Natural Language Processing

Corpora and Preprocessing

Ramaseshan Ramachandran

NATURAL LANGUAGE

- Ambiguous, context-dependent
- Expresses thoughts, emotions, ideas
- Multiple ways to convey same idea
- Handles sarcasm, metaphor, humor
- Can be poetic, narrative, abstract
- Possible to Express simple ideas in complex forms
- Non-algorithmic, non-deterministic

PROGRAMMING LANGUAGE

- Algorithmic and precise
- Deterministic execution
- Strict syntax and semantics
- Designed for unambiguous logic
- Single best way to express a given task
- Machine-readable instructions

THE GAP

- NL: Flexible, rich, human-oriented
- PL: Rigid, exact, machine-oriented
- Challenge: Mapping ambiguity to precision
- Requires preserving meaning while enforcing structure

Bridging the Gap

ONE REQUEST, WORKED END TO END

We will take a single English sentence and turn it into a program.

> “Show me the common words in this file.”

And here is the whole file. Fourteen words, so we can check every number by hand.

```
The cat sat on the mat. The Cat did not sit on the dog.
```

Any programmer would say the request is clear. It is not.

STEP 1: UNDERSTAND THE INTENT

- The person wants to know what this text is about
- They believe repetition is a clue to the topic
- So the program must count something, then rank it

That much is settled. Counting and ranking are the objective.
Everything still undecided lives inside three words.

STEP 2: CLARIFY THE AMBIGUITIES

The sentence has three loose words. Each one forces a decision.

Word	**Question the machine asks**	**A choice we must make**
word	Is `Cat` the same as `cat`?	Fold case, or keep it
word	Is `mat.` the same as `mat`?	Strip punctuation, or keep it
common	Does `the` count as a common word?	Drop stop words, or keep them
common	How many do you want back?	Top 3, top 10, or all
show	In what order do ties come out?	Alphabetical, or first seen

A human never asks these. A machine cannot start without the answers.

STEP 3: THREE HONEST READINGS OF THE SAME SENTENCE

Nothing here is wrong. Each reading obeys the English faithfully.

- **Reading A.** A word is whatever spaces separate. Keep everything
- **Reading B.** A word is a run of letters. Fold case, drop punctuation
- **Reading C.** Reading B, then remove stop words such as `the` and `on`

The tokens already differ before we count anything.

A	`The cat sat on the mat. The Cat did not sit on the dog.`
B	`the cat sat on the mat the cat did not sit on the dog`

STEP 4: THREE ANSWERS TO ONE QUESTION

Same file, same sentence, three programs, three results.

	Reading A	**Reading B**	**Reading C**
Tokens	14	14	6
Vocabulary	11	9	5
1st	`The` (2)	`the` (4)	`cat` (2)
2nd	`on` (2)	`cat` (2)	`dog` (1)
3rd	`the` (2)	`on` (2)	`mat` (1)
Rank of `cat`	5th (count 1)	2nd (count 2)	1st (count 2)

The text is about a cat. Only reading C says so.

WHY READING A LOSES THE CAT

Reading A splits on spaces and nothing else. So the animal arrives as two different words, and neither one is frequent.

Token	Count	Why
`cat`	1	sentence one, lower case
`Cat`	1	sentence two, capital after a full stop
`mat.`	1	the full stop is glued to the word
`dog.`	1	same problem at the end of the file

The capital letter is an accident of position. The full stop belongs to the sentence, not to the word. The machine cannot know either.

STEP 5: MAP THE DECISIONS TO STRUCTURE

Once the choices are named, the code is short and dull. That is the point.

```
# every line below is a decision from Step 2
text   = open(path).read()
text   = text.lower()                    # Cat == cat
tokens = re.findall(r"[a-z]+", text)     # mat. -> mat
tokens = [t for t in tokens
          if t not in STOPWORDS]         # drop the, on
counts = Counter(tokens)
rank   = lambda kv: (-kv[1], kv[0])      # ties: alphabetical
top    = sorted(counts.items(), key=rank)[:3]
```

Four English words became five decisions and seven lines.

STEP 6: TEST ON SOMETHING YOU CAN CHECK BY HAND

Fourteen words is a feature, not a toy.

- You can count `the` yourself and get 4
- So a program reporting 2 is wrong, and you know it in one second
- Then ask what breaks: `isn't`, `U.S.`, `re-run`, `3.14`, `#nlp`

Each of those is a new question of the same kind. Is this one word or two?
Refining the program means answering more of them, not writing more code.

STEP 7: WRITE THE DECISIONS DOWN

The program records what it does. It does not record what it chose to ignore.

- "Case folded, so `US` and `us` are now the same token"
- "Punctuation stripped, so `U.S.` became `u` and `s`"
- "Stop list has 10 words, tuned for topic finding, wrong for authorship"

Without these notes the next reader sees an answer and no assumptions.
The ambiguity comes back, hidden inside a number.

