* You can get the link of slides at E3 “Hands-on 1”
* We will start the course at 9:05.

109-2
Natural Language Processing

Lab - 1

Yun-Zhu Song , Yi-Syuan Chen

Colab Link HackMD Link

Topics

- Dataset
  - Tweets
- Data Processing
  - Use NLTK to tokenize data
  - Remove punctuations and lower the cases
- N-gram Model
  - Build the model
  - Generate a sequence
- POS Tagging
  - Use NLTK to do POS tagging for the generated sequence

N-Gram Model

- Reference

Unigram Model

$$p(w_i) = \frac{c(w_i)}{\sum_w c(w)}$$

Bigram Model

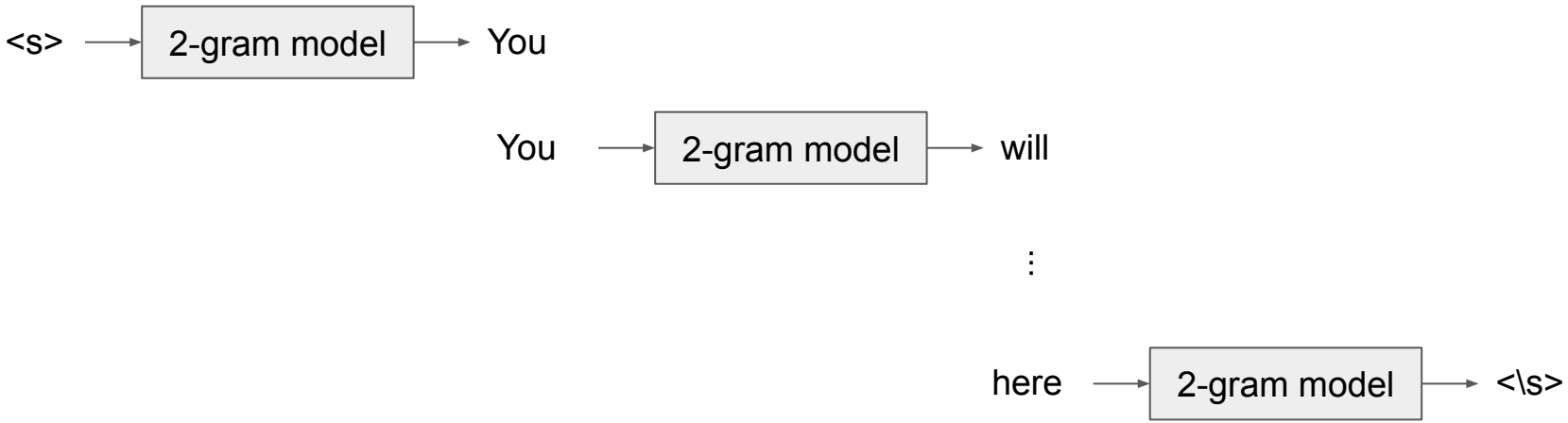
$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{\sum_w c(w_{i-1}, w)}$$

Trigram Model

$$p(w_i|w_{i-1}, w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{\sum_w c(w_{i-2}, w_{i-1}, w)}$$

