COMP 554 / CSDS 553 Advanced NLP

Faizad Ullah

Evaluation Metrics

Evaluations: Intrinsic vs. Extrinsic

- Intrinsic Evaluation: Measures the model's performance on a specific, well-defined task (e.g., translation, sentiment classification).

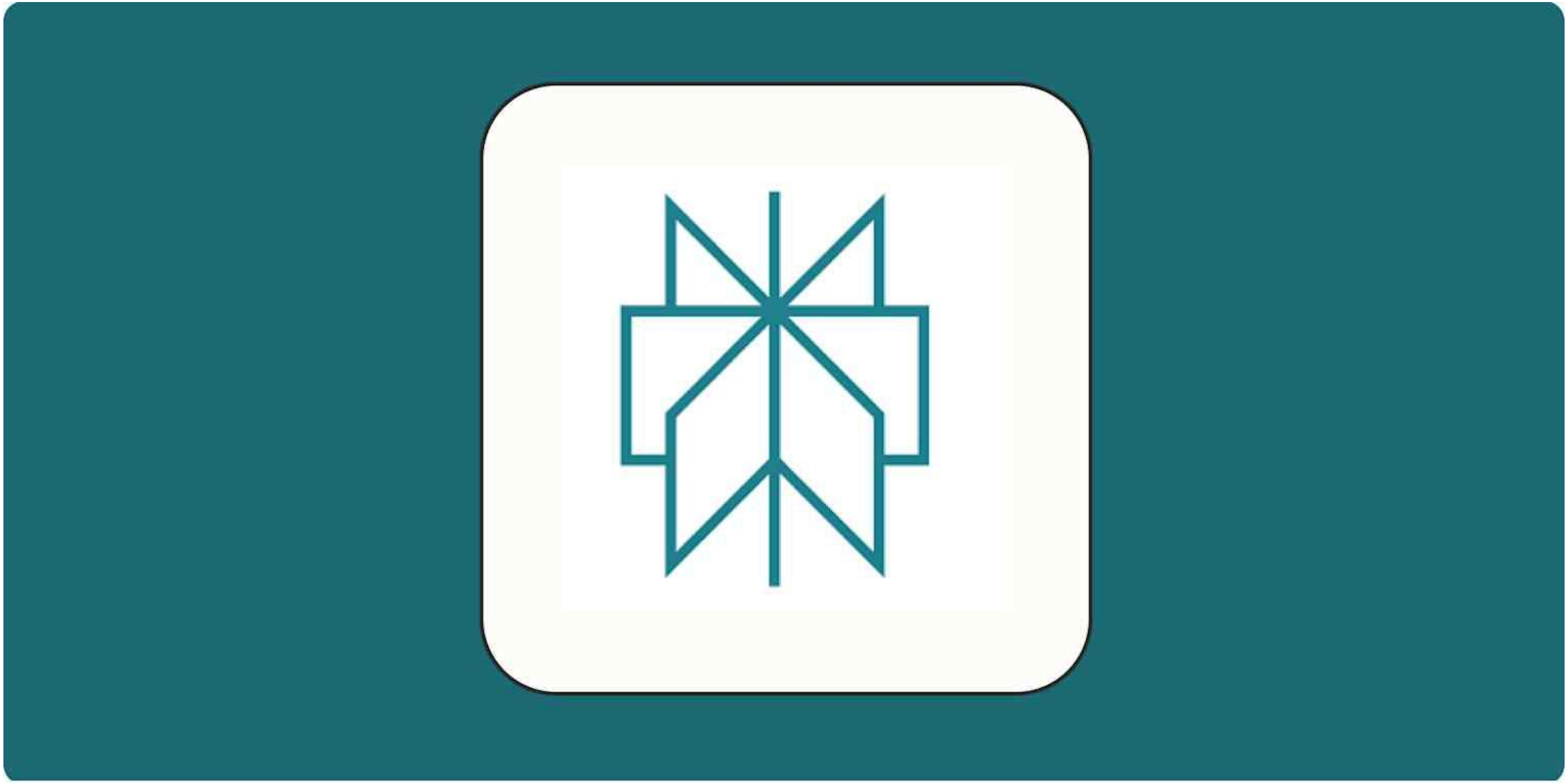
- Examples:
  - Language Modeling: Perplexity (PPL) (lower is better)
  - Text Classification: Accuracy, Precision, Recall, F1-score
  - Machine Translation: BLEU, ROUGE, METEOR
  - Word Embeddings: Cosine Similarity, Word Analogy Tasks

Evaluations: Intrinsic vs. Extrinsic

- Measures the real-world impact of the model in a practical application.
- Evaluates how well the model contributes to an external task.
- More time-consuming but provides application-level insights.
- Examples:
  - Chatbot Performance: How well a chatbot improves customer satisfaction.
  - Search Engine Ranking: How well an NLP model improves search results.
  - Summarization Impact: Does a summarization model help users understand documents faster?

