**Big data challenge**

**Group:T006**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **T006P1** | Shravan | Singh | [ssingh15@stevens.edu](mailto:ssingh15@stevens.edu) | 10397088 | T006 |
| **T006P2** | Feng | Chen | [fchen6@stevens.edu](mailto:fchen6@stevens.edu) | 10400586 | T006 |
| **T006P3** | Xiaotian | Zhang | [xzhang90@stevens.edu](mailto:xzhang90@stevens.edu) | 10402569 | T006 |

**Flow Chart：**

Input whole text

Split sentence

Word analysis

Split one sentence into different sense-groups

Give positive and negative score of one sentence

Positive score >negative score

No

Yes

Return negative

Return positive

End

**Two codes one for analysis sentence one by one another for analysis the whole 24kData.**

**Code and code explanation:**

**Code to analysis sentence one by one:**

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

import os

import sys

import numpy as np

from string import punctuation

##########################################################################################################

#

# Sentiment dictionary analysis basic function

#

##########################################################################################################

# (1) Function of matching adverbs of degree and set weights:

# (a). If there have adjectives or adverb before the sentiment word, they will be given different score

# according to different catagories.

# (b). Sentiment\_value will multiply to the value of those words before return to the recallee.

def match(word, sentiment\_value):

if word in mostdict:

sentiment\_value \*= 2.0

elif word in verydict:

sentiment\_value \*= 1.5

elif word in moredict:

sentiment\_value \*= 1.25

elif word in ishdict:

sentiment\_value \*= 0.5

elif word in insufficientdict:

sentiment\_value \*= 0.25

elif word in inversedict:

sentiment\_value \*= -1

return sentiment\_value

# (2) Function of transforming negative score to positive score for comparing:

# Formate each combination of poscount and the negcount to positive numbers so that they will easily be

# compared.

# Example: [5, -2] → [7, 0]; [-4, 8] → [0, 12];

def transform\_to\_positive\_num(poscount, negcount):

pos\_count = 0

neg\_count = 0

if poscount < 0 and negcount >= 0:

neg\_count += negcount - poscount

pos\_count = 0

elif negcount < 0 and poscount >= 0:

pos\_count = poscount - negcount

neg\_count = 0

elif poscount < 0 and negcount < 0:

neg\_count = -poscount

pos\_count = -negcount

else:

pos\_count = poscount

neg\_count = negcount

return [pos\_count, neg\_count]

# (3) Function of calculating review's every sentence sentiment score:

# Return the list containing positive value and negative value of the whole sentence

def sumup\_sentence\_sentiment\_score(score\_list):

score\_array = np.array(score\_list) # Change list to a numpy array

Pos = np.sum(score\_array[:,0]) # Compute positive score

Neg = np.sum(score\_array[:,1])

return [Pos, Neg] #, AvgPos, AvgNeg, StdPos, StdNeg]

# (4) Function for single review's positive and negative score:

# (a). Seperate the whole sentence into several meaning group by "and", "or","but" and punctuation.

# (b). In each meaning group, searching for adjective and adverb will be taken if finding a sentiment word,

# and searching range is from the word behind the last sentiment word to the current sentiment word.

# (c). Return the final sentiment score by using variable review\_sentiment\_score.

def single\_review\_sentiment\_score(seg\_sent):

single\_review\_senti\_score = []

word\_posit = 1 # word position counter for whole seg\_sent

group\_start\_bit = 0 # start position for each meaning group

start\_bit = 0 # sentiment word position

stop\_bit = 0 # word position counter for slice of seg\_sent[a:i]

poscount = 0 # count a positive score

negcount = 0 # count a negative score

pos\_flag = 0 # if there is a positive word, the pos\_flag will equal to 1, else equal to 0

neg\_flag = 0 # if there is a negative word, the neg\_flag will equal to 1, else equal to 0

for word in seg\_sent:

# seperate into different meaning group

if word == "and" or word == "or" or word in list(punctuation):

start\_bit = stop\_bit = group\_start\_bit

for x in seg\_sent[group\_start\_bit : word\_posit]:

stop\_bit += 1

if x in posdict:

poscount += 1

pos\_flag = 1

neg\_flag = 0

for w in seg\_sent[start\_bit : word\_posit]:

poscount = match(w, poscount)

start\_bit = stop\_bit + 1

elif x in negdict:

negcount += 1

pos\_flag = 0

neg\_flag = 1

for w in seg\_sent[start\_bit : word\_posit]:

negcount = match(w, negcount)

start\_bit = stop\_bit + 1

elif stop\_bit == word\_posit:

if pos\_flag == 1:

for w in seg\_sent[start\_bit : word\_posit]:

poscount = match(w, poscount)

elif neg\_flag == 1:

for w in seg\_sent[start\_bit : word\_posit]:

negcount = match(w, negcount)

group\_start\_bit = word\_posit + 1

pos\_flag = 0

neg\_flag = 0

# if one meaning group has negative words, it means positive sentiment even if having positive words

if negcount > 0 :

poscount = 0

# arriving at the end of the seg\_sent

elif word\_posit == len(seg\_sent):

start\_bit = group\_start\_bit

stop\_bit = word\_posit

for x in seg\_sent[ group\_start\_bit : word\_posit ]:

stop\_bit += 1

if x in posdict:

poscount += 1

pos\_flag = 1

neg\_flag = 0

for w in seg\_sent[start\_bit : word\_posit]:

poscount = match(w, poscount)

start\_bit = stop\_bit + 1

elif x in negdict:

negcount += 1

pos\_flag = 0

neg\_flag = 1

for w in seg\_sent[start\_bit : word\_posit]:

negcount = match(w, negcount)

start\_bit = stop\_bit + 1

elif stop\_bit == word\_posit:

if pos\_flag == 1:

for w in seg\_sent[start\_bit : word\_posit]:

poscount = match(w, poscount)

elif neg\_flag == 1:

for w in seg\_sent[start\_bit : word\_posit]:

negcount = match(w, negcount)

