Summary of Wordnet: Wordnet is a database of words split by parts of speech and similarity. It splits words into synsets based on like words. I can see that it is very useful for NLP applications when a computer guesses what realm of things a particular text input is.

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('omw-1.4')
nltk.download('book')
nltk.download('sentiwordnet')
```

```
    [nltk_data]    | Downloading package senseval to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/senseval.zip.
    [nltk_data]    | Downloading package state_union to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/state_union.zip.
    [nltk_data]    | Downloading package stopwords to /root/nltk_data...
    [nltk_data]    |   Package stopwords is already up-to-date!
    [nltk_data]    | Downloading package swadesh to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/swadesh.zip.
    [nltk_data]    | Downloading package timit to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/timit.zip.
    [nltk_data]    | Downloading package treebank to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/treebank.zip.
    [nltk_data]    | Downloading package toolbox to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/toolbox.zip.
    [nltk_data]    | Downloading package udhr to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/udhr.zip.
    [nltk_data]    | Downloading package udhr2 to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/udhr2.zip.
    [nltk_data]    | Downloading package unicode_samples to
    [nltk_data]    |     /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/unicode_samples.zip.
    [nltk_data]    | Downloading package webtext to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/webtext.zip.
    [nltk_data]    | Downloading package wordnet to /root/nltk_data...
    [nltk_data]    |   Package wordnet is already up-to-date!
    [nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/wordnet_ic.zip.
    [nltk_data]    | Downloading package words to /root/nltk_data...
    [nltk_data]    |   Unzipping corpora/words.zip.
    [nltk_data]    | Downloading package maxent_treebank_pos_tagger to
    [nltk_data]    |     /root/nltk_data...
    [nltk_data]    |   Unzipping taggers/maxent_treebank_pos_tagger.zip.
    [nltk_data]    | Downloading package maxent_ne_chunker to
    [nltk_data]    |     /root/nltk_data...
    [nltk_data]    |   Unzipping chunkers/maxent_ne_chunker.zip.
    [nltk_data]    | Downloading package universal_tagset to
    [nltk_data]    |     /root/nltk_data...
    [nltk_data]    |   Unzipping taggers/universal_tagset.zip.
    [nltk_data]    | Downloading package punkt to /root/nltk_data...
    [nltk_data]    |   Package punkt is already up-to-date!
    [nltk_data]    | Downloading package book_grammars to
```

```
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/book_grammars.zip.
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/city_database.zip.
[nltk_data]    | Downloading package tagsets to /root/nltk_data...
[nltk_data]    |   Unzipping help/tagsets.zip.
[nltk_data]    | Downloading package panlex_swadesh to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    |
[nltk_data]  Done downloading collection book
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/sentiwordnet.zip.
True
```

```
from nltk.corpus import wordnet as wn
```

```
# output all synsets for the noun brick
wn.synsets('brick')
```

```
[Synset('brick.n.01'), Synset('brick.n.02')]
```

```
#3
print(wn.synset('brick.n.01').definition())
print(wn.synset('brick.n.01').examples())
print(wn.synset('brick.n.01').lemmas())

hyp = wn.synset('brick.n.01').hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
rectangular block of clay baked by the sun or in a kiln; used as a building or p
[]
[Lemma('brick.n.01.brick')]
Synset('building_material.n.01')
Synset('artifact.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Wordnet organizes its nouns in a heirarchial fashion where the top most level is entity and every level below it is a hyponym. As seen above the original noun is in the synset brick and each layer above is a hypernym. This structure makes a lot of logical sense when using the wordnet heirarchy

```
#4
brick = wn.synset('brick.n.01')
print('hypernyms: ', brick.hypernyms())
print('hyponyms: ', brick.hyponyms())
print('meronyms: ', [])
print('holonyms: ', [])
print('antonym: ', [])
```

```
hypernyms:  [Synset('building_material.n.01'), Synset('ceramic.n.01')]
hyponyms:  [Synset('adobe.n.02'), Synset('clinker.n.02'), Synset('firebrick.n.01
meronyms:  []
holonyms:  []
antonym:  []
```

```
#5
wn.synsets('slanting')
```

```
[Synset('slant.v.01'),
 Synset('slant.v.02'),
 Synset('lean.v.01'),
 Synset('cant.v.01'),
 Synset('aslant.s.01')]
```

```
#6
print(wn.synset('slant.v.01').definition())
print(wn.synset('slant.v.01').examples())
print(wn.synset('slant.v.01').lemmas())
print("Heirarchy of slant.v.01")
hyp = wn.synset('slant.v.01').hypernyms()[0]
hashset = set()
while hyp and hyp not in hashset:
    print(hyp)
    hashset.add(hyp)
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
lie obliquely
['A scar slanted across his face']
[Lemma('slant.v.01.slant')]
Heirarchy of slant.v.01
Synset('lie.v.01')
Synset('be.v.03')
```

