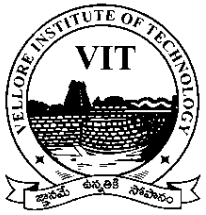


VIT-AP UNIVERSITY

Natural Language Processing

(Course Code: CSE 3015)

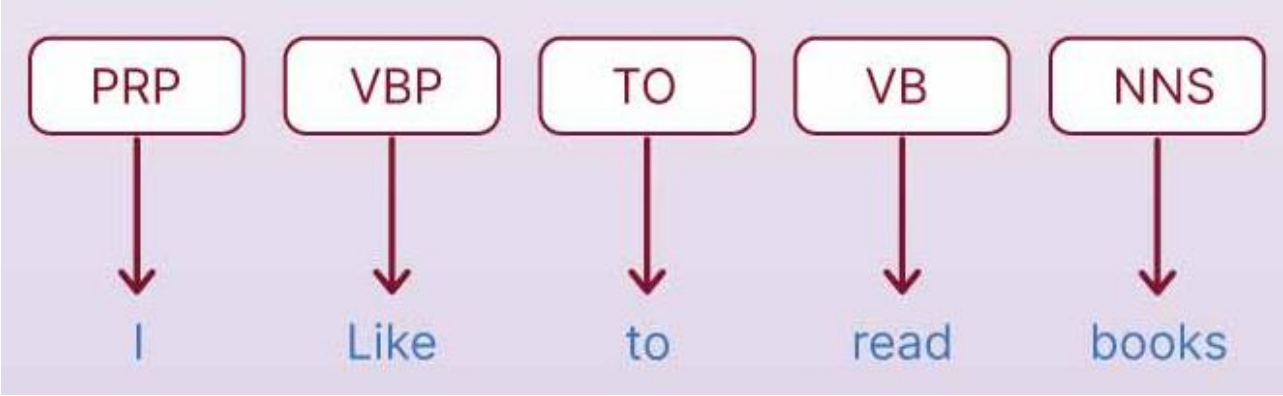
Module-2:Lecture-2: Parts of Speech Tagging

Gundimeda Venugopal, Professor of Practice, SCOPE

What is Parts of Speech (POS) tagging?

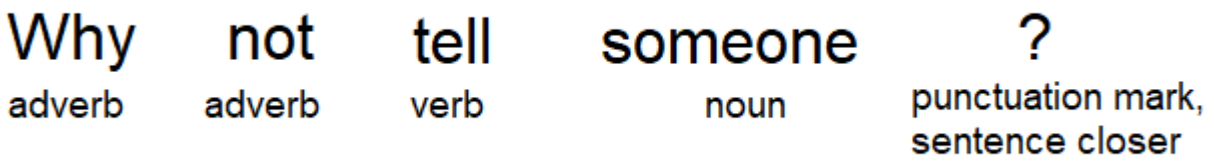
- Parts of Speech (POS) tagging is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)).
- The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.
- A POS tag (or part-of-speech tag) is a special label assigned to each token (word) in a text corpus to indicate the part of speech and often also other grammatical categories such as tense, number plural/singular), case etc.
- POS tags are used in corpus searches and in text analysis tools and algorithms.

Tag	Description
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Pre determiner
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle



Part Of Speech Tagging

❖Part-of-speech (POS) tagging is a popular Natural Language Processing process which refers to categorizing words in a text (corpus) in correspondence with a particular part of speech, depending on the definition of the word and its context.



❖In above figure, we can see **each word has its own lexical term** written underneath, however, having to constantly write out these full terms when we perform text analysis can very quickly become cumbersome — especially as the size of the corpus grows. Then, we use a short representation referred to as “tags” to represent the categories.

Part-of-Speech (POS) Tags

1. **Noun**: A noun is the name of a person, place, thing, or idea.
2. **Pronoun**: A pronoun is a word used in place of a noun.
3. **Verb**: A verb expresses action or being.
4. **Adjective**: An adjective modifies or describes a noun or pronoun.
5. **Adverb**: An adverb modifies or describes a verb, an adjective, or another adverb.
6. **Preposition**: A preposition is a word placed before a noun or pronoun to form a phrase modifying another word in the sentence.
7. **Conjunction**: A conjunction joins words, phrases, or clauses.
8. **Interjection**: An interjection is a word used to express emotion.
9. **Determiner or Article**: A grammatical marker of definiteness (the) or indefiniteness (a, an).

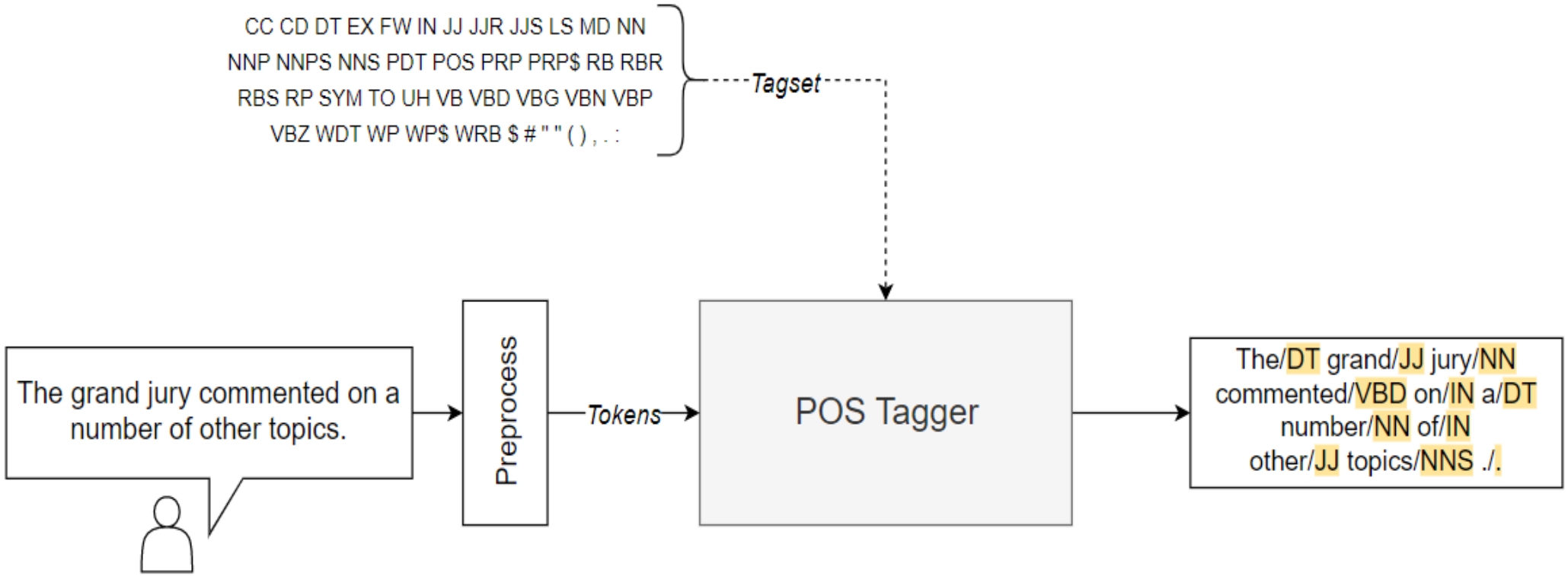
Let's take an example,

Text: "The cat sat on the mat."

POS tags:

- The: determiner
- cat: noun
- sat: verb
- on: preposition
- the: determiner
- mat: noun

POS Tagging architecture



Write the POS Tag for the Sentence

1. It is a nice night.

It/PRP is/VBZ a/DT nice/JJ night/NN ./.

2. Time flies like an Arrow

Time/[V,N] flies/[V,N] like/[V,Prep] an/Det arrow/N

Time/NN flies/V like/IN an/Det arrow/NN

3. Fruit flies like a banana

Fruit/NN flies/NN like/VB a/DET banana/NN

Fruit/NN flies/VB like/VB a/DET banana/NN (not correct)

4. The flies like a banana

The/Det flies/N like/VB a/DET banana/NN

Penn Treebank Tagset

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WPS	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	$
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

POS Tagset

- Tagset is the collection of tags from which the tagger finds appropriate tags and attaches to the word
- Different POS tag sets are used depending on the language and the application.

Tagset	Number of Tags
Penn Treebank	36+ (includes punctuation)
Universal Dependencies (UD)	17
Brown Corpus	87
CLAWS	70-100
Stanford POS Tagset	47
Simplified Tagset (UD)	12

As earlier mentioned, the process of assigning a specific tag to a word in our corpus is referred to as part-of-speech tagging (POS tagging for short) since the POS tags are used to describe the lexical terms that we have within our text.

