TEXAS
The University of Texas at Austin

I320D – Topics in Human Centered Data Science

Text Mining and NLP Essentials

Week 3: Finite State Machines, Regular Expressions, preprocessing raw text data, Morphology Analysis,

Dr. Abhijit Mishra

Week 2: Recap

- Lecture topics:
  - Fundamentals layers of NLP,
  - Applications and Ambiguity
  - Multilingualism and its importance in the context of NLP
  - Overview of text corpora and datasets
- Python Tutorial:
  - Reading and manipulating PDF text
  - Regular Expressions – basics

Before we start …

- Assignment 1 (due this week) clarifications
  - Question (McKenna) : what do we mean by frequency analysis of unique words and two-word phrases for each sentiment category?
- Example:
  - {“Text” : “the movie was a good movie”, “Label” : 1}
  - {“Text” : “the play was bad”, “Label” : 2}
  - {“Text” : “the acting was OK”, “Label” : 0}
- Expected output (after cleaning and removing redundant words)
  - freq_dict_1 = {“movie”: 2, “good” : 1, “movie good”: 1, “good movie”: 1}
  - freq_dict_2 = {“play”: 2, “bad” : 1, “play bad”: 1}
  - freq_dict_3 = {“acting”: 1, “OK” : 1, “acting OK”: 1}

Also …

- General observations / guidelines reg assignments
  - Acknowledge and give attributions when you copy code (you should not be copying code in the first place, rather reimplementing)
    - Else would result in plagiarism
  - Submit in time / seek extensions “explicitly” if you are struggling

Week 2: Roadmap

- Lecture:
  - Regular Expressions and Finite State Automata
  - Text Pre-processing Techniques – cleaning text, normalization, stop word removal
    - Morphological Analysis – stemming , lemmatization
    - When to apply which operations
- Tutorial:
  - Text pre-processing using NLTK and SpaCy libraries

W3 Reference on Canvas

References:

[1] Chapter 3. Words and Transducers, Book: Jurafsky, D., & Manning, C. (2012). Natural language processing. Instructor, 212(998), 3482.

[2] https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/

[3] Text Preprocessing: NLP fundamentals with spaCy https://medium.com/eni-digitalks/text-preprocessing-nlp-fundamentals-with-spacy-54f32e520bc8

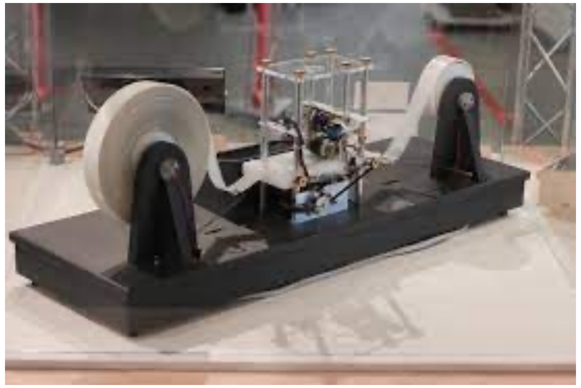
Part 1: Regular Expressions and Finite State Automata

What is an Automaton ?

- An automaton is a mathematical model that represents a system or a machine that can transition through a finite set of states based on inputs.
- Involves using these models to automate certain computations, recognizing patterns, or solving specific problems.

Three Types of Automata

- **Finite State Automata (FSA):**
  - Automation is achieved through finite state machines, where the system transitions between a finite set of states based on input symbols.
  - FSAs are often used to recognize regular languages.
- **Pushdown Automata (PDA):**
  - Extend the concept of finite state machines by introducing a stack
  - PDAs are associated with context-free languages.
- **Turing Machines:**
  - Represent a more powerful form of automation
  - Consist of an infinite tape and a read/write head
  - Associated with recursively enumerable languages



Regular Languages and Expressions

- A **regular language** is a type of language that can be recognized by a finite state machine or described by a regular expression.
  - No nesting, recursive operations allowed
- **Regular Expressions:**
  - Sequence of characters that defines a search pattern
  - Simple and well-defined structures, no nesting / recursion
  - Widely used in text processing, searching, and manipulating strings in various programming languages and tools

