Logistic Regression for Text Classification

CS 485, Fall 2023
Applications of Natural Language Processing
https://people.cs.umass.edu/~brenocon/cs485_f23/

Brendan O'Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

[With slides from Ari Kobren and SLP3]

BOW linear model for text classif.

- Problem: classify doc *d* into one of *k*∈*1..K* classes

- Parameters: For each class *k*, and word type *w*, there is a *word weight*

$$\beta_{w,k}$$

$$\beta_{love,\,pos} = 3.8$$

- Representation: bag-of-words vector of doc *d*'s word counts

text

$$x = [x_1, \dots x_V]$$

$$x_{510} = \text{Count}(\text{"love"})$$

$$x_1 = \text{Count}(\text{"sky"})$$

$$x_2 = \text{Count}(\text{"the"})$$

- Prediction rule: choose class *y* with highest score

$$\hat{y} = \underset{y \in \{1..K\}}{\operatorname{argmax}} \underbrace{\left[\sum_{j=1}^{V} \beta_{j,y} x_j \right]}_{= (\beta_{\cdot,y})^T x}$$

Keyword count as linear model

- Problem: classify doc *d* into one of *k*∈*1..K* classes

- Parameters: For each class *k*, and word type *w*, there is a *word weight*

- Representation: bag-of-words vector of doc *d*'s word counts

$\vec{x}$

$$\beta_{w,POS} = \begin{cases} 1 & \text{if } w \in \text{PosLexicon} \\ 0 & \text{otherwise} \end{cases}$$

- Prediction rule: choose class *y* with highest score

scores:

$$z_{POS} = \sum_{j=1}^{V} (\beta_{j,POS}) x_j$$

= Num tokens in doc, that are in PosLex

$$z_{NEG} = \sum_j \beta_{j,NEG} x_j$$

$$\hat{y} = \text{argmax} [z_{POS}, z_{NEG}]$$

Naive Bayes as linear model

- Problem: classify doc *d* into one of *k*∈*1..K* classes

- Parameters: For each class *k*, and word type *w*, there is a *word weight*

learned from tr. data

$$\beta_{w,k} = \log P(w \mid y=k)$$

- Representation: bag-of-words vector of doc *d*'s word counts

$\vec{x}$ BOW

Also need bias term?

- Prediction rule: choose class *y* with highest score

$$\arg\max_y \left[P(y)\, P(x \mid y) \right] = \arg\max_y \left[\log P(y) + \log P(x \mid y) \right]$$

$$= \arg\max_y \log P(y) + \sum_{i=1}^{N_{tok}} \log P(w_i \mid y)$$

$$= \arg\max_y \log P(y) + \sum_{j=1}^{V} \underbrace{\mathrm{count}(j)}\, \log P(w=j \mid y)$$

Linear classification models

- The foundational model for machine learning-based NLP!

- Examples
  - The humble "keyword count" classifier (no ML)
  - Naive Bayes ("generative" ML)

- Today: **Logistic Regression**
  - probabilistic model directly geared for *prediction*
  - allows for *features*
  - used within more complex models (neural networks)

Motivation: feature engineering

- For Naive Bayes, we used counts of each word in the vocabulary (BOW representation). But why not also use....
  - Number of words from "CS485 Crowdsource Positive Lexicon"
  - ...from "CS485 Crowdsource Negative Lexicon" ... or another....
  - Phrases?
  - Words/phrases with negation markers?
  - Number of "!" occurrences?
  - or...?

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ 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	$\begin{cases} \cancel{1 \text{ if "!"} \in \text{doc}} \\ 0 \text{ otherwise} \end{cases}$	0
x_6	ln(word count of doc)	$\ln(66) = 4.19$

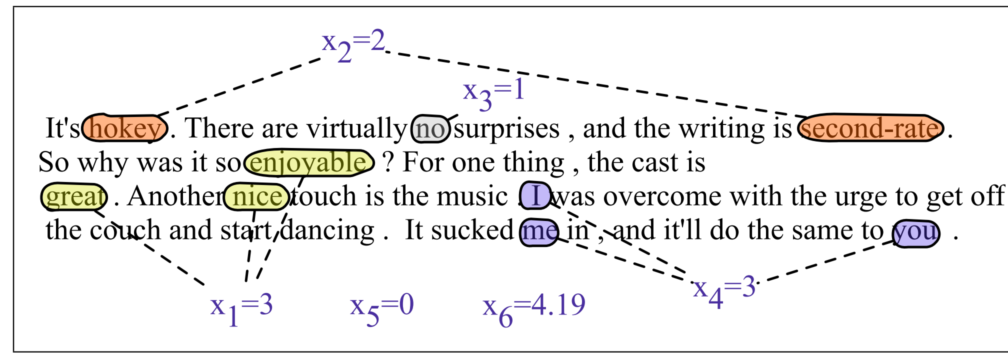


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

- Logistic regression can accommodate ***any arbitrary features***
- 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)

$$x_i = (\text{count "nigerian"}, \text{count "prince"}, \text{count "nigerian prince"})$$

- given **weights** (betas)

$$\beta = (-1.0, \quad -1.0, \quad 4.0)$$

Binary logs. regr.

