Introduction to NLP

- Aiswarya Ramachandran

Objective

- **NLP Applications**
- **Understanding Text**
    - Lexical, Syntactic and Semantic Analysis
- **Basic Lexical Analysis**
    - Word Frequencies
    - Text Preprocessing
    - Bag of Words
    - TF-IDF
- **Regular Expressions**
    - Different Quantifiers
    - Finding Pattern Matches
    - Replacing Pattern Matches

Introduction to NLP

- Blend of Language, Machine Learning and Artificial Intelligence
- Taking text to analyze and extract the related information in a format that can be used in decision-making capabilities
- Allows machines to understand human queries
- Allows Business to allow customer expectations
    - **“The quality of the battery and performance is very good, but the phone is heavy”**

How Business use NLP for Social Media Analytics

- **Pizza Hut**
  - Posted two images of pizza with white and black background
  - Monitored conversation around it
  - Black background shot was performing three times better
- **Zomato:**
  - #yummyyatra campaign live using twitter
  - Twitter and Facebook comments are monitored for customer response
  - Recommending restaurants
- **Competitor Intelligence**
- **Understanding reviews can help figure out what captures people's attention, why they bought a product and what makes them loyal to a brand**

Why are Customer Reviews Important

How Business use NLP for Social Media Analytics

- **Pizza Hut**
  - Posted two images of pizza with white and black background
  - Monitored conversation around it
  - Black background shot was performing three times better
- **Zomato:**
  - #yummyyatra campaign live using twitter
  - Twitter and Facebook comments are monitored for customer response
  - Recommending restaurants
- **Competitor Intelligence**
- **Understanding reviews can help figure out what captures people's attention, why they bought a product and what makes them loyal to a brand**

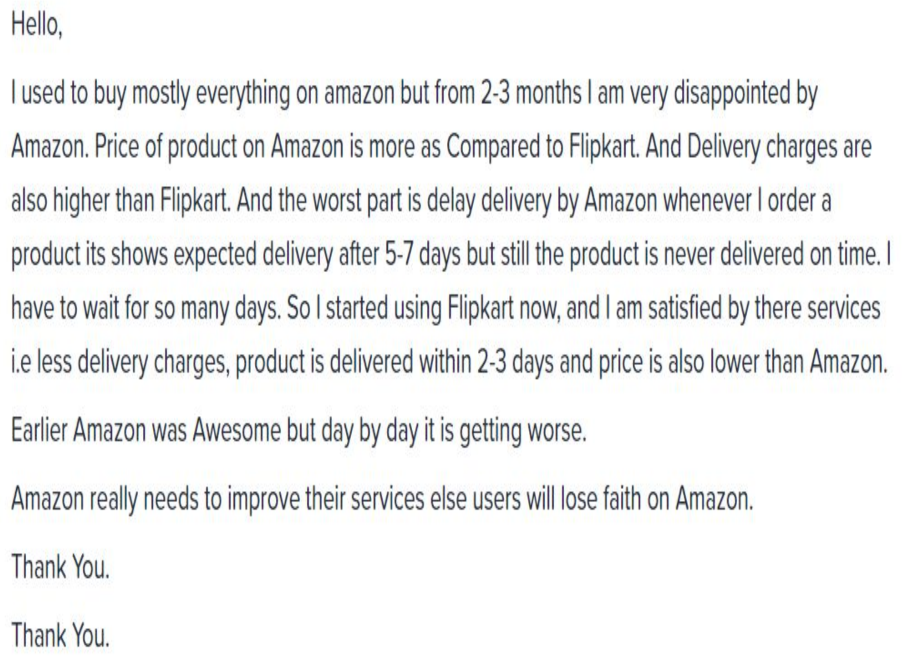
What makes NLP Hard?

- Cannot be defined with explicit rules
- Language is **ambiguous**
  - **“The man saw a girl with the telescope”**
- Same word can mean different things
  - **The man asked his friend to tie his tie**
- It is difficult for machines to understand sarcasm
- In customer reviews, people **tend to use multiple languages**.
  - The food was “Bakwas” but the ambience was awesome
- **Grammatical Errors** :
  - Word “Awesome” can be wriiten as awsum, awesommeeee,osm, awesoneee etc
- **Domain Specific Words**

Understanding Text

Imagine you want to summarise this review, what would you do?

