Logistic Regression for Text Classification

CS 485, Spring 2026
Applications of Natural Language Processing

Katrin Erk
College of Information and Computer Sciences
University of Massachusetts Amherst

[With slides from Brendan O'Connor, Ari Kobren and SLP3]

Linear models for text classification

- Problem: classify data point (document) into one of K classes

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

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

- Prediction rule: choose class y with highest score

Keyword count as a linear model

- Problem: classify data point (document) into one of K classes
- Parameters: For each class *c*, and word type *w*, there is a *word weight*
- Representation: bag-of-words vector of doc d's word counts
- Prediction rule: choose class y with highest score

Keyword count as a linear model

- Problem: classify data point (document) into one of K classes

- Parameters: For each class *c*, and word type *w*, there is a *word weight*
  Wt(w, c) = 1 if w is in the lexicon for c

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

- Prediction rule: choose class y with highest score
  Sum_c = sum of words in the documents that belong to the c lexicon
  Choose class c with the highest sum

Naive Bayes as a linear model

- Problem: classify data point (document) into one of K classes

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

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

- Prediction rule: choose class y with highest score

Naive Bayes as a linear model

- Problem: classify data point (document) into one of K classes

- Parameters: For each class *c*, and word type *w*, there is a *word weight*
  log P(w | c)

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

- Prediction rule: choose class y with highest score
  $P(c \mid d) \sim \sum_{w \text{ in } d} \log P(w \mid c) + \log P(c)$
  choose argmax_c P(c | d)

Logistic Regression as a linear model

- Problem: classify data point (document) into one of K classes

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

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

- Prediction rule: choose class y with highest score

This is what we'll discuss in the next two sessions

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**
  - a linear classification model, trained to be good at *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 "COMPSCI485 Crowdsource Positive/ Negative Lexicon"
  - Phrases?
  - Words/phrases with negation markers?
  - Number of "!" occurrences?
  - Or other features…?

- NB tends to work poorly when there are many potentially repetitive features (why?)

Features!

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	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	ln(word count of doc)	$\ln(66) = 4.19$

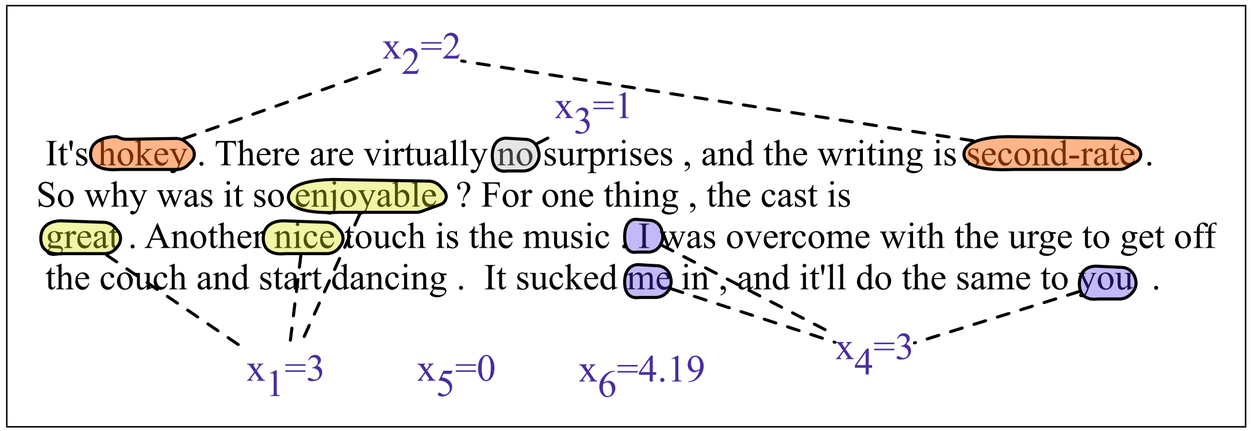


Figure 5.2 A sample mini test document showing the extracted features in the vector x.

