THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 36: Natural Language Processing (NLP)

Sergei V. Kalinin

Language processing is everywhere!

- **Natural language (and speech) interfaces:** search/IR, database access, image search, image description
- **Dialog systems** (e.g. customer service, robots, cars, tutoring), chatbots
- **Information extraction, summarization, translation:** Process (large amounts of) text automatically to obtain meaning/knowledge contained in the text
- **Analysis:** Identify/analyze trends, opinions, etc. (e.g. in social media)
- **Translation:** Translate text automatically from one language to another
- **Convenience:** Grammar/style checking, automate email filing, autograding

Adapted from J. Hockenmaier, UIUC CS447

Language processing is everywhere!

- **Natural language understanding:** Extract information (e.g. about entities, events or relations between them) from text
- **Translate raw text into a meaning representation:** data mining, causal structures, knowledge graphs
- **Execute NL instructions:** ChatGPT for coding
- **Summarize** (newspaper or scientific) articles, describe images

Adapted from J. Hockenmaier, UIUC CS447

Why language analysis is difficult?

- **Ambiguity:** Words and phrases often have multiple meanings (e.g., "bank" can mean a financial institution or a riverbank).
- **Context Dependency:** The meaning of words and sentences often depends on context, making it difficult to infer without understanding surrounding text.
- **Idioms and Expressions:** Non-literal expressions like "kick the bucket" or "hit the road" do not translate directly, making them challenging to interpret.
- **Grammar and Syntax Variability:** Natural language is full of syntactical complexities, including irregular structures and sentence forms that challenge rule-based and ML-based approaches.

Adapted from J. Hockenmaier, UIUC CS447

Why language analysis is difficult?

- **Morphology:** Words can change form based on tense, plurality, etc., which complicates accurate tokenization and representation.
- **Long-Range Dependencies:** Important information in a sentence or paragraph may be far apart, making it hard to connect related concepts.
- **Noise and Informality in Text:** Real-world text (e.g., tweets, chat messages) often includes typos, slang, and abbreviations, complicating processing.
- **High Dimensionality:** Language data is represented by large vocabularies and embeddings, leading to high-dimensional data that can be computationally demanding.
- **Evolving Vocabulary:** Language changes over time, with new words and phrases continually emerging, requiring models to adapt.

When did it start?

A real human-computer dialogue from the 1960s: ELIZA (Weizenbaum, 1966)

Computer: TELL ME MORE ABOUT YOUR FAMILY

Human: My mother takes care of me.

Computer: WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU

Human: My father.

Computer: YOUR FATHER

Human: You are like my father in some ways.

Computer: WHAT RESEMBLANCE DO YOU SEE

Human: You are not very aggressive but I think you don't want me to notice that.

Computer: WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE

Human: You don't argue with me.

Computer: WHY DO YOU THINK I DON'T ARGUE WITH YOU

Human: You are afraid of me.

… and what it was like

The Eliza Effect

People *really* trusted this system:

Weizenbaum: *"I had not realized … that extremely short exposures to a relatively simple computer program could induce powerful delusional thinking in quite normal people"*

ELIZA was based on a cascade of finite-state transducers:

Human: *You don't argue with me.*

1. Replace **you** with ***I*** and ***me*** with ***you:***
   I don't argue with you.
2. Replace ***<...>*** with ***Why do you think <...>:***
   Why do you think I don't argue with you.

Computer: *WHY DO YOU THINK I DON'T ARGUE WITH YOU*

— How could you break a system like ELIZA?
— Where do you see the ELIZA effect today?

General NLP workflow

- **Tokenizer/Segmenter** to identify words and sentences
- **Morphological analyzer/POS-tagger** to identify the part of speech and structure of words
- **Word sense disambiguation** to identify the meaning of words
- **Syntactic/semantic Parser** to obtain the structure and meaning of sentences
- **Coreference resolution** to keep track of the various entities mentioned

Adapted from J. Hockenmaier, UIUC CS447

General NLP workflow

Each step in the NLP pipeline embellishes the input with explicit information about its linguistic structure:

- POS tagging: Parts of speech of word,
- Syntactic parsing: Grammatical structure of sentence,….

