Natural Language Processing (NLP)

(CTMTAICS SII P3)

Tuesday : 10:00 a.m. – 12:00 noon
Wednesday : 10:00 a.m. – 12:00 noon
Friday : 10:00a.m. – 11:00 a.m.

Natural Language Processing (NLP)

- Unit 1: Introduction (Regular expression and Finite state Automaton
- Unit 2: Word Level Analysis
- Unit 3: Syntactic Analysis
- Unit 4: Semantic and Pragmatics
- Unit 5: Discourse Analysis and Lexical Resources

Prerequisites :

- ✓ Theory of computation
- ✓ Probability and Statistics
- ✓ Python scripts

TA1 (25)	Mid Sem (50)	End Sem (100)
Unit 1	Unit 1 , 2 and 3	Unit 1,2,3,4 and 5

- ➢ TA2 : Project
- ➢ Lab : 10 -12 Labs

Lab Marks

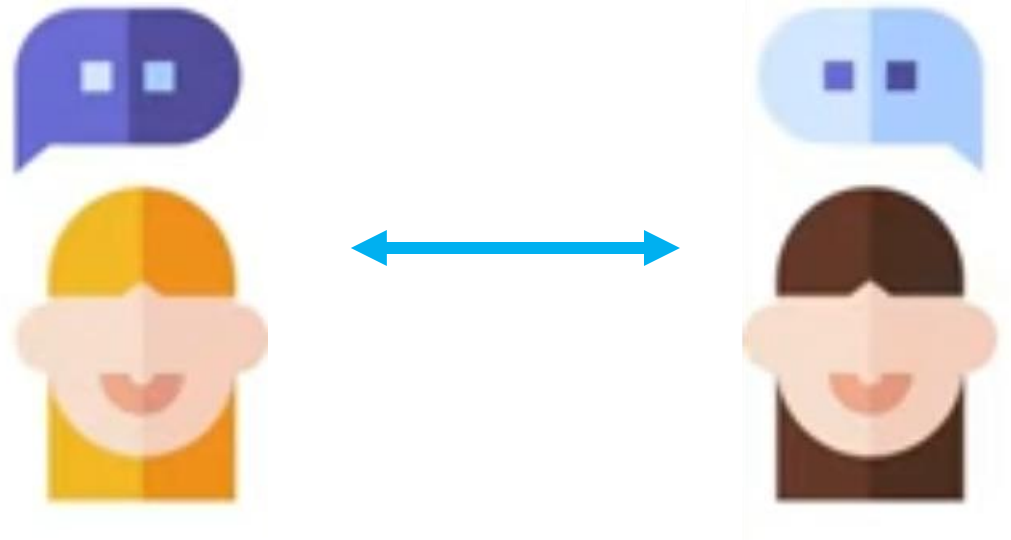
- Continuous Evaluation (100 marks)
- External Examination (100 marks)

Pattern

- ✓ Aim of Assignment
- ✓ Tools used
- ✓ Theory
- ✓ Procedure
- ✓ Result and Discussion
- ✓ Conclusion
- ✓ Code

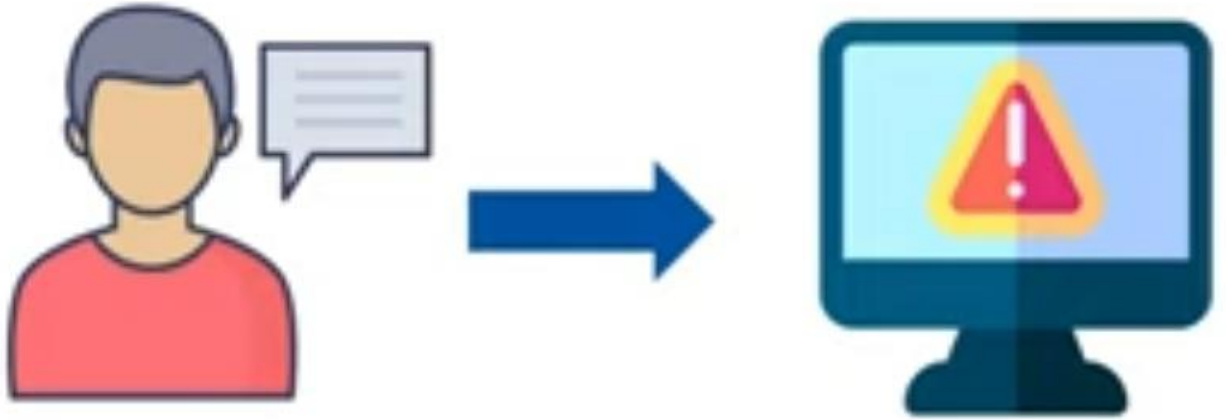
What is Natural Language ?

Natural language refers to the human way of communicating, i.e. through text and speech.

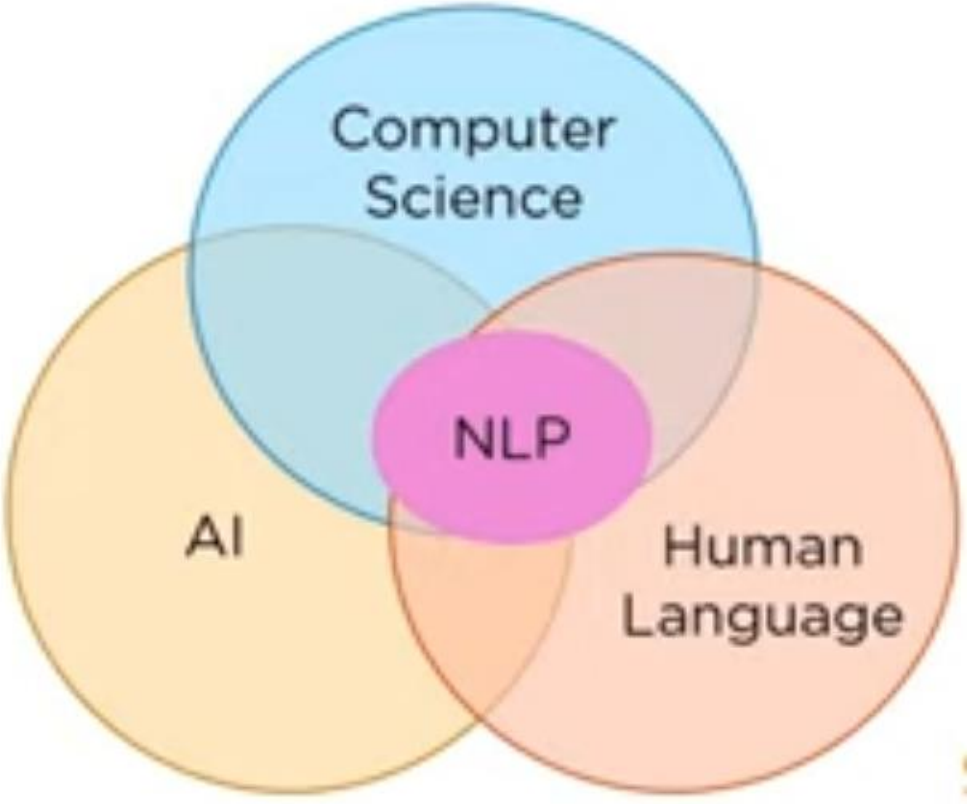


What is Natural Language Processing (NLP)

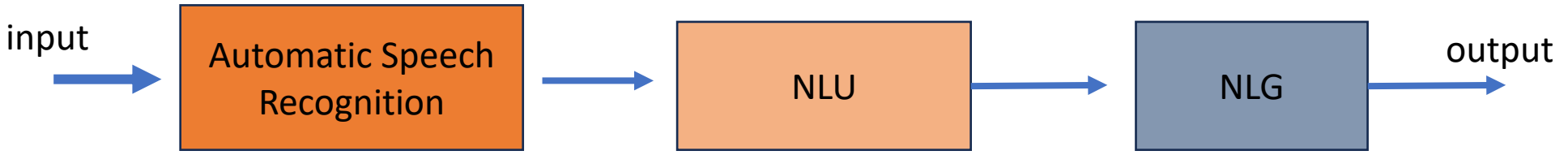
NLP refers to the branch of artificial intelligence that allows the machines to understand the natural language.



What is Natural Language Processing (NLP)



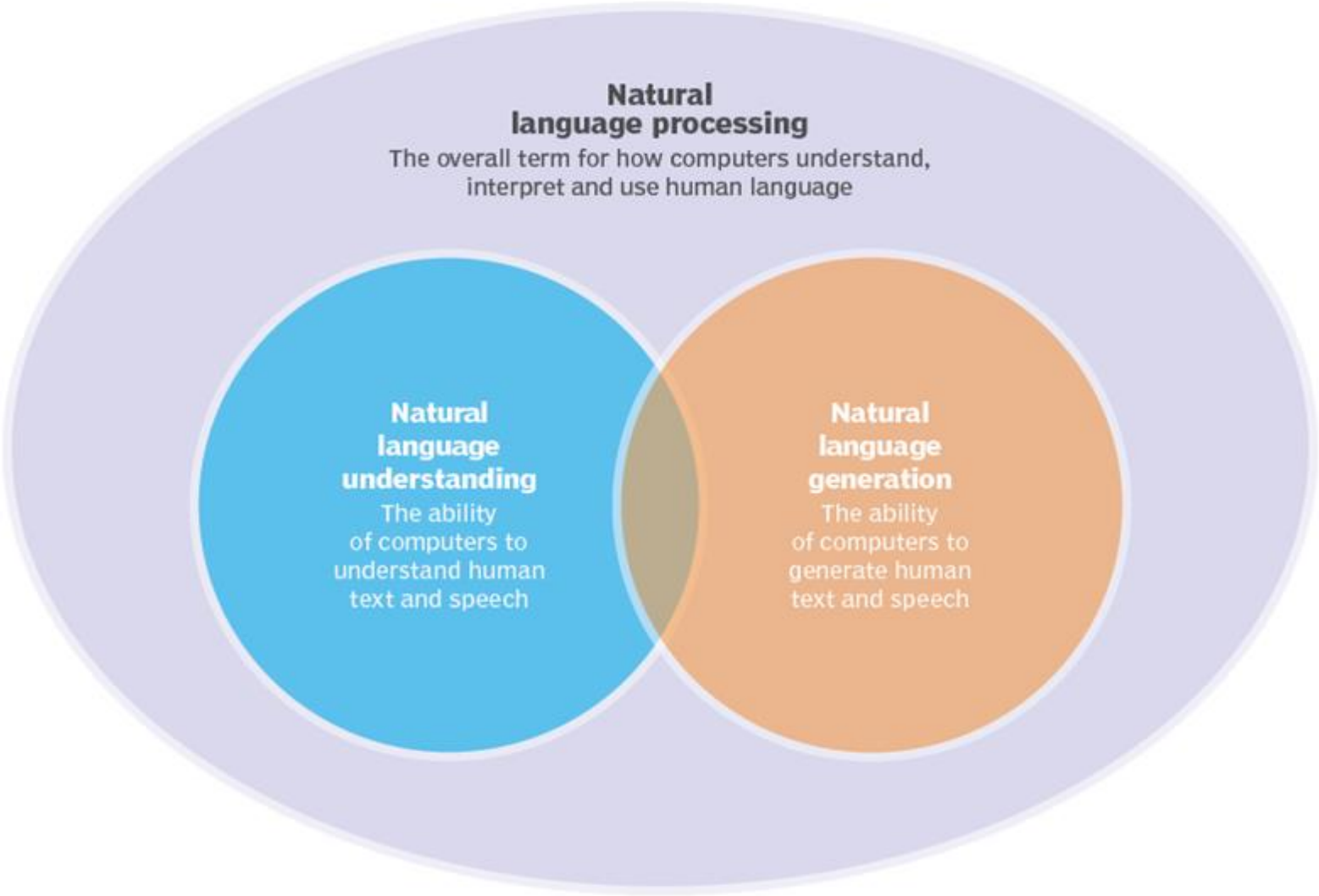
Natural Language Processing



NLU : What do the users say? Their intent and Meaning.

NLG : What should say to user?
It should be Intelligent and conversational.
Deal with structured data.
Text/Sentences Planning (Corpus).

How NLP, NLU and NLG are related



What is Natural Language Processing ?

1. NLP is a subfield of linguistics, computer science and artificial intelligence that deals with the interaction between computers and human languages.
2. It enables computers to understand, interpret, and generate natural language, the way humans do.
3. It involves a variety of techniques, including computational linguistics, machine learning, and statistical modeling.
4. It is used in a wide range of industries, including finance, healthcare, education, and entertainment.
5. Some of the main applications of NLP include language translation, speech recognition, sentiment analysis, text classification, and information retrieval.
6. NLP is a rapidly evolving field that is driving new advances in computer science and artificial intelligence.
7. NLP has the potential to transform the way we interact with technology in our daily lives.

Advantages

- **Improves human-computer interaction:** Enables computers to understand and respond to human languages, which improves the overall user experience and makes it easier for people to interact with computers.

- **Automates repetitive tasks:** NLP techniques can be used to automate repetitive tasks, such as text summarization, sentiment analysis, and language translation, which can save time and increase efficiency.

- **Enables new applications:** NLP enables the development of new applications, such as virtual assistants, chatbots, and question answering systems, that can improve customer service, provide information, and more.

- **Improves decision-making:** NLP techniques can be used to extract insights from large amounts of unstructured data, such as social media posts and customer feedback, which can improve decision-making in various industries.

Advantages

- **Improves accessibility:** NLP can be used to make technology more accessible, such as by providing text-to-speech and speech-to-text capabilities for people with disabilities.

- **Facilitates multilingual communication:** NLP techniques can be used to translate and analyze text in different languages, which can facilitate communication between people who speak different languages.

- **Improves information retrieval:** NLP can be used to extract information from large amounts of data, such as search engine results, to improve information retrieval and provide more relevant results.

- **Enables sentiment analysis:** NLP techniques can be used to analyze the sentiment of text, such as social media posts and customer reviews, which can help businesses understand how customers feel about their products and services.

Advantages

- **Improves content creation:** NLP can be used to generate content, such as automated article writing, which can save time and resources for businesses and content creators.

- **Supports data analytics:** NLP can be used to extract insights from text data, which can support data analytics and improve decision-making in various industries.

- **Enhances natural language understanding:** NLP research and development can lead to improved natural language understanding, which can benefit various industries and applications.

Disadvantages

- **Limited understanding of context:** NLP systems have a limited understanding of context, which can lead to misinterpretations or errors in the output.
- **Requires large amounts of data:** NLP systems require large amounts of data to train and improve their performance, which can be expensive and time-consuming to collect.
- **Limited ability to understand idioms and sarcasm:** NLP systems have a limited ability to understand idioms, sarcasm, and other forms of figurative language, which can lead to misinterpretations or errors in the output.
- **Limited ability to understand emotions:** NLP systems have a limited ability to understand emotions and tone of voice, which can lead to misinterpretations or errors in the output.
- **Difficulty with multi-lingual processing:** NLP systems may struggle to accurately process multiple languages, especially if they are vastly different in grammar or structure.

Disadvantages

- **Dependency on language resources:** NLP systems heavily rely on language resources, such as dictionaries and corpora, which may not always be available or accurate for certain languages or domains.

- **Difficulty with rare or ambiguous words:** NLP systems may struggle to accurately process rare or ambiguous words, which can lead to errors in the output.

- **Lack of creativity:** NLP systems are limited to processing and generating output based on patterns and rules, and may lack the creativity and spontaneity of human language use

- **Ethical considerations:** NLP systems may perpetuate biases and stereotypes, and there are ethical concerns around the use of NLP in areas such as surveillance and automated decision-making.

Application Areas of NLP

- **Speech recognition and transcription:** NLP techniques are used to convert speech to text, which is useful for tasks such as dictation and voice-controlled assistants. (Ex: Google Assistant)
- **Language translation:** NLP techniques are used to translate text from one language to another, which is useful for tasks such as global communication and e-commerce. (Ex: Google Translator)
- **Text summarization:** NLP techniques are used to summarize long text documents into shorter versions, which is useful for tasks such as news summarization and document indexing.
- **Sentiment analysis**: NLP techniques are used to determine the sentiment or emotion expressed in text, which is useful for tasks such as customer feedback analysis and social media monitoring. (Ex: Twitter Information's)
- **Question answering:** NLP techniques are used to answer questions asked in natural language, which is useful for tasks such as chatbots and virtual assistants. (Ex: Chatbots)

Application Areas of NLP

- Contextual Advertisements
- Email Clients (Spam filtering / Smart Reply)
- Social Media (Removing adult content and opinion mining)

Tasks on NLP

- Text / Document Classification
- Sentiment Analysis
- Information Retrieval
- Parts of Speech tagging
- Language Detection and Machine Translation
- Conversational Agents Design
- Knowledge Graph and QA Systems
- Text Summarization
- Topic Modelling
- Spell Checking and Grammar correction
- Speech to Text conversation
- Text Parsing

Approaches in NLP

- Heuristic Approaches (Regular Expression, DFA, etc.)
- Machine Learning Approaches
- Deep Learning Approaches (ANN, RNN, CNN, etc)

Challenges of NLP

1. Ambiguity

Example :

I saw the boy on the beach with my binoculars.

I have never tasted a cake quite like that one before.

Types of Ambiguity

- Lexical Ambiguity (The tank was full of water)
- Syntactic Ambiguity (Old man and women were taken to a safe places)
- Semantic Ambiguity (The car hit the pole while it was moving)
- Pragmatic Ambiguity (The police are comming)

Challenges of NLP

2. Contextual words

Example: I ran to the store because we ran out of milk.

3. Colloquialisms and slang

- Piece of cake, Pulling your leg

4. Synonyms

5. Irony, Sarcasm and tonal difference

Example : That’s just what I needed today!.

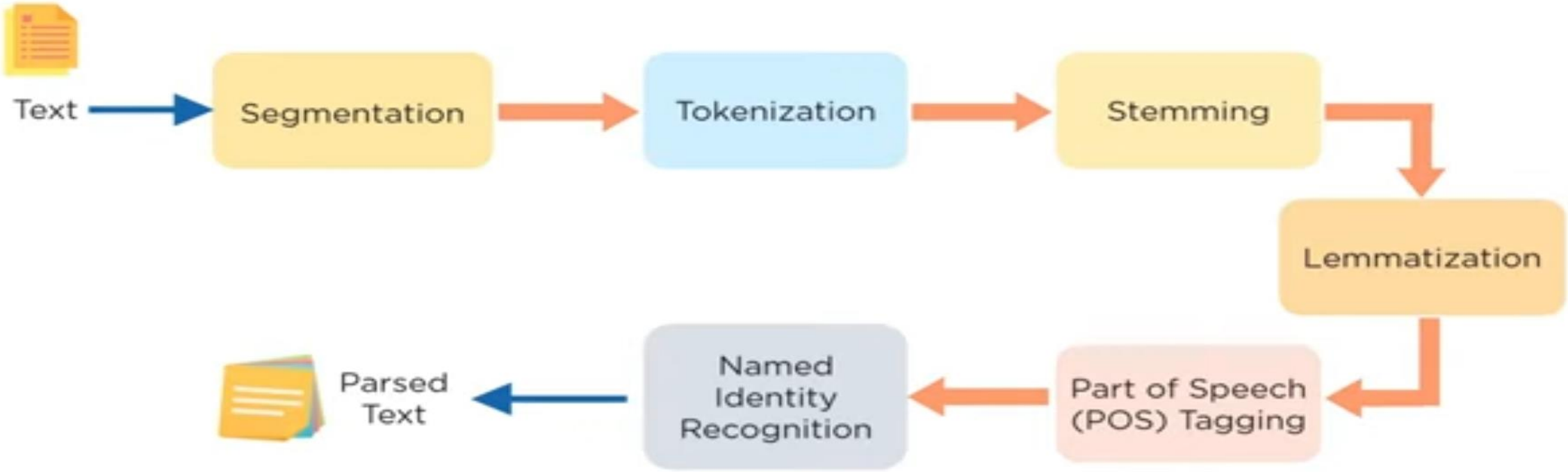
6. Spelling Errors

7.Creativity

Example : Poem, Dialogue, Scripts

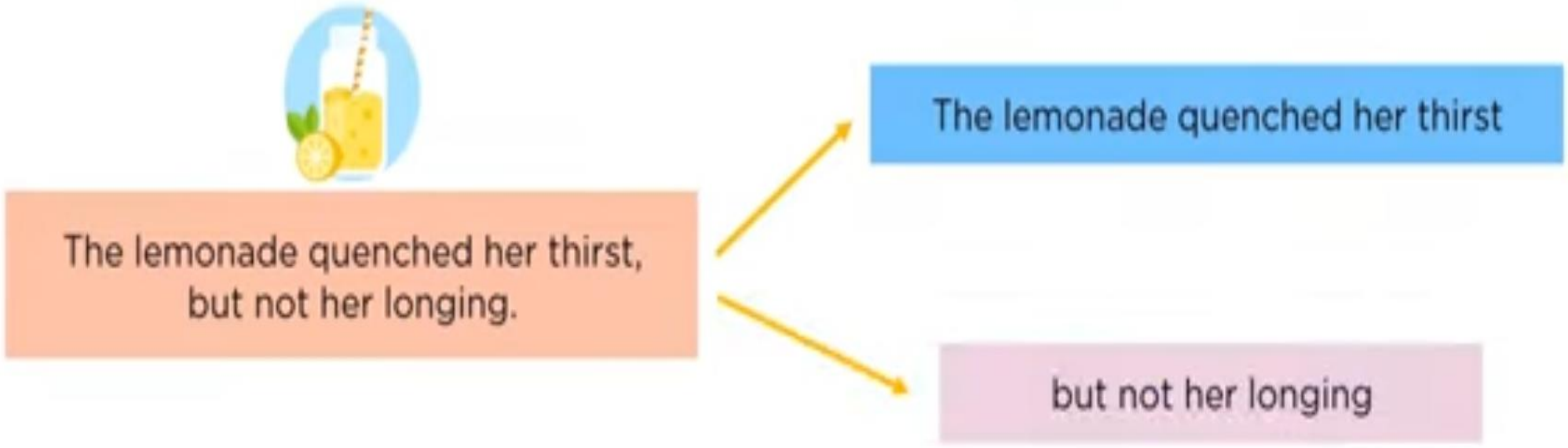
NLP Pipeline

NLP pipeline is a set of steps followed to build an end to end NLP software.



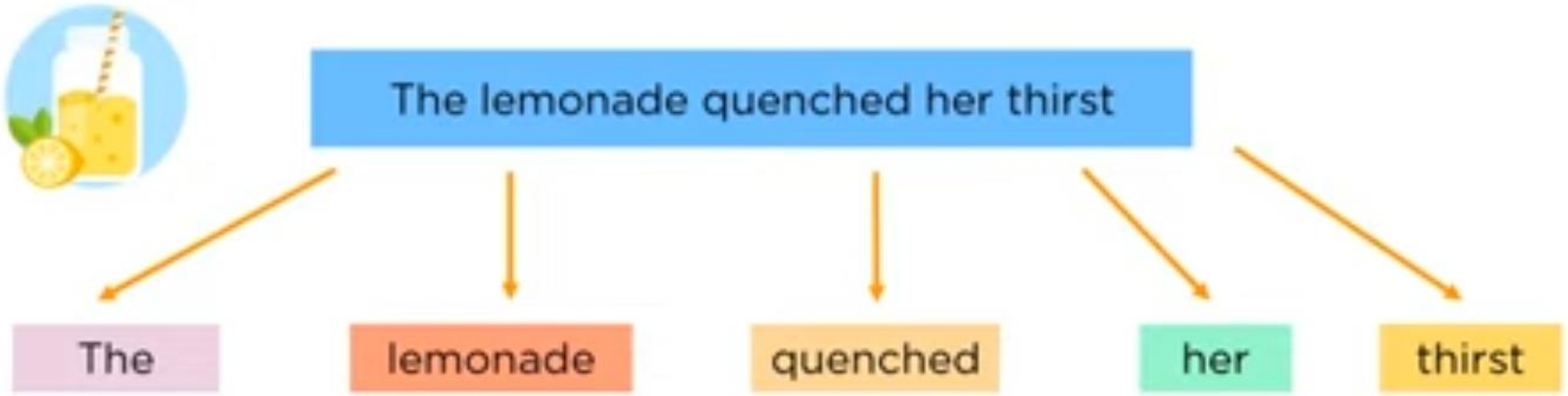
1. Segmentation

- The process of dividing a sentence into its component sentences, Usually along punctuation marks.



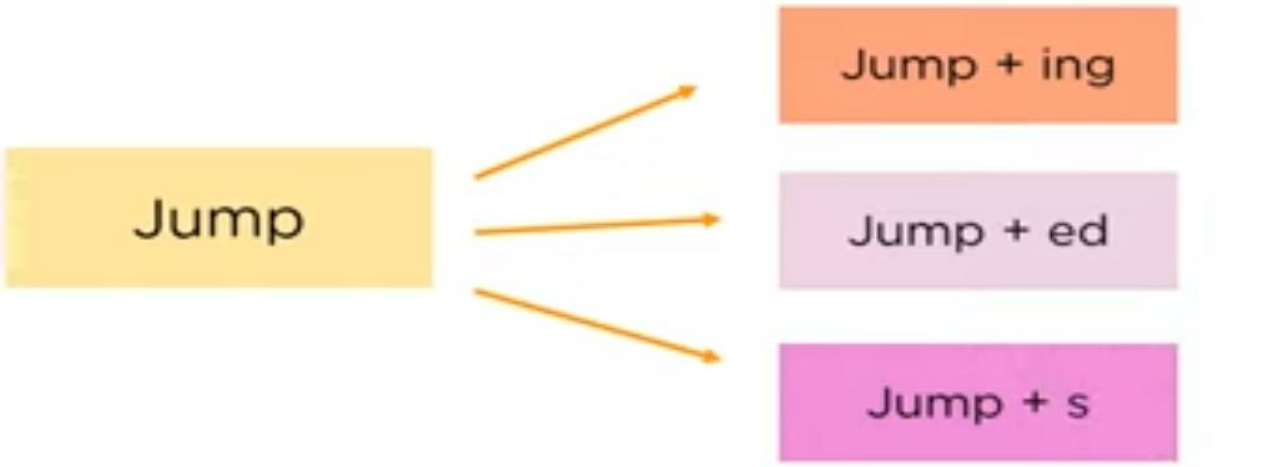
2. Tokenization

- The process of splitting sentences into their constituent words is called Tokenization



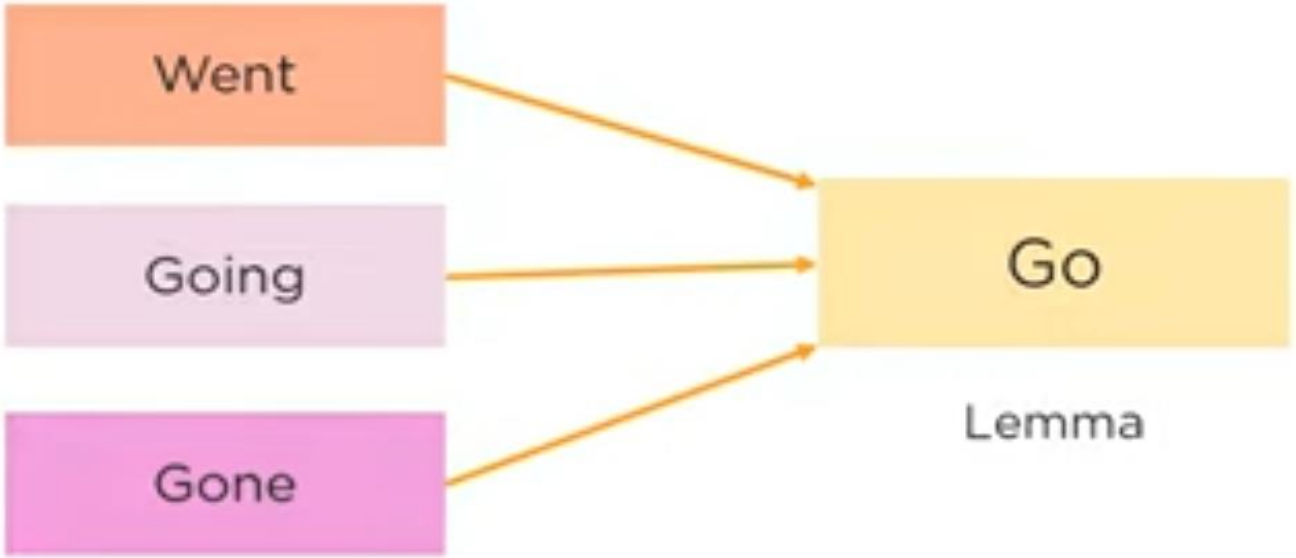
3. Stemming

- The process of obtaining the word stem of a new word.
- Word Stem give new words upon adding affixes to them.



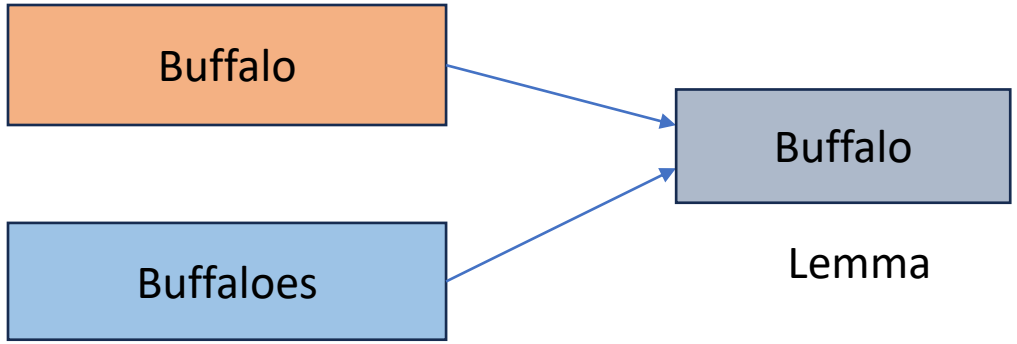
4. Lemmatization

- The process of obtaining the Root Stem of a word.
- Root Stem give new base form of a word.



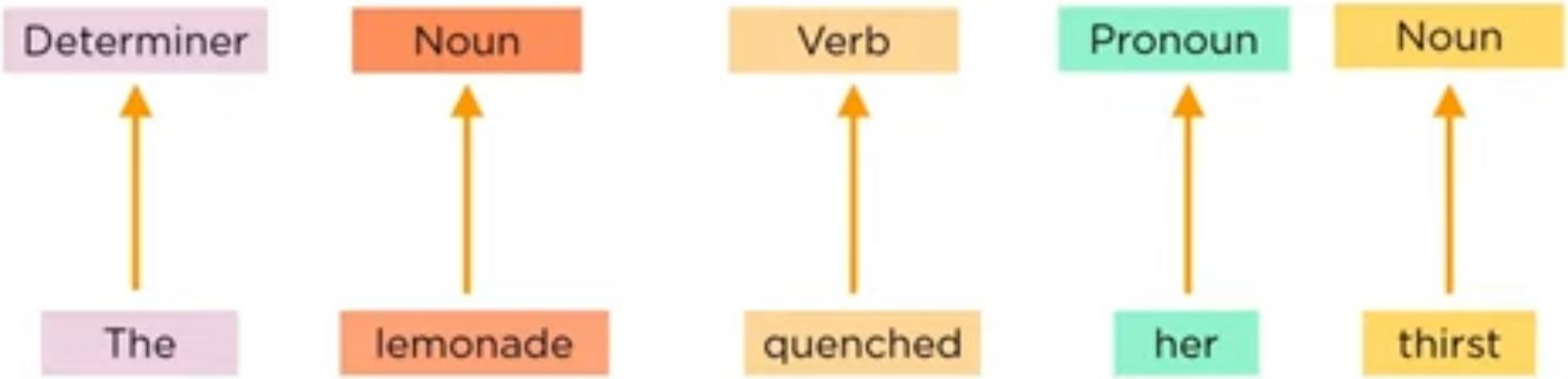
Example:

There's a Buffalo grazing in the field.
There are Buffaloes grazing in the field.



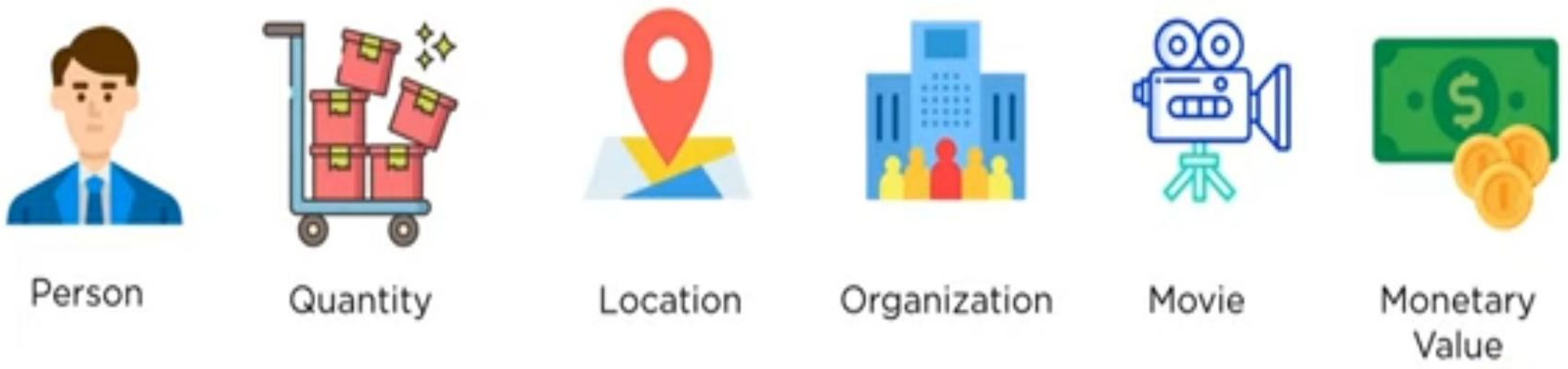
5. Part of Speech Tagging

- Identifies which part of speech a word belongs to.
- It tags a word as a verb, noun, pronoun etc.



6. Named Entity Recognition

- Classifying the words into subcatagories.
- The subcatagories are person, Quantity, Location, Organization, Movie, Monetary value etc.



Step #1: Sentence Segmentation Breaking the piece of text in various sentences.

- ***Input :*** *San Pedro is a town on the southern part of the island of Ambergris Caye in the Belize District of the nation of Belize in Central America. According to 2015 mid-year estimates the town has a population of about 16444. It is the second largest town in the Belize District and largest in the Belize Rural South constituency.*

- *Output :*

✓*1.San Pedro is a town on the southern part of the island of Ambergris Caye in the Belize District of the nation of Belize, in Central America.*

✓*2. According to 2015 mid-year estimates the town has a population of about 16444.*

✓*3. It is the second-largest town in the Belize District and largest in the Belize Rural South constituency.*

Step #2: Word Tokenization Breaking the sentence into individual words called as tokens. We can tokenize them whenever we encounter a space, we can train a model in that way. Even punctuations are considered as individual tokens as they have some meaning.

- ***Input :*** *San Pedro is a town on the southern part of the island of Ambergris Caye in the Belize District of the nation of Belize in Central America. According to 2015 mid-year estimates the town has a population of about 16444. It is the second-largest town in the Belize District and largest in the Belize Rural South constituency.*

- ***Output :*** *'San', 'Pedro', 'is', 'a', 'town', 'on', 'the', 'southern', 'part', 'of', 'island, 'Ambergris', 'Caye', 'in', 'Belize', 'District', 'nation' 'Central' 'America', 'According', 'to', '2015', 'mid-year', 'estimates', 'has', 'population', 'about', '16444', 'It', 'second-largest', 'and', 'largest', 'Rural', 'South', 'constituency'.*

Step #3: Lemmatization Feeding the model with the root word.

- Large
- Largest

✓ South
✓ Southern

➢ Estimate
➢ Estimates
➢ Estimation

Step #4: Predicting Parts of Speech for each token

Input :

'San', 'Pedro', 'is', 'a', 'town', 'on', 'the', 'southern', 'part', 'of', 'island, 'Ambergris', 'Caye', 'in', 'Belize', 'District', 'nation' 'Central' 'America', 'According', 'to', '2015', 'mid-year', 'estimates', 'has', 'population', 'about', '16444', 'It', 'second-largest', 'and', 'largest', 'Rural', 'South', 'constituency'.

Output :

Town - common noun

Is - verb

The - determiner

Of- Preposition

In-Preposition

San – noun

And so on.

- **Step #5:** Identifying stop words.

- There are various words in the English language that are used very frequently like ‘a’, ‘and’, ‘the’ etc. These words make a lot of noise while doing statistical analysis.
- We can take these words out.
- Some NLP pipelines will categorize these words as stop words, they will be filtered out while doing some statistical analysis.
- Definitely, they are needed to understand the dependency between various tokens to get the exact sense of the sentence.
- The list of stop words varies and depends on what kind of output are you expecting.

- **Step 6.1:** Dependency Parsing
- This means finding out the relationship between the words in the sentence and how they are related to each other.
- We create a parse tree in dependency parsing, with root as the main verb in the sentence.
- If we talk about the first sentence in our example, then ‘is’ is the main verb and it will be the root of the parse tree.
- We can construct a parse tree of every sentence with one root word(main verb) associated with it. We can also identify the kind of relationship that exists between the two words.
- In our example, ‘San Pedro’ is the subject and ‘island’ is the attribute. Thus, the relationship between ‘San Pedro’ and ‘is’, and ‘island’ and ‘is’ can be established. Just like we trained a Machine Learning model to identify various parts of speech, we can train a model to identify the dependency between words by feeding many words. It’s a complex task though. In 2016, Google released a new dependency parser Parsey McParseface which used a deep learning approach.

- **Step 6.2:** Finding Noun Phrases We can group the words that represent the same idea.
-
- For example – It is the second-largest town in the Belize District and largest in the Belize Rural South constituency. Here, tokens ‘second’, ‘largest’ and ‘town’ can be grouped together as they together represent the same thing ‘Belize’.
- We can use the output of dependency parsing to combine such words.
- Whether to do this step or not completely depends on the end goal, but it’s always quick to do this if we don’t want much information about which words are adjective, rather focus on other important details.

•**Step #7:** Named Entity Recognition(NER)

•San Pedro is a town on the southern part of the island of Ambergris Caye in the Belize District of the nation of Belize, in Central America.
•Here, the NER maps the words with the real world places.
•The places that actually exist in the physical world. We can automatically extract the real world places present in the document using NLP.

•If the above sentence is the input, NER will map it like this way:
San Pedro - Geographic Entity
Ambergris Caye - Geographic Entity
Belize - Geographic Entity
Central America - Geographic Entity

- **Step #8:** Coreference Resolution

- San Pedro is a town on the southern part of the island of Ambergris Caye in the Belize District of the nation of Belize, in Central America. According to 2015 mid-year estimates, the town has a population of about 16, 444. It is the second-largest town in the Belize District and largest in the Belize Rural South constituency.

- Here, we know that ‘it’ in the sentence 6 stands for San Pedro, but for a computer, it isn’t possible to understand that both the tokens are same because it treats both the sentences as two different things while it’s processing them. Pronouns are used with a high frequency in English literature and it becomes difficult for a computer to understand that both things are same.

THANK YOU

Language Model in NLP

What is language model in NLP?

- It is a model which knows our language.
- A model of the probability of a sequence of words.
- It estimates the relative likelihood of different phases and are useful in many NLP applications.
- The goal of probabilistic language model is to calculate the probability of a sentence of sequence of words.

$$P(w)=P(w_1, w_{2,} w_{3,} w_{4, \ldots,} w_n)$$

- It can be used to find the probability of the next word in the sentence.

$$P(w_s)=P(w_s|w1, w_{2,} w_{3,} w_{4, \ldots,} w_{s-1})$$

Language Model

I am the king of the ……world……

Input Predicted Word

I am the king of the world ……of……

Input Predicted Word

Example: keyboard of Mobile phone

Advantages of LM

✓It can predict what words are likely to come next in a text.

Ex: Suggest completions for an email o text message.

✓Capable to compute more probable alterations to a text

Ex: Suggest spelling or Grammar corrections

✓With a pair of models, we can compute the most probable translation of sentences.

✓With some example questions/answer pairs as training data, we can compute the most likely answer to a question.

Corpus

- **corpus** is a collection of texts, on which we can perform various natural language processing (NLP) functions.

- In simplest terms, a corpus is a folder of text files on your computer, and corpus readers process all these text files at once, though each file can be called on individually.

Feature, Document and Corpus

Feature

Every unique word in the corpus is considered as a feature.

Document

A document is a single text data point (a text file, book, blog, article, webpage).

Tokenization

It is the process of breaking text into pieces (called tokens).

Corpus

It a collection of all the documents present in our dataset.

Example :

Dog hates a cat. It loves to go out and play. Cat loves to play with a ball.

Corpus = "Dog hates a cat. It loves to go out and play. Cat loves to play with a ball."
Documents = [1. dog hates a cat.
2. it loves to go out and play.
3. cat loves to play with a ball.]

Features= ['and', 'ball', 'cat', 'dog', 'go', 'hates', 'it', 'loves', 'out', 'play', 'to', 'with']

Types of Language Model

- The bag-of-words model
- N-gram word models
- Other n-gram models
- Smoothing n-gram models
- Word representations
- Parts-of-speech (POS) tagging
- Grammar based Language modelling
- Statistical language modelling

N-gram Language Modelling (Probabilistic Model)

- n-gram : sequence of n words
- 1-gram (Unigram)(having no history word)
- 2-gram (Bigram) (having one history word)
- 3-gram(Trigram) (having three history word)
- N-gram (having n-1 history word)

Example: I am the king

[I] [am] [the] [king]

[I am] [am the] [the king]

[I am the] [am the king]

N-gram Language Modelling (Probabilistic Model)

- Unigram Probability $P(w) = \frac{count(w)}{N}$
- Bayes Rule $P(A|B) = \frac{P(A \cap B)}{P(B)}$
- Bigram Probability $P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
- Trigram Probability $P(w_i|w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{count(w_{i-2} w_{i-1})}$

N-gram Language Modelling (Probabilistic Model)

- Corpus

The girl bought a chocolate

The boy ate the chocolate

The girl bought a toy

The girl played with the toy

Example : For Unigram

P(the)=0.6
P(girl)=0.3
P(bought)=0.2
P(a)=0.2
P(chocolate)=0.2
P(boy)=0.1
P(ate)=0.1
P(toy)=0.2
P(played)=0.1
P(with)=0.1

- Vocabulary/Feature

{the, girl, bought, a, chocolate, boy, ate, toy, played, with}

N=No of features = 10

N-gram Language Modelling (Probabilistic Model)

- Input : The
- Output: The girl
- P(girl|the) = $\frac{count(the,girl)}{count(the)} = \frac{3}{6}$ = 0. 5
- P(boy|the)= $\frac{count(the,boy)}{count(the)} = \frac{1}{6}$=0.166
- Probabilities for our vocabulary for input: The ……

the = 0
girl = 0.5
bought=0
a=0
chocolate =0.166
boy=0.166
ate=0.166
toy=0
played=0
with=0

N-gram Language Modelling (Probabilistic Model)

- Input : The girl
- Output: The girl bought
- P(bought|the,girl) = $\frac{count(the,girl,bought)}{count(the\ girl)} = \frac{2}{3} = 0.67$
- P(played|the ,girl) = $\frac{count(the,girl,played)}{count(the,girl)} = \frac{1}{3} = 0.33$
- Probabilities for our vocabulary for input: The girl ……

The = 0
Girl = 0
Bought=o.67
A=0
Chocolate =0
Boy=0
Ate=0
Toy=0
Played=0.33
With=0

N-gram Language Modelling (Probabilistic Model)

- Input : The boy
- Output: The boy ate
- P(ate|the,boy) = $\frac{count(the,boy,ate)}{count(the\ boy)} = \frac{1}{1} = 1$
- Probabilities for our vocabulary for input: The boy ……

the = 0
girl = 0
bought=0
a=0
chocolate =0
boy=0
ate=1
toy=0
played=0
with=0

N-gram Language Modelling (Probabilistic Model)

Disadvantages of N-Grams

1. It has too many features.

2. Due to too many features, the feature set becomes too dense and is computationally expensive.

3. Choose the optimal value of N is not that easy task.

THANK YOU

Minimum Edit Distance

Application

- Spell correction
  - The user typed “graffe”

  Which is closest?
    - graf
    - graft
    - grail
    - giraffe

- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

  - Resulting alignment:

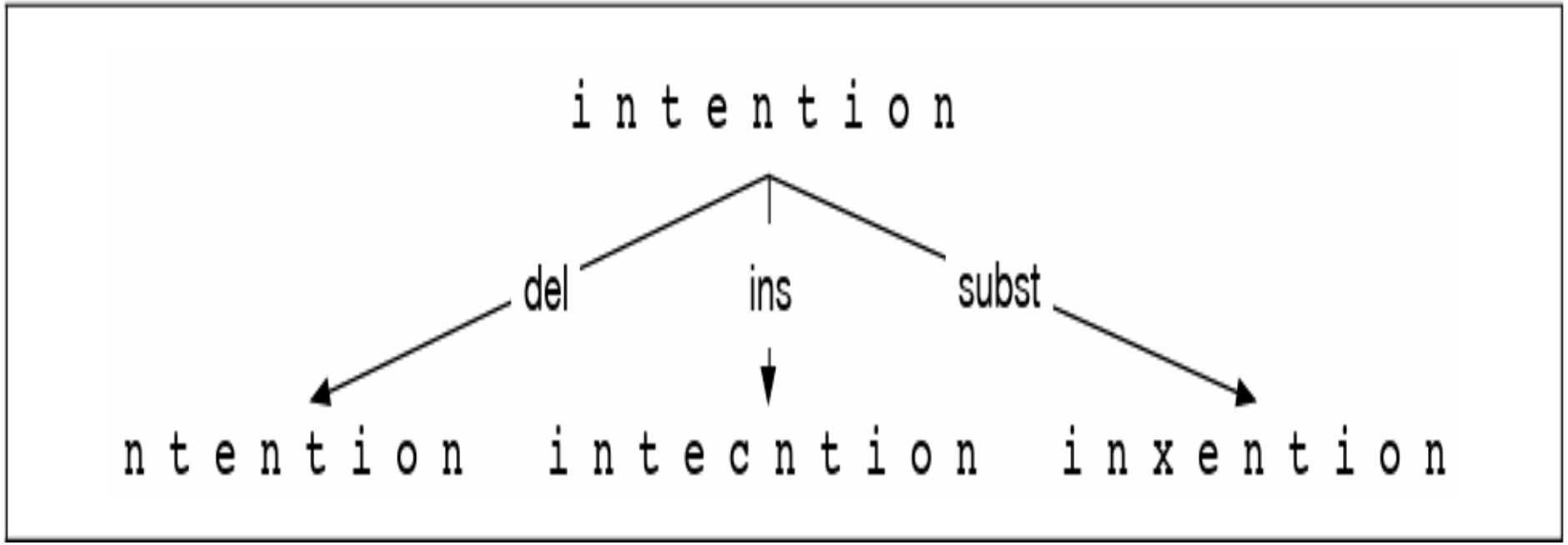
    −AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
    TAG−CTATCAC--GACCGC--GGTCGATTTGCCCGAC

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
  - Insertion
  - Deletion
  - Substitution
- Needed to transform one into the other

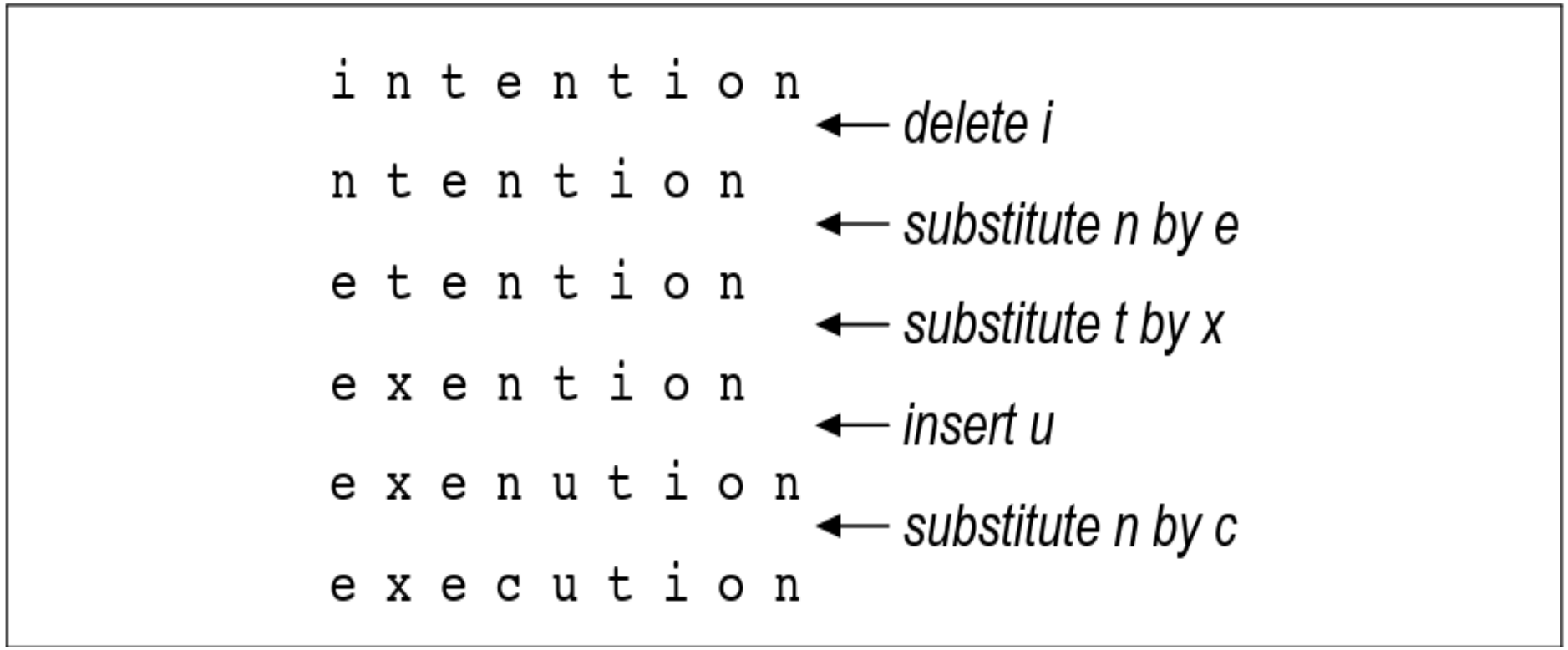
Meaning of Edit Operation



Minimum Edit Distance

- Two strings and their **alignment**:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```


i n t e n t i o n
delete i
n t e n t i o n
substitute n by e
e t e n t i o n
substitute t by x
e x e n t i o n
insert u
e x e n u t i o n
substitute n by c
e x e c u t i o n

Example

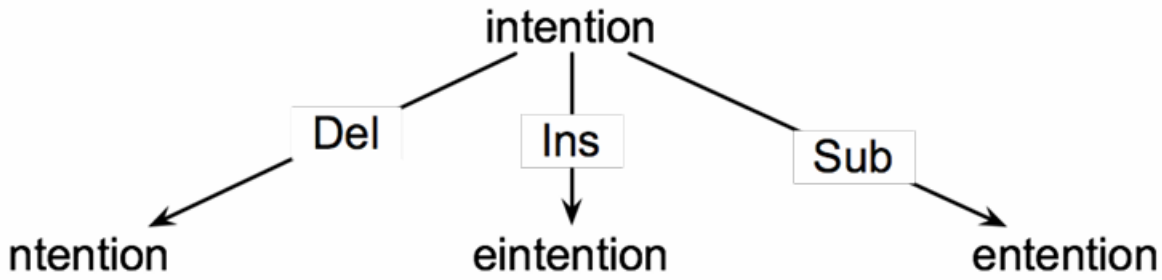
Minimum Edit Distance

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

How to find Minimum Edit Distance

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits



Minimum Edit as Search

- But the space of all edit sequences is huge!
  - We can't afford to navigate naïvely
  - Lots of distinct paths wind up at the same state.
    - We don't have to keep track of all of them
    - Just the shortest path to each of those revisted states.

Defining Minimum Edit Distance

- For two strings
  - X of length *n*
  - Y of length *m*
- We define D(*i*,*j*)
  - the edit distance between X[1..*i*] and Y[1..*j*]
    - i.e., the first *i* characters of X and the first *j* characters of Y
  - The edit distance between X and Y is thus D(*n*,*m*)

Dynamic Programming for Minimum Edit Distan

- **Dynamic programming**: A tabular computation of D(*n,m*)
- Solving problems by combining solutions to subproblems.
- Bottom-up
  - We compute D(i,j) for small *i,j*
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(*i,j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

Defining Minimum Edit Distance (LAVENSHTEIN)

- Initialization

  $D(i,0) = i$

  $D(0,j) = j$

- Recurrence Relation:

  For each $i = 1 \ldots M$

  For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

  $D(N,M)$ is distance

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 \\ D[i,j-1] + 1 \\ D[i-1,j-1] + \begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

function MIN-EDIT-DISTANCE(*source*, *target*) **returns** *min-distance*

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix *D[n+1,m+1]*

 # Initialization: the zeroth row and column is the distance from the empty string
$D[0,0] = 0$
for each row *i* **from** 1 **to** *n* **do**
    $D[i,0] \leftarrow D[i\text{-}1,0] + del\text{-}cost(source[i])$
for each column *j* **from** 1 **to** *m* **do**
    $D[0,j] \leftarrow D[0,j\text{-}1] + ins\text{-}cost(target[j])$

 # Recurrence relation:
for each row *i* **from** 1 **to** *n* **do**
    for each column *j* **from** 1 **to** *m* **do**
        $D[i,j] \leftarrow$ MIN($D[i-1,j] + del\text{-}cost(source[i])$,
                $D[i-1,j-1] + sub\text{-}cost(source[i], target[j])$,
                $D[i,j-1] + ins\text{-}cost(target[j])$)
 # Termination
return *D*[n,m]

Source string = n Target = m j →

i ↓

Src/Tar	#	E	X	E	C	U	T	I	O	N
#										
I										
N										
T										
E										
N										
T										
I										
O										
N										

11×11

← INSERT ↑ DELETE

Src/Tar	#	E	X	E	C	U	T	I	O	N
#	0 ←	1 ←	2 ←	3 ←	4 ←	5 ←	6 ←	7 ←	8 ←	9 ←
I	↑ 1									
N	↑ 2									
T	↑ 3									
E	↑ 4									
N	↑ 5									
T	↑ 6									
I	↑ 7									
O	↑ 8									
N	↑ 9									

11×11

INSERT DELETE

Src/Tar	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2	3	4	5	6	7	6	7	8
N	2									
T	3									
E	4									
N	5									
T	6									
I	7									
O	8									
N	9									

11×11

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

Figure 2.18 Computation of minimum edit distance between *intention* and *execution* with the algorithm of Fig. 2.17, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions.

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ **1**	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖←↑ **3**	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	↑ 3	↖←↑ 4	↖←↑ **5**	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← **5**	← **6**	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	↑ 5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ **8**	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ **8**	← 9	← 10	←↑ 11
i	↑ 7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ **8**	← 9	← 10
o	↑ 8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	↑ 9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**

Figure 2.19 When entering a value in each cell, we mark which of the three neighboring cells we came from with up to three arrows. After the table is full we compute an **alignment** (minimum edit path) by using a **backtrace**, starting at the **8** in the lower-right corner and following the arrows back. The sequence of bold cells represents one possible minimum cost alignment between the two strings. Diagram design after Gusfield (1997).

THANK YOU

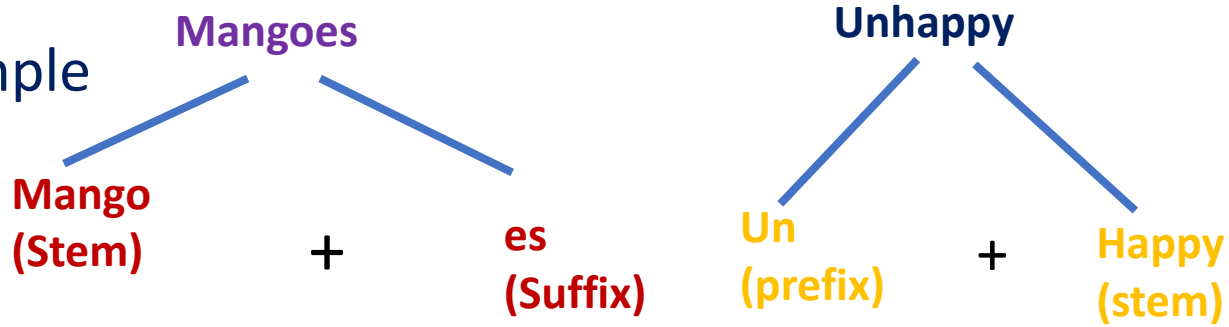
Morphological Parsing

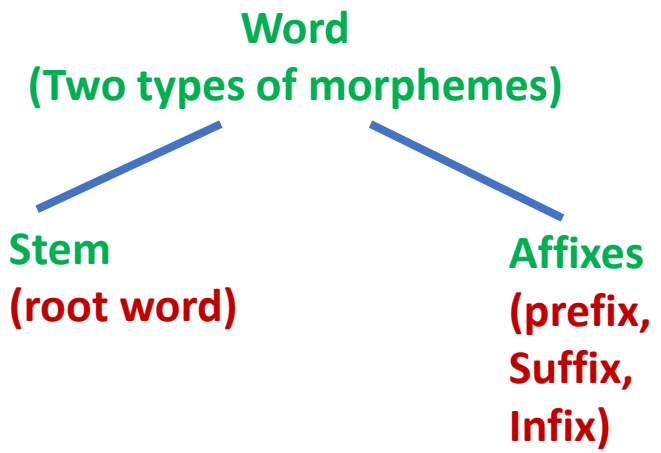
Morphological Parsing

- It is used to identify and find the number of morphemes in a given word.

- Morphemes are the smallest indivisible meaningful units of a language which builds a word.

- Example





- Steps to design Morphological parser
  - ✓ Lexicon
  - ✓ Morphotactic
  - ✓ Orthographic Rules.

Morphological Parsing

Lexicon

- ✓ Stores basic information about a word.
- ✓ Word is stem or affix
- ✓ If Stem, then whether a verb stem or noun stem.
- ✓ If affix, then whether a prefix, infix or suffix

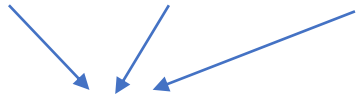
Morphotactic

- ✓ Set of rules to make a decisions
- ✓ Decides a word appear/not appear before, after or in between other words.
- ✓ For example

Use able ness

Useableness (Valid rule)

Able use ness



Ableuseness (Invalid rule)

Orthographic rules

Set of rules used to decide spelling changes.

For example

Baby + S =Babys

Baby+s = Babies

Morphological Analysis

- Morphology means study of word / making of word.
- Some words has their own meaning.
- Example

Camera Board Pen Table

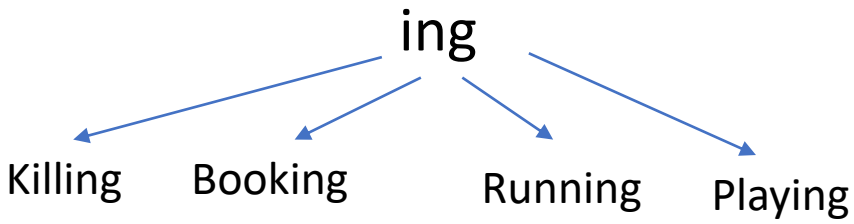
- Some words are there which when divided into different words, those new words have their own meaning.

Example

Morphological Analysis

- Some words are there which does not have their own meaning but when they are combined with other words, they become meaningful.

Example:



Finite State Automata

- A finite state automata is defined as M = {Q,Σ, δ, q_0, F}

Where

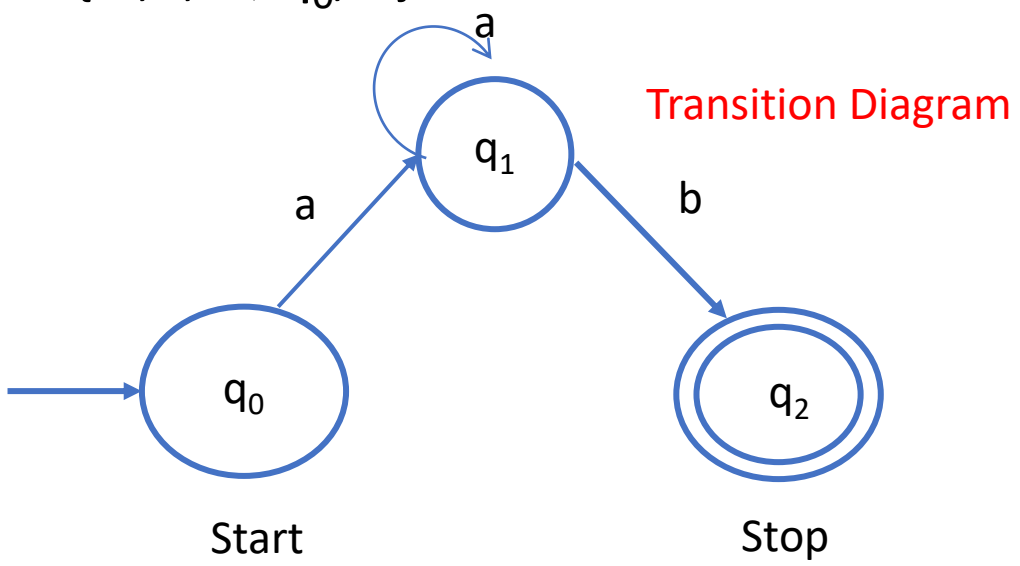
✓Q: Finite Non-empty Set of States

✓Σ : Finite Set of Input symbols

✓δ :Transition Mapping: Q × Σ ⟶ Q

✓q_0: Initial State of the Finite Automata

✓F : Finite Set of Final States



Example = a^+b

Q = {q_0,q_1,q_2}
Σ = {a,b}

Finite State Automata

Let's begin with the "sheep language" we discussed previously. Recall that we defined the sheep language as any string from the following (infinite) set:

baa!
baaa!
baaaa!
baaaaa!
...

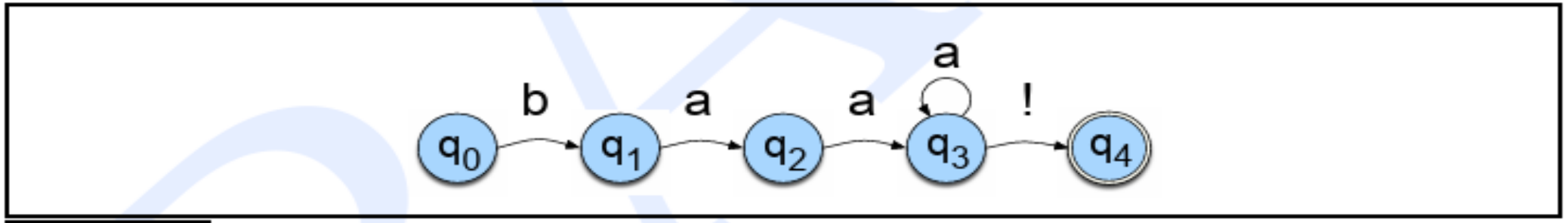


Figure 2.10 A finite-state automaton for talking sheep.

Algorithm for FSA

```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
   if End of input has been reached then
    if current-state is an accept state then
      return accept
    else
      return reject
   elsif transition-table[current-state,tape[index]] is empty then
     return reject
   else
     current-state ← transition-table[current-state,tape[index]]
     index ← index + 1
  end
```

Figure 2.12 An algorithm for deterministic recognition of FSAs. This algorithm returns *accept* if the entire string it is pointing at is in the language defined by the FSA, and *reject* if the string is not in the language.

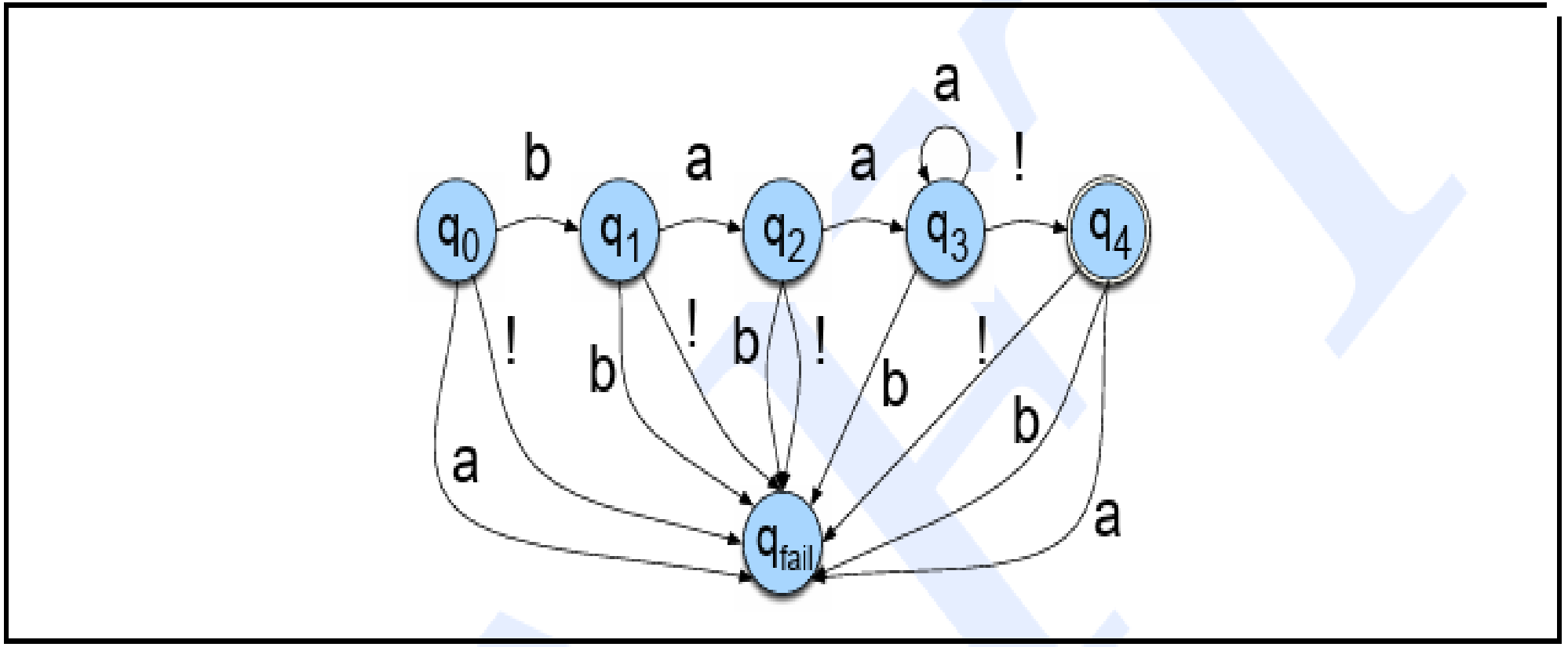
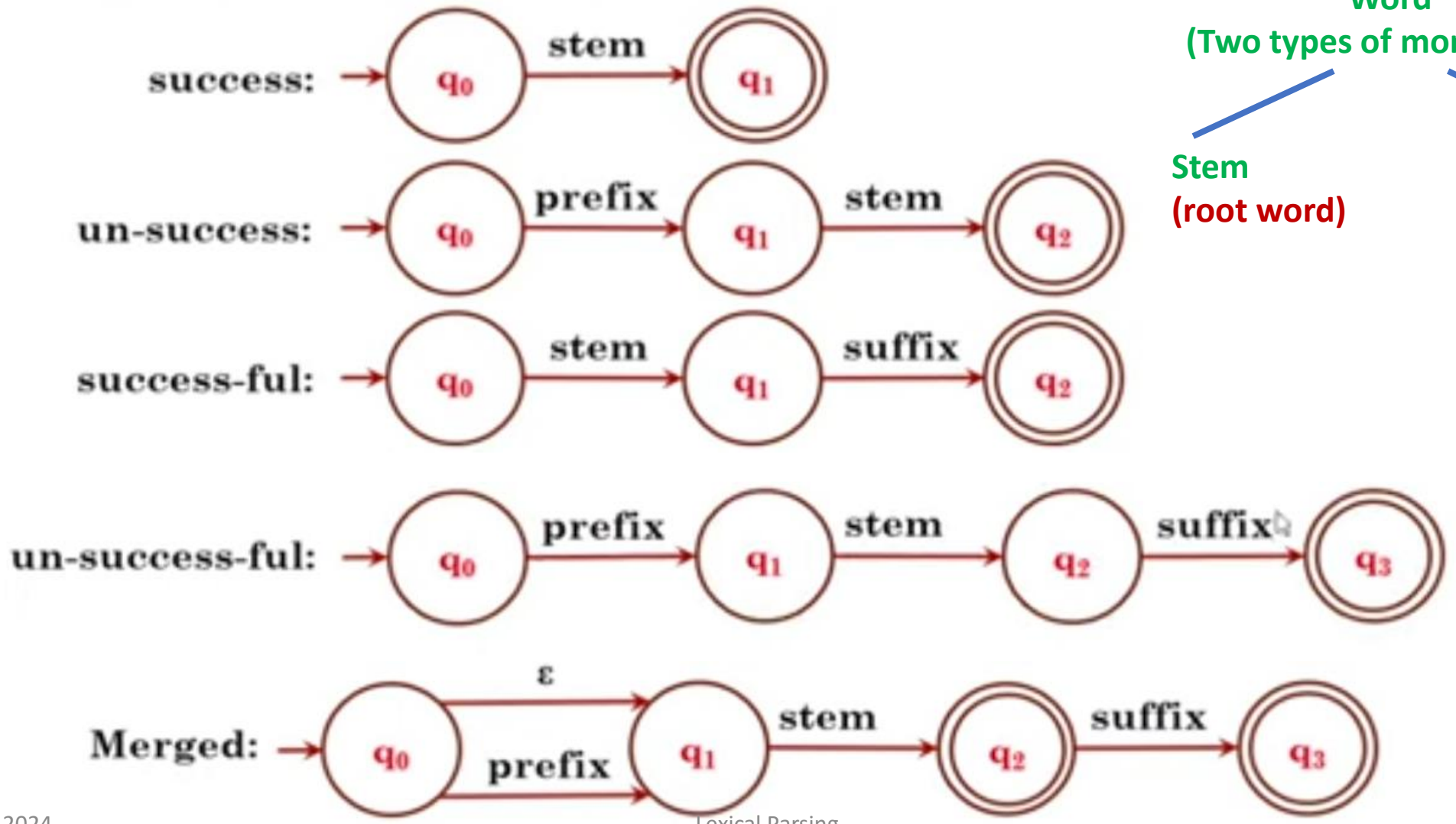


Figure 2.14 Adding a fail state to Fig. 2.10.

Finite State Automata for Morphology



Word
(Two types of morphemes)

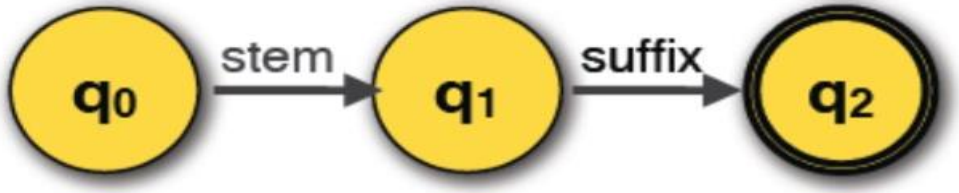
Stem
(root word)

Affixes
(prefix,
Suffix,
Infix)

grace:

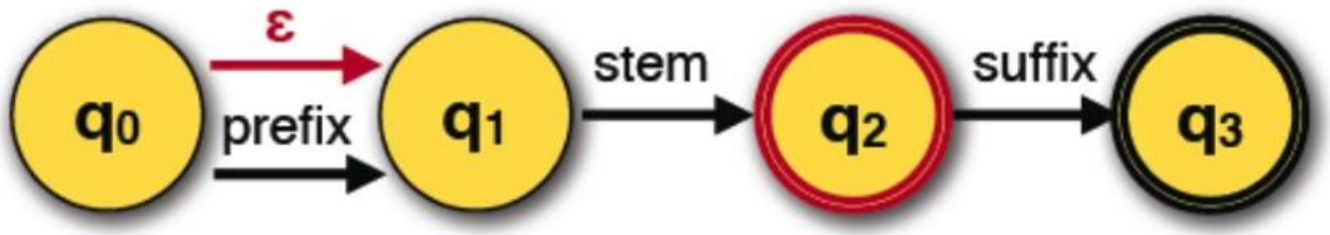
dis-grace:

grace-ful:



dis-grace-ful:

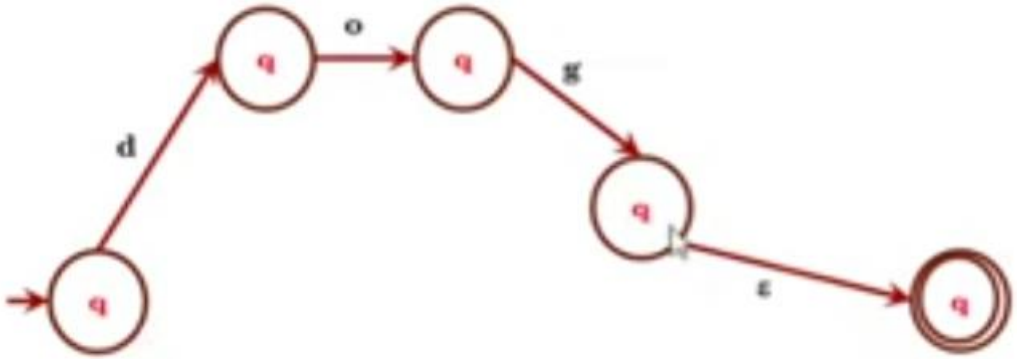
grace,
dis-grace,
grace-ful,
dis-grace-ful

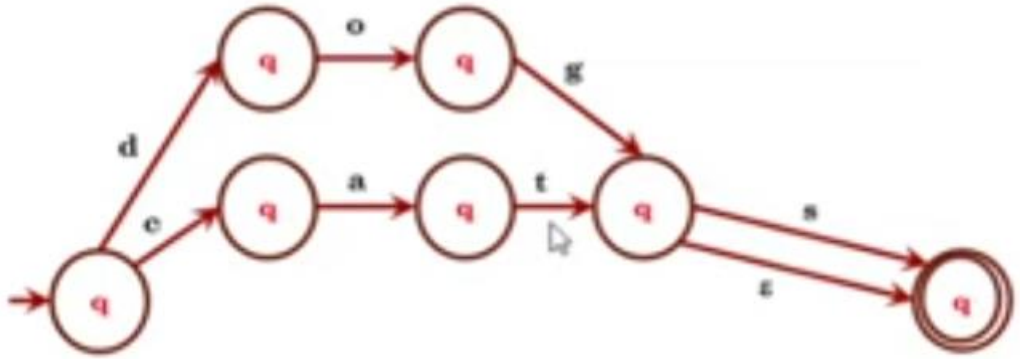


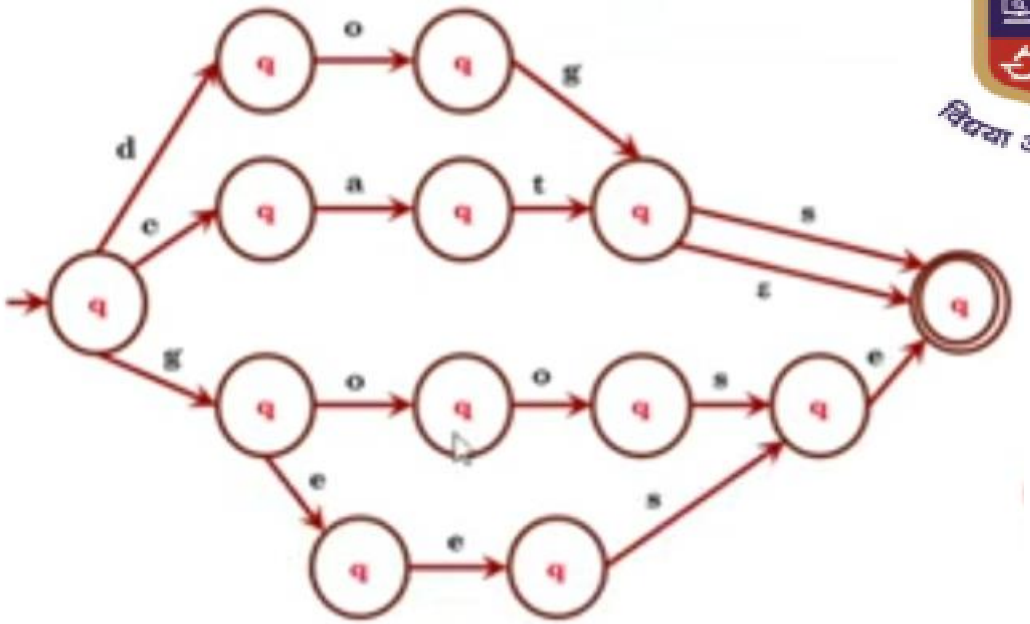
STEM CHANGES

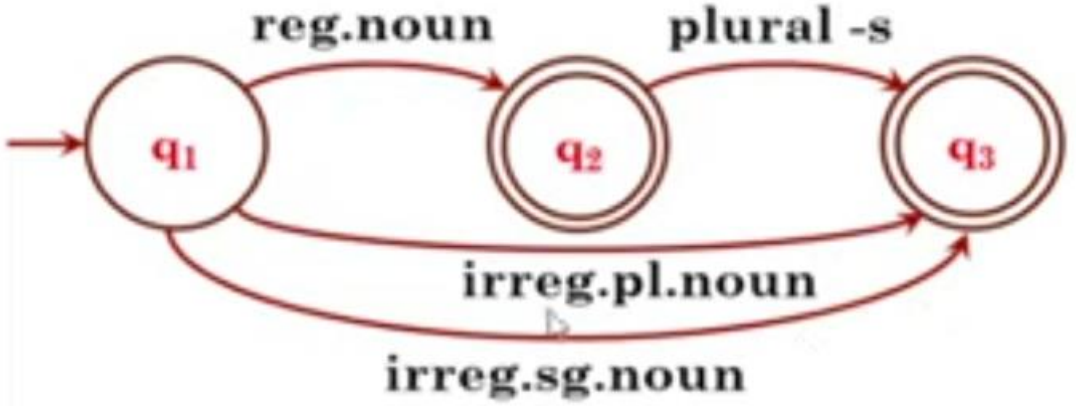
- Some irregular word requires stem changes.
- Example
  - ✓Goose -> geese
  - ✓Mouse ->Mice
  - ✓Teach -> Taught
  - ✓Go -> went

Reg-noun	Irreg-pl-noun	Irreg –sg -noun	plural
rat	geese	goose	-s
cat	taught	teach	
dog	mice	mouse	


d
o
g
s


d
o
g
a
t
s


d
o
g
a
t
s


reg.noun
plural -s
irreg.pl.noun
irreg.sg.noun

RECOGNITION vs ANALYSIS

- Finite State Automata can recognize/accept a string, but they cannot tell its internal structure.
- Thus, a machine is required to map/transduce the input string into an output string that encodes its internal structure.
- Finite State Transducers has two tapes for input and output as:

  Lexical Tape and Surface Tape.

Any one of the two tape can be either input tape or output tape.

Finite State Transducer

- A formal language is not the natural language, but it can be used to model part of natural languages such as phonology, morphology, etc.
- FSTs are FSAs with two tapes.
- AFST is 7 tuple, T= (Q,Σ, Γ, q0, F, δ, λ) where
  - ✓ Q: Finite Set of States
  - ✓ Σ : Finite set of Input Symbols
  - ✓ Γ : Finite set of output symbols
  - ✓ q_0 : Initial State
  - ✓ F : Set of final states
  - ✓ δ : Transition Function Mapping δ:Q x Σ -> 2Q
  - ✓ λ : Output Function Mapping λ :Q x{Σ U ε} -> Q x{ΓU ε}

Example on FST

- $Q = \{q_0 , q_1\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{x, y, z\}$
- $\{\Sigma \cup \varepsilon\} = \{a, b, \varepsilon\}$
- $\{\Gamma \cup \varepsilon\} = \{x, y, z, \varepsilon\}$
- λ : $\{ <0,a> , <0,b>, <0,c>, < 0, \varepsilon > \; <1,a> , <1,b>, <1,c>, <1, \varepsilon > \}$ x $\{ <0,x> , <0,y>, <0,z>, < 0, \varepsilon > \; <1,x> , <1,y>, <1,z>, <1, \varepsilon > \}$

THANK YOU

Regular Expression

Regular Expression

- A regular expression is a set of characters or pattern which is used to find substring in given string.

- Example:

✓Extracting hashtags from a given string

✓Getting Email Id or phone number from a larger unstructured text content

Basics of Regular Expression

- **Bracket([])**
- **Dash (–)**
- **Caret (^)**
- **Question Mark (?)**
- **Period (.)**

Basics of Regular Expression

1. Brackets ([]) :

- They are used to specify a disjunction of characters.
- For instance using brackets to put w/W allow us to return W or w.

- **Example**
  - ✓/[Ww]oodchuck/ -> Woodchuck or woodchuck
  - ✓/[abc]/ -> ‘a’ or ‘b’ or ‘c’
  - ✓/[1234567890]/ -> Any digit

Regex	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	“Woodchuck”
/[abc]/	‘a’, ‘b’, *or* ‘c’	“In uomini, in soldati”
/[1234567890]/	any digit	“plenty of 7 to 5”

Figure 2.2 The use of the brackets [] to specify a disjunction of characters.

Basics of Regular Expression

2. Dash (-)

- To specify a range.
- For instance putting A-Z in brackets allows R to return all matches of an upper case letters.

- **Example**
  - ✓[0-9] -> matches single digit
  - ✓/[A-Z]/ ->matches uppercase letters
  - ✓/[a-z]/-> matches lower case letters

Regex	Match	Example Patterns Matched
/[A-Z]/	an upper case letter	"we should call it 'Drenched Blossoms' "
/[a-z]/	a lower case letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

Figure 2.3 The use of the brackets [] plus the dash - to specify a range.

Basics of Regular Expression

3. Caret (^)

- It can be used for negation or just to mean ^.

Example

- ✓/[^A-Z]/-> not an upper case letters
- ✓/[^a-z]/-> not an lower case letters
- ✓/[^Ss]/-> Neither S nor s
- ✓/[^\.]/-> Not a period
- ✓/[e^]/->Either e or ^
- ✓/[a^b]/-> The pattern a^b

Regex	Match (single characters)	Example Patterns Matched
/[^A-Z]/	not an upper case letter	"Oyfn pripetchik"
/[^Ss]/	neither 'S' nor 's'	"I have no exquisite reason for't"
/[^.]/	not a period	"our resident Djinn"
/[e^]/	either 'e' or '^'	"look up ^ now"
/a^b/	the pattern 'a^b'	"look up a^ b now"

Figure 2.4 The caret ^ for negation or just to mean ^. See below re: the backslash for escaping the period.

Basics of Regular Expression

4. Question marks (?)

- It marks optionality of the previous expression.
- For instance putting a? at the end of chucks returns results for woodchuck (without an s) and woodchucks (with an s)

- **Example**
  - ✓/woodchucks?/ -> woodchuck/woodchucks
  - ✓/colou?r/ -> color or colour

Basics of Regular Expression

5. Period(.)

- Used to specify any characters between expressions
- For instance putting beg.n will return begin or began

- **Example**
  - ✓/beg.n -> match any characters between a to z

Anchor

- They are used to assert something about string or the matching process.

- **Example:** ^ and $
- ^The : Matches any string that starts with ‘The’.
- End$: Matches any string that ends with ‘End’.

Regex	Match
^	start of line
$	end of line
\b	word boundary
\B	non-word boundary

Figure 2.7 Anchors in regular expressions.

Character Classes

\d	**Returns a match where the string contains digits (numbers from 0-9)**
\D	Returns a match where the string DOES NOT contain digits
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)
\W	Returns a match where the string DOES NOT contain any word characters
\s	Returns a match where the string contains a white space character
\S	Returns a match where the string DOES NOT contain a white space character
\Z	Returns a match if the specified characters are at the end of the string
\A	Returns a match if the specified characters are at the beginning of the string

Quantifier (* + ? And {})

- abc* : ab followed by zero or more c.
- abc+ : ab followed by one or more c.
- abc? : ab followed by one or zero c.
- abc{2} : ab followed by 2c.
- abc{2,} : ab followed by 2 or more c.
- abc{2,5} : ab follower by 2 up to 5c.
- A{bc}* : a followed by zero or more copies of the sequence bc.

Program 1

```
import re
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
print (x)
```

Program 2

```
import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start())
```

Program 3

```
import re

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

Program 4

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

Program 5

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

Program 6

Program 7

THANK YOU

CHAPTER

A Hidden Markov Models

Chapter 17 introduced the Hidden Markov Model and applied it to part of speech tagging. Part of speech tagging is a fully-supervised learning task, because we have a corpus of words labeled with the correct part-of-speech tag. But many applications don't have labeled data. So in this chapter, we introduce the full set of algorithms for HMMs, including the key unsupervised learning algorithm for HMM, the Forward-Backward algorithm. We'll repeat some of the text from Chapter 17 for readers who want the whole story laid out in a single chapter.

A.1 Markov Chains

Markov chain

The HMM is based on augmenting the Markov chain. A **Markov chain** is a model that tells us something about the probabilities of sequences of random variables, *states*, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. The states before the current state have no impact on the future except via the current state. It's as if to predict tomorrow's weather you could examine today's weather but you weren't allowed to look at yesterday's weather.

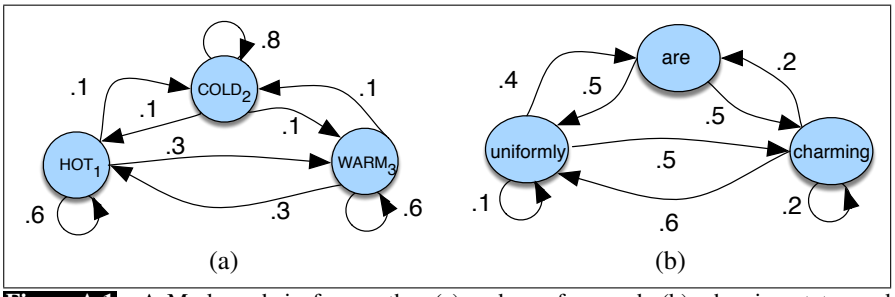


Figure A.1 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Markov assumption

More formally, consider a sequence of state variables $q_1, q_2, ..., q_i$. A Markov model embodies the **Markov assumption** on the probabilities of this sequence: that when predicting the future, the past doesn't matter, only the present.

$$\textbf{Markov Assumption:} \quad P(q_i = a | q_1 ... q_{i-1}) = P(q_i = a | q_{i-1}) \tag{A.1}$$

Figure A.1a shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of HOT, COLD, and WARM. The states are represented as nodes in the graph, and the transitions, with their probabilities, as edges. The transitions are probabilities: the values of arcs leaving a given

state must sum to 1. Figure A.1b shows a Markov chain for assigning a probability to a sequence of words $w_1 ... w_n$. This Markov chain should be familiar; in fact, it represents a bigram language model, with each edge expressing the probability $p(w_i|w_j)$! Given the two models in Fig. A.1, we can assign a probability to any sequence from our vocabulary.

Formally, a Markov chain is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{N} \pi_i = 1$

Before you go on, use the sample probabilities in Fig. A.1a (with $\pi = [.1, .7., 2]$) to compute the probability of each of the following sequences:

(A.2) hot hot hot hot
(A.3) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. A.1a?

A.2 The Hidden Markov Model

A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are **hidden**: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence. We call the tags **hidden** because they are not observed.

hidden

Hidden Markov model

A **hidden Markov model** (**HMM**) allows us to talk about both *observed* events (like words that we see in the input) and *hidden* events (like part-of-speech tags) that we think of as causal factors in our probabilistic model. An HMM is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t (drawn from a vocabulary $V = v_1, v_2, ..., v_V$) being generated from a state q_i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

The HMM is given as input $O = o_1 o_2 \ldots o_T$: a sequence of T **observations**, each one drawn from the vocabulary V.

A first-order hidden Markov model instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state depends

only on the previous state:

$$\textbf{Markov Assumption:} \quad P(q_i|q_1 \ldots q_{i-1}) = P(q_i|q_{i-1}) \tag{A.4}$$

Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations:

$$\textbf{Output Independence:} \quad P(o_i|q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i) \tag{A.5}$$

To exemplify these models, we'll use a task invented by Jason Eisner (2002). Imagine that you are a climatologist in the year 2799 studying the history of global warming. You cannot find any records of the weather in Baltimore, Maryland, for the summer of 2020, but you do find Jason Eisner's diary, which lists how many ice creams Jason ate every day that summer. Our goal is to use these observations to estimate the temperature every day. We'll simplify this weather task by assuming there are only two kinds of days: cold (C) and hot (H). So the Eisner task is as follows:

> Given a sequence of observations O (each an integer representing the number of ice creams eaten on a given day) find the 'hidden' sequence Q of weather states (H or C) which caused Jason to eat the ice cream.

Figure A.2 shows a sample HMM for the ice cream task. The two hidden states (H and C) correspond to hot and cold weather, and the observations (drawn from the alphabet $O = \{1,2,3\}$) correspond to the number of ice creams eaten by Jason on a given day.

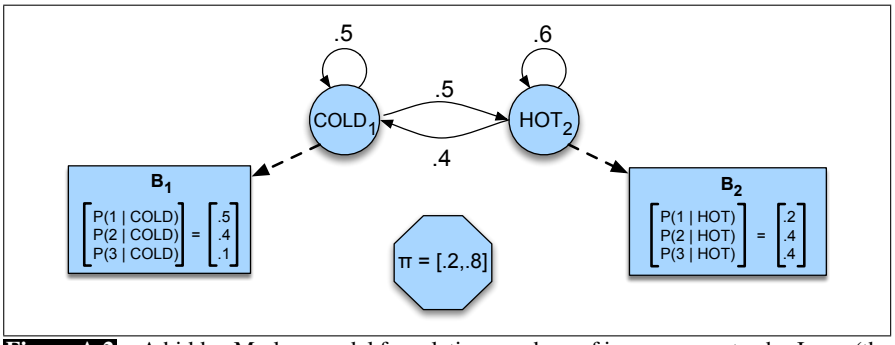


Figure A.2 A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

An influential tutorial by Rabiner (1989), based on tutorials by Jack Ferguson in the 1960s, introduced the idea that hidden Markov models should be characterized by **three fundamental problems**:

Problem 1 (Likelihood):	Given an HMM $\lambda = (A,B)$ and an observation sequence O, determine the likelihood $P(O\|\lambda)$.
Problem 2 (Decoding):	Given an observation sequence O and an HMM $\lambda = (A,B)$, discover the best hidden state sequence Q.
Problem 3 (Learning):	Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

We already saw an example of Problem 2 in Chapter 17. In the next two sections we introduce the Forward and Forward-Backward algorithms to solve Problems 1 and 3 and give more information on Problem 2

A.3 Likelihood Computation: The Forward Algorithm

Our first problem is to compute the likelihood of a particular observation sequence. For example, given the ice-cream eating HMM in Fig. A.2, what is the probability of the sequence *3 1 3*? More formally:

> **Computing Likelihood:** Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$.

For a Markov chain, where the surface observations are the same as the hidden events, we could compute the probability of *3 1 3* just by following the states labeled *3 1 3* and multiplying the probabilities along the arcs. For a hidden Markov model, things are not so simple. We want to determine the probability of an ice-cream observation sequence like *3 1 3*, but we don't know what the hidden state sequence is!

Let's start with a slightly simpler situation. Suppose we already knew the weather and wanted to predict how much ice cream Jason would eat. This is a useful part of many HMM tasks. For a given hidden state sequence (e.g., *hot hot cold*), we can easily compute the output likelihood of *3 1 3*.

Let's see how. First, recall that for hidden Markov models, each hidden state produces only a single observation. Thus, the sequence of hidden states and the sequence of observations have the same length. [1]

Given this one-to-one mapping and the Markov assumptions expressed in Eq. A.4, for a particular hidden state sequence $Q = q_1, q_2, ..., q_T$ and an observation sequence $O = o_1, o_2, ..., o_T$, the likelihood of the observation sequence is

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i) \tag{A.6}$$

The computation of the forward probability for our ice-cream observation *3 1 3* from one possible hidden state sequence *hot hot cold* is shown in Eq. A.7. Figure A.3 shows a graphic representation of this computation.

$$P(3\ 1\ 3|\text{hot hot cold}) = P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \tag{A.7}$$

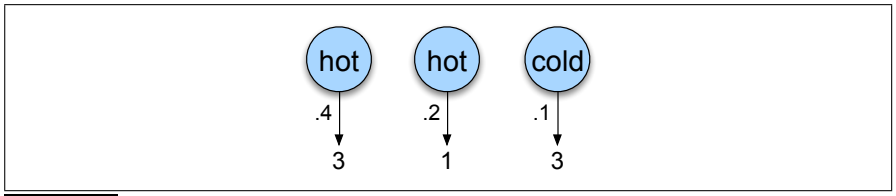


Figure A.3 The computation of the observation likelihood for the ice-cream events *3 1 3* given the hidden state sequence *hot hot cold*.

But of course, we don't actually know what the hidden state (weather) sequence was. We'll need to compute the probability of ice-cream events *3 1 3* instead by

[1] In a variant of HMMs called **segmental HMMs** (in speech recognition) or **semi-HMMs** (in text processing) this one-to-one mapping between the length of the hidden state sequence and the length of the observation sequence does not hold.

summing over all possible weather sequences, weighted by their probability. First, let's compute the joint probability of being in a particular weather sequence Q and generating a particular sequence O of ice-cream events. In general, this is

$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{T} P(o_i|q_i) \times \prod_{i=1}^{T} P(q_i|q_{i-1}) \tag{A.8}$$

The computation of the joint probability of our ice-cream observation *3 1 3* and one possible hidden state sequence *hot hot cold* is shown in Eq. A.9. Figure A.4 shows a graphic representation of this computation.

$$\begin{aligned} P(3\ 1\ 3, \text{hot hot cold}) &= P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned} \tag{A.9}$$

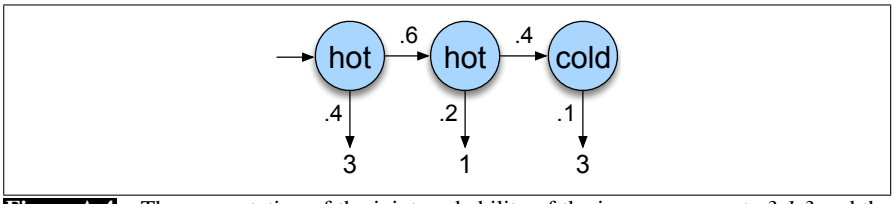


Figure A.4 The computation of the joint probability of the ice-cream events *3 1 3* and the hidden state sequence *hot hot cold*.

Now that we know how to compute the joint probability of the observations with a particular hidden state sequence, we can compute the total probability of the observations just by summing over all possible hidden state sequences:

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q) \tag{A.10}$$

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3\ 1\ 3) = P(3\ 1\ 3, \text{cold cold cold}) + P(3\ 1\ 3, \text{cold cold hot}) + P(3\ 1\ 3, \text{hot hot cold}) + \ldots$$

For an HMM with N hidden states and an observation sequence of T observations, there are N^T possible hidden sequences. For real tasks, where N and T are both large, N^T is a very large number, so we cannot compute the total observation likelihood by computing a separate observation likelihood for each hidden state sequence and then summing them.

forward algorithm

Instead of using such an extremely exponential algorithm, we use an efficient $O(N^2T)$ algorithm called the **forward algorithm**. The forward algorithm is a kind of **dynamic programming** algorithm, that is, an algorithm that uses a table to store intermediate values as it builds up the probability of the observation sequence. The forward algorithm computes the observation probability by summing over the probabilities of all possible hidden state paths that could generate the observation sequence, but it does so efficiently by implicitly folding each of these paths into a single **forward trellis**.

Figure A.5 shows an example of the forward trellis for computing the likelihood of *3 1 3* given the hidden state sequence *hot hot cold*.

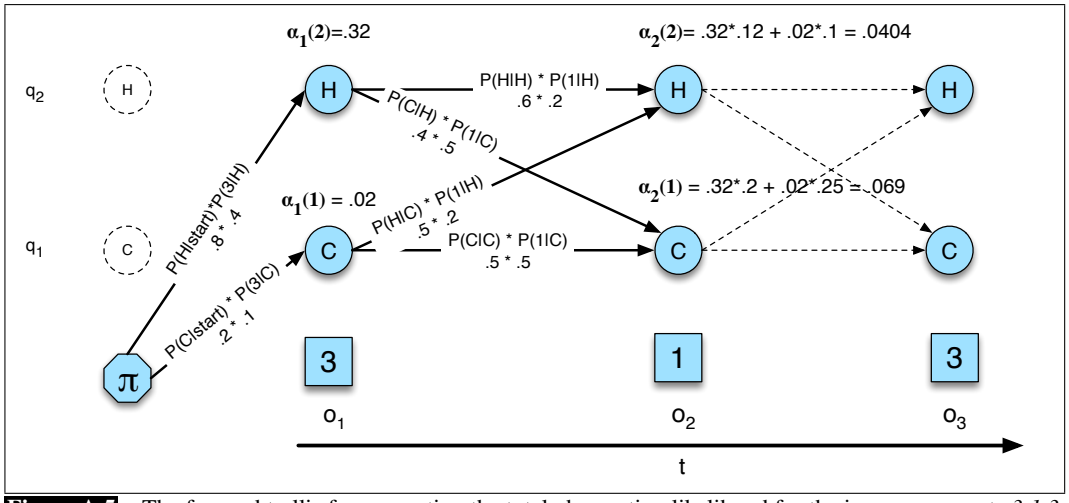


Figure A.5 The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Each cell of the forward algorithm trellis $\alpha_t(j)$ represents the probability of being in state j after seeing the first t observations, given the automaton λ. The value of each cell $\alpha_t(j)$ is computed by summing over the probabilities of every path that could lead us to this cell. Formally, each cell expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda) \tag{A.11}$$

Here, $q_t = j$ means "the t^{th} state in the sequence of states is state j". We compute this probability $\alpha_t(j)$ by summing over the extensions of all the paths that lead to the current cell. For a given state q_j at time t, the value $\alpha_t(j)$ is computed as

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{A.12}$$

The three factors that are multiplied in Eq. A.12 in extending the previous paths to compute the forward probability at time t are

$\alpha_{t-1}(i)$	the **previous forward path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

Consider the computation in Fig. A.5 of $\alpha_2(2)$, the forward probability of being at time step 2 in state 2 having generated the partial observation *3 1*. We compute by extending the α probabilities from time step 1, via two paths, each extension consisting of the three factors above: $\alpha_1(1) \times P(H|C) \times P(1|H)$ and $\alpha_1(2) \times P(H|H) \times P(1|H)$.

Figure A.6 shows another visualization of this induction step for computing the value in one new cell of the trellis.

We give two formal definitions of the forward algorithm: the pseudocode in Fig. A.7 and a statement of the definitional recursion here.

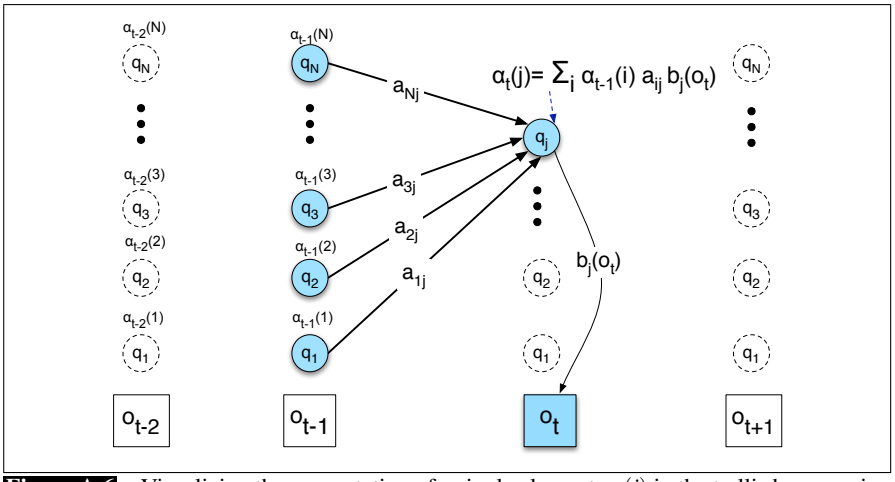


Figure A.6 Visualizing the computation of a single element $\alpha_t(i)$ in the trellis by summing all the previous values α_{t-1}, weighted by their transition probabilities a, and multiplying by the observation probability $b_i(o_t)$. For many applications of HMMs, many of the transition probabilities are 0, so not all previous states will contribute to the forward probability of the current state. Hidden states are in circles, observations in squares. Shaded nodes are included in the probability computation for $\alpha_t(i)$.

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$forward[s,1] \leftarrow \pi_s * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T]$$; termination step
return *forwardprob*

Figure A.7 The forward algorithm, where $forward[s,t]$ represents $\alpha_t(s)$.

1. Initialization:

$$\alpha_1(j) = \pi_j b_j(o_1) \quad 1 \le j \le N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

A.4 Decoding: The Viterbi Algorithm

decoding

For any model, such as an HMM, that contains hidden variables, the task of determining which sequence of variables is the underlying source of some sequence of observations is called the **decoding** task. In the ice-cream domain, given a sequence of ice-cream observations *3 1 3* and an HMM, the task of the **decoder** is to find the best hidden weather sequence (*H H H*). More formally,

> **Decoding**: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

We might propose to find the best sequence as follows: For each possible hidden state sequence (*HHH*, *HHC*, *HCH*, etc.), we could run the forward algorithm and compute the likelihood of the observation sequence given that hidden state sequence. Then we could choose the hidden state sequence with the maximum observation likelihood. It should be clear from the previous section that we cannot do this because there are an exponentially large number of state sequences.

Viterbi algorithm

Instead, the most common decoding algorithms for HMMs is the **Viterbi algorithm**. Like the forward algorithm, **Viterbi** is a kind of **dynamic programming** that makes uses of a dynamic programming trellis. Viterbi also strongly resembles another dynamic programming variant, the **minimum edit distance** algorithm of Chapter 2.

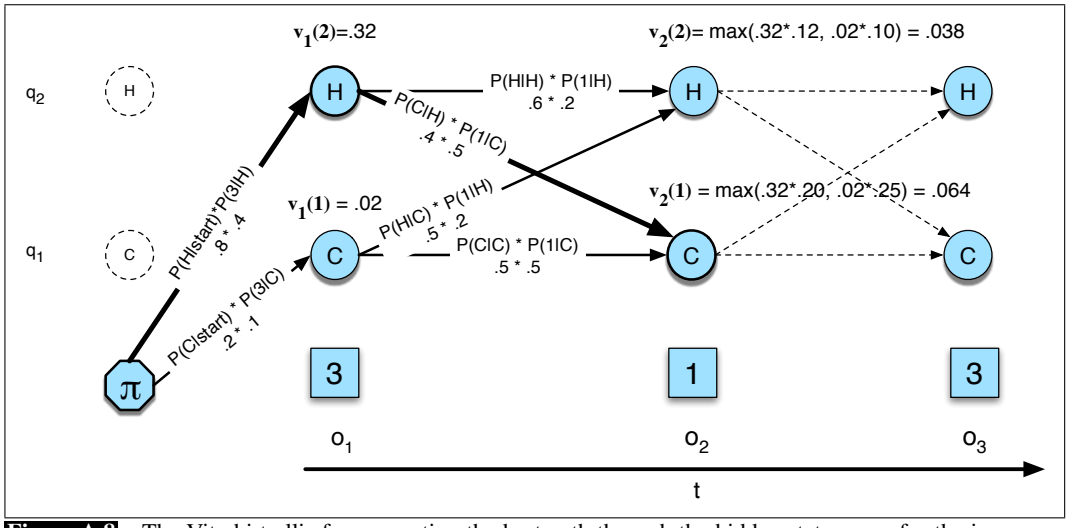


Figure A.8 The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $v_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.14: $v_t(j) = \max_{1 \le i \le N-1} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.13: $v_t(j) = P(q_0, q_1, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j|\lambda)$.

Figure A.8 shows an example of the Viterbi trellis for computing the best hidden state sequence for the observation sequence *3 1 3*. The idea is to process the observation sequence left to right, filling out the trellis. Each cell of the trellis, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $q_1, ..., q_{t-1}$, given the

automaton λ. The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability

$$v_t(j) = \max_{q_1,\cdots,q_{t-1}} P(q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda) \tag{A.13}$$

Note that we represent the most probable path by taking the maximum over all possible previous state sequences $\max_{q_1,\cdots,q_{t-1}}$. Like other dynamic programming algorithms, Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time $t-1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state q_j at time t, the value $v_t(j)$ is computed as

$$v_t(j) \;=\; \max_{i=1}^{N} v_{t-1}(i)\; a_{ij}\; b_j(o_t) \tag{A.14}$$

The three factors that are multiplied in Eq. A.14 for extending the previous paths to compute the Viterbi probability at time t are

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  viterbi[s,1] $\leftarrow \pi_s * b_s(o_1)$
  backpointer[s,1] $\leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    viterbi[s,t] $\leftarrow \max_{s'=1}^{N}$ *viterbi*$[s',t-1] * a_{s',s} * b_s(o_t)$
    backpointer[s,t] $\leftarrow \operatorname{argmax}_{s'=1}^{N}$ *viterbi*$[s',t-1] * a_{s',s} * b_s(o_t)$
bestpathprob $\leftarrow \max_{s=1}^{N}$ *viterbi*$[s,T]$; termination step
bestpathpointer $\leftarrow \operatorname{argmax}_{s=1}^{N}$ *viterbi*$[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure A.9 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Figure A.9 shows pseudocode for the Viterbi algorithm. Note that the Viterbi algorithm is identical to the forward algorithm except that it takes the **max** over the previous path probabilities whereas the forward algorithm takes the **sum**. Note also that the Viterbi algorithm has one component that the forward algorithm doesn't

have: **backpointers**. The reason is that while the forward algorithm needs to produce an observation likelihood, the Viterbi algorithm must produce a probability and also the most likely state sequence. We compute this best state sequence by keeping track of the path of hidden states that led to each state, as suggested in Fig. A.10, and then at the end backtracing the best path to the beginning (the Viterbi **backtrace**).

Viterbi backtrace

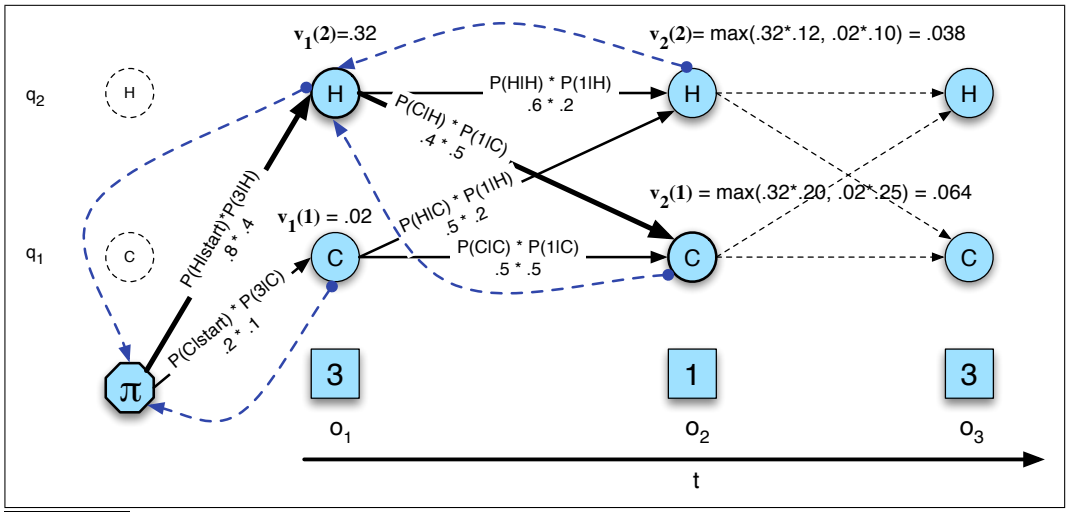


Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Finally, we can give a formal definition of the Viterbi recursion as follows:

1. **Initialization:**

$$\begin{aligned} v_1(j) &= \pi_j b_j(o_1) \qquad 1 \le j \le N \\ bt_1(j) &= 0 \qquad 1 \le j \le N \end{aligned}$$

2. **Recursion**

$$\begin{aligned} v_t(j) &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \\ bt_t(j) &= \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \end{aligned}$$

3. **Termination:**

$$\begin{aligned} \text{The best score:} \quad P* &= \max_{i=1}^{N} v_T(i) \\ \text{The start of backtrace:} \quad q_T* &= \operatorname*{argmax}_{i=1}^{N} v_T(i) \end{aligned}$$

A.5 HMM Training: The Forward-Backward Algorithm

We turn to the third problem for HMMs: learning the parameters of an HMM, that is, the A and B matrices. Formally,

> **Learning:** Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B.

The input to such a learning algorithm would be an unlabeled sequence of observations O and a vocabulary of potential hidden states Q. Thus, for the ice cream task, we would start with a sequence of observations $O = \{1, 3, 2, ..., \}$ and the set of hidden states H and C.

Forward-backward
Baum-Welch
EM

The standard algorithm for HMM training is the **forward-backward**, or **Baum-Welch** algorithm (Baum, 1972), a special case of the **Expectation-Maximization** or **EM** algorithm (Dempster et al., 1977). The algorithm will let us train both the transition probabilities A and the emission probabilities B of the HMM. EM is an *iterative* algorithm, computing an initial estimate for the probabilities, then using those estimates to compute a better estimate, and so on, iteratively improving the probabilities that it learns.

Let us begin by considering the much simpler case of training a fully visible Markov model, where we know both the temperature and the ice cream count for every day. That is, imagine we see the following set of input observations and magically knew the aligned hidden state sequences:

```
3    3   2        1    1    2        1    2  3
hot  hot cold     cold cold cold    cold hot hot
```

This would easily allow us to compute the HMM parameters just by maximum likelihood estimation from the training data. First, we can compute π from the count of the 3 initial hidden states:

$$\pi_h = 1/3 \quad \pi_c = 2/3$$

Next we can directly compute the A matrix from the transitions, ignoring the final hidden states:

$$p(hot|hot) = 2/3 \quad p(cold|hot) = 1/3$$
$$p(cold|cold) = 2/3 \quad p(hot|cold) = 1/3$$

and the B matrix:

$$P(1|hot) = 0/4 = 0 \quad p(1|cold) = 3/5 = .6$$
$$P(2|hot) = 1/4 = .25 \quad p(2|cold = 2/5 = .4$$
$$P(3|hot) = 3/4 = .75 \quad p(3|cold) = 0$$

For a real HMM, we cannot compute these counts directly from an observation sequence since we don't know which path of states was taken through the machine for a given input. For example, suppose I didn't tell you the temperature on day 2, and you had to guess it, but you (magically) had the above probabilities, and the temperatures on the other days. You could do some Bayesian arithmetic with all the other probabilities to get estimates of the likely temperature on that missing day, and use those to get expected counts for the temperatures for day 2.

But the real problem is even harder: we don't know the counts of being in **any** of the hidden states!! The Baum-Welch algorithm solves this by *iteratively* estimating the counts. We will start with an estimate for the transition and observation probabilities and then use these estimated probabilities to derive better and better probabilities. And we're going to do this by computing the forward probability for an observation and then dividing that probability mass among all the different paths that contributed to this forward probability.

backward probability

To understand the algorithm, we need to define a useful probability related to the forward probability and called the **backward probability**. The backward probabil-

ity β is the probability of seeing the observations from time $t+1$ to the end, given that we are in state i at time t (and given the automaton λ):

$$\beta_t(i) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda) \tag{A.15}$$

It is computed inductively in a similar manner to the forward algorithm.

1. **Initialization:**

$$\beta_T(i) \;=\; 1, \quad 1 \le i \le N$$

2. **Recursion**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \le i \le N, 1 \le t < T$$

3. **Termination:**

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j\, b_j(o_1)\, \beta_1(j)$$

Figure A.11 illustrates the backward induction step.

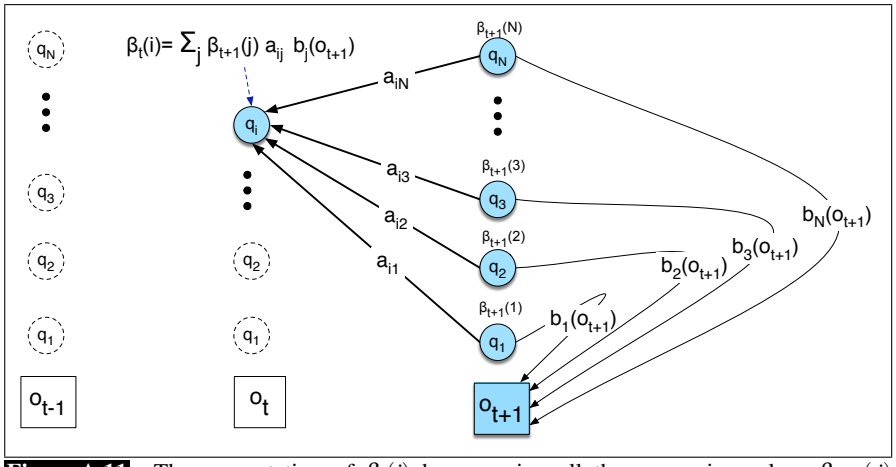


Figure A.11 The computation of $\beta_t(i)$ by summing all the successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and their observation probabilities $b_j(o_{t+1})$.

We are now ready to see how the forward and backward probabilities can help compute the transition probability a_{ij} and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path taken through the model is hidden.

Let's begin by seeing how to estimate $\hat{a}_{ij}$ by a variant of simple maximum likelihood estimation:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \tag{A.16}$$

How do we compute the numerator? Here's the intuition. Assume we had some estimate of the probability that a given transition $i \to j$ was taken at a particular point in time t in the observation sequence. If we knew this probability for each particular time t, we could sum over all times t to estimate the total count for the transition $i \to j$.

More formally, let's define the probability ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence and of course the model:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \tag{A.17}$$

To compute ξ_t, we first compute a probability which is similar to ξ_t, but differs in including the probability of the observation; note the different conditioning of O from Eq. A.17:

$$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O | \lambda) \tag{A.18}$$

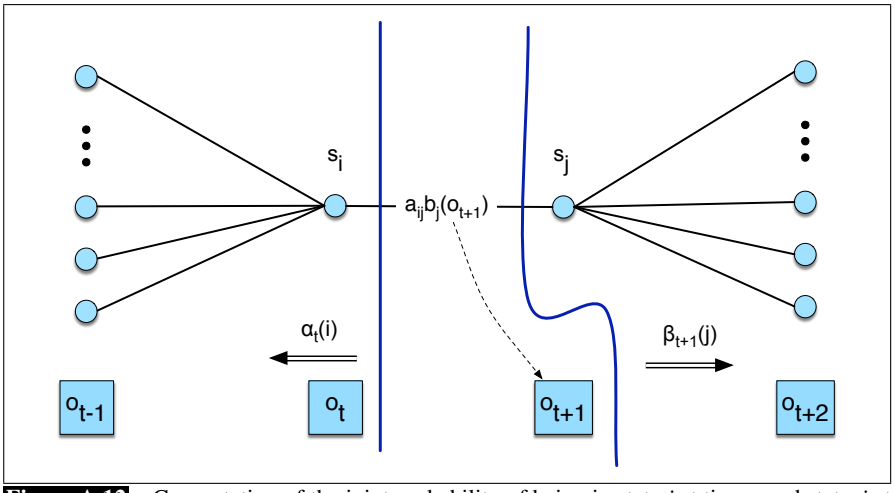


Figure A.12 Computation of the joint probability of being in state i at time t and state j at time $t+1$. The figure shows the various probabilities that need to be combined to produce $P(q_t = i, q_{t+1} = j, O|\lambda)$: the α and β probabilities, the transition probability a_{ij} and the observation probability $b_j(o_{t+1})$. After Rabiner (1989) which is ©1989 IEEE.

Figure A.12 shows the various probabilities that go into computing not-quite-ξ_t: the transition probability for the arc in question, the α probability before the arc, the β probability after the arc, and the observation probability for the symbol just after the arc. These four are multiplied together to produce *not-quite-*ξ_t as follows:

$$\text{not-quite-}\xi_t(i,j) = \alpha_t(i)\, a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \tag{A.19}$$

To compute ξ_t from *not-quite-*ξ_t, we follow the laws of probability and divide by $P(O|\lambda)$, since

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{A.20}$$

The probability of the observation given the model is simply the forward probability of the whole utterance (or alternatively, the backward probability of the whole utterance):

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_t(j) \beta_t(j) \tag{A.21}$$

So, the final equation for ξ_t is

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{A.22}$$

The expected number of transitions from state i to state j is then the sum over all t of ξ. For our estimate of a_{ij} in Eq. A.16, we just need one more thing: the total expected number of transitions from state i. We can get this by summing over all transitions out of state i. Here's the final formula for $\hat{a}_{ij}$:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)} \tag{A.23}$$

We also need a formula for recomputing the observation probability. This is the probability of a given symbol v_k from the observation vocabulary V, given a state j: $\hat{b}_j(v_k)$. We will do this by trying to compute

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \tag{A.24}$$

For this, we will need to know the probability of being in state j at time t, which we will call $\gamma_t(j)$:

$$\gamma_t(j) = P(q_t = j | O, \lambda) \tag{A.25}$$

Once again, we will compute this by including the observation sequence in the probability:

$$\gamma_t(j) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \tag{A.26}$$

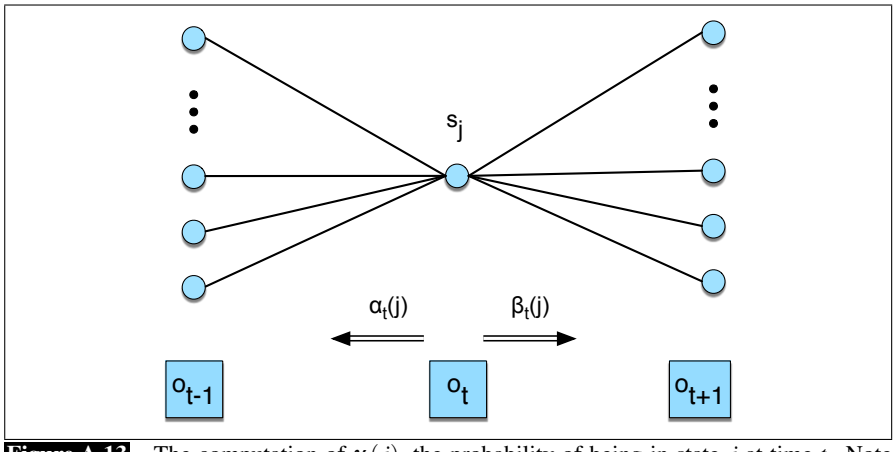


Figure A.13 The computation of $\gamma_t(j)$, the probability of being in state j at time t. Note that γ is really a degenerate case of ξ and hence this figure is like a version of Fig. A.12 with state i collapsed with state j. After Rabiner (1989) which is ©1989 IEEE.

As Fig. A.13 shows, the numerator of Eq. A.26 is just the product of the forward probability and the backward probability:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \tag{A.27}$$

We are ready to compute b. For the numerator, we sum $\gamma_t(j)$ for all time steps t in which the observation o_t is the symbol v_k that we are interested in. For the denominator, we sum $\gamma_t(j)$ over all time steps t. The result is the percentage of the times that we were in state j and saw symbol v_k (the notation $\sum_{t=1\ s.t. O_t=v_k}^{T}$ means "sum over all t for which the observation at time t was v_k"):

$$\hat{b}_j(v_k) = \frac{\sum_{t=1\ s.t. O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{A.28}$$

We now have ways in Eq. A.23 and Eq. A.28 to *re-estimate* the transition A and observation B probabilities from an observation sequence O, assuming that we already have a previous estimate of A and B.

These re-estimations form the core of the iterative forward-backward algorithm. The forward-backward algorithm (Fig. A.14) starts with some initial estimate of the HMM parameters $\lambda = (A, B)$. We then iteratively run two steps. Like other cases of the EM (expectation-maximization) algorithm, the forward-backward algorithm has two steps: the **expectation** step, or **E-step**, and the **maximization** step, or **M-step**.

E-step

M-step

In the E-step, we compute the expected state occupancy count γ and the expected state transition count ξ from the earlier A and B probabilities. In the M-step, we use γ and ξ to recompute new A and B probabilities.

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** *HMM=(A,B)*

initialize A and B
iterate until convergence
E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

Figure A.14 The forward-backward algorithm.

Although in principle the forward-backward algorithm can do completely unsupervised learning of the A and B parameters, in practice the initial conditions are very important. For this reason the algorithm is often given extra information. For example, for HMM-based speech recognition, the HMM structure is often set by hand, and only the emission (B) and (non-zero) A transition probabilities are trained from a set of observation sequences O.

A.6 Summary

This chapter introduced the **hidden Markov model** for probabilistic **sequence classification**.

- Hidden Markov models (**HMMs**) are a way of relating a sequence of **observations** to a sequence of **hidden classes** or **hidden states** that explain the observations.
- The process of discovering the sequence of hidden states, given the sequence of observations, is known as **decoding** or **inference**. The **Viterbi** algorithm is commonly used for decoding.
- The parameters of an HMM are the A transition probability matrix and the B observation likelihood matrix. Both can be trained with the **Baum-Welch** or **forward-backward** algorithm.

Bibliographical and Historical Notes

As we discussed in Chapter 17, Markov chains were first used by Markov (1913) (translation Markov 2006), to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant. The hidden Markov model was developed by Baum and colleagues at the Institute for Defense Analyses in Princeton (Baum and Petrie 1966, Baum and Eagon 1967).

The **Viterbi** algorithm was first applied to speech and language processing in the context of speech recognition by Vintsyuk (1968) but has what Kruskal (1983) calls a "remarkable history of multiple independent discovery and publication". Kruskal and others give at least the following independently-discovered variants of the algorithm published in four separate fields:

Citation	Field
Viterbi (1967)	information theory
Vintsyuk (1968)	speech processing
Needleman and Wunsch (1970)	molecular biology
Sakoe and Chiba (1971)	speech processing
Sankoff (1972)	molecular biology
Reichert et al. (1973)	molecular biology
Wagner and Fischer (1974)	computer science

The use of the term **Viterbi** is now standard for the application of dynamic programming to any kind of probabilistic maximization problem in speech and language processing. For non-probabilistic problems (such as for minimum edit distance), the plain term **dynamic programming** is often used. Forney, Jr. (1973) wrote an early survey paper that explores the origin of the Viterbi algorithm in the context of information and communications theory.

Our presentation of the idea that hidden Markov models should be characterized by three fundamental problems was modeled after an influential tutorial by Rabiner (1989), which was itself based on tutorials by Jack Ferguson of IDA in the 1960s. Jelinek (1997) and Rabiner and Juang (1993) give very complete descriptions of the forward-backward algorithm as applied to the speech recognition problem. Jelinek (1997) also shows the relationship between forward-backward and EM.

Baum, L. E. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities III: Proceedings of the 3rd Symposium on Inequalities*. Academic Press.

Baum, L. E. and J. A. Eagon. 1967. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360–363.

Baum, L. E. and T. Petrie. 1966. Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics*, 37(6):1554–1563.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, 39(1):1–21.

Eisner, J. 2002. An interactive spreadsheet for teaching the forward-backward algorithm. *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*.

Forney, Jr., G. D. 1973. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Jelinek, F. 1997. *Statistical Methods for Speech Recognition*. MIT Press.

Kruskal, J. B. 1983. An overview of sequence comparison. In D. Sankoff and J. B. Kruskal, eds, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, 1–44. Addison-Wesley.

Markov, A. A. 1913. Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7:153–162.

Markov, A. A. 2006. Classical text in translation: A. A. Markov, an example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. *Science in Context*, 19(4):591–600. Translated by David Link.

Needleman, S. B. and C. D. Wunsch. 1970. A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Rabiner, L. R. and B. H. Juang. 1993. *Fundamentals of Speech Recognition*. Prentice Hall.

Reichert, T. A., D. N. Cohen, and A. K. C. Wong. 1973. An application of information theory to genetic mutations and the matching of polypeptide sequences. *Journal of Theoretical Biology*, 42:245–261.

Sakoe, H. and S. Chiba. 1971. A dynamic programming approach to continuous speech recognition. *Proceedings of the Seventh International Congress on Acoustics*, volume 3. Akadémiai Kiadó.

Sankoff, D. 1972. Matching sequences under deletion-insertion constraints. *Proceedings of the National Academy of Sciences*, 69:4–6.

Vintsyuk, T. K. 1968. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57. Original Russian: Kibernetika 4(1):81-88. 1968.

Viterbi, A. J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269.

Wagner, R. A. and M. J. Fischer. 1974. The string-to-string correction problem. *Journal of the ACM*, 21:168–173.

NGram Language Model
Log of Probabilities
Laplace Smoothing
Perplexity

N-gram Model

An **n-gram** is a contiguous sequence of **n** items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The **n-grams** typically are collected from a text or speech corpus.

Conditional Probability: $P(B \mid A) = \frac{P(A,B)}{P(A)}$ $\qquad P(A,B) = P(A)P(B \mid A)$

More variables: $P(A,B,C,D) = P(A)P(B \mid A)P(C \mid A,B)P(D \mid A,B,C)$

Chain Rule:

$$P(x_1, x_2, \ldots x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2) \ldots\ldots P(x_n \mid x_1, \ldots\ldots, x_{n-1})$$

P("about five minutes from")= P(about) × P(five |about) × P(minutes | about five) × P(from | about five minutes)

Probability of words in sentences:

$$P(w_1, w_2, \ldots w_n) = \prod_i P(w_i \mid w_1, w_2, w_3 \ldots\ldots, w_{i-1})$$

Unigram(1-gram): **No history is used.**

Bi-gram(2-gram): **One word history**

Tri-gram(3-gram): **Two words history**

Four-gram(4-gram): **Three words history**

Five-gram(5-gram):**Four words history**

Generally in practical applications, Bi-gram(previous one word), Tri-gram(previous two word, Four-gram (previous three word) are used.

Unigram(1-gram): No history is used.

"about five minutes from....."

Assume in corpus dinner word is present with highest probability.

Unigram doesn't take into account probabilities with previous words like from, minutes.

Unigram will predict dinner.

"about five minutes from dinner"

Bi-gram(2-gram): **One word history**

$$P(w_1, w_2) = \prod_{i=2} P(w_2 \mid w_1) \qquad P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

"about five minutes from....."

Assumption: Next word may be college, class

$$P(college \mid about\ \ five\ \ minutes\ \ from) = \frac{count(about\ \ five\ \ minutes\ \ from\ \ college)}{count(about\ \ five\ \ minutes\ \ from)}$$

$$P(class \mid about\ \ five\ \ minutes\ \ from) = \frac{count(about\ \ five\ \ minutes\ \ from\ \ class)}{count(about\ \ five\ \ minutes\ \ from)}$$

"about five minutes from....."

Count(about five minutes from)= P(about|<S>) × P(five| about) × P(minutes | five)
× P(from| minutes)

Count(about five minutes from **college**)= P(about|<S>) × P(five| about) × P(minutes | five)
× P(from| minutes) × P(college| from)

Count(about five minutes from **class**)= P(about|<S>) × P(five| about) × P(minutes | five)
× P(from| minutes) × P(class| from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college| from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(class| from)

Tri-gram(2-gram): **Two words history**

$$P(w_1, w_2, w_3) = \prod_{i=3} P(w_3 \mid w_1, w_2) \qquad P(w_i \mid w_{i-1}, w_{i-2}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$$

Count(about five minutes from)= P(five|<S>, about) × P(minutes| about, five) × P(from| five , minutes)

Count(about five minutes from **college**)= P(five|<S>, about) × P(minutes| about, five) × P(from| five , minutes) × P(college| minutes from)

Count(about five minutes from **class**)= P(five|<S>, about) × P(minutes| about, five) × P(from| five , minutes) × P(class| minutes from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college| minutes from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(class| minutes from)

$$P(\text{college} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from college)}}{\text{count(about five minutes from)}}$$

=P(college| five minutes from)

$$P(\text{class} \mid \text{about five minutes from}) = \frac{\text{count(about five minutes from class)}}{\text{count(about five minutes from)}}$$

=P(college| five minutes from)

As no. of previous state (history) increases, it is very difficult to match that set of words in corpus.

Probabilities of **larger collection of word is minimum.** To overcome this problem, Bi-gram model is used

Exercise 1: Estimating Bi-gram probabilities

What is the most probable next word predicted by the model for the following word sequence?

Given Corpus

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1) <S> Do ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability W_{i-1}=do

$$\frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Next word	Probability Next Word=
P(</S> | do)	0/4
P(<I> | do)	**2/4**
P(<am>| do)	0/4
P(<Henry>| do)	1/4
P(<like | do)	1/4
P(<college | do)	0/4
P(do | do)	0/4

I is more probable

2) <S> I like Henry ?

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability **W_{i-1}=Henry**

Next word	Probability Next Word= $\frac{N}{D} = \frac{count(w_{i-1}.w_i)}{count(w_{i-1})}$
P(</S> | Henry)	**3/5**
P(<I> | Henry)	1/5
P(<am>| Henry)	0
P(<Henry>| Henry)	0
P(<like | Henry)	1/5
P(<college | Henry)	0
P(do | Henry)	0

</S> is more probable

3) <S> Do I like ?

Use Tri-gram

P<I like>=3

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Next word prediction probability

W_{i-2}=I and W_{i-1}=like

Next word	Probability Next Word= $\frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$
P(</S> |I like)	0/3
P(<I> | I like)	0/3
P(<am>| I like)	0/3
P(<Henry>| I like)	1/3
P(<like | I like)	0/3
P(<college | I like)	2/3
P(do | I like)	0/3

College is probable

4) <S> Do I like college ?

Use Four-gram

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Next word prediction probability

W_{i-3}=I, W_{i-2}=like W_{i-1}=college

Next word	Probability Next Word= $\frac{\text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-3}, w_{i-2}, w_{i-1})}$
P(</S> |I like college)	2/2
P(<I> | I like college)	0/2
P(<am>| I like college)	0/2
P(<Henry>| I like college)	0/2
P(<like | I like college)	0/2
P(<college | I like college)	0/2
P(do | I like college)	0/2

</S> is more probable

Which of the following sentence is better. i.e. Gets a higher probability with this model.
Use Bi-gram

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

1. <S> I like college </S>

<S> like college </S>=?

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

=3/7 × 3/6 × 3/5 ×3/3 = 9/70=0.13

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=3/7 × 2/4 × 3/6 ×2/5 ×3/5 = 9/350=0.0257

ANS: First statement is more probable

Which of the following sentence is better. i.e. Gets a higher probability with Bi-gram model.

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

First statement is more probable

1. <S> I like college </S>

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

=3/7 × 3/6 × 3/5 ×3/3 = 9/70=**0.13**

= log(3/7) +log(3/6) + log(3/5) + log(3/3)= **-2.0513**

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=3/7 × 2/4 × 3/6 ×2/5 ×3/5 = 9/350=**0.0257**

=log(3/7)+log(2/4) +log(3/6) + log(2/5) + log(3/5)= **-3.6607**

<S> I am Henry </S>

<S> I like college </S>

<S> Do Henry like college </S>

<S> Henry I am </S>

<S> Do I like Henry </S>

<S> Do I like college </S>

<S> I do like Henry </S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Second statement is more probable

1. <S> like college </S>

=P(like | <S>) × P(college| like) × P(</S> | college)

=0/7 × 3/5 ×3/3 =**0**

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=3/7 × 2/4 × 3/6 ×2/5 ×3/5 = 9/350=**0.0257**

Laplace Smoothing

Corpus

<S> I am Henry </S>

<S>I like college</S>

<S>Do Henry like college</S>

<S>Henry I am</S>

<S>Do I like Henry</S>

<S>Do I like college</S>

<S>I do like Henry</S>

Word	Frequency
<S>	7
</S>	7
I	6
am	2
Henry	5
like	5
college	3
Do	4

Features : <S>, </S>, I, am, Henry, like, college, Do

Total number of unique words : 8

But we exclude <S> as it is not used in bigram.

So, total unique word is 7.

Give the following bi-gram probabilities estimated by Laplace model.

Add one Smoothing

1. <S> like college </S>

=P(like | <S>) × P(college| like) × P(</S> | college)

=(0+1)/(7+7) × (3+1)/(5+7) ×(3+1)/(3+7)

=1/14 × 4/12 × 4/10

=0.0095

2. <S> Do I like Henry </S>

=P(do | <S>) × P(I | do) × P(like | I) × P(Henry | like) × P(</S> | Henry)

=(3+1)/(7+7) × (2+1)/(4+7) × (3+1)/(6+7) ×(2+1)/(5+7) ×(3+1)/(5+7)

= 4/14 × 3/11 × 4/13 × 3/12 × 4/12

=0.0020

We are given the following corpus, modified from the one in the chapter:

<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like green eggs and Sam </s>

Using a bigram language model with add-one smoothing, what is P(Sam | am)? Include <s> and </s> in your counts just like any other token.

Word	Frequency
I	4
am	3
do	1
not	1
like	1
green	1
eggs	1
Sam	4
and	1

$P(Sam|am) = \frac{2}{3}$ (Bigram Model)

$P(Sam|am) = \frac{3}{14}$ (Bigram Model with add-one smoothing)

Language Model Evaluation

- LM is better if it is assigning a high probability to the real, frequently observed and grammatical sentence over false, rarely observed and ungrammatical sentences.
- Two different criteria for evaluation
  - ✓Extrinsic
  - ✓Intrinsic
- Extrinsic Evaluation
- It evaluates the language model when solving a specific task.
- Ex: Speech recognition accuracy, Machine translation accuracy, spelling correction accuracy
- Compare 2 or more models and check which one is better.
- Disadvantages
  - ✓Expensive and Time consuming

Language Model Evaluation

- Intrinsic Evaluation:
- The language model is best when it predicts an unseen test set.
- Definition of Perplexity
- It is the inverse probability of the test data which is normalized by the number of words.
- Lower the value of perplexity : **Better model**
- More value of perplexity : **Confused for prediction**

Perplexity (Intrinsic Evaluation model)

- The language model is best when it predicts an unseen test set.

Definition of Perplexity:

- It is the inverse probability of the test data which is normalized by the number of words.

$$PP(w) = P(w_1,w_2,w_3,w_4,\ldots,\ldots,wN)^{\frac{1}{N}}$$

$$PP(w) = \left(\prod_i \frac{1}{P(w_i \mid w_1, w_2, \ldots\ldots, w_{i-1})}\right)^{\frac{1}{N}} \qquad PP(w) = \left(\prod_i \frac{1}{P(w_i \mid w_{i-1})}\right)^{\frac{1}{N}}$$

- Lower the value of perplexity : **Better model**
- More value of perplexity : **Confused for prediction**

Perplexity for Bigram <S> I like college </S>

=P(I| <S>) × P(like | I) × P(college| like) × P(</S> | college)

=3/7 × 3/6 × 3/5 ×3/3 = 9/70=**0.13**

PP(w)=(1/0.13)$^{1/4}$=1.67

Perplexity for Trigram <S> I like college </S>

P(w)=P(like| <S> I) × P(college | I like) × P(</S>| like college)

P(w) =1/3 × 2/3 × 3/3 = 2/9=**0.22**

PP(w)=(1/0.22)$^{1/3}$=1.66

Advantages:

- Easy to understands, implement
- Can be easily convert to any gram

Disadvantages:

- Underflow due to multiplication of probabilities
- **Solution:** Use log. Add probabilities.
- Zero probability problem
- **Solution:** Use Laplace smoothing

References:

Daniel Jurafsky, James H. Martin —Speech and Language Processing‖ Second Edition, Prentice Hall, 2008.

Christopher D.Manning and Hinrich Schutze, — Foundations of Statistical Natural Language Processing , MIT Press, 1999.

THANK YOU

How NLP does this task???

Answer is Text Vectorization or Word Embedding

- **Text vectorization is a process** to convert each text document into a numeric vector.
- One hot encoding
- Word embeding

Why do we need Word Embeddings?

- As we know that many Machine Learning algorithms and almost all Deep Learning Architectures are not capable of processing strings or plain text in their raw form.

- In a broad sense, they require numerical numbers as inputs to perform any sort of task, such as classification, regression, clustering, etc.

- Also, from the huge amount of data that is present in the text format, it is imperative to extract some knowledge out of it and build any useful applications.

Text Vectorization & Transformation

Text vectorization is two types.

- Frequency-based or Statistical based Word Embedding
- Prediction based Word Embedding

Methods used for text vectorization

- Binary Term Frequency.
- Bag of Words (BoW) Term Frequency.
- TF-IDF
- Word2Vec
- Glove embedding

One-Hot Encoding (OHE)

Sentence: I am teaching NLP in Python

A word in this sentence may be “NLP”, “Python”, “teaching”, etc.
Since a dictionary is defined as the list of all unique words present in the sentence.

So, a dictionary may look like –
Dictionary: [‘I’, ’am’, ’teaching’,’ NLP’,’ in’, ’Python’]
Therefore, the vector representation in this format according to the above dictionary is

Vector for NLP: [0,0,0,1,0,0]
Vector for Python: [0,0,0,0,0,1]

Disadvantages of One-hot Encoding

1. One of the disadvantages of One-hot encoding is that the Size of the vector is equal to the count of unique words in the vocabulary.

2. One-hot encoding does not capture the relationships between different words. Therefore, it does not convey information about the context.

Count Vectorizer

- It creates a document term matrix, which is a set of dummy variables that indicates if a particular word appears in the document.

- Count vectorizer will fit and learn the word vocabulary and try to create a document term matrix in which the individual cells denote the frequency of that word in a particular document, which is also known as term frequency, and the columns are dedicated to each word in the corpus.

- Matrix Formulation

✓Consider a Corpus C containing D documents $\{d_1, d_2 \ldots . d_D\}$ from which we extract N unique tokens.

✓Now, the dictionary consists of these N tokens, and the size of the Count Vector matrix M formed is given by D X N.

✓Each row in the matrix M describes the frequency of tokens present in the document D(i).

Count Vectorizer

Document-1: He is a smart boy. She is also smart.
Document-2: Chirag is a smart person.
The dictionary created contains the list of unique tokens(words) present in the corpus
Unique Words: [‘He’, ’She’, ’smart’, ’boy’, ’Chirag’, ’person’]
Here, D=2, N=6
So, the count matrix M of size 2 X 6 will be represented as –

	He	She	smart	boy	Chirag	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

Vector for ‘smart’ is [2,1],
Vector for ‘Chirag’ is [0, 1], and so on.

Bag-of-Words (BoW)

- This vectorization technique converts the text content to numerical feature vectors.

- Bag of Words takes a document from a corpus and converts it into a numeric vector by mapping each document word to a feature vector for the machine learning model.

- It follows two steps

✓Tokenization

✓Vectors Creation

Bag-of-Words (BoW)

- **Tokenization**

It is the process of dividing each sentence into words or smaller parts, which are known as tokens. After the completion of tokenization, we will extract all the unique words from the corpus. Here corpus represents the tokens we get from all the documents and used for the bag of words creation.

- **Create vectors for each Sentence**

Here the vector size for a particular document is equal to the number of unique words present in the corpus. For each document, we will fill each entry of a vector with the corresponding word frequency in a particular document.

Bag-of-Words (BoW)

This burger is very tasty and affordable
This burger is not tasty and is affordable
This burger is very very delicious

Corpus →

this burger is very tasty and affordable.
this burger is not tasty and is affordable.
this burger is very very delicious.

↓ Tokenization

Unique words: [“and”, “affordable.”, “delicious.”, “is”, “not”, “burger”, “tasty”, “this”, “very”]

←

[[1,1,0,1,0,1,1,1,1],
[1,1,0,2,2,2,2,2,0],
[0,0,1,1,0,1,0,1,2]]

	and	affordable	delicious	is	not	burger	tasty	this	very
D1	1	1	0	1	0	1	1	1	1
D2	1	1	0	2	1	1	1	1	0
D3	0	0	1	1	0	1	0	1	2

- corpus = ['the quick brown fox jumped over the brown dog.',
  'the quick brown fox.',
  'the brown brown dog.',
  'the fox ate the dog.']

	ate	brown	dog	fox	jumped	over	quick	the
Document 1	0	2	1	1	1	1	1	2
Document 2	0	1	0	1	0	0	1	1
Document 3	0	2	1	0	0	0	0	1
Document 4	1	0	1	1	0	0	0	2

```
[0, 2, 1, 1, 1, 1, 0, 2],
[0, 1, 0, 1, 0, 0, 1, 1],
[0, 2, 1, 0, 0, 0, 0, 1],
[1, 0, 1, 1, 0, 0, 0, 2]
```

Bag-of-Words (BoW)

Disadvantages

- This method doesn’t preserve the word order.
- It does not allow to draw of useful inferences for downstream NLP tasks.

TF-IDF Vectorization

- As we discussed in the above techniques that the BOW method is simple and works well, but the problem with that is that it treats all words equally. As a result, it cannot distinguish very common words or rare words. So, to solve this problem, TF-IDF comes into the picture!

- Term frequency-inverse document frequency (TF-IDF) gives a measure that takes the importance of a word into consideration depending on how frequently it occurs in a document and a corpus.

- Term frequency (TF) : the frequency of a word in a document
- Inverse document frequency (IDF) : It measures how common a particular word is across all the documents in the corpus.

- $\text{TF(term)}=\frac{Number\ of\ times\ term\ appear\ in\ a\ document}{total\ number\ of\ items\ in\ the\ document}$
- $\text{IDF (term)} = \log \frac{Total\ number\ of\ document}{Number\ of\ document\ with\ term\ in\ it}$
- TFIDF (term) = TF(term) * IDF(term)

TF – frequency of world(term) in the corpus.

Doc 1 . "I will **go** to Mumbai then go to Pune"

I	1
Will	1
Go	2
To	2
Mumbai	1
Then	1
Pune	1

NTF = total count for word/total words (normalize)

I	Will	Go	To	Mumbai	Then	Pune
1/7	1/7	2/7	2/7	1/7	1/7	1/7

1 . "I will go to Mumbai then go to Pune"
2. "Yesterday I went to Mumbai"

I	1/7
Will	1/7
Go	2/7
To	2/7
Mumbai	1/7
Then	1/7
Pune	1/7

Yesterday	1/5
I	1/5
Went	1/5
To	1/5
Mumbai	1/5

World frequency in all documents

I	2
Will	1
Go	2
To	3
Mumbai	2
Then	1
Pune	1
Yesterday	1
went	1

TF

word / document	I	Will	Go	To	Mumbai	Then	Pune	Yesterday	Went
TF1	0.14	0.14	0.28	0.28	0.14	0.14	0.14	0	0
TF2	0.25	0	0	0.25	0.25	0	0	0.25	0.25

1 . "I will go to Mumbai then go to Pune"
2. "Yesterday I went to Mumbai"

I	1/7
Will	1/7
Go	2/7
To	2/7
Mumbai	1/7
Then	1/7
Pune	1/7

Yesterday	1/5
I	1/5
Went	1/5
To	1/5
Mumbai	1/5

IDF – inverse document frequency

$$IDF = LOG\left(\frac{\text{No of documents}}{\text{(No of documents that contains a word)}}\right)$$

IDF is used to calculate weight of words.

Words	Word frequency in all documents	IDF	IDF
I	2	log(2/2)	0
Will	1	log(2/1)	0.69
Go	2	log(2/1)	0.69
To	2	log(2/2)	0
Mumbai	2	log(2/2)	0
Then	1	log(2/1)	0.69
Pune	1	log(2/1)	0.69
Yesterday	1	log(2/2)	0
Went	1	log(2/1)	0.69

TF-IDF

TF-IDF TF-IDF

words	Word frequency in all documents	TF1	TF2	IDF	IDF	IDF*TF1	IDF*TF2
I	2	0.14	0.25	Log(2/2)	0	0	0
Will	1	0.14	0	Log(2/1)	0.69	0.1	0
Go	2	0.28	0	Log(2/1)	0.69	0.19	0
To	3	0.28	0.25	Log(2/2)	0	0	0
Mumbai	2	0.14	0.25	Log(2/2)	0	0	0
Then	1	0.14	0	Log(2/1)	0.69	0.1	0
Pune	1	0.14	0	Log(2/1)	0.69	0.1	0
Yesterday	1	0	0.25	Log(2/1)	0.69	0	0.17
went	1	0	0.25	Log(2/1)	0.69	0	0.17

Thank You

Unit 4

Semantics

Requirements for Representation.

Expressiveness: Expressiveness refers to the ability of the representation system to capture a wide range of meanings, nuances, and contexts.

- Rich Vocabulary: The system should have a rich set of symbols to describe various entities, actions, properties, and relationships.
- Complex Structures: It should support the representation of complex structures such as nested and hierarchical relationships.
- Context Sensitivity: The representation should capture context to accurately reflect meaning (e.g., the difference between "bank" in "river bank" and "financial bank").
- Ambiguity Resolution: It should handle and resolve ambiguities in natural language.

Formalism: Formalism ensures that the representation has a well-defined syntax and semantics, allowing for unambiguous interpretation and processing.

- Clear Syntax: The rules for forming valid expressions should be precise and unambiguous.
- Defined Semantics: Each expression should have a clear meaning, defined in a way that machines can interpret consistently.
- Standardization: Use of standard representation languages (e.g., First-Order Logic, Description Logic) can help in maintaining consistency and interoperability.

Inference: Inference capability allows the system to derive new information from existing knowledge, which is essential for reasoning and decision-making.

- **Logical Deduction**: The system should support logical inference mechanisms such as modus ponens, modus tollens, and syllogism.
- **Probabilistic Inference**: For handling uncertainty, the system might need to support probabilistic reasoning (e.g., Bayesian networks).
- **Temporal Reasoning**: For applications involving time, the system should support reasoning about temporal relationships and events.

Propositional Logic

1. "If it is raining, then the ground is wet." - $P \rightarrow Q$
2. "Either the system is down, or the network is slow." - $P \vee Q$
3. "If the user is logged in and the session is active, then the user can access the dashboard." - $(P \wedge Q) \rightarrow R$
4. "The application will fail unless it is updated." - $\neg Q \rightarrow P$ or $P \vee Q$

5. "If the file is not found, an error message is displayed." - $\neg P \rightarrow Q$6. "The server is running if and only if the power is on." - $P \leftrightarrow Q$
7. "If the temperature is above 30°C, then the air conditioner turns on." - $P \rightarrow Q$
8. "If the database is backed up, the system will restart, and the logs will be cleared." - $P \rightarrow (Q \wedge R)$
9. "If the user presses the submit button, then the form is validated and the data is sent." - $P \rightarrow (Q \wedge R)$
10. "The document is either saved or discarded." - $P \vee Q$

First-Order Logic (FOL),

- **First-Order Logic (FOL)**, also known as Predicate Logic or First-Order Predicate Calculus, is a formal system used to express statements about objects, their properties, and their relationships. FOL extends propositional logic by incorporating quantifiers and predicates, making it much more expressive.

Key Components of First-Order Logic

Terms:

- Constants: Represent specific objects in the domain (e.g., a, b, Raju).
- Variables: Represent any object in the domain (e.g., x, y, z).
- Functions: Map a set of objects to another object
  - e.g., ‘fatherOf(Raju)’

Predicates: Represent properties or relationships between objects. For example,

- Student(Raju) means “Raju is a student," and Likes(Raju, Pizza) means "John likes pizza."

Logical Connectives:

- Conjunction (∧): P ∧ Q means "P and Q."
- Disjunction (∨): P ∨ Q means "P or Q."
- Negation (¬): ¬P means "not P."
- Implication (→): P → Q means "if P then Q."
- Biconditional (↔): P ↔ Q means "P if and only if Q."

Quantifiers:

- Universal Quantifier (∀): ∀x P(x) means "for all x, P(x) is true."
- Existential Quantifier (∃): ∃x P(x) means "there exists an x such that P(x) is true."

- "All humans are mortal."
- "Some students are brilliant." –
- "No dogs can fly.“
- "If a person is a parent, then they have a child." -
- "There is a cat that is black."
- "Every student loves some book." –
- "Someone likes everyone." -
- "If an animal is a bird, then it can fly." -
- "There exists a unique number that is even."
- "For every number, there is a greater number." -

1. "All humans are mortal." - $\forall x(\mathrm{Human}(x) \rightarrow \mathrm{Mortal}(x))$ x: human
2. "Some students are brilliant." - $\exists x(\mathrm{Student}(x) \wedge \mathrm{Brilliant}(x))$ x: student
3. "No dogs can fly." - $\forall x(\mathrm{Dog}(x) \rightarrow \neg \mathrm{CanFly}(x))$ x: dog
4. "If a person is a parent, then they have a child." - $\forall x(\mathrm{Parent}(x) \rightarrow \exists y(\mathrm{Child}(y, x)))$ x: parent, y: child
5. "There is a cat that is black." - $\exists x(\mathrm{Cat}(x) \wedge \mathrm{Black}(x))$ x: cat
6. "Every student loves some book." - $\forall x(\mathrm{Student}(x) \rightarrow \exists y(\mathrm{Book}(y) \wedge \mathrm{Loves}(x, y)))$ x: student, y: book
7. "Someone likes everyone." - $\exists x(\forall y(\mathrm{Person}(y) \rightarrow \mathrm{Likes}(x, y)))$ x: someone, y: person
8. "If an animal is a bird, then it can fly." - $\forall x(\mathrm{Bird}(x) \rightarrow \mathrm{CanFly}(x))$ x: bird

9. "There exists a unique number that is even." - $\exists x(\mathbf{Even}(x) \land \forall y(\mathbf{Even}(y) \rightarrow x = y))$ x: number, y: number
10. "For every number, there is a greater number." - $\forall x(\mathbf{Number}(x) \rightarrow \exists y(\mathbf{Number}(y) \land \mathbf{Greater}(y, x)))$ x: number, y: number

CMSC 471

First-Order Logic

Chapter 8.1-8.3

Adapted from slides by
Tim Finin and
Marie desJardins.

Some material adopted from notes
by Andreas Geyer-Schulz

Outline

- First-order logic
  - Properties, relations, functions, quantifiers, …
  - Terms, sentences, axioms, theories, proofs, …
- Extensions to first-order logic
- Logical agents
  - Reflex agents
  - Representing change: situation calculus, frame problem
  - Preferences on actions
  - Goal-based agents

First-order logic

- First-order logic (FOL) models the world in terms of
  - **Objects,** which are things with individual identities
  - **Properties** of objects that distinguish them from other objects
  - **Relations** that hold among sets of objects
  - **Functions,** which are a subset of relations where there is only one “value” for any given “input”
- Examples:
  - Objects: Students, lectures, companies, cars ...
  - Relations: Brother-of, bigger-than, outside, part-of, has-color, occurs-after, owns, visits, precedes, ...
  - Properties: blue, oval, even, large, ...
  - Functions: father-of, best-friend, second-half, one-more-than ...

User provides

- **Constant symbols,** which represent individuals in the world
  - Mary
  - 3
  - Green
- **Function symbols,** which map individuals to individuals
  - father-of(Mary) = John
  - color-of(Sky) = Blue
- **Predicate symbols,** which map individuals to truth values
  - greater(5,3)
  - green(Grass)
  - color(Grass, Green)

FOL Provides

- **Variable symbols**
  - E.g., x, y, foo
- **Connectives**
  - Same as in PL: not ($\neg$), and ($\wedge$), or ($\vee$), implies ($\rightarrow$), if and only if (biconditional $\leftrightarrow$)
- **Quantifiers**
  - Universal $\forall$**x** or **(Ax)**
  - Existential $\exists$**x** or **(Ex)**

Sentences are built from terms and atoms

- A **term** (denoting a real-world individual) is a constant symbol, a variable symbol, or an n-place function of n terms.
  x and $f(x_1, ..., x_n)$ are terms, where each x_i is a term.
  A term with no variables is a **ground term**
- An **atomic sentence** (which has value true or false) is an n-place predicate of n terms
- A **complex sentence** is formed from atomic sentences connected by the logical connectives:
  $\neg P$, $P \vee Q$, $P \wedge Q$, $P \rightarrow Q$, $P \leftrightarrow Q$ where P and Q are sentences
- A **quantified sentence** adds quantifiers $\forall$ and $\exists$
- A **well-formed formula (wff)** is a sentence containing no "free" variables. That is, all variables are "bound" by universal or existential quantifiers.
  $(\forall x)P(x,y)$ has x bound as a universally quantified variable, but y is free.

Quantifiers

- **Universal quantification**
  - $(\forall x)P(x)$ means that P holds for **all** values of x in the domain associated with that variable
  - E.g., $(\forall x)$ dolphin(x) $\rightarrow$ mammal(x)
- **Existential quantification**
  - $(\exists x)P(x)$ means that P holds for **some** value of x in the domain associated with that variable
  - E.g., $(\exists x)$ mammal(x) $\wedge$ lays-eggs(x)
  - Permits one to make a statement about some object without naming it

Quantifiers

- Universal quantifiers are often used with “implies” to form “rules”:
  $(\forall x)$ student(x) $\rightarrow$ smart(x) means “All students are smart”
- Universal quantification is *rarely* used to make blanket statements about every individual in the world:
  $(\forall x)$student(x)$\wedge$smart(x) means “Everyone in the world is a student and is smart”
- Existential quantifiers are usually used with “and” to specify a list of properties about an individual:
  $(\exists x)$ student(x) $\wedge$ smart(x) means “There is a student who is smart”
- A common mistake is to represent this English sentence as the FOL sentence:
  $(\exists x)$ student(x) $\rightarrow$ smart(x)
  - But what happens when there is a person who is *not* a student?

Quantifier Scope

- Switching the order of universal quantifiers *does not* change the meaning:
  - $(\forall x)(\forall y)P(x,y) \leftrightarrow (\forall y)(\forall x)\ P(x,y)$
- Similarly, you can switch the order of existential quantifiers:
  - $(\exists x)(\exists y)P(x,y) \leftrightarrow (\exists y)(\exists x)\ P(x,y)$
- Switching the order of universals and existentials *does* change meaning:
  - Everyone likes someone: $(\forall x)(\exists y)$ likes(x,y)
  - Someone is liked by everyone: $(\exists y)(\forall x)$ likes(x,y)

Connections between All and Exists

We can relate sentences involving $\forall$ and $\exists$ using De Morgan's laws:

$(\forall x)\ \neg P(x) \leftrightarrow \neg(\exists x)\ P(x)$

$\neg(\forall x)\ P \leftrightarrow (\exists x)\ \neg P(x)$

$(\forall x)\ P(x) \leftrightarrow \neg\ (\exists x)\ \neg P(x)$

$(\exists x)\ P(x) \leftrightarrow \neg(\forall x)\ \neg P(x)$

Quantified inference rules

1. Universal instantiation: If a property is true for all elements in a domain, it is true for any specific element in that domain.
    1. $\forall x\ P(x) \therefore P(A)$
2. Universal generalization: If a property is true for an arbitrary element in a domain, then it is true for all elements in the domain.
    1. $P(A) \wedge P(B) \ldots \therefore \forall x\ P(x)$
3. Existential instantiation: If there exists an element in a domain for which a property is true, we can infer that the property is true for some specific element.
    1. $\exists x\ P(x) \therefore P(F)$
4. Existential generalization: If a property is true for some specific element in a domain, then there exists an element in the domain for which the property is true.
    1. $P(A) \therefore \exists x\ P(x)$

1. Statement: "All dogs bark."

Inference: "Rover is a dog, so Rover barks.“

2. **Observation**: "If any student studies hard, they pass the exam."

Generalization: "Therefore, all students who study hard pass the exam.“

3. **Statement**: "There is someone in the office who can help you."

Inference: "Let's call this person Alex. Alex can help you.“

4. **Observation**: "Maria can solve the puzzle."

Generalization: "Therefore, there exists someone who can solve the puzzle."

Universal instantiation (a.k.a. universal elimination)

- If $(\forall x)$ P(x) is true, then P(C) is true, where C is *any* constant in the domain of x
- Example:

  $(\forall x)$ eats(Ziggy, x) $\Rightarrow$ eats(Ziggy, IceCream)
- The variable symbol can be replaced by any ground term, i.e., any constant symbol or function symbol applied to ground terms only

Existential instantiation (a.k.a. existential elimination)

- From $(\exists x)\ P(x)$ infer $P(c)$
- Example:
  - $(\exists x)$ eats(Ziggy, x) $\rightarrow$ eats(Ziggy, Stuff)
- Note that the variable is replaced by a **brand-new constant** not occurring in this or any other sentence in the KB
- Also known as skolemization; constant is a **skolem constant**
- In other words, we don't want to accidentally draw other inferences about it by introducing the constant
- Convenient to use this to reason about the unknown object, rather than constantly manipulating the existential quantifier

Existential generalization (a.k.a. existential introduction)

- If P(c) is true, then $(\exists x)\ P(x)$ is inferred.
- Example

  $\text{eats(Ziggy, IceCream)} \Rightarrow (\exists x)\ \text{eats(Ziggy, x)}$
- All instances of the given constant symbol are replaced by the new variable symbol
- Note that the variable symbol cannot already exist anywhere in the expression

Translating English to FOL

Every gardener likes the sun.

$\forall x$ gardener(x) $\rightarrow$ likes(x,Sun)

You can fool some of the people all of the time.

$\exists x \; \forall t$ person(x) $\wedge$time(t) $\rightarrow$ can-fool(x,t)

You can fool all of the people some of the time.

$\forall x \; \exists t$ (person(x) $\rightarrow$ time(t) $\wedge$can-fool(x,t))

$\forall x$ (person(x) $\rightarrow \exists t$ (time(t) $\wedge$can-fool(x,t))

Equivalent

All purple mushrooms are poisonous.

$\forall x$ (mushroom(x) $\wedge$ purple(x)) $\rightarrow$ poisonous(x)

No purple mushroom is poisonous.

$\neg\exists x$ purple(x) $\wedge$ mushroom(x) $\wedge$ poisonous(x)

$\forall x$ (mushroom(x) $\wedge$ purple(x)) $\rightarrow \neg$poisonous(x)

Equivalent

There are exactly two purple mushrooms.

$\exists x \; \exists y$ mushroom(x) $\wedge$ purple(x) $\wedge$ mushroom(y) $\wedge$ purple(y) ^ $\neg$(x=y) $\wedge \; \forall z$ (mushroom(z) $\wedge$ purple(z)) $\rightarrow$ ((x=z) $\vee$ (y=z))

Clinton is not tall.

$\neg$tall(Clinton)

X is above Y iff X is on directly on top of Y or there is a pile of one or more other objects directly on top of one another starting with X and ending with Y.

$\forall x \; \forall y$ above(x,y) $\leftrightarrow$ (on(x,y) $\vee \; \exists z$ (on(x,z) $\wedge$ above(z,y)))

Example: A simple genealogy KB by FOL

- **Build a small genealogy knowledge base using FOL that**
  - contains facts of immediate family relations (spouses, parents, etc.)
  - contains definitions of more complex relations (ancestors, relatives)
  - is able to answer queries about relationships between people
- **Predicates:**
  - parent(x, y), child(x, y), father(x, y), daughter(x, y), etc.
  - spouse(x, y), husband(x, y), wife(x,y)
  - ancestor(x, y), descendant(x, y)
  - male(x), female(y)
  - relative(x, y)
- **Facts:**
  - husband(Joe, Mary), son(Fred, Joe)
  - spouse(John, Nancy), male(John), son(Mark, Nancy)
  - father(Jack, Nancy), daughter(Linda, Jack)
  - daughter(Liz, Linda)
  - etc.

- **Rules for genealogical relations**
  - $(\forall x,y)$ parent(x, y) $\leftrightarrow$ child (y, x)
    $(\forall x,y)$ father(x, y) $\leftrightarrow$ parent(x, y) $\wedge$ male(x) (similarly for mother(x, y))
    $(\forall x,y)$ daughter(x, y) $\leftrightarrow$ child(x, y) $\wedge$ female(x) (similarly for son(x, y))
  - $(\forall x,y)$ husband(x, y) $\leftrightarrow$ spouse(x, y) $\wedge$ male(x) (similarly for wife(x, y))
    $(\forall x,y)$ spouse(x, y) $\leftrightarrow$ spouse(y, x) (**spouse relation is symmetric**)
  - $(\forall x,y)$ parent(x, y) $\rightarrow$ ancestor(x, y)
    $(\forall x,y)(\exists z)$ parent(x, z) $\wedge$ ancestor(z, y) $\rightarrow$ ancestor(x, y)
  - $(\forall x,y)$ descendant(x, y) $\leftrightarrow$ ancestor(y, x)
  - $(\forall x,y)(\exists z)$ ancestor(z, x) $\wedge$ ancestor(z, y) $\rightarrow$ relative(x, y)
    (related by common ancestry)
    $(\forall x,y)$ spouse(x, y) $\rightarrow$ relative(x, y) (related by marriage)
    $(\forall x,y)(\exists z)$ relative(z, x) $\wedge$ relative(z, y) $\rightarrow$ relative(x, y) (**transitive**)
    $(\forall x,y)$ relative(x, y) $\leftrightarrow$ relative(y, x) **(symmetric**)
- **Queries**
  - ancestor(Jack, Fred) /* the answer is yes */
  - relative(Liz, Joe) /* the answer is yes */
  - relative(Nancy, Matthew)
    /* no answer in general, no if under closed world assumption */
  - $(\exists z)$ ancestor(z, Fred) $\wedge$ ancestor(z, Liz)

Semantics of FOL

- **Domain M**: the set of all objects in the world (of interest)
- **Interpretation I**: includes
  - Assign each constant to an object in M
  - Define each function of n arguments as a mapping M^n => M
  - Define each predicate of n arguments as a mapping M^n => {T, F}
  - Therefore, every ground predicate with any instantiation will have a truth value
  - In general there is an infinite number of interpretations because |M| is infinite
- **Define logical connectives**: **~, ^, ∨, =>, <=>** as in PL
- **Define semantics of (∀x) and (∃x)**
  - (∀x) P(x) is true iff P(x) is true under all interpretations
  - (∃x) P(x) is true iff P(x) is true under some interpretation

- **Model**: an interpretation of a set of sentences such that every sentence is *True*
- **A sentence is**
  - **satisfiable** if it is true under some interpretation
  - **valid** if it is true under all possible interpretations
  - **inconsistent** if there does not exist any interpretation under which the sentence is true
- **Logical consequence**: S |= X if all models of S are also models of X

Axioms, definitions and theorems

- **Axioms** are facts and rules that attempt to capture all of the (important) facts and concepts about a domain; axioms can be used to prove **theorems**
  - Mathematicians don't want any unnecessary (dependent) axioms –ones that can be derived from other axioms
  - Dependent axioms can make reasoning faster, however
  - Choosing a good set of axioms for a domain is a kind of design problem
- A **definition** of a predicate is of the form "p(X) $\leftrightarrow$ …" and can be decomposed into two parts
  - **Necessary** description: "p(x) $\rightarrow$ …"
  - **Sufficient** description "p(x) $\leftarrow$ …"
  - Some concepts don't have complete definitions (e.g., person(x))

More on definitions

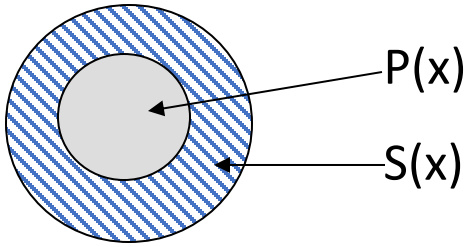
- A **necessary** condition must be satisfied for a statement to be true.
- A **sufficient** condition, if satisfied, assures the statement's truth.
- Duality: "P is sufficient for Q" is the same as "Q is necessary for P."
- Examples: define father(x, y) by parent(x, y) and male(x)
  - parent(x, y) is a necessary (**but not sufficient**) description of
    father(x, y)
    - father(x, y) $\rightarrow$ parent(x, y)
  - parent(x, y) ^ male(x) ^ age(x, 35) is a **sufficient** (**but not necessary**) description of father(x, y):

    father(x, y) $\leftarrow$ parent(x, y) ^ male(x) ^ age(x, 35)
  - parent(x, y) ^ male(x) is a **necessary and sufficient** description of father(x, y)

    parent(x, y) ^ male(x) $\leftrightarrow$ father(x, y)

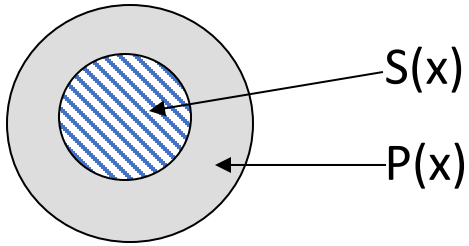
More on definitions

S(x) is a necessary condition of P(x)



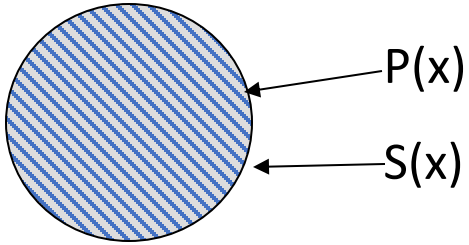
$(\forall x)\ P(x) \Rightarrow S(x)$

S(x) is a sufficient condition of P(x)



$(\forall x)\ P(x) \Leftarrow S(x)$

S(x) is a necessary and sufficient condition of P(x)



$(\forall x)\ P(x) \Leftrightarrow S(x)$

Higher-order logic

- FOL only allows to quantify over variables, and variables can only range over objects.
- HOL allows us to quantify over relations
- Example: (quantify over functions)

  "two functions are equal iff they produce the same value for all arguments"

  $\forall f\ \forall g\ (f = g) \leftrightarrow (\forall x\ f(x) = g(x))$
- Example: (quantify over predicates)

  $\forall r$ transitive(r) $\rightarrow (\forall xyz)\ r(x,y) \wedge r(y,z) \rightarrow r(x,z))$
- More expressive, but **undecidable**. (there isn't an effective algorithm to decide whether all sentences are valid)
  - First-order logic is decidable only when it uses predicates with only one argument.

Expressing uniqueness

- Sometimes we want to say that there is a single, unique object that satisfies a certain condition
- "There exists a unique x such that king(x) is true"
  - $\exists x\ king(x) \wedge \forall y\ (king(y) \rightarrow x=y)$
  - $\exists x\ king(x) \wedge \neg\exists y\ (king(y) \wedge x\neq y)$
  - $\exists!\ x\ king(x)$
- "Every country has exactly one ruler"
  - $\forall c\ country(c) \rightarrow \exists!\ r\ ruler(c,r)$
- Iota operator: "ι x P(x)" means "the unique x such that p(x) is true"
  - "The unique ruler of Freedonia is dead"
  - $dead(\iota x\ ruler(freedonia,x))$

Notational differences

- **Different symbols** for *and, or, not, implies, …*
  - ∀ ∃ ⇒ ⇔ ∧ ∨ ¬ • ⊃
  - p v (q ^ r)
  - p + (q * r)
  - etc
- **Prolog**

  cat(X) :- furry(X), meows (X), has(X, claws)
- **Lispy notations**

```
(forall ?x (implies (and (furry ?x)
                         (meows ?x)
                         (has ?x claws))
                    (cat ?x)))
```

Logical agents for the Wumpus World

Three (non-exclusive) agent architectures:

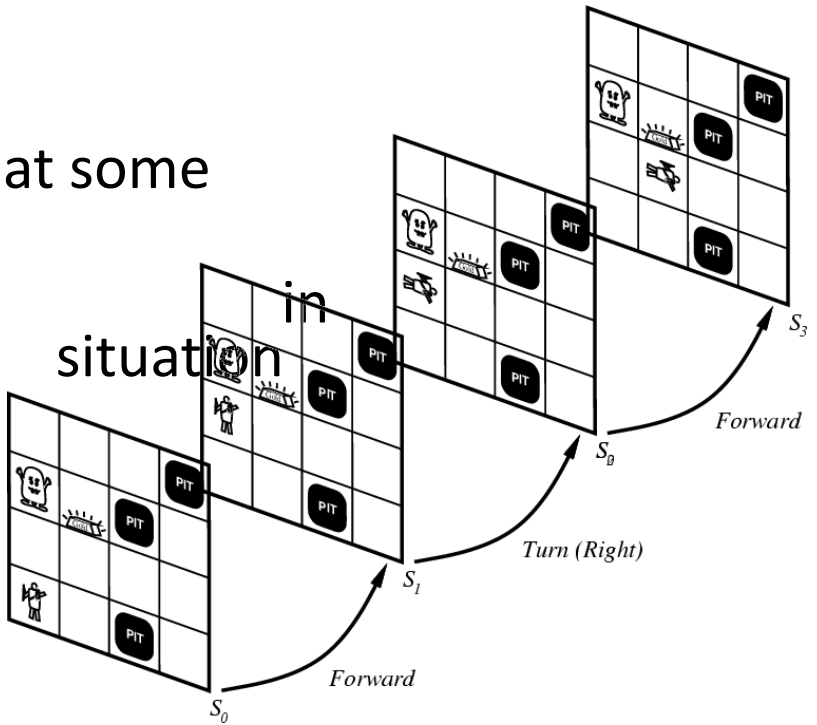
- Reflex agents
  - Have rules that classify situations, specifying how to react to each possible situation
- Model-based agents
  - Construct an internal model of their world
- Goal-based agents
  - Form goals and try to achieve them

A simple reflex agent

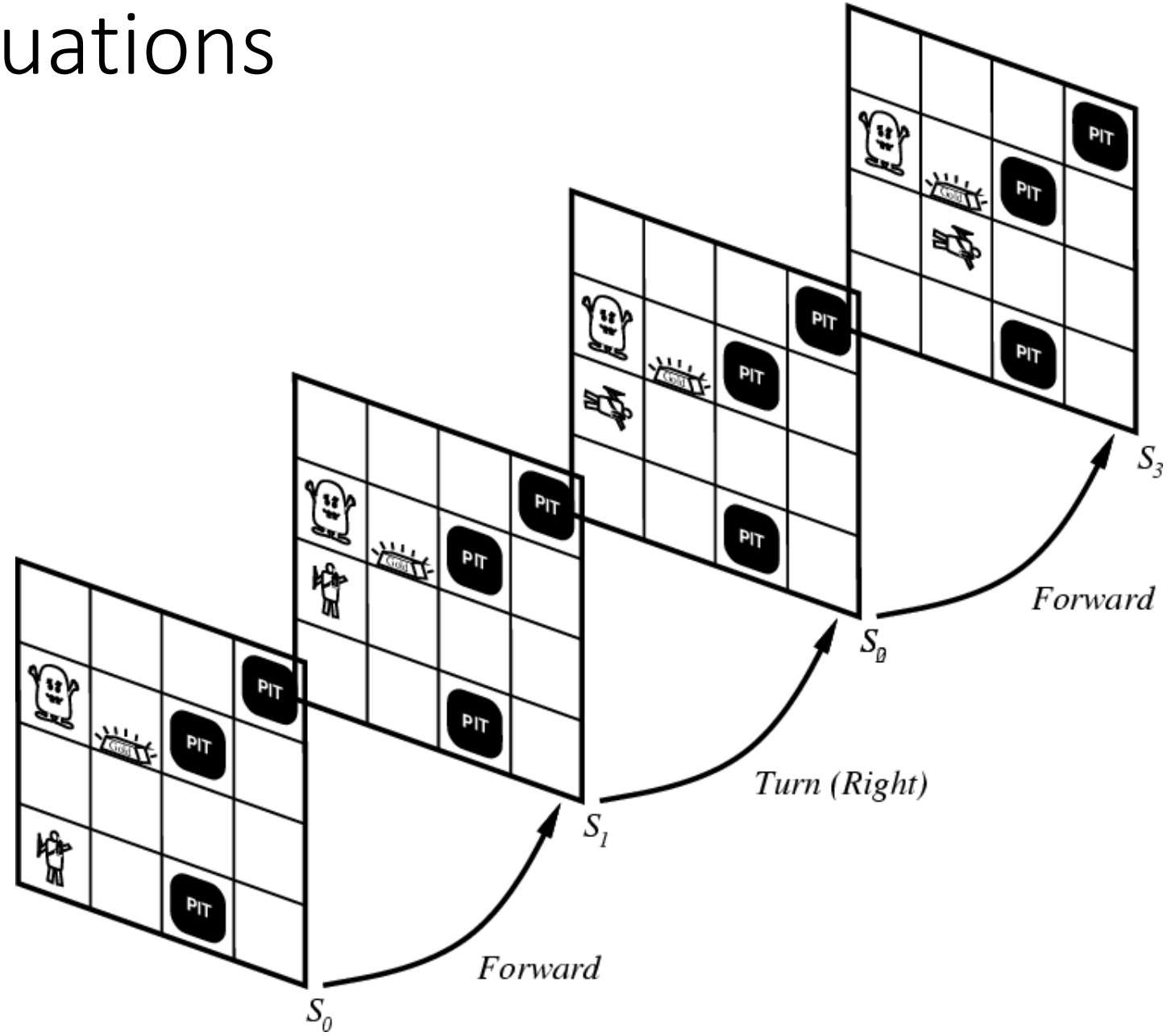
- Rules to map percepts into observations:
  $\forall b,g,u,c,t$ Percept([Stench, b, g, u, c], t) $\rightarrow$ Stench(t)
  $\forall s,g,u,c,t$ Percept([s, Breeze, g, u, c], t) $\rightarrow$ Breeze(t)
  $\forall s,b,u,c,t$ Percept([s, b, Glitter, u, c], t) $\rightarrow$ AtGold(t)
- Rules to select an action given observations:
  $\forall t$ AtGold(t) $\rightarrow$ Action(Grab, t);
- Some difficulties:
  - Consider Climb. There is no percept that indicates the agent should climb out – position and holding gold are not part of the percept sequence
  - Loops – the percept will be repeated when you return to a square, which should cause the same response (unless we maintain some internal model of the world)

Representing change

- Representing change in the world in logic can be tricky.
- One way is just to change the KB
  - Add and delete sentences from the KB to reflect changes
  - How do we remember the past, or reason about changes?
- **Situation calculus** is another way
- A **situation** is a snapshot of the world at some instant in time
- When the agent performs an action A in situation S1, the result is a new situation S2.



Situations


PIT
PIT
PIT
PIT
PIT
PIT
PIT
PIT
PIT
S_0
S_1
S_2
S_3
Forward
Turn (Right)
Forward

Situation calculus

- A **situation** is a snapshot of the world at an interval of time during which nothing changes
- Every true or false statement is made with respect to a particular situation.
  - Add **situation variables** to every predicate.
  - at(Agent,1,1) becomes at(Agent,1,1,s0): at(Agent,1,1) is true in situation (i.e., state) s0.
  - Alternatively, add a special 2nd-order predicate, **holds(f,s),** that means "f is true in situation s." E.g., holds(at(Agent,1,1),s0)
- Add a new function, **result(a,s),** that maps a situation s into a new situation as a result of performing action a. For example, result(forward, s) is a function that returns the successor state (situation) to s
- Example: The action agent-walks-to-location-y could be represented by
  - $(\forall x)(\forall y)(\forall s)\ (at(Agent,x,s) \wedge \neg onbox(s)) \rightarrow at(Agent,y,result(walk(y),s))$

Deducing hidden properties

- From the perceptual information we obtain in situations, we can **infer properties of locations**

  $\forall l,s\ at(Agent,l,s) \wedge Breeze(s) \rightarrow Breezy(l)$

  $\forall l,s\ at(Agent,l,s) \wedge Stench(s) \rightarrow Smelly(l)$

- Neither Breezy nor Smelly need situation arguments because pits and Wumpuses do not move around

Deducing hidden properties II

- We need to write some rules that relate various aspects of a single world state (as opposed to across states)
- There are two main kinds of such rules:
  - **Causal rules** reflect the assumed direction of causality in the world:

    $(\forall l1,l2,s)\ At(Wumpus,l1,s) \wedge Adjacent(l1,l2) \rightarrow Smelly(l2)$

    $(\forall\ l1,l2,s)\ At(Pit,l1,s) \wedge Adjacent(l1,l2) \rightarrow Breezy(l2)$

    Systems that reason with causal rules are called **model-based reasoning systems**
  - **Diagnostic rules** infer the presence of **hidden properties** directly from the percept-derived information. We have already seen two diagnostic rules:

    $(\forall\ l,s)\ At(Agent,l,s) \wedge Breeze(s) \rightarrow Breezy(l)$

    $(\forall\ l,s)\ At(Agent,l,s) \wedge Stench(s) \rightarrow Smelly(l)$

Representing change: The frame problem

- **Frame axioms**: If property x doesn't change as a result of applying action a in state s, then it stays the same.
  - On (x, z, s) $\wedge$ Clear (x, s) $\rightarrow$
    On (x, table, Result(Move(x, table), s)) $\wedge$
    $\neg$On(x, z, Result (Move (x, table), s))
  - On (y, z, s) $\wedge$ y$\neq$ x $\rightarrow$ On (y, z, Result (Move (x, table), s))
  - The proliferation of frame axioms becomes very cumbersome in complex domains

The frame problem II

- **Successor-state axiom**: General statement that characterizes every way in which a particular predicate can become true:
  - Either it can be **made true**, or it can **already be true and not be changed**:
  - On (x, table, Result(a,s)) $\leftrightarrow$
    [On (x, z, s) $\wedge$ Clear (x, s) $\wedge$ a = Move(x, table)] $\wedge$
    [On (x, table, s) $\wedge$ a $\neq$ Move (x, z)]
- In complex worlds, where you want to reason about longer chains of action, even these types of axioms are too cumbersome
  - Planning systems use special-purpose inference methods to reason about the expected state of the world at any point in time during a multi-step plan

Qualification problem

- Qualification problem:
  - How can you possibly characterize every single effect of an action, or every single exception that might occur?
  - When I put my bread into the toaster, and push the button, it will become toasted after two minutes, unless…
    - The toaster is broken, or…
    - The power is out, or…
    - I blow a fuse, or…
    - A neutron bomb explodes nearby and fries all electrical components, or…
    - A meteor strikes the earth, and the world we know it ceases to exist, or…

Ramification problem

- Similarly, it's just about impossible to characterize every side effect of every action, at every possible level of detail:
  - When I put my bread into the toaster, and push the button, the bread will become toasted after two minutes, and…
    - The crumbs that fall off the bread onto the bottom of the toaster over tray will also become toasted, and…
    - Some of the aforementioned crumbs will become burnt, and…
    - The outside molecules of the bread will become "toasted," and…
    - The inside molecules of the bread will remain more "breadlike," and…
    - The toasting process will release a small amount of humidity into the air because of evaporation, and…
    - The heating elements will become a tiny fraction more likely to burn out the next time I use the toaster, and…
    - The electricity meter in the house will move up slightly, and…

Knowledge engineering!

- Modeling the “right” conditions and the “right” effects at the “right” level of abstraction is very difficult
- Knowledge engineering (creating and maintaining knowledge bases for intelligent reasoning) is an entire field of investigation
- Many researchers hope that automated knowledge acquisition and machine learning tools can fill the gap:
  - Our intelligent systems should be able to **learn** about the conditions and effects, just like we do!
  - Our intelligent systems should be able to learn when to pay attention to, or reason about, certain aspects of processes, depending on the context!

Preferences among actions

- A problem with the Wumpus world knowledge base that we have built so far is that it is difficult to decide which action is best among a number of possibilities.
- For example, to decide between a forward and a grab, axioms describing when it is OK to move to a square would have to mention glitter.
- This is not modular!
- We can solve this problem by **separating facts about actions from facts about goals**. This way our **agent can be reprogrammed just by asking it to achieve different goals**.

Preferences among actions

- The first step is to describe the desirability of actions independent of each other.
- In doing this we will use a simple scale: actions can be Great, Good, Medium, Risky, or Deadly.
- Obviously, the agent should always do the best action it can find:

  $(\forall a,s)\ \text{Great}(a,s) \rightarrow \text{Action}(a,s)$

  $(\forall a,s)\ \text{Good}(a,s) \wedge \neg(\exists b)\ \text{Great}(b,s) \rightarrow \text{Action}(a,s)$

  $(\forall a,s)\ \text{Medium}(a,s) \wedge (\neg(\exists b)\ \text{Great}(b,s) \vee \text{Good}(b,s)) \rightarrow \text{Action}(a,s)$

  ...

Preferences among actions

- We use this action quality scale in the following way.
- Until it finds the gold, the basic strategy for our agent is:
  - Great actions include picking up the gold when found and climbing out of the cave with the gold.
  - Good actions include moving to a square that’s OK and hasn't been visited yet.
  - Medium actions include moving to a square that is OK and has already been visited.
  - Risky actions include moving to a square that is not known to be deadly or OK.
  - Deadly actions are moving into a square that is known to have a pit or a Wumpus.

Goal-based agents

- Once the gold is found, it is necessary to change strategies. So now we need a new set of action values.
- We could encode this as a rule:
  - $(\forall s)$ Holding(Gold,s) $\rightarrow$ GoalLocation([1,1]),s)
- We must now decide how the agent will work out a sequence of actions to accomplish the goal.
- Three possible approaches are:
  - **Inference**: good versus wasteful solutions
  - **Search**: make a problem with operators and set of states
  - **Planning**: to be discussed later

Unit 4- Continue

Word Senses and relations

- **Word Senses** refer to the different meanings a word can have in various contexts.
- Each distinct meaning of a word is called a "sense."
- The study of word senses is critical in natural language processing (NLP) for understanding and generating human language accurately.
- **Relationships Between Senses** are the semantic connections that exist between different senses of words.
- WordNet, a lexical database of English, extensively documents these relationships.

Types of Word Senses

- **Synonymy**:
  - Words with the **same** or nearly the same meaning.
  - Example: {car, automobile}
- **Antonymy**:
  - Words with **opposite** meanings.
  - Example: {hot, cold}
- **Hyponymy**:
  - A specific word whose meaning is included in a more **general** word.
  - Example: {dog} (hyponym of {animal})
- **Hypernymy**:
  - A general word that includes the meanings of more **specific** words.
  - Example: {animal} (hypernym of {dog})

- **Meronymy**:
  - A word that denotes a **part** of a larger whole.
  - Example: {wheel} (meronym of {car})
- **Holonymy**:
  - A word that denotes a **whole** of which a part is mentioned.
  - Example: {car} (holonym of {wheel})
- **Troponymy**:
  - A verb that denotes a specific **manner** of performing another verb.
  - Example: {to jog} (troponym of {to run})
- **Entailment**:
  - A verb that **implies** the action of another verb.
  - Example: {to snore} (entails {to sleep})
- **Coordinate Terms**:
  - Words that share a common hypernym and are at the same level of specificity.
  - Example: {car, bus, bicycle} (coordinate terms under {vehicle})

Word Sense Disambiguation (WSD)

- Word Sense Disambiguation (WSD) is the process of determining which sense of a word is used in a given context. This is essential for accurate language understanding in applications such as machine translation, information retrieval, and more.

Approaches to WSD:

1. Supervised Methods:

1. Use labeled data to train machine learning models to predict the correct word sense based on context.
2. Techniques: Support Vector Machines (SVM), Neural Networks.

2. Unsupervised Methods:

1. Cluster contexts of word usage to identify different senses without labeled data.
2. Techniques: Clustering algorithms, distributional similarity.

3. Knowledge-Based Methods:

1. Leverage lexical resources like WordNet to determine the correct sense.
2. Techniques: Lesk algorithm, similarity measures based on definitions.

4. Contextualized Embeddings:

1. Use deep learning models (e.g., BERT) to generate context-aware word embeddings that capture different senses.
2. Example: BERT can distinguish between "bank" in "river bank" and "financial bank" based on context.

Disambiguation Process:

1. Identify the context words surrounding the target word.
2. Compare the context with the definitions (glosses) of each sense in WordNet.
3. Choose the sense with the highest overlap or most relevant meaning based on the context.

The Simplified Lesk algorithm

- Let's disambiguate "**bank**" in this sentence:

  The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

- given the following two WordNet senses:

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

The Simplified Lesk algorithm

Choose sense with most word overlap between gloss and context
(not counting function words)

The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

Drawback

- Glosses and examples migh be too short and may not provide enough chance to overlap with the context of the word to be disambiguated.

The Corpus(--based) Lesk algorithm

- Assumes we have some sense--labeled data (like SemCor)
- Take all the sentences with the relevant word sense:

  These short, "streamlined" meetings usually are sponsored by local ***banks***[1]*, Chambers of Commerce, trade associations, or other civic organizations.*

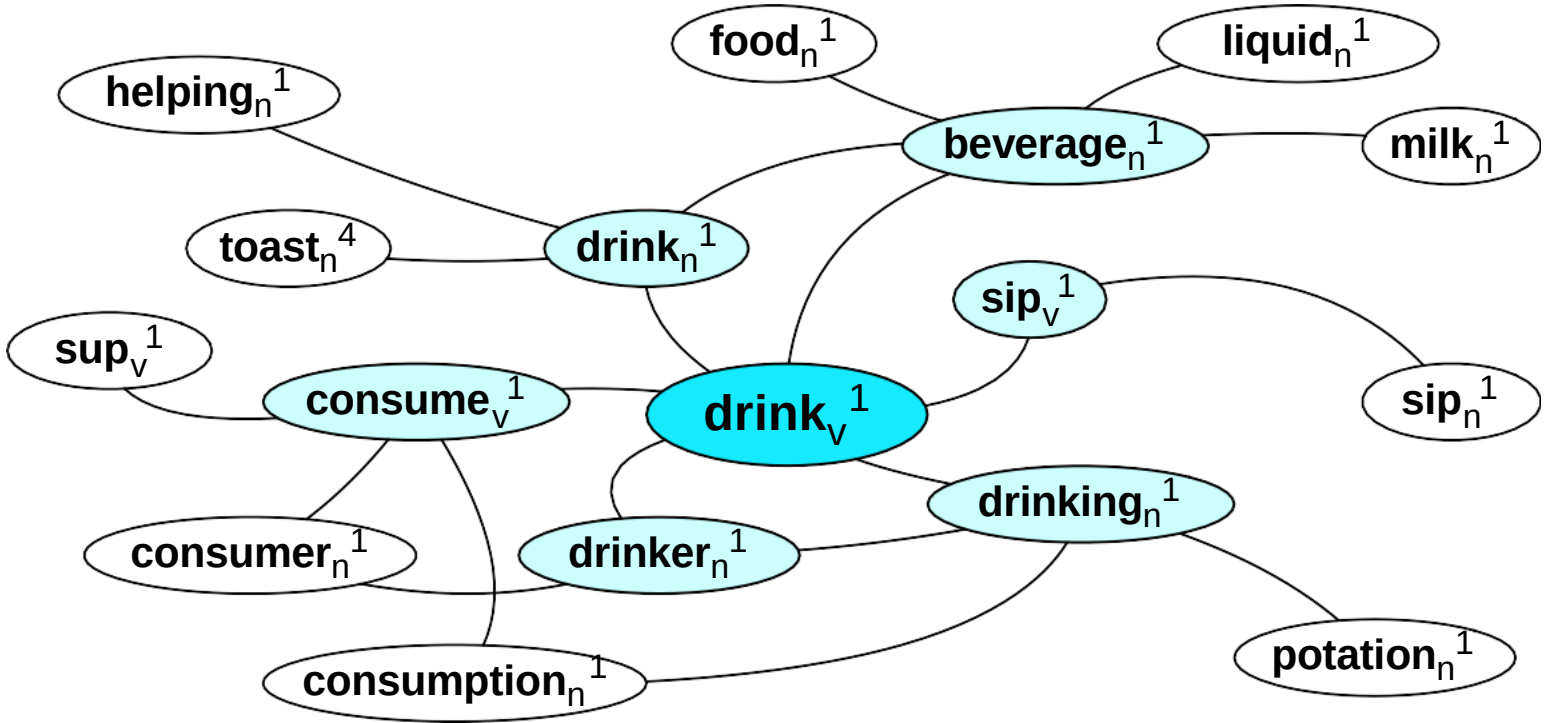
- Now add these to the gloss + examples for each sense, call it the "signature" of a sense. Basically, it is an expansion of the dictionary entry.
- Choose sense with most word overlap between context and signature (ie. the context words provided by the resources).

Corpus Lesk: IDF weighting

- Instead of just removing function words
  - Weigh each word by its `promiscuity' across documents
  - Down--weights words that occur in every `document' (gloss, example, etc)
  - These are generally function words, but is a more fine--grained measure
- Weigh each overlapping word by **inverse document frequency (IDF)**.

Graph based methods

- First, WordNet can be viewed as a graph
  - senses are nodes
  - relations (hypernymy, meronymy) are edges
  - Also add edge between word and unambiguous gloss words



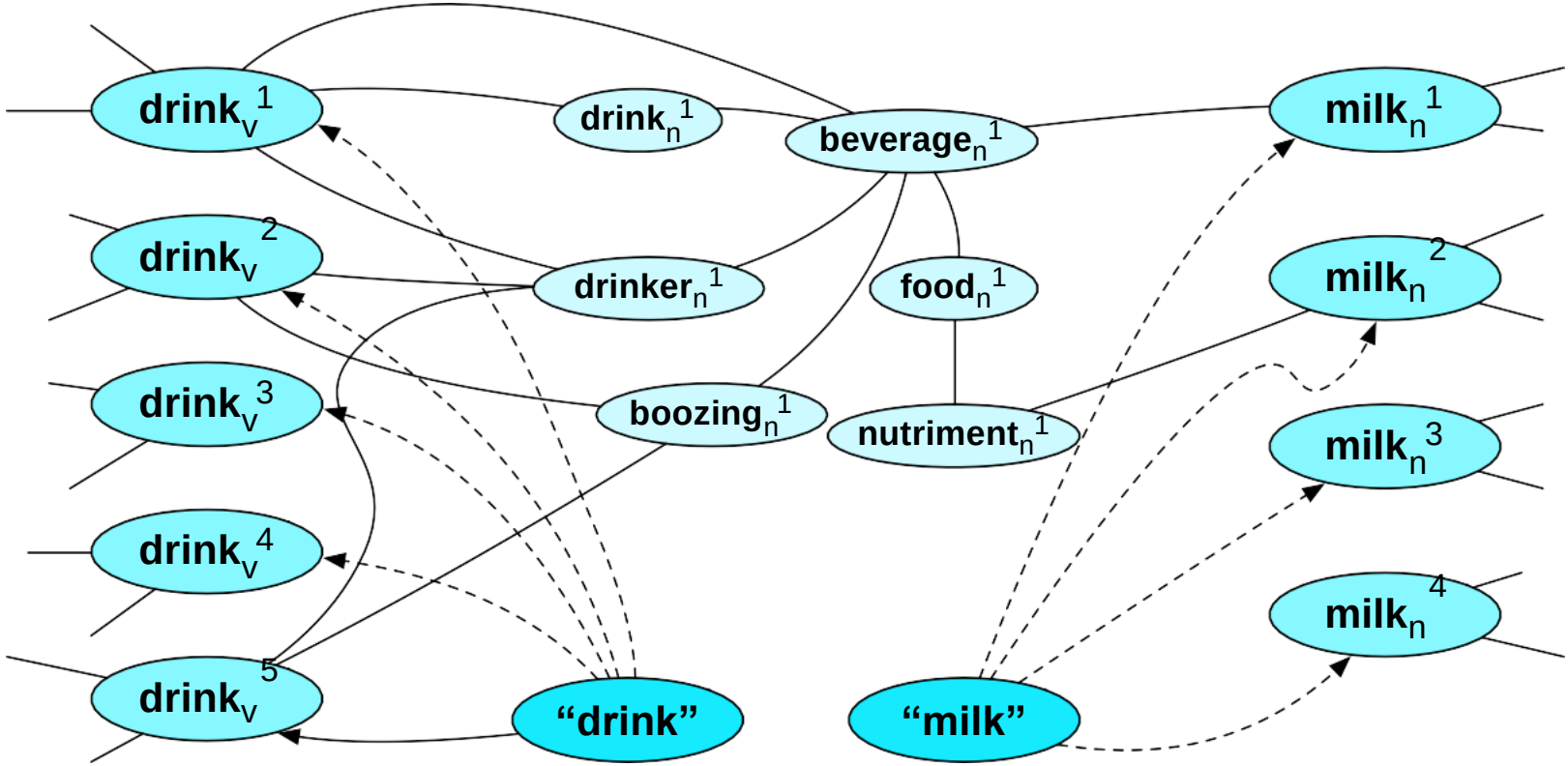
An undirected graph is set of nodes tha are connected together by bidirectional edges (lines).

How to use the graph for WSD

"She drank some milk"

- choose the *most central* sense

(several algorithms have been proposed recently)



Word Meaning and Similarity

Word Similarity: Thesaurus Methods

Word Similarity

- **Synonymy**: a binary relation
  - Two words are either synonymous or not
- **Similarity** (or **distance**): a looser metric
  - Two words are more similar if they share more features of meaning
- Similarity is properly a relation between **senses**
  - We do not say “The word “`bank`” is not similar to the word “`slope`” “, bu w say.
    - Bank[1] is similar to fund[3]
    - Bank[2] is similar to slope[5]
- But we’ll compute similarity over both words and senses

Why word similarity

- Information retrieval
- Question answering
- Machine translation
- Natural language generation
- Language modeling
- Automatic essay grading
- Plagiarism detection
- Document clustering

Word similarity and word relatedness

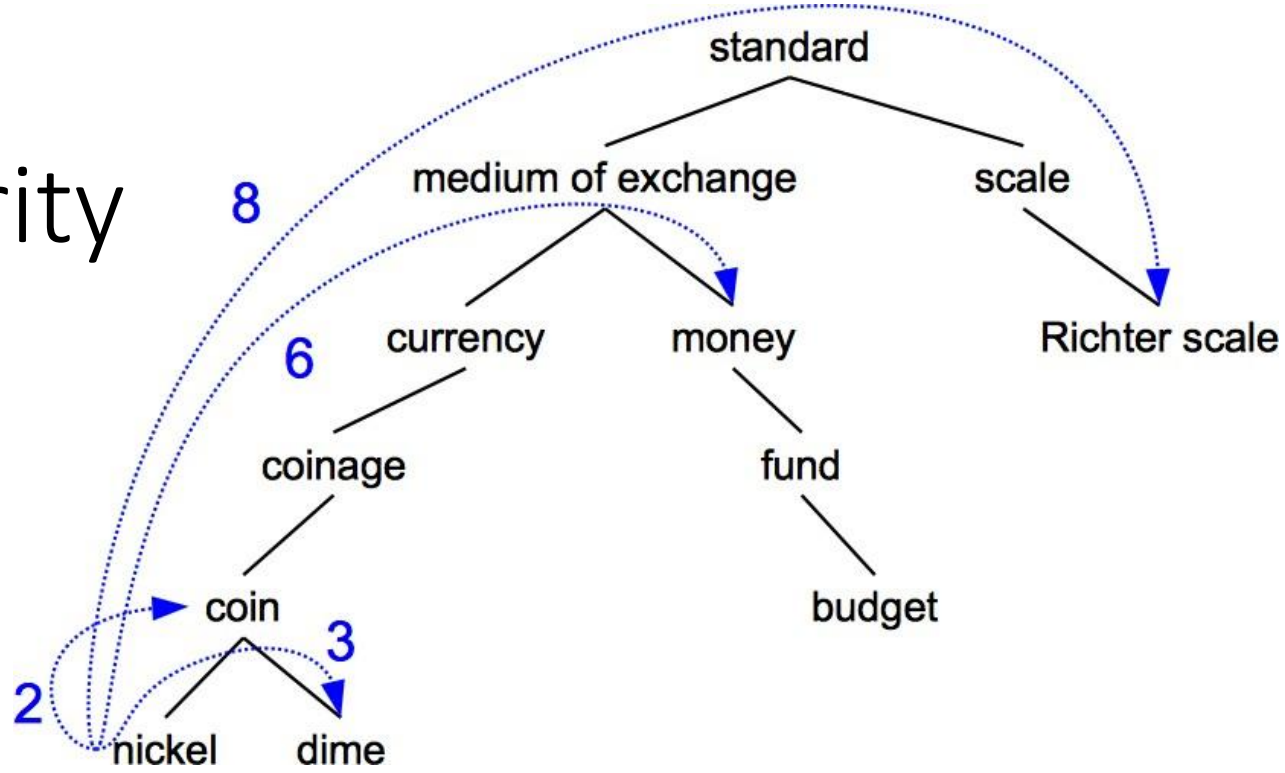
- We often distinguish **word similarity** from **word relatedness**
  - **Similar words**: near--synonyms
    - `car, bicycle:` **similar**
  - **Related words**: can be related any way
    - `car, gasoline:` **related**, not similar

Cf. Synonyms: car & automobile

Two classes of similarity algorithms

- Thesaurus--based algorithms
  - Are words “nearby” in hypernym hierarchy?
  - Do words have similar glosses (definitions)?
- Distributional algorithms:

Path-based similarity



- Two concepts (senses/synsets) are similar if they are near each other in the thesaurus hierarchy
  - =have a short path between them
  - concepts have path 1 to themselves

Refinements to path--based similarity

- pathlen(c_1,c_2) = **(distance metric)** = 1 + number of edges in the shortest path in the hypernym graph between *sense nodes* c_1 and c_2

- $\text{simpath}(c_1,c_2) = \frac{1}{\text{pathlen}(c_1,c_2)}$

Sense similarity metric: 1 over the distance!

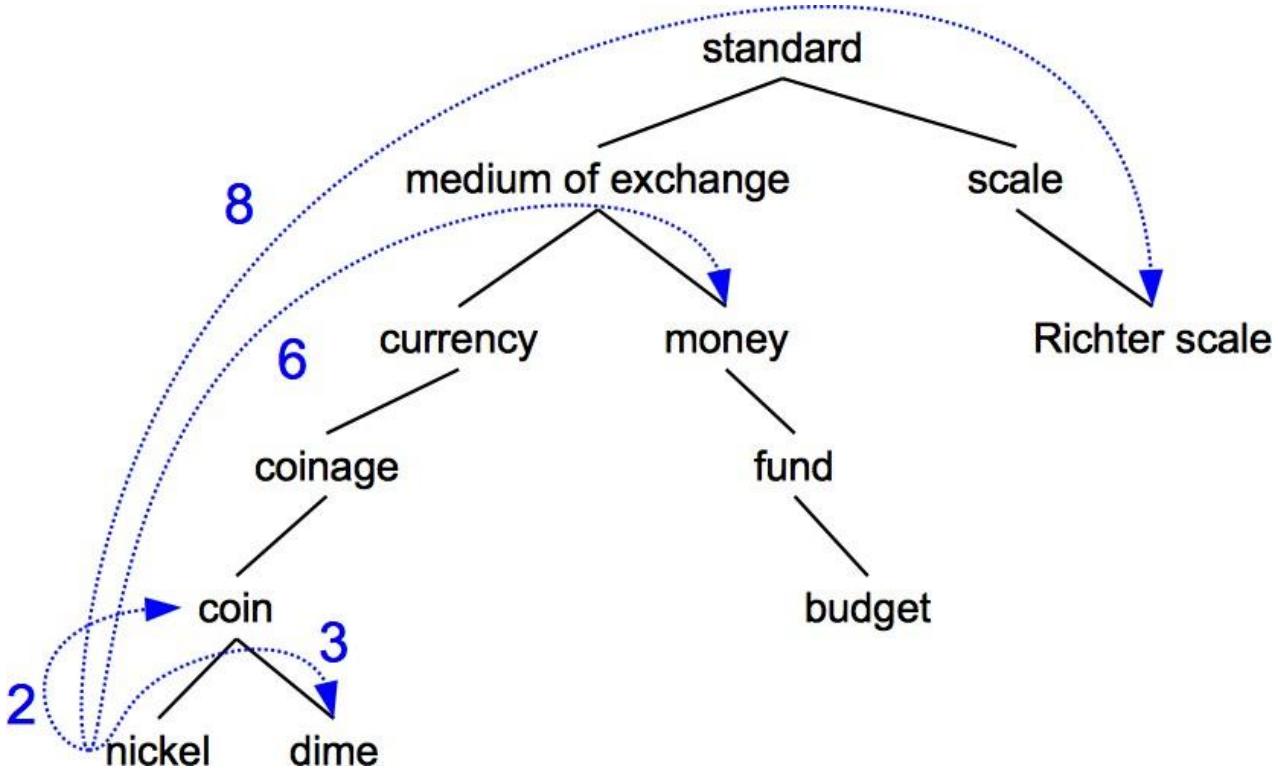
- $\text{wordsim}(w_1,w_2) = \max_{c_1 \in \text{senses}(w_1), c_2 \in \text{senses}(w_2)} \text{sim}(c_1,c_2)$

Word similarity metric: max similarity among pairs of senses.

For all senses of w1 and all senses of w2, take the similarity between each of the senses of w1 and each of the senses of w2 and then take the maximum similarity between those pairs.

Example: path--based similarity

simpath(c_1,c_2) = 1/pathlen(c_1,c_2)



simpath(*nickel,coin*) = 1/2 = .5

simpath(*fund,budget*) = 1/2 = .5

simpath(*nickel,currency*) = 1/4 = .25

simpath(*nickel,money*) = 1/6 = .17

simpath(*coinage,Richter scale*) = 1/6 = .17

Problem with basic path--based similarity

- Assumes each link represents a uniform distance
  - But *nickel* to *money* seems to us to be closer than *nickel* to *standard*
  - Nodes high in the hierarchy are very abstract
- We instead want a metric that
  - Represents the cost of each edge independently
  - Words connected only through abstract nodes
    - are less similar

Information content similarity metrics

Resnik 1995. Using information content to evaluate semantic similarity in a taxonomy. IJCAI

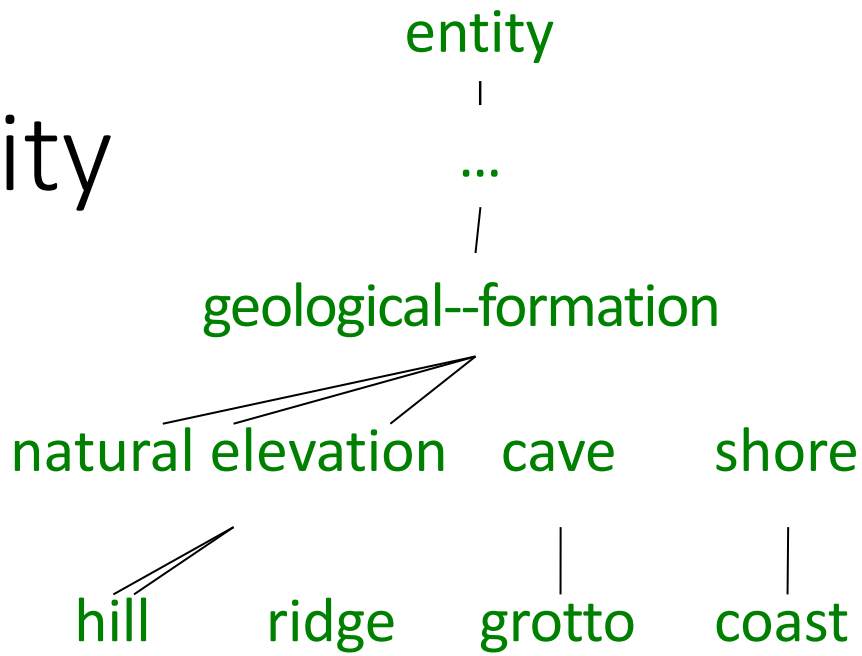
- In simple words:
  - We define the probability of a concept C as the probability that a randomly selected word in a corpus is an instance of that concept.
  - Basically, for each random word in a corpus we compute how probable it is that it belongs to a certain concepts.

Formally: Information content similarity metrics

Resnik 1995. Using information content to evaluate semantic similarity in a taxonomy. IJCAI

- Let's define $P(c)$ as:
  - The probability that a randomly selected word in a corpus is an instance of concept c
  - Formally: there is a distinct random variable, ranging over words, associated with each concept in the hierarchy
    - for a given concept, each observed noun is either
      - a member of that concept with probability $P(c)$
      - not a member of that concept with probability $1\text{-}P(c)$
  - All words are members of the root node (Entity)
    - $P(root)=1$
  - The lower a node in hierarchy, the lower its probability

Information content similarity



- For every word (ex “natural elevation”), we count all the words in that concepts, and then we normalize by the total number of words in the corpus.
- we get a probability value that tells us how probable it is that a random word is a an instance of that concept

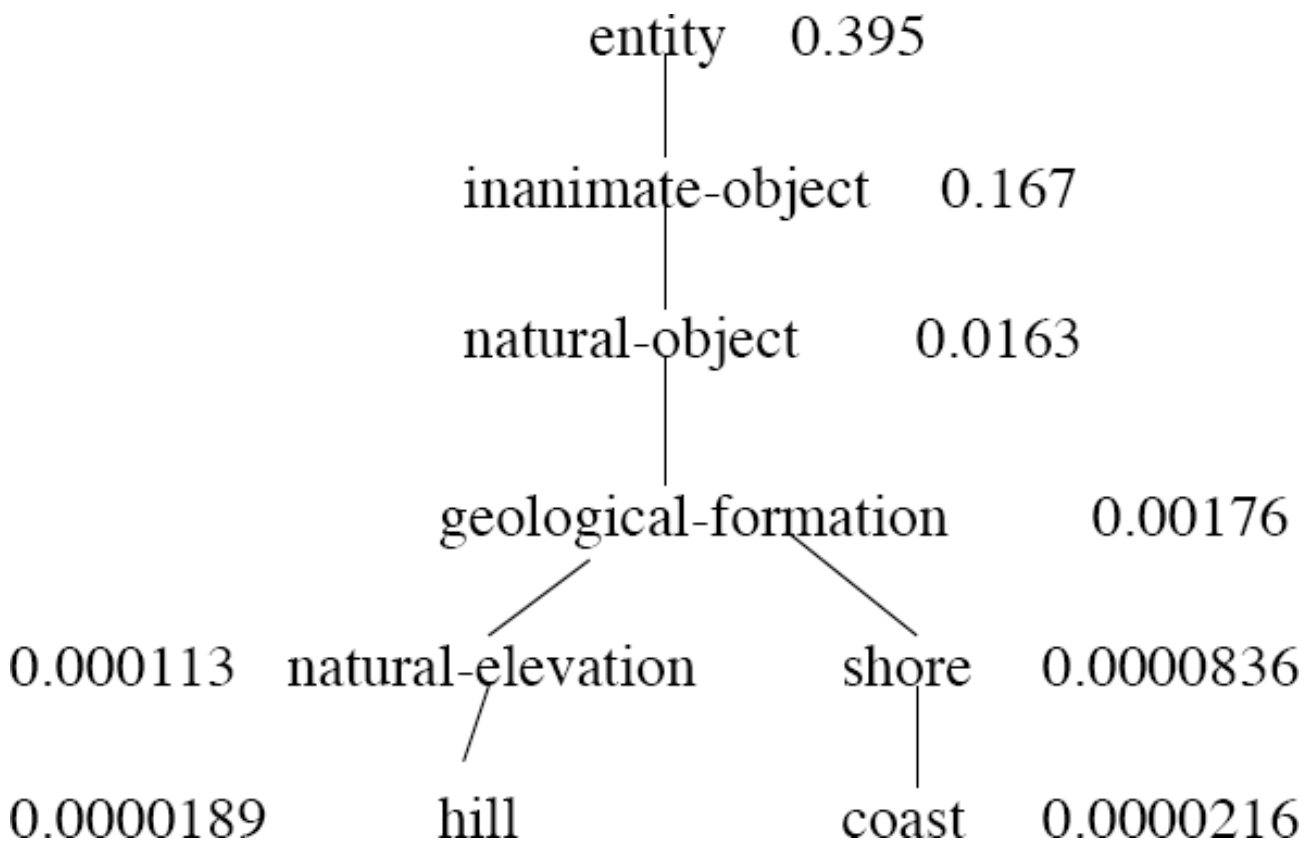
$$P(c) = \frac{\sum_{w \in words(c)} count(w)}{N}$$

In order o compute the probability of the term "natural elevation", we take ridge, hill + natural elevation itself

Information content similarity

- WordNet hierarchy augmented with probabilities P(c)

D. Lin. 1998. An Information--Theoretic Definition of Similarity. ICML 1998

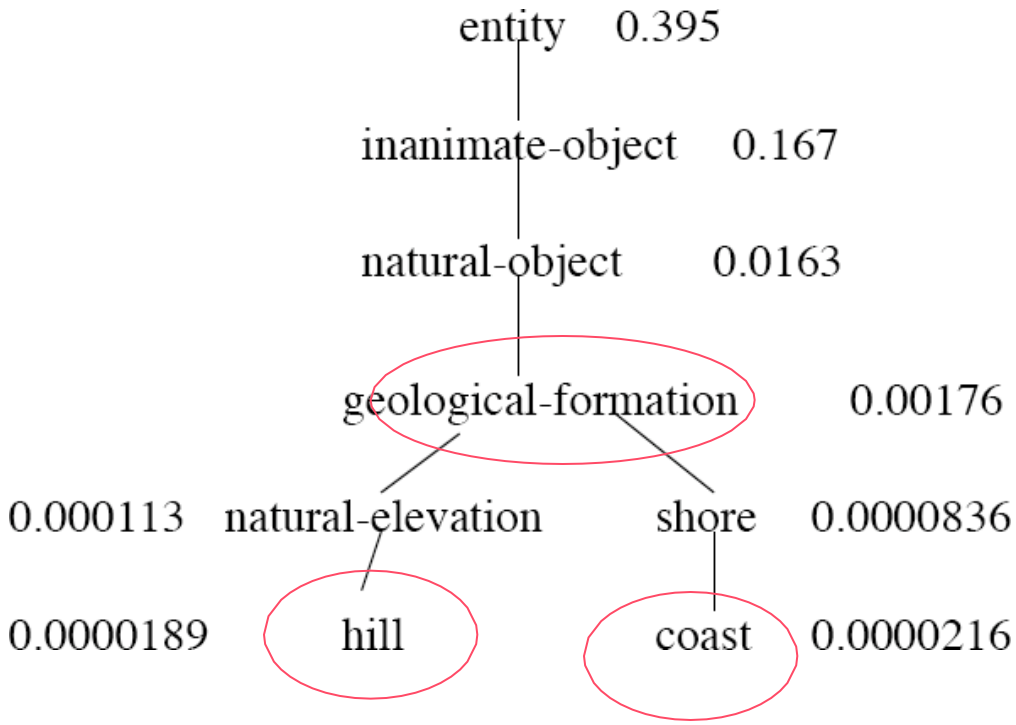


Information content: definitions

1. Information content:
   1. $IC(c) = -\log P(c)$
2. Most informative subsumer
   (Lowest common subsumer)

$LCS(c_1,c_2) =$

The most informative (lowest) node in the hierarchy subsuming both c_1 and c_2



- A lot of people prefer the term **surprisal** to information or to information content.

$$-\log p(x)$$

It measures the amount of surprise generated by the event x.

The smaller the probability of x, the bigger the surprisal is.

It's helpful to think about it this way, particularly for linguistics examples.

Using information content for similarity: the Resnik method

Philip Resnik. 1995. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. IJCAI 1995.
Philip Resnik. 1999. Semantic Similarity in a Taxonomy: An Information--Based Measure and its Application to Problems of Ambiguity in Natural Language. JAIR 11, 95–130.

- The similarity between two words is related to their common information
- The more two words have in common, the more similar they are
- Resnik: measure common information as:
  - The information content of the most informative (lowest) subsumer (MIS/LCS) of the two nodes
  - $\text{sim}_{\text{resnik}}(c_1,c_2) = -\log P(\text{LCS}(c_1,c_2))$

Dekang Lin method

Dekang Lin. 1998. An Information--Theoretic Definition of Similarity. ICML

- Intuition: Similarity between A and B is not just what they have in common
- The more **differences** between A and B, the less similar they are:
  - Commonality: the more A and B have in common, the more similar they are
  - Difference: the more differences between A and B, the less similar
- Commonality: IC(common(A,B))
- Difference: IC(description(A,B)--IC(common(A,B))

Dekang Lin similarity theorem

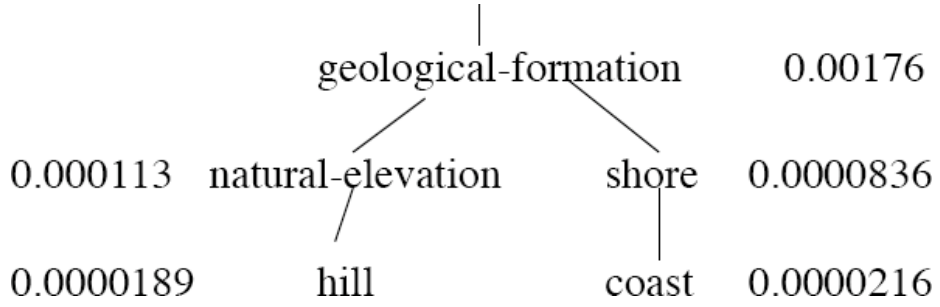
- The similarity between A and B is measured by the ratio between the amount of information needed to state the commonality of A and B and the information needed to fully describe what A and B are

$$sim_{Lin}(A,B) \propto \frac{IC(common(A,B))}{IC(description(A,B))}$$

- Lin (altering Resnik) defines IC(common(A,B)) as 2 x information of the LCS

$$sim_{Lin}(c_1,c_2) = \frac{2\log P(LCS(c_1,c_2))}{\log P(c_1)+\log P(c_2)}$$

Lin similarity function



$$sim_{Lin}(A,B) = \frac{2\log P(LCS(c_1,c_2))}{\log P(c_1)+\log P(c_2)}$$

$$sim_{Lin}(\text{hill},\text{coast}) = \frac{2\log P(\text{geological-formation})}{\log P(\text{hill})+\log P(\text{coast})}$$

$$= \frac{2\ln 0.00176}{\ln 0.0000189+\ln 0.0000216}$$

$$= .59$$

The (extended) Lesk Algorithm

- A thesaurus--based measure that looks at **glosses**
- Two concepts are similar if their glosses contain similar words
  - ***Drawing paper***: paper that is specially prepared for use in drafting
  - ***Decal***: the art of transferring designs from specially prepared paper to a wood or glass or metal surface
- For each *n*–word phrase that's in both glosses
  - Add a score of n^2
  - Paper and specially prepared for $1 + 2^2 = 5$
  - Compute overlap also for other relations
    - glosses of hypernyms and hyponyms

Summary: thesaurus--based similarity

$$\text{sim}_{\text{path}}(c_1, c_2) = \frac{1}{pathlen(c_1, c_2)}$$

$$\text{sim}_{\text{resnik}}(c_1, c_2) = -\log P(LCS(c_1, c_2))$$

$$\text{sim}_{\text{lin}}(c_1, c_2) = \frac{2 \log P(LCS(c_1, c_2))}{\log P(c_1) + \log P(c_2)}$$

$$\text{sim}_{eLesk}(c_1, c_2) = \sum_{r,q \in RELS} \text{overlap}(\text{gloss}(r(c_1)), \text{gloss}(q(c_2)))$$

Libraries for computing thesaurus--based similarity

- NLTK
  - http://nltk.github.com/api/nltk.corpus.reader.html?highlight=similarity – nltk.corpus.reader.WordNetCorpusReader.res_similarity

- WordNet::Similarity
  - http://wn--similarity.sourceforge.net/
  - Web-based interface:
    - http://marimba.d.umn.edu/cgi--bin/similarity/similarity.cgi

Machine Learning based approach

Basic idea

- If we have data that has been hand--labelled with correct word senses, we can used a supervised learning approach and learn from it!
  - We need to extract features and train a classifier
  - The output of training is an automatic system capable of assigning **sense labels** TO **unlabelled words** in a context.

Two variants of WSD task

- Lexical Sample task
  - (we need labelled corpora for individual senses)
  - Small pre--selected set of target words (*ex difficulty*)
  - And inventory of senses for each word
  - **Supervised machine learning: train a classifier for each word**
- All-words task
  - (each word in each sentence is labelled with a sense)
  - Every word in an entire text
  - A lexicon with senses for each word

SENSEVAL 1–2–3

Supervised Machine Learning Approaches

- Summary of what we need:
  - the **tag set** (“sense inventory”)
  - the **training corpus**
  - A set of **features** extracted from the training corpus
  - A **classifier**

Supervised WSD 1: WSD Tags

- What's a tag?

  A dictionary sense?
- For example, for WordNet an instance of "bass" in a text has 8 possible tags or labels (bass1 through bass8).

8 senses of “bass” in WordNet

1. bass –(the lowest part of the musical range)
2. bass, bass part –(the lowest part in polyphonic music)
3. bass, basso –(an adult male singer with the lowest voice)
4. sea bass, bass –(flesh of lean-fleshed saltwater fish of the family Serranidae)
5. freshwater bass, bass –(any of various North American lean-fleshed freshwater fishes especially of the genus Micropterus)
6. bass, bass voice, basso –(the lowest adult male singing voice)
7. bass –(the member with the lowest range of a family of musical instruments)
8. bass –(nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

SemCor

SemCor: 234,000 words from Brown Corpus, manually tagged with WordNet senses

<wf pos=PRP>**He**</wf>

<wf pos=VB lemma=recognize wnsn=4 lexsn=2:31:00::>**recognized**</wf>

<wf pos=DT>**the**</wf>

<wf pos=NN lemma=gesture wnsn=1 lexsn=1:04:00::>**gesture**</wf>

<punc>.</punc>

Supervised WSD: Extract feature vectors Intuition from Warren Weaver (1955):

"If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words…

But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word… the window

The practical question is : ``What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?"

Feature vectors

- **Vectors** of sets of feature/value pairs

Two kinds of features in the vectors

- **Collocational** features and **bag–of–words** features

  - **Collocational/Paradigmatic**
    - Features about words at **specific** positions near target word
      - Often limited to just word identity and POS
  - **Bag–of–words**
    - Features about words that occur anywhere in the window (regardless of position)
      - Typically limited to frequency counts

Generally speaking, a **collocation** is a sequence of words or terms that co-occur **more often than would be expected by chance.** But here the meaning is not exactly this…

Examples

- Example text (WSJ):

  An electric guitar and **bass** player stand off to one side not really part of the scene

- Assume a window of +/–2 from the target

Examples

- Example text (WSJ)

  An electric guitar and **bass** player stand off to

  one side not really part of the scene,

  - Assume a window of +/-2 from the target

Collocational features

- Position--specific information about the words and collocations in window
- guitar | and | bass | player | stand

$$[w_{i-2}, \mathrm{POS}_{i-2}, w_{i-1}, \mathrm{POS}_{i-1}, w_{i+1}, \mathrm{POS}_{i+1}, w_{i+2}, \mathrm{POS}_{i+2}, w_{i-2}^{i-1}, w_i^{i+1}]$$

[guitar, NN, and, CC, player, NN, stand, VB, and guitar, player stand]

- word 1,2,3 grams in window of ±3 is common

Bag-of-words features

- “an unordered set of words” – position ignored
- Choose a vocabulary: a useful subset of words in a training corpus
- Either: the count of how often each of those terms occurs in a given window OR just a binary “indicator” 1 or 0

Co–Occurrence Example

- Assume we've settled on a possible vocabulary of 12 words in "bass" sentences:

[*fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band*]

- The vector for:

  guitar and bass player stand

  [0,0,0,1,0,0,0,0,0,0,1,0]

Word Sense Disambiguation

Classification

Classification

- *Input*:
  - a word w and some features f
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

Any kind of classifier
- Naive Bayes
- Logistic regression
- Neural Networks
- Support--vector machines
- k-Nearest Neighbors
- etc.

- *Output*: a predicted class $c \in C$

Coherence

- Coherence in discourse refers to the way in which the sentences of a text relate to each other to form a unified whole. An important aspect of coherence is how references (like pronouns, noun phrases, etc.) are used to maintain the flow and connectivity of ideas. Coherence reference phenomena are mechanisms that help achieve this connection.

Coherence Reference Phenomena

1. **Anaphora**: Anaphora is a reference to something previously mentioned in the discourse.
    1. Example: "John went to the store. He bought some milk." ("He" refers to "John")
2. **Cataphora**: Cataphora is a reference to something that is mentioned later in the discourse.
    1. Example: "Before he could leave, John had to finish his work." ("he" refers to "John")
3. **Exophora**: Exophora is a reference to something outside the text, often in the physical or situational context.
    1. Example: "Look at that!" (where "that" refers to something in the physical environment)
4. **Endophora**: Endophora is a general term for both anaphora and cataphora, i.e., references within the text.
    1. Anaphoric endophora: "He was hungry. John ate an apple."
    2. Cataphoric endophora: "When he arrived, John was tired."
5. **Coreference**: Coreference occurs when two or more expressions in a text refer to the same entity.
    1. Example: "Alice lost her book. She can't find it anywhere." ("her" and "she" refer to "Alice", and "it" refers to "her book")

Penn Tree Bank

Penn Treebank is a large annotated corpus of English that is widely used in computational linguistics and natural language processing (NLP) for training and evaluating algorithms. It was created by the University of Pennsylvania's Linguistic Data Consortium and has been a foundational resource in the field.

Key Features of the Penn Treebank

- Annotated Corpus: The Penn Treebank includes syntactic and semantic annotations of texts, making it a valuable resource for developing and testing NLP models.
- Part-of-Speech (POS) Tagging: It provides POS tags for each word, which are essential for various NLP tasks such as lemmatization, parsing, and machine translation.
- Syntactic Trees: The corpus contains syntactic trees that represent the grammatical structure of sentences. These trees are crucial for tasks such as syntactic parsing and grammar induction.
- Wide Coverage: It includes texts from various genres, such as Wall Street Journal articles, telephone conversations, and more, offering a broad spectrum of English language use.

Penn Treebank POS Tags

- The Penn Treebank uses a set of POS tags to annotate words. Here is a list of some common tags:
- CC: Coordinating conjunction
- CD: Cardinal number
- DT: Determiner
- EX: Existential there
- FW: Foreign word
- IN: Preposition or subordinating conjunction
- JJ: Adjective
- JJR: Adjective, comparative
- JJS: Adjective, superlative
- LS: List item marker

- MD: Modal
- NN: Noun, singular or mass
- NNS: Noun, plural
- NNP: Proper noun, singular
- NNPS: Proper noun, plural
- PDT: Predeterminer
- POS: Possessive ending
- PRP: Personal pronoun
- PRP$: Possessive pronoun
- RB: Adverb
- RBR: Adverb, comparative
- RBS: Adverb, superlative

- RP: Particle
- SYM: Symbol
- TO: to
- UH: Interjection
- VB: Verb, base form
- VBD: Verb, past tense
- VBG: Verb, gerund or present participle
- VBN: Verb, past participle
- VBP: Verb, non-3rd person singular present
- VBZ: Verb, 3rd person singular present
- WDT: Wh-determiner
- WP: Wh-pronoun
- WP$: Possessive wh-pronoun
- WRB: Wh-adverb

- **Example Sentence with Penn Treebank POS Tags**
- Consider the sentence: "The quick brown fox jumps over the lazy dog."
- Here is how it might be tagged using Penn Treebank POS tags:
- The/DT quick/JJ brown/JJ fox/NN jumps/VBZ over/IN the/DT lazy/JJ dog/NN

- **Applications of the Penn Treebank**

1. **POS Tagging**: The POS-tagged data is used to train and evaluate POS taggers.
2. **Syntactic Parsing**: The syntactic trees are used to train parsers that can predict the syntactic structure of sentences.
3. **Machine Learning**: The annotated data serves as a benchmark for various machine learning models in NLP.
4. **Linguistic Research**: Researchers use the Penn Treebank to study syntactic and semantic phenomena in English.