Evaluations: Intrinsic vs. Extrinsic

Feature	Intrinsic Evaluation	Extrinsic Evaluation
Goal	Measures performance on a specific task	Measures impact on a real-world application
Metrics	PPL, BLEU, F1-score, Accuracy	User satisfaction, search ranking, response time
Time Required	**Faster** (less expensive)	**Slower** (more expensive)
Example	Evaluating **perplexity** of a language model	Measuring **how well** the model improves a chatbot's response quality




The more I think
The more confused I get

Perplexity

Perplexity

- Perplexity is a common metric to measure the performance of a language model.
- Smaller value better performance
- A text written by humans are more likely to have lower perplexity score.
- Perplexity is basically the inverse probability of a test set, normalized by the number of words in the test set.

Perplexity: Likelihood

$$P(X) = \prod_{i=0}^{t} p(x_i \mid x_{<i})$$

Likelihood of a Sequence

Perplexity: Cross-Entropy

$$CE(X) = -\frac{1}{t}\log P(X)$$

Cross-Entropy

Perplexity

$$PPL(X) = e^{CE(X)}$$
$$= e^{-\frac{1}{t}\sum_{i=0}^{t}\log p(x_i|x_{<i})}$$

Perplexity

Perplexity

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m} \frac{1}{P(w_i|w_{i-1})}}$$

$$logPP(W) = -\frac{1}{m}\sum_{i=1}^{m} log_2(P(w_i|w_{i-1}))$$

Perplexity: Example

- `Sentence = I love NLP`
- `P(I) = 0.2`
- `P(love|I) = 0.3`
- `P(NLP|love) = 0.4`

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

$$PP(W) = \left(\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, \dots, w_{i-1})} \right)^{\frac{1}{N}}$$

Perplexity: Example

- `Sentence = I love NLP`
- `P(I) = 0.2`
- `P(love|I) = 0.3`
- `P(NLP|love) = 0.4`

$$PP(W) = e^{-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_1 \ldots w_{i-1})}$$

Text Normalization

Tokenization

- Before almost any natural language processing of a text, the text has to be normalized, a task called text normalization.

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

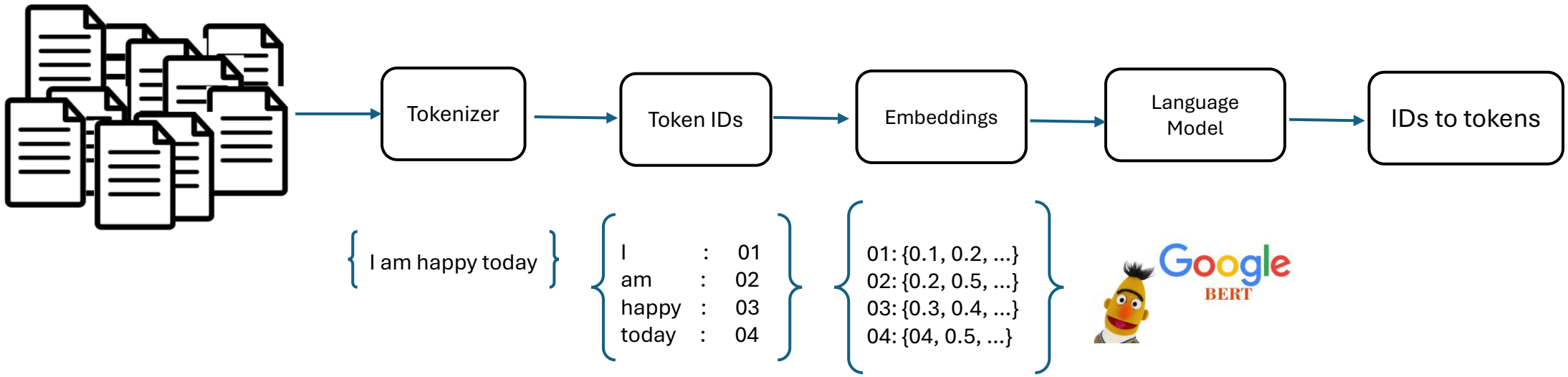
Tokenization

- Tokenization: To extract linguistic unit of interests from running text
- Linguistic Unit?
- character, word, sentence, paragraph, …
- The most common is word
- The 1989 edition of the Oxford English Dictionary had 615,000 entries.

