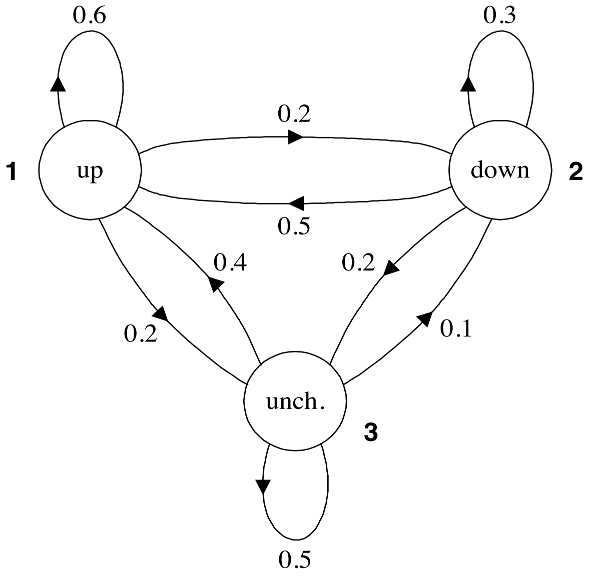
Sequence Tagging: hidden Markov models

COMP90042 Lecture 15



POS tagging recap

- Janet will back the bill
- Janet/NNP will/MB back/VP the/DT bill/NN
- Local classifier: prone to **error propagation**
- What about treating the full sequence as a “class”?
  - Output: “NNP_MB_VP_DT_NN”
- Problems:
  - Exponentially many combinations: $|Tags|^M$,for length M
  - How to tag sequences of different lengths?

A better approach

- Tagging is a sentence-level task but as humans we **decompose** it into small word-level tasks.
  - Janet/NNP will/MB back/VP the/DT bill/NN
- Solution:
  - Define a model that decomposes process into individual word level steps
  - But that takes into account the whole sequence when learning and predicting (no error propagation)
- This is the idea of **sequence labelling**, and more general, **structured prediction.**

A probabilistic model

- Goal: obtain best tag sequence **t** from sentence **w**
  - $\hat{\boldsymbol{t}} = argmax_{\boldsymbol{t}}\, P(\boldsymbol{t}|\boldsymbol{w})$
  - $\hat{\boldsymbol{t}} = argmax_{\boldsymbol{t}}\, \frac{P(\boldsymbol{w}|\boldsymbol{t})P(\boldsymbol{t})}{P(\boldsymbol{w})} = argmax_{\boldsymbol{t}}\, P(\boldsymbol{w}|\boldsymbol{t})\, P(\boldsymbol{t})$ [Bayes]
- Let's decompose:
  - $P(\boldsymbol{w}|\boldsymbol{t}) = \prod_{i=1}^{n} P(w_i|t_i)$ [Prob. of a word depends only on the tag]
  - $P(\boldsymbol{t}) = \prod_{i=1}^{n} P(t_i|t_{i-1})$ [Prob. of a tag depends only on the previous tag]
- These are **independence assumptions** (remember Naïve Bayes? Language models?)
- This is a Hidden Markov Model (HMM)

Hidden Markov model

$$\hat{\boldsymbol{t}} = argmax_{\boldsymbol{t}}\, P(\boldsymbol{w}|\boldsymbol{t})\, P(\boldsymbol{t})$$

$$P(\boldsymbol{w}|\boldsymbol{t}) = \prod_{i=1}^{n} P(w_i|t_i)$$

$$P(\boldsymbol{t}) = \prod_{i=1}^{n} P(t_i|t_{i-1})$$

- Why “Markov”?
  - Because it assumes the sequence follows a Markov chain: probability of an event (tag) depends only on the previous event (last tag)
- Why “Hidden”?
  - Because the events (tags) are not seen: goal is to find the best sequence

