Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

# Latent Semantic Analysis (LSA) tools

Natalia Derbentseva
Peter J. Kwantes
Philip Terhaar

**Defence R&D Canada**
Technical Note
DRDC Toronto TN 2012-079
July 2012

Canada

# Latent Semantic Analysis (LSA) tools

Natalia Derbentseva
Peter J. Kwantes
Philip Terhaar

**Defence R&D Canada – Toronto**

Technical Note
DRDC Toronto TN 2012-079
July 2012

Principal Author

*Original signed by Natalia Derbentseva*

Natalia Derbentseva

Defence Scientist

Approved by

*Original signed by Keith Stewart*

Keith Stewart

Head of Socio-Cognitive Systems Section

# Table of contents

This page intentionally left blank.

# 1 Introduction

Latent Semantic Analysis (LSA; Landauer & Dumais, 1997; Landauer, Foltz, & Laham, 1998) is a computational model that uses a large collection of unstructured documents to construct semantic representations for words. The representations are based on a statistical analysis of the terms' occurrences within and across documents, and take the form of a vector. The semantic similarity between resulting word representations can be compared by calculating their cosine. After an LSA space is created, it can be queried to provide a word-to-word, word-to-document, or document-to-document comparisons to determine their semantic similarity by taking the cosine of the angle formed by their vector representations. When comparing words to documents (and sometimes documents to each other), LSA uses what is called a "bag of words" approach to representing the semantic contents of a document. In a "bag of words", the order of terms in a document does not matter, and the semantic representation of a document is formed by summing the vectors of all its content words.

LSA requires a relatively large set of short documents to generate a semantic space (i.e., from several hundreds to tens of thousands). The large number helps to ensure that many words from a variety of contexts (operationally defined as documents) are available for analysis.

Before a semantic space for the terms in the corpus is constructed by our version of LSA, the documents need to be prepared. Punctuation is removed from the documents, words are brought to lower case, and words that appear in most or all documents in the corpus, such as articles, prepositions, and others that do not help to differentiate the contextual uses of the terms are removed from the documents, so called "stop words". Typically, the same set of stop words is removed from every document in a corpus before analysis. However, under certain conditions different stop word lists have to be removed from different groups of documents before they can be combined into a single corpus and analyzed. Because LSA usually requires several thousands of documents, removing custom stop word lists from them is a time-consuming and labour intensive task. One of the tools reported in this Technical Note (TN) supports this task.

One of LSA's requirements is that for a word to be included in the space, it must occur in at least two documents. This poses a potential challenge when a corpus for LSA analysis needs to be created from a limited number of relatively short documents. In this situation, a large number of words can potentially be excluded from the analysis because they do not meet the minimum document occurrence criteria. At least partially, it occurs not because there are many unique words, but because the same word is used in different forms in different documents (for example, singular and plural forms of nouns). Thus, bringing different forms of the same word to a single form across documents in a training corpus can partially diminish their unnecessary exclusion from the corpus. The tools described in this TN implemented stemming of words prior to the construction of the corpus and the LSA space.

LSA builds its semantic representation from a consideration of words' contextual use. That is, words that tend to be used in similar contexts will tend to have a semantic association. Not surprisingly then, the same two words can have different representations depending on the collection of documents used to create a semantic space, For example, the word "mouse" means something different in the context of computers than in the context of animals or vermin. Hence, depending on the purpose of the analysis, the same set of words or texts can be analyzed with different LSA spaces. The WIKIPEDIA subcorpora tool (Stone and Dennis, 2012) allows the user to create custom corpora from a Wikipedia archive.

Finally, the existing functionality of DRDC's SEMMOD package allows pair-wise comparisons of words or texts, one pair at a time with a single LSA space. Therefore, a large number of such comparisons are a time-consuming and a labour intensive task. One of the tools described in this TN provides automation of multiple LSA comparisons and supports application of several LSA spaces.

## 1.1 Utilities to Support Latent Semantic Analysis (LSA)

This TN documents a set of tools that were developed to support the ongoing work at DRDC Toronto that applies. The described set of tools builds on the WIKIPEDIA Subcorpora Tool (Stone & Dennis, 2012) and the Semantic Models (SEMMOD) module (v 1.5, Dennis & Stone, 2011) that were developed by the Ohio State University (OSU) under the contract W7711-067985/001/TOR for a Technology Investment Fund (TIF) project (15da05) and an Applied Research Project (ARP, 15ah) . The tools described in this TN extend the capabilities offered by the OSU modules and support 2 main functions:

1. Preparation of a custom corpus ready for LSA's semantic space generation. In addition to standard procedures that are performed on a collection of documents to prepare them for LSA space generation, such as removing punctuation and stop words, the tools developed also allow the following operations:
   - The application of a customized stop word list to a single document file prior to its inclusion into a corpus;
   - The stemming the words in a file using the Porter stemming algorithm;
   - The exclusion of words that occur in fewer than a specified number of documents.
2. Automating the process of conducting document-to-document comparisons where documents are created from different LSA spaces, with the following options:
   - application of a customized stop word list to the input file;
   - stemming the words in the input file;
   - specifying multiple LSA spaces to be applied to the input file;
   - generating output in the form of a document-by-document table or a single column to facilitate subsequent analysis of the results.

-

# 2 LSA supporting tools

This section describes the tools that support LSA including their functions, input requirements, output formats, and execution requirements. The tools were developed in the Python 2.6.5 programming language for the Linux environment. In the interest of time no graphical user interface (GUI) was developed, and individual files containing scripts are run from the command line. The source code is currently stored on DRDC Toronto public server Pluto at the following location: \\Pluto\public\VISTIF\LSATools. Print outs of the source code are provided in Annexes A-C.

## 2.1 Tools to support custom corpus generation

This sub-section describes the tools that were developed to support custom corpus generation, including i) possibility to apply a custom stop words list to a single document prior to its inclusion into a corpus, ii) stemming the words in a file using the Porter stemming model; iii) excluding from the corpus words that occur in fewer than the specified number of documents.

### 2.1.1 Removing custom stop words from a file

This module was developed to allow the user to remove a custom set of words from a single text file before the file is included into a corpus file. This module is useful when different stop word lists have to be applied to different documents before they are combined into a single corpus. NOTE: prior to removing stop words, punctuation is also removed from the file.

***File:*** *CustomStopW.py* (source code is in Appendix A)

***Command line prompt example:***

```
$ python CustomStopW.py -i /home/MyDocuments/TextFiles/Agreeableness.txt -s /home/MyDocuments/StopListFiles/Agreeableness_Stop.txt
```

***Options and arguments:***

*-i* followed by the input file name. Requires either a filename with a complete path, or just a name of a file in the default location: "*input/*" subfolder. If this option is omitted, the default input file will be processed, which is "*input/default.txt*".

*-s* followed by the stop words list file. Requires either a filename with a complete path, or just a name of the file in the default location: "*CleaningFiles/*" subfolder. If this option is omitted, the default stop words list file will be used: "*CleaningFiles/stopList_Words.txt*".

***Input requirements:***

- Input file: Plain text, preferably in UTF-8 format with no special requirements. Default input file location is "*input/*" subfolder.

- Stop words list file: Plain text, preferably in UTF-8 format. Each stop word must appear on a separate line.

***Output:***

Punctuation and the words found in the stop words list file are removed from the input file, and the result is stored in a new file, which is saved in the same location as the input file. The line breaks in the input file are preserved. The name of the output file complies with the following naming convention:

<input file>_CSWr.<input file extension>.

***Execution requirements:***

The folder containing the *CustomStopW.py* must also contain the following files:

- *corpusCleaningTools.py*
- *EntityClassify.py*

The folder containing the stop word list file must also contain the punctuation stop list: "*stopList_Punctuation.txt*".

### 2.1.2 Custom corpus preparation

This module prepares a text document for LSA space generation, and outputs a collection of documents in a file with the extension *.cor*. It allows the use to apply a custom stop words list to the input file, stem the words, and specify the minimum frequency of word's occurrence.

***File:*** *CustomCorpusPreparation.py* (source code is in Appendix B)

***Command line prompt example:***

```
$ python CustomCorpusPreparation.py -i /home/MyDocuments/TextFiles/Agreeableness.txt –f 3
-s /home/MyDocuments/StopListFiles/Agreeableness_Stop.txt –t
```

***Options and arguments:***

*-i* followed by the input file name. Requires either a filename with a complete path, or just a name of a file in the default location: "*input/*" subfolder. If this option is omitted, the default input file will be processed, which is "*input/default.txt*".

*-f* to indicate the minimum number of documents in which a word must occur to be included in the corpus. This argument must be followed by an argument. The default value is 2.

*-s* followed by the stop words list file. Requires either a filename with a complete path, or just a name of the file in the default location: "*CleaningFiles/*" subfolder. If this

option is omitted, the default stop word list file will be used: "*CleaningFiles/stopList_Words.txt*".

*-t* no argument is required. If "-*t*" is included, the words will be stemmed after the documents are cleaned, and they will not be stemmed if "-*t*" is omitted. The default is to omit stemming.

***Input requirements:***

- Input file: Plain text, preferably in UTF-8 format with no special requirements. Default input file location is "*input/*" subfolder.
- Stop words list file: Plain text, preferably in UTF-8 format. Each stop word must appear on a separate line.
- The folder containing the stop word list file must also contain the punctuation stop list: "*stopList_Punctuation.txt*".

***Output:***

A corpus ready for LSA space generation: Punctuation and stop words removed, only words that appear with sufficient frequency across documents are retained. The documents are separated by a blank line. The result is stored in a new file which is placed in the same location as the input file with the following naming convention:

<input file>.cor

The words that were removed from the documents are stored in a file with the name

<input file>.cor.removed

***Execution requirements:***

- *nltk.stem.porter*
- *semmod.lsa*
- The folder containing the *CustomStopW.py* must also contain the following files:
  - *corpusCleaningTools.py*
  - *EntityClassify.py*
- The folder containing the stop word list file must also contain the punctuation stop list: "*stopList_Punctuation.txt*".

#### 2.1.2.1 Word stemming module

Words can appear in several forms, like singular and plural versions of the nouns *book* and *books.* Different forms of a word are treated as different terms by LSA. As a result, one could argue that the number of unique words recognised by the system in a collection of documents is somewhat inflated. Such inflation can pose a problem for generating a semantic space from a relatively small collection of short documents. In such a collection, a substantial number of words could be excluded from the analysis because of their “uniqueness” in the corpus, thus, resulting in a limited semantic space.

Word stemming can be used to mitigate this issue. Word stemming is a process that brings affixed forms of a word to its base form, that is, its stem. Reducing words to their stems decreases the number of unique words in a corpus, and increasing their frequency in the corpus. Such processing could improve the quality of a semantic space constructed from a relatively small number of short documents.