Basic Elements of Regular Expressions

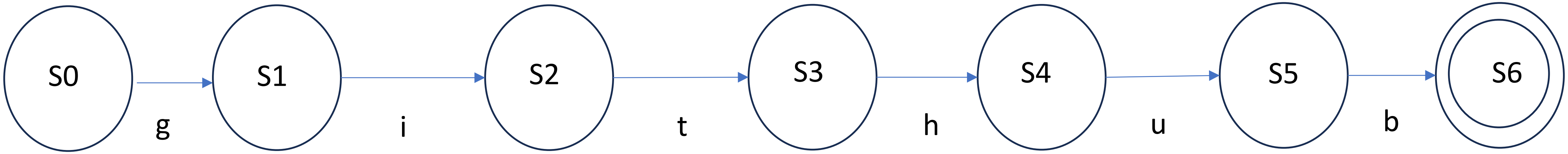
1. **Literals:** Characters that match themselves.
   e.g., the regular expression **abc** matches the string "abc" exactly.
2. **Alternation (|):** Represents a choice between alternatives.
   e.g., a|b matches either "a" or "b".
3. **Concatenation:** Represents the concatenation of two expressions.
   e.g., **ab** matches the string formed by concatenating "a" and "b".
4. **Kleene Star (*):** Represents zero or more occurrences of the preceding expression.
   e.g., **a*** matches zero or more occurrences of the character "a".
5. **Plus (+): Like "*" but** represents ONE or more occurrences
6. **Start and end symbols: "^"** is start and "$" is end

Representation of RegEx in FSA

- States: {S0, S1, S2, …. S_N }
- Alphabet: {set of allowed characters}
  - E.g., {A, B, C, …, a, b, c, …, 0, 1, 2, …}
- Initial State: S0
- Accepting State: S2 (marked by doubled circles
- Transitions:
  - Arrows connecting states
- **Matching criteria:** Final state is reached upon processing of all inputs
- **Non-Matching criteria:** Final state is NOT reached upon processing of all inputs

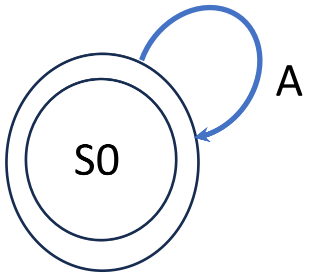
Basic Example(s)

- Represent “github” in regular expression
  - exp = r“github”
  - Won’t match anything except “github”



Basic Example(s)

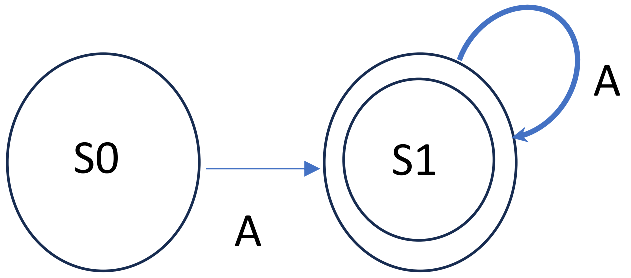
- Represent a sequence of **A**s any number of **A**s (also allow *None*)
    - exp = r "A*"
    - State diagram (FSA)



- Represent a bit sequence of **A**s any number of **A**s (at least one A should be present)

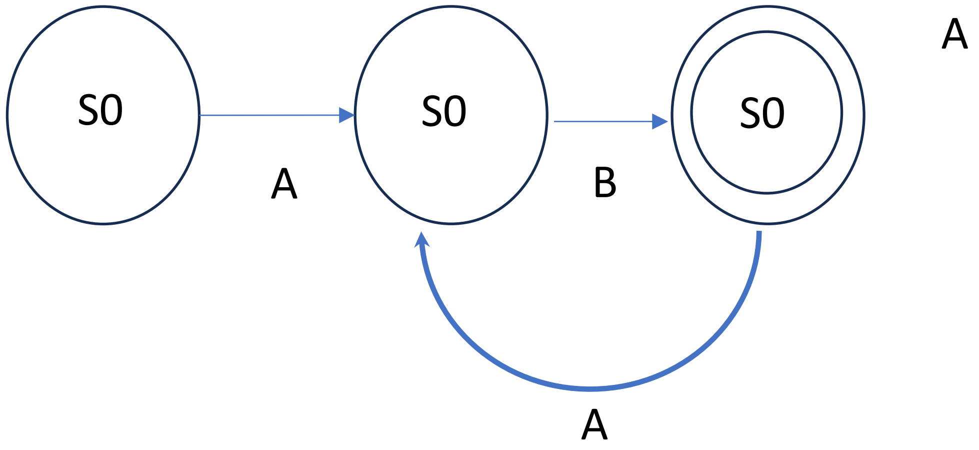
Basic Example(s)