Classification: LogReg (II)

- Compute the **dot product**

$$z = \beta^T x = \sum_{j=1}^{N_{feat}} \beta_j x_j$$

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

$$P(y=1 \mid x) = g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

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

$$z = -1000, \quad \frac{1}{1+e^{+1000}} \approx 0$$

$$z = \beta^T x = 1\cdot(-1) + 1\cdot(-1) + 1\cdot(4) = 2$$

LogReg Exercise

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

x = (1, 1, 1)

β = (-1.0, -1.0, 4.0)

P(y=1 | x) = $g(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-2}} = .88$

Classification: Dot Product

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

$\hat{y} = 1$ if $z \geq 0$

Why the **logistic function**?

Logistic Function

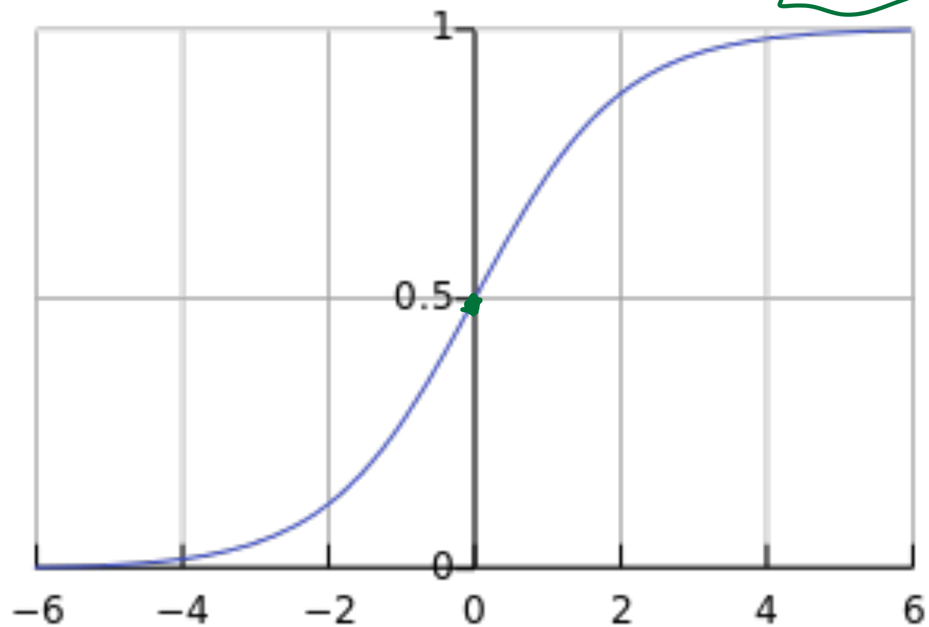
$$g(z) = \frac{e^z}{e^z+1} = \frac{1}{1+e^{-z}}$$