Each step in the NLP pipeline requires its own explicit (“symbolic”) output representation:

- POS tagging requires a POS tag set (e.g. NN=common noun singular, NNS = common noun plural, …)
- Syntactic parsing requires constituent or dependency labels (e.g. NP = noun phrase, or nsubj = nominal subject)

These representations should capture linguistically appropriate generalizations/abstractions. Designing these representations requires linguistic expertise

Adapted from J. Hockenmaier, UIUC CS447

Ambiguity in NLP

I made her duck

What does this sentence mean?

“I made her crouch”,
“I cooked duck for her”,
“I cooked her [pet] duck (perhaps just for myself)”,
“I have promoted her duck to be a mafia boss”

… “duck”: noun or verb?
“make”: “cook X” or “cause X to do Y” or mafia slang?
“her”: “for her” or “belonging to her” ?

Adapted from J. Hockenmaier, UIUC CS447

Ambiguity in NLP

Structural ambiguity:
"I eat sushi with tuna" vs. "I eat sushi with chopsticks"
"I saw the man with the telescope on the hill"

Lexical (word sense) ambiguity:
"I went to the bank": financial institution or river bank?

Referential ambiguity:
"John saw Jim. He was drinking coffee."
Who was drinking coffee?

Adapted from J. Hockenmaier, UIUC CS447

Data in NLP

A corpus (plural: corpora) is a collection of natural language data (i.e. a data set)

Raw corpora have only minimal (or no) processing:

- Sentence boundaries may or may not be identified.
- There may or may not be metadata
- Typos (written text) or disfluencies (spoken language) may or may not be corrected.

Annotated corpora contain some labels (e.g. POS tags, sentiment labels), or linguistic structures (e.g syntax trees, semantic interpretations), etc.

Adapted from J. Hockenmaier, UIUC CS447

How difficult is parsing?

For a language like English, this seems like a really easy problem: A word is any sequence of alphabetical characters between whitespaces that's not a punctuation mark? That works to a first approximation, but…

… what about abbreviations like D.C.?
… what about complex names like New York?
… what about contractions like doesn't or couldn't've?
… what about New York-based ?
… what about names like SARS-Cov-2, or R2-D2?
… what about languages like Chinese that have no whitespace, or languages like Turkish where one such "word" may express as much information as an entire English sentence?

Adapted from J. Hockenmaier, UIUC CS447

How large is vocabulary?

Vocabulary size = the number of distinct word types

Google N-gram corpus: 1 trillion tokens, 13 million word types that appear 40+ times

- If you count words in text, you will find that… …a few words (mostly closed-class) are very frequent
- (the, be, to, of, and, a, in, that,…) Most words (all open class) are very rare. … Even if you've read a lot of text, you will keep finding words you haven't seen before.

Word frequency: the number of occurrences of a word type in a text (or in a collection of texts

The number of distinct word types (vocabulary size) increases with the size of the corpus Herdan's Law/Heap's Law:

$$|V| = kN^{\beta} \qquad k \text{ and } 0 < \beta$$

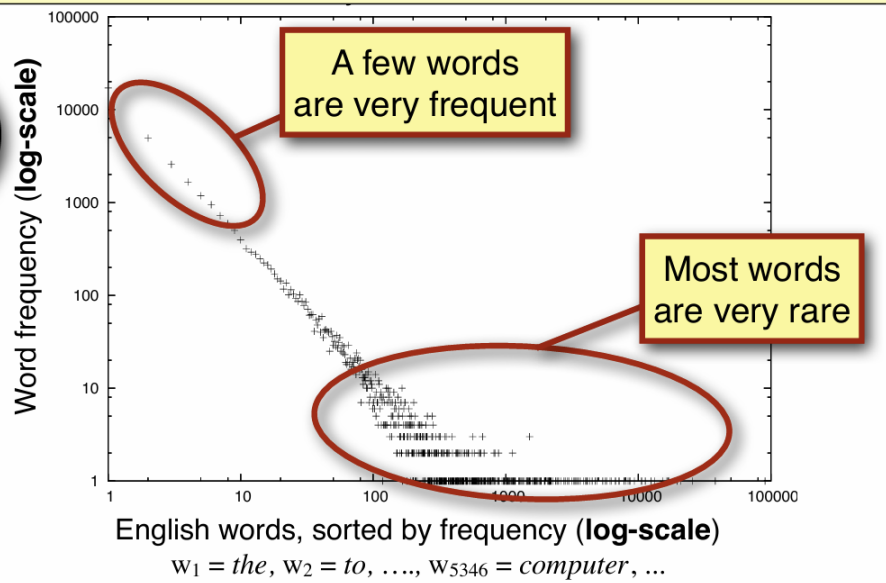
Adapted from J. Hockenmaier, UIUC CS447

Zipf's law

Zipf's law: the long tail

How many words occur once, twice, 100 times, 1000 times?