- Represent a sequence of **A**s any number of **A**s (*at least 1 A*)
  - exp = r "A+"
  - State diagram (FSA)



Basic Example(s)

- Represent ABABABAB……. (at least ONE “AB”) :
  - exp = r “(AB)+”
  - State diagram (FSA)



Exercise: why won’t it accept ABBB ?

Exercise

- Design RegEx for any sequence of “ABs” with odd number of ABs

Exercise (Take home)

- Design RegEx for any sequence of “A”s and “B”s with odd number of “B”s

Some other details about RegEx and programing languages

- Regular Expressions are Programming language independent
- However, syntax for handling matches, grouping words ***etc. can vary***
  - ***E.g,*** *findall( ) is specific to Python, Java and JavaScript*
- Some languages allow shortening/truncation of expressions
  - E.g., [A-Za-z0-9] = (A | B | C | … | Z|a|b|…|z|0|1|2|…|9) = “\w”

Part 2: Text pre-processing

What is Text Pre-processing?

- Text pre-processing in NLP involves cleaning and transforming raw text data into a format that is suitable for analysis or modeling
- Main goal:
  - Enhance the quality of textual data by addressing various challenges that arise due to data sparsity (non-matching) in tasks such as search and text classification

NLP – Two "Main" Tasks

- **Text Classification**
- **Search**

- *Both are affected by "data sparsity problem"*

* Text generation is a special case of text classification

The Problem of Data Sparsity

- Data sparsity in NLP refers to the situation where the available data is insufficient or incomplete to effectively cover the entire linguistic space

Sparsity in Search

- Consider hypothetical example in healthcare domain

Document 1: "Common_1 Cold Symptms and Treatments"
Document 2: "Understanding ¶ Diabetes: Causes and Management"
Document 3: "Healthy Living Tips for Everyone"
Document 4: "Cardiovascular Diseaseshttps://www.webmd.com : A Comprehensive Guide"
Document 5: "An Overview of Respiratory Conditions"
Document 6: "Managing Chronic Illnesses in Adults"
Document 7: "Preventing Infectious Diseases: Best Practices"
Document 8: "Mental Health Awareness and Support"
Document 9: "Types of Allergies and How to Manage Them"
Document 10: "Rare Disordersÿ in Medicine: A Closer Look at XYZ Disorder"

User's query
"Common-cold Symptoms"

NO-match

Sparsity in Search

- Consider hypothetical example in healthcare domain

Document 1: "Common_1 Cold Symptms and Treatments"
Document 2: "Understanding ¶ Diabetes: Causes and Management"
Document 3: "Healthy Living Tips for Everyone"
Document 4: "Cardiovascular Diseaseshttps://www.webmd.com : A Comprehensive Guide"
Document 5: "An Overview of Respiratory Conditions"
Document 6: "Managing Chronic Illnesses in Adults"
Document 7: "Preventing Infectious Diseases: Best Practices"
Document 8: "Mental Health Awareness and Support"
Document 9: "Types of Allergies and How to Manage Them"
Document 10: "Rare Disordersÿ in Medicine: A Closer Look at XYZ Disorder"

User's query
"Cardio Disease"

NO-match

Data Sparsity in Classification

- Say we train a Machine Learning based text classifier to classify a text into topic categories