A function was developed that transforms a string into a collection of candidate stems. To reduce noise introduced by the stemming process itself, a validation step checks whether a stemmed form is a recognized word itself. If the stem is a recognized word, then the original word in the document is changed into its stemmed form. If the stemmed form is not recognized, the letters that were removed by stemming are added back to the stemmed form one by one. The check is repeated after each letter is added until a recognized word is found.

This function uses the Porter stemmer module from the Natural Language Toolkit (NLTK) package and it also uses an existing semantic space built using LSA to check for the words’ existence. This function can be called from other modules by importing it and passing the required arguments to it.

***Resides in file:*** *CustomCorpusPreparation.py* (code is in Appendix B)

***Usage example:***

```
from CustomCorpusPreparation import StemmerWithLSAcheck

…

        StemmedDocuments = StemmerWithLSAcheck (ListOfDocuments, FileNameBase)
```

***Required arguments:***

- **ListOfDocuments** – either a list or a dictionary that contains strings to be stemmed
- **FileNameBase** – input file name without extension. It is used to create stemming output files

***Returns:*** Stemmed strings in the form they were provided to the function, i.e., either a list or a dictionary

Also creates:

- a file with the record of stemming steps, file name <inputfilename>_STemDebug" and
- a file that contains a list of the original words and their form after stemming, file name: <inputfilename>_AfterStemmingWords.txt"

***Execution requirements:***

All modules required by the *CustomCorpusPreparation.py* file (see 2.1.2), and the LSA semantic space contained in the file, *LSAspaces/tasaCleaned_fromPluto.lsa*

## 2.2 Tools supporting document-by-document comparisons using LSA with multiple semantic spaces

This module automates document-by-document comparisons, and it allows the user to measure the semantic similarity among all document pairs in an input file. Further, the comparisons can be conducted on representations derived from multiple semantic spaces from LSA at the same time. The module generates a document-by-document similarity matrix and saves it as comma separated values (CSV) in a separate ASCII file for each semantic space. It supports the following options:

- applying a customized stop words list to the input file;
- stemming the words in the input file;
- analysing the input file with multiple LSA spaces;
- generating output in the form of a document-by-document table , as well as a single column created by concatenating the columns of the matrix to facilitate subsequent analysis.

***File:*** *doc_by_doc_Multi_LSA_Custom_STOPlist.py* (source code is in Appendix C)

***Command line prompt example:***

```
$ python doc_by_doc_Multi_LSA_Custom_STOPlist.py -i
/home/MyDocuments/TextFiles/Agreeableness.txt -f 3 -s
/home/MyDocuments/StopListFiles/Agreeableness_Stop.txt -l /home/MyDocuments/LSAspaces -
t
```

***Options and arguments:***

*-i* followed by the input file name. Requires either a filename with a complete path, or just a name of a file in the default location: "*input/*" subfolder. If this option is omitted, the default input file will be processed, which is "*input/default.txt*". The input file has to be properly formatted, see section "Input requirements" below for instructions.

*-s* followed by the stop words list file. Requires either a filename with a complete path, or just a name of the file in the default location: "*CleaningFiles/*" subfolder. If this option is omitted, the default stop words list file will be used: "*CleaningFiles/stopList_Words.txt*".

*-l* followed by the name of the directory that contains all of the LSA spaces to be applied to the input file. If not specified, the default folder is "*LSAspaces/*".

*-t* no argument is required. If "*-t*" is included, the words will be stemmed after the docs are cleaned. They will not be stemmed if "*-t*" is omitted. The default is to omit stemming.

***Input requirements:***

- Input file: All documents to be analyzed must be compiled into a single plain text file, in which each document is on a separate line; and each line (document) must begin with the document code, which will be used as the document identifier in all output files. Preferred format for the input file is UTF-8. Default input file location is "*input/*" subfolder.
- Stop words list file: Plain text, preferably in UTF-8 format. Each stop word must appear on a separate line.
- The folder containing the stop word list file must also contain the punctuation stop list: "*stopList_Punctuation.txt*".

***Output:***

Two CSV files for each semantic space are created in the "*csv/*" sub-folder. One of the *.csv* files contains a document-by-document similarity matrix populated with cosine values above the upper diagonal from the given semantic space; the second *.csv* file (with the *_COLUMN* suffix) formats the same information into a single column. Files are named with the following convention:

<input-file name>_<StopList file name>_LSA_<name of the LSA space>.csv

<input-file name>_<StopList file name>_LSA_<name of the LSA space>_COLUMN.csv

***Execution requirements:***

- *nltk.stem.porter*
- *semmod.lsa*
- *numpy*
- The folder containing the *doc_by_doc_Multi_LSA_Custom_STOPlist.py* must also contain the following files:
  - *corpusCleaningTools.py*
  - *EntityClassify.py*
  - *CustomCorpusPreparation.py (*contains the stemming function*)*
- The folder containing the stop word list file must also contain the punctuation stop list: "*stopList_Punctuation.txt*".

# 3 Concluding remarks

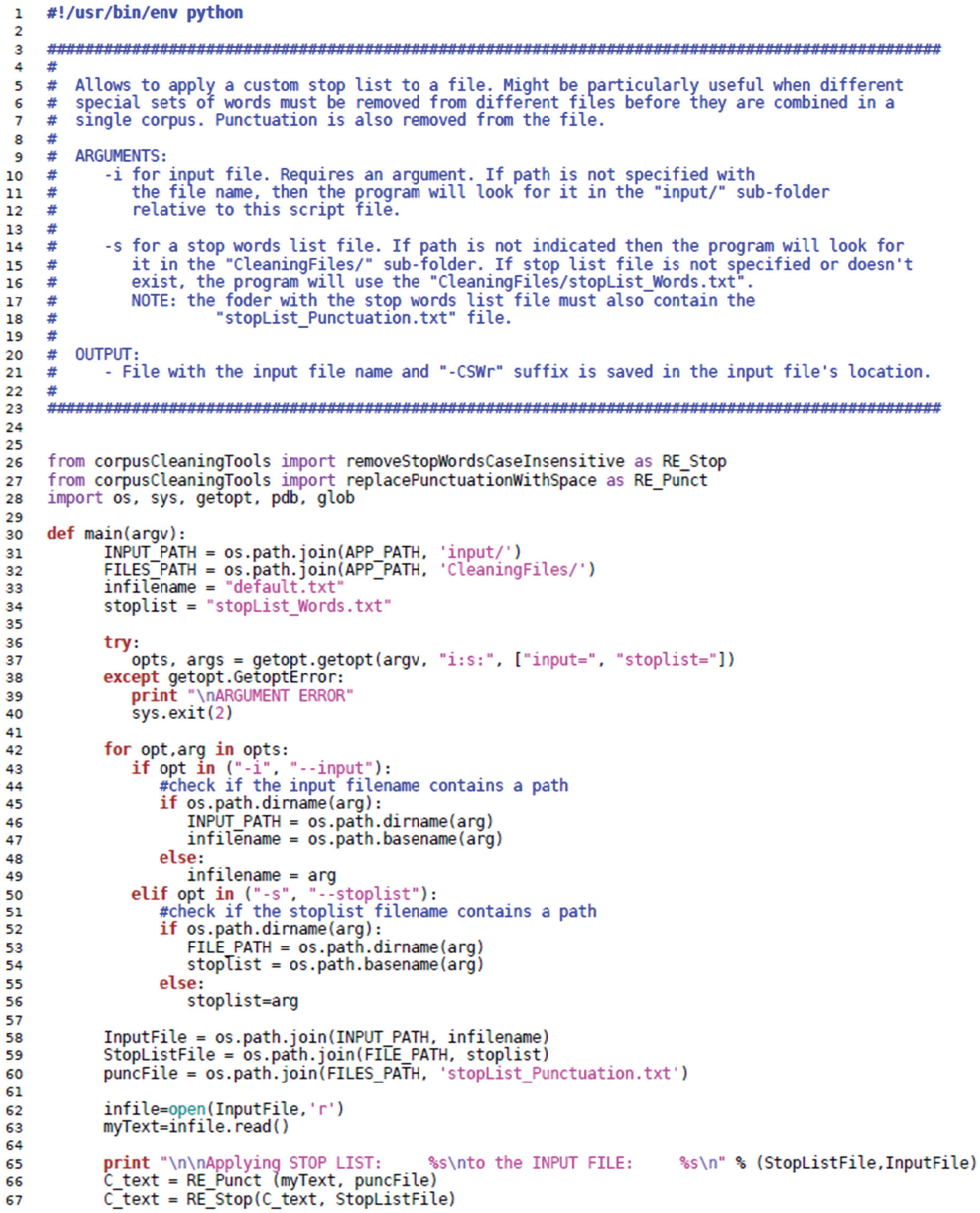
The tools described in this TN automate certain aspects of the otherwise labour-intensive and time-consuming process of pre-processing text for LSA and conducting multiple analyses. Although these tools were developed to address specific data analysis needs, the functionality that they support (e.g., document-by-document comparison) is fairly general. For example, the document-by-document comparison could be used in validating the semantic analysis component of the Analysis of Semantic and Social Networks (ASSN) tool.

The tools described in this TN will likely be developed further, given the nature of the ongoing work at DRDC Toronto. The purpose of this TN is to document the functionality developed up to date, to disseminate the availability of such functionality among DRDC Toronto colleagues who might benefit from them, and to reduce duplication of efforts in the future.

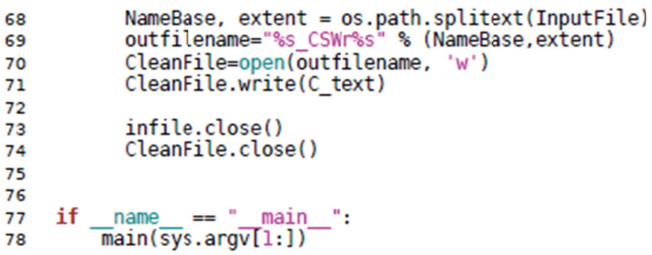
We expect to improve the flexibility and usability of these tools in the future, and to develop other functionality to support LSA and semantic analysis in general. DRDC Toronto can take the lead in developing a more comprehensive Python-based LSA toolkit, and make it available as an open source library to the general community of users interested in application of LSA. Such sharing could facilitate further co-development of the toolkit by the community.

This page intentionally left blank.

