## Owen Galvin

## HU Extension Assignment 12 E63 Big Data Analytics

### Handed out: 04/23/2016 Due by 11:30PM EST on Friday, 04/29/2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board. If you use some other language other than Python in your daily work with NLP, please be free to use that language and a framework of your choice to do this assignment.

**Problem 1.** Create atable displaying **relative** frequencies with which “modals” (can, could, may, might, will, would and should) are used in 18 texts provided by NLTK in their extract from Gutenberg Corpus. For two modals with the largest span of relative frequencies (most used minus least used), select a text which uses it the most and the text that uses it the least. Compare usage in both texts by examining the concordances of those modals in two texts. Perhaps try to understand how are those words used in different texts.

**Solution:**

**I’ve read some of the questions on how to interpret this problem on the discussion boards and posted one of my own , I’ll go with the straightforward approach and just do what the problem states.**

**My code is being written and tested in Jupyter/IPython notebook format and I plan to submit it as such but there will also be a regular .py file as backup, because I haven’t used Jupyter much before this class.**

**Start with standard imports but alias gutenberg so I don’t need to worry about misspelling. Create a simple list of the seven modal text words that are being analyzed. Note that the modal values I’m using are all lower case and are being counted against the “as-is” text, i.e. the text used for conditional frequency has not been converted to lower-case. This means any occurrences of the modals that begin a sentence or are otherwise capitalized, will not be counted. I figure usage at the beginning of a sentence would differ from intra-sentence context. Create a books variable to hold list of 18 book text values that make up the gutenberg corpus. Then we have the piece of code that actually runs the analysis, populating the cfd variable with a collection of frequency distribution (FreqDist) objects. In this case the conditions will be the list of 18 books and for each of those we’ll get FreqDist containing a list of words and corresponding counts.**

|  |
| --- |
| import nltk  from nltk.corpus import gutenberg as gut  modals = ['can', 'could', 'may', 'might', 'will', 'would', 'should']  books = gut.fileids() #[:2]  cfd = nltk.ConditionalFreqDist(  (book, word)  for book in books  for word in gut.words(fileids=book) ) |

**To prepare for printing out a table of books and modals, get an offset equal to the longest book name + 5 char for some space. Beginning at that width, print the modals list going across, reserving 9 char for each value. Iterate through the keys of the cfd object, with each key being a different book/fileid. Print the book moniker, followed by the relative frequency for each of the modal values in that book. The list comprehension formats them to 6 decimals, basically allowing for auto-spacing - to get the values simple call .freq() on the underlying FreqDist object, passing in each of the modal text values.**

|  |
| --- |
| width = max(len(book) for book in cfd.iterkeys()) + 5  print ''.ljust(width) + ''.join(modal.ljust(9) for modal in modals)  for book in cfd.iterkeys():  print book.ljust(width) + ' '.join(['{0:.6f}'.format(cfd[book].freq(modal)) for modal in modals]) |

**Below we see the table output by above code.**

|  |
| --- |
| can could may might will would should  milton-paradise.txt 0.001105 0.000640 0.001198 0.001012 0.001663 0.000506 0.000568  shakespeare-macbeth.txt 0.000908 0.000648 0.001296 0.000216 0.002679 0.001815 0.001772  austen-emma.txt 0.001403 0.004287 0.001107 0.001673 0.002905 0.004235 0.001902  chesterton-ball.txt 0.001351 0.001206 0.000928 0.000711 0.002041 0.001433 0.000773  bible-kjv.txt 0.000211 0.000163 0.001013 0.000470 0.003767 0.000438 0.000760  chesterton-thursday.txt 0.001690 0.002138 0.000809 0.001026 0.001575 0.001676 0.000780  blake-poems.txt 0.002394 0.000359 0.000599 0.000239 0.000359 0.000359 0.000718  shakespeare-caesar.txt 0.000619 0.000697 0.001355 0.000465 0.004994 0.001548 0.001471  whitman-leaves.txt 0.000568 0.000316 0.000549 0.000168 0.001685 0.000549 0.000271  melville-moby\_dick.txt 0.000843 0.000824 0.000882 0.000702 0.001453 0.001614 0.000694  austen-persuasion.txt 0.001019 0.004523 0.000886 0.001691 0.001650 0.003575 0.001884  edgeworth-parents.txt 0.001614 0.001994 0.000760 0.000603 0.002454 0.002388 0.001286  carroll-alice.txt 0.001671 0.002140 0.000322 0.000821 0.000704 0.002052 0.000792  bryant-stories.txt 0.001350 0.002772 0.000324 0.000414 0.002592 0.001980 0.000684  burgess-busterbrown.txt 0.001213 0.002953 0.000158 0.000896 0.001002 0.002426 0.000686  chesterton-brown.txt 0.001464 0.001975 0.000546 0.000825 0.001290 0.001534 0.000651  shakespeare-hamlet.txt 0.000883 0.000696 0.001499 0.000749 0.003506 0.001606 0.001392  austen-sense.txt 0.001455 0.004012 0.001194 0.001519 0.002500 0.003581 0.001610 |