# if one meaning group has negative words, it means positive sentiment even if having positive words

if negcount > 0:

poscount = 0

word\_posit += 1

# formating the sentiment sorce and sendingsentiment score of the whole sentence to the review\_sentiment\_score

single\_review\_senti\_score.append(transform\_to\_positive\_num(poscount, negcount))

review\_sentiment\_score = sumup\_sentence\_sentiment\_score(single\_review\_senti\_score)

return review\_sentiment\_score

# (5) Function for Sentiment analysis and output:

# (a). Processing the sentences input from outside before sentiment analysis.

# (b). Recall the function of single\_review\_sentiment\_score to process sentiment analysis.

def review\_sentiment\_analysis(review):

# change the letter into lowercase and delete symbols including "\t" and "\r"

tweet\_processed=review.lower()

tweet\_processed=tweet\_processed.replace("'",'')

tweet\_processed=tweet\_processed.replace('"','')

for p in list(punctuation):

temp\_str=' '+p

tweet\_processed=tweet\_processed.replace(p,temp\_str)

tweet\_processed=tweet\_processed.replace("1",'')

tweet\_processed=tweet\_processed.replace("5",'')

tweet\_processed=tweet\_processed.replace("\t",'')

tweet\_processed=tweet\_processed.replace("\r",'')

words=tweet\_processed.split(' ')

# sentiment analysis

sentiment\_score = single\_review\_sentiment\_score(words)

if sentiment\_score[0] > sentiment\_score[1]:

return 'positive'

elif sentiment\_score[0] <= sentiment\_score[1]:

return 'negative'

##########################################################################################################

#

# Main function

# (1).Documents reading

#

##########################################################################################################

# list of files needed by the program, and they should be put at the same file path as the file of review\_sentiment\_analysis.py

filelist=['negative.txt','positive.txt','mostdict.txt', 'verydict.txt', 'moredict.txt','ishdict.txt','insufficientdict.txt','inversedict.txt']

# opening and reading from the files

for num in range(0, 8):

filelist[num] = os.path.join(os.getcwd(), filelist[num])

negative\_word = file(filelist[0], "r").read()

negative\_word=negative\_word.replace("\r",'')

negdict = negative\_word.split('\n')

positive\_word = file(filelist[1], "r").read()

positive\_word=positive\_word.replace("\r",'')

posdict = positive\_word.split('\n')

most\_word = file(filelist[2], "r").read()

most\_word=most\_word.replace("\r",'')

mostdict = most\_word.split('\n')

very\_word = file(filelist[3], "r").read()

very\_word=very\_word.replace("\r",'')

verydict = very\_word.split('\n')

more\_word = file(filelist[4], "r").read()

more\_word=more\_word.replace("\r",'')

moredict = more\_word.split('\n')

ish\_word = file(filelist[5], "r").read()

ish\_word=ish\_word.replace("\r",'')

ishdict = ish\_word.split('\n')

insufficient\_word = file(filelist[6], "r").read()

insufficient\_word=insufficient\_word.replace("\r",'')

insufficientdict = insufficient\_word.split('\n')

inverse\_word=file(filelist[7],"r").read()

inverse\_word=inverse\_word.replace("\r",'')

inversedict=inverse\_word.split('\n')

##########################################################################################################

#

# Main function

# (2).Documents processing

# (a) input one comment on the screen to get the sentiment analysis

# (b) input "exit" to exit

#

##########################################################################################################

print "Sentiment analysis"

print "Please input you recommend(input \"exit\" to exit):"

review = raw\_input()

while review != "exit":

# sentiment analysis and output the result

print\_string="the recommend is: " + review\_sentiment\_analysis(review) + '\n'

print print\_string

print "Sentiment analysis"

print "Please input you recommend(input \"exit\" to exit):"

review = raw\_input()

print "\nThanks very much!"

**Code for analysis the whole 24kData:**

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

#from \_\_future\_\_ import division

import os

import sys

import numpy as np

from string import punctuation

##########################################################################################################

#

# Sentiment dictionary analysis basic function

#

##########################################################################################################

# (1) Function of matching adverbs of degree and set weights:

# (a). If there have adjectives or adverb before the sentiment word, they will be given different score

# according to different catagories.

# (b). Sentiment\_value will multiply to the value of those words before return to the recallee.

def match(word, sentiment\_value):

if word in mostdict:

sentiment\_value \*= 2.0

elif word in verydict:

sentiment\_value \*= 1.5

elif word in moredict:

sentiment\_value \*= 1.25

