School of Computing and Information Systems
The University of Melbourne
COMP90042
WEB SEARCH AND TEXT ANALYSIS (Semester 1, 2019)

Sample solutions for discussion exercises: Week 9

Discussion

1. What are the assumptions that go into a **Hidden Markov Model**? What is the time complexity of the **Viterbi algorithm**? Is this practical?

   - **Markov assumption**: the likelihood of transitioning into a given state depends only on the current state, and not the previous state(s) (or output(s))
   - **Output independence assumption**: the likelihood of a state producing a certain word (as output) does not depend on the preceding (or following) state(s) (or output(s)).
   - The time complexity of the Viterbi algorithm, for an HMM with T possible states, and a sentence of length W, is $\mathcal{O}(T^2W)$. In POS tagging, there might typically be approximately 100 possible tags (states), and a typical sentence might have 10 or so tokens, so ... yes, it is practical (unless we need to tag really, really, quickly, e.g. tweets as they are posted).

   (a) How can an HMM be used for POS tagging a text? For the purposes of POS tagging:

   - Tokens (possibly ignoring punctuation) correspond to outputs, and POS tags correspond to states of the HMM.
   - We will build a model based around sentences in a tagged corpus. We could instead use a document–based model; in which case, the model will probably discover that the initial distribution of states is very similar to the transition from end–of–sentence punctuation (`.?!` etc.), which probably represents a needless complication.

   i. How can the initial state probabilities π be estimated?
   - Record the distribution of tags for the first token of each sentence in a tagged corpus.

   ii. How can the transition probabilities A be estimated?
   - For each tag, record the distribution of tags of the immediately following token in the tagged corpus. (We might need to introduce an end–of–sentence dummy for the probabilities to add up correctly.)

   iii. How can the emission probabilities B be estimated?
   - For each tag, record the distribution of corresponding tokens in the tagged corpus.

   (b) Estimate π, A and B for POS tagging, based on the following tagged corpus:

```
1. silver-JJ wheels-NNS turn-VBP
2. wheels-NNS turn-VBP right-JJ
3. right-JJ wheels-NNS turn-VBP
```

- For π, there are three sentences: two begin with `JJ` and one with `NNS`. Consequently:

$$\pi[JJ, NNS, VBP] = [\frac{2}{3}, \frac{1}{3}, 0]$$

- For A, we need to observe the distribution for each tag. Even with such a small collection, this is somewhat tedious:
  - For `JJ`, two instances are immediately followed by `NNS` and one instance occurs at the end of the sentence.
  - For `NNS`, all three instance are followed by `VBP`.
  - For `VBP`, it is followed by `JJ` once, and the end of the sentence twice.
- If we don't use an explicit end–of–sentence marker, we end up with the (somewhat ill-formed) A on the left; if we include it, we end up with the A on the right (note that `EOS` doesn't require it's own transition probabilities):

A	JJ	NNS	VBP
(from) JJ	0	1	0
NNS	0	0	1
VBP	1	0	0

A	JJ	NNS	VBP	EOS
JJ	0	$\frac{2}{3}$	0	$\frac{1}{3}$
NNS	0	0	1	0
VBP	$\frac{1}{3}$	0	0	$\frac{2}{3}$

- For B, we can simply read off the corresponding words for each tag in the corpus:
  - For `JJ`, there is one instance of `silver` and two of `right`.
  - For `NNS`, there are three instances of `wheels`.
  - For `VBP`, there are three instances of `turn`.
- Consequently:

B	right	silver	turn	wheels
(from) JJ	$\frac{2}{3}$	$\frac{1}{3}$	0	0
NNS	0	0	0	1
VBP	0	0	1	0

2. Consider using the following Hidden Markov Model to tag the sentence `silver wheels turn`:
   π[JJ, NNS, VBP] $= [0.3, 0.4, 0.3]$

A	JJ	NNS	VBP
JJ	0.4	0.5	0.1
NNS	0.1	0.4	0.5
VBP	0.4	0.5	0.1

B	silver	wheels	turn
JJ	0.8	0.1	0.1
NNS	0.3	0.4	0.3
VBP	0.1	0.3	0.6

   (a) Visualise the HMM as a graph.

   (b) Use the **Viterbi algorithm** to find the most likely tag for this sequence.

   - The most likely tag sequence can be read right-to-left, based upon the maximum probability we've observed: in this case, 0.0072 when *turn* is a VBP; this value is derived from the NNS $\rightarrow$ VBP transition, so we can infer that *wheels* is an NNS; that in turn comes from the JJ $\rightarrow$ NNS transition, so *silver* is a JJ.