- **Break text into sentences and words**
- **Identify the sentiment of each sentence**
- **Try understanding meaning of the words**



Hello,

I used to buy mostly everything on amazon but from 2-3 months I am very disappointed by Amazon. Price of product on Amazon is more as Compared to Flipkart. And Delivery charges are also higher than Flipkart. And the worst part is delay delivery by Amazon whenever I order a product its shows expected delivery after 5-7 days but still the product is never delivered on time. I have to wait for so many days. So I started using Flipkart now, and I am satisfied by there services i.e less delivery charges, product is delivered within 2-3 days and price is also lower than Amazon.

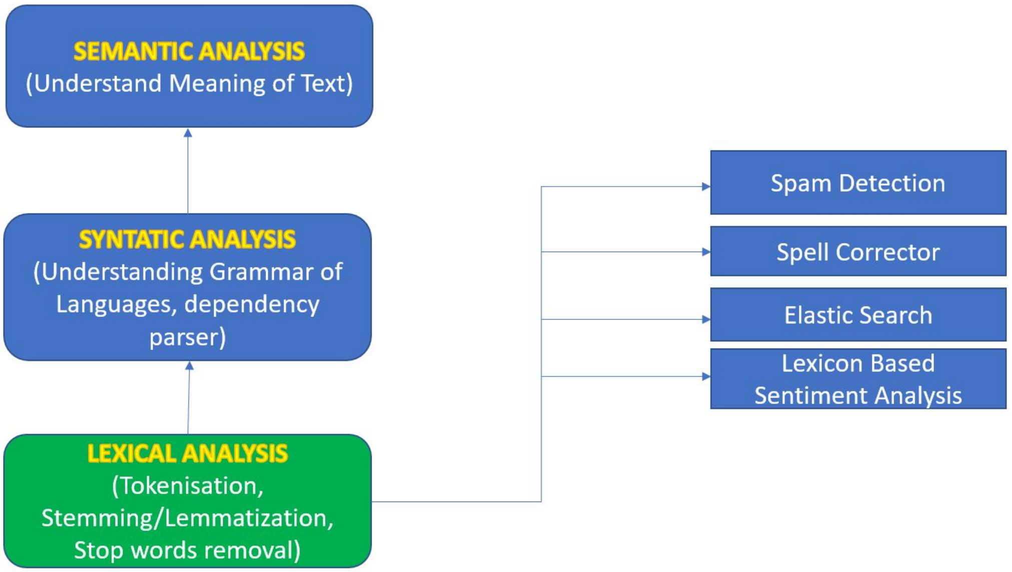
Earlier Amazon was Awesome but day by day it is getting worse.

Amazon really needs to improve their services else users will lose faith on Amazon.

Thank You.

Thank You.

