Data in Systems Sciences

Part 2 - Text mining

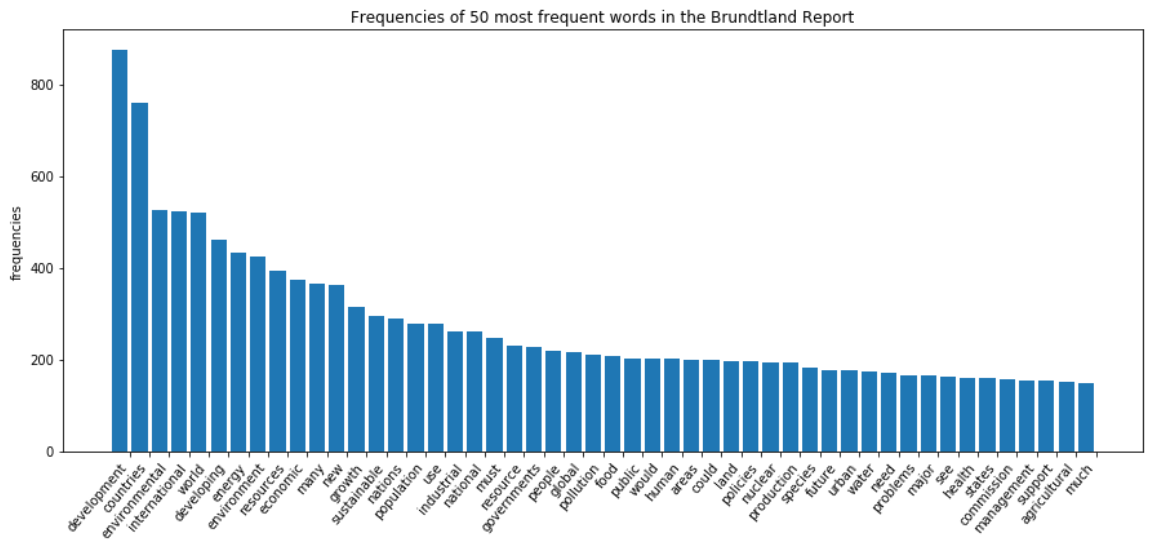
In [88]:

```python
# import necessary libraries
# for numerical calculations
import numpy as np
# for text editing
import re
import os, string
from string import digits
# for text analysis
import nltk
from nltk.corpus import stopwords
# if first time
#nltk.download('stopwords')
#nltk.download('punkt')
from nltk.stem.snowball import EnglishStemmer as ES

# plotting
import matplotlib.pyplot as plt
%matplotlib inline

# network analysis
import networkx as nx

#function to read content
def read_content(filename):
    fdcontent = open(filename, 'r')
    content = []
    for line in fdcontent:
        line = line.replace("u'","")
        line = line.replace('u"',"")
        line = line.replace("\n","")
        line = line.replace("\r","")
        line = line.replace("\r","")
        line = line.replace("'","")
        content.append(line)
    fdcontent.close()
    return content


# load content
datei = 'Brundtlandreport'
documents = read_content('C:\\temp\\Text-mining\\' + datei + '.txt')
#print(documents)      #### careful, this is large!

#filter out punctuation, digits and special characters
delete_table = str.maketrans(string.punctuation + string.ascii_uppercase,
                             " " * len(string.punctuation) + string.ascii_lowercase)
#print(delete_table)

digits_table= str.maketrans('', '', digits)


# remove common words and tokenize
stope = stopwords.words("english")
#stopg = stopwords.words("german")

#stoplist can be extended like this:
stope.extend(['et', 'al', 'per', 'cent', 'one', 'may', 'also'])

#texts are cleaned (characters only), filtered (stopwords removed) and stemmed (reduced
to word stem)
#texts = [[ES().stem(word.translate(delete_table)) for word in document.lower().split()
if word.translate(delete_table)
#          not in stope] for document in documents]

# without stemming
texts = [[word.translate(delete_table) for word in document.lower().split()
          if word.translate(delete_table) not in stope and len((word.translate(delete_t
able))) > 2] for document in documents]

#texts = filter(lambda c: not c.isdigit(), texts)
#print(texts)

# without stopwords exclusion
#texts = [[word.translate(delete_table) for word in document.lower().split()] for docum
ent in documents]
# print(texts)

# flatten texts (in case they should be in the form of lists within lists)
flattened_texts = [val for sublist in texts for val in sublist]
# use the digits_table from above to remove digits
flattened_texts = [word.translate(digits_table) for word in flattened_texts]

# alter texts from list to words
texts = (" ".join(flattened_texts))
#print(texts)

# alter texts from words to tokens and on to nltk.Text
tokens = nltk.word_tokenize(texts)
texts = nltk.Text(tokens)
print(len(texts))
print(texts[2090:2200])
```

```
78215
['size', 'india', 'much', 'forest', 'converted', 'low', 'grade', 'farmlan
d', 'unable', 'support', 'farmers', 'settle', 'it', 'europe', 'acid', 'pre
cipitation', 'kills', 'forests', 'lakes', 'damages', 'artistic', 'architec
tural', 'heritage', 'nations', 'acidified', 'vast', 'tracts', 'soil', 'bey
ond', 'reasonable', 'hope', 'repair', 'burning', 'fossil', 'fuels', 'put
s', 'atmosphere', 'carbon', 'dioxide', 'causing', 'gradual', 'global', 'wa
rming', 'greenhouse', 'effect', 'early', 'next', 'century', 'increased',
'average', 'global', 'temperatures', 'enough', 'shift', 'agricultural', 'p
roduction', 'areas', 'raise', 'sea', 'levels', 'flood', 'coastal', 'citie
s', 'disrupt', 'national', 'economies', 'industrial', 'gases', 'threaten',
'deplete', 'planets', 'protective', 'ozone', 'shield', 'extent', 'number',
'human', 'animal', 'cancers', 'would', 'rise', 'sharply', 'oceans', 'foo
d', 'chain', 'would', 'disrupted', 'industry', 'agriculture', 'put', 'toxi
c', 'substances', 'human', 'food', 'chain', 'underground', 'water', 'table
s', 'beyond', 'reach', 'cleansing', 'growing', 'realization', 'national',
'governments', 'multilateral', 'institutions', 'impossible', 'separate',
'economic']
```

