1. Naive Bayes

2. Logistic Regression Classifiers

CS 490A, Fall 2020

Applications of Natural Language Processing

https://people.cs.umass.edu/~brenocon/cs490a_f20/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

[Many slides from Ari Kobren]

Logistic regression

- a.k.a. "Maximum Entropy" a.k.a. "Log-Linear Classifier"
- Kinda like Naive Bayes, but:
  - No Bayes Rule. Just directly define a function for *p(y | text)*.
  - Learn all feature weights jointly
    - Correlated features don't "double count"
    - Better to "throw in lots of features"
  - Discriminative training: optimize for label predictions
    - *p(y | text)*, not *p(y, text)*
- Tends to work a bit better than NB - best baseline method
- Good off-the-shelf implementations: use scikit-learn
- Requires regularization (similar to NB pseudocount smoothing)
- Advanced: integrate with neural networks

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

f(d) → 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 several different "happy" emoticons
Count of "of the",
Length of document,
...
)

Typically these use feature templates:
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!
- Feature engineering: when you spend a lot of trying and testing new features. Very important!! This is a place to put linguistics in, or just common sense about your data.

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
```

[Slide: SLP3]

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)

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

bigram

- given **weights** (betas)

β = (-1.0, -1.0, 4.0)

Params of the model

Classification: LogReg (II)

- Compute the **dot product**

$$z = x_i^T \beta = \sum_{j=1}^{N_{feat}} \beta_j x_{i,j}$$

