Module 2
Parts of Speech Tagging

Syllabus

Parts of Speech Tagging and Named Entities –Tagging in NLP, Sequential tagger, N-gram tagger, Regex

tagger,Brill tagger, NER tagger; Machine learning taggers-MEC,HMM,CRF

Introduction to POS Tagging

Parts of Speech (POS) Tagging is a fundamental task in Natural Language Processing (NLP) that involves assigning each word in a text with its corresponding part of speech, such as noun, verb, adjective

- Assume we have
  - A tagset
  - A dictionary that gives you the possible set of tags for each entry
  - A text to be tagged
- Output
  - Single best tag for each word
  - E.g., Book/VB that/DT flight/NN

Important 9 POS Tags

- **Common Parts of Speech Tags**
- **Noun (NN)**: Person, place, or thing (e.g., "dog", "city").
- **Verb (VB)**: Action or state (e.g., "run", "is").
- **Adjective (JJ)**: Describes a noun (e.g., "quick", "blue").
- **Adverb (RB)**: Modifies a verb, adjective, or another adverb (e.g., "quickly", "very").
- **Pronoun (PRP)**: Replaces a noun (e.g., "he", "they").
- **Preposition (IN)**: Shows relationships between a noun (or pronoun) and other words (e.g., "on", "at").
- **Conjunction (CC)**: Connects words, phrases, or clauses (e.g., "and", "but").
- **Determiner (DT)**: Introduces a noun (e.g., "the", "a").
- **Interjection (UH)**: Expresses emotion (e.g., "wow", "ouch").

Some example

I slowly eats many fruits.

My cake should have **sixteen** candles.

She quickly learn NLP

she is brave woman

Jackson always scores above 90% in the exam

I parked my car in the garage

Thor is on the bus.

Whoa, *this city view is amazing!*

Hey! Give me back my Lamborghini

She usually studies in the library **or** at a café

He was clever **but** lazy

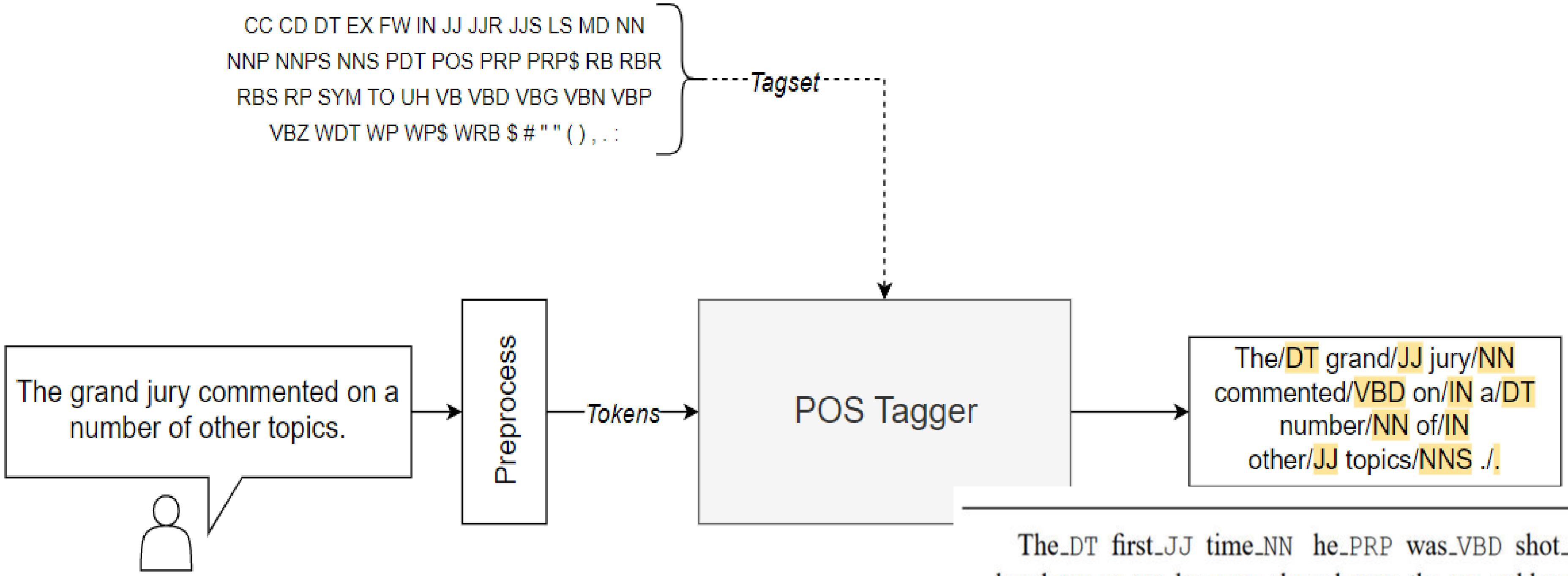
Introduction to POS Tagging

- E.G
  - Plays well with others
  - Plays (NNS/VBZ)
  - well (UH/JJ/NN/RB)
  - with (IN)
  - others (NNS)
  - Plays[VBZ] well[RB] with[IN] others[NNS]
- Let’s take another example,
  - Text: “The cat sat on the mat.”
  - POS tags:
    - The: determiner(DT)
    - cat: noun(NN)
    - sat: verb (VBD)
    - on: preposition(IN)
    - the: determiner(DT)
    - mat: noun (NN)

Some more Examples of POS tags

- Noun: book/books, nature, Germany, Sony
- Verb: eat, wrote
- Auxiliary: can, should, have
- Adjective: new, newer, newest
- Adverb: well, urgently
- Number: 872, two, first
- Article/Determiner: the, some
- Conjuction: and, or, but,else, if
- Pronoun: he, my
- Preposition: to, in
- Particle: off, up
- Interjection: Ow, Eh

POS Tagging architecture



The_DT first_JJ time_NN he_PRP was_VBD shot_VBN in_IN the_DT hand_NN as_IN he_PRP chased_VBD the_DT robbers_NNS outside_RB ._.

first	time	shot	in	hand	as	chased	outside
JJ	NN	NN	IN	NN	IN	JJ	IN
RB	VB	VBD	RB	VB	RB	VBD	JJ
		VBN	RP			VBN	NN
							RB

Use of Parts of Speech Tagging in NLP

- To understand the grammatical structure of a sentence
- To disambiguate words with multiple meanings
- To improve the accuracy of NLP tasks
- To facilitate research in linguistics

How many word classes are there?

- A basic set: – N, V, Adj, Adv, Prep, Det, Aux, Part(iciple), Conj(unction), Num
- A simple division: open/content vs. closed/function
  - – Open: N, V, Adj, Adv
  - – Closed: Prep, Det, Aux, Part, Conj, Num
- Many subclasses,
  - e.g. – eats/V ⇒ eat/VB, eat/VBP, eats/VBZ, ate/VBD, eaten/VBN, eating/VBG, ...
  - Reflect morphological form & syntactic function
  - VB: verb, base form; VBP: verb, non-3person singular present; VBZ: verb 3sg present; VBD: verb, past tense; VBN: verb, past participle

Word classes

- Open classes
  - Nouns, Verbs, Adjectives, Adverbs
- Closed classes
  - Auxiliaries and modal verbs, Prepositions, Conjunctions, Pronouns, Determiners, Particles, Numerals.

Application of POS Tagging

- **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.
- **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.
- **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.
- **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.
- **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.

Challenges in POS Tagging

- **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).
- **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 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 Tag sets

A **POS tagset** is a predefined list of part-of-speech (POS) tags used to label words in a text according to their grammatical category. Each tag in the set represents a specific syntactic category, such as nouns, verbs, adjectives

- Tag types
  - Coarse-grained
    - Noun, verb, adjective, ...
  - Fine-grained
    - noun-proper-singular, noun-proper-plural, nouncommon-mass, ..
    - verb-past, verb-present-3rd, verb-base, ...
    - adjective-simple, adjective-comparative, ..
- Brown tagset (87 tags) – Brown corpus
- C5 tagset (61 tags)
- C7 tagset (146 tags!)
- Penn TreeBank (45 tags) – most used – A large annotated corpus of English tagset

		87-tag Original Brown	45-tag Treebank Brown
Unambiguous (1 tag)		44,019	38,857
Ambiguous (2–7 tags)		5,490	8844
Details:	2 tags	4,967	6,731
	3 tags	411	1621
	4 tags	91	357
	5 tags	17	90
	6 tags	2 (*well, beat*)	32
	7 tags	2 (*still, down*)	6 (*well, set, round, open, fit, down*)
	8 tags		4 (*'s, half, back, a*)
	9 tags		3 (*that, more, in*)

Penn TreeBank tagset Tags

Widely used in English language processing.
Contains 36 POS tags, including

Abbreviation	Meaning
CC	coordinating conjunction(FANBOYS)(for, and, nor, but, or, yet and so)
CD	cardinal digit(one , two , three)
DT	determiner(a , an , the, this, that)
EX	existential there(there)
FW	foreign word
IN	preposition/subordinating conjunction(of, for, in ,by, at)
JJ	This NLTK POS Tag is an adjective (large)
JJR	adjective, comparative (larger)
JJS	adjective, superlative (largest)
LS	list market
MD	modal (could, will, Would)
NN	noun, singular (cat, tree)
NNS	noun plural (desks)
NNP	proper noun, singular (sarah)
NNPS	proper noun, plural (indians or americans)

PDT	predeterminer (all, both, half)
POS	possessive ending (parent\ 's)
PRP	personal pronoun (hers, herself, him, himself)
PRP$	possessive pronoun (her, his, mine, my, our)
RB	adverb (occasionally, swiftly)
RBR	adverb, comparative (greater)
RBS	adverb, superlative (biggest)
RP	particle (about, up, off)
TO	infinite marker (to)
UH	interjection (goodbye)
VB	verb (ask)
VBG	verb gerund (judging)
VBD	verb past tense (pleaded)
VBN	verb past participle (reunified)
VBP	verb, present tense not 3rd person singular(wrap)
VBZ	verb, present tense with 3rd person singular (bases)
WDT	wh-determiner (what)
WP	wh- pronoun (who), who is the CEO of google?
WRB	wh- adverb (how, where) **Where** is my passport? , How are you?

