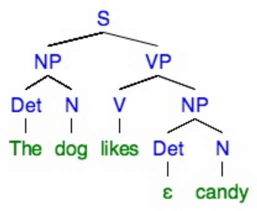
Caroline Butler
Introduction to Natural Language Processing
Kibble
20 November 2015

Q1

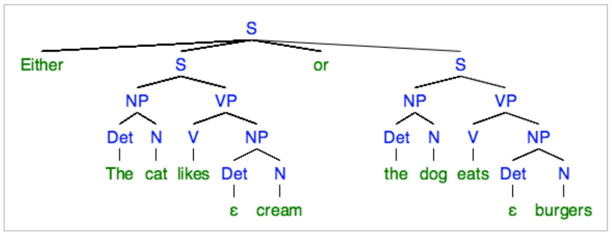
Between context-free and regular grammars, regular grammars are the simpler of the two kinds of grammars. Thus, they are not very useful for expressing anything in real speech because of their simplicity. The only modifications that one could make on regular grammar syntax would be to add in a single adjective or multiple adjectives before a noun. In addition, there can only be one noun and one determiner used. An example is: Det, ADJ, N. Because of this, it requires absolutely no memory to parse. On the other hand, context-free grammars are slightly more sophisticated. Instead of only being able to use one verb, context-free grammars can be mounted on to one another to create a sentence like so: Det, N, Det, N, V, V. In fact, one could create a sentence following that exact same structure infinitely many times. The finite kinds of these sentences can be parsed with a machine, because it will know to match each "Det, N," with each corresponding, "V." Context-free grammars are generally considered more appropriate for expressing natural language syntax, such as syntax trees. This is because they are structured based on a grammatical hierarchy. Natural anguage is so complex and intelligently created that regular grammars would never come close to accurately representing it.

Q2 a)

i. The dog likes candy

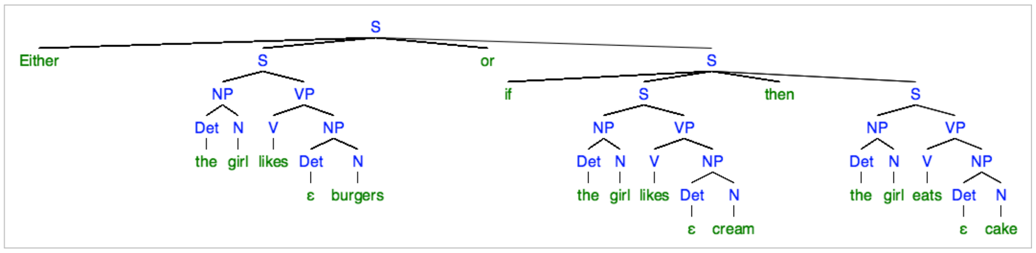


ii. Either the cat likes cream or the dog eats burgers



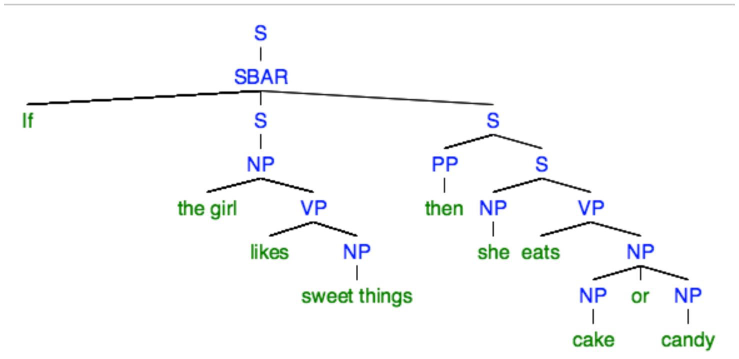
iii. Either the girl likes burgers or if the girl likes cream then the girl eats a cake

[S [Either]
[S [NP [Det the][N girl]] [VP [V likes][NP [Det ε] [N burgers]]]]
[or]
[S [if] [S[NP[Det the][N girl]] [VP[V likes][NP[Det ε][N cream]]]] [then]
[S[NP[Det the][N girl]][VP[V eats][NP[Det ε][N cake]]]

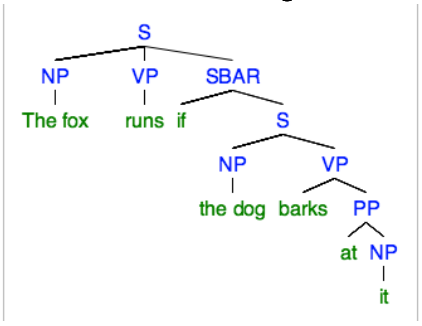


b)