- Logistic regression can accommodate ***any arbitrary features***
- Feature engineering: when you spend a lot of trying and testing new features. 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 the following punctuation:

didn’t like this movie , but I

didn’t NOT_like NOT_this NOT_movie , but I

Slide: SLP3

Logistic regression

- Also called: Maximum Entropy model, log-linear classifier
- Similar to Naive Bayes but
  - Learn all feature weights at the same time
  - Weights do not come directly from relative frequencies but are adapted to provide the best possible predictions
  - Also, correlated features don't "double count", this is why we can throw in as many features as we like
- Tends to work a bit better than Naive Bayes: best baseline method
- Good off-the-shelf implementations: use scikit-learn
- Requires regularization (similar to NB pseudocount smoothing)
- Preview: This method will become important again later as a "building block" of neural networks

Logistic regression classifier

- Compute features: x's
  x_i = < count("nigerian"), count("prince"), count("nigerian prince")>
- Use weights: β's
  < -1.0, -1.0, 4.0>

positive weight -> points towards non-spam
negative weight -> points towards spam

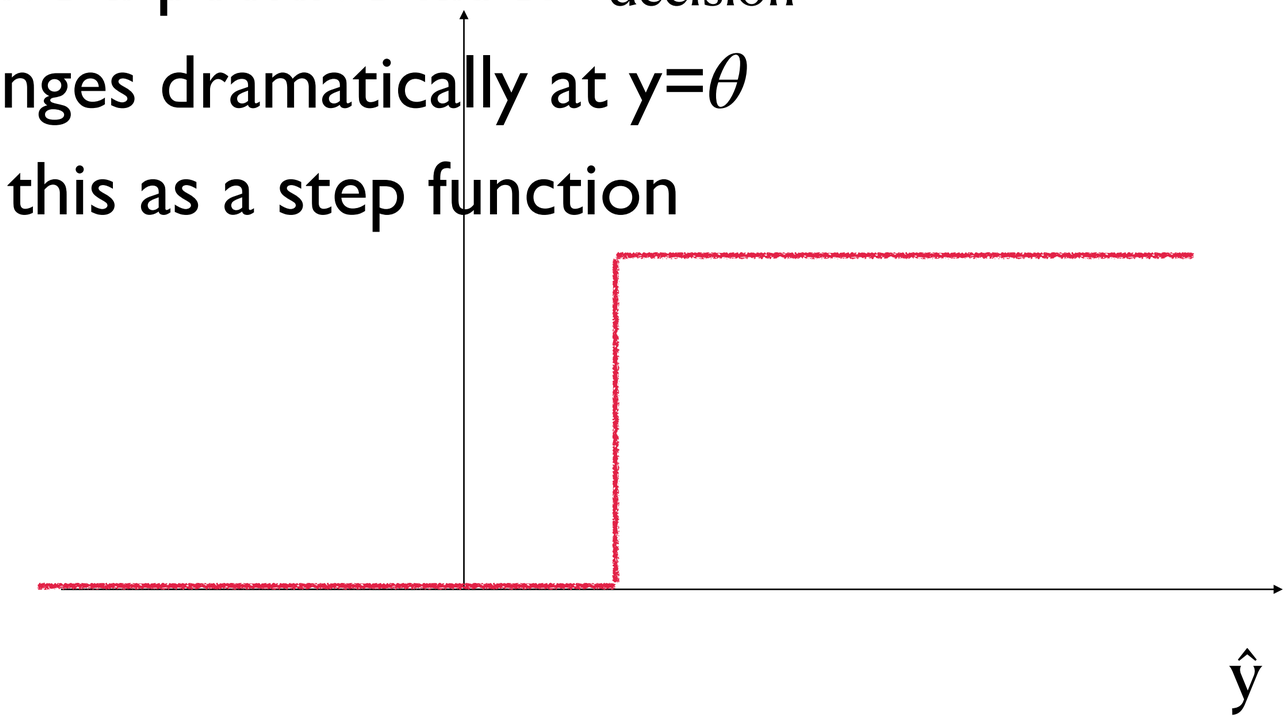
Logistic regression classifier

- Compute dot product: multiply each feature by its weight
  count("nigerian") * 1.0 + count("prince") * 1.0 +
  count("nigerian prince") * -4.0

- Then what do we do with that score?
- Simplest option would be: Set threshold θ. If sum is below that threshold, say "spam"
- What we do instead: **compute the logistic function**