Complete Penn TreeBank tagset Tags

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*	WP$	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 ”
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	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*			

Universal POS Tags

Tag	Meaning	Examples
ADJ	adjective	new, good, high, special, big, local
ADP	adposition	on, of, at, with, by, into, under
ADV	adverb	really, already, still, early, now
CONJ	conjunction	and, or, but, if, while, although
DET	determiner	the, a, some, most, every, no
NOUN	noun	year, home, costs, time, education
NUM	number	twenty-four, fourth, 1991, 14:24
PRON	pronoun	he, their, her, its, my, I, us
PRT	particle	at, on, out, over per, that, up, with
VERB	verb	is, say, told, given, playing, would
.	punctuation marks	. , ; !
X	other	ersatz, esprit, dunno, univeristy

A simplified, language-independent tagset.Contains 17 tags

Simple example of POS tagging in Python

python

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Sample sentence
sentence = "The quick brown fox jumps over the lazy dog."

# Tokenize the sentence
tokens = word_tokenize(sentence)

# Perform POS tagging
pos_tags = pos_tag(tokens)

# Print the tokens with their POS tags
for word, tag in pos_tags:
    print(f"{word}: {tag}")
```

```
The: DT
quick: JJ
brown: JJ
fox: NN
jumps: VBZ
over: IN
the: DT
lazy: JJ
dog: NN
.: .
```

Approaches to POS Tagging

- Rule-based Approach

  – Uses handcrafted sets of rules to tag input sentences

  e.g. RegExp Tagger

- Statistical approaches(Machine Learning Based)

  – Use training corpus to compute probability of a tag to every token in given text.

  e.g N-gram tagger, HMM(Hidden Markov Model), CRF(conditional random field)

- Transformation based(Hybrid)

  Rules + machine learning(1 gram tagger)

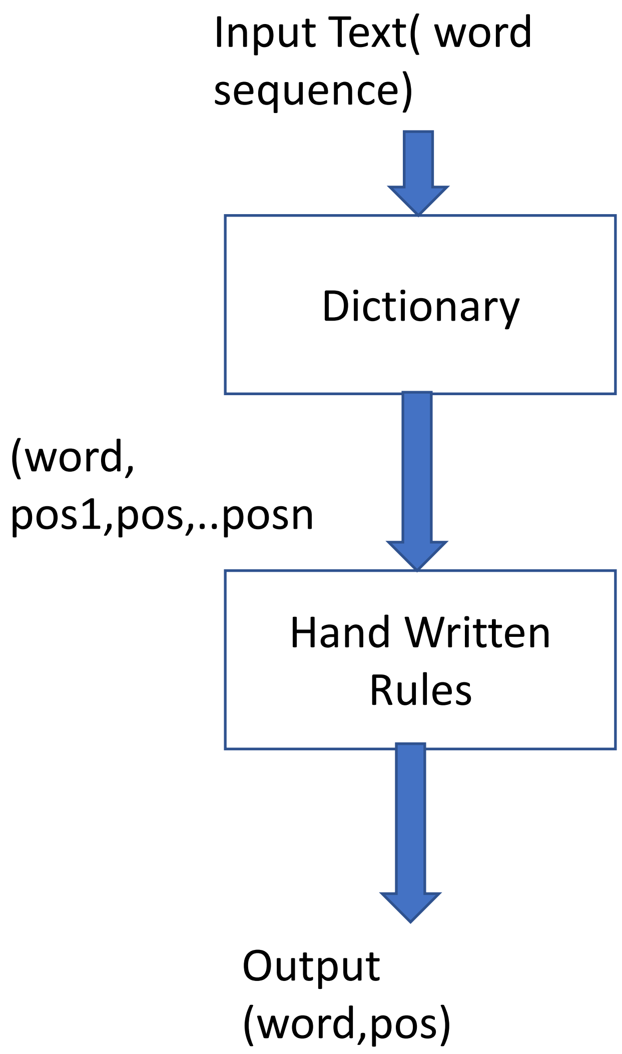
  e.g Brill Tagger

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.
- A Two-stage architecture
  - Use dictionary to tag each word with all possible POS
  - Apply hand-written rules to eliminate ambiguous tags.

    The rules eliminate tags that are inconsistent with the context, and should reduce the list of POS tags to a single POS per word
- For example
  - Assign the tag "noun" to any word that ends in "-tion" or "-ment," as these suffixes are often used to form nouns.
  - If an ambiguous word follows a determiner, tag it as a 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."

Iterate through the words in the text and apply the rules to each word in turn



“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.

Output the POS tags for each word in the text

Adverbial-that Rule

Rule: If a word follows a verb and modifies it, it could be an adverb.

Example:

Sentence: "She runs fast."

Word: "fast" (following the verb "runs")

POS Tag: Adverb (RB)

```
def rule_based_tagger(tokens):
    # Sample lexicon with possible tags
  lexicon = {     "The": ["DT"],
       "cat": ["NN"],
       "sleeps": ["VBZ", "NNS"]
    }
 # Initial tagging based on the lexicon
    tagged_tokens = [(word, lexicon.get(word, ["NN"])[0]) for word in tokens]
    # Applying contextual rules (example)
    for i in range(1, len(tagged_tokens)):
        word, tag = tagged_tokens[i]
        prev_word, prev_tag = tagged_tokens[i - 1]
 # Rule: If previous tag is DT, current word is likely NN
        if prev_tag == "DT":
            tagged_tokens[i] = (word, "NN")

        # Rule: If the word ends with 's' and is ambiguous, tag as VBZ (verb)
        if word.endswith("s") and tag in ["VBZ", "NNS"]:
            tagged_tokens[i] = (word, "VBZ")

    return tagged_tokens
# Example usage
sentence = ["The", "cat", "sleeps"]
print(rule_based_tagger(sentence))
[('The', 'DT'), ('cat', 'NN'), ('sleeps', 'VBZ')]
```

RegExp Tagger

- A **Regular Expression (RegEx) Tagger** is a simple rule-based POS tagger that uses predefined patterns and regular expressions to assign tags to words. It is particularly useful for tagging words based on morphological features like suffixes, prefixes, or specific word forms.
- 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".
- The PoS of a word depends not only on the word itself, e.g. pre- and suffixes, length of the word, etc. but also on surrounding words.
- Therefore, rules on the word itself, e.g. does the word end with ing, and rules on the surrounding words, e.g. is the previous word a determiner (the), must be defined.
- An example of a small set of rules is given below.
- This small set contains rules only on the word itself

```
#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'.*ful$', 'JJ')               # adjective
    (r'.*', 'NN'),                 # nouns (default)
]
```

RegExp Tagger

```
#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())
```

Output

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

Steps Involved in Machine learning POS tagging

- **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.
- **Preprocess the text:** This may include tasks such as tokenization (splitting the text into individual words), lowercasing, and removing punctuation.
- **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.
- **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.
- **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.
- **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.
- **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.

Unigram Tagger

- 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:

$$word \rightarrow PoS(word), \quad \forall word \in V,$$

where V
is the applied vocabulary.

Having two steps

Training and testing

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.

In such corpora each word is associated with its PoS, as can be seen in the following sentence 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'), ('.', '.')]
```

Unigram Tagger

- 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:

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

Unigram tagger Example

```
from nltk import UnigramTagger, DefaultTagger, BigramTagger
from nltk.corpus import brown
from nltk.corpus import treebank

nltk.help.brown_tagset()

brown_tagged_sents=brown.tagged_sents(tagset="universal")
print(brown_tagged_sents[:1])

complete_tagger=UnigramTagger(train=brown_tagged_sents)
mySent1="the cat is on the mat".split()
print(complete_tagger.tag(mySent1))

output
[('the', 'DET'), ('cat', 'NOUN'), ('is', 'VERB'), ('on', 'ADP'), ('the',
'DET'), ('mat', 'NOUN')]
```

```
mySent2="This is major tom calling ground control
from space".split()
print("Unigram Tagger:
\n",complete_tagger.tag(mySent2))
print("\nCurrent Tagger applied for NLTK pos_tag():
\n",nltk.pos_tag(mySent2,tagset='universal'))
```

```
Unigram Tagger:
 [('This', 'DET'), ('is', 'VERB'), ('major', 'ADJ'), ('tom',
None), ('calling', 'VERB'), ('ground', 'NOUN'),
('control', 'NOUN'), ('from', 'ADP'), ('space', 'NOUN')]

Current Tagger applied for NLTK pos_tag():
 [('This', 'DET'), ('is', 'VERB'), ('major', 'ADJ'), ('tom',
'ADJ'), ('calling', 'VERB'), ('ground', 'NOUN'),
('control', 'NOUN'), ('from', 'ADP'), ('space', 'NOUN')]
```

```
print("Performance of complete Tagger:
",complete_tagger.evaluate(brown_tagged_sents))
```