Lexical Term	Tag	Example
Noun	NN	Paris, France, Someone, Kurtis
Verb	VB	work, train, learn, run, skip
Determiner	DT	the, a
...	...	

Why	not	tell	someone	?
WRB	**RB**	**VB**	**NN**	**.**

POS Tagging Challenges

Some common challenges in part-of-speech (POS) tagging include:

- **Ambiguity:** Some words can have multiple POS tags depending on the context in which they appear, making it difficult to determine their correct tag.
  - For example, the word “bass” can be a noun (a type of fish) or an adjective (having a low frequency or pitch).
  - “bank” can be a financial institution or river bank
- **Out-of-vocabulary (OOV) words:** Words that are not present in the training data of a POS tagger can be difficult to tag accurately, especially if they are rare or specific to a particular domain.
- **Complex grammatical structures:** Languages with complex grammatical structures, such as languages with many inflections (e.g., Ancient Greek) or free word order , can be more challenging to tag accurately.
- **Lack of annotated training data:** Some languages or domains may have limited annotated training data, making it difficult to train a high-performing POS tagger.
- **Inconsistencies in annotated data:** Annotated data can sometimes contain errors or inconsistencies, which can negatively impact the performance of a POS tagger.

POS Tagging Approaches

1. Rules-based POS tagging

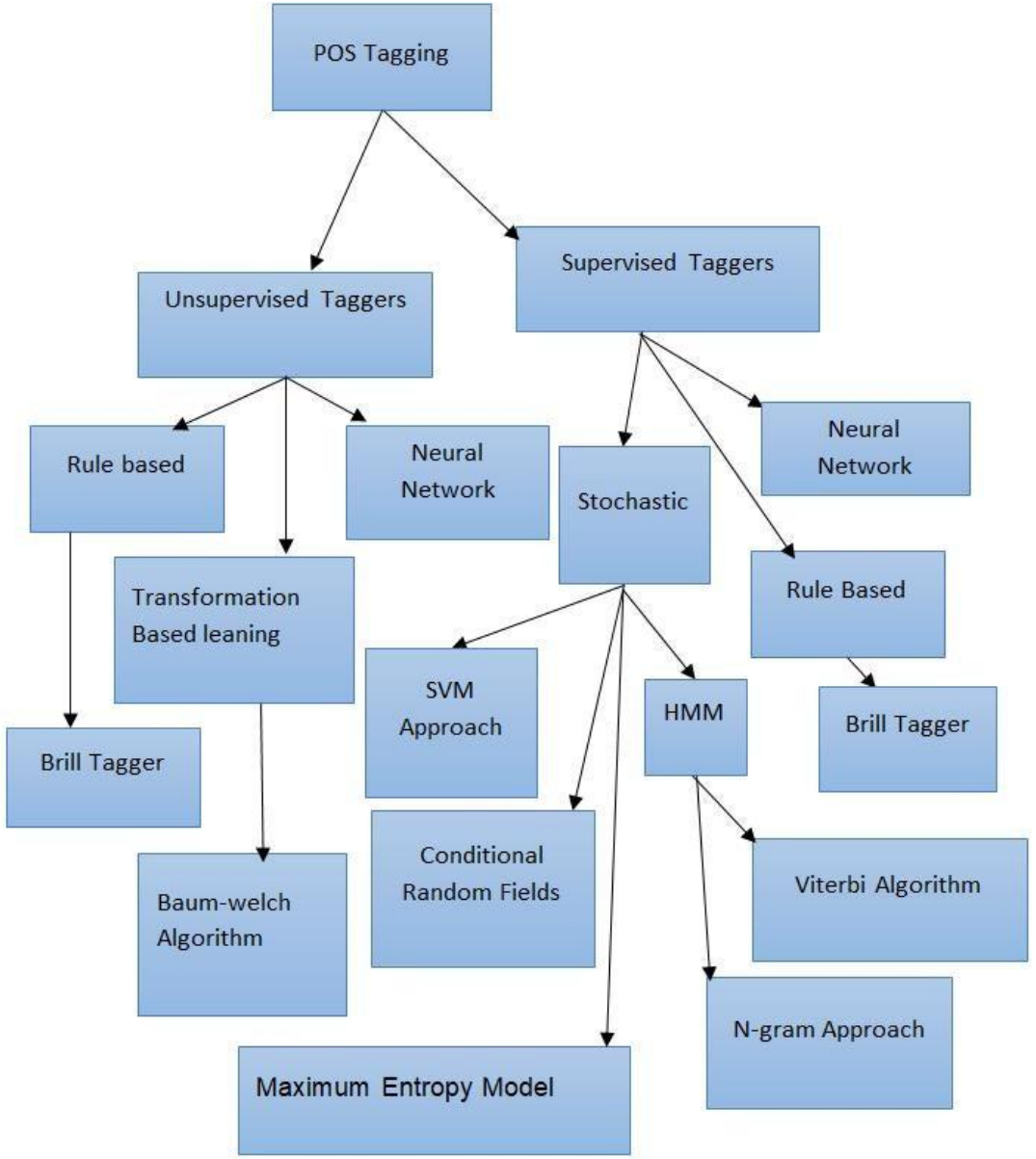
- Human crafted rules on lexical and other linguistic knowledge
- Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word.
- If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag.
- Disambiguation by analyzing the linguistic features of a word along with its preceding as well as following words.
- Example Rule: "if the preceding word of a word is article, then word must be a noun."

2. Learning based: Trained on human annotated corpora like the Penn Treebank

- Statistical models (Stochastic/ Probabilistic POS Tagging):
  - A stochastic approach involves frequency, probability or statistics.
  - Resolve Tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in the given context.
  - Examples include N-grams, Hidden Markov Model (HMM), Maximum Entropy Markov Model (MEMM) and Conditional Random Fields (CRF).
- Rule Learning: Transformation based Learning (TBL)
  - The transformation-based approaches use a pre-defined set of handcrafted rules as well as automatically induced rules that are generated during training.
  - Transformation based tagging is also called Brill tagging. It is an instance of the transformation-based learning (TBL), which is a rule-based algorithm for automatic tagging of POS to the given text.
  - TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.
  - Learns symbolic rules based on a corpus
- Neural networks: Recurrent networks like Long Short Term Memory (LSTM).

3. Ensemble Methods: Combine the results of multiple taggers.

POS Tagging Approaches



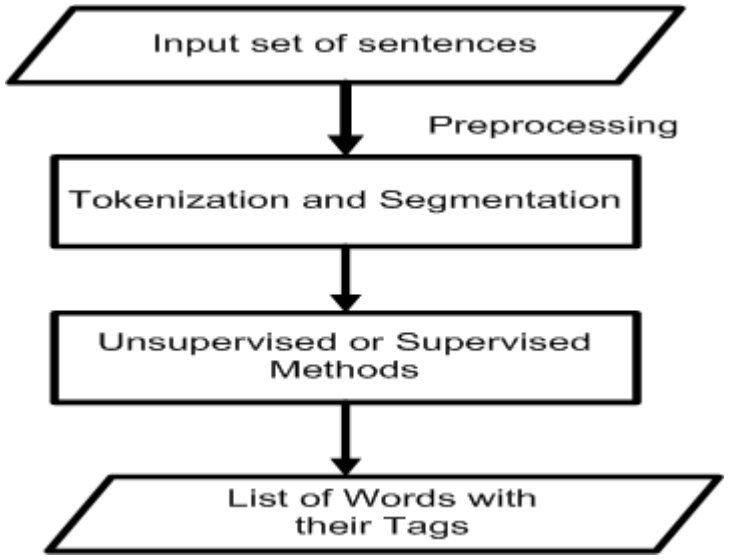
How Parts of Speech Tagging is useful?

❖ There are several reasons why we might tag words with their parts of speech (POS) in natural language processing (NLP):

- **To understand the grammatical structure of a sentence:** By labeling each word with its POS, we can better understand the syntax and structure of a sentence. This is useful for tasks such as machine translation and information extraction, where it is important to know how words relate to each other in the sentence. We can use them for making assumptions about semantics.
- **To disambiguate words with multiple meanings:** Some words, such as "bank," can have multiple meanings depending on the context in which they are used. By labeling each word with its POS, we can disambiguate these words and better understand their intended meaning.
- **To improve the accuracy of NLP tasks:** POS tagging can help improve the performance of various NLP tasks, such as Named Entity Recognition, Co-reference Resolution and Text Classification. By providing additional context and information about the words in a text, we can build more accurate and sophisticated algorithms.
- **To facilitate research in linguistics:** POS tagging can also be used to study the patterns and characteristics of language use and to gain insights into the structure and function of different parts of speech.