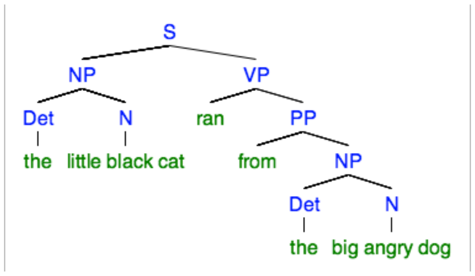
i. If the girl likes sweet things then she eats cake or candy



ii. The fox runs if the dog barks at it



iii. The little black cat ran from the big angry dog.



Q3 a)

1. a) Synonym: Two or more words are synonyms if they have the same or very similar meanings. Some examples are, “funny,” and, “humorous,” and “intelligent,” and, “smart.”
   b) Antonym: Two or more words that have opposite meanings. Some examples are, “win,” and, “lose,” and, “smart,” and, “stupid.”
   c) Hyponym: A word is a hyponym of another word if it is within the group of words that the second word describes. Some examples are, “apple,” to, “fruit,” and, “television,” to, “electronics.”
   d) Hypernyms: This is the opposite of hyponym, and refers to the word describing a category. Some examples are, “keyboard,” to, “key,” and, “alphabet,” to, “letter.”
   2. Isolated corpus structure: An isolated corpus structure is a body of text with no organization in particular. These are the simplest kinds of corpus structures (NLTK Book, Section 1.8).
   An example is the Gutenberg corpus (NLP Subject Guide, 37).
   3. Temporal corpus structure: A temporal corpus structure must exhibit a change in language and sentiment over a course of time. (NLP Subject Guide, 37) An example is the Inaugural Addresses corpus of the United States’ presidents.
   4. Collocations: Collocations are used in Natural Language Processing to give an idea of the main point of a text. They are pairs of sets of words that appear the most frequently in a text (NLP Subject Guide, 38).
   5. Lexical Diversity: Lexical diversity is a measure of the variance in types of words and symbols in a text. A high lexical diversity means that there is a lot of variance, and a low lexical diversity means that there is not a lot of variance (NLTK Book, Section 1.4).

b)

i. (More information in Python Appendix - Q3)
The presidents do appear to have a few preferred words. For instance, in George Bush’s second inaugural address, he said, “freedom,” 25 times. The second highest-used word by one of the presidents is, “peace,” by Richard Nixon in 1973. He used it 19 times in 1973 and 12 times in 1969. Similarly, Ronald Reagan said, “freedom,” 12 times in his inaugural address.

ii. (More information in Python Appendix - Q3 (b) ii.)
Yes, one thing that I notice is that Republican presidents use the words that we searched for more frequently than the Democratic presidents. In particular, they used the words, "war,", "peace,", and, "freedom" more. The Democratic presidents use each of the words less overall.

iii. (Answer is in Python Appendix - Q3 (b) iii.)

Q4 a)
1. a) tag: A tag the association of a part of speech with a string in a body of text.
b) supervised learning: Supervised Learning is the process of training a machine using data that has already been analyzed and marked by humans in order to test it using new data. This is often done with, "training" and, "testing" sets of data.
c) n-gram tagger: N-gram taggers try to label a tag based on the previous n-1 (where n>1) words in a set.
d) inter-annotator agreement: Inter-annotator agreement is a measure of how individual annotators interpret a body of text, in comparison with an outline created by a human.
e) backoff tagger: A backoff tagger combines the functionalities of several distinct taggers to create a better-performing tagger.

b) (More information in Python Appendix – Q4 (b))
After I tested my new RE tagger, which included various preposition and determiners, I found that it actually decreased the accuracy completely. At first, the accuracy was about .2, and it fell to 0. I think this is because I may have tried to over fit the news corpus.

Q5 a) (More information in Python Appendix – Q5 (a))
The most informative feature found was the word, "outstanding," which was ranked as positive 10.9:1. This makes sense, as outstanding generally implies that something had not been done before, or was a leader of its kind in a field. What was interesting was that both, "mulan," and, "damon," were considered to be positive attributes. This must be because Mulan is a very highly celebrated film, and Matt Damon is one of Hollywood's favorite actors.

b) (More information in Python Appendix – Q5 (b), and Appendix (1) and (2))
Interestingly, the classifier wrongly classified one of the reviews. The review by Ali Arikan on Roger Ebert's website was poor, but it must have used positive words sarcastically, thereby tricking the machine. Beyond that, each of the rest of the reviews were done correctly.

