Lecture 6
Logistic Regression for Text Classification

CS 490A, Fall 2021

https://people.cs.umass.edu/~brenocon/cs490a_f21/

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

[Many slides from Ari Kobren]

- HW1!
- Zoom OH demand?
- Accept exercises late? Yes, for check-minus: see the "Grading and Policies" page. Please let us know if you can't upload to Gradescope.

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*

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

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

Have we seen a text classification model that can be seen as an instance of this??

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**
  - "discriminative" ML
  - allows for features
  - used within more complex models (neural networks)

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

f(d) ⟶ x

f(d) = (
Count of "happy",
(Count of "happy") / (Length of doc),
log(1 + count of "happy"),
Count of "not happy",
Count of words in my pre-specified word list, "positive words according to my favorite psychological theory",
Count of several different "happy" emoticons
Count of "of the",
Length of document,
...
)

Typically these use feature templates:
Generate many features at once

for each word w:
- ${w}_count
- ${w}_is_count_nonzero
- NOT_${w}_bigram_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!! 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 = (count “nigerian”, count “prince”, count “nigerian prince”)

- given **weights** (betas)

β = (-1.0, -1.0, 4.0)

Classification: LogReg (II)

- Compute the **dot product**

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

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) =

Classification: Dot Product

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

Why the **logistic function**?

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

Multiclass Logistic Regression

- Binary logreg: let x be a feature vector for the doc, and y either 0 or 1

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

β is a weight vector across the *x* features.

- Multiclass logistic regression: weight vector for each class

- Prediction: dot product for each class

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

NB as Log-Linear Model

- What are the **features** in Naive Bayes?

- What are the **weights** in Naive Bayes?

NB as Log-Linear Model

In both NB and LogReg

we **compute the dot product!**

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

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

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

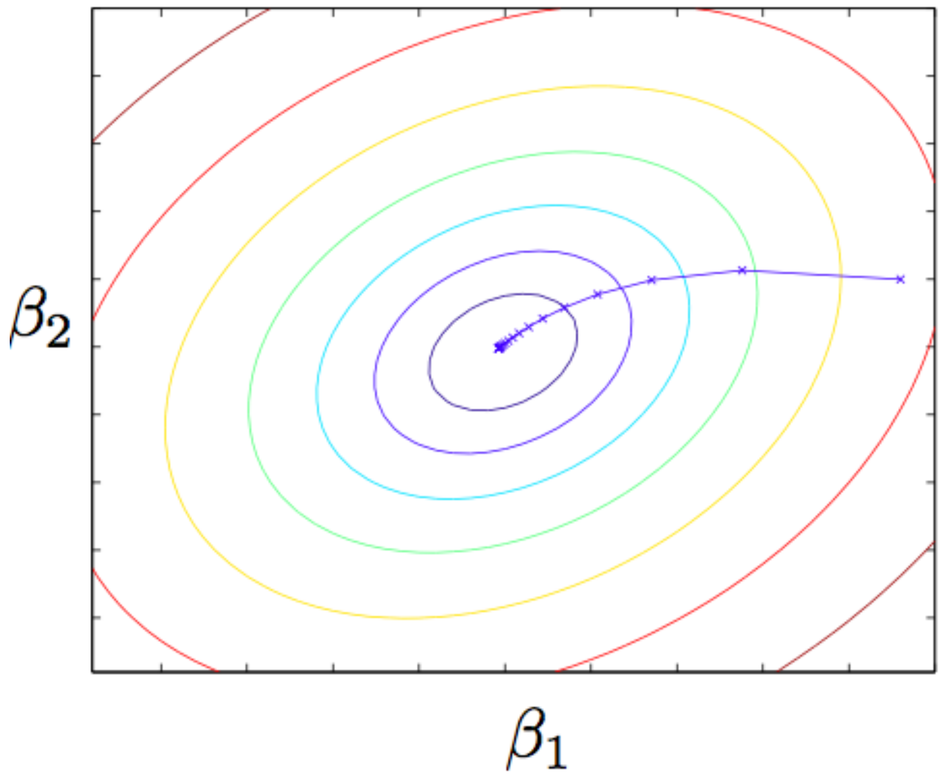
So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

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

Gradient ascent learning

- Follow direction of *steepest ascent*. Iterate

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



$$\ell(\beta) = \sum_{i=0}^{|X|} \log P(y_i | \mathbf{x_i}; \beta)$$

ℓ: Training set log-likelihood

η: Step size (a.k.a. learning rate)

$\left(\frac{\partial \ell}{\partial \beta_1}, \ldots, \frac{\partial \ell}{\partial \beta_J} \right)$: 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.

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

LogReg: Important Details!

- Overfitting / regularization
- Visualizing decision boundary / bias term
- Multiclass LogReg

You can use scikit-learn (python) to test it out!

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.
- Why might logreg (or NB) overfit to the training set?

Regularization to address overfitting

- How did we regularize Naive Bayes and language modeling?

- For logistic regression: L2 regularization for training

Regularization tradeoffs

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

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**.

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

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

"Regularizer constant": Strength of penalty

"Quadratic penalty" or "L2 regularizer": Squared distance from origin

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

$\beta^T x \leq 0$