**A quick sanity check on above, to make sure that a simple FreqDist() on one of the texts comes up with a value for ‘can’ that matches same cell in above table, which it does. And a couple of more lines to make sure I understand the cfd object and can manipulate it.**

|  |
| --- |
| #sanity check  emma = gut.words('austen-emma.txt')  fdist\_emma = nltk.FreqDist(emma)  print fdist\_emma.freq('can')  print cfd['austen-emma.txt'].freq('can')  print cfd['austen-emma.txt']['can']/(len(emma) \* 1.0) |

**Output of above.**

|  |
| --- |
| 0.00140312949846  0.00140312949846  0.00140312949846 |

**Next up is the code that finds the two modals with the largest span of relative frequencies. The rng\_lookup var is defined as empty dictionary. Then loop through the list of modals:**

1. **Use max() on collection of iterkeys() as sorted by the modal frequency belonging to each book. Previously I’ve used key param when sorting lists, along with operator.itemgetter() but the lambda syntax is very clean – get the FreqDist for each book and call freq(), passing in the current modal. That winds up being the value upon which max() is really based, even though the output is a “book”.**
2. **Same syntax, but for min()**
3. **Now calculate the range aka span, between min and max frequency distribution for current modal.**
4. **And create a key in rng\_lookup dictionary = book name, assign value = tuple of above span + book with min frequency distribution + book with max.**
5. **Then print out the current modal, range, and other debug info to make sure the final answer is actually correct.**

**After loop has completed do a sort on the dictionary holding {modal: (range, min\_book, max\_book)}, sorting on the range value, though in reverse in order to get largest ranges at the front. Do simple slice [:2] to grab the top two dictionary keys (modal values) and reach back into dictionary to print out associated tuple.**

|  |
| --- |
| rng\_lookup = {}  for modal in modals:  max\_modal\_book = max(cfd.iterkeys(), key = (lambda key: cfd[key].freq(modal)))  min\_modal\_book = min(cfd.iterkeys(), key = (lambda key: cfd[key].freq(modal)))  rng = cfd[max\_modal\_book].freq(modal) - cfd[min\_modal\_book].freq(modal)  rng\_lookup[modal] = (rng, min\_modal\_book, max\_modal\_book )  print 'modal: "{0}" range, max - min: {1:.6f}'.format(  modal, rng)  print '\t {0} only has {1} relative freq'.format(min\_modal\_book, cfd[min\_modal\_book].freq(modal))  print '\t {0} has most used relative freq, at {1}'.format(max\_modal\_book, cfd[max\_modal\_book].freq(modal))  print  print 'Two modals with with the largest span of relative frequencies:'  for k in sorted(rng\_lookup, key = (lambda key: rng\_lookup[key][0]), reverse = True)[:2]:  print '{0}: {1}'.format(k, rng\_lookup[k]) |

