```
-->
```

Chapter 2: Regular Expressions, Text Normalization, and Edit Distance

Covered Material	Value
Chapter	Chapter 2
Book	Speech and Language Processing (Jurafsky & Martin, 3rd ed.)
Related Week	Week 2
Related Slides / Files	week2.md, week2_inperson_slides.md, 2.md

Index

Brief Content Summary

This chapter addresses the preprocessing and normalization of text data, essential steps preceding any Natural Language Processing (NLP) or text mining task. Regular expressions are introduced as a formal language for pattern specification, enabling precise text search, extraction, and substitution. The chapter then examines tokenization—segmenting text into meaningful units—distinguishing between rule-based (top-down) and data-driven (bottom-up) approaches, with particular attention to the Byte-Pair Encoding (BPE) algorithm. Text normalization techniques including case folding, lemmatization, and stemming are formalized. Evaluation methodology is presented through the confusion matrix framework, defining precision, recall, and F-score metrics. Finally, the minimum edit distance algorithm is derived using dynamic programming, providing a principled measure of string similarity with applications in spelling correction and sequence alignment.

Abstract

Text data rarely arrives in a form suitable for computational analysis. This chapter provides a rigorous treatment of the preprocessing pipeline that transforms raw text into structured representations. Beginning with regular expressions—an algebraic notation for characterizing string sets—we establish the theoretical foundations for pattern matching, including concatenation, disjunction, Kleene closure, and anchoring. The chapter formalizes tokenization as the segmentation of character sequences into discrete units, comparing rule-based approaches with the Byte-Pair Encoding algorithm that automatically induces subword vocabularies. Heaps' Law ($|V| = kN^\beta$) is presented to characterize the relationship between corpus size and vocabulary growth. Text normalization operations—case folding, lemmatization, and stemming—are defined and contrasted. A formal framework for system evaluation introduces the confusion matrix, from which precision ($\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$), recall (

$\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$), and F-score are derived. The chapter culminates with the minimum edit distance algorithm, formulated as a dynamic programming solution that computes the optimal sequence of insertions, deletions, and substitutions to transform one string into another, with the recurrence relation $D[i, j] = \min\{D[i-1, j] + 1, D[i, j-1] + 1, D[i-1, j-1] + \text{cost}\}$.

Keywords

Regular Expressions; Tokenization; Byte-Pair Encoding (BPE); Text Normalization; Lemmatization; Stemming; Heaps' Law; Confusion Matrix; Precision; Recall; F-Score; Minimum Edit Distance; Levenshtein Distance; Dynamic Programming; String Alignment

1. Introduction

1.1 Purpose and Scope

This chapter establishes the foundational techniques for preparing text data for computational analysis. Whereas Chapter 1 addressed the representation of processed text as vectors, this chapter examines the prerequisite steps: extracting structured tokens from raw character sequences, normalizing variant forms, and measuring string similarity.

The objectives are threefold: (i) to formalize regular expressions as a language for pattern specification, (ii) to present tokenization and normalization algorithms that reduce textual variance, and (iii) to derive the minimum edit distance algorithm for quantifying string similarity.

1.2 The Data Collection Pipeline

Text processing follows a systematic pipeline from raw data to structured representation:

```
Data Sources → Preliminary Formatting → Tokenization → Normalization → Analysis
```

Data Sources: Text may originate from structured databases (CSV, JSON), web scraping, APIs (Twitter, Reddit), or document collections. Each source introduces characteristic noise and formatting challenges.

Preliminary Formatting: Raw text often contains HTML markup, encoding artifacts, or platform-specific formatting that must be stripped or converted.

Tokenization: The continuous character stream is segmented into discrete units (tokens) suitable for further processing.

Normalization: Variant forms are mapped to canonical representations, reducing vocabulary size and improving generalization.

The Garbage-In-Garbage-Out Principle: Data quality fundamentally constrains analytical outcomes. User-generated text exhibits substantial creativity and variance—typos, abbreviations, character flooding, dialect features—that can dramatically inflate vocabulary size and degrade similarity calculations. Systematic preprocessing mitigates these effects.