Note: When we perform POS tagging, it's often the case that our tagger will encounter words that were not within the vocabulary that was used. Consequently, augmenting your dataset to include unknown word tokens will aid the tagger in selecting appropriate tags for those words.

Part-of-speech (POS) tagging Steps



Steps involved in a typical part-of-speech (POS) tagging example:

1. Collect a dataset of annotated text: This dataset will be used to train and test the POS tagger. The text should be annotated with the correct POS tags for each word.
2. Preprocess the text: This may include tasks such as tokenization (splitting the text into individual words), lowercasing, and removing punctuation.
3. Divide the dataset into training and testing sets: The training set will be used to train the POS tagger, and the testing set will be used to evaluate its performance.
4. Train the POS tagger: This may involve building a statistical model, such as a hidden Markov model (HMM), or defining a set of rules for a rule-based or transformation-based tagger. The model or rules will be trained on the annotated text in the training set.
5. Test the POS tagger: Use the trained model or rules to predict the POS tags of the words in the testing set. Compare the predicted tags to the true tags and calculate metrics such as precision and recall to evaluate the performance of the tagger.
6. Fine-tune the POS tagger: If the performance of the tagger is not satisfactory, adjust the model or rules and repeat the training and testing process until the desired level of accuracy is achieved.
7. Use the POS tagger: Once the tagger is trained and tested, it can be used to perform POS tagging on new, unseen text. This may involve preprocessing the text and inputting it into the trained model or applying the rules to the text. The output will be the predicted POS tags for each word in the text.

POS Tagging applications

1. **Information extraction:** POS tagging can be used to identify specific types of information in a text, such as names, locations, and organizations. This is useful for tasks such as extracting data from news articles or building knowledge bases for artificial intelligence systems.

2. **Named entity recognition:** POS tagging can be used to identify and classify named entities in a text, such as people, places, and organizations. This is useful for tasks such as building customer profiles or identifying key figures in a news story.

3. **Text classification:** POS tagging can be used to help classify texts into different categories, such as spam emails or sentiment analysis. By analyzing the POS tags of the words in a text, algorithms can better understand the content and tone of the text.

4. **Machine translation:** POS tagging can be used to help translate texts from one language to another by identifying the grammatical structure and relationships between words in the source language and mapping them to the target language.

5. **Natural language generation:** POS tagging can be used to generate natural-sounding text by selecting appropriate words and constructing grammatically correct sentences. This is useful for tasks such as chatbots and virtual assistants.

6. Other applications of POS tagging include: Co-reference Resolution and Speech Recognition

Text Sequences: Words and their Roles

- Words are sequential building blocks of sentences. Each word contributes syntactic and semantic properties to a sentence.

For example, a word can be an adjective, which can give positive semantics (delicate). By doing that, this adjective can describe a noun (delicate boy). This sequential relationship can go recursively and unbounded. This shows us words are related to each other.

- A sequence is a series of tokens where tokens are not independent of each other. Series in mathematics and sentences in linguistics are both sequences. Because; in both of them, the next token depends on the previous ones or vice versa.

Sequence labeling

❖ Sequence labeling is a NLP task that is used to identify and label the components of a sequence, such as words or phrases in a sentence.

❖Sequence labeling has been one of the most discussed topics in Linguistics and Computational Linguistics history. Challenges like Dependency Parsing, Word Sense Disambiguation and Sequence Labeling etc. arose with the formal definition of the syntax.

❖ Sequence labeling aims to classify each token (word) in a class space **C**. This classifying approach can be independent (each word is treated as an independent), or dependent (each word is dependent on other words).

Sequence Labeling Tasks

- ❖ Parts of Speech tagging
- ❖ Named Entity Recognition
- ❖ Chunking and Semantic Role labeling

Sequence Labeling Models

- ❖ Sequence labeling can be done with various methods. While traditional models are based on corpus statistics (Hidden Markov Models, Maximum Entropy Markov Models, Conditional Random Field, etc.), recent models are based on neural networks (Recurrent Neural Networks, Long Short-Term Memory, BERT, etc.).

Sequence Role Labeling

❖ Chunking:

- Chunking is a task of sequence labeling that involves dividing a sequence of words into chunks or non-overlapping sub-sequences.
- These chunks are typically tagged with a label that indicates their type or role in the sequence.

Sentence: "*The quick brown fox jumps over the lazy dog.*"

After Chunking:

"***The quick brown fox***" (noun phrase)
"***jumps***" (verb)
"***over the lazy dog***" (noun phrase)

❖ Named Entity Recognition

- Named entity recognition (NER) is the task of identifying and classifying named entities (such as people, organizations, Geo Political Entity (GPE) and locations) in text.

```
Sentence = "Barack Obama was born in Hawaii."

# Output : [('PERSON', 'Barack Obama'), ('GPE', 'Hawaii')]
```

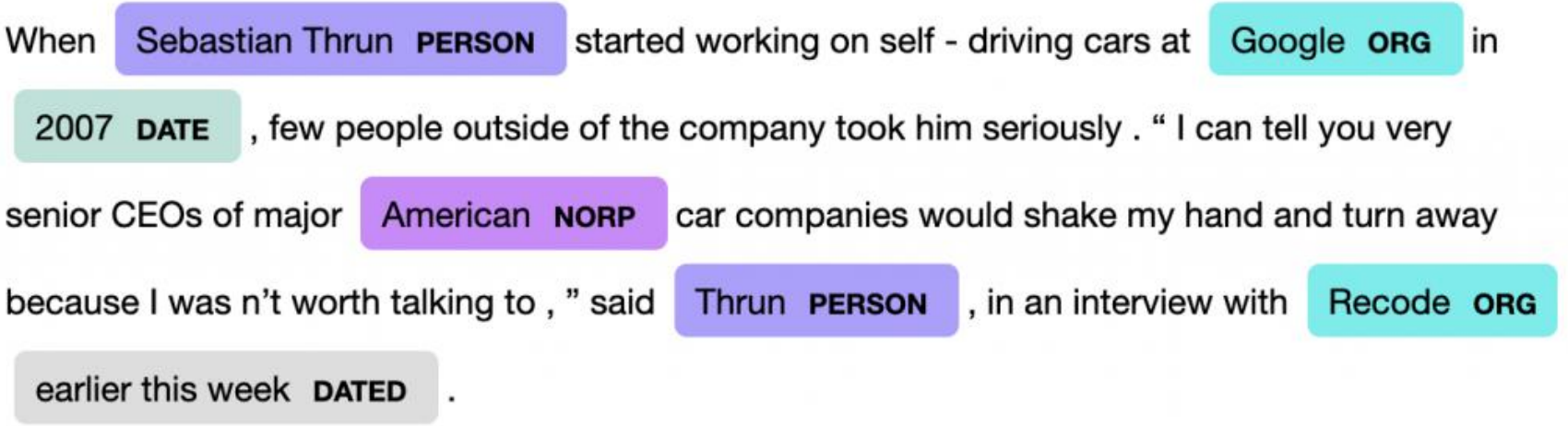
❖ Semantic Role Labeling:

- Semantic Role Labeling goes beyond identifying the grammatical role of words as Part-Of-Speech Tagging but focuses on determining their meaning and the relationships between them.
- It typically involves the following steps :
  a) Identifying the predicate in a sentence
  b) Identifying the arguments of the predicate
  c) Labeling the arguments with their corresponding roles

```
Sentence = "The boy kicked the ball."

# Output :
# (('kicked', 'VBD'), 'nsubj', ('boy', 'NN'))
# (('kicked', 'VBD'), 'dobj', ('ball', 'NN'))
```

Named Entity Recognition (NER)

The first step in information extraction is to detect the entities in the text. A named entity is, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization. NER helps us to identify and extract critical elements from the text.



NER on a paragraph

NER can be used for customer services, medical purposes, document categorization. For example, extracting aspects from customer reviews with NER helps identification of semantics. Or extracting medical diseases and drugs from a text helps identification of treatment.

Sequence Labeling Tasks: Approaches

❖Rule-based approaches :

- Rule based approaches rely on a set of manually-defined rules going from predefined rules to tag each word in a sentence or identifying named entities in text.
- These do the job for simple tasks but can be error-prone and time consuming.

❖Machine learning-based approaches :

- These approaches use machine learning techniques to learn the patterns for the given tasks from annotated training data.
- They range from Stochastic approaches to Deep learning-based approaches such as Transformers.

❖Hybrid approaches :

- These approaches combine the strengths of rule-based and statistical approaches, using a combination of hand-written rules and machine learning techniques to identify arguments and roles.

Rule based POS tagging

- Rule-based part-of-speech (POS) tagging is a method of labeling words with their corresponding parts of speech using a set of pre-defined rules.
- In a rule-based POS tagging system, words are assigned POS tags based on their characteristics and the context in which they appear.
- Example: A rule-based POS tagger might assign the tag "noun" to any word that ends in "-tion" or "-ment," as these suffixes are often used to form nouns.

