**Semantic Text Similarity:**

**Take Home Concepts:**

1. Finding similarity between words and text is non-trivial.
2. WordNet is a useful resource for semantic relationships between words.
3. NLTK is a useful package for many such tasks.

Definition**:** **Semantic** – the study of relationships of words and how they draw meaning from the words around them.

E.g. which pair of words are most similar: deer and elk, deer and giraffe, deer and horse, deer and mouse? Of course, we think that deer and elk are the most similar, but how? We need to use semantic similarities.

**Applications of Semantic Similarity:**

Semantic similarity is when you’re grouping words that have similar meaning together. These concepts can be very useful for tasks such as **paraphrasing** and **textual entailment**.

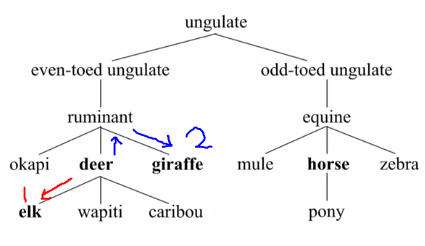
**Paraphrasing** is when a block of text has the same meaning as another block of text but with different order and words used.

**Textual** **entailment** is when a sentence or part of a sentence derives its meaning from the text before it, i.e. the 2 sentences before it.

**WordNet:**

This is a semantic dictionary with links between other relevant words. This dictionary includes lots of linguistic information e.g. POS, word senses (different meanings of the same word), synonyms, hypernyms, etc… WordNet is also machine readable and free to use, so it extensively used in industry.

WordNet organises information in a tree structure of hierarchy. E.g. for our deer example:



One way to measure how similar these words are to one another is to use **Path Similarity**.We first need to find the shortest path to the related words; the similarity is then the inverse of this distance.

PathSim(deer, elk) = 1/( 1 + 1) = 0.5

PathSim(deer, giraffe) = 1/ (2 + 1) = 0.33

PathSim(deer, horse) = 1/(6+1) = 0.143

Another way of doing this is to use a method called **Lowest Common Subsumer (LCS)**. This finds the lowest **ancestor** that relates the word to the other word. An ancestor is the branch in the hierarchy tree. E.g.

LCS(deer, elk) = deer

LCS(deer, giraffe) = ruminant

LCS(deer, horse) = ungulate

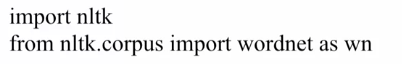
**Lin Similarity:**

This is a similarity measure base on the information contained in the LCS of the two words. This looks at how interchangeable two words are given the method is trained on the text provided.



Where P(LCS(u,v)) is the probability of the lowest common subsumer, where u and v are the words being compared. The probabilities are calculated over a large corpus of words.

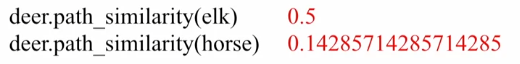
**Python:**



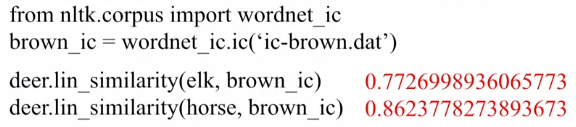
Then we find the appropriate sense of the words:



Where “deer.n.01” means find the word deer which is a noun (n) and the first one (01). Once you have the correct sense of the word you could use it to find similarity between them:



For **Lin Similarity:**



We can see that the **Lin Similarity** It thinks that horse and deer are more similar even though the distance between them in the hierarchy tree is much larger. This is because the words deer and horse are used in a similar context more often, so it thinks that the words can be interchanged with the least loss of information.

**Collocations and distributional Similarity:**

“You know a word by the company it keeps”, meaning that two words that are frequently appearing in similar contexts are more likely to be similar (semantically related). E.g.:

* The friends met at a **café**.
* Amy met Ray at a **pizzeria**.
* Let’s meet up near the **coffee shop**.
* The secret meeting at the **restaurant** soon became public.

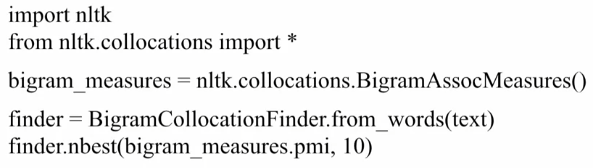
Here the highlighted words could be easily interchanged, and the meaning would hardly change.

**Distributional Similarity** is based on the context of the word being examined. This is done by looking at the words before, after, and within a small window of that word. E.g. the words before are all “a” or “the”. We could also use POS of the words before, after, and within a small window of the word.

Once you have found this context you can compute the strength of association between words based on how frequently the words occur together. If you find two words that commonly come together in a document of text, you would want to say they collocate well together. Its also important to see how frequently words occur, for example, the word “the” will occur many times with other words but we’re not very interested in this relationship. We can use **Pointwise Mutual Information (PMI)** to ensure that these words aren’t prioritized.



**Python:**



Bigrams is used to check for important pairs of words, then use from\_word(…) to learn the text we want to work on. Then find the 10 best words using PMI.

We could also use this finder to filter out bigrams that don’t appear a certain number of times.

