Text Preprocessing and Edit Distance

Natalie Parde, Ph.D.

Department of Computer Science

University of Illinois at Chicago

CS 421: Natural Language Processing

Fall 2019

Many slides adapted from Jurafsky and Martin (https://web.stanford.edu/~jurafsky/slp3/) and Stanford's NLP Coursera course (https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html).

What is text preprocessing?

- Automated organization, normalization, and manipulation of text such that it can more easily be handled by downstream language processing tasks.

"Have some wine," the March Hare said in an encouraging tone.

Alice looked all round the table, but there was nothing on it but tea. "I don't see any wine," she remarked.

"There isn't any," said the March Hare.

- Lewis Carroll, *Alice's Adventures in Wonderland*

have some wine [PERSON 1] said in an encouraging tone

[PERSON 2] looked all round the table but there was nothing on it but tea

i don't see any wine she remarked

there isn't any said [PERSON 1]

- Lewis Carroll, *Alice's Adventures in Wonderland*

Text preprocessing steps can (and should!) vary depending on your needs.

capitalization

written numbers vs. digits

punctuation

proper nouns

British vs. American spellings (for English text)

Important

?

Not Important

One way to preprocess text is by using **regular expressions**.

- Regular expressions: A formal language for specifying text strings.
- How can we search for any of these?
  - Donut
  - donut
  - Doughnut
  - doughnut
  - Donuts
  - doughnuts

Regular Expression Terminology

Regex: Common abbreviation for **reg**ular **ex**pression

Disjunction: Logical OR

Range: All characters in a sequence from c_1-c_2

Negation: Logical NOT

Scope: Indicates to which characters the regex applies

Anchor: Matches the beginning or end of a string

Regular Expressions: Disjunctions (and Ranges)

- Disjunction: Letters inside square brackets [az]
- Range: Hyphen between the first and last characters in the range [a-z]

Pattern	Matches	Example
[dD]onut	donut, Donut	This morning would be better with a **donut**.
[0123456789]	Any digit	This morning would be better with **5** donuts.
[A-Z]	An uppercase letter	**D**onuts are an excellent way to start the day.
[a-z]	A lowercase letter	What is your favorite kind of **d**onut?
[0-9]	Any digit	I just ate **5** donuts.

Regular Expressions: Negation in Disjunction

- Negation: A caret (^) at the beginning of a disjunction [^az]
  - The caret must be at the beginning of the disjunction to negate it

Pattern	Matches	Example
[^dD]onut	Any letter except "d" or "D" before the sequence "onut"	This morning would be better with a co**c**onut.
[^A-Z]	Not an uppercase letter	D**o**nuts are an excellent way to start the day.
[^^]	Not a caret	**W**hat is your favorite kind of donut?
D^o	The pattern "D^o"	Is **D^o**nut a good name for my donut shop?

Regular Expressions: More Disjunction

- The pipe | indicates the union (logical OR) of two smaller regular expressions
- a|b|c is equivalent to [abc]

Pattern	Matches	Example
d|D	“d” or “D”	This morning would be better with a **d**onut.

Regular Expressions: Special Characters

- *: Means that there must be 0 or more occurrences of the preceding expression
- .: A wildcard that can mean any character
- **+**: Means that there must be 1 or more occurrences of the preceding expression
- ?: Means that there must be 0 or 1 occurrences of the preceding expression
- **{m}**: Means that there must be *m* instances of the preceding expression
- **{m,n}**: Means that there must be between *m* and *n* instances of the preceding expression

Regular Expressions: Special Characters

Pattern	Matches	Example
donuts*	"donut" or "donuts" or "donutss" or "donutsss"….	This morning I had many **donuts**.
.onut	Any character followed by "onut"	Can I have a co**conut donut**?
donuts+	"donuts" or "donutss" or "donutsss"….	Do you want one donut or two **donuts**?
donuts?	"donut" or "donuts"	Do you want one **donut** or two **donuts**?
donuts{1}	"donuts"	Do you want one donut or two **donuts**?
donuts{0,1}	"donut" or "donuts"	Do you want one **donut** or two **donuts**?