Performance of complete Tagger: 0.9570777270253326

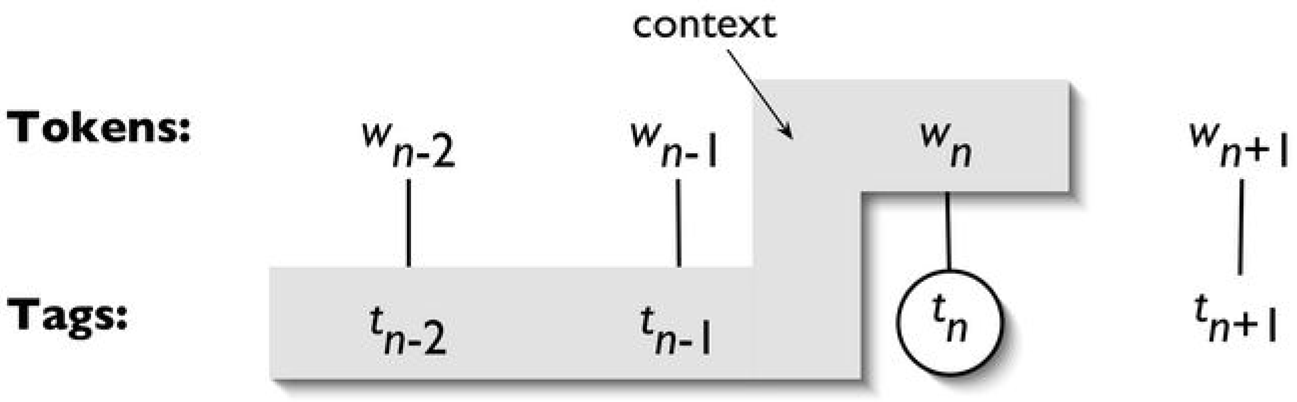
Evaluating the unigram Tagger

```
size = int(len(brown_tagged_sents) * 0.9)
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]
unigram_tagger = UnigramTagger(train_sents,backoff=DefaultTagger("NN"))
print("Performance of Tagger with 90% Training and 10% Testdata:
",unigram_tagger.evaluate(test_sents))
```

```
Output
Performance of Tagger with 90% Training and 10% Testdata:
0.9156346262651662
```

N-Gram Tagger

- Unigram taggers assign to each wort w_n the tag t_n, which is the most frequent tag for w_n in the training corpus.
- N-Gram taggers are a generalization of Unigram-Taggers.
- During training they determine for each combination of N−1 previous tags $t_{n-1}, t_{n-2}, \ldots$ and the current word w_n the most frequent tag t_n.
- Tagging is then realized, by inspecting the n−1 previous tags and the current word w_n and assigning the most frequent tag, which appeared for this combination in the training corpus.



N gram Tagger Example

```
from nltk import UnigramTagger, DefaultTagger, BigramTagger
from nltk import FreqDist,ConditionalFreqDist4
from nltk.corpus import brown
size = int(len(brown_tagged_sents) * 0.9)
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]
baseline=nltk.DefaultTagger('NOUN')
unigram = UnigramTagger(train=train_sents,backoff=baseline)
bigram = BigramTagger(train=train_sents,backoff=unigram)
bigram.evaluate(test_sents)
```

0.9485446658703716

Affix Tagger

- 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.

Example of Affix Tagger

```
# loading libraries
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))
```

```
# Specifying 3 character prefixes
prefix_tag = AffixTagger(train_data,

                         affix_length = 3)

# Testing
accuracy = prefix_tag.evaluate(test_data)

print ("Accuracy : ", accuracy)
```

```
# Specifying 2 character suffixes
sufix_tag = AffixTagger(train_data, affix_length = -2)
# Testing
accuracy = sufix_tag.evaluate(test_data)
print ("Accuracy : ", accuracy)
```

Output :

```
Train data :
[('Mr.', 'NNP'), ('Vinken', 'NNP'), ('is', 'VBZ'), ('chairman', 'NN'),
('of', 'IN'), ('Elsevier', 'NNP'), ('N.V.', 'NNP'), (', ', ', '), ('the',
'DT'),
('Dutch', 'NNP'), ('publishing', 'VBG'), ('group', 'NN'), ('.', '.')]

Accuracy : 0.27558817181092166
```

Brill Tagger

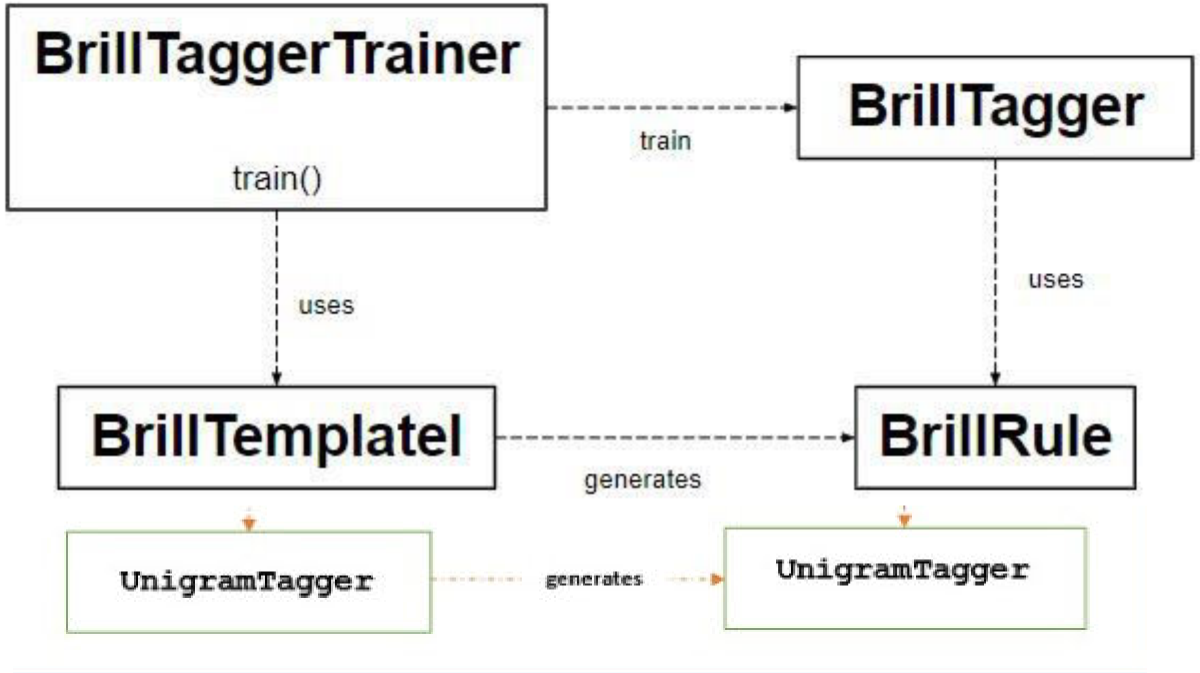
- Brill tagger is a transformation based tagger, where the idea is to start with a guess for the given tag and, in next iteration, go back and fix the errors based on the next set of rules the tagger learned. It's also a supervised way of tagging, but unlike N-gram tagging where we count the N-gram patterns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

Replace NN with VB when the previous word is TO.

- After we already have some tags based on UnigramTagger, we can refine if with just one simple rule. This is an interactive process. With a few iterations and some more optimized rules, the brill tagger can outperform some of the N-gram taggers. The only piece of advice is to look out for over-fitting of the tagger for the training set.

Brill Tagger

- **BrillTagger class** is a **transformation-based tagger**. It is not a subclass of SequentialBackoffTagger.
- Moreover, it uses a series of rules to correct the results of an initial tagger.
- These 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.



Example

```
# Loading Libraries
from nltk.tag import brill, brill_trainer

def train_brill_tagger(initial_tagger, train_sents, **kwargs):
    templates = [
            brill.Template(brill.Pos([-1])),
            brill.Template(brill.Pos([1])),
            brill.Template(brill.Pos([-2])),
            brill.Template(brill.Pos([2])),
            brill.Template(brill.Pos([-2, -1])),
            brill.Template(brill.Pos([1, 2])),
            brill.Template(brill.Pos([-3, -2, -1])),
            brill.Template(brill.Pos([1, 2, 3])),
            brill.Template(brill.Pos([-1]), brill.Pos([1])),
            brill.Template(brill.Word([-1])),
            brill.Template(brill.Word([1])),
            brill.Template(brill.Word([-2])),
            brill.Template(brill.Word([2])),
            brill.Template(brill.Word([-2, -1])),
            brill.Template(brill.Word([1, 2])),
            brill.Template(brill.Word([-3, -2, -1])),
            brill.Template(brill.Word([1, 2, 3])),
            brill.Template(brill.Word([-1]), brill.Word([1])),
            ]

    # Using BrillTaggerTrainer to train
    trainer = brill_trainer.BrillTaggerTrainer(
            initial_tagger, templates, deterministic = True)

    return trainer.train(train_sents, **kwargs)
```

As we can see, this function requires **initial_tagger** and **train_sentences**. It takes an **initial_tagger** argument and a list of templates, which implements the **BrillTemplate** interface. The **BrillTemplate** interface is found in the **nltk.tbl.template** module. One of such implementation is **brill.Template** class.

