NLP Tuts

Week 1

- What is tokenisation and why is it important?
  - Tokenisation is the act of transforming a (long) document into a set of meaningful substrings, so that we can compare with other (long) documents.
  - In general, a document is too long — and contains too much information — to manipulate directly. There are some counter-examples, like language identification, which we need to perform before we decide how to tokenise anyway.

- What are stemming and lemmatisation, and how are they different?
  - Both stemming and lemmatisation are mechanisms for transforming a token into a canonical (base, normalised) form. For example, turning the token walking into its base form walk.
  - Both operate by applying a series of rewrite operations to remove or replace (parts of) affixes (primarily suffixes). (In English, anyway.)
  - However, **lemmatisation works in conjunction with a lexicon**: a list of valid words in the language. The goal is to turn the input token into an element of this list (a valid word) using the rewrite rules. If the re-write rules can't be used to transform the token into a valid word, then the token is left alone. (For example, the token lemming wouldn't be transformed into lemm because the latter isn't in the word list.)
  - **Stemming simply applies the rewrite rules, even if the output is a garbage token (like lemm).**
  - One further idea is the difference between inflectional morphology and derivational morphology:
    - Inflectional morphology is the systematic process (in many but not all languages) by which tokens are altered to conform to certain grammatical constraints: for example, if the English noun teacher is plural, then it must be represented as teachers. The idea is that these changes don't really alter the meaning of the term. Consequently, **both stemming and lemmatisation attempt to remove this kind of morphology**.
    - Derivational morphology is the (semi-)systematic process by which we transform terms of one class into a different class. For example, if we would like to make the English verb teach into a noun (someone who performs the

action of teaching), then it must be represented as teacher. This kind of morphology tends to produce terms that differ (perhaps subtly) in meaning, and the two separate forms are usually both listed in the lexicon. Consequently, **lemmatisation doesn't usually remove derivational morphology in its normalisation process, but stemming usually does.**

Week 2

i. What is the continuation probability of `chuck` and `wood`?

- unique context words before `chuck` = {`wood`, `would`, `could`, `chuck`}
- unique context words before `wood` = {`the`, `much`, `a`, `chuck`}

$$P_{cont}(\texttt{chuck}) = \frac{\#_{cont}(\texttt{chuck})}{\#_{cont}(\texttt{a}) + \ldots + \#_{cont}(\texttt{</s>}) + \#_{cont}(\texttt{chuck}) + \#_{cont}(\texttt{wood})} = \frac{4}{2+1+1+0+1+1+1+2+1+4+4}$$

$$P_{cont}(\texttt{wood}) = \frac{\#_{cont}(\texttt{wood})}{\#_{cont}(\texttt{a}) + \ldots + \#_{cont}(\texttt{</s>}) + \#_{cont}(\texttt{chuck}) + \#_{cont}(\texttt{wood})} = \frac{4}{2+1+1+0+1+1+1+2+1+4+4}$$

Week 3

- What is a **POS tag**?
  - A POS tag, AKA word classes, is a label assigned to a word token in a sentence which indicates some grammatical (primarily syntactic) properties of its function in the sentence.
- What are some common approaches to POS tagging? What aspects of the data might allow us to predict POS tags systematically?
  - **Unigram**: Assign a POS tag to a token according to the most common observation in a tagged corpus; many words are unambiguous, or almost unambiguous.
  - **N-gram**: Assign a POS tag to a token according to the most common tag in the same sequence (based on the sentence in which the token occurs) of n tokens (or tags) in the tagged corpus; context helps disambiguate.
  - **Rule-based**: Write rules (relying on expertise of the writer) that disambiguate unigram tags.
  - **Sequential**: Learn a Hidden Markov Model (or other model) based on the observed tag sequences in a tagged corpus.
  - **Classifier**: Treat as a supervised machine learning problem, with tags from a tagged corpus as training data.