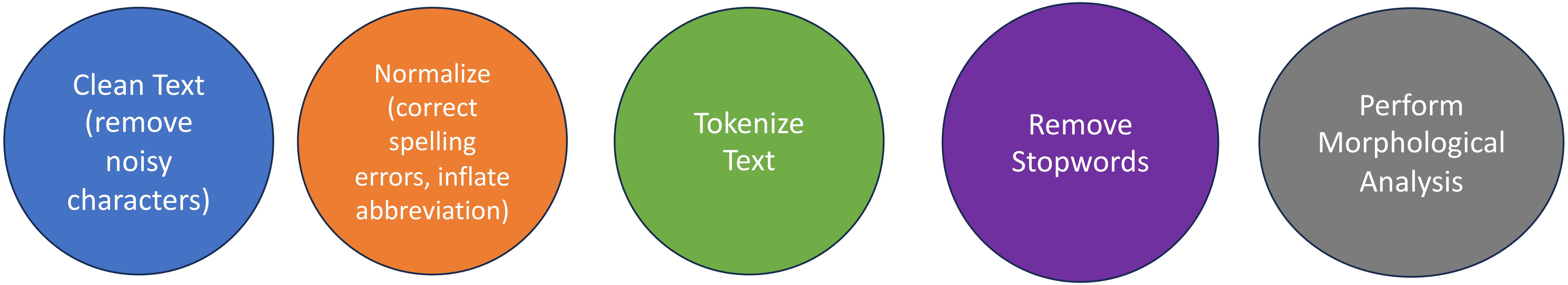
Text	Category
"Team A Wins the Championship¶ Title"	Sports
"Football World Cuphttp://sports.com : Exciting Matches Ahead"	Sports
"New Smartphone Released with Advanced Features"	Technology
"Artificial Intelligence Transforming Industries"	Technology
"Breakthrough in Cancer Research: Promising Results"	Health
"Healthy Eating Habits for a Better Lifestyle"	Health
"Movie Review: Blockbuster Hits the Theaters"	Entertainment
"Upcoming Music¶ Festival to Feature Top Artists"	Entertainment
"Election Results: New Government Takes Office"	Politics
"Debates on Economic Policies in Parliament"	Politics

Test input:
"TeamA becomes champion"
Predicted Label: None

Test input:
"Phone with advanced tech"
Predicted Label: None

Out of vocabulary words

Text Pre-processing for Data Sparsity Reduction



Step 1: Cleaning Text

- Involves:
  - Removing unwanted characters (e.g., printable or non-printable UNICODE characters such as “¶”)
  - Removing spurious entries, URLs, Hashtags (in tweets), Mentions

- Additional steps (optional and task dependent)
  - Lowercasing of Roman Characters to ensure better matching

Step 2: Text Normalization

- Involves spell checking and spell correction
  - E.g., Common_1 Cold Symptms (=>Symptoms) and Treatments
- Normalizing repeated characters
  - E.g., “I loooooved the movie” => “I loved the movie”
- Techniques used:
  - Dictionary based character sequence matching
  - Machine learning based text classification for text normalization

Example spell checker - HunSpell

```
from hunspell import Hunspell

# Load the English dictionary
hunspell = Hunspell('en_US')

# Check if a word is spelled correctly
if hunspell.spell('example'):
    print("The word is spelled correctly.")
else:
    # Get suggestions for corrections
    suggestions = hunspell.suggest('example')
    print(f"The word is misspelled. Suggestions: {suggestions}")
```

Step 3: Tokenization

- Act of extracting tokens from text
- Basic tokenization: “white space” based splitting
- Advanced tokenization : Consider punctuations

Input Text: "I bought apples, oranges, and bananas."

- **Tokens:** ["I", "bought", "apples", ",", "oranges", ",", "and", "bananas", "."]

Input Text: "She said, 'I'll be there at 3:00.'"