2. Background

2.1 Text as Data

Text data presents unique challenges compared to structured numerical data:

Unstructured Format: Unlike database records with explicit schemas, text requires inference of structure through tokenization and parsing.

Language Variation: Text exhibits variation across multiple dimensions:

- Abbreviations and acronyms (NLP, lol, u)
- Capitalization conventions
- Character flooding (soooo, !!!!!)
- Concatenations and compounds
- Emoticons and emoji
- Dialect, slang, and code-switching
- Typographical errors

Platform-Specific Artifacts: Web-scraped data contains HTML tags, Reddit/Twitter posts include platform-specific markup, and API responses embed metadata in structured formats (JSON, XML).

2.2 Corpora and Language Variation

A **corpus** (plural: **corpora**) is a computer-readable collection of text or speech. Important considerations in corpus construction include:

Language and Dialect: The world contains over 7,000 languages (Simons & Fennig, 2018). Even within a single language, substantial dialectal variation exists—for example, African American English (AAE) features constructions like *iont* (corresponding to Mainstream American English *I don't*) that affect tokenization.

Code-Switching: Speakers frequently mix languages within single utterances:

> "Por primera vez veo a @username actually being hateful!"
> [For the first time I get to see @username actually being hateful!]

Genre: Text characteristics vary substantially across newswire, fiction, scientific articles, social media, spoken transcripts, and legal documents.

Temporal Variation: Language evolves; historical texts require period-appropriate processing.

Documentation Standards: Best practice calls for **datasheets** (Gebru et al., 2020) or **data statements** (Bender et al., 2021) documenting corpus motivation, collection process, language variety, speaker demographics, and annotation procedures.

---

3. Regular Expressions

3.1 Fundamentals and Notation

A **regular expression** (regex) is an algebraic notation for characterizing a set of strings. Regular expressions enable:

- **Search**: Finding strings matching a pattern within a corpus
- **Extraction**: Isolating substrings matching specified criteria
- **Substitution**: Replacing matched patterns with alternative strings

We denote regular expressions within forward slashes: `/pattern/`. The pattern `/woodchuck/` matches any string containing the substring "woodchuck."

Regular expressions are processed by **regex engines** that compile patterns into efficient finite-state automata. Major implementations exist in programming languages (Python `re`, JavaScript), text editors (vim, Emacs), and command-line tools (grep, sed).

3.2 Basic Patterns: Concatenation and Disjunction

Concatenation: Characters placed in sequence form a pattern matching that exact sequence. The pattern `/abc/` matches strings containing "abc."

Case Sensitivity: Regular expressions are case-sensitive by default. `/the/` matches "the" but not "The."

Disjunction with Brackets: Square brackets specify a set of alternative characters:

Pattern	Matches	Example
`/[wW]oodchuck/`	"woodchuck" or "Woodchuck"	"Woodchuck"
`/[abc]/`	'a', 'b', or 'c'	"cab"
`/[0-9]/`	Any single digit	"test5"

Character Ranges: The dash specifies ranges within brackets:

Pattern	Matches
`/[A-Z]/`	Any uppercase letter
`/[a-z]/`	Any lowercase letter
`/[0-9]/`	Any digit
`/[A-Za-z]/`	Any letter

3.3 Negation and Character Classes

Negation with Caret: When `^` appears as the first character inside brackets, it negates the character class:

Pattern	Matches
`/[^A-Z]/`	Any character except uppercase letters
`/[^0-9]/`	Any non-digit character
`/[^aeiou]/`	Any character except lowercase vowels

Important: The caret only indicates negation when it is the *first* character after `[`. In other positions, it represents a literal caret character.