- What are the **assumptions that go into a Hidden Markov Model**? What is the time complexity of the **Viterbi algorithm**? Is this practical?
    - **Markov assumption**: the likelihood of transitioning into a given state depends only on the current state, and not the previous state(s) (or output(s))
    - **Output independence assumption**: the likelihood of a state producing a certain word (as output) does not depend on the preceding (or following) state(s) (or output(s)).
    - The time complexity of the Viterbi algorithm, for an HMM with T possible states, and a sentence of length W, is O(T2W). In POS tagging, there might typically be approximately 100 possible tags (states), and a typical sentence might have 10 or so tokens, so ... yes, it is practical (unless we need to tag really, really, quickly, e.g. tweets as they are posted).
- how can the **initial state probabilities** π be esitmated
    - Record the distribution of tags for the first token of each sentence in a tagged corpus.
- How can the **transition probabilities** A be estimated?
    - For each tag, record the distribution of tags of the immediately following token in the tagged corpus. (We might need to introduce an end–of–sentence dummy for the probabilities to add up correctly.)
- How can the **emission probabilities** B be estimated?
    - For each tag, record the distribution of corresponding tokens in the tagged corpus.

```
1. silver-JJ wheels-NNS turn-VBP
2. wheels-NNS turn-VBP right-JJ
3. right-JJ wheels-NNS turn-VBP
```

- For π, there are three sentences: two begin with `JJ` and one with `NNS`. Consequently:

$$\pi[JJ, NNS, VBP] = [\frac{2}{3}, \frac{1}{3}, 0]$$

- For A, we need to observe the distribution for each tag.
  - For JJ, two instances are immediately followed by NNS.
  - For NNS, all three instance are followed by VBP.
  - For VBP, it is followed by JJ once.

A	JJ	NNS	VBP
(from) JJ	0	1	0
NNS	0	0	1
VBP	1	0	0

- For B, we can simply read off the corresponding words for each tag in the corpus:
  - For JJ, there is one instance of silver and two of right.
  - For NNS, there are three instances of wheels.
  - For VBP, there are three instances of turn.
- Consequently:

B	right	silver	turn	wheels
(from) JJ	$\frac{2}{3}$	$\frac{1}{3}$	0	0
NNS	0	0	0	1
VBP	0	0	1	0

3. Consider using the following Hidden Markov Model to tag the sentence silver wheels turn:
   $\pi[\text{JJ}, \text{NNS}, \text{VBP}] = [0.3, 0.4, 0.3]$

A	JJ	NNS	VBP
JJ	0.4	0.5	0.1
NNS	0.1	0.4	0.5
VBP	0.4	0.5	0.1

B	silver	wheels	turn
JJ	0.8	0.1	0.1
NNS	0.3	0.4	0.3
VBP	0.1	0.3	0.6

   (a) Visualise the HMM as a graph.

   (b) Use the **Viterbi algorithm** to find the most likely tag for this sequence.

   - The most likely tag sequence can be read right-to-left, based upon the maximum probability we've observed: in this case, 0.0144 when *turn* is a VBP; this value is derived from the NNS → VBP transition, so we can infer that *wheels* is an NNS; that in turn comes from the JJ → NNS transition, so *silver* is a JJ.

α	1:*silver*		2:*wheels*	3:*turn*
JJ:	JJ	$\pi[\text{JJ}]B[\text{JJ}, \text{silver}]$ $0.3 \times 0.8 = 0.24$		
NNS:	NNS	$\pi[\text{NNS}]B[\text{NNS}, \text{silver}]$ $0.4 \times 0.3 = 0.12$		
VBP:	VBP	$\pi[\text{VBP}]B[\text{VBP}, \text{silver}]$ $0.3 \times 0.1 = 0.03$		

α	*1:silver*	*2:wheels*		*3:turn*
JJ:	0.24	JJ → JJ	A[JJ,JJ]B[JJ, wheels]	
	0.24		$\times 0.4 \times 0.1 = \mathbf{0.0096}$	
		NNS → JJ	A[NNS,JJ]B[JJ, wheels]	
		0.12	$\times 0.1 \times 0.1 = 0.0012$	
		VBP → JJ	A[VBP,JJ]B[JJ, wheels]	
		0.03	$\times 0.4 \times 0.1 = 0.0012$	
NNS:	0.12	JJ → NNS	A[JJ,NNS]B[NNS, wheels]	
	0.24		$\times 0.5 \times 0.4 = \mathbf{0.048}$	
		NNS → NNS	A[NNS,NNS]B[NNS, wheels]	
		0.12	$\times 0.4 \times 0.4 = 0.0192$	
		VBP → NNS	A[VBP,NNS]B[NNS, wheels]	
		0.03	$\times 0.5 \times 0.4 = 0.006$	
VBP:	0.03	JJ → VBP	A[JJ,VBP]B[VBP, wheels]	
	0.24		$\times 0.1 \times 0.3 = 0.0072$	
		NNS → VBP	A[NNS,VBP]B[VBP, wheels]	
		0.12	$\times 0.5 \times 0.3 = \mathbf{0.018}$	
		VBP → VBP	A[VBP,VBP]B[VBP, wheels]	
		0.03	$\times 0.1 \times 0.3 = 0.0009$	

