• A simple introduction to maximum entropy models for NLP. Adwait Ratnaparkhi.

• A maximum entropy model for part-of-speech tagging. Adwait Ratnaparkhi.
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 $\theta$ 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: $p(\text{semantics}) \times p(\text{syntax} \mid \text{semantics}) \times p(\text{morphology} \mid \text{syntax}) \times p(\text{phonology} \mid \text{morphology}) \times p(\text{sounds} \mid \text{phonology})$
Remember the humble Naive Bayes Classifier

\begin{itemize}
  \item $P(c_k \mid x) = \frac{P(c_k) \times P(x \mid c_k)}{P(x)}$
  \item $P(x \mid c_k) = \prod_{j=1}^{d} P(x_j \mid c_k)$
  \item $P(c_k \mid x) = P(c_k) \times \prod_{j=1}^{d} P(x_j \mid c_k)$
\end{itemize}
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. $25 \times 9 = 225x$ 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 $\text{AMOUNT}$ into
     - in spam: $0.5$  log prob $= -1$
     - in normal email: $0.02$  log prob $= -5.64$

  2. contains $\text{AMOUNT}$
     - 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

• \( \log p(feats \mid spam) = 4.2 \)

• but then, \( p(feats \mid spam) = \exp(4.2) = 66.68 \)

• how do we compute keep arbitrary scores and still get probabilities?
Log linear model

- Renormalize!

- \( p(x \mid spam) = \frac{1}{Z(\lambda)} \exp \sum_i \lambda_i f_i(x) \)
  - \( x \) is the email message
  - \( \lambda_i \) 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 \( \sum_x p(x \mid spam) = 1 \)

- called log-linear: why?
Log linear model

- Now we can get the weights from data

- Choose $\lambda_i$ 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
  \[ p(\text{spam}) \times p(x \mid \text{spam}) \quad \text{vs.} \quad p(\text{normal}) \times p(x \mid \text{normal}) \]

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

- Equivalent to changing features into:
  message is spam \textit{and} contains turn $\text{AMOUNT}$ 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:

\[ p(x \mid \text{spam}) = \frac{1}{Z(\lambda)} \exp \sum_j \alpha_j f_j(x, \text{spam}) \]

- Doesn’t it look familiar?

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

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: $\lambda_j$: Generalized Iterative Scaling

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

$$\pi = \sum_{x} \prod_{j=1}^{k} \lambda_j^{f_j(x, h)}$$
Learning the weights: $\lambda_j$: Generalized Iterative Scaling

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

For each iteration

expected[1 .. # of features] ← 0

For $t = 1$ to $|\text{training data}|$

For each feature $f_j$

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

For each feature $f_j$

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

For each feature $f_j$

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

cf. Goodman, NIPS '01
Logistic Regression

- models effects of explanatory variables on binary valued variable

- observations $x = \{x_1, \ldots, x_j\}$ with success given by $q(x)$:

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

and

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