Regular Expressions: Anchors

- Indicate that a pattern should be matched only at the beginning or end of a word

Pattern	Matches	Example
^Donuts	"Donuts" only when it is at the beginning of a string	**Donuts** are an excellent way to start the day.
$donuts\.	"donuts." only when it is at the end of the string	I just ate 5 **donuts.**
$donuts.	"donuts" + one additional character, only when it is at the end of the string	I just ate 12 **donuts!**

Simple(?) Task: Create a regular expression to match the word "the"

https://www.google.com/search?q=timer

Pattern	Matches
[dD]onut	donut, Donut
[0123456789]	Any digit
[A-Z]	An uppercase letter
[a-z]	A lowercase letter
[0-9]	Any digit
[^dD]onut	Any letter except "d" or "D" before the sequence "onut"
[^A-Z]	Not an uppercase letter
donut|doughnut	"donut" or "doughnut"
[dD]onut|[dD]oughnut	"donut" or "Donut" or "doughnut" or "Doughnut"
donuts*	"donut" or "donuts" or "donutss" or "donutsss"….
.onut	Any character followed by "onut"
donuts+	"donuts" or "donutss" or "donutsss"….
donuts?	"donut" or "donuts"
donuts{1}	"donuts"

Possible Solutions

the

- Fails on test case: The

[tT]he

- Fails on test case: other

[^a-zA-Z][tT]he[^a-zA-Z]

- :-) ?

Errors

- In iterating through possible solutions to avoid the first two failures, we were trying to fix two types of errors:
  - Matching strings that we should not have matched (there, then, other)
    - False positives (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)

Errors

- This is a recurring theme in NLP!
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing **accuracy** or **precision** (minimizing false positives)
  - Increasing **coverage** or **recall** (minimizing false negatives)

Regular Expressions: Takeaway Points

Regular expressions are a surprisingly powerful tool!

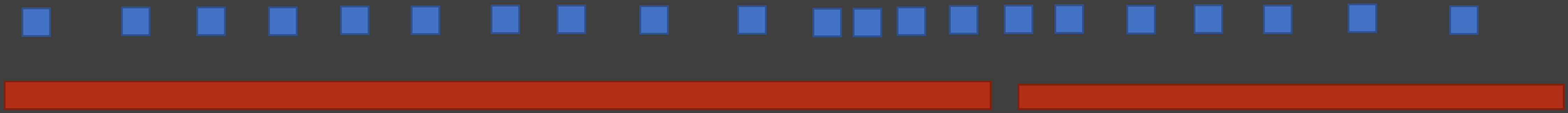
They are critical to text tokenization and normalization.

They may also be used to extract **features** for machine learning classifiers.

Text Tokenization and Normalization

- Text tokenization and normalization are critical to most (all?) NLP tasks
- A typical NLP pipeline begins by:
  - Separating words in running text
  - Normalizing word formats (e.g., favourite = favorite)
  - Segmenting sentences in running text

Alice looked all round the table, but there was nothing on it but tea. “I don't see any wine," she remarked.



How many words?

- I do uh main- mainly business data processing
  - Fragments, filled pauses
- Seuss's **cat** in the hat is different from other **cats**!
  - **Lemma**: same stem, part of speech, rough word sense
    - cat and cats = same lemma
  - **Wordform**: the full inflected surface form
    - cat and cats = different wordforms

How many words?

Alice looked all round the table, but there was nothing on it but tea.

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
  - 14 tokens (or 15?)
  - 13 types (or 14?)

How many words?

N = number of tokens

V = vocabulary = set of types

|*V*| is the size of the vocabulary

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

Simple Tokenization in Python

- Given a string of text, output the word tokens (assuming all words are delimited by whitespace) and their frequencies

```
sentence = “Alice looked all round the table, but there was nothing on it but tea.”
tokens = sentence.split()
```

['Alice', 'looked', 'all', 'round', 'the', 'table,', 'but', 'there', 'was', 'nothing', 'on', 'it', 'but', 'tea.']

Simple Tokenization in Python

- Given a string of text, output the word tokens (assuming all words are delimited by whitespace) and their frequencies

```
types = set(tokens)
for t in types:
   freq = tokens.count(t)
   print("Word: {0}\tFreq: {1}".format(t, freq))
```

```
Word: was        Freq: 1
Word: Alice      Freq: 1
Word: table,     Freq: 1
Word: looked     Freq: 1
Word: it         Freq: 1
Word: but        Freq: 2
Word: tea.       Freq: 1
Word: there      Freq: 1
Word: round      Freq: 1
Word: all        Freq: 1
Word: nothing    Freq: 1
Word: on         Freq: 1
Word: the        Freq: 1
```

Issues in Tokenization

- Finland's capital → Finland Finlands 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 lower case ?
- San Francisco → one token or two?
- m.p.h., PhD. → ??

Tokenization: Language Issues

French

- ***L'ensemble*** → one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***

German noun compounds are not segmented

- ***Lebensversicherungsgesellschaftsangestellter***
- 'life insurance company employee'
- German information retrieval needs a **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
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats

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

End-users can express queries entirely in hiragana!

