COMP 6721 Applied Artificial Intelligence (Fall 2023)

Lab Exercise #10: Introduction to NLP

Solutions

Question 1 Assume that we are working with the Shloutan language. If you don't know Shloutan, don't worry; it is a simple language made of only 5 words: loola nikee aloka bibi vo.
You want to build a word language model for Shloutan. The training corpus that you use is the following:

"Loola nikee. Aloka bibi vo. Vo bibi loola. Loola nikee bibi vo. Vo. Vo. Aloka bibi loola. Loola aloka aloka. Loola loola. Nikee nikee nikee. Bibi vo. Bibi vo. Vo Vo. Nikee loola."

You can ignore case distinctions and sentence boundaries when answering the following questions.

(a) What is the value of P(vo | bibi)?

$$P(vo|bibi) = \frac{\text{freq}(bibi\ vo)}{\sum_{w_i} \text{freq}(bibi\ w_i)} = \frac{4}{6}$$

(b) What is the value of P(bibi vo)?

$$P(bibi\ vo) = \frac{\text{freq}(bibi\ vo)}{\sum_{w_j} \sum_{w_i} \text{freq}(w_j\ w_i)} = \frac{4}{32}$$

(c) Build a bigram language model based on this training corpus. Show the frequencies and the probabilities for each bigram.

Frequencies

	loola	nikee	aloka	bibi	vo
loola	3	3	1	0	0
nikee	1	2	1	2	0
aloka	1	0	1	2	0
bibi	2	0	0	0	4
vo	0	1	1	2	5

Conditional Probabilities

	loola	nikee	aloka	bibi	vo
loola	0.43	0.43	0.14	0	0
nikee	0.17	0.33	0.17	0.33	0
aloka	0.25	0	0.25	0.50	0
bibi	0.33	0	0	0	0.67
vo	0	0.11	0.11	0.22	0.56

(d) Smooth your bigram language model using "add 0.5". Show the frequencies and the probabilities for each bigram.

Frequencies

	loola	nikee	aloka	bibi	vo
loola	3.5	3.5	1.5	0.5	0.5
nikee	1.5	2.5	1.5	2.5	0.5
aloka	1.5	0.5	1.5	2.5	0.5
bibi	2.5	0.5	0.5	0.5	4.5
vo	0.5	1.5	1.5	2.5	5.5

Conditional Probabilities

	loola	nikee	aloka	bibi	vo
loola	0.37	0.37	0.16	0.05	0.05
nikee	0.18	0.29	0.18	0.29	0.06
aloka	0.23	0.08	0.23	0.38	0.08
bibi	0.29	0.06	0.06	0.06	0.53
vo	0.04	0.13	0.13	0.22	0.48

(e) Using each language model from parts (c) and (d), which of the following two sentences is more probable. Show all your work.

sentence 1: Aloka vo nikee aloka.

Bigrams: (aloka vo), (vo nikee), (nikee aloka)
Model (c): $0 \times 0.11 \times 0.17 = 0$
Model (d): $0.08 \times 0.13 \times 0.18 \approx 0.0019$

sentence 2: Vo nikee nikee aloka.

Bigrams: (vo nikee), (nikee nikee), (nikee aloka)
Model (c): $0.11 \times 0.33 \times 0.17 \approx 0.0062$
Model (d): $0.13 \times 0.29 \times 0.18 \approx 0.0068$

Sentence 2 is more probable using either model.

