Nature Language Processing

Dihui Lai

dlai@wustl.edu

July 13, 2019

Regular Expression

- Algebraic notation for characterizing a set of strings.
- Useful for searching in corpus of texts
- Python example: str = "The rain in Spain"; x = re.findall("Spain", str)

Basic Regular Expression

- []: Used to indicate a set of characters e.g. [abc] will match 'a', 'b', or 'c'.
- '\d': matches any decimal digit; equivalent to the set [0-9].
- '+': match 1 or more repetitions of the preceding RE
- '\D': anything but a number (a non-digit)
- '\s': space (tab,space,newline etc.)
- '\S': anything but a space e.g. '\S+@\S+' represents email address
- '^': This expression matches the start of a string

Regular Expression in Python

re.search(regex, text) returns a match object when the pattern is found or not match if the pattern is not found.

```
1 import re
2 m = re.search('hello', 'hello world, hello all, good
      afternoon')
3 print m.group(0)
4 #hello
```

re.findall(regex, text) will return a list of all the matches.

```
1 m = re.findall('hello', 'hello world, hello all, good
      afternoon')
2 print m
3 #['hello', 'hello']
```

N-gram Language Models

- Models that assign probabilities to sequences of words are called language models or LM.
- An n-gram is a sequence of N words e.g. 2-gram (or bigram) "Good Morning", 3-gram "Turn it on"
- N-gram lanuage models estimate the probability of the last word of an n-gram given the previous words

N-gram Language Models

LM: What is the probability of having a sentence that consists a sequence of words: w_1, w_2, w_3 ... w_N, i.e. $P(w_1, w_2, w_3 ... w_N)$.
Recall the chain rule:

$$P(w_1, w_2, w_3 ... w_N)$$
$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)P(w_4|w_1, w_2, w_3)...P(w_N|w_1, w_2, ...w_{N-1})$$

In the case of bigram, we assume $P(w_N|w_1, ..., w_{N-1}) = P(w_N|w_{N-1})$, since the word is only dependent on the previous word, it is also called Markov assumption. In general case of an n-gram, we assume

$$P(w_N|w_1, w_2, ...w_{N-1}) = P(w_N|w_{N-1}, w_{N-2}, ...w_{N-n+1})$$

MLE Estimation for bigram

In the case of bigram, the MLE estimation can be formulated as

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

Here, C is the count of the words' occurence

Example: MLE Estimation for bigram

Estimate the bigram for the following corpus, here $\langle s \rangle$ and $\langle /s \rangle$ are introduced as the symbols that represents the begining and end of a setence.

$\langle s \rangle$ I am Sam $\langle /s \rangle$

$\langle s \rangle$ Sam I am $\langle /s \rangle$

$\langle s \rangle$ I do not like green eggs and ham $\langle /s \rangle$

We begin buy counting the words occurence and have $C(I) = 3$, $C(Sam) = 2$, $C(\langle /s \rangle) = 3$, $C(\langle s \rangle) = 3$ ...$C(\langle s \rangle I) = 2$, $C(\langle s \rangle Sam) = 1$

So we have $P(I|\langle s \rangle) = \frac{2}{3}$, $P(Sam|\langle s \rangle) = \frac{1}{3}$, $P(do|I) = \frac{1}{3}$, $P(am|I) = \frac{2}{3}$, $P(Sam|am) = \frac{1}{2}$, $P(\langle /s \rangle|Sam) = \frac{1}{2}$

The in-sample probability of $P(\langle s \rangle I\ am\ Sam \langle /s \rangle) = P(I|\langle s \rangle)P(am|I)P(Sam|am)P(\langle /s \rangle|Sam) = 2/3x2/3x1/2x1/2$

Evaluating Language Models

How do we compare two LM?

- A test data/hold out data set can be used to evaluate a LM. Apply the estiamated conditional probability to the test data set and compare the resulting probability.
- Perplexity is used instead of the raw probability.

$$PP(W) = P(w_1, w_2, ...w_N)^{-\frac{1}{N}}$$
$$= \sqrt[N]{\frac{1}{P(w_1, w_2, ...w_N)}}$$

- Maximize probability is equivalent to minimize perplexity

Smoothing

What do we do with words that appear in a test set with an unseen context for example, $P(John|am) = 0$ because "John" has never appear in training text. We end up getting $P(w_1, w_2, ...w_N) = 0$. One possible solution is smoothing

- Laplace smoothing: increase the bigram count by 1, so what was counted 0 now becomes 1

$$P(w_N|w_{N-1}) = \frac{C(w_{N-1}w_N) + 1}{\sum_w C(w_{N-1}w) + 1} = \frac{C(w_{N-1}w_N) + 1}{C(w_{N-1}) + V}$$