**Print out of above where the code finds “will” and “could” to be the two modals with largest range of frequency distributions. First book in each line, after the range, has the lowest freq dist and is followed by the book with highest freq dist for that modal.**

|  |
| --- |
| modal: "can" range, max - min: 0.002183  bible-kjv.txt only has 0.000210754620276 relative freq  blake-poems.txt has most used relative freq, at 0.00239406272444  modal: "could" range, max - min: 0.004359  bible-kjv.txt only has 0.00016326062134 relative freq  austen-persuasion.txt has most used relative freq, at 0.00452272055902  modal: "may" range, max - min: 0.001341  burgess-busterbrown.txt only has 0.00015820281601 relative freq  shakespeare-hamlet.txt has most used relative freq, at 0.00149892933619  modal: "might" range, max - min: 0.001523  whitman-leaves.txt only has 0.000167868649238 relative freq  austen-persuasion.txt has most used relative freq, at 0.00169092705585  modal: "will" range, max - min: 0.004635  blake-poems.txt only has 0.000359109408667 relative freq  shakespeare-caesar.txt has most used relative freq, at 0.00499361282081  modal: "would" range, max - min: 0.003876  blake-poems.txt only has 0.000359109408667 relative freq  austen-emma.txt has most used relative freq, at 0.00423537237498  modal: "should" range, max - min: 0.001631  whitman-leaves.txt only has 0.000271172433385 relative freq  austen-emma.txt has most used relative freq, at 0.0019020199868  Two modals with with the largest span of relative frequencies:  will: (0.004634503412144084, u'blake-poems.txt', u'shakespeare-caesar.txt')  could: (0.004359459937684319, u'bible-kjv.txt', u'austen-persuasion.txt') |

**The problem stated “For two modals with the largest span…** **select a text which uses it the most and the text that uses it the least”, so I take that to mean I should examine statistics on “will” and “could” within min & max books my code returned. First up is “will”, which had highest span. Create variables for each of the books, poems with min frequency distribution and caesar with max. For each book print out 10 lines of concordance, displaying the context for “will”, i.e. with surrounding words. Then in an attempt to understand how those words are used in the different texts, use similar() function to see what other words appear in a similar range of contexts. Take the first word returned by similar() and use common\_contexts() to see how “will” and that similar word (“have” for poems, “shall” for caesar) appear in similar contexts within their respective books.**

|  |
| --- |
| # modal = 'will', highest relative frequency difference across gutenberg corpus: 0.0046345  word = 'will'  poems = nltk.Text(gut.words('blake-poems.txt'))  caesar = nltk.Text(gut.words('shakespeare-caesar.txt'))  books = [poems, caesar]  for book in books:  print '>>> Concordance for "{0}" in {1}'.format(word, book)  book.concordance(word, lines=10)  print  #can't really modularize since can't capture output of .similar() w/o redirecting stdout... not worth the trouble  print '>>> words in {0} with similar context to "{1}"'.format(poems, word)  poems.similar(word)  sim = 'have'  print '>>> context of word in {0} with similar context to "{1}"'.format(poems, sim)  poems.common\_contexts([word, sim])  print  print '>>> words in {0} with similar context to "{1}"'.format(caesar, word)  caesar.similar(word)  sim = 'shall'  print '>>> context of word in {0} with similar context to "{1}"'.format(caesar, sim)  caesar.common\_contexts([word, sim]) |

**Below is a printout of above. Partly due to the brevity of the poems and partly due to the archaic language of the texts I’m not able to really understand how the modal is used differently in the two. Of course there also may not be any identifiable pattern.**

|  |
| --- |
| >>> Concordance for "will" in <Text: Poems by William Blake 1789>  Displaying 3 of 3 matches:  arn ' d the heat to bear , The cloud will vanish , we shall hear His voice , S  lver hair , And be like him , and he will then love me . THE BLOSSOM Merry , m  alone nor or itself : fear not and I will call , The weak worm from its lowly  >>> Concordance for "will" in <Text: The Tragedie of Julius Caesar by William Shakespeare 1599>  Displaying 10 of 163 matches:  way towards the Capitoll , This way will I : Disrobe the Images , If you do f  eathers , pluckt from Caesars wing , Will make him flye an ordinary pitch , Wh  eunt . Manet Brut . & Cass . Cassi . Will you go see the order of the course ?  , That you haue no such Mirrors , as will turne Your hidden worthinesse into y  l as by Reflection ; I your Glasse , Will modestly discouer to your selfe That  eye , and Death i ' th other , And I will looke on both indifferently : For le  s heauy : Coniure with ' em , Brutus will start a Spirit as soone as Caesar .  er moou ' d : What you haue said , I will consider : what you haue to say I wi  ll consider : what you haue to say I will with patience heare , and finde a ti  Plucke Caska by the Sleeue , And he will ( after his sowre fashion ) tell you  >>> words in <Text: Poems by William Blake 1789> with similar context to "will"  have  >>> context of word in <Text: Poems by William Blake 1789> with similar context to "have"  i\_call  >>> words in <Text: The Tragedie of Julius Caesar by William Shakespeare 1599> with similar context to "will"  shall would did know should must do house are be capitoll were if  looke hands streetes name found leaue like  >>> context of word in <Text: The Tragedie of Julius Caesar by William Shakespeare 1599> with similar context to "shall"  you\_not it\_please i\_not that\_be i\_be we\_be caesar\_not i\_see he\_be |

