



ST. ANNE'S COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai)

Accredited by NAAC

ANGUCHETTYPALAYAM, PANRUTI – 607 106.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

**Subject Name : NATURAL LANGUAGE PROCESSING
SCIENCE**

Subject code : AL3501

Regulation : 2021

Year / Branch : III / AIML

Semester : V

**Faculty Name : Mr. R. MANICKAVASAGAN,
AP/AIML**

UNIT II

WORD LEVEL ANALYSIS

9

Unsmoothed N-grams, Evaluating N-grams, Smoothing, Interpolation and Backoff – Word Classes, Part-of-Speech Tagging, Rule-based, Stochastic and Transformation-based tagging, Issues in PoS tagging – Hidden Markov and Maximum Entropy models.

NATURAL LANGUAGE PROCESSING WORD LEVEL ANALYSIS

Word-level analysis in NLP involves examining individual words within a text to understand their structure, meaning, and relationships to other words. This includes techniques like tokenization, stemming, and part-of-speech tagging, which are crucial for various NLP tasks like text processing and understanding.

Key aspects of word-level analysis:

- **TOKENIZATION:**

The process of breaking down a text into individual units (tokens), which can be words, punctuation marks, or other meaningful elements. For example, the sentence "The quick brown fox" would be tokenized into: "The", "quick", "brown", "fox".

- **STEMMING:**

Reducing words to their root or base form. For example, "running", "runs", and "ran" would all be stemmed to "run".

- **LEMMATIZATION:**

Similar to stemming, but lemmatization considers the context and grammatical structure of the word to return its lemma (dictionary form). For example, "better" would be lemmatized to "good".

- **PART-OF-SPEECH (POS) TAGGING:**

Assigning grammatical tags to each word, indicating whether it's a noun, verb, adjective, etc. This helps understand the role of each word in a sentence.

- **MORPHOLOGICAL ANALYSIS:**

Examining the internal structure of words, breaking them down into morphemes (smallest units of meaning). This helps in understanding word formation and variations.

- **N-GRAMS:**

Sequences of n words. For example, bigrams (sequences of two words) like "New York" and trigrams (sequences of three words) like "I love NLP".

- **WORD EMBEDDINGS:**

Representing words as numerical vectors, capturing semantic relationships between words. Examples include Word2Vec, GloVe, and BERT.

- **REGULAR EXPRESSIONS:**

Used for pattern matching in text, useful for tasks like tokenization, identifying specific word patterns, and data cleaning.

Applications of Word-Level Analysis:

- **TEXT CLASSIFICATION:**

Analyzing word frequencies and relationships to categorize text into different classes (e.g., spam/not spam, positive/negative sentiment).

- **INFORMATION RETRIEVAL:** Identifying keywords and relevant terms to improve search engine results.
- **MACHINE TRANSLATION:** Understanding the grammatical structure and relationships between words in different languages.
- **CHAT BOTS AND CONVERSATIONAL AI:** Enabling chatbots to understand user input and generate appropriate responses.
- **SENTIMENT ANALYSIS:** Identifying the emotional tone of text by analyzing the words used.
- **Text Summarization:** Identifying key words and phrases to create concise summaries of longer texts. Word-level analysis is a fundamental aspect of NLP, providing the foundation for more complex language processing tasks. Understanding the structure and meaning of individual words is crucial for effectively processing and understanding human language.

UNSMOOTHED N-GRAMS

Unsmoothed n-grams in NLP refer to a method of calculating word sequence probabilities where the probabilities are directly derived from the observed frequencies in a training corpus, without any adjustments to account for unseen sequences. This means that if a specific sequence of words (an n-gram) doesn't appear in the training data, its probability will be zero, which can be problematic.

Here's a more detailed explanation:

What are N-grams?

N-grams are sequences of n consecutive items (words, characters, etc.) from a given text. Unigrams (1-grams) are individual words, bigrams (2-grams) are pairs of words, trigrams (3-grams) are sequences of three words, and soon.

Unsmoothed N-gram Probability Calculation:

In an unsmoothed n-gram model, the probability of an n-gram is calculated as:

$$P(w_1, w_2, \dots, w_n) = \text{Count}(w_1, w_2, \dots, w_n) / \text{Count}(w_1, w_2, \dots, w_{n-1})$$

Where:

- $P(w_1, w_2, \dots, w_n)$ is the probability of the word sequence.
- $\text{Count}(w_1, w_2, \dots, w_n)$ is the number of times the entire sequence appears in the training data.
- $\text{Count}(w_1, w_2, \dots, w_{n-1})$ is the number of times the preceding (n-1) words appear.

