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!

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

\[
f(d) \quad \rightarrow \quad x
\]

\[
f(d) = \left( \begin{array}{c}
\text{Count of “happy”}, \\
(\text{Count of “happy”}) / (\text{Length of doc}), \\
\log(1 + \text{count of “happy”}), \\
\text{Count of “not happy”}, \\
\text{Count of words in my pre-specified word list, “positive words according to my favorite psychological theory”}, \\
\text{Count of “of the”}, \\
\text{Length of document}, \\
\text{...}
\end{array} \right)
\]

Typically these use feature templates:
Generate many features at once

for each word \( w \):
- \( \$_{w\_count} \)
- \( \$_{w\_log\_l\_plus\_count} \)
- \( \$_{w\_with\_NOT\_before\_it\_count} \)
- ....

- 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.
First, we’ll discuss how LogReg works.
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 \((xs)\)

\[ x_i = \text{(count “nigerian”, count “prince”, count “nigerian prince”)} \]
Classification: LogReg (I)

- compute **features** \((xs)\)

\[ x_i = (\text{count “nigerian”, count “prince”, count “nigerian prince”}) \]

- given **weights** \((betas)\)
Classification: LogReg (I)

- **compute features** \( (xs) \)

\[
\vec{x}_i = (\text{count "nigerian"}, \text{count "prince"}, \text{count "nigerian prince"})
\]

- **given weights** \( (\text{betas}) \)

\[
\vec{\beta} = (-1.0, -1.0, 4.0)
\]
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

\[ z = \sum_{i=0}^{\text{size of } X} \beta_i x_i \]
Classification: LogReg (II)

- compute the **dot product**
  \[ z = \sum_{i=0}^{\mid X \mid} \beta_i x_i \]

- compute the **logistic function**
  \[ P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}} \]
LogReg Exercise

features: \((\text{count "nigerian"}, \text{count "prince"}, \text{count "nigerian prince"})\)

\[ \mathbf{x} = (1, 1, 1) \]

\[ \mathbf{\beta} = (-1.0, -1.0, 4.0) \]

\[ P(\mathbf{x}) = ??? \]
LogReg Exercise

\[ \mathbf{x} = (1, 1, 1) \]

\[ \mathbf{\beta} = (-1.0, -1.0, 4.0) \]

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

\[ P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}} \]
Classification: LogReg

OK, let’s take this step by step...

- Why dot product?

\[ z = \sum_{i=0}^{\|X\|} \beta_i x_i \]
Classification: LogReg

OK, let’s take this step by step...

- Why dot product?

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

- Why would we use the logistic function?
Classification: Dot Product

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

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

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

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

\[ \log[P(\text{spam}|D)] \propto \log[P(\text{spam})] + \sum_{w_i \in \text{Vocab}} x_i \cdot \log[P(w_i|\text{spam})] \]
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 \( P(z) : \mathcal{R} \to [0, 1] \)

- decision boundary is dot product = 0 (2 class)

- comes from linear log odds
  \[
  \log \frac{P(x)}{1 - P(x)} = \sum_{i=0}^{\frac{|X|}{2}} \beta_i x_i
  \]
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

<table>
<thead>
<tr>
<th>$x_{00}$</th>
<th>$x_{01}$</th>
<th>$\ldots$</th>
<th>$x_{0m}$</th>
<th>$y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{10}$</td>
<td>$x_{11}$</td>
<td>$\ldots$</td>
<td>$x_{1m}$</td>
<td>$y_1$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$x_{n0}$</td>
<td>$x_{n1}$</td>
<td>$\ldots$</td>
<td>$x_{nm}$</td>
<td>$y_n$</td>
</tr>
</tbody>
</table>

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**

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

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

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

$$= \arg \max_{\beta} \sum_{i=0}^{|X|} \log P(y_i|x_i; \beta)$$
Learning the weights

Maximize the training set’s (log-)likelihood?

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

$$\log p(y_1 \ldots y_n | x_1 \ldots x_n, \beta) = \sum_{i} \log p(y_i | x_i, \beta) = \sum_{i} \log \begin{cases} p_i & \text{if } y_i = 1 \\ 1 - p_i & \text{if } y_i = 0 \end{cases}$$

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:

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

- $\ell$: Training set log-likelihood
- $\eta$: Step size (a.k.a. learning rate)

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

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”)

\[ \beta^{(0)} = (1.0, -3.0, 2.0) \rightarrow 63\%\text{ accuracy} \]

\[ \beta^{(1)} = (0.5, -1.0, 3.0) \rightarrow 75\%\text{ accuracy} \]

\[ \beta^{(2)} = (-1.0, -1.0, 4.0) \rightarrow 81\%\text{ accuracy} \]
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:

● 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.

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

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

“Regularizer constant”: Strength of penalty

“Quadratic penalty” or “L2 regularizer”: Squared distance from origin
Visualization of a classifier in feature space

Feature vector \( x = (1, \text{count "happy"}, \text{count "hello"}, \ldots) \)

Weights/parameters \( \beta = \)

50% probability where \( \beta^T x = 0 \)

Predict \( y = 1 \) when \( \beta^T x > 0 \)

Predict \( y = 0 \) when \( \beta^T x \leq 0 \)

Graph showing the relationship between \( \beta^T x \) and the count of "happy" and "hello".
Binary vs Multiclass logreg

- Binary logreg: let $x$ be a feature vector, and $y$ either 0 or 1
  
  $\beta$ is a weight vector across the $x$ features.

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

- 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.

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