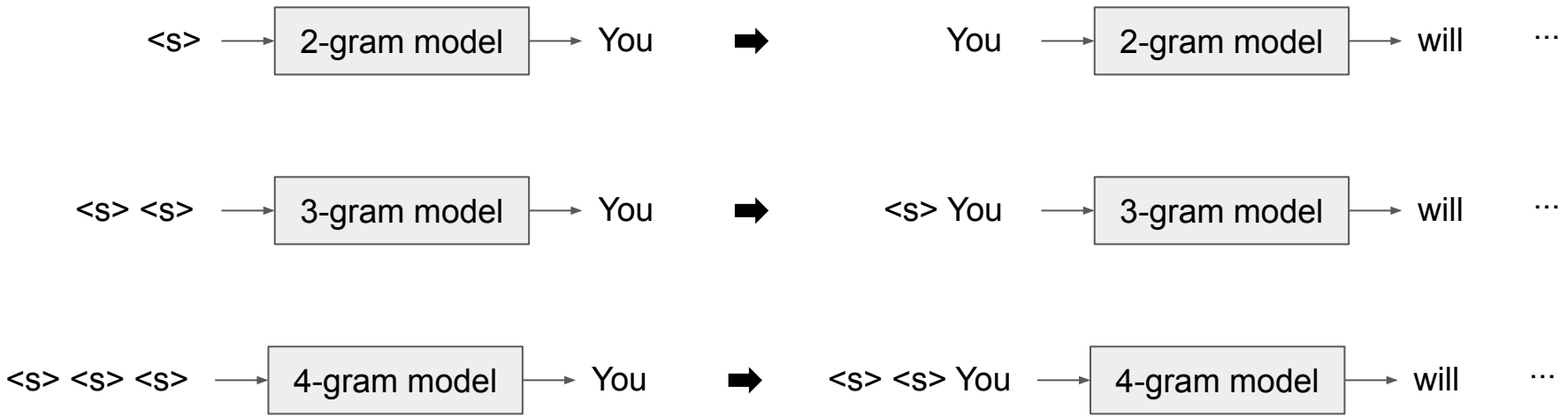
N-Gram Model

- When using n-gram model to generate sequence, we use add special tokens <s> and <\s> as the beginning and ending.



N-Gram Model

- When using n-gram model to generate sequence...



POS-Tagging

- Reference

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**	**.**

Requirements

- You can write your own code or use the code provided by TA to fulfill the following requirements.
- Part 1. Data Preprocessing
  - Show the top-10 common words and their counts before/after preprocessing.
- Part 2. N-Gram Model and POS Tagging
  - Build 2-gram / 4-gram model with processed dataset.
  - Show the top-5 probable next words and their probability after initial token ‘<s>’ by 2-gram model.
  - Generate a sentence with 2-gram model and find the POS taggings.
  - Generate a sentence with 4-gram model and find the POS taggings.

Requirements

- Inputs
  - CSV file , use the content

```
,content
0,The president is unhinged
1,Shucks..they made Acosta cry
2,This is what divides us and caused the hate
3,"Keep telling yourself those lies and Bot Sentinel is set up by people like you
going to be a lot of people who don't agree with you."
4,Already debunked last weekend. You're late to the party.
5,Scottish sun! Full of shit as usual
6,"But you never, ever hear it reported that they got it wrong so the morons out
7,Jim Acosta is a nothingburger he'll never be relevant as a journalist is like a
laughable
8,It's hardly hard reading an autocue! Time this tv licence was scrapped!
9,Have some more Kool-Aide with your delusions
```

Requirements

- Outputs
  - Part 1.1 : show the top-10 common words and their counts before/after preprocessing

Fixed

```
Before preprocessing: [(',', 175690), ('the', 170721), ('.', 146750), ('to', 102328), ('of', 86019), ('a', 80936), ('and', 79132), ('in', 78560), ('"', 64865), ('’', 53377)]
After preprocesing: [('the', 196586), ('to', 102786), ('of', 86330), ('a', 85708), ('in', 82951), ('and', 79803), ('s', 59795), ('’', 53377), ('on', 46551), ('said', 39352)]
```

  - Part 2.1 : Build 2-gram / 4-gram model by processed dataset
    - No need, we will check the code.
  - Part 2.2 : Show the top-5 probable next words and their probability after initial token ‘<s>’ by 2-gram model

Fixed

```
Next word predictions of two gram model: [('the', 0.14570487715636984), ('it', 0.027895599853644655), ('in', 0.0260932608784031), ('a', 0.02275283563481631), ('but', 0.021506104915100348)]
```

Requirements

- Outputs
  - Part 2.3 : Generate a sentence with 2-gram model and find the POS taggings

Fixed

```
Generation results of two gram model: 2 elections opinion poll
[('2', 'CD'), ('elections', 'NNS'), ('opinion', 'NN'), ('poll', 'NN')]
```

  - Part 2.4 : Generate a sentence with 4-gram model and find the POS taggings

Fixed

```
Generation results of four gram model:  cannava ' s lawyer sven mary said on sunday
[('cannava', 'NN'),
 (''', 'NNP'),
 ('s', 'VBZ'),
 ('lawyer', 'NN'),
 ('sven', 'NN'),
 ('mary', 'NNP'),
 ('said', 'VBD'),
 ('on', 'IN'),
 ('sunday', 'NN')]
```

Notes

- **You should follow the format but not the content of the example outputs. (which is generated from other datasets)**
- Long lists will be displayed in multiple line, but it will not affect our judgements.