WHAT THE GAP ACTUALLY COSTS

- Four words of English hid five decisions
- Every decision changed the answer, and none of them was wrong
- `cat` ranked 5th, 2nd or 1st depending only on those choices

Bridging the gap is not translation. It is deciding what the sentence left out. Those five decisions have names. Tokenization, case folding, punctuation handling, stop word removal, tie breaking. They are the rest of this lecture.

Corpus Capital

IDEAL PROPERTIES OF A CORPUS

- Corpus is huge – Several billions of words
- Useful to verify a hypothesis about a language
- Find usage of a particular sound, word, or syntactic construction varies in different contexts
- Collection of most of the words of a language
- Even distribution of texts from all domains of language use

- The boys play cricket on the river bank.
- The boys play cricket by the side of a nationalized bank

LEXICAL RESOURCES

- The Brown Corpus contains a collection of written American English
- SUSANNE is a freely available, annotated subset of the Brown Corpus
- The Canadian Hansards, a bilingual parallel corpus, contains English and French transcripts of parliamentary proceedings
- The Penn Treebank contains syntactically annotated text from the Wall Street Journal
- NLP toolkits such as NLTK and spaCy ship with several corpora for learning purposes
- Hugging Face and Kaggle host thousands of text, image, and multimodal datasets
- Wikipedia dumps are available for hundreds of languages
- Web-scale corpora used for LLM pretraining: Common Crawl, C4, The Pile, RefinedWeb, FineWeb

DISAMBIGUATION OF BANK

Synset('bank.n.01') sloping land (especially the slope beside a body of water)
Synset('depository-financial-institution.n.01') a financial institution that accepts deposits and channels the money into lending activities
Synset('bank.n.03') a long ridge or pile
Synset('savings-bank.n.02') a container (usually with a slot in the top) for keeping money at home
Synset('bank.n.10') a flight maneuver (manoeuvre); aircraft tips laterally about its
longitudinal axis (especially in turning) Synset('bank.v.01') tip laterally
Synset('bank.v.02') enclose with a bank
Synset('bank.v.03') do business with a bank or keep an account at a bank
Synset('bank.v.04') act as the banker in a game or in gambling
Synset('bank.v.05') be in the banking business
Synset('deposit.v.02') put into a bank account
Synset('trust.v.01') have confidence or faith in

COMMON LAYERS OF NLP APPLICATIONS

- **Preprocessing layer**
- Data extraction layer
- Analysis of extracted information
- Semantic understanding
- Human/automatic evaluation of word meaning, sentence structure using the content obtained from the previous layers

OPERATIONS ON A CORPUS

- Text normalization converts text into a single canonical form: case folding, Unicode normalization, handling of foreign words, ...
- Tokenization divides input text into tokens by identifying word boundaries (modern LLMs use subword tokenizers such as BPE)
- Identification/extraction picks out particular tokens, sentences, and paragraphs
- Corpus statistics cover token count, vocabulary size and type–token ratio

Preprocessing

PREPROCESSING

- Corpus creation
- Modifications required to make raw text suitable for processing
- Understanding the problem is crucial in choosing the preprocessing steps
- Preprocessing steps are specific to the problem at hand
- Improper preprocessing may destroy lexical context (e.g., case folding loses “US” vs. “us”)
- Note: modern LLM pipelines do minimal preprocessing. They keep case and punctuation and rely on subword tokenization; aggressive cleaning is mostly for classical/statistical pipelines

COMMONLY USED PREPROCESSING STEPS FOR ENGLISH

Preprocessing consists of (a) tokenization, (b) normalization and (c) substitution

- Tokenization splits text into words/tokens
- Case folding converts all text to lower case
- Stemming turns running $\rightarrow$ run (crude suffix stripping)
- Lemmatization turns best $\rightarrow$ good (dictionary-based root form)
- Correct misspellings

- Strip punctuation
- Normalize white space, newlines, tabs, ...
- Expand contractions, so isn't $\rightarrow$ is not and I'd $\rightarrow$ I would
- Remove scripts and form variables from HTML/XML
- Remove stop words (task-dependent)

CORPUS

- An airplane accelerates down a runway at **3.20m/s2** for **32.8 s** until is finally lifts off the ground. Determine the distance traveled before takeoff
- How far will a car travel in **25 min** at **12 kmh**?
- A jalopy with an initial speed of 23.7 km/h accelerates at a uniform rate of **0.92 m/s2**ăfor **3.6 s**. Find the final speed and the displacement of the jalopy during this time
- A college student wants to toss a textbook to his roommate who is leaning out of a window directly above him. He throws the book upwards with an initial velocity of **8.0 m/s**. The roommate catches it while it is traveling at **3.0 m/s** [up]. a) How long was the book in the air? b) How far vertically did the book travel? A car accelerates in a straight line from rest at the rate of **2.3 m/s 2** . What is its final velocity after **55 m**? What is its time?
- 4. What height will a dart achieve **7 seconds** after being blown straight up at **50 m/s**?