  The denominator is adjusted by the vocabulary size of V

- Add-k smoothing, increase the count by a fraction of k (0.5, 0.8 ...) and we have

$$P(w_N|w_{N-1}) = \frac{C(w_{N-1}w_N) + k}{\sum_w C(w_{N-1}w) + k} = \frac{C(w_{N-1}w_N) + k}{C(w_{N-1}) + kV}$$

Unknown Words

What do we do if a word in the test data is not in the vocabulary i.e. out of vocabulary (OOV)

- Choose a fixed vocabulary. If a word in the training set is OOV, convert it to $\langle UNK \rangle$. Estimate the probability of $\langle UNK \rangle$ as a regular word.
- replace low frequency word in the training dataset by $\langle UNK \rangle$. Treating $\langle UNK \rangle$ as regular word.

Word Semantics and Representations

- Homonymous: a word can have multiple definitions e.g. mouse could mean small rodents or it could mean computer devices.
- Synonyms/antonym (words' relations): couch/sofa, vomit/throw up, filbert/hazelnut; long/short, big/little
- Word sentiments
- Can we represent a word using vectors and quantify those measures?

Document Representations

A document can be represented by the words and the number of their occurence using term-document matrxix

Words	As You Like It	Twelfth Night	Ulius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	5
wit	20	15	2	3

tf-idf Weighted Measure

Certain words are more common in all documents e.g. the, it, they. The less frequent like "litigation" might be more important that the more frequent word "good". The tf-idf algorithms can be used to adjust the intuition.

$$w_{t,d} = tf_{t,d} \times idf_t$$

Here,

$$tf_{t,d} = 1 + log_{10}(count(t,d)) \text{ if } count(t,d) > 0, \text{ else } 0$$

With $count(t,d)$ the number of occurence of the term t in the document d

$$idf_t = log_{10}\left(\frac{N}{df_t}\right)$$

N is the total number of documents in the collection, and df_t is the number of documents in which term t occurs. If word good appears in all collected document, $idf_t = 0$

Word Vector Representations

Term-term matrix or word-word matrix: count the number of times a word occurs in a context window around the target word (e.g. ± 7)
sugar, a sliced lemon, a tablespoonful of, **apricot** jam, a pinch each of,

	aardvark	...	computer	data	pinch	result	sugar	..
apricot	0	...	0	0	1	0	1	..
pineapple	0	...	0	0	1	0	1	..
digital	0	...	2	1	0	1	0	..
information	0	...	1	6	0	4	0	..

It can be inferred from the word-word matrxi that apricot and pineapple are more simliar to each other.

Consine Similarity

The similarity of two words could be measured by dot-products of their vector representation

$$\vec{v} \cdot \vec{w} = \sum_{i=1}^{N} v_i w_i$$

The dot-product favors vectors of higher frequency to normalize the similarity without considering word frequency, we use cosine similarity meature

$$cosine(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_1^N v_i^2}\sqrt{\sum_1^N w_i^2}}$$