Week 4

- How does a neural network language model (feedforward or recurrent) **handle a large vocabulary**, and how does it **deal with sparsity** (i.e. unseen sequences of words)?
    - A neural language model projects words into a continuous space and **represents each word as a low dimensional vector** known as **word embeddings**. These word embeddings **capture semantic and syntactic relationships between words**, allowing the model to generalise better to unseen sequences of words.
    - For example, having seen the sentence the cat is walking in the bedroom in the training corpus, the model should understand that a dog was running in a room is just as likely, as (the, a), (dog, cat), (walking, running) have similar semantic/ grammatical roles.
    - Count-based N-gram language model would struggle in this case, as (the, a) or (dog, cat) are distinct word types from the model's perspective.

Why do we say most parameters of a neural network (feedforward or recurrent) language model is in their input and output word embeddings?

- Assume we have a vocabulary size of 10K word types, and we are using the feedforward neural network language model in the lecture (L7 page 21). If the dimension of the embeddings and the hidden representation (h_1) is 300, the number of parameters in our model is:
  - input word embeddings $= 300 \times 10K = 3,000,000$
  - $W_1 = 300 \times 600 = 180,000$
  - $b_1 = 300$
  - output word embeddings (W_2) $= 10K \times 300 = 3,000,000$
- Similarly for a simple recurrent neural language model (L8 page 14):
  - input word embeddings $W_x = 300 \times 10K = 3,000,000$
  - $W_s = 300 \times 300 = 90,000$
  - $b = 300$
  - output word embeddings (W_y) $= 10K \times 300 = 3,000,000$

4. What is the vanishing gradient problem in RNN, and what causes it? How do we tackle vanishing gradient for RNN?

   - An unrolled RNN is just a very deep network.
   - When we do backpropagation to compute the gradients, the gradients tend to get smaller and smaller as we move backward through the network.
   - The net effect is that neurons in the earlier layers learn very slowly, as their gradients are very small. If the unrolled RNN is deep enough we might start seeing gradients "vanish".
   - More sophisticated RNN models such as LSTM or GRU are introduced to tackle vanishing gradients.
   - They do so by introducing "memory cells" that preserve gradients across time.

Week 5

- **Synonyms and hypernyms**
  - Two words are synonyms when they share (mostly) the same meaning, for example: snake and serpent are synonyms.
  - One word is a hypernym of a second word when it is a more general instance ("higher up" in the hierarchy) of the latter, for example, reptile is the hypernym of snake (in its animal sense).
- **Hyponyms and meronyms**
  - One word is a hyponym of a second word when it is a more specific instance ("lower down" in the hierarchy) of the latter, for example, snake is one hyponym of reptile. (The opposite of hypernymy.)

- One word is a meronym of a second word when it is a part of the whole defined by the latter, for example, scales (the skin structure) is a meronym of reptile.

  - The word *information* has five different senses in Wordnet; I've reproduce the fragment of the hierarchy above these senses below:

entity abstraction... communication message...	entity abstraction... psychological... cognition...	entity abstraction... communication message... statement pleading charge... accusation...	entity abstraction... group... collection...	entity abstraction... measure system of meas... information meas...

information

- Here's the corresponding fragment of the three senses above `retrieval`:

entity physical... process... processing data process... operation computer op...	entity abstraction... psychological... cognition... process... basic cog... memory...	entity abstraction... psychological... event act...

retrieval

- what is word sense disambiguation
  - Word sense disambiguation is the computational problem of automatically determining which sense (usually, Wordnet synset) of a word is intended for a given token instance with a document.

4. For the following term co-occurrence matrix (suitably interpreted):

	cup	not(cup)
world	55	225
not(world)	315	1405

(a) Find the Point-wise Mutual Information (PMI) between these two terms in this collection.

- To evaluate PMI, we need the joint and prior probabilities of the two event (in this case, probably w: document contains `world` and c: document contains `cup`.
- We estimate these based on their appearance out of the total number of instances in the collection (2000), and then substitute:

$$\begin{aligned} P(w) &= 280/2000 = 0.14 \\ P(c) &= 370/2000 = 0.185 \\ P(w,c) &= 55/2000 = 0.0275 \\ PMI(w,c) &= \log_2 \frac{P(w,c)}{P(w)P(c)} \\ &= \log_2 \frac{0.0275}{0.14 \times 0.185} \\ &\approx 0.0865 \end{aligned}$$

(b) What does the value from (a) tell us about **distributional similarity**?

- This value is slightly positive, which means that the two events occur together (in documents) slightly more commonly than would occur purely by chance. There is some possibility that `world` and `cup` occurring together is somehow meaningful for documents in this collection.

- What is the **Singular Value Decomposition (SVD)** method used for here? Why is this helpful?
    - We are using the SVD method to build a representation of our matrix which can be used to identify the most important characteristics of words.
    - By throwing away the less important characteristics, we can have a smaller representation of the words, which will save us (potentially a great deal of) time when evaluating the cosine similarities between word pairs.

- What is a **word embedding** and how does it relate to **distributional hypothesis**?
    - We're going to have a representation of words (based on their contexts) in a vector space, such that other words "nearby" in the space are similar
    - This is broadly the same as what we expect in distributional similarity ("you shall know a word by the company it keeps.")
    - The row corresponding to the word in the relevant (target/context) matrix is known as the "embedding".

Week 6

- What are **contextual representations**?
    - The contextual representation of a word is a representation of the word based on a particular usage. It captures the different senses or nuances of the word depending on the context.
    - Contextualised representations are different to word embeddings (e.g.Word2Vec) which gives a single representation for every word type.
    - Contextual representations that are pre-trained on large data can be seen as a model that has obtained fairly comprehensive knowledge about the language.
- How does a **transformer** captures dependencies between words? What advantages does it have compared to RNN?
    - Transformer uses **attention** to capture dependencies between words.
    - For each target word in a sentence, transformer attends to every other words in the sentence to create its contextual embeddings.
    - As the computation of the contextual embeddings of a target word is independent to other target words, we can **parallelise the computation**. This is an important distinction to RNN, which rely on sequential processing: the contextual embedding of the current target word cannot be computed until we have processed the previous word.
    - This allows transformer-based models to scale to very large data that is difficult for RNN-based models.
- What is **discourse segmentation**? What do the segments consist of, and what are some methods we can use to find them?
    - In Discourse Segmentation, we try to divide up a text into discrete, cohesive units based on sentences.
    - By interpretting the task as a boundary–finding problem, we can use rule– based or unsupervised methods to find sentences with little lexical overlap (suggesting a discourse boundary). We can also use supervised methods, by training a classifier around paragraph boundaries.

Week 7

- What are **regular grammar** and **regular language**? How are they different?
    - A language is a set of acceptable strings and a grammar is a **generative description** of a language.
    - Regular language is a formal language that can be expressed using a regular expression.
    - Regular grammar is a formal grammar defined by a set of productions rules in the

form of A→xB, A→x and A→ ϵ , where A and B are non-terminals, x is a terminal and ϵ is the empty string.

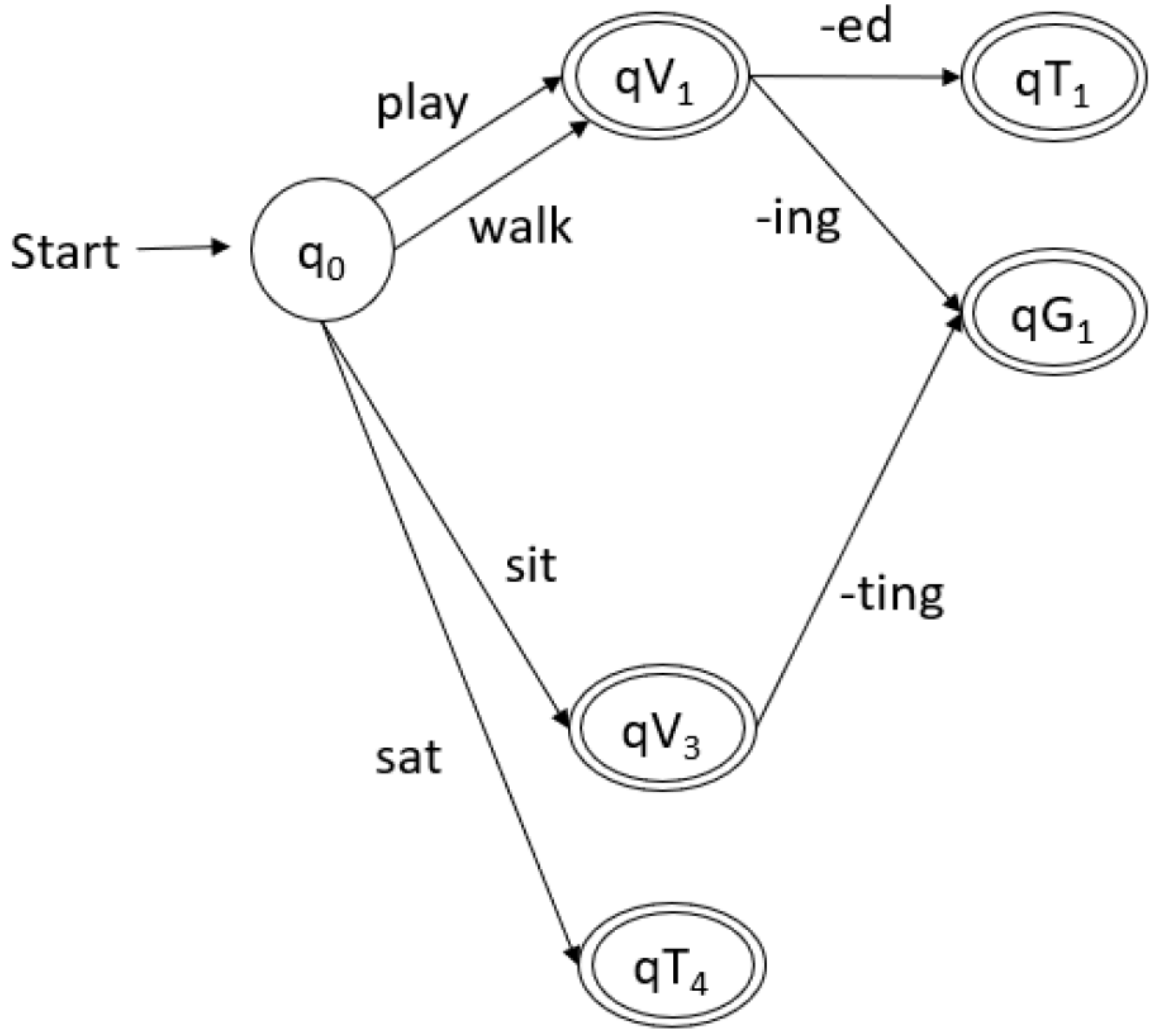
- A language is regular if and only if it can be generated by a regular grammar. For example: A simple regular grammar
  - Rules: S→ A, A→ aA, A→ ϵ
  - S is the start symbol
  - It will generate words such as a, aa, aaa, aaa.
  - The set of words generated by this regular grammar is a regular language.
  - This regular language **can also be expressed in regular expression** (a)*.

- Regular languages are

$$closedunderunion, intersectionandconcatenation$$

. What does it mean? Why is it important?

  - This means that if L1 and L2 are two regular languages, then L1 [L2, L1 \ L2, and the language strings that are the concatenation of L1 and L2 are also regular languages.
  - This closure property allows us to apply operations on regular languages to produce a regular language.
  - This allows for NLP problems to be factored into small simple parts, such that we can **develop regular languages for each part, and combine them into a complex system to handle the NLP problems**. This is particularly relevant for transducers and the composition operation, which are used in many NLP pipelines. (Note that FSTs implement “regular relations” rather that regular languages.



- What are **Weighted Finite State Acceptors (WFSAs)**? When and why are they useful?
    - WFSAs are **generalizations of FSAs**, with each path assigned a score, computed from the transitions, the initial state, and the final state. The total score for any path is equal to the sum of the scores of the initial state, the transitions, and the final state.
    - WFSAs can produce a score for each valid word for sub-word decomposition problems or sentence for words-in-sentence problems, while FSAs have no way to express preferences among technically valid words or sentences.
    - For example, WFSAs can assign scores to all strings of characters forming words, so that spelling mistakes, new words, or strange-but-acceptable words can still be handled.
    - The same argument holds for sequences of words forming sentences. Clearly some word sequences are gibberish, but being able to provide a numerical score can help in many applications, like how LMs can be used in sentence generation, speech recognition, OCR, translation, etc.

- what is parsing
    - Parsing in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.

Week 8

- What differentiates **probabilistic CYK parsing from CYK parsing**? Why is this important? How does this affect the algorithms used for parsing?

    - Parsing in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.
    - In general, the search space is too large to do this efficiently, so we use a dynamic programming method to keep track of partial solutions. The data structure we use for this is a chart in CYK, where entries in the chart correspond to partial parses (licensed structures) for various spans (sequences of tokens) within the sentence. **Probabilistic CYK parsing adds real-valued weights (probability) to each production in the grammar, such that parse trees can be assigned a score, namely the product of the probabilities of the productions in the tree**. This is important as it allows for discrimination between likely and unlikely parses, rather than just provide a list of all parses, as in standard CYK parsing. This affects the algorithms as they need to track the maximum probability analysis for each span, rather than the set of grammar symbols. However the parsing algorithms are very closely related.

- What is a **probabilistic context-free grammar** and what problem does it attempt to solve?

    - A probabilistic context-free grammar adds real-valued weights (probability) to each production in the context-free grammar. This attempts to provide a “language model”, that is, describe the likely sentences in a language, which is facilitated by their grammatical structure.

- In what ways is (transition–based, probabilistic) **dependency parsing similar to (probabilistic) CYK parsing**? In what ways is it different?

    - The connections are a little tenuous, but let’s see what we can come up with:
        - Both methods are attempting to determine the structure of a sentence;
        - both methods are attempting to disambiguate amongst the (perhaps many) possible structures licensed by the grammar by using a probabilistic grammar
        - to determine the most probable structure.
        - Both methods process the tokens in the sentence one–by–one, left–to– right.

    - There are numerous differences (probably too many to enumerate here), for

example:

- Although POS tags are implicitly used in constructing the “oracle” (training), the depedency parser doesn’t explicitly tag the sentence.
- The dependency parser can potentially take into account other (non–local) relations in the sentence, whereas CYK’s probabilities depend only on the (local) sub-tree.
- CYK adds numerous fragments to the chart, which don’t end up getting used in the final parse structure, whereas the transition–based dependency parser only adds edges that will be in the final structure.

Week 9

- What aspects of human language make **automatic translation difficult**?
    - The whole gamut of linguistics, from lexical complexity, morphology, syntax, semantics etc. In particular if the two languages have very different word forms (e.g., consider translating from an morphologically light language like English into Turkish, which has very complex morphology), or very different syntax, leading to different word order. These raise difficult learning problems for a translation system, which needs to capture these differences in order to learn translations from bi-texts, and produce these for test examples.
- What is **Named Entity Recognition** and why is it difficult? What might make it more difficult for persons rather than places, and vice versa?
    - We want to find named entities — mostly proper nouns, but also sometimes times or numerical values of significance — within a document. Often context (which is not always present within the sentence, or even the document!) is needed to disambiguate the type of named entity, or even whether a given phrase is a named entity at all.
    - One common problem, that we see with both people’s names and places, is that they are ambiguous with common nouns. Generally speaking, we can write a (somewhat) exhaustive list of names of places — a gazetteer — but we can’t with names of people, which are constantly changing. On the other hand, locations can also be quite ambiguous, e.g. “New York” in “New York Times” is less of a location than an organisation.

Week 10

- What is **semantic parsing**, and why might it be desirable for QA? Why might approaches like NER be more desirable?

- As opposed to syntactic parsing — which attempts to define the structural relationship between elements of a sentence — we instead want to define the (meaning–based) relations between those elements.
- For example, in the sentence Donald Trump is president of the United States. we can deduce that Donald Trump is the subject of the verb is, and so on, but in semantic parsing, we might be trying to generate a logical relationship like is(Donald Trump, president(United States)).
- This format allows us to answer questions like “Who is president of the United States?” by generating an equivalent representation like: is(?,president(United States))

Week 11

(b) What is the log likelihood ratio of word w in document d if: $F_d(w) = 1$, $F_b(w) = 20$, $N_d = 30$, and $N_b = 4000$, where b denotes the background corpus, F the frequency and N the total number of word tokens.

$$
\begin{aligned}
p &= \frac{1+20}{30+4000} \\
p_d &= \frac{1}{30} \\
p_b &= \frac{20}{4000} = \frac{1}{200}
\end{aligned}
$$

As $p_d = \frac{1}{30}$ which is greater than $p_b = \frac{1}{200}$, this indicates that w is a word that is rather prominent (i.e. it is statistically more probable in the document than it typically is), but is it *salient* under its log likelihood ratio?

$$
\begin{aligned}
\text{LLR}(w) &= -2 \times \log \left(\frac{\binom{30}{1} p^1 (1-p)^{29} \times \binom{4000}{20} p^{20} (1-p)^{3980}}{\binom{30}{1} p_d^1 (1-p_d)^{29} \times \binom{4000}{20} p_b^{20} (1-p_b)^{3980}} \right) \\
&= 2.08
\end{aligned}
$$