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 \( k \in 1..K \) 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

- Prediction rule: choose class \( y \) with highest score
Keyword count as linear model

- Problem: classify doc $d$ into one of $k \in 1..K$ 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

\[
\beta_{w,\text{POS}} = \begin{cases} 
1 & \text{if } w \in \text{Post lexicon} \\
0 & \text{otherwise}
\end{cases}
\]

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

\[
\text{scores} = z_{\text{POS}} = \sum_{j=1}^{n} \left( \beta_{j,\text{POS}} \right) x_j
\]

\[
= \text{Num tokens in doc that are in PostLex}
\]

\[
2_{\text{NEG}} = \sum_{j} \beta_{j,\text{NEG}} x_j
\]

\[
\hat{y} = \text{argmax} \left[ z_{\text{POS}}, 2_{\text{NEG}} \right]
\]
Naive Bayes as linear model

- Problem: classify doc $d$ into one of $k \in 1..K$ classes

- Parameters: For each class $k$, and word type $w$, there is a word weight $B_{w,k} = \log P(w|y=k)$

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

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

$$\text{argmax}_y \left[ \log P(y) \prod_{w \in d} P(w|y) \right] = \text{argmax}_y \left[ \log P(y) + \sum_{w \in d} \log P(w|y) \right]$$
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...?
Logistic regression can accommodate \textit{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.
Add NOT_ to every word between negation and following punctuation:

didn’t NOT_like NOT_this NOT_movie, but 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** \((xs)\)
  \[
  x_i = \text{(count “nigerian”, count “prince”, count “nigerian prince”)}
  \]

- given **weights** \((betas)\)
  \[
  \beta = (-1.0, -1.0, 4.0)
  \]
Classification: LogReg (II)

• Compute the **dot product**

\[
Z = \mathbf{\beta}^T \mathbf{x} = \sum_{j=1}^{N} \beta_j x_j
\]

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

\[
P(y=1|x) = g(z) = \frac{e^z}{1 + e^z}
\]

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

\[z = \begin{cases} -1000, & \frac{1}{1+e^{-1000}} = 0 \\
1000, & \frac{1}{1+e^{1000}} \approx 1 \end{cases}\]
LogReg Exercise

features: (count “nigerian”, count “prince”, count “nigerian prince”)

\[ x = (1, 1, 1) \]

\[ \beta = (-1.0, -1.0, 4.0) \]

\[ P(y=1 \mid x) = g(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-2}} \approx 0.88 \]
Classification: Dot Product

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

\( \hat{y} = 1 \quad \text{if} \quad z \geq 0 \)
Why the **logistic function**?
Logistic Function

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

\[ P(y=1|x) = \frac{1}{2} \iff \hat{y} = 1 \]

\[ P(y=1|x) < \frac{1}{2} \iff \hat{y} = 0 \]
Multi-class Logistic Regression

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

**Binary Logistic Regression**
- Features act like features in binary logistic regression.
- Weights are a row of weights corresponding to weight vector \( w \) for class \( 1 \).
- The estimated score of \( \hat{y} = \sigma^T x \) where \( \sigma \) is the sigmoid function.

**Multinomial Logistic Regression**
- Each class has its own weight vector (across all features; e.g. BOW counts).
- There are \( K \) separate weight vectors corresponding to the \( K \) classes.
- The estimated score of each class is \( \hat{y}_k = \sigma^T W_k x \) where \( \sigma \) is the softmax function and \( W_k \) is the weight vector for class \( k \).

**Examples**
- Input words: dessert, was, great
- Features:
  - Wordcount: \( x_1 \) (value 3)
  - Positive lexicon: \( x_2 \) (value 1)
  - "No": \( x_3 \) (value 0)

**Output**
- Binary logistic regression uses a single weight vector and has a scalar output \( \hat{y} \).
- Multinomial logistic regression uses a weight matrix and has a vector output \( \hat{y} \).

**Figure 5.3**
- Shows an intuition of the role of the weight vector versus weight matrix.
Multiclass Logistic Regression

• Weight vector for each class

\[ B_{i,k} = \begin{bmatrix} -1, 0, 1, 3 \end{bmatrix} \]

• Prediction: dot product for each class

\[ \forall k: z_k = (B_{i,k})^T x \]

• Predicted probabilities: apply the softmax function to normalize

\[ P(y | x) = \left[ \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \ldots + e^{z_K}} \quad \frac{e^{z_2}}{e^{z_1} + e^{z_2} + \ldots + e^{z_K}} \quad \ldots \quad \frac{e^{z_K}}{e^{z_1} + e^{z_2} + \ldots + e^{z_K}} \right] \]
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

Let's consider the following table:

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

- **n examples; xs - features; ys - class**
Learning Weights

We know:

\[
g(z) = \frac{1}{1 + e^{-z}} \quad P(y = 1 \mid x) = g \left( \sum_{j=1}^{\text{Nfeat}} \beta_j x_{ij} \right)
\]

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**

\[
\beta^{MLE} = \arg \max_{\beta} \log P(y_0, \ldots, y_n | x_0, \ldots, x_n; \beta)
\]
Gradient ascent/descent learning

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

$$L(\beta) = \sum_i \log P(y_i | x_i; \beta)$$

Follow direction of steepest ascent. Iterate:

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

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

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:

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

● 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

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?