# Annex A *CustomStopW.py* Source Code

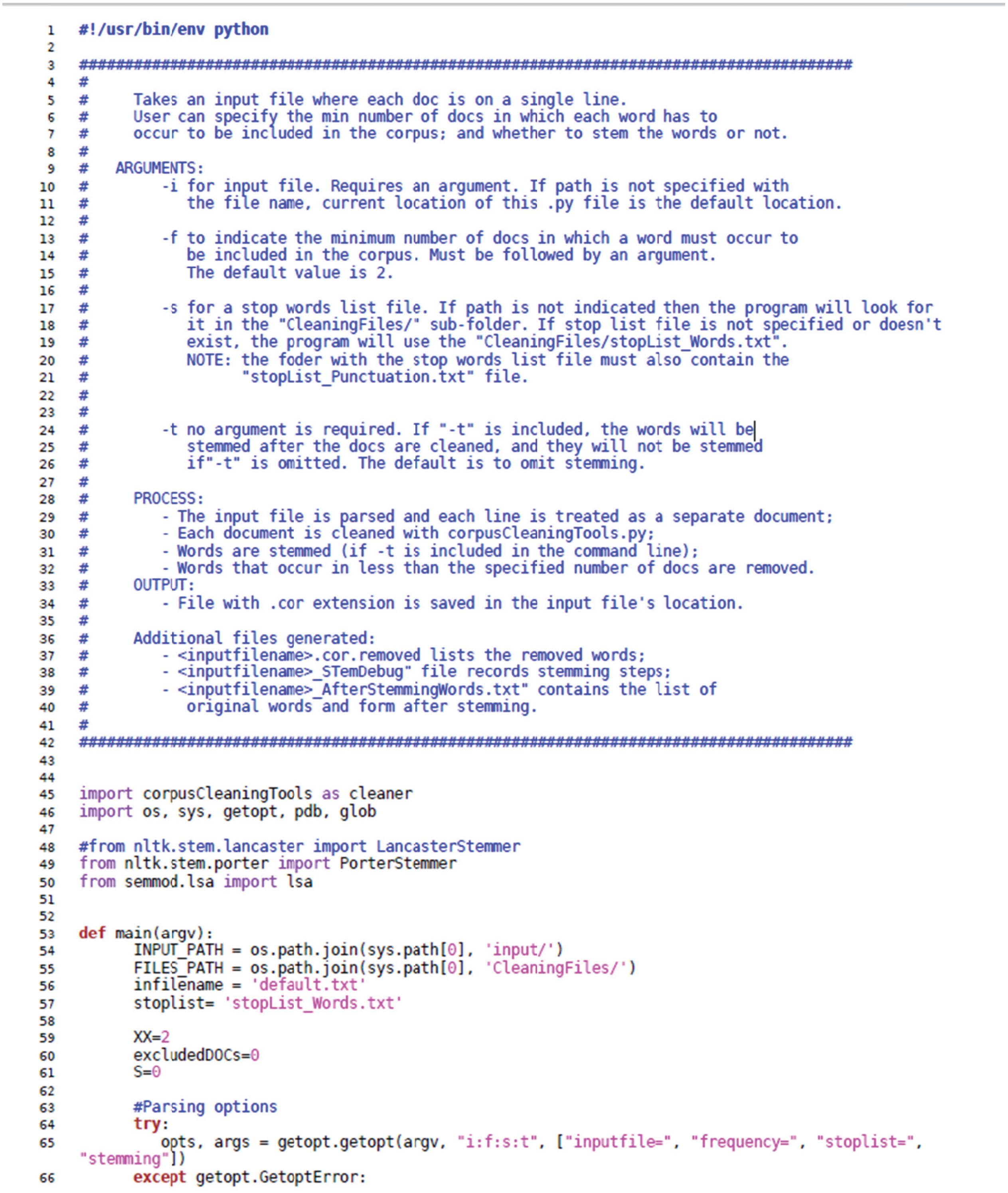


```
#!/usr/bin/env python

################################################################################################
#
#  Allows to apply a custom stop list to a file. Might be particularly useful when different
#  special sets of words must be removed from different files before they are combined in a
#  single corpus. Punctuation is also removed from the file.
#
#  ARGUMENTS:
#     -i for input file. Requires an argument. If path is not specified with
#        the file name, then the program will look for it in the "input/" sub-folder
#        relative to this script file.
#
#     -s for a stop words list file. If path is not indicated then the program will look for
#        it in the "CleaningFiles/" sub-folder. If stop list file is not specified or doesn't
#        exist, the program will use the "CleaningFiles/stopList_Words.txt".
#        NOTE: the foder with the stop words list file must also contain the
#                 "stopList_Punctuation.txt" file.
#
#  OUTPUT:
#     - File with the input file name and "-CSWr" suffix is saved in the input file's location.
#
################################################################################################


from corpusCleaningTools import removeStopWordsCaseInsensitive as RE_Stop
from corpusCleaningTools import replacePunctuationWithSpace as RE_Punct
import os, sys, getopt, pdb, glob

def main(argv):
      INPUT_PATH = os.path.join(APP_PATH, 'input/')
      FILES_PATH = os.path.join(APP_PATH, 'CleaningFiles/')
      infilename = "default.txt"
      stoplist = "stopList_Words.txt"

      try:
         opts, args = getopt.getopt(argv, "i:s:", ["input=", "stoplist="])
      except getopt.GetoptError:
         print "\nARGUMENT ERROR"
         sys.exit(2)

      for opt,arg in opts:
         if opt in ("-i", "--input"):
            #check if the input filename contains a path
            if os.path.dirname(arg):
               INPUT_PATH = os.path.dirname(arg)
               infilename = os.path.basename(arg)
            else:
               infilename = arg
         elif opt in ("-s", "--stoplist"):
            #check if the stoplist filename contains a path
            if os.path.dirname(arg):
               FILE_PATH = os.path.dirname(arg)
               stoplist = os.path.basename(arg)
            else:
               stoplist=arg

      InputFile = os.path.join(INPUT_PATH, infilename)
      StopListFile = os.path.join(FILE_PATH, stoplist)
      puncFile = os.path.join(FILES_PATH, 'stopList_Punctuation.txt')

      infile=open(InputFile,'r')
      myText=infile.read()

      print "\n\nApplying STOP LIST:     %s\nto the INPUT FILE:     %s\n" % (StopListFile,InputFile)
      C_text = RE_Punct (myText, puncFile)
      C_text = RE_Stop(C_text, StopListFile)
```

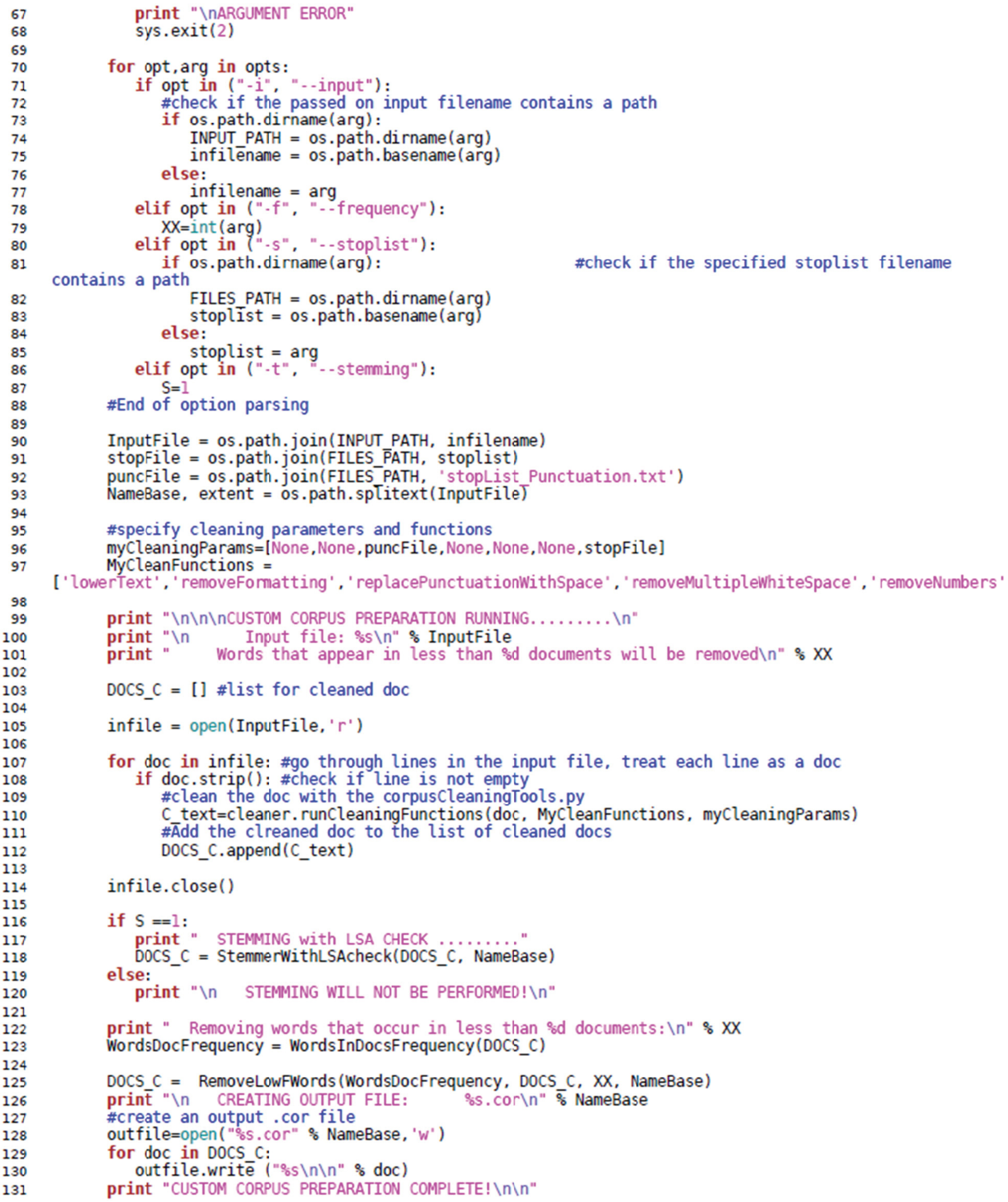


```
        NameBase, extent = os.path.splitext(InputFile)
        outfilename="%s_CSWr%s" % (NameBase,extent)
        CleanFile=open(outfilename, 'w')
        CleanFile.write(C_text)

        infile.close()
        CleanFile.close()


if __name__ == "__main__":
    main(sys.argv[1:])
```

