School of Computing and Information Systems
The University of Melbourne
COMP90042
WEB SEARCH AND TEXT ANALYSIS (Semester 1, 2017)

Sample solutions for discussion exercises: Week 3

Discussion

1. What is a **POS tag**?

   - A POS tag is a label assigned to a token in a sentence which indicates some grammatical (primarily syntactic) properties of its function in the sentence.

   (a) What are some common approaches to POS tagging? What aspects of the data might allow us to predict POS tags systematically?

   - **Unigram**: Assign a POS tag to a token according to the most common observation in a tagged corpus; many words are unambiguous, or almost unambiguous.
   - **N-gram**: Assign a POS tag to a token according to the most common tag in the same sequence (based on the sentence in which the token occurs) of n tokens (or tags) in the tagged corpus; context helps disambiguate.
   - **Hand-coding**: Write rules (relying on expertise of the writer) that disambiguate unigram tags.
   - **Transformational**: Similar to hand-coding, but rules are derived automatically (using templates) by observing mistakes of the unigram tagger on a tagged corpus.
   - **Sequential**: Learn a Hidden Markov Model (or other model) based on the observed tag sequences in a tagged corpus.
   - **Classifier**: Treat as a supervised machine learning problem, with tags from a tagged corpus as training data.

   (b) POS tag (by hand) the following sentence: `Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.` according to the Penn Treebank tags. (Note that some of the tags are somewhat obscure.)

   - According to the Penn Treebank:

```
NNP Pierre
NNP Vinken
  , ,
 CD 61
NNS years
 JJ old
  , ,
 MD will
 VB join
 DT the
 NN board
 IN as
 DT a
 JJ nonexecutive
 NN director
NNP Nov.
 CD 29
  . .
```

2. 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 O be estimated?
      - For each tag, record the distribution of corresponding tokens in the tagged corpus.

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

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

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

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

- For A, we need to observe the distribution for each tag. Even with such a small collection, this is somewhat tedious:
  - For `JJ`, both instances are immediately followed by `NN`.
  - For `NN`, all three instance are followed by `VBP`.
  - For `RB`, it only occurs at the end of the sentence; there is no immediate following context.
  - For `VBP`, it is followed by `RB` once, and the end of the setence 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	NN	RB	VBP
(from) JJ	0	1	0	0
NN	0	0	0	1
RB	0	0	0	0
VBP	0	0	1	0

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

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

O	right	silver	turn	wheels
(from) JJ	$\frac{1}{2}$	$\frac{1}{2}$	0	0
NN	0	0	0	1
RB	1	0	0	0
VBP	0	0	1	0

3. Consider using the following Hidden Markov Model to tag the sentence `silver wheels turn`:

$\pi[\text{JJ}, \text{NN}, \text{VBP}] = [0.3, 0.4, 0.3]$

A	JJ	NN	VBP
JJ	0.4	0.5	0.1
NN	0.1	0.4	0.5
VBP	0.4	0.5	0.1

O	silver	wheels	turn
JJ	0.8	0.1	0.1
NN	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.

- See overleaf.
- 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 NN $\rightarrow$ VBP transition, so we can infer that *wheels* is an NN; that in turn comes from the JJ $\rightarrow$ NN transition, so *silver* is a JJ.

α	*1:silver*		*2:wheels*	*3:turn*
JJ:	JJ	π[JJ]O[JJ, silver]		
		$0.3 \times 0.8 = 0.24$		
NN:	NN	π[NN]O[NN, silver]		
		$0.4 \times 0.3 = 0.12$		
VBP:	VBP	π[VBP]O[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]O[JJ, wheels]	
		0.24	$\times 0.4 \times 0.1 =$ **0.0096**	
		NN $\rightarrow$ JJ	A[NN,JJ]O[JJ, wheels]	
		0.12	$\times 0.1 \times 0.1 = 0.0012$	
		VBP $\rightarrow$ JJ	A[VBP,JJ]O[JJ, wheels]	
		0.03	$\times 0.4 \times 0.1 = 0.0012$	
NN:	0.12	JJ $\rightarrow$ NN	A[JJ,NN]O[NN, wheels]	
		0.24	$\times 0.5 \times 0.4 =$ **0.048**	
		NN $\rightarrow$ NN	A[NN,NN]O[NN, wheels]	
		0.12	$\times 0.4 \times 0.4 = 0.0192$	
		VBP $\rightarrow$ NN	A[VBP,NN]O[NN, wheels]	
		0.03	$\times 0.5 \times 0.4 = 0.006$	
VBP:	0.03	JJ $\rightarrow$ VBP	A[JJ,VBP]O[VBP, wheels]	
		0.24	$\times 0.1 \times 0.3 = 0.0072$	
		NN $\rightarrow$ VBP	A[NN,VBP]O[VBP, wheels]	
		0.12	$\times 0.5 \times 0.3 =$ **0.018**	
		VBP $\rightarrow$ VBP	A[VBP,VBP]O[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]O[JJ, turn]
		JJ $\rightarrow$ JJ	0.0096	$\times 0.4 \times 0.1 = 0.000384$
			NN $\rightarrow$ JJ	A[NN,JJ]O[JJ, turn]
			0.048	$\times 0.1 \times 0.1 = 0.00048$
			VBP $\rightarrow$ JJ	A[VBP,JJ]O[JJ, turn]
			0.018	$\times 0.4 \times 0.1 =$ **0.00072**
NN:	0.12	0.048	JJ $\rightarrow$ NN	A[JJ,NN]O[NN, turn]
		JJ $\rightarrow$ NN	0.0096	$\times 0.5 \times 0.3 = 0.00144$
			NN $\rightarrow$ NN	A[NN,NN]O[NN, turn]
			0.048	$\times 0.4 \times 0.3 =$ **0.00576**
			VBP $\rightarrow$ NN	A[VBP,NN]O[NN, turn]
			0.018	$\times 0.5 \times 0.3 = 0.0027$
VBP:	0.03	0.018	JJ $\rightarrow$ VBP	A[JJ,VBP]O[VBP, turn]
		NN $\rightarrow$ VBP	0.0096	$\times 0.1 \times 0.3 = 0.000288$
			NN $\rightarrow$ VBP	A[NN,VBP]O[VBP, turn]
			0.048	$\times 0.5 \times 0.3 =$ **0.0072**
			VBP $\rightarrow$ VBP	A[VBP,VBP]O[VBP, turn]
			0.018	$\times 0.1 \times 0.3 = 0.00054$

Programming

1. In the iPython notebook `hidden_markov_models.ipynb`:
   - The Viterbi algorithm is implemented with loops. Try to implement Viterbi using recursion instead.
   - Can you see the difference between the speed of the Viterbi algorithm and the exhaustive search over the lattice? How much faster is Viterbi than exhaustive search on an example problem? (hint: *time* or *clock* functions from the *time* package can be useful)

Catch-up

- Revise the terms **noun**, **verb**, **adjective**, and **determiner**.
- What is **lemmatisation**? Why would it be easier if we knew in advance that a token was a noun (or verb, etc.)?
- What is an n-gram?
- Who left waffles on the Falkland Islands? Why?
- What is the **BIO labelling trick** and why is it important?

Get ahead

- NLTK has extensive support for POS tagging. Have a read through Chapter 5 of the NLTK book. (`http://www.nltk.org/book/ch05.html`)
- Work through the advanced material in the `hidden_markov_models.ipynb` notebook.