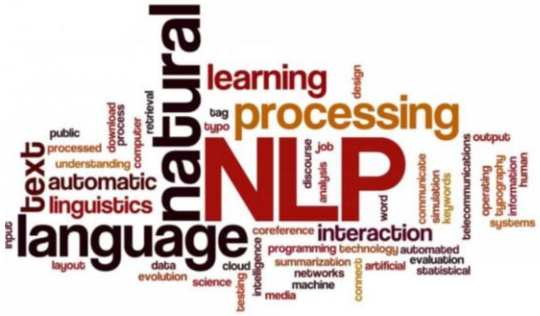
University of
Northampton



Week 3

Preprocessing and Text Analytics

CSY3055 – Natural Language Processing

Dr Oluseyi Oyedeji

Learning Goals for Week 3

By the end of week three, you should be able to:

- Describe and apply key preprocessing techniques.
- Implement Bag-of-Words and TF-IDF models.
- Explain the difference between frequency-based and weighted features.
- Reflect on context-sensitive preprocessing decisions.
- Appreciate how these “basic” models still power real-world applications.

Today's Session

- Text Preprocessing
- Part-of-Speech (POS) Tagging
- Feature Extraction
- Bag-of-Words (BOW)
- TF-IDF
- Exploratory Text Analytics

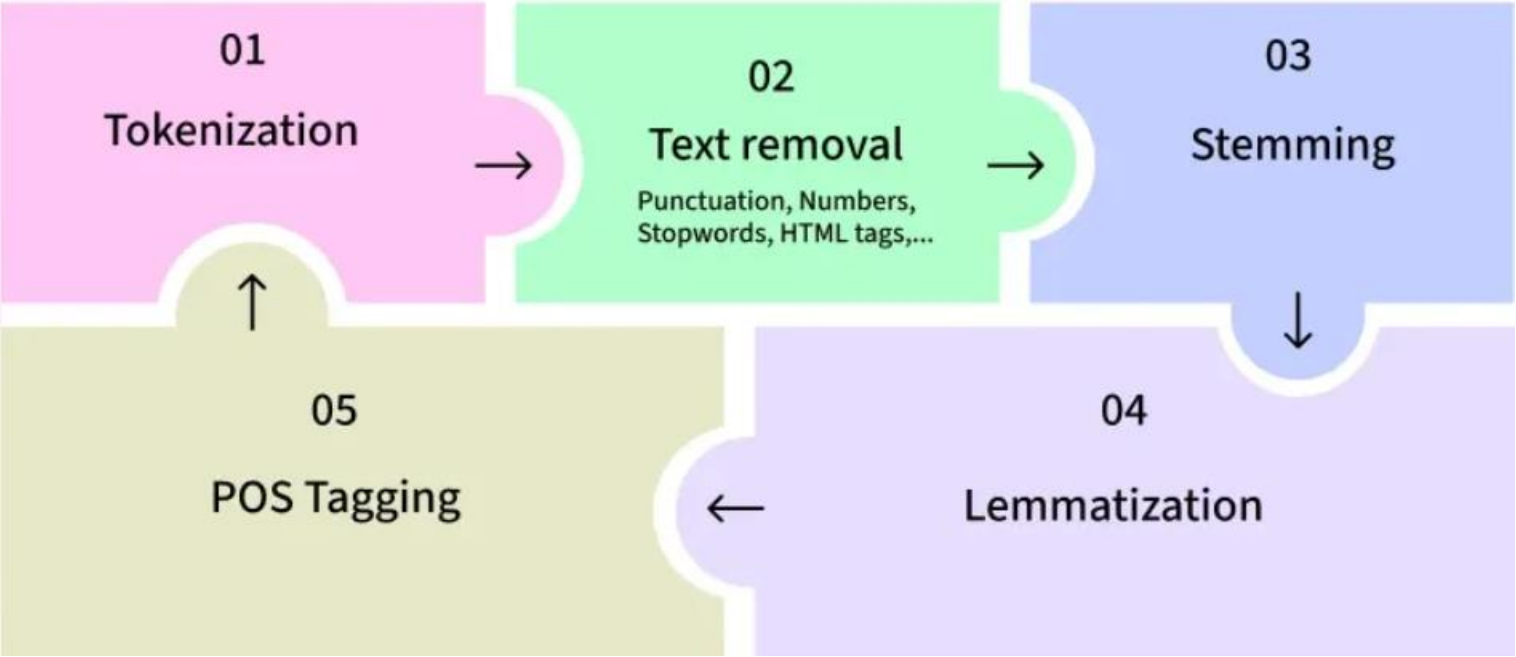
Introduction

- Text is mostly **unstructured** in its raw form. Machines cannot "read" these; they need **proper structure** e.g. **numbers.**
- It is necessary to explore how to:
    - **Preprocess** text (clean and normalize it)
    - **Extract** features (convert it into vectors)
    - **Analyze** those features for insights
- This is the foundation of all NLP models
    - From spam filters to chatbots to language models.

