Logistic regression classifiers

CS 585, Fall 2017

Introduction to Natural Language Processing

http://people.cs.umass.edu/~brenocon/inlp2017

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

[incl. slides from Ari Kobren]

Naive Bayes: bag of words

Naive Bayes: bag of words

- BoW - Order independent

Naive Bayes: bag of words

- BoW - Order independent
- Can we add more features to the model?

Naive Bayes: bag of words

- BoW - Order independent
- Can we add more features to the model?

- NB assumes: Feature statistically independent, given class

Naive Bayes: bag of words

- BoW - Order independent
- Can we add more features to the model?

- NB assumes: Feature statistically independent, given class
- Examples of non-independent features?

Naive Bayes: bag of words

- BoW - Order independent
- Can we add more features to the model?

- NB assumes: Feature statistically independent, given class
- Examples of non-independent features?

- Correlated features => double counting

Naive Bayes: bag of words

- BoW - Order independent
- Can we add more features to the model?

- NB assumes: Feature statistically independent, given class
- Examples of non-independent features?

- Correlated features => double counting
- Can hurt classifier accuracy and calibration

Logistic regression

- Log Linear Model - a.k.a. Logistic regression classifier
- Kinda like Naive Bayes, but:
  - Doesn't assume features are independent
    - Correlated features don't "double count"
  - Discriminative training: optimize *p(y | text)*, not *p(y, text)*
  - Tends to work better - state of the art for doc classif, widespread hard-to-beat baseline for many tasks
  - Good off-the-shelf implementations (e.g. scikit-learn)

Features! Features! Features!

- Input document **d** (a string...)
- Engineer a <u>feature function</u>, f(d), to generate <u>feature vector</u> ***x***

$$f(d) \longrightarrow x$$

f(d) = (
Count of "happy",
(Count of "happy") / (Length of doc),
log(1 + count of "happy"),
Count of "not happy",
Count of words in my pre-specified word list, "positive words according to my favorite psychological theory",
Count of "of the",
Length of document,
...
)

Typically these use <u>feature templates</u>:
Generate many features at once

for each word w:
- ${w}_count
- ${w}_log_1_plus_count
- ${w}_with_NOT_before_it_count
- ....

- Not just word counts. Anything that might be useful!
- <u>Feature engineering</u>: when you spend a lot of trying and testing new features. Very important!! This is a place to put linguistics in.

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.