Predefined Character Classes (aliases):

Alias	Expansion	Meaning
`\d`	`[0-9]`	Any digit
`\D`	`[^0-9]`	Any non-digit
`\w`	`[a-zA-Z0-9_]`	Any "word" character
`\W`	`[^\w]`	Any non-word character
`\s`	`[ \t\n\r\f]`	Any whitespace
`\S`	`[^\s]`	Any non-whitespace

3.4 Kleene Operators and Quantifiers

Quantifiers specify how many times a pattern element may occur:

The Kleene Star (`*`): Zero or more occurrences of the preceding element.

$$/a*/ \text{ matches } \varepsilon, a, aa, aaa, \ldots$$

where ε denotes the empty string.

The Kleene Plus (`+`): One or more occurrences of the preceding element.

$$/a+/ \text{ matches } a, aa, aaa, \ldots \text{ (but not } \varepsilon)$$

The Optional Operator (`?`): Zero or one occurrence.

$$/colou?r/ \text{ matches } color, colour$$

Explicit Counters:

Pattern	Meaning
`{n}`	Exactly n occurrences
`{n,m}`	From n to m occurrences
`{n,}`	At least n occurrences
`{,m}`	At most m occurrences

The Wildcard (`.`): Matches any single character except newline.

$$/beg.n/ \text{ matches } begin, begun, beg'n, \ldots$$

Combined Pattern: The expression `/.*/` matches any string of characters.

3.5 Anchors and Word Boundaries

Anchors constrain where in the string a pattern may match:

Anchor	Meaning
`^`	Start of line/string
`$`	End of line/string
`\b`	Word boundary
`\B`	Non-word boundary

Examples:

- `/^The/` matches "The" only at line start
- `/end$/` matches "end" only at line end
- `/\bthe\b/` matches "the" but not "other" or "there"

Word Boundary Definition: A word boundary `\b` occurs between a word character (`\w`) and a non-word character (`\W`), or at string boundaries adjacent to word characters.

3.6 Grouping, Precedence, and Capture Groups

Operator Precedence (highest to lowest):

Precedence	Operators
1 (highest)	Parentheses `()`
2	Counters `*`, `+`, `?`, `{}`
3	Sequences and anchors
4 (lowest)	Disjunction `|`

Grouping with Parentheses: Parentheses override default precedence:

- `/the*/` matches "th" followed by zero or more 'e's: th, the, thee, theee, ...
- `/(the)*/` matches zero or more occurrences of "the": ε, the, thethe, ...

The Disjunction Operator (`|`): Matches either the left or right pattern:

$$/cat|dog/ \text{ matches ”cat” or ”dog”}$$

Combined with grouping:

$$/gupp(y|ies)/ \text{ matches ”guppy” or ”guppies”}$$

Capture Groups: Parentheses also create capture groups, storing matched substrings in numbered registers:

```
s/([0-9]+)/<\1>/
```

This substitution wraps integers in angle brackets, with `\1` referring to the first captured group.

Backreferences: Capture groups enable matching repeated patterns:

```
/the (.*)er they were, the \1er they will be/
```

This matches "the bigger they were, the bigger they will be" but not "the bigger they were, the faster they will be."

Non-Capturing Groups: The syntax `(?:pattern)` groups without capturing:

```
/(?:some|a few) (people|cats) like some \1/
```

3.7 Greedy versus Non-Greedy Matching

Greedy Matching (default): Quantifiers match as much text as possible.

Given text "the cat sat on the mat" and pattern `/the.*the/`, the match is "the cat sat on the" (maximal).

Non-Greedy (Lazy) Matching: Adding `?` after a quantifier makes it match minimally:

Greedy	Non-Greedy	Behavior
`*`	`*?`	Match as few as possible
`+`	`+?`	Match as few as possible
`?`	`??`	Prefer zero over one

3.8 Lookahead Assertions

Lookahead assertions test whether a pattern matches ahead without consuming characters:

Positive Lookahead `(?=pattern)`: Succeeds if pattern matches ahead.

Negative Lookahead `(?!pattern)`: Succeeds if pattern does not match ahead.

Example: Match words not starting with "Volcano":

$$\texttt{/\textbackslash\textasciicircum(?!Volcano)[A-Za-z]+/}$$

These are **zero-width** assertions—they don't advance the match position.

