Case Assignment #2
OPIM 311, Spring 2012
"Scripting for Business Analytics"

Steven O. Kimbrough

April 2, 2012

This case 2 assignment is due by 5 p.m. on Saturday 5 May 2012.

This due date is different from what's on the syllabus (it's later) and what was stated (mistakenly, a typo) in the case 2 assignment document (infeasible for handing in grades).

Contents

1 The Concept Vector Matching Process

An influential paper by James March begins as follows.

> A central concern of studies of adaptive processes is the relation between the exploration of new possibilities and the exploitation of old certainties (Schumpeter 1934; Holland 1975; Kuran 1988). ***Exploration includes things captured by terms such as search, variation, risk taking, experimentation, play, flexibility, discovery, innovation. Exploitation includes such things as refinement, choice, production, efficiency, selection, implementation, execution.*** Adaptive systems that engage in exploration to the exclusion of exploitation are likely to find that they suffer the costs of experimentation without gaining many of its benefits. They exhibit too many undeveloped new ideas and too little distinctive competence. Conversely, systems that engage in exploitation to the exclusion of exploration are likely to find themselves trapped in suboptimal stable equilibria. As a result, maintaining an appropriate balance between exploration and exploitation is a primary factor in system survival and prosperity.
>
> This paper considers some aspects of such problems in the context of organizations. Both exploration and exploitation are essential for organizations, but they compete for scarce resources. As a result, organizations make explicit and implicit choices between the two. The explicit choices are found in calculated decisions about alternative investments and competitive strategies. The implicit choices are buried in many features of organizational forms and customs, for example, in organizational procedures for accumulating and reducing slack, in search rules and practices, in the ways in which targets are set and changed, and in incentive systems. Understanding the choices and improving the balance between exploration and exploitation are complicated by the fact that returns from the two options vary not only with respect to their expected values, but also with respect to their variability, their timing, and their distribution within and beyond the organization. (March, 1991, Emphasis added)

In the remainder of the paper, March develops and explores two rather abstract, stylized models of organizational learning. These models, or rather March's discussion of them, shed light on the exploration–exploitation tradeoffs that organizations must make. The paper promotes what has subsequently been called the *ambidexterity hypothesis,* which holds that successful organizations engage in both exploration and exploitation, in spite of the added burden of managing the two rather different kinds of operation.

How might we recognize exploration and exploitation activities with objective measures applied to real organizations? We cannot expect much help here from stylized models, however useful they are for other purposes. A fairly obvious and straightforward, not to say naïve, idea would be to develop lists of words having to do, in the present case, with exploitation and with exploration. One might (1) simply start with March's two lists, emphasized in the above quoted passage from his paper. Then, (2) one might collect documents pertaining to the firms in question.

Finally, (3) one could score the documents with respect to the several lists of words developed in step (1).

This assignment is about supporting this process with Python code, so let us be a bit more general about the overall process.

1. Develop concept vectors.
2. Collect and process relevant documents.
3. Match the concept vectors to the documents and produce relevance scores.

And we need a name. We'll call it the *concept vector matching* process. A word or two now on each of the steps.

1.1 Developing concept vectors

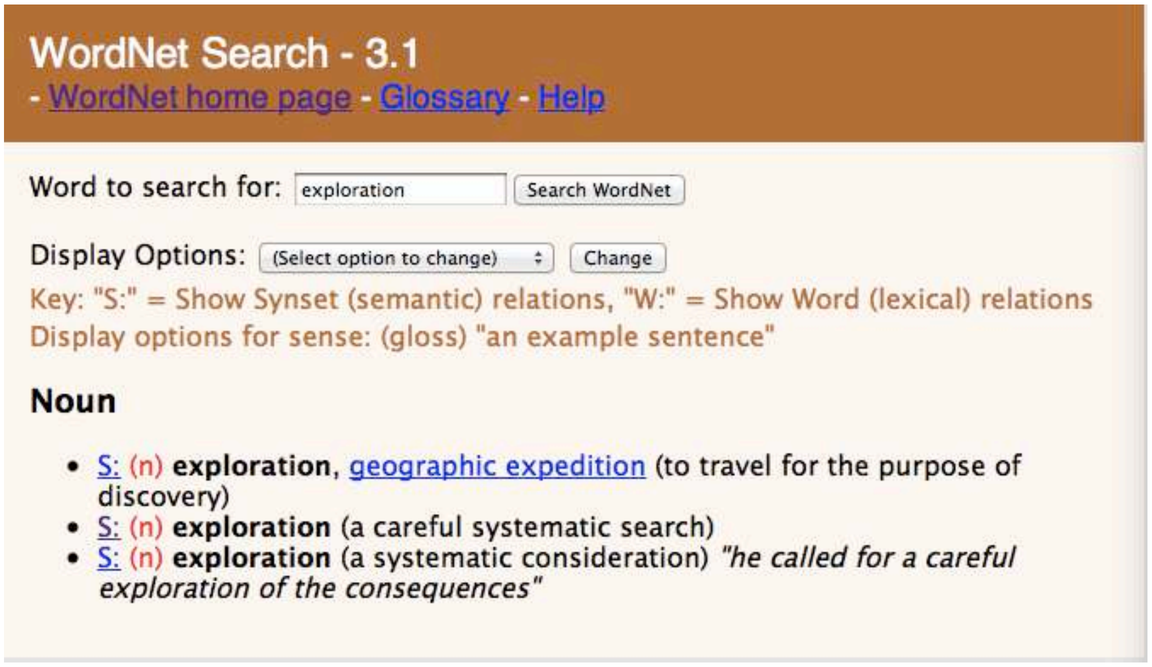
Basically, a *concept vector* (aka: topic vector, vocabulary (Loewenstein et al., 2012), n-gram list) is a list of words that we think all have some indicative power for the concept on which we wish to score documents. The words might be weighted so that seemingly better indicators would count more. Further, we do not need to be limited to single words. We call a one-word phrase a unigram, a two word phrase a bi-gram, and in general an n word phrase an n-gram. So the elements in our concept vector may be n-grams generally. We can also specify more complicated arrangements. For example, that n-gram #1 should precede n-gram #2 and be within k words of n-gram #2. These are all complications to be dealt with later. The short of it is that a concept vector is a, possibly weighted, *list of lexical patterns*; in the simplest case it is just a list words. Of course, we want the lexical patterns to indicate effectively the concept we have in mind.

Directly to the point, how do we develop a list of n-grams to constitute a concept vector? This is an art, although it can be augmented with science. One begins with list developed intuitively or from a credible source (such as the person or organization for whom the project is undertaken) and then one modifies the list, adding to and deleting from it. I can recommend two specific tools for doing this.

The first is WordNet: `http://wordnet.princeton.edu/`. Figure 1 shows the report you get if you use WordNet online to search on the term "exploration." It shows three distinct synsets (S, synonym set, groupings of similar meanings) for "exploration." You can then explore and find synonyms, hypernyms (more general terms), and hyponyms (more specific terms). Which are relevant? This requires judgment on the analyst's part. Perhaps, for example, synonyms for "exploration" in the sense of geographic expedition are relevant because the documents often use metaphoric language.

Note that "probe" is a direct hypernym of "exploration" in the sense of "a careful systematic search," yet it is not in March's list of exploration-related terms. Should it be?

Besides WordNet, other kinds of lexicons are often available. Particularly useful are industry-specific or scientific lexicons and classification systems. They will very often be the best source


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

Word to search for: exploration [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) **exploration**, geographic expedition (to travel for the purpose of discovery)
- S: (n) **exploration** (a careful systematic search)
- S: (n) **exploration** (a systematic consideration) *"he called for a careful exploration of the consequences"*

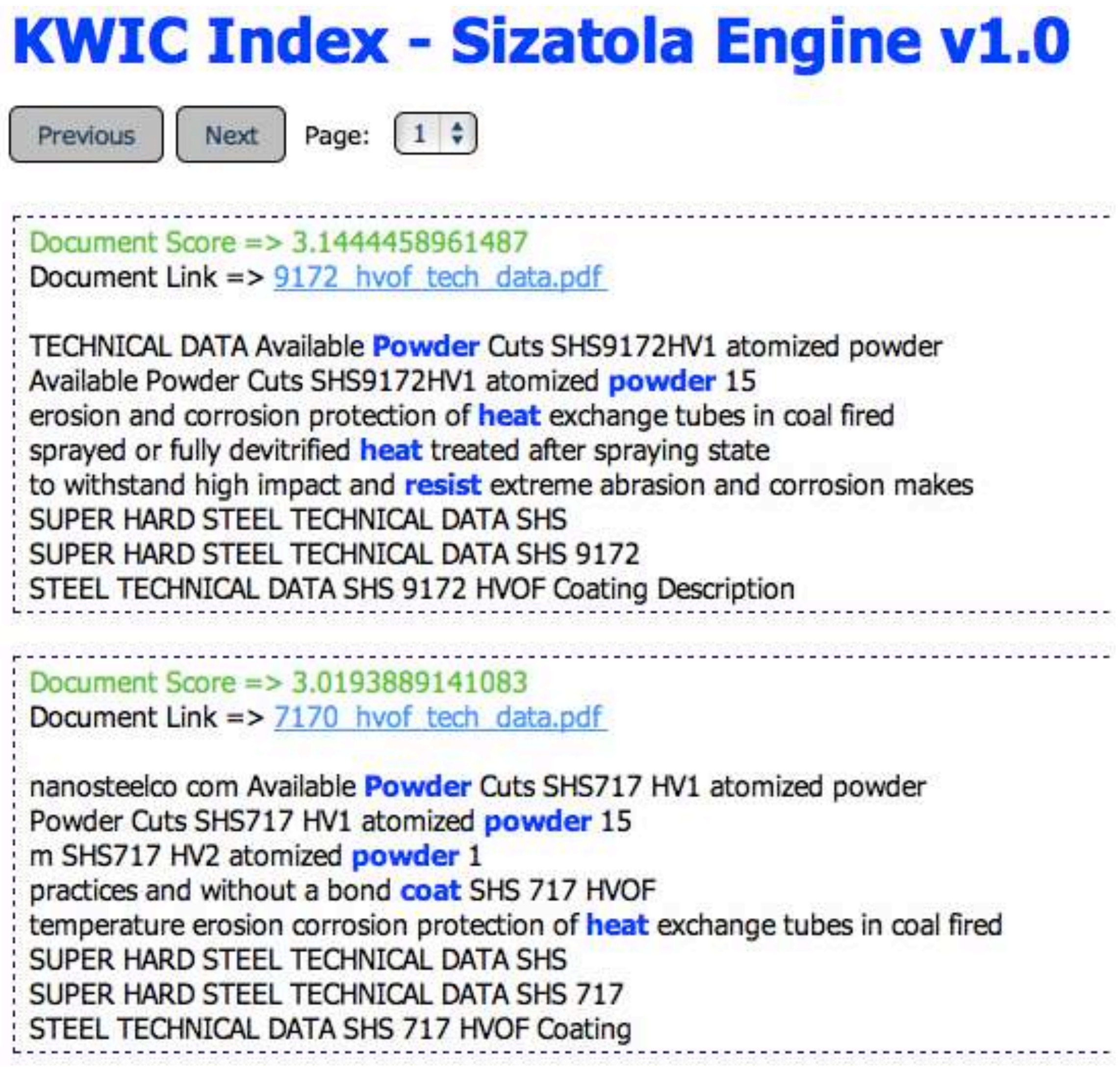
Figure 1: WordNet report on "exploration"

of terminology for technical terms, those outside of common language. WordNet's aspirations are limited to ordinary English.