In [89]:

```
# most frequent words and their frequencies = frequency distribution
fdist = nltk.FreqDist(texts)

# transform into tuples
tuples = zip([x for x in fdist],[y for y in fdist.values()])
sorted_tuples = sorted(tuples, key=lambda x: x[1], reverse = True)
#print(len(sorted_tuples))

# save sorted_tuples to file
#f = open( 'C:\\...\\sorted_tuples', 'w' )
#f.write( repr(sorted_tuples))
#f.close()

# exclude stopwords
w =  [x[0] for x in sorted_tuples if x[0] not in stope]
fr = [x[1] for x in sorted_tuples if x[0] not in stope]

von = 0
bis = 50
numwords = bis - von
most_freq_words = w[von:bis]
frequencies = fr[von:bis]

# plot frequency distribution
fig = plt.figure(figsize=(15, 6))
x = range(numwords)
plt.bar(x, frequencies)
# set ticks position 0.5 to the right, so that labels are centered
xticks_pos = np.arange(0.5, numwords + 0.5, 1)
# specify a rotation for the tick labels in degrees or with keywords (e.g.'vertical');
 ha = horizontal alignment of ticklabels
plt.xticks(xticks_pos, most_freq_words, rotation='50', ha = 'right')
#plt.title('Frequencies of %s to %s most frequent words in %s' %(von, bis, datei))
plt.title('Frequencies of %s most frequent words in the Brundtland Report' %(numwords))
plt.ylabel('frequencies')
# Plot margins so that markers don't get clipped by the axes
plt.margins(0.05)
# Tweak spacing to prevent clipping of tick-labels; can also be done with figsize
#plt.subplots_adjust(bottom = 0.5)
plt.show()
```



In [90]:

```
from __future__ import division

# how many times does word x appear
def appearance(word):
    return texts.count(word)

print(appearance('sustainable'))
#print(appearance('energy'))

def percentage(count, total):
    return (100 * count / total)

# what percentage of the text does word x take up
print(percentage(texts.count('environmental'), len(texts)))
print(percentage(texts.count('waste'), len(texts)))
```

```
294
0.672505273924439
0.07671162820430864
```