The Problem of Zero Probabilities:

The main issue with unsmoothed n-grams is that they assign a probability of zero to any n-gram that doesn't exist in the training data. This is a significant limitation because:

- **DATA SPARSITY:**

Real-world text data is vast, but it's still finite. There will always be n-grams present in the evaluation data that were not seen during training.

- **MODEL INSTABILITY:**

Assigning a zero probability to unseen sequences can lead to inaccurate predictions and instability in language models.

Example:

Let's say your training data contains the following bigrams: "the cat", "cat sat", "sat on". If you then encounter the bigram "the dog" (which is not in the training data), an unsmoothed bigram model would assign it a probability of zero.

In essence, unsmoothed n-grams are a basic approach to language modeling, but they are rarely used in practice due to the zero-probability problem. Smoothing techniques are typically employed to address this issue.

EVALUATING N-GRAMS

N-gram models in NLP are evaluated using both intrinsic and extrinsic methods. Intrinsic evaluation focuses on metrics like perplexity, which measures how well the model predicts the next word in a sequence, with lower perplexity indicating better performance. Extrinsic evaluation assesses the model's performance in a real-world application, such as speech recognition or machine translation, by measuring its impact on the overall system's accuracy.

Intrinsic Evaluation:

- **PERPLEXITY:**

This metric quantifies the uncertainty of a language model. A lower perplexity score means the model is more certain about the sequence of words it is predicting, indicating better performance. Perplexity is calculated as the inverse probability of the test sentence, normalized by the number of words, according to a blog post.

- **OTHER METRICS:**

While perplexity is common, other metrics can be used depending on the specific task and model. These might include measures of word or sentence similarity, or metrics tailored to specific applications.

Extrinsic Evaluation:

- **Real-World Applications:**

This involves testing the n-gram model within a larger application, like a speech recognition system or a machine translation engine. The evaluation focuses on how well the model contributes to the overall success of the application.

- **End-to-End Testing:**

This approach is more time-consuming and expensive but provides a more realistic assessment of the model's practical value, [according to Engati](#).

Key Considerations:

- **Corpus Size:**

The size and quality of the training data significantly impact the performance of n-gram models. Larger and more representative corpora generally lead to better results.

- **Markov Assumption:**

N-gram models rely on the Markov assumption, which simplifies calculations by assuming that the probability of a word only depends on the preceding n-1 words. This assumption can limit the model's ability to capture long-range dependencies in text.

- **Language Evolution:**

Language changes over time, which can affect the accuracy of n-gram models. New words and phrases emerge, while others become obsolete, requiring models to be updated or adapted.

- **Computational Cost:**

Building and evaluating large n-gram models can be computationally expensive, especially for high-order n-grams.

- **Pruning:**

Techniques like pruning are used to reduce the size of n-gram models by removing infrequent n-grams, but this can also impact accuracy, [says NVIDIA Docs](#).

Smoothing

In NLP, smoothing refers to techniques used to adjust probabilities in statistical language models to handle unseen n-grams (sequences of words) and improve model generalization. Essentially, it prevents zero probabilities for word combinations not encountered during training, allowing the model to make more robust predictions on new data.

Why is smoothing needed?

- **Zero Probability Issues:**

Language models trained on finite datasets often encounter n-grams that weren't present in the training data. Standard probability calculations would assign a probability of zero to these unseen n-grams, making it impossible to predict their occurrence or use them in larger sequences.

- **Overfitting:**

Without smoothing, models can overfit to the training data, performing poorly on data outside the training set.

- **Generalization:**
Smoothing techniques help models generalize better by distributing some probability mass to unseen n-grams, allowing them to handle variations in language more effectively.
Common Smoothing Techniques:
- **Laplace Smoothing (Add-one Smoothing):**
Adds one to all counts, including those of unseen n-grams.
- **Add-k Smoothing:***
* Similar to Laplace, but adds a value 'k' (usually between 0 and 1) instead of 1.
- **Good-Turing Smoothing:***
* Adjusts counts based on how often they appear (frequency of frequencies), giving more weight to rare events.
- **Kneser-Ney Smoothing:***
* A more advanced technique that considers the number of unique contexts a word appears in, rather than just the raw count.
- **Witten-Bell Smoothing:***
* Uses a combination of counts and the number of unique words to estimate probabilities.
In essence, smoothing ensures that language models assign non-zero probabilities to all possible n-grams, even those not seen during training, leading to more reliable and accurate predictions.

