#Part 1. NLTK Pipeliner

* f = open('suess.txt')

raw = f.read()

print("Imported Text: " + raw + '\n')

Imported Text: I do not like green eggs and ham. I do not like them, Sam-I-am. I will not eat them in a box. I will not eat them with a fox.

* lemmer = nltk.WordNetLemmatizer()

lemmas= [lemmer.lemmatize(t) for t in text]

print("Lemmas: " + str(lemmas) + ‘\n’)

Lemmas: ['I', 'do', 'not', 'like', 'green', 'egg', 'and', 'ham', '.', 'I', 'do', 'not', 'like', 'them', ',', 'Sam-I-am', '.', 'I', 'will', 'not', 'eat', 'them', 'in', 'a', 'box', '.', 'I', 'will', 'not', 'eat', 'them', 'with', 'a', 'fox', '.']

* text = nltk.word\_tokenize(raw)

tagged = nltk.pos\_tag(text)

print('POS Tagging: ' + str(tagged) + '\n')

POS Tagging: [('I', 'PRP'), ('do', 'VBP'), ('not', 'RB'), ('like', 'IN'), ('green', 'JJ'), ('eggs', 'NNS'), ('and', 'CC'), ('ham', 'NN'), ('.', '.'), ('I', 'PRP'), ('do', 'VBP'), ('not', 'RB'), ('like', 'VB'), ('them', 'PRP'), (',', ','), ('Sam-I-am', 'NNP'), ('.', '.'), ('I', 'PRP'), ('will', 'MD'), ('not', 'RB'), ('eat', 'VB'), ('them', 'PRP'), ('in', 'IN'), ('a', 'DT'), ('box', 'NN'), ('.', '.'), ('I', 'PRP'), ('will', 'MD'), ('not', 'RB'), ('eat', 'VB'), ('them', 'PRP'), ('with', 'IN'), ('a', 'DT'), ('fox', 'NN'), ('.', '.')]

#Part 2. The Regular Expression Tagger.

* Just from using the regular expression tagger from the textbook, we get 8.006643196690336% accuracy. Which is pretty sorry. To try to bump this up a little I tried adding a few more regular expression searches, such as plural common genitive nouns, possessive pronouns, ordinal number words, reflexive pronouns and adverbs as well as 2 common noun suffixes. The results of which are visible below. The most effective of which seemed to be the ‘-tion’ suffix filter, upping the accuracy nearly 1 whole percent in itself, followed by the adverb filter, which raised the rating by almost .7 percent.

0.08006643196690336 – with no extras

0.08006643196690336 - plural common genitive nouns

0.08100125305805836 - ordinal number words

0.08100125305805836 - possessive pronouns

0.08135926964616028 - reflexive pronouns

0.08135926964616028 - singular adverbial genitive nouns

0.09050858245320922 - common nouns (-tion)

0.09763907949957237 - adverbs

0.09776836326749806 - more common nouns (-gy)

* The result without the backoff tagger maximally were 0.09776836326749806. Whereas with the backoff autotagger performance was more than doubled to reach 0.2097380511963721.

#Part 3. Lookup Tagger

Using the lookup tagger from the (latest version of the) book I constructed the lower left graph for the accuracy of the tagger based on the model size variable. It seems clear that the more samples the tagger is fed the effectiveness increases, although there seems to be the law of diminishing returns at play. The first few hundred benchmarks show dramatic increase in performance however several thousand more are needed to raise this number any more after about 2000.



Brown News Corpus - 94.069% Brown Adventure Corpus - 93.8337%

Using the same code for the Brown Advenure genre I noticed that performance was wuicker to increase in the initial increase of the model size however this meant a plateau effect appeared and again, the model had to increase by several orders of magnitude after about 2000 to increase much more than 85%

#Part 4 Conditional Frequency Distribution and Dictionaries

cfd = nltk.ConditionalFreqDist(brown.tagged\_words(categories='news'))

dict = {}

for key in cfd.keys():

if len(cfd[key]) > 1:

dict[key] = cfd.get(key)

print(repr(cfd.get('play')))

#FreqDist({'VB': 30, 'NN': 11})

print(str(len(dict)/len(cfd)))

#0.11970265388356259

The point of this exercise was to use see how many words in the tagged news corpus appeared to be used in more than one part of speech. That is, what percentage of words were used in more than way?

(For example, the word ‘play’ appears as a verb 33 times and a noun 11 times in the news corpus.) The result was 11.97%.