CMPS 143: Intro to NLP
## HW4 Competition Results

<table>
<thead>
<tr>
<th>Part 1 Restaurant</th>
<th>Best Combination</th>
<th>Dev</th>
<th>Test</th>
<th>Heldout</th>
<th>Part 2 Fable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frodo</td>
<td>NB#32 (uni+bi+liwc+pos)</td>
<td>0.81</td>
<td>0.8</td>
<td>0.86</td>
<td>0.9063</td>
</tr>
<tr>
<td>Samwise</td>
<td>NB#28772(uni+bi+liwc+unipos)</td>
<td>0.82</td>
<td>0.84</td>
<td>0.8</td>
<td>0.875</td>
</tr>
<tr>
<td>Thranduil</td>
<td>DT(uni #15891)</td>
<td>0.81</td>
<td>0.67</td>
<td>0.79</td>
<td>0.7812</td>
</tr>
<tr>
<td>Eowyn</td>
<td>NB#64(uni+bi+liwc)</td>
<td>0.85</td>
<td>-</td>
<td>0.9063</td>
<td>0.5</td>
</tr>
<tr>
<td>Peregrin</td>
<td>NB#128(uni+bi+liwc+pos)</td>
<td>0.82</td>
<td>0.79</td>
<td>-</td>
<td>0.4583</td>
</tr>
<tr>
<td>Bilbo</td>
<td>NB#128(uni+liwc)</td>
<td>0.79</td>
<td>-</td>
<td>0.83</td>
<td>0.2</td>
</tr>
<tr>
<td>Aragorn</td>
<td>NB#64(uni+liwc+unipos)</td>
<td>0.76</td>
<td>0.78</td>
<td>0.77</td>
<td>0.7879</td>
</tr>
<tr>
<td>Legolas</td>
<td>NB#14194(bi+liwc+binning)</td>
<td>0.82</td>
<td>0.81</td>
<td>0.8</td>
<td>0.8333</td>
</tr>
<tr>
<td>Gimli</td>
<td>NB#128(uni+binning)</td>
<td>0.82</td>
<td>0.79</td>
<td>0.74</td>
<td>0.875</td>
</tr>
<tr>
<td>Arwen</td>
<td>NB#128(bi+liwc)</td>
<td>0.82</td>
<td>0.81</td>
<td>0.8</td>
<td>0.9063</td>
</tr>
<tr>
<td>Gollum</td>
<td>NB#128(uni+liwc)</td>
<td>0.81</td>
<td>0.79</td>
<td>0.83</td>
<td>0.9063</td>
</tr>
<tr>
<td>Saruman</td>
<td>NB#all(uni+liwc)</td>
<td>0.78</td>
<td>0.8</td>
<td>0.83</td>
<td>0.0625</td>
</tr>
<tr>
<td>Gandalf</td>
<td>NB#all(bi+liwc)</td>
<td>0.81</td>
<td>0.81</td>
<td>0.86</td>
<td>0.9375</td>
</tr>
<tr>
<td>Merry</td>
<td>NB#4096(uni+adjuni+liwc_bin)</td>
<td>0.83</td>
<td>0.85</td>
<td>0.84</td>
<td>0.9583</td>
</tr>
<tr>
<td>Elrond</td>
<td>NB#32(bi+liwc+binning)</td>
<td>0.82</td>
<td>0.81</td>
<td>0.8</td>
<td>0.8125</td>
</tr>
<tr>
<td>Galadriel</td>
<td>NB#64(uni+bi+liwc)</td>
<td>0.8</td>
<td>0.81</td>
<td>0.85</td>
<td>0.8125</td>
</tr>
<tr>
<td>Radagast</td>
<td></td>
<td>0.4688</td>
<td>0.833</td>
<td>0.6731</td>
<td></td>
</tr>
</tbody>
</table>
HW4: Multiply the 20 points by 5.
Grades (no HW5, no curve, A+ 2add)

<table>
<thead>
<tr>
<th>Name</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Median</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course Grade</td>
<td>79.37</td>
<td>17.74</td>
<td>83.46</td>
<td>95.08</td>
</tr>
<tr>
<td>Homework 5: Midterm R…</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Homework 5B: Scheher…</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Homework 3</td>
<td>7.38</td>
<td>2.89</td>
<td>8.80</td>
<td>10.00</td>
</tr>
<tr>
<td>Homework 2</td>
<td>15.28</td>
<td>4.36</td>
<td>16.55</td>
<td>16.10, 19.20, 18.90</td>
</tr>
<tr>
<td>Homework 4</td>
<td>15.98</td>
<td>5.30</td>
<td>18.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Entry Quiz: Homework 0</td>
<td>5.00</td>
<td>0.00</td>
<td>5.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Homework 1</td>
<td>8.30</td>
<td>2.01</td>
<td>9.00</td>
<td>9.80</td>
</tr>
</tbody>
</table>
CMPS 143: Intro to NLP
Midterm Review
Thurs right here
Bring a PINK Scantron
Any particular questions?

- From Homework 5 the midterm review?

- Solutions posted, can pull them up.
NLP Pipeline
NLP PIPELINE

WORDS
(MORPHOLOGY)

PATTERNS OF WORDS
(DISTRIBUTIONAL ANALYSIS, LEXICAL SEMANTICS)

PHRASES AND SENTENCES
(SYNTAX)

CLASSIFYING TEXTS
(SEMANTICS)

SENTENCE MEANING
(SEMANTICS)

DISCOURSE MEANING NARRATIVE STRUCTURES
(SEMANTICS, PRAGMATICS, DISCOURSE)
Regular Expressions
Regular Expressions (Regexes)

- Many linguistic processing tasks involve pattern matching.
- Regular Expressions are a compact textual representation of a set of strings representing a language
  - Classically, they represent *regular* languages
  - Most regex interpreters have additional features
- To use regular expressions in Python we need to import the `re` library using: `import re`
Regexes as Finite State Acceptors

- We’ll focus on sheep talk...
  - /baa+/!
    - Note the convention:
      - Regex delimited by forward slashes
    - Other things to note:
      - + is a quantifier (one or more of the symbol immediately on the left)
  - A graph in this sense is simply:
    - A set of nodes
    - A set of edges connecting them
    - Edges can have labels
Recognition Algorithm