Skip-Gram

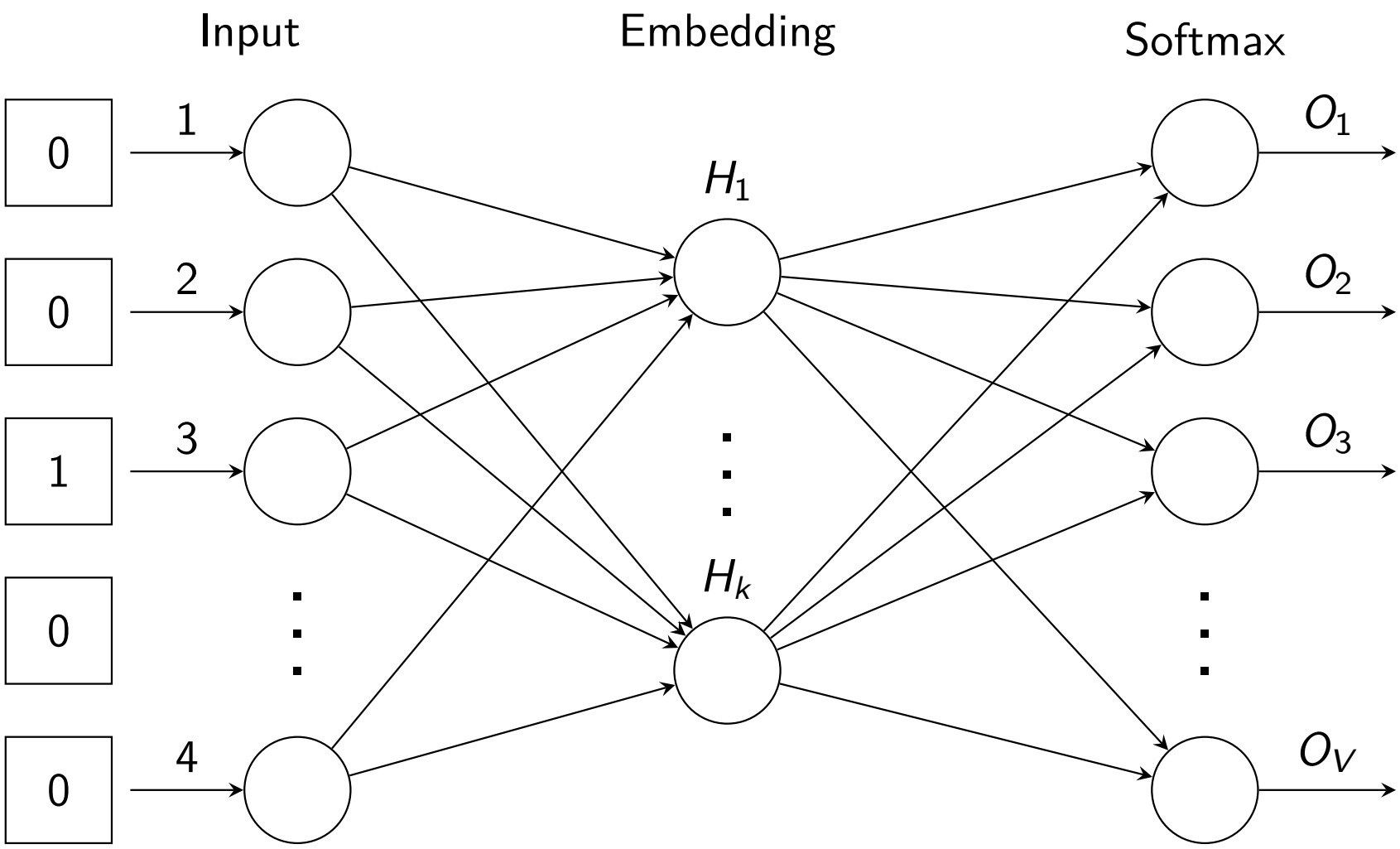
- The model try to increase the similarity between $\vec{t}$ and $\vec{c}$ if the a context word appears next to the target word
- Skip-Gram uses neural network to predict the probability of the words c is is within the context a target word t, $P(+|t,c)$ or is NOT within the context a target word $P(-|t,c)$
- The likelihood function is defined by

$$L = logP(+|t,c) + \sum_{i=1}^{k} logP(-|t,n_i)$$

$$= log\frac{1}{1+e^{-\vec{c}\cdot\vec{t}}} + \sum_{i=1}^{k} log\frac{1}{1+e^{\vec{n_i}\cdot\vec{t}}}$$

- There are two types of embeddings for a word in the neural network model: target embedding $\vec{t}$ and context embedding $\vec{c}$,

Skip-Gram: Neural Network Architecture



A vocabulary is fed into the nerual network using one-hot encoding methods. For a vocabulary of size V, the input vector is of size 1xV

Parts of Speech

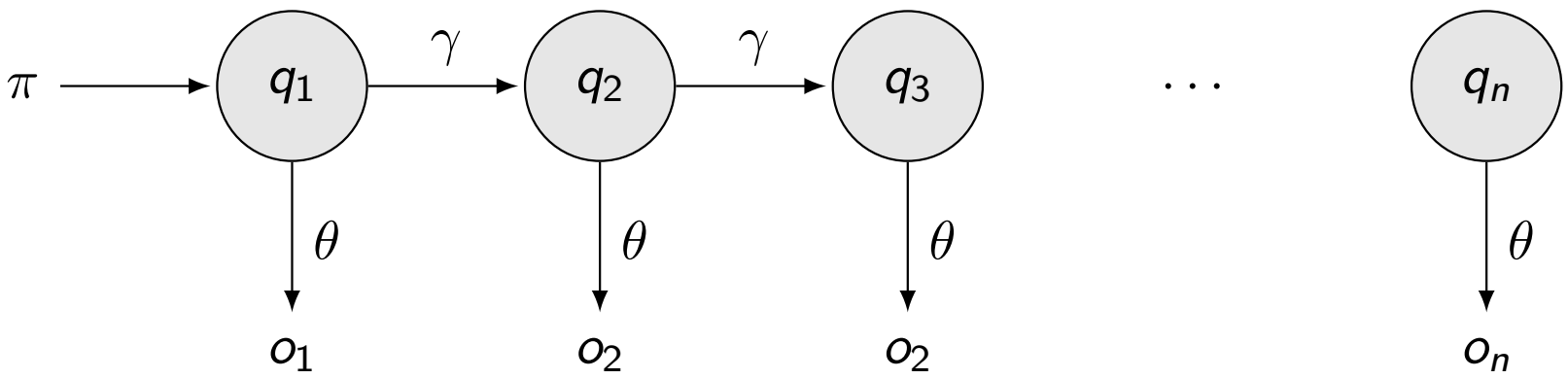
- Parts-of-speech (POS): noun, verb, pronoun, preposition, adverb, conjunction, participle, and article
- For example: Can (modal) you (personal pronoun) buy (verb) me (personal pronoun) a (determiner) tea (proper noun)?
- POS reveal the property of a word and its neighbors