z is high if $y=1$ is likely

- Compute the **logistic function** for the label probability

$$P(y=1 \mid x) = g(z)$$

$g(z): \mathbb{R} \to [0,1]$

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

LogReg Exercise

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

x = (1, 1, 1)

β = (-1.0, -1.0, 4.0)

$P(y=1 \mid x) = g(x_1\beta_1 + x_2\beta_2 + x_3\beta_3)$

$= g(-1 - 1 + 4) = g(2) = \frac{1}{1+e^{-2}}$

.88

Classification: Dot Product

"score"

$$z = \sum_{j=1}^{\text{Nfeat}} \beta_j x_{ij}$$

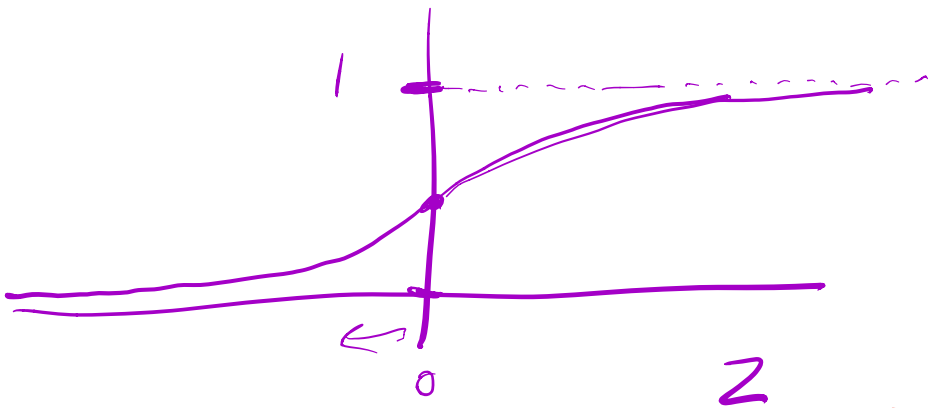
weighted sum of feature values

linear function

Why the **logistic function**?

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

$z = 0 \quad g = \frac{1}{1+e^0} = \frac{1}{2}$



"sigmoid"

S-shape

Outputs a prob

NB as Log-Linear Model

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

word counts $\vec{x} = (n_{dog}, n_{cat}, \ldots, n_{zebra})$

V

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

↳ word assoc w/ label

$$\beta_w^{NB} = \log \frac{p(w|y=1)}{p(w|y=0)}$$

NB as Log-Linear Model

In both NB and LogReg

we **compute the dot product!**

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**

Training Time!

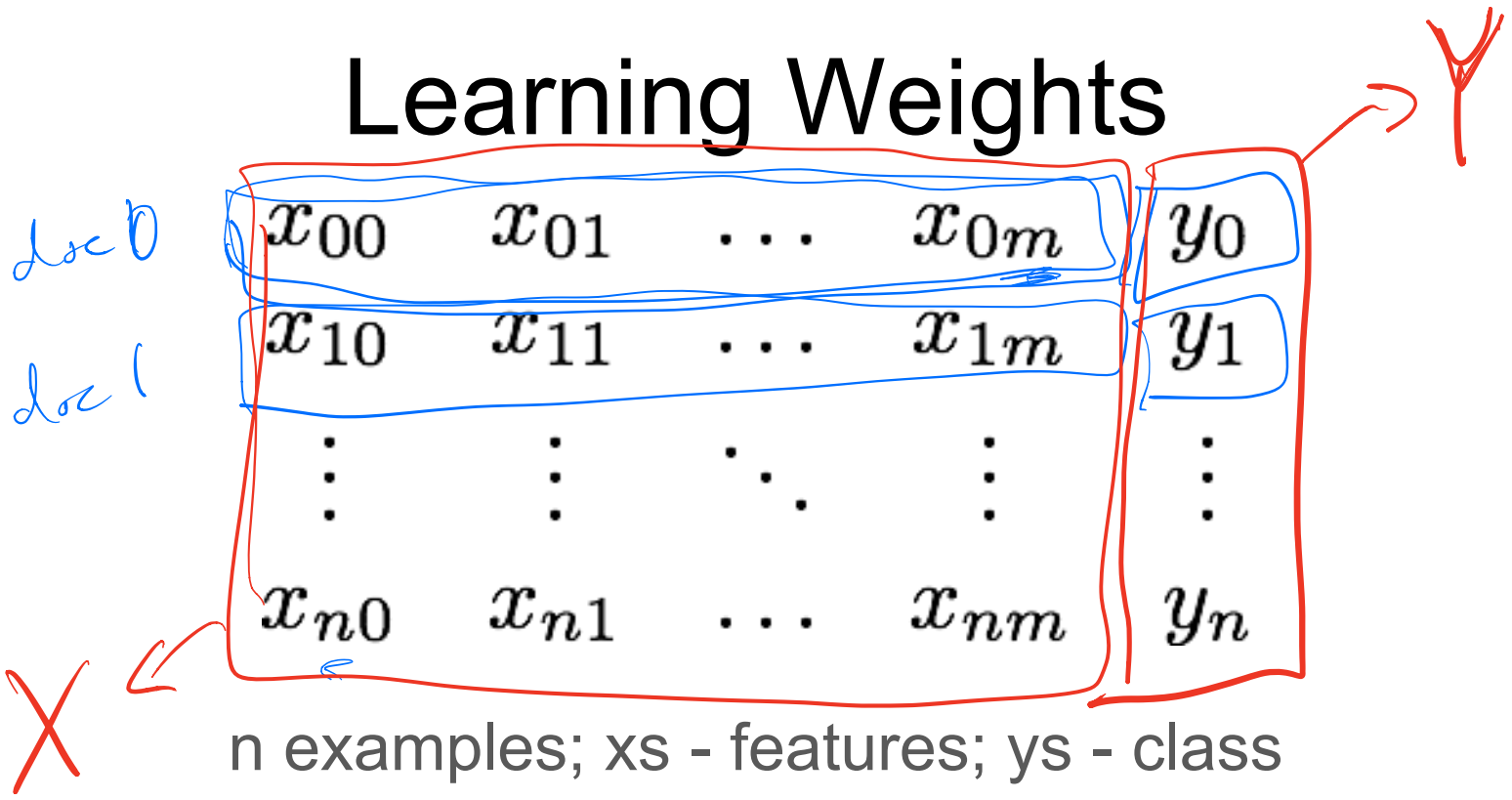
Learning Weights

- given: a set of **feature vectors** and **labels**

$\vec{x}_1, \vec{x}_2 \cdots \vec{x}_n$ $\quad y_1, y_2 \cdots y_n$

- goal: learn the weights. $\Rightarrow \vec{\beta}$

Learning Weights



n examples; xs - features; ys - class

Learning Weights

We know:

$$g(z) = \frac{1}{1 + e^{-z}} \qquad P(y = 1 \mid x) = g\left(\sum_{j=1}^{\text{Nfeat}} \beta_j x_{ij}\right)$$

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

