
# T. Y. Artificial Intelligence & Data Science
# Semester VI

**Prof. Mandar Diwakar**
*Asst. Professor*
*Department of AI & DS*
*Vishwakarma Institute of Information Technology*

# ADUA32203: Natural Language Processing

# Unit II

# Morphological Analysis

| Unit II: | Morphological Analysis | 7 Hrs |
|---|---|---|
| **Types of Morphology:** Survey of English and Indian Languages, Finite-State Morphological Parsing, Building a Finite-State Lexicon, Finite-State Transducers, FSTs for Morphological Parsing, The Porter Stemmer, Word and Sentence Tokenization, Detecting and Correcting Spelling Errors, Minimum Edit Distance, Human Morphological Processing, N –Grams- N-gram language model, N-gram for spelling correction | | |

Morphological Analysis

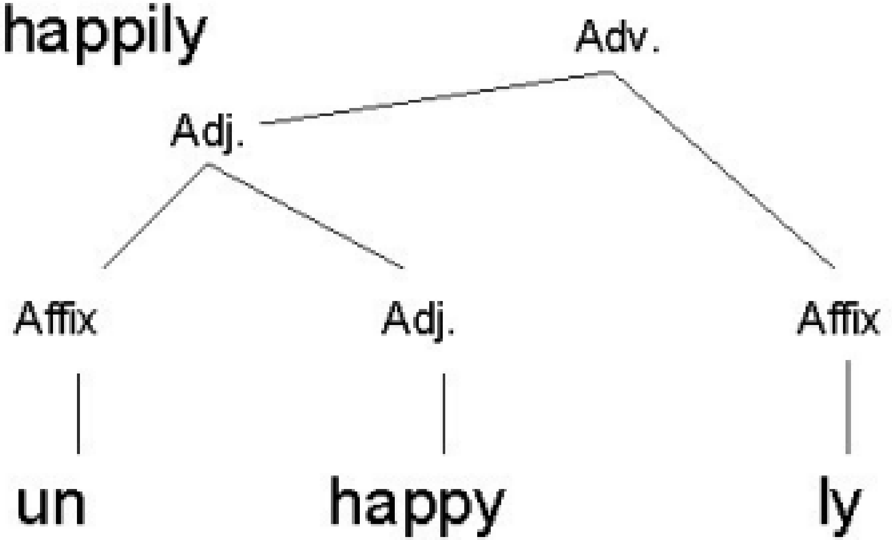
# Survey of English and Indian Languages

Survey of English and Indian Languages

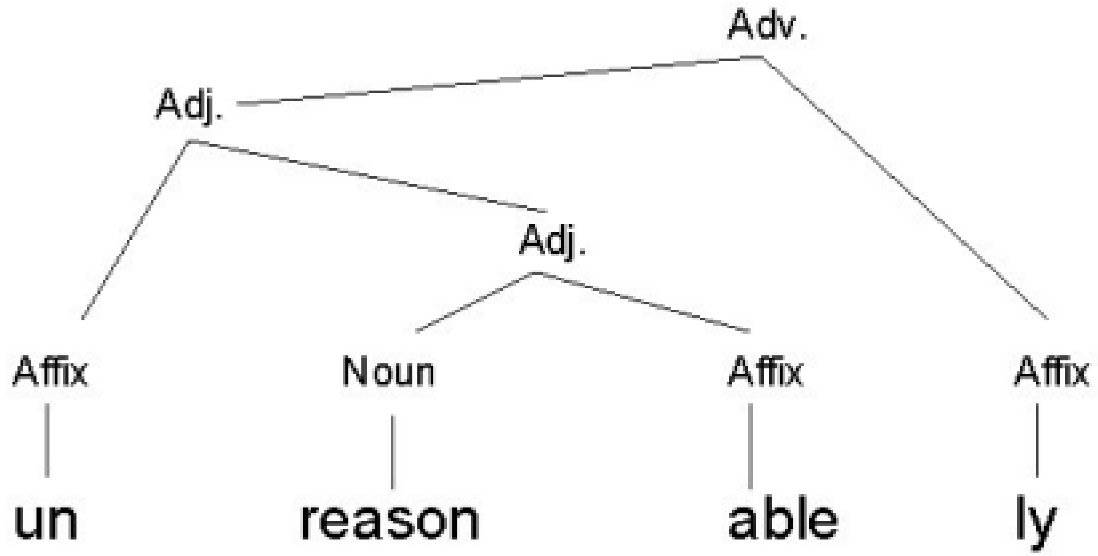
## SURVEY OF ENGLISH MORPHOLOGY

 It is often useful to distinguish two broad classes of **morphemes**: ***stems*** and ***affixes***.  The ***stem*** is the **'main'** morpheme of the word, supplying the main meaning,  The ***affixes*** add **'additional'** meanings of various kinds.

Unhappily




Unreasonably



# Survey of English and Indian Languages

## SURVEY OF ENGLISH MORPHOLOGY

- ***Affixes*** are further divided into ***prefixes, suffixes, infixes***, and ***circumfixes***.

- ***Prefixes*** precede the stem(root).

| Prefix | Meaning | Example |
|---|---|---|
| de- | undo | derail |
| ex- | non, out | ex-president, extend |
| in- | negate | incapable |
| anti- | negate | anti-social |
| pre- | before | predate |
| sub- | under, below | subway |
| un- | negate | undo |
| dis- | negate | disengage |
| mis- | wrongly | mistreat |
| non- | negate | nonsense |
| pro- | for | proclaim |
| re- | again, repeat | reread |
| trans- | across | transatlantic |
| bi- | two, twice | bilingual |
| co- | along with | co-author |

Survey of English and Indian Languages

## SURVEY OF ENGLISH MORPHOLOGY

- *Suffixes* follow the stem(root)

| Suffix | Meaning | Example |
|---|---|---|
| -s | plural | horses |
| -s | third person singular verbal inflection | likes |
| -‘s | possession | Mary’s |
| -ed | past tense | walked |
| -en | past participle | eaten |
| -ing | progressive verbal inflection | reading |
| -er | comparative | brighter |
| -est | superlative | brightest |

# Survey of English and Indian Languages

## SURVEY OF ENGLISH MORPHOLOGY

- *Circumfixes* do both ***precede the stem*** as well as ***follow the stem***.

| Root | Word class | Circumfixation | Gloss | Word class |
|---|---|---|---|---|
| Legal | Adjective | Il-legal-ity | Illegality | Noun |
| Liberal | Adjective | Il-liberal-ity | Illiberality | Noun |

| Roots | Word class | Circumfixation | Gloss | Word class |
|---|---|---|---|---|
| Imagine | verb | un-imagin-able | unimaginable | adjective |
| Accept | verb | **un**-accept-**able** | unacceptable | adjective |
| Question | verb/noun | **un**-question-**able** | unquestionable | adjective |

| Stem | Word class | Circumfixation | Gloss | Word class |
|---|---|---|---|---|
| Advisable | adjective | in-advisab-ly | inadvisably | adverb |
| Correct | adjective | in-correct-ly | incorrectly | adverb |

# Survey of English and Indian Languages

SURVEY OF ENGLISH MORPHOLOGY

- *Infixes* are inserted inside the stem.

- Infixes are relatively rare in English, but you can find them in the plural forms of some words.

- For example, **cupful, spoonful** and **passerby** can be pluralized as ***cupsful, spoonsful,*** and ***passersby***, using "s" as an infix.

- Another example is the insertion of an (often offensive) ***intensifier*** into a word, as in ***"fan-freakin'-tastic."*** Such whole-word insertions are sometimes called infixes, though this phenomenon is more traditionally known as tmesis.

□ English has a relatively simple inflectional system; only ***nouns, verbs,*** and sometimes ***adjectives*** can be ***inflected***, and the number of possible inflectional affixes is quite small. □ English ***nouns*** have only two kinds of inflection: an affix that marks **plural** and an affix that marks **possessive**.

# Survey of English and Indian Languages

## Inflection MORPHOLOGY

□ English verbal inflection is more complicated than nominal inflection.

| Morphological Form Classes | Regularly Inflected Verbs | | | |
|---|---|---|---|---|
| stem | walk | merge | try | map |
| -*s* form | walks | merges | tries | maps |
| -*ing* participle | walking | merging | trying | mapping |
| Past form or -*ed* participle | walked | merged | tried | mapped |

| Morphological Form Classes | Irregularly Inflected Verbs | | |
|---|---|---|---|
| stem | eat | catch | cut |
| -*s* form | eats | catches | cuts |
| -*ing* participle | eating | catching | cutting |
| Past form | ate | caught | cut |
| -*ed* participle | eaten | caught | cut |

# Survey of English and Indian Languages

## Derivational MORPHOLOGY

 A very common kind of derivation in English is the formation of new nouns, often from verbs or adjectives. Process is called ***NOMINALIZATION.***

 For example,

- the suffix -***ation*** produces nouns from verbs ending often in the
- suffix -***ize*** (computerize !computerization).

| Suffix | Base Verb/Adjective | Derived Noun |
|---|---|---|
| -ation | computerize (V) | computerization |
| -ee | appoint (V) | appointee |
| -er | kill (V) | killer |
| -ness | fuzzy (A) | fuzziness |

# Survey of English and Indian Languages

## Derivational MORPHOLOGY

- Adjectives can also be derived from nouns and verbs.

| Suffix | Base Noun/Verb | Derived Adjective |
|---|---|---|
| -al | computation (N) | computational |
| -able | embrace (V) | embraceable |
| -less | clue (N) | clueless |

# Survey of English and Indian

# Languages *SURVEY*

*OF Hindi MORPHOLOGY*

- Hindi morphological structure consists of various word classes in


Hindi Varnamala Chart

which their derivational and inflection forms are described.

- Word classes include ***nouns, verbs, adjectives, pronouns, particls, connections*** and ***interjections***.

# Survey of English and Indian Languages

## *SURVEY OF Hindi MORPHOLOGY*

**Prefixes In Hindi**

**Suffixes In Hindi**

- Circumfixes In Hindi

**Survey of English and Indian Languages**

-ई -अ

- **Noun ---> Feminine Noun (Vocation)**

नौकर (servant) ---> नौकरी (job, service)

- **Verb Stem ---> Abstract Feminine Noun**
  बोल(ना) (to speak)---> बोली (speech, language)

- **Masculine Noun ---> Diminutive Feminine**
  **Noun** डंडा (stick) ---> डंडी (small stick)

- **Adjective ---> Noun**
  अच्छा (good) ---> अच्छाई (goodness)

- **Noun ---> Noun / Adjective**
  हिन्दुस्तान ---> हिन्दुस्तानी
  - The prefix अ means “not”, “un-“, “im-“, “without”, “-less”, etc. It comes from Sanskrit/Hindi.

  - **With adjectives:**
    अ + संभव (possible) = असंभव (impossible) अ + सफ़ल (successful) = असफ़ल (unsuccessful)

- **With nouns:**
  अ + हिंसा (violence) = अहिंसा (nonviolence)

- -आई

# Survey of English and Indian Languages

-मान

The suffix -ान converts a noun into an adjective.

The suffix **-आई** transforms a verb stem into a feminine noun. It comes from

Hindi.

पढ़(ना) (to read/study) + आई =
पढ़ाई (study, education)

- -इक

The suffix -इक transforms a noun into an adjective. It comes from Sanskrit.

धर्म(religion) + इक = धाहर्मक (religious) परंपरा (tradition) + इक = पारंपररक (traditional)

It comes from Sanskrit.

बुद्धि (wisdom/intelligence) + ◌्रान = बुद्धि◌्रान (wise, intelligent)

-आना

The suffix -आना converts a noun into an adjective or different noun. It comes from Persian. साल (year) + आना = सालाना (annual)

-दार

The suffix -दार converts a noun into another noun or an adjective. It comes from Persian.

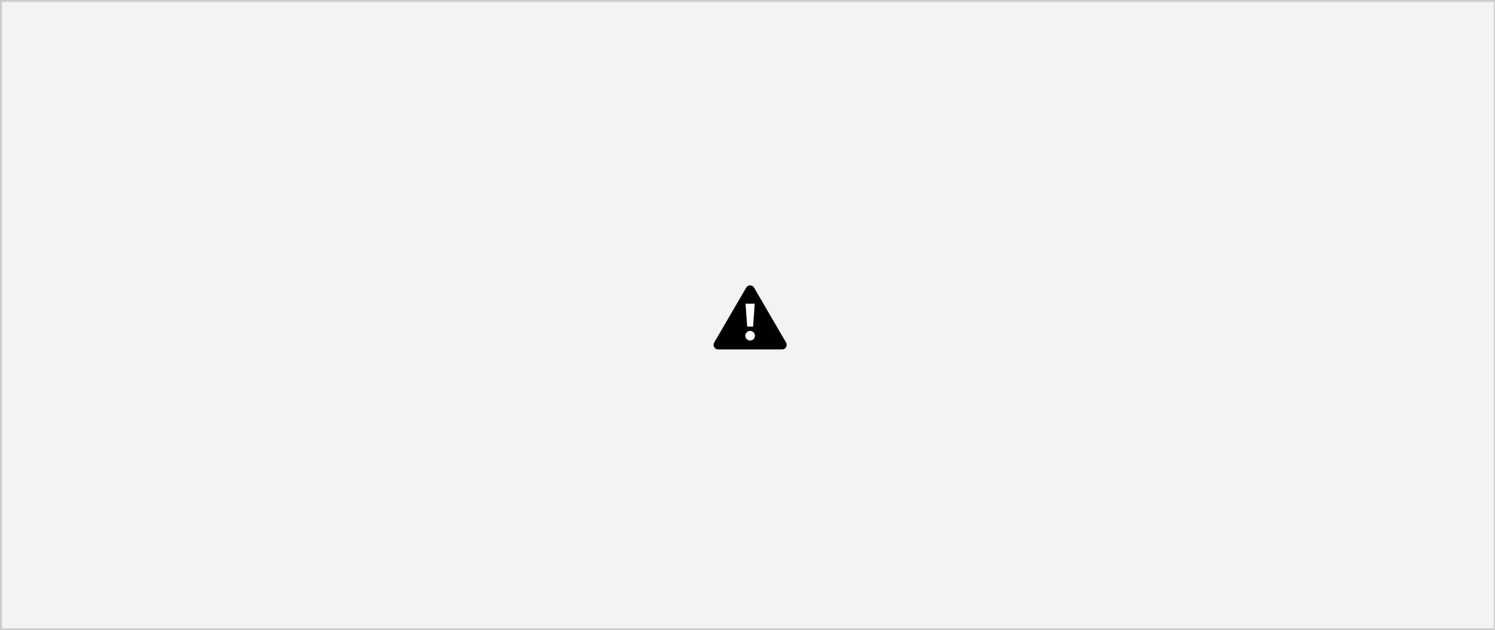
दुकान (shop) + दार = दुकानदार (shopkeeper) ई◌्रान (trust) + दार = ई◌्रानदार (honest)

NLP

#  *Finite-State Morphological Parsing*

## Finite-State Morphological Parsing

parsing just the productive **nominal plural (-s)** and the **verbal progressive (-ing).**



+N means that the word is a noun;
+SG means it is singular,
+PL that it is plural.

## Finite-State Morphological Parsing

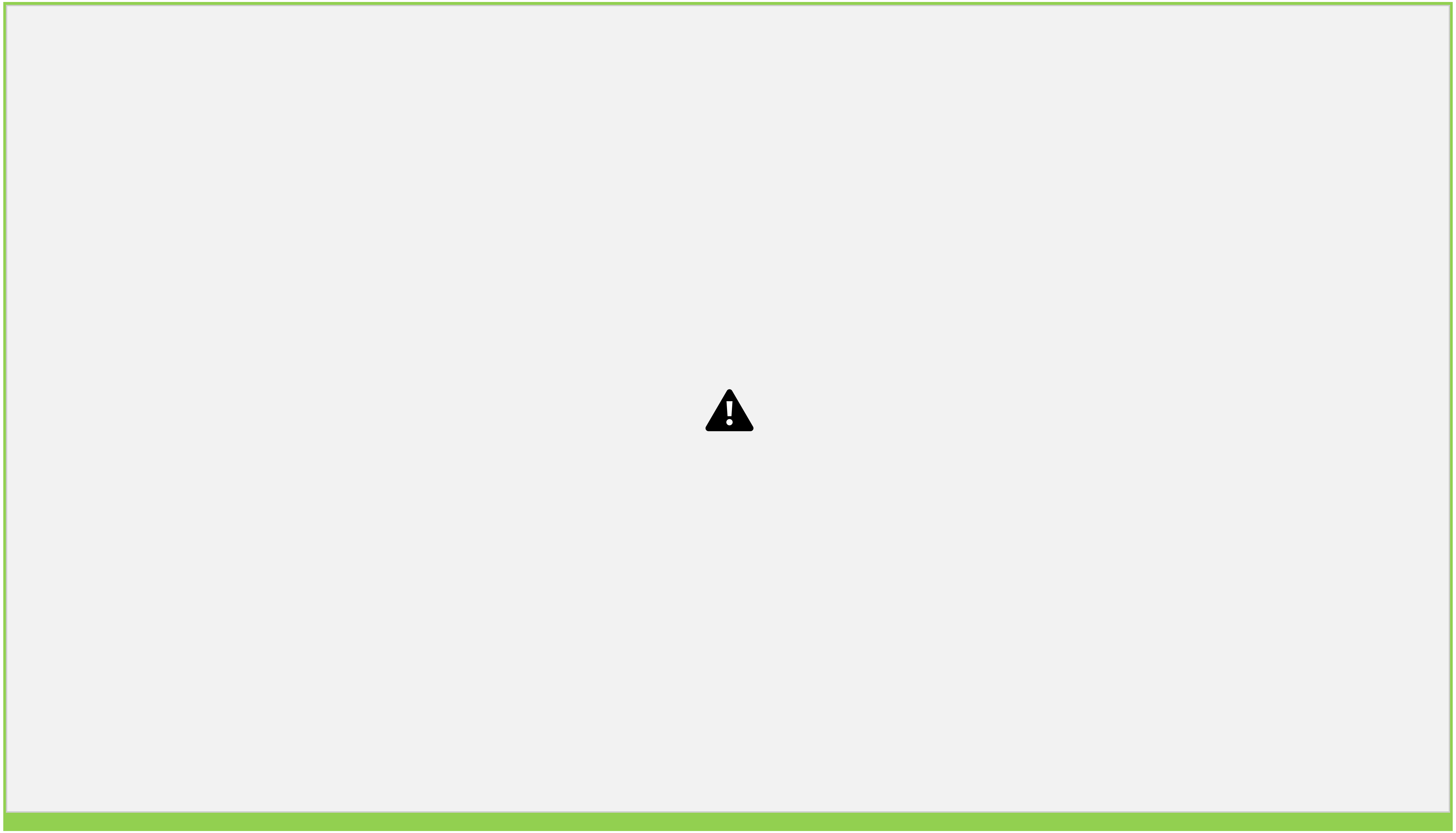
In order to build a ***morphological parser***, we'll need at least the following:

- **A lexicon**: The list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc).

- **morphotactics**: the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the rule that the English plural morpheme follows the noun rather than preceding it.

- **orthographic rules**: these spelling rules are used to model the changes that occur in a word, usually when two morphemes combine (for example the **y ----> ie** spelling rule discussed above that changes city + -s to cities rather than citys).

# Finite-State Morphological Parsing



# Finite-State Morphological Parsing

- lexicon includes ***regular nouns (reg-noun)*** that take the ***regular -s plural*** (e.g. cat, dog, fox).
- Plural of words like **fox** have an inserted **e**: foxes.
- The lexicon also includes **irregular noun** forms that don't take **-s**, both singular **irreg-sg noun** (goose, mouse) and plural **irreg-pl-noun** (geese, mice).

A finite-state automaton for English nominal inflection

• The automaton ***STATE*** has ***03*** states, which are represented by nodes in the graph. • ***State 0*** is the ***START STATE*** which we represent by the incoming arrow. • ***State 2*** is the ***final state*** or accepting state, which we represent by the ***double circle***. • It also has four transitions, which we represent by arcs in the graph.

# Finite-State Morphological Parsing

- This lexicon has three stem classes (**reg-verb-stem, irreg-verb-stem, and irreg past-verb-form**),
- 4 more affix classes **(-ed past, -ed participle, -ing participle, and 3rd singular -s)** A finite-state automaton for English verbal inflection

- The automaton STATE has 04 states, which are represented by nodes in the graph.
- State 0 is the START STATE which we represent by the incoming arrow.
- State 4 is the final state or accepting state, which we represent by the double circle.
- It also has 08 transitions, which we represent by arcs in the graph.

# Finite-State Morphological Parsing

- **Data on English adjectives**

**adj-root1** would include adjectives that can occur with ***un- and -ly*** (clear, happy, and real) while **adj-root2** will include adjectives that can't (big, cool, and red).

it will also recognize ungrammatical forms like **unbig, redly, and** realest.

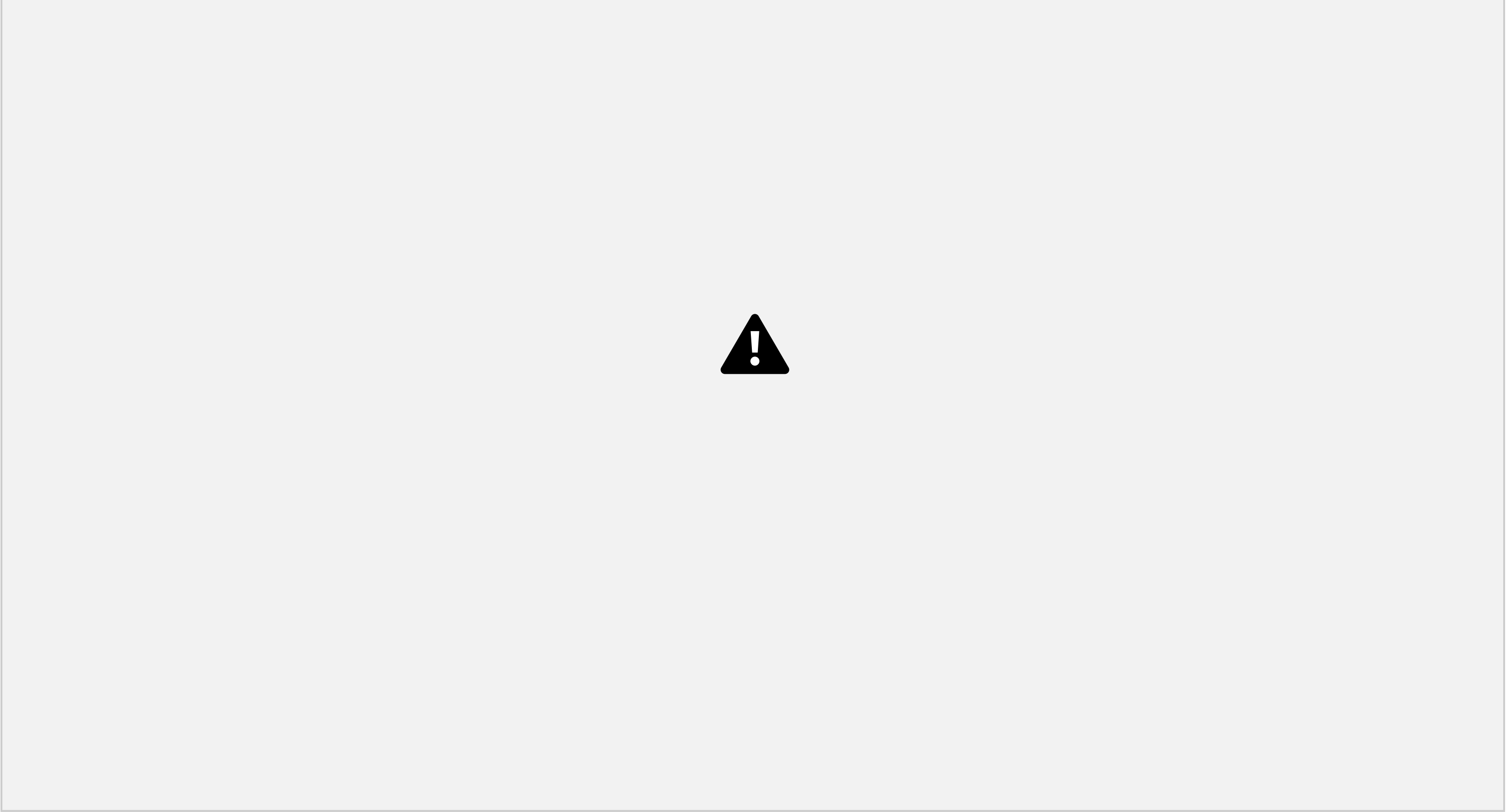
# Finite-State Morphological Parsing

# Finite-State Morphological

# Parsing

• Visual (Adjective) --> Visualize (verb) --> Visualization (noun) {q0 **--> q1 --> q2-->** q3} • Author (Noun) --> Authorize (verb) --> Authorization (noun) {q0 **--> q1--> q2-->** q3} • Skill (Noun) --> Skillful (Adjective) --> Skillfulness (noun) {q0 **--> q11 --> q8 -->** q6} • --> Skillfully (Adverb) {q0 **--> q11 --> q8 -->** q9} • Create (verb) --> Creative (Adjective) --> Creativeness (noun) {q0 **--> q7--> q8 -->** q6} • Normal(Noun) --> Normalize (Verb) --> Normalizer (noun) {q0 **--> q1 --> q2-->** q4}

# Finite-State Morphological Parsing



- *fox*
- *geese*
- *goose*
- *cat*
- *sheep*
- *sheep*
- *dog*
- *mice*
- *mouse*
- *aardvark*

Compiled FSA for a few English nouns with their inflection

NLP

# Morphology and Finite-State Transducers

## Finite-State Transducers

- Morphological parsing is implemented by ***building mapping rules that map letter sequences*** like cats on the surface level into morpheme and features sequences like cat +N +PL on the lexical level.
- Figure shows these two levels for the word cats. Note that the ***lexical level has the stem for a word***, followed by the ***morphological information +N +PL*** which tells us that cats is a plural noun.