An Example rule-based POS tagger:

1.Define a set of rules for assigning POS tags to words. For example:

•If the word ends in "-tion," assign the tag "noun."

•If the word ends in "-ment," assign the tag "noun."

•If the word is all uppercase, assign the tag "proper noun."

•If the word is a verb ending in "-ing," assign the tag "verb."

2. Iterate through the words in the text and apply the rules to each word in turn. For example:

•"Nation" would be tagged as "noun" based on the first rule.

•"Investment" would be tagged as "noun" based on the second rule.

•"UNITED" would be tagged as "proper noun" based on the third rule.

•"Running" would be tagged as "verb" based on the fourth rule.

3.Output the POS tags for each word in the text.

Note:

- This is in contrast to machine learning-based POS tagging, which relies on training a model on a large annotated corpus of text
- Rule-based POS taggers can be relatively simple to implement and are often used as a starting point for more complex machine learning-based taggers. However, they can be less accurate and less efficient than machine learning-based taggers, especially for tasks with large or complex datasets.

Rule based Tagging Example

❖ **Rule:** Assign the POS tag “noun” to words ending in “-tion” or “-ment.”

❖ **Text:** “The presentation highlighted the key achievements of the project’s development.”

❖ **Rule based Tags:**

- “The” – Determiner (DET)
- “presentation” – Noun (N)
- “highlighted” – Verb (V)
- “the” – Determiner (DET)
- “key” – Adjective (ADJ)
- “achievements” – Noun (N)
- “of” – Preposition (PREP)
- “the” – Determiner (DET)
- “project’s” – Noun (N)
- “development” – Noun (N)

❖ In this instance, the predetermined rule is followed by the rule-based POS tagger to label words. “Noun” tags are applied to words like “presentation,” “achievements,” and “development” because of the aforementioned restriction.

❖ Despite the simplicity of this example, rule-based taggers may handle a broad variety of linguistic patterns by incorporating different rules, which makes the tagging process transparent and comprehensible.

Rule based taggers: Regex Tagger Example

```
#1. Define Pattern:
patterns = [
    (r'.*ing$', 'VBG'),               # gerunds
    (r'.*ed$', 'VBD'),                # simple past
    (r'.*es$', 'VBZ'),                # 3rd singular present
    (r'.*ould$', 'MD'),               # modals
    (r'.*\'s$', 'NN$'),               # possessive nouns
    (r'.*s$', 'NNS'),                 # plural nouns
    (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),  # cardinal numbers
    (r'the', 'DT'),                   # Determiner
    (r'in','IN'),                     # preposition
    (r'.*', 'NN')                     # nouns (default)
]
```

2. Once a rule-set (=pattern) is defined, it can be applied by, e.g. a RegexpTagger-object from the NLTK package as follows:

```
#2.Generate RegexpTagger
from nltk import RegexpTagger
regexp_tagger = nltk.RegexpTagger(patterns)
#3.Tag a sentence. Note that the string, which contains the sentence must be segmented into words
regexp_tagger.tag("5 friends have been singing in the rain".split())
```

3. The out of the tagger this Regex tagger is:

```
[('5', 'CD'),
 ('friends', 'NNS'),
 ('have', 'NN'),
 ('been', 'NN'),
 ('singing', 'VBG'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('rain', 'NN')]
```

Unigram Tagger

- A unigram is just a single word. A unigram-tagger is probably the simplest data-based tagger.
- As all data-based taggers it requires a labeled training data set (corpus), from which it learns a mapping from a single word to its PoS:

word → PoS(word), ∀ word ∈ V, where V is the applied vocabulary.

Training: For training a Unigram-Tagger a large PoS-tagged corpus is required. Such corpora are publicly available for almost all common languages, e.g. the Brown Corpus for English and the Tiger Corpus for German. A sentence example from the Brown corpus:

[('The', 'DET'), ('Fulton', 'NOUN'), ('County', 'NOUN'), ('Grand', 'ADJ'), ('Jury', 'NOUN'), ('said', 'VERB'), ('Friday', 'NOUN'), ('an', 'DET'), ('investigation', 'NOUN'), ('of', 'ADP'), ("Atlanta's", 'NOUN'), ('recent', 'ADJ'), ('primary', 'NOUN'), ('election', 'NOUN'), ('produced', 'VERB'), ('``', '.'), ('no', 'DET'), ('evidence', 'NOUN'), ("''", '.'), ('that', 'ADP'), ('any', 'DET'), ('irregularities', 'NOUN'), ('took', 'VERB'), ('place', 'NOUN'), ('.', '.')]

- **During training the Unigram-Tagger determines for each word in the corpus which PoS-Tag is associated most often with the word in the training corpus.** The result of the training is a table of two columns, the first column is a word and the second the most-frequent PoS of this word:
- Tagging: **The learned mapping is the two-column table of word and associated most frequent PoS.** This table can be applied to tag each word with its PoS.
- Properties: A unigram tagger is simple to learn and to apply. However, it suffers from the drawback that only the word itself, but not its context is applied to determine the tag. Consequently a word is always tagged with the same PoS, independent of its context. Unigram-Tagging is erroneous whenever a PoS applies to a word, which is not the PoS that appeared most often with this word in the training corpora.

Word	Most Frequent Tag
control	noun
run	verb
love	verb
red	adjective
:	:

Unigram Tagger

Tagging:

- During tagging, for a given word in a sentence, the tagger looks up the most likely tag associated with that word from the training data.
- If the word was seen during training, it assigns the most common tag associated with that word.
- If the word wasn't seen during training, it assigns a default tag (usually NN).

Cons:

- It doesn’t consider the context or POS tags of previous words. Consequently, a word is always tagged with the same POS, independent of its context.

Implementation steps:

- Import necessary modules.
- Load the brown corpus and divide the data into training and testing data
- Train the UnigramTagger on training data
- Tag a sentence using the tag() method of UnigramTagger.

Unigram Tagger

```
import nltk
from nltk.corpus import brown
from nltk.tag import UnigramTagger

# Download the required NLTK resources
nltk.download('brown')
nltk.download('punkt')

# Load the brown dataset for training
train_data = brown.tagged_sents()[:3000]   # Use first 3000 sentences for training
test_data = treebank.tagged_sents()[3000:]   # Use remaining sentences for testing

# Create a UnigramTagger (1-gram model)
unigram_tagger = UnigramTagger(train_data)

# Tag a sentence using the trained UnigramTagger
sentence = "The dog sat on the mat".split()
tagged_sentence = unigram_tagger.tag(sentence)

print(tagged_sentence)

print(unigram_tagger.evaluate(test_data))
```

```
[('The', 'AT'), ('dog', 'NN'), ('sat',
'VBD'), ('on', 'IN'), ('the', 'AT'),
('mat', 'NN')]

 0.7641347348147013
```

N-Gram Tagger

- A **Bigram-Tagger** assigns the **PoS-tag of the current word** by taking into account the **current word** itself **and** the **PoS-tag of the preceding word**.

  (***PoS***($word_{i-1}$), $word_i$)→**PoS**($word_i$),∀ word ∈ V, the Vocabulary

- A **N-Gram-Tagger** assigns the **PoS-tag of the current word** by taking into account the **current word** itself **and** the **PoS-tag of the N-1 preceding words**.

  (***PoS***($word_{i-N+1}$) … ***PoS***($word_{i-1}$), $word_i$)→**PoS**($word_i$),∀ word ∈ V, the Vocabulary

- Training: A Large POS tagged Corpus is required. During training the N-Gram-Tagger determines for each combination of word plus N−1 preceding PoS-Tags in the corpus which PoS-Tag is associated most often with the word.

  The result of the training is a table of N+1 columns, the first N−1 columns contain the PoS-tags of the preceding words, followed by a column with the current word and the column, which contains the most frequent PoS-tag for this combination.

- Tagging: The learned mapping is the (N+1) column table. This table can be applied to tag each PoS-tag-sequence-word-combination with the PoS of the current word.

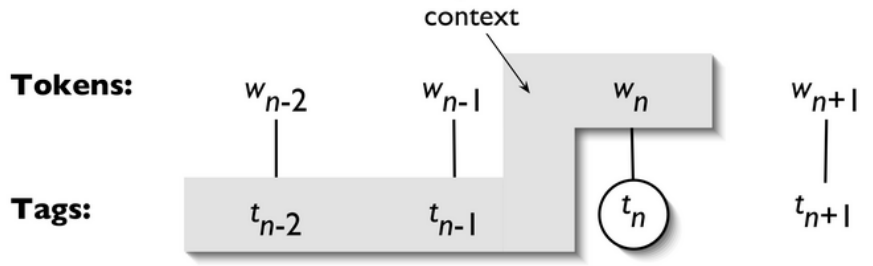


Figure: A 3-Gram-Tagger determines the PoS-Tag of the current word, by taking into the account the current word and the PoS-Tags of 2 preceding words.