Q6 a) (More information in Python Appendix – Q6 (a))
I chose to add in JJR (comparative adjective) , JJS (superlative adjective), V (verb), and RB (adverb). The tagger already included regular adjectives, so I thought it

would be good to include different kinds of adjectives. This is especially important in a corpus as well-written as the Wall Street Journal. In addition, it is hardly sufficient to only have noun parts of speech. So, I added in some verb ones as well.

b) (More information in Python Appendix – Q6 (b))

The most obvious difference between the two outputs is the different classifications between parts of speech. The universal tagset also caught words such as, “urban”, whereas the regular tagset did not. Overall, the universal tagset did a better job of sorting the words.

PYTHON APPENDIX:

IMPORTS:

In [2]:

```
import re, nltk
from nltk.corpus import inaugural
from nltk.corpus import PlaintextCorpusReader
from nltk.corpus import inaugural
from nltk.corpus import stopwords
from nltk.corpus import words
import nltk.tree
from nltk.corpus import treebank as wsj
from nltk import *
```

Q3:

In [70]:

```
# Q3

cfd = nltk.ConditionalFreqDist(
   (presidents, word)
   for presidents in inaugural.fileids()
   for word in inaugural.words(fileids=presidents))

presidents = ['1961-Kennedy.txt',
 '1965-Johnson.txt',
 '1969-Nixon.txt',
 '1973-Nixon.txt',
 '1977-Carter.txt',
 '1981-Reagan.txt',
 '1985-Reagan.txt',
 '1989-Bush.txt',
 '1993-Clinton.txt',
 '1997-Clinton.txt',
 '2001-Bush.txt',
 '2005-Bush.txt',
 '2009-Obama.txt']

modals = ['war', 'peace', 'freedom', 'terror', 'economy', 'prosperity', 'jobs', 'wealth']

cfd.tabulate(conditions=presidents, samples=modals)
```

```
               war peace freedom terror economy prosperity jobs wealth
1961-Kennedy.txt    4    4    4    1    0    0    0    0
1965-Johnson.txt    1    0    2    0    0    0    0    1
  1969-Nixon.txt    3   12    2    0    1    0    0    2
  1973-Nixon.txt    2   19    4    0    0    0    0    0
 1977-Carter.txt    1    1    4    0    0    0    0    0
 1981-Reagan.txt    1    2    7    1    4    0    1    1
 1985-Reagan.txt    1    8   12    0    5    0    0    0
   1989-Bush.txt    2    3    4    0    0    1    0    0
1993-Clinton.txt    0    0    3    0    3    1    1    0
1997-Clinton.txt    1    1    2    2    2    0    0    1
   2001-Bush.txt    1    1    5    0    2    0    0    0
   2005-Bush.txt    0    1   25    0    0    0    0    1
  2009-Obama.txt    2    4    3    1    3    3    3    2
```

In [235]:

```
# Q3 (b) ii.

cfd = nltk.ConditionalFreqDist(
    (presidents, word)
    for presidents in inaugural.fileids()
    for word in inaugural.words(fileids=presidents))

presidents = ['1961-Kennedy.txt',
 '1969-Nixon.txt',
 '1973-Nixon.txt',
 '1977-Carter.txt',
 '1981-Reagan.txt',
 '1985-Reagan.txt',
 '1989-Bush.txt',
 '1993-Clinton.txt',
 '1997-Clinton.txt',
 '2001-Bush.txt',
 '2005-Bush.txt',
 '2009-Obama.txt']

modals = ['war', 'peace', 'freedom', 'terror', 'economy', 'prosperity', 'jobs', 'wealth']
```

```
cfd.tabulate(conditions=presidents, samples=modals)
```

```
                 war peace freedom terror economy prosperity jobs wealth
1961-Kennedy.txt    4    4    4    1    0    0    0    0
  1969-Nixon.txt    3   12    2    0    1    0    0    2
  1973-Nixon.txt    2   19    4    0    0    0    0    0
 1977-Carter.txt    1    1    4    0    0    0    0    0
 1981-Reagan.txt    1    2    7    1    4    0    1    1
 1985-Reagan.txt    1    8   12    0    5    0    0    0
   1989-Bush.txt    2    3    4    0    0    1    0    0
1993-Clinton.txt    0    0    3    0    3    1    1    0
1997-Clinton.txt    1    1    2    2    2    0    0    1
   2001-Bush.txt    1    1    5    0    2    0    0    0
   2005-Bush.txt    0    1   25    0    0    0    0    1
  2009-Obama.txt    2    4    3    1    3    3    3    2
```