**The comparison of the modal with the next-largest span is essentially identical to the first one above, except of course the most/least books are changing in addition to the modal itself. So below is for “could” with lowest relative frequency belonging to king james bible and most to Austen’s persuasion. The corresponding similar words in the two texts are “have” and “would ” respectively.**

|  |
| --- |
| # modal = 'could', 2nd highest relative frequency difference across gutenberg corpus: 0.00435946  (0.004359459937684319, u'bible-kjv.txt', u'austen-persuasion.txt')  word = 'could'  bible = nltk.Text(gut.words('bible-kjv.txt'))  persuasion = nltk.Text(gut.words('austen-persuasion.txt'))  books = [bible, persuasion]  for book in books:  print '>>> Concordance for "{0}" in {1}'.format(word, book)  book.concordance(word, lines=10)  print  #can't really modularize since can't capture output of .similar() w/o redirecting stdout... not worth the trouble  print '>>> words in {0} with similar context to "{1}"'.format(bible, word)  bible.similar(word)  sim = 'have'  print '>>> context of word in {0} with similar context to "{1}"'.format(bible, sim)  bible.common\_contexts([word, sim])  print  print '>>> words in {0} with similar context to "{1}"'.format(persuasion, word)  persuasion.similar(word)  sim = 'would'  print '>>> context of word in {0} with similar context to "{1}"'.format(persuasion, sim)  persuasion.common\_contexts([word, sim]) |

**Printout for “could” comparison. Apparently KJV of the bible was translated around 200 years before Austen wrote Persuasion and we can see the that language surrounding “could” in the bible is much more archaic vs. Persuasion, where the latter doesn’t have the more formal language. The KJV concordance (and a longer sampling I output initially) does show that “could not” bigram was much more common than any other “could” + 1 bigram. I would not be surprised to see a similar pattern in the bible for other modals, the phrase “shalt not” certainly comes to mind.**

|  |
| --- |
| >>> Concordance for "could" in <Text: The King James Bible>  Displaying 10 of 166 matches:  r substance was great , so that they could not dwell together . 13 : 7 And ther  , and his eyes were dim , so that he could not see , he called Esau his eldest  the land wherein they were strangers could not bear them because of their cattl  his brethren , they hated him , and could not speak peaceably unto him . 37 :  his dream ; but there was none that could interpret them unto Pharaoh . 41 : 9  And when they had eaten them up , it could not be known that they had eaten the  magicians ; but there was none that could declare it to me . 41 : 25 And Josep  ording to the tenor of these words : could we certainly know that he would say  me on my father . 45 : 1 Then Joseph could not refrain himself before all them  y father yet live ? And his brethren could not answer him ; for they were troub  >>> Concordance for "could" in <Text: Persuasion by Jane Austen 1818>  Displaying 10 of 451 matches:  every other leaf were powerless , he could read his own history with an interes  as still a very fine man . Few women could think more of their personal appeara  ersonal appearance than he did , nor could the valet of any new made lord be mo  l ; but it was only in Anne that she could fancy the mother to revive again . A  mild dark eyes from his own ), there could be nothing in them , now that she wa  ood looks of everybody else ; for he could plainly see how old all the rest of  self - possession and decision which could never have given the idea of her bei  heir , and whose strong family pride could see only in him a proper match for S  aronet from A to Z whom her feelings could have so willingly acknowledged as an  ing black ribbons for his wife , she could not admit him to be worth thinking o  >>> words in <Text: The King James Bible> with similar context to "could"  shall will is had should do be would doth was did hath may have said  are can fear were might  >>> context of word in <Text: The King James Bible> with similar context to "have"  we\_no i\_not and\_not which\_not neither\_they brethren\_not that\_not  ye\_not what\_ye i\_no men\_not israel\_not they\_not  >>> words in <Text: Persuasion by Jane Austen 1818> with similar context to "could"  would had was should might must did may will can is to cannot have  does knew i shall do were  >>> context of word in <Text: Persuasion by Jane Austen 1818> with similar context to "would"  he\_no it\_not anne\_have she\_feel there\_be who\_not i\_not she\_only  she\_not he\_be it\_be they\_have russell\_not who\_be wentworth\_not she\_be  nothing\_be she\_see you\_have they\_not |