A second tool is also very helpful, although it requires specialized software and this is a *concordance* or KWIC (keyword in context) index. Figure 2 presents an example. The concept vector terms are highlighted in blue (if you have color) and for each occurrence in the document we see a few words occurring before and after the term in question. By reading these lines, an analyst can come to an informed judgment regarding whether the term in question is actually being used very often in a way that indicates the concept we wish to match. In the example related to Figure 2, the concept in play has to do with tungsten coatings. It appears that each of the highlighted words is indeed being used in a relevant context. Concordance or KWIC index software is widely available. See the *PyTAbook* for how to do it in Python.

1.2 Collect and process relevant documents

If we are going to match our concept vectors to documents and score them, then we had better obtain the documents. I will do that for you (as described below). The general point is that useful documents can come from many sources, including: SEC filings, newspaper stories, annual reports, Web pages, and transcribed telephone calls (Larcker and Zakolyukina, 2012).


KWIC Index - Sizatola Engine v1.0

Document Score => 3.1444458961487
Document Link => 9172_hvof_tech_data.pdf

TECHNICAL DATA Available **Powder** Cuts SHS9172HV1 atomized powder
Available Powder Cuts SHS9172HV1 atomized **powder** 15
erosion and corrosion protection of **heat** exchange tubes in coal fired
sprayed or fully devitrified **heat** treated after spraying state
to withstand high impact and **resist** extreme abrasion and corrosion makes
SUPER HARD STEEL TECHNICAL DATA SHS
SUPER HARD STEEL TECHNICAL DATA SHS 9172
STEEL TECHNICAL DATA SHS 9172 HVOF Coating Description

Document Score => 3.0193889141083
Document Link => 7170_hvof_tech_data.pdf

nanosteelco com Available **Powder** Cuts SHS717 HV1 atomized powder
Powder Cuts SHS717 HV1 atomized **powder** 15
m SHS717 HV2 atomized **powder** 1
practices and without a bond **coat** SHS 717 HVOF
temperature erosion corrosion protection of **heat** exchange tubes in coal fired
SUPER HARD STEEL TECHNICAL DATA SHS
SUPER HARD STEEL TECHNICAL DATA SHS 717
STEEL TECHNICAL DATA SHS 717 HVOF Coating

Figure 2: Example KWIC index on multiple words

1.3 Match the concept vectors to the documents

Recall that a concept vector is a list of lexical patterns and may be weighted. We match a concept vector to a document or body of text by counting the number of times each lexical pattern in the concept vector occurs in the text. We may then weight the counts and use the resulting numbers to form an overall score. This may be done in several ways. We may simply add them up for a single score. We may divide the individual scores by the length of the document and then add everything up. And so on. In the tasks that follow, we will explore some options.

2 Does It Work?

Broadly speaking, yes. Uotila et al. (2009) used concept vectors very close to March's and matched them to newspaper stories on Factiva covering the years 1989–2004 for the 279 manufacturing firms in the 1989 Standard & Poor's 500 index. Here are the two concept vectors they actually used (Uotila et al., 2009):

> Appendix 1. Words and Word Roots in Content Analysis
>
> The wildcard '*' can represent any characters. [*sic: any sequence of alphabetic characters –sok*]
> Exploratory action: explor*, search*, variation*, risk*, experiment*, play*, flexib*, discover*, innovat*
> Exploitative action: exploit*, refine*, choice*, production*, efficien*, select*, implement*, execut*

They got results that are interesting and significant, pertaining to investment in exploration and financial performance.

Larcker and Zakolyukina (2012), mentioned above, were able to detect lying or prevarication by CEOs in conference calls with securities analysts using these methods.

Tetlock et al. (2008) uses "happy" and "sad" words from a standard lexicon as concept vectors and demonstrated their predictive value for firm performance when matched to newspaper stories about the firms.

Much earlier, the psychologist Gottschalk used these methods for *The Measurement of Psychological States Through the Content Analysis of Verbal Behavior* (Gottschalk and Gleser, 1969; Gottschalk et al., 1969; Gottschalk, 1995). From carefully developed word lists (treated as concept vectors in our sense), Gottschalk and his co-workers achieved considerable success in automated diagnosis of mental illnesses from transcribed interviews with patients. (Wouldn't you like to know what words are in the lists? Well, it's public information.)

One could go on, but what's here should suffice to make the point for the utility of the concept vector matching process.

3 Tasks

3.1 Get a Concept Vector from a File

Write a function and associated code that conforms to the following template and documentation.

```
def getConceptVectorLabeled(daDir,daFile):
    '''
Assumes that daFile is in standard concept vector form:
<weight>$$$<pattern>$$$<pattern label>, where <pattern>
is an re string and <pattern label> is for display only.
Example: MarchExplorLabeled.txt:
1$$$\bexplor.*?\b$$$explor*
1$$$\bsearch.*?\b$$$search*
1$$$\bvariation.*?\b$$$variation*
1$$$\brisk.*?\b$$$risk*
1$$$\bexperiment.*?\b$$$experiment*
1$$$\bplay.*?\b$$$play*
1$$$\bflexib.*?\b$$$flexib*
1$$$\bdiscover.*?\b$$$discover*
1$$$\binnovat.*?\b$$$innovat*
Here the weights are all equal to 1. The $$$ is a separator.
Then there is a regular expression.Note that here we
are insisting on word boundaries, with \b.
The function returns a list in which the first item
is the concept name, taken as the file name minus the
".txt" extension. More precisely, it is the string in
the file name to the left of the first period.
The second item in the list is another list. This
is a list of 2-tuples


[No, it's 3-tuples The third item is the
label on the regular expression which is the second item.]


 in which the first item is the
<weight> value from a line and the second is the
<pattern> from the line, as a string. For example,
this is returned for MarchExplore.txt:
Run the script
def q1():
    conceptVector = \
                 mytexan.getConceptVectorLabeled(\
                     pathToConceptVectors,MarchExplore)
    print(conceptVector)
and you should get

['MarchExploreLabeled', [
('1', '\\bexplor.*?\\b', 'explor*'),
('1', '\\bsearch.*?\\b', 'search*'),
('1', '\\bvariation.*?\\b', 'variation*'),
('1', '\\brisk.*?\\b', 'risk*'),
('1', '\\bexperiment.*?\\b', 'experiment*'),
('1', '\\bplay.*?\\b', 'play*'),
('1', '\\bflexib.*?\\b', 'flexib*'),
('1', '\\bdiscover.*?\\b', 'discover*'),
('1', '\\binnovat.*?\\b', 'innovat*')]]

Note that the backslashes have been escaped (this is
done automatically by Python when you read the file),
so you do not need to worry about using a raw string.
    '''
[Your code here.]
```

When you run it, say with

```
def q1():
    '''
A little script to exercise the code for question 1.
    '''
    conceptVector = \
                 mytexan.getConceptVectorLabeled(\
                     pathToConceptVectors,MarchExplore)
    print(conceptVector)

q1()
```

(where `dirConceptVectors` is the path to the directory holding the concept vectors files) it should behave as indicated above in the function comments (edited for display). See myexercise-texan.py for valid assignments of `dirConceptVectors` and `MarchExplore`.

Answer:

```
def getConceptVectorLabeled(daDir,daFile):
    '''
```

```
Assumes that daFile is in standard concept vector form:
<weight>$$$<pattern>$$$<pattern label>, where <pattern>
is an re string and <pattern label> is for display only.
Example: MarchExplorLabeled.txt:
1$$$\bexplor.*?\b$$$explor*
1$$$\bsearch.*?\b$$$search*
1$$$\bvariation.*?\b$$$variation*
1$$$\brisk.*?\b$$$risk*
1$$$\bexperiment.*?\b$$$experiment*
1$$$\bplay.*?\b$$$play*
1$$$\bflexib.*?\b$$$flexib*
1$$$\bdiscover.*?\b$$$discover*
1$$$\binnovat.*?\b$$$innovat*
Here the weights are all equal to 1. The $$$ is a separator.
Then there is a regular expression.Note that here we
are insisting on word boundaries, with \b.
The function returns a list in which the first item
is the concept name, taken as the file name minus the
".txt" extension. More precisely, it is the string in
the file name to the left of the first period.
The second item in the list is another list. This
is a list of 2-tuples in which the first item is the
<weight> value from a line and the second is the
<pattern> from the line, as a string. For example,
this is returned for MarchExplore.txt:
Run the script
def q1():
    conceptVector = \
                 mytexan.getConceptVectorLabeled(\
                     pathToConceptVectors,MarchExplore)
    print(conceptVector)
and you should get

['MarchExploreLabeled', [
('1', '\\bexplor.*?\\b', 'explor*'),
('1', '\\bsearch.*?\\b', 'search*'),
('1', '\\bvariation.*?\\b', 'variation*'),
('1', '\\brisk.*?\\b', 'risk*'),
('1', '\\bexperiment.*?\\b', 'experiment*'),
('1', '\\bplay.*?\\b', 'play*'),
('1', '\\bflexib.*?\\b', 'flexib*'),
('1', '\\bdiscover.*?\\b', 'discover*'),
('1', '\\binnovat.*?\\b', 'innovat*')]]

Note that the backslashes have been escaped (this is
done automatically by Python when you read the file),
so you do not need to worry about using a raw string.
    '''
    f = open(daDir + os.sep + daFile,'r')
    flines = f.readlines()
    tupleList = []
    for line in flines:
        daSplit = line.strip('\n').split('$$$')
        #print(daSplit[2])
        if len(daSplit) > 2:
            tupleList.append((daSplit[0], daSplit[1], daSplit[2]))

    conceptName = daFile.split('.')[0]
    return [conceptName, tupleList]
```

