Maximum Entropy Markov Models (log-linear model for tagging)

Instructor: Wei Xu

Many slides from Michael Collins

Overview

- Log-linear models
- Parameter estimation in log-linear models
- Smoothing/regularization in log-linear models

The General Problem

- We have some **input domain** $\mathcal{X}$
- Have a finite **label set** $\mathcal{Y}$
- Aim is to provide a **conditional probability** $p(y \mid x)$ for any x, y where $x \in \mathcal{X}$, $y \in \mathcal{Y}$

Feature Vector Representations

- Aim is to provide a conditional probability $p(y \mid x)$ for "decision" y given "history" x
- A **feature** is a function $f_k(x, y) \in \mathbb{R}$
  (Often **binary features** or **indicator functions** $f_k(x, y) \in \{0, 1\}$).
- Say we have m features f_k for $k = 1 \dots m$
  $\Rightarrow$ A **feature vector** $f(x, y) \in \mathbb{R}^m$ for any x, y

features are a property of both observation *x* and the candidate output class *y*

Parameter Vectors

- Given features $f_k(x, y)$ for $k = 1 \ldots m$, also define a **parameter vector** $v \in \mathbb{R}^m$

all possible m-dimensional real value vectors

- Each (x, y) pair is then mapped to a "score"

$$v \cdot f(x, y) = \sum_k v_k f_k(x, y)$$

However, this doesn't produce a legal probability

Log-Linear Models

- We have some input domain $\mathcal{X}$, and a finite label set $\mathcal{Y}$. Aim is to provide a conditional probability $p(y \mid x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- A feature is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$
  (Often binary features or indicator functions $f_k : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$).
- Say we have m features f_k for $k = 1 \ldots m$
  $\Rightarrow$ A feature vector $f(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- We also have a **parameter vector** $v \in \mathbb{R}^m$
- We define

$$p(y \mid x; v) = \frac{e^{v \cdot f(x,y)}}{\sum_{y' \in \mathcal{Y}} e^{v \cdot f(x,y')}}$$

Softmax!

Exercise

Why the name?

$$\log p(y \mid x; v) = \underbrace{v \cdot f(x, y)}_{\text{Linear term}} - \underbrace{\log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x, y')}}_{\text{Normalization term}}$$

Overview

- Log-linear models
- Parameter estimation in log-linear models
- Smoothing/regularization in log-linear models

Maximum-Likelihood Estimation

- Maximum-likelihood estimates given training sample $(x^{(i)}, y^{(i)})$ for $i = 1 \ldots n$, each $(x^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y}$:

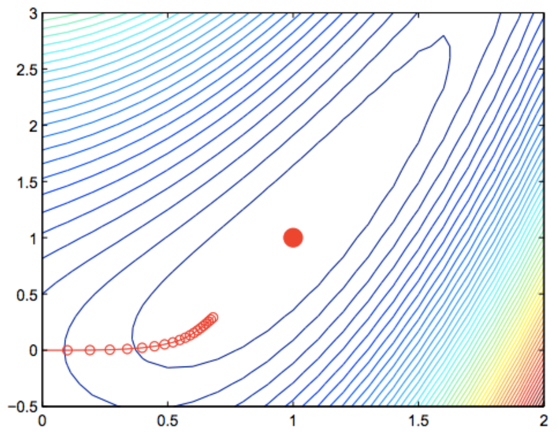
$$v_{ML} = \operatorname{argmax}_{v \in \mathbb{R}^m} L(v)$$

where

$$L(v) = \sum_{i=1}^{n} \log p(y^{(i)} \mid x^{(i)}; v) = \sum_{i=1}^{n} v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')}$$

concave function!