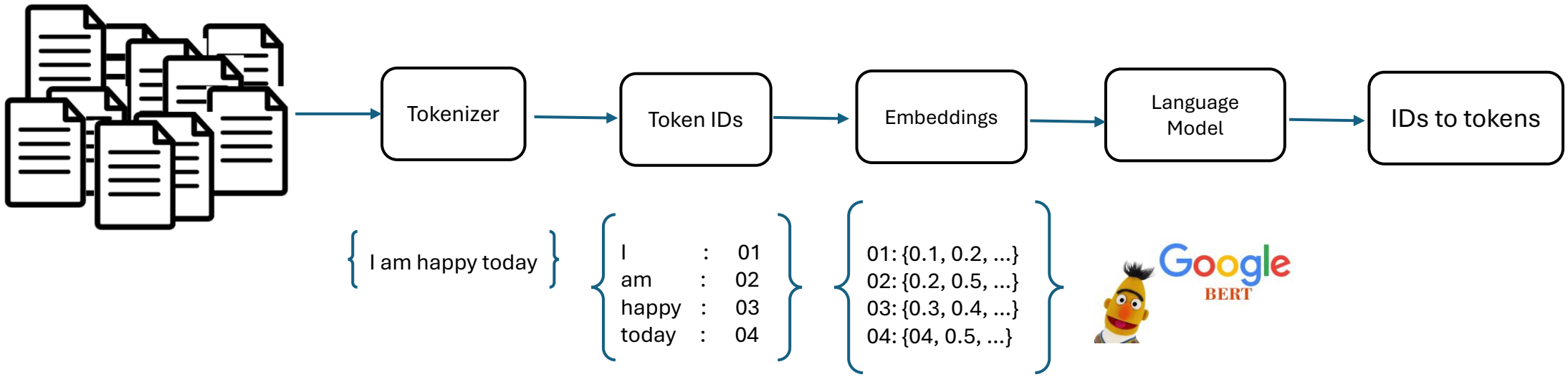
Tokenization

- The most intuitive and simple way is to tokenize using white spaces.
- What if the language doesn’t use white spaces?
- What if some nouns contain spaces? (e.g., New York, Lakki Marwat, Kot Lakhpat, etc.)

