# Text Processing

Dr. Demetrios Glinos
University of Central Florida

CAP6640 – Computer Understanding of Natural Language

# Today

- Regular Expressions
- Word Tokenization
- Word Normalization and Stemming
- Sentence Segmentation and Decision Trees

# Build an NLP Pipeline

- NLP programs are often built as pipelines of components that act as filters or transducers

- Typical components

  1. Sentence splitting
  2. Tokenization (optional: stopword removal)
  3. Lemmatization
  4. Part-of-speech tagging
  5. Named entity recognition
  6. Parsing (constituency or dependency)
  7. Coreference resolution
  8. Semantic analysis (sentiment, QA, summarization, etc.)

# Regular Expressions

- A formal language for matching text strings

- We can use a single RegEx to match any of these words
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

# Regular Expressions: Disjunctions

- Characters within square brackets []

| Pattern | Matches |
|---|---|
| [wW]oodchuck | Woodchuck, woodchuck |
| [1234567890] | Any digit |

- Ranges of characters

| Pattern | Matches | Examples |
|---|---|---|
| [A-Z] | An upper case letter | Monty Python |
| [a-z] | A lower case letter | any time |
| [0-9] | A single digit | Chapter 1: Down the Rabbit Hole |

# Regular Expressions: Negation in Disjunctions

- Negations **[^Ss]**
  - Carat means negation only when first in []

| Pattern | Matches | Examples |
|---|---|---|
| [^A-Z] | Not an upper case letter | Monty Python |
| [^Ss] | Neither 'S' nor 's' | In the morning |
| [^e^] | Neither 'e' nor '^' | Look here |
| [a^b] | The pattern 'a^b' | Look up a^b now |

# Regular Expressions: More Disjunctions

- We can also use the pipe ('|') for disjunction

| Pattern | Matches |
|---|---|
| groundhog\|woodchuck | groundhog<br>woodchuck |
| yours\|mine | yours<br>mine |
| a\|b\|c | = [abc] |
| [gG]roundhog\|[Ww]oodchuck | groundhog<br>Groundhog<br>Woodchuck<br>woodchuck |

# Regular Expressions: ? * + .

| Pattern | Meaning | Matches |
|---|---|---|
| colou?r | previous character is optional | color<br>colour |
| oo*h! | 0 or more of the previous character | oh! ooh! oooh! ooooh! |
| o+h! | 1 or more of the previous character | oh! ooh! oooh! ooooh! |
| baa+ | | baa baaa baaaa baaaaa |
| beg.n | any character (wildcard) | begin began begun beg3n |

# Regular Expressions: Anchors ^ $ and Escape \

| Pattern | Meaning | Matches |
|---|---|---|
| ^[A-Z] | begins with upper case letter | New York |
| ^[^A-Za-z] | does not begin with a letter | 1 "Hello" |
| \.$ | ends with a period<br>(backslash is escape character) | The end. |
| .$ | any character followed by anything | 'T', 'h', 'e', ' ', 'e', 'n', 'd', '.' |

# Example

- Want: Find all instances of the word "the" in some text
- Try regular expression: **the** → misses upper case The
- Try regular expression: **[tT]he** → returns "other", "theology", etc.
- Try regular expression: **^[tT]he$** → returns only "The" and "the"

# Errors

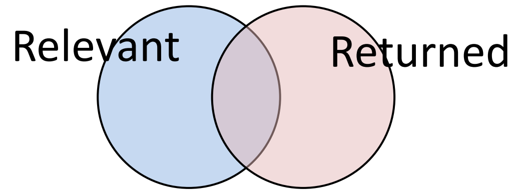
- We just went through a process of fixing two kinds of errors
  - **False positive** (Type I) errors
    - Matching strings that we should not have matched
    - "there", "then", "other"
  - **False negative** (Type II) errors
    - Not matching strings that we should have matched
    - "The"