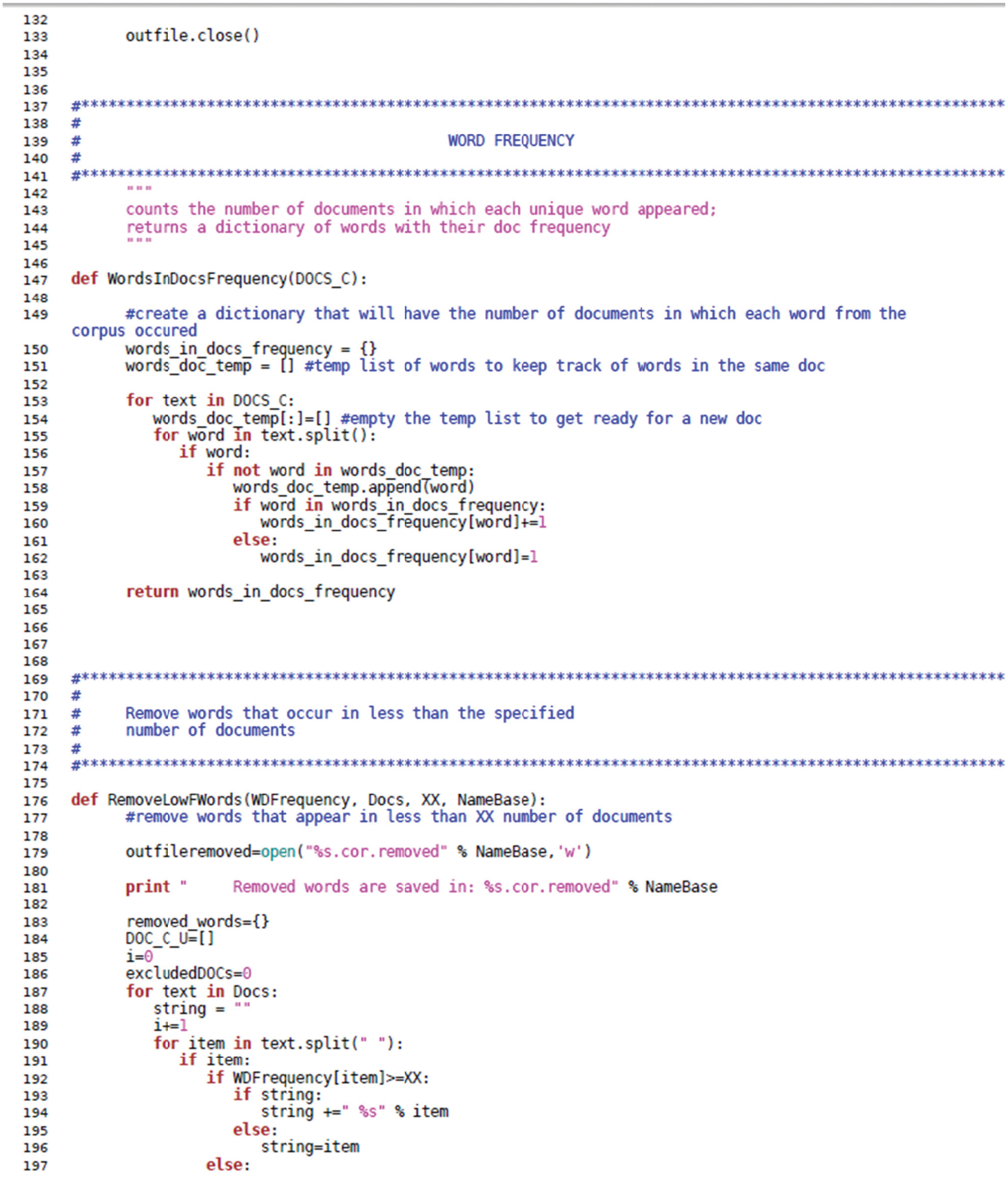
# Annex B *CustomCorpusPreparation.py* Sourse Code



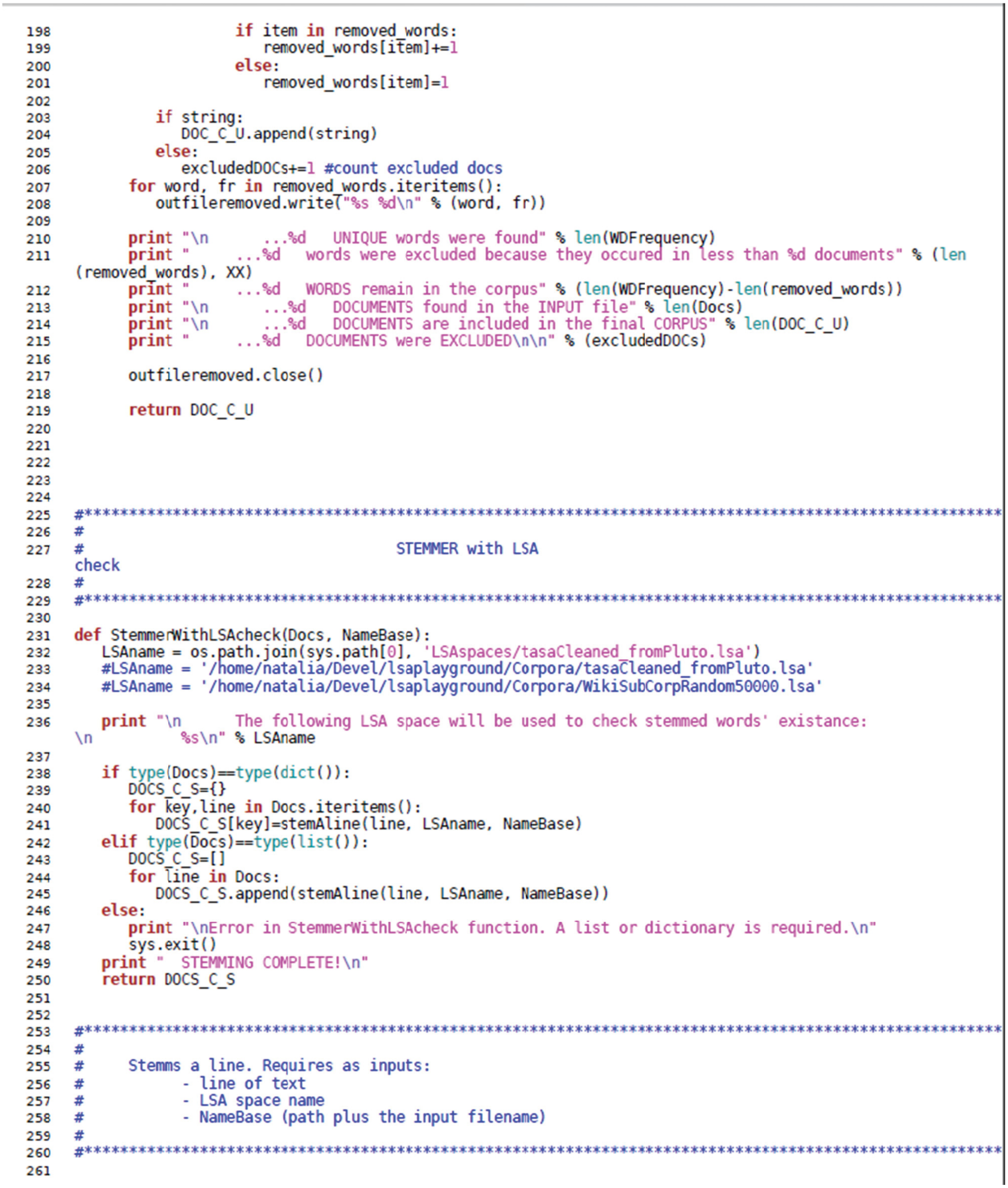
```
#!/usr/bin/env python

#####################################################################################
#
#     Takes an input file where each doc is on a single line.
#     User can specify the min number of docs in which each word has to
#     occur to be included in the corpus; and whether to stem the words or not.
#
#   ARGUMENTS:
#        -i for input file. Requires an argument. If path is not specified with
#           the file name, current location of this .py file is the default location.
#
#        -f to indicate the minimum number of docs in which a word must occur to
#           be included in the corpus. Must be followed by an argument.
#           The default value is 2.
#
#        -s for a stop words list file. If path is not indicated then the program will look for
#           it in the "CleaningFiles/" sub-folder. If stop list file is not specified or doesn't
#           exist, the program will use the "CleaningFiles/stopList_Words.txt".
#           NOTE: the foder with the stop words list file must also contain the
#                 "stopList_Punctuation.txt" file.
#
#
#        -t no argument is required. If "-t" is included, the words will be
#           stemmed after the docs are cleaned, and they will not be stemmed
#           if"-t" is omitted. The default is to omit stemming.
#
#     PROCESS:
#        - The input file is parsed and each line is treated as a separate document;
#        - Each document is cleaned with corpusCleaningTools.py;
#        - Words are stemmed (if -t is included in the command line);
#        - Words that occur in less than the specified number of docs are removed.
#     OUTPUT:
#        - File with .cor extension is saved in the input file's location.
#
#     Additional files generated:
#        - <inputfilename>.cor.removed lists the removed words;
#        - <inputfilename>_STemDebug" file records stemming steps;
#        - <inputfilename>_AfterStemmingWords.txt" contains the list of
#           original words and form after stemming.
#
#####################################################################################


import corpusCleaningTools as cleaner
import os, sys, getopt, pdb, glob

#from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.porter import PorterStemmer
from semmod.lsa import lsa


def main(argv):
      INPUT_PATH = os.path.join(sys.path[0], 'input/')
      FILES_PATH = os.path.join(sys.path[0], 'CleaningFiles/')
      infilename = 'default.txt'
      stoplist= 'stopList_Words.txt'

      XX=2
      excludedDOCs=0
      S=0

      #Parsing options
      try:
         opts, args = getopt.getopt(argv, "i:f:s:t", ["inputfile=", "frequency=", "stoplist=", "stemming"])
      except getopt.GetoptError:
```