- Start in the start state
- Examine the current input
- Consult the table to identify the next state
- Go to the new state and update the pointer.
- ...Until you run out of input symbols.
Another FSA example

- \(((\text{chicka})|(\text{wicka}))(\text{chicka})|(\text{wicka}))(\text{boom})^*(\text{zoom})\)
- \(((\text{chicka})|(\text{wicka}))^2(\text{boom})^*(\text{zoom})\)
Constructing Regexes

- Often need to refine the pattern to fix:
  - **False positives** (Type I errors)
    - Strings we should not have matched but did: *thespian, atheist, etc.*
  - **False negatives** (Type II errors)
    - Strings we should have matched but didn’t: *The*
## Regex Meta-Characters

<table>
<thead>
<tr>
<th>Operator</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Wildcard, matches any character</td>
</tr>
<tr>
<td>^abc</td>
<td>Matches some pattern (abc) at the start of a string</td>
</tr>
<tr>
<td>$abc$</td>
<td>Matches some pattern (abc) at the end of a string</td>
</tr>
<tr>
<td>[abc]</td>
<td>Matches one of a set of characters</td>
</tr>
<tr>
<td>[A-Z0-9]</td>
<td>Matches one of a range of characters</td>
</tr>
<tr>
<td>ed</td>
<td>ing</td>
</tr>
<tr>
<td>*</td>
<td>Zero or more of previous item, e.g. (a^+), ([a-z]^+) (also known as Kleene Closure)</td>
</tr>
<tr>
<td>+</td>
<td>One or more of previous item, e.g. (a^+), ([a-z]^+)</td>
</tr>
<tr>
<td>?</td>
<td>Zero or one of the previous item (i.e. optional), e.g. (a^?), ([a-z]^?)</td>
</tr>
<tr>
<td>{n}</td>
<td>Exactly (n) repeats where (n) is a non-negative integer</td>
</tr>
<tr>
<td>{n,}</td>
<td>At least (n) repeats</td>
</tr>
<tr>
<td>{n,}</td>
<td>No more than (n) repeats</td>
</tr>
<tr>
<td>{m,}</td>
<td>At least (m) and no more than (n) repeats</td>
</tr>
<tr>
<td>a(b</td>
<td>c)+</td>
</tr>
</tbody>
</table>
### Homework “Ing”

```python
>>> wordlist=["farming", "fishing", "run", "dining", "smile"]
>>> print([w for w in wordlist if re.search(r'ing$', w)])
['farming', 'fishing', 'dining']
```

I think it will return only one last word that match "ing" if we feed re.search() a sentence. However, we feed it a word w, so its find.

Counter Example: If it's a sentence, it only return the last "ing" due to the "$\)$. But that's not the question.

```python
>>> match= re.search(r'ing$', "I am going to fishing")
>>> if match: print(match.group())
... ing
```
import nltk
>>> import random
>>> from nltk.corpus import brown

>>> hobbies_learned = nltk.Text(brown.words(categories=['hobbies', 'learned']))

>>> hobbies_learned.findall(r"<\w*> and<\w*> such as<\w*,>")

>>> hobbies_learned.findall(r"<\w*> such as<\w*>")

speed and other activities; water and other liquids; tomb and other landmarks; Statues and other monuments; pearls and other jewels; charts and other items; roads and other features; figures and other objects; military and other areas; demands and other factors; abstracts and other compilations; iron and other metals

powerplants such as these; boats such as outboards; attractions such as the; features such as a; values such as ancient; stress such as shipping; conditions such as vaccinating; organizations such as American; subjects such as architecture; trappings such as private; manufacturers such as Advance; pastes such as printing; impurities
Words & Frequencies
Morphology

- Words can have compositional meaning from their parts
- Morphology is the study of the internal structure of words, of the way words are built up from smaller meaning units.
- Morpheme:
  - The smallest meaningful unit in the grammar of a language.
- Two classes of morphemes
  - Stems: “main” morpheme of the word, supplying the main meaning (i.e. establish in the example below)
  - Affixes: add additional meaning
    - Prefixes: Antidisestablishmentarianism
    - Suffixes: Antidisestablishmentarianism
    - Infixed: hingi (borrow) – humingi (borrower) in Tagalog
    - Circumfixes: sagen (say) – gesagt (said) in German
Stemming

- The removal of the inflectional ending from words (strip off any affixes)
  - Laughing, laugh, laughs, laughed $\rightarrow$ laugh
- Problems
  - Can conflate semantically different words
    - Gallery and gall may both be stemmed to gall
- A further step is to make sure that the resulting form is a known word in a dictionary, a task known as lemmatization.
Is Stemming Useful

- For information retrieval, some improvement for smaller documents
  - Helps a lot for some queries, hurts a lot in other cases

- Mixed results for language modeling

- Problems
  - Word sense disambiguation on query terms: *business* may be stemmed to *busy*, *saw* (the tool) *to see*
  - A truncated stem can be unintelligible to users

- **However**, finding the root word (lemma) may be necessary to use lexical resources
Type / Token Distinction

- **token** = any word in the corpus
  - # tokens is an estimate of the corpus size

- **type** = unique representatives of the tokens
  - # types is an estimate of the vocabulary size

**Example:**
- *I spoke to the chap who spoke to the child*
- 10 tokens
- 7 types
General Observations

- There are always a few very high-frequency words, and many low-frequency words.

- Among the top ranks, frequency differences are big.

- Among bottom ranks, frequency differences are very small.
Typical Shape of the Rank/Freq Curve
Ziph's Law

- Frequency decreases non-linearly with rank.

\[ f(w) = \frac{C}{r(w)^a} \]

- Suppose \( a = 1 \), and \( C = 60,000 \).

- The model predicts:
  - 2\textsuperscript{nd} most frequent word will be \( C/2 = 30,000 \)
  - 3\textsuperscript{rd} most frequent: \( C/3 = 20,000 \)
  - 20\textsuperscript{th} most frequent = \( C/20 = 3000 \)

- So frequency decreases very rapidly (exponentially) as rank increases.

\( a \) is a constant, determined from data, roughly the frequency of the most frequent word.

\( C \) is a constant, determined from data.
Ngrams & Counting Other Things

- Can basically count anything with FreqDist

- N-grams are sequences of \( n \) consecutive words e.g., "more is said than done"
  - Unigrams: "more", "is", "said", "than", "done"
  - Bigrams: "more is", "is said", "said than", "than done"
  - Trigrams: "more is said", "is said than", "said than done"
  - ...

