Logistic Regression

Some slides adapted from Dan Jurafsky and Brendan O’Connor
Naïve Bayes Recap

• Bag of words (order independent)

• Features are assumed independent given class

\[ P(x_1, \ldots, x_n | c) = P(x_1 | c) \ldots P(x_n | c) \]

Q: Is this really true?
The problem with assuming conditional independence

• Correlated features -> double counting evidence
  – Parameters are estimated independently

• This can hurt classifier accuracy and calibration
Logistic Regression

- (Log) Linear Model – similar to Naïve Bayes
- Doesn’t assume features are independent
- Correlated features don’t “double count”
What are “Features”? 

• A feature function, $f$
  – Input: Document, $D$ (a string)
  – Output: Feature Vector, $X$
What are “Features”? 

\[ f(d) = \begin{pmatrix} 
\text{count(“boring”)} \\
\text{count(“not boring”)} \\
\text{length of document} \\
\text{author of document} \\
\vdots
\end{pmatrix} \]

Doesn’t have to be just “bag of words”
Feature Templates

• Typically “feature templates” are used to generate many features at once

• For each word:
  – ${w}_count
  – ${w}_lowercase
  – ${w}_with_NOT_before_count
Logistic Regression: Example

• Compute Features:

\[ f(d_i) = x_i = \begin{pmatrix} \text{count(“nigerian”)} \\ \text{count(“prince”)} \\ \text{count(“nigerian prince”)} \end{pmatrix} \]

• Assume we are given some weights:

\[ w = \begin{pmatrix} -1.0 \\ -1.0 \\ 4.0 \end{pmatrix} \]
Logistic Regression: Example

- Compute Features
- We are given some weights
- Compute the dot product:

\[ z = \sum_{i=0}^{|X|} w_i x_i \]
Logistic Regression: Example

• Compute the dot product:

\[ z = \sum_{i=0}^{\mid X \mid} w_i x_i \]

• Compute the logistic function:

\[ P(\text{spam} \mid x) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}} \]
The Logistic function

\[ P(\text{spam}|x) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}} \]
The Dot Product

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

- Intuition: weighted sum of features
- All Linear models have this form
NAIVE BAYES

IS ALSO A LOG-LINEAR MODEL?
Naïve Bayes as a log-linear model

• Q: what are the features?

• Q: what are the weights?
Naïve Bayes as a Log-Linear Model

\[ P(\text{spam}|D) \propto P(\text{spam}) \prod_{w \in D} P(w|\text{spam}) \]

\[ P(\text{spam}|D) \propto P(\text{spam}) \prod_{w \in \text{Vocab}} P(w|\text{spam})^{x_i} \]

\[ \log P(\text{spam}|D) \propto \log P(\text{spam}) + \sum_{w \in \text{Vocab}} x_i \cdot \log P(w|\text{spam}) \]
Naïve Bayes as a Log-Linear Model

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

In both Naïve Bayes and Logistic Regression we compute the dot product!
NB vs. LR

• Both compute the dot product

• NB: sum of log probabilities

• LR: logistic function
NB vs. LR: Parameter Learning

• Naïve Bayes:
  – Learn conditional probabilities independently by counting

• Logistic Regression:
  – Learn weights jointly
LR: Learning Weights

- Given: a set of feature vectors and labels
- Goal: learn the weights
Learning Weights

\[
\begin{bmatrix}
  x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\
  x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  x_{d1} & x_{d2} & x_{d3} & \cdots & x_{dn}
\end{bmatrix}
\]

\[
\begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n
\end{pmatrix}
\]
Q: what parameters should we choose?

• What is the right value for the weights?