In [4]:

```
# Q3 (b) iii.

import re, nltk
from nltk.corpus import inaugural
from nltk.corpus import PlaintextCorpusReader
from nltk.corpus import stopwords
from nltk.corpus import words
import nltk.tree
from nltk.corpus import treebank as wsj
from nltk import *

# iii.

stopwords = stopwords.words('english')

def freq_dist(text1,text2):
    new_text = [w for w in text1 and text2 if w not in stopwords and w.isalpha()]
    fd = nltk.FreqDist(new_text)
    return fd.most_common(10)

def freq_dist2(text1,text2,text3,text4):
    new_text = [w for w in text1 and text2 and text3 and text4 if w not in stopwords and
w.isalpha()]
    fd = nltk.FreqDist(new_text)
    return fd.most_common(10)


corpus_root = "nltk_data/corpora"
```

```
mycorpus = PlaintextCorpusReader(corpus_root,'.*\.txt')

#def freq_dist(text,):
#    for word in text:

Lincoln1 = nltk.Text(mycorpus.words('inaugural/1861-Lincoln.txt'))
Lincoln2 = nltk.Text(mycorpus.words('inaugural/1865-Lincoln.txt'))

Lincoln_fd = freq_dist(Lincoln1, Lincoln2)
print("Lincoln most common:", Lincoln_fd)

FDR1 = nltk.Text(mycorpus.words('inaugural/1933-Roosevelt.txt'))
FDR2 = nltk.Text(mycorpus.words('inaugural/1937-Roosevelt.txt'))
FDR3 = nltk.Text(mycorpus.words('inaugural/1941-Roosevelt.txt'))
FDR4 = nltk.Text(mycorpus.words('inaugural/1945-Roosevelt.txt'))

FDR_fd = freq_dist2(FDR1,FDR2,FDR3,FDR4)
print("FDR most common:", FDR_fd)

Reagan1 = nltk.Text(mycorpus.words('inaugural/1981-Reagan.txt'))
Reagan2 = nltk.Text(mycorpus.words('inaugural/1985-Reagan.txt'))

Reagan_fd = freq_dist(Reagan1,Reagan2)
print("Reagan most common:", Reagan_fd)
#fd = nltk.FreqDist(Lincoln1)
#fd.tabulate()
#fd.values(Lincoln1)
```

```
Lincoln most common: [ (war , 12), ( God , 6), ( shall , 5), ( let , 4), ( Union , 4), ( years , 4), ( may , 3), ( His , 3), ( offenses , 3), ( would , 3)]
FDR most common: ([We , 11), ( shall , 7), ( peace , 6), ( learned , 5), ( I , 4), ( men , 4), ( today , 4), ( way , 3), ( test , 3), ( The , 3)]
Reagan most common: [(us , 27), ( We , 24), ( people , 16), ( world , 16), ( I , 12), ( must , 12), ( one , 12), ( freedom , 12), ( time , 10), ( And , 10)]
```

Q4:

In [10]:

```
# Q4 (b)

import re, nltk
from nltk.corpus import inaugural
from nltk.corpus import PlaintextCorpusReader
from nltk.corpus import brown
from nltk.corpus import stopwords
from nltk.corpus import words
import nltk.tree
from nltk.corpus import treebank as wsj
from nltk import *

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)
]

new_patterns = [
   (r'^I|you|he|him|she|her|it|they|them$', 'PRON'), # pronouns
   (r'^and|but|or|if|while|although$','CONJ'), # conjunctions
   (r'^a|the|some|many|most$','DET'), # determiner
   (r'^[A-Z].*','NOUN'), # proper noun
   (r'.*ing$', 'VERB'), # gerunds
   (r'.*ed$', 'VERB'), # simple past
   (r'.*es$', 'VERB'), # 3rd singular present
   (r'.*ould$', 'VERB'), # modals
   (r'.*\'s$', 'NOUN'), # possessive nouns
   (r'.*s$', 'NOUN'), # plural nouns
   (r'^-?[0-9]+(.[0-9]+)?$', 'NUM'), # cardinal numbers
   (r'.*', 'NOUN'), # nouns (default)
   (r'quite|such|rather','ABL'), #determiner
   (r'all|half|many|nary','ABN'), #determiner
   (r'both','ABX'), #determiner
 ]

bts=brown.tagged_sents(categories='news')
bs = brown.sents(categories = 'news')
size = int(len(bts)*.9)
train_set = bts[:size]
test_set = bts[size:]
regexp_tagger = nltk.RegexpTagger(patterns)
print("Without editing:",regexp_tagger.evaluate(test_set))
```

```
regexp_tagger1 = nltk.RegexpTagger(new_patterns)
print("With editing:", regexp_tagger1.evaluate(test_set))
```

```
Without editing: 0.2047244094488189
With editing: 0.0
```