- Used a lot in NLP applications
  - Language models (next week)
  - Collocation (next)
  - Language Identification
  - ASR
  - Machine Translation
Get some data, count some stuff

- Use the nltk.FreqDist class

```python
>>> from nltk.corpus import gutenberg
>>> words_per_sentence= [gutenberg.words(fileid) for fileid in gutenberg.fileids ()]

>>> words = [word.lower() for sublist in words_per_sentence for word in sublist]

>>> fdist = nltk.FreqDist(words)
>>> fdist.N()
2621613
>>> fdist.B()
42339
>>> fdist.hapaxes()

>>> fdist.most_common(10)
[('(', 186091), ('the', 133583), ('and', 95442), ('.', 73746), ('of', 71267), ('to', 48057), (':', 47406), ('a', 33960), ('in', 33580), ('i', 30265)]
```
- NLTK comes with lots of corpora
- Corpora may have structure & annotations

Table 1.2:

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Compiler</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown Corpus</td>
<td>Francis, Kucera</td>
<td>15 genres, 1.15M words, tagged, categorized</td>
</tr>
<tr>
<td>CESS Treebanks</td>
<td>CLiC-UB</td>
<td>1M words, tagged and parsed (Catalan, Spanish)</td>
</tr>
<tr>
<td>Chat-80 Data Files</td>
<td>Pereira &amp; Warren</td>
<td>World Geographic Database</td>
</tr>
<tr>
<td>CMU Pronouncing Dictionary</td>
<td>CMU</td>
<td>127k entries</td>
</tr>
<tr>
<td>CoNLL 2000 Chunking Data</td>
<td>CoNLL</td>
<td>270k words, tagged and chunked</td>
</tr>
<tr>
<td>CoNLL 2002 Named Entity</td>
<td>CoNLL</td>
<td>700k words, pos- and named-entity-tagged (Dutch, Spanish)</td>
</tr>
<tr>
<td>CoNLL 2007 Dependency Treebanks (scl)</td>
<td>CoNLL</td>
<td>150k words, dependency parsed (Basque, Catalan)</td>
</tr>
<tr>
<td>Dependency Treebank</td>
<td>Narad</td>
<td>Dependency parsed version of Penn Treebank sample</td>
</tr>
<tr>
<td>FrameNet</td>
<td>Fillmore, Baker et al</td>
<td>10k word senses, 170k manually annotated sentences</td>
</tr>
<tr>
<td>Floresta Treebank</td>
<td>Diana Santos et al</td>
<td>9k sentences, tagged and parsed (Portuguese)</td>
</tr>
<tr>
<td>Gazettee Lists</td>
<td>Various</td>
<td>Lists of cities and countries</td>
</tr>
<tr>
<td>Genesis Corpus</td>
<td>Misc web sources</td>
<td>6 texts, 200k words, 6 languages</td>
</tr>
<tr>
<td>Gutenberg (selections)</td>
<td>Hart, Newby et al</td>
<td>18 texts, 2M words</td>
</tr>
<tr>
<td>Inaugural Address Corpus</td>
<td>CSpan</td>
<td>US Presidential Inaugural Addresses (1789-present)</td>
</tr>
<tr>
<td>Indian POS-Tagged Corpus</td>
<td>Kumaran et al</td>
<td>60k words, tagged (Bangla, Hindi, Marathi, Telugu)</td>
</tr>
<tr>
<td>MacMorpho Corpus</td>
<td>NILC, USP, Brazil</td>
<td>1M words, tagged (Brazilian Portuguese)</td>
</tr>
<tr>
<td>Movie Reviews</td>
<td>Pang, Lee</td>
<td>2k movie reviews with sentiment polarity classification</td>
</tr>
<tr>
<td>Names Corpus</td>
<td>Kantrowitz, Ross</td>
<td>8k male and female names</td>
</tr>
<tr>
<td>NIST 1999 Info Extr (selections)</td>
<td>Garzio</td>
<td>63k words, newswire and named-entity SGML markup</td>
</tr>
<tr>
<td>Nombank</td>
<td>Meyers</td>
<td>115k propositions, 1400 noun frames</td>
</tr>
<tr>
<td>NPS Chat Corpus</td>
<td>Forsyth, Martell</td>
<td>10k IM chat posts, POS-tagged and dialogue-act tagged</td>
</tr>
<tr>
<td>Open Multilingual WordNet</td>
<td>Bond et al</td>
<td>15 languages, aligned to English WordNet</td>
</tr>
<tr>
<td>PP Attachment Corpus</td>
<td>Ratnaparkhi</td>
<td>28k prepositional phrases, tagged as noun or verb modifiers</td>
</tr>
<tr>
<td>Proposition Bank</td>
<td>Palmer</td>
<td>113k propositions, 3300 verb frames</td>
</tr>
<tr>
<td>Question Classification</td>
<td>Li, Roth</td>
<td>6k questions, categorized</td>
</tr>
<tr>
<td>Reuters Corpus</td>
<td>Project Gutenberg</td>
<td>1.3M words, 10k news documents, categorized</td>
</tr>
<tr>
<td>Roger’s Thesaurus</td>
<td>Dagan et al</td>
<td>200k words, formatted text</td>
</tr>
<tr>
<td>RTE Textual Entailment</td>
<td></td>
<td>8k sentence pairs, categorized</td>
</tr>
</tbody>
</table>
Conditional Counts

- We can also get some more interesting information by using a ConditionalFreqDist.
- How often have I seen word$_2$ given that word$_1$ immediately preceded it?
- *fox* is seen exactly twice after having seen *the*.

```python
>>> import nltk
>>> fables_text = open('cmps143/fables/TheFoxAndTheCrow.txt').read()
>>> sentences = nltk.sent_tokenize(fables_text)
>>> words = [nltk.word_tokenize(sentence) for sentence in sentences]
>>> flat_words = [word.lower() for sentence in words for word in sentence]
>>> bigrams = nltk.bigrams(flat_words)
>>> bgcdist = nltk.ConditionalFreqDist(bigrams)
>>> bgcdist.tabulate(conditions=["the", "fox", "and", "the", "crow"])
```

<table>
<thead>
<tr>
<th></th>
<th>birds</th>
<th>cheese</th>
<th>crow</th>
<th>fox</th>
<th>hue</th>
<th>just</th>
<th>observed</th>
<th>said</th>
<th>set</th>
<th>standing</th>
<th>that</th>
<th>the</th>
<th>tree</th>
<th>was</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fox</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>and</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>the</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>crow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Probability
Some notation

- $A \cup B$ = events in $A$ and events in $B$
- $A \cap B$ = events which are in both $A$ and $B$

- $P(A \cup B) = \text{probability that something which is either in } A \text{ OR } B \text{ occurs} - \text{also denoted } p(A \text{ or } B)$
- $P(A \cap B) = \text{probability that something which is in both } A \text{ AND } B \text{ occurs} - \text{also denoted } p(A \& B), p(A \text{ and } B), p(A,B)$
Estimation of conditional probability

- Intuition:
  - $P(A|B)$ is a ratio of the chances that both $A$ and $B$ happen, by the chances of $B$ happening alone.