HMMs - training

- Parameters are the individual probabilities $P(w_i|t_i)$ and $P(t_i|t_{i-1})$
  - Respectively, **emission** (*O*) and **transition** (*A*) probabilities
- Training uses Maximum Likelihood Estimation (MLE)
  - In Naïve Bayes & n-gram LMs, this is done by simply counting word frequencies according to the class.
- We do **exactly the same** in HMMs!
  - $P(like|VB) = \frac{count(VB,like)}{count(VB)}$
  - $P(NN|DT) = \frac{count(DT,NN)}{count(DT)}$

HMMs - training

- What about the first tag?
  - Assume we have a symbol “<s>” that represents the start of your sentence.

$$P(NN| < s >) = \frac{count(< s >,\ NN)}{count(< s >)}$$

- What about the last tag?
  - Assume we have a symbol “</s>” that represents the end of sentence.
- What about unseen (word,tag) and (tag, previous) combinations?
  - Smoothing techniques, like NB/n-gram LMs

Transition Matrix

	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 10.5 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

Emission (observation) Matrix

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 10.6 Observation likelihoods *B* computed from the WSJ corpus without smoothing.

HMMs – prediction (decoding)

$$\hat{\boldsymbol{t}} = argmax_{\boldsymbol{t}}\, P(\boldsymbol{w}|\boldsymbol{t})\, P(\boldsymbol{t})$$

$$= argmax_{\boldsymbol{t}}\, \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

- Simple idea: for each word, take the tag that maximises $P(w_i|t_i)P(t_i|t_{t-1})$. Do it left-to-right, in *greedy* fashion.
- This is wrong! We are looking for $\mathrm{argmax}_{\boldsymbol{t}}$, not individual $argmax_{t_i}$ terms.
  * This is a local classifier: error propagation
- Correct way: take **all** possible tag combinations, evaluate them, take the max (like Naïve Bayes)
  * Problem: exponential number of sequences.

The Viterbi algorithm

- Dynamic Programming to the rescue!
  - We can still proceed sequentially, as long as we careful.
- "can play" -> can/MD play/VB
- Best tag for "can" is easy: $argmax_t\ P(\text{can}|t)P(t|\text{<s>})$
  - We can do that because first "tag" is always "<s>"
- Suppose best tag for "can" is NN. To get the tag for "play", we can take $argmax_t\ P(\text{play}|t)P(t|\text{NN})$ but this is wrong.
- Instead, we keep track of **scores for each tag** for "can" and check **what would happen** if "can" had a different tag.

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP					
MD					
VB					
JJ					
NN					
RB					
DT					

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	P(Janet|NNP) * P(NNP|<s>)				
MD	P(Janet|MD) * P(MD|<s>)				
VB	…				
JJ	…				
NN	…				
RB	…				
DT	…				

The viterbi algorithm

	Janet	will	back	the	bill
NNP	0.000032 * 0.2767				
MD	0 * 0.0006				
VB	…				
JJ	…				
NN	…				
RB	…				
DT	…				

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	8.8544e-06				
MD	0				
VB	0				
JJ	0				
NN	0				
RB	0				
DT	0				

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	P(will|NNP) * P(NNP|t_{Janet}) * s(t_{Janet}|Janet)			
MD	0	…			
VB	0	…			
JJ	0	…			
NN	0	…			
RB	0	…			
DT	0	…			

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	P(will|NNP) * P(NNP|t_{Janet}) * s(t_{Janet}|Janet)			
MD	0	…			
VB	0	…			
JJ	0	…			
NN	0	…			
RB	0	…			
DT	0	…			

Calculate this for all tags, take the max.