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
```

Classification: LogReg (I)

First, we'll discuss **how LogReg works.**

Classification: LogReg (I)

First, we'll discuss **how LogReg works.**

Then, **why** it's set up the way that it is.

Application: **spam filtering**

Classification: LogReg (I)

- compute **features** (xs)

Classification: LogReg (I)

- compute **features** (xs)

x_i = (count “nigerian”, count “prince”, count “nigerian prince”)

Classification: LogReg (I)

- compute **features** (xs)

x_i = (count "nigerian", count "prince", count "nigerian prince")

- given **weights** (betas)

Classification: LogReg (I)

- compute **features** (xs)

x_i = (count “nigerian”, count “prince”, count “nigerian prince”)

- given **weights** (betas)

β = (-1.0, -1.0, 4.0)

Classification: LogReg (I)

- compute **features** (x's)
- given **weights** (betas)
- compute the **dot product**

Classification: LogReg (I)

- compute **features** (x's)
- given **weights** (betas)
- compute the **dot product**

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

Classification: LogReg (II)

- compute the **dot product**

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

Classification: LogReg (II)

- compute the **dot product**

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

- compute the **logistic function**

$$\boxed{P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}}$$

LogReg Exercise

features: (count “nigerian”, count “prince”, count “nigerian prince”)

x = (1, 1, 1)

β = (-1.0, -1.0, 4.0)

$$P(x) = ???$$

LogReg Exercise

x = (1, 1, 1)

β = (-1.0, -1.0, 4.0)

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

Classification: LogReg

OK, let's take this step by step...

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

- **Why dot product?**

