Markov Models for NLP: an Introduction

J. Savoy
Université de Neuchâtel

C. D. Manning & H. Schütze : *Foundations of statistical natural language processing*. The MIT Press, Cambridge (MA)

P. M. Nugues: *An introduction to language processing with Perl and Prolog*. Springer, Berlin

Markov Models

- Different possible models
- Classical (visible, discrete) Markov Models (MM) (chains)
- Based on a set of states
- Transitions from one state to the other at each “period”
- The transitions are random (stochastic model)
- Modeling the system in terms of
  - states
  - change from one state to the other
- *Memoryless* property: the future depends only of the current state (not of all previous states)

Markov Models

- Transitions from one state to the other is a probabilistic one
- Interesting questions:
- Compute the probability of being in a given state in the next step / in the next two steps
- Compute the probability of a given sequence of states
- Examples:
  - Generating a DNA sequence
  - Decoding a DNA sequence

Example: DNA sequence

To encode the genetic information (DNA or RNA), cells are using the genetic code based on four nucleotides

A	Adenine
G	Guanine
C	Cytosine
T	Thymine

The system is based on sequences of three nucleotides (from “AAA”, “AAG”, …, “TTC”, “TTT”, $4^3 = 64$) (defining a codon to denote either an amino acid or a stop signal).

Decoding a DNA sequence means finding the start / stop signal and the sequence different amino acids.

How can we model this system as a Markov Model?

Example

- Where are the states?
- Where are the (random) transitions?
- The four different nucleotides (letters A, C, G, T) represent the four states
- We assume that going from one state to the other is a random process. We are not sure about the future (it is not deterministic).
- Big idea: modeling the randomness by using numbers (probability) to represent our certainty (uncertainty) about an event.

  “Prob[A]” or “P[A]“ (or “P(A)”)

  where A denotes the corresponding event

Random Process

- We are using three axioms:

1. $0 \le \text{Prob}[A] \le 1$
   The measure is limited between 0 and 1.
2. Prob [certain event] = 1 (or Prob [Ω] = 1)
   For an event that is absolutely sure, we assign a probability of 1.
3. If A and B are two mutually exclusive events
   Prob [A or B] = Prob [A] + Prob [B]
   There is nothing in common between A and B
   (the additive law)

- But noting is specified about "how to obtain the probability"

Random Process

- Define your own random process: the dice problem. Six possible outcomes, define their probabilities.

1. Subjective approach
2. Common sense
3. Frequentist: doing some experiments and evaluate the probability as
   Prob [A] = #outcome A / number of trials

- What is our random model for throwing a dice?

Random Process

- By common sense
- Six possible outcomes

  Prob[“1”] = Prob[“2”] = Prob[“3”] =
  Prob[“4”] = Prob[“5”] = Prob[“6”]
- Their sum must be equal to 1.
- Thus we must have: Prob[“1”] = ⅙
- Other computations