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

- $P(\text{ADJ} \mid \text{DET}) = P(\text{DET+ADJ}) / P(\text{DET})$
Probability Rules Cheat Sheet

- Must sum to 1: \( P(\text{all events}) \)
  \[ P(\Omega) = \sum_{e \in \Omega} P(e) = 1 \]

- Complement Rule: \( P(A \) doesn’t happen) \)
  \[ P(\overline{A}) = 1 - P(A) \]

- Addition Rule: \( P(A \cup B) = P(A) + P(B) - P(A \cap B) \)

- Multiplication Rule: \( P(A \cap B) = P(A)P(B \mid A) \)

- Can be extended indefinitely
  - E.g. chances of drawing 4 straight aces from a pack
  \[ P(A_1 \& A_2 \& A_3 \& A_4) = P(A_1)P(A_2 \mid A_1)P(A_3 \mid A_1 \& A_2)P(A_4 \mid A_1 \& A_2 \& A_3) \]
Basic Probability
Probability: classical interpretation

- Given $n$ equally possible outcomes, and $m$ events of interest, the probability that one of the $m$ events occurs is $m/n$.
- If we call our set of events of interest $A$, then:

$$P(A) = \frac{|A|}{|\Omega|}$$

- Principle of insufficient reason (Laplace):
  - We should assume that events are equally likely, unless there is good reason to believe they are not.
Compound vs. simple events

- If $A$ is a compound event, then $P(A)$ is the sum of the probabilities of the simple events making it up:

$$P(A) = \sum_{a \in A} P(a)$$

The sum of probabilities, for all elements $a$ of $A$

- Recall, that $P(\text{Even}) = \frac{3}{6} = 0.5$

- In a throw of the Dice, the simple events are \{1,2,3,4,5,6\}, each with probability $\frac{1}{6}$

- $P(\text{Even}) = P(2), P(4), P(6) = \frac{1}{6} \times 3 = 0.5$
More rules...

- Since, for any compound event A:
  \[
P(A) = \sum_{a \in A} P(a)
  \]
  the probability of all events, \(P(\Omega)\) is:
  \[
P(\Omega) = \sum_{e \in \Omega} P(e) = 1
  \]
  (this is the likelihood of “anything happening”, which is always 100% certain)
Yet more rules...

- If A is any event, the probability that A does not occur is the probability of the complement of A:

$$P(\overline{A}) = 1 - P(A)$$

i.e. the likelihood that anything which is not in A happens.

- Impossible events are those which are not in $\Omega$. They have probability of 0.

- For any event A: $P(A) \in [0,1]$
Addition rule

To estimate probability of A OR B happening, we need to remove the probability of A AND B happening, to avoid double-counting events. (flip back three slides)

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

In our case:
- $P(A) = 2/4$
- $P(B) = 2/4$
- $P(A \text{ AND } B) = \frac{1}{4}$
- $P(A \text{ OR } B) = 2/4 + 2/4 - \frac{1}{4} = \frac{3}{4} = 0.75$
Conditional Probability
Another example

- If we throw a die, what’s the probability that the number we get is even, given that the number we get is larger than 4?
  - works out as the probability of getting the number 6

- \[ P(A|B) = \frac{P(A \& B)}{P(B)} \]

- \[ P(\text{even}|>4) = \frac{P(\text{even} \& >4)}{P(>4)} \]
  \[ = \frac{1/6}{2/6} \]
  \[ = \frac{1}{2} = 0.5 \]

- Note the difference from simple, prior probability.

- Using only frequency, \( P(6) = 1/6 \)
**Multiplication Rule**

\[ P(A \cap B) = P(A)P(B \mid A) \]

- Probability that both A and B occur
- Probability of A happening overall
- Probability of B happening given that A has happened

*Source:* [nlds.soe.ucsc.edu](http://nlds.soe.ucsc.edu)
Multiplication rule: example 1

- We have a standard deck of 52 cards

- What’s the probability of pulling out two aces in a row?
  - NB Standard deck has 4 aces

- Let A1 stand for “an ace on the first pick”, A2 for “an ace on the second pick”

- We’re interested in $P(A_1 \text{ AND } A_2)$
Example 1 continued

- $P(A_1 \text{ AND } A_2) = P(A_1)P(A_2|A_1)$
- $P(A_1) = \frac{4}{52}$
  - (since there are 4 aces in a 52-card pack)
- If we do pick an ace on the first pick, then we diminish the odds of picking a second ace (there are now 3 aces left in a 51-card pack).
  - $P(A_2|A_1) = \frac{3}{51}$
- Overall: $P(A_1 \text{ AND } A_2) = \left(\frac{4}{52}\right)\left(\frac{3}{51}\right)$
  $= .0045$
Extending the multiplication rule

- The logic of the “A AND B” rule is:
  - Both conditions, A and B have to be met
  - A is met a fraction of the time
  - B is met a fraction of the times that A is met

- Can be extended indefinitely
  - E.g. chances of drawing 4 straight aces from a pack
  - \( P(A_1 & A_2 & A_3 & A_4) \)
    
    \[ = P(A_1) P(A_2|A_1) P(A_3|A_1 & A_2) P(A_4|A_1 & A_2 & A_3) \]
Bayes' Theorem
Deriving Bayes’ rule from the multiplication rule

- Given symmetry of intersection, multiplication rule can be written in two ways

\[
P(A \cap B) = P(A)P(B \mid A)
\]

\[
P(A \cap B) = P(B)P(A \mid B)
\]