Classification: LogReg

OK, let's take this step by step...

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

- Why dot product?

- **Why would we use the logistic function?**

Classification: Dot Product

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

Intuition: **weighted sum of features**

All linear models have this form!

NB as Log-Linear Model

Recall that Naive Bayes is also a linear model...

NB as Log-Linear Model

- What are the **features** in Naive Bayes?

- What are the **weights** in Naive Bayes?

NB as Log-Linear Model

$$P(\text{spam}|D) \propto P(\text{spam}) \cdot \prod_{w_i \in D} P(w_i|\text{spam})$$

NB as Log-Linear Model

$$P(\text{spam}|D) \propto P(\text{spam}) \cdot \prod_{w_i \in D} P(w_i|\text{spam})$$

$$P(\text{spam}|D) \propto P(\text{spam}) + \prod_{w_i \in \text{Vocab}} \cdot P(w_i|\text{spam})^{x_i}$$

NB as Log-Linear Model

$$P(\text{spam}|D) \propto P(\text{spam}) \cdot \prod_{w_i \in D} P(w_i|\text{spam})$$

$$P(\text{spam}|D) \propto P(\text{spam}) \quad \cdot \prod_{w_i \in \text{Vocab}} \cdot P(w_i|\text{spam})^{x_i}$$

$$\log[P(\text{spam}|D)] \propto \log[P(\text{spam})] + \sum_{w_i \in \text{Vocab}} x_i \cdot \log[P(w_i|\text{spam})]$$

NB as log-linear model

$$P(\text{spam} \mid D) = \frac{1}{Z} P(\text{spam}) \prod_{t=1}^{\text{len}(D)} P(w_t \mid \text{spam})$$

$$P(\text{spam} \mid D) = \frac{1}{Z} P(\text{spam}) \prod_{w \in \mathcal{V}} P(w \mid \text{spam})^{x_w}$$

$$\log P(\text{spam} \mid D) = \log P(\text{spam}) + \sum_{w \in \mathcal{V}} x_w \log P(w \mid \text{spam})$$
$$- \log Z$$

NB as Log-Linear Model

In both NB and LogReg

we **compute the dot product!**

Logistic Function

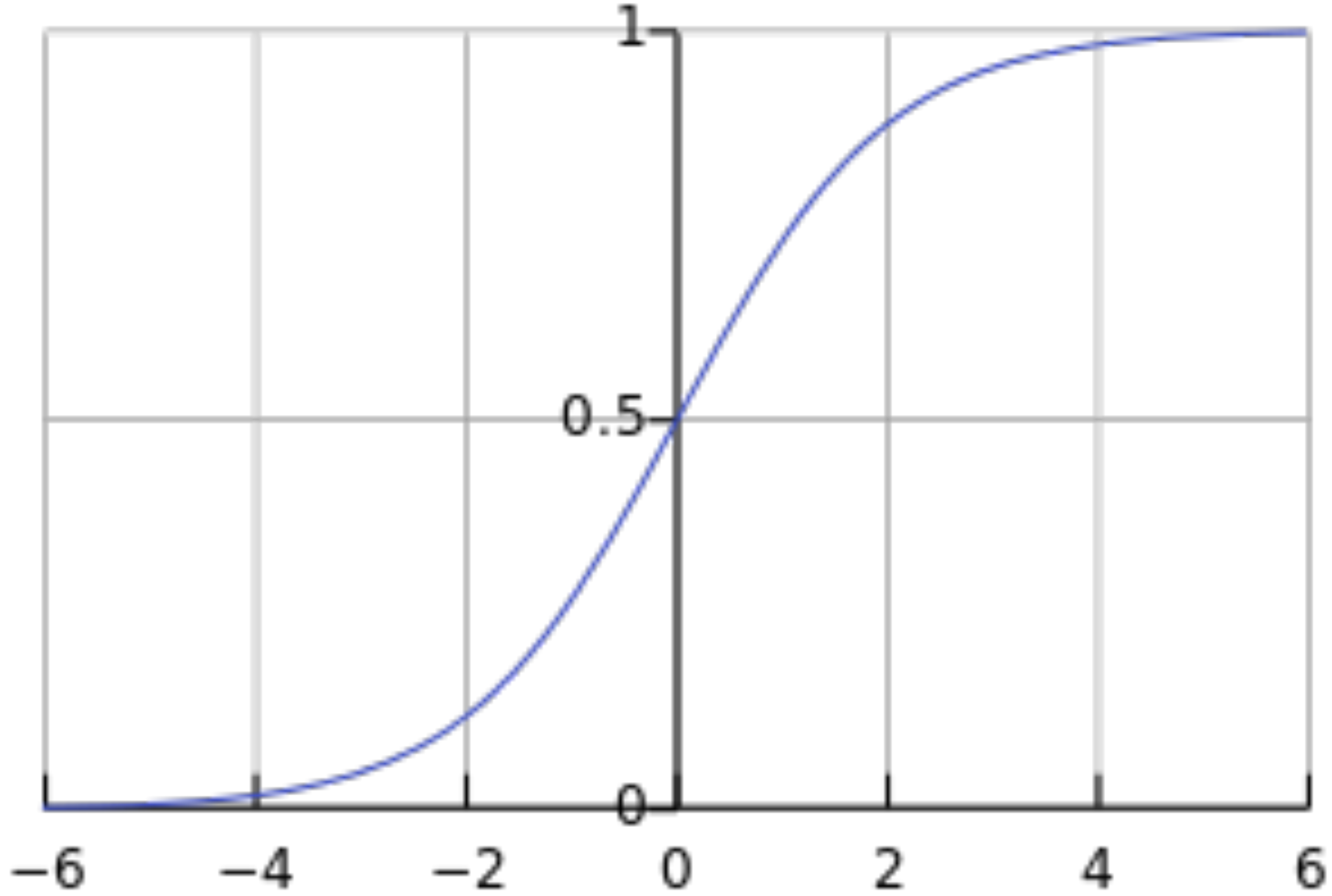
$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

What does this function look like?