What a hard threshold would be like

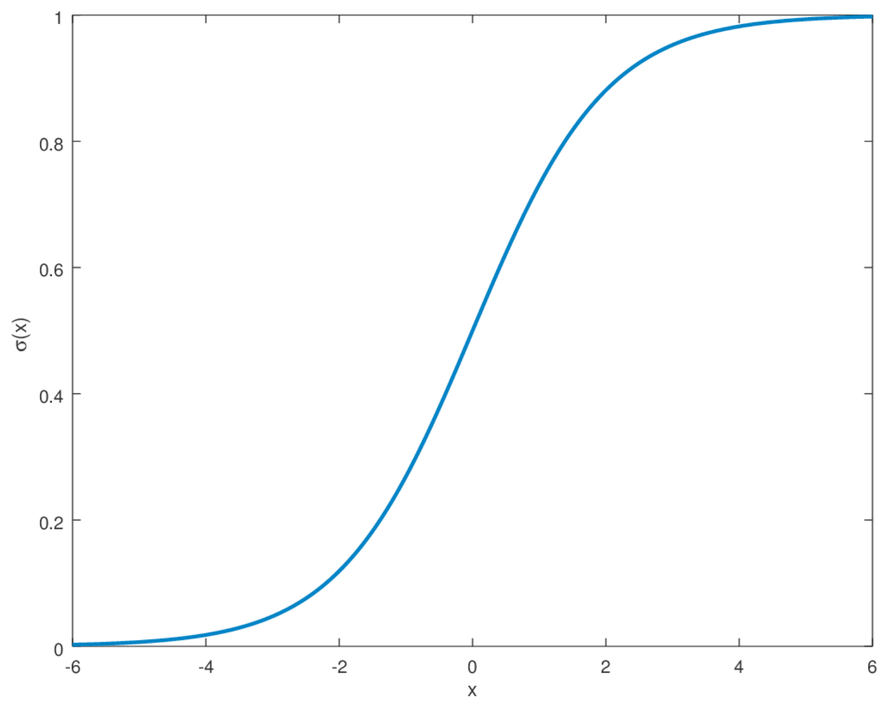
- Preliminary formulation: $y = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$
- If y > θ, say we have a positive label
- Our decision changes dramatically at y=θ
- We can describe this as a step function



Logistic regression classifier: logistic (sigmoid) function

- Instead of using a step function, use a smooth function
- The **logistic or sigmoid ("s-like") function** as a soft step function
- Defined as: $y = \sigma(z) = \dfrac{e^z}{e^z + 1}$

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



Logistic regression

- Aim: predict probability of y being 1 for a given feature vector x, versus 0

- We need probabilities to sum up to one, so:
- $P(y = 1) = \sigma(z) = \sigma(\beta \mathbf{x})$
- $P(y = 0) = 1 - \sigma(z)$

- Decision boundary is at 0.5

Logistic regression exercise

- Features: count(nigerian), count(prince), count(nigerian prince)

x = <. 1, 1, 1>

- Weights:

β = < -1.0, -1.0, 4.0>

- P(y=1 | x) =

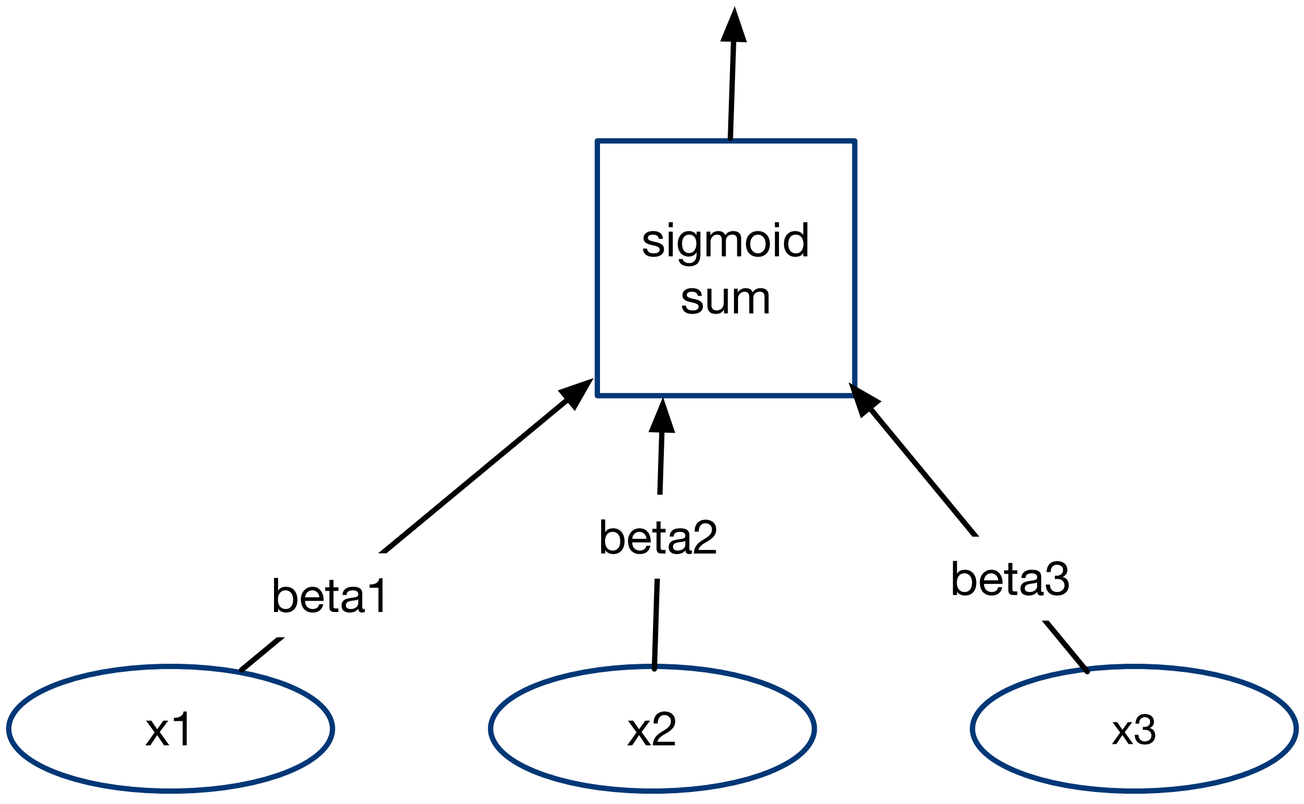
Logistic regression as a linear model

- Weighted sum of features: Dot product

$$z = \sum_{i=1}^{Numfeatures} \beta_i x_i$$

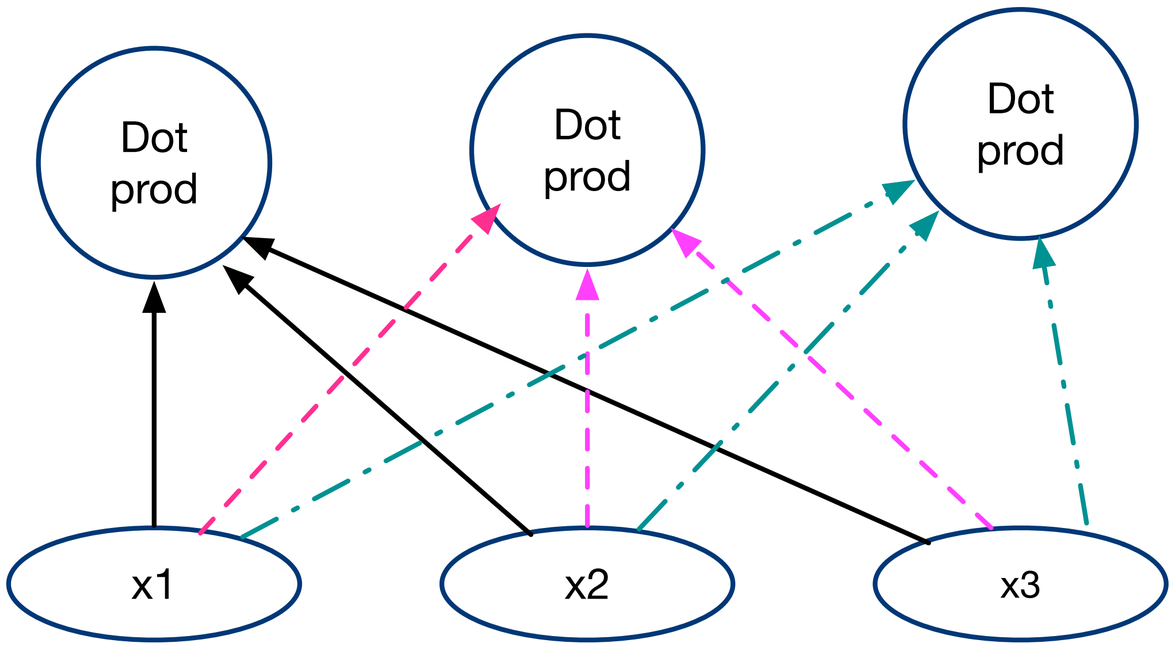
- Decision procedure: compute logistic function / sigmoid to predict P(1 | x)