- Bayes’ rule involves the substitution of one equation into the other, to replace \(P(A\) and \(B)\)

\[
P(B \mid A) = \frac{P(B)P(A \mid B)}{P(A)}
\]

Eq both sides, divide through by \(P(A)\)
N-Gram Language Models
The Markov Assumption

- **Markov models:**
  - probabilistic models which predict the likelihood of a future unit based on **limited history**

- in language modelling, this pans out as the **local history assumption**:
  - the probability of $w_n$ depends on a limited number of prior words

- **utility of the assumption:**
  - we can rely on a small $n$ for our n-gram models (bigram, trigram)
  - long *n-grams* become exceedingly sparse
  - Probabilities become very small with long sequences
The structure of an n-gram model

- The task can be re-stated in conditional probabilistic terms:

\[ P(w_n \mid w_1 \ldots w_{n-1}) \]

- Limiting \( n \) under the Markov Assumption means:
  - greater chance of finding more than one occurrence of the sequence \( w_1 \ldots w_{n-1} \)
  - more robust statistical estimations

- N-grams are essentially *equivalence classes* or *bins*
  - every unique n-gram is a type or *bin*
  - \( BIN= \text{A CHosen Representation of the Context} \)
Maximum Likelihood Estimation Approach

- Basic equation:
  \[ P(w_n \mid w_1...w_{n-1}) = \frac{P(w_1...w_n)}{P(w_1...w_{n-1})} \]

- \( P(A \mid B) \) is a ratio of the chances that both A and B happen, by the chances of B happening alone.
  \[ P(A \mid B) = \frac{P(A \cap B)}{P(B)} \]

- \( P(\text{ADJ} \mid \text{DET}) = \frac{P(\text{DET}+\text{ADJ})}{P(\text{DET})} \)

- In a unigram model, this reduces to simple probability.

- MLE models estimate probability using relative frequency.
What N should we use?

- Unigram models:
  - not entirely hopeless because most sentences contain a majority of highly common words
  - BUT ignore syntax completely:
    - \( P(\text{the tall man}) = P(\text{man the tall}) \)

- Bigrams: improve situation dramatically

- Trigram models great *if you’ve seen it before!!*
  - Capture a surprising amount of contextual variation
  - Biggest limitation: *most* new trigrams in test data will not have been seen in training data.

- Even worse for 4-grams

- NOTE: Google provides a pre-compiled 5 gram list
**HW Q16**

Table 1 shows the unigram and bigram frequencies of a normalized document. The content of the document is: "The cat was on the mat. The cat slept on the bed. The cat was on the desk. The dog slept on the mat." Calculate the following probabilities:

a) \( \text{P(cat)} \)

b) \( \text{P(the | on)} \)

c) \( \text{P(cat, slept)} \)

d) \( \text{P(Sentence1)} \) using a bigram model

**Solution:**

a) \( \text{P(cat)} = \frac{3}{24} = \frac{1}{8} \)

b) \( \text{P(the | on)} = \frac{\text{p(on, the)}}{\text{p(on)}} = \frac{4}{4} = 1 \)

c) \( \text{P(cat, slept)} = \text{p(slept | cat)} \times \text{p(cat)} = \left(\frac{2}{3}\right) \times \left(\frac{3}{24}\right) \)

d) \( \text{P(the, cat, slept, on, the, desk)} = \text{P(the)} \times \text{P(cat | the)} \times \text{P(slept | cat)} \times \text{P(on | slept)} \times \text{P(the | on)} \times \text{P(desk | the)} \)
\[
= \left(\frac{8}{24}\right) \times \left(\frac{3}{8}\right) \times \left(\frac{2}{4}\right) \times 1 \times 1 \times \left(\frac{1}{8}\right) = \frac{1}{128}
\]
Moving Beyond Words: Part-of-Speech
Part-Of-Speech (POS)

- "a linguistic category of words (or more precisely lexical items), which is generally defined by the syntactic or morphological behaviour of the lexical item in question" – Wikipedia

- 8 traditional POS tags in English
  - Noun, pronoun, adjective, verb, adverb, preposition, conjunction, interjection

- Can be further subcategorized by function

- Difficult due to ambiguity when words have more than one possible tag.
  - need context to make a good guess about POS
  - How much context?
Example: Different ways of POS Tagging

POS tagging as a classification problem
Some Features for Automatic Tagging

- **Syntagmatic information**: the tags of other words in the context of $w$
  - Not sufficient on its own.

- **Lexical information** ("dictionary"): most common tag(s) for a given word
  - E.g. in English, many nouns can be used as verbs *(flour the pan, wax the car...)*
  - However, their most likely tag remains NN
  - Distribution of a word’s usages across different POSs is uneven: usually, one highly likely, other much less

- **Other lexical features**
  - Is it uppercase?
  - Prefix, suffix?
Section 5.4 Tagging

- Tag everything  NN: 13% Accuracy!

```python
>>> tags = [tag for (word, tag) in brown.tagged_words(categories='news')]
>>> nltk.FreqDist(tags).max()
'NN'

Now we can create a tagger that tags everything as NN.

```python
>>> raw = 'I do not like green eggs and ham, I do not like them Sam I am!

```python
>>> tokens = word_tokenize(raw)
>>> default_tagger = nltk.DefaultTagger('NN')
>>> default_tagger.tag(tokens)
[('I', 'NN'), ('do', 'NN'), ('not', 'NN'), ('like', 'NN'), ('green', 'NN'), ('eggs', 'NN'), ('and', 'NN'), ('ham', 'NN'), ('Sam', 'NN'), ('I', 'NN'), ('am', 'NN'), ('like', 'NN'), ('them', 'NN'), ('Sam', 'NN'), ('I', 'NN'), ('am', 'NN'), ('like', 'NN'), ('I', 'NN'), ('am', 'NN'), ('like', 'NN'), ('I', 'NN'), ('am', 'NN'), ('like', 'NN')]

Unsurprisingly, this method performs rather poorly. On a typical corpus, it will tag only about an eighth of the tokens correctly, as we see below:

```python
>>> default_tagger.evaluate(brown_tagged_sents)
0.13089484257215028
```
RegExp Tagger: A rule-based classifier

- Use lists and morphology of the word (suffixes etc)
- Defaults to NN (last pattern)
- GETS 20%!

Note that these are processed in order, and the first one that matches is applied. Now we can set up a tagger and use it to tag a sentence. Now it's right about a fifth of the time.

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

>>> regexp_tagger = nltk.RegexpTagger(patterns)
>>> regexp_tagger.tag(brown_sents[3])
[('``', 'NN'), ('Only', 'NN'), ('a', 'NN'), ('relative', 'NN'), ('handful', 'NN'), ('of', 'NN'), ('such', 'NN'), ('reports', 'NNS'), ('was', 'NNS'), ('received', 'VBD'), ('``', 'NN'), ('the', 'NN'), ('jury', 'NN'), ('said', 'NN'), ('considering', 'VBG'), ('the', 'NN'), ('widespread', 'NN'), '...

>>> regexp_tagger.evaluate(brown_tagged_sents)
0.20326391789486245
```
What do you think the unigram tagger does?
Unigram Tagger

- Unigram Tagger: A simple approach which assigns only the most common tag to each word gets greater than 80% accuracy!

```python
>>> size = int(len(brown_tagged_sents) * 0.9)
>>> size
4160
>>> train_sents = brown_tagged_sents[:size]
>>> test_sents = brown_tagged_sents[size:]
>>> unigram_tagger = nltk.UnigramTagger(train_sents)
>>> unigram_tagger.evaluate(test_sents)
0.811721...
```
Bigram and Trigram Taggers

- The bigram tagger finds the most likely tag for each word, given the preceding tag. Gets 89.6%!!

- What is this backoff thing?

```python
>>> bigram_tagger = nltk.BigramTagger(brown_train, backoff=unigram_tagger_2)
>>> print bigram_tagger.size()
3379
>>> print 'Accuracy: %4.1f%%' % (...
100.0 * bigram_tagger.evaluate(brown_test))
Accuracy: 89.6%
```

- Trigram Tagger: 89% !! Why no better?

```python
>>> trigram_tagger = nltk.TrigramTagger(brown_train, backoff=bigram_tagger)
>>> print trigram_tagger.size()
1495
>>> print 'Accuracy: %4.1f%%' % (...
100.0 * trigram_tagger.evaluate(brown_test))
Accuracy: 89.0%
```
Contextual View of Meaning
The Empiricist's View of Meaning

- Firth’s view (1957):
  - "You shall know a word by the company it keeps"

- This is a **contextual** view of meaning, akin to that espoused by Wittgenstein (1953).
  - i.e., meaning is how it's used

- In the Firthian tradition, attention is paid to patterns that crop up with regularity in language.

- Statistical work on collocations tends to follow this tradition.
Another View of Lexical Categories

- Word categories can be defined by the context in which they appear.
- For a word $w$ find all the contexts $w_1ww_2$ in which $w$ appears.
- Find all words $w'$ that share many frequent contexts.
Finding Similar Words in NLTK

- Use the nltk.Text class
- Use the similar function.

```python
>>> gutext
<Text: emma by jane austen 1816>
>>> gutext.similar('woman')
man people time day thing men one king place lord house night land
word other earth way lady world gentleman
```
Collocations
Definition

“Collocations ... are statements of the habitual or customary places of [a] word.” (Firth 1957)

Characteristics/Expectations:
- regular/frequently attested
- occur within a narrow window (span of few words)
- not fully compositional
- non-substitutable
- subject to category restrictions
Narrow Window (Textual Proximity)

- Usually, we specify an **n-gram** window within which to analyse collocations:
  - bigram: *credit card, credit crunch*
  - trigram: *credit card fraud, credit card expiry*
  - ...

- The idea is to look at co-occurrence of words within a specific n-gram window

- We can also count n-grams with intervening words:
  - *federal (.*) subsidy*
  - matches: *federal subsidy, federal farm subsidy, federal manufacturing subsidy*...
Non-Compositionality

- **white wine**
  - not really “white”, meaning not fully predictable from component words + syntax

- **signal interpretation**
  - a term used in Intelligent Signal Processing: connotations go beyond compositional meaning

- Similarly:
  - *regression coefficient*
  - *good practice guidelines*

- Extreme cases:
  - idioms such as *kick the bucket*
  - meaning is completely frozen
Frequency alone doesn’t indicate collocational strength:
- *by the* is a very frequent phrase in English
- not a collocation

Collocations tend to be formed from content words:
- A+N: *powerful tea*
- N+N: *regression coefficient, mass demonstration*
- N+PREP+N: *degrees of freedom*
Collocations in NLTK

- Use the `nltk.Text` module

```python
from nltk.corpus import gutenberg
words_per_sentence = [gutenberg.words(fileid) for fileid in gutenberg.fileids()]
words = [word.lower() for sublist in words_per_sentence for word in sublist]
gutext = nltk.Text(words)
gutext.collocations()

>>> gutext.collocations()
thou shalt; said unto; thou hast; thus saith; thou art; captain wentworth; lord god; frank churchill; unto thee; every one; sperm whale; burnt offering; jesus christ; lady russell; colonel brandon; say unto; miss woodhouse; father brown; spake unto; buster bear
```
Classification
Diagram of supervised classification

(a) Training
- Input
- Feature extractor
- Features
- Machine learning algorithm
- Label

(b) Prediction
- Input
- Feature extractor
- Features
- Classifier model
- Label

http://www.nltk.org/howto/classify.html
What is a machine learning algorithm?

**Its just a function.** Takes features as input and returns an output.

Simplest example: Linear regression

\[ Y = Mx + B \]

\[ Y = w_1x_1 + w_2x_2 + w_3x_3 \ldots + B \]

What format does the machine learning algorithm want?