the *r*-th most common word w_r has $P(w_r) \propto 1/r$



In natural language:
A small number of events (e.g. words) occur with high frequency
A large number of events occur with very low frequency

- **The good:** Any text will contain a number of words that are very common. We have seen these words often enough that we know (almost) everything about them. These words will help us get at the structure (and possibly meaning) of this text.
- **The bad:** Any text will contain a number of words that are rare. We know something about these words, but haven't seen them often enough to know everything about them. They may occur with a meaning or a part of speech we haven't seen before.
- **The ugly:** Any text will contain a number of words that are unknown to us. We have never seen them before, but we still need to get at the structure (and meaning) of these texts.

Adapted from J. Hockenmaier, UIUC CS447

How do we identify “elementary” words?

Option 1: Words are atomic symbols

- Each (surface) word form is its own symbol
- Add some generalization by mapping different forms of a word to the same symbol

- **Normalization:** map all variants of the same word (form) to the same canonical variant (e.g. lowercase everything, normalize spellings, perhaps spell-check)
- **Lemmatization:** map each word to its lemma (esp. in English, the lemma is still a word in the language, but lemmatized text is no longer grammatical)
- **Stemming:** remove endings that differ among word forms (no guarantee that the resulting symbol is an actual word)

Adapted from J. Hockenmaier, UIUC CS447

How do we identify “elementary” words?

Option 2: Represent the structure of each word “books” => “book N pl” (or “book V 3rd sg”) This requires a morphological analyzer. The output is often a lemma (“book”) plus morphological information (“N pl” i.e. plural noun)

Turkish: uygarlaştıramadıklarımızdanmışsınızcasına

uygar_laş_tır_ama_dık_lar_ımız_dan_mış_sınız_casına

“as if you are among those whom we were not able to civilize (=cause to become civilized)”

uygar: civilized
_laş: become
_tır: cause somebody to do something
_ama: not able
_dık: past participle
_lar: plural
_ımız: 1st person plural possessive (our)
_dan: among (ablative case)
_mış: past
_sınız: 2nd person plural (you)
_casına: as if (forms an adverb from a verb)

Adapted from J. Hockenmaier, UIUC CS447

Let's start with basics: Regex

• **Definition**: Regular expressions (regex) are sequences of characters that form search patterns.
• **Purpose**: Used for finding, matching, and manipulating text in strings or files.
• **Applications**: Common in data processing, programming, and text editing.

Simple patterns:

- Standard characters match themselves: 'a', '1'
- Character classes: '[abc]', '[0-9]', negation: '[^aeiou]' (Predefined: \s (whitespace), \w (alphanumeric), etc.)
- Any character (except newline) is matched by '.'

Complex patterns: (e.g. ^[A-Z]([a-z])+\s)

- Group: '(...)'
- Repetition: 0 or more times: '*', 1 or more times: '+'
- Disjunction: '...|...'
- Beginning of line '^' and end of line '$

Adapted from J. Hockenmaier, UIUC CS447

Using Regex

- **Pattern Matching:** Identify specific patterns in text (e.g., finding all email addresses in a document).
- **Data Validation:** Ensure inputs meet a certain format (e.g., valid phone numbers, email addresses, or ZIP codes).
- **Text Searching:** Locate keywords or phrases within large documents or datasets.
- **Text Extraction:** Pull out important data, such as dates, currency amounts, or IDs, from unstructured text.
- **Replacing Text:** Substitute parts of text based on patterns (e.g., censoring profanity or formatting dates).
- **String Splitting:** Split strings into components by specific patterns (e.g., splitting a sentence into words).
- **Removing Unwanted Characters:** Clean data by removing extra whitespace, punctuation, or formatting artifacts.