In [91]:

```
# consider context -1 and 1 (= adjacents) and loop through most frequent words to show
 how their context is interconnected

def determine_adjacents(word, k):
    # finding the words that surround the concordanced word in distance dis
    c = nltk.ConcordanceIndex(texts.tokens, key = lambda s: s.lower())
    # the words around the concordanced word, -1 = before, + 1 = after the word
    b = [texts.tokens[offset - k] for offset in c.offsets(word)]
    a = [texts.tokens[offset + k] for offset in c.offsets(word)]
    #print(a)

    # remove duplicates and stopwords
    b_cleared = []
    for x in b:
        if x not in b_cleared and x not in stope:
            b_cleared.append(x)
    a_cleared = []
    for x in a:
        if x not in a_cleared and x not in stope:
            a_cleared.append(x)

    # count number of occurrences of contextwords in a, b
    b_occurrences = [b.count(item) for item in b_cleared]
    a_occurrences = [a.count(item) for item in a_cleared]

    # append distance indicators -1 and +1 and number of occurrences
    j = 0
    before = []
    after = []
    i = 0
    for item in b_cleared:
        before.append((item, -k, b_occurrences[i]))
        i += 1
    j = 0
    for item in a_cleared:
        after.append((item, +k, a_occurrences[j]))
        j += 1
    j += 1

    # take maximum
    sorted_before = sorted(before,key = lambda x: x[2], reverse = True)
    max_before = sorted_before[0]
    sorted_after = sorted(after,key = lambda x: x[2], reverse = True)
    max_after = sorted_after[0]
    max_adjacent = (max_before + max_after)
    dict_adjacent = {word: max_adjacent}
    return dict_adjacent

# word = the concordanced word
# k = the number of occurences considered
max_adjacents = []
for item in most_freq_words:
    try:
        max_adjacents.append(determine_adjacents(item, 1))
    except:
        stope.append(item)
        continue

print(max_adjacents)
```

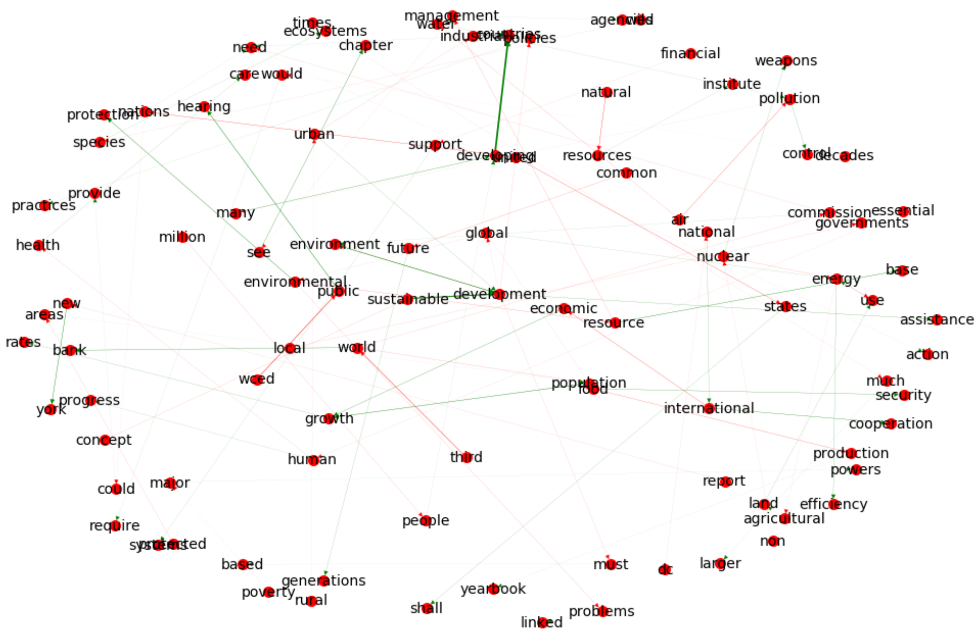
```
[{'development': ('sustainable', -1, 198, 'assistance', 1, 32)}, {'countri
es': ('developing', -1, 341, 'need', 1, 9)}, {'environmental': ('global',
-1, 7, 'protection', 1, 61)}, {'international': ('national', -1, 26, 'coop
eration', 1, 52)}, {'world': ('third', -1, 73, 'bank', 1, 51)}, {'developi
ng': ('many', -1, 45, 'countries', 1, 341)}, {'energy': ('nuclear', -1, 1
9, 'efficiency', 1, 29)}, {'environment': ('development', -1, 15, 'develop
ment', 1, 95)}, {'resources': ('natural', -1, 56, 'institute', 1, 13)},
{'economic': ('international', -1, 41, 'growth', 1, 42)}, {'many': ('pollu
tion', -1, 5, 'developing', 1, 45)}, {'new': ('report', -1, 11, 'york', 1,
71)}, {'growth': ('population', -1, 70, 'rates', 1, 23)}, {'sustainable':
('concept', -1, 15, 'development', 1, 198)}, {'nations': ('united', -1, 4
5, 'need', 1, 6)}, {'population': ('world', -1, 20, 'growth', 1, 70)}, {'u
se': ('energy', -1, 20, 'energy', 1, 8)}, {'industrial': ('urban', -1, 12,
'countries', 1, 64)}, {'national': ('dc', -1, 8, 'international', 1, 26)},
{'must': ('development', -1, 12, 'based', 1, 6)}, {'resource': ('environme
ntal', -1, 27, 'base', 1, 60)}, {'governments': ('local', -1, 12, 'need',
1, 11)}, {'people': ('million', -1, 14, 'developing', 1, 8)}, {'global':
('development', -1, 8, 'energy', 1, 16)}, {'pollution': ('air', -1, 37, 'c
ontrol', 1, 26)}, {'food': ('global', -1, 7, 'security', 1, 39)}, {'publi
c': ('wced', -1, 91, 'hearing', 1, 79)}, {'would': ('decades', -1, 3, 'req
uire', 1, 9)}, {'human': ('essential', -1, 8, 'progress', 1, 16)}, {'area
s': ('protected', -1, 14, 'population', 1, 4)}, {'could': ('nations', -1,
5, 'provide', 1, 6)}, {'land': ('agricultural', -1, 10, 'use', 1, 18)},
{'policies': ('development', -1, 13, 'practices', 1, 8)}, {'nuclear': ('no
n', -1, 8, 'weapons', 1, 22)}, {'production': ('food', -1, 33, 'yearbook',
1, 5)}, {'species': ('wild', -1, 8, 'ecosystems', 1, 15)}, {'future': ('co
mmon', -1, 24, 'generations', 1, 23)}, {'urban': ('rural', -1, 6, 'develop
ment', 1, 20)}, {'water': ('air', -1, 17, 'pollution', 1, 13)}, {'need':
('governments', -1, 11, 'action', 1, 4)}, {'problems': ('environmental', -
1, 14, 'linked', 1, 3)}, {'major': ('poverty', -1, 3, 'powers', 1, 6)},
{'see': ('management', -1, 5, 'chapter', 1, 47)}, {'health': ('human', -1,
12, 'care', 1, 15)}, {'states': ('united', -1, 41, 'shall', 1, 19)}, {'com
mission': ('world', -1, 14, 'environment', 1, 11)}, {'management': ('resou
rce', -1, 19, 'agencies', 1, 7)}, {'support': ('financial', -1, 8, 'system
s', 1, 13)}, {'agricultural': ('energy', -1, 6, 'land', 1, 10)}, {'much':
('times', -1, 6, 'larger', 1, 6)}]
```