What properties does it have?

Logistic Function

$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$



Logistic Function

- logistic function $P(z) : \mathcal{R} \rightarrow [0, 1]$

Logistic Function

- logistic function $P(z) : \mathcal{R} \rightarrow [0, 1]$
- decision boundary is dot product = 0 (2 class)

Logistic Function

- logistic function $P(z) : \mathcal{R} \rightarrow [0, 1]$

- decision boundary is dot product = 0 (2 class)

- comes from linear log odds $\log \frac{P(x)}{1 - P(x)} = \sum_{i=0}^{|X|} \beta_i x_i$

NB vs. LogReg

- Both compute the dot product

- **NB**: sum of log probs; **LogReg**: logistic fun.

Learning Weights

- **NB**: learn conditional probabilities separately via **counting**

- **LogReg**: learn weights **jointly**

Learning Weights

- given: a set of **feature vectors** and **labels**
- goal: learn the weights.

Learning Weights

$$
\begin{array}{ccccc}
x_{00} & x_{01} & \cdots & x_{0m} & y_0 \\
x_{10} & x_{11} & \cdots & x_{1m} & y_1 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
x_{n0} & x_{n1} & \cdots & x_{nm} & y_n
\end{array}
$$

n examples; xs - features; ys - class

Learning Weights

We know:

$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

Learning Weights

So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

$$\beta^{MLE} = \arg\max_{\beta} \log P(y_0, \ldots, y_n | \mathbf{x_0}, \ldots, \mathbf{x_n}; \beta)$$

Learning Weights

So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

$$\beta^{MLE} = \arg\max_{\beta} \log P(y_0, \ldots, y_n | \mathbf{x_0}, \ldots, \mathbf{x_n}; \beta)$$

$$= \arg\max_{\beta} \sum_{i=0}^{|X|} \log P(y_i | \mathbf{x_i}; \beta)$$

Learning the weights

Maximize the training set's (log-)likelihood?

$$\beta^{\text{MLE}} = \arg\max_{\beta} \; \log p(y_1..y_n | x_1..x_n, \beta)$$