3.2 Process a Single Pattern on a Collection

Write a function and associated code that conforms to the following template and documentation.

```
def dirFilesCountWord(dirTxts,daWord):
    '''
Given a directory, dirTxts, holding a number text files,
and a search term, daWord, returns a list of 2-tuples in
which the first item is the file name and the second item
is the number of times the search term appears in the file.
This function DOES use the re module to accomplish this task.
It works fine, however, if daWord is just a simple string.
    '''
[Your code here.]
```

When you run it, say with

```
def q2():
    '''
A little script to exercise the code for question 2.
    '''
    theDirTexts = pathToTextCollections + os.sep + DowIndTxts
    listDirFilesCounts = \
        mytexan.dirFilesCountWord(theDirTexts,r'\bexpan.*?\b')
    print(theDirTexts)
    print(listDirFilesCounts)
q2()
```

where `DowIndTxts` is defined as

```
DowIndTxts = 'DowJonesIndustrials2009'
```

then it should behave as follows (edited for display):

```
../collections/text/DowJonesIndustrials2009
[('3M-2010.pdf.txt', 8), ('Alcoa-2010.pdf.txt', 46),
('ATT-2010.pdf.txt', 25), ('AXP-2010.pdf.txt', 13),
('Boeing-2010.pdf.txt', 8), ('Caterpillar-2010.pdf.txt', 16),
('Chevron-2010.pdf.txt', 10), ('Cisco-2011.pdf.txt', 16),
('Coca-Cola-2010.pdf.txt', 5), ('Dupont-2010.pdf.txt', 2),
('Exxon-2010.pdf.txt', 21), ('GE-2010.pdf.txt', 20),
('HomeDepot-2011.pdf.txt', 10), ('HP-2011.pdf.txt', 3),
('IBM-2010.pdf.txt', 37), ('Intel-2010.pdf.txt', 1),
('JPMC-2010.pdf.txt', 45), ('Kraftfoods-2010.pdf.txt', 8),
('McDonalds-2010.pdf.txt', 11), ('Merck-2010.pdf.txt', 6),
('Microsoft-2011.pdf.txt', 5), ('Pfizer-2010.pdf.txt', 7),
('PG-2011.pdf.txt', 42), ('Travelers-2010.pdf.txt', 41),
('United-Technologies-2010.pdf.txt', 17),
('Verizon-2010.pdf.txt', 25), ('Wal-mart-2010.pdf.txt', 35),
('Walt-Disney-2010.pdf.txt', 22)]
```

Answer:

```
def dirFilesCountWord(dirTxts,daWord):
    '''
Given a directory, dirTxts, holding a number text files,
and a search term, daWord, returns a list of 2-tuples in
which the first item is the file name and the second item
is the number of times the search term appears in the file.
This function DOES use the re module to accomplish this task.
It works fine, however, if daWord is just a simple string.
    '''
    daFileNames = os.listdir(dirTxts)
    toReturn = []
    compiledPattern = re.compile(daWord,re.I)
    for daFileName in daFileNames:
        f = open(dirTxts + os.sep + daFileName,'r').read()
        daCount = len(compiledPattern.findall(f))
        toReturn.append((daFileName, daCount))

    return toReturn
```

3.3 Sort a List of Tuples

Write a function and associated code that conforms to the following template and documentation.

```
def sortListOfTuples(daList,daItem,descend=True):
    '''
Given a list of tuples, daList, sorts the tuples
on item daItem and returns the sorted list of
tuples. By default the list is in descending order,
but if descend is False, then in ascending order.
    '''
[Your code here.]
```

When you run it, it should behave as follows. This script

```
def q3():
    '''
A little script to exercise the code for question 3.
    '''
    theDirTexts = pathToTextCollections + os.sep + DowIndTxts
    listDirFilesCounts = \
        mytexan.dirFilesCountWord(theDirTexts,r'\bexpan.*?\b')
    sortedList = mytexan.sortListOfTuples(\
        listDirFilesCounts,1)
    print(sortedList)
q3()
```

produces

```
[('Alcoa-2010.pdf.txt', 46), ('JPMC-2010.pdf.txt', 45),
('PG-2011.pdf.txt', 42), ('Travelers-2010.pdf.txt', 41),
('IBM-2010.pdf.txt', 37),
('Wal-mart-2010.pdf.txt', 35), ('ATT-2010.pdf.txt', 25),
('Verizon-2010.pdf.txt', 25), ('Walt-Disney-2010.pdf.txt', 22),
('Exxon-2010.pdf.txt', 21), ('GE-2010.pdf.txt', 20),
('United-Technologies-2010.pdf.txt', 17),
('Caterpillar-2010.pdf.txt', 16), ('Cisco-2011.pdf.txt', 16),
('AXP-2010.pdf.txt', 13), ('McDonalds-2010.pdf.txt', 11),
('Chevron-2010.pdf.txt', 10), ('HomeDepot-2011.pdf.txt', 10),
('3M-2010.pdf.txt', 8), ('Boeing-2010.pdf.txt', 8),
('Kraftfoods-2010.pdf.txt', 8), ('Pfizer-2010.pdf.txt', 7),
('Merck-2010.pdf.txt', 6), ('Coca-Cola-2010.pdf.txt', 5),
('Microsoft-2011.pdf.txt', 5), ('HP-2011.pdf.txt', 3),
('Dupont-2010.pdf.txt', 2), ('Intel-2010.pdf.txt', 1)]
```

(edited for display).

Answer:

```
def sortListOfTuples(daList,daItem,descend=True):
    '''
Given a list of tuples, daList, sorts the tuples
on item daItem and returns the sorted list of
tuples. By default the list is in descending order,
but if descend is False, then in ascending order.
    '''
    return sorted(daList, cmp=lambda x,y: \
                  cmp(x[daItem], y[daItem]), \
                      reverse=descend)
```

3.4 Write a Simple Report to a CSV File

Write a function and associated code that conforms to the following template and documentation.

```
def writeCSVFileCount2Tuples(da2Tuples,daFile,conceptName):
    '''
Given a list of 2-tuples, assumed to have the form
(<file name>,<concept score>), writes to the file daFile
the conceptName in the first line as
"Concept: " + conceptName.
The second line has "Source File,Score".
Following that writes each
tuple to a line, with a comma as the separater. The
tuples are written in the order given in da2Tuples.The
function returns a message string when it has completed:
"Done writing file " +  daFile
    '''
[Your code here.]
```

When you run it, it should behave as follows. Running the following script (after importing texan.py)

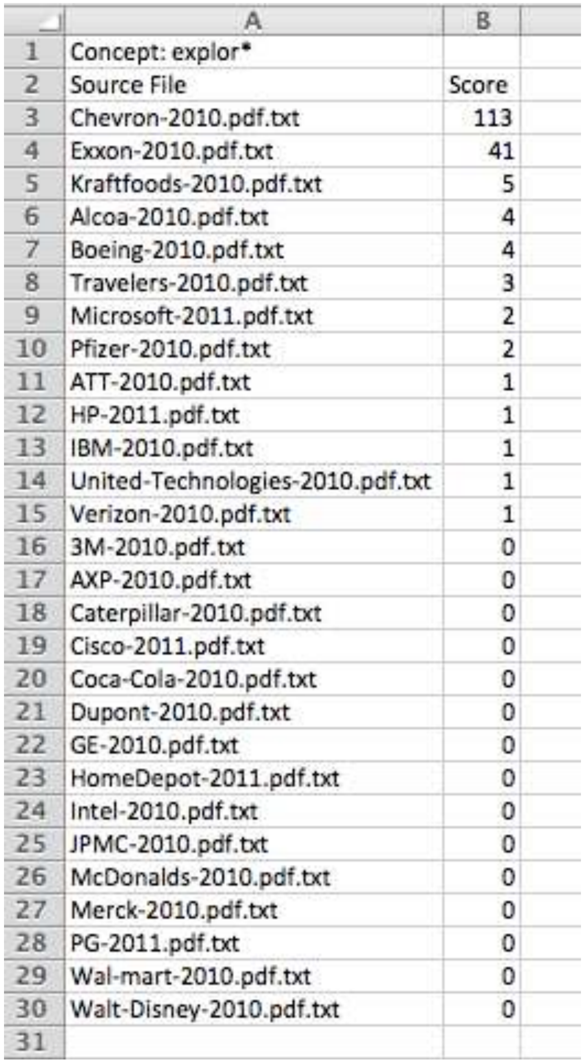
```
def q4():
    '''
A little script to exercise the code for question 4.
    '''
    conceptVector = \
        mytexan.getConceptVectorLabeled(pathToConceptVectors,MarchExplore)
    theDirTexts = pathToTextCollections + os.sep + DowIndTxts
    listDirFilesCounts = \
        mytexan.dirFilesCountWord(theDirTexts,conceptVector[1][0][1])
    print(mytexan.sortListOfTuples(listDirFilesCounts,1))
    print(mytexan.sortListOfTuples(listDirFilesCounts,1,False))
    print('\n\n')
    daSortedTuples = mytexan.sortListOfTuples(listDirFilesCounts,1)
    print(mytexan.writeCSVFileCount2Tuples(\
        daSortedTuples,\
        '..' + os.sep + 'outputs' + os.sep + 'tuplesScores.txt',\
        conceptVector[1][0][2]))

q4()
```

produces the message `Done writing file ../outputs/tuplesScores.txt.` when it completes. (It also produces output that illustrates ascending and descending sorting on `listDirFilesCounts`.) If you then launch Excel and import the file tuplesScores.txt, what you see will be (something very like) Figure 3. **Note well: The output file tuplesScores.txt is written to reside in the `outputs` folder. This is required.**

Answer:

```
def writeCSVFileCount2Tuples(da2Tuples,daFile,conceptName):
    '''
Given a list of 2-tuples, assumed to have the form
(<Given a list of 2-tuples, assumed to have the form
(<file name>,<concept score>), writes to the file daFile
the conceptName in the first line as
"Concept: " + conceptName.
The second line has "Source File,Score:".
Following that writes each
tuple to a line, with a comma as the separater. The
tuples are written in the order given in da2Tuples.The
function returns a message string when it has completed:
"Done writing file " +  daFile
    '''
```