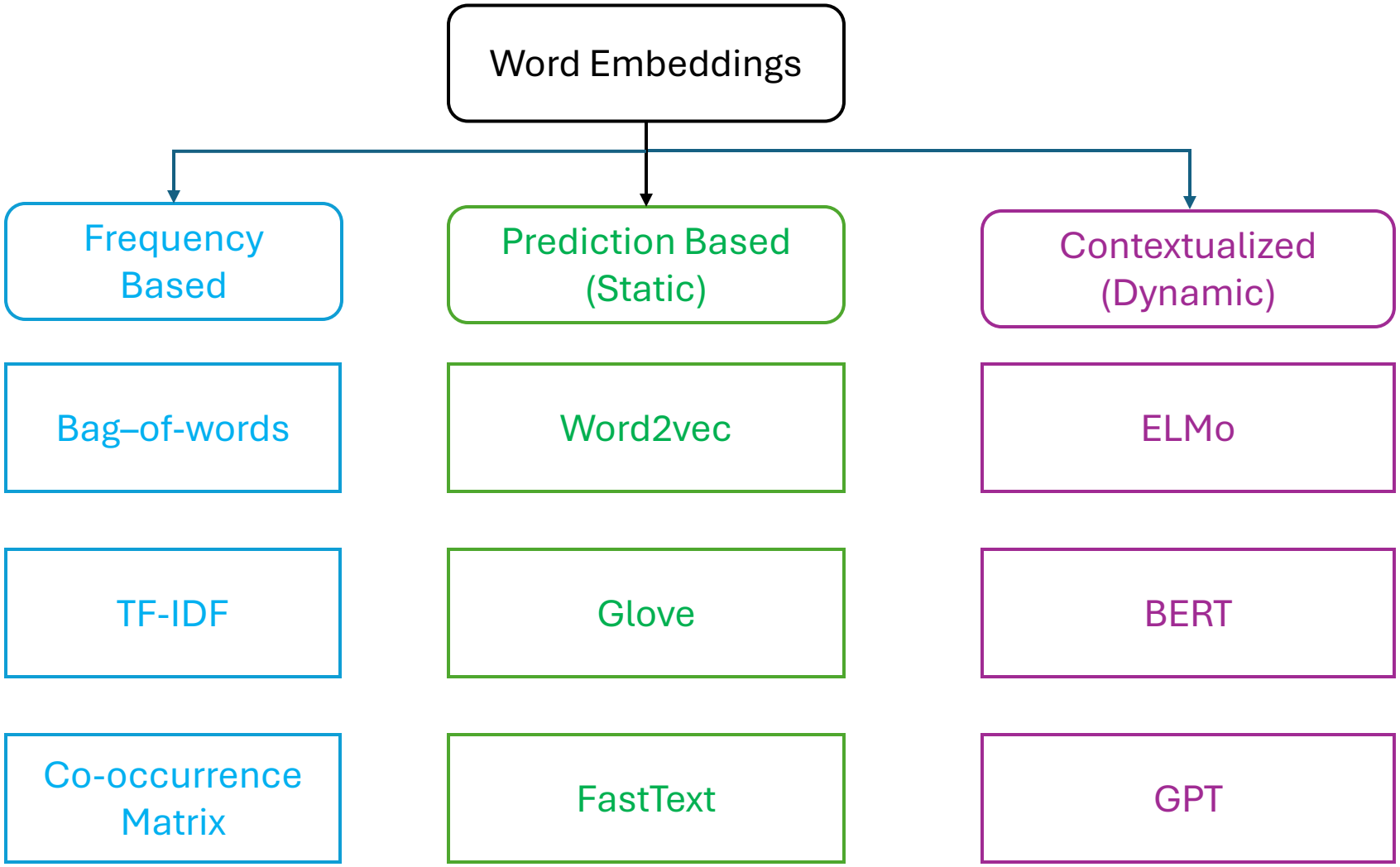
Tokenization



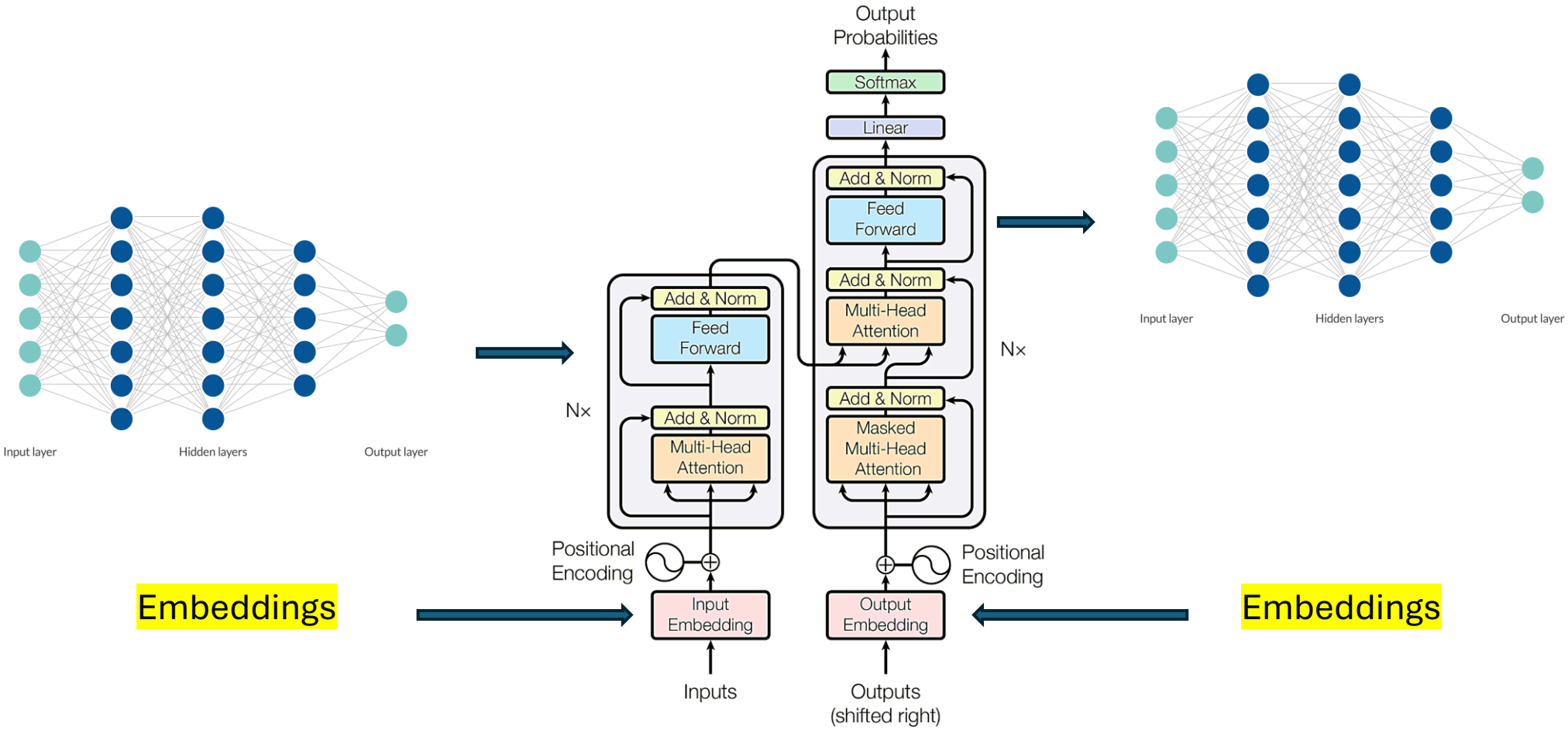
Tokenization



Word Embeddings



Transformers



Words

- Type: An element of the vocabulary or the number of distinct words in a corpus

- If the set of words in the vocabulary is word instance V, the number of types is the vocabulary size |V|.