α	1:*silver*		*2:wheels*	3:*turn*
JJ:	JJ	π[JJ]B[JJ, silver] $0.3 \times 0.8 = 0.24$		
NNS:	NNS	π[NNS]B[NNS, silver] $0.4 \times 0.3 = 0.12$		
VBP:	VBP	π[VBP]B[VBP, silver] $0.3 \times 0.1 = 0.03$		

α	*1:silver*	*2:wheels*		*3:turn*
JJ:	0.24	JJ $\rightarrow$ JJ	A[JJ,JJ]B[JJ, wheels]	
	0.24		$\times 0.4 \times 0.1 =$ **0.0096**	
		NNS $\rightarrow$ JJ	A[NNS,JJ]B[JJ, wheels]	
		0.12	$\times 0.1 \times 0.1 = 0.0012$	
		VBP $\rightarrow$ JJ	A[VBP,JJ]B[JJ, wheels]	
		0.03	$\times 0.4 \times 0.1 = 0.0012$	
NNS:	0.12	JJ $\rightarrow$ NNS	A[JJ,NNS]B[NNS, wheels]	
		0.24	$\times 0.5 \times 0.4 =$ **0.048**	
		NNS $\rightarrow$ NNS	A[NNS,NNS]B[NNS, wheels]	
		0.12	$\times 0.4 \times 0.4 = 0.0192$	
		VBP $\rightarrow$ NNS	A[VBP,NNS]B[NNS, wheels]	
		0.03	$\times 0.5 \times 0.4 = 0.006$	
VBP:	0.03	JJ $\rightarrow$ VBP	A[JJ,VBP]B[VBP, wheels]	
		0.24	$\times 0.1 \times 0.3 = 0.0072$	
		NNS $\rightarrow$ VBP	A[NNS,VBP]B[VBP, wheels]	
		0.12	$\times 0.5 \times 0.3 =$ **0.018**	
		VBP $\rightarrow$ VBP	A[VBP,VBP]B[VBP, wheels]	
		0.03	$\times 0.1 \times 0.3 = 0.0009$	

α	*1:silver*	*2:wheels*	*3:turn*	
JJ:	0.24	0.0096	JJ $\rightarrow$ JJ	A[JJ,JJ]B[JJ, turn]
		JJ $\rightarrow$ JJ	0.0096	$\times 0.4 \times 0.1 = 0.000384$
			NNS $\rightarrow$ JJ	A[NNS,JJ]B[JJ, turn]
			0.048	$\times 0.1 \times 0.1 = 0.00048$
			VBP $\rightarrow$ JJ	A[VBP,JJ]B[JJ, turn]
			0.018	$\times 0.4 \times 0.1 =$ **0.00072**
NNS:	0.12	0.048	JJ $\rightarrow$ NNS	A[JJ,NNS]B[NNS, turn]
		JJ $\rightarrow$ NNS	0.0096	$\times 0.5 \times 0.3 = 0.00144$
			NNS $\rightarrow$ NNS	A[NNS,NNS]B[NNS, turn]
			0.048	$\times 0.4 \times 0.3 =$ **0.00576**
			VBP $\rightarrow$ NNS	A[VBP,NNS]B[NNS, turn]
			0.018	$\times 0.5 \times 0.3 = 0.0027$
VBP:	0.03	0.018	JJ $\rightarrow$ VBP	A[JJ,VBP]B[VBP, turn]
		NNS $\rightarrow$ VBP	0.0096	$\times 0.1 \times 0.6 = 0.000576$
			NNS $\rightarrow$ VBP	A[NNS,VBP]B[VBP, turn]
			0.048	$\times 0.5 \times 0.6 =$ **0.0144**
			VBP $\rightarrow$ VBP	A[VBP,VBP]B[VBP, turn]
			0.018	$\times 0.1 \times 0.6 = 0.00108$