# Errors (cont.)

- In NLP, we are always dealing with these kinds of errors
- Reducing the error rate for an application often involves two competing goals
  - increasing accuracy or precision (minimizing false positives)
  - increasing coverage or recall (minimizing false negatives)

# Recall and Precision

- Traditional measure for IR systems
  - compute recall and precision based on human relevance judgments




  - recall: the proportion of relevant documents that are returned (found)

$$R = \frac{|Relevant \cap Returned|}{|Relevant|}$$

  - precision: the proportion of returned documents that are relevant

$$P = \frac{|Relevant \cap Returned|}{|Returned|}$$

  - "f-measure": the harmonic mean or recall and precision

$$F_1 = \frac{2PR}{(P+R)}$$

# Regular Expressions in NLP

- Regular expressions play a surprisingly large role
  - Often, sophisticated sequences of regular expressions are the first model for any text processing task (i.e., prototyping)
- For many difficult tasks, we typically use machine learning classifiers
  - However, regular expression results are often used as features
  - Regular expressions can be very useful in capturing general properties

# Today

- Regular Expressions
- Word Tokenization
- Word Normalization and Stemming
- Sentence Segmentation and Decision Trees

# Text Normalization

- Every NLP task needs to do text normalization (preprocessing)

  1. Segmenting/tokenizing words in running text
  2. Normalizing word forms
  3. Segmenting sentences in running text

- These tasks are not as simple as they may appear

# How many words?

- Consider: "I do, uh main- mainly business data processing"
  - Here, there are fragments and filled pauses

- Consider: "Seuss's cat in the hat is different from other cats!"
  - **Wordform**
    - the full inflected surface form
    - cat and cats are different wordforms
  - **Lemma**
    - same stem, part-of-speech, rough word sense
    - cat and cats have the same lemma

# How many words?

- Consider: "He said he will return in a New York minute"
  - **Type**
    - an element of the vocabulary
  - **Token**
    - an instance of a type in running text
  - Number of tokens: 10 or 9?
  - Number of types: 10, 9, or 8?

# How many words?

- Let **N** = number of tokens
  **V** = set of types (vocabulary) , so that |V| = size of vocabulary
- Then **$|V| = O(N^{½})$** (per Church and Gale, 1990)
- Some relevant data points:

| Domain | # Tokens = N | # Types = \|V\| |
|---|---|---|
| Switchboard phone conversations | 2.4 million | 20,000 |
| Shakespeare | 884,000 | 31,000 |
| Google N-grams | 1 trillion | 13 million |

# Simple tokenization in UNIX

- Given a text file, output the word tokens and their frequencies (inspired by Ken Church's "Unix for Poets")

```
tr –sc 'A-Za-z' '\n' < shakes.txt
     | sort
     | uniq –c
```

← change all non-alpha to newlines

← sort in alphabetical order

← merge and count each type

```
Demetrioss-MBP:resources glinosd$ tr -sc 'A-Za-z' '\n' < shakes.txt | sort | uniq -c
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
   8 ABERGAVENNY
  18 ABHORSON
   1 ABOUT
  88 ACHILLES
 259 ACT
  17 ADAM
   1 ADO
  14 ADRIAN
  87 ADRIANA
   3 ADRIANO
```

# Step 1: tokenizing

Command:

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

Console output:

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
we
desire
...
```

# Step 2: sorting

Command:

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

Console output:

```
A
A
A
A
A
A
A
A
A
A
```

. . .

# More counting

- Merging upper and lower case

```
Demetrioss-MBP:resources glinosd$ tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
14725 a
   97 aaron
    1 abaissiez
   10 abandon
    2 abandoned
    2 abase
    1 abash
   14 abate
    3 abated
```

- And sorting the counts