Word Tokenization in Chinese

- Also called **word segmentation**
- Chinese words are composed of characters
  - Generally one syllable each
  - Average word is 2.4 characters long
- Standard baseline segmentation algorithm:
  - Maximum Matching

Maximum Matching Word Segmentation Algorithm

Given a wordlist of Chinese and a string:

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

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

Doesn’t generally transfer to English….

Thecatinthehat → the cat in the hat

Thetabledownthere → ?

theta bled own there

the table down there

- Nice Python tokenizers:
  - NLTK: http://www.nltk.org/api/nltk.tokenize.html
  - spaCy: https://spacy.io/api/tokenizer
  - StanfordNLP: https://stanfordnlp.github.io/stanfordnlp/

Text Normalization

- **Normalization:** Manipulating text such that all **forms** of the same word match (e.g., U.S.A. = USA, flavour = flavor, etc.)
- To normalize text, you must define **equivalence classes**
  - Example: Periods in a term → not important
- Words with the same characters but different capitalization are often considered equivalent to one another (referred to as **case folding**)
  - Example: Hello = hello
  - Not a perfect strategy!
    - US != us
- Useful equivalence classes vary depending on task
  - Capitalization can be very important in sentiment analysis

Lemmatization

- Reduce inflections or variant forms to base form
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
- *the boy's cars are different colors* $\rightarrow$ *the boy car be differ color*
- Tricky because you need to find the correct dictionary headword form
- Very useful for machine translation

Morphology

- **Morphemes**:
  - Small meaningful units that make up words
  - **Stems**: The core meaning-bearing units
  - **Affixes**: Bits and pieces that adhere to stems and add additional information
    - -ed
    - -ing
    - -s

Stemming

- Automatically reduces words to their stems using simple rules
  - language dependent
  - Example: {automate(s), automatic, automation} → automat
- Pros: Very quick, simple to implement
- Cons: Groups together some words that don’t really mean the same thing, and doesn’t group together some words that do mean the same thing
  - {meanness, meaning} → mean
  - {goose} → goos, {geese} → gees

Porter Stemming

- Step 1a
  - sses → ss caresses → caress
  - ies → i ponies → poni
  - ss → ss caress → caress
  - s → ø cats → cat
- Step 1b
  - (*v*)ing → ø walking → walk
  - sing → sing
  - (*v*)ed → ø plastered → plaster
  - …
- Step 2 (for long stems)
  - ational→ ate relational→ relate
  - izer→ ize digitizer → digitize
  - ator→ ate operator → operate
  - …
- Step 3 (for longer stems)
  - al → ø revival → reviv
  - able → ø adjustable → adjust
  - ate → ø activate → activ
  - …

Much like tokenization, stemming methods are difficult to transfer across languages….

- Some languages requires complex morpheme segmentation
- Example from 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’