Logistic regression model, different visualization



Multiclass logistic regression

Generalize to more than 2 classes:



Each class has its own weight set (picture: black, magenta, green)

Multiclass logistic regression

- Each class has its own set of weights

- Prediction: dot product for each class
  $\beta_1^c w_1 + \beta_2^c w_2 + \ldots + \beta_n^c w_n$

- Normalizing to obtain probabilities for each class: softmax function

Why the softmax function?

Naive Bayes as log-linear model

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

Naive Bayes as log-linear model

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

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

Naive Bayes as log-linear model

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

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

$$\log \left[P(\text{spam} \mid D)\right] \propto \log \left[P(\text{spam})\right] + \sum_{w_i \in D} x_i \log \left[P(w_i \mid \text{spam})\right]$$

Naive Bayes as log-linear model

$$P(\text{spam} \mid D) = \frac{1}{Z} P(\text{spam}) \cdot \prod_{w_i \in D} P(w_i \mid \text{spam})$$

$$P(\text{spam} \mid D) = \frac{1}{Z} P(\text{spam}) \cdot \prod_{w_i \in \text{Vocab}} P(w_i \mid \text{spam})^{x_i}$$

$$\log \Big[P(\text{spam} \mid D) \Big] \propto \log \Big[P(\text{spam}) \Big] + \sum_{w_i \in D} x_i \log \Big[P(w_i \mid \text{spam}) \Big] - \log Z$$

Naive Bayes as a log-linear model

In both Naive Bayes and logistic regression
we compute the dot product.

Naive Bayes versus logistic regression

- Both compute the dot product

- Naive Bayes: sum of log probabilities
  Logistic regression: logistic function

Learning weights

- Naive Bayes: learn conditional probabilities separately via counting

- Logistic regression: learn weights jointly

Learning weights

- Given training data: a set of feature vectors and labels

- Goal: learn the weights

Learning weights: feature vector and gold label for each datapoint

x_{00}	x_{01}	…	x_{0m}	y_0
x_{10}	x_{11}	…	x_{1m}	y_1
…	…	…	…	…
x_{n0}	x_{n1}	…	x_{nm}	y_m

n datapoints, m features each. x's: features. y's: class

Learning weights

We know:

Logistic function $g(z) = \dfrac{1}{1 + e^{-z}}$

$$P(y_i = 1 \mid x_i) = g\left(\sum_{j=1}^{\text{Nfeatures}} \beta_j x_{ij}\right)$$

So let's try to maximize the probability of the whole dataset:
maximum likelihood estimation

Learning weights

So let's try to maximize the probability of the whole dataset:
maximum likelihood estimation

$$\beta^{MLE} = \textbf{argmax}_{\beta} \log P(y_0, \ldots, y_n \mid x_0, \ldots x_n; \beta)$$

Learning weights

So let's try to maximize the probability of the whole dataset:
maximum likelihood estimation

$$\beta^{MLE} = \textbf{argmax}_{\beta} \log P(y_0, \ldots, y_n \mid x_0, \ldots x_n; \beta)$$

Learning weights

So let's try to maximize the probability of the whole dataset:
maximum likelihood estimation