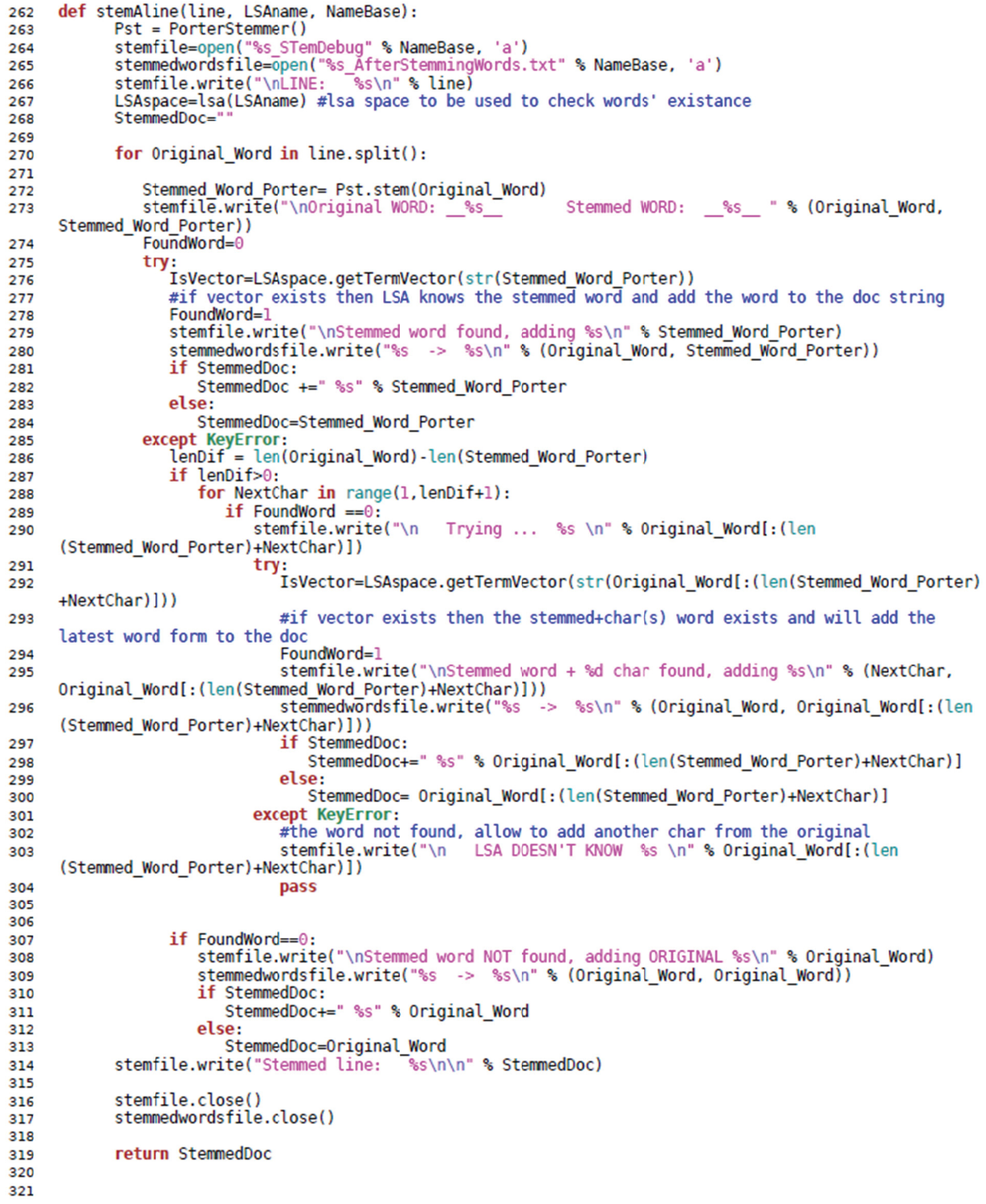
```
         print "\nARGUMENT ERROR"
         sys.exit(2)

      for opt,arg in opts:
         if opt in ("-i", "--input"):
            #check if the passed on input filename contains a path
            if os.path.dirname(arg):
               INPUT_PATH = os.path.dirname(arg)
               infilename = os.path.basename(arg)
            else:
               infilename = arg
         elif opt in ("-f", "--frequency"):
            XX=int(arg)
         elif opt in ("-s", "--stoplist"):
            if os.path.dirname(arg):                    #check if the specified stoplist filename
contains a path
               FILES_PATH = os.path.dirname(arg)
               stoplist = os.path.basename(arg)
            else:
               stoplist = arg
         elif opt in ("-t", "--stemming"):
            S=1
      #End of option parsing

      InputFile = os.path.join(INPUT_PATH, infilename)
      stopFile = os.path.join(FILES_PATH, stoplist)
      puncFile = os.path.join(FILES_PATH, 'stopList_Punctuation.txt')
      NameBase, extent = os.path.splitext(InputFile)

      #specify cleaning parameters and functions
      myCleaningParams=[None,None,puncFile,None,None,None,stopFile]
      MyCleanFunctions =
['lowerText','removeFormatting','replacePunctuationWithSpace','removeMultipleWhiteSpace','removeNumbers'

      print "\n\n\nCUSTOM CORPUS PREPARATION RUNNING.........\n"
      print "\n      Input file: %s\n" % InputFile
      print "     Words that appear in less than %d documents will be removed\n" % XX

      DOCS_C = [] #list for cleaned doc

      infile = open(InputFile,'r')

      for doc in infile: #go through lines in the input file, treat each line as a doc
         if doc.strip(): #check if line is not empty
            #clean the doc with the corpusCleaningTools.py
            C_text=cleaner.runCleaningFunctions(doc, MyCleanFunctions, myCleaningParams)
            #Add the clreaned doc to the list of cleaned docs
            DOCS_C.append(C_text)

      infile.close()

      if S ==1:
         print "  STEMMING with LSA CHECK ........."
         DOCS_C = StemmerWithLSAcheck(DOCS_C, NameBase)
      else:
         print "\n   STEMMING WILL NOT BE PERFORMED!\n"

      print "  Removing words that occur in less than %d documents:\n" % XX
      WordsDocFrequency = WordsInDocsFrequency(DOCS_C)

      DOCS_C =  RemoveLowFWords(WordsDocFrequency, DOCS_C, XX, NameBase)
      print "\n   CREATING OUTPUT FILE:      %s.cor\n" % NameBase
      #create an output .cor file
      outfile=open("%s.cor" % NameBase,'w')
      for doc in DOCS_C:
         outfile.write ("%s\n\n" % doc)
      print "CUSTOM CORPUS PREPARATION COMPLETE!\n\n"
```



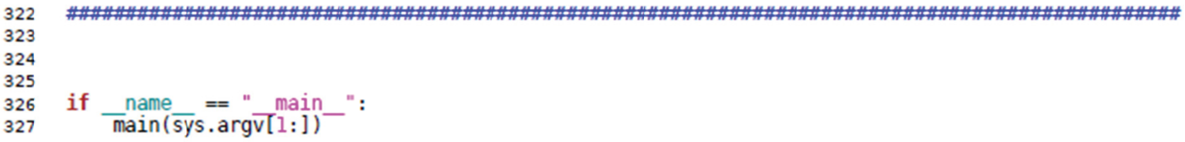
```

      outfile.close()



#*****************************************************************************************************
#
#                                          WORD FREQUENCY
#
#*****************************************************************************************************
      """
      counts the number of documents in which each unique word appeared;
      returns a dictionary of words with their doc frequency
      """

def WordsInDocsFrequency(DOCS_C):

      #create a dictionary that will have the number of documents in which each word from the corpus occured
      words_in_docs_frequency = {}
      words_doc_temp = [] #temp list of words to keep track of words in the same doc

      for text in DOCS_C:
         words_doc_temp[:]=[] #empty the temp list to get ready for a new doc
         for word in text.split():
            if word:
               if not word in words_doc_temp:
                  words_doc_temp.append(word)
                  if word in words_in_docs_frequency:
                     words_in_docs_frequency[word]+=1
                  else:
                     words_in_docs_frequency[word]=1

      return words_in_docs_frequency




#*****************************************************************************************************
#
#     Remove words that occur in less than the specified
#     number of documents
#
#*****************************************************************************************************

def RemoveLowFWords(WDFrequency, Docs, XX, NameBase):
      #remove words that appear in less than XX number of documents

      outfileremoved=open("%s.cor.removed" % NameBase,'w')

      print "     Removed words are saved in: %s.cor.removed" % NameBase

      removed_words={}
      DOC_C_U=[]
      i=0
      excludedDOCs=0
      for text in Docs:
         string = ""
         i+=1
         for item in text.split(" "):
            if item:
               if WDFrequency[item]>=XX:
                  if string:
                     string +=" %s" % item
                  else:
                     string=item
               else:
```



```
                  if item in removed_words:
                     removed_words[item]+=1
                  else:
                     removed_words[item]=1

         if string:
            DOC_C_U.append(string)
         else:
            excludedDOCs+=1 #count excluded docs
      for word, fr in removed_words.iteritems():
         outfileremoved.write("%s %d\n" % (word, fr))

      print "\n      ...%d   UNIQUE words were found" % len(WDFrequency)
      print "     ...%d   words were excluded because they occured in less than %d documents" % (len(removed_words), XX)
      print "     ...%d   WORDS remain in the corpus" % (len(WDFrequency)-len(removed_words))
      print "\n      ...%d   DOCUMENTS found in the INPUT file" % len(Docs)
      print "\n      ...%d   DOCUMENTS are included in the final CORPUS" % len(DOC_C_U)
      print "     ...%d   DOCUMENTS were EXCLUDED\n\n" % (excludedDOCs)

      outfileremoved.close()

      return DOC_C_U





#*********************************************************************************************************
#
#                                    STEMMER with LSA check
#
#*********************************************************************************************************

def StemmerWithLSAcheck(Docs, NameBase):
   LSAname = os.path.join(sys.path[0], 'LSAspaces/tasaCleaned_fromPluto.lsa')
   #LSAname = '/home/natalia/Devel/lsaplayground/Corpora/tasaCleaned_fromPluto.lsa'
   #LSAname = '/home/natalia/Devel/lsaplayground/Corpora/WikiSubCorpRandom50000.lsa'

   print "\n      The following LSA space will be used to check stemmed words' existance:\n         %s\n" % LSAname

   if type(Docs)==type(dict()):
      DOCS_C_S={}
      for key,line in Docs.iteritems():
         DOCS_C_S[key]=stemAline(line, LSAname, NameBase)
   elif type(Docs)==type(list()):
      DOCS_C_S=[]
      for line in Docs:
         DOCS_C_S.append(stemAline(line, LSAname, NameBase))
   else:
      print "\nError in StemmerWithLSAcheck function. A list or dictionary is required.\n"
      sys.exit()
   print "  STEMMING COMPLETE!\n"
   return DOCS_C_S


#*********************************************************************************************************
#
#     Stemms a line. Requires as inputs:
#           - line of text
#           - LSA space name
#           - NameBase (path plus the input filename)
#
#*********************************************************************************************************