Q5:

In [201]:

```
# Q5 (a)
import nltk
from nltk.corpus import movie_reviews
import random

def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features

documents = [(list(movie_reviews.words(fileid)), category)
for category in movie_reviews.categories()
for fileid in movie_reviews.fileids(category)]

random.shuffle(documents)
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

word_features = [w for (w,ct) in all_words.most_common(2000)]
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
#print(train_set)
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
classifier.show_most_informative_features(30)
```

```
0.86
Most Informative Features
  contains(outstanding) = True           pos : neg    =     10.9 : 1.0
        contains(mulan) = True           pos : neg    =      8.9 : 1.0
       contains(seagal) = True           neg : pos    =      8.3 : 1.0
  contains(wonderfully) = True           pos : neg    =      7.2 : 1.0
        contains(damon) = True           pos : neg    =      6.3 : 1.0
   contains(ridiculous) = True           neg : pos    =      5.8 : 1.0
        contains(flynt) = True           pos : neg    =      5.6 : 1.0
       contains(wasted) = True           neg : pos    =      5.5 : 1.0
```

```
        contains(lame) = True              neg : pos    =     5.3 : 1.0
       contains(waste) = True              neg : pos    =     5.2 : 1.0
       contains(awful) = True              neg : pos    =     5.2 : 1.0
      contains(poorly) = True              neg : pos    =     4.8 : 1.0
       contains(worst) = True              neg : pos    =     4.6 : 1.0
      contains(allows) = True              pos : neg    =     4.5 : 1.0
         contains(era) = True              pos : neg    =     4.5 : 1.0
       contains(hanks) = True              pos : neg    =     4.1 : 1.0
        contains(mess) = True              neg : pos    =     4.0 : 1.0
    contains(terrific) = True              pos : neg    =     3.9 : 1.0
      contains(stupid) = True              neg : pos    =     3.9 : 1.0
     contains(unfunny) = True              neg : pos    =     3.9 : 1.0
       contains(bland) = True              neg : pos    =     3.9 : 1.0
   contains(portrayal) = True              pos : neg    =     3.8 : 1.0
   contains(memorable) = True              pos : neg    =     3.8 : 1.0
   contains(fantastic) = True              pos : neg    =     3.7 : 1.0
        contains(dull) = True              neg : pos    =     3.7 : 1.0
        contains(jedi) = True              pos : neg    =     3.6 : 1.0
   contains(laughable) = True              neg : pos    =     3.6 : 1.0
        contains(zero) = True              neg : pos    =     3.6 : 1.0
      contains(boring) = True              neg : pos    =     3.6 : 1.0
       contains(snake) = True              neg : pos    =     3.6 : 1.0
```