3.9 Substitution and ELIZA

Substitution: The operation `s/pattern/replacement/` replaces matched text:

$$s/colour/color/ \quad \text{(British} \rightarrow \text{American spelling)}$$

ELIZA: The classic chatbot (Weizenbaum, 1966) demonstrates regex-based dialogue:

```
User:  Men are all alike.
ELIZA: IN WHAT WAY
User:  They're always bugging us about something.
ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE
User:  He says I'm depressed much of the time.
ELIZA: I AM SORRY TO HEAR YOU ARE DEPRESSED
```

ELIZA implements cascaded substitution rules:

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1?/
s/.* all .*/IN WHAT WAY?/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/
```

First, input transformations convert first-person to second-person (MY → YOUR, I'M → YOU ARE). Then pattern-matching rules generate contextually appropriate responses.

4. Words and Tokenization

4.1 What Counts as a Word?

The definition of "word" varies by application and language:

Punctuation: Should periods, commas, and quotation marks be separate tokens? For parsing and part-of-speech tagging, punctuation typically forms distinct tokens.

Contractions: Should "don't" be one token or two (*do* + *n't*)? The Penn Treebank standard separates clitics: *doesn't* → *does* + *n't*.

Multi-word Expressions: Should "New York" or "rock 'n' roll" be single tokens? This depends on downstream applications.

Disfluencies: In spoken language, fragments (*main*-) and fillers (*uh*, *um*) may be retained or removed depending on the task.

Non-Spaced Languages: Chinese, Japanese, and Thai lack whitespace word boundaries, requiring specialized segmentation algorithms.

4.2 Types versus Instances

Word Types ($|V|$): The number of distinct words in the vocabulary.

Word Instances (N): The total count of running words (tokens).

Example: "They picnicked by the pool, then lay back on the grass and looked at the stars."

- Ignoring punctuation: 16 instances, 14 types
- "the" appears 3 times (3 instances, 1 type)

Corpus	Types ($\|V\|$)	Instances (N)
Shakespeare	31,000	884,000
Brown corpus	38,000	1,000,000
Switchboard	20,000	2,400,000
COCA	2,000,000	440,000,000
Google n-grams	13,000,000	1,000,000,000,000

4.3 Herdan's Law (Heaps' Law)

The relationship between vocabulary size and corpus size follows a power law:

$$|V| = kN^{\beta}$$

where:

- $|V|$ = vocabulary size (number of types)
- N = corpus size (number of instances)
- k = constant
- β = exponent, typically $0.67 < \beta < 0.75$

Interpretation: Vocabulary grows sublinearly with corpus size, but faster than the square root. Doubling corpus size does not double vocabulary—many new tokens are repetitions of existing types—but novel words continue to appear.

Implication: Larger corpora always contain words unseen in smaller samples, motivating subword tokenization approaches.

4.4 Top-Down Tokenization

Rule-based tokenization applies deterministic algorithms, typically implemented as cascaded regular expressions or finite-state automata.

Penn Treebank Standard:

- Separates clitics: *doesn't* → *does* + *n't*
- Preserves hyphenated words: *San Francisco-based*
- Separates punctuation: periods, commas, quotes become tokens

Example:

Input: `"The San Francisco-based restaurant," they said, "doesn't charge $10".`

Output: `" The San_Francisco-based restaurant , " they said , " does n't charge $ 10 " .`

NLTK Regular Expression Tokenizer:

```
pattern = r'''(?x)
    (?:[A-Z]\.)+          # abbreviations, e.g. U.S.A.
  | \w+(?:-\w+)*          # words with optional internal hyphens
  | \$?\d+(?:\.\d+)?%?    # currency, percentages
  | \.\.\.                # ellipsis
  | [][.,;"'?():_`-]      # punctuation
'''
```

4.5 Subword Tokenization: Byte-Pair Encoding

Motivation: Unknown words (words not seen in training) present challenges for NLP systems. Subword tokenization addresses this by segmenting words into smaller units that may be recombined.

Byte-Pair Encoding (BPE) (Sennrich et al., 2016) is a data-driven algorithm with two phases:

Token Learner: Iteratively merges frequent adjacent symbol pairs.

Algorithm:

```
function BPE(corpus C, number of merges k):
    V ← all unique characters in C
    for i = 1 to k:
        (t_L, t_R) ← most frequent adjacent pair in C
        t_NEW ← t_L + t_R
        V ← V ∪ {t_NEW}
        Replace all (t_L, t_R) in C with t_NEW
    return V
```

Worked Example:

Initial corpus with word frequencies:

```
5 low_    2 lowest_    6 newer_    3 wider_    2 new_
```

Initial vocabulary: `{_, d, e, i, l, n, o, r, s, t, w}`

Iteration 1: Most frequent pair is `(e, r)` with count 9 (newer: 6, wider: 3). Merge to `er`.

Iteration 2: Most frequent is `(er, _)` with count 9. Merge to `er_`.

Iteration 3: `(n, e)` has count 8. Merge to `ne`.

Continuing yields vocabulary:

```
{_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_}
```

Token Segmenter: Applies learned merges to new text in the order they were learned.

For unknown word *lower_*: segments to `low` + `er_` using learned subwords.

Properties:

- Most common words remain whole tokens
- Rare words decompose into subword units
- Handles unseen words through character-level fallback

---

5. Text Normalization

5.1 Case Folding

Case folding maps all characters to a uniform case (typically lowercase):

$$\text{The} \to \text{the}, \quad \text{IMPORTANT} \to \text{important}$$

Advantages: Reduces vocabulary size; improves generalization for information retrieval and speech recognition.

Disadvantages: Loses potentially useful distinctions:

- *US* (country) vs. *us* (pronoun)
- Proper nouns vs. common nouns

Practice: Many systems maintain both cased and uncased model variants.

5.2 Lemmatization

Lemmatization maps inflected forms to their base **lemma** (dictionary form):

$$am, are, is \to be$$

$$sang, sung, sings \to sing$$

$$cats, cat's \rightarrow cat$$

Example: "He is reading detective stories" → "He be read detective story"

Morphological Analysis: Full lemmatization requires parsing words into **morphemes**:

- **Stem**: Core meaning-bearing unit (*wash* in *unwashable*)
- **Affixes**: Meaning-modifying attachments (*un-*, *-able*)

Importance: Essential for morphologically rich languages (Arabic, Polish) where a single lemma may have dozens of surface forms.

5.3 Stemming: The Porter Algorithm

Stemming is a simplified, rule-based approximation to lemmatization that strips affixes without full morphological analysis.

The Porter Stemmer (Porter, 1980) applies cascaded rewrite rules:

Rule	Example
ATIONAL → ATE	relational → relate
ING → ε (if stem has vowel)	motoring → motor
SSES → SS	grasses → grass
IES → I	ponies → poni

Example Output:

Input: "This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things..."

Stemmed: "Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing..."

Limitations:

- **Over-generalization**: *policy* → *polic* (conflated with *police*)
- **Under-generalization**: *European* not mapped to *Europe*

Trade-off: Stemming is fast and simple but less accurate than full lemmatization.

5.4 Sentence Segmentation

Sentence segmentation identifies sentence boundaries, typically marked by:

- Periods (.)
- Question marks (?)
- Exclamation points (!)

Ambiguity: Periods are ambiguous between sentence boundaries and abbreviations:

- "Dr. Smith arrived." — period after "Dr" is abbreviation, after "arrived" is boundary
- "The U.S.A. won." — both uses in single sentence

Approaches:

- Abbreviation dictionaries (hand-built or learned)
- Machine learning classifiers for boundary detection
- Rule-based systems in tools like Stanford CoreNLP

---

6. System Evaluation

6.1 Error Types: False Positives and False Negatives

Evaluation of text processing systems distinguishes two error types:

False Positives (FP): Incorrectly matched items (matched when should not have).

False Negatives (FN): Missed items (not matched when should have).

Example: Regex `/[tT]he/` to find the word "the":

- False positives: matches "there", "then", "other" (embedded occurrences)
- False negatives: misses "THE" (all caps)

Tension: Reducing one error type often increases the other. Broadening patterns captures more true positives but increases false positives; narrowing patterns reduces false positives but increases false negatives.

6.2 The Confusion Matrix

For binary classification, outcomes are organized in a **confusion matrix**:

	Predicted Positive ($\hat{y} = 1$)	**Predicted Negative ($\hat{y} = 0$)**
Actual Positive ($y = 1$)	True Positive (TP)	False Negative (FN)
