Rule-based NLP and Regular Expressions

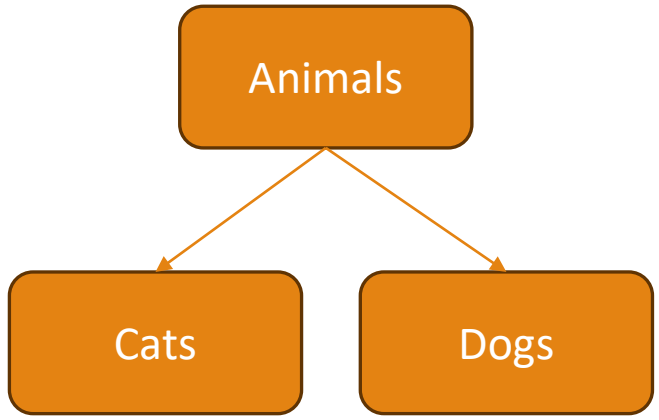
PREPARED BY: AHMAD ALAA ALDINE

PRESENTED BY: AHMAD ALAA ALDINE

What are Rule-based Approaches?

Rule-based NLP systems rely on handcrafted linguistic rules written by experts (linguists, programmers) to analyze and generate language.

"I like animals such as cats and dogs" →



How does it works?

1. Rule Creation
   - Experts create Rules
2. Rule Application
   - Apply rules to capture matching patterns
3. Rule Processing
   - Extract information from the matched patterns and take decisions
4. Rule Refinement
   - Refine rules to enhance accuracy and repeat again

Types of Rules in NLP

Lexical Rules

- Define how individual words are recognized and analyzed
- Rule: if a word ends with “ed”, it may be past tense

Syntactic Rules

- Define sentence structure (grammar and word order)
- Rule: Det + Adj + Noun → Noun Phrase

Semantic Rules

- Define how meaning is derived from words and structures
- Word-sense disambiguation
  - “Bank” = financial institution if near “money”, “loan”
  - “Bank” = river edge if near “water”, “shore”

Advantages

High Precision for well defined rules

Interpretable and Explainable

- Easy for human to understand and interpret results

No data Needed

Limitations

Not Scalable

- Difficult to create new rules

Labor Intensive

- Heavy work to create and refine rules

Low Recall

- Hard to create rules that cover all language variations leading to lower recall

Regular Expressions

Regular Expressions are a sequence of characters that define a search pattern.

They are used in NLP to search for specific patterns or structures in text data for:

- Data Cleaning
- Information Extraction

Regular expressions are highly expressive and can match many patterns, including numbers, dates, email addresses, and phone numbers.

Regular Expressions

Use strings to build patterns

```
import re

pattern = r'car'
text = r'''
    Here is the car of the future! This car is the opposite of todays' cars;
    today's cars are noisy and dirty, this one is silent and clean.
    Today's cars have wheels, this one does not. It moves like a snail, but much faster than a snail !
'''

result = re.findall(pattern, text)
print(result)
```

```
['car', 'car', 'car', 'car', 'car']
```

Regular Expressions

Use strings to build patterns

```
import re

pattern = r'car '
text = r'''
    Here is the car of the future! This car is the opposite of todays' cars;
    today's cars are noisy and dirty, this one is silent and clean.
    Today's cars have wheels, this one does not. It moves like a snail, but much faster than a snail !
'''

result = re.findall(pattern, text)
print(result)
```

```
['car ', 'car ']
```

Regular Expressions

Regular expressions are case sensitive

```
import re

pattern = r'Car '
text = r'''
    Here is the car of the future! This car is the opposite of todays' cars;
    today's cars are noisy and dirty, this one is silent and clean.
    Today's cars have wheels, this one does not. It moves like a snail, but much faster than a snail !
'''

result = re.findall(pattern, text)
print(result)
```

```
[]
```

Regular Expressions

Use special characters to build more complex patterns

Special characters

.	match any char except newline (eg., ac..ve)
^	match at beginning of string (eg., ^active)
$	match at end of string (eg, state$)
[3a-c]	match any char (ie., 3 or a or b or c)
[^x-z1]	match any char except x, y, z or 1
A|S	match either A or S regex
()	capture & match a group of chars (eg., (8097ba))
\	escape special characters

```
import re

pattern = r"car[ |s]"
text = r'''
    Here is the car of the future! This car is the opposite of todays' cars;
    today's cars are noisy and dirty, this one is silent and clean.
    Today's cars have wheels, this one does not.
    ]It moves like a snail, but much faster than a snail !
'''

result = re.findall(pattern, text)
print(result)
```

```
['car ', 'car ', 'cars', 'cars', 'cars']
```