Text Preprocessing

Text Preprocessing

- **Preprocessing** ensures that text is *consistent, structured, and informative* before numerical transformation.
- It means tidying your data before you let a machine learn from it.



University of Northampton

Core Preprocessing Steps

Step	Purpose	Example
Lowercasing	Removes case variations	“Free” → “free”
Removing punctuation & symbols	Clears non-textual noise	“WIN!!!” → “win”
Removing numbers	Optional (depends on task)	“Win 1000 now” → “Win now”
Tokenization	Splits text into words	“I love NLP” → [“I”, “love”, “NLP”]
Stopword removal	Removes uninformative words	“the”, “and”, “to”
Lemmatization/Stemming	Reduces to base form	“running” → “run”

UO N University of Northampton

Preprocessing extracts the core meaning that is crucial for analysis.

Other Preprocessing Actions

- Removing **URLS**
- Removing remove **non-word** and **non-whitespace characters**
- Handling Contractions (don’t, I’ll, we’ll etc.)
- Handling Emojis and Emoticons
- Spell Checking
- Remove HTML tags
- Handling Chat words (u, ur, brb, lol etc.)

Preprocessing task [Code]

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

text = "Congratulations! You have won 1000 dollars. Claim now."
tokens = word_tokenize(text.lower())
words = [w for w in tokens if w.isalpha()]
filtered = [w for w in words if w not in stopwords.words('english')]
lemmatizer = WordNetLemmatizer()
cleaned = [lemmatizer.lemmatize(w) for w in filtered]
print(cleaned)
```

Context-Specific Preprocessing

Domain	Adjustments
Social Media	Retain emojis, hashtags; handle @mentions
Legal/Medical	Keep numbers and acronyms
Chatbots	Keep pronouns and contractions
Multilingual corpora	Detect language before cleaning
Deep learning pipelines	Minimal preprocessing
Traditional ML (BoW/TF-IDF)	Heavier cleaning, full normalization

Principle: Strike a balance between noise removal and information preservation

Part-of-Speech Tagging

Part-of-Speech Tagging

- **Part-of-Speech (POS)** Tagging may not be completely considered a preprocessing technique
  - It usually exists between the Preprocessing and Feature Extraction Stage in the NLP pipeline
- **Part-of-Speech (POS)** tagging is the process of **assigning** a **grammatical category** (e.g., **noun**, **verb**, **adjective**) to each word in a sentence.

Part-of-Speech Tagging – Example 1

- Sentence
  - "*The cat sat on the mat*."
- Tagged
  - [('The', '**DET**'), ('cat', '**NOUN**'), ('sat', '**VERB**'), ('on', '**ADP**'), ('the', '**DET**'), ('mat', '**NOUN**')]

Part-of-Speech Tagging – Example 2

- Sentence
  - "*Artificial intelligence is transforming industries*."
- Tagged
  - [('Artificial', '**ADJ**'), ('intelligence', '**NOUN**'), ('is', '**VERB**'), ('transforming', '**VERB**'), ('industries', '**NOUN**')]

POS Tagging Matters

Application	Role of POS Tagging
Information extraction	Identify entities and relationships
Sentiment analysis	Adjectives/adverbs carry emotion
Text classification	Improves features for ML models
Translation	Maintains grammatical order
Chatbots	Understands question type

Universal POS Tagsets

Tag	Meaning	Example
NOUN	Noun	student, book
VERB	Verb	study, learn
ADJ	Adjective	smart, busy
ADV	Adverb	quickly, very
PRON	Pronoun	he, they
DET	Determiner	the, a
ADP	Adposition	in, on, at
CONJ	Conjunction	and, but
NUM	Numeral	one, ten
INTJ	Interjection	wow, oh

used in NLTK/spaCy

POS Tagging Methods

- Rule-Based
  - Uses grammar rules and dictionaries.
  - E.g. words ending in *-ly* = adverbs.
- Statistical
  - Uses probabilities from labelled data (e.g., Hidden Markov Models).
  - Predicts most likely tag sequence.
- Neural
  - Deep learning (LSTM, Transformer).
  - Learns context automatically
  - **Has best accuracy**.

POS Tagging with NLTK

```python
import nltk
nltk.download('punkt'); nltk.download('averaged_perceptron_tagger')

text = "The quick brown fox jumps over the lazy dog"
tokens = nltk.word_tokenize(text)
tags = nltk.pos_tag(tokens)
print(tags)
```

```
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'J
J'), ('dog', 'NN')]
```

NLTK uses the Penn Treebank tagset (e.g., JJ = adjective, NN = noun)

POS Tagging with spaCy

```python
import spacy
nlp = spacy.load("en_core_web_sm")

doc = nlp("The quick brown fox jumps over the lazy dog")
for token in doc:
    print(token.text, token.pos_, token.tag_, token.dep_)
```