- Challenges:
  - What should be the size of vocabulary?
  - Out-of-vocabulary (OOV)
  - New words
  - Handling misspelled words in corpus

Words

- Token/Instance: An instance of that type in running text

- Word instances are the total number N of running words.

Words

- How many words/tokens and types are in the following sentence?

- `He stepped out into the hall, was delighted to encounter a water brother.`

- This sentence has 13 words if we don't count punctuation marks as words, 15 if we count punctuation.

- Whether we treat period (“.”), comma (“,”), and so on as words depends on the task.

Words

- `I do uh main- mainly business data processing.`

- This utterance has two kinds of disfluencies.
  - The broken-off word main- is fragment called a fragment.
  - Words like uh and um are called fillers or filled pauses.

- We consider these to be words?
- It depends on the application.

Morphology

- A morpheme is the smallest meaning-bearing unit of a language; for example the word unwashable has the morphemes un-, wash, and -able.

- Some languages, like Japanese, don’t have spaces between words, so word tokenization becomes more difficult.

Word Normalization

- Word normalization is the task of putting words or tokens in a standard format.

- The world has 7097 languages at the time of this writing, according to the online Ethnologue catalog (Simons and Fennig, 2018).

- It is important to test algorithms on more than one language, and particularly on languages with different properties; by contrast there is an unfortunate current tendency for NLP algorithms to be developed or tested just on English (Bender, 2019).

- code switching

How many words?

N = number of tokens/Instances

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Corpus	Types = $\|V\|$	Instances = N
Shakespeare	31 thousand	884 thousand
Brown corpus	38 thousand	1 million
Switchboard telephone conversations	20 thousand	2.4 million
COCA	2 million	440 million
Google n-grams	13 million	1 trillion

Issues in Tokenization

- `Finland’s capital` →
- `what’re, I’m, isn’t` →
- `Hewlett-Packard` →
- `state-of-the-art` →
- `Lowercase` →
- `San Francisco` →
- `m.p.h., Ph.D.` →

Word Tokenization in Chinese

- Also called Word Segmentation
- Chinese words are composed of characters
- Characters are generally 1 syllable and 1 morpheme.
- Standard baseline segmentation algorithm:
- Maximum Matching (also called Greedy)

Maximum Matching Word Segmentation

- Given a wordlist of Chinese, and a string.
- Start a pointer at the beginning of the string
- Find the longest word in dictionary that matches the string starting at pointer
- Move the pointer over the word in string
- Go to 2

Maximum Matching Word Segmentation

- Thecatinthehat
- Thetabledownthere

the cat in the hat

the table down there

theta bled own there

- Doesn’t generally work in English!

Example

- `He sat on the chair, but he likes sitting on the floor.`

- N = ?
- V = ?

- Normalization: lowercasing, stemming, lemmatization
- stopwords removing, punctuation removing, vectorization

Stemming and Lemmatization

- Stemming: Reduces words to their base or root form by chopping off affixes (e.g., "running" → "run").