```
Demetrioss-MBP:resources glinosd$ tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
27597 the
26738 and
22538 i
19771 to
18138 of
14725 a
13826 you
12489 my
11536 that
11112 in
9755 is
8960 d
```

What happened here?

# Tokenization Issues

- Finland's capital → Finland, Finlands, or Finland's ?
- what're, I'm, isn't → what are, I am, is not ?
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art?
- Lowercase → lower-case, lowercase, or lower case ?
- San Francisco → one token or two ?
- m.p.h., PhD. → how many tokens?

# Tokenization: language issues

- French
  - ***L'ensemble*** ⟶ one token or two?
    - ***L*** , ***L'*** , or ***Le*** ?
    - We want ***l'ensemble*** to match with ***un ensemble***
- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - *'life insurance company employee'*
  - German information retrieval needs **compound splitter**

# Tokenization: language issues

- Chinese and Japanese – no spaces between words

莎拉波娃现在居住在美国东南部的佛罗里达。

莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

Sharapova now lives in US southeastern Florida

- Japanese also allows intermingling of multiple alphabets

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Katakana Hiragana Kanji Romaji

  - Query to match can be in any of the alphabets!
  - Note also: dates and amounts are in multiple formats

# Word Tokenization in Chinese

- Also called word segmentation
- Chinese words are composed of characters
  - characters are generally 1 syllable and 1 morpheme
  - average word is 2.4 characters long
- Baseline word segmentation algorithm for Chinese is called "Maximum Matching"

Maximum Matching algorithm:

Given a wordlist of Chinese, and a string to segment:

1. Start a pointer at the beginning of the string
2. Find the longest word in dictionary that matches the string starting at the pointer
3. Move the pointer to the end of the word
4. Go to step 2

# Example: Maximum Matching

- Doesn't generally work for English
  - Thecatinthehat → the cat in the hat
  - Thetabledownthere → theta bled own there
    the table down there
- Maximum matching works surprisingly well for Chinese

莎拉波娃现在居住在美国东南部的佛罗里达。

莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

  - Modern probabilistic segmentation algorithms work even better

# Today

- Regular Expressions
- Word Tokenization
- Word Normalization and Stemming
- Sentence Segmentation and Decision Trees

# Normalization

- Why?
  - For Information Retrieval to work, query terms and indexed text must have the same forms
    - e.g., to match U.S.A. and USA

- How?
  - We implicitly define equivalence classes of terms
    - e.g., deleting periods in terms
  - Alternative: asymmetric expansion
    - Query: window Search: window, windows
    - Query: windows Search: Windows, windows, window
    - Query: Windows Search: Windows
    - potentially more powerful, but less efficient

# Case folding

- For applications like IR
  - reduce all letters to lower case
  - users tend to use lower case
  - possible exception: upper case in mid-sentence?
    - General Motors
    - Fed v. fed
    - SAIL v. sail

- For sentiment analysis, MT, information extraction
  - case is helpful
  - US v. us

# Lemmatization

- Reduce inflections or variant forms to base form
  - *am, are, is* ⟶ *be*
  - *car, cars, car's, cars'* ⟶ car
  - *the boys' cars are different colors* ⟶ *the boy car be different color*
- Lemmatization for MT
  - more difficult since often words don't map 1:1
  - e.g., Spanish quiero ('I want'), quires ('you want') have same lemma as querer ('want')

# Morphology

- **Morphemes**
  - the small, meaningful units that make up words
  - **stems**
    - the core meaning-bearing units
  - **affixes**
    - bits and pieces that adhere to stems, often with grammatical functions
    - prefixes, suffixes, infixes, circumfixes
    - e.g., in German: verb *sagen* (to *say*)
      past tense uses circumfix: gesagt

# Stemming

- Reduce terms to their stems for information retrieval
- **Stemming** is crude chopping of affixes
  - language dependent
  - e.g., ***automate(s), automatic, automation*** ⟶ ***automat***

| for example compressed and compression are both accepted as equivalent to compress | ➡ | for exampl compressed and compress ar both accept as equival to compress |
| --- | --- | --- |

# Porter's algorithm

The most commonly used English stemmer: https://tartarus.org/martin/PorterStemmer/

**NOTE: We show a simplified version**

Step 1a

| | |
|---|---|
| sses ⟶ ss | caresses ⟶ caress |
| ies ⟶ i | ponies ⟶ poni |
| ss ⟶ ss | caress ⟶ caress |
| s ⟶ ∅ | cats ⟶ cat |

Step 2 (for long stems)

| | |
|---|---|
| ational ⟶ ate | relational ⟶ relate |
| izer ⟶ ize | digitizer ⟶ digitize |
| ator ⟶ ate | operator ⟶ operate |

Step 1b

| | |
|---|---|
| (*v*)ing ⟶ ∅ | walking ⟶ walk |
| | sing ⟶ sing |
| (*v*)ed ⟶ ∅ | plastered ⟶ plaster |
| | bled ⟶ bled |

Step 3 (for longer stems)

| | |
|---|---|
| al ⟶ ∅ | revival ⟶ reviv |
| able ⟶ ∅ | adjustable ⟶ adjust |
| ate ⟶ ∅ | activate ⟶ activ |

# Viewing morphology in a corpus

- Why strip –ing only if there is a preceding vowel?

| | |
|---|---|
| (*v*)ing $\longrightarrow \varnothing$ | walking $\longrightarrow$ walk |
| | sing $\longrightarrow$ sing |