Interpolation and Backoff

In Natural Language Processing (NLP), interpolation and backoff are smoothing techniques used in language modeling, particularly with n-gram models, to address the issue of data sparsity and unseen events.

Backoff:

Backoff is a strategy where, if a higher-order n-gram (e.g., a trigram) is not found in the training data or has insufficient counts, the model "backs off" to a lower-order n-gram (e.g., a bigram or unigram) to estimate the probability. This process continues until a sufficiently frequent n-gram is found. For example, in Katz's Backoff model, if a trigram is unseen, the probability is estimated using a discounted bigram probability, and if the bigram is also unseen, it backs off to a unigram probability.

Interpolation:

Interpolation, in contrast, combines the probabilities from multiple n-gram models (e.g., unigram, bigram, and trigram) at all times, using a weighted average. Each n-gram model contributes to the final probability estimate, with weights typically assigned to reflect the importance of each order (e.g., higher weights for higher-order n-grams). The sum of these weights usually equals one. This ensures that all available information from different n-gram orders is utilized, even if a higher-order n-gram is present.

Key Differences:

- **Strategy:**
Backoff is a sequential, conditional approach that falls back to lower orders only when higher orders are absent or unreliable. Interpolation is a parallel, always-on approach that

combines all orders simultaneously.

- **Data Sparsity Handling:**

Backoff directly addresses unseen n-grams by resorting to lower orders. Interpolation smooths probabilities by combining information from various orders, even for seen n-grams, reducing the impact of low counts.

- **Performance:**

Interpolation often performs better on smaller datasets as it leverages all available information. Backoff, particularly "stupid backoff," can be more efficient on very large datasets due to its simpler probability estimation when counts are high.

In Natural Language Processing (NLP), word classes, also known as [parts of speech](#), are categories that classify words based on their grammatical function within a sentence. Common word classes include nouns, verbs, adjectives, adverbs, prepositions, pronouns, conjunctions, and interjections. Understanding these classes is crucial for tasks like [sentence parsing](#), [machine translation](#), and information retrieval.

Here's a breakdown with examples:

- **Nouns:** Represent people, places, things, or ideas (e.g., dog, house, freedom, happiness).
- **Verbs:** Indicate actions or states of being (e.g., run, sleep, is, think).
- **Adjectives:** Modify or describe nouns (e.g., big, red, happy).
- **Adverbs:** Modify verbs, adjectives, or other adverbs (e.g., quickly, very, extremely).
- **Prepositions:** Show relationships between nouns/pronouns and other words (e.g., on, in, under).
- **Pronouns:** Replace nouns (e.g., he, she, it, they).
- **Conjunctions:** Connect words, phrases, or clauses (e.g., and, but, or).
- **Interjections:** Express sudden emotion (e.g., Ouch!, Wow!).

Example:

In the sentence, "The quick brown fox jumps over the lazy dog," the words can be classified as follows:

- "The": Determiner (DT)
- "quick": Adjective (JJ)
- "brown": Adjective (JJ)
- "fox": Noun (NN)
- "jumps": Verb (VBZ)
- "over": Preposition (IN)
- "the": Determiner (DT)
- "lazy": Adjective (JJ)
- "dog": Noun (NN)

Part-of-Speech Tagging

Part-of-speech (POS) tagging in NLP is the process of assigning grammatical tags (like noun, verb, adjective, etc.) to each word in a sentence. This helps computers understand sentence

structure and the role of each word. For example, the sentence "The quick brown fox jumps over the lazy dog" would be tagged as: "The (DT) quick (JJ) brown (JJ) fox (NN) jumps (VBZ) over (IN) the (DT) lazy (JJ) dog (NN)".

Here's a more detailed breakdown:

What is POS tagging?

- POS tagging is a fundamental task in NLP that involves identifying the grammatical category of each word in a sentence.
- These categories, or tags, include nouns, verbs, adjectives, adverbs, pronouns, prepositions, conjunctions, and interjections.
- By assigning these tags, computers can better understand the structure and meaning of sentences.

Example:

Let's take the sentence: "The quick brown fox jumps over the lazy dog."

- "The" is a determiner (DT).
- "quick" and "brown" are adjectives (JJ).
- "fox" is a noun (NN).
- "jumps" is a verb (VBZ).
- "over" is a preposition (IN).
- "lazy" is an adjective (JJ).
- "dog" is a noun (NN).

Why is it important?

