FoLT Tutorial 3 Summary

N-Gram Language Models:

Part 1: Text Generation:

- we are gonna make a function has 3 parameters:
    - `nltk.ConditionalFreqDist` that takes a word and generate the next word that has the highest probability
    - `start_word` that we are gonna use as the root of the phrase
    - `number_of_words` which gives us how many words will be generated after `start_word`
- Code **`generate_sentence`**:

```python
def generate_sentence(
    bigram_model: nltk.ConditionalFreqDist,
    start_word: str,
    number_of_words=20
) -> str:

    prev_word = start_word
    words = [prev_word]
    for i in range(number_of_words):
        # select the next word with the highest probability
        word = bigram_model[prev_word].max()
        prev_word = word
        words.append(word)
    return " ".join(words)
```

- Example of usage:

```python
generate_sentence(cfd, "living", 4)
```

- Result:

```
'living creature that he said'
```

Part 2: Build bigram model from a corpus

- In this part we are gonna use the `austen-persuasion.tx` fom the `gutenberg` corpus
- We are gonna make a function that does the following:
  - take all the words from `austen-persuasion.txt`
  - make a bigram out of those words using `nltk.bigrams()`
  - initiate `bigram_model` for **`generate_sentence`** using `nltk.ConditionalFreqDist`
- Code **`build_bigram_models_from_austen_persuasion`**:

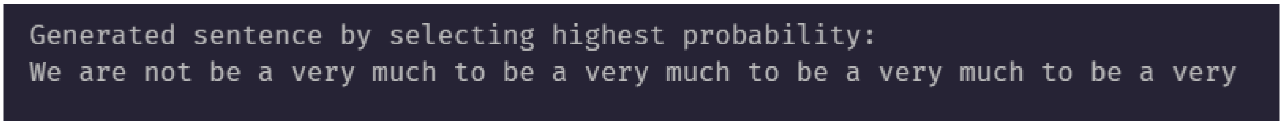
```
def build_bigram_models_from_austen_persuasion():
    gutenberg_words = gutenberg.words('austen-persuasion.txt')
    gutenberg_bigrams = nltk.bigrams(gutenberg_words)
    bigram_model = nltk.ConditionalFreqDist(gutenberg_bigrams)
    return bigram_model

bigram_model = build_bigram_models_from_austen_persuasion()
```

- Example:

```
start_word = "We"
generated_text_w_highest_prob = generate_sentence(bigram_model, start_word)
print("Generated sentence by selecting highest probability:\n{}".format(
    generated_text_w_highest_prob))
```

- Result:



```
Generated sentence by selecting highest probability:
We are not be a very much to be a very much to be a very much to be a very
```

Part 3: Generate sentence from most common words

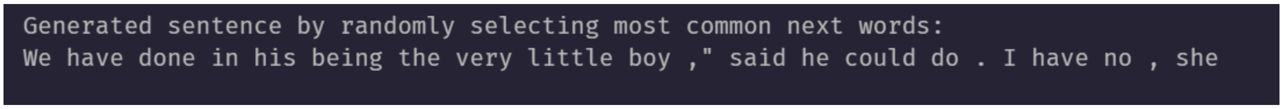
- This part is is similar to **PART 2**, but the only difference, is that we are gonna use randomness to pick 1 of the top 5 words
- Things you should know:
  - `ConditionalFreqDist.most_common(n: int)` takes a number `n` as input and returns a list of `n` most common words and their counts:
    ->[(word_1, n_1), (word_2, n_2), ...]
  - `random.choice` takes input as a list of tokens (top 5 possible tokens given the previous token) and randomly selects one.
- Code **`generate_sentence_from_most_common_words`**:

```python
def generate_sentence_from_most_common_words(
        bigram_model: nltk.ConditionalFreqDist,
        start_word: str,
        number_of_words=20,
        number_of_most_common_words=5
) -> str:
    for i in range(number_of_words):
        candidates = [word
                      for (word, count) in
bigram_model[prev_word].most_common(number_of_most_common_words)]
        next_word = random.choice(candidates)
        list_word.append(next_word)
        prev_word = next_word
```

- Example:

```
start_word = "We"
generated_text_w_most_common =
generate_sentence_from_most_common_words(bigram_model,
start_word)
print("Generated sentence by randomly selecting most common
next words: \n{}".format(
    generated_text_w_most_common))
```

- Result:



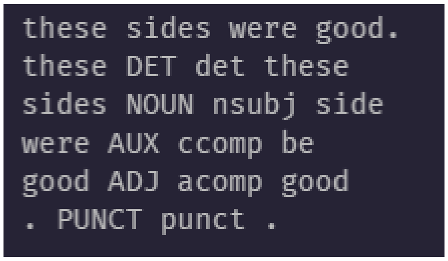
```
Generated sentence by randomly selecting most common next words:
We have done in his being the very little boy ," said he could do . I have no , she
```

Part 4: Documents and spans with spaCy (We are gonna use concepts from Lecture 1)

- In this section we are gonna use spacy to get a `token's` :
  - Part Of Speech *(Assigning word types to tokens, like verb or noun.)*
  - Dependency label *(Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.)*
  - Lemmatization *(Assigning the base forms of words.)*
- Code + Example:

```
raw = "Hard to judge whether these sides were good. We were
grossed " \
      "out by the melted styrofoam and didn't want to eat
it for fear of getting sick."
nlp = spacy.load("en_core_web_sm")
doc = nlp(raw)
span = doc[4:9]
print(span)
for token in span:
    print(token.text, token.pos_, token.dep_, token.lemma_)
```

- Result:



```
these sides were good.
these DET det these
sides NOUN nsubj side
were AUX ccomp be
good ADJ acomp good
. PUNCT punct .
```