In [92]:

```python
# define nodelist and edgelist
n =[(list(x.keys())[0], list(x.values())[0][0], list(x.values())[0][3]) for x in max_ad
jacents]
# flatten
nl = [item for sublist in n for item in sublist]
# remove duplicates
nodelist = []
for item in nl:
    if item not in nodelist:
        nodelist.append(item)
#print nodelist

e = [((list(x.keys())[0], list(x.values())[0][0], list(x.values())[0][1], list(x.values
())[0][2]),
      (list(x.keys())[0], list(x.values())[0][3], list(x.values())[0][4], list(x.values
())[0][5])) for x in max_adjacents]
# flatten
edgelist = [item for sublist in e for item in sublist]
#print edgelist
```

In [93]:

```
# draw network
import networkx as nx
import matplotlib.colors as colors
import matplotlib.cm as cmx

G = nx.DiGraph()

# nodes
G.add_nodes_from(nodelist)

# edges
#G.add_edges_from(edgelist)

reduce_factor = 200.
for item in edgelist:
    if item[2] == -1:
        # reverse direction in this case, because it is the antecedent word
        G.add_edge(item[1],item[0],color = 'r', weight = item[3]/reduce_factor)
    else:
        G.add_edge(item[0],item[1],color = 'g', weight = item[3]/reduce_factor)

edges = G.edges()
#print(edges)
colors = [G[u][v]['color'] for u,v in edges]
weights = [G[u][v]['weight'] for u,v in edges]

#layout
# k controls the distance between the nodes and varies between 0 and 1
# iterations is the number of times simulated annealing is run
# default k = 0.1 and iterations = 50
pos = nx.spring_layout(G, k = 0.3, iterations = 50)             # positions for all n
odes

plt.figure(figsize = (15,10))
nx.draw(G, pos, edges = edges, edge_color = colors, width = weights, arrows=True, with_
labels=True,
        font_size=14, node_size = 100, alpha = 1)

plt.axis('off')
plt.show() # display

#import pandas as pd
#M = pd.DataFrame(nx.adjacency_matrix(G, nodelist=nodelist))
#M.index = nodelist
#M.columns = nodelist
#M
```