English Penn Treebank part-of-speech Tagset

An important tagset for English is the 45-tag Penn Treebank tagset.

`https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

Hidden Markov Model



Hidden Markov Model

$$
\begin{aligned}
& p(o_1, o_2, \ldots, o_T, q_1, q_2, \ldots, q_T) \\
&= p(q_1, q_2, \ldots, q_T) \prod_{i=1}^{T} p(o_i \mid q_i) \\
&= p(q_T \mid q_1, q_2, \ldots, q_{T-1}) p(q_1, q_2, \ldots, q_{T-1}) \prod_{i=1}^{T} p(o_i \mid q_i) \\
&= p(q_T \mid q_{T-1}) p(q_1, q_2, \ldots, q_{T-1}) \prod_{i=1}^{T} p(o_i \mid q_i) \\
&= \prod_{i=1}^{T} p(q_i | q_{i-1}) \prod_{i=1}^{T} p(o_i \mid q_i)
\end{aligned}
$$

At each time step, the probability of being in state j after seeing the first t observations, given the automaton λ can be calculated as

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)\gamma_{ij}\theta_j(o_t),$$

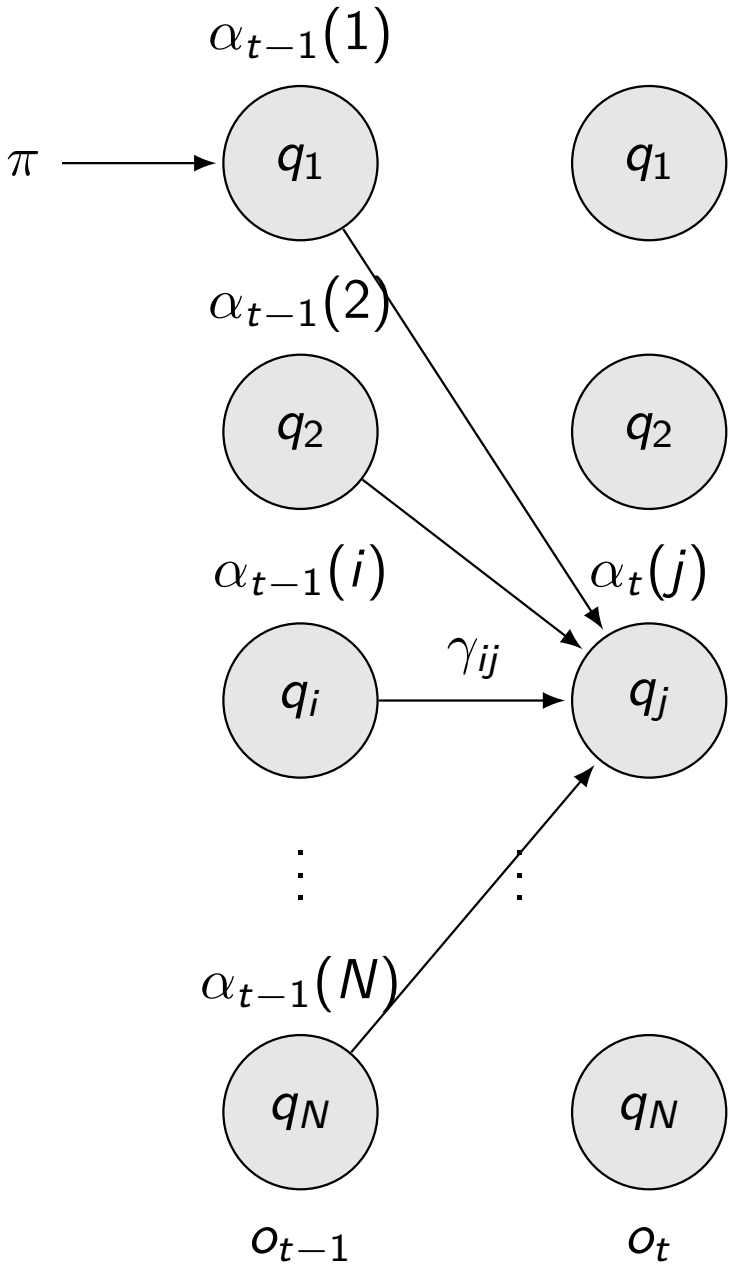
where

$$1 \leq t \leq T, 1 \leq j \leq N$$

where $\theta_j(o_t)$ is the likelihood of observing o_t given the current state is j, λ_{ij} is the probability of transition from hidden state i to hidden state j. The probability of state q_j at time t is a summation over all paths that could lead to the state. Therefore, the probability of seeing the observed sequence can be calculated as

$$P(o \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Hidden Markov Model



HMM: Viterbi Algorithm

Given that we know the HMM $\lambda = (\Gamma, \Theta)$ and an observed squence of $O = o_1, o_2, ...o_T$, find the most probable sequence of states $Q = q_1, q_2, q_2, ..., q_T$.
The viterbi path probablity: the probability of the most probable path that leads to the jth state at time step t. Denoted as

$$\nu_t(j) = \max_{q1,q2,q3,...q_{t-1}} p(q_1, q_2, ...q_{t-1}, q_t = j, o_1, o_2, ..., o_t \mid \lambda)$$

We can calculate the viterbi path probabilty of being at state q_j at time t as

$$\nu_t(j) = \max_{i=1}^{N} \nu_{t-1}(i)\gamma_{ij}\theta_j(o_t)$$