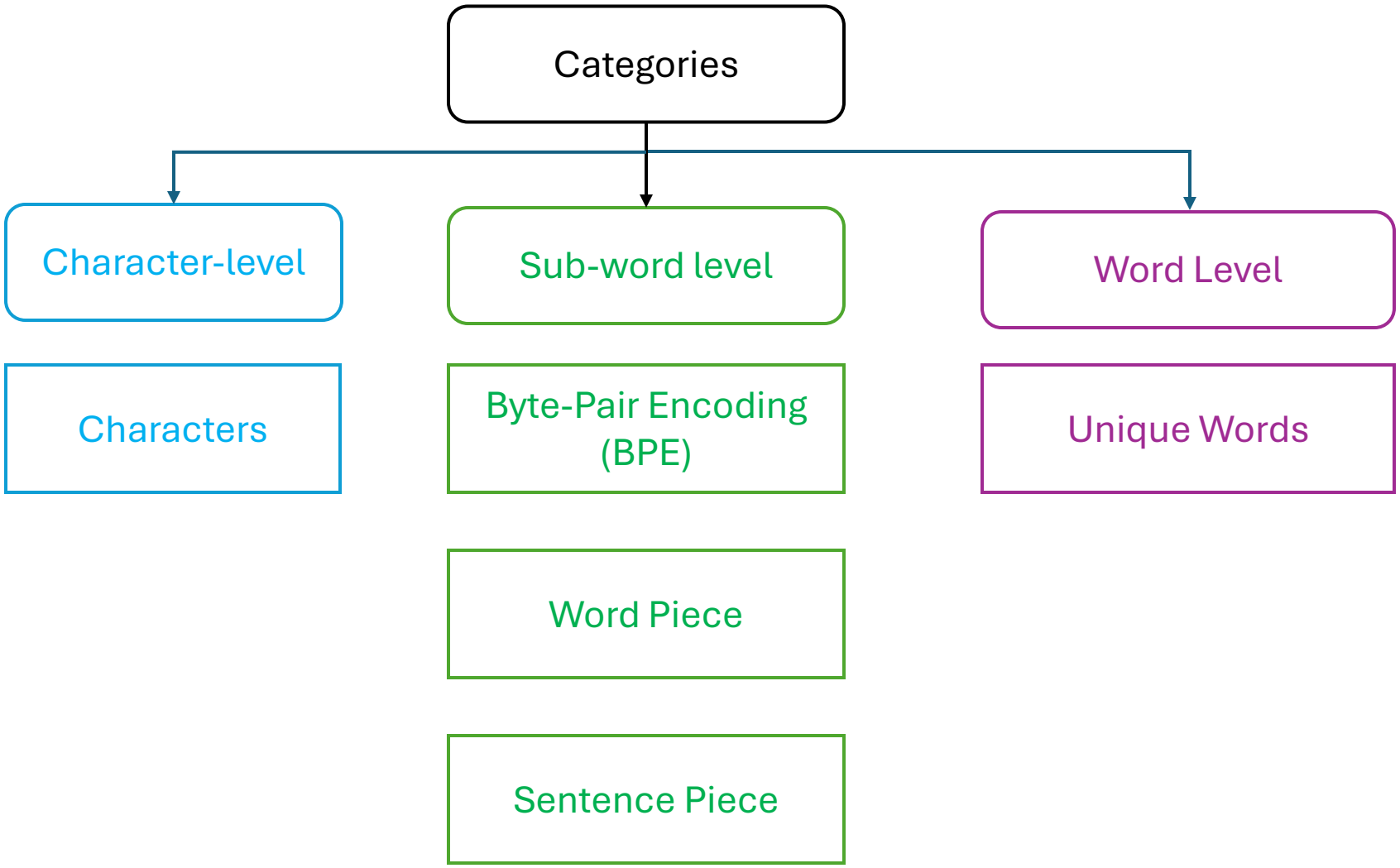
- Lemmatization: Converts words to their dictionary form (e.g., "better" → "good" or "running" → "run") using context and linguistic rules.

Stemming and Lemmatization

- He sat on the chair but he likes sitting on the floor
- he sit <SW> <SW> chair but he like sit <SW> <SW> floor
- N = ?
- V = ?
- <DATE>, <UNK>

Word Tokenization

Word Tokenization



Byte-Pair Encoding: A Bottom-up Tokenization Algorithm

Byte-Pair Encoding

- BPE is most commonly used by large language models for word tokenization.

- Instead of defining tokens as words (whether delimited by spaces or more complex algorithms), or as characters (as in Chinese), we can use our data to automatically tell us what the tokens should be.

- NLP algorithms often learn some facts about language from one corpus (a training corpus) and then use these facts to make decisions about a separate test corpus and its language.

Byte-Pair Encoding

- Thus, if our training corpus contains, say the words low, new, newer, but not lower, then if the word lower appears in our test corpus, our system will not know what to do with it.

- To deal with this unknown word problem, modern tokenizers automatically induce sets of tokens that include tokens smaller than words, called subwords.

Byte-Pair Encoding Algorithm

- The BPE algorithm starts with a vocabulary containing only individual characters.
- It scans the training corpus to find the two symbols that are most frequently adjacent (e.g., ‘A’ and ‘B’).
- A new merged symbol (e.g., ‘AB’) is added to the vocabulary, and every occurrence of adjacent ‘A’ and ‘B’ is replaced with ‘AB’ in the corpus.
- This process of counting and merging continues, forming longer character strings until k merges have been completed, resulting in k novel tokens.
- K is a parameter of the algorithm, determining the number of new tokens.
- The final vocabulary consists of the original set of characters plus the k new symbols.

Byte-Pair Encoding Algorithm

- The algorithm is usually run inside words (not merging across word boundaries).
- The input corpus is first white-space-separated to give a set of strings, each corresponding to the characters of a word, plus a special end-of-word symbol , and its counts.

Byte-Pair Encoding Algorithm

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Byte-Pair Encoding Algorithm

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w

Corpus of 18 word tokens with counts for each word (the word low appears 5 times, the word newer 6 times, and so on), which would have a starting vocabulary of 11 letters.