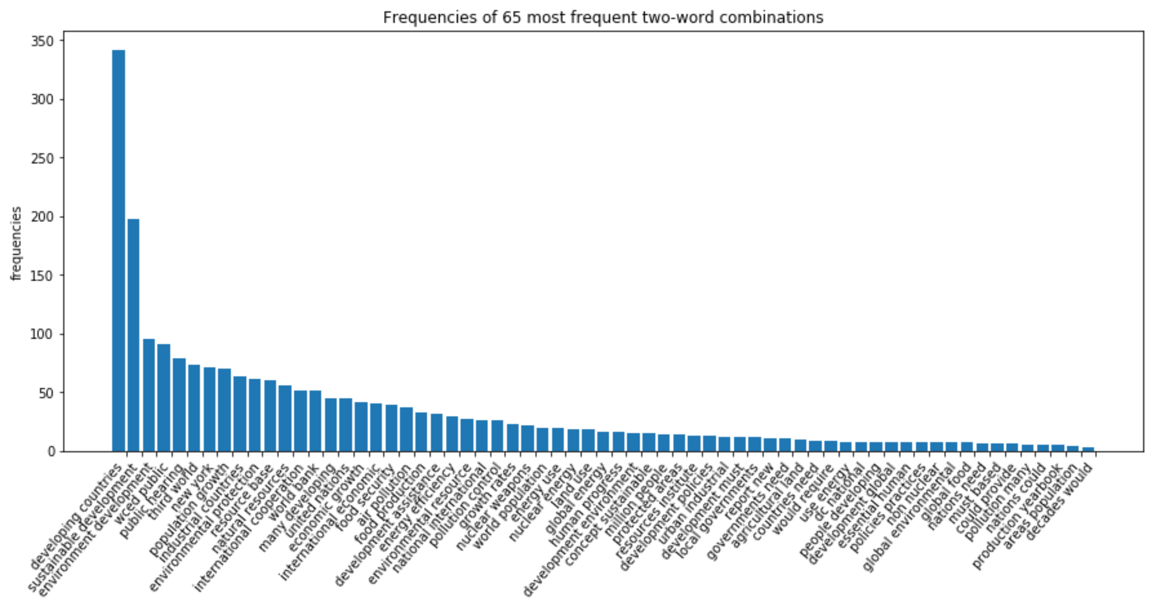


In [110]:

```
# export as graph-ml for Gephi
#write_graphml(G, path[, encoding, prettyprint])
nx.write_graphml(G, "C:\\temp\\Text-mining\\brundtlad.graphml", prettyprint = True)
```

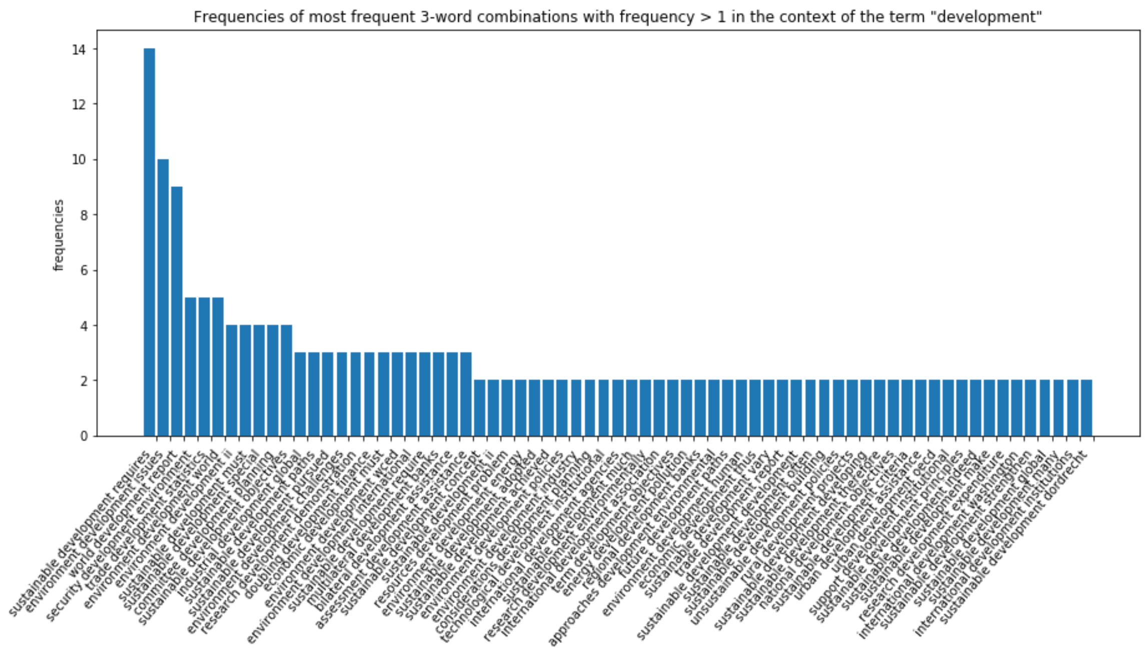
In [94]:

```python
import pandas as pd
connection_list = []

for item in edgelist[:70]:
    if item[2]== -1:
        if [item[1],item[0],item[3]] not in connection_list:
            connection_list.append([item[1],item[0],item[3]])
    else:
        if [item[0],item[1],item[3]] not in connection_list:
            connection_list.append([item[0],item[1],item[3]])
#print(connection_list)
# sort
sorted_cl = sorted(connection_list,key=lambda x: x[2], reverse = True)

words = [(x[0] + " " + x[1]) for x in sorted_cl]
frequences = [x[2] for x in sorted_cl]

# plot
fig = plt.figure(figsize=(15, 6))
numwords = len(sorted_cl)
xcor = range(numwords)
plt.bar(xcor, frequences)
# set ticks position 0.5 to the right, so that labels are centered
xticks_pos = np.arange(0.5, numwords + 0.5, 1)
# specify a rotation for the tick labels in degrees or with keywords (e.g.'vertical');
 ha = horizontal alignment of ticklabels
plt.xticks(xticks_pos, words, rotation='50', ha = 'right')
plt.title('Frequencies of %s most frequent two-word combinations' %numwords)
plt.ylabel('frequencies')
# Plot margins so that markers don't get clipped by the axes
plt.margins(0.05)
# Tweak spacing to prevent clipping of tick-labels; can also be done with figsize
#plt.subplots_adjust(bottom = 0.5)
plt.show()
```