• Maximum Likelihood Principle:
  – Pick the parameters that maximize the probability of the data
Maximum Likelihood Estimation

\[ w_{\text{MLE}} = \arg \max_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w) \]

\[ = \arg \max_w \sum_i \log P(y_i | x_i; w) \]

\[ = \arg \max_w \sum_i \log \begin{cases} 
  p_i, & \text{if } y_i = 1 \\
  1 - p_i, & \text{if } y_i = 0 
\end{cases} \]

\[ = \arg \max_w \sum_i \log p_i^{\mathbb{I}(y_i=1)}(1 - p_i)^{\mathbb{I}(y_i=0)} \]
Maximum Likelihood Estimation

\[ = \arg\max_w \sum_i \log p_i^{\mathbb{I}(y_i=1)} (1 - p_i)^{\mathbb{I}(y_i=0)} \]

\[ = \arg\max_w \sum_i y_i \log p_i + (1 - y_i) \log(1 - p_i) \]
Maximum Likelihood Estimation

• Unfortunately there is no closed form solution
  – (like there was with naïve bayes)

• Solution:
  – Iteratively climb the log-likelihood surface through the derivatives for each weight

• Luckily, the derivatives turn out to be nice
Gradient ascent

Loop While not converged:

For all features \( j \), compute and add derivatives

\[
\omega_j^{\text{new}} = \omega_j^{\text{old}} + \eta \frac{\partial}{\partial \omega_j} \mathcal{L}(w)
\]

\( \mathcal{L}(w) \): Training set log-likelihood

\[
\left( \frac{\partial \mathcal{L}}{\partial \omega_1}, \frac{\partial \mathcal{L}}{\partial \omega_2}, \ldots, \frac{\partial \mathcal{L}}{\partial \omega_n} \right) : \text{Gradient vector}
\]
Gradient ascent
Gradient ascent
LR Gradient

\[
\frac{\partial L}{\partial w_j} = \sum_i (y_i - p_i) x_j
\]
Derivative of Sigmoid

\[
\frac{ds(x)}{dx} = \frac{1}{1 + e^{-x}}
\]

\[
= \left(\frac{1}{1 + e^{-x}}\right)^2 \frac{d}{dx}(1 + e^{-x})
\]

\[
= \left(\frac{1}{1 + e^{-x}}\right)^2 e^{-x}(-1)
\]

\[
= \left(\frac{1}{1 + e^{-x}}\right)^2 \left(\frac{1}{1 + e^{-x}}\right)(-e^{-x})
\]

\[
= \left(\frac{1}{1 + e^{-x}}\right)^2 \left(\frac{-e^{-x}}{1 + e^{-x}}\right)
\]

\[
= s(x)(1 - s(x))
\]
Logistic Regression: Pros and Cons

• Doesn’t assume conditional independence of features
  – Better calibrated probabilities
  – Can handle highly correlated overlapping features

• NB is faster to train, less likely to overfit
NB & LR

- Both are linear models

- Training is different:
  - NB: weights are trained independently
  - LR: weights trained jointly

\[ z = \sum_{i=0}^{\mid X \mid} w_i x_i \]
Perceptron Algorithm

- Very similar to logistic regression
- Not exactly computing gradients
Perceptron Algorithm

- Algorithm is Very similar to logistic regression
- Not exactly computing gradients

Initialize weight vector \( w = 0 \)
Loop for \( K \) iterations
  Loop For all training examples \( x_i \)
    if \( \text{sign}(w \cdot x_i) \neq y_i \)
      \[ w += (y_i - \text{sign}(w \cdot x_i)) \cdot x_i \]
Differences between LR and Perceptron

- Online learning vs. Batch

- Perceptron doesn’t always make updates
MultiClass Classification

• Q: what if we have more than 2 categories?
  – Sentiment: Positive, Negative, Neutral
  – Document topics: Sports, Politics, Business, Entertainment, ...

• Could train a separate logistic regression model for each category...