$$\log p(y_1..y_n | x_1..x_n, \beta) = \sum_i \log p(y_i | x_i, \beta) = \sum_i \log \begin{Bmatrix} p_i & \text{if } y_i = 1 \\ 1 - p_i & \text{if } y_i = 0 \end{Bmatrix}$$

where $p_i \equiv p(y_i = 1 | x, \beta)$

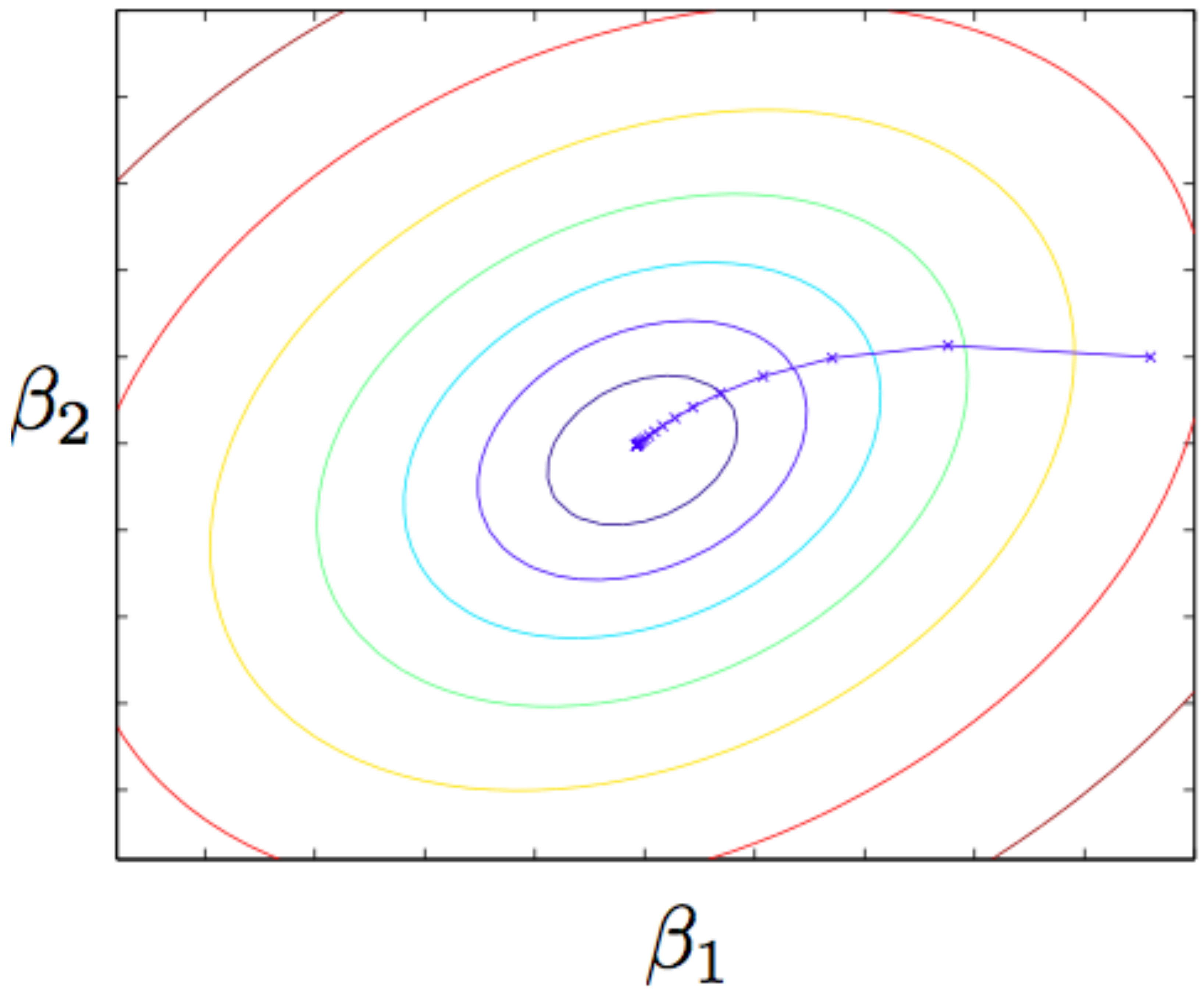
- No analytic form, unlike our counting-based multinomials in NB, n-gram LM's, or Model 1.
- Use *gradient ascent*: iteratively climb the log-likelihood surface, through the derivatives for each weight.
- Luckily, the derivatives turn out to look nice...

Gradient ascent

Loop while not converged (or as long as you can):

For all features ***j***, compute and add derivatives:

$$\beta_j^{(new)} = \beta_j^{(old)} + \eta \frac{\partial}{\partial \beta_j} \ell(\beta^{(old)})$$



ℓ: Training set log-likelihood

η: Step size (a.k.a. learning rate)

$\left(\frac{\partial \ell}{\partial \beta_1}, \ldots, \frac{\partial \ell}{\partial \beta_J}\right)$: Gradient vector (vector of per-element derivatives)

This is a generic optimization technique. Not specific to logistic regression! Finds the maximizer of any function where you can compute the gradient.

Perceptron learning algorithm (binary classif.)

Perceptron learning algorithm (binary classif.)

- Close cousin of MLE gradient ascent
- Loop through dataset many times. For each example:
  - Predict = $\text{argmax}_y\, p(y \mid \mathbf{x}, \boldsymbol{\beta})$
  - If y_{pred} != y_{gold}:
    - $\boldsymbol{\beta}$ += $(y_{gold} - y_{pred})\, \mathbf{x}$
- Does this converge and when?
  - If no errors, finishes. If not linearly separable: doesn't converge
- What does an update do?
  - e.g. for false negative: Increase weights for features in example

LogReg Exercise

features: (count "nigerian", count "prince", count "nigerian prince")

$\beta^{(0)}$ = (1.0, -3.0, 2.0) → **63% accuracy**

LogReg Exercise