Training of Initial Tagger and utilizing in brill tagger

```
from nltk.tag import brill, brill_trainer
from nltk.tag import DefaultTagger
from nltk.corpus import treebank
from tag_util import train_brill_tagger

# Initializing
default_tag = DefaultTagger('NN')

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

initial_tag = UnigramTagger(train_data, backoff = default_tagger)

a = initial_tag.evaluate(test_data)
print ("Accuracy of Initial Tag : ", a)
```

```
brill_tag = train_brill_tagger(initial_tag,
train_data)
b = brill_tag.evaluate(test_data)

print ("Accuracy of brill_tag : ", b)
```

```
Accuracy of Initial Tag : 0.8806820634578028
```

```
Accuracy of brill_tag : 0.8827541549751781
```

Sequential Tagging

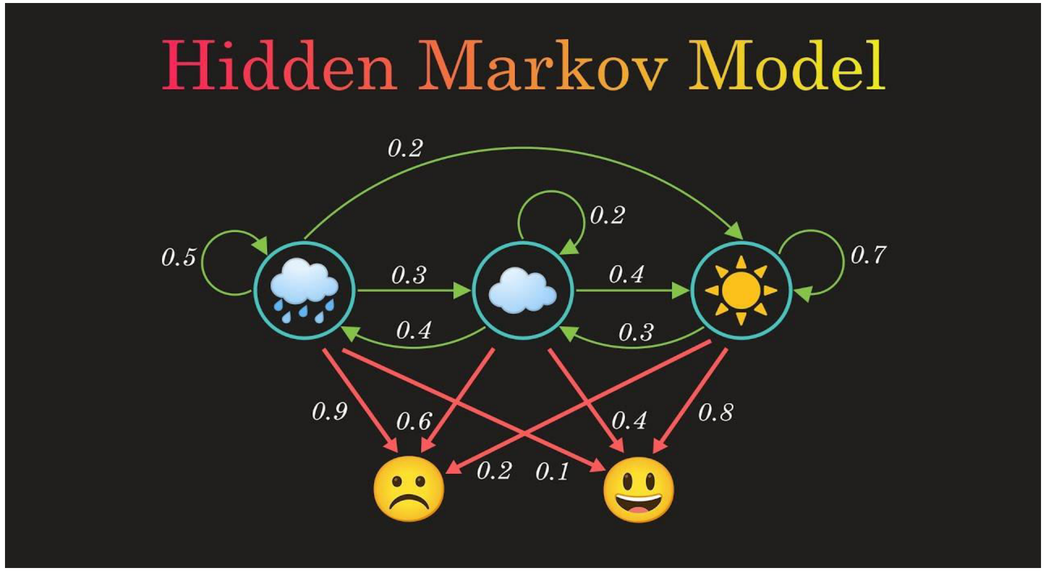
- Classes for tagging sentences sequentially, left to right.
- The abstract base class SequentialBackoffTagger serves as the base class for all the taggers in this module.
- Tagging of individual words is performed by the method choose_tag(), which is defined by subclasses of SequentialBackoffTagger.
- If a tagger is unable to determine a tag for the specified token, then its backoff tagger is consulted instead.
- Any SequentialBackoffTagger may serve as a backoff tagger for any other SequentialBackoffTagger.
- nltk.tag.sequential
  - AffixTagger
  - NgramTagger
  - RegexpTagger

HMM(Hidden Markov Model)Tagger

- HMM (Hidden Markov Model) is a Stochastic technique for POS tagging.
- Markov Chain
  - A Markov chain is a model that tells us something about the probabilities of sequences of random states/variables.
  - A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state.
  - All the states before the current state have no impact on the future except via the current state

Hidden Markov Model

- want to predict is a sequence of states that aren't directly observable in the environment.
- Though we are given another sequence of states that are observable in the environment
- these hidden states have some dependence on the observable states.



- In the above HMM, we are given sad and smile as observable states.
- But we are more interested in tracing the sequence of the hidden states that will be followed which are Rainy & Sunny, and cloudy.

HMM tagger

Q: Set of possible Tags (hidden states)

A: The A matrix contains the tag transition probabilities P(ti|ti−1) which represent the probability of a tag occurring given the previous tag. Example: Calculating A[Verb][Noun]:

P (Noun|Verb): Count(Verb&Noun)/Count(Verb)

O: Sequence of observation (words in the sentence)

B: The B emission probabilities, P(wi|ti), represent the probability, given a tag (say Verb), that it will be associated with a given word (say Playing). The emission probability B[Verb][Playing] is calculated using:

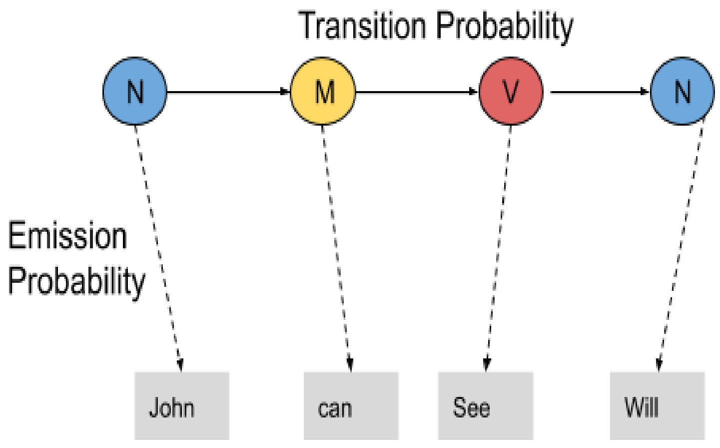
P(Playing | Verb): Count (Playing & Verb)/ Count (Verb)

It must be noted that we get all these Count() from the corpus itself used for training.

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \;\; \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_i
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Transition and Emission Probabilities

- “I love Artificial Intelligence” and we need to assign POS tags to each word.
- It is clear that the POS tags for each word are “Pronoun (PRP), Verb (VBP), Adjective (JJ), Noun (NN) respectively.
- To calculate the probabilities associated with the tags
    - we need to first know how likely it is for a pronoun to be followed by a verb, then an adjective, and finally a noun. These probabilities are typically called **transitions probabilities**.
    - Secondly, we need to know how likely that the word ‘’I” would be a pronoun, the word “love” would be a verb, the word “Artificial” would be an adjective, and the word “Intelligence” would be a noun. These probabilities are called **emission probabilities**.
- The transition probability is the probability that connects the change from one state to the next in the system.
- The emission probability is the probability that quantifies the possibility of making a particular observation given a defined state.

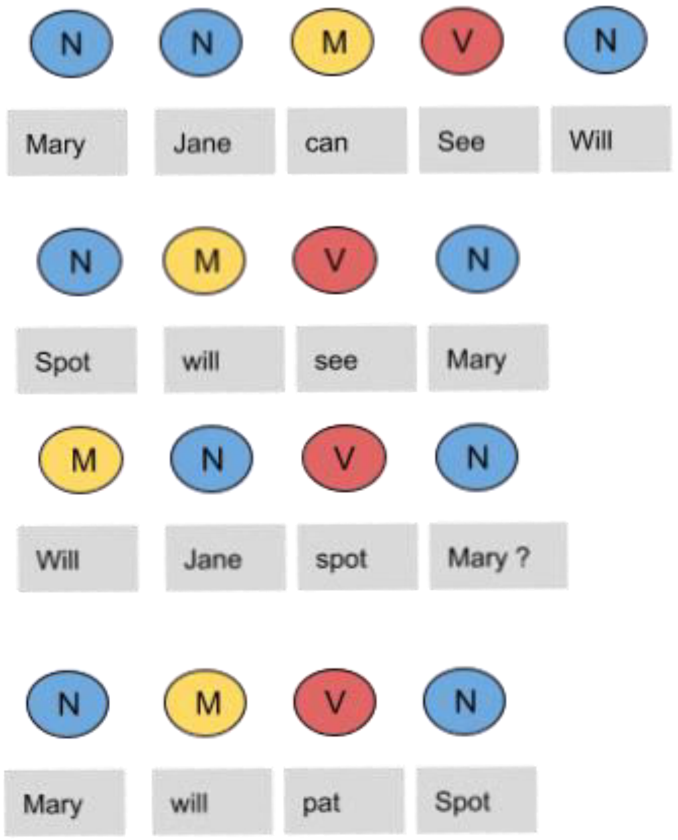


Example

Let us calculate the above two probabilities for the set of sentences below

Mary Jane can see Will
Spot will see Mary
Will Jane spot Mary?
Mary will pat Spot

Note that Mary Jane, Spot, and Will are all names.