Verbs in wordnet are also organized in a heirarchy, but not all verbs have a common hypernym. For instance above, the verb slant only has two hypernyms, lie, and be.

```
#7
wn.morphy('slanting', wn.VERB)
```

```
'slant'
```

```
#8
# run # jog
run = wn.synset('run.n.01')
jog = wn.synset('jog.n.01')
print('Wu-Palmer',wn.wup_similarity(run, jog))
```

```
0.23529411764705882
```

```
#8
from nltk.wsd import lesk
sentence = ['You', 'can', 'run', 'or', 'jog', 'but', 'you', 'cannot', 'hide', '.']
print(lesk(sentence, 'run', 'n'))
print(lesk(sentence, 'jog', 'n'))
```

```
Synset('run.n.10')
Synset('nudge.n.01')
```

SentiWordNet is a dialect of WordNet , but in SeniWordNet every word is assigned a score that says if the word is negative or positive. This can be used to find how positive or negative an sentence or paragraph of text is. As seen in class, this is a very useful tool but can show results that are not entirely accurate.

```
#9
from nltk.corpus import sentiwordnet as swn
depression = swn.senti_synset('depression.n.02')
print(depression)
print("Positive score = ", depression.pos_score())
print("Negative score = ", depression.neg_score())
print("Objective score = ", depression.obj_score())
```

```
<depression.n.02: PosScore=0.0 NegScore=0.25>
Positive score =  0.0
Negative score =  0.25
Objective score =  0.75
```

```
#9
sent = 'this was a very fun day'
tokens = sent.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        syn = syn_list[0]
        print(token)
        print('neg:', syn.neg_score(), " pos:", syn.pos_score(), " obj:" , syn.obj_sco
```

```
was
neg: 0.0  pos: 0.0  obj: 1.0
a
neg: 0.0  pos: 0.0  obj: 1.0
very
neg: 0.0  pos: 0.5  obj: 0.5
fun
neg: 0.0  pos: 0.375  obj: 0.625
day
neg: 0.0  pos: 0.0  obj: 1.0
```

It looks like the positive negative evaluations are fairly accurate for this sentence. I also see that all the scores are 1/(2^N). This seems very useful when evaluating what type of emotion a sentence is said with.

A collocation is a phrase containing multiple words which can be used on their own. It is really interesting because the meaning changes or becomes unclear when a like word or a synonym is used in place of one of the original words.

```
#10
from nltk.book import *
text4.collocations()
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
United States; fellow citizens; years ago; four years; Federal
Government; General Government; American people; Vice President; God
bless; Chief Justice; one another; fellow Americans; Old World;
Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
tribes; public debt; foreign nations
```

```
#10
text = ' '.join(text4.tokens)
text[:50]
```

```
import math
vocab = len(set(text4))
hg = text.count('fellow citizens')/vocab
print("p(fellow citizens) = ",hg )
h = text.count('fellow')/vocab
print("p(fellow) = ", h)
g = text.count('citizens')/vocab
print('p(citizens) = ', g)
pmi = math.log2(hg / (h * g))
print('pmi = ', pmi)
```

```
    p(fellow citizens) =  0.006084788029925187
    p(fellow) =  0.013665835411471322
    p(citizens) =  0.026932668329177057
    pmi =  4.0472042737811735
```

The mutual information formula essentially finds the relationship between two variables. In this case we can see the probability of fellow, the probability of citizens, and the probability of fellow citizens all occuring. This information is used to find the mutual information value which is 4.04. This is a fairly high values which means that the likelyhood of the word citizens following the word fellow is higher than normal.