Logistic Regression

September 17, 2015

Questions?

- From previous lecture?
- From HW?

Naive Bayes Recap

- What do you remember about classification with Naive Bayes?

Naive Bayes Recap

- What do you remember about classification with Naive Bayes?
- What statistics do you need to make a classification?

Naive Bayes: Bag of Words

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

Naive Bayes: Bag of Words

- Features statistically independent given class
- Examples of non-independent features?

Independence Assumption

- Correlated features -> **double counting**
- Can hurt classifier accuracy & calibration

Logistic Regression

- **(Log) Linear Model** - similar to Naive Bayes
- Doesn’t assume features are independent
- Correlated features don’t “double count”

Features! **Features!** **Features!**

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

$$f(d) \longrightarrow \mathbf{x}$$

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 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 for effective classifiers!! This is a place to put linguistics in.

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

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

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

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

NB as Log-Linear Model

In both NB and LogReg

we **compute the dot product!**

Logistic Function

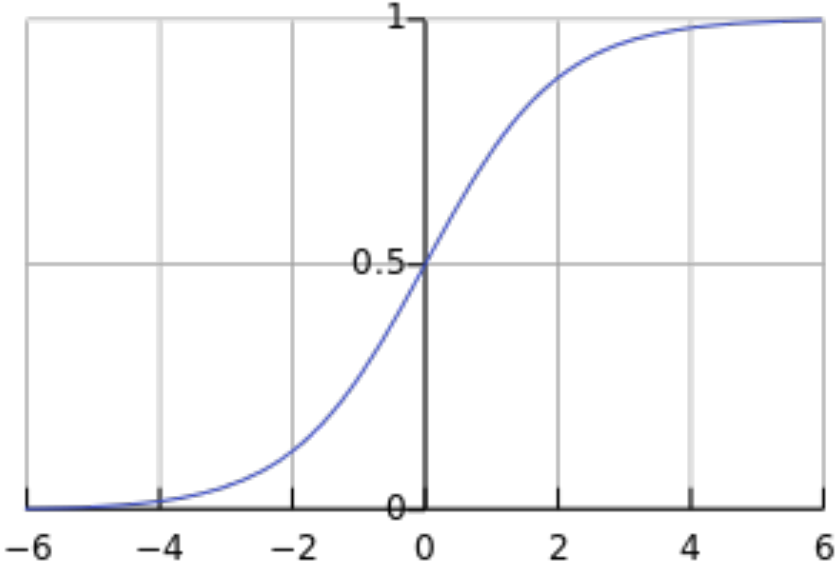
$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

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

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

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?

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

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

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

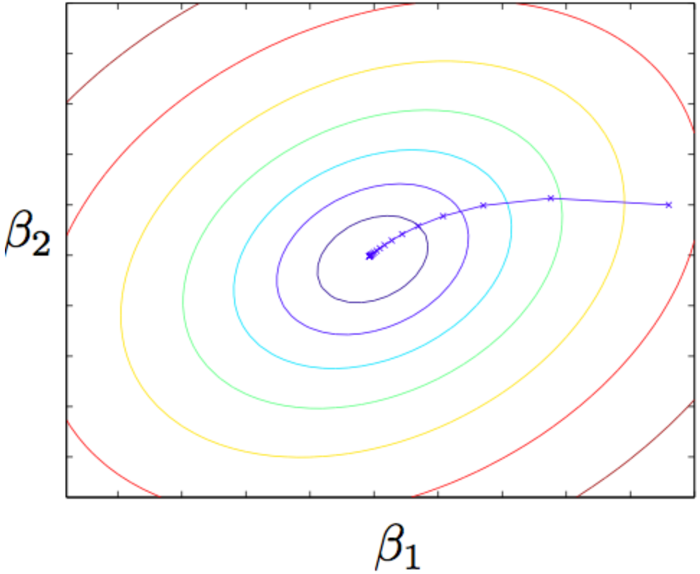
- No analytic form, unlike our counting-based multinomials in NB, n-gram LM's, or Model 1.
- Use *gradient ascent*: iteratively climb the log-likelihood surface, through the derivatives for each weight.
- Luckily, the derivatives turn out to look nice...

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.

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**.

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

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

"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

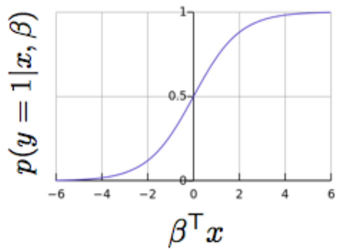
$$\beta^{\mathsf{T}} x = 0$$

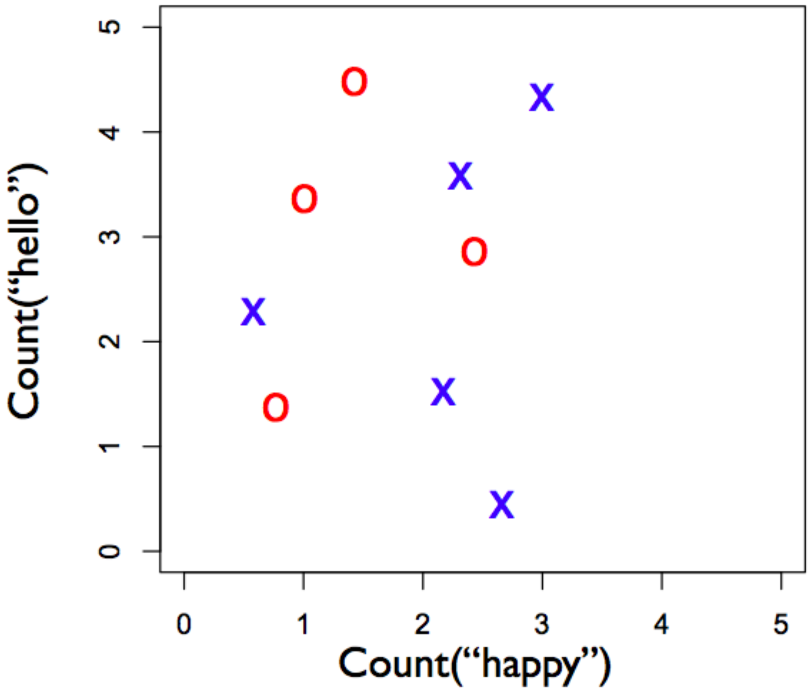
Predict y=1 when

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

Predict y=0 when

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





Binary vs Multiclass logreg

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

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

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

- Multiclass logreg: *y* is a categorical variable, attains one of several values in *Y*

  Each $\beta_{y'}$ is a weight vector across all *x* features.

$$p(y|x,\beta) = \frac{\exp(\beta_y^\mathsf{T} x)}{\sum_{y' \in \mathcal{Y}} \exp(\beta_{y'}^\mathsf{T} x)}$$