3. What are regular grammar and regular language? How are they different?

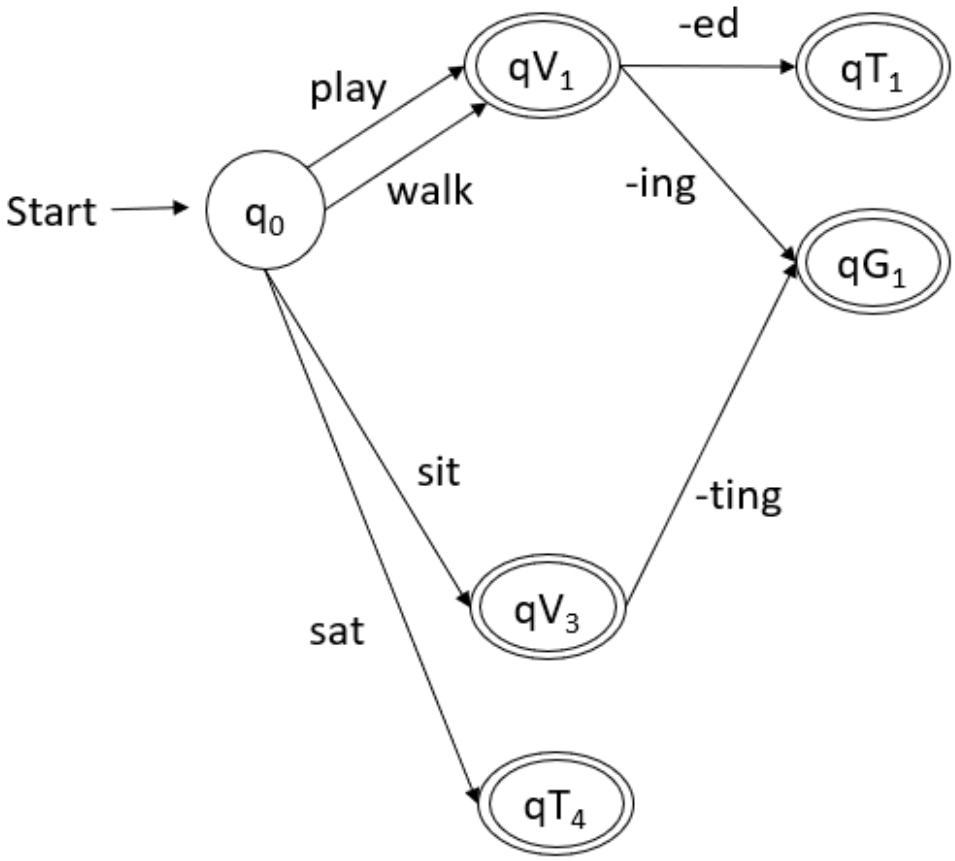
   - A language is a set of acceptable strings and a grammar is a generative description of a language.
   - Regular language is a formal language that can be expressed using a regular expression.
   - Regular grammar is a formal grammar defined by a set of productions rules in the form of A $\rightarrow$ xB, A $\rightarrow$ x and A $\rightarrow \epsilon$, where A and B are non-terminals, x is a terminal and ϵ is the empty string.
   - A language is regular if and only if it can be generated by a regular grammar.
   - For example: A simple regular grammar
     - Rules: S $\rightarrow$ A, A $\rightarrow$ a A, A $\rightarrow \epsilon$
     - S is the start symbol
     - It will generate words such as a, aa, aaa, aaa.
     - The set of words generated by this regular grammar is a regular language.
     - This regular language can also be expressed in regular expression (a)*.

   (a) Regular languages are closed under union, intersection and concatenation. What does it mean? Why is it important?

   - This means that if L1 and L2 are two regular languages, then L1 $\cup$ L2, L1 $\cap$ L2, and the language strings that are the concatenation of L1 and L2 are also regular languages.
   - This closure property allows us to apply operations on regular languages to produce a regular language.
   - This allows for NLP problems to be factored into small simple parts, such that we can develop regular languages for each part, and combine them into a complex system to handle the NLP problems. This is particularly relevant for transducers and the composition operation, which are used in many NLP pipelines. (Note that FSTs implement "regular relations" rather that regular languages, but the distinction is a bit subtle and is not something we will dive into.)

   (b) Draw a Finite State Acceptor (FSA) for word morphology to show the possible derivations from root forms using the words: play, played, playing; walk, walked, walking; sit, sat, sitting.

   - There are many correct answers. Below is one:



(c) What are Weighted Finite State Acceptors (WFSAs)? When and why are they useful?

- WFSAs are generalizations of FSAs, with each path assigned a score, computed from the transitions, the initial state, and the final state. The total score for any path is equal to the sum of the scores of the initial state, the transitions, and the final state.
- WFSAs can produce a score for each valid word for sub-word decomposition problems or sentence for words-in-sentence problems, while FSAs have no way to express preferences among technically valid words or sentences.
- For example, WFSAs can assign scores to all strings of characters forming words, so that spelling mistakes, new words, or strange-but-acceptable words can still be handled.
- The same argument holds for sequences of words forming sentences. Clearly some word sequences are gibberish, but being able to provide a numerical score can help in many applications, like how LMs can be used in sentence generation, speech recognition, OCR, translation, etc.