	A	B
1	Concept: explor*	
2	Source File	Score
3	Chevron-2010.pdf.txt	113
4	Exxon-2010.pdf.txt	41
5	Kraftfoods-2010.pdf.txt	5
6	Alcoa-2010.pdf.txt	4
7	Boeing-2010.pdf.txt	4
8	Travelers-2010.pdf.txt	3
9	Microsoft-2011.pdf.txt	2
10	Pfizer-2010.pdf.txt	2
11	ATT-2010.pdf.txt	1
12	HP-2011.pdf.txt	1
13	IBM-2010.pdf.txt	1
14	United-Technologies-2010.pdf.txt	1
15	Verizon-2010.pdf.txt	1
16	3M-2010.pdf.txt	0
17	AXP-2010.pdf.txt	0
18	Caterpillar-2010.pdf.txt	0
19	Cisco-2011.pdf.txt	0
20	Coca-Cola-2010.pdf.txt	0
21	Dupont-2010.pdf.txt	0
22	GE-2010.pdf.txt	0
23	HomeDepot-2011.pdf.txt	0
24	Intel-2010.pdf.txt	0
25	JPMC-2010.pdf.txt	0
26	McDonalds-2010.pdf.txt	0
27	Merck-2010.pdf.txt	0
28	PG-2011.pdf.txt	0
29	Wal-mart-2010.pdf.txt	0
30	Walt-Disney-2010.pdf.txt	0
31		

Figure 3: Imported file produced by texan.sortListOfTuples()

```
    f = open(daFile,'w')
    f.write('Concept: ' + conceptName + '\n')
    f.write('Source File,Score\n')
    #daSortedTuples = sortListOfTuples(da2Tuples,1)
    for aTuple in da2Tuples:
        f.write(aTuple[0] + ',' + str(aTuple[1]) + '\n')

    f.close()
    return "Done writing file " + daFile + "."
```

3.5 Count the Hits and Show the Words

Write a function and associated code that conforms to the following template and documentation.

```
def dirFilesCountWordHits(dirTxts,daWord):
    '''
Similar to but adds to dirFilesCountWord. Instead of
returning a list of 2-tuples, it returns a list of 3-tuples.
Given a directory, dirTxts, holding a number text files,
and a search term, daWord, returns a list of 3-tuples in
which the first item is the file name, the second item
is the number of times the search term appears in the file, and the
third item is a list of the terms in the document matching
the input patter, daWord. This list is in alphabetical order.
This function DOES use the re module to accomplish this task.
It works fine, however, if daWord is just a simple string.
The function processes its input file after converting it to
lower case, so in the hit list we do not have duplications due
to capitalization.
    '''
[Your code here.]
```

When you run it, it should behave as follows. Running the following script (after importing texan.py)

```
def q5():
    '''
A little script to exercise the code for question 5.
    '''
    theDirTexts = pathToTextCollections + os.sep + DowIndTxts
    countsAndHits = \
        mytexan.dirFilesCountWordHits(theDirTexts,r'\bexpan.*?\b')
    print(countsAndHits)

q5()
```

produces the output (in part):

```
[('3M-2010.pdf.txt', 8, ['expanded', 'expanding', 'expansions']),
('Alcoa-2010.pdf.txt', 46, ['expand', 'expanded', 'expanding',
'expands', 'expansion', 'expansions']), ('ATT-2010.pdf.txt', 25,
['expand', 'expanded', 'expanding', 'expands', 'expansion']),
('AXP-2010.pdf.txt', 13, ['expand', 'expanded', 'expands',
'expansion']), ('Boeing-2010.pdf.txt', 8, ['expand',
'expandboeing', 'expanded', 'expanding']),
('Caterpillar-2010.pdf.txt', 16, ['expand', 'expanded',
'expanding', 'expansion']), ('Chevron-2010.pdf.txt', 10,
['expanded', 'expands', 'expansion']),
('Cisco-2011.pdf.txt', 16, ['expand', 'expanded', 'expanding', 'expansion']),
('Coca-Cola-2010.pdf.txt', 5, ['expand', 'expanded', 'expansion']),
('Dupont-2010.pdf.txt', 2, ['expanded', 'expanding']),
('Exxon-2010.pdf.txt', 21,
['expand', 'expanded', 'expanding', 'expands', 'expansion']),
```

(Notice the typo discovered in the Boeing report. Or is it a slogan?)

Answer:

```
def dirFilesCountWordHits(dirTxts,daWord):
    '''
Similar to but adds to dirFilesCountWord. Instead of
returning a list of 2-tuples, it returns a list of 3-tuples.
Given a directory, dirTxts, holding a number text files,
and a search term, daWord, returns a list of 3-tuples in
which the first item is the file name, the second item
is the number of times the search term appears in the file, and the
third item is a list of the terms in the document matching
the input patter, daWord. This list is in alphabetical order.
This function DOES use the re module to accomplish this task.
It works fine, however, if daWord is just a simple string.
The function processes its input file after converting it to
lower case, so in the hit list we do not have duplications due
to capitalization.
    '''
    daFileNames = os.listdir(dirTxts)
    toReturn = []
    compiledPattern = re.compile(daWord,re.I)
    for daFileName in daFileNames:
        f = open(dirTxts + os.sep + daFileName,'r')
        ftext = f.read().lower()
        f.close()
        #print(daFileName + " " + str(len(ftext)) + '\n')
        daFindings = compiledPattern.findall(ftext)
        #print(daFindings)
        daCount = len(daFindings)
        daHits = sorted(set(daFindings))
        toReturn.append((daFileName, daCount,list(daHits)))

    return toReturn
```

3.6 Write a CSV File to Report the Hit Counts and the Words

Write a function and associated code that conforms to the following template and documentation.

```
def writeCSVFileCountHits3Tuples(da3Tuples,daFile,conceptName):
    '''
Similar to writeCSVFileCount2Tuples, but adds hits list.
Given a list of 3-tuples, assumed to have the form
(<file name>,<concept score>, <hits list>), writes to the file
daFilethe conceptName in the first line.
The second line has "Source File,Score,Hits:".
Following that writes each 3-tuple to a line, with a comma
as the separater. The third item, a litst, is "flattened" and
each item in the list is written out with comma separations.
The tuples are written in the order given in da2Tuples.The
function returns a message string when it has completed:
"Done writing file " +  daFile
    '''
    f = open(daFile,'w')
    f.write('Concept: ' + conceptName + '\n')
    f.write('Source File,Score,Hits:\n')
    #daSortedTuples = sortListOfTuples(da2Tuples,1)
    for aTuple in da3Tuples:
        f.write(aTuple[0] + ',' + str(aTuple[1]))
        for aWord in aTuple[2]:
            f.write(','+aWord)
        f.write('\n')

    f.close()
    return "Done writing file " + daFile + "."
```

When you run it, it should behave as follows. Running the following script (after importing texan.py)

```
def q6():
    '''
A little script to exercise the code for question 6.
    '''
    theDirTexts = pathToTextCollections + os.sep + DowIndTxts
    countsAndHits = \
        mytexan.dirFilesCountWordHits(theDirTexts,r'\bexpan.*?\b')
    sorted3tuples = mytexan.sortListOfTuples(countsAndHits,1)
    result = mytexan.writeCSVFileCountHits3Tuples(sorted3tuples, \
        '..' + os.sep + 'outputs' + os.sep +'tuples3.txt', \
        'expan*')
    print(result)

q6()
```

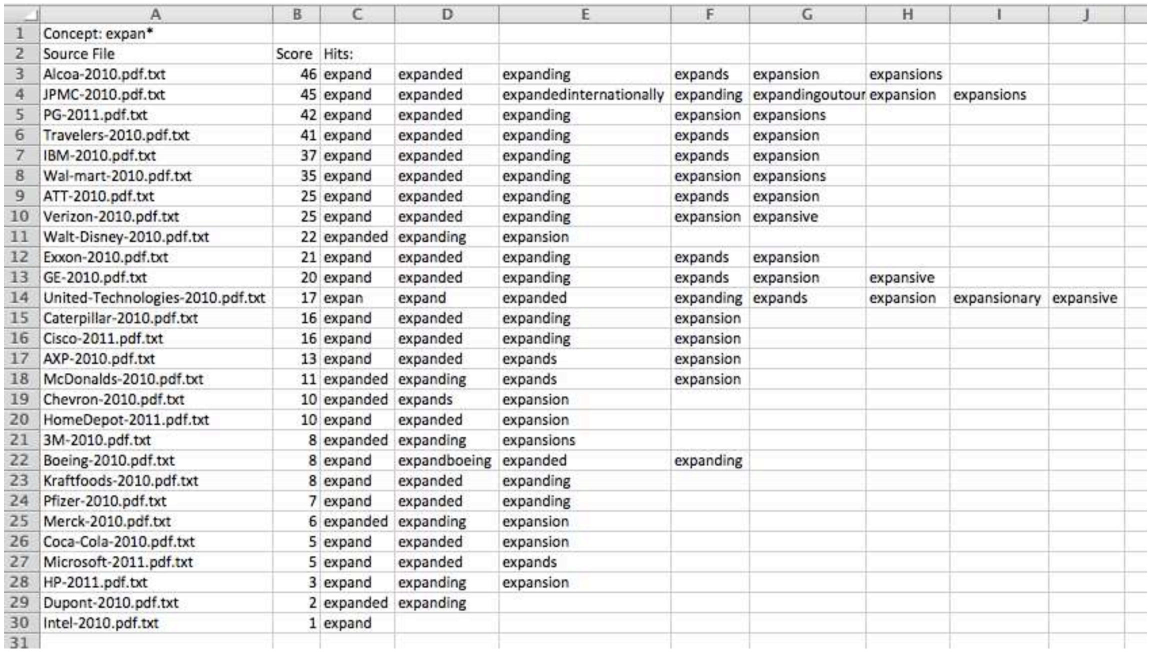
produces the message

```
Done writing file ../outputs/tuples3.txt.
```

when it completes. If you then launch Excel and import the file tuples3.txt, what you see will be (something very like) Figure 4.