In [200]:

```
# Q5 (b)

import nltk, random
from nltk.corpus import movie_reviews
from nltk.corpus import PlaintextCorpusReader

documents = [(list(movie_reviews.words(fileid)), category)
for category in movie_reviews.categories()
for fileid in movie_reviews.fileids(category)]

random.shuffle(documents)

def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features

all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
```

```
word_features = [w for (w,ct) in all_words.most_common(2000)]
featuresets = [(document_features(d), c) for (d,c) in documents]

train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)


# ALI ARIKAN:
corpus_root = "/Users/cmbutler/Documents/NLP/corpora"
mycorpus = PlaintextCorpusReader(corpus_root,'.*\.txt')
arikan = mycorpus.words('AliArikan.txt')
arikan_review_tokens =nltk.word_tokenize(mycorpus.raw('AliArikan.txt'))
arikan_df = document_features(arikan_review_tokens)
print("Ali Arkan's review:", classifier.classify(arikan_df))


# PETER BRADSHAW:
corpus_root = "/Users/cmbutler/Documents/NLP/corpora"
mycorpus = PlaintextCorpusReader(corpus_root,'.*\.txt')
bradshaw = mycorpus.words('PeterBradshaw.txt')
bradshaw_review_tokens =nltk.word_tokenize(mycorpus.raw('PeterBradshaw.txt'))
bradshaw_df = document_features(bradshaw_review_tokens)
print("Peter Bradshaw's review:", classifier.classify(bradshaw_df))


#IMDB 1
corpus_root = "/Users/cmbutler/Documents/NLP/corpora"
mycorpus = PlaintextCorpusReader(corpus_root,'.*\.txt')
imdb1 = mycorpus.words('imdb1.txt')
imdb1_review_tokens =nltk.word_tokenize(mycorpus.raw('imdb1.txt'))
imdb1_df = document_features(imdb1_review_tokens)
print("First IMDB review:", classifier.classify(imdb1_df))

#IMDB 2
corpus_root = "/Users/cmbutler/Documents/NLP/corpora"
mycorpus = PlaintextCorpusReader(corpus_root,'.*\.txt')
imdb2 = mycorpus.words('imdb2.txt')
imdb2_review_tokens =nltk.word_tokenize(mycorpus.raw('imdb2.txt'))
imdb2_df = document_features(imdb2_review_tokens)
print("Second IMDB review:", classifier.classify(imdb2_df))
```

```
Ali Arkan s review: pos
Peter Bradshaw s review: pos
First IMDB review: pos
```

Second IMDB review: neg

Q6:

In [4]:

```
# Q6 (a)

import nltk
from nltk.corpus import treebank as wsj
import nltk.tree
from nltk.tree import Tree
import re

grammar = r"""
    NP: {<DT|PP\$>?<JJ|JJR|JJS>*<NN>}
        {<NNP>+}
    VP: {<V><RB>*}
"""
uni_grammar = "NP: {<DET>?<ADJ>*<NOUN>+}"
cp = nltk.RegexpParser(grammar)



wsj_tagged_sents = wsj.tagged_sents()
chunked_sents = [cp.parse(sent) for sent in wsj_tagged_sents]
print(chunked_sents[0])
```

```
(S
  (NP Pierre/NNP Vinken/NNP)
  ,/,
  61/CD
  years/NNS
  old/JJ
  ,/,
  will/MD
  join/VB
  (NP the/DT board/NN)
  as/IN
  (NP a/DT nonexecutive/JJ director/NN)
  (NP Nov./NNP)
  29/CD
  ./.)
```