```
Demetrioss-MBP:resources glinosd$ tr -sc 'A-Za-z' '\n' < shakes.txt | grep 'ing$' | sort | uniq -c | sort -nr
1312 King
 548 being
 541 nothing
 388 king
 375 bring
 358 thing
 307 ring
 152 something
 145 coming
 130 morning
 127 sing
 122 having
```

13080

```
Demetrioss-MBP:resources glinosd$ tr -sc 'A-Za-z' '\n' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort -nr
 548 being
 541 nothing
 152 something
 145 coming
 130 morning
 122 having
 120 living
 117 loving
 116 Being
 102 going
 100 anything
```

9730

# Dealing with complex morphology

- Some languages require complex morpheme segmentation
  - Turkish
    - Uygarlastiramadiklarimizdanmissinizcasina
    - '(behaving) as if you are among those whom we could not civilize'
    - Uygar ('civilized') + las ('become')
      + tir ('cause') + ama ('not able')
      + dik ('past) + lar ('plural')
      + imiz ('people') + dan ('able')
      + mis ('past') + siniz ('people')
      + casina ('as if')

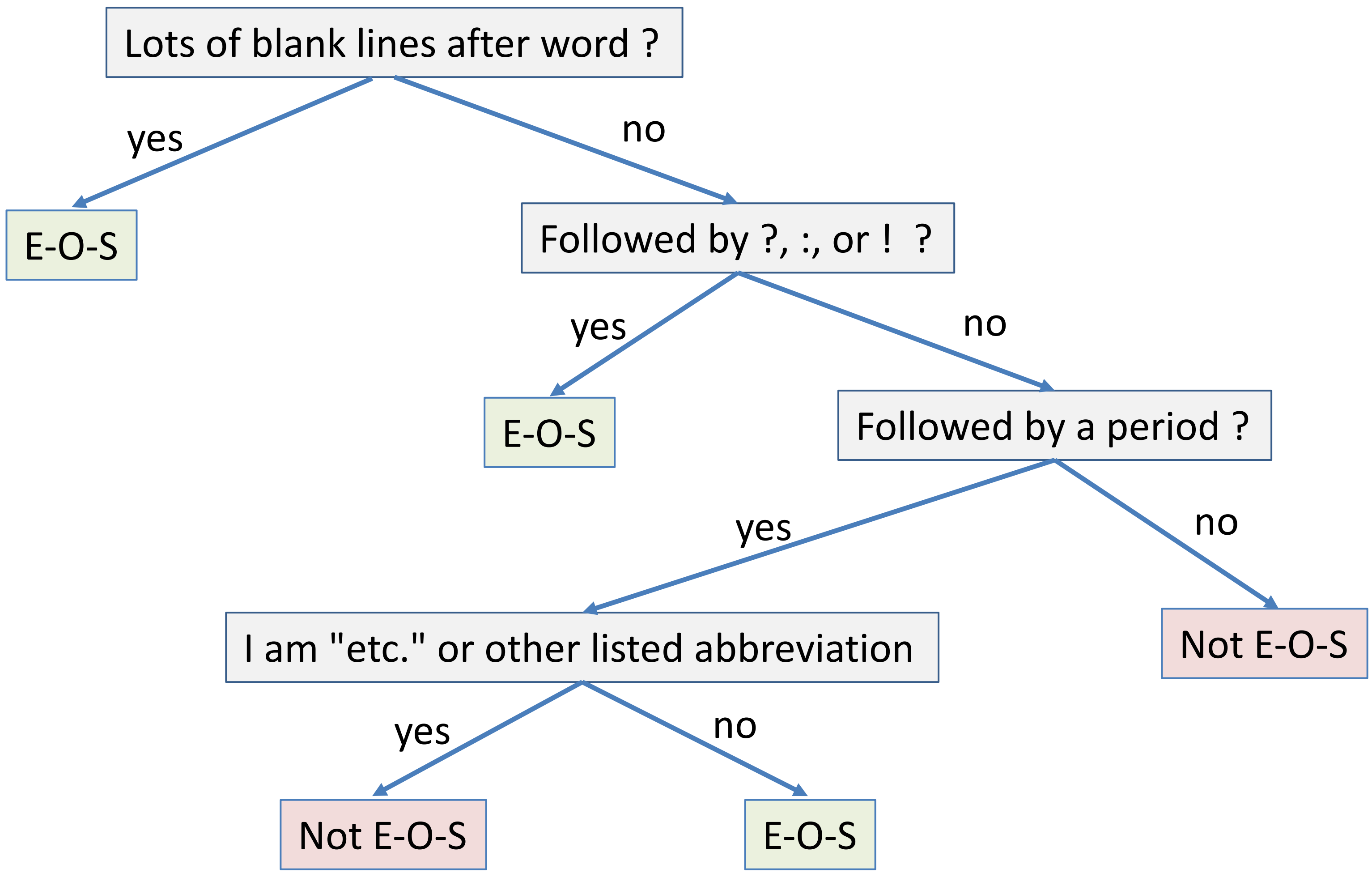
# Today

- Regular Expressions
- Word Tokenization
- Word Normalization and Stemming
- Sentence Segmentation and Decision Trees

# Sentence Segmentation

- ! and ? are relatively unambiguous
- Period (".") is quite ambiguous
  - sentence boundary
  - abbreviations like "Inc." or "Dr."
  - numbers like .02%, 12. , or 4.3
- Solution: build a binary classifier
  - examines each instance of a "." and decides whether or not end of sentence
  - classifier
    - hand-written rules
    - regular expressions
    - machine-learning

# Decision tree: is a word at end-of-sentence



# More sophisticated decision tree features

- Case of word with "." : upper, lower, capitalized, number
- Case of word after "." : upper, lower, capitalized, number
- Numeric features
  - length of the word with "."
  - Probability( word with "." occurs at end-of-sentence )
  - Probability( word after "." occurs at beginning-of-sentence )

# Implementing Decision Trees

- A decision tree can be implemented by simple if-then-else constructs
- The interesting part is choosing the order in which to test the features
- Setting up the structure is often too difficult to do by hand
  - Hand-building only possible for very simple features and problem domains
  - For numeric features, it is generally too difficult to choose each threshold by hand
  - Structure is usually learned by machine learning from a training corpus
    - e.g., ID3 algorithm, which uses concept of "information gain" for choosing features

# Decision Trees and other Classifiers

- The questions in a decision tree are features that could be exploited by any kind of classifier
    - Naive Bayes
    - Logistic regression
    - SVM
    - Neural Net
    - etc.