Q: all $\vec{\beta} = \vec{0}$ $p(y=1|x;\beta) = \frac{1}{1+e^0}$ $z = \beta^T x = 0$

≥ 0.5

Learning Weights

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

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

$= \arg\max_{\beta} \sum_{i=1}^{n} \log P(y_i | \vec{x}_i, \vec{\beta})$

gold std. label

(# "n...", # "pr", # "u pronc")

$\beta = (2, 3, -2)$ $\Rightarrow$ 60% acc

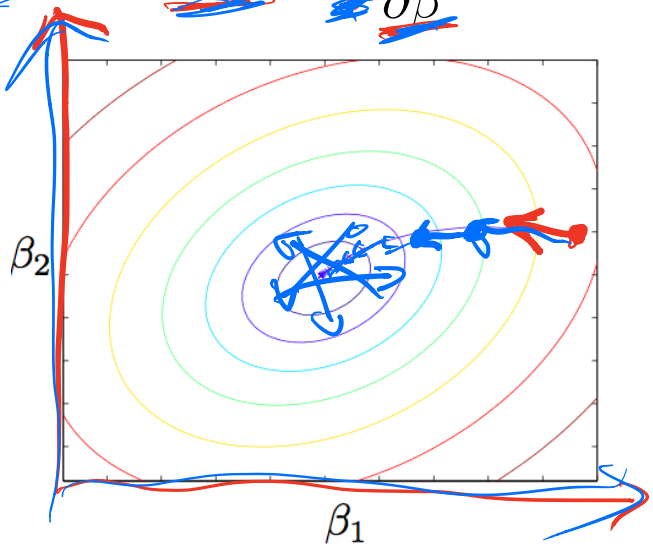
$\beta = (1, 1.3, .5)$ $\Rightarrow$ 73% acc

$\beta = (-1, -2, 6.1)$ $\Rightarrow$ 78% acc

Gradient ascent learning

- Follow direction of *steepest ascent*. Iterate

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



$$\ell(\beta) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x_i};\beta)$$

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.

Pros & Cons

- LogReg doesn't assume independence
  - better calibrated probabilities
  - allows for repetitive features
- 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 count thresholding: throw out features that occur in < L documents (e.g. L=5). This is OK, and makes training faster, but not as good as....
- *Regularized* logistic regression: add a new term to penalize solutions with large weights. Controls the **bias/variance tradeoff**.

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

$$\beta^{\text{Regul}} = \arg\max_{\beta} \; \left[\log p(y_1..y_n|x_1..x_n, \beta) - \lambda \underbrace{\sum_j (\beta_j)^2}\right]$$

"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

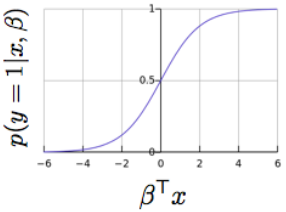
$\beta^T x = 0$

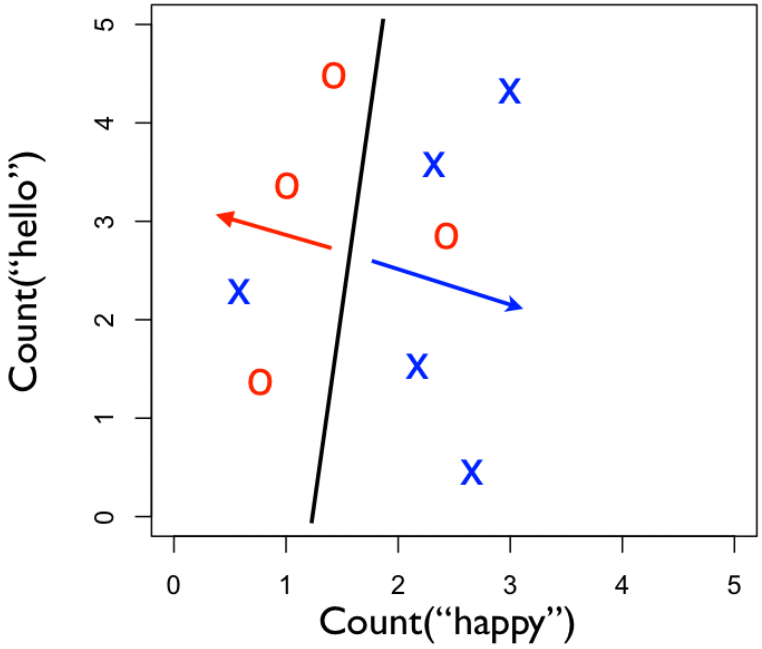
Predict y=1 when

$\beta^T x > 0$

Predict y=0 when

$\beta^T x \leq 0$





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: