NTNU

Norwegian University of Science and Technology

TDT4310, Lab session 3

Stemming/Lemmatization, TF-IDF, and part-of-speech tagging

Tollef Jørgensen, February 13, 2024.

The course so far

- Few seem to have problems with the course material
  - Surprisingly few questions on forums
    - Please don't hesitate to ask us about whatever you're wondering about!

A recap: working with the labs

- The git repo will update throughout. You can clone it, and make sure you update with **git pull** when necessary
  - oooor: just check by the github page and download the exercise files directly.
    - Sometimes, I will change the material, but there will be no consequences if you deliver an "out of date" file.

NTNU

Before we continue

- Are there any concerns about the setup so far?
- Are the topics interesting?
- I wish we could delve right into the most "hyped" implementation parts, but that would require another course :-)

Lab 3 topics

- Stemming/lemmatization
- TF-IDF
- Part-of-speech tagging

Lab 3 topics

- Stemming/lemmatization
- TF-IDF
- Part-of-speech tagging

I will cover some basics of each topic,
then you work on the lab questions related to it:

stemming -> work -> tfidf -> work -> tagging -> work work work

But why?

- **Stemming/lemmatization**
  - Common word forms, making your inputs more predictable
  - Mostly used in "basic" NLP systems (classification, for example)
- **TF-IDF**
  - Weight the importance of words relative to the current document in context of all available documents
    - Given a query, how relevant is it for each document in my collection?
- **Part-of-speech tagging**
  - Word structure!
    - Can be used to generate groups of words as we'll see in the next lab (dependency parsing)
  - If done correctly, we can resolve ambiguities.
    - Obviously relies on context, and this can be (naively) implemented with n-grams.

Stemming and Lemmatization

Stemming and Lemmatization

- Part of what we call "text normalization" or "text cleaning"
- Converting text to its standard form - either by:
  - Stemming - reducing words to their "stem", by removing suffixes
  - Lemmatization - using dictionaries and definitions to reduce words to their base form

Word	Stemming	Lemmatization
information	inform	information
informative	inform	informative
computers	comput	computer
feet	feet	foot

Stemming and Lemmatization

- Both approaches will result in weird and ungrammatical sentences.
  - But... For many applications, this won't matter!
    - Any guesses?

Word	Stemming	Lemmatization
information	inform	information
informative	inform	informative
computers	comput	computer
feet	feet	foot

```
"this lecture describes some concepts within computational linguistics"
# STEMM.: "thi lectur describ some concept within comput linguist"
# LEMMA.: "this lecture describes some concept within computational linguistics"
```

Using wordnet to lemmatize

```
some
Synset('some.a.01') quantifier; used with either mass nouns or plural count nouns
Synset('some.s.02') relatively much but unspecified in amount or extent
Synset('some.s.03') relatively many but unspecified in number
Synset('some.s.04') remarkable
Synset('approximately.r.01') (of quantities) imprecise but fairly close to correct
concepts
Synset('concept.n.01') an abstract or general idea inferred or derived from specif
within
Synset('inside.r.02') on the inside
computational
Synset('computational.a.01') of or involving computation or computers
linguistics
Synset('linguistics.n.01') the scientific study of language
Synset('linguistics.n.02') the humanistic study of language and literature
```

Get to work :-)

1. Given the list of pluralized words below, define your own simple word stemmer function or class, limited to only simple rules and regex. No libraries! It should strip basic endings.

```
plurals = [
    "flies",
    "denied",
    "itemization",
    "sensational",
    "reference",
    "colonizer",
]

# TODO: implement your own simple stemmer
```

[ ] MagicPython

2. After your initial implementation, run it on the following words:

```
new_words = [
    "friendly",
    "puzzling",
    "helpful",
]
# TODO: run your stemmer on the new words
```

[ ] MagicPython

NTNU

3. Realizing that fixing future words manually can be problematic, use a desired NLTK stemmer and run it on all the words:

```
import nltk

all_words = plurals + new_words

# TODO: use an nltk stemming implementation to stem `all_words`
```

MagicPython

4. There are likely a few words in the outputs above that would cause issues in real-world applications. Pick some examples, and show how they are solved with a lemmatizer. Use either spaCy or nltk.

Your answer here! Code below.

```
# TODO: basic observations on which examples are problematic with stemming + implement lemmatization
with spacy/nltk
```

MagicPython

NTNU

Stemming/Lemmatization - Practical Example

Using the news corpus (subset/category of the Brown corpus), perform common text normalization techniques such as stopword filtering and stemming/lemmatization. Compare the top 10 most common **words** before and after these normalization techniques.

```python
# import nltk; nltk.download('brown')  # ensure we have the data
from nltk.corpus import brown
news = brown.words(categories='news')

# TODO: find the top 10 most common words
```

MagicPython

```python
# TODO: find the top 10 most common words after applying text normalization techniques
```

NTNU

TF-IDF

TF-IDF

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

- Term Frequency (TF)
  - How often a word appears in a document.
  - The more frequent, the higher its TF score.
- Inverse Document Frequency (IDF)
  - How rare a word is across the corpus.
  - The rarer, the higher its IDF score

TF-IDF

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

- A way to determine the *importance of a word*
  - In a document relative to a collection of documents
- Example:
  - You're interested in looking up info about "robots"
    - Probably frequent in sci-fi books, less so in historical novels, probably.
      - If the word is *rare*, it will receive a higher score

TF-IDF

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

- Many applications!
  - Information retrieval (IR)
    - Given a query, find the relevant documents
  - Finding important keywords
    - Topic modeling and summarization
  - Sentiment analysis
    - By using the scores and weights of negative/ positive words

NTNU

Get to work :-)

TF-IDF (term frequency-inverse document frequency) is a way to measure the importance of a word in a document.

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Where:

- t is the term (word)
- d is the document
- D is the corpus

1. Implement TF-IDF using NLTKs FreqDist (no use of e.g. scikit-learn and other high-level libraries).
2. With your TF-IDF function in place, calculate the TF-IDF for the following words in the first document of the news articles found in the Brown corpus:

- *the*
- *nevertheless*
- *highway*
- *election*

Perform any preprocessing steps you deem necessary. Comment on your findings.

NTNU

3. While TF-IDF is primarily used for information retrieval and text mining, reflect on how TF-IDF could be used in a language modeling context.

Your answer here!

4. You were previously introduced to word representations. TF-IDF can be considered one. What are some differences between the TF-IDF output and one that is computed once from a vocabulary (e.g. one-hot encoding)?

Your answer here!

TF-IDF - Practical Example

You will again be looking at specific words for a document, but this time weighted by their TF-IDF scores. Ideally, the scoring should be able to retrieve representative words for this document in context of its document collection or category.

You will do the following:

- Select a category from the Reuters (news) corpus
- Perform preprocessing
- Calculate TF-IDF scores
- Find the top 5 words for a subset of documents in your collection (e.g. 5, 10, ..)
- Inspect whether these words make sense for a given document, and comment on your findings.

NTNU

Part-of-speech (POS) tagging

Tagging

- What?
  - Part-of-speech (POS) tagging
    - Adverb, verb, noun? Based on context
  - Labels that include extratextual information about a word
    - Not inferrable from the word itself, needs a predefined tagset
- Why?
  - Resolving ambiguities
  - Aiding downstream tasks
    - Speech recognition
    - Named entity recognition
    - Coreference resolution

Terms

- Tags
  - The labels
- Tagging
  - The process of assigning tags to a token
- Tag set
  - The collection of tags for a corpus/task

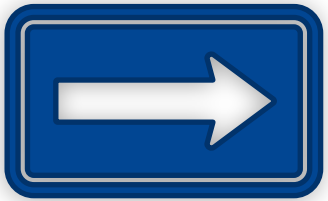
Let's look at some tags

- NLTK has different tagsets of tags, related to each corpora
  - nltk.help.upenn_tagset()
  - nltk.help.brown_tagset()
- With a given corpus:
  - nltk.corpus.<insert_corpus_here>.readme()
- Can be called with queries:
  - nltk.help.brown_tagset("VB.*")
    - Match all POS tags containing VB
      - VB+AT, VBN, VBZ, ...

nltk.help.brown_tagset("VB.*")

```
VB: verb, base: uninflected present, imperative or infinitive
    investigate find act follow inure achieve reduce take remedy re-set
    distribute realize disable feel receive continue place protect
    eliminate elaborate work permit run enter force ...
VB+AT: verb, base: uninflected present or infinitive + article
    wanna
VB+IN: verb, base: uninflected present, imperative or infinitive + prep
osition
    lookit
VB+JJ: verb, base: uninflected present, imperative or infinitive + adje
ctive
    die-dead
VB+PPO: verb, uninflected present tense + pronoun, personal, accusative
    let's lemme gimme
VB+RP: verb, imperative + adverbial particle
    g'ahn c'mon
VB+TO: verb, base: uninflected present, imperative or infinitive + infi
nitival to
    wanta wanna
```

Ambiguity



Right (direction)

Right (correct)

- Resolving homonyms
  - Definition
    - *each of two or more words having the same spelling or pronunciation but different meanings and origins*
- *You shall know a word by the company it keeps*
  - Not only relevant to POS-tagging, but especially to word embeddings, as you've seen already.

Ok... so how do we "tag"?

- Rule-based
  - Regex (typically suffix-based)
- Transformation-based
  - Rule-based + learning
- Stochastic
  - Probability-based
  - N-gram
  - Hidden Markov Models
- Deep learning

Rule-based tagging

- Regex

```
>>> patterns = [
...     (r'.*ing$', 'VBG'),                 # gerunds
...     (r'.*ed$', 'VBD'),                  # simple past
...     (r'.*es$', 'VBZ'),                  # 3rd singular present
...     (r'.*ould$', 'MD'),                 # modals
...     (r'.*\'s$', 'NN$'),                 # possessive nouns
...     (r'.*s$', 'NNS'),                   # plural nouns
...     (r'^-?[0-9]+(\.[0-9]+)?$', 'CD'),   # cardinal numbers
...     (r'.*', 'NN')                       # nouns (default)
... ]
```

  - See section 4.2 in ch. 5 (nltk book)

Transformation-based tagging

- Brill tagging
  - Regex + training template rules
  - Read up in section 6, ch. 5, if you're interested! Not required for the lab

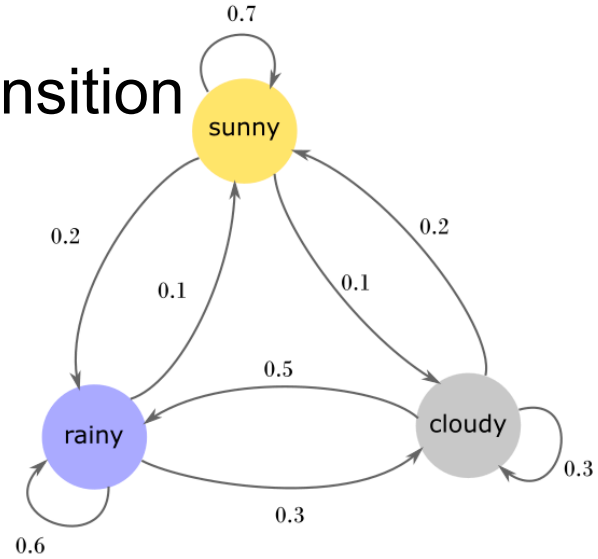
```
NN -> VB if the tag of the preceding word is 'TO'
NN -> VBD if the tag of the following word is 'DT'
NN -> VBD if the tag of the preceding word is 'NNS'
NN -> NNP if the tag of words i-2...i-1 is '-NONE-'
NN -> NNP if the tag of the following word is 'NNP'
NN -> NNP if the text of words i-2...i-1 is 'like'
NN -> VBN if the text of the following word is '*-1'
```

Stochastic tagging

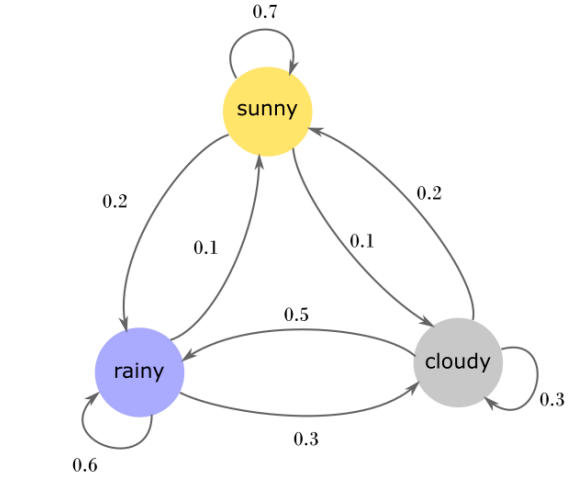
- What is the probability of the tag NN being assigned...
  - NN: Noun, singular
- Based on observed data
  - Find the probability of the next tag occurring
- Requires data
  - Train/Test/Eval (only train/test for now)

Hidden Markov Models (HMMs)

- Example:
  - The weather for any given day can be in any of the three states
    - Sunny, rainy, cloudy, with transition probabilities:



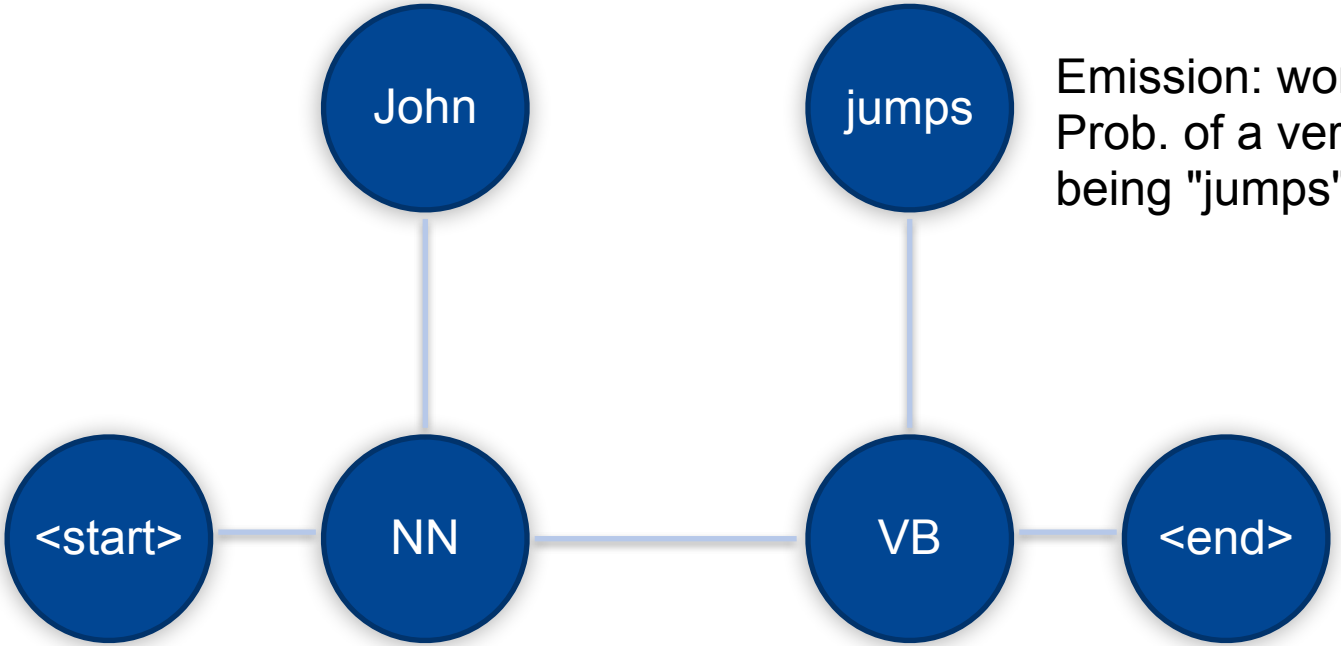
Hidden Markov Models (HMMs)



$$P(q_1, \ldots, q_n) = \prod_{i=1}^{n} P(q_i|q_{i-1})$$

- It is now rainy, what's the chance of it raining tomorrow and then sunny the day after?
  = P(Rainy|Rainy) * P(Sunny|Rainy)
  = 0.1 * 0.9 = 0.09 = 9%
- This applies just as well to POS tags!

HMMs for POS-tagging



Emission: word-to-tag
Prob. of a verb (VB) being "jumps"?

Transition: tag-to-tag
Prob. of a verb (VB) occurring after a noun (NN)?

Example task

- Find the most ambiguous words in the bible
- Steps:
  - POS-tag the bible
  - Generate sets of tags for each word
- Why
  - Practical example of ambiguity
  - Illustrate the importance of a good POS-tagger

- Notebook in the examples folder on GitHub

Example task

Result: some of the words with > 8 tags

```
unto: {'NNS', 'PRP$', 'RB', 'NNP', 'CC', 'RP', 'MD', 'NN', 'VBD', 'JJ', 'VBP', 'VBZ', 'RBR'
, 'IN', 'VB'}
forth: {'NNS', 'RB', 'JJS', 'RP', 'PDT', 'NN', 'VBD', 'VBN', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'
}
hath: {'RB', 'PDT', 'NN', 'VBD', 'MD', 'PRP', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'}
wherein: {'RB', 'NNP', 'CC', 'EX', 'WP', 'NN', 'VBD', "''", 'WRB', 'VBP', 'VBZ', 'WDT', 'JJ
', 'IN', 'JJR', 'VB'}
behold: {'RB', 'CC', 'VBN', 'NN', 'VBD', 'UH', 'JJ', 'VBP', 'VB'}
till: {'RB', 'CC', 'EX', 'NN', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'}
evil: {'RB', 'CC', 'EX', 'VBN', 'NN', 'VBD', 'JJ', 'VBP', 'NNS', 'VBZ', 'FW', 'VB'}
goeth: {'NNS', 'RB', 'NN', 'VBD', 'VBG', 'JJ', 'VBP', 'VBZ', 'VB'}
thou: {'NNS', 'RB', 'NNP', 'CC', 'EX', 'RP', 'VBN', 'NN', 'VBD', "''", 'PRP', 'MD', 'JJ', '
VBP', 'VBZ', 'IN', 'JJR', 'VB'}
eat: {'RB', 'NNP', 'NN', 'VBD', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'}
shalt: {'RB', 'MD', 'NN', 'VBD', 'VBN', 'PRP', 'JJ', 'VBP', 'NNS', 'VBZ', 'JJR', 'FW', 'VB'
}
thereof: {'NNS', 'RB', 'EX', 'RP', 'NN', 'VBD', 'PRP', 'JJ', 'VBP', 'VBZ', 'WDT', 'VB'}
meet: {'RB', 'VBN', 'NN', 'VBD', 'JJ', 'VBP', 'NNS', 'VBZ', 'FW', 'VB'}
```

Example task

Result: some of the words with > 8 tags

```
unto: {'NNS', 'PRP$', 'RB', 'NNP', 'CC', 'RP', 'MD', 'NN', 'VBD', 'JJ', 'VBP', 'VBZ', 'RBR'
, 'IN', 'VB'}
forth: {'NNS', 'RB', 'JJS', 'RP', 'PDT', 'NN', 'VBD', 'VBN', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'
}
hath: {'RB', 'PDT', 'NN', 'VBD', 'MD', 'PRP', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'}
wherein: {'RB', 'NNP', 'CC', 'EX', 'WP', 'NN', 'VBD', "''", 'WRB', 'VBP', 'VBZ', 'WDT', 'JJ
', 'IN', 'JJR', 'VB'}
behold: {'RB', 'CC', 'VBN', 'NN', 'VBD', 'UH', 'JJ', 'VBP', 'VB'}
till: {'RB', 'CC', 'EX', 'NN', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'}
evil: {'RB', 'CC', 'EX', 'VBN', 'NN', 'VBD', 'JJ', 'VBP', 'NNS', 'VBZ', 'FW', 'VB'}
goeth: {'NNS', 'RB', 'NN', 'VBD', 'VBG', 'JJ', 'VBP', 'VBZ', 'VB'}
thou: {'NNS', 'PRP', 'RB', 'NNP', 'CC', 'EX', 'WP', 'NN', 'VBD', 'VBN', 'JJ', '
VBP', 'VBZ', 'IN', 'JJR', 'VB'}
eat: {'RB', 'NNP', 'NN', 'VBD', 'JJ', 'VBP', 'VBZ', 'IN', 'VB'}
shalt: {'RB', 'MD', 'NN', 'VBD', 'VBN', 'PRP', 'JJ', 'VBP', 'NNS', 'VBZ', 'JJR', 'FW', 'VB'
}
thereof: {'NNS', 'RB', 'EX', 'RP', 'NN', 'VBD', 'PRP', 'JJ', 'VBP', 'VBZ', 'WDT', 'VB'}
meet: {'RB', 'VBN', 'NN', 'VBD', 'JJ', 'VBP', 'NNS', 'VBZ', 'FW', 'VB'}
```

Most of these are unintuitive... VBN, VBP, VBZ

Simplified "Universal" POS-tags

Tag	Meaning	English Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, univeristy*

- corpus.tagged_words(**tagset='universal'**)

Bible tagged with "simple" POS

- More understandable, less flexible for real-world systems

```
wherein: {'.', 'DET', 'NOUN', 'ADP', 'ADV', 'VERB', 'CONJ', 'ADJ', 'PRON'}
thou: {'.', 'DET', 'NOUN', 'ADP', 'PRT', 'ADV', 'VERB', 'CONJ', 'ADJ', 'PRON'}
ye: {'.', 'DET', 'NOUN', 'NUM', 'PRT', 'ADP', 'ADV', 'VERB', 'CONJ', 'ADJ', 'X', 'PRON
'}
doth: {'DET', 'NOUN', 'ADP', 'PRT', 'ADV', 'VERB', 'CONJ', 'ADJ', 'X'}
thee: {'DET', 'NOUN', 'ADP', 'PRT', 'ADV', 'VERB', 'CONJ', 'ADJ', 'PRON'}
```

Custom taggers

- What
  - A tagger that incorporates backoff
- How
  - Split data
  - Create custom backoff taggers
  - POS tag
  - Train taggers
  - Evaluate
- Why
  - Attempt to improve performance over a singular tagger

```
Default (no backoff) acc:           0.1314479001902567
Unigram (backoff: def) acc:         0.8944463322541195
Bigram (backoff: uni) acc:          0.913963601112907
Trigram (backoff: bi) acc:          0.9137859361820668
```

spaCy!

- As one of the last tasks, you will use spaCy for tagging
- Very high level APIs

```
import spacy
nlp = spacy.load("en_core_web_sm")
text = "The quick brown fox jumps over the lazy dog."
doc = nlp(text)
for token in doc:
    print(token.text, token.pos_)
    # Detailed tag description
    print(token.text, token.pos_, spacy.explain(token.pos_))
```

spaCy!

```
custom_nlp = spacy.load("en_core_web_sm")

# Add a custom rule for hashtags
custom_nlp.add_pipe("tagger", last=True, name="custom_tagger")

@custom_nlp.annotators["custom_tagger"]
def custom_tagger(doc):
    for token in doc:
        if token.text.startswith("#"):
            token.pos_ = "HASHTAG"
    return doc

text = "I love #nlp!"
doc = custom_nlp(text)
```

Get to work... again :-)

1. Briefly describe your understanding of POS tagging and its possible use-cases in context of text generation applications/language modeling.

Your answer here!

2. Train a UnigramTagger (NLTK) using the Brown corpus.

Hint: the taggers in nltk require a list of sentences containing tagged words.

3. Use this tagger to tag the text given below. Print out the POS tags for all variants of "justify"

```
text = """
Imagine a situation where you have to explain why you did something – that's when you justify your actions. So, let's say you made a decision; you, as the justifier, need to give good reasons (justifications) for your choice. You might use justifying words to make your point clear and reasonable. Justifying can be a bit like saying, "Here's why I did what I did." When you justify things, you're basically providing the why behind your actions. So, being a good justifier involves carefully explaining, giving reasons, and making sure others understand your choices
"""
```

4. Your results may be disappointing. Repeat the same task as above using both the default NLTK pos-tagger and with spaCy. Compare the results

```
# TODO: use the default NLTK tagger
```

MagicPython

```
# TODO: use spacy to fetch pos tags from the document
```

MagicPython

5. Finally, explore more features of the what the spaCy *document* includes related to topics covered in this lab.