- **Tokens:** ["She", "said", ",", "'", "I'll", "be", "there", "at", "3:00", ".", "'"]

- Mostly rule/pattern based
- Sentence tokenization: Extracting sentences from documents

Importance of Tokenization: non-space delimited languages

1. **Input Text:** "你好，你在做什么？"
   - **Tokens:** ["你好", "，", "你", "在", "做", "什么", "？"]
   - **Gloss:** ["Hello", ",", "you", "at", "do", "what", "?"]
   - **Translation:** "Hello, what are you doing?"
2. **Input Text:** "我喜欢吃中餐和日餐。"
   - **Tokens:** ["我", "喜欢", "吃", "中餐", "和", "日餐", "。"]
   - **Gloss:** ["I", "like", "to eat", "Chinese food", "and", "Japanese food", "."]
   - **Translation:** "I like to eat Chinese and Japanese food."

Advanced tokenization

- Word pieces: involve breaking down words into smaller units called subword tokens (e.g., "unhappiness" => ["un", "##happiness"])
  - Unsupervised machine learning for splitting words using byte level information
- Sentence pieces: sentencepieces consider smaller segments that represent meaningful parts of the text
  - Unsupervised machine learning for splitting words using byte level information
- Will be discussed more in week 9

Step 4: Removing stopwords

- Certain words are redundant
- Typically function words (i.e., prepositions, articles, *etc.)*
- Sometimes removing such words after tokenization and lemmatization helps reduce vocabulary

```
import nltk
from nltk.corpus import stopwords

stops = set(stopwords.words('english'))
print(stops)
```

```
stops = set(stopwords.words('german'))
stops = set(stopwords.words('indonesia'))
stops = set(stopwords.words('portuguese'))
stops = set(stopwords.words('spanish'))
```

Step 5: Morphological Analysis

- Segment and identify
- **Science goal:** Study language and complexity of words in a language (e.g., how compounding happens in a language)
- **Engineering:** segment and identify all constituents of a word so as to reduce "data sparsity"

Morphological Analysis (1)

- German Word: "Staubsaugerbeutel"
- Meaning : "Vacuum cleaner bag"
- After morphological analysis
  - "Staub" (Dust)
  - "sauger" (Sucker)
  - "Beutel" (Bag)
- Reduces space requirements to store complex words

Morphological Analysis (2)

- Two types of analysis: **stemming** and **lemmatzation**
- **Stemming :** reducing words to their base or root form by removing suffixes or prefixes

1. **Original:** jumping
   - **Stem:** jump
2. **Original:** walked
   - **Stem:** walk
3. **Original:** apples
   - **Stem:** appl (Note: Stemming might not always result in valid words.)
4. **Original:** swimming
   - **Stem:** swim

Morphological Analysis (2)