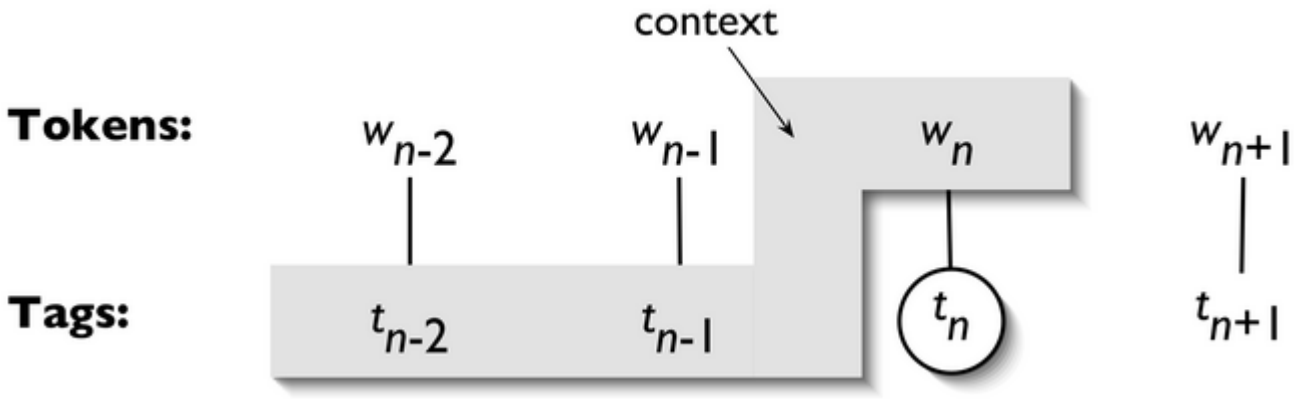
for a Bigram-Tagger (N=2N=2) the table-structure is as follows:

PoS-Tag of previous word	Word	Most Frequent Tag
article	control	noun
pronoun	control	verb
pronoun	run	verb
article	run	noun
pronoun	love	verb
article	love	adjective
:	:	:

- Key Ideas: The larger the value N , the more context is taken into account and the higher the probability, that the correct PoS-Tag is assigned. However, with an increasing value N, also the number of PoS-tag-sequence-word-combinations increases exponentially. Therefore the probability that the text, which must be tagged, contains a combination, which has not been in the training corpus and therefore is not listed in the mapping table increases.
- What should be done in the case of such an unknown combination?
- A standard solution is to train and implement a sequence of N-Gram-Taggers with varying N. For example a Unigram-, Bigram-, 3-Gram and 4-Gram-Tagger is trained. For tagging the 4-Gram tagger is applied.
- If this tagger faces a 4-combination (sequence of 3 PoS-tags plus following word), which is not in its table, a Backup-Tagger, the 3-Gram-Tagger in this case, is applied for this combination. If the corresponding 3-combination is also not in the table of the 3-Gram-Tagger the next Backup-Tagger, which is the Bigram-Tagger, is applied and so on.

N-Gram tagger

- **N-gram tagger** in NLP uses the POS tags of *N-1* previous words to predict the tag for a current word.
- **N-gram-Tagger** assigns the PoS-tag of the current word by taking into account the current word itself and the PoS-tag of the N-1 preceding words.



N-gram Models:

•**Unigram Tagger (1-gram)**: It predicts the tag for the current word based on the word alone.

•**Bigram Tagger (2-gram)**: It predicts the tag for the current word based on the previous word and its tag.

•**Trigram Tagger (3-gram)**: It predicts the tag for the current word based on the previous two words and their tags.

N-gram tagger

How N-gram Taggers Work:

Training:
• The taggers learn patterns from a labeled training dataset (e.g., the brown corpus).

Tagging:
•When a sentence is given to the tagger, it assigns the most probable tag for each word based on the tags of previous *N-1* words.

Steps :
1. Import UnigramTagger, BigramTagger, and TrigramTagger classes.
2. Load the brown Corpus
3. Create Taggers: UnigramTagger,BigramTagger,TrigramTagger.
4. Combine Taggers: A chain of taggers is applied, starting from the unigram tagger and moving to the bigram and trigram taggers. If the higher-order taggers fail to find a tag, the lower-order taggers (unigram) are used as a fallback.
5. Test the Tagger: The sentence is tokenized and tagged.

Advantages:
- ❖ Captures Local Context
- ❖ Improves Predictions
- ❖ Easy to Implement
- ❖ Computational Efficiency
- ❖ Flexibility in N-gram Size

Estimating N-Gram Probabilities

- Estimating n-gram 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.
- Estimating bigram probabilities:

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

where C is the count of that pattern in the corpus

- Estimating N-Gram probabilities:

$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

For example, to compute a particular bigram probability of a word w_n given a previous word w_{n-1}, we'll compute the count of the bigram $C(w_{n-1}w_n)$ and normalize by the sum of all the bigrams that share the same first word w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \quad (3.10)$$

We can simplify this equation, since the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1} (the reader should take a moment to be convinced of this):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad (3.11)$$

The equation estimates the n-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix. This ratio is called a relative frequency.

This use of relative frequencies as a way to estimate probabilities is an example of Maximum Likelihood Estimation or MLE.

```python
import nltk
from nltk.corpus import brown
from nltk.tag import UnigramTagger, BigramTagger, TrigramTagger

nltk.download('brown')
nltk.download('punkt')

train_data = brown.tagged_sents()[:3000]  # First 3000 sentences for training
test_data = brown.tagged_sents()[3000:]   # Remaining sentences for testing

# Create and train a Unigram Tagger
unigram_tagger = UnigramTagger(train_data)

# Create and train a Bigram Tagger
bigram_tagger = BigramTagger(train_data, backoff=unigram_tagger)

# Create and train a Trigram Tagger
trigram_tagger = TrigramTagger(train_data, backoff=bigram_tagger)

# Test the trigram tagger on a new sentence
sentence = "The dog sat on the mat".split()
tagged_sentence = trigram_tagger.tag(sentence)

print(tagged_sentence)

# Print Performnces
print(unigram_tagger.evaluate(test_data))
print(bigram_tagger.evaluate(test_data))
print(trigram_tagger.evaluate(test_data))
```

```
[('The', 'AT'), ('dog', 'NN'), ('sat', 'VBD'),
('on', 'IN'), ('the', 'AT'), ('mat', 'NN')]
```

```
0.7641347348147013
0.7734657846307614
0.772435283624883
```

Transformation-Based Tagging

- A **combination** of rule-based and stochastic tagging methodologies:
  - like the rule-based tagging because rules are used to specify tags in a certain environment;
  - like stochastic tagging, because machine learning is used.
  - uses Transformation-Based Learning (TBL)
- Input:
  - tagged corpus → dictionary (with most frequent tags)
- Basic Process:
  - Set the most probable tag for each word as a start value, e.g. tag all "race" with NN
    P(NN|race) = .98
    P(VB|race) = .02
  - The set of possible transformations is limited
    - by using a fixed number of rule templates, containing slots and
    - allowing a fixed number of fillers to fill the slots
- Example: Brill Tagger

Brill Tagger Approach: 3 stages

1) **Initial Labeling**: Label every word with its most likely tag using a tagged Corpus (e.g., Brown Corpus) without regards to context

   Set the most probable tag for each word as a start value, e.g. tag all "race" with NN

   P(NN|race) = .98

   P(VB|race) = .02

   Example: In both sentences below, run would be tagged as a verb:

   The **run** lasted thirty minutes.

   We **run** three miles every day.

Brown Corpus (POS tagged)
Training data 90%
Patch corpus data 5%
Testing 5%

2) **Rules Learning:**
   a) List all possible tags for each word in the Development/Training data by looking up provided 'english_pos_brown.txt'
   b) Disambiguate tags for each word manually, considering context and lexical properties & make rules (Transformation) in the process
   c) Examine every possible transformation, and select the one that results in the most improved tagging.

3) **Re-tagging**: Re-tag the data according to the selected rule.
   a)Analyze annotation of the Testing data, reformalize the rule-set

4) Go to 2 until stopping criterion is reached.

Stopping: Insufficient improvement over previous pass.