Regular Expressions

To build more complex patterns

Special characters

.	match any char except newline (eg., ac..ve)
^	match at beginning of string (eg., ^active)
$	match at end of string (eg, state$)
[3a-c]	match any char (ie., 3 or a or b or c)
[^x-z1]	match any char except x, y, z or 1
A|S	match either A or S regex
()	capture & match a group of chars (eg., (8097ba))
\	escape special characters

```
import re

pattern = r"and (.....)[ |,|\.]"
text = r'''
    Here is the car of the future! This car is the opposite of todays' cars;
    today's cars are noisy and dirty, this one is silent and clean.
    Today's cars have wheels, this one does not.
    It moves like a snail, but much faster than a snail !
'''

result = re.findall(pattern, text)
print(result)
```

```
['dirty', 'clean']
```

Regular Expressions

To build more complex patterns

Special sequences

\d	match a digit
\D	match a non-digit
\s	match any whitespace char: [\t\n\r\f\v]
\S	match any non-whitespace char
\w	match any alphanumeric: [0-9a-zA-Z_]
\W	match any non-alphanumeric

```
import re

pattern = r"\d\d"
text = r'''
     However you probably won't ever drive a "snail car", even if you're under 20 today.
     This, perhaps, is the car of the year 2100, the car that your grandchildren will maybe drive.
'''

result = re.findall(pattern, text)
print(result)
```

```
['20', '21', '00']
```

Regular Expressions

To build more complex patterns

Quantifiers

*	match 0 or more occurrences (eg., py*)
+	match 1 or more occurrences (eg., py+)
?	match 0 or 1 occurrences (eg., py?)
{m}	match exactly m occurrences (eg., py{3})
{m,n}	match from m to n occurrences (eg., py{1,3})
{,n}	match from 0 to n occurrences (eg., py{,3})
{m,}	match m to infinite occurrences (eg., py{3,})

```
import re

pattern = r"\d+"
text = r'''
     However you probably won't ever drive a "snail car",
     even if you're under 20 today.
     This, perhaps, is the car of the year 2100,
     the car that your grandchildren will maybe drive.
'''

result = re.findall(pattern, text)
print(result)
```

```
['20', '2100']
```

Regular Expressions

To build more complex patterns

Quantifiers

*	match 0 or more occurrences (eg., py*)
+	match 1 or more occurrences (eg., py+)
?	match 0 or 1 occurrences (eg., py?)
{m}	match exactly m occurrences (eg., py{3})
{m,n}	match from m to n occurrences (eg., py{1,3})
{,n}	match from 0 to n occurrences (eg., py{,3})
{m,}	match m to infinite occurrences (eg., py{3,})

```
import re

pattern = r"\d{4}"
text = r'''
     However you probably won't ever drive a "snail car",
     even if you're under 20 today.
     This, perhaps, is the car of the year 2100,
     the car that your grandchildren will maybe drive.
'''

result = re.findall(pattern, text)
print(result)
```

```
['2100']
```

Regular Expressions

Example 1

Write a regex pattern to find email addresses:

Here is my email address: aalaaeddein@gmail.com

I have another email address: Test.2@gmail.com

Here is my Outlook email: Test_1994@outlook.gov

Regular Expressions

Solution

```
import re

pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
text = r'''
     Here is my email address: aalaaeddein@gmail.com
I have another email address: Test.2@gmail.com
Here is my Outlook email: Test_1994@outlook.gov
'''

result = re.findall(pattern, text)
print(result)
```

```
['aalaaeddein@gmail.com', 'Test.2@gmail.com', 'Test_1994@outlook.gov']
```

Regular Expressions

Example 2

Write a regex pattern to find Lebanese phone numbers:

Here is my phone number: 78888222

I have another number: 76 - 443 222

If you want to contact me from outside Lebanon use this: (+961) 71 321123

Regular Expressions

Solution

```
import re

pattern = r"\(?\+?\d*\)? *\d\d *-? *\d\d\d *\d\d\d"
text = r'''
     Here is my phone number: 78888222
I have another number: 76 - 443 222
If you want to contact from outside Lebanon use this: (+961) 71 321123
'''

result = re.findall(pattern, text)
print(result)
```

```
[' 78888222', ' 76 - 443 222', '(+961) 71 321123']
```