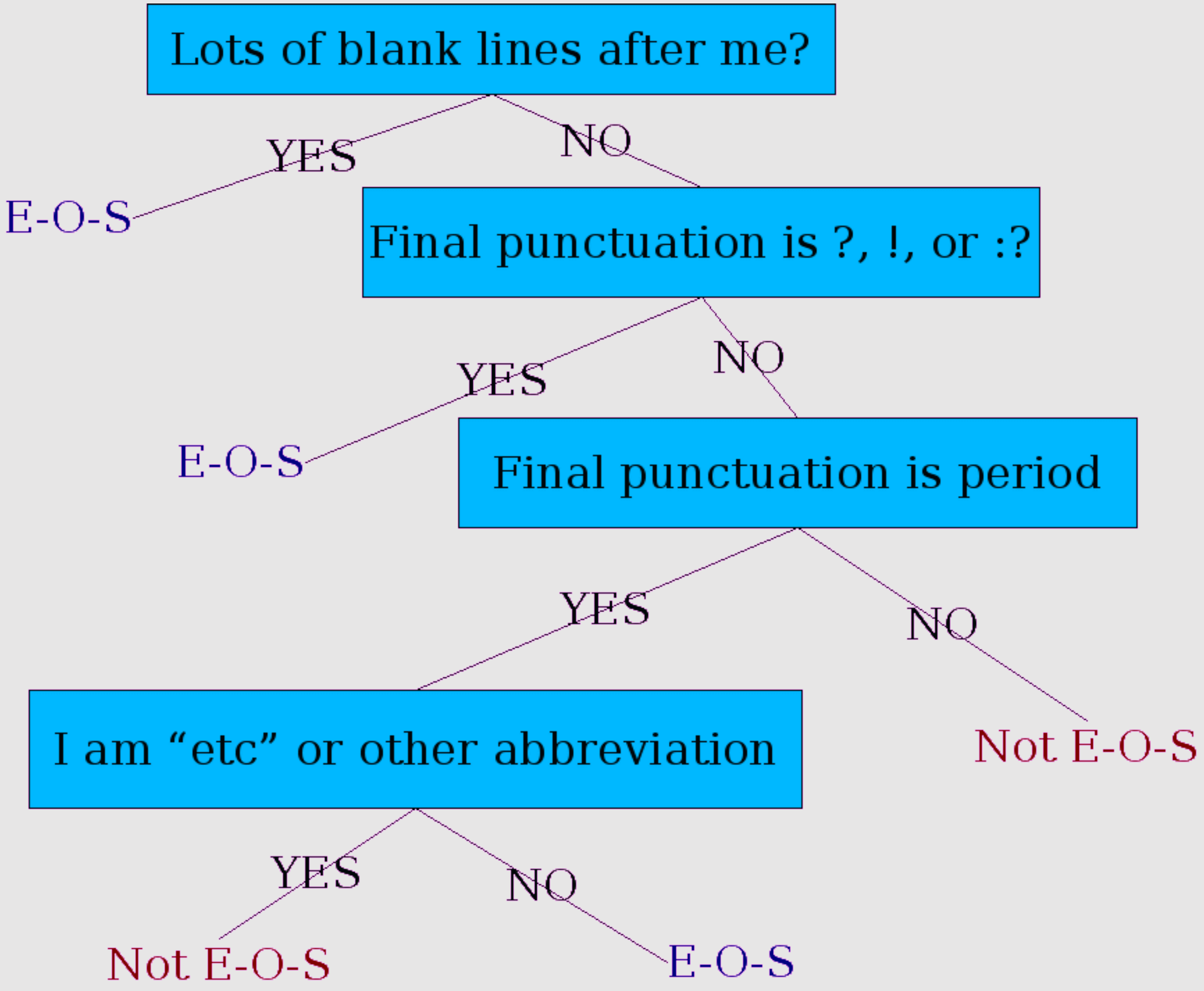
    + imiz ‘p1pl’ + dan ‘abl’

    + mis ‘past’ + siniz ‘2pl’ + casina ‘as if’

Sentence Segmentation

- !, ? are relatively unambiguous
- . is more ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Simple sentence segmentation:
  - Build a binary **classifier** that checks for "."
    - Classifier: A statistical or rule-based model that predicts labels for unseen test input
  - At each token, decides EndOfSentence/NotEndOfSentence

More Complex Sentence Segmentation: Decision Tree



More Sophisticated Decision Tree Features

- Case of word before “.”: Upper, Lower, Number
- Case of word after “.”: Upper, Lower, Number

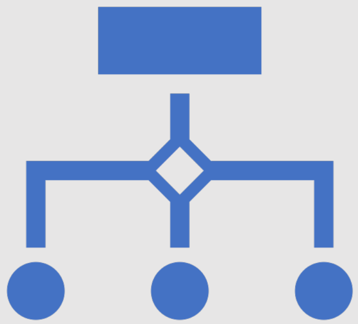
- Numeric features:
  - Length of word before “.”
  - Probability(word before “.” occurs at end-of-s)
  - Probability(word after “.” occurs at beginning-of-s)

Implementing Decision Trees

- Decision trees are fancy if-then-else statements
- The interesting part: Choosing the features!
- Unless there are very few features, it is too complex to choose the tree's structure (e.g., which features are closer to the top) by hand
  - Instead, that structure is usually learned via machine learning from a training corpus

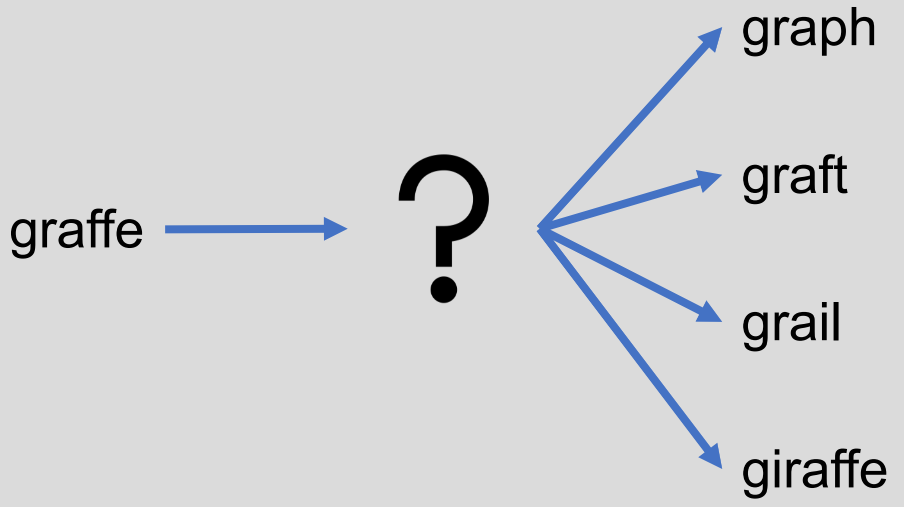
We'll learn more about text classification later in the semester!