**Problem 2**. In the Inaugural corpus identify 10 most frequently used words longer than 7 characters. Which one of those has the largest number of synonyms? List all synonyms for those 10 words. Which one of those 10 words has the largest number of hyponyms? List all hyponyms of those 10 most frequently used “long” words.

**Solution:**

**After importing nltk and the inaugural module holding the text in question, create a generator that holds all words longer than 7 characters. Force them to lower case since I think it makes sense for this assignment to ignore casing when comparing words. The problem only concerns raw counts, so don’t need to worry about “different” relative frequencies of the long words, i.e. in the full text vs. within the long words subset. Create a FreqDist on these long words, to get nltk to perform its word counting function. Similar to some of the sorting in previous problem, get a list of words returned (the .keys() of the FreqDist) but sorted by the value passed into the key argument of sorted(). Here it is the word count returned by the lookup of the word on the fidist object. Also use reverse=True argument so that words with highest count come first. Perform a slice on the sorted list so that it only pulls in the first 10 words. After that iterate through the list of long words, pulling in the word count from fdist once more for display purposes.**

|  |
| --- |
| import nltk  from nltk.corpus import inaugural  #choose to lower-case all words to avoid casing issues  long\_words = (w.lower() for w in inaugural.words() if len(w) > 7)  fdist = nltk.FreqDist(long\_words)  #textbook indicated a FreqDist came sorted, with most frequent counts first, but I didn't see that  freq\_long\_words = sorted(fdist.keys(), key = lambda w: fdist[w], reverse = True)[:10]  long\_words\_with\_counts = ['{0}: {1}'.format(k, fdist[k]) for k in freq\_long\_words]  print '\r\n'.join(long\_words\_with\_counts) |

**Print out of above code. Including word counts wasn’t part of the assignment but it helps affirm the sorting has gone correctly.**

|  |
| --- |
| government: 593  citizens: 237  constitution: 205  national: 154  american: 147  congress: 129  interests: 113  political: 106  executive: 97  principles: 93 |

**The next task is to find the “long word” with the most synoms, which is accomplished with a defaultdict object instanciated with a set object. That way the value for a given key (where the key will be the original word) can be treated as a set even if it has not yet been created. Iterate through the words and for each call the synsets() function of WordNet, which in turn will return a series of Synset objects, more or less representing different meaning so the word. For each of those pull out the lemma\_names, which will be the simple text values that can be considered synonyms. The come out as a list and use .update() to essentially append that list to the value, a set object. As a set, the value will never contain any duplicate values. Once the syns dictionary is populated, pull out the max synonym, where max is measured in terms of the count of values for each long word, similar to syntax used in Problem 1. Second part of the task is to print out all the synonyms for each long word. Using the syns dictionary, this is simple enough, key = word, values = synonyms.**

|  |
| --- |
| syns = defaultdict(set)  for w in freq\_long\_words:  for ss in wn.synsets(w):  names = ss.lemma\_names()  syns[w].update(names)  most\_synonyms = max(syns.keys(), key = lambda k: len(syns[k]))  print 'Word with most synonyms: "{0}", count: {1}'.format(most\_synonyms, len(syns[most\_synonyms]))  print  print '"WORD"'.ljust(15) + 'SYNONYMS'  for k,v in syns.items():  print '"{0}"'.format(k).ljust(15) + ', '.join(v)  print |

