Probabilistic Pronunciation + N-gram Models

CMSC 35100

Natural Language Processing

April 15, 2003

The ASR Pronunciation Problem

Given a series of phones, what is the most probable word?

Simplification: Assume phone sequence known, word boundaries known

Approach: Noisy channel model

Surface form is an instance of lexical form that has passed through a noisy communication path

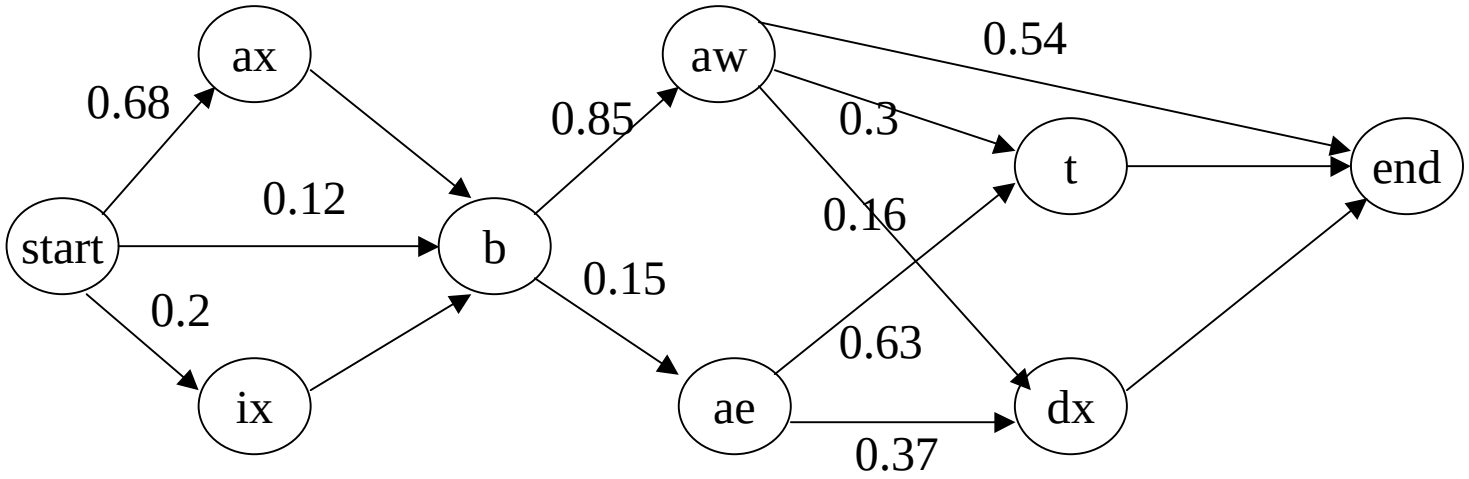
Model channel to remove noise, find original

Bayesian Model

- Pr(w|O) = Pr(O|w)Pr(w)/P(O)
- Goal: Most probable word
  - Observations held constant
  - Find w to maximize Pr(O|w)*Pr(w)
- Where do we get the likelihoods? – Pr(O|w)
  - Probabilistic rules (Labov)
    - Add probabilities to pronunciation variation rules
      - Count over large corpus of surface forms wrt lexicon
- Where do we get Pr(w)?
  - Similarly – count over words in a large corpus

Weighted Automata

- Associate a weight (probability) with each arc
  - Determine weights by decision tree compilation or counting from a large corpus



Computed from Switchboard corpus

Forward Computation

- For a weighted automaton and a phoneme sequence, what is its likelihood?
  - Automaton: Tuple
    - Set of states Q: q0,…qn
    - Set of transition probabilities between states aij,
      - Where aij is the probability of transitioning from state i to j
    - Special start & end states
  - Inputs: Observation sequence: O = o1,o2,…,ok
  - Computed as:
    - forward[t,j] = $P(o1,o2\ldots ot,qt=j|\lambda)p(w)=\Sigma i$ forward[t-1,i]*aij*bjt
      - Sums over all paths to qt=j

Viterbi Decoding

- Given an observation sequence o and a weighted automaton, what is the mostly likely state sequence?
  - Use to identify words by merging multiple word pronunciation automata in parallel
  - Comparable to forward
    - Replace sum with max
- Dynamic programming approach
  - Store max through a given state/time pair

Viterbi Algorithm

Function Viterbi(observations length T, state-graph) returns best-path
Num-states<-num-of-states(state-graph)
Create path prob matrix viterbi[num-states+2,T+2]
Viterbi[0,0]<- 1.0
For each time step t from 0 to T do
  for each state s from 0 to num-states do
    for each transition s' from s in state-graph
      new-score<-viterbi[s,t]*at[s,s']*bs'(ot)
      if ((viterbi[s',t+1]=0) || (viterbi[s',t+1]<new-score))
        then
          viterbi[s',t+1] <- new-score
          back-pointer[s',t+1]<-s
Backtrace from highest prob state in final column of viterbi[] & return

Segmentation

- Breaking sequence into chunks
  - Sentence segmentation
    - Break long sequences into sentences
  - Word segmentation
    - Break character/phonetic sequences into words
      - Chinese: typically written w/o whitespace
        - » Pronunciation affected by units
      - Language acquisition:
        - » How does a child learn language from stream of phones?

Models of Segmentation

- Many:
  - Rule-based, heuristic longest match
- Probabilistic:
  - Each word associated with its probability
  - Find sequence with highest probability
    - Typically compute as log probs & sum
  - Implementation: Weighted FST cascade
    - Each word = chars + probability
    - Self-loop on dictionary
    - Compose input with dict*
    - Compute most likely

N-grams

- Perspective:
  - Some sequences (words/chars) are more likely than others
  - Given sequence, can guess most likely next
- Used in
  - Speech recognition
  - Spelling correction,
  - Augmentative communication
  - Other NL applications

Corpus Counts

- Estimate probabilities by counts in large collections of text/speech
- Issues:
  - Wordforms (surface) vs lemma (root)
  - Case? Punctuation? Disfluency?
  - Type (distinct words) vs Token (total)

Basic N-grams

- Most trivial: 1/#tokens: too simple!
- Standard unigram: frequency
  - # word occurrences/total corpus size
    - E.g. the=0.07; rabbit = 0.00001
  - Too simple: no context!
- Conditional probabilities of word sequences

$$P(w_1^n) = P_n(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^n)$$
$$= \prod_{k=1} P(w_k \mid w_1^{k-1})$$

Markov Assumptions

- Exact computation requires too much data
- Approximate probability given all prior wds
  - Assume *finite* history
  - Bigram: Probability of word given 1 previous
    - First-order Markov
  - Trigram: Probability of word given 2 previous
- N-gram approximation

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

Bigram sequence $$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

Issues

- Relative frequency
  - Typically compute count of sequence
    - Divide by prefix

$$P(w_n \mid w_{n-1}) = \frac{C(w_n w_{n-1})}{C(w_{n-1})}$$

- Corpus sensitivity
  - Shakespeare vs Wall Street Journal
    - Very unnatural
- Ngrams
  - Unigram: little; bigrams: colloc; trigrams:phrase