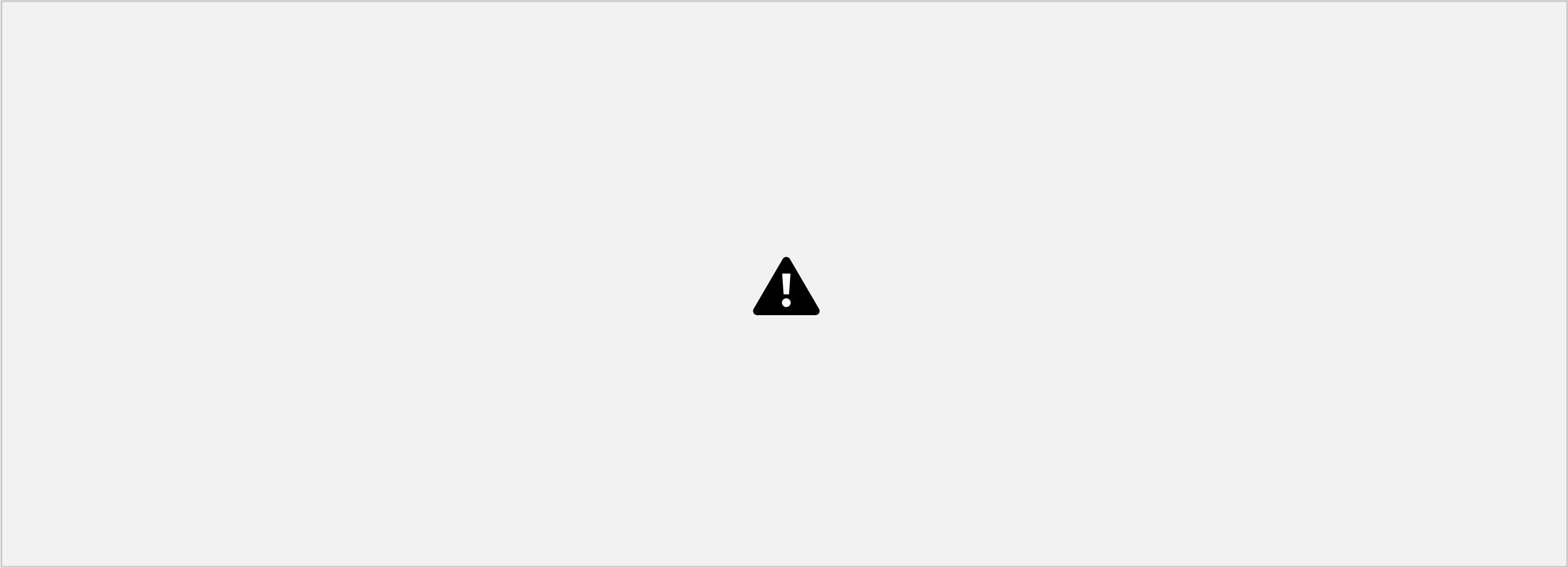
# Finite-State Transducers

- The automaton that we use for performing the mapping between these two levels is the ***finite-state transducer or FST.***
- A transducer maps between FST one set of symbols and another; a finite-state transducer does this via a finite automaton.
- ***four-fold way of thinking about transducers:***

- **FST as recognizer** : a transducer that takes a pair of strings as input and outputs accept if the string pair is in the string-pair language, and a reject if it is not.
- **FST as generator** : a machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.
- **FST as translator** : a machine that reads a string and outputs another string. •

**FST as set relater** : a machine that computes relations between sets.

# Finite-State Transducers



Since both q1 and q2 are accepting states, regular nouns can have the plural suffix or not. The morpheme-boundary symbol ˆ and word-boundary marker # will be discussed below.

# Finite-State Transducers

This transducer will map plural nouns into the stem plus the morphological marker +PL, and singular nouns into the stem plus the morpheme +SG. Thus a surface cats will map to cat +N +PL as follows:

**c:c a:a t:t +N:e +PL:^s#**

# Finite-State Transducers

# Finite-State Transducers

## Orthographic Rules and Finite-State Transducers

Finite-State Transducers

# Orthographic Rules and Finite-State Transducers

something like "insert an *e* on the surface tape just when the lexical tape has a morpheme  ending in *x* (or *z*, etc.) and the next morpheme is *-s*. Here's a formalization of the rule:

# THE PORTER STEMMER

- THE PORTER

# *STEMMER*

# THE PORTER STEMMER

➢ **Stemming** is the process of producing morphological variants of a root/base word. ➢ Stemming programs are commonly referred to as stemming algorithms or stemmers.

➢ ***A stemming algorithm reduces the words*** "chocolates", "chocolatey", "Choco" to the root word, "chocolate" and "retrieval", "retrieved", "retrieves" reduce to the stem "retrieve".

- Stemming is an important part of the pipelining process in Natural language processing.
- ***The input to the stemmer is tokenized words***.
- How do we get these tokenized words? Well, tokenization involves breaking down the document into different words.

# THE PORTER STEMMER

- **The Porter Stemming algorithm (or Porter Stemmer**) is used to remove the suffixes from an English word and obtain its stem.
- This algorithm was developed by a British Computer Scientist named **Martin F. Porter.**

- Some more example of stemming for root word **"like"** include:

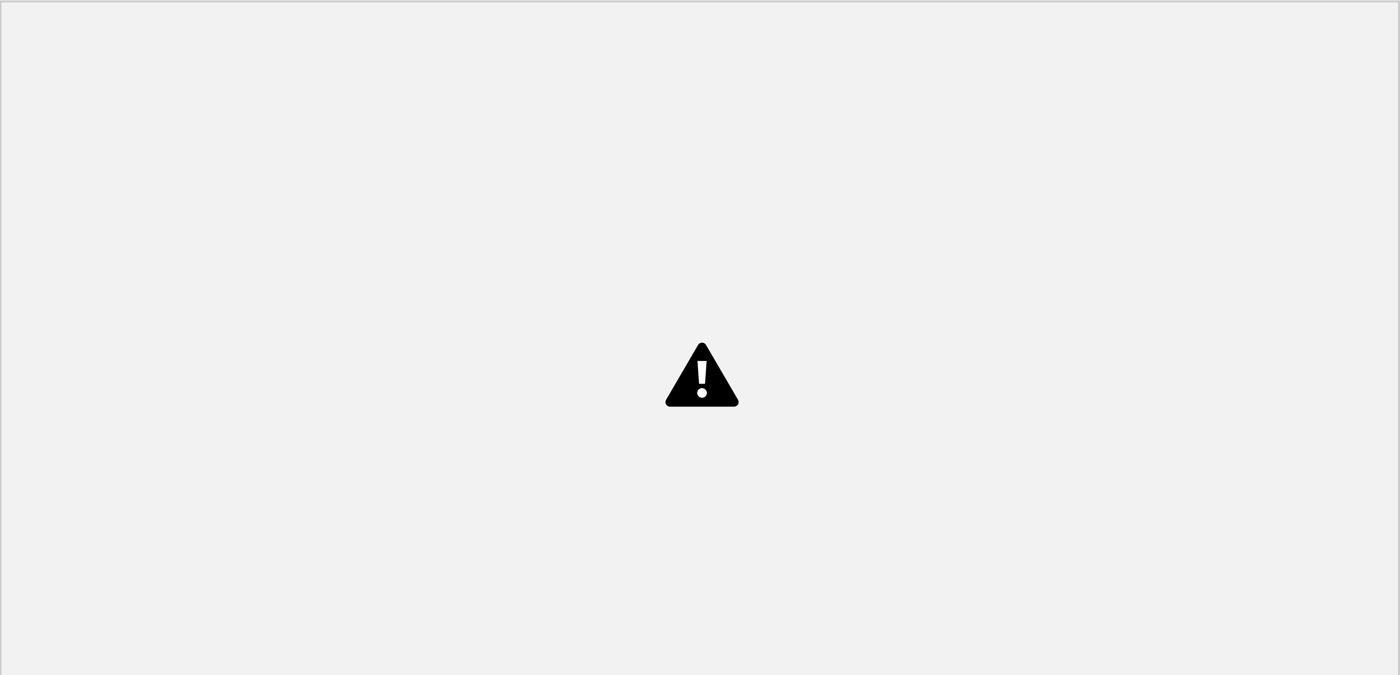
->"likes"

->"liked"

->"likely"

->"liking"

# THE PORTER STEMMER



**Vowels and consonants**

- Words are built from ***vowels (a, e, i, o, u)*** and ***consonants (the rest of the alphabet).***

- The letter 'y' is a bit different, because sometimes it acts as a consonant and sometimes it acts as a vowel. **$[C]\ (VC)^m\ [V]$.**

# THE PORTER STEMMER

**m=0 TR, EE, TREE, Y, BY.**

**m=1 TROUBLE, OATS, TREES, IVY.**

**m=2 TROUBLES, PRIVATE, OATEN, ORRERY.**

# THE PORTER STEMMER

# THE PORTER STEMMER

# THE PORTER STEMMER

# THE PORTER STEMMER

# THE PORTER STEMMER

# Tokenization

**Tokenization** in simple words is the process of splitting a phrase, sentence, paragraph, one or multiple text documents into smaller units. Each of these smaller units is called a **token**.

Tokenization

- A tokenizer breaks unstructured data and natural language text into chunks of information that can be considered as discrete elements.

- This immediately turns an unstructured string (text document) into a numerical data structure suitable for machine learning.

- Tokenization can separate sentences, words, characters, or subworlds. •

When we split the text into sentences, we call it sentence tokenization. •

For words, we call it word tokenization.

# Word-based tokenization

➢ It splits a piece of text into words **based on a delimiter**.

- The most commonly used delimiter is **space**.
- You can also split your text using more than one delimiter, like space and punctuation marks(full stop, comma, semicolon, colon).
- Word-based tokenization can be easily done using custom RegEx or Python's split() method. Apart from that, there are plenty of libraries in Python — NLTK, spaCy, Keras, Gensim, which can help you perform tokenization easily.
- Example:

**"Is it weird I don't like coffee?"**

By performing word-based tokenization with space as a delimiter, we get:

**["Is", "it", "weird", "I", "don't", "like", "coffee?"]**

# Word-based tokenization

# Sentence Tokenization

➢ This is similar to word tokenization. Here, we study the structure of sentences in the analysis. A sentence usually ends with a **full stop (.)**, so we can use "." as a separator to break the string. ➢ you need to count average words per sentence, how you will calculate? For accomplishing such a task, you need both **NLTK sentence tokenizer as well as NLTK word tokenizer** to calculate the ratio

# Detecting and Correcting Spelling Errors

**Detecting and Correcting Spelling Errors**

- The detection and correction of spelling errors is an integral part of **modern word processors.**
- The very same algorithms are also important in applications in which even the individual letters aren’t guaranteed to be accurately identified: optical character recognition (OCR) and on-line handwriting OCR recognition.
- Optical character recognition is the term used for automatic recognition of machine or hand-printed characters

# Detecting and Correcting Spelling Errors

**Kukich (1992),** in her survey article on spelling correction, breaks the field down into three increasingly broader problems:

- **non-word error detection:**
  detecting spelling errors which result in non-words (like graffe for giraffe).

- **isolated-word error correction**:
  correcting spelling errors which result in non-words, for example correcting graffe to giraffe, but looking only at the word in isolation.

- **context-dependent error detection and correction**: Using the context to help detect and correct spelling errors even if they accidentally result in an actual word of English (real-REALWORD word errors). which accidently produce a real word **(e.g. there for three),** or because the writer substituted the wrong spelling of a homophone or near-homophone (**e.g. dessert for desert, or piece for peace).**

# Detecting and Correcting Spelling Errors

## SPELLING ERROR PATTERNS

- Gruden (1983) found spelling error rates of between 1% and 3% in human typewritten text (this includes both non-word errors and real-word errors).
- The rate of spelling errors in handwritten text itself is similar; word error rates of between 1.5% and 2.5% have been reported (Kukich, 1992).
- Kukich (1992) breaks down human typing errors into two classes.
- **Typographic errors** ***(for example misspelling spell as speel),*** are generally related to the keyboard.
- **Cognitive errors** ***(for example misspelling separate as seperate)*** are

caused by writers who not know how to spell the word.