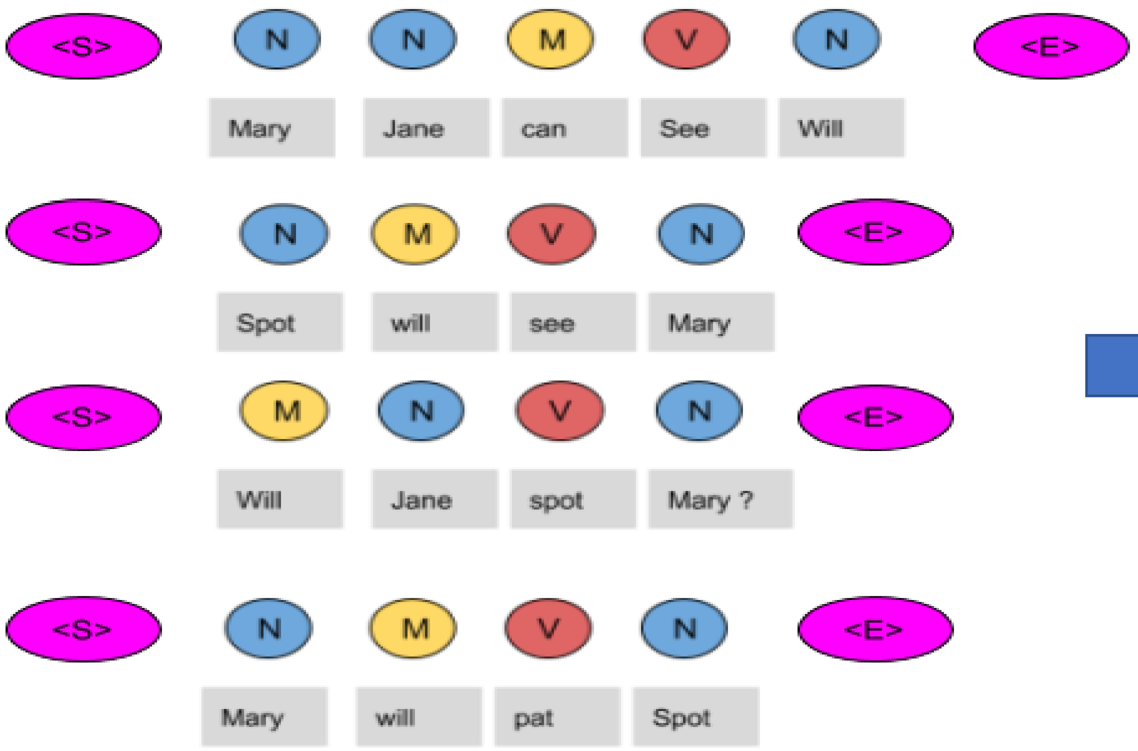
To calculate the emission probabilities, let us create a counting table.

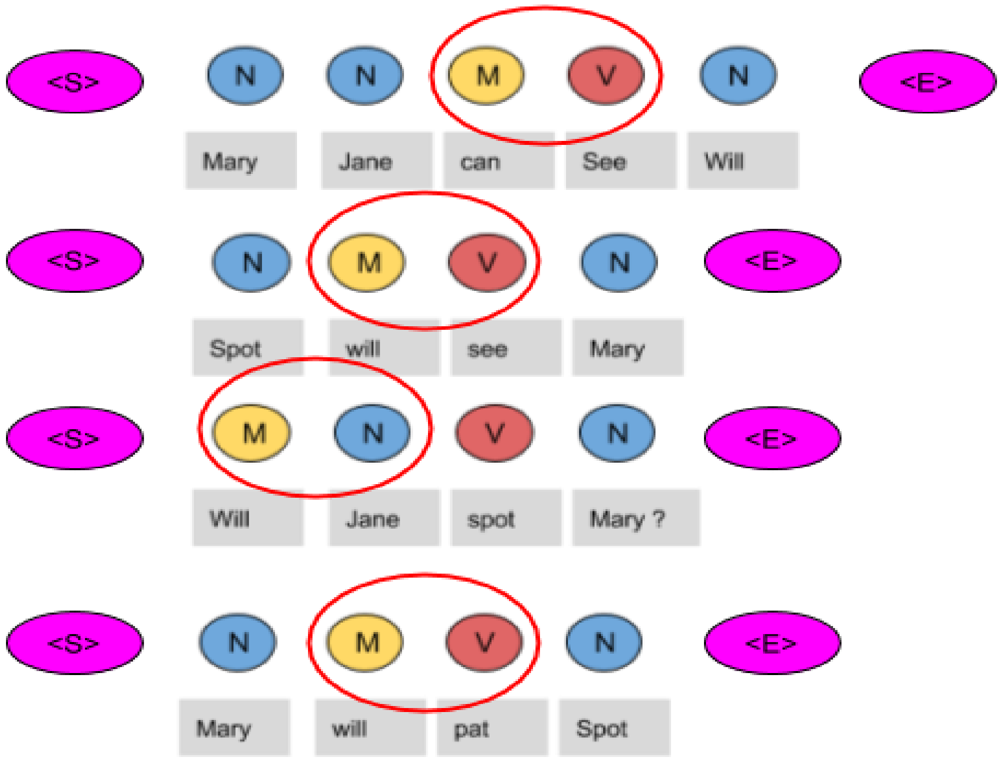
Words	Noun	Model	Verb
Mary	4	0	0
Jane	2	0	0
Will	1	3	0
Spot	2	0	1
Can	0	1	0
See	0	0	2
pat	0	0	1

emission probabilities

Words	Noun	Model	Verb
Mary	4/9	0	0
Jane	2/9	0	0
Will	1/9	3/4	0
Spot	2/9	0	1/4
Can	0	1/4	0
See	0	0	2/4
pat	0	0	1

Transition Probabilities





	N	M	V	<E>
<S>	3	1	0	0
N	1	3	1	4
M	1	0	3	0
V	4	0	0	0

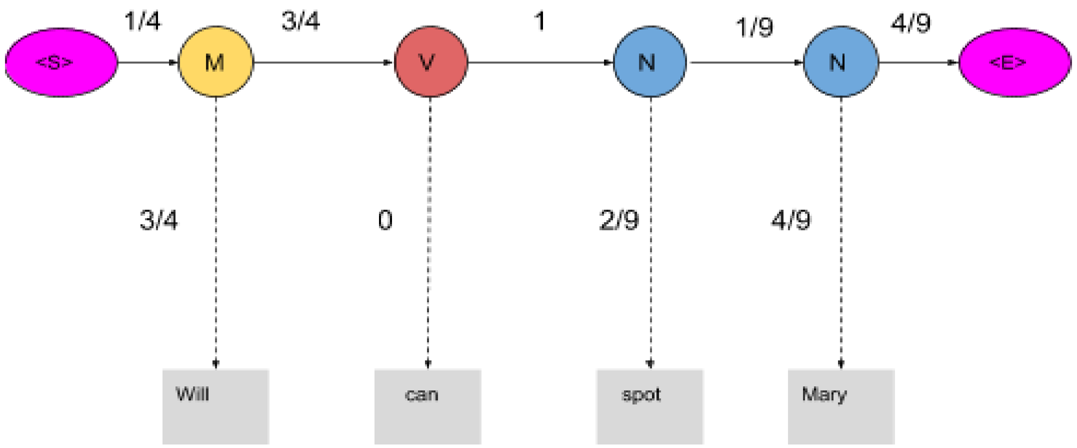
	N	M	V	<E>
<S>	3/4	1/4	0	0
N	1/9	3/9	1/9	4/9
M	1/4	0	3/4	0
V	4/4	0	0	0

Calculating tag sequence probabilities

Take a new sentence and tag them with wrong tags.
Let the sentence, ‘ **Will can spot Mary**’ be tagged as-

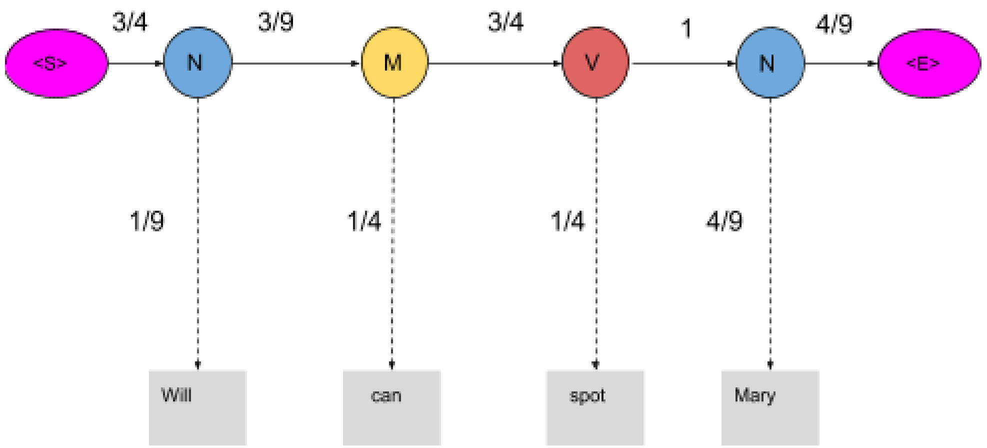
Will as a model
Can as a verb
Spot as a noun
Mary as a noun

Now calculate the probability of this sequence being correct in the following manner.