Output: Ordered list of transformations. These constitute a tagging procedure

Brill tagger

- Brill tagger is "**an error-driven transformation based tagger**".
- The idea is to start with assigning a POS tag to each word its most likely tag, estimated by examining a large tagged Corpus without regards to context and, in next iteration, go back and fix the tagging errors using a set of predefined rules that the tagger learned.
- It uses a series of rules to correct the results of an initial tagger. These transformation rules it follows are scored based. This score is equal to the no. of errors they correct minus the no. of new errors they produce. Rules are learnt with an aim to minimize the POS tagging errors (a form of supervised learning).
- In N-gram tagging, we count the N-gram patterns in training data, here we look for transformation rules.
- If the tagger starts with a Unigram / Bigram tagger with an acceptable accuracy, then brill tagger, instead looking for a trigram tuple, will be looking for rules based on tags, position and the word itself.
- Although rule-based taggers are usually subpar to stochastic taggers, the Brill tagger uses statistics as a means of matching the results of stochastic taggers.

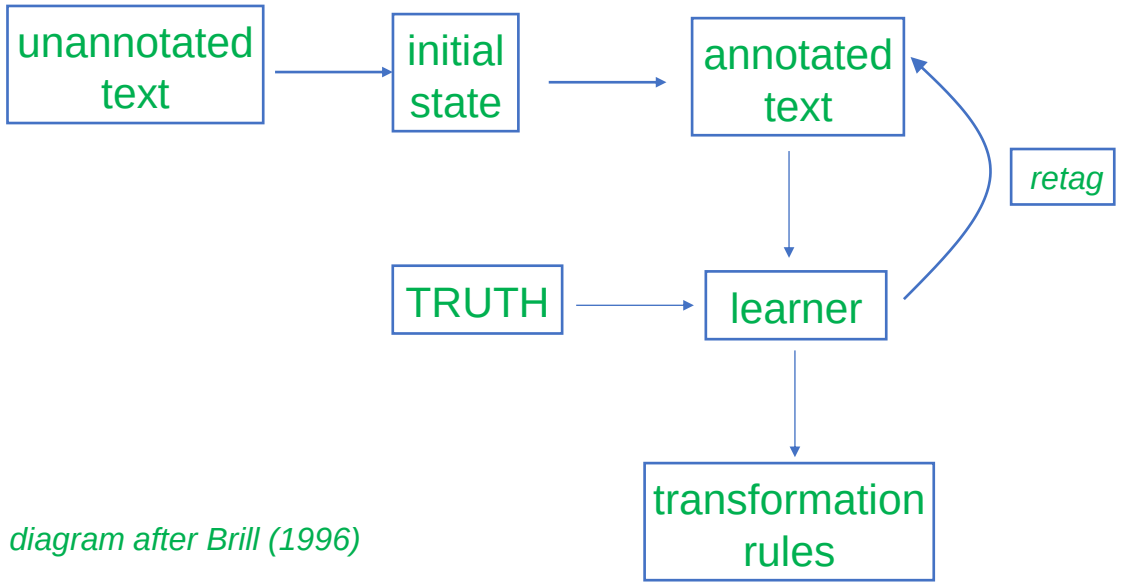
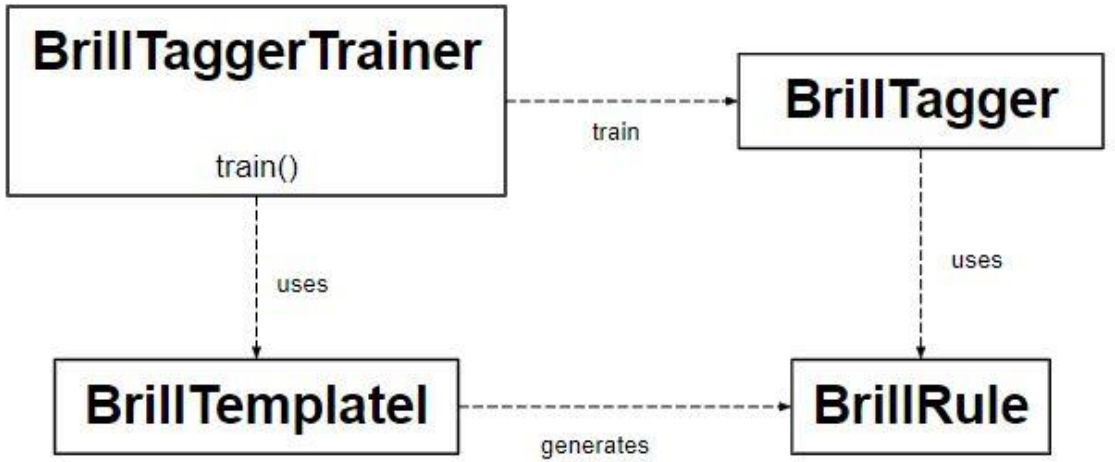


diagram after Brill (1996)



BrillTagger class is a transformation-based tagger.

It is not a subclass of SequentialBackoffTagger.

Brill Tagger Example

❖ **By analyzing sentences with their words mapped to their default tags, we find the following discrepancies:**

- "It doesn't matter": "matter" should be a verb
- "I canceled...": "cancelled" should be a VBD (Verb past tense) and not VBN (Verb past participle)
- NOT_FOUND tags becomes Proper Nouns and we check for 's or s' to assign corresponding $ tags
- "to Smriti Sthal": "to" should be IN (preposition) and not TO
- Some Proper Nouns listed as NN (Noun Singular)

❖ **Disambiguation Rules:**

1. NN → VB if DO* before NN
2. VBN → VBD unless [HV, HVZ, HVD, HVN, HVG] VBN
3. VBD → VBN if [HV, HVZ, HVD, HVN, HVG] VBD
4. NULL → NP, if ends with 's or s' : consider NP$
5. TO → IN unless "[VB(#, D, G, N, P, Z)] to" OR "to [VB(#, D, G, N, P, Z)]"
6. NN with first letter capitalized is NP

❖ **The final tagging results by applying the handcrafted rules to the testing data have been provided.**

Penn Treebank Tagset

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WPS	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	$
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	(],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

AT = article, HVD = had, IN = preposition, MD = modal, NN = sing. noun, NP = proper noun, PPS = 3rd sing. nom. pronoun, PPO = obj. personal pronoun, TO = infinitive to, VB = verb, VBN = past part. verb, VBD = past verb

HV=have, HVN=had (verb past participle)
HVZ=Has, HVG=having

Brill tagger – Example (continued)

❖ **Observations from System Tagged Test Dataset and Newly Proposed Rules:**

1. 'how to assess player value' : "how" should be WRB and not QL
2. Rule 7: if WRB tag is available for a word and the next word is not a JJ or NN, change the tag to WRB
3. 'tests theory by asking' : "tests" should be VBZ and not NNS
4. Rule 8: if VB* tag is available for a word and the next word is a NN*, change tag to VB*
5. 'assistant manager' : 'assistant' should be JJ and not NN
6. Rule 9: Change tag from NN to JJ if JJ is an available tag and the tag of the next word is NN*
7. 'hugely promising' : The problem arised since the Brown Dataset did not have "hugely" as a separate word and hence it got the NULL tag. By default, I assigned NNP tags to NULL values and hence this problem arose.
8. Rule 10: For a word with a NULL tag, check if it ends with "ly" and if it is not the first word in the sentence, check if the first letter is lower case. If the above checks hold, Assign to the word the RB tag for adverbs.

- These are the only 4 discrepancies found. And 4 new rules have been assigned to deal with these.
- Note that these scenarios were not encountered in the the training data and hence could not be taken care of.
- Another issue lies in determining whether the first word in a sentence starting with a capitalized letter is an NNP or not.
- Suppose the word also exists as a NN, then we have to look at the discourse level to analyze whether it is an NNP or an NN. This is precisely the problem of Named Entity Recognition which is beyond the scope of our simple rule based disambiguation scheme.

Brill Tagging: Transformation-Based Error Driven Learning

1. Rule based: Brill tagging has 2 initial procedures.
   a) Any words that are not in the training corpus (collection of words and texts) and are capitalized generally tend to be proper nouns, and should be tagged accordingly.
   b) For the other words not in the training corpus, assign them to the tag most common for such words ending in the same 3 letters.
      Ex. _____ous = adjective — enormous, numerous, tremendous, …
      Ex. _____ion = noun — rejection, objection, inflammation, …
      Ex. _____ly = adjective

      In Brill tagger, there are different sets of POS tagging rules. According to Eric Brill, this simple algorithm itself has an error rate of about 5–8%!

2. Tagging with a Separate Corpus (90% of Brown corpus) and Error Correction using Correctly annotated patch corpus (5% of Brown corpus) :
   a) Next, a patch acquisition procedure in which there is a separate corpus which the tagger trains on as well as an already correctly annotated patch corpus.
   b) A list of tagging errors is compiled by comparing the output of the tagger to the correct tagging of the patch corpus.
   c) The list consists of elements <tag_a, tag_b, number> in which tag_a is mistagged for tag_b for the specified number of times.