**Print out of above, was able to do color in Jupyter notebook to distinguish words vs. synonyms but that broke the .ljust() and wasn’t worth troubleshooting.**

|  |
| --- |
| Word with most synonyms: "constitution", count: 17  "WORD" SYNONYMS  "interests" sake, stake, interest\_group, matter\_to, interestingness, occupy, involvement, pursuit, interest, pastime, worry, concern  "executive" administrator, executive, executive\_director  "constitution" make-up, makeup, constitution, US\_Constitution, organisation, composition, Constitution\_of\_the\_United\_States, fundamental\_law, physical\_composition, formation, U.S.\_Constitution, Old\_Ironsides, organization, United\_States\_Constitution, establishment, organic\_law, Constitution  "congress" Congress, congress, sexual\_intercourse, sex\_act, coition, sexual\_relation, carnal\_knowledge, intercourse, sexual\_congress, relation, U.S.\_Congress, copulation, coitus, United\_States\_Congress, US\_Congress  "government" government, administration, governance, political\_science, government\_activity, governing, authorities, politics, regime  "national" interior, home, national, internal, subject  "citizens" citizen  "political" political  "principles" precept, rationale, principle, rule  "american" American, American\_English, American\_language |

**Final part of Problem 2 is to essentially perform above tasks but in terms of hyponyms this time. The general loop code is quite similar to that used for synonyms only as the collection of Synsets is iterated over, the hyponyms are pulled out as a further collection of Synsets. Call .lemma\_names() on each these to get the actual textual representation of the related hyponyms – add these to the set() belonging to that “long word”. At this point the logic for getting word with largest number of hyponyms is identical to that for synonyms. The printout of the actual hyponyms is similar also, though it should output only 5 on a line, to make things cleaner. (I did consider a recursive function to get all the hyponyms of the first layer of hyponyms and then go deeper as necessary but that seemed a little much and put my attempt to the side to get things tied up.)**

|  |
| --- |
| hypo\_dict = defaultdict(set)  for w in freq\_long\_words:  for ss in wn.synsets(w):  for hyp in ss.hyponyms():  names = hyp.lemma\_names()  hypo\_dict[w].update(names)  most\_hyponyms = max(hypo\_dict.keys(), key = lambda k: len(hypo\_dict[k]))  print 'Word with most hyponyms: "{0}", count: {1}'.format(most\_hyponyms, len(hypo\_dict[most\_hyponyms]))  print  for k,v in hypo\_dict.items():  print 'WORD: "{0}"'.format(k)  print 'HYPONYMS: '  for i in range(0, len(v), 5):  print ''.ljust(5) + ', '.join(list(v)[i:i+5])  print |