In [110]:

```
# find frequencies of x most frequent n-word combinations in context of ward x

# word = the concordanced word
# von = the lower end of the context considered
# bis = the (excluded) upper end of the context considered
# k = the number of occurences considered

def context(word, von, bis, k):
    # finding the words that surround the concordanced word in distance dis
    c = nltk.ConcordanceIndex(texts.tokens, key = lambda s: s.lower())
    # +1 indicates the words around the concordanced word, from negcontext (incl) to po
scontext (excl)
    negcontext = von    # inclusive
    poscontext = bis    # exclusive
    d = [texts.tokens[offset + negcontext : offset + poscontext] for offset in c.offset
s(word)]
    #print(d)

    dd = []
    for x in d:
        i = 0
        xx = ""
        while i < len(x):
            xx += x[i] + " "
            i += 1
        dd.append(xx)

    # count number of occurrences of contextwords-distance-tuples in dd
    occurrences = [dd.count(item) for item in dd]
    # generate tuples of word-distance-tuples and the number of their occurrences in dd
    tuples = zip(dd, occurrences)

    # remove duplicates in tuples
    cleared = []
    for item in tuples:
        if item not in cleared:
            cleared.append(item)

    # sort tuples by occurrences
    sorted_c = sorted(cleared,key=lambda x: x[1], reverse = True)
    context_words = [x[0] for x in sorted_c if x[1] > k]
    frequences = [x[1] for x in sorted_c if x[1] > k]

    # plot
    fig = plt.figure(figsize=(15, 6))
    numwords = len(context_words)
    xcor = range(numwords)
    plt.bar(xcor, frequences)
    # set ticks position 0.5 to the right, so that labels are centered
    xticks_pos = np.arange(0.5, numwords+0.5, 1)
    # specify a rotation for the tick labels in degrees or with keywords (e.g.'vertica
l'); ha = horizontal alignment of ticklabels
    plt.xticks(xticks_pos, context_words, rotation='50', ha = 'right')
    plt.title('Frequencies of most frequent %s-word combinations with frequency > %s in
the context of the term "%s"'
              %(bis - von, k, word))
    plt.ylabel('frequencies')
    # Plot margins so that markers don't get clipped by the axes
    plt.margins(0.05)
    # Tweak spacing to prevent clipping of tick-labels; can also be done with figsize
    #plt.subplots_adjust(bottom = 0.5)
    plt.show()


for item in most_freq_words[:1]:
    # def context(word, von, bis, k):
    context(item, -1, 2, 1)
```