Calculating the Maximum-Likelihood Estimates



- Need to maximize:

$$L(v) = \sum_{i=1}^{n} v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')}$$

- Calculating gradients:

$$\begin{aligned}
\frac{dL(v)}{dv_k} &= \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \frac{\sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') e^{v \cdot f(x^{(i)}, y')}}{\sum_{z' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, z')}} \\
&= \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \frac{e^{v \cdot f(x^{(i)}, y')}}{\sum_{z' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, z')}} \\
&= \underbrace{\sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)})}_{\text{Empirical counts}} - \underbrace{\sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' \mid x^{(i)}; v)}_{\text{Expected counts}}
\end{aligned}$$

Gradient Ascent Methods

- Need to maximize $L(v)$ where

$$\frac{dL(v)}{dv} = \sum_{i=1}^{n} f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} f(x^{(i)}, y') p(y' \mid x^{(i)}; v)$$

Initialization: $v = 0$

Iterate until convergence:

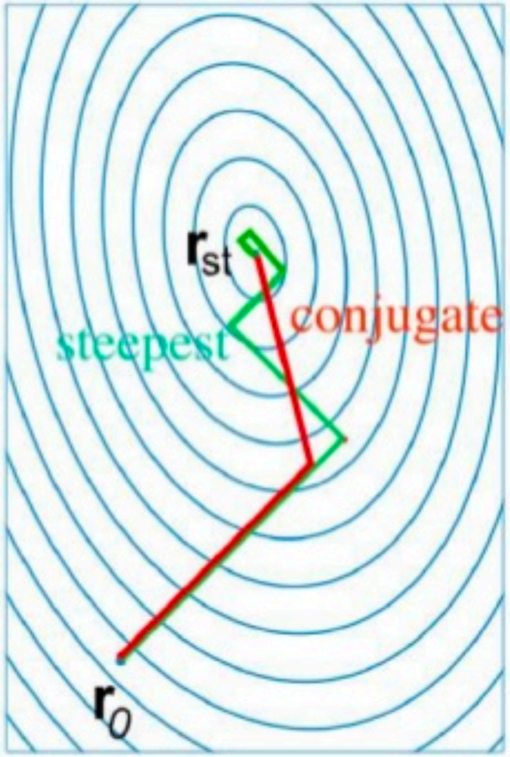
- Calculate $\Delta = \frac{dL(v)}{dv}$
- Calculate $\beta_* = \text{argmax}_\beta L(v + \beta \Delta)$ (Line Search)
- Set $v \leftarrow v + \beta_* \Delta$

Conjugate Gradient Methods

- (Vanilla) gradient ascent can be very slow
- Conjugate gradient methods require calculation of gradient at each iteration, but do a line search in **a direction which is a function of the current gradient, and the previous step taken**.
- Conjugate gradient packages are widely available
  In general: they require a function

$$\texttt{calc\_gradient}(v) \rightarrow \left(L(v), \frac{dL(v)}{dv} \right)$$

and that's about it!



e.g. LBFGS Algorithm (Limited-memory Broyden–Fletcher–Goldfarb–Shanno)

https://www.youtube.com/watch?v=h4cG8jLGmKg

Overview

- Log-linear models
- Parameter estimation in log-linear models
- Smoothing/regularization in log-linear models

Smoothing in Log-Linear Models

- Say we have a feature:

$$f_{100}(x, y) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } y = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases}$$

- In training data, base is seen 3 times, with Vt every time
- Maximum likelihood solution satisfies

$$\sum_i f_{100}(x^{(i)}, y^{(i)}) = \sum_i \sum_y p(y \mid x^{(i)}; v) f_{100}(x^{(i)}, y)$$

$\Rightarrow p(\mathsf{Vt} \mid x^{(i)}; v) = 1$ for any history $x^{(i)}$ where $w_i =$ base
$\Rightarrow v_{100} \rightarrow \infty$ at maximum-likelihood solution (most likely)
$\Rightarrow p(\mathsf{Vt} \mid x; v) = 1$ for any test data history x where $w =$ base

Regularization

- Modified loss function

$$L(v) = \sum_{i=1}^{n} v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')} - \frac{\lambda}{2} \sum_{k=1}^{m} v_k^2$$

- Calculating gradients:

$$\frac{dL(v)}{dv_k} = \underbrace{\sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)})}_{\text{Empirical counts}} - \underbrace{\sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' \mid x^{(i)}; v)}_{\text{Expected counts}} - \lambda v_k$$

- Can run conjugate gradient methods as before
- Adds a penalty for large weights

Log Linear Models for Tagging

Overview

- Recap: The Tagging Problem
- Log-linear taggers

Tagging (Sequence Labeling)