```
The DET DT det
quick ADJ JJ amod
brown ADJ JJ amod
fox NOUN NN nsubj
jumps VERB VBZ ROOT
over ADP IN prep
the DET DT det
lazy ADJ JJ amod
dog NOUN NN pobj
```

spaCy provides **POS**, **fine-grained tags**, and **dependency relations**.

POS Tagging for Feature Engineering

- Keep **nouns**
  - topics and entities
- Keep **adjectives/adverbs**
  - sentiment-bearing words
- Remove **determiners**, **prepositions**, etc.
- Use filtered text for TF-IDF or embeddings

Using POS Tags to Filter Words

```
from nltk.corpus import stopwords
tokens = nltk.word_tokenize("This excellent camera captures amazing photos in low light.")
tags = nltk.pos_tag(tokens)

filtered = [word for word, tag in tags if tag.startswith('JJ') or tag.startswith('NN')]
print(filtered)
```

```
['excellent', 'camera', 'amazing', 'photos', 'low', 'light']
```

*Some POS tagging challenges includes **Ambiguity**, **informal texts**, **Domain variation** and **language differences***

Feature Extraction

Bag-of-Words (BoW)

- Represents text as a “bag” of words, ignoring grammar and word order.
- Each document becomes a vector of word counts.

Document	win	money	claim	now
“Win money now”	1	1	0	1
“Claim your prize now”	0	0	1	1

N-grams

- N-grams are sequences of N consecutive words.
    - Unigrams (n=1): “win”, “money”, “now”
    - Bigrams (n=2): “win money”, “money now”
    - Trigrams (n=3): “claim your prize”

N-grams define what goes into the Bag-of-Words model, more context, richer features.

N-grams [Code]

```
from sklearn.feature_extraction.text import CountVectorizer

docs = ["Win money now", "Claim your free prize now"]
vectorizer = CountVectorizer(ngram_range=(1,2))
X = vectorizer.fit_transform(docs)
print(vectorizer.get_feature_names_out())
print(X.toarray())
```

Each row = document; each column = token; values = word counts.

TF-IDF (Term Frequency–Inverse Document Frequency)

TF-IDF

- **Statistical method** used in NLP and information retrieval to evaluate how **important a word** is to a document in relation to a larger collection of documents.
- The problem In BoW is that **common words dominate**.
- TF-IDF solves this by adjusting word weights by rarity.

TF-IDF contd.

$$TF-IDF(t,d) = TF(t,d) \times \log\left(\frac{N}{DF(t)}\right)$$

where

TF(t,d) = term frequency of *t* in *d*

DF(t) = number of documents containing *t*

N = total documents

log(N / DF(t)) = inverse document frequency

TF-IDF contd.

- Interpretation
  - **High TF** = frequent in a document
  - **Low DF** = rare across documents
  - **Result** = higher importance

TF-IDF [CODE]

```
from sklearn.feature_extraction.text import TfidfVectorizer

docs = ["Win money now", "Claim your free prize now"]
vectorizer = TfidfVectorizer(ngram_range=(1,2))
X = vectorizer.fit_transform(docs)
print(vectorizer.get_feature_names_out())
print(X.toarray())
```

BoW vs TF-IDF

Aspect	Bag-of-Words	TF-IDF
Representation	Raw counts	Weighted counts
Common words	Dominant	Down-weighted
Unique words	Equal treatment	Up-weighted
Vector type	Integer	Float
Emphasis	Frequency	Informativeness
Example Use	Quick baseline	Discriminative models

BoW counts every voice; TF-IDF amplifies unique ones.

Exploratory Text Analytics

Understand what your text “says” before modeling.

Exploratory Text Analytics

- *Recap from Week 1*
- **Frequency Distribution**
  - Rank most common terms to see vocabulary richness and noise
- **Word Cloud**
  - Visual summary of word frequencies.
  - Class Comparison

Industry Applications

Application	BoW/TF-IDF Role
Email Spam Detection	Classic Naïve Bayes + TF-IDF
Search Engines (Lucene/BM25)	TF-IDF-based ranking
Customer Feedback Analysis	Sentiment scoring
Fake News Detection	Frequency + stylistic features
Recommendation Systems	Similarity using TF-IDF vectors

Transition to Modern NLP

- **Modern models** (like BERT, GPT, and word2vec) learn their own representations, replacing manual feature engineering.
- But understanding BoW and TF-IDF is essential
  - They form the conceptual DNA of how machines process text.
- Before machines could understand meaning, they had to learn to count.

Ethics, Fairness & Transparency

- **Dataset Imbalance**
  - Unequal spam/ham distribution causes bias , fix via balancing or weighting.
- **Preprocessing Bias**
  - Removing rare dialect or slang terms can erase identity markers.
- **Transparency**
  - Always document (record) preprocessing, it’s part of ethical AI practice.

Bias can start as early as data cleaning.

Summary

- Preprocessing = cleaning text.
- Feature Extraction = turning text into numbers.
- BoW counts; TF-IDF weighs importance.
- N-grams capture local order.
- Fairness and ethics matter early.
- These foundations still power real-world NLP.

Thank you