**A VECTOR of features**
What the representation looks like

Vectors of features, the label

```python
>>> train = [
    ... (dict(a=1, b=1, c=1), 'y'),
    ... (dict(a=1, b=1, c=1), 'x'),
    ... (dict(a=1, b=1, c=0), 'y'),
    ... (dict(a=0, b=1, c=1), 'x'),
    ... (dict(a=0, b=1, c=1), 'y'),
    ... (dict(a=0, b=0, c=1), 'y'),
    ... (dict(a=0, b=0, c=0), 'x'),
    ... (dict(a=0, b=1, c=0), 'x'),
    ... (dict(a=0, b=1, c=1), 'y'),
    ...
]

>>> test = [
    ... (dict(a=1, b=0, c=1)), # unseen
    ... (dict(a=1, b=0, c=0)), # unseen
    ... (dict(a=0, b=1, c=1)), # seen 3 times, labels=y, y, x
    ... (dict(a=0, b=1, c=0)), # seen 1 time, label=x
    ...
]
```
Text Classification Algorithm

- Divide the corpus into three sets:
  - training set
  - test set
  - development (dev-test) set

1. Choose the features that will be used to classify the corpus.
2. Train the classifier on the training set.
3. Run it on the development set.
4. ANALYSE YOUR ERRORS: Refine the feature extractor from any errors produced on the development set.
5. REPEAT 1 THRU 4 UNTIL RUN OUT OF TIME OR IDEAS.
6. Run the improved classifier on the test set. CALCULATE YOUR FINAL RESULTS
See which features are helping

- Usually want to look at more than just the top five features
- 38 times more likely to see “a” as the last letter of a female name
- 31 times more likely to see “k” as the last letter of a male name
- Classifiers often work better with **fewer features**

```python
>>> classifier.show_most_informative_features(5)
Most Informative Features
last_letter = 'a'               female : male   = 38.3 : 1.0
last_letter = 'k'               male : female = 31.4 : 1.0
last_letter = 'f'               male : female = 15.3 : 1.0
last_letter = 'p'               male : female = 10.6 : 1.0
last_letter = 'w'               male : female = 10.6 : 1.0
```
def evaluate(classifier, dev_data, test_data):
    # Test on the development and test data
    dev_accuracy = nltk.classify.accuracy(classifier, develop_data)
    test_accuracy = nltk.classify.accuracy(classifier, test_data)

    print "{0:6s} {1:8.5f}".format("Dev", dev_accuracy)
    print "{0:6s} {1:8.5f}".format("Test", test_accuracy)

    features_only = [example[0] for example in develop_data]

    reference_labels = [example[1] for example in develop_data]
    predicted_labels = classifier.batch_classify(features_only)
    reference_text = [review[0] for review in reviews[1700:1800]]

    confusion_matrix = nltk.ConfusionMatrix(reference_labels, predicted_labels)

    print confusion_matrix

    for reference, predicted, text in zip(
        reference_labels,
        predicted_labels,
        reference_text
    ):
        if reference != predicted:
            print "{0} {1}\n{2}\n\n".format(reference, predicted, text)
## Baseline Confusion Matrix

- **Dev Accuracy**: 0.71000
- **Test Accuracy**: 0.74000

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>p</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td></td>
<td></td>
<td>o</td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td>s</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>neg</th>
<th>&lt;20&gt;29</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>.&lt;51&gt;</td>
<td></td>
</tr>
</tbody>
</table>
```

(row = reference; col = test)
Typical Problems

- Most distinctions between things can be framed in terms of categories
- Part of Speech Tagging: ADJ, VB, NN...
- Word categorization in general: LIWC categories
- Parsing and Chunking: Classify or Rank possible structures for a sentence
- Sentiment analysis on reviews, blogs, spoken language
Naïve Bayes Model

- Assume all features are independent given the class label Y.

\[
p(X_1, ..., X_n \mid Y) = p(X_1 \mid Y)p(X_2 \mid Y)...p(X_n \mid Y)
\]

\[
= \prod_{i=1}^{n} p(X_i \mid Y)
\]
Massive Reduction in Parameters

- # of parameters for modeling $P(X_1, ..., X_n | Y)$:
  - $2(2^n - 1)$

- # of parameters for modeling $P(X_1 | Y), ..., P(X_n | Y)$:
  - $2n$

- Counts are easy to obtain!
Other problems

- What happens to
  \[
  \prod_{i=1}^{n} p(X_i \mid Y)p(Y)
  \]

- If we have 100,000 features?

- E.g. \((0.5)^{100000}\)

- Try it on your calculator or computer
Other problems

- What happens to

\[
\prod_{i=1}^{n} p(X_i \mid Y)p(Y)
\]

- If we have 100,000 features?

- E.g. \((0.5)^{100000}\)

- Try it on your calculator or computer

- Hint it returns 0

- This is called underflow
Numerical Stability

- Work with log probabilities instead

- Remember
  \[ \log (0.5^{100000}) = 100000 \times \log(0.5) \approx -30,103 \]

- Much longer before underflowing

- So

  \[
  = \log \left( \prod_{i=1}^{n} p(X_i \mid Y) p(Y) \right)
  = \sum_{i=1}^{n} \log \left( p(X_i \mid Y) \right) + \log p(Y)
  \]
Lexical Resources
Lexical Resources in NLTK

- NLTK includes some corpora that are nothing more than **wordlists** (e.g., the Words Corpus)

- What can they be useful for?

- There is also a corpus of **stopwords**, that is, high-frequency words like *the*, *to* and *also* that we sometimes want to filter out of a document before further processing.

- Stopwords usually have little lexical content, and their presence in a text fails to distinguish it from other texts.