Answer:

```
def writeCSVFileCountHits3Tuples(da3Tuples,daFile,conceptName):
    '''
Similar to writeCSVFileCount2Tuples, but adds hits list.
Given a list of 3-tuples, assumed to have the form
(<file name>,<concept score>, <hits list>), writes to the file
daFilethe conceptName in the first line.
The second line has "Source File,Score,Hits:".
Following that writes each 3-tuple to a line, with a comma
as the separater. The third item, a litst, is "flattened" and
each item in the list is written out with comma separations.
The tuples are written in the order given in da2Tuples.The
```



	A	B	C	D	E	F	G	H	I	J
1	Concept: expan*									
2	Source File	Score	Hits:							
3	Alcoa-2010.pdf.txt	46	expand	expanded	expanding	expands	expansion	expansions		
4	JPMC-2010.pdf.txt	45	expand	expanded	expandedinternationally	expanding	expandingoutour	expansion	expansions	
5	PG-2011.pdf.txt	42	expand	expanded	expanding	expansion	expansions			
6	Travelers-2010.pdf.txt	41	expand	expanded	expanding	expands	expansion			
7	IBM-2010.pdf.txt	37	expand	expanded	expanding	expands	expansion			
8	Wal-mart-2010.pdf.txt	35	expand	expanded	expanding	expansion	expansions			
9	ATT-2010.pdf.txt	25	expand	expanded	expanding	expands	expansion			
10	Verizon-2010.pdf.txt	25	expand	expanded	expanding	expansion	expansive			
11	Walt-Disney-2010.pdf.txt	22	expanded	expanding	expansion					
12	Exxon-2010.pdf.txt	21	expand	expanded	expanding	expands	expansion			
13	GE-2010.pdf.txt	20	expand	expanded	expanding	expands	expansion	expansive		
14	United-Technologies-2010.pdf.txt	17	expan	expand	expanded	expanding	expands	expansion	expansionary	expansive
15	Caterpillar-2010.pdf.txt	16	expand	expanded	expanding	expansion				
16	Cisco-2011.pdf.txt	16	expand	expanded	expanding	expansion				
17	AXP-2010.pdf.txt	13	expand	expanded	expands	expansion				
18	McDonalds-2010.pdf.txt	11	expanded	expanding	expands	expansion				
19	Chevron-2010.pdf.txt	10	expanded	expands	expansion					
20	HomeDepot-2011.pdf.txt	10	expand	expanded	expansion					
21	3M-2010.pdf.txt	8	expanded	expanding	expansions					
22	Boeing-2010.pdf.txt	8	expand	expandboeing	expanded	expanding				
23	Kraftfoods-2010.pdf.txt	8	expand	expanded	expanding					
24	Pfizer-2010.pdf.txt	7	expand	expanded	expanding					
25	Merck-2010.pdf.txt	6	expanded	expanding	expansion					
26	Coca-Cola-2010.pdf.txt	5	expand	expanded	expansion					
27	Microsoft-2011.pdf.txt	5	expand	expanded	expands					
28	HP-2011.pdf.txt	3	expand	expanding	expansion					
29	Dupont-2010.pdf.txt	2	expanded	expanding						
30	Intel-2010.pdf.txt	1	expand							
31										

Figure 4: Imported file produced by texan.writeCSVFileCountHits3Tuples()

```
function returns a message string when it has completed:
"Done writing file " +  daFile
    '''
    f = open(daFile,'w')
    f.write('Concept: ' + conceptName + '\n')
    f.write('Source File,Score,Hits:\n')
    #daSortedTuples = sortListOfTuples(da2Tuples,1)
    for aTuple in da3Tuples:
        f.write(aTuple[0] + ',' + str(aTuple[1]))
        for aWord in aTuple[2]:
            f.write(','+aWord)

        f.write('\n')

    f.close()
    return "Done writing file " + daFile + "."
```

3.7 Count and Sum the Pattern Hits

Write a function and associated code that conforms to the following template and documentation.

```
def dirFilesCountWordVectorHits(dirTxts,daConceptVector):
    '''
Similar to but adds to dirFilesCountWord.
And similar to dirFilesCountWordHits, but adds to it.
Modifies the second input parameter, daConceptVector, which
now it is assumed has the form as returned by
getConceptVector(). That is, a list whose first element
is the concept name and whose second element is a list
of tuples. Instead of a single search pattern, as before,
we have a concept vector.

The function returns a list of 3-tuples.
Given a directory, dirTxts, holding a number text files,
and a concept vector, daConceptVector, returns a list of 3-tuples
in which the first item is the file name, the second item is the
total number of times the search term in the search vector
appear in the file. The
third item is a list of the counts of patterns in the document matching
the input patterns in the concept vector. This list is in the order
in which the search patterns appear in the concept vector.
This function DOES use the re module to accomplish this task.
The function processes its input file after converting it to
lower case, so in the hit list we do not have duplications due
to capitalization.
    '''
[Your code here.]
```

When you run it, it should behave as follows. Running the following script (after importing texan.py) produces the following output.

```
def q7():
    '''
A little script to exercise the code for question 7.
    '''
    conceptVector = \
        mytexan.getConceptVectorLabeled(pathToConceptVectors,MarchExplore)
    #print(conceptVector)
    theDirTexts = pathToTextCollections + os.sep + DowIndTxts
    countWordHits = \
        mytexan.dirFilesCountWordVectorHits(theDirTexts,conceptVector)
    print(countWordHits)

q7()
```

When you examine the output, this is what you see:

```
>>>
['MarchExploreLabeled', [('3M-2010.pdf.txt', 101.0,
[0.0, 0.0, 1.0, 74.0, 0.0, 1.0, 5.0, 2.0, 18.0]),
('Alcoa-2010.pdf.txt', 86.0,
[4.0, 0.0, 0.0, 55.0, 0.0, 1.0, 7.0, 9.0, 10.0]),
('ATT-2010.pdf.txt', 82.0,
[1.0, 4.0, 2.0, 49.0, 1.0, 1.0, 2.0, 0.0, 22.0]),
('AXP-2010.pdf.txt', 270.0,
[0.0, 1.0, 1.0, 254.0, 0.0, 1.0, 7.0, 2.0, 4.0]),
('Boeing-2010.pdf.txt', 138.0,
[4.0, 0.0, 2.0, 96.0, 4.0, 0.0, 7.0, 9.0, 16.0]),
('Caterpillar-2010.pdf.txt', 9.0,
[0.0, 0.0, 0.0, 5.0, 0.0, 1.0, 0.0, 0.0, 3.0]),
```

(and so on).

Answer:

```
def dirFilesCountWordVectorHits(dirTxts,daConceptVector):
    '''
Similar to but adds to dirFilesCountWord.
And similar to dirFilesCountWordHits, but adds to it.
Modifies the second input parameter, daConceptVector, which
now it is assumed has the form as returned by
getConceptVector(). That is, a list whose first element
is the concept name and whose second element is a list
of tuples. Instead of a single search pattern, as before,
we have a concept vector.

The function returns a list of 3-tuples.
Given a directory, dirTxts, holding a number text files,
and a concept vector, daConceptVector, returns a list of 3-tuples
in which the first item is the file name, the second item is the
total number of times the search term in the search vector
appear in the file. The
third item is a list of the counts of patterns in the document matching
```

```
the input patterns in the concept vector. This list is in the order
in which the search patterns appear in the concept vector.
This function DOES use the re module to accomplish this task.
The function processes its input file after converting it to
lower case, so in the hit list we do not have duplications due
to capitalization.
    '''
    daFileNames = os.listdir(dirTxts)
    toReturnData = []
    # First element is the name of the concept:
    #toReturn.append(daConceptVector[0])
    # Get the list of tuples for the concept:
    conceptList = daConceptVector[1]
    #compiledPattern = re.compile(daWord,re.I)
    for daFileName in daFileNames:
        f = open(dirTxts + os.sep + daFileName,'r')
        ftext = f.read().lower()
        f.close()
        findingsCounts = []
        for (weight,term) in conceptList:
            wt = float(weight)
            compiledPattern = re.compile(term,re.I)
            daFindings = compiledPattern.findall(ftext)
        #print(daFindings)
            daCount = len(daFindings)
            daScore = daCount * wt
            findingsCounts.append(daScore)
            #daHits = sorted(set(daFindings))

        toReturnData.append((daFileName, sum(findingsCounts), findingsCounts))

    return [daConceptVector[0], toReturnData]
```

3.8 Write the Full Report

Write a function and associated code that conforms to the following template and documentation.

```
def writeCSVFileSumScoresShowScores(daData,daFile,conceptVector):
    '''
Similar to writeCSVFileCount2Tuples, but adds hits list.
Given daData, assumed to have the form of the return value
```

```
from dirFilesCountWordVectorHits(, writes to the file
daFile the conceptName in the first line.
The second line has "Source File,Score," followed by the
labels from the conceptVector.
Following that it writes each tuple to a line, with a comma
as the separater. The third item, a listt, is "flattened" and
each item (a hit count) in the list is written out with
comma separations.
The tuples are written in the order given in da2Tuples.The
function returns a message string when it has completed:
"Done writing file " +  daFile
    '''
[Your code here.]
```

Here's a script for running the function:

```
def q8():
    '''
A little script to exercise the code for question 8.
    '''
    aConVec = mytexan.getConceptVectorLabeled(pathToConceptVectors, \
        'MarchExploreLabeled.txt')
    theDirTexts = pathToTextCollections + os.sep + DowIndTxts
    someData = mytexan.dirFilesCountWordVectorHits(theDirTexts, aConVec)
    coreData = mytexan.sortListOfTuples(someData[1],1)
    theData = [someData[0], coreData]
    daFile = '..' + os.sep + 'outputs' + os.sep + 'SumShowScores.txt'
    aResult = mytexan.writeCSVFileSumScoresShowScores(theData, \
        daFile, aConVec)
    print("Done. " + daFile + ".")

q8()
```