In [3]:

```
# Q6 (b)
```

```
uni_grammar = "NP: {<DET>?<ADJ>*<NOUN>+}"
up = nltk.RegexpParser(uni_grammar)

all_chunks = \
[tree.leaves() for s in chunked_sents for tree in s\
 if isinstance(tree,Tree)]
chunk_list = [i for i in all_chunks]
chunk_list.sort(key = len, reverse = True)
print(chunk_list[:10])

uni_wsj_tagged_sents = wsj.tagged_sents(tagset = 'universal')
uni_chunked_sents = [up.parse(sent) for sent in uni_wsj_tagged_sents]
uni_all_chunks = \
[tree.leaves() for s in uni_chunked_sents for tree in s\
 if isinstance(tree,Tree)]
uni_new_chunks = [j for j in uni_all_chunks]
uni_new_chunks.sort(key = len, reverse = True)
print(uni_new_chunks[:10])
```

```
[,;Leighton ,  NNP ), ( E. ,  NNP ), ( Cluff ,  NNP ), ( M.D. ,  NNP ), ( President ,  NNP ), ( Robert ,
NNP ), ( Wood ,  NNP ), ( Johnson ,  NNP ), ( Foundation ,  NNP ), ( Princeton ,  NNP )], ,;William ,
NNP ), ( R. ,  NNP ), ( Breakey ,  NNP ), ( M.D. ,  NNP ), ( Pamela ,  NNP ), ( J. ,  NNP ), ( Fischer ,
NNP ), ( M.D. ,  NNP ), ( Department ,  NNP )], ,;Big ,  NNP ), ( Board ,  NNP ), ( Chairman ,  NNP ),
( John ,  NNP ), ( J. ,  NNP ), ( Phelan ,  NNP ), ( Jr. ,  NNP )], ,;U.S. ,  NNP ), ( Assistant ,  NNP ), (
Treasury ,  NNP ), ( Secretary ,  NNP ), ( David ,  NNP ), ( Mulford ,  NNP )], ,;U.S. ,  NNP ), (
District ,  NNP ), ( Judge ,  NNP ), ( Mary ,  NNP ), ( Johnson ,  NNP ), ( Lowe ,  NNP )], ,;Kansas ,
NNP ), ( City ,  NNP ), ( Fed ,  NNP ), ( President ,  NNP ), ( Roger ,  NNP ), ( Guffey ,  NNP )],
,;Georgia ,  NNP ), ( Gulf ,  NNP ), ( President ,  NNP ), ( Jerry ,  NNP ), ( R. ,  NNP ), ( Satrum ,
NNP )], ,;Texas ,  NNP ), ( Air ,  NNP ), ( Corp. ,  NNP ), ( Chairman ,  NNP ), ( Frank ,  NNP ), (
Lorenzo ,  NNP )], ,;New ,  NNP ), ( York ,  NNP ), ( Stock ,  NNP ), ( Exchange ,  NNP ), ( Composite
,  NNP ), ( Index ,  NNP )], ,;American ,  NNP ), ( Stock ,  NNP ), ( Exchange ,  NNP ), ( Market ,
NNP ), ( Value ,  NNP ), ( Index ,  NNP )]]
[,;Leighton ,  NOUN ), ( E. ,  NOUN ), ( Cluff ,  NOUN ), ( M.D. ,  NOUN ), ( President ,  NOUN ), (
Robert ,  NOUN ), ( Wood ,  NOUN ), ( Johnson ,  NOUN ), ( Foundation ,  NOUN ), ( Princeton ,
NOUN )], ,;William ,  NOUN ), ( R. ,  NOUN ), ( Breakey ,  NOUN ), ( M.D. ,  NOUN ), ( Pamela ,
NOUN ), ( J. ,  NOUN ), ( Fischer ,  NOUN ), ( M.D. ,  NOUN ), ( Department ,  NOUN )], ,;the ,  DET
), ( Palestine ,  NOUN ), ( Liberation ,  NOUN ), ( Organization ,  NOUN ), ( news ,  NOUN ), (
agency ,  NOUN ), ( WAFA ,  NOUN )], ,;a ,  DET ), ( New ,  NOUN ), ( York ,  NOUN ), ( exchange ,
NOUN ), ( board ,  NOUN ), ( meeting ,  NOUN ), ( today ,  NOUN )], ,;the ,  DET ), ( powerful ,
ADJ ), ( Hawaii ,  NOUN ), ( Democrat ,  NOUN ), ( Sen. ,  NOUN ), ( Daniel ,  NOUN ), ( Inouye ,
NOUN )], ,;Big ,  NOUN ), ( Board ,  NOUN ), ( Chairman ,  NOUN ), ( John ,  NOUN ), ( J. ,  NOUN
), ( Phelan ,  NOUN ), ( Jr. ,  NOUN )], ,;Duke ,  NOUN ), ( University ,  NOUN ), ( law ,  NOUN ), (
professor ,  NOUN ), ( William ,  NOUN ), ( Van ,  NOUN ), ( Alstyne ,  NOUN )], ,;the ,  DET ), (
New ,  NOUN ), ( York ,  NOUN ), ( Stock ,  NOUN ), ( Exchange ,  NOUN ), ( Composite ,  NOUN ),
( Index ,  NOUN )], ,;The ,  DET ), ( American ,  NOUN ), ( Stock ,  NOUN ), ( Exchange ,  NOUN ), (
Market ,  NOUN ), ( Value ,  NOUN ), ( Index ,  NOUN )], ,;the ,  DET ), ( New ,  NOUN ), ( York ,
NOUN ), ( State ,  NOUN ), ( Urban ,  NOUN ), ( Development ,  NOUN ), ( Corp. ,  NOUN )]]
```

In [ ]:

APPENDIX:

(1) IMDB1:
url: http://www.imdb.com/title/tt1602613/reviews
title: An atmospheric tale of revenge, and not the sequel for Drive
author: Some random dude from France
date published: 24 May 2013
date visited: 19 Nov 2015

"Critics have gone way too hard on this movie. Lots of violent, strange et slow films have been presented at the Cannes film festival since its creation but yet every time a film pushes the boundaries of violence while keeping its own style, most critics go mad and sometimes shout at the screening, even leaving the theater before the end and calling it "outrageous". This film, along with "Anti-Christ" is a perfect example of the type of scandals that go on at Cannes for quite stupid reasons.