PREPROCESSING SAMPLE - ONE RULE PER STEP

Each step is a function from string to string. One regex, one job.

```
import re
from functools import reduce

def sub(pattern, repl):
    return lambda text: re.sub(pattern, repl, text)

remove_newline = sub(r"[\r\n]+", " ")
remove_hints = sub(r"\((?:hint|Hint)[^)]*\)", "")
remove_math_minus = sub(r"[\u2212\u2013]", "-")
separate_number_alpha = sub(r"(?<=\d)(?=[A-Za-z])", " ")
fix_ms_pattern = sub(r"\bm\s*/\s*s\s*2\b", "m/s**2")
ten_power_to_e = sub(r"(\d+) ?[x*] ?10\^?(-?\d+)", r"\1e\2")
mark_directions = sub(r"\[(up|down|left|right)\]", r"<\1>")
mark_degree = sub(r"(\d+)\s*\u00b0", r"\1 <degree>")
single_space = sub(r"\s+", " ")
```

PREPROCESSING SAMPLE - THE PIPELINE

The document flows through the steps in order.

```
UNITS = {"m/s**2": "<m/s**2>", "km/h": "<km/h>",
         "m/s": "<m/s>", "min": "<minute>", "s": "<s>"}
keys = sorted(UNITS, key=len, reverse=True)
alts = "|".join(map(re.escape, keys))
unit_re = r"(?<=\d )(" + alts + r")\b"
mark_units = sub(unit_re, lambda m: UNITS[m.group(1)])

PIPELINE = [remove_newline, remove_hints, remove_math_minus,
            separate_number_alpha, ten_power_to_e,
            fix_ms_pattern, mark_directions, mark_units,
            mark_degree, single_space, str.strip, str.lower]

def preprocess(doc):
    return reduce(lambda t, step: step(t), PIPELINE, doc)
```

A unit is marked only after a number. Case folding is last, so drop it when case matters.

PROCESSED CORPUS

Raw text	Preprocessed Text
An airplane accelerates down a runway at **3.20m/s2** for **32.8 s** until is finally lifts off the ground. Determine the distance traveled before takeoff	An airplane accelerates down a runway at **3.20** $< m/s**2 >$ for **32.8** $< s >$ until is finally lifts off the ground. Determine the distance traveled before takeoff
A college student wants to toss a textbook to his roommate who is leaning out of a window directly above him. He throws the book upwards with an initial velocity of **8.0 m/s**. The roommate catches it while it is traveling at **3.0 m/s** [up]. a) How long was the book in the air? b) How far vertically did the book travel?	A college student wants to toss a textbook to his roommate who is leaning out of a window directly above him. He throws the book upwards with an initial velocity of **8.0** $< m/s >$. The roommate catches it while it is traveling at **3.0** $< m/s >$ [up]. a) How long was the book in the air? b) How far vertically did the book travel?

HTML PREPROCESSING

1. Case folding - convert content to lower case
2. Remove scripts
3. Tokenize
4. Term Frequency
5. Extract Vocabulary
6. Remove stop words
7. Stemming removes suffixes (e.g., the Porter stemmer)
8. Lemmatization maps a word to its dictionary root (e.g., Stanford CoreNLP, spaCy)
   - Stemming: running $\rightarrow$ run
   - Stemming: studies, studied, studying $\rightarrow$ studi
   - Lemmatization: best $\rightarrow$ good
9. Correct misspellings
10. Strip punctuation
11. Normalize white space, newlines, tabs, ...
12. Expand contractions, so isn't $\rightarrow$ is not and I'd $\rightarrow$ I would

PREPROCESSING EXAMPLE - HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="description" content="A sample HTML page for NLP preprocessing.">
    <meta name="keywords" content="HTML, NLP, preprocessing">
    <title>Sample NLP Page</title>
    <style>
        .highlight {
            color: red;
        }
    </style>
</head>
<body>
    <header>
        <h1>Welcome to the Sample Page</h1>
    </header>
    <div id="content">
        <p>This is some sample text for NLP processing. It includes <span class="highlight">important</span> keywords.</p>
        <p>Another paragraph with more text. We want to extract this text.</p>
        <script>
            function sayHello() {
                console.log("Hello from JavaScript!");
            }
        </script>
    </div>
    <footer>
        <p>&copy; 2023 Sample Site</p>
    </footer>
</body>
</html>
```

Processed text

```
Welcome to the Sample Page This is some sample text for NLP processing.
It includes important keywords. Another paragraph with more text.
We want to extract this text. 2023 Sample Site
```

PREPROCESSING HTML

```
import re
from bs4 import BeautifulSoup
def preprocess_html(html_file):
    with open(html_file, 'r', encoding='utf-8') as f:
        html_content = f.read()
    soup = BeautifulSoup(html_content, 'html.parser')
    # Remove script and style elements
    for script in soup(["script", "style", "meta"]):
        script.decompose()
    # Get extracted text
    text = soup.get_text()
    # Remove extra whitespace and newlines
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```