**Output from above.**

|  |
| --- |
| Word with most hyponyms: "american", count: 109  WORD: "interests"  HYPONYMS:  right, hobby, transfix, color, by-line  special\_interest, vested\_interest, avocation, newsworthiness, controlling\_interest  compound\_interest, concern, fee, occupy, charisma  equity, enthusiasm, grip, security\_interest, reversion  absorb, topicality, grubstake, sideline, pursuit  undivided\_interest, news, spellbind, insurable\_interest, behalf  vividness, engage, simple\_interest, fascinate, spare-time\_activity  colour, shrillness, terminable\_interest, engross, personal\_magnetism  personal\_appeal, undivided\_right, intrigue  WORD: "executive"  HYPONYMS:  triumvir, Carter\_administration, Clinton\_administration, Bush\_administration, Reagan\_administration  government\_minister, DCI, prefect, Director\_of\_Central\_Intelligence, commissioner  Secretary\_General, Surgeon\_General, vice\_president, corporate\_executive, minister  V.P., business\_executive, rainmaker  WORD: "constitution"  HYPONYMS:  karyotype, federation, colonisation, collectivization, texture  colonization, unionization, grain, phenotype, structure  communisation, collectivisation, settlement, unionisation, genetic\_constitution  communization, genotype  WORD: "congress"  HYPONYMS:  ass, screw, piece\_of\_tail, Continental\_Congress, fucking  piece\_of\_ass, shtup, screwing, fuck, hank\_panky  penetration, roll\_in\_the\_hay, criminal\_congress, unlawful\_carnal\_knowledge, nookie  defloration, nooky, shag  WORD: "government"  HYPONYMS:  palace, geopolitics, ancien\_regime, papacy, authoritarian\_regime  empire, pupet\_regime, misrule, court, puppet\_government  misgovernment, legislation, state, lawmaking, realpolitik  puppet\_state, royal\_court, bureaucracy, military\_government, local\_government  stratocracy, authoritarian\_state, Downing\_Street, government-in-exile, legislating  practical\_politics, federal\_government, pontificate, totalitarian\_state, totalitation\_regime  trust\_busting, state\_government  WORD: "national"  HYPONYMS:  nationalist, compatriot, patriot, citizen  WORD: "citizens"  HYPONYMS:  freeman, thane, freewoman, civilian, private\_citizen  repatriate, active\_citizen, voter, elector  WORD: "principles"  HYPONYMS:  fundamental\_principle, Tao, chivalry, accounting\_principle, Gestalt\_principle\_of\_organization  Le\_Chatelier\_principle, mass-energy\_equivalence, localization\_principle, Le\_Chatelier-Braun\_principle, localisation\_of\_function  Ockham's\_Razor, pleasure-pain\_principle, value-system, Hellenism, dictate  principle\_of\_superposition, judicial\_principle, hypothetical\_imperative, insurrectionism, ethic  pleasure-unpleasure\_principle, principle\_of\_liquid\_displacement, caveat\_emptor, yin, higher\_law  Huygens'\_principle\_of\_superposition, conservation, value\_orientation, moral\_principle, mass-action\_principle  Naegele's\_rule, legal\_principle, Gresham's\_Law, localization\_of\_function, bedrock  localization, fundamentals, pleasure\_principle, localisation\_principle, dialectics  localisation, principle\_of\_parsimony, judicial\_doctrine, superposition, feng\_shui  scruple, basic\_principle, accounting\_standard, basics, superposition\_principle  Gestalt\_law\_of\_organization, pillar, reality\_principle, law\_of\_parsimony, Le\_Chatelier's\_principle  principle\_of\_equivalence, Occam's\_Razor, yang, mass\_action, logic  knightliness, Le\_Chatelier's\_law  WORD: "american"  HYPONYMS:  Black\_English\_Vernacular, Texan, North\_Carolinian, Southerner, Black\_Vernacular  Kansan, Yank, Volunteer, African\_American\_Vernacular\_English, Virginian  Nisei, Oregonian, Idahoan, Floridian, Utahan  New\_Yorker, North\_American, Illinoisan, Granite\_Stater, Puerto\_Rican  Badger, Delawarian, Afro-American, South\_Carolinian, Arizonan  Kentuckian, Coloradan, Carolinian, Keystone\_Stater, Mainer  Rhode\_Islander, North\_Dakotan, Hawaiian, New\_Englander, Georgian  Pennsylvanian, Alabaman, Bostonian, Ohioan, Garden\_Stater  Northerner, Wyomingite, Hispanic\_American, Black\_English, Bay\_Stater  Arkansawyer, New\_Jerseyan, Latin\_American, Alabamian, AAVE  Beaver, Yankee-Doodle, Louisianian, Arizonian, Arkansan  New\_Hampshirite, West\_Virginian, Michigander, Louisianan, Creole  Tennessean, Anglo-American, African-American, Hoosier, Vermonter  Latino, Mississippian, Missourian, Hispanic, Ebonics  Asian\_American, Connecticuter, Californian, Wolverine, Minnesotan  Appalachian, Nebraskan, Gopher, African\_American\_English, Buckeye  Montanan, Bluegrass\_Stater, Yankee, Mesoamerican, Iowan  Wisconsinite, Down\_Easter, Tory, Delawarean, African\_American  Sooner, German\_American, Washingtonian, Franco-American, Indianan  Black\_American, Marylander, Alaskan, Spanish\_American, South\_American  Cornhusker, Nevadan, Oklahoman, West\_Indian, Black\_Vernacular\_English  New\_Mexican, Tarheel, New\_Jerseyite, South\_Dakotan |

**~~Problem 3.~~** ~~Create for us one graph displaying cumulative word length distribution for six different genres in Brown corpus. Create a tabular display of basic word statistics for all genres in Brown corpus. Include: average word length, average sentence length, number of concurrences in each genre, percentage of the text consumed by conditional words: would, could and should.~~

You literature for this assignment are chapters 1 and 2 of Natural Language Processing with Python book by Steven Bird et al.