Understanding Text – Lexical Analysis

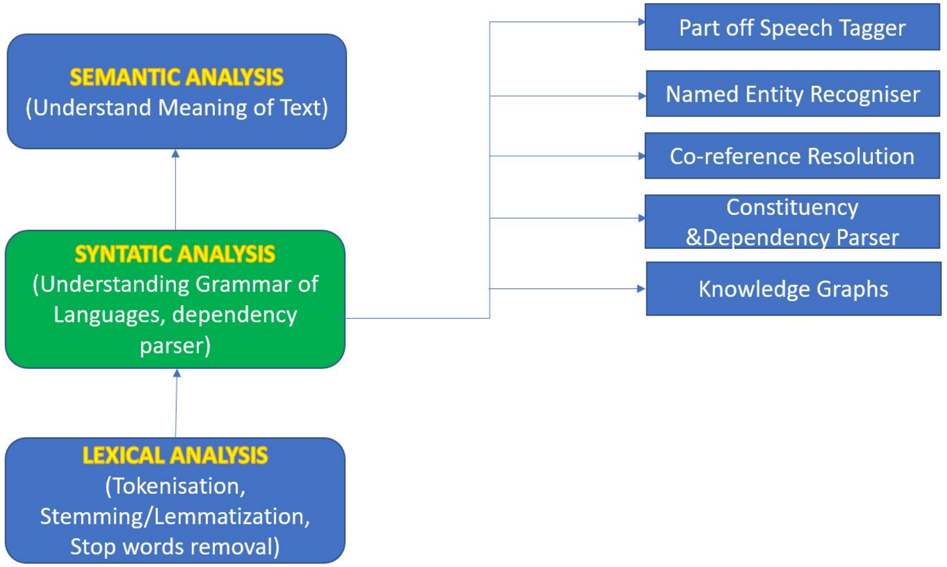

SEMANTIC ANALYSIS
(Understand Meaning of Text)
SYNTATIC ANALYSIS
(Understanding Grammar of Languages, dependency parser)
LEXICAL ANALYSIS
(Tokenisation, Stemming/Lemmatization, Stop words removal)
Spam Detection
Spell Corrector
Elastic Search
Lexicon Based Sentiment Analysis

- Break the document into sentences and words
- If a email contains words like "lucky, winner, prize , money" you know it is an spam
- The set of words in a document, gives an idea of what the document is all about
- Generally used as an preprocessing step

Cannot distinguish between

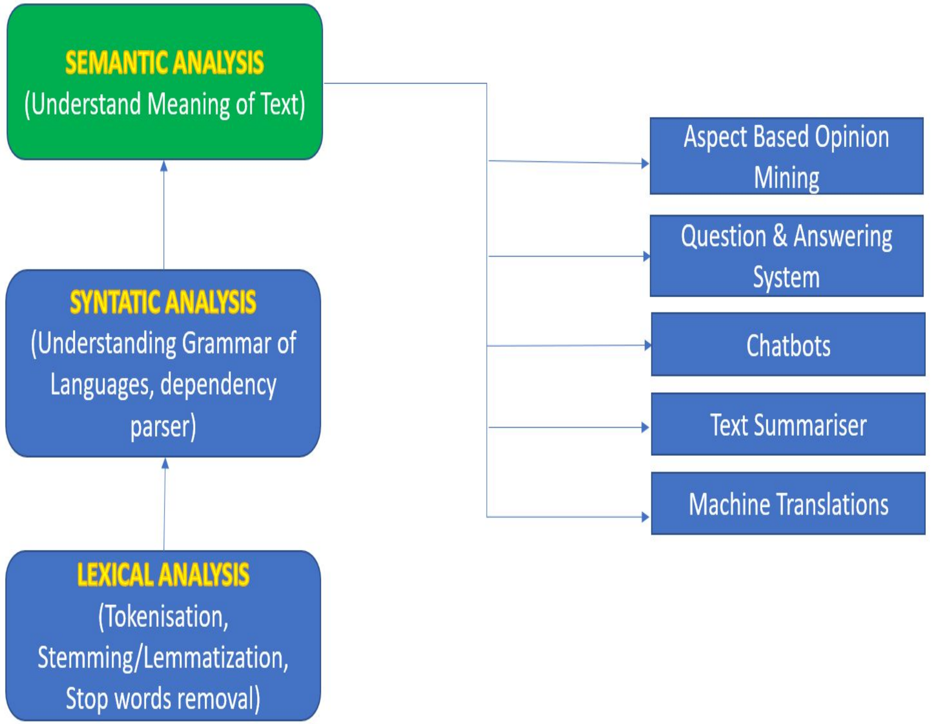
"Ram took money from Shyam" & "Shyam took money from Ram"

Understanding Text - Syntactic Analysis


SEMANTIC ANALYSIS
(Understand Meaning of Text)
SYNTATIC ANALYSIS
(Understanding Grammar of Languages, dependency parser)
LEXICAL ANALYSIS
(Tokenisation, Stemming/Lemmatization, Stop words removal)
Part off Speech Tagger
Named Entity Recogniser
Co-reference Resolution
Constituency &Dependency Parser
Knowledge Graphs

