Natural Language Processing

Professor John Licato
University of South Florida

Chapter 2:
RegEx, Edit Distance

ADVANCING MACHINE AND
HUMAN REASONING (AMHR) LAB

UNIVERSITY OF
SOUTH FLORIDA

Regular Expressions

"Knowing [regular expressions] can mean the difference between solving a problem in 3 steps and solving it in 3,000 steps. When you're a nerd, you forget that the problems you solve with a couple keystrokes can take other people days of tedious, error-prone work to slog through."

Regular Expressions

The following function called `isPhoneNumber(text)` is designed to check if the provided string is a phone number in a specific format using regex.

```
def isPhoneNumber(text):
    if len(text) != 12:
        return False
    for i in range(0, 3):
        if not text[i].isdecimal():
            return False
    if text[3] != '-':
        return False
    for i in range(4, 7):
        if not text[i].isdecimal():
            return False
    if text[7] != '-':
        return False
    for i in range(8, 12):
        if not text[i].isdecimal():
            return False
    return True

print('415-555-4242 is a phone number:')
print(isPhoneNumber('415-555-4242'))
print('Moshi moshi is a phone number:')
print(isPhoneNumber('Moshi moshi'))
```

Regular Expressions

The following Python code uses the previously defined `isPhoneNumber` function within a loop to search through a longer string for valid phone number formats.

```
message = 'Call me at 415-555-1011 tomorrow. 415-555-9999 is my office.'
for i in range(len(message)):
    chunk = message[i:i+12]
    if isPhoneNumber(chunk):
        print('Phone number found: ' + chunk)
print('Done')
```

Creating regex objects

```
>>> import re
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

r' = raw string

\d – placeholder for a single digit

Matching regex objects

```
>>> import re
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> print('Phone number found: ' + mo.group())

Phone number found: 415-555-4242
```

mo = match object – contains the result of our search

Text Normalization

- We will work a lot with large datasets / corpora
- We often need to pre-process text
- Tokenizing (segmenting) words
- Normalizing word formats
- Segmenting sentences (e.g. by using punctuation)

Tokenization – segmenting running text into words (or word-like units)

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r''', (?x)  # set flag to allow verbose regexps
...     ([A-Z]\.)+       # abbreviations, e.g. U.S.A.
...     | \w+(-\w+)*      # words with optional internal hyphens
...     | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...     | \.\.\.          # ellipsis
...     | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Subword tokenization

- How do we capture relations between words like:
  - new, newer
  - blow, blowing
  - precipitation, precipitate
- Often useful to break tokens into *sub*words
- Usually split into token learners, and token segmenters

Byte-pair encoding (BPE)

- A way of performing subword tokenization

```
function BYTE-PAIR ENCODING(strings C, number of merges k) returns vocab V
V <- all unique characters in C                # initial set of tokens is characters
for i = 1 to k do                              # merge tokens til k times
    t_L, t_R <- Most frequent pair of adjacent tokens in C
    t_new <- t_L + t_R                         # make new token by concatenating
    V <- V + t_new                             # update the vocabulary
    Replace each occurrence of t_L, t_R in C with t_new # and update the corpus
return V
```

corpus
5 low_
2 lowest_
6 newer_
3 wider_
2 new_

vocabulary
_, d, e, i, l, n, o, r, s, t, w

corpus
5 low _
2 lowest _
6 newer _
3 wider _
2 new _

vocabulary
_, d, e, i, l, n, o, r, s, t, w, er

Word normalization

- Case folding – e.g., making everything lowercase
- Lemmatization – folding lemmas together if they have the same root (dinner / dinners, am / are / is, etc.).
- Stemming – performing lemmatization by removing all but the roots of words (running / runner -> run)

Minimum Edit Distance

Definition of Minimum Edit Distance

How similar are two strings?

- Spell correction
  - The user typed "graffe"

  Which is closest?
    - graf
    - graft
    - grail
    - giraffe

- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

  - Resulting alignment:

    –AGG**CTATCAC**C**TGACCT**C**A**G**G**C**G**A**--TGCCC**---
    T**AG**–**CTATCAC**--**GACC**G**C**--**GG**T**CGA**TT**TGCCC**GAC

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
  - Insertion
  - Deletion
  - Substitution
- Needed to transform one into the other

Minimum Edit Distance

- Two strings and their **alignment**:

```
INTENTION
| | | | | | |
*EXECUTION
```

Minimum Edit Distance

INTENTION
| | | | | | |
*EXECUTION
d s s is s

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

Alignment in Computational Biology

- Given a sequence of bases

  AGGCTATCACCTGACCTCCAGGCCGATGCCC
  TAGCTATCACGACCGCGGTCGATTTGCCCGAC

- An alignment:

  -AGG**CTATCAC**C**T**T**GACC**T**C**C**A**T**GG**C**CGA--TGCCC**---
  TT**AG-CTATCAC--GACC**G**C--GG**T**CGA**T**T**TGCCC**GAC

- Given two sequences, align each letter to a letter or gap

Other uses of Edit Distance in NLP

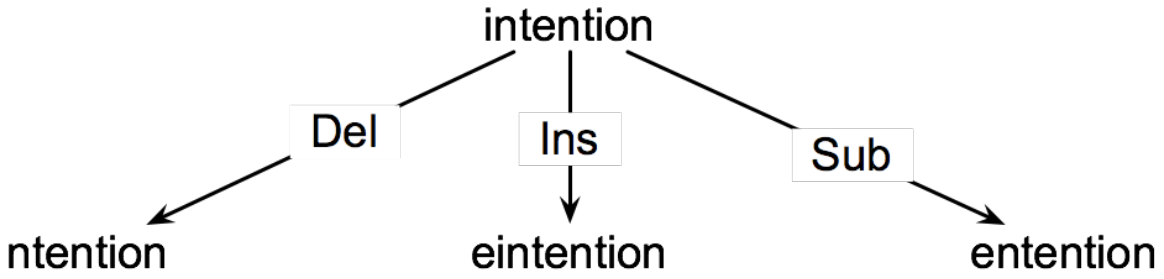
- Evaluating Machine Translation and speech recognition

```
R Spokesman confirms    senior government adviser was shot
H Spokesman said    the senior            adviser was shot dead
              S      I             D                        I
```

- Named Entity Extraction and Entity Coreference
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announced yesterday
  - for Stanford University President John Hennessy

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits



Minimum Edit as Search

- But the space of all edit sequences is huge!
  - We can't afford to navigate naïvely
  - Lots of distinct paths wind up at the same state.
    - We don't have to keep track of all of them
    - Just the shortest path to each of those revisted states.

Defining Min Edit Distance

- For two strings
  - X of length *n*
  - Y of length *m*
- We define D(*i,j*)
  - the edit distance between X[1..*i*] and Y[1..*j*]
    - i.e., the first *i* characters of X and the first *j* characters of Y
  - The edit distance between X and Y is thus D(*n,m*)

Dynamic Programming for Minimum Edit Distance

- **Dynamic programming**: A tabular computation of D(*n,m*)
- Solving problems by combining solutions to subproblems.
- Bottom-up
  - We compute D(i,j) for small *i,j*
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(*i,j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

Defining Min Edit Distance (Levenshtein)

- Initialization

  D(i,0) = i

  D(0,j) = j

- Recurrence Relation:

  For each i = 1…M

  For each j = 1…N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

  D(N,M) is distance

The Edit Distance Table

The minimum edit distance between two strings is the minimum number of editing operations needed to transform one string into the other. The typical operations allowed are:

1. **Insertion (Ins)**: Add one character to the string.
2. **Deletion (Del)**: Remove one character from the string.
3. **Substitution (Sub)**: Replace one character with another.

Computing alignments

- Edit distance isn't sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a "backtrace"
- Every time we enter a cell, remember where we came from
- When we reach the end,
  - Trace back the path from the upper right corner to read off the alignment

MinEdit with Backtrace

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↙ **8**
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↙ **8**	← 9
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↙ **8**	← 9	← 10
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙ **8**	← 9	← 10	←↓ 11
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ **8**	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10
e	4	↙ 3	← 4	↙← **5**	← **6**	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9
t	3	↙←↓ 4	↙←↓ **5**	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8
n	2	↙←↓ **3**	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙ 6	← 7	← 8
#	**0**	1	2	3	4	5	6	7	8	9
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**

Adding Backtrace to Minimum Edit Distance

- Base conditions:

  `D(i,0) = i` `D(0,j) = j`

  Termination:

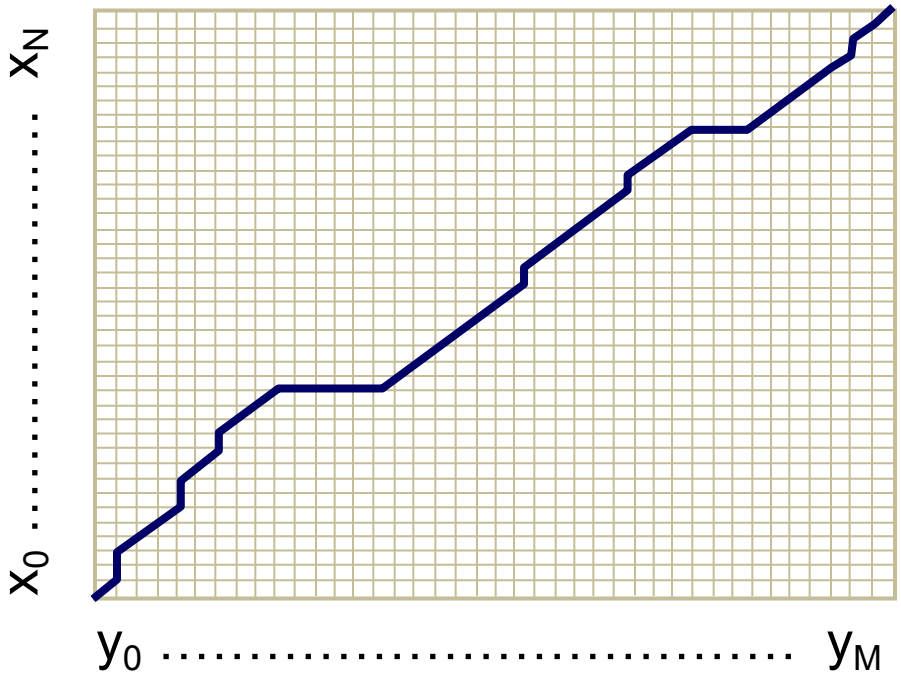
  `D(N,M) is distance`

- Recurrence Relation:

```
For each  i = 1…M
      For each  j = 1…N
                          D(i-1,j) + 1        deletion
D(i,j)= min               D(i,j-1) + 1        insertion
                          D(i-1,j-1) +  2;    if X(i) ≠ Y(j)   substitution
                                        0;    if X(i) = Y(j)

                          LEFT        insertion
ptr(i,j)=                 DOWN        deletion
                          DIAG        substitution
```

The Distance Matrix



Every non-decreasing path

from (0,0) to (M, N)

corresponds to
an alignment
of the two sequences

An optimal alignment is composed of optimal subalignments

Slide adapted from Serafim Batzoglou

Result of Backtrace

- Two strings and their **alignment**:

```
INTENTION
| | | | | | |
*EXECUTION
```

Performance

- Time:

  O(nm)

- Space:

  O(nm)

- Backtrace

  O(n+m)

Hearst Patterns for Hypernymy

- Hyp**o**nym – “Is-A” relationship
- Hyp**er**nym – Opposite of hypernym
- **Color** is a hypernym of **red**; **cat** is a hypernym of **white cat**.
- A rule-based way of detecting hypernym relationships in text is through *Hearst patterns*

Some Hearst Patterns

- All bolded symbols (**a**, **b**, **…**) are noun phrases
- Type 1 - Extract (b,a)
  - "**a** is **b**"
  - "**a** is a type of **b**"
  - "**a** is a kind of **b**"
  - "**a** was **b**"
  - "**a** was a type of **b**"
  - "**a** was a kind of **b**"
  - "**a** are **b**"
  - "**a** are a type of **b**"
  - "**a** are a kind of **b**"
- Type 2 - Extract (a,b), (a,c), …, (a,d)
  - "**a**, including **b**"
  - "**a**, including **b**, **c**, …, and **d**"
  - "**a**, including **b**, **c**, …, or **d**"
  - "**a**, such as **b**"
  - "**a**, such as **b**, **c**, …, and **d**"
  - "**a**, such as **b**, **c**, …, or **d**"
- There are many others!