3. Rule Learning:
   a) For each error in the list, the Brill tagger computes the error loss for each set of POS tagging rules and selects the most optimal POS tagging rule for each error.
   b) The rules which result in the greatest improvement to the patch corpus is added to the set of POS tagger rules for that specific Brill model.
   c) Once the ultimate set of POS tagger rules has been acquired by training the Brill model, new text can be tested / tagged by applying those optimal rules only.
   d) Note that there is no need to be too cautious of specific rules because naturally only the best rules will statistically show the best results / least error loss. This also makes it easy to experiment with new rules and ascertain results.

Types of Tagging

- **Regex tagging**
- **Affix tagging**
- Stanford tagger
- **Brill Tagger**

Regex tagging

- There is one more class of sequential tagger that is a regular expression based taggers. Here, instead of looking for the exact word, we can define a regular expression, and at the same time we can define the corresponding tag for the given expressions
- Regular expression matching is used to tag words. Consider the example, numbers can be matched with \d to assign the tag CD (which refers to a Cardinal number). Or one can match the known word patterns, such as the suffix “ing”.

```
>>>from nltk.tag.sequential import RegexpTagger
>>>regexp_tagger = RegexpTagger(
[( r'^-?[0-9]+(.[0-9]+)?$', 'CD'), # cardinal numbers
( r'(The|the|A|a|An|an)$', 'AT'), # articles
( r'.*able$', 'JJ'), # adjectives
( r'.*ness$', 'NN'), # nouns formed from adj
( r'.*ly$', 'RB'), # adverbs
( r'.*s$', 'NNS'), # plural nouns
( r'.*ing$', 'VBG'), # gerunds
(r'.*ed$', 'VBD'), # past tense verbs
(r'.*', 'NN') # nouns (default)
])
>>>print regexp_tagger.evaluate(test_data)
0.303627342358
```

Affix tagging

It is a subclass of ContextTagger. In the case of AffixTagger class, the context is either the suffix or the prefix of a word. So, it clearly indicates that this class can learn tags based on fixed-length substrings of the beginning or end of a word.

It specifies the three-character suffixes. That words must be at least 5 characters long and None is returned as the tag if a word is less than five character.

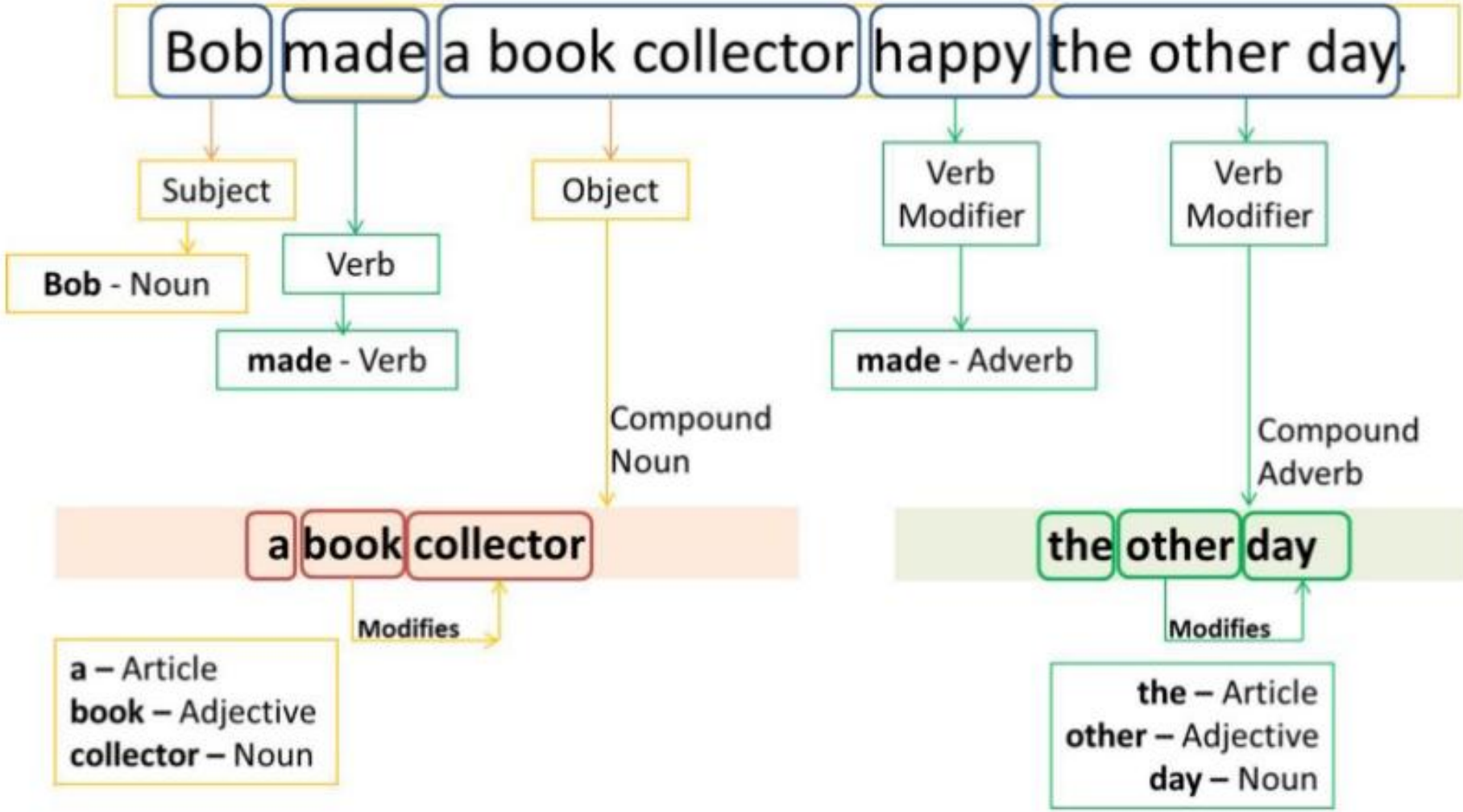
```
# loading libraries
from tag_util import word_tag_model
from nltk.corpus import treebank
from nltk.tag import AffixTagger

# initializing training and testing set
train_data = treebank.tagged_sents()[:3000]
test_data = treebank.tagged_sents()[3000:]

print ("Train data : \n", train_data[1])

# Initializing tagger
tag = AffixTagger(train_data)

# Testing
print ("\nAccuracy : ", tag.evaluate(test_data))
```

Stanford tagger

Another awesome feature of NLTK is that it also has many wrappers around other pre-trained taggers, such as Stanford tools. A Part-Of-Speech Tagger (POS Tagger) is **a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc**., although generally computational applications use more fine-grained POS tags like 'noun-plural'.

A common example of a POS tagger is shown here:

```
>>>from nltk.tag.stanford import POSTagger
>>>import nltk
>>>stan_tagger = POSTagger('models/english-bidirectional-distdim.
tagger','standford-postagger.jar')
>>>tokens = nltk.word_tokenize(s)
>>>stan_tagger.tag(tokens)
```


Bob made a book collector happy the other day.
Subject
Verb
Object
Verb Modifier
Verb Modifier
Bob - Noun
made - Verb
made - Adverb
Compound Noun
Compound Adverb
a book collector
the other day
Modifies
Modifies
a – Article
book – Adjective
collector – Noun
the – Article
other – Adjective
day – Noun

Sample (POS) tags

SYM	Symbol (mathematical or scientific)
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund/present participle

Tag	Description
VBN	Verb, past
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb
#	Pound sign
$	Dollar sign
.	Sentence-final punctuation
,	Comma
:	Colon, semi-colon
(	Left bracket character
)	Right bracket character

"	Straight double quote
'	Left open single quote
"	Left open double quote
'	Right close single quote
"	Right open double quote

Tag	Description
VBG	Verb, gerund/present participle
VBN	Verb, past
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb
#	Pound sign
$	Dollar sign
.	Sentence-final punctuation
,	Comma
:	Colon, semi-colon
(	Left bracket character
)	Right bracket character
"	Straight double quote
'	Left open single quote
"	Left open double quote
'	Right close single quote
"	Right open double quote

```
>>>import nltk
>>>from nltk import word_tokenize
>>>s = "I was watching TV"
>>>print
nltk.pos_tag(word_tokenize(s))
[('I', 'PRP'), ('was', 'VBD'), ('watching',
'VBG'), ('TV', 'NN')]
```