- **Understanding Sentence Structure:**

POS tagging helps NLP systems understand how words relate to each other in a sentence.

- **Ambiguity Resolution:**

Words can have multiple meanings depending on their context. POS tagging helps to resolve this ambiguity by assigning the correct grammatical role.

- **Downstream NLP Tasks:**

POS tagging is a crucial pre-processing step for many other NLP tasks, such as [named entity recognition](#), [dependency parsing](#), and [machine translation](#).

How does it work?

- POS tagging can be done using various methods, including rule-based systems, statistical models (like [Hidden Markov Models](#)), and neural networks.
- **NLTK:** The Natural Language Toolkit (NLTK) provides tools for POS tagging, including the `pos_tag` function.
- **SpaCy:** SpaCy is another popular NLP library that offers efficient POS tagging capabilities, including detailed tag explanations.

A common example of a rule-based system in Natural Language Processing (NLP) is a grammar checker.

Here's how a rule-based grammar checker operates:

- **Predefined Rules:**

It utilizes a set of explicit, human-defined linguistic rules. These rules cover various grammatical aspects, such as:

- **Subject-verb agreement:** "If the subject is singular, the verb must be singular."
- **Punctuation:** "A period must follow a complete sentence."
- **Article usage:** "Use 'an' before words starting with a vowel sound."
- **Common spelling errors:** Specific rules for frequently misspelled words.

Pattern Matching:

The system analyzes input text and attempts to match patterns within the text against these predefined rules.

Error Identification and Correction:

If a portion of the text violates a rule, the system identifies it as an error and may suggest a correction based on the rule.

Example Scenario:

Consider the sentence: "The cat eat the fish."

A rule-based grammar checker would apply its rule for subject-verb agreement. It would identify "cat" as a singular subject and "eat" as a plural verb form. Since this violates the rule, the system would flag "eat" as incorrect and suggest changing it to "eats" to achieve subject-verb agreement.

In Natural Language Processing (NLP), stochastic and transformation-based approaches are two distinct yet sometimes complementary methods used, particularly in tasks like Part-of-Speech (POS) tagging.

1. Stochastic Approach (Probabilistic):

This approach relies on probability and statistics to assign the most likely tag to a word based on its frequency and context within a corpus. Hidden Markov Models (HMMs) are a classic example.

- **Example:** Consider the word "can" in the sentences:

- "I can swim."
- "Open the can."

A stochastic tagger, trained on a large corpus, learns the probability of "can" being a modal verb (MD) when followed by a verb, and the probability of "can" being a noun (NN) when preceded by a determiner like "the." It then assigns the tag with the highest probability based on the observed context.

2. Transformation-Based Learning (TBL) / Brill Tagging:

This approach, also known as Brill tagging, starts with an initial tagging (often using a simple rule-based or stochastic method) and then applies a series of learned transformation rules to correct errors. These rules are automatically induced from data.

- **Example:** Imagine an initial tagger tags "can" as a modal verb (MD) in "The can was rusted." A transformation rule might be: "Change the tag of a word from MD to NN if it is preceded by a determiner (DT)." Applying this rule would correct the initial error, transforming the tag of "can" from MD to

NN, recognizing its role as a noun in that specific context.

In essence:

- **Stochastic:** methods calculate probabilities to make tagging decisions.
- **Transformation-based:** methods learn and apply rules to refine initial tag assignments, often drawing inspiration from both rule-based and stochastic principles.

-based tagging

Rule-based tagging in Natural Language Processing (NLP) involves assigning grammatical categories (Part-of-Speech tags) to words based on predefined linguistic rules. These rules can consider word characteristics, context, and patterns.

Example:

Consider the sentence: "The man fans the flame."

A rule-based tagger might apply rules like the following:

- **Rule 1:** Assign "DT" (Determiner) to words like "the," "a," "an."
 - Result: "The" is tagged as DT.
- **Rule 2:** Assign "NN" (Noun) to words that are common nouns and follow a determiner.
 - Result: "man" is tagged as NN.
- **Rule 3:** Assign "VBZ" (Verb, third person singular present) to words that are verbs and appear after a singular noun or pronoun.
 - Result: "fans" is tagged as VBZ (as in, the man performs the action of fanning).
- **Rule 4:** Reapply Rule 1 for "the."
 - Result: "the" is tagged as DT.
- **Rule 5:** Reapply Rule 2 for "flame."
 - Result: "flame" is tagged as NN.