Actual Negative ($y = 0$)	False Positive (FP)	True Negative (TN)

- **True Positive (TP)**: Correctly identified positives
- **True Negative (TN)**: Correctly identified negatives
- **False Positive (FP)**: Type I error (false alarm)
- **False Negative (FN)**: Type II error (miss)

6.3 Precision, Recall, and F-Score

Accuracy: Overall correctness.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: Of items labeled positive, what fraction are correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (Sensitivity): Of actual positives, what fraction were found?

$$\text{Recall} = \frac{TP}{TP + FN}$$

F-Score: Harmonic mean of precision and recall.

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

Generalized F-Score with parameter β:

$$F_\beta = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta^2 \cdot FN + FP}$$

- $\beta = 1$: Equal weight to precision and recall

- $\beta > 1$: Weights recall higher
- $\beta < 1$: Weights precision higher

6.4 Information Retrieval Metrics

Mean Reciprocal Rank (MRR): For queries with ranked results, measures the position of the first relevant result.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

where $|Q|$ is the number of queries and rank_i is the rank of the first relevant result for query i.

Average Precision (AP):

$$\text{AP} = \frac{\sum_{k=1}^{n} P(k) \cdot \text{rel}(k)}{\text{number of relevant items}}$$

where $\text{rel}(k)$ is an indicator (1 if item at rank k is relevant, 0 otherwise) and $P(k)$ is precision at rank k.

7. Minimum Edit Distance

7.1 Definition and Motivation

Edit distance quantifies the similarity between two strings based on the number of operations required to transform one into the other.

Applications:

- **Spelling correction**: Finding intended words from misspellings (*graffe* → *giraffe*)
- **Coreference resolution**: Determining if strings refer to the same entity
- **Speech recognition**: Computing word error rate
- **Bioinformatics**: Sequence alignment

Minimum Edit Distance: The minimum number of editing operations (insertions, deletions, substitutions) required to transform source string X into target string Y.

7.2 Levenshtein Distance

Levenshtein distance assigns uniform costs to operations:

Standard Version:

- Insertion: cost 1
- Deletion: cost 1
- Substitution: cost 1 (or 0 if characters match)

Alternative Version (used in some formulations):

- Insertion: cost 1
- Deletion: cost 1
- Substitution: cost 2 (equivalent to deletion + insertion)

Alignment Visualization:

Transforming *intention* to *execution*:

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

Operations: delete I, substitute N→E, substitute T→X, insert C, substitute N→U.

Edit distance: 5 (with cost-1 substitutions).

7.3 The Dynamic Programming Algorithm

Dynamic programming solves problems by combining solutions to subproblems, storing intermediate results to avoid recomputation.

Formulation:

Let $X[1..n]$ be the source string of length n and $Y[1..m]$ be the target string of length m.

Define $D[i, j]$ as the minimum edit distance between $X[1..i]$ and $Y[1..j]$.

Base Cases:

- $D[0,0] = 0$ (empty string to empty string)
- $D[i,0] = i$ (delete all i characters from source)
- $D[0,j] = j$ (insert all j characters into empty source)

Recurrence Relation:

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(X[i]) \\ D[i,j-1] + \text{ins-cost}(Y[j]) \\ D[i-1,j-1] + \text{sub-cost}(X[i],Y[j]) \end{cases}$$

For standard Levenshtein with substitution cost 2:

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

Result: $D[n,m]$ contains the minimum edit distance.

Complexity: Time $O(nm)$, Space $O(nm)$ (reducible to $O(\min(n,m))$ if only distance needed).

7.4 Worked Example: Computing the Edit Distance Matrix

Source: *intention* (n=9)
Target: *execution* (m=9)
Cost model: Insertion/deletion = 1, Substitution = 2, Match = 0

Matrix Construction:

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

	#	e	x	e	c	u	t	i	o	n
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	**8**

Result: Minimum edit distance = 8 (using substitution cost 2).

7.5 Alignment and Backtrace

Alignment: A correspondence between positions in two strings showing which characters match, are substituted, inserted, or deleted.

Backtrace: To recover the optimal alignment:

1. Store **backpointers** during matrix computation, recording which cell(s) contributed the minimum
