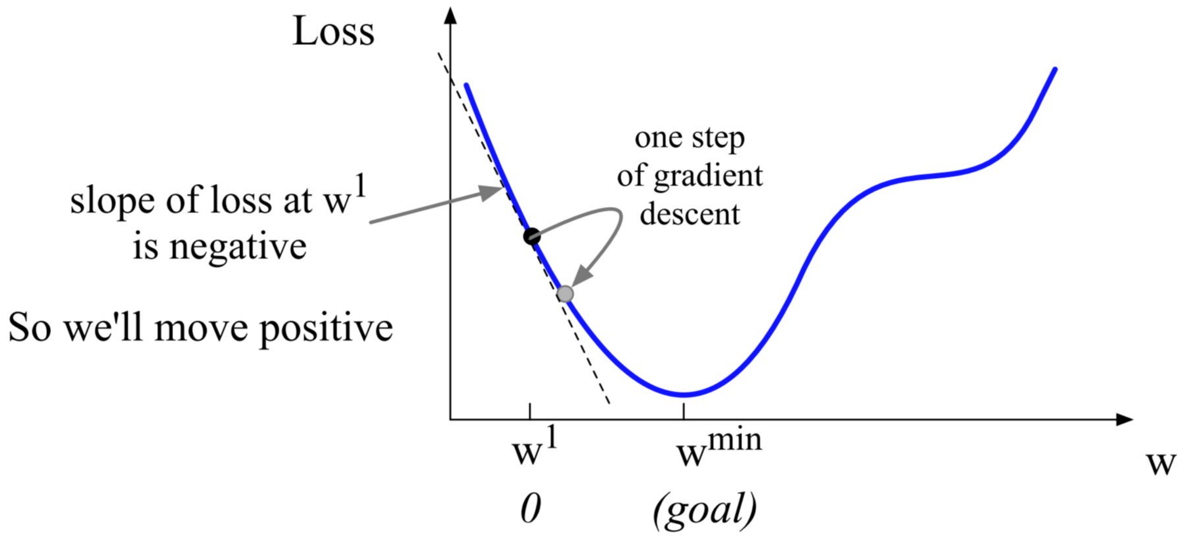
A: Move w in the reverse direction from the slope of the function



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) $\frac{d}{dw}L(f(x;w),y)$
  - weighted by a learning rate η

- Higher learning rate means move w faster

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

Now let's consider N dimensions

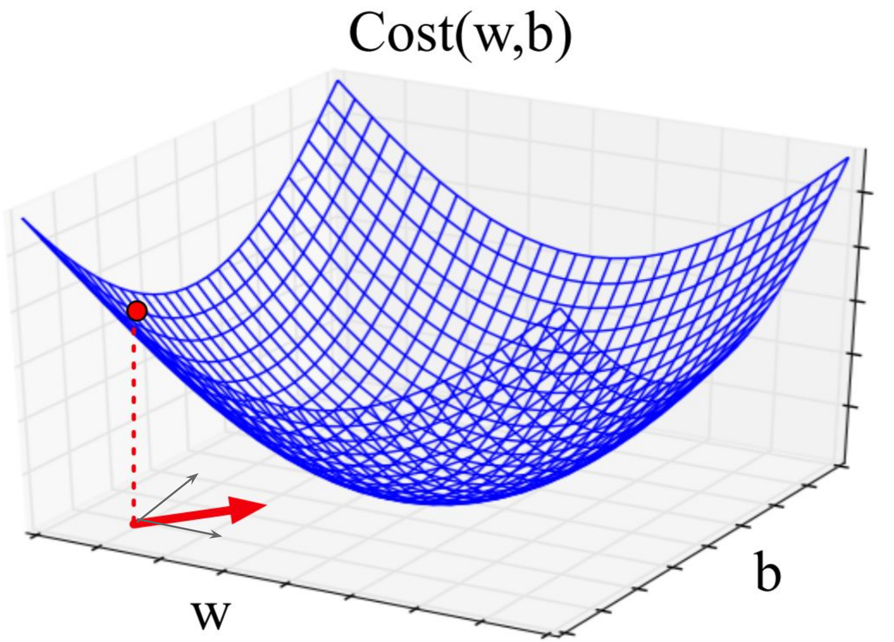
We want to know where in the N-dimensional space (of the N parameters that make up θ) we should move.

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 $\frac{\partial}{\partial w_i}$

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

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

The elegant derivative of this function (see Section 5.10 for the derivation)

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

$$= (\hat{y} - y) \mathbf{x}_j$$

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
  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

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.

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)

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**

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)

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

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

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

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

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

L1 regularized objective function:

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

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 $z = [z_1, z_2, ..., z_k]$ of k arbitrary values
- Outputs a probability distribution
- each value in the range $[0,1]$
- all the values summing to 1

We'll discuss it more when we talk about neural networks

Components of a probabilistic machine learning classifier

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

1. A **feature representation** for the input. For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \ldots, x_n]$. Feature j for input $x^{(i)}$ is x_j, more completely $x_1^{(i)}$, or sometimes $f_j(x)$.

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**

Next class:

- Language models