- Given a sequence (in NLP, words), assign appropriate labels to each word.
- Many NLP problems can be viewed as sequence labeling:
  - POS Tagging
  - Chunking
  - Named Entity Tagging
- Labels of tokens are dependent on the labels of other tokens in the sequence, particularly their neighbors

Plays	well	with	others.
VBZ	RB	IN	NNS

Log-Linear Models for Tagging

- We have an input sentence $w_{[1:n]} = w_1, w_2, \ldots, w_n$
  (w_i is the i'th word in the sentence)

Log-Linear Models for Tagging

- We have an input sentence $w_{[1:n]} = w_1, w_2, \ldots, w_n$
  (w_i is the i'th word in the sentence)

- We have a tag sequence $t_{[1:n]} = t_1, t_2, \ldots, t_n$
  (t_i is the i'th tag in the sentence)

Log-Linear Models for Tagging

- We have an input sentence $w_{[1:n]} = w_1, w_2, \ldots, w_n$
  (w_i is the i'th word in the sentence)

- We have a tag sequence $t_{[1:n]} = t_1, t_2, \ldots, t_n$
  (t_i is the i'th tag in the sentence)

- We'll use an log-linear model to define

$$p(t_1, t_2, \ldots, t_n | w_1, w_2, \ldots, w_n)$$

  for any sentence $w_{[1:n]}$ and tag sequence $t_{[1:n]}$ of the same length.
  (Note: contrast with HMM that defines $p(t_1 \ldots t_n, w_1 \ldots w_n)$)

Log-Linear Models for Tagging

- We have an input sentence $w_{[1:n]} = w_1, w_2, \ldots, w_n$
  (w_i is the i'th word in the sentence)

- We have a tag sequence $t_{[1:n]} = t_1, t_2, \ldots, t_n$
  (t_i is the i'th tag in the sentence)

- We'll use an log-linear model to define

$$p(t_1, t_2, \ldots, t_n | w_1, w_2, \ldots, w_n)$$

  for any sentence $w_{[1:n]}$ and tag sequence $t_{[1:n]}$ of the same length.
  (Note: contrast with HMM that defines $p(t_1 \ldots t_n, w_1 \ldots w_n)$)

- Then the most likely tag sequence for $w_{[1:n]}$ is

$$t^*_{[1:n]} = \text{argmax}_{t_{[1:n]}} p(t_{[1:n]} | w_{[1:n]})$$

How to model $p(t_{[1:n]}|w_{[1:n]})$?

A Trigram Log-Linear Tagger:

$p(t_{[1:n]}|w_{[1:n]}) = \prod_{j=1}^{n} p(t_j \mid w_1 \ldots w_n, t_1 \ldots t_{j-1})$ Chain rule

How to model $p(t_{[1:n]}|w_{[1:n]})$?

A Trigram Log-Linear Tagger:

$p(t_{[1:n]}|w_{[1:n]}) = \prod_{j=1}^{n} p(t_j \mid w_1 \ldots w_n, t_1 \ldots t_{j-1})$ Chain rule

$= \prod_{j=1}^{n} p(t_j \mid w_1, \ldots, w_n, t_{j-2}, t_{j-1})$

Independence assumptions

- We take $t_0 = t_{-1} = *$

How to model $p(t_{[1:n]}|w_{[1:n]})$?

A Trigram Log-Linear Tagger:

$p(t_{[1:n]}|w_{[1:n]}) = \prod_{j=1}^{n} p(t_j \mid w_1 \ldots w_n, t_1 \ldots t_{j-1})$ Chain rule

$= \prod_{j=1}^{n} p(t_j \mid w_1, \ldots, w_n, t_{j-2}, t_{j-1})$

Independence assumptions

- We take $t_0 = t_{-1} = *$
- Independence assumption: each tag only depends on previous two tags

$$p(t_j|w_1, \ldots, w_n, t_1, \ldots, t_{j-1}) = p(t_j|w_1, \ldots, w_n, t_{j-2}, t_{j-1})$$

An Example

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere .

- There are many possible tags in the position ??