• Pretty clear what to do with Naive Bayes.
Log-Linear Models

\[ P(y|x) \propto e^{w \cdot f(d,y)} \]

\[ P(y|x) = \frac{1}{Z(w)} e^{w \cdot f(d,y)} \]
MultiClass Logistic Regression

\[ P(y|x) \propto e^{w \cdot f(d,y)} \]

\[ P(y|x) = \frac{1}{Z(w)} e^{w \cdot f(d,y)} \]

\[ P(y|x) = \frac{e^{w \cdot f(d,y)}}{\sum_{y' \in Y} e^{w \cdot f(d,y')}} \]
MultiClass Logistic Regression

- Binary logistic regression:
  - We have one feature vector that matches the size of the vocabulary

- Multiclass in practice:
  - One weight vector for each category

Can represent this in practice with one giant weight vector and repeated features for each category.

\[ w_{pos} \quad w_{neg} \quad w_{neut} \]
Q: How to compute posterior class probabilities for multiclass?

\[
P(y = j | x_i) = \frac{e^{w_j \cdot x_i}}{\sum_k e^{w_k \cdot x_i}}
\]
Maximum Likelihood Estimation

\[ w_{\text{MLE}} = \arg \max_w \log P(y_1, \ldots, y_n | x_1, \ldots, x_n; w) \]

\[ = \arg \max_w \sum_i \log P(y_i | x_i; w) \]

\[ = \arg \max_w \sum_i \log \left( \frac{e^{w \cdot f(x_i, y_i)}}{\sum_{y' \in Y} e^{w \cdot f(x_i, y_i)}} \right) \]
Multiclass LR Gradient

\[
\frac{\partial L}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} \sum_{y \in Y} f_j(y, d_i) P(y|d_i)
\]
MAP-based learning
(perceptron)

\[
\frac{\partial L}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} f_j(\arg\max_{y \in Y} P(y|d_i), d_i)
\]
Online Learning (perceptron)

• Rather than making a full pass through the data, compute gradient and update parameters after each training example.
MultiClass Perceptron Algorithm

Initialize weight vector $w = 0$
Loop for $K$ iterations
  Loop For all training examples $x_i$
    $y_{\text{pred}} = \text{argmax}_y w_y \times x_i$
    if $y_{\text{pred}} \neq y_i$
      $w_{y_{\text{gold}}} += x_i$
      $w_{y_{\text{pred}}} -= x_i$
Q: what if there are only 2 categories?

\[ P(y = j|x_i) = \frac{e^{w_j \cdot x_i}}{\sum_k e^{w_k \cdot x_i}} \]
Q: what if there are only 2 categories?

\[ P(y = 1|x) = \frac{e^{w_1 \cdot x}}{e^{w_0 \cdot x} + e^{w_1 \cdot x} - w_1 \cdot x + e^{w_1 \cdot x}} \]
Q: what if there are only 2 categories?

\[ P(y = 1|x) = \frac{e^{w_1 \cdot x}}{e^{w_0 \cdot x} - w_1 \cdot x} \frac{e^{w_1 \cdot x}}{e^{w_1 \cdot x} + e^{w_1 \cdot x}} \]
Q: what if there are only 2 categories?

\[ P(y = 1|x) = \frac{e^{w_1 \cdot x}}{e^{w_1 \cdot x} (e^{w_0 \cdot x} - w_1 \cdot x + 1)} \]
Q: what if there are only 2 categories?

\[ P(y = 1|x) = \frac{1}{e^{w_0 \cdot x - w_1 \cdot x} + 1} \]
Q: what if there are only 2 categories?

\[ P(y = 1|x) = \frac{1}{e^{-\mathbf{w}' \cdot \mathbf{x}} + 1} \]
Regularization

• Combating over fitting

• Intuition: don’t let the weights get very large

\[ w_{\text{MLE}} = \arg \max_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w) \]

\[ \arg \max_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w) - \delta \sum_{i=1}^{V} w_i^2 \]
Regularization in the Perceptron Algorithm

• Can’t directly include regularization in gradient

• # of iterations

• Parameter averaging