Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

October 7, 2014

Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

Part 1: Classification tasks in NLP

Classification tasks in NLP

Probability models for Classification

Naive Bayes Classifier

Log linear models

Tagging tasks in NLP

Prepositional Phrases

- noun attach: *I bought the shirt with pockets*
- verb attach: *I washed the shirt with soap*
- As in the case of other attachment decisions in parsing: it depends on the meaning of the entire sentence – needs world knowledge, etc.
- Maybe there is a simpler solution: we can attempt to solve it using heuristics or associations between words

Ambiguity Resolution: Prepositional Phrases in English

- Learning Prepositional Phrase Attachment: Annotated Data

v	n_1	p	n_2	Attachment
join	board	as	director	V
is	chairman	of	N.V.	N
using	crocidolite	in	filters	V
bring	attention	to	problem	V
is	asbestos	in	products	N
making	paper	for	filters	N
including	three	with	cancer	N
⋮	⋮	⋮	⋮	⋮

Prepositional Phrase Attachment

Method	Accuracy
Always noun attachment	59.0
Most likely for each preposition	72.2
Average Human (4 head words only)	88.2
Average Human (whole sentence)	93.2

Back-off Smoothing

- Random variable a represents attachment.
- $a = n_1$ or $a = v$ (two-class classification)
- We want to compute probability of noun attachment: $p(a = n_1 \mid v, n_1, p, n_2)$.
- Probability of verb attachment is $1 - p(a = n_1 \mid v, n_1, p, n_2)$.

Back-off Smoothing

1. If $f(v, n_1, p, n_2) > 0$ and $\hat{p} \neq 0.5$

$$\hat{p}(1 \mid v, n_1, p, n_2) = \frac{f(1, v, n_1, p, n_2)}{f(v, n_1, p, n_2)}$$

2. Else if $f(v, n_1, p) + f(v, p, n_2) + f(n_1, p, n_2) > 0$ and $\hat{p} \neq 0.5$

$$\hat{p}(1 \mid v, n_1, p, n_2) = \frac{f(1, v, n_1, p) + f(1, v, p, n_2) + f(1, n_1, p, n_2)}{f(v, n_1, p) + f(v, p, n_2) + f(n_1, p, n_2)}$$

3. Else if $f(v, p) + f(n_1, p) + f(p, n_2) > 0$

$$\hat{p}(1 \mid v, n_1, p, n_2) = \frac{f(1, v, p) + f(1, n_1, p) + f(1, p, n_2)}{f(v, p) + f(n_1, p) + f(p, n_2)}$$

4. Else if $f(p) > 0$

$$\hat{p}(1 \mid v, n_1, p, n_2) = \frac{f(1, p)}{f(p)}$$

5. Else $\hat{p}(1 \mid v, n_1, p, n_2) = 1.0$

Prepositional Phrase Attachment: Results

- **Results (Collins and Brooks 1995)**: 84.5% accuracy with the use of some limited word classes for dates, numbers, etc.
- **Toutanova, Manning, and Ng, 2004**:
  use sophisticated smoothing model for PP attachment
  86.18% with words & stems; with word classes: 87.54%
- **Merlo, Crocker and Berthouzoz, 1997**:
  test on multiple PPs, generalize disambiguation of 1 PP to 2-3 PPs
  1PP: 84.3%   2PP: 69.6%   3PP: 43.6%
  Note that this is still not the real problem faced in parsing natural language

Classification tasks in NLP

Probability models for Classification

Naive Bayes Classifier

Log linear models

Tagging tasks in NLP

Probability Models

- $p(x, y)$: x = input, y = labels
- Pick best prob distribution $p(x, y)$ to fit the data
- Max likelihood of the data *according to the prob model* equivalent to minimizing entropy

Probability Models

- Max likelihood of the data *according to the prob model*
- Equivalent to picking best parameter values θ such that the data gets highest likelihood:

$$\max_\theta p(\theta \mid \mathrm{data}) = \max_\theta p(\theta) \cdot p(\mathrm{data} \mid \theta)$$

Log probabilities v.s. scores

- n-grams: $\ldots + \log\, p(w_8 \mid w_6, w_7) + \ldots$
- HMM: $\ldots + \log\, p(t_5 \mid t_4) + \log\, p(w_5 \mid t_5) + \ldots$
- Naive Bayes: $\log\, p(\text{class}) + \log\, p(\text{feature}_1 \mid \text{class}) + \log\, p(\text{feature}_2 \mid \text{class}) + \ldots$

Advantages of probability models

- parameters can be estimated automatically, while scores have to twiddled by hand
- parameters can be estimated from supervised or unsupervised data
- probabilities can be used to quantify confidence in a particular state and used to compare against other probabilities in a strictly comparable setting
- modularity: $p(\mathit{semantics}) \cdot p(\mathit{syntax} \mid \mathit{semantics}) \cdot p(\mathit{morphology} \mid \mathit{syntax}) \cdot p(\mathit{phonology} \mid \mathit{morphology}) \cdot p(\mathit{sounds} \mid \mathit{phonology})$

Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

Part 2: Probabilistic Classifiers

Classification tasks in NLP

Probability models for Classification

Naive Bayes Classifier

Log linear models

Tagging tasks in NLP

Naive Bayes Classifier

- $\mathbf{x}$ is the input that can be represented as d independent features f_j, $1 \leq j \leq d$
- y is the output classification
- $P(y \mid \mathbf{x}) = \frac{P(y) \cdot P(\mathbf{x}|y)}{P(\mathbf{x})}$ (Bayes Rule)
- $P(\mathbf{x} \mid y) = \prod_{j=1}^{d} P(f_j \mid y)$
- $P(y \mid \mathbf{x}) = P(y) \cdot \prod_{j=1}^{d} P(f_j \mid y)$

Using Naive Bayes for Document Classification

- Spam text: `Learn how to make $38.99 into a money making machine that pays ... $7,000 / month !`
- Distinguish spam text from regular email text
- Find useful features to make this distinction

Using Naive Bayes

- Useful features
  1. contains `turn $AMOUNT into`
  2. contains `$AMOUNT`
  3. contains `Learn how to`
  4. contains exclamation mark at end of sentence

Using Naive Bayes

- how many times do these features occur?
  1. contains: `turn $AMOUNT into`
     in spam text: 50
     in normal email: 2
     i.e. 25x more likely in spam
  2. contains: `$AMOUNT`
     in spam text: 90
     in normal email: 10
     i.e. 9x more likely in spam

Using Naive Bayes

- How likely is it for *both* features to occur at the same time in a spam message?
  1. contains: `turn $AMOUNT into`
  2. contains: `$AMOUNT`
- Assume we have a new feature, contains: `turn $AMOUNT into` *and* `$AMOUNT`
- The model predicts that the event that both features occur simultaneously has probability $\frac{140}{152} = 0.92$
- But Naive Bayes assumes that these features are independent and should occur with probability: $0.92 \cdot 0.9 = 0.864$

Using Naive Bayes

- Naive Bayes needs overlapping but independent features
- How can we use all of the features we want?
  1. contains `turn $AMOUNT into`
  2. contains `$AMOUNT`
  3. contains `Learn how to`
  4. contains exclamation mark at end of sentence
- how about giving each feature a weight w equal to its log probability: $w = \log p(f, y)$

Using Naive Bayes

- each feature gets a score equal to its log probability
- Assign scores to features:
  1. $w_1 = +1$ contains `turn $AMOUNT into`
  2. $w_2 = +5$ contains `$AMOUNT`
  3. $w_3 = +1$ contains `Learn how to`
  4. $w_4 = -2$ contains exclamation mark at end of sentence

Using Naive Bayes

- so add the scores and treat it like a log probability
- $\log p(\mathit{spam} \mid \mathit{feats}) = 4.2$
- but then, $p(\mathit{spam} \mid \mathit{feats}) = \mathit{exp}(4.2) = 66.68$
- how do we compute keep arbitrary scores and still get probabilities?

Classification tasks in NLP

Probability models for Classification

Naive Bayes Classifier

Log linear models

Tagging tasks in NLP

Log linear model

- Let there be m features, $f_k(\mathbf{x}, y)$ for $k = 1, \ldots, m$
- Define a parameter vector $\mathbf{w} \in \mathbb{R}^m$
- Each $(\mathbf{x}, y)$ pair is mapped to score:

$$s(\mathbf{x}, y) = \sum_k w_k \cdot f_k(\mathbf{x}, y)$$

- Using inner product notation:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y) &= \sum_k w_k \cdot f_k(\mathbf{x}, y) \\ s(\mathbf{x}, y) &= \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y) \end{aligned}$$

- To get a probability from the score: Renormalize!

$$\Pr(y \mid \mathbf{x}, \mathbf{w}) = \frac{exp\left(s(\mathbf{x}, y)\right)}{\sum_{y'} exp\left(s(\mathbf{x}, y')\right)}$$

Log linear model

- The name 'log-linear model' comes from:

$$\log \Pr(y \mid \mathbf{x}, \mathbf{w}) = \underbrace{\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y)}_{\text{linear term}} - \underbrace{\log \sum_{y'} exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y')\right)}_{\text{normalization term}}$$

- Once the weights are learned, we can perform predictions using these features.
- The goal: to find $\mathbf{w}$ that maximizes the log likelihood $L(\mathbf{w})$ of the labeled training set containing $(\mathbf{x}_i, y_i)$ for $i = 1 \ldots n$

$$\begin{aligned} L(\mathbf{w}) &= \sum_i \log \Pr(y_i \mid \mathbf{x}_i, \mathbf{w}) \\ &= \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \log \sum_{y'} exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right) \end{aligned}$$