Part-of-speech (POS) tagging Example

Example:

```
>>>import nltk
>>>from nltk import word_tokenize
>>>s = "I was watching TV"
>>>print nltk.pos_tag(word_tokenize(s))
[('I', 'PRP'), ('was', 'VBD'), ('watching', 'VBG'), ('TV', 'NN')]
```

Now this code snippet will give us all the nouns in the given sentence:

```
>>>tagged = nltk.pos_tag(word_tokenize(s))
>>>allnoun = [word for word,pos in tagged if pos in ['NN','NNP'] ]
```

Stanford tagger

Another awesome feature of NLTK is that it also has many wrappers around other pre-trained taggers,such as **Stanford tools.**

A common example of a POS tagger is shown here:

```
>>>from nltk.tag.stanford import POSTagger
>>>import nltk
>>>stan_tagger = POSTagger('models/english-bidirectional-istdim.tagger',
                                                'standford-postagger.jar')
>>>tokens = nltk.word_tokenize(s)
>>>stan_tagger.tag(tokens)
```

The following snippet gives us the frequency distribution of POS tags in the Brown corpus:

```
>>>from nltk.corpus import brown
>>>import nltk
>>>tags = [tag for (word, tag) in
brown.tagged_words(categories='news')]
>>>print nltk.FreqDist(tags)
```

Output:

```
<FreqDist: 'NN': 13162, 'IN': 10616, 'AT': 8893,
 'NP': 6866, ',‘:5133, 'NNS': 5066, '.': 4452, 'JJ': 4392 >
```

DefaultTagger function

- NLTK has a DefaultTagger function.
- DefaultTagger function is part of the Sequence tagger.
- There is a function called evaluate() that gives the accuracy of the correctly predicted POS of the words.
- Thisis used to benchmark the tagger against the brown corpus.
- In the default_tagger case, we aregetting approximately 13 percent of the predictions correct.
- We will use the same benchmark for all thetaggers moving forward.

Example:

```
>>>brown_tagged_sents = brown.tagged_sents(categories='news')
>>>default_tagger = nltk.DefaultTagger('NN')
>>>print default_tagger.evaluate(brown_tagged_sents)
```

Output: 0.130894842572

Sequential tagger

- Not surprisingly, the above Default tagger performed poorly. The DefaultTagger is part of a base class.
- SequentialBackoffTagger that serves tags based on the Sequence. Tagger tries to model the tags based on the context, and if it is not able to predict the tag correctly, it consults a BackoffTagger.
- Typically, the DefaultTagger parameter could be used as a BackoffTagger.

What is Backoff Tagging? It is one of the most important features of **SequentialBackoffTagger** as it allows to combine the taggers together. The advantage of doing this is that if a tagger doesn't know about the tagging of a word, then it can pass this tagging task to the next backoff tagger. If that one can't do it, it can pass the word on to the next backoff tagger, and so on until there are no backoff taggers left to check.

Regex tagger

There is one more class of sequential tagger that is a regular expression based taggers. Here, instead of looking for the exact word, we can define a regular expression, and at the same time we can define the corresponding tag for the given expressions.

```
>>>from nltk.tag.sequential import RegexpTagger
>>>regexp_tagger = RegexpTagger([( r'^-?[0-9]+(.[0-9]+)?$', 'CD'), # cardinal numbers
( r'(The|the|A|a|An|an)$', 'AT'),          # articles
( r'.*able$', 'JJ'),                       # adjectives
( r'.*ness$', 'NN'),                       # nouns formed from adj
( r'.*ly$', 'RB'),                         # adverbs
( r'.*s$', 'NNS'),                          # plural nouns
( r'.*ing$', 'VBG'),                        # gerunds
(r'.*ed$', 'VBD'),                          # past tense verbs
(r'.*', 'NN')                               # nouns (default)
])
>>>print regexp_tagger.evaluate(test_data)
0.303627342358
```

Brill tagger

•Brill tagger is a transformation based tagger, where the idea is to start with a guess for the given tag and,in next iteration, go back and fix the errors based on the next set of rules the tagger learned. It's also a supervised way of tagging, but unlike N-gram tagging where we count the N-gram patterns in training data, we look for transformation rules.

•If the tagger starts with a Unigram / Bigram tagger with an acceptable accuracy, then brill tagger, instead looking for a trigram tuple, will be looking for rules based on tags, position and the word itself.

Machine learning based tagger

- Until now we have just used some of the pre-trained taggers from NLTK or Stanford.
- **pos_tag** internally uses a **Maximum Entropy Classifier (MEC).**
- **StanfordTagger** also uses a modified version of Maximum Entropy.
- There are other models like **Hidden Markov Model (HMM) and Conditional Random Field (CRF)** based taggers, these are generative models.

Named Entity Recognition (NER)

NER constitutes name, location, and organizations. There are NER systems that tag more entities than just three of these. The problem can be seen as a sequence, labeling the Named entities using the context and other features.

NE chunking is loosely used in the same way as Named entity:

```
>>>import nltk
>>>from nltk import ne_chunk
>>>Sent = "Mark is studying at Stanford University in California"
>>>print(ne_chunk(nltk.pos_tag(word_tokenize(sent)), binary=False))
(S
        (PERSON Mark/NNP)
        is/VBZ
        studying/VBG
        at/IN
        (ORGANIZATION Stanford/NNP University/NNP)
        in/IN
        NY(GPE California/NNP))
)
```

The **ne_chunking** method recognizes people (names), places (location), and organizations.

- If binary is set to True then it provides the output for the entire sentence tree and tags everything.
- Setting it to False will give us detailed person, location and organizations information.

References

- https://npogeant.medium.com/sequence-labeling-the-basis-of-nlp-a52c8fffa567
- https://hannibunny.github.io/nlpbook/03postagging/01tagsetsAndAlgorithms.html
- https://www.geeksforgeeks.org/nlp-part-of-speech-default-tagging/
- https://files.ifi.uzh.ch/cl/volk/LexMorphVorl/Brill92.pdf (Eric Brill original paper)
- https://github.com/sayarghoshroy/Brill-Tagger-Illustration
- https://dream-y.medium.com/parts-of-speech-tagging-and-brill-tagging-ee57a157dfe5
- NLP Demystified 5: Basic Bag-of-Words and Measuring Document Similarity
- Lecture 9 : N-Gram Language Models by Pawan Goyal IIT Kharagpur NPTEL
- NLP Demystified 8: Text Classification With Naive Bayes (+ precision and recall)
- https://www.shiksha.com/online-courses/articles/pos-tagging-in-nlp/
- https://github.com/japerk/nltk-trainer
- http://en.wikipedia.org/wiki/Part-of-speech_tagging
- http://en.wikipedia.org/wiki/Named-entity_recognition
- http://www.inf.ed.ac.uk/teaching/courses/icl/nltk/tagging.pdf
- http://www.nltk.org/api/nltk.tag.html
- https://devopedia.org/n-gram-model

Appendix

Parts of Speech (POS) tagging Approaches

1. **rule-based**: involve a large database of hand-written disambiguation rules, e.g. that specify that an ambiguous word is a noun rather than a verb if it follows a determiner.
2. **probabilistic**: resolve tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context.

   - HMM tagger
3. **hybrid corpus-/rule-based**: E.g. transformation-based tagger (Brill tagger); learns symbolic rules based on a corpus.
4. **ensemble methods**: combine the results of multiple taggers.

N-gram tagger

- **N-gram tagger is a subclass of SequentialTagger, where the tagger takes previous *n words in the*** context, to predict the POS tag for the given token.
- There are variations of these taggers where people have tried it with UnigramsTagger, BigramsTagger, and TrigramTagger:

Example:

```
>>>from nltk.tag import UnigramTagger
>>>from nltk.tag import DefaultTagger
>>>from nltk.tag import BigramTagger
>>>from nltk.tag import TrigramTagger
# we are dividing the data into a test and train to evaluate our taggers.
>>>train_data = brown_tagged_sents[:int(len(brown_tagged_sents) *0.9)]
>>>test_data = brown_tagged_sents[int(len(brown_tagged_sents) *0.9):]
```

```
>>>unigram_tagger = UnigramTagger(train_data,backoff=default_tagger)
>>>print unigram_tagger.evaluate(test_data)
0.826195866853
>>>bigram_tagger = BigramTagger(train_data, backoff=unigram_tagger)
>>>print bigram_tagger.evaluate(test_data)
0.835300351655
>>>trigram_tagger = TrigramTagger(train_data,backoff=bigram_tagger)
>>>print trigram_tagger.evaluate(test_data)
0.83327713281
```

- Unigram just considers the conditional frequency of tags and predicts the most frequent tag for the every given token.
- The bigram_tagger parameter will consider the tags of the given word and the previous word, and tag as tuple to get the given tag for the test word.
- The TrigramTagger parameter looks for the previous two words with a similar process.