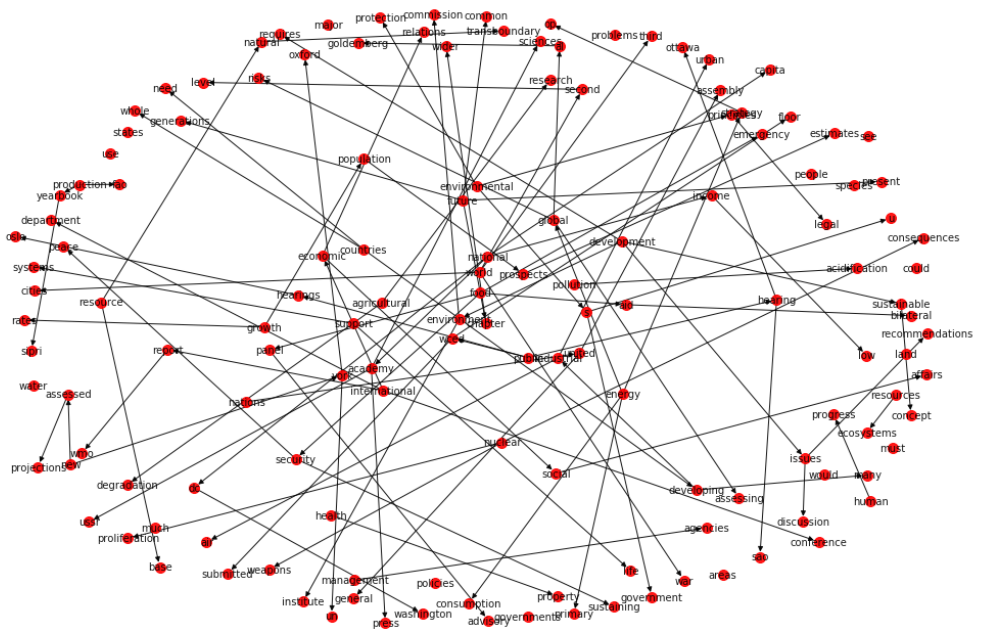
In [107]:

```
# draw network without weights, just connections among context-words
def context(word, von, bis, k):
    # finding the words that surround the concordanced word in distance dis
    c = nltk.ConcordanceIndex(texts.tokens, key = lambda s: s.lower())
    # +1 indicates the words around the concordanced word, from negcontext (incl) to po
scontext (excl)
    negcontext = von    # inclusive
    poscontext = bis    # exclusive
    d = [texts.tokens[offset + negcontext : offset + poscontext] for offset in c.offset
s(word)]
    #print d

    dd = []
    for x in d:
        i = 0
        xx = ""
        while i < len(x):
            xx += x[i] + " "
            i += 1
        dd.append(xx)

    #dd = [(x[0] + " " + x[1] + " " + x[2]) for x in d]
    #print dd

    # count number of occurrences of contextwords-distance-tuples in dd
    occurrences = [dd.count(item) for item in dd]
    # generate tuples of word-distance-tuples and the number of their occurrences in dd
    tuples = zip(dd, occurrences)

    #print tuples

    # remove duplicates in tuples
    cleared = []
    for item in tuples:
        if item not in cleared:
            cleared.append(item)

    #print cleared

    # sort tuples by occurrences
    sorted_c = sorted(cleared,key=lambda x: x[1], reverse = True)
    #print sorted_c

    context_words = [x[0] for x in sorted_c if x[1] > k]
    #print context_words
    frequences = [x[1] for x in sorted_c if x[1] > k]
    #print frequences
    return context_words

c =[]

nodelist = []
edgelist = []

def network_context(contexted_term, k, kk):
    global nodelist
    global edgelist
    nodelist.append(contexted_term)
    # consider triples with frequency more than k
    # context(word, von, bis, k)
    cont = context(contexted_term, -1, 2, k)
    # consider kk context triples
    for item in cont[:kk]:
        tokens = nltk.word_tokenize(item)
        for tok in tokens:
            if tok not in nodelist:
                nodelist.append(tok)
                edgelist.append((contexted_term, tok))

k = 3    # minimum frequency
kk = 2    # number of contexts considered

for item in most_freq_words:
    network_context(item, k, kk)
for item in nodelist[:100]:
    network_context(item, k, kk)
print(len(nodelist))
#print(edgelist)
```

```
254
```

In [109]:

```
G = nx.DiGraph()

# nodes
G.add_nodes_from(nodelist)

# edges
G.add_edges_from(edgelist)

#layout
# k controls the distance between the nodes and varies between 0 and 1
# iterations is the number of times simulated annealing is run
# default k = 0.1 and iterations = 50
pos = nx.spring_layout(G, k = 0.5, iterations = 50)           # positions for all n
odes

plt.figure(figsize=(15,10))
nx.draw(G, pos, edges = edges, arrows=True, with_labels=True,
        font_size=10, node_size = 100, alpha = 0.9)

plt.axis('off')
plt.show() # display
```

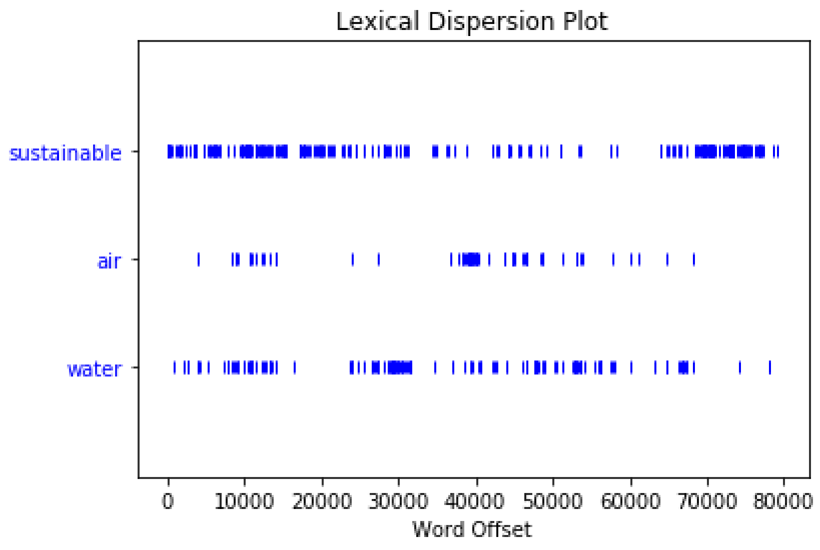


In [71]:

```
# export as graph-ml for Gephi
#write_graphml(G, path[, encoding, prettyprint])
nx.write_graphml(G, "C:\\temp\\Text-mining\\brundtland_2.graphml", prettyprint = True)
```

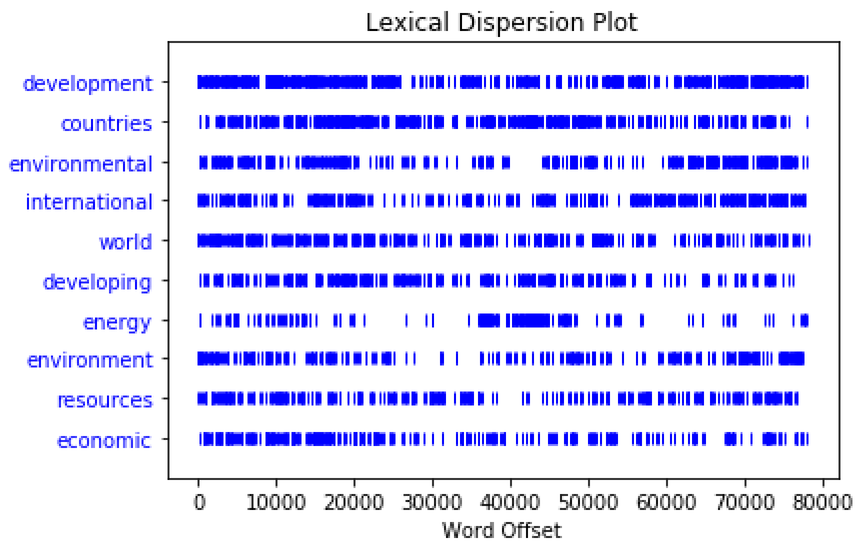
In [66]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

texts.dispersion_plot(["sustainable", "air", "water"])
```



In [111]:

```
mfw_list = []
for mfw in most_freq_words[:10]:
    mfw_list.append(mfw)

texts.dispersion_plot(mfw_list)
```



In [114]:

```
term = 'water'
texts.concordance(term)
```

```
Displaying 25 of 25 matches:
ple adequate food sound housing safe water access means choosing size fami
lies
 not read write numbers without safe water safe sound homes numbers short
woodf
stances human food chain underground water tables beyond reach cleansing g
rowin
g millions fish threatening drinking water federal republic germany nether
lands
eal diseases related unsafe drinking water malnutrition victims children m
ainsp
 ecological stress degradation soils water regimes atmosphere forests upon
econ
within limits mandates improving air water quality enhancing resources muc
h wor
nomic burdens inherited problems air water pollution depletion groundwater
prol
ged overuse soil chemicals pollution water resources foods chemicals degra
datio
ies needed adequate human life clean water sanitation schools transport re
sult
ing growing number lack access clean water sanitation hence prey diseases
arise
surface runoff soil erodes soil gone water retained land longer produce en
ough
turies human existence interventions water cycles increased greatly massiv
e dam
ge proportion river flow europe asia water use reached annual run off figu
re ex
 could ever change colour rivers big water plentiful amount human activity
coul
ons affected spreading sands changes water regimes increased risks soil er
osion
velopment environmental progress air water pollution control example incre
asing
 agricultural policies lie root land water forest degradation energy polic
ies a
osystems respect national boundaries water pollution moves shared rivers l
akes
ernational hydrological decade world water balance water resources earth p
aris
rological decade world water balance water resources earth paris unesco wm
o rep
ent limits hold use energy materials water land many manifest form rising
costs
n planned effects soil erosion rates water regimes genetic losses taken ac
count
pecies so called free goods like air water resources raw materials energy
produ
requires adverse impacts quality air water natural elements minimized sust
ain e
```

In [ ]:

```

```