Byte-Pair Encoding Algorithm

corpus

```
5   l o w _
2   l o w e s t _
6   n e w er _
3   w i d er _
2   n e w _
```

vocabulary

```
_, d, e, i, l, n, o, r, s, t, w, er
```

Byte-Pair Encoding Algorithm

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Byte-Pair Encoding Algorithm

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2 l o w e s t _	
6 ne w er_	
3 w i d er_	
2 ne w _	

Byte-Pair Encoding Algorithm

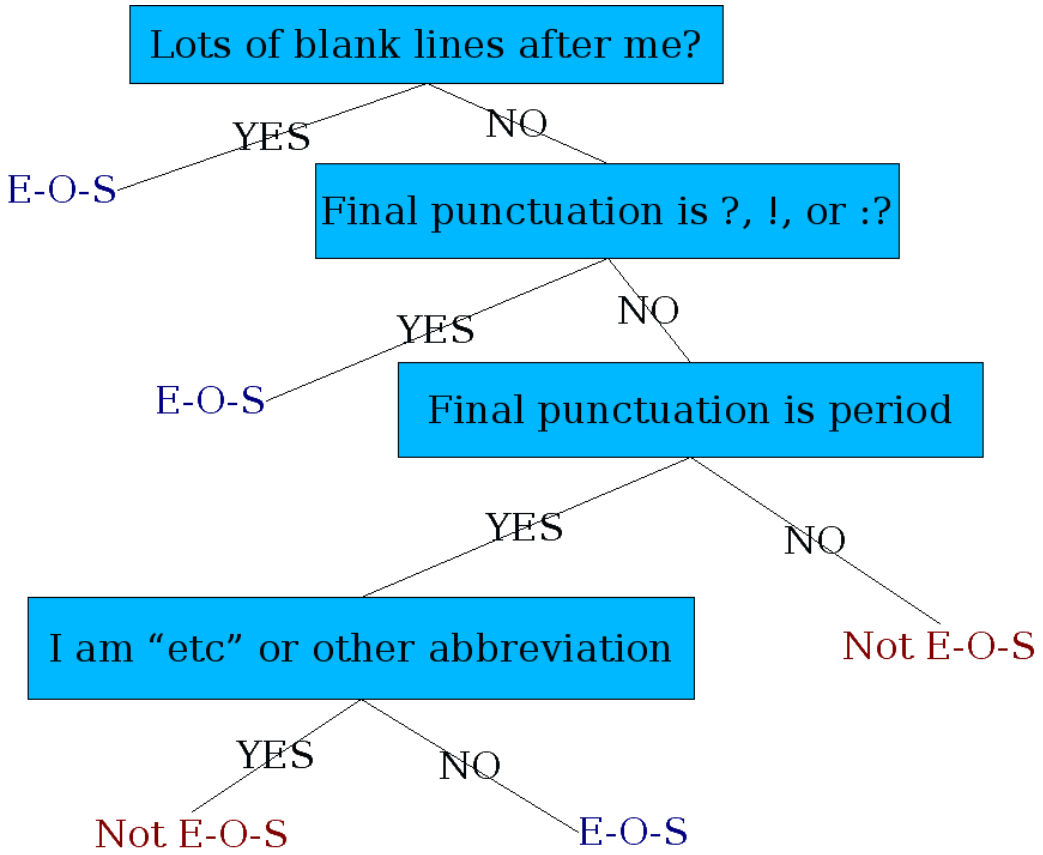
merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Sentence Segmentation

Sentence Segmentation

- `!, ?` are relatively unambiguous
- Period "`.`" is quite ambiguous
- Sentence boundary
  - Abbreviations like `Inc.` or `Dr.`
  - Numbers like `.02%` or `4.3`
- Build a binary classifier
  - Looks at a "`.`"
  - Decides End-of-Sentence/Not-End-of-Sentence
  - Classifiers: hand-written rules, regular expressions, or machine learning

Determining if a word is end-of-sentence



Language Models

Language Models

- A language model is a machine learning model LM that predicts upcoming words.

- More formally, a language model assigns a probability to each possible next word, or equivalently gives a probability distribution over possible next works.

- Language models can also assign a probability to an entire sentence.

Language Models

- Thus, an LM could tell us that the following sequence has a much higher probability of appearing in a text:

- `all of a sudden I notice three guys standing on the sidewalk`

- than does this same set of words in a different order:

- `on guys all I of notice sidewalk three a sudden standing the`

Language Models

$$
\begin{aligned}
P(X_1 \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1}) \\
&= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \\
P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \ldots P(w_n|w_{1:n-1}) \\
&= \prod_{k=1}^{n} P(w_k|w_{1:k-1})
\end{aligned}
$$

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Probability of the next word

- Islamabad is a capital of _____
- The baby started ____ when she saw her mother.
- The weather today is very ____.