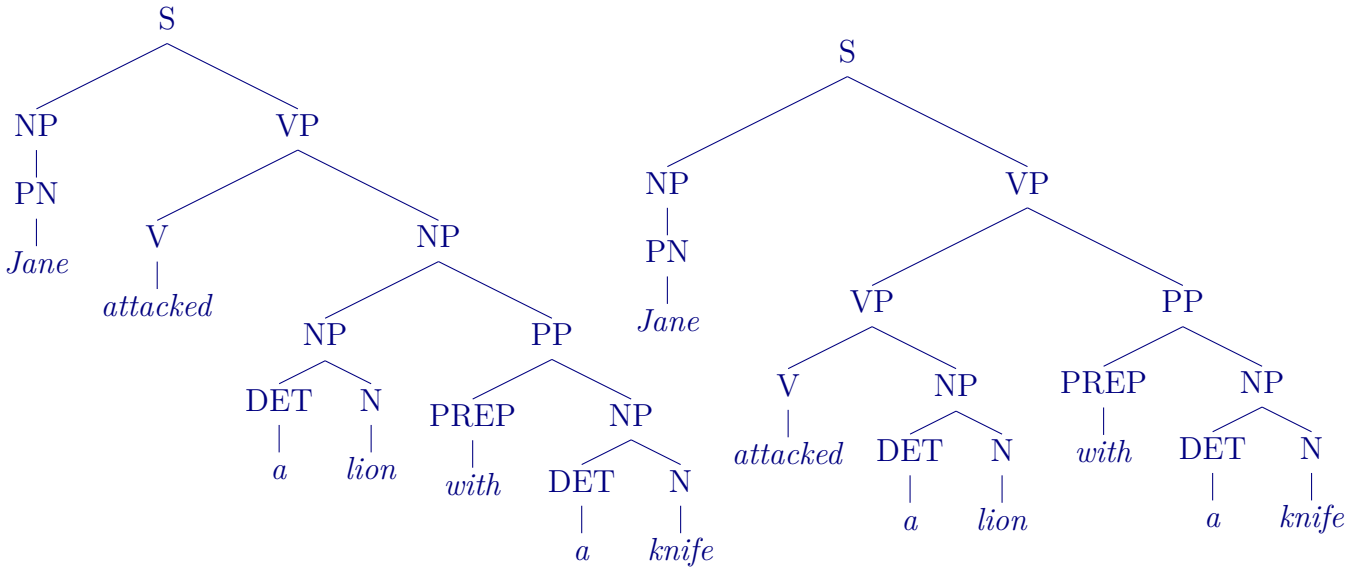
Question 2 Consider the following context-free grammar:

S → NP VP	N → lion
VP → V	N → knife
VP → V NP	N → zoo
VP → VP PP	V → attacked
NP → DET N	PN → Jane
NP → PN	DET → the
NP → NP PP	DET → a
PP → PREP NP	PREP → with
	PREP → in

(a) Generate all possible parse trees for the following sentence:

i *Jane attacked a lion with a knife*

There are two possible parse trees:



In the sentence *"Jane attacked a lion with a knife,"* the context-free grammar rules lead to two different parse trees, demonstrating syntactic ambiguity. The left tree utilizes the rule NP → NP PP, interpreting *"with a knife"* as part of the noun phrase, resulting in *"a lion with a knife."* This suggests the lion, associated with the knife, is the object of the attack. The right tree follows the rule VP → VP PP, where *"with a knife"* modifies the verb phrase *"attacked a lion,"* indicating that Jane used a knife in the attack. These variations highlight how specific grammar rules, can yield different meanings from the same sentence structure in natural language processing.

Question 3 Consider the following sentences as training set for disambiguating the sense of the word *light*.

Sentence	**Sense**
There is a ray of light even if it is dim.	Sense1
This faint ray of light is barely enough to see.	Sense1
It needs to be light in weight to be portable.	Sense2
I can see dim red light coming from that room.	Sense1
This laptop is light enough to carry around.	Sense2
I am not hurt since I was hit by something as light as a candle.	Sense2
There is a faint hint of light at the end of the street.	Sense1
But the room was bright and light from the candle was warm as it fell on the surroundings.	Sense1
My bright red overcoat is light but fairly warm.	Sense2
This red candle stand is light enough to carry it to the room.	Sense2

Further consider this list of stop words:

a, and, are, as, at, be, for, from, in, is, it, if, of, that, this, the, to, was

Using a Naive Bayes approach with:

- a context window of ± 3 words,
- a vocabulary with the size of 31,
- smoothing with the value of 0.2, and
- stop-word removal

Calculate the scores of each possible sense and find the most probable sense of the word *light* in the following sentence:

In her dark red room, that dim light from the candle was warm enough.

The sentence with the word to disambiguate-
~~*In*~~ *her dark* *red* *room*, ~~*that*~~ *dim* ***light*** ~~*from*~~ ~~*the*~~ *candle* ~~*was*~~ *warm* *enough*.

Note: The stop words have been depicted as strike outs, since they will not be considered in predicting the sense. The words in the box are the words that are a part of the sentence and lie within the context window. The probabilities of these words for each sense in the training set will be considered.