$\mathcal{Y} = \{$NN, NNS, Vt, Vi, IN, DT, $\ldots\}$

Representation: Histories

- A **history** is a 4-tuple $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$
- t_{-2}, t_{-1} are the previous two tags.
- $w_{[1:n]}$ are the n words in the input sentence.
- i is the index of the word being tagged
- $\mathcal{X}$ is the set of all possible histories

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere .

- $t_{-2}, t_{-1} =$ DT, JJ
- $w_{[1:n]} = \langle Hispaniola, quickly, became, \ldots, Hemisphere, . \rangle$
- $i = 6$

An Example (continued)

- $\mathcal{X}$ is the set of all possible histories of form $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$
- $\mathcal{Y} = \{$NN, NNS, Vt, Vi, IN, DT, $\ldots\}$
- We have m features $f_k : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ for $k = 1 \ldots m$

For example:

$$f_1(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } t = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases}$$

analogy to e(base|Vt) in HMMs

$$f_2(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \texttt{ing} \text{ and } t = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases}$$

difficult for HMMs

$\ldots$

$f_1(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6 \rangle, \text{Vt}) = 1$
$f_2(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle, 6 \rangle, \text{Vt}) = 0$

$\ldots$

Training the Log-Linear Model

- To train a log-linear model, we need a training set (x_i, y_i) for $i = 1 \ldots n$. Then search for

$$v^* = \text{argmax}_v \left(\underbrace{\sum_i \log p(y_i | x_i; v)}_{Log-Likelihood} - \underbrace{\frac{\lambda}{2} \sum_k v_k^2}_{Regularizer} \right)$$

- Training set is simply all history/tag pairs seen in the training data

The Viterbi Algorithm

Problem: for an input $w_1 \ldots w_n$, find

$$\arg\max_{t_1 \ldots t_n} p(t_1 \ldots t_n \mid w_1 \ldots w_n)$$

We assume that p takes the form

$$p(t_1 \ldots t_n \mid w_1 \ldots w_n) = \prod_{i=1}^{n} q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

(In our case $q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$ is the estimate from a log-linear model.)

The Viterbi Algorithm

- Define n to be the length of the sentence
- Define

$$r(t_1 \ldots t_k) = \prod_{i=1}^{k} q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

- Define a dynamic programming table

$\pi(k, u, v)$ = maximum probability of a tag sequence ending in tags u, v at position k

that is,

$$\pi(k, u, v) = \max_{\langle t_1, \ldots, t_{k-2} \rangle} r(t_1 \ldots t_{k-2}, u, v)$$

A Recursive Definition

Base case:

$$\pi(0, *, *) = 1$$

Recursive definition:
For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v | t, u, w_{[1:n]}, k) \right)$$

where $\mathcal{S}_k$ is the set of possible tags at position k

The Viterbi Algorithm with Backpointers

Input: a sentence $w_1 \ldots w_n$, log-linear model that provides $q(v|t, u, w_{[1:n]}, i)$ for any tag-trigram t, u, v, for any $i \in \{1 \ldots n\}$
Initialization: Set $\pi(0, *, *) = 1$.
Algorithm:

- For $k = 1 \ldots n$,
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\begin{aligned}
\pi(k, u, v) &= \max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k) \right) \\
bp(k, u, v) &= \arg\max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k) \right)
\end{aligned}$$

- Set $(t_{n-1}, t_n) = \arg\max_{(u,v)} \pi(n, u, v)$
- For $k = (n-2) \ldots 1$, $t_k = bp(k+2, t_{k+1}, t_{k+2})$
- **Return** the tag sequence $t_1 \ldots t_n$

Summary

- Key ideas in log-linear taggers:
  - Decompose

$$p(t_1 \ldots t_n | w_1 \ldots w_n) = \prod_{i=1}^{n} p(t_i | t_{i-2}, t_{i-1}, w_1 \ldots w_n)$$

  - Estimate

$$p(t_i | t_{i-2}, t_{i-1}, w_1 \ldots w_n)$$

    using a log-linear model
  - For a test sentence $w_1 \ldots w_n$, use the Viterbi algorithm to find

$$\arg\max_{t_1 \ldots t_n} \left(\prod_{i=1}^{n} p(t_i | t_{i-2}, t_{i-1}, w_1 \ldots w_n) \right)$$

- Key advantage over HMM taggers: flexibility in the features they can use