$P(y=1|x) \geq .5$

$\iff \hat{y} = 1$

$P(y=1|x) < .5$

$\iff \hat{y} = 0$

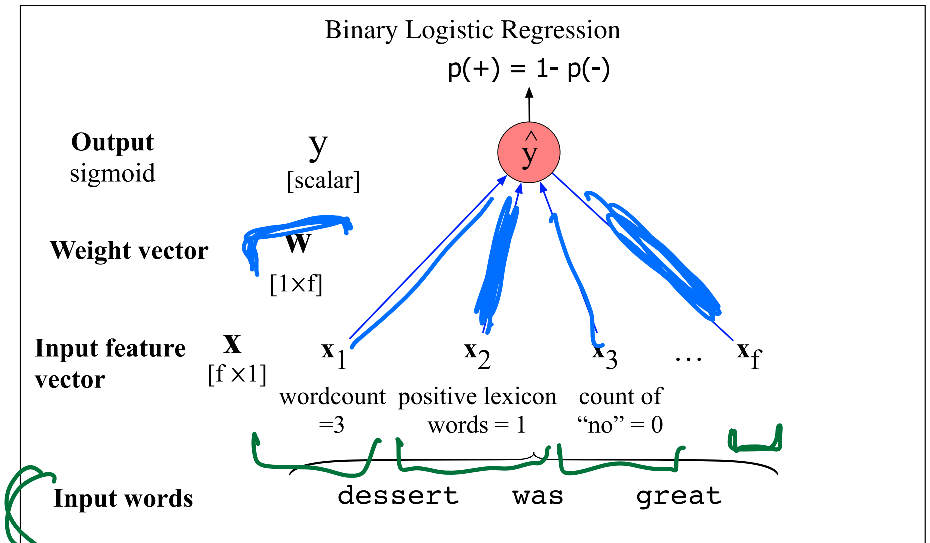


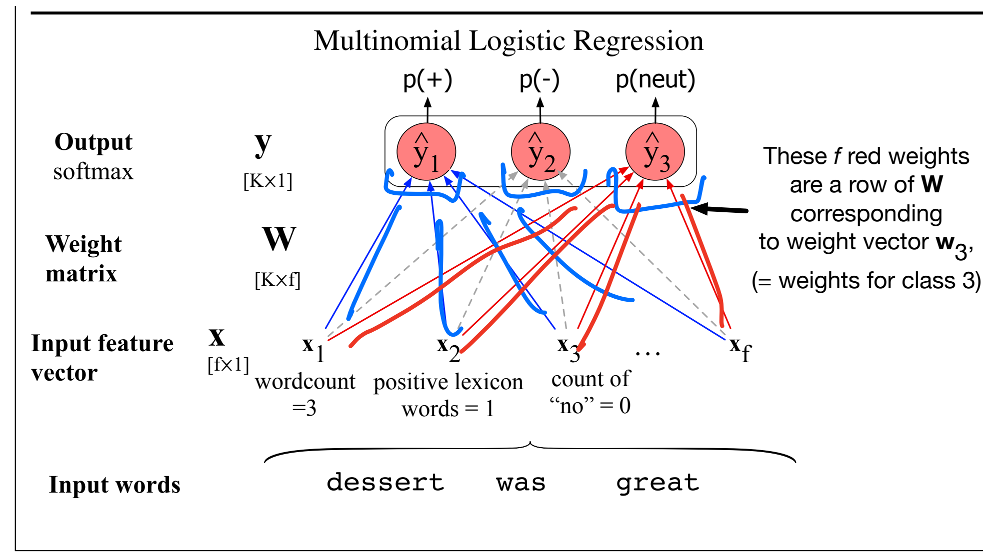
$\uparrow z$

$P(y=1|x)$

Multiclass Logistic Regression

- Generalize to K>2 classes
- Each class has its own weight vector (across all features; e.g. BOW counts)





Multiclass Logistic Regression

- Weight vector for each class

$$\beta_{\cdot,k} = [\underbrace{-1, 0}_{\text{Count(dog)}}, \underbrace{+1, 3}_{\text{Has "n"?}}, \dots]$$

- Prediction: dot product for each class

$$\forall k: \quad z_k = (\beta_{\cdot,k})^T x$$

- Predicted probabilities: apply the *softmax function* to normalize

$$P(y|x) = \left[\frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots e^{z_k}}, \frac{e^{z_2}}{\sum_j e^{z_j}}, \dots, \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}\right]$$

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

$$\begin{pmatrix} x_{00} & x_{01} & \dots & x_{0m} & y_0 \\ x_{10} & x_{11} & \dots & x_{1m} & y_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n0} & x_{n1} & \dots & x_{nm} & y_n \end{pmatrix} \Rightarrow \beta$$

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**

Learning Weights

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

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

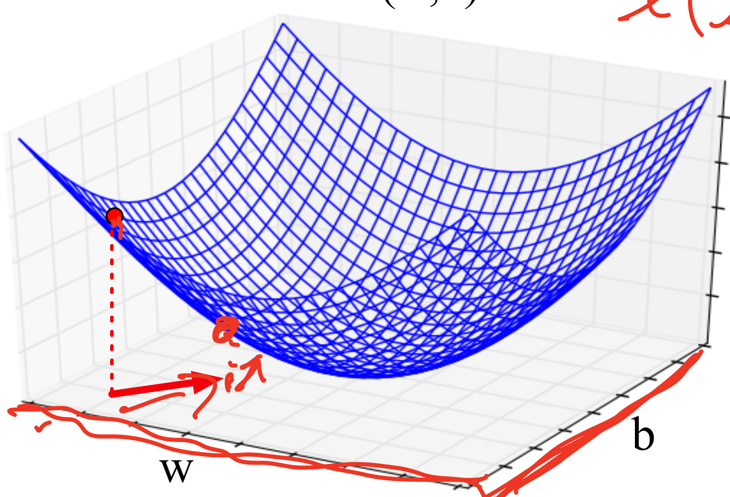
Gradient ascent/descent learning

$$\beta^{MLE} = \arg\max_{\beta} \underbrace{\log P(y_0, \ldots, y_n | \mathbf{x_0}, \ldots, \mathbf{x_n}; \beta)}_{\ell(\beta) = \sum_i \log P(y_i | x_i; \beta)}$$

- Follow direction of *steepest ascent*. Iterate: $\beta^{(new)} = \beta^{(old)} + \underset{\text{Step size}}{\eta} \frac{\partial \ell}{\partial \beta}$

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

Cost(w,b) $= -\ell(\beta)$



GD is a generic method for optimizing differentiable functions — widely used in machine learning!