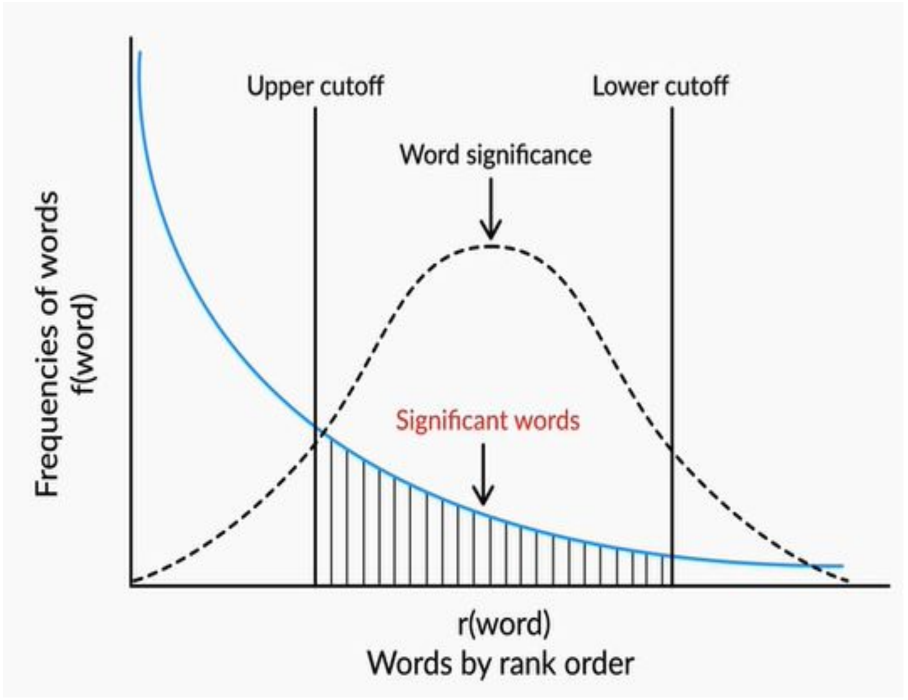
- Use syntax to extract more meaning from sentence
- Look at grammar of the sentence
- By looking at the subject and object this analysis can distinguish between “Ram took money from Shyam” & ‘Shyam took money from Ram”
- Knowledge Graph can learn relationship between words to answer questions
  - Like asking google for **Who is the PM of India?**
- For Flight Information applications, , understanding what is the source, destination and date of travel from user Query can help retrieve results from database

Understanding Text – Semantic Analysis


SEMANTIC ANALYSIS
(Understand Meaning of Text)
SYNTATIC ANALYSIS
(Understanding Grammar of Languages, dependency parser)
LEXICAL ANALYSIS
(Tokenisation, Stemming/Lemmatization, Stop words removal)
Aspect Based Opinion Mining
Question & Answering System
Chatbots
Text Summariser
Machine Translations

- After Syntactic Analysis, the machine may not be able to understand that PM and Prime Minister mean the same thing
- **“A word is known by the company it keeps”**
- To understand PM and Prime Minister mean the same thing, we can look at the words that occur with those words. If they have similar words occurring with each other , it means they have similar meaning
- Machines need to understand semantic relationships as well
    - King and Queen are related to each other

Basic Lexical Analysis – Word Frequencies



Zipf’s Law or Power Law

- Frequency of the word is inversely proportional to the rank of the word, where rank 1 is given to the most frequent word, rank 2 to the second most frequent word etc
- The words that are very frequent and appear almost is all documents are called “**stopwords**”
- Words like “the”,”a”,”an”,”is” are stopwords
- Towards the right end are the rarely occurring words
- In most analysis, we remove “stopwords” as a preprocessing step

Rationale behind removing Stopwords

- They provide **no useful information**, may it be in a spam detector or an search engine
- Removing stopwords r**educes the size** of the data significantly
  - Reduction of computation time
  - Lesser vocabulary or features to handle
- **Where do we not remove stopwords??**
  - POS Tagging
  - Parsing
  - In these applications stopwords provide meaningful grammatical information