Each of the words in the box above is also highlighted (boxed) in each of the training sentences below, when it lies within the context window of the word we are trying to disambiguate:

There ~~*is*~~ ~~*a*~~ *ray* ~~*of*~~ ***light*** *even* ~~*if*~~ ~~*it*~~ ~~*is*~~ *dim*.
~~*This*~~ *faint* *ray* ~~*of*~~ ***light*** ~~*is*~~ *barely* *enough* ~~*to*~~ *see*.
~~*It*~~ *needs* ~~*to*~~ ~~*be*~~ ***light*** ~~*in*~~ *weight* ~~*to*~~ ~~*be*~~ *portable*.

I can see dim red ***light*** *coming ~~from~~ ~~that~~ room.*
~~This~~ laptop ~~is~~ ***light*** *enough ~~to~~ carry around.*
I am not hurt since I ~~was~~ hit by something ~~as~~ ***light*** *~~as~~ ~~a~~ candle.*
There ~~is~~ ~~a~~ faint hint ~~of~~ ***light*** *~~at~~ ~~the~~ end ~~of~~ ~~the~~ street.*
But ~~the~~ room ~~was~~ bright ~~and~~ ***light*** *~~from~~ ~~the~~ candle ~~was~~ warm ~~as~~ ~~it~~ fell on ~~the~~ surroundings.*
My bright red overcoat ~~is~~ ***light*** *but fairly warm.*
~~This~~ red candle stand ~~is~~ ***light*** *enough ~~to~~ carry ~~it~~ ~~to~~ ~~the~~ room.*

$P(Sense1) = 5/10 = 1/2 = 0.5$
$P(Sense2) = 5/10 = 1/2 = 0.5$

$$P(red|Sense1) = \frac{1 + 0.2}{25 + (31 \times 0.2)} = \frac{1.2}{31.2} = 0.038461538$$
$$P(room|Sense1) = \frac{2 + 0.2}{25 + (31 \times 0.2)} = \frac{2.2}{31.2} = 0.070512821$$
$$P(dim|Sense1) = \frac{2 + 0.2}{25 + (31 \times 0.2)} = \frac{2.2}{31.2} = 0.070512821$$
$$P(candle|Sense1) = \frac{1 + 0.2}{25 + (31 \times 0.2)} = \frac{1.2}{31.2} = 0.038461538$$
$$P(warm|Sense1) = \frac{1 + 0.2}{25 + (31 \times 0.2)} = \frac{1.2}{31.2} = 0.038461538$$
$$P(enough|Sense1) = \frac{1 + 0.2}{25 + (31 \times 0.2)} = \frac{1.2}{31.2} = 0.038461538$$

$$P(red|Sense2) = \frac{2 + 0.2}{23 + (31 \times 0.2)} = \frac{2.2}{29.2} = 0.075342466$$
$$P(room|Sense2) = \frac{1 + 0.2}{23 + (31 \times 0.2)} = \frac{1.2}{29.2} = 0.04109589$$
$$P(dim|Sense2) = \frac{0 + 0.2}{23 + (31 \times 0.2)} = \frac{0.2}{29.2} = 0.006849315$$
$$P(candle|Sense2) = \frac{2 + 0.2}{23 + (31 \times 0.2)} = \frac{2.2}{29.2} = 0.075342466$$
$$P(warm|Sense2) = \frac{1 + 0.2}{23 + (31 \times 0.2)} = \frac{1.2}{29.2} = 0.04109589$$
$$P(enough|Sense2) = \frac{2 + 0.2}{23 + (31 \times 0.2)} = \frac{2.2}{29.2} = 0.075342466$$

$score(Sense1)$
$= log(Sense1)+log(P(red|Sense1))+log(P(room|Sense1))+log(P(dim|Sense1)+$
$log(P(candle|Sense1) + log(P(warm|Sense1) + log(P(enough|Sense1)$
$= log(0.5)+log(0.038461538)+log(0.070512821)+log(0.070512821)+log(0.038461538)+$
$log(0.038461538) + log(0.038461538)$
$= -8.264387214$

score(Sense2)
$= log(Sense2) + log(P(red|Sense2)) + log(P(room|Sense2)) + log(P(dim|Sense2) +$
$log(P(candle|Sense2) + log(P(warm|Sense2) + log(P(enough|Sense2)$
$= log(0.5) + log(0.075342466) + log(0.006849315) + log(0.04109589) + log(0.075342466) +$
$log(0.04109589) + log(0.075342466)$
$= -8.606666574$

Since score(Sense1) > score(Sense2), ***Sense1*** *is more probable.*

Question 4 Preprocessing text is a very crucial step for any NLP related task. There are many frameworks (GATE[1], UIMA[2]) and libraries (CoreNLP[3], NLTK[4], spaCy[5]) available out there to facilitate text analysis.