features: (count “nigerian”, count “prince”, count “nigerian prince”)

$\beta^{(0)}$ = (1.0, -3.0, 2.0) → **63% accuracy**

$\beta^{(1)}$ = (0.5, -1.0, 3.0) → **75% accuracy**

LogReg Exercise

features: (count "nigerian", count "prince", count "nigerian prince")

$\beta^{(0)}$ = (1.0, -3.0, 2.0) → 63% accuracy

$\beta^{(1)}$ = (0.5, -1.0, 3.0) → 75% accuracy

$\beta^{(2)}$ = (-1.0, -1.0, 4.0) → 81% accuracy

Pros & Cons

- LogReg doesn't assume independence
  - better calibrated probabilities

- NB is faster to train; less likely to overfit

NB & Log Reg

- Both are linear models:

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

- Training is different:
  - NB: weights trained independently
  - LogReg: weights trained jointly

LogReg: Important Details!

- Overfitting / regularization
- Visualizing decision boundary / bias term
- Multiclass LogReg

You can use scikit-learn (python) to test it out!

Regularization

- Just like in language models, there's a danger of overfitting the training data. (For LM's, how did we combat this?)
- One method is <u>count thresholding</u>: throw out features that occur in < L documents (e.g. L=5). This is OK, and makes training faster, but not as good as....
- <u>*Regularized* logistic regression</u>: add a new term to penalize solutions with large weights. Controls the **bias/variance tradeoff**.

$$\beta^{\text{MLE}} = \arg\max_{\beta} \; [\log p(y_1..y_n | x_1..x_n, \beta)]$$

$$\beta^{\text{Regul}} = \arg\max_{\beta} \; \left[\log p(y_1..y_n | x_1..x_n, \beta) - \lambda \underbrace{\sum_j (\beta_j)^2}_{\text{“Quadratic penalty” or “L2 regularizer”: Squared distance from origin}} \right]$$

"Regularizer constant": Strength of penalty

"Quadratic penalty" or "L2 regularizer": Squared distance from origin

Visualizing a classifier in feature space

