Homework1

September 12, 2023

[ ]:
```
import nltk, re
from collections import Counter, defaultdict

nltk.download("punkt")

nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/magnusde93/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/magnusde93/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[ ]:
```
True
```

[ ]:
```
pattern = r"(\b\w+(?:'\w+)?|[.,!?;:-])"

with open("of-mice-and-men.txt", "r") as file:
    text = file.read()

tokenizer = re.findall(pattern, text)
print(f"There are {len(tokenizer)} tokens overall")
print(f"There are {len(set(tokenizer))} unique tokens")
print("10 of the most common tokens are", Counter(tokenizer).most_common(10))

lower_tokenizer = re.findall(pattern, text.lower())
print("10 of the most common tokens are", Counter(lower_tokenizer).
 ↪most_common(10))

words_8 = [word for word in tokenizer if len(word) > 8]
print(f"{len(words_8)} words are longer than 8 characters and they are␣
 ↪{set(words_8[:10])} and more.")

longestWordLength = len(max(tokenizer, key=len))
```

```
print(f"The longest words are {set([textword for textword in tokenizer if␣
 ↪len(textword) == longestWordLength])} and are {longestWordLength} characters␣
 ↪long")
```

```
There are 35603 tokens overall
There are 3291 unique tokens
10 of the most common tokens are [('.', 3246), (',', 1679), ('the', 1191),
('and', 780), ('a', 683), ('I', 658), ('to', 579), ('you', 491), ('he', 462),
('in', 438)]
10 of the most common tokens are [('.', 3246), (',', 1679), ('the', 1319),
('and', 836), ('he', 757), ('a', 721), ('you', 673), ('i', 660), ('to', 580),
('his', 457)]
571 words are longer than 8 characters and they are {'sculptured', 'skittering',
'mountains', 'recumbent', 'sycamores', 'horizontal', 'junctures', 'direction',
'twinkling'} and more.
The longest words are {'apologetically', 'sonofabitching', 'leatherworking',
'contemptuously', 'disapprovingly'} and are 14 characters long
```

[ ]:
```
tokens = nltk.word_tokenize(text)
tagged_tokens = nltk.pos_tag(nltk.word_tokenize(text))

def get_pos_tag_frequencies(tagged_tokens, num_tags=10):
    pos_tags = [pos for _, pos in tagged_tokens]
    tag_freq = Counter(pos_tags)
    return tag_freq.most_common(num_tags)

top_pos_tags = get_pos_tag_frequencies(tagged_tokens)

print(top_pos_tags)
```

```
[('NN', 4251), ('PRP', 3783), ('.', 3492), ('IN', 2867), ('DT', 2854), ('VBD',
2702), ('RB', 2213), ('NNP', 2191), ("'", 1702), (',', 1679)]
```

[ ]:
```
def find_words_with_multiple_tags(tagged_tokens):
    word_pos_mapping = defaultdict(set)

    for word, pos in tagged_tokens:
        word_pos_mapping[word].add(pos)

    ambiguous_words = {word: tags for word, tags in word_pos_mapping.items() if␣
 ↪len(tags) > 1}
    return ambiguous_words


ambiguous_words = find_words_with_multiple_tags(tagged_tokens)
ambiguous_items = list(ambiguous_words.items())[:10]
```

```
for word, tags in ambiguous_items:
    print(f"{word}: {tags}")
```

```
A: {'DT', 'NNP'}
south: {'RB', 'NN'}
drops: {'NNS', 'VBZ'}
close: {'NN', 'VBZ', 'RB', 'JJ'}
warm: {'VB', 'NN', 'JJ'}
slipped: {'VBD', 'VBN'}
over: {'RB', 'RP', 'NN', 'IN'}
yellow: {'NN', 'JJ'}
before: {'RB', 'IN'}
one: {'NN', 'CD'}
```

[ ]:
```
def word_with_most_tags(ambiguous_words):
    if not ambiguous_words:
        return None

    max_tags = max(len(tags) for tags in ambiguous_words.values())
    words_with_max_tags = [word for word, tags in ambiguous_words.items() if␣
 ↪len(tags) == max_tags]

    return words_with_max_tags, max_tags

words_with_max_tags, max_tags = word_with_most_tags(ambiguous_words)