```python
>>> from nltk.corpus import stopwords
>>> stopwords.words('english')
['a', 'a\'s', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', 'afterwards', 'again', 'against', 'ain\'t', 'all', 'allow', 'allows', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', '...']
```
Feature Extraction: Lexical Dictionaries

- LIWC: Linguistic Inquiry and Word Count: categorizes words into a hierarchical set of lexical categories

- Sentiment Lexicons: Classify words into positive and negative polarity, either binary or scalar

- Wordnet:
  - classifies words hierarchically according to an ontology of things in the world, e.g. cup ISA container
  - Tells you the different senses of a word and groups words with their synonyms “service”

- Verbnet:
  - groups verbs by their meaning into an ontology
  - Tells you how verbs ‘subcategorize’ for their arguments
WordNet
WordNet

- **WordNet** is a large lexicon of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.

- Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser.

- NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets.

- **Senses and Synonyms**
  - Consider the 2 sentences:
    - *Benz is credited with the invention of the motorcar*
    - *Benz is credited with the invention of the automobile.*
  - *motorcar* and *automobile* have the same meaning, i.e. they are **synonyms**.
WordNet

- Like a dictionary and thesaurus on steroids
- Each entry has a list of senses for Nouns, Verbs, Adjectives and Adverbs
- Each sense has:
  - A **synset**: other words that have the same meaning (i.e., synonyms)
  - A **gloss**: a short definition
  - An **example**: an example usage of the word in context form a corpus
  - A set of **lexical relations**:
    - **Hypernym**: "is-a" relation. A car is a kind of motor vehicle
    - **Hyponym**: "instance of". A car is an instance of motor vehicle
    - **Meronym**: "part of". An engine is a part of a car
    - **Holonym**: "composed of". A forest is composed of trees
    - **Troponym**: "manner/style". Sneak is the style of walking
    - **Entailment**: "must also be true". Walking entails stepping.
The WordNet Hierarchy

WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: lynx

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) lynx (a text browser)
  ○ instance
- S: (n) lynx, catamount (short-tailed wildcats with usually tufted ears; valued for their fur)
  ○ direct hyponym / full hyponym
  ○ member holonym
  ○ direct hypernym / inherited hypernym / sister term
The WordNet Hierarchy

- S: (n) car#1, auto#1, automobile#1, machine#6, motorcar#1 (a motor vehicle with four wheels; usually propelled by an internal combustion engine) "he needs a car to get to work"
  - direct hyponym / full hyponym
    - S: (n) ambulance#1 (a vehicle that takes people to and from hospitals)
    - S: (n) funny wagon#1 (an ambulance used to transport patients to a mental hospital)
    - S: (n) beach wagon#1, station wagon#1, wagon#5, estate car#1, beach waggon#1, station waggon#1, waggon#2 (a car that has a long body and rear door with space behind rear seat)
    - S: (n) shooting brake#1 (another name for a station wagon)
    - S: (n) bus#4, jalopy#1, heap#3 (a car that is old and unreliable)
      "the fenders had fallen off that old bus"
Meronyms: Part Whole Relations

WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: automobile Search WordNet

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) car, auto, automobile, machine, motorcar (a motor vehicle with four wheels; usually propelled by an internal combustion engine) "he needs a car to get to work"
  - direct hyponym / full hyponym
  - part meronym
    - S: (n) accelerator, accelerator pedal, gas pedal, gas, throttle, gun (a pedal that controls the throttle valve) "he stepped on the gas"
    - S: (n) air bag (a safety restraint in an automobile; the bag inflates on collision and prevents the driver or passenger from being thrown forward)
    - S: (n) auto accessory (an accessory for an automobile)
    - S: (n) automobile engine (the engine that propels an automobile)
    - S: (n) automobile horn, car horn, motor horn, horn, hooter (a device on an automobile for making a warning noise)
    - S: (n) buffer, fender (a cushion-like device that reduces shock due to an impact)
    - S: (n) bumper (a mechanical device consisting of bars at either end of a vehicle to absorb shock and prevent serious damage)
    - S: (n) car door (the door of a car)
    - S: (n) car mirror (a mirror that the driver of a car can use)
    - S: (n) car seat (a seat in a car)
    - S: (n) car window (a window in a car)
    - S: (n) fender, wing (a barrier that surrounds the wheels of a vehicle to block splashing water or mud) "in Britain they call a fender a wing"
VerbNet
VerbNet: A Verb Lexicon

- VerbNet, a hierarchical verb lexicon linked to WordNet. It can be accessed with nltk.corpus.verbnet.

- *VerbNet is the largest on-line verb lexicon currently available for English.*

- It is a hierarchical domain-independent, broad-coverage verb lexicon with mappings to other lexical resources such as WordNet and FrameNet.
Each VerbNet class contains a set of syntactic descriptions, depicting the possible surface realizations of the argument structure for constructions such as transitive, intransitive, prepositional phrases, etc.

Semantic restrictions (such as animate, human, organization) are used to constrain the types of thematic roles allowed by the arguments.

Syntactic frames may also be constrained in terms of which prepositions are allowed.

Each frame is associated with explicit semantic information.

Adapted from VerbNet website.
### VerbNet: A Verb Lexicon

- Each verb argument is assigned one (usually unique) thematic role within the class.

#### Table 2: Thematic roles and example classes that use them

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>used for some communication classes (e.g., Chitchat-37.6, Marry-36.2, Meet-36.2) when both arguments can be considered symmetrical (pseudo-agents).</td>
</tr>
<tr>
<td>Agent</td>
<td>generally a human or an animate subject. Used mostly as a volitional agent, but also used in VerbNet for internally controlled subjects such as forces and machines.</td>
</tr>
<tr>
<td>Asset</td>
<td>used for the Sum of Money Alternation, present in classes such as Build-26.1, Get-13.5.1, and Obtain-13.5.2 with 'currency' as a selectional restriction.</td>
</tr>
<tr>
<td>Attribute</td>
<td>attribute of Patient/Theme refers to a quality of something that is being changed, as in (The price)att of oil soared. At the moment, we have only one class using this role Calibratable cos-45.6 to capture the Possessor Subject Possessor-Attribute Factoring Alternation. The selectional restriction 'scalar' (defined as a quantity, such as mass, length, time, or temperature, which is completely specified by a number on an appropriate scale) ensures the nature of Attribute.</td>
</tr>
<tr>
<td>Beneficiary</td>
<td>the entity that benefits from some action. Used by such classes as Build-26.1, Get-13.5.1, Performance-26.7, Preparing-26.3, and Steal-10.5. Generally introduced by the preposition 'for', or double object variant in the benefactive alternation.</td>
</tr>
<tr>
<td>Cause</td>
<td>used mostly by classes involving Psychological Verbs and Verbs Involving the Body.</td>
</tr>
<tr>
<td>Location, Destination, Source</td>
<td>used for spatial locations.</td>
</tr>
<tr>
<td>Destination</td>
<td>end point of the motion, or direction towards which the motion is directed. Used with a 'to' prepositional phrase by classes of change of location, such as Banish-10.2, and Verbs of Sending and Carrying. Also used as location direct objects in classes where the concept of destination is implicit (and location could not be Source), such as Butter-9.9, or Image impression-25.1.</td>
</tr>
<tr>
<td>Source</td>
<td>start point of the motion. Usually introduced by a source prepositional phrase (mostly headed by 'from' or 'out of'). It is also used as a direct object in such classes as Clear-10.3, Leave-51.2, and Wipe instr-10.4.2.</td>
</tr>
<tr>
<td>Location</td>
<td>underspecified destination, source, or place, in general introduced by a locative or path prepositional phrase.</td>
</tr>
</tbody>
</table>