# Minimum Edit Distance

**Minimum Edit distance** between two strings str1 and str2 is defined as *the minimum number of insert/delete/substitute operations* required to transform str1 into str2.

For example

- if str1 = "ab", str2 = "abc"
- then making an insert operation of character 'c' on str1 transforms str1 into str2.
- Therefore, edit distance between str1 and str2 is 1.
- You can also calculate edit distance as number of operations required to transform str2 into str1.
- For above example, if we perform a delete operation of character 'c' on str2, it is

transformed into str1 resulting in same edit distance of 1.

# Minimum Edit Distance

if str1 = "INTENTION" and str2 = "EXECUTION", then the minimum edit distance between str1 and str2 turns out to be 5 as shown below. All operations are performed on str1.

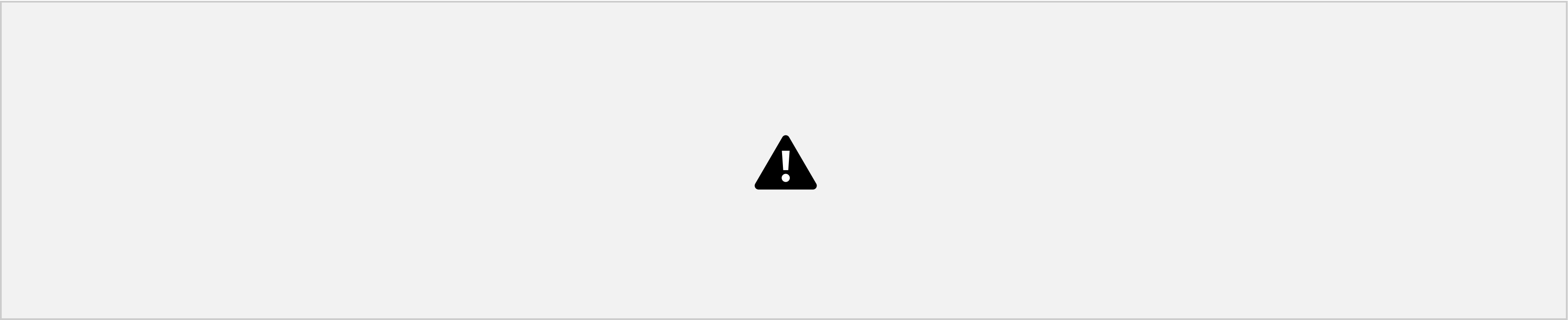
# Minimum Edit Distance

# Minimum Edit Distance

- The minimum edit distance is computed by **dynamic programming**.
- Dynamic programming algorithms for sequence comparison work by creating a distance matrix with *one column for each symbol in the target sequence and one row for each symbol in the source sequence* (i.e. target along the bottom, source along the side). • For minimum edit distance, this matrix is the **edit-distance matrix**.
- Each cell edit-distance[i,j] contains the distance between the first i characters of the target and the first j characters of the source.
- Each cell can be computed as a simple function of the surrounding cells; thus starting from the beginning of the matrix it is possible to fill in every entry.

# Human Morphological Processing

- *lifting* primed *lift*, and *burned* primed *burn*, but for example *selective* didn't prime *select*. Figure sketches one possible representation of their finding:



# Human Morphological Processing

• ***government***

primes ***govern***, but ***department*** does not prime ***depart***.

- human lexicon represents some morphological structure comes from **speech errors**, also called **slips of the tongue**.
- In normal conversation, speakers often mix up the order of the words or initial
- sounds:

if you **break** it it'll **drop**
I don't have time to **work** to watch television because I have to
**work**

# Human Morphological Processing

- it's not only us who have screw **looses** (for 'screws loose')
- words of rule formation (for 'rules of word formation') •

**easy enoughly** (for 'easily enough')

# What are n-grams?

- **N-grams** are continuous sequences of words or symbols or tokens in a document. •

In technical terms, they can be defined as the neighboring sequences of items in a document.

- They come into play when we deal with text data in NLP(Natural Language Processing) tasks

- ***How are n-grams classified?***

1. n’ in the term “n-grams”? **size(n)**

2. ’n’ is just a variable that can have positive integer values including 1,2,3 and so on 3. if n=1 we call it as Unigram, n=2 Bigram & n=3 Trigram and so on.

# What are n-grams?

- For example, for the sentence “The cow jumps over the moon”.
- If N=2 (known as bigrams), then the ngrams would be:

{the cow
cow jumps
jumps over
over the
the moon}

So you have 5 n-grams in this case.

- If N=3, the n-grams would be:

{the cow jumps
cow jumps over
jumps over the
over the moon}

So you have 4 n-grams in this case.

# What are n-grams?

- **Unigram Language Model Example**
- Let's say we want to determine the probability of the sentence, "***Which is the best car insurance package***". Based on Unigram language model, probability can be calculated as following:

**P("Which is best car insurance package")= P(which)P(is)…P(insurance)P(package)**

The probability of any word, … ng:

where *wi* is ith word,

*c(wi)* is count of *wi* in the corpus

*c(w)* is count of all the words.

# What are n-grams?

- **Bigram Language Model Example**

Let's say we want to determine the probability of the sentence, "***Which is the best car insurance package***". … the probability of any word given previous word, (wi/wi−1) can be calculated as following:

where *wi* is ith word,

*c(wi)* is count of *wi* in the corpus

*c(w)* is count of all the words.