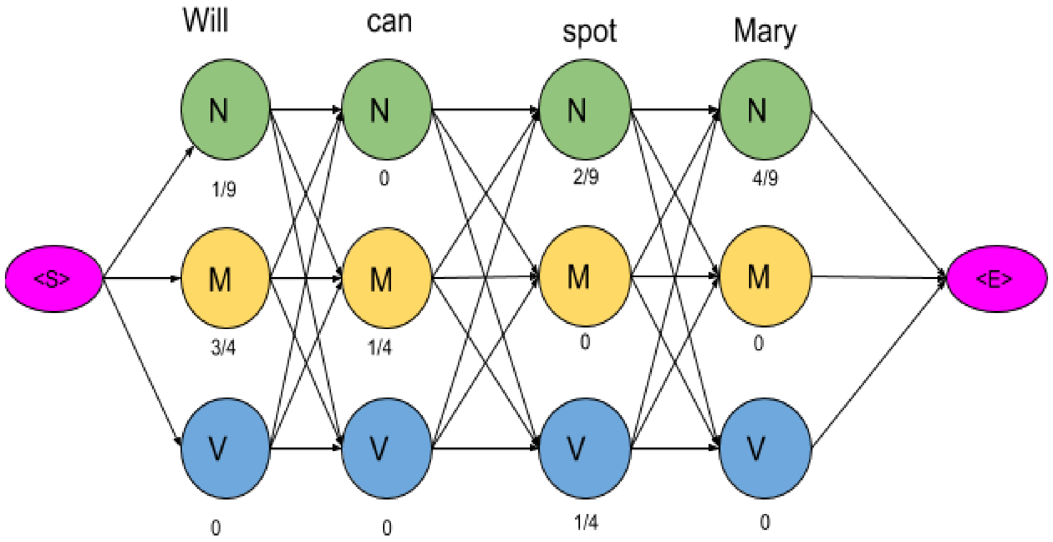
1/4*3/4*3/4*0*1*2/9*1/9*4/9*4/9=0

Calculating tag sequence probabilities



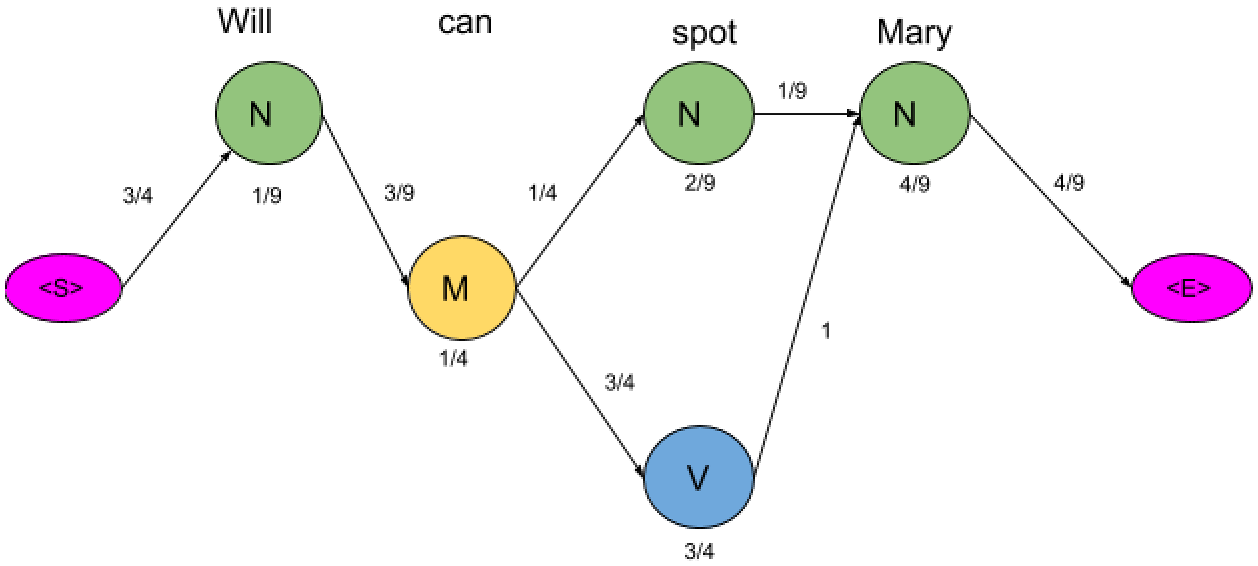
3/4*1/9*3/9*1/4*3/4*1/4*1*4/9*4/9=0.00025720164

All possible combinations with three tags(N, M and V)



Removal of unwanted links and nodes

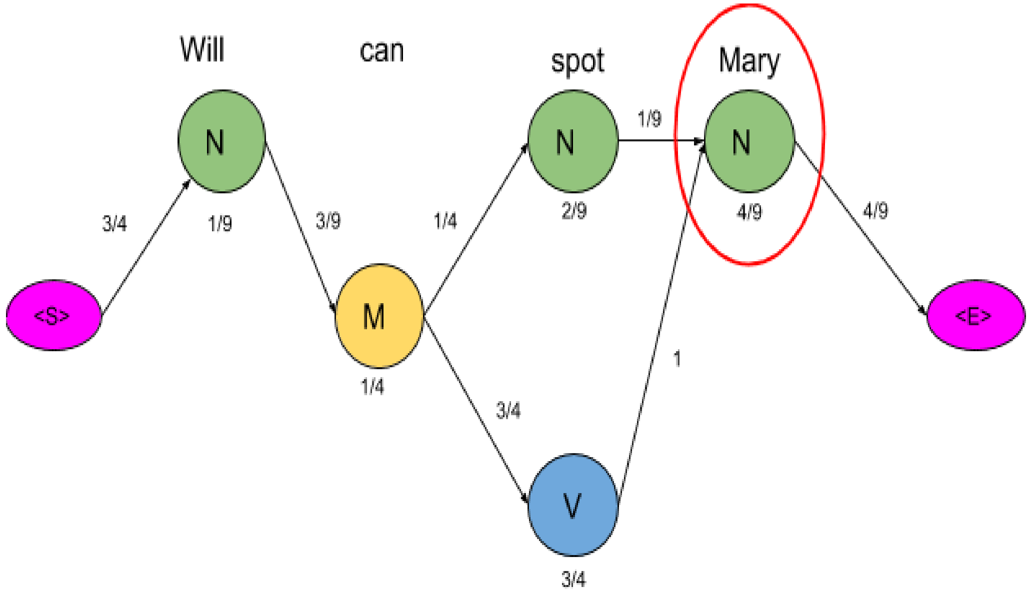
The next step is to delete all the vertices and edges with probability zero, also the vertices which do not lead to the endpoint are removed.

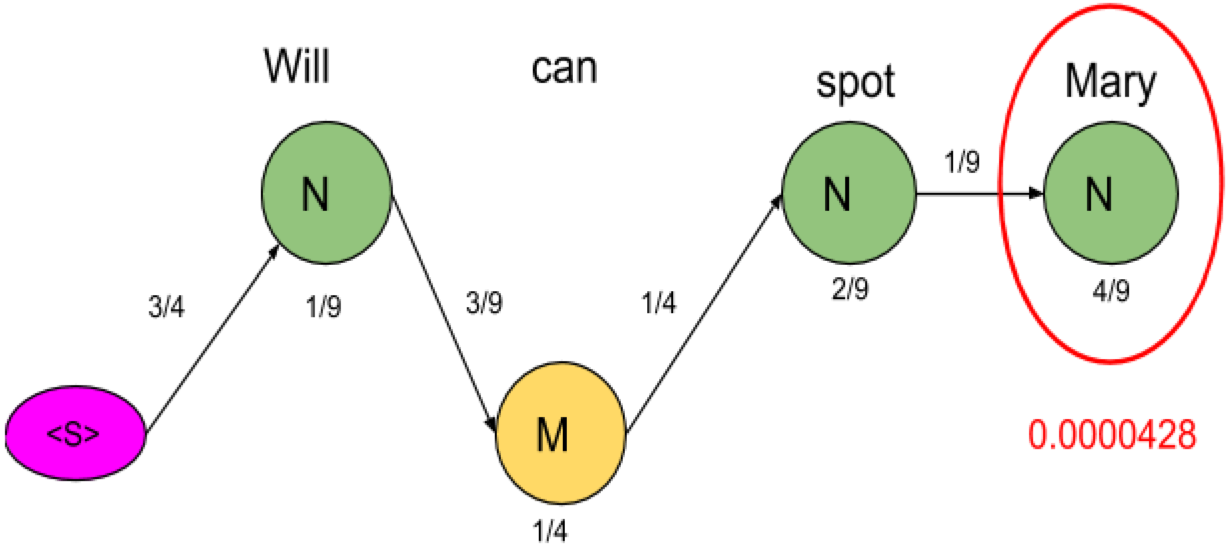


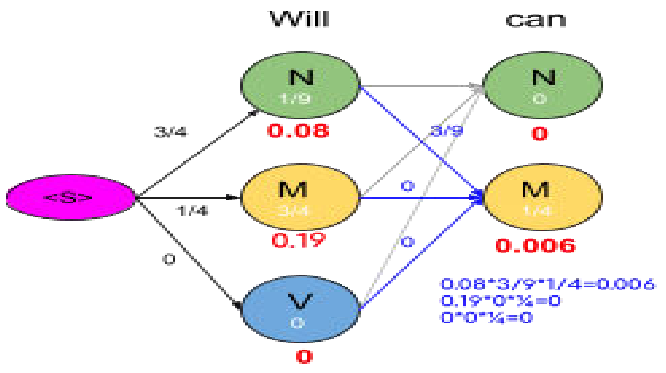
Clearly, the probability of the second sequence is much higher and hence the HMM is going to tag each word in the sentence according to this sequence.