```

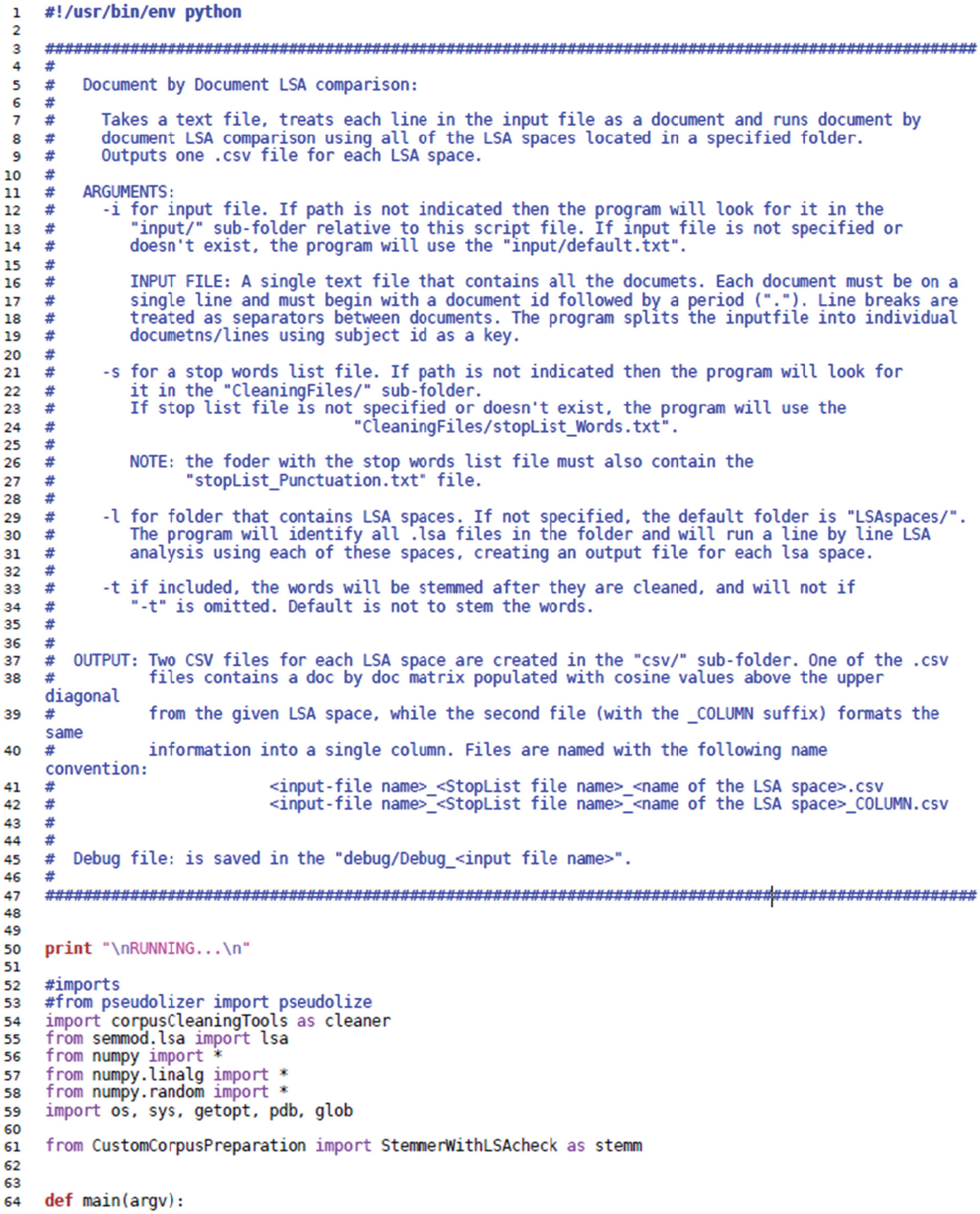


```
def stemAline(line, LSAname, NameBase):
      Pst = PorterStemmer()
      stemfile=open("%s_STemDebug" % NameBase, 'a')
      stemmedwordsfile=open("%s_AfterStemmingWords.txt" % NameBase, 'a')
      stemfile.write("\nLINE:   %s\n" % line)
      LSAspace=lsa(LSAname) #lsa space to be used to check words' existance
      StemmedDoc=""

      for Original_Word in line.split():

         Stemmed_Word_Porter= Pst.stem(Original_Word)
         stemfile.write("\nOriginal WORD: __%s__      Stemmed WORD:  __%s__ " % (Original_Word, Stemmed_Word_Porter))
         FoundWord=0
         try:
            IsVector=LSAspace.getTermVector(str(Stemmed_Word_Porter))
            #if vector exists then LSA knows the stemmed word and add the word to the doc string
            FoundWord=1
            stemfile.write("\nStemmed word found, adding %s\n" % Stemmed_Word_Porter)
            stemmedwordsfile.write("%s  ->  %s\n" % (Original_Word, Stemmed_Word_Porter))
            if StemmedDoc:
               StemmedDoc +=" %s" % Stemmed_Word_Porter
            else:
               StemmedDoc=Stemmed_Word_Porter
         except KeyError:
            lenDif = len(Original_Word)-len(Stemmed_Word_Porter)
            if lenDif>0:
               for NextChar in range(1,lenDif+1):
                  if FoundWord ==0:
                     stemfile.write("\n   Trying ...  %s \n" % Original_Word[:(len(Stemmed_Word_Porter)+NextChar)])
                     try:
                        IsVector=LSAspace.getTermVector(str(Original_Word[:(len(Stemmed_Word_Porter)+NextChar)]))
                        #if vector exists then the stemmed+char(s) word exists and will add the latest word form to the doc
                        FoundWord=1
                        stemfile.write("\nStemmed word + %d char found, adding %s\n" % (NextChar, Original_Word[:(len(Stemmed_Word_Porter)+NextChar)]))
                        stemmedwordsfile.write("%s  ->  %s\n" % (Original_Word, Original_Word[:(len(Stemmed_Word_Porter)+NextChar)]))
                        if StemmedDoc:
                           StemmedDoc+=" %s" % Original_Word[:(len(Stemmed_Word_Porter)+NextChar)]
                        else:
                           StemmedDoc= Original_Word[:(len(Stemmed_Word_Porter)+NextChar)]
                     except KeyError:
                        #the word not found, allow to add another char from the original
                        stemfile.write("\n   LSA DOESN'T KNOW  %s \n" % Original_Word[:(len(Stemmed_Word_Porter)+NextChar)])
                        pass


            if FoundWord==0:
               stemfile.write("\nStemmed word NOT found, adding ORIGINAL %s\n" % Original_Word)
               stemmedwordsfile.write("%s  ->  %s\n" % (Original_Word, Original_Word))
               if StemmedDoc:
                  StemmedDoc+=" %s" % Original_Word
               else:
                  StemmedDoc=Original_Word
      stemfile.write("Stemmed line:   %s\n\n" % StemmedDoc)

      stemfile.close()
      stemmedwordsfile.close()

      return StemmedDoc


```

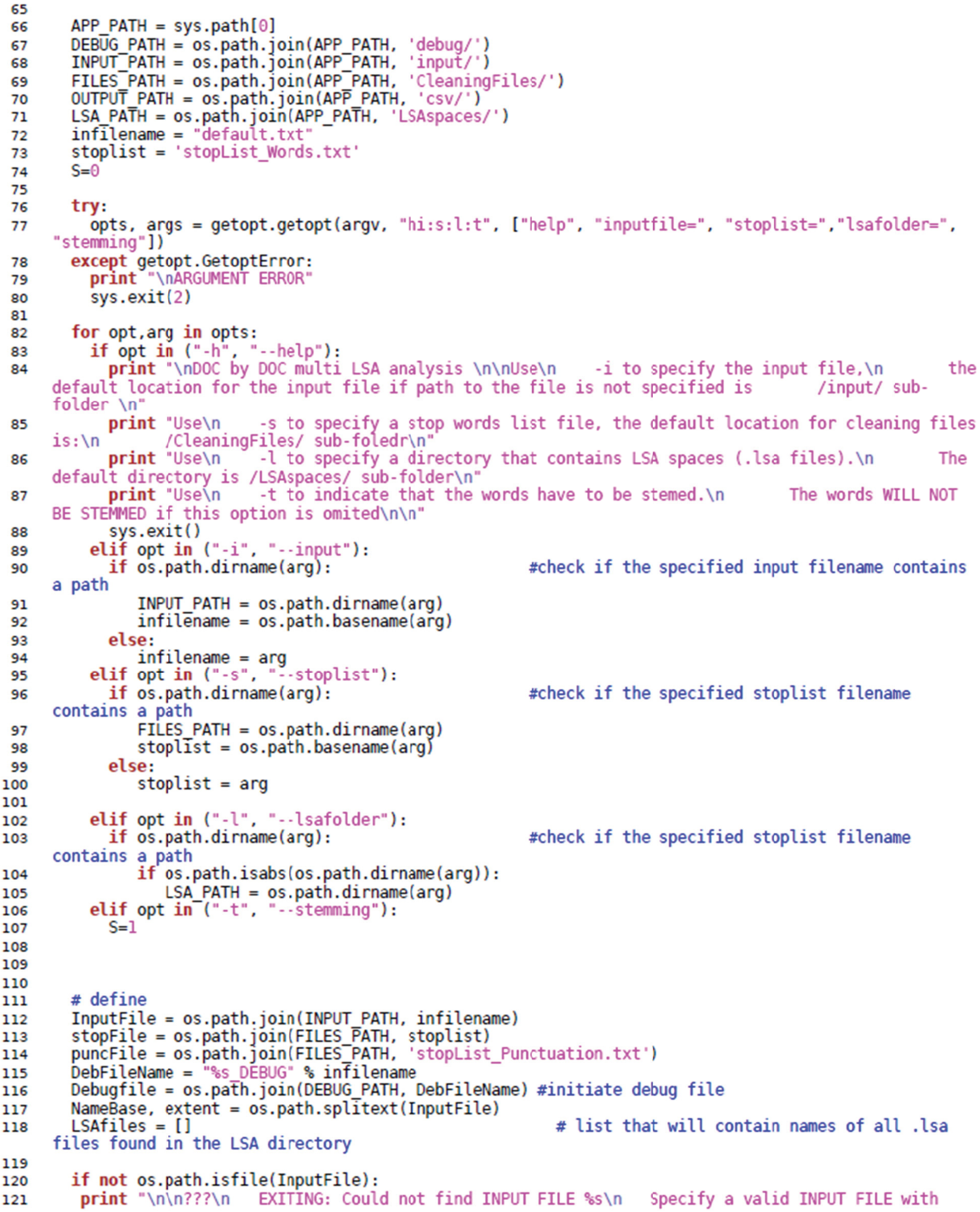


```
######################################################################################################



if __name__ == "__main__":
    main(sys.argv[1:])
```