  Prob[“1” or “2”] = 1/6 + 1/6 = 2/6 = ⅓

  Prob[“1” or “3” or “4”] = 1/6 + 1/6 + 1/6 = 3/6 = ½

  Prob[“even”] = #even / # possible = 3/6 = ½

Random Process

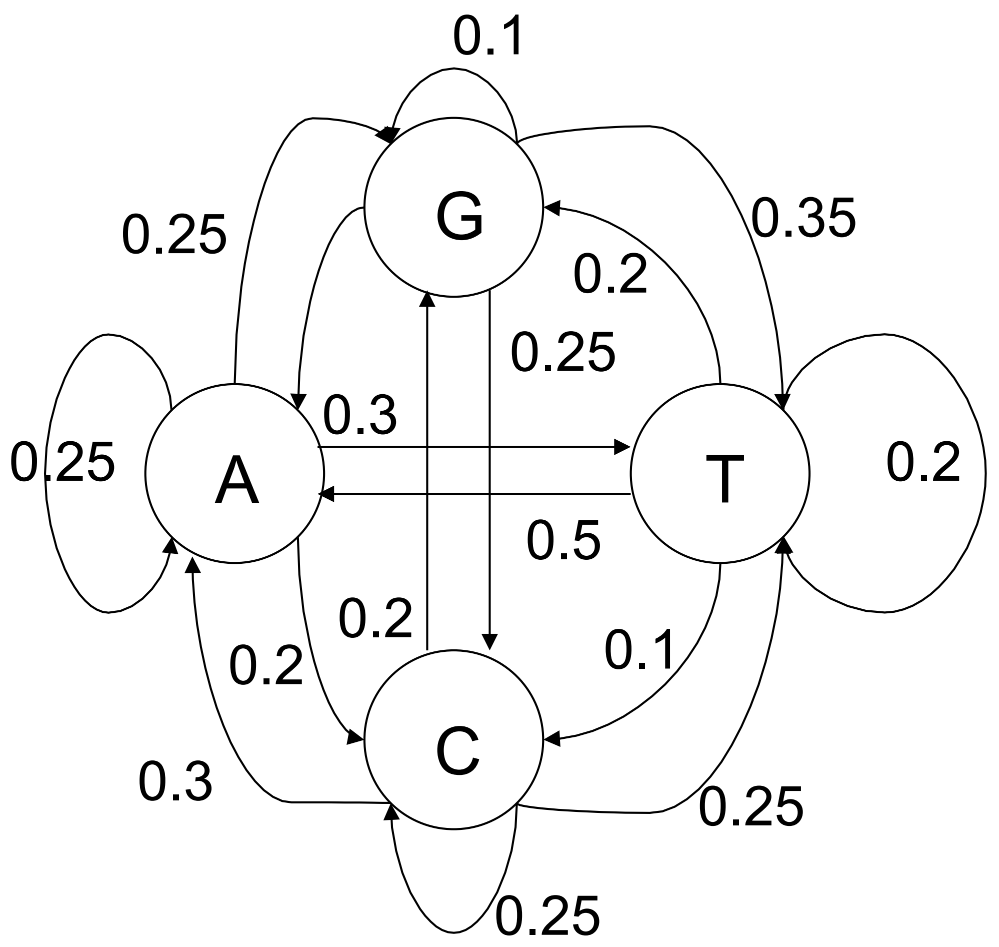
- Probability of an even outcome in the first trial, and an even outcome in the second trial?
- First reasoning:
  Enumerate all possible outcomes
  {1,2}, {1,3}, {1,4}, ... {6,4}, {6,5}, {6,6}
  Prob [“even,even”] = #even,even / #possible = 9/36 = 2/13 =¼
- Second reasoning: the multiplicative law
  The outcome (“even”/”odd”) in the first trial has no influence on the outcome in the second trial (the dice does not have any memory). The two events are *independent*.
  Prob[“even,even”] = Prob[“even”]·Prob[“even”] = ½ · ½ = ¼

Random Process

- Back to our DNA problem...
- Where are the states of our Markov model?
- Where are the (random) transitions?
- The four different nucleotides (letters A, C, G, T) represent the four states
- We assume that going from one state to the other is a random process

From \ To	A	C	G	T
A	0.25	0.2	0.25	0.3
C	0.3	0.25	0.2	0.25
G	0.3	0.25	0.1	0.35
T	0.5	0.1	0.2	0.2

Markov Example



Generating a DNA sequence

The sequences AAT or AAC define the asparagine.

Markov Example

- We can define various sequences of three letters such as
  The sequences TAT or TAC define the tyrosime.
  The sequences TCT, TCC, TCA, TCG, AGT, AGC = serine
  The sequence TAA, TAG or TGA the stop signal.
- Can we compute the probability of a sequence (e.g., TAC)?

We will note

Prob[T]	probability of being in the state "T"
Prob[A|T]	probability of being in state "A", knowing that previously (just before) we were in state "T"
Prob[C|AT]	probability of being in state "C", knowing that previously we were in state "A", and before "T"

Markov Example

- Computing the probability of a sequence (e.g., TAC as Prob [TAC])?
- First reasoning:
  Enumerate all possible sequences observed during a (long) period.
  Prob [TAC] = #seq ”TAC” / # all seq generated
- Difficult to find the data
- Time consuming
- Difficult to observe some phenomena

Markov Example

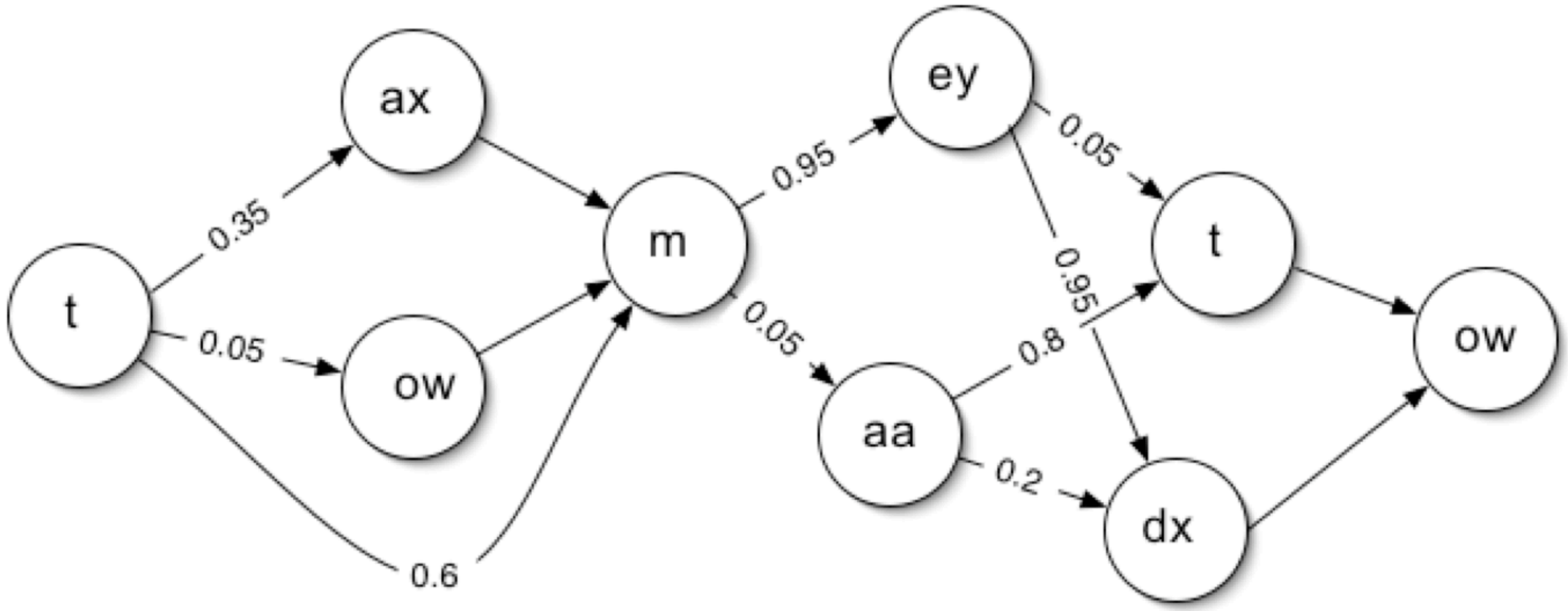
- Computing the probability of a sequence (e.g., TAC as Prob[TAC])?
- Second reasoning: we have a (Markov) model, and we can use it.
  Decompounding into simpler parts...
  Prob [TAC] = Prob [T] · Prob [A|T] · Prob [C|AT]
  and more simpler (Markov property of *memoryless*: the future depends only of current state)
  Prob [TAC] = Prob [T] · Prob [A|T] · Prob [C|A]
  where Prob [A|T] and Prob [C|A] are the transition probability
- Prob [TAC] = 1 · 0.5 · 0.2 = 0.1

Markov Example

Sequence	First	Second	Third	Prob
TTT	1	0.2	0.2	0.04
TTA	1	0.2	0.5	0.1
TTG	1	0.2	0.2	0.04
TTC	1	0.2	0.1	0.02
TAT	1	0.5	0.3	**0.15**
TAA	1	0.5	0.25	0.125
TAG	1	0.5	0.25	0.125
TAC	1	0.5	0.2	0.1
TGT	1	0.2	0.35	0.07
TGA	1	0.2	0.3	0.06
TGG	1	0.2	0.1	0.02
TGC	1	0.2	0.25	0.05
TCT	1	0.1	0.25	0.025
TCA	1	0.1	0.3	0.03
TCG	1	0.1	0.2	0.02
TCC	1	0.1	0.25	0.025

Speech Synthesis

The possible pronunciation of the word “tomato”.
The states are the possible phonemes for the word “tomato”.



Speech Synthesis

- The computation of the different pronunciations

t - ax - m - ey - t - ow	0,35 . 1 . 0,95 . 0,05 . 1 =	0,016625
t - ax - m - ey - dx - ow	0,35 . 1 . 0,95 . 0,95 . 1 =	**0,315875**
t - ax - m - aa - t - ow	0,35 . 1 . 0,05 . 0,8 . 1 =	0,014
t - ax - m - aa - dx - ow	0,35 . 1 . 0,05 . 0,2 . 1 =	0,0035
t - ow - m - ey - t - ow	0,05 . 1 . 0,95 . 0,05 . 1 =	0,002375
t - ow - m - ey - dx - ow	0,05 . 1 . 0,95 . 0,95 . 1 =	0,045125
t - ow - m - aa - t - ow	0,05 . 1 . 0,05 . 0,8 . 1 =	0,002
t - ow - m - aa - dx - ow	0,05 . 1 . 0,05 . 0,2 . 1 =	0,0005
t - m - ey - t - ow	0,6 . 0,95 . 0,05 . 1 =	0,0285
t - m - ey - dx - ow	0,6 . 0,95 . 0,95 . 1 =	**0,5415**
t - m - aa - t - ow	0,6 . 0,05 . 0,8 . 1 =	0,024
t - m - aa - dx - ow	0,6 . 0,05 . 0,2 . 1 =	0,006

POS Tagging

- We can use Markov Model to design a Part-Of-Speech (POS) tagger
  Assign a POS tag for each surface word (token)
  No guarantee that the whole sentence is correct.
- Example
  Input: "Time flies like an arrow."
  Output: "Time/NNP flies/VBZ like/IN an/DT arrow/NN ./."
  Input: "time flies like an arrows." (*)
  Output: "time/NN flies/VBZ like/IN an/DT arrows/NNS ./."

POS Tagging

- Different POS taggers freely available online
- May work at different levels
  “aimes” tag simply as “VB” or “VB” and the needed morphological information
  Usually select only one tag per word (but no guarantee this will be the correct one)
- Other example
  “The brown cat eats the gray mouse.”
  “The/DT brown/JJ cat/NN eats/VBZ the/DT gray/JJ mouse/ NN ./.”

POS Tagset (Penn Treebank)

DT	determiner
IN	preposition
JJ	adjective
NN	singular or mass noun
NNP	singular proper noun
NNS	plural noun
RB	adverb
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle, gerund
VBP	verb, non-3rd person singular present
VBZ	verb, 3rd singular present
WDT	wh-determiner
WP	wh-pronoun

POS Tagging (word-based)

- A given token (word) may belong to more than one POS. “record” is it a noun (NN) or a verb (VB)?

  and maybe it is better to have a small dictionary than a larger one
- Most words taken from the dictionary have only one part of speech or have a strong preference for only one of them.
- For French & English, 50% to 60% of words have a unique possible tag, and 15% to 25% have only two tags.

POS Tagging (word-based)

- Tagging a word with its most common POS, success rate of around 75% (for both French and English)
- Usage in corpus

  Assign the most common tag for each known word and the tag “proper noun” to all unknown

  Success rate: around 90% accuracy for the English language (Charniak)

E. Charniak: Statistical Language Learning, The MIT Press, 1993.

POS Tagging (sequence-based)

- A given token (word) may belong to more than one POS. E.g., “record” as noun or verb?
- Based on previous POS tags (syntagmatic information) Assign the most frequent tag for each word based on previous tags (how many previous tags?) Having the sequence “determinant-adjective-?? (DT–JJ -??) the next POS tag will certainly be “noun” (NN)
- Success rate: around 77% accuracy for the English language (using only the most frequent rules, without considering the word)
- Add a special tag (e.g., “null”) beginning of the sentence

POS Tagging (combining)

- Brill’s tagger (1995)
- Based on a dictionary
  - contains all words
  - list all legal tags in an frequently-based order

1. Tag each word with its most likely tag
2. Apply a list of transformation to modify the initial tagging (contextual rules)

- Using 500 rules, accuracy of 97% for the English language

POS Tagging (combining)

- Back to a Markov Model
- Combing both sources of information
  - the word itself
  - the context (limited to the previous tag in a Markov model)
  - ignore the position in the sentence.
- States?

  Possible POS tag
- Transitions?

  Possible sequence of two adjacent POS tags

POS Tagging (combining)

- Estimate the probabilities of a sequence of two tags (e.g., ”NN VB”, or in general t_j t_k)

  Prob [t_k | t_j] = C(t_k t_j) / C(t_j)

  with C(t_j) = number of tags t_j (e.g., # NN in the corpus) and C(t_k t_j) = number of times we have the bigram t_j t_k (e.g., “NN VB” in the corpus)
- We need some training examples (manually tagged corpus, see next slide)
- Example: we have 833 tags NN and 358 times the sequence “NN VB”, Prob [VB | NN] = 358 / 833 = 0.429772

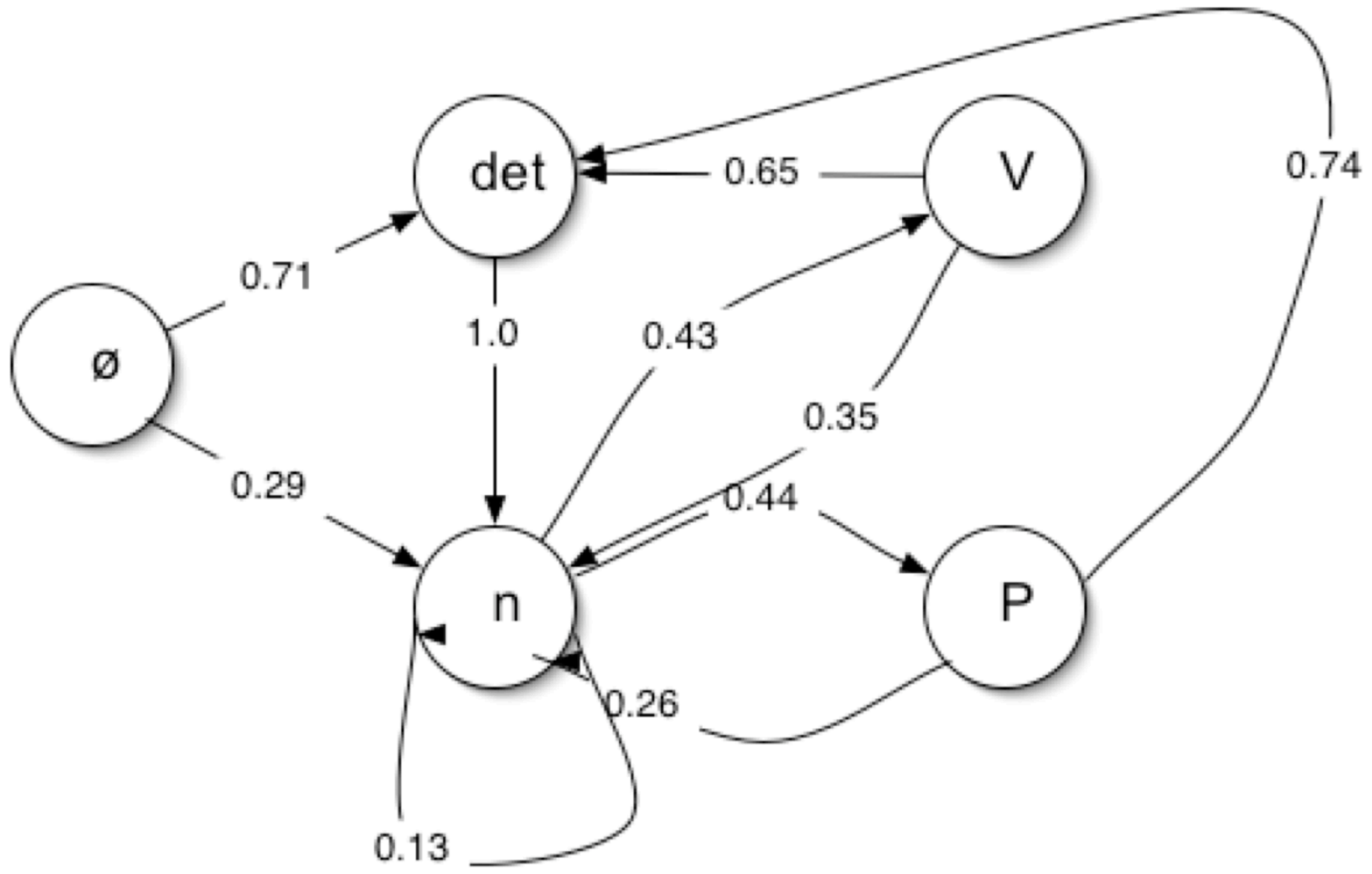
POS Tagging

We have a corpus (manually annotated) of 300 sentences

POS	freq	bigram	freq	Prob	estimate
Ø	300	Ø, DT	213	P [DT| Ø]	0.71
		Ø, NN	87	P [NN| Ø]	0.29
DT	558	DT, NN	558	P [NN| DT]	1.00
NN	833	NN, VB	358	P [VB| NN]	0.43
		NN, NN	108	P [NN| NN]	0.13
		NN, IN	366	P [IN| NN]	0.44
VB	300	VB, NN	75	P [NN| VB]	0.35
		VB, IN	194	P [IN| VB]	0.65
IN	307	IN, DT	226	P [DT| IN]	0.74
		IN, NN	81	P [NN| IN]	0.26

POS Tagging

And the corresponding Markov model



POS Tagging

- We can compute the probability of a tags sequence (e.g., an English sentence or a part of it)
  "Ø DT NN VB IN DT NN" (e.g., "the cat sat on the mat")
  Prob ["Ø DT NN VB IN DT NN"] =
  $0.71 \cdot 1.0 \cdot 0.43 \cdot 0.65 \cdot 0.74 \cdot 1.0 = 0.14685$
- Other example (composed also of six tags)
  Prob ["Ø NN VB NN IN NN NN"] =
  $0.29 \cdot 0.43 \cdot 0.35 \cdot 0.44 \cdot 0.26 \cdot 0.13 = 0.00065$
- This is the first part. We may compute the probability of a sequence of *n* tags (Prob[$t_{1,n}$]) and thus we can estimate that some syntactic structures are more probable than others.

POS Tagging

- We then need to estimate the probability that knowing a given POS tag, the next word (token) will be the given word. E.g., Prob [record| VB] = …, Prob [like | VB] = … Prob [flower | NN] = ?
  Given that the next word is a NN, estimate the probability that this word will be “flower”?
- The corresponding word has only one POS.
- But in general…

POS Tagging

- In general

1. We may use a machine-readable dictionary and when a word owns *n* possible tags, each of them as the probability 1/*n* to be the correct one.
2. We need to have a (manually) tagged corpus to
3. …

POS Tagging

- Estimate the probability that knowing a given POS tag, the next word (token) will be a given word.
  Prob [flower | NN] = ?
  (e.g., ”NN flower”, or in general the tag t_j with the word w_k)
  $$\text{Prob } [w_k \mid t_j] = C(w_k\ t_j)\ /\ C(t_j)$$
  with $C(t_j)$ = number of tags t_j (# NN in the corpus = 833)
  and $C(w_k\ t_j)$ = number of times we have the word w_k has the POS tag t_k (e.g., “flower” as NN = 53, see next slide)
- This probability is different from Prob [w_k and t_j] (e.g., Prob[flower and NN] = 53/68 = 0.78

POS Tagging

This a example of the needed information

	NN	VB	DT	IN	Total
flies	21	23	0	0	44
fruit	49	5	1	0	55
like	10	30	0	21	61
a	1	0	201	0	202
the	1	0	300	2	303
flower	53	15	0	0	68
flowers	42	16	0	0	58
birds	64	1	0	0	65
others	592	210	56	284	1142
total	833	300	558	307	1998

POS Tagging

- We may compute some examples
- Prob [flower | NN] = 53 / 833 = 0.06363
- Prob [flies | NN] = 21 / 833 = 0.02521
- Prob [flies | VB] = 23 / 300 = 0.07667
- Prob [the | DT] = 300 / 558 = 0.5102
- Prob [the | VB] = 0 / 300 = 0.0
- Prob [the | NN] = 1 / 833 = 0.0012

POS Tagging

- The problem: Determine the most probable sequence of POS tags $t_{1,n}$ for the input sequence (sentence) $w_{1,n}$ (word from 1 to *n*).

  $$\text{Arg Max } (t_{1,n}) \text{ Prob}[\, w_{1,n} \mid t_{1,n} \,] \cdot \text{Prob}[\, t_{1,n} \,]$$

- Example

  Input: $w_{1,n} =$ « *Flies like a flower* » (*n*=4)

  Output: « *Flies*/NN *like*/VB *a*/DT *flower*/NN »?

- Example

  « *Flies*/NN *like*/VB *a*/DT *flower*/NN »

  or « *Flies*/VBZ *like*/IN *a*/DT *flower*/NN »

POS Tagging

- « *Flies*/NN *like*/VB *a*/DT *flower*/NN »?
- Prob [$w_{1,n}$ | $t_{1,n}$] · Prob [$t_{1,n}$]
- Prob [$t_{1,n}$] = 0.29 · 0.43 · 0.65 · 1 = 4.68E-06
- Prob [$w_{1,n}$ | $t_{1,n}$] = 0.0252 · 0.1 · 0.3602 · 0.0636 = 5.778E-05
- Prob [$w_{1,n}$ | $t_{1,n}$] · Prob [$t_{1,n}$] = 4.68E-06
- After normalization: 0.556
- Second most probable 0.443 (NN-IN-DT-NN)

Conclusion

- Markov model use in various contexts
  - Random walk
  - Surfing on the Internet (PageRank)
- Use to model spam email (and thus use to filter the e-mail traffic)
- Extension: We have shown only *first-order* MM. We can consider second-order (Prob [A|BC]).
- Variant: Hidden Markov Chain: the states and the transitions cannot be observed directly but only by the observation of a symbols
- The Hidden Markov Model (HMM) is the basic model of speech recognition.

Surfing on the Internet

- Each page is a state
- Transition: from a given we may reach pages that are related by a hyperlink
- Select the next page randomly based on available hyperlinks in the source page.
  Probability to go from Page i to Page j =
  - $1/d_i$ if there is a hyperlink from Page i to Page j
  - 0 otherwise
- Extension:
  Consider the position of the hyperlink in Page i
  Consider the words in the anchor text