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	0 * $P(NNP\|t_{Janet})$ * $s(t_{Janet}\|Janet)$			
MD	0	…			
VB	0	…			
JJ	0	…			
NN	0	…			
RB	0	…			
DT	0	…			

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	8.8544e-06 ●	0			
MD	0	P(will|MD) * P(MD|t_{Janet}) * s(t_{Janet}|Janet)			
VB	0	…			
JJ	0	…			
NN	0	…			
RB	0	…			
DT	0	…			

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	0			
MD	0	3.004e-8			
VB	0	…			
JJ	0	…			
NN	0	…			
RB	0	…			
DT	0	…			

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	8.8544e-06	0			
MD	0	3.004e-8			
VB	0	2.231e-13			
JJ	0	0			
NN	0	1.034e-10			
RB	0	0			
DT	0	0			

The iterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06 ●	0			
MD	0	3.004e-8			
VB	0	2.231e-13			
JJ	0	0			
NN	0	1.034e-10			
RB	0	0			
DT	0	0			

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	0	0		
MD	0	3.004e-8	0		
VB	0	2.231e-13	P(back|VB) * P(VB|t_{will}) * s(t_{will}|will)		
JJ	0	0			
NN	0	1.034e-10			
RB	0	0			
DT	0	0			

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	0	0		
MD	0	3.004e-8	0		
VB	0	2.231e-13	**MD: 1.6e-11** **VB: 7.5e-19** **NN: 9.7e-17**		
JJ	0	0			
NN	0	1.034e-10			
RB	0	0			
DT	0	0			

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	8.8544e-06	0	0		
MD	0	3.004e-8	0		
VB	0	2.231e-13	**MD: 1.6e-11** **VB: 7.5e-19** **NN: 9.7e-17**		
JJ	0	0			
NN	0	1.034e-10			
RB	0	0			
DT	0	0			

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	0	0		
MD	0	3.004e-8	0		
VB	0	2.231e-13	1.6e-11		
JJ	0	0			
NN	0	1.034e-10			
RB	0	0			
DT	0	0			

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	0	0		
MD	0	3.004e-8	0		
VB	0	2.231e-13	1.6e-11		
JJ	0	0	5.1e-15		
NN	0	1.034e-10	5.4e-15		
RB	0	0	5.3e-11		
DT	0	0	0		

The viterbi algorithm

	Janet	will	back	the	bill
NNP	8.8544e-06	0	0	2.5e-17	
MD	0	3.004e-8	0	0	
VB	0	2.231e-13	1.6e-11	0	
JJ	0	0	5.1e-15	5.2e-16	
NN	0	1.034e-10	5.4e-15	5.9e-18	
RB	0	0	5.3e-11	0	
DT	0	0	0	1.8e-12	

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	8.8544e-06	0	0	2.5e-17	0
MD	0	3.004e-8	0	0	0
VB	0	2.231e-13	1.6e-11	0	1.0e-20
JJ	0	0	5.1e-15	5.2e-16	0
NN	0	1.034e-10	5.4e-15	5.9e-18	2.0e-15
RB	0	0	5.3e-11	0	0
DT	0	0	0	1.8e-12	0

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	8.8544e-06	0	0	2.5e-17	0
MD	0	3.004e-8	0	0	0
VB	0	2.231e-13	1.6e-11	0	1.0e-20
JJ	0	0	5.1e-15	5.2e-16	0
NN	0	1.034e-10	5.4e-15	5.9e-18	**2.0e-15**
RB	0	0	5.3e-11	0	0
DT	0	0	0	1.8e-12	0

The viterbi algorithm

	Janet	**will**	**back**	**the**	**bill**
NNP	8.8544e-06	0	0	2.5e-17	0
MD	0	3.004e-8	0	0	0
VB	0	2.231e-13	1.6e-11	0	1.0e-20
JJ	0	0	5.1e-15	5.2e-16	0
NN	0	1.034e-10	5.4e-15	5.9e-18	**2.0e-15**
RB	0	0	5.3e-11	0	0
DT	0	0	0	1.8e-12	0