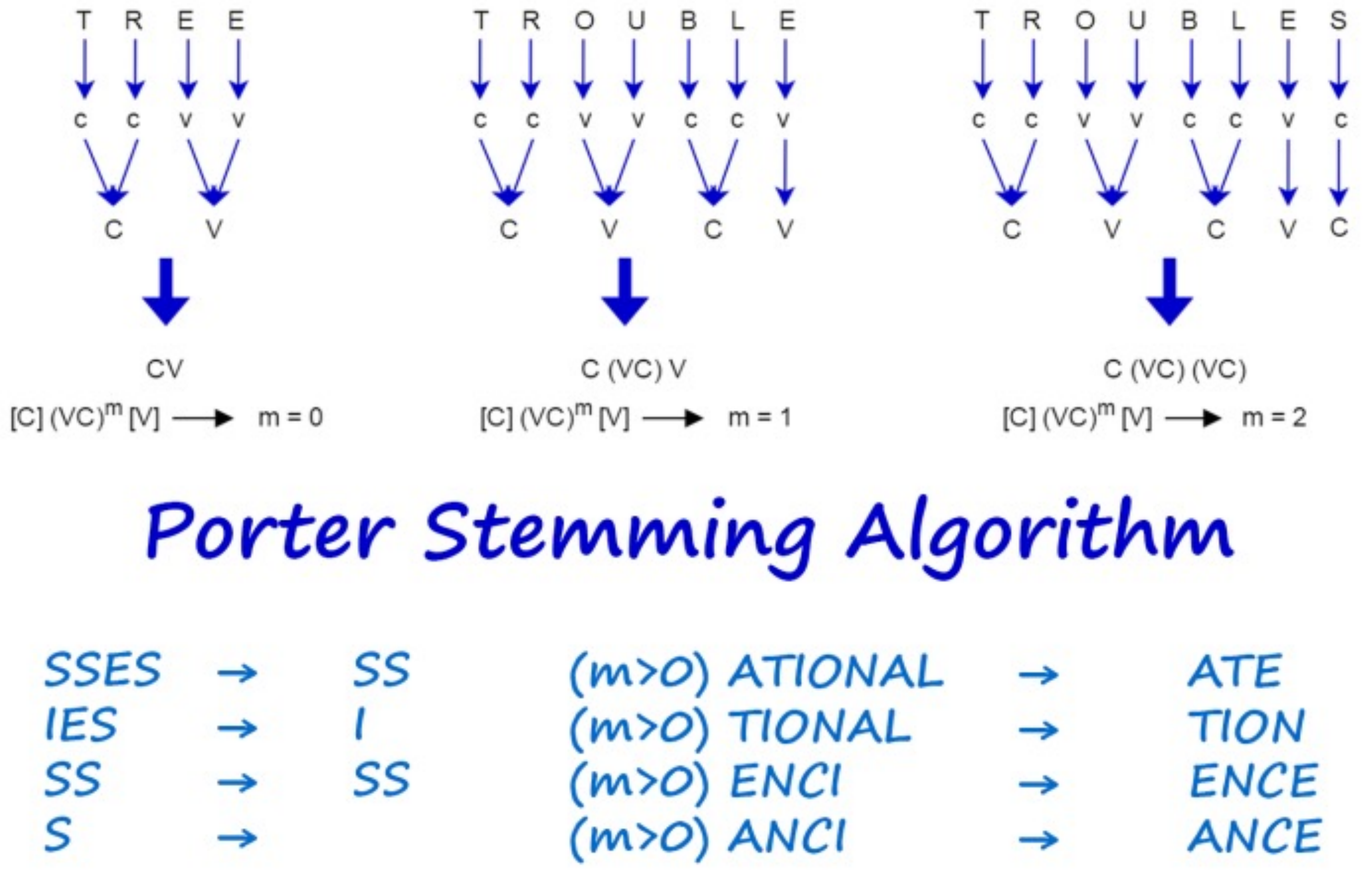
- Two types of analysis: **stemming** and **lemmatization**
- **Lemmatization:** reducing words to their canonical or dictionary form (a.k.a **lemmas**)

1. **Original:** jumping
   - **Lemma:** jump
2. **Original:** walked
   - **Lemma:** walk
3. **Original:** apples
   - **Lemma:** apple
4. **Original:** swimming
   - **Lemma:** swim

Stemming

- Comparatively less resource intensive
- Built by looking at patterns in character sequences in corpora
- Rules are pre-determined to split words into stems