Basic Lexical Analysis – Tokenization

- **Tokenization** - process used to break text into smaller elements
- The elements can be characters, words, sentences or paragraph based on the application
- NLTK package has various tokeniser like
  - Word Tokeniser
  - Sentence Tokeniser
  - Tweet Tokeniser
  - Regex Tokeniser
-

Basic Lexical Analysis - Stemming and Lemmatization

Consider the documents:

- My friends are **going** for a **movie**
- I love **going** to the **movies**
- Lets **go** to a movie

The word going and go are all the form of the same word "go"

Word "movies" is just a plural form of the word "movie"

Imagine, if we were to consider all forms of a word in our vocabulary - the size will become very very large and computation will become slower.

To represent various forms of the word into a single token we use Stemming or Lemmatization

Basic Lexical Analysis - Stemming

- Rule based technique that cuts of the suffix to get the root word
- **Example**:

  The driver is racing his boss's car

- Stemming will convert **driver to driv** and **racing to rac**

They don't resemble the exact root word. But, stemming will convert all forms of drive like driving, driven, drive to driv.

- **Porter Stemmer - works only on English words**
- **Snowball Stemmer - works on many language like English, French, German etc**

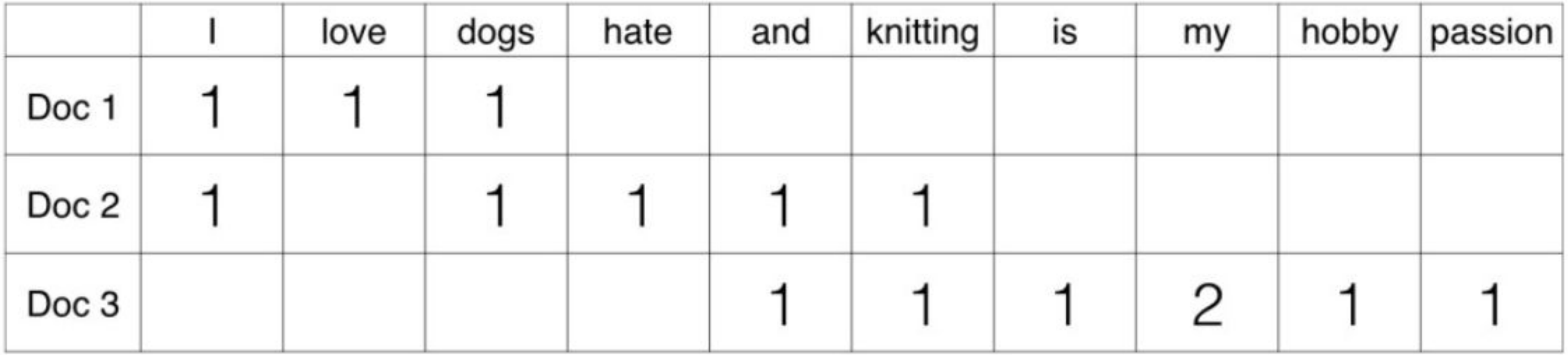
Basic Lexical Analysis - Lemmatization

- It searches for the base word by going through various variations of the dictionary word
- The base word is called the **lemma**
- **Lemmatizer takes word and POS tag associated with the word to come to the correct root form of the word**
- Stemmer cannot convert arose and bought to their root form while Lemmatization can
- Consider example:
  - I saw an man cutting a tree with a saw
- In the above example, a stemmer will convert both occurrences of “saw” to see, while Lemmatizer will convert the first “saw” to “see” and the second “saw” will remain as “saw”
- Lemmatization is more computationally expensive than an stemmer

Basic Lexical Analysis – Bag of Words

- Given any piece of text, they can be represented as a list of words that occur in it, where the **sequence of occurrence doesn't matter**

- **Consider the sentences:**
  - **I love dogs**
  - **I hate dogs and knitting**
  - **Knitting is my hobby and passion**



	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

Basic Lexical Analysis - TF-IDF

Consider a set of reviews all talking about "Iphone 11". Iphone is not a stopword, but in the set of reviews if we just go by the frequency of the word - iphone will have a higher importance.