2. Starting from $D[n, m]$, follow pointers back to $D[0, 0]$
3. Path direction indicates operation:
    - Diagonal (↖): Match or substitution
    - Left (←): Insertion
    - Up (↑): Deletion

Multiple Optimal Paths: When multiple cells yield the same minimum, multiple optimal alignments exist.

8. Conclusion

This chapter has established the preprocessing foundations essential for text analysis:

Key Contributions:

1. **Regular Expressions**: A formal language for pattern specification enabling precise text search, extraction, and transformation. Core operations include concatenation, disjunction, Kleene closure, anchoring, and grouping.

2. **Tokenization**: The segmentation of text into discrete units. Top-down approaches apply deterministic rules; bottom-up methods like BPE automatically induce vocabularies from data, addressing the unknown word problem.
3. **Vocabulary Growth**: Heaps' Law ($|V| = kN^\beta$) characterizes the sublinear relationship between corpus size and vocabulary, motivating subword approaches.
4. **Text Normalization**: Operations including case folding, lemmatization, and stemming reduce textual variance, improving generalization at the potential cost of information loss.
5. **Evaluation Framework**: The confusion matrix provides the foundation for precision, recall, and F-score metrics that quantify system performance while distinguishing error types.
6. **Minimum Edit Distance**: A dynamic programming algorithm computing optimal string transformations, with applications throughout NLP including spelling correction and sequence alignment.

These preprocessing steps precede the classification and representation learning methods to be covered in subsequent chapters, ensuring that text data is in suitable form for computational analysis.

9. References

1. Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing* (3rd ed. draft). Retrieved from https://web.stanford.edu/~jurafsky/slp3/
2. Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3), 130–137.
3. Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. *Proceedings of ACL*, 1715–1725.
4. Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1), 36–45.
5. Heaps, H. S. (1978). *Information Retrieval: Computational and Theoretical Aspects*. Academic Press.
6. Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8), 707–710.

7. Gebru, T., et al. (2020). Datasheets for datasets. *Communications of the ACM*, 64(12), 86–92.

8. Bender, E. M., & Friedman, B. (2018). Data statements for natural language processing. *TACL*, 6, 587–604.

Appendix A: Regular Expression Quick Reference

Pattern	Meaning	Example	Matches
`.`	Any character (except newline)	`n.p`	nop, nlp, n9p
`[abc]`	Character class (any of a, b, c)	`analy[zs]e`	analyze, analyse
`[a-z]`	Character range	`[A-Z][a-z]+`	Hello, World
`[^abc]`	Negated class	`[^0-9]+`	abc, xyz
`|`	Disjunction (or)	`cat|dog`	cat, dog
`^`	Start of string/line	`^The`	The (at start)
`$`	End of string/line	`end$`	end (at end)
`?`	Zero or one	`colou?r`	color, colour
`*`	Zero or more	`a*b`	b, ab, aab
`+`	One or more	`a+b`	ab, aab, aaab
`{n}`	Exactly n	`a{3}`	aaa
`{n,m}`	n to m occurrences	`a{2,4}`	aa, aaa, aaaa
`{n,}`	At least n	`a{2,}`	aa, aaa, aaaa, …
`()`	Grouping / capture	`(ab)+`	ab, abab
`(?:)`	Non-capturing group	`(?:ab)+`	ab, abab
`(?=)`	Positive lookahead	`a(?=b)`	a (before b)
`(?!)`	Negative lookahead	`a(?!b)`	a (not before b)
`\b`	Word boundary	`\bthe\b`	the (whole word)
`\d`	Digit `[0-9]`	`\d+`	123, 45
`\w`	Word char `[a-zA-Z0-9_]`	`\w+`	hello_123
`\s`	Whitespace	`\s+`	spaces, tabs
`\`	Escape special char	`\.`	. (literal)

Appendix B: Minimum Edit Distance Algorithm Pseudocode

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance:
    n ← LENGTH(source)
    m ← LENGTH(target)

    // Create distance matrix D[0..n, 0..m]
    D[0,0] ← 0

    // Initialize first column (deletions from source)
    for i from 1 to n:
        D[i,0] ← D[i-1,0] + del-cost(source[i])

    // Initialize first row (insertions to empty source)
    for j from 1 to m:
        D[0,j] ← D[0,j-1] + ins-cost(target[j])

    // Fill matrix using recurrence relation
    for i from 1 to n:
        for j from 1 to m:
            D[i,j] ← MIN(
                D[i-1,j] + del-cost(source[i]),        // deletion
                D[i,j-1] + ins-cost(target[j]),        // insertion
                D[i-1,j-1] + sub-cost(source[i], target[j])  // substitution
            )

    return D[n,m]
```