آج موسم بہت ____ ہے۔

میں صبح اٹھ کر سب سے پہلے ____ پیتا ہوں۔

بارش کے بعد ہوا بہت ____ ہو گئی۔

Uses

- Machine Translation
- Spell Correction
- Speech Recognition
- Text Generation ...

Probability of a sentence

- `P(all of a sudden I notice three guys standing on the sidewalk)?`
- `P(on guys all I of notice sidewalk three a sudden standing the)?`
- Counting of such sentences?

N-Gram: Basics of Counting

- Sources for Language models are Corpora

- Count words forms not lemmas

N-Grams

- Let's begin with the task of computing `P(w|h)`, the probability of a word `w` given some history `h`.

- Suppose:
  - `h = "The water of Walden Pond is so beautifully"`

- The probability of blue is:
  - `P(w|h) = P(blue|The water of Walden Pond is so beautifully)`

N-Grams

- One way to count this is by relative frequency.
- Relative frequency-dividing the observed frequency of a particular sequence by the observed frequency of a prefix.
- This would be answering the question "Out of the times we saw the history h, how many times was it followed by the word w", as follows:

$$P(\text{blue}|\text{The water of Walden Pond is so beautifully}) = \frac{C(\text{The water of Walden Pond is so beautifully blue})}{C(\text{The water of Walden Pond is so beautifully})}$$

N-Grams

- If we had a large enough corpus, we could compute these two counts and estimate the probability.
- Let see on the Google search engine.

- “FCCU is the best university in Pakistan”
- Why?
- This is because language is creative.

Calculate the probability of a sentence

- We need more clever ways to estimate $P(w|h)$ or the probability of an entire word sequence W.

- Let $W = w_1, w_2, \ldots, w_n$ be a sentence
  - $P(W) = P(w_1, w_2, \ldots, w_n)$

- Now, the question is how we compute the $P(w_1, w_2, w_3, \ldots w_n)$?
  - $P(x,y) = p(x).p(y)$ if x and y are independent.
  - $P(x,y) = P(x).P(y|x)$ otherwise

Chain rule of probability

Chain rule of probability

- The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.

- We could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities.

Chain rule of probability

$$P(X_1 \ldots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1})$$
$$= \prod_{k=1}^{n} P(X_k|X_{1:k-1})$$

Applying the chain rule to words, we get

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \ldots P(w_n|w_{1:n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

$P(W) = P(w_1).P(w_2|w_1).P(w_3|w_1w_2).P(w_4|w_1w_2w_3)...P(w_n|w_1w_2...w_{n-1})$

The Markov assumption

The Markov assumption

- The probability of a word depends only on the previous word is called a Markov assumption.

$$P(\texttt{blue}|\texttt{The water of Walden Pond is so beautifully})$$

$$P(\texttt{blue}|\texttt{beautifully})$$

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

The Markov assumption (n-gram)

- The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.
  - N=1 → Unigram
  - N=2 → Bigram
  - N=3 → Trigram
  - N=4→ 4-gram

How to estimate probabilities

- In simple words, an intuitive way to estimate probabilities is called maximum likelihood estimation or MLE.

- We get the MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1.

How to estimate probabilities

- A bigram probability of a word w_n given a previous word w_{n-1}
  1. Compute the count of the bigram $C(w_{n-1}w_n)$
  2. Normalize by the sum of all the bigrams that share the same first word w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

How to estimate probabilities: an example

- Let's work through an example using a mini-corpus of three sentences.
- Augmenting sentences with <s> and </s>.

Estimates the n-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix.

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

How to estimate probabilities: Unigram

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Unigram	Count	Probability
I	3	3/|N|
Sam	2	2/|N|
am	2	2/|N|
do	1	1/|N|

…

How to estimate probabilities: Bigram

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Bigram	Count	Probability
<s> I	2	?
I am	2	?
am Sam	1	?
Sam </s>	1	?

...

$P(\text{I}|\text{<s>}) = \frac{2}{3} = 0.67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = 0.33$ $P(\text{am}|\text{I}) = \frac{2}{3} = 0.67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = 0.5$ $P(\text{do}|\text{I}) = \frac{1}{3} = 0.33$

How to estimate probabilities: Trigram

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Trigram	Count	Probability
<s> <s> I	2	?
<s> I am	1	?
I am Sam	1	?
am Sam </s>	1	?

…

Log probabilities

- Probabilities are always between 0 and 1.
- When we multiply many small probabilities, the result becomes even smaller and can approach zero (a problem called numerical underflow).
- To avoid this, we use logarithms:

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Sources

- https://web.stanford.edu/~jurafsky/slp3/3.pdf
- https://web.stanford.edu/~jurafsky/slp3/4.pdf