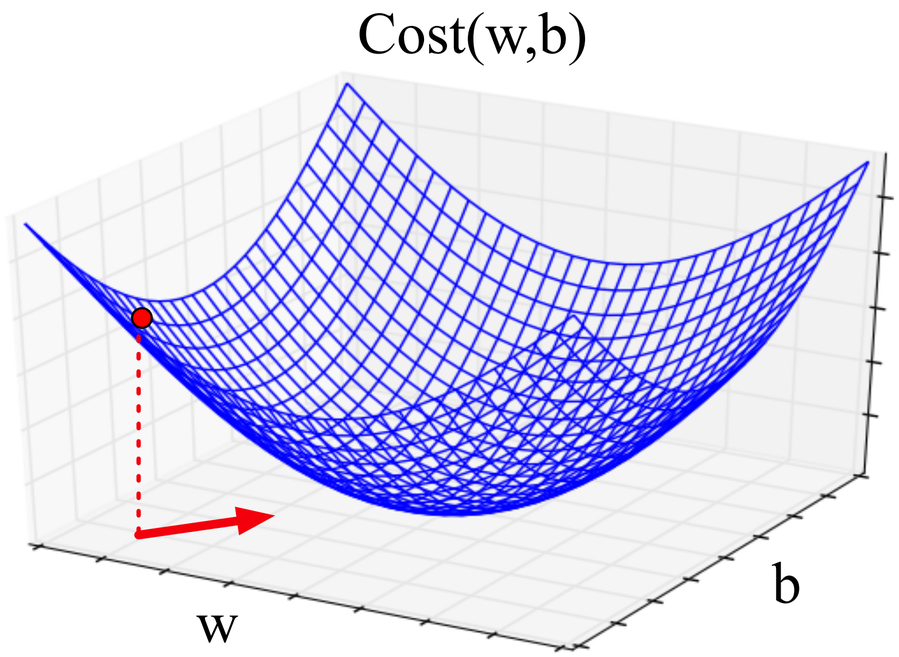
$$\beta^{MLE} = \textbf{argmax}_{\beta} \log P(y_0, \ldots, y_n \mid x_0, \ldots x_n; \beta)$$

$$= \textbf{argmax}_{\beta} \sum_{i=0}^{n} \log P(y_i \mid \mathbf{x}_i; \beta)$$

Gradient ascent/descent learning

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

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



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

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

Gradient descent in action

- One-dimensional demo: https://uclaacm.github.io/gradient-descent-visualiser/
- Three-dimensional demo: https://blog.skz.dev/gradient-descent

Comparing Naive Bayes and Logistic Regression

- Logistic regression does not assume independence:
  Better calibrated probabilities

- Naive Bayes is faster to train, and less likely to overfit

Comparing Naive Bayes and Logistic Regression

- Both are linear models: $z_i = \sum_{j=1}^{Nfeatures} \beta_j x_{ij}$

- Training is different:
  - Naive Bayes: weights trained independently
  - Logistic regression: 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.
- For logistic regression: L2 regularization for training

Regularization

- One method is **count thresholding**: Throw out any feature that appears in fewer than T documents, for example T=5. This is okay, and makes training faster, but not as good as L2 regularization
- Regularized logistic regression: Penalize solutions with large weights. This discourages overly complex models and makes the model fit better to unseen data.

  Regularization term: $\lambda \cdot \left(\beta_1^2 + \ldots + \beta_m^2\right)$

Regularization

- Regularization term: $\lambda \cdot \left(\beta_1^2 + \ldots + \beta_m^2\right)$
- Which beta's do we want to learn?

$$\beta^{\text{MLE}} = \text{argmax}_\beta P(y_1, \ldots, y_n \mid x_1, \ldots, x_n, \beta)$$

$$\beta^{\text{REGUL}} = \text{argmax}_\beta \left(P(y_1, \ldots, y_n \mid x_1, \ldots, x_n, \beta) - \lambda \underbrace{\left(\beta_1^2 + \ldots + \beta_m^2\right)} \right)$$