Adapted from J. Hockenmaier, UIUC CS447

Text statistics: Bag of Words

- The Bag of Words model is a simple and commonly used method for representing text data in natural language processing (NLP).
- Treats each document as a "bag" of words, ignoring grammar, word order, and structure.
- Represents each document by counting the occurrences (frequency) of each word in the text.

Process:

1. **Tokenization:** Split each document into individual words (tokens).
2. **Vocabulary Creation:** Create a unique list (vocabulary) of all words across the dataset.
3. **Vectorization:** For each document, create a vector where each position corresponds to a word in the vocabulary, and the value at each position is the count (frequency) of that word in the document.

Adapted from J. Hockenmaier, UIUC CS447

Text statistics: Bag of Words

1. Documents: ["The cat sat on the mat", "The dog sat on the log"]
2. Vocabulary: ["The", "cat", "sat", "on", "mat", "dog", "log"]
3. Vector Representation:
    - Document 1: [2, 1, 1, 1, 1, 0, 0]
    - Document 2: [2, 0, 1, 1, 0, 1, 1]

Pros:

- Simple and easy to implement.
- Effective for basic text representation in classification and clustering tasks.

Cons:

- Ignores word order, grammar, and semantics.
- Produces large, sparse vectors, especially for big vocabularies.
- Common words can overshadow the meaningful words

Adapted from J. Hockenmaier, UIUC CS447

Term Frequency – Inverse Document Frequency

TF-IDF is a numerical statistic used to reflect the importance of a word in a document relative to a collection of documents (corpus).

Purpose: Enhances the Bag of Words model by giving less weight to common words and more weight to unique or informative words, improving relevance in text analysis.

Components:

- Term Frequency (TF): Measures how often a word appears in a document.
  Document Frequency (IDF): Measures how unique a word is across all documents (Total number of documents/Number of documents containing te word)
- TF-IDF Score: Calculated by multiplying TF and IDF, giving a higher score to unique, informative words.

Adapted from J. Hockenmaier, UIUC CS447

Term Frequency – Inverse Document Frequency

Process:

1. Tokenization: Split each document into words.
2. Calculate TF for each word in each document.
3. Calculate IDF for each word across the entire corpus.
4. Compute TF-IDF score for each word in each document.

Example:

- Documents: ["The cat sat on the mat", "The dog sat on the log"]
- Word "sat" appears once in each document, so it has a lower IDF (more common across documents), reducing its TF-IDF score.
- Rare words like "cat" or "dog" have a higher IDF, boosting their TF-IDF scores.

Pros:

- Highlights informative words by down-weighting common words.
- Improves relevance in tasks like document classification and information retrieval.

Cons:

- Does not capture word order or semantics.
- Creates large, sparse vectors with big vocabularies.
- Provides a more nuanced representation than Bag of Words by emphasizing distinctive terms across documents, which is useful for tasks requiring word importance differentiation.

Term Frequency – Inverse Document Frequency

Creation:

- Transform each document into a vector of TF-IDF scores, where each element represents a word in the vocabulary.
- Compile these vectors into a TF-IDF matrix, with documents as rows and words as columns, making the text data ready for machine learning.

Input for Machine Learning Models:

- Use TF-IDF features as inputs for algorithms such as Naive Bayes, Support Vector Machines, and Logistic Regression in text classification tasks.
- Train models to classify documents, detect sentiment, or categorize topics based on the TF-IDF vector representation.

- Similarity Measurement: Use TF-IDF vectors to calculate cosine similarity between documents to find related content or cluster similar documents.
- Dimensionality Reduction: Reduce the TF-IDF matrix's dimensionality with techniques like PCA or LSA for efficient processing and better model performance.
- Information Retrieval: Rank documents based on their TF-IDF scores, with higher scores indicating more relevant matches to search queries.

There must be order: N-gram models

N-gram models *assume* each word (event) depends only on the previous n–1 words (events):

Unigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$

Bigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$

Trigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)}, w^{(i-2)})$

NB: Independence assumptions where the n-th event in a sequence depends only on the last n-1 events are called Markov assumptions (of order n–1).

Adapted from J. Hockenmaier, UIUC CS447