

Text and Sequence Analytics

Compiled Revision Notes

Table of Contents

Week 1: Introduction to Text & Sequence Analytics	3
Fundamentals of Text Analytics	3
The DIKW Pyramid	4
Artificial Intelligence: Capabilities & Branches	4
Sequence Analysis & Bioinformatics	4
Historical Context & Thought Experiments	5
Modern Tools & Frameworks	5
Week 2: Text Processing & Normalization	5
Text Representation	5
Regular Expressions (Regex)	6
Text Normalization	6
Key Stemming Algorithms	6
Part of Speech (POS) Tagging	6
Performance Metrics & Data Structures	7
Week 3: Tokenization & Sequence Segmentation	7
Text Tokenization Basics	7
Sub-Word Tokenization	7
Sequence Segmentation	9
SentencePiece Framework	10
Week 4: Sequence Similarity & Alignment	10
Mathematical Fundamentals of Similarity	10
Distance Calculations	11
Sequence Alignment	12
Advanced Scoring Parameters	12
BLAST Statistical Significance	13

Week 1: Introduction to Text & Sequence Analytics

Fundamentals of Text Analytics

Text analytics, or **text mining**, is the process of extracting high-quality information from unstructured text. It differs from standard data processing because text is inherently ‘noisy’ and lacks a regular syntax or pattern.

- **Definition:** Transforming unstructured text into useful data through NLP, algorithms, and analytical methods.
- **The 3 Vs:** The vast amount of data created yearly (Volume, Velocity, Variety) necessitates modern storage like NoSQL and Vector Databases.
- **Core Tasks:**
 - Translation and Language Identification
 - Sentiment Analysis and Topic Modelling
 - Classification and Clustering

The DIKW Pyramid

This hierarchy illustrates how we move from raw facts to actionable insight. In exams, this is often used to explain the goal of information management vs. knowledge management.

Level	Description	Focus
Wisdom	Understanding ‘Why’; used for high-level decision making	Know Why
Knowledge	Finding patterns, rules, and building predictive models	Know How
Information	Data that has been categorized, combined, and calculated	Know What
Data	Raw facts, figures, and measurements	Raw Input

Artificial Intelligence: Capabilities & Branches

AI is categorized by its ‘reach’ relative to human intelligence.

AI by Capability

- **Artificial Narrow Intelligence (ANI):** Specialized in one task (e.g., Alexa, Siri, or Generative AI).
- **Artificial General Intelligence (AGI):** Can perform any task a human can.
- **Artificial Super Intelligence (ASI):** Surpasses all human capabilities.

The Seven Branches (IEEE)

- **Machine Learning (ML):** Statistical models that learn from data.
- **Natural Language Processing (NLP):** Understanding and generating human language.
- **Neural Networks:** Models based on the human brain (interconnected nodes).
- **Computer Vision:** Interpreting visual or image data.
- **Robotics:** Programming machines for autonomous tasks.
- **Expert Systems:** Simulating human expertise in specific fields.
- **Fuzzy Logic:** Handling vagueness and uncertainty in reasoning.

Sequence Analysis & Bioinformatics

A unique aspect of this course is the crossover between human language and biological ‘languages’ like DNA.

- **Goal:** Analyzing DNA and protein sequences to discover biological insights.
- **The Connection:** Biological sequences use symbols (like English) where the order is the most critical factor, just like NLP.
- **Key Growth:** The volume of data in GenBank (Nucleotides, Amino Acids) has seen exponential growth over 30 years.

Historical Context & Thought Experiments

Dr. Healy highlights several milestones that defined the field:

- **Memex (1945):** Vannevar Bush's concept for 'associative memory' storage, the precursor to hypertext.
 - **Turing Test (1950):** A test of whether a machine can 'imitate' a human well enough to fool an interrogator.
 - **Searle's Chinese Room (1980):** A critique of 'Strong AI'. It argues that a machine can manipulate symbols (syntax) without ever understanding their meaning (semantics).
 - **AI Winters:**
 - 1st AI Winter: Triggered by the 1966 ALPAC report.
 - 2nd AI Winter: Mid-1980s to early 1990s.
-

Modern Tools & Frameworks

To implement these theories, the industry uses specific Python and Java ecosystems.

- **General ML:** scikit-learn, PyTorch, TensorFlow, Keras.
 - **NLP Specific:** NLTK, spaCy, Gensim.
 - **LLM Ecosystem:** Hugging Face, LangChain, and models like GPT, Gemini, and Llama.
 - **Data Interchange:** ONNX (Open Neural Network Exchange) allows models to move between different languages like Python and Java.
-

Week 2: Text Processing & Normalization

Text Representation

Computers do not process characters directly; they represent them as numeric values.

- **ASCII:** A basic numeric mapping for 128 characters, including control characters (like 'Line Feed') and standard Latin letters.
- **ISO-8859-1 (Latin 1):** An extension of ASCII standardized in 1987 to include 256 characters, covering most Latin and Greek alphabets.
- **Unicode:** The modern standard for world languages.
 - **UTF-8:** Uses 1 byte for basic characters; compatible with ISO-8859-1.
 - **UTF-16:** Uses 2 bytes (16 bits) per character, allowing for 65,536 characters, including ancient scripts like Ogham.

Java Strings

- Java uses UTF-16 internally for `char` and `String`.
 - **Immutability:** Once a `String` is declared, it cannot be changed; modifying it actually creates a new object to improve performance.
 - **String Pooling:** Literal strings (e.g., `String t = "Happy Days!"`) are shared in a 'String Pool' in the JVM heap to save memory.
-

Regular Expressions (Regex)

Developed in the 1950s, Regex is the ‘de facto’ standard for pattern matching and text manipulation.

Symbol	Meaning	Example
.	Any single character	a.b → acb
\d	Any digit (0-9)	\d{4} → 1970
\w	Word character (a-z, 0-9)	\w+ → Hello123
^ / \$	Start / End of string	^Hi / end\$
[^a-zA-Z]	Match only letters	Useful for removing numbers or symbols

Text Normalization

Before analysis, text must be ‘wrangled’ or cleaned through a series of steps:

- **Tokenization:** Decomposing text into minimal meaningful units like words or sentences.
 - **Morpheme Analysis:** Breaking words into stems and affixes (prefixes or suffixes) to find the ‘base’ meaning.
 - **Stemming:** A heuristic ‘affix stripper’ that reduces a word to a root.
 - **Lemmatization:** A more accurate, context-aware method that uses a dictionary (like WordNet) to find the valid base form (lemma).
 - **Stop Word Removal:** Filtering out noisy, insignificant words (like ‘the’, ‘and’) that often make up 25%+ of a document.
-

Key Stemming Algorithms

Stemming is a trade-off between speed and accuracy.

- **Porter’s Algorithm (1980):** The most widely used; classifies characters as Vowels (v) or Consonants (c) and applies a 5-step rule list.
- **Snowball Stemmer:** An updated version of Porter’s algorithm supporting multiple languages.
- **Lancaster (Paice-Husk) Stemmer:** A ‘greedy’ iterative rules-based approach.

Common Issues:

- **Over-stemming:** Too aggressive; stems ‘university’ to ‘universe’.
 - **Under-stemming:** Too lazy; fails to stem ‘knavish’ to ‘knave’.
-

Part of Speech (POS) Tagging

POS tagging labels words with lexical classes (noun, verb, etc.) based on their role in a sentence.

- **Penn Treebank:** The standard notation for tagging.
- **Common Tags:**

- NN: Noun, singular
 - VB: Verb, base form
 - JJ: Adjective
 - CD: Cardinal number
 - PRP: Personal pronoun
-

Performance Metrics & Data Structures

- **Bloom Filter:** A probabilistic data structure used to test set membership efficiently; ideal for representing large sets of stop words or computing similarity metrics while saving memory.
- **Index Compression Factor (ICF):** Measures a stemmer's strength by the percentage of distinct words it reduces.

$$\text{ICF} = \frac{n - s}{n} \times 100$$

where n is the number of distinct words before stemming and s is the number after stemming.