To avoid this, instead of just going by raw frequency of a word in a document, we use TF-IDF

$$tf_{t,d} = \frac{frequency\ of\ term\ 't'\ in\ document\ 'd'}{total\ terms\ in\ document\ 'd'}$$

$$idf_t = log\frac{total\ number\ of\ documents}{total\ documents\ that\ have\ the\ term\ 't'}$$

$$tf - idf = tf_{t,d} * idf_t$$

- Higher weightage to words that are frequently present in a document, but rare across documents
- Low score for words that are common across documents

Introduction to Regular Expressions

- Powerful programming tool used for string manipulations, feature extraction from text and string replacement
- Regular Expression is an set of characters or **pattern** used to find substrings in a given string
- Example :
  - Extracting user mentions from Tweets
  - Extracting phone numbers from text
  - Extracting email id's from text
- **"re" package is used**
- To understand this lecture better please download the notebook from the below link:
  - https://jovian.ml/aiswaryasrinivas/regularexpressions

re.search()

- Used to find if substring present in string
- re.search(pattern,string) - returns a regex object if the pattern is found, else returns None

Practice

text=”The roots of education are bitter, but the fruit is sweet.”

- **Write a regular expression pattern to check whether the word ‘education’ is present in the given string or not.**
- **Extract the starting and ending position of the word “education”**

Quantifiers

- Quantifiers allow you to mention and control how many times you want a character(s) to occur in a pattern
- Consider a list [yummy, yummyyy, yummyyyyyy]
- Say, I want to extract the elements where there are more than one ‘y’ at the end of word yummy - Quantifiers are used in these scenarios
- 4 types of Quantifiers
    - ‘?’ operator
    - ‘*’ operator
    - ‘+’ operator
    - ‘{m.n}’ operator

“?” Quantifier

- Used when preceding character of your string is **optional**
- **For example:** if you want to match “dog” or “dogs” we can use this quantifier

  re.search(“dogs?”,”I love dogs”)

  re.search(“dogs?”,”I have a dog”)

Here **?** indicates that the character “s” is optional and hence will match both dog and dogs

Practice:

- **Write a regular expression that will match “know” or “knows” in a given string**

“*” Quantifier

- Matches the preceding character any number of times.

Practice

- **Match all strings that start with word “yumm” and end with any number of “y”s**
- **Match all strings that start with word “yum” followed by “m” any number of times and “y” any number of times**

“+” Quantifier

- Matches the preceding character any number of times.

Practice

- **Match all strings that start with word “yumm” and end with one or more “y”**
- **Match all strings that start with word “yu” followed by “m” one or more times and “y” any number of times**

{m,n} Quantifier

- Specify occurrence of preceding character a fixed number of times
- **{m,} :** Matches preceding character m or more times
- **{,n}:** Matches preceding character 0 to n times
- **{m,n} :** Matches preceding character m to n times
- **{n} :** matches preceding character exactly n times

Practice

- **Write a Regular expression to match word “awesome” where there are 2 or more “e” at the end and 3 to 6 “m” in the middle**
- **Write a regular expression where**

Other Operators

- **whitespace :** can be tab, single space or multi-space.
- **Parentheses :** say instead of looking for preceding character, you want to look for a set of characters that occur one or more times . A parentheses is used in this case

  Example: **(abc){1,3}   will match abc, abcabc and abcabcabc**

- **Pipe operator (|): This is a "OR" operator**
- **Example:** say you can write awesome as "awesome" or "awesume", then regular expression will be **awes(u|e)me**

Practice

- **Write a regular expression which matches where '23' occurs one or more times followed by 56 three to five times**

How to use the special characters like “?”, “+” in Regular Expressions

- If i want to find out all strings which have “?” , since “?” is an quantifier, we have to use “\?” .
- The “\” is called the “**escape sequence**”
- To match “\” you need to use “\\”