Log linear model

- Maximize:

$$L(\mathbf{w}) = \sum_i \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \log \sum_{y'} exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right)$$

- Calculate gradient:

$$\left.\frac{dL(\mathbf{w})}{d\mathbf{w}}\right|_{\mathbf{w}}$$

$$= \sum_i \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \frac{1}{\sum_{y''} exp\left(\mathbf{w} \cdot \mathbf{f}(x_i, y'')\right)} \sum_{y'} \mathbf{f}(\mathbf{x}_i, y') \cdot exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right)$$

$$= \sum_i \mathbf{f}(\mathbf{x}_i, y_i) - \sum_i \sum_{y'} \mathbf{f}(\mathbf{x}_i, y') \frac{exp\left(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i, y')\right)}{\sum_{y''} exp\left(\mathbf{w} \cdot \mathbf{f}(x_i, y'')\right)}$$

$$= \underbrace{\sum_i \mathbf{f}(\mathbf{x}_i, y_i)}_{\text{Observed counts}} - \underbrace{\sum_i \sum_{y'} \mathbf{f}(\mathbf{x}_i, y') \Pr(y' \mid \mathbf{x}_i, \mathbf{w})}_{\text{Expected counts}}$$

Log linear model

- Init: $\mathbf{w}^{(0)} = \mathbf{0}$
- $t \leftarrow 0$
- Iterate until convergence:
  - Calculate: $\Delta = \left.\frac{dL(\mathbf{w})}{d\mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{(t)}}$
  - Find $\beta^* = \arg\max_\beta L(\mathbf{w}^{(t)} + \beta\Delta)$
  - Set $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \beta^*\Delta$

Learning the weights: $\mathbf{w}$: Generalized Iterative Scaling

$f^{\#} = max_{x,y} \sum_j f_j(x, y)$
(the maximum possible feature value; needed for scaling)
Initialize $\mathbf{w}^{(0)}$
For each iteration t

    `expected[j]` $\leftarrow$ 0 for j = 1 .. # of features
    For i = 1 to $|\text{training data}|$
        For each feature f_j
            `expected[j]` $\mathrel{+}= f_j(x_i, y_i) \cdot P(y_i \mid x_i, \mathbf{w}^{(t)})$
    For each feature $f_j(x, y)$
        `observed[j]` $= f_j(x, y) \cdot \frac{c(x,y)}{|\text{training data}|}$
    For each feature $f_j(x, y)$
        $w_j^{(t+1)} \leftarrow w_j^{(t)} \cdot \sqrt[f^{\#}]{\frac{\texttt{observed[j]}}{\texttt{expected[j]}}}$

cf. Goodman, NIPS '01

Natural Language Processing

Anoop Sarkar
anoopsarkar.github.io/nlp-class

Simon Fraser University

Part 3: Linear models for Tagging

Classification tasks in NLP

Probability models for Classification

Naive Bayes Classifier

Log linear models

Tagging tasks in NLP

Tagging Tasks

Tagged Sequences

a b e e a f h j ⇒ a/Y b/Z e/Y e/Y a/Z f/X h/Z j/Y

Example 1: Part-of-speech tagging

Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV topping/V forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.

Example 2: Named Entity Recognition

Profits/O soared/O at/O Boeing/B-CO Co./I-CO ,/O easily/O topping/O forecasts/O on/O Wall/B-LOC Street/I-LOC ,/O as/O their/O CEO/O Alan/B-PER Mulally/I-PER announced/O first/O quarter/O results/O ./O

Notation for Tagging Tasks

- Set of possible input words: $\mathcal{V}$
- Set of possible tags: $\mathcal{T}$
- Tag sequence: $t_{[1:n]} = [t_1, \ldots, t_n]$
- Training data is N tagged sentences, the i^{th} sentence has length n_i:

$$(w^{(i)}_{[1:n]}, t^{(i)}_{[1:n]}) \text{ for } i = 1, \ldots, n$$

Independence Assumptions for Tagging

Chain Rule

$$P(t_{[1:n]} \mid w_{[1:n]}) = \prod_{j=1}^{n} P(t_j \mid t_{j-1}, \ldots, t_1, w_{[1:n]}, j)$$

Make independence assumptions

$$P(t_{[1:n]} \mid w_{[1:n]}) \approx \prod_{j=1}^{n} P(t_j \mid t_{j-1}, w_{[1:n]}, j)$$

j is the word being tagged

Questions

- Split up $P(t_j \mid t_{j-1}, w_{[1:n]}, j)$ into parameters?
- How to find $\arg\max_{t_{[1:n]}} P(t_{[1:n]} \mid w_{[1:n]})$?

Acknowledgements

Many slides borrowed or inspired from lecture notes by Michael Collins, Chris Dyer, Adam Lopez, and Luke Zettlemoyer from their NLP course materials. All mistakes are my own.