Week 3: Tokenization & Sequence Segmentation

Text Tokenization Basics

- **Definition:** Segmenting text into meaningful atomic units (tokens) such as characters, sub-words, or words.
- **Vocabulary:** The complete set of unique tokens used by a specific tokenizer.
- **Out-of-Vocabulary (OOV):** Words appearing in text that are missing from the tokenizer's pre-defined vocabulary.

Granularity Trade-offs

- **Word-level:** Often used for semantic analysis and Part-of-Speech tagging, but results in large vocabularies and frequent OOV misses.
 - **Character-level:** Useful for spelling correction and OOV handling, but creates long sequences that are computationally expensive to process.
-

Sub-Word Tokenization

Decomposes words into smaller units to handle OOV words and provide better morphological understanding while keeping vocabulary size manageable.

Byte-Pair Encoding (BPE)

- **Concept:** A data compression algorithm adapted for tokenization by iteratively merging the most frequent adjacent pairs.
- **Models:** Used in GPT-2, RoBERTa, and XLM.

BPE Training Algorithm

1. Initialize vocabulary: $V = \text{Set of characters in } S$
 2. **Do:**
 - Find: $(\text{Token}_{Left}, \text{Token}_{Right}) = \text{getMostFrequentAdjacentTokens}(S)$
 - Concatenate: $\text{Token}_{new} = \text{Token}_{Left} + \text{Token}_{Right}$
 - Update: $V = V + \text{Token}_{new}$
 - Replace all adjacent $(\text{Token}_{Left}, \text{Token}_{Right})$ in S with Token_{new} .
 3. **Loop Until maxMerges reached.**
 - **Complexity:**
 - Training time: $O(VC)$ where V is vocabulary size and C is corpus size.
 - Tokenization for text T : $O(T \log V)$
-

WordPiece

- **Concept:** Iterative sub-word tokenization similar to BPE but uses a likelihood scoring system for merges.
- **Models:** Primarily used by BERT.

Merging Score Formula

$$\text{score} = \frac{\text{freq}(\text{Token}_{Left}) \times \text{freq}(\text{Token}_{Right})}{\text{freq}(\text{Token}_{new})}$$

Markers

- Uses `##` to track sub-word positions (start, middle, or end) and preserve morphological structure.

Special Tokens

- `[CLS]`: Classification token at the start of every sequence.
- `[SEP]`: Separator for different text segments (e.g., Q&A).
- `[UNK]`: Unknown token for OOV words.

- [PAD]: Padding to reach fixed-size vector lengths.
 - [MASK]: Used in BERT pre-training to hide words for prediction.
-

Unigram Tokenization

- **Concept:** A probabilistic algorithm that starts with a large vocabulary and iteratively removes tokens that cause the least log-likelihood loss.
- **Base Rule:** Base characters are never removed to ensure all strings remain tokenizable.

Log Loss Formula

$$\text{loss} = \sum_{i=1}^n \log \left(\sum_{w \in S(w_i)} p(w) \right)$$

Where $S(w_i)$ is the set of all possible tokenizations of a word w_i .

- **Training Complexity:** $O(nv)$ where n is training examples and v is vocabulary size.
-

Sequence Segmentation

Division of already tokenized text into smaller, overlapping sub-sequences, often using a sliding window.

N-grams

- **Definition:** A sequence of n contiguous characters.
 - **Applications:** Spell checking, language modeling, and plagiarism detection.
 - **Biological Context:** Called **k-mers** when used with DNA or protein sequences.
 - **DNA Encoding:**
 - DNA sequences ($|\Sigma| = 4$) can be encoded with 2 bits:
 - * A: 00
 - * T: 01
 - * G: 10
 - * C: 11
 - A 64-bit long can encode a 32-mer.
-

Shingles

- **Definition:** A word or group of words, similar to n-grams but typically used for document-level analysis.
 - **Applications:** Near-duplicate detection, document clustering, and data deduplication.
-

Skip-grams

- **Definition:** Similar to n-grams but allows gaps (skips) between words.
 - **Value:** Captures semantic relationships between non-adjacent words, essential for creating dense word embeddings.
-

SentencePiece Framework

- **Origin:** Developed by Meta.