The Viterbie Algorithm needs to back trace the most likely state sequence that leads to the probability. $bt_t(j)$, which can be found by

$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} \nu_{t-1}(i)\gamma_{ij}\theta_j(o_t)$$

HMM: Backward Algorithm

Denote the probability of seeing the observation after $t+1$ as $o_{t+1}, o_{t+2}, ...o_T$, given that we are at state j at time t.

$$\beta_t(j) = p(o_{t+1}, o_{t+2}, ..., o_T \mid q_t = j, \lambda)$$

The $\beta_t(j)$ can be expressed as the following formula

$$\begin{aligned}\beta_t(j) &= \sum_{k=1}^{N} \gamma_{jk}\theta_k(o_{t+1})p(o_{t+2}, o_{t+3}, ..., o_T \mid q_{t+1} = k, \lambda) \\ &= \sum_{k=1}^{N} \gamma_{jk}\theta_k(o_{t+1})\beta_{t+1}(k)\end{aligned}$$

The transition probability of going from *ith* state at time $t-1$ to *jth* state at time t can be calculated as

$$p(O|\lambda) = \sum_{t=1}^{N}\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)$$

$$p(q_t = i, q_{t+1} = j, O \mid \lambda) = \sum_{t=1}^{N}\sum_{j=1}^{N}\alpha_t(i)\gamma_{ij}\theta_j(o_{t+1})\beta_{t+1}(j)$$

$$\begin{aligned}\hat{\gamma_{ij}} &= p(q_{t-1} = i, q_t = j, \mid O, \lambda) \\ &= p(q_{t-1} = i, q_t = j, O \mid \lambda)/p(O \mid \lambda) \\ &= \frac{\sum_{t=1}^{N}\alpha_t(i)\gamma_{ij}\theta_j(o_{t+1})\beta_{t+1}(j)}{\sum_{t=1}^{N}\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}\end{aligned}$$

Similarly, to calculate the emission probability $\theta_j(o_t)$

$$\begin{aligned}\eta_t(j) &= p(q_t = j \mid O, \lambda)) \\ &= \frac{p(q_t = j, O \mid \lambda)}{p(O \mid \lambda)} \\ &= \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}\end{aligned}$$

The emission probability can be estimated as

$$\hat{b_j}(k) = \frac{\sum_{t=1, s.t. o_t = k}^{T} \eta_t(j)}{\sum_{t=1}^{T} \eta_t(j)}$$

Name Entity Recognition (NER)

- NER labels words in a texts that are names of things e.g. person, organization, money amount, gene/protein names
- John (person) Lee (person) is the chief of CBSE (organization).