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:

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

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

Overfitting and generalization

- Overfitting: your model performs overly optimistically on training set, but generalizes poorly to other data (even from same distribution)
- To diagnose: separate training set vs. test set.
- How did we regularize Naive Bayes and language modeling?

- For logistic regression: L2 regularization for training

Regularization tradeoffs

- No regularization <---------------> Very strong regularization

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

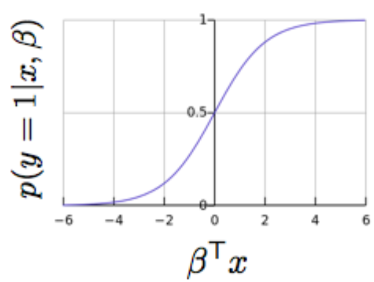
$\beta^\mathsf{T} x = 0$

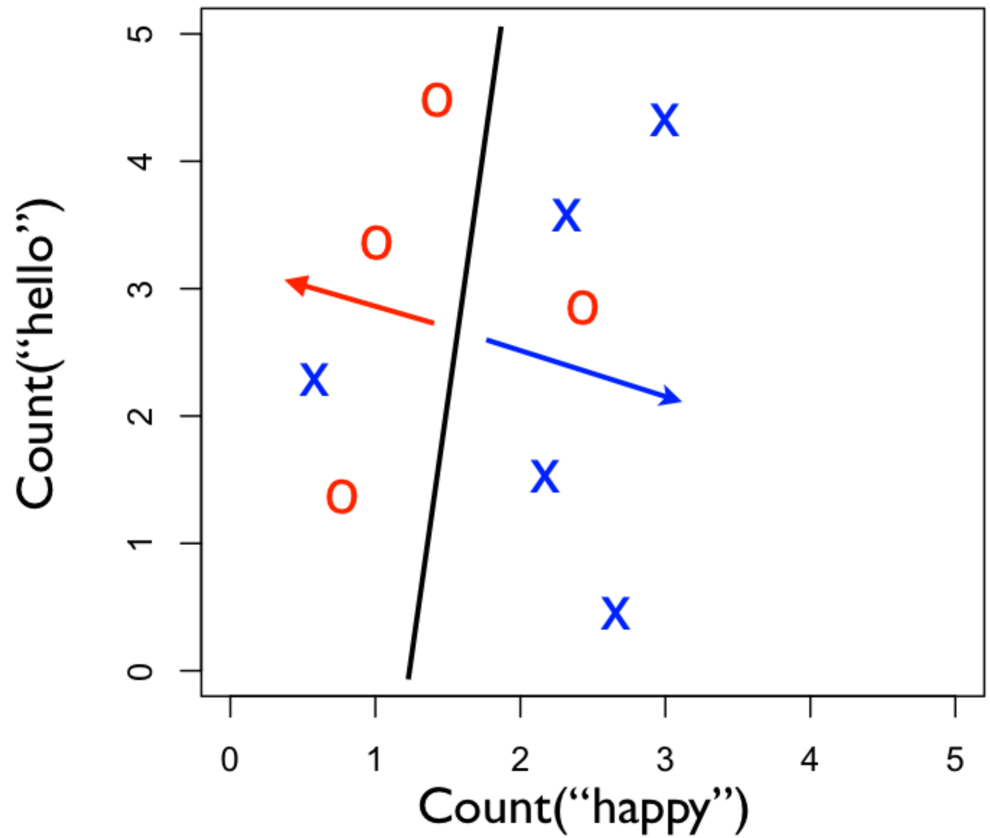
Predict y=1 when

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

Predict y=0 when

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





Logistic regression wrap-up

- Given you can extract features from your text, logistic regression is the best, easy-to-use, method
  - Logistic regression with BOW features is an excellent baseline method to try at first
  - Will be a foundation for more sophisticated models, later in course
- Always regularize your LR model
- We recommend using the implementation in scikit-learn
  - Useful: CountVectorizer to help make BOW count vectors

- Next: but where do the LABELS in supervised learning come from?