- The same features that decision trees use can also be used to train **logistic regression models**, **support vector machines**, **neural networks**, etc.



Edit Distance

- Simple way to answer the question: How similar are two strings?
- Useful for spelling correction



Minimum Edit Distance

- Minimum number of editing operations needed to transform one string into another
- Possible editing operations:
  - Insertion
  - Deletion
  - Substitution

Minimum Edit Distance

- Two strings and their **alignment**:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

Minimum Edit Distance

- If each operation has a cost of 1 (Levenshtein distance)
  - Distance between these is 5
- If substitutions cost 2 (alternative also proposed by Levenshtein)
  - Distance between them is 8

I N T E * N T I O N

* E X E C U T I O N

d s s   i s

Other Uses of Edit Distance in NLP

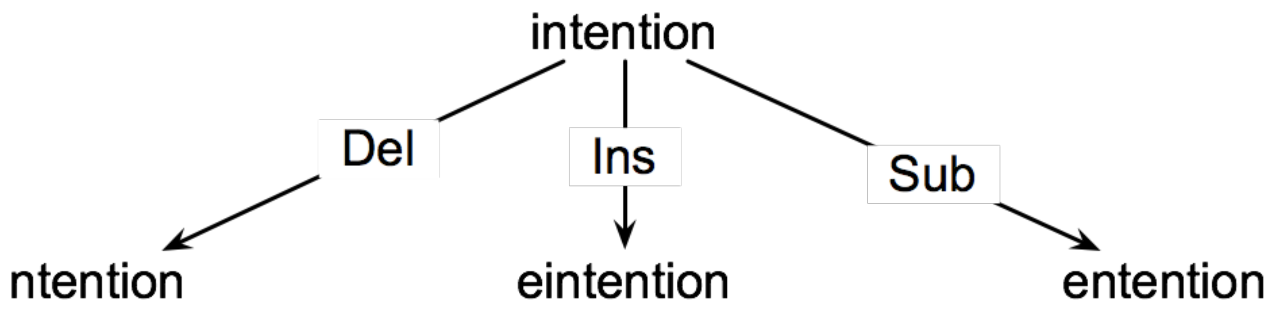
- Evaluating Machine Translation and speech recognition

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

- Named Entity Extraction and Entity Coreference
  - **IBM Inc.** announced today
  - **IBM** profits

How to find the minimum edit distance?

- Search 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)



However, the search 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 paths)

Formal Definition: Minimum Edit Distance

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

Intuition: Dynamic Programming

- Minimum edit distance can be solved using **dynamic programming**
  - Stores intermediate outputs in a table
  - Intuition: If some string B is in the optimal path from string A to string C, then that path must also include the optimal path from A to B
- D(n,m) is computed tabularly, 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* ($0 < j < m$)

Formal Definition: Minimum Edit Distance

- 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

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2								
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3	4	5	6	7			
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3	4	5	6	7	6		
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

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

Backtrace for Computing Alignments

- We know the minimum edit distance now …but what is the alignment between the two strings?
- We can figure this out by maintaining a **backtrace**
  - For each new cell, remember where we came from!
    - D(i-1,j) ?
    - D(i,j-1) ?
    - D(i-1,j-1) ?
- Once we reach the end of the table (upper right corner), we can trace backward using these pointers to figure out the alignment

The Edit Distance Table

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

Formal Definition: Minimum Edit Distance with Backtrace

- 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
```

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

$$
ptr(i,j) = \begin{cases} \text{LEFT} & \boxed{\text{insertion}} \\ \text{DOWN} & \boxed{\text{deletion}} \\ \text{DIAG} & \boxed{\text{substitution}} \end{cases}
$$

The Final Product

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```

Summary

- **Text Preprocessing:** Preparing text for downstream language processing tasks
  - Tokenization
  - Normalization
  - Lemmatization
  - Stemming
- Regular expressions are a powerful tool for text preprocessing!
- **Edit Distance:** Determining the similarity between two strings based on the number of insertions, deletions, and substitutions needed to transform one to another
- Minimum edit distance, computed using dynamic programming, allows you to find the smallest number of edits needed to do so.