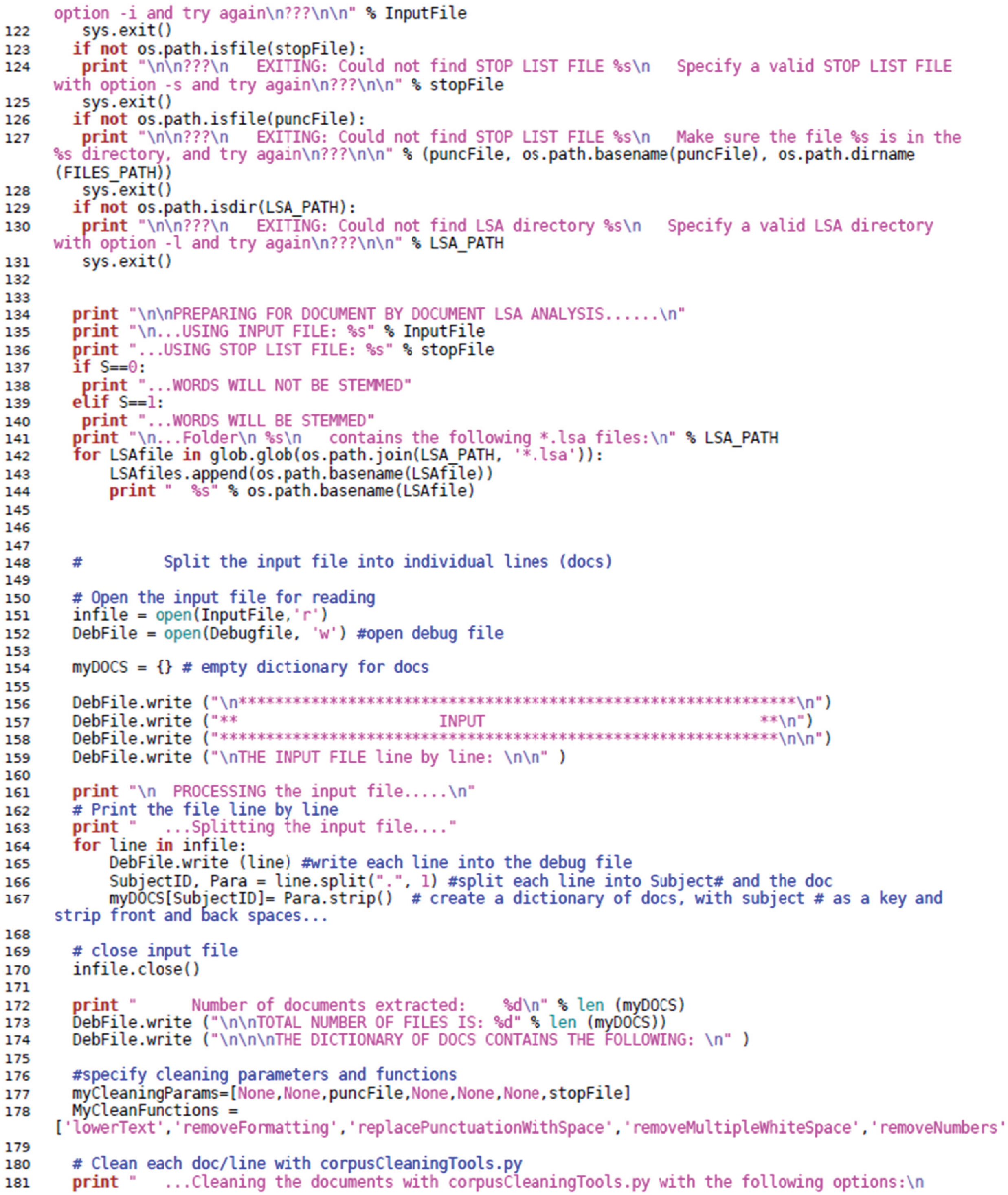
# Annex C *doc_by_doc_Multi_LSA_Custom_STOPlist.py* ***Sourse Code***



```
#!/usr/bin/env python

####################################################################################################
#
#   Document by Document LSA comparison:
#
#     Takes a text file, treats each line in the input file as a document and runs document by
#     document LSA comparison using all of the LSA spaces located in a specified folder.
#     Outputs one .csv file for each LSA space.
#
#   ARGUMENTS:
#     -i for input file. If path is not indicated then the program will look for it in the
#        "input/" sub-folder relative to this script file. If input file is not specified or
#        doesn't exist, the program will use the "input/default.txt".
#
#        INPUT FILE: A single text file that contains all the documets. Each document must be on a
#        single line and must begin with a document id followed by a period ("."). Line breaks are
#        treated as separators between documents. The program splits the inputfile into individual
#        documetns/lines using subject id as a key.
#
#     -s for a stop words list file. If path is not indicated then the program will look for
#        it in the "CleaningFiles/" sub-folder.
#        If stop list file is not specified or doesn't exist, the program will use the
#                                "CleaningFiles/stopList_Words.txt".
#
#        NOTE: the foder with the stop words list file must also contain the
#              "stopList_Punctuation.txt" file.
#
#     -l for folder that contains LSA spaces. If not specified, the default folder is "LSAspaces/".
#        The program will identify all .lsa files in the folder and will run a line by line LSA
#        analysis using each of these spaces, creating an output file for each lsa space.
#
#     -t if included, the words will be stemmed after they are cleaned, and will not if
#        "-t" is omitted. Default is not to stem the words.
#
#
#  OUTPUT: Two CSV files for each LSA space are created in the "csv/" sub-folder. One of the .csv
#          files contains a doc by doc matrix populated with cosine values above the upper
diagonal
#          from the given LSA space, while the second file (with the _COLUMN suffix) formats the
same
#          information into a single column. Files are named with the following name
convention:
#                        <input-file name>_<StopList file name>_<name of the LSA space>.csv
#                        <input-file name>_<StopList file name>_<name of the LSA space>_COLUMN.csv
#
#
#  Debug file: is saved in the "debug/Debug_<input file name>".
#
####################################################################################################


print "\nRUNNING...\n"

#imports
#from pseudolizer import pseudolize
import corpusCleaningTools as cleaner
from semmod.lsa import lsa
from numpy import *
from numpy.linalg import *
from numpy.random import *
import os, sys, getopt, pdb, glob

from CustomCorpusPreparation import StemmerWithLSAcheck as stemm


def main(argv):
```

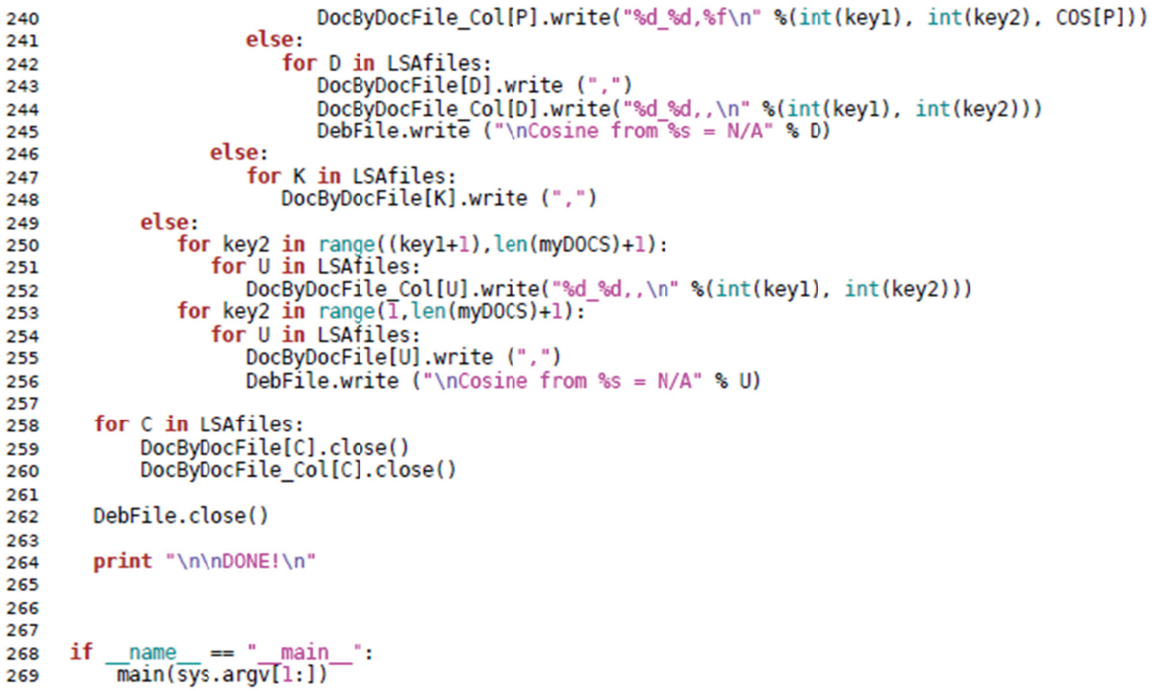


```
  APP_PATH = sys.path[0]
  DEBUG_PATH = os.path.join(APP_PATH, 'debug/')
  INPUT_PATH = os.path.join(APP_PATH, 'input/')
  FILES_PATH = os.path.join(APP_PATH, 'CleaningFiles/')
  OUTPUT_PATH = os.path.join(APP_PATH, 'csv/')
  LSA_PATH = os.path.join(APP_PATH, 'LSAspaces/')
  infilename = "default.txt"
  stoplist = 'stopList_Words.txt'
  S=0

  try:
    opts, args = getopt.getopt(argv, "hi:s:l:t", ["help", "inputfile=", "stoplist=","lsafolder=",
"stemming"])
  except getopt.GetoptError:
    print "\nARGUMENT ERROR"
    sys.exit(2)

  for opt,arg in opts:
    if opt in ("-h", "--help"):
      print "\nDOC by DOC multi LSA analysis \n\nUse\n    -i to specify the input file,\n       the
default location for the input file if path to the file is not specified is       /input/ sub-
folder \n"
      print "Use\n    -s to specify a stop words list file, the default location for cleaning files
is:\n       /CleaningFiles/ sub-foledr\n"
      print "Use\n    -l to specify a directory that contains LSA spaces (.lsa files).\n       The
default directory is /LSAspaces/ sub-folder\n"
      print "Use\n    -t to indicate that the words have to be stemed.\n       The words WILL NOT
BE STEMMED if this option is omited\n\n"
      sys.exit()
    elif opt in ("-i", "--input"):
      if os.path.dirname(arg):                     #check if the specified input filename contains
a path
         INPUT_PATH = os.path.dirname(arg)
         infilename = os.path.basename(arg)
      else:
         infilename = arg
    elif opt in ("-s", "--stoplist"):
      if os.path.dirname(arg):                     #check if the specified stoplist filename
contains a path
         FILES_PATH = os.path.dirname(arg)
         stoplist = os.path.basename(arg)
      else:
         stoplist = arg

    elif opt in ("-l", "--lsafolder"):
      if os.path.dirname(arg):                     #check if the specified stoplist filename
contains a path
         if os.path.isabs(os.path.dirname(arg)):
            LSA_PATH = os.path.dirname(arg)
    elif opt in ("-t", "--stemming"):
      S=1



  # define
  InputFile = os.path.join(INPUT_PATH, infilename)
  stopFile = os.path.join(FILES_PATH, stoplist)
  puncFile = os.path.join(FILES_PATH, 'stopList_Punctuation.txt')
  DebFileName = "%s_DEBUG" % infilename
  Debugfile = os.path.join(DEBUG_PATH, DebFileName) #initiate debug file
  NameBase, extent = os.path.splitext(InputFile)
  LSAfiles = []                                    # list that will contain names of all .lsa
files found in the LSA directory

  if not os.path.isfile(InputFile):
   print "\n\n???\n   EXITING: Could not find INPUT FILE %s\n   Specify a valid INPUT FILE with
```



```
option -i and try again\n???\n\n" % InputFile
   sys.exit()
  if not os.path.isfile(stopFile):
   print "\n\n???\n   EXITING: Could not find STOP LIST FILE %s\n   Specify a valid STOP LIST FILE
with option -s and try again\n???\n\n" % stopFile
   sys.exit()
  if not os.path.isfile(puncFile):
   print "\n\n???\n   EXITING: Could not find STOP LIST FILE %s\n   Make sure the file %s is in the
%s directory, and try again\n???\n\n" % (puncFile, os.path.basename(puncFile), os.path.dirname
(FILES_PATH))
   sys.exit()
  if not os.path.isdir(LSA_PATH):
   print "\n\n???\n   EXITING: Could not find LSA directory %s\n   Specify a valid LSA directory
with option -l and try again\n???\n\n" % LSA_PATH
   sys.exit()


  print "\n\nPREPARING FOR DOCUMENT BY DOCUMENT LSA ANALYSIS......\n"
  print "\n...USING INPUT FILE: %s" % InputFile
  print "...USING STOP LIST FILE: %s" % stopFile
  if S==0:
   print "...WORDS WILL NOT BE STEMMED"
  elif S==1:
   print "...WORDS WILL BE STEMMED"
  print "\n...Folder\n %s\n   contains the following *.lsa files:\n" % LSA_PATH
  for LSAfile in glob.glob(os.path.join(LSA_PATH, '*.lsa')):
      LSAfiles.append(os.path.basename(LSAfile))
      print "  %s" % os.path.basename(LSAfile)



  #         Split the input file into individual lines (docs)

  # Open the input file for reading
  infile = open(InputFile,'r')
  DebFile = open(Debugfile, 'w') #open debug file

  myDOCS = {} # empty dictionary for docs

  DebFile.write ("\n*************************************************************\n")
  DebFile.write ("**                      INPUT                              **\n")
  DebFile.write ("*************************************************************\n\n")
  DebFile.write ("\nTHE INPUT FILE line by line: \n\n" )

  print "\n  PROCESSING the input file.....\n"
  # Print the file line by line
  print "   ...Splitting the input file...."
  for line in infile:
      DebFile.write (line) #write each line into the debug file
      SubjectID, Para = line.split(".", 1) #split each line into Subject# and the doc
      myDOCS[SubjectID]= Para.strip()  # create a dictionary of docs, with subject # as a key and
strip front and back spaces...

  # close input file
  infile.close()

  print "      Number of documents extracted:    %d\n" % len (myDOCS)
  DebFile.write ("\n\nTOTAL NUMBER OF FILES IS: %d" % len (myDOCS))
  DebFile.write ("\n\n\nTHE DICTIONARY OF DOCS CONTAINS THE FOLLOWING: \n" )

  #specify cleaning parameters and functions
  myCleaningParams=[None,None,puncFile,None,None,None,stopFile]
  MyCleanFunctions =
['lowerText','removeFormatting','replacePunctuationWithSpace','removeMultipleWhiteSpace','removeNumbers'

  # Clean each doc/line with corpusCleaningTools.py
  print "   ...Cleaning the documents with corpusCleaningTools.py with the following options:\n
```

```
* lowerText\n     * removeFormatting\n     * replacePunctuationWithSpace\n     * removeMultipleWhiteSpace\n     * removeNumbers\n     * removeSingleCharacters\n     * removeStopWordsCaseInsensitive\n"
  for n, t in myDOCS.iteritems():
      DebFile.write ("\nSubject # %s:\nORIGINAL TEXT: %s" % (n,t))
      if t.strip(): #check if line is not empty
         myDOCS[n]= cleaner.runCleaningFunctions(t, MyCleanFunctions, myCleaningParams) #clean the doc with the corpusCleaningTools.py
         DebFile.write ("\nCLEANED TEXT: %s\n\n" % myDOCS[n])
  if S==1:  #stem the words with LSA check
   print "\n   ...STEMMING THE WORDS...\n"
   myDOCS = stemm(myDOCS,NameBase)
   for g,k in myDOCS.iteritems():
      DebFile.write ("\nSTEMMED TEXT #%d: %s\n\n" % (int(g),k))
  else:
   print "\n   !!STEMMING WILL NOT BE PERFORMED!\n"


  # The program generates one output file for each LSA spaces in the LSA folder.

  OutFileName = {}
  DocByDocFile = {}
  DocByDocFile_Col = {} #additional column file output
  LSAspaces = {}
  COS = {}

  print "   ...GENERATING OUTPUT FILES in the directory  %s:\n" % os.path.join(APP_PATH,OUTPUT_PATH)
  DebFile.write ("\n\n\n*****************************************************************")

  for space in LSAfiles:
      DocByDocFile[space]=open(os.path.join(APP_PATH,OUTPUT_PATH,'%s_%s_%s.csv' % (infilename,stoplist,space)),'w')
      DocByDocFile_Col[space]=open(os.path.join(APP_PATH,OUTPUT_PATH,'%s_%s_%s_COLUMN.csv' % (infilename,stoplist,space)),'w')
      DocByDocFile[space].write (",")                             #leaving the first cell of the first row empty
      LSAspaces[space]=lsa(os.path.join(LSA_PATH,space))
      print "  * FILE: %s_%s_%s.csv" % (infilename,stoplist,space)
      print "  * FILE: %s_%s_%s_COLUMN.csv" % (infilename,stoplist,space)
      DebFile.write ("* OUTPUT FILE: %s\n" % os.path.join(APP_PATH,OUTPUT_PATH,'%s_%s_%s.csv' % (infilename,stoplist,space)))

  DebFile.write ("\n*****************************************************************")
  DebFile.write ("\n*                    LSA ANALYSIS                              *")
  DebFile.write ("\n*              USING SIMILARITY method                         *")
  DebFile.write ("\n*****************************************************************\n\n")

  for counter in range(1, len(myDOCS)+1): #populating line 1 of the output file with DOC numbers
      for F in LSAfiles:
         DocByDocFile[F].write ("%d," % counter)

  for key1 in range(1,len(myDOCS)+1): #populating row 1 of the output file with DOC numbers
      for F in LSAfiles:
         DocByDocFile[F].write ("\n%d" % key1)
      DOC1=myDOCS[str(key1)]
      if DOC1:
         for key2 in range(1, len(myDOCS)+1):
            DOC2=myDOCS[str(key2)]
            if key1<key2: #adding for diagonal
               DebFile.write ("\n\nSubject #%d and Subject #%d\n" % (key1, key2))
               DebFile.write ("\nSubject #%d para = %s \n\nSubject #%d para = %s \n" %(key1, DOC1, key2, DOC2))
               if DOC2:
                  for P in LSAfiles:
                     COS[P] = LSAspaces[P].Similarity(DOC1, DOC2)
                     DocByDocFile[P].write (",%f" % COS[P])
                     DebFile.write ("\nCosine from %s = %f" % (P, COS[P]))
```



```
                     DocByDocFile_Col[P].write("%d_%d,%f\n" %(int(key1), int(key2), COS[P]))
               else:
                  for D in LSAfiles:
                     DocByDocFile[D].write (",")
                     DocByDocFile_Col[D].write("%d_%d,,\n" %(int(key1), int(key2)))
                     DebFile.write ("\nCosine from %s = N/A" % D)
            else:
               for K in LSAfiles:
                  DocByDocFile[K].write (",")
      else:
         for key2 in range((key1+1),len(myDOCS)+1):
            for U in LSAfiles:
               DocByDocFile_Col[U].write("%d_%d,,\n" %(int(key1), int(key2)))
         for key2 in range(1,len(myDOCS)+1):
            for U in LSAfiles:
               DocByDocFile[U].write (",")
               DebFile.write ("\nCosine from %s = N/A" % U)

  for C in LSAfiles:
      DocByDocFile[C].close()
      DocByDocFile_Col[C].close()

  DebFile.close()

  print "\n\nDONE!\n"


if __name__ == "__main__":
    main(sys.argv[1:])
```

# Bibliography

Dennis, S., & Stone, B. (2011). *SEMMOD: Semantic models package* (Version 1.5): Ohio State University.

Landauer, T. K., & Dumais, S. (1997). A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge. *Prychological Review, 104(2)*, 211-240.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes, 25(2-3)*, 259-284.

Stone, B., & Dennis, S. (2012). *Wikipedia Subcorpora Tool* (Version 1.0): Ohio State University.

# List of symbols/abbreviations/acronyms/initialisms

| | |
|---|---|
| ARP | Applied Research Project |
| ASSN | Analysis of Semantic and Social Networks |
| CSV | Comma Separated Values |
| DND | Department of National Defence |
| DRDC | Defence Research & Development Canada |
| DRDKIM | Director Research and Development Knowledge and Information Management |
| GUI | Graphical User Interface |
| LSA | Latent Semantic Analysis |
| NLTK | Natural Language Toolkit |
| R&D | Research & Development |
| OSU | Ohio State University |
| SEMMOD | Semantic Models |
| TIF | Technology Investment Fund |
| TN | Technical Note |
| UTF | Unicode Transformation Formats |

This page intentionally left blank.

## DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

|  |  |
|---|---|
| 1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)<br>Defence R&D Canada – Toronto<br>1133 Sheppard Avenue West<br>P.O. Box 2000<br>Toronto, Ontario<br>M3K 2C9 | 2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)<br>UNCLASSIFIED<br>(NON-CONTROLLED GOODS)<br>DMC A<br>REVIEW: GCEC JUNE 2010 |

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Latent Semantic Analysis (LSA) tools:

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Derbentseva, N.; Kwantes, P.; Terhaar, P.

|  |  |  |
|---|---|---|
| 5. DATE OF PUBLICATION (Month and year of publication of document.)<br>July 2012 | 6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)<br>35 | 6b. NO. OF REFS (Total cited in document.)<br>4 |

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Note

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Toronto1133 Sheppard Avenue WestP.O. Box 2000Toronto, Ontario M3K 2C9

|  |  |
|---|---|
| 9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)<br>15ah | 9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) |
| 10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)<br>DRDC Toronto TN 2012-079 | 10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.) |

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

NOT REQUIRED.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Latent Semantic Analysis, LSA, semantic models, semantic analysis, python