Today we will be looking into spaCy to get you started with the basic steps.

Installing spaCy. To install spaCy using Conda, use:

```
conda install -c conda-forge spacy
```

After the installation, execute the following command to verify that you have installed spaCy version 3 or upper.

```
python -m spacy info
```

Downloading language models. Not all languages are preprocessed in the same way. Depending on the language, tokenization, sentence splitting, POS tagging, etc. varies. To facilitate analyzing different languages and genres, spaCy provides different pre-trained language pipelines to work with.[6] For starters, we will be working with the basic English model. We can download the model with the following command:

```
python -m spacy download en_core_web_sm
```

Let's start coding! Use the following to import the spaCy library and load the language model:

```
import spacy
nlp = spacy.load("en_core_web_sm")
```

Now let's try to preprocess the text in your worksheet and check if the POS tags we assigned are what spaCy returns as well:

```
doc = nlp("I prefer a direct flight to Chicago.")
```

Depending on the language model and sentence provided as input, spaCy outputs an object `doc`, in our case with a variety of annotations encoded within. Linguistic annotations are available as token attributes. Let's try to print some basic attributes:

[1] https://gate.ac.uk/
[2] http://uima.apache.org/
[3] https://stanfordnlp.github.io/CoreNLP/index.html
[4] https://www.nltk.org/book/
[5] https://spacy.io/usage/spacy-101
[6] https://spacy.io/usage/models

```
for token in doc:
   print(token.text, token.lemma_, token.tag_, token.dep_)
```

(a) Now try to process the following paragraph to see how spaCy performs sentence splitting:

> *"Superman was born on the planet Krypton and was given the name Kal-El at birth. As a baby, his parents sent him to Earth in a small spaceship moments before Krypton was destroyed in a natural cataclysm. His ship landed in the American countryside, near the fictional town of Smallville. He was found and adopted by farmers Jonathan and Martha Kent, who named him Clark Kent. Clark developed various superhuman abilities, such as incredible strength and impervious skin. His adoptive parents advised him to use his abilities for the benefit of humanity, and he decided to fight crime as a vigilante. To protect his privacy, he changes into a colorful costume and uses the alias "Superman" when fighting crime. Clark Kent resides in the fictional American city of Metropolis, where he works as a journalist for the Daily Planet. Superman's supporting characters include his love interest and fellow journalist Lois Lane, Daily Planet photographer Jimmy Olsen and editor-in-chief Perry White. His classic foe is Lex Luthor, who is either a mad scientist or a ruthless businessman, depending on the story."*

Start by processing the text:

```
doc = nlp(text)
```

This performs *sentence splitting* as part of the analysis pipeline. Now add code to print all the resulting sentences.

```
sentence_spans = list(doc.sents)
for sentences in sentence_spans:
   print(sentences)
```

(b) In addition to the constituent parse trees discussed in the lecture and Question 2 above, dependency parsing[7] is another formalism used in English grammars. In dependency parse trees, each token (word) is connected to another by a single arc, where the arc's label indicates the syntactic relationship between them. The starting node of the arc is known as the *Governor* (or *head* in spaCy terminology), and the endpoint is the *Dependant* (called *child* in spaCy). Each 'Dependant/Child' can have only

[7] See https://web.stanford.edu/~jurafsky/slp3/18.pdf and https://spacy.io/usage/linguistic-features#dependency-parse for more details on dependency parse trees

one 'Governor/Head,' but a 'Governor/Head' may have multiple dependents. Every dependency tree also has a root, typically the main verb of the sentence, and the arcs point from heads to their children.

When you process text with spaCy's `nlp` function (like in `doc = nlp(text)`), a dependency parse tree is automatically created for each sentence. Now, try to use spaCy's "displaCy visualizer"[8] to visualize the dependency parse tree for the sentence given in your worksheet — the result should look like Figure 1.

