Hw6
Format of HMM

\text{state\_num}=\text{nn} \quad \#\# \text{the number of states}
\text{sym\_num}=\text{nn} \quad \#\# \text{the size of output symbol alphabet}
\text{init\_line\_num}=\text{nn} \quad \#\# \text{the number of lines for the initial probability}
\text{trans\_line\_num}=\text{nn} \quad \#\# \text{the number of lines for the transition probability}
\text{emiss\_line\_num}=\text{nn} \quad \#\# \text{the number of lines for the emission probability}

\textbf{\texttt{\textbackslash init}} \\
\text{state \ prob \ \ lg\_prob} \quad \#\# \text{prob}=\pi(\text{state}), \ \lg\_prob=\lg(\text{prob})
\ldots

\textbf{\texttt{\textbackslash transition}} \\
\text{from\_state \ to\_state \ \ prob \ \ lg\_prob} \quad \#\# \text{prob}=P(\text{to\_state} \mid \text{from\_state})
\ldots

\textbf{\texttt{\textbackslash emission}} \\
\text{state \ symbol \ \ prob \ \ lg\_prob} \quad \#\# \text{prob}=P(\text{symbol} \mid \text{state})
\ldots
Q1: HMM for a bigram tagger

• cat training_data | create_2gram_hmm.sh output_hmm

• training data: w1/t1 ... wn/tn

• No smoothing
Q2: HMM for a trigram tagger

- cat training_data | create_3gram_hmm.sh output_hmm l1 l2 l3 unk_prob_file

- unk_prob_file has the format “tag prob”, which means $P(\langle unk \rangle | tag) = prob$

- Smoothing:
  \[
P(t_3 | t_1, t_2) = \lambda_3 P_3(t_3 | t_1, t_2) + \lambda_2 P_2(t_3 | t_2) + \lambda_1 P_1(t_3)
  \]
  if w is a known word
  then $P_{smooth}(w | tag) = P(w | tag) * (1 - P(\langle unk \rangle | tag))$
  else $P_{smooth}(w | tag) = P(\langle unk \rangle | tag)$
Q3: read and check HMM

- `check_hmm.sh input_hmm > warning_file`

state_num=6
sym_num=11

warning: different numbers of init_line_num: claimed=2, real=1
warning: different numbers of trans_line_num: claimed=13, real=15
warning: different numbers of emission_line_num: claimed=11, real=12

warning: the trans_prob_sum for state N is 0.9
warning: the trans_prob_sum for state V is 1.1

warning: the emiss_prob_sum for state BOS is 0
warning: the emiss_prob_sum for state N is 0.5
warning: the emiss_prob_sum for state V is 0.85
warning: the emiss_prob_sum for state Adv is 0
Implementation issue: storing HMM

Approach #1: use hash tables
- $\pi_i$: $\{\text{state\_str}\}$
- $a_{ij}$: $\{\text{from\_state\_str}\} \{\text{to\_state\_str}\}$
- $b_{jk}$: $\{\text{state\_str}\} \{\text{symbol}\}$

Approach #2: map a string to an index first
- $\text{state2idx}\{\text{state\_str}\} = \text{state\_idx}$
- $\text{symbol2idx}\{\text{symbol\_str}\} = \text{symbol\_idx}$

- $\pi_i$: $\text{pi}[\text{state\_idx}] = \text{prob}$
- $a_{ij}$: $\text{a}[\text{from\_state\_idx}] [\text{to\_state\_idx}] = \text{prob}$
- $b_{jk}$: $\text{b}[\text{state\_idx}] [\text{symbol\_idx}] = \text{prob}$

- $\text{idx2state}[\text{state\_idx}] = \text{state\_str}$
- $\text{Idx2symbol}[\text{symbol\_idx}] = \text{symbol\_str}$
Storing HMM: sparse matrix

- Two-dimensional array:
  - $a_{ij}$: $a[i][j] = \text{prob}$
  - $b_{jk}$: $b[j][k] = \text{prob}$

- One-dimensional array:
  - $a_{ij}$: $a[i] = \text{“j1 p1 j2 p2 ...”}$, or
  - $a_{ij}$: $a[j] = \text{“i1 p1 i2 p2 ...”}$
  - $b_{jk}$: $b[j] = \text{“k1 p1 k2 p2 ....”}$, or
  - $b_{jk}$: $b[k] = \text{“j1 p1 j2 p2 ...”}$
Other implementation issues

• Index starts from 0 in programming, but often starts from 1 in algorithms

• The sum of \( \log \text{prob} \) is used in practice to replace the product of \text{prob}.

• Check constraints and print out warning if the constraints are not met.