Cost Functions (Levenshtein):

- `del-cost(c) = 1` for all characters
- `ins-cost(c) = 1` for all characters
- `sub-cost(c1, c2) = 0` if c1 = c2, else 2 (or 1 in some formulations)

Glossary

Alignment: A correspondence between positions in two strings showing the relationship between characters.

Anchor: A regex element matching a position rather than a character (e.g., `^`, `$`, `\b`).

Backtrace: The process of following pointers backward through a dynamic programming matrix to recover the optimal solution path.

Byte-Pair Encoding (BPE): A subword tokenization algorithm that iteratively merges frequent adjacent symbol pairs.

Capture Group: Parentheses in a regex that store the matched substring in a numbered register.

Case Folding: Converting all characters to a uniform case (typically lowercase).

Clitic: A morpheme that cannot stand alone and must attach to another word (e.g., *n't* in *don't*).

Confusion Matrix: A table organizing classification outcomes into true/false positives/negatives.

Corpus: A collection of text or speech data (plural: corpora).

Disjunction: A regex operator (`|` or `[]`) matching any of several alternatives.

Dynamic Programming: An algorithmic paradigm solving problems by combining solutions to subproblems.

Edit Distance: A measure of string similarity based on required transformation operations.

F-Score: The harmonic mean of precision and recall.

False Negative (FN): An item incorrectly classified as negative.

False Positive (FP): An item incorrectly classified as positive.

Greedy Matching: Regex behavior matching as much text as possible.

Heaps' Law: The relationship $|V| = kN^\beta$ between vocabulary size and corpus size.

Kleene Star: The regex operator `*` matching zero or more occurrences.

Lemma: The canonical dictionary form of a word.

Lemmatization: Mapping inflected forms to their lemma.

Levenshtein Distance: Edit distance with unit costs for insertion, deletion, and substitution.

Lookahead Assertion: A zero-width regex test for upcoming patterns.

Morpheme: The smallest meaning-bearing unit of language.

Precision: The fraction of predicted positives that are correct: TP/(TP+FP).

Recall: The fraction of actual positives that were found: TP/(TP+FN).

Regular Expression (Regex): An algebraic notation for specifying string patterns.

Stemming: Stripping affixes to reduce words to approximate stems.

Subword Token: A token representing part of a word, used to handle unknown words.

Tokenization: Segmenting text into discrete units (tokens).

Word Instance: A single occurrence of a word in text.

Word Type: A unique word in the vocabulary.