Stemming Example: Porter Stemmer



https://vijinimallawaarachchi.com/2017/05/09/porter-stemming-algorithm/

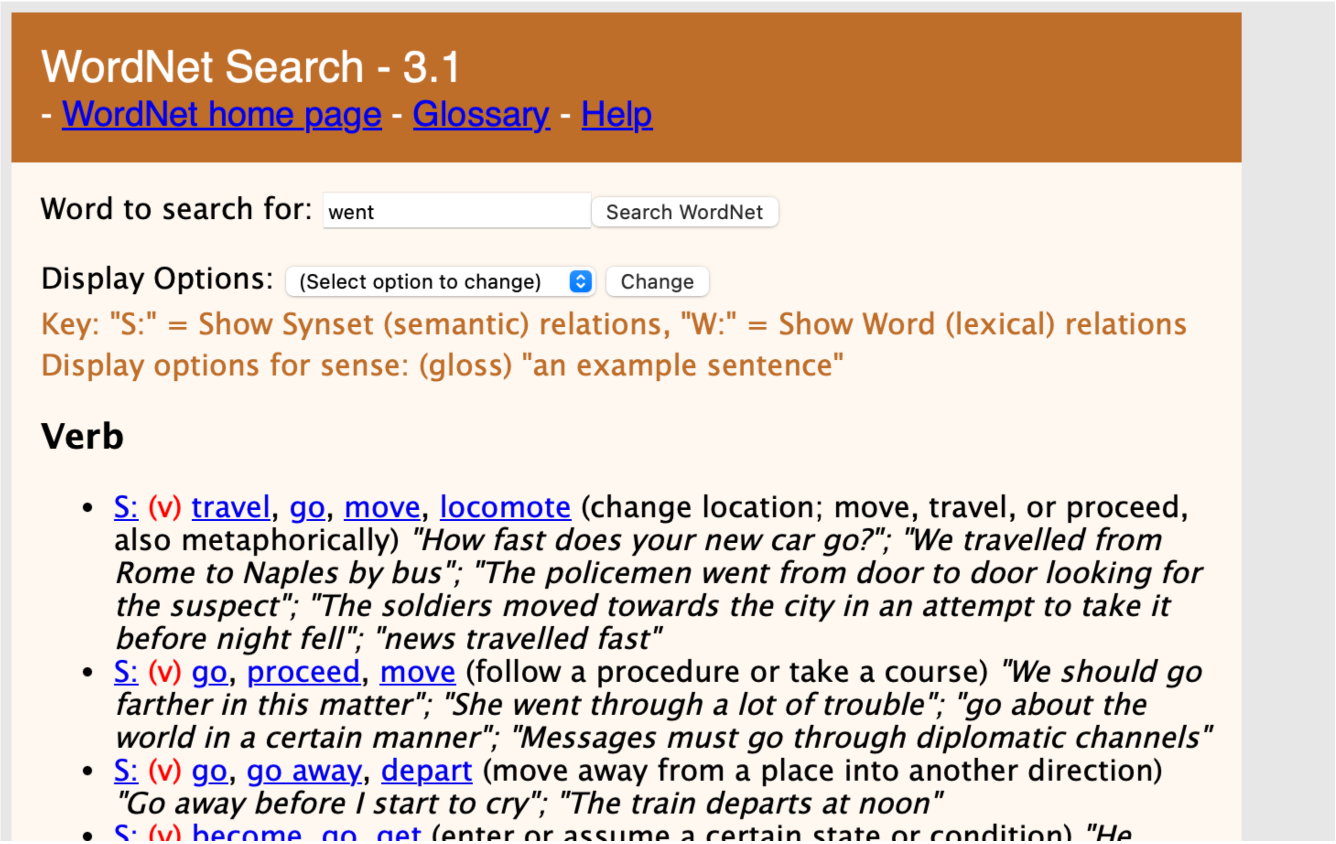
Lemmatization

- More resource intensive: requires some sort of a dictionary for valid word identification
- E.g., “went” => “go” (requires a dictionary)

Lemmatization Example: Wordnet Lemmatizer

```
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

# Create a lemmatizer object
lemmatizer = WordNetLemmatizer()

# Example words to lemmatize
words_to_lemmatize = ["running", "better", "cats", "ate", "happily"]

# Lemmatize each word
lemmatized_words = [lemmatizer.lemmatize(word, pos='v') for word in
words_to_lemmatize]

# Print the original and lemmatized words
for original, lemmatized in zip(words_to_lemmatize, lemmatized_words):
    print(f"Original: {original}, Lemmatized: {lemmatized}")
```

Original: running, Lemmatized: run
Original: better, Lemmatized: better
Original: cats, Lemmatized: cat
Original: ate, Lemmatized: eat
Original: happily, Lemmatized: happily



When to use stemming and when to use lemmatization

- Stemming: Shallow tasks (do not require understanding meaning), e.g., Information retrieval
- Lemmatization: Tasks requiring meaning analysis
  - Example?

Summary

- We discussed
  - Regular Expression basics and basics of Finite State Automata
  - Text Preprocessing techniques
- Choosing appropriate text pre-processing steps is IMPORTANT
  - E.g., Stopword removal won't benefit a text –translation system
  - E.g., Stemming won't benefit a deep semantic-analysis based task such as summarization

Next Class

- Text pre-processing using NLTK and SpaCy modules

Assignment 2 (to be posted on Thursday) : Text pre-processing + Search