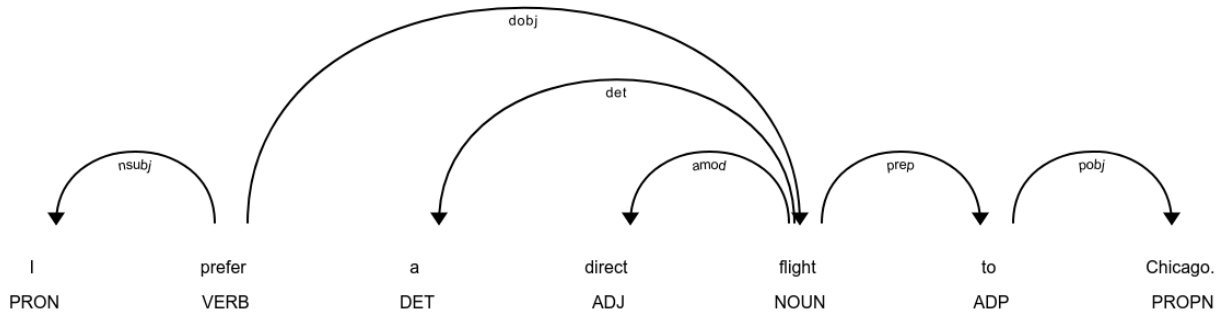


Figure 1: Dependency parse tree created by spaCy for the sentence we used on the worksheet

```
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("I prefer a direct flight to Chicago.")
displacy.serve(doc, style="dep")
```

You should be able to visualize the graph by clicking the link from the run command window or by opening your browser at `http://0.0.0.0:5000`

As depicted in the graph, the token which has no arrows points pointed at (or the token from which the arcs starts) is known as the *root*, which is the main action in a sentence. Then, `nsubj` (i.e., *noun subject*) refers to the main subject of the sentence, which is *"I"* in our case and `dobj` (i.e., *direct object*) refers to the object that we are talking about in a sentence; here it is about *"flight"*. The tokens *"a"* and *"direct"* are *determiners* (`det`) and *adjective modifiers* (`amod`), respectively; hence, for the token *"flight"* you see the arc starting from here.

Through a dependency graph, you can see how tokens are connected with one another and you can extract the main subject, object and the action given a sentence, no matter how long it is. This is an important aspect in many language related tasks, for example to summarize a text.

[8] See `https://spacy.io/usage/visualizers`

(c) Now we will look at *Named Entity Recognition* (NER), an essential task in natural language processing (NLP) that involves identifying and classifying proper nouns in text into predefined categories such as names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. NER is crucial for understanding the content of a document by highlighting the key entities and is widely used in information extraction, content classification, and data analysis.

Using spaCy, we can easily perform NER with minimal coding. spaCy's NER model is pre-trained on a large corpus and can recognize a wide range of entity types. Start by processing the following text with spaCy and explore the identified named entities:

```
text = "Apple is looking at buying U.K. startup for $1 billion. Barack
    ↪ Obama visited Apple headquarters in California."
```

Process this text using the spaCy pipeline and iterate through the named entities in the processed document, which are stored in the `ents` property of a `doc` object. For each entity, print out its text and the entity type it has been classified as. For example, "Apple" should be recognized as an *Organization* and "Barack Obama" as a *Person*.

To print out the detected entities, you can use:

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

After processing the text with spaCy, the model identifies and categorizes each entity. For example, 'Apple' is recognized as an organization, 'U.K.' as a geopolitical entity, '1 billion' as a monetary value, and 'Barack Obama' as a person. This exercise demonstrates spaCy's capability in accurately identifying and classifying entities in text, which is crucial for various NLP applications.

You can also use displaCy to visualize the detected entities:

```
displacy.serve(doc, style='ent')
```

This lab marks your initial foray into Natural Language Processing (NLP), a foundational aspect of AI that bridges language and technology. The experience gained with spaCy is crucial for understanding how machines interpret human language. The practical skills you've developed are applicable in various real-world scenarios, from enhancing search engine algorithms to improving automated customer support systems. NLP's relevance spans multiple industries, including healthcare, finance, and media, where insights from text data are invaluable. As you delve deeper into NLP, you'll find the field replete with opportunities for innovation and advancement. Continue building upon these skills, and you may soon find yourself adept at creating applications that significantly impact the way we interact with technology in language-driven contexts.