Practice

- **Sachin scored 100(98) in the match with South Africa**

  Extract (98) from the string

Regex Flags

- **re.I** : allows you to ignore the case of the text.
- **re.M**: allows you to search in multi lines
- Example: if i want to match the word “New Delhi” and “new dElHi”

  re.search(“new delhi”,text,flag=re.I)

- Multiple flags can be passed using the “|” operator

Anchors

- Anchors specify start and end of a string
- “^” specify the start of a string
  - The character followed by “^” should be the first character of the string
- “$” specify end of string
  - The character before a “$” should be the last character of the string

Practice

- **Write a regular expression to match a string that ends with “all”**

Wildcards

- “.” (dot) can match any character and is known as the wildcard character

Practice

- **Match any string that contains “some”**

Character Sets

- If you want to match a phone number, you are not looking for any particular character but looking for an expression that contains only digits and no numbers
- For such cases Character Sets are used
- Character sets can be specified with or without quantifiers - if no quantifier matches only one character
- Character Sets are represented within a []
- [a-z] - represents all lower case alphabets
- [0-9] represents all digits
- **Example** : If i want to match all three letter words that end with "ed"
  - **[a-z]ed**
- **If I want to match anything other than [a-z], we will use the "^" within the character set**
- **[^a-z] will match any character other than lower case alphabet**

Character Sets

- Quantifier losses its meaning within a character set
- [?a-z] will match any lowercase character and a ?

Practice

- **Match any string that contains a ten digit phone number**
- **Match any string that contains only alphabets and ends with “ed”**

Meta Sequences

- Shorthand to write character sets
- \w+ or [\w+] will match any alphanumeric character
- [\w+\s+] will match any alphanumeric character and space
- The character sets and meta sequences can be found in the jupyter notebook

Practice

- **Match a string that starts with any number, followed by space and ends with any alphabets**

Greedy vs Non-Greedy Approach

- By default Regular Expression looks for the longest possible pattern in the string
  - If looking for a pattern 30{3,5} and your text is 300000, it will return 300000 as match
- If you want the regex to stop once a particular condition is satisfied, then we use a non-greedy approach
  - In the non-greedy approach 30{3,5} will return 30000
- To use the non-greedy approach use ? after the quantifiers

Practice

- **What does the greedy and non-greedy approach return for pattern bat* on word batsmen**

Commonly used re functions

- **re.match(pattern,string) :** it will match only if the pattern is present in the very beginning of a string
- **re.search(pattern,string) :** scan through a string, looking for any location where this pattern matches
- **re.findall(pattern,string) :** returns a list of substring where the pattern matches
- **re.finditer(pattern,string):** returns an iter object of substring where the pattern matches
- **re.sub(pattern,replacement_string,string):** finds all the substrings where a pattern matches and replaces them with the replacement_string.

Grouping

- Extract **sub-patterns out of a large pattern**
- Example : **Extracting year from dates**
- **group() function is used for getting the year.**

- **Consider string :**

  Mahatma Gandhi was born on 2-10-1869

  To extract date from this string, the regular expression would be

  “\d{1,2}-\d{1,2}-\d{4}”

  To extract just the year, you can put () around the year regular expression. So it becomes

  “\d{1,2}-\d{1,2}-(\d{4})”

  If you want to extract the month and date too seperately, put **()** around each of the sub-string

Homework

- Use Regular Expressions to extract all hashtags and user mentions from Twitter data on Article 370

  The dataset can be found at : https://www.kaggle.com/aiswaryaramachandran/twitter-data-on-abrogation-of-article-370

Resources

- https://biznology.com/2016/08/12-inspiring-social-media-monitoring-case-studies/
- https://marketing.twitter.com/na/en/insights/twitter-transforms-conversations-between-companies-and-customers
- https://www.nextiva.com/blog/importance-of-customer-reviews.html
- https://www.managerskills.org/hospitality/social-media-impact/
- https://medium.com/@AI_with_Kain/why-nlp-is-hard-for-reviews-1730d6fd9f10