Output:

The resulting tagged sentence would be:

Code

```
[('The', 'DT'), ('man', 'NN'), ('fans', 'VBZ'), ('the', 'DT'), ('flame', 'NN')]
```

This example demonstrates how rules based on word form and surrounding context can be used to determine the grammatical role of each word in a sentence. Rule-based systems are often handcrafted and can be effective for specific linguistic phenomena, though they can struggle with ambiguity and require extensive rule sets for comprehensive coverage.

Issues in PoS tagging

Part-of-speech (POS) tagging, a fundamental task in Natural Language Processing (NLP), faces several challenges, primarily stemming from ambiguity in word usage and the limitations of training data. Key issues include ambiguity in word meaning, handling out-of-vocabulary words, dealing with complex grammatical structures, and the availability and quality of training data.

1. Ambiguity:

- Many words can have multiple POS tags depending on the context, making it difficult to determine the correct tag.
- For example, "book" can be a noun (e.g., "I need a book") or a verb (e.g., "Book the flight").
- Similarly, "this" can be a determiner, a pronoun, or an adverb.
- POS taggers need to resolve this ambiguity by considering the surrounding words and sentence structure.

2. Out-of-Vocabulary (OOV) Words:

- Words not encountered during training can be difficult to tag correctly.
- This is especially problematic for rare words, proper nouns, or words specific to a particular domain.
- For instance, if a tagger is trained on general text, it might struggle with technical jargon or slang.

3. Complex Grammatical Structures:

- Languages with rich morphology (inflections, conjugations) or free word order present unique challenges.
- These structures can make it harder to establish clear patterns for POS tagging.
- Examples include languages like Russian or Czech, which have extensive inflectional systems.

4. Training Data Issues:

• **Limited Data:**

Some languages or domains may lack sufficient annotated data for training POS taggers.

• **Inconsistencies:**

Training data might contain errors or inconsistencies, which can negatively impact the tagger's performance.

• **Data Bias:**

Training data might not be representative of the language or domain being used, leading to biased tagging.

5. Domain Dependence:

- POS taggers trained on one domain might not generalize well to other domains.
- A tagger trained on news articles might not perform as well on social media text or scientific papers.

6. Propagation of Errors:

- Errors in POS tagging can propagate to downstream NLP tasks like parsing, semantic analysis, or machine translation, affecting overall accuracy.
- For example, if a verb is incorrectly tagged as a noun, subsequent parsing steps might fail to identify the correct sentence structure.

Hidden Markov and Maximum Entropy models

Hidden Markov Models (HMMs) and Maximum Entropy (MaxEnt) models are both used in Natural Language Processing (NLP) for sequence labeling tasks, but they approach the

problem differently. HMMs are generative models, while MaxEnt models are discriminative. A Maximum Entropy Markov Model (MEMM) combines aspects of both, using a discriminative approach with features and normalization while also considering state dependencies.

Hidden Markov Model (HMM)

- **Generative Model:**

HMMs model the joint probability of observations and hidden states.

- **Structure:**

HMMs consist of hidden states (e.g., part-of-speech tags) and observable outputs (e.g., words).

- **Example:**

In part-of-speech (POS) tagging, the hidden states are the POS tags, and the observations are the words in a sentence.

- **Key Components:**

HMMs use transition probabilities (probability of moving from one state to another) and emission probabilities (probability of observing a word given a state).

- **Training:**

HMMs are trained to estimate these probabilities, often using algorithms like the [Baum-Welch algorithm](#) or the [Viterbi algorithm](#).

Maximum Entropy (MaxEnt) Model

- **Discriminative Model:**

MaxEnt models directly model the conditional probability of a state given the observations.

- **Features:**

MaxEnt models use various features (e.g., words, capitalization, part-of-speech tags) to predict the state.

- **Example:**

In POS tagging, a MaxEnt model might consider features like the current word, its capitalization, the previous word, and the next word.

- **Training:**

MaxEnt models are trained to find the probability distribution that maximizes entropy (uniformity) while satisfying constraints based on the observed data and features.

Maximum Entropy Markov Model (MEMM)

- **Combination:**

MEMMs combine the strengths of both HMMs and MaxEnt models.

- **State Dependencies:**

They model state transitions and observation probabilities like HMMs, but they use a discriminative approach with features like MaxEnt models.

- **Advantages:**

MEMMs can leverage richer feature sets than HMMs and can model state dependencies

more effectively.

- **Example:**
In information extraction, MEMMs can consider various overlapping features of the observation sequence (words, capitalization, etc.) and the previous state to predict the current state.