First of all, forget about Drive. If you know Nicolas Winding Refn's style and like it then you'll enjoy this movie but if you've only seen Drive and believe this is going to be in the same style (because of the same actor, similar cinematography, same musical style...) believe me you'll be disappointed. The trailer might give this impression, but this film is very different. The director had already made other movies just like this, but they did not encounter a really large audience. His works were mostly known by cinephiles, artsy people and intellectuals interested in film analysis (in a general way of course). Drive was his first really big success and also his first film taking place in America, starring a worldwide known star (Gosling) and going deep into its message while keeping a more specific style than his other films.

Here Refn feels a lot more philosophical, and comes back to his original style in directing films such as Valhalla Rising : great visuals, slow-pasted action, scenes that seem a bit detached from one-another, deep character development, little dialogue, extreme violence mixed with soft and/or trance-electro music... all of which are here to deal with philosophical, deep, hard subjects like revenge, good and bad, mother/son relationship etc...

When it comes to the acting Gosling does not disappoints however this time Refn wanted to do the opposite that he did in Drive : showing the weakness of his character. Also, even though he does pull-off a very convincing performance, Kristin Scott Thomas is surprisingly captivating and gives her character a much more "real" dimension than it could have been (like it is most of the time, when a woman is supposed to play a drug-lord badass). But saving the best for the end, Vithaya Pansringarm, an actor totally unknown to me until know, plays wonderfully his role as the mystical bad guy, and really did surprise me by the quality of his acting. He completely understood the movie's atmosphere and makes his character feel mysterious and fascinating.

To sum-up this is a very atmospheric, deep movie with great actors/actresses and dealing with difficult and serious themes, with some philosophical analysis possible, but definitely not in the same style as Drive, even though it has some similarities with it."

(2) IMDB2:

url: http://www.imdb.com/title/tt1602613/reviews
title: Slow, so slow
author: Havgar from France
date published: 24 May 2-13
date visted: 19 November 2015

"If you've seen Drive, then you should know that this movie is nothing like it, except perhaps in the fact that they are both beautifully shot. Drive had a pretty brisk pace, good dialogue, a plot that went somewhere, and a likable character.

Only God Forgives had none of that. This is a movie which moves along at a snail's pace, and even at a runtime of 90 minutes, it feels like many hours go by before even a single thing happens. Even the characters move and turn slowly.

The plot, such as it is, you would probably find worth watching, but Nicolas Winding Refn peppers it with pseudo-dream sequences and many pointless scenes that drag on for ever, so that the plot becomes hard to stay interested in.

Now, some things you might care about.

The acting. Ryan Gosling, of whom I was a fan in his earlier days, plays the same character from Drive, except that here he is indeed even more emotionless. He speaks about 5 lines during the whole movie, and has fewer different facial expressions. Kristin Scott Thomas is very good, although she feels underused. She is definitely the strong point of this movie. Vithaya Pansringarm, who plays a prominent role in the movie, is as expressionless as Gosling, although he is somewhat better, in my opinion.

Action scenes do exist, and they do resemble those from Drive, in that they are very matter-of-factly and visceral. Here, Winding Refn has really indulged in a lot of gratuitous gore, although overall, I found the action scenes quite entertaining. One particular one showcases Byron Gibson's acting talents, and it is particularly (and hilariously) cringe-worthy.

All the characters in this movie are unlikable. It is extremely difficult to get yourself to care for any of them, including Gosling's, who is arguably the protagonist here.

Aside from Scott Thomas' acting, the only other redeeming quality of this film is the excellent way in which most scenes are set up and shot. The sets, the camera movement, the placement of the actors, all of these make up for some truly gorgeous shots.

Overall, sad as I am to say it, I cannot recommend seeing Only God Forgives.”

WORKS CITED:

1. Bird, Steven, Ewan Klein, and Edward Loper. "Natural Language Processing with Python." *NLTK Book*. Creative Commons Attribution Noncommercial No-Derivative-Works 3.0 US License, 2009. Web. 19 Nov. 2015.
2. Kibble, Rodger. “Introduction to natural language processing.” Goldsmiths, University of London, 2013.
3. Kibble, Rodger. “Introduction to natural language processing, Weekly Slides 1-8.” Goldsmiths, University of London, 2015.