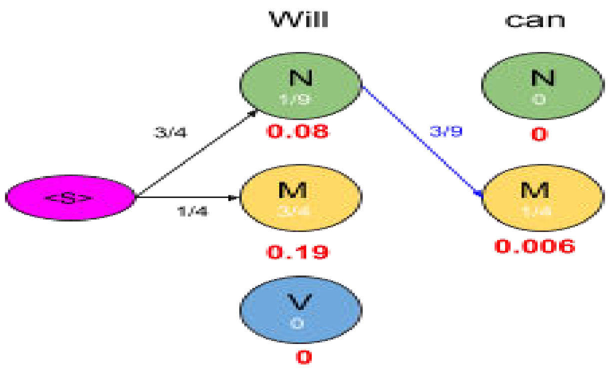
<S>→N→M→N→N→<E> =**3/4*1/9*3/9*1/4*1/4*2/9*1/9*4/9*4/9=0.00000846754**
<S>→N→M→N→V→<E>=**3/4*1/9*3/9*1/4*3/4*1/4*1*4/9*4/9=0.00025720164**

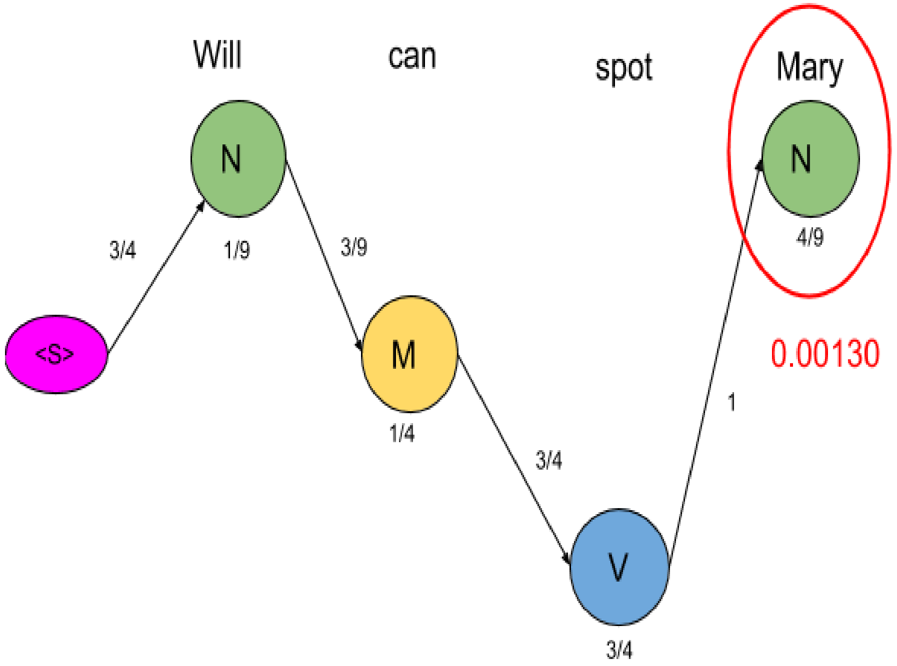
Optimizing HMM with Viterbi Algorithm



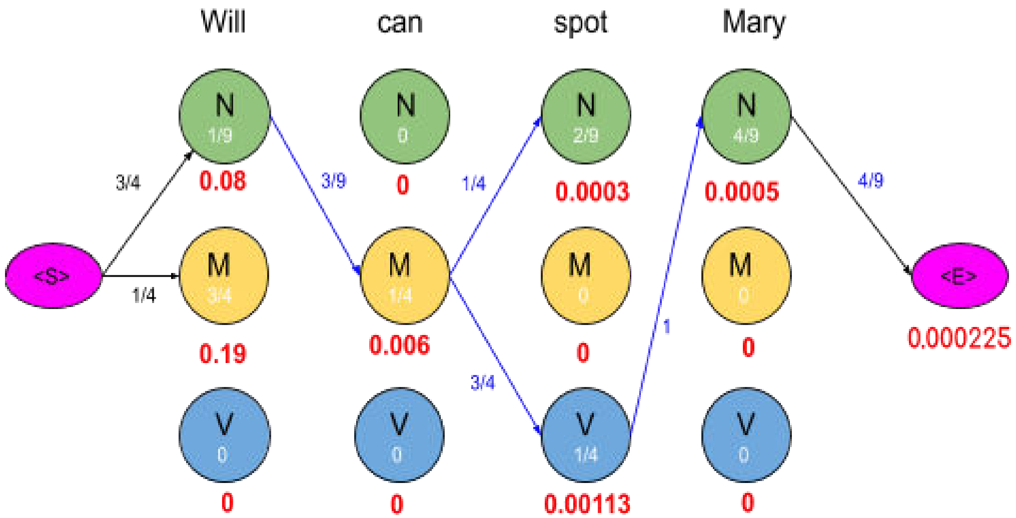




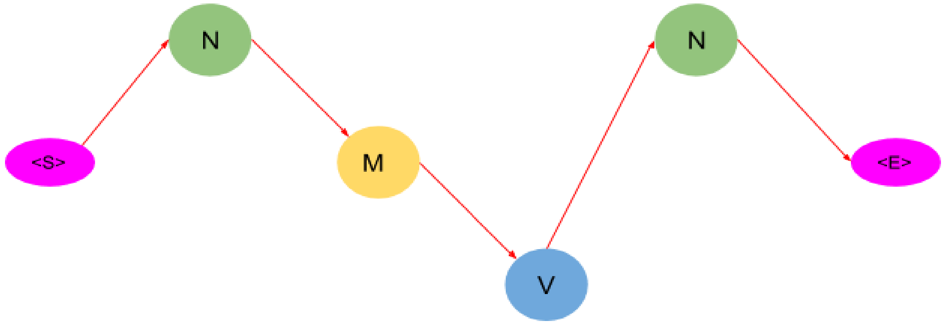




- the probabilities of all paths leading to a node are calculated and we remove the edges or path which has lower probability cost.
- Also, you may notice some nodes having the probability of zero and such nodes have no edges attached to them as all the paths are having zero probability.
- The graph obtained after computing probabilities of all paths leading to a node is shown below:

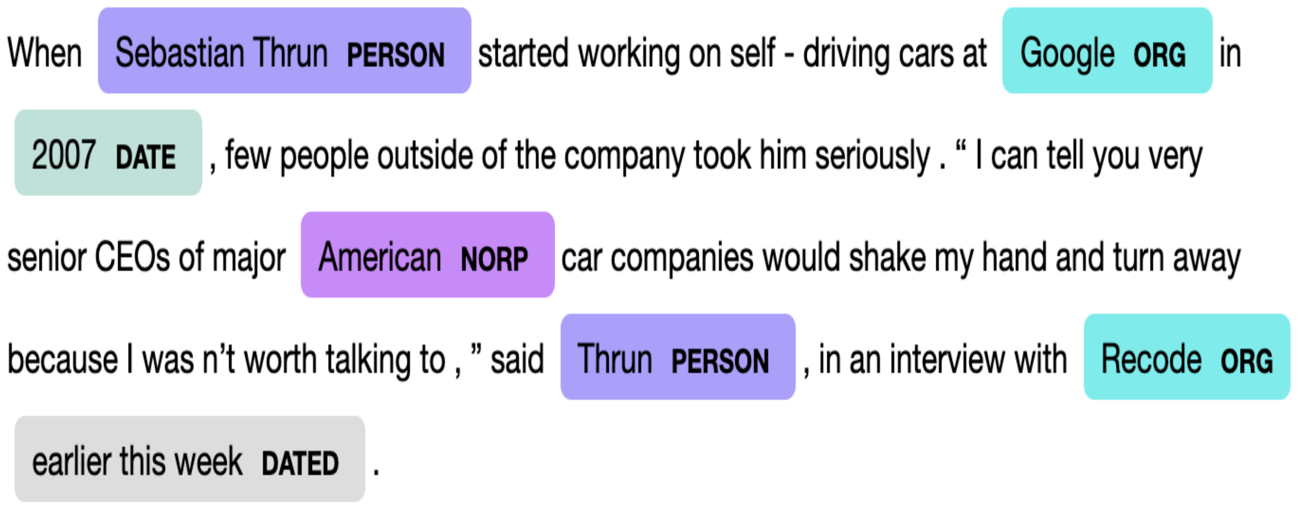


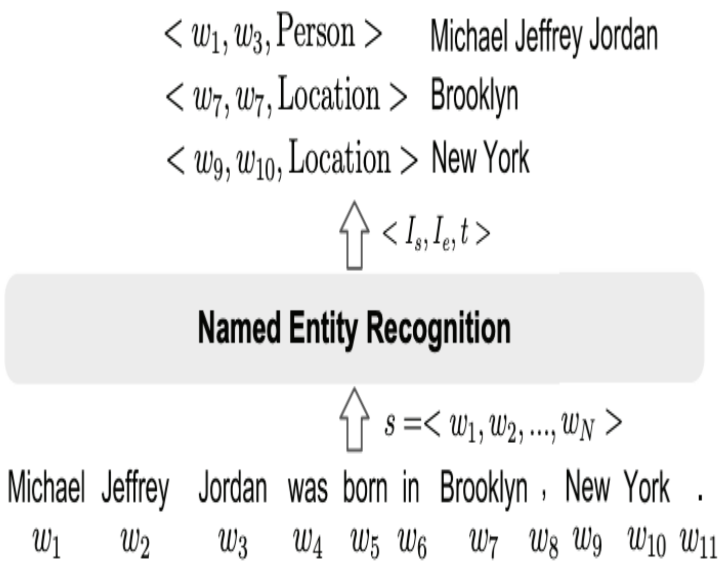
- To get an optimal path, we start from the end and trace backward, since each state has only one incoming edge,
- This gives us a path as shown below



