CMPT-825

Natural Language Processing

Anoop Sarkar

`http://www.cs.sfu.ca/~anoop`

Probability Models

- $p(a,b)$: a = input, b = labels

- Pick best prob distribution $p(a,b)$ 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:

$$\max_{\theta} p(\theta \mid \text{data}) = \max_{\theta} p(\theta) \times p(\text{data} \mid \theta)$$

What happened to good, old fashioned AI?

- No stinkin' probabilities: real AI is done with heuristic scores
- Assign scores ($+$ score or $-$ score) – sum it all up – and then use it to weight alternatives
- So are probability models any better than this approach?
- Worse: are they the same?

Aren't log probabilities just scores

- n-grams: $\ldots + \log p(w_8 \mid w_6, w_7) + \ldots$
- HMM: $\ldots + \log p(t_5 \mid t_3, t_4) + \log p(w_5 \mid t_5) + \ldots$
- Naive Bayes:
  $\ldots + \log p(\text{class}) + \log p(\text{feature}_1 \mid \text{class}) + \log p(\text{feature}_2 \mid \text{class}) + \ldots$

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: $p(semantics) \times p(syntax \mid semantics) \times p(morphology \mid syntax) \times p(phonology \mid morphology) \times p(sounds \mid phonology)$

Remember the humble Naive Bayes Classifier

- $P(c_k \mid \mathbf{x}) = \frac{P(c_k) \times P(\mathbf{x} | c_k)}{P(\mathbf{x})}$

- $P(\mathbf{x} \mid c_k) = \prod_{j=1}^{d} P(x_j \mid c_k)$

- $P(c_k \mid \mathbf{x}) = P(c_k) \times \prod_{j=1}^{d} P(x_j \mid c_k)$

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: 0.5
     in normal email: 0.02
     i.e. 25x more likely in spam

  2. contains `$AMOUNT`
     in spam text: 0.9
     in normal email: 0.1
     i.e. 9x more likely in spam

Using Naive Bayes

- How likely is it for *both* features to occur at the same time
  1. contains `turn $AMOUNT into`
  2. contains `$AMOUNT`
- The model predicts that the event that both features occur simultaneously has probability 0.45
  i.e. 25x9 = 225x more likely in spam than in normal email.
- **What went wrong?**

Using Naive Bayes

- How likely is it for *both* features to occur at the same time

  1. contains `turn $AMOUNT into`
     in spam: 0.5 log prob = -1
     in normal email: 0.02 log prob = -5.64

  2. contains `$AMOUNT`
     in spam: 0.9 log prob = -0.15
     in normal email: 0.1 log prob = -3.3

- **tweak it by hand**
  in spam: 0.85 log prob = -2.3
  But what is the basic problem

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 score equal to its log probability

Using Naive Bayes

- each feature gets a score equal to its log probability

- Assign scores to features:

  1. $\lambda_1 = +1$ contains `turn $AMOUNT into`
  2. $\lambda_2 = +5$ contains `$AMOUNT`
  3. $\lambda_3 = +0.2$ contains `Learn how to`
  4. $\lambda_4 = -2$ contains exclamation mark at end of sentence

Using Naive Bayes

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

Log linear model

- Renormalize! $P(\textit{spam} \mid x) = \frac{P(\textit{spam},x)}{P(x)}$

$$p(\textit{spam}, x) = \frac{1}{Z(\lambda)} exp \sum_i \lambda_i f_i(x)$$

  - x is the email message
  - λ_i is the weight of feature i
  - $f_i(x) \in \{0, 1\}$ tells us whether x has feature i
  - $\frac{1}{Z(\lambda)}$ is a normalizing factor making $\sum_x p(\textit{spam}, x) = 1$

- called log-linear: **why?**

Log linear model

- Now we can get the weights from data
- Choose λ_i such that the log prob of the training data is maximized:
  $log \prod_j p(c_j) \times p(x_j \mid c_j)$
- log linear models are convex functions – easy to maximize **why?**

Log linear model

- Instead of having separate models
  $p(\textit{spam} \mid x) = p(\textit{spam}) \times p(x \mid \textit{spam})$ vs.
  $p(\textit{normal} \mid x) = p(\textit{normal}) * p(x \mid \textit{normal})$

- Have one model $p(x, c)$

- Equivalent to changing features into:
  message is spam *and* contains `turn $AMOUNT into`

Maximum Entropy

- The maximum entropy principle: related to Occam's razor and other similar justifications for scientific inquiry

- Make the minimum possible assumptions about unseen data

- Also: Laplace's *Principle of Insufficient Reason*: when one has no information to distinguish between the probability of two events, the best strategy is to consider them equally likely

Maximum Entropy

- Amazing theorem: Maximum Likelihood estimate equals Maximum Entropy estimate

$$p(\textit{spam}, x) = \frac{1}{Z(\lambda)} exp \sum_j \alpha_j f_j(x, spam)$$

- Doesn't it look familiar?

$$p^*(h, x) = \pi \prod_{j=1}^{k} \lambda_j^{f_j(x,h)}, 0 < \lambda_j < \infty$$

$$\text{where} \sum_j \lambda_j f_j(x, h) = \log(\prod_{j=1}^{k} \alpha_j^{f_j(x,h)}); \pi = \frac{1}{Z(\lambda)}$$

Learning the weights: λ_j: Generalized Iterative Scaling

$$p^*(h,x) = \pi \prod_{j=1}^{k} \lambda_j^{f_j(x,h)}, 0 < \lambda_j < \infty$$

$$\pi = \sum_x \prod_{j=1}^{k} \lambda_j^{f_j(x,h)}$$

Learning the weights: λ_j: Generalized Iterative Scaling

$f^{\#} = max_{x,h} \sum_{j=1}^{k} f_j(x,h)$

For each iteration

    expected[1 .. # of features] $\leftarrow$ 0

    For t = 1 to $|$ training data $|$

        For each feature f_j

            expected[j] += $f_j(x, h_t) \times P(x, h_t)$

    For each feature f_j

        observed[j] = $f_j(x,h) \times \frac{c(x,h)}{|\text{training data}|}$

    For each feature f_j

        $\lambda_i \leftarrow \lambda_i \times \sqrt[f^{\#}]{\frac{\text{observed[j]}}{\text{expected[j]}}}$

cf. Goodman, NIPS ’01

Logistic Regression

- models effects of explanatory variables on binary valued variable
- observations $\mathbf{x} = \{x_1, \ldots, x_j\}$ with success given by $q(\mathbf{x})$:

$$q(\mathbf{x}) = \frac{e^{g(\mathbf{x})}}{1 + e^{g(\mathbf{x})}}$$

and

$$g(\mathbf{x}) = \beta_0 + \sum_{j=1}^{k} \beta_j x_j$$

Logistic Regression

- probability that observations lead to success, or $p(a = 1 \mid b)$:

$$p(a = 1 \mid b) = \frac{e^{g(b)}}{1 + e^{g(b)}}$$

where

$$g(b) = \beta_0 f_0(1, b) + \sum_{j=1}^{k} \beta_j f_j(1, b)$$

- $\beta_j = \log \alpha_j$, $f_0(1, b) = 1$ and $f_j(1, b) = x_j$