"Bias term"

Feature vector $x = (1,\ \text{count "happy"},\ \text{count "hello"}, \ldots)$

Weights/parameters $\beta =$

50% prob where

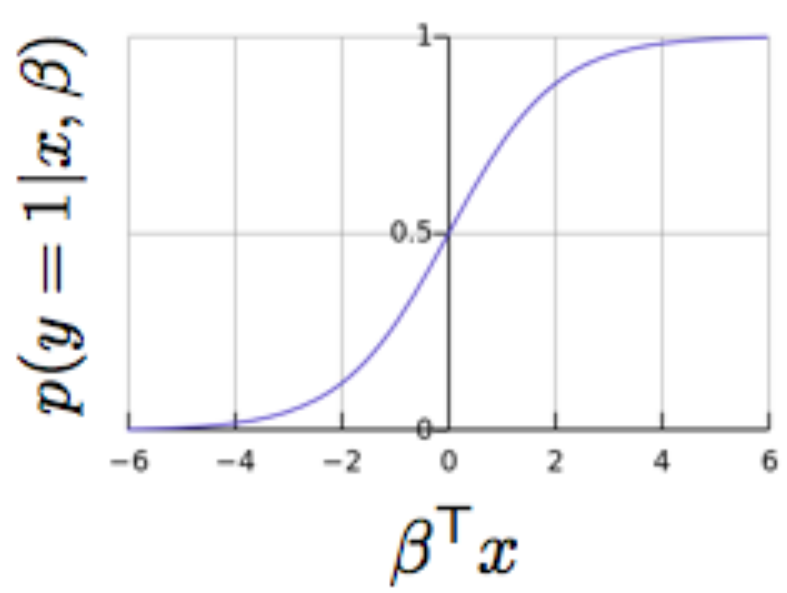
$\beta^{\mathsf{T}} x = 0$

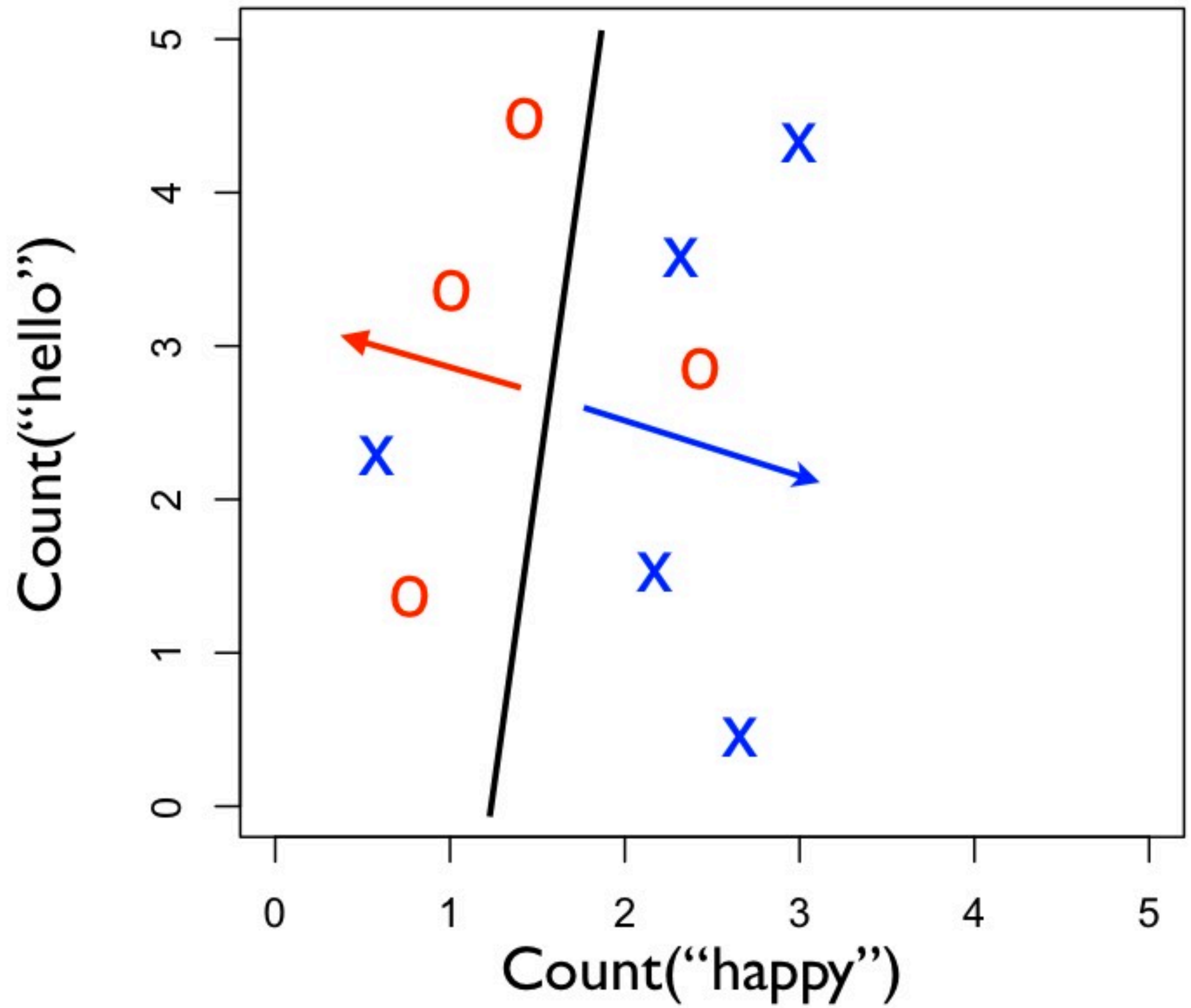
Predict y=1 when

$\beta^{\mathsf{T}} x > 0$

Predict y=0 when

$\beta^{\mathsf{T}} x \leq 0$





Binary vs. Multiclass

- Binary logreg: let x be a feature vector for the doc, and y either 0 or 1

$$p(y=1|x,\beta) = \frac{\exp(\beta^\mathsf{T} x)}{1+\exp(\beta^\mathsf{T} x)}$$

β is a weight vector across the *x* features.

- Multiclass logistic regression, in "log-linear" form:
  Features are jointly of document ***and output class***

$$p(c|x) \;=\; \frac{1}{Z}\exp\left(\sum_i w_i f_i(c,x)\right)$$

w is a weight vector across the *x* features.

Multiclass log. reg.

$$p(c|x) \; = \; \frac{1}{Z}\exp\left(\sum_i w_i f_i(c,x)\right)$$

$$p(c|x) \; = \; \frac{\exp\left(\sum_{i=1}^{N} w_i f_i(c,x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=1}^{N} w_i f_i(c',x)\right)}$$

Multiclass log. reg.

$$
f_1(c,x) = \begin{cases} 1 & \text{if “great”} \in x \ \& \ c = + \\ 0 & \text{otherwise} \end{cases}
$$

$$
f_2(c,x) = \begin{cases} 1 & \text{if “second-rate”} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}
$$

$$
f_3(c,x) = \begin{cases} 1 & \text{if “no”} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}
$$

$$
f_4(c,x) = \begin{cases} 1 & \text{if “enjoy”} \in x \ \& \ c = - \\ 0 & \text{otherwise} \end{cases}
$$

$$
p(c|x) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(c,x)\right)
$$

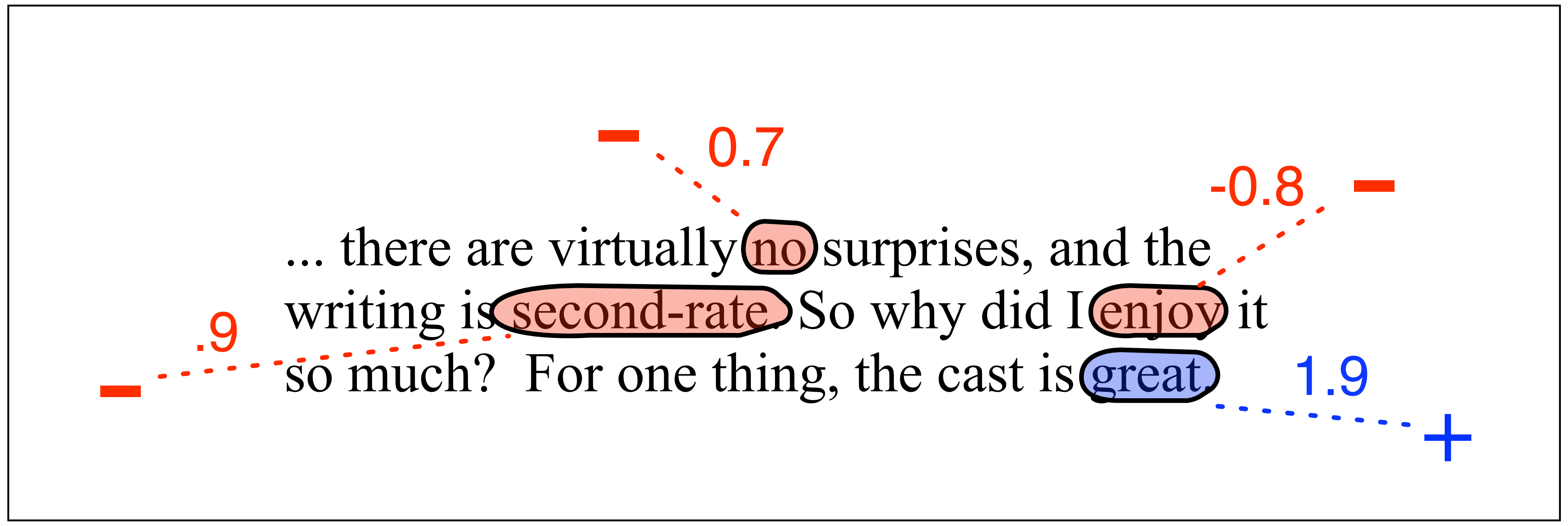


Figure 7.1 Some features and their weights for the positive and negative classes. Note the *negative* weight for *enjoy* meaning that it is evidence against the class negative $-$.