Log-linear models (part II)

CS 690N, Spring 2018

Advanced Natural Language Processing

http://people.cs.umass.edu/~brenocon/anlp2018/

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

Learning

- Log-likelihood is concave
  - (At least with regularization... need since typically linearly separable)

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

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

Learning

- Log-likelihood is concave
  - (At least with regularization... need since 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

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

Learning

- Log-likelihood is concave
  - (At least with regularization... need since 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

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

Learning

- Log-likelihood is concave
  - (At least with regularization... need since 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?

Learning

- Log-likelihood is concave
  - (At least with regularization... need since 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
  - Model-expected feature count = Empirical feature count
    For each feature j:
    $E_{y\sim p(y|x;\, v)}[\ f_j(x,y)\ ] = E_{y\sim Pempir(y|x)}[f_j(x,y)]$

Moment matching

- Example: Rosenfeld's trigger words
- ".... loan .... went into the <u>bank</u>"

Empirical history prob.
(Bigram model estimate)

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

Log-linear model:
has weaker property

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

- AVERAGED model probability over all ".... the __" instances.
  (Not same for each!)
- Maximum Entropy view of a log-linear model:
  - Start with feature expectations as constraints.
    What is the highest entropy distribution that satisfies them?

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.
    - Moment matching intuition!
    - Issue: Combining per-example sparse updates with regularization updates (lazy updates, occasional regularization sweeps)

Triggers: will they help?

HARVEST $\Leftarrow$ CROP HARVEST CORN SOYBEAN SOYBEANS AGRICULTURE GRAIN DROUGHT GRAINS BUSHELS

HARVESTING $\Leftarrow$ CROP HARVEST FORESTS FARMERS HARVESTING TIMBER TREES LOGGING ACRES FOREST

HASHEMI $\Leftarrow$ IRAN IRANIAN TEHRAN IRAN'S IRANIANS LEBANON AYATOLLAH HOSTAGES KHOMEINI ISRAELI HOSTAGE SHIITE ISLAMIC IRAQ PERSIAN TERRORISM LEBANESE ARMS ISRAEL TERRORIST

HASTINGS $\Leftarrow$ HASTINGS IMPEACHMENT ACQUITTED JUDGE TRIAL DISTRICT FLORIDA

HATE $\Leftarrow$ HATE MY YOU HER MAN ME I LOVE

HAVANA $\Leftarrow$ CUBAN CUBA CASTRO HAVANA FIDEL CASTRO'S CUBA'S CUBANS COMMUNIST MIAMI REVOLUTION

Table 7: The best triggers "A" for some given words "B", in descending order, as measured by $\mathrm{MI}(A_{\circ\text{-}3g} : B)$.

Triggers help

vocabulary	top 20,000 words of WSJ corpus	
training set	5MW (WSJ)	
test set	325KW (WSJ)	
trigram perplexity (baseline)	173	173
ME experiment	top 3	top 6
ME constraints:		
unigrams	18400	18400
bigrams	240000	240000
trigrams	414000	414000
triggers	36000	65000
ME perplexity	134	130
perplexity reduction	23%	25%
0.75·ME + 0.25·trigram perplexity	129	127
perplexity reduction	25%	27%

Table 8: Maximum Entropy models incorporating *N*-gram and trigger constraints.

note (1) feature explosion, (2) ensembling helps

Stemming: will it help?

[ACCRUAL]	:	ACCRUAL
[ACCRUE]	:	ACCRUE, ACCRUED, ACCRUING
[ACCUMULATE]	:	ACCUMULATE, ACCUMULATED, ACCUMULATING
[ACCUMULATION]	:	ACCUMULATION
[ACCURACY]	:	ACCURACY
[ACCURATE]	:	ACCURATE, ACCURATELY
[ACCURAY]	:	ACCURAY
[ACCUSATION]	:	ACCUSATION, ACCUSATIONS
[ACCUSE]	:	ACCUSE, ACCUSED, ACCUSES, ACCUSING
[ACCUSTOM]	:	ACCUSTOMED
[ACCUTANE]	:	ACCUTANE
[ACE]	:	ACE
[ACHIEVE]	:	ACHIEVE, ACHIEVED, ACHIEVES, ACHIEVING
[ACHIEVEMENT]	:	ACHIEVEMENT, ACHIEVEMENTS
[ACID]	:	ACID

Table 9: A randomly selected set of examples of stem-based clustering, using morphological analysis provided by the 'morphe' program.

Stemming doesn't help (much..)

vocabulary	top 20,000 words of WSJ corpus	
training set	300KW (WSJ)	
test set	325KW (WSJ)	
unigram perplexity	903	
model	word self-triggers	class self-triggers
ME constraints:		
unigrams	9017	9017
word self-triggers	2658	—
class self-triggers	—	2409
training-set perplexity	745	740
test-set perplexity	888	870

Table 10: Word self-triggers vs. class self-triggers, in the presence of unigram constraints. Stem-based clustering does not help much.

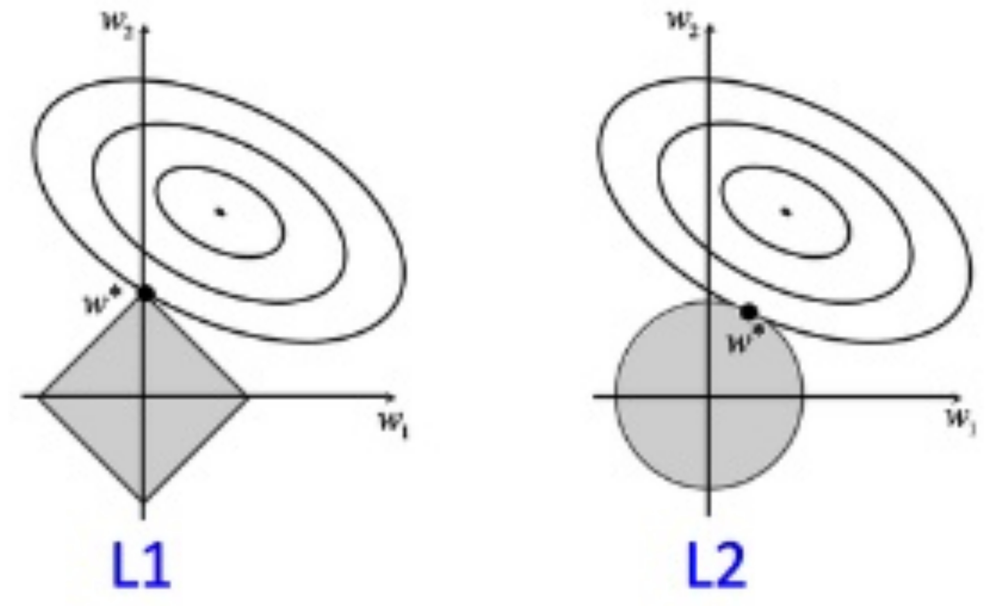
Engineering

- 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