Named Entity Recognition

- Named entity recognition (NER) is an NLP based technique to identify mentions of rigid designators from text belonging to particular semantic types such as a person, location, organisation etc.
- Named entities are proper nouns that refer to specific entities that can be a person, organization, location, date, etc. Consider this example – “Mount Everest is the tallest mountain”. Here Mount Everest is a named entity of type location as it refers to a specific entity.
- Below is an screenshot of how a NER algorithm can highlight and extract particular entities from a given text document:





examples of named entities

Sl. No	Named Entity	Examples
1	ORGANIZATION	SEI, BCCI, Pakistan Cricket Board
2	PERSON	Barack Obama, Narendra Modi, Kohli
3	MONEY	7 million dollars, INR 7 Crore
4	GPE	India, Australia, South East Asia
5	LOCATION	Mount Everest, River Nile
6	DATE	8th June 1998, 7 April
7	TIME	8:45 A.M., two-fifty am

NER Example in NLTK

```
# Step One: Import nltk and download necessary packages

import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

# Step Two: Load Data

sentence = "WASHINGTON -- In the wake of a string of abuses by New York police officers in the 1990s, Loretta E. Lynch, the top federal prosecutor in Brooklyn, spoke forcefully about the pain of a broken trust that African-Americans felt and said the responsibility for repairing generations of miscommunication and mistrust fell to law enforcement."

# Step Three: Tokenise, find parts of speech and chunk words

for sent in nltk.sent_tokenize(sentence):
  for chunk in nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sent))):
     if hasattr(chunk, 'label'):
        print(chunk.label(), ' '.join(c[0] for c in chunk))
```

```
GPE WASHINGTON GPE New York PERSON Loretta E.
Lynch GPE Brooklyn
```

NER with Spacy

Install: python -m spacy download en_core_web_sm

```
# import spacy
import spacy

# load spacy model
nlp = spacy.load('en_core_web_sm')

# load data
sentence = "Apple is looking at buying U.K. startup for $1 billion"
doc = nlp(sentence)

# print entities
for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
Apple 0 5 ORG
U.K. 27 31 GPE
$1 billion 44 54 MONEY
```

Medium and large models

```
nlp = spacy.load('en_core_web_md')
nlp = spacy.load('en_core_web_lg')
```

MEC(Maximum Entropy Classifier)

- The MaxEnt is based on the principle of Maximum Entropy, which states that of all models (features) that match training data, the one with the highest entropy should be chosen
- When the conditional independence of the features cannot be assumed, the maximum entropy classifier is utilized.
- Features are interdependent.
- Machine learning framework
- Taking single observation, extracting some useful features describing the observation, and then classify the observation into one of the set of discrete classes depending on these features.
- Probabilistic classifier: likelihood that the observation belongs to that category.
- Classification is the problem of taking a single observation, extracting some useful features describing he observation, and classify the observation into one of a set of discrete classes depending on these features.

Entropy formula

- P(c|x) = $\frac{exp(\sum_{i=1}^{N} f_i(x) * w_i(d))}{\sum_d exp(\sum_{i=1}^{N} f_i(x) * w_i(d))}$

- P(c|x) = $\frac{1}{Z}\exp(\sum_i w_i f_i)$

    - C is POS tag and x is observed word

    - Z represent normalizing constant

Example

e.g. The complex record

The	Complex	record
Det.	Verb	Verb
Noun	Adj.	Noun

POS Tags	Ti-1	Ti	
Word	Wi-1	Wi	Wi+1

Assumptions: all features having same weight as 1
Beam Size = 2

Set of features learned from training

Features are taken as below. Observed word(W_i). Its previous word (W_{i-1}) and POS Tag (T_i and T_{i-1}). **Stentence is: "The complex record"**

Feature	The (Det.)	The (Noun)	Complex (Verb)	Complex (Adj.)	Record (Verb)	Record (Noun)
F1:T_{i-1}=Det and T_i=Adj				1		
F2:T_{i-1}=Noun and T_i=Verb			1			
F3:T_{i-1}=Adj. And T_i=Noun						1
F4: W_{i-1}=the and T_i=Adj				1		
F5:W_{i-1}=the & W_{i+1}=record and T_i=Adj				1		
F6=:W_{i-1}=complex and T_i=Noun						1
F7:W_{i+1}=complex and T_i=Det	1					
F8:W_{i-1}=NULL and T_i=Noun		1				

The Word

The (Det)   F7 feature is present

The(Noun) F8 feature is present

$$p(\mathrm{Det}|\mathrm{The}) = \frac{\exp(1*1)}{\exp(1*1)+\exp(1*1)} = 0.5$$

$$p(\mathrm{Noun}|\mathrm{The}) = \frac{\exp(1*1)}{\exp(1*1)+\exp(1*1)} = 0.5$$

Prvious word The(Det), The (Noun) 0.5 probability

Observed Word is complex Word

For The(Det)

complex(Verb) No feature is present

complex (Adj.) F1,F4,F5 feature is present

For The(Noun)

complex(Verb) F2 feature is present

complex (Adj.) F4,F5 feature is present

Prvious word The(Det), The (Noun) 0.5 probability.   Observed Word is complex Word

For The(Det) :   complex(Verb)   No feature is present

complex (Adj.) F1, F4, F5 feature is present

$$p(\text{Verb}|\text{complex}) = \frac{\exp(0)}{[\exp(1*1)+\exp(1*1)+\exp(1*1)]+\exp(0)} = 0.11$$

$$= 0.11 \times 0.5 = 0.055$$

$$p(\text{Adj.}|\text{complex}) = \frac{\exp(1*1)+\exp(1*1)+\exp(1*1)}{[\exp(1*1)+\exp(1*1)+\exp(1*1)]+\exp(0)} = 0.89$$

$$= 0.89 \times 0.5 = 0.455$$

Prvious word The(Det), The (Noun) 0.5 probability.   Observed Word is **complex** Word

For The(Noun)

complex(Verb)   F2 feature is present

complex (Adj.) F4,F5 feature is present

$$p(\text{Verb}|\text{complex}) = \frac{\exp(1)}{[\exp(1 \cdot 1) + \exp(1 \cdot 1)] + \exp(1)} = 0.33$$

$$= 0.33 \times 0.5 = 0.165$$

$$p(\text{Adj.}|\text{complex}) = \frac{\exp(1 \cdot 1) + \exp(1 \cdot 1)}{[\exp(1 \cdot 1) + \exp(1 \cdot 1)] + \exp(1)} = 0.66$$

$$= 0.66 \times 0.5 = 0.33$$

First Word: The	Second Word:Complex	Answer
Det	Verb	0.055
Det	Adj.	**0.455**
Noun	Verb	0.165
Noun	Adj.	**0.330**

Beam Size=2

First Word: The	Second Word:Complex	Answer
Det	Adj.	**0.455**
Noun	Adj.	**0.330**

First Word: The	Second Word:Complex	Answer
Det	Adj.	**0.455**

For record : record(Verb) No feature is present

record (Noun) F3, F6 feature is present

$$p(\text{Verb}|\text{record}) = \frac{\exp(0)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.16$$

$$= 0.16 \times 0.455 = 0.0728$$

$$p(\text{Noun}|\text{record}) = \frac{\exp(1 * 1) + \exp(1 * 1)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.85$$

$$= 0.85 \times 0.455 = 0.387$$

First Word: The	Second Word:Complex	Answer
Noun	Adj.	**0.330**

For record : record(Verb) No feature is present

record (Noun) F3, F6 feature is present

$$p(\text{Verb}|\text{record}) = \frac{\exp(0)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.16$$

$$= 0.16 \times 0.330 = 0.0528$$

$$p(\text{Noun}|\text{record}) = \frac{\exp(1 * 1) + \exp(1 * 1)}{[\exp(1 * 1) + \exp(1 * 1)] + \exp(0)} = 0.85$$

$$= 0.85 \times 0.330 = 0.281$$

First Word The	Second Word Complex	Third Word Record	Answer
Det	Adj. (0.455)	Verb	0.0728
Det	Adj. (0.455)	Noun	**0.387**
Noun	Adj. (0.330)	Verb	0.0528
Noun	Adj. (0.330)	Noun	0.281

The	complex	record
Det.	Adj	Noun