Submission

- Deadline
  - Submit files to E3 before **11:59 AM (before the class)** today.
  - Accept late submissions before **11:59 PM** today (80% of original scores).
  - No grades after **11:59 PM** today.
- Format
  - Lab_1_<StudentID>.ipynb (ex: Lab_1_1234567.ipynb)
  - Make sure the .ipynb file contains the **correct execution results**.

Grading Policy

- Correctness
  - We have 5 problems, and **each accounts for 20 points**.
  - Use **“Runtime/Restart and run all”** to check your execution results on Colab.
- Clarity
  - Write your code with clear logic, and documents it properly.

Utilities

- NLTK
  - word_tokenize
  - sent_tokenize

`nltk.tokenize.`**`sent_tokenize`**(*text*, *language='english'*) [source]

Return a sentence-tokenized copy of *text*, using NLTK's recommended sentence tokenizer (currently `PunktSentenceTokenizer` for the specified language).

Parameters

- **text** – text to split into sentences
- **language** – the model name in the Punkt corpus

`nltk.tokenize.`**`word_tokenize`**(*text*, *language='english'*, *preserve_line=False*) [source]

Return a tokenized copy of *text*, using NLTK's recommended word tokenizer (currently an improved `TreebankWordTokenizer` along with `PunktSentenceTokenizer` for the specified language).

Parameters

- **text** (*str*) – text to split into words
- **language** (*str*) – the model name in the Punkt corpus
- **preserve_line** (*bool*) – An option to keep the preserve the sentence and not sentence tokenize it.

Utilities

- NLTK
  - pos_tag

`nltk.tag.pos_tag(tokens, tagset=None, lang='eng')` [source]

Use NLTK's currently recommended part of speech tagger to tag the given list of tokens.

```
>>> from nltk.tag import pos_tag
>>> from nltk.tokenize import word_tokenize
>>> pos_tag(word_tokenize("John's big idea isn't all that bad."))
[('John', 'NNP'), ("'s", 'POS'), ('big', 'JJ'), ('idea', 'NN'), ('is', 'VBZ'),
("n't", 'RB'), ('all', 'PDT'), ('that', 'DT'), ('bad', 'JJ'), ('.', '.')]
>>> pos_tag(word_tokenize("John's big idea isn't all that bad."), tagset='universal')
[('John', 'NOUN'), ("'s", 'PRT'), ('big', 'ADJ'), ('idea', 'NOUN'), ('is', 'VERB'),
("n't", 'ADV'), ('all', 'DET'), ('that', 'DET'), ('bad', 'ADJ'), ('.', '.')]
```

NB. Use *pos_tag_sents()* for efficient tagging of more than one sentence.

Parameters

- **tokens** (*list(str)*) – Sequence of tokens to be tagged
- **tagset** (*str*) – the tagset to be used, e.g. universal, wsj, brown
- **lang** (*str*) – the ISO 639 code of the language, e.g. 'eng' for English, 'rus' for Russian

Returns

The tagged tokens

Return type

list(tuple(str, str))

Utilities

- collections
  - Counter : a special dict, convenient for counting

```
>>> c = Counter()                           # a new, empty counter
>>> c = Counter('gallahad')                 # a new counter from an iterable
>>> c = Counter({'red': 4, 'blue': 2})      # a new counter from a mapping
>>> c = Counter(cats=4, dogs=8)             # a new counter from keyword args
```

```
>>> Counter('abracadabra').most_common(3)
[('a', 5), ('r', 2), ('b', 2)]
```

Utilities

- collections
  - namedtuple : a special tuple, convenient to access.

```
>>> # Basic example
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)     # instantiate with positional or keyword arguments
>>> p[0] + p[1]             # indexable like the plain tuple (11, 22)
33
>>> x, y = p                # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y               # fields also accessible by name
33
>>> p                       # readable __repr__ with a name=value style
Point(x=11, y=22)
```

Utilities

- pdb
  - set_trace()
    - n : execute next line
    - q : exit

Let’s start

Demonstration code