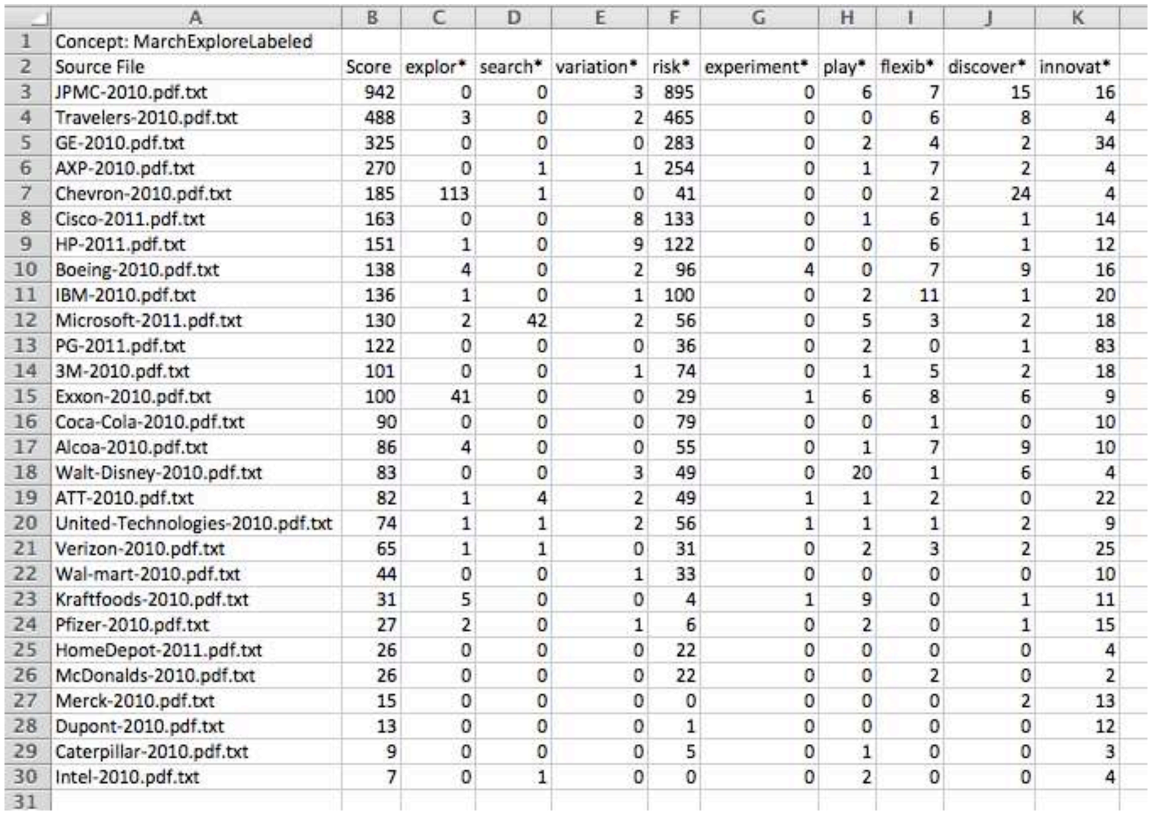
When you run it, it should behave as follows:

```
Done. ../outputs/SumShowScores.txt.
```

If you then launch Excel and import the file SumShowScores.txt, what you see will be (something very like) Figure 5.

Answer:

```
def writeCSVFileSumScoresShowScores(daData,daFile,conceptVector):
    '''
```



	A	B	C	D	E	F	G	H	I	J	K
1	Concept: MarchExploreLabeled										
2	Source File	Score	explor*	search*	variation*	risk*	experiment*	play*	flexib*	discover*	innovat*
3	JPMC-2010.pdf.txt	942	0	0	3	895	0	6	7	15	16
4	Travelers-2010.pdf.txt	488	3	0	2	465	0	0	6	8	4
5	GE-2010.pdf.txt	325	0	0	0	283	0	2	4	2	34
6	AXP-2010.pdf.txt	270	0	1	1	254	0	1	7	2	4
7	Chevron-2010.pdf.txt	185	113	1	0	41	0	0	2	24	4
8	Cisco-2011.pdf.txt	163	0	0	8	133	0	1	6	1	14
9	HP-2011.pdf.txt	151	1	0	9	122	0	0	6	1	12
10	Boeing-2010.pdf.txt	138	4	0	2	96	4	0	7	9	16
11	IBM-2010.pdf.txt	136	1	0	1	100	0	2	11	1	20
12	Microsoft-2011.pdf.txt	130	2	42	2	56	0	5	3	2	18
13	PG-2011.pdf.txt	122	0	0	0	36	0	2	0	1	83
14	3M-2010.pdf.txt	101	0	0	1	74	0	1	5	2	18
15	Exxon-2010.pdf.txt	100	41	0	0	29	1	6	8	6	9
16	Coca-Cola-2010.pdf.txt	90	0	0	0	79	0	0	1	0	10
17	Alcoa-2010.pdf.txt	86	4	0	0	55	0	1	7	9	10
18	Walt-Disney-2010.pdf.txt	83	0	0	3	49	0	20	1	6	4
19	ATT-2010.pdf.txt	82	1	4	2	49	1	1	2	0	22
20	United-Technologies-2010.pdf.txt	74	1	1	2	56	1	1	1	2	9
21	Verizon-2010.pdf.txt	65	1	1	0	31	0	2	3	2	25
22	Wal-mart-2010.pdf.txt	44	0	0	1	33	0	0	0	0	10
23	Kraftfoods-2010.pdf.txt	31	5	0	0	4	1	9	0	1	11
24	Pfizer-2010.pdf.txt	27	2	0	1	6	0	2	0	1	15
25	HomeDepot-2011.pdf.txt	26	0	0	0	22	0	0	0	0	4
26	McDonalds-2010.pdf.txt	26	0	0	0	22	0	0	2	0	2
27	Merck-2010.pdf.txt	15	0	0	0	0	0	0	0	2	13
28	Dupont-2010.pdf.txt	13	0	0	0	1	0	0	0	0	12
29	Caterpillar-2010.pdf.txt	9	0	0	0	5	0	1	0	0	3
30	Intel-2010.pdf.txt	7	0	1	0	0	0	2	0	0	4
31											

Figure 5: Imported file produced by texan.writeCSVFileSumScoresShowScores()

```
Similar to writeCSVFileCount2Tuples, but adds hits list.
Given daData, assumed to have the form of the return value
from dirFilesCountWordVectorHits(, writes to the file
daFile the conceptName in the first line.
The second line has "Source File,Score," followed by the
labels from the conceptVector.
Following that it writes each tuple to a line, with a comma
as the separater. The third item, a listt, is "flattened" and
each item (a hit count) in the list is written out with
comma separations.
The tuples are written in the order given in da2Tuples.The
function returns a message string when it has completed:
"Done writing file " +  daFile
    '''
    conceptName = daData[0]
    f = open(daFile,'w')
    f.write('Concept: ' + conceptName + '\n')
    f.write('Source File,Score')
    conceptTuples = conceptVector[1]
    for (a, b, label) in conceptTuples:
        f.write(',' + label)
    f.write('\n')
    da3Tuples = daData[1]
    #daSortedTuples = sortListOfTuples(da2Tuples,1)
    for aTuple in da3Tuples:
        f.write(aTuple[0] + ',' + str(aTuple[1]))
        for aCount in aTuple[2]:
            f.write(','+ str(aCount))

        f.write('\n')

    f.close()
    return "Done writing file " + daFile + "."
```

---

4 Generalizing All This a Bit

We have developed some useful functions in the context of exploring concept matching between a concept vector based on March's exploration vocabulary and a corpus (document collection) composed of annual reports from the Dow Jones Industrials. In general, there will be a many-to-many relationship between concept vectors and corpora (plural of "corpus"). We should like our software to accommodate exploring such many-to-many relationships.

To begin, we have a concept vector for March's exploitation vocabulary:

```
1$$$\bexploit.*?\b$$$exploit*
1$$$\brefine.*?\b$$$refine*
1$$$\bchoice.*?\b$$$choice*
1$$$\bproduction.*?\b$$$production*
1$$$\befficien.*?\b$$$efficien*
1$$$\bselect.*?\b$$$select*
1$$$\bimplement.*?\b$$$implement*
1$$$\bexecut.*?\b$$$execut*
```

It resides in the MarchExploitLabeled.txt file in the `concepts` directory, whose contents are listed by running this script:

```
def listConceptVectorFiles():
    '''
Prints a list of the file names of the available
concept vector files.
    '''
    print('pathToConceptVectors = ' + pathToConceptVectors)
    daList = os.listdir(pathToConceptVectors)
    print(daList)
listConceptVectorFiles()
```

When you run it you will see

```
pathToConceptVectors = ../concepts
['MarchExploitLabeled.txt', 'MarchExploreLabeled.txt']
```

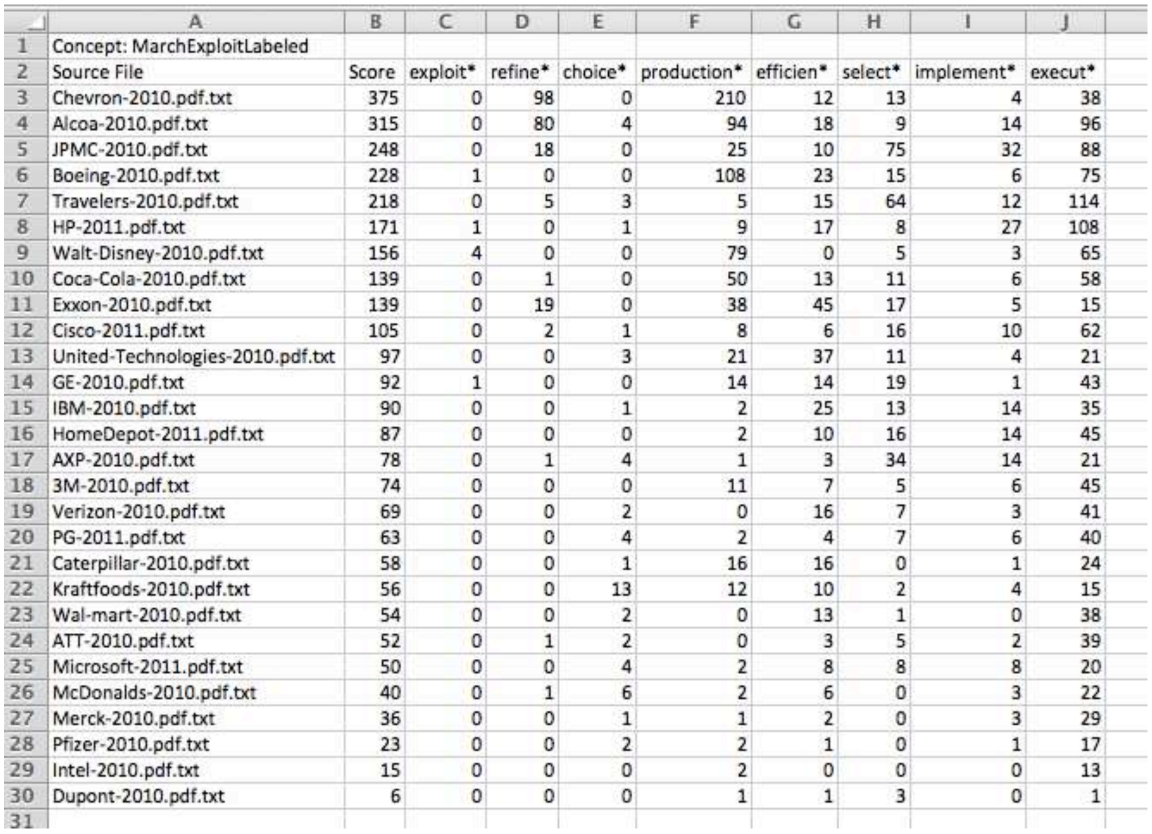
We can generalize the procedure of question 8 to accommodate arbitrary concept vectors and arbitrary corpora as follows:

```
def reportCorpusConcept(corpus, conceptFile):
    '''
A little script to exercise the code for question 8.
    '''
    print('pathToConceptVectors = ' + pathToConceptVectors)
    print('pathToTextCollections = ' + pathToTextCollections)
    aConVec = mytexan.getConceptVectorLabeled(pathToConceptVectors, \
        conceptFile)
    theDirTexts = pathToTextCollections + os.sep + corpus
    someData = mytexan.dirFilesCountWordVectorHits(theDirTexts, aConVec)
    coreData = mytexan.sortListOfTuples(someData[1],1)
    theData = [someData[0], coreData]
    daFile = '..' + os.sep + 'outputs' + os.sep + 'Results' + conceptFile
    aResult = mytexan.writeCSVFileSumScoresShowScores(theData, \
        daFile, aConVec)
    print("Done. " + daFile + ".")
```