lambda: strength of penalty

L2 regularizer, quadratic penalty: penalizes large beta's

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

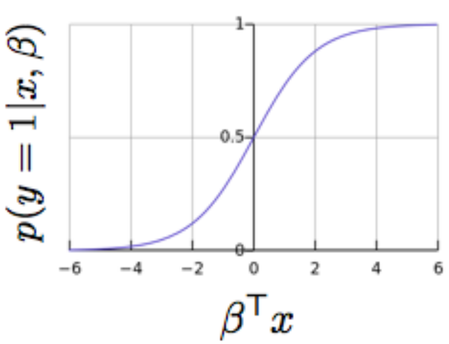
$\beta^T x = 0$

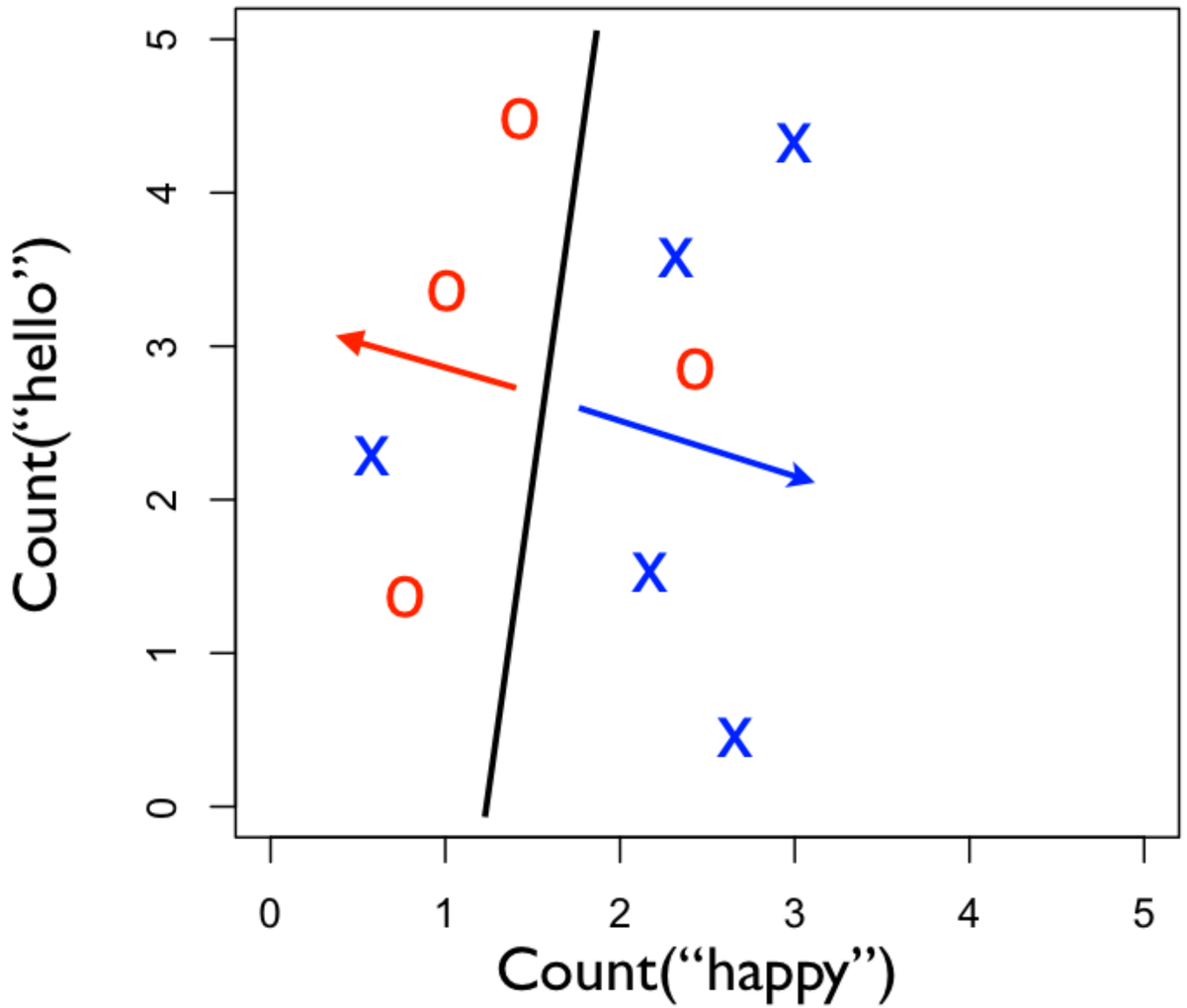
Predict y=1 when

$\beta^T x > 0$

Predict y=0 when

$\beta^T x \leq 0$





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?