print(words_with_max_tags, max_tags)
```

```
[''] 15
```

[ ]:
```
def most_common_word_pos_pairs(tagged_tokens, num_pairs=10):
    word_pos_pairs = [(word, pos) for word, pos in tagged_tokens]
    word_pos_freq = Counter(word_pos_pairs)
    return word_pos_freq.most_common(num_pairs)

top_word_pos_pairs = most_common_word_pos_pairs(tagged_tokens, num_pairs=10)

print(top_word_pos_pairs)
```

```
[(('.', '.'), 3123), ((',', ','), 1679), (("''", "''"), 1269), ((``, ``),
1237), (('the', 'DT'), 1190), (('and', 'CC'), 777), (('I', 'PRP'), 741), (('a',
'DT'), 674), (('to', 'TO'), 578), (('you', 'PRP'), 508)]
```

[ ]:
```
def get_pos_tag_trigrams(tagged_tokens, num_trigrams=10):
    pos_tags = [pos for _, pos in tagged_tokens]
    pos_trigrams = [(pos_tags[i], pos_tags[i+1], pos_tags[i+2]) for i in␣
 ↪range(len(pos_tags) - 2)]
```

```
    return pos_trigrams[:num_trigrams]

top_pos_tag_trigrams = get_pos_tag_trigrams(tagged_tokens, num_trigrams=10)

print(top_pos_tag_trigrams)
```

```
[('DT', 'JJ', 'NNP'), ('JJ', 'NNP', 'RB'), ('NNP', 'RB', 'IN'), ('RB', 'IN',
'NNP'), ('IN', 'NNP', ','), ('NNP', ',', 'DT'), (',', 'DT', 'NNP'), ('DT',
'NNP', 'NNP'), ('NNP', 'NNP', 'VBZ'), ('NNP', 'VBZ', 'IN')]
```

[ ]:
```
sentence = "The complex houses married and single soldiers and their families."

tagged_sentence = nltk.pos_tag(nltk.word_tokenize(sentence))

print(tagged_sentence)


'''I tried to find a sentence that would mess with the sentence and interpret␣
 ↪the words in a wrong way but it seems as if NLTK is very good at␣
 ↪interpreting the words in the right way.'''
```

```
[('The', 'DT'), ('complex', 'JJ'), ('houses', 'NNS'), ('married', 'VBD'),
('and', 'CC'), ('single', 'JJ'), ('soldiers', 'NNS'), ('and', 'CC'), ('their',
'PRP$'), ('families', 'NNS'), ('.', '.')]
```

[ ]:
```
'This may cause'
```

1. Manually gathering data. Pros: Extremely precise. Since it's a human gathering the data they can be very anal about the data. Cons: It's very expensive and it takes a long time.

Scraping the internet. Pros: A lot of data available. Very cheap and so many different topics to choose from. Cons: The internet is full of useless data. Some websites have rules so you can't scrape them.

Using crowd-sourcing. Pros: Large data available and not expensive. Can also help to have contributors of all backgrounds. Cons: Hard to control the quality especially when contributors have different skill levels.

Using NLP systems. Pros: It's very fast and consistent across large datasets. There are pre-trained models out there to use. Cons: They aren't flawless and you can only use pre-trained models.

It all depends on what we're working with. If I have enough money I would obviously manually gather the data, unless there's a website I can scrape from that's extremely reliable, like visindavefurinn. If I don't have money nor time I would scrape the internet, anything I could find that might be handy.

2. It's important to anonymize data but some types of NLP systems require access to non-anonymized data like language models for content generation. GPT-3 needs to access diverse rext to generate human-like behaviour. Chatbots need data to provide information to clients. Speech recognition systems rely on human voices to convert it into text.

3. If the AI is trained with prejudice it can have negative implications of these models. It can be racist, discriminate or use offensive language. If it's trained on prejudice it can base it's answer on the human it's trying to help. If it's racist it could give a bad answer to someone who's of a minority while giving others the answer needed. Gender, race, sexuality and more issues come up when the AI model is trained on prejudice. You're less likely to trust an AI model if it's prejudice. Great examlpe of a AI model is the Tay chatbot that was trained by human interaction. It became so offensive and racist that Microsoft had to take down the bot and issue an apology for the bot.

4. I did run into some trouble with the regex. Decided to just put "wasn't" and words with "n't" as a single token. That fixed one problem. I wish the dots and commas weren't a part of the token even though I know they serve a big part of it all. It did take me time to google all of the tags for every word in NLTK.