if you add

```
reportCorpusConcept('DowJonesIndustrials2009','MarchExploitLabeled.txt')
```

to the module and execute it, a CSV file called ResultsMarchExploitLabeled.txt will be written to the `outputs` directory. If you then import this file into Excel, you get what you see in Figure 6. Compare it with Figure 5, found on page 25.



	A	B	C	D	E	F	G	H	I	J
1	Concept: MarchExploitLabeled									
2	Source File	Score	exploit*	refine*	choice*	production*	efficien*	select*	implement*	execut*
3	Chevron-2010.pdf.txt	375	0	98	0	210	12	13	4	38
4	Alcoa-2010.pdf.txt	315	0	80	4	94	18	9	14	96
5	JPMC-2010.pdf.txt	248	0	18	0	25	10	75	32	88
6	Boeing-2010.pdf.txt	228	1	0	0	108	23	15	6	75
7	Travelers-2010.pdf.txt	218	0	5	3	5	15	64	12	114
8	HP-2011.pdf.txt	171	1	0	1	9	17	8	27	108
9	Walt-Disney-2010.pdf.txt	156	4	0	0	79	0	5	3	65
10	Coca-Cola-2010.pdf.txt	139	0	1	0	50	13	11	6	58
11	Exxon-2010.pdf.txt	139	0	19	0	38	45	17	5	15
12	Cisco-2011.pdf.txt	105	0	2	1	8	6	16	10	62
13	United-Technologies-2010.pdf.txt	97	0	0	3	21	37	11	4	21
14	GE-2010.pdf.txt	92	1	0	0	14	14	19	1	43
15	IBM-2010.pdf.txt	90	0	0	1	2	25	13	14	35
16	HomeDepot-2011.pdf.txt	87	0	0	0	2	10	16	14	45
17	AXP-2010.pdf.txt	78	0	1	4	1	3	34	14	21
18	3M-2010.pdf.txt	74	0	0	0	11	7	5	6	45
19	Verizon-2010.pdf.txt	69	0	0	2	0	16	7	3	41
20	PG-2011.pdf.txt	63	0	0	4	2	4	7	6	40
21	Caterpillar-2010.pdf.txt	58	0	0	1	16	16	0	1	24
22	Kraftfoods-2010.pdf.txt	56	0	0	13	12	10	2	4	15
23	Wal-mart-2010.pdf.txt	54	0	0	2	0	13	1	0	38
24	ATT-2010.pdf.txt	52	0	1	2	0	3	5	2	39
25	Microsoft-2011.pdf.txt	50	0	0	4	2	8	8	8	20
26	McDonalds-2010.pdf.txt	40	0	1	6	2	6	0	3	22
27	Merck-2010.pdf.txt	36	0	0	1	1	2	0	3	29
28	Pfizer-2010.pdf.txt	23	0	0	2	2	1	0	1	17
29	Intel-2010.pdf.txt	15	0	0	0	2	0	0	0	13
30	Dupont-2010.pdf.txt	6	0	0	0	1	1	3	0	1
31										

Figure 6: Imported file produced by texan.writeCSVFileSumScoresShowScores() with MarchExploitLabeled.txt concept vector

So that's for the concept vector side. On the corpora side, if we add additional corpora we can use them as well. And we have. If you run this script:

```
def listTextCorpora():
    '''
Prints a list of the directory names of the available
corpora (document collections) in text format.
    '''
    print('pathToTextCollections = ' + pathToTextCollections)
    daList = os.listdir(pathToTextCollections)
    print(daList)

#
listTextCorpora()
```

you will see on output

```
pathToTextCollections = ../collections/text
['Consumer Goods', 'DowJonesIndustrials2009', 'Healthcare',
'IndustrialGoods']
```

Running

```
reportCorpusConcept('Healthcare','MarchExploitLabeled.txt')
```

leads to the report in Figure 7.

Running

```
reportCorpusConcept('Healthcare','MarchExploreLabeled.txt')
```

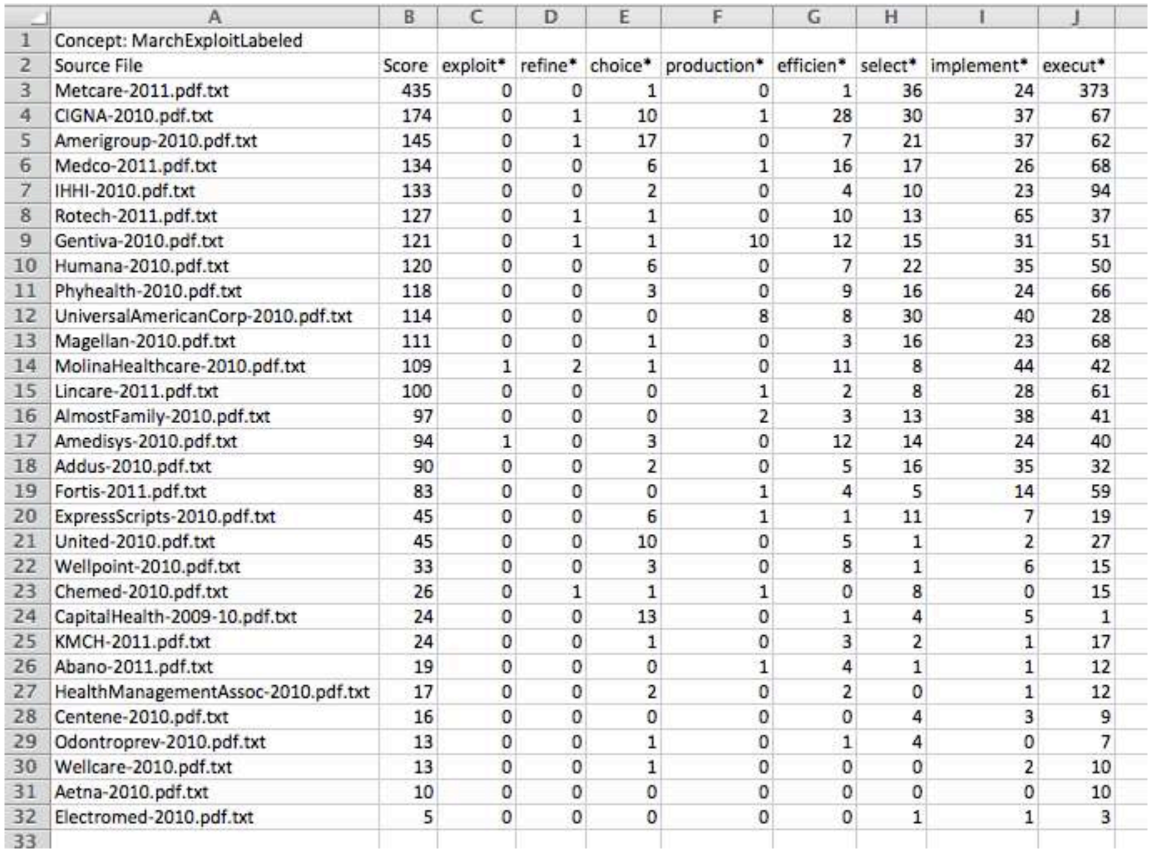
leads to the report in Figure 8.

So, by these means we are able to general a great deal of interesting and useful data.