Features

- Implements optimized BPE, WordPiece, and Unigram (default).
- Reversible tokenization: (token id) \leftrightarrow (Unicode)
- Excellent for languages without whitespaces (Chinese/Japanese) because it treats the whole sentence as a Unicode stream.
- Training time is reduced to $O(n \log n)$ using an **Enhanced Suffix Array (ESA)**.

Week 4: Sequence Similarity & Alignment

Mathematical Fundamentals of Similarity

To compare sequences, we distinguish between measuring how much they are alike versus how much they differ.

Similarity (s)

Measures the degree of correspondence between sequences. Higher similarity implies a closer relationship.

Distance (d)

Measures the number of changes required to transform one sequence into another.

Identity (I)

A Boolean indicator function:

$$I(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$$

Metric Properties

For a distance function $d(x, y)$ to be a true metric, it must satisfy:

1. **Non-negativity:** $d(x, y) \geq 0$
 2. **Identity of Indiscernibles:** $d(x, y) = 0 \iff x = y$
 3. **Symmetry:** $d(x, y) = d(y, x)$
 4. **Triangle Inequality:** $d(x, z) \leq d(x, y) + d(y, z)$
-

Distance Calculations

These quantify the cost of transforming one string into another.

Hamming Distance

The number of positions at which corresponding symbols differ.

Constraint: Sequences must be of equal length.

$$d_H(s_1, s_2) = \sum_{i=1}^n \mathbf{1}[s_{1i} \neq s_{2i}]$$

where $\mathbf{1}[\cdot]$ is the indicator function.

Levenshtein (Edit) Distance

The minimum number of insertions, deletions, or substitutions required to transform string A into string B .

Let $d(i, j)$ denote the distance between the first i characters of A and the first j characters of B .

Recurrence Relation:

$$d(i, j) = \min \begin{cases} d(i-1, j) + 1 & \text{(Deletion)} \\ d(i, j-1) + 1 & \text{(Insertion)} \\ d(i-1, j-1) + \text{cost} & \text{(Substitution)} \end{cases}$$

$$\text{where cost} = \begin{cases} 0 & \text{if } A[i] = B[j] \\ 1 & \text{if } A[i] \neq B[j] \end{cases}$$

Sequence Alignment

Alignment algorithms use a scoring matrix H and defined transition rules to find an optimal alignment.

Needleman-Wunsch (Global Alignment)

Aligns sequences from beginning to end. Suitable for closely related sequences of similar length.

Initialization: $H(i, 0) = i \times \text{gap}$ and $H(0, j) = j \times \text{gap}$

Recurrence Relation:

$$H(i, j) = \max \begin{cases} H(i - 1, j - 1) + S(a_i, b_j) & (\text{Match/Mismatch}) \\ H(i - 1, j) + \text{gap} & (\text{Gap in B}) \\ H(i, j - 1) + \text{gap} & (\text{Gap in A}) \end{cases}$$

Time complexity: $O(mn)$

Smith-Waterman (Local Alignment)

Finds the highest scoring local subsequences within two sequences.

Initialization: $H(i, 0) = 0$ and $H(0, j) = 0$

Recurrence Relation:

$$H(i, j) = \max \begin{cases} 0 & (\text{Restart}) \\ H(i - 1, j - 1) + S(a_i, b_j) & (\text{Match/Mismatch}) \\ H(i - 1, j) + \text{gap} & (\text{Gap in B}) \\ H(i, j - 1) + \text{gap} & (\text{Gap in A}) \end{cases}$$

The inclusion of 0 prevents negative scores and allows the alignment to restart at any position.

Advanced Scoring Parameters

Affine Gap Penalty

More realistic than a linear gap penalty because opening a gap is typically more costly than extending one.

$$W = g + (L - 1)e$$

where:

- g = gap opening penalty
 - e = gap extension penalty
 - L = total length of the gap
-

BLAST Statistical Significance

BLAST evaluates whether an alignment is statistically meaningful.

Bit Score S'

A normalized alignment score independent of database size.

E-value (Expect Value)

The expected number of matches occurring by chance.

$$E = m \times n \times 2^{-S'}$$

where:

- m = query length
- n = database length
- S' = bit score

A very small or zero E-value indicates a statistically significant match.