The viterbi algorithm

	Janet/NNP	**will/MD**	**back/VB**	**the/DT**	**bill/NN**
NNP	8.8544e-06	0	0	2.5e-17	0
MD	0	3.004e-8	0	0	0
VB	0	2.231e-13	1.6e-11	0	1.0e-20
JJ	0	0	5.1e-15	5.2e-16	0
NN	0	1.034e-10	5.4e-15	5.9e-18	**2.0e-15**
RB	0	0	5.3e-11	0	0
DT	0	0	0	1.8e-12	0

The Viterbi algorithm

- Complexity: $O(T^2N)$, where T is the size of the tagset and N is the length of the sequence.
  - T * N matrix, each cell performs T operations.
- Why does it work?
  - Because of the **independence assumptions** that decompose the problem (specifically, the Markov property). Without these, we cannot apply DP.

Viterbi Pseudocode

```
alpha = np.zeros(M, T)
for t in range(T):
  alpha[1, t] = pi[t] * O[w[1], t]

for i in range(2, M):
  for t_i in range(T):
    for t_last in range(T):        # t_last means t_{i-1}
      s = alpha[i-1, t_last] * A[t_last, t_i] * O[w[i], t_i]
      if s > alpha[i,t_i]:
        alpha[i,t_i] = s
        back[i,t_i] = t_last

best = np.max(alpha[M-1,:])
return backtrace(best, back)
```

- Good practice: work with **log** probabilities to prevent underflow (multiplications become sums)
- Vectorisation (use matrix-vector operations)

HMMs in practice

- We saw HMM taggers based on **bigrams.** State-of-the-art use tag **trigrams.**
  - $P(\boldsymbol{t}) = \prod_{i=1}^{n} P(t_i | t_{i-1}, t_{i-2})$ Viterbi now O(T^{3}N)
- Need to deal with sparsity: some tag trigram sequences might not be present in training data
  - Backoff: $P(t_i | t_{i-1}, t_{i-2})$ = $\lambda_3 \hat{P}(t_i | t_{i-1}, t_{i-2})$ + $\lambda_2 \hat{P}(t_i | t_{i-1})$ + $\lambda_1 \hat{P}(t_i)$
  - $\lambda_1 + \lambda_2 + \lambda_3$ = 1
  - Can learn the weights using **deleted interpolation**.
- With additional features, reach 96.5% accuracy on Penn Treebank (Brants, 2000)

Other variant Taggers

- HMM is **generative**, *P(**t**, **w**)*, ‘creates’ the input
  - allows for unsupervised HMMs: learn model without any tagged data!
- **Discriminative** models describe *P(**t** | **w**)* directly
  - supports richer feature set, generally better accuracy when trained over large supervised datasets
  - E.g., Maximum Entropy Markov Model (MEMM), Conditional random field (CRF), Connectionist Temporal Classification (CTC)
  - Most *deep learning* models of sequences are discriminative (e.g., encoder-decoders for translation), similar to an MEMM

HMMs in NLP

- HMMs are highly effective for part-of-speech tagging
  - trigram HMM gets 96.5% accuracy (TnT)
  - related models are state of the art
    - MEMMs 97%
    - CRFs 97.6%
    - Deep CRF 97.9%
  - *English Penn Treebank* tagging accuracy
    https://aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art)
- Apply out-of-the box to other sequence labelling tasks
  - named entity recognition, shallow parsing, alignment …
  - In other fields: DNA, protein sequences, image lattices…

A final word

- HMMs are a simple, yet effective way to perform sequence labelling.
- Can still be competitive, and fast. Natural baseline for other sequence labelling tasks.
- Main drawback: not very flexible in terms of feature representation, compared to MEMMs and CRFs.

Readings

- JM3 Appendix A A.1-A.2, A.4
- See also E18, parts of Chapter 7
- References:
  - Rabiner's HMM tutorial http://tinyurl.com/2hqaf8
  - Lafferty et al, Conditional random fields: Probabilistic models for segmenting and labeling sequence data (2001)