5 Assignment

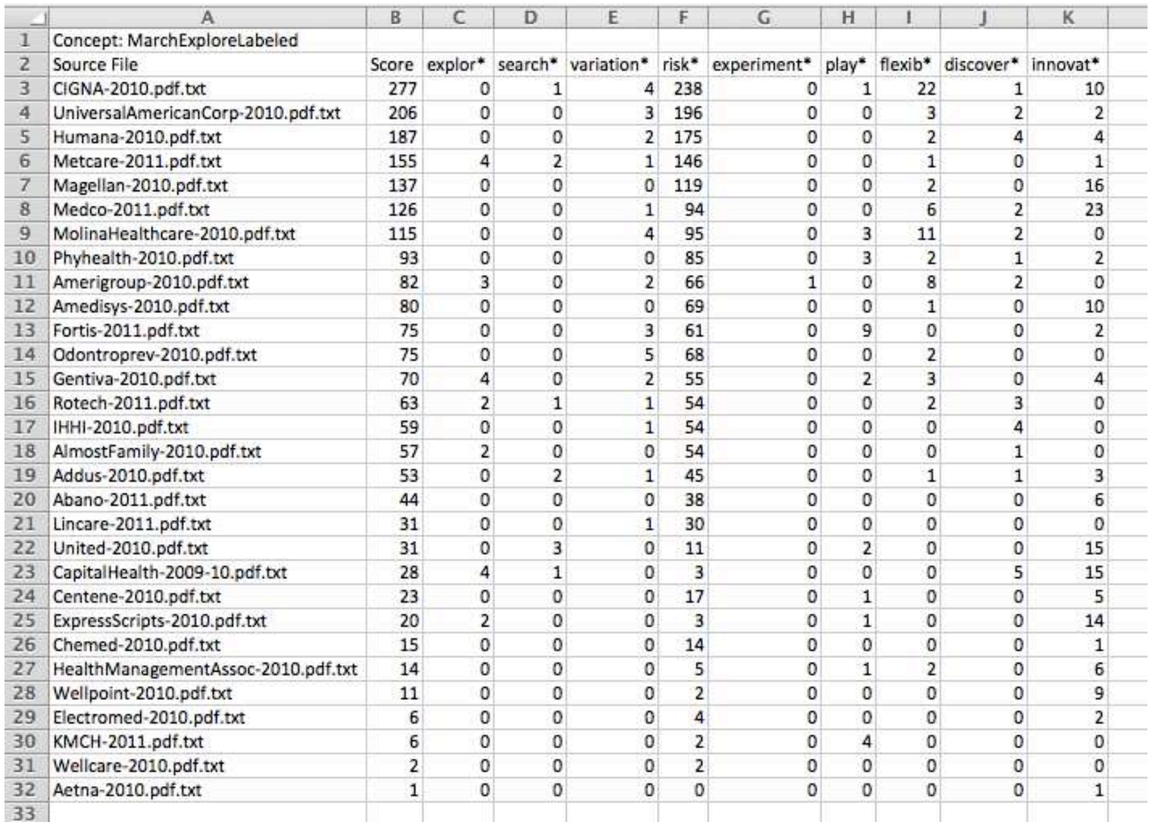
On WebCafé you will find a file named rtr.zip. You should download it to a place of convenience for you and unzip it. This will produce a directory called `rtr` ("Root of the Text Repository"), with the following subdirectories (folders):

```
rtr/
    code/
    collections/
        text/
    concepts/
    outputs/
```



	A	B	C	D	E	F	G	H	I	J
1	Concept: MarchExploitLabeled									
2	Source File	Score	exploit*	refine*	choice*	production*	efficien*	select*	implement*	execut*
3	Metcare-2011.pdf.txt	435	0	0	1	0	1	36	24	373
4	CIGNA-2010.pdf.txt	174	0	1	10	1	28	30	37	67
5	Amerigroup-2010.pdf.txt	145	0	1	17	0	7	21	37	62
6	Medco-2011.pdf.txt	134	0	0	6	1	16	17	26	68
7	IHHI-2010.pdf.txt	133	0	0	2	0	4	10	23	94
8	Rotech-2011.pdf.txt	127	0	1	1	0	10	13	65	37
9	Gentiva-2010.pdf.txt	121	0	1	1	10	12	15	31	51
10	Humana-2010.pdf.txt	120	0	0	6	0	7	22	35	50
11	Phyhealth-2010.pdf.txt	118	0	0	3	0	9	16	24	66
12	UniversalAmericanCorp-2010.pdf.txt	114	0	0	0	8	8	30	40	28
13	Magellan-2010.pdf.txt	111	0	0	1	0	3	16	23	68
14	MolinaHealthcare-2010.pdf.txt	109	1	2	1	0	11	8	44	42
15	Lincare-2011.pdf.txt	100	0	0	0	1	2	8	28	61
16	AlmostFamily-2010.pdf.txt	97	0	0	0	2	3	13	38	41
17	Amedisys-2010.pdf.txt	94	1	0	3	0	12	14	24	40
18	Addus-2010.pdf.txt	90	0	0	2	0	5	16	35	32
19	Fortis-2011.pdf.txt	83	0	0	0	1	4	5	14	59
20	ExpressScripts-2010.pdf.txt	45	0	0	6	1	1	11	7	19
21	United-2010.pdf.txt	45	0	0	10	0	5	1	2	27
22	Wellpoint-2010.pdf.txt	33	0	0	3	0	8	1	6	15
23	Chemed-2010.pdf.txt	26	0	1	1	1	0	8	0	15
24	CapitalHealth-2009-10.pdf.txt	24	0	0	13	0	1	4	5	1
25	KMCH-2011.pdf.txt	24	0	0	1	0	3	2	1	17
26	Abano-2011.pdf.txt	19	0	0	0	1	4	1	1	12
27	HealthManagementAssoc-2010.pdf.txt	17	0	0	2	0	2	0	1	12
28	Centene-2010.pdf.txt	16	0	0	0	0	0	4	3	9
29	Odontroprev-2010.pdf.txt	13	0	0	1	0	1	4	0	7
30	Wellcare-2010.pdf.txt	13	0	0	1	0	0	0	2	10
31	Aetna-2010.pdf.txt	10	0	0	0	0	0	0	0	10
32	Electromed-2010.pdf.txt	5	0	0	0	0	0	1	1	3
33										

Figure 7: From running `reportCorpusConcept('Healthcare','MarchExploitLabeled.txt')`



	A	B	C	D	E	F	G	H	I	J	K
1	Concept: MarchExploreLabeled										
2	Source File	Score	explor*	search*	variation*	risk*	experiment*	play*	flexib*	discover*	innovat*
3	CIGNA-2010.pdf.txt	277	0	1	4	238	0	1	22	1	10
4	UniversalAmericanCorp-2010.pdf.txt	206	0	0	3	196	0	0	3	2	2
5	Humana-2010.pdf.txt	187	0	0	2	175	0	0	2	4	4
6	Metcare-2011.pdf.txt	155	4	2	1	146	0	0	1	0	1
7	Magellan-2010.pdf.txt	137	0	0	0	119	0	0	2	0	16
8	Medco-2011.pdf.txt	126	0	0	1	94	0	0	6	2	23
9	MolinaHealthcare-2010.pdf.txt	115	0	0	4	95	0	3	11	2	0
10	Phyhealth-2010.pdf.txt	93	0	0	0	85	0	3	2	1	2
11	Amerigroup-2010.pdf.txt	82	3	0	2	66	1	0	8	2	0
12	Amedisys-2010.pdf.txt	80	0	0	0	69	0	0	1	0	10
13	Fortis-2011.pdf.txt	75	0	0	3	61	0	9	0	0	2
14	Odontroprev-2010.pdf.txt	75	0	0	5	68	0	0	2	0	0
15	Gentiva-2010.pdf.txt	70	4	0	2	55	0	2	3	0	4
16	Rotech-2011.pdf.txt	63	2	1	1	54	0	0	2	3	0
17	IHHI-2010.pdf.txt	59	0	0	1	54	0	0	0	4	0
18	AlmostFamily-2010.pdf.txt	57	2	0	0	54	0	0	0	1	0
19	Addus-2010.pdf.txt	53	0	2	1	45	0	0	1	1	3
20	Abano-2011.pdf.txt	44	0	0	0	38	0	0	0	0	6
21	Lincare-2011.pdf.txt	31	0	0	1	30	0	0	0	0	0
22	United-2010.pdf.txt	31	0	3	0	11	0	2	0	0	15
23	CapitalHealth-2009-10.pdf.txt	28	4	1	0	3	0	0	0	5	15
24	Centene-2010.pdf.txt	23	0	0	0	17	0	1	0	0	5
25	ExpressScripts-2010.pdf.txt	20	2	0	0	3	0	1	0	0	14
26	Chemed-2010.pdf.txt	15	0	0	0	14	0	0	0	0	1
27	HealthManagementAssoc-2010.pdf.txt	14	0	0	0	5	0	1	2	0	6
28	Wellpoint-2010.pdf.txt	11	0	0	0	2	0	0	0	0	9
29	Electromed-2010.pdf.txt	6	0	0	0	4	0	0	0	0	2
30	KMCH-2011.pdf.txt	6	0	0	0	2	0	4	0	0	0
31	Wellcare-2010.pdf.txt	2	0	0	0	2	0	0	0	0	0
32	Aetna-2010.pdf.txt	1	0	0	0	0	0	0	0	0	1
33											

Figure 8: From running `reportCorpusConcept('Healthcare','MarchExploreLabeled.txt')`

You are to develop your code in the context of this directory system. In the `code` directory you will find a Python module called `myexercisetexan.py`. It contains useful material, discussed above, for testing your code and helping you to develop your code.

Your assignment is to complete each of the tasks in §3 (all 8 of them). Put all of your code in a single Python module called `mytexan.py` ("MY TEXt ANalytics" module). Put your module in the `code` directory and be sure it works when called and tested by the `myexercisetexan.py` module. Once you have completed this assignment, create a folder on WebCafé called `case2` and put just your `mytexan.py` module in it.

This case 2 assignment is due by 5 p.m. on Saturday 5 May 2012.

This due date is different from what's on the syllabus (it's later) and what was stated (mistakenly, a typo) in the case 2 assignment document (infeasible for handing in grades).

References

Gottschalk, L. A. (1995). *Content Analysis of Verbal Behavior: New Findings and Clinical Applications*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Gottschalk, L. A. and Gleser, G. C. (1969). *The Measurement of Psychological States Through the Content Analysis of Verbal Behavior*. University of California Press, Berkeley and Los Angeles, CA.

Gottschalk, L. A., Winget, C. N., and Gleser, G. C. (1969). *Manual of Instructions for Using the Gottschalk-Gleser Content Analysis Scales: Anxiety, Hostility, and Social Alienation–Personal Disorganization*. University of California Press, Berkeley and Los Angeles, CA.

Larcker, D. F. and Zakolyukina, A. A. (2012). Detecting deceptive discussions in conference calls. *Journal of Accounting Research*, pages no–no. `http://dx.doi.org/10.1111/j.1475-679X.2012.00450.x`.

Loewenstein, J., Ocasio, W., and Jones, C. (2012). Vocabularies and vocabulary structure: A new approach linking categories, practices, and institutions. *The Academy of Management Annals*. Available online 13 March 2012. `http://dx.doi.org/10.1080/19416520.2012.660763`.

March, J. G. (1991). Exploration and exploitation in organizational learning. *Organization Science*, 2:71–87.

Tetlock, P. C., Saar-Tsechansky, M., and Macskassy, S. (2008). More than words: Quantifying language to measure firms' fundamentals. *The Journal of Finance*, LXIII(3):1437–1467.

Uotila, J., Maula, M., Keil, T., and Zahra, S. A. (2009). Exploration, exploitation, and financial performance: Analysis of S&P 500 corporations. *Strategic Management Journal*, 30:221–231.