Log-linear models (part II)

Lecture, Feb 2

CS 690N, Spring 2017

Advanced Natural Language Processing

http://people.cs.umass.edu/~brenocon/anlp2017/

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

MaxEnt / Log-Linear models

- **x**: input (all previous words)
- **y**: output (next word)
- **f(x,y)** => R^d feature function [[domain knowledge here!]]
- **v**: R^d parameter vector (weights)

$$p(y|x;v) = \frac{\exp\left(v \cdot f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)}$$

Application to history-based LM:

$$P(w_1..w_T) = \prod_t P(w_t \mid w_1..w_{t-1})$$

$$= \prod_t \frac{\exp(v \cdot f(w_1..w_{t-1}, w_t))}{\sum_{w \in \mathcal{V}} \exp(v \cdot f(w_1..w_{t-1}, w))}$$

$$f_1(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model} \text{ and } w_{i-1} = \texttt{statistical} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model},\ w_{i-2} = \texttt{any},\ w_{i-1} = \texttt{statistical} \\ 0 & \text{otherwise} \end{cases}$$

$$f_4(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model},\ w_{i-2} = \texttt{any} \\ 0 & \text{otherwise} \end{cases}$$

$$f_5(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model},\ w_{i-1} \text{ is an adjective} \\ 0 & \text{otherwise} \end{cases}$$

$$f_6(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model},\ w_{i-1} \text{ ends in ``ical''} \\ 0 & \text{otherwise} \end{cases}$$

$$f_7(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model}, \text{ ``model'' is not in } w_1, \ldots w_{i-1} \\ 0 & \text{otherwise} \end{cases}$$

$$f_8(x,y) = \begin{cases} 1 & \text{if } y = \texttt{model}, \text{ ``grammatical'' is in } w_1, \ldots w_{i-1} \\ 0 & \text{otherwise} \end{cases}$$

Figure 1: Example features for the language modeling problem, where the input x is a sequence of words $w_1 w_2 \ldots w_{i-1}$, and the label y is a word.

- These are sparse. But still very useful.

Feature templates

- Generate large collection of features from single template
  - Not part of (standard) log-linear mathematics, but how you actually build these things
- e.g. Trigram feature template:
  For every (u,v,w) trigram in training data, create feature

$$f_{N(u,v,w)}(x,y) = \begin{cases} 1 & \textit{if } y = w,\ w_{i-2} = u,\ w_{i-1} = v \\ 0 & \textit{otherwise} \end{cases}$$

where $N(u,v,w)$ is a function that maps each trigram in the training data to a unique integer.

- At training time: record N(u,v,w) mapping
- At test time: extract trigram features and check if they are in the feature vocabulary

- Feature engineering: iterative cycle of model development

Feature subtleties

- On training data, generate all features under consideration
  - Subtle issue: partially unseen features
  - At testing time, a completely new feature has to be ignored (weight 0)
- Assuming a conditional log-linear model,
  - Features typically conjoin between aspects of both input and output
  - Features can only look at the output f(y)
  - Invalid: Features that only look at the input

Multiclass Log. Reg.

- What does this look like in log-linear form?

$$P(y \mid x) = \frac{\exp(\sum_j \theta_{j,y} x_j)}{\sum_{y'} \exp(\sum_j \theta_{j,y'} x_j)}$$

- "Complete input-output conjunctions" generator: very common and effective
- Log-linear models give more flexible forms
(e.g. disjunctions on output classes)
- Ambiguous term: "feature"
- Partially "unseen" features: typically helpful

Learning

- Log-likelihood is concave
  - (At least with regularization: typically linearly separable)

$$\log p(y|x;v) \quad = \quad v \cdot f(x,y) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)$$

$$\frac{\partial}{\partial v_j} \log p(y|x;v) \quad =$$

$$\mathop{\mathbf{E}}_{h \text{ ends in "THE"}} \left[\ P_{\text{COMBINED}}(\text{BANK}|h)\ \right] \ = \ K_{\{\text{THE},\text{BANK}\}}$$

Learning

- Log-likelihood is concave
  - (At least with regularization: typically linearly separable)

$$\log p(y|x;v) \quad = \quad v \cdot f(x,y) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)$$

fun with the chain rule

$$\frac{\partial}{\partial v_j} \log p(y|x;v) \quad =$$

$$\mathop{\mathbf{E}}_{h \text{ ends in ``THE''}} [\ P_{\text{COMBINED}}(\text{BANK}|h)\ ] \ = \ K_{\{\text{THE},\text{BANK}\}}$$

Learning

- Log-likelihood is concave
  - (At least with regularization: typically linearly separable)

$$\log p(y|x;v) \quad = \quad v \cdot f(x,y) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)$$

fun with the chain rule

$$\frac{\partial}{\partial v_j} \log p(y|x;v) \quad = \quad f_j(x,y) \quad - \quad \sum_{y'} p(y'|x;v) f_j(x,y')$$

$$\underset{h \text{ ends in “THE”}}{\mathbf{E}} \left[\; P_{\text{COMBINED}}(\text{BANK}|h) \; \right] \; = \; K_{\{\text{THE},\text{BANK}\}}$$

Learning

- Log-likelihood is concave
  - (At least with regularization: typically linearly separable)

$$\log p(y|x;v) \quad = \quad v \cdot f(x,y) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)$$

fun with the chain rule

$$\frac{\partial}{\partial v_j} \log p(y|x;v) \quad = \quad f_j(x,y) \quad - \quad \sum_{y'} p(y'|x;v) f_j(x,y')$$

Feature in data? Feature in posterior?

$$\underset{h \text{ ends in ``THE''}}{\mathbf{E}} \left[\; P_{\text{COMBINED}}(\text{BANK}|h) \; \right] \; = \; K_{\{\text{THE,BANK}\}}$$

Learning

- Log-likelihood is concave
  - (At least with regularization: typically linearly separable)

$$\log p(y|x;v) \quad = \quad v \cdot f(x,y) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)$$

fun with the chain rule

$$\frac{\partial}{\partial v_j} \log p(y|x;v) \quad = \quad f_j(x,y) \quad - \quad \sum_{y'} p(y'|x;v) f_j(x,y')$$

Feature in data? Feature in posterior?

- Gradient at a single example: can it be zero?
- Full dataset gradient: First moments match at mode

$$\mathop{\mathbf{E}}_{h \text{ ends in ``THE''}} \left[\; P_{\text{COMBINED}}(\text{BANK}|h) \; \right] \; = \; K_{\{\text{THE,BANK}\}}$$

Moment matching

- Example: Rosenfeld's trigger words
- "…. loan …. went into the bank"

Empirical history prob.
(Bigram model estimate)

$$P_{\text{BIGRAM}}(\text{BANK}|\text{THE}) = K_{\{\text{THE,BANK}\}}$$

Log-linear model:
has weaker property

$$\mathop{\mathbf{E}}_{h \text{ ends in ``THE''}} [\ P_{\text{COMBINED}}(\text{BANK}|h)\ ] = K_{\{\text{THE,BANK}\}}$$

- Maximum Entropy view of a log-linear model:
- Start with feature expectations as constraints.
  What is the highest entropy distribution that satisfies them?

- stopped here 2/2

Gradient descent

- Batch gradient descent -- doesn't work well by itself
- Most commonly used alternatives
  - LBFGS (adaptive version of batch GD)
  - SGD, one example at a time
    - and adaptive variants: Adagrad, Adam, etc.
    - Intuition
    - Issue: Combining per-example sparse updates with regularization updates
      - Lazy updates
      - Occasional regularizer steps (easy to implement)

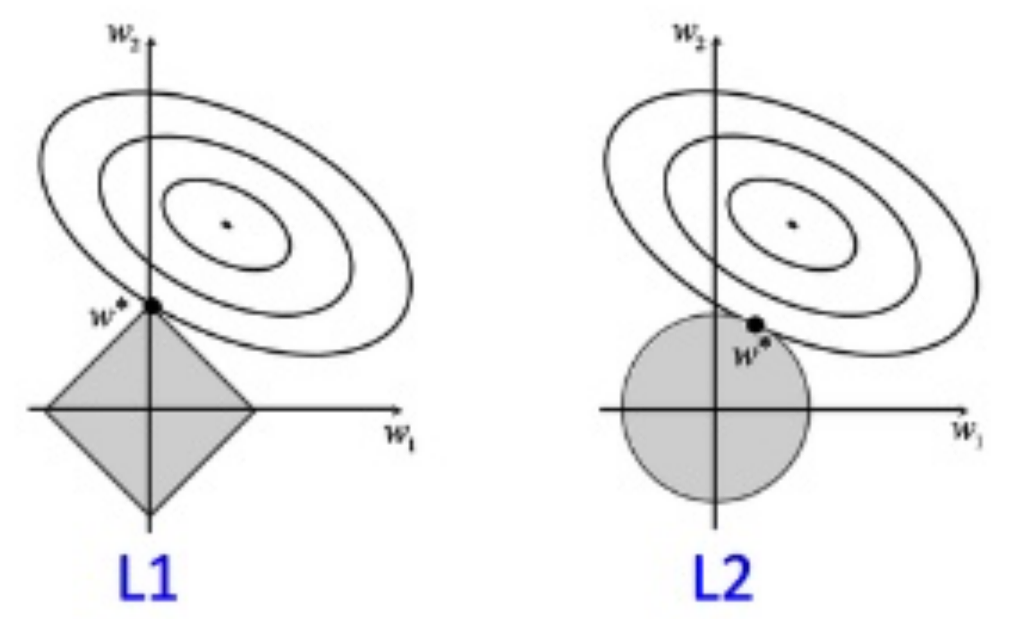
Engineering

- Sparse dot products are crucial!
- Lots and lots of features?
    - Millions to billions of features: performance often keeps improving!
    - Features seen only once at training time typically help
    - Feature name=>number mapping is the problem; the parameter vector is fine
- Feature hashing: make e.g. N(u,v,w) mapping random with collisions (!)
    - Accuracy loss low since features are rare. Works really well, and extremely practical computational properties (memory usage known in advance)
    - Practically: use a fast string hashing function (murmurhash or Python's internal one, etc.)

Feature selection

- Count cutoffs: computational, not performance
- Offline feature selection: MI/IG vs. chi-square
- L1 regularization: encourages θ sparsity

$$\min_{\theta} \; -\log p_\theta(y|x) + \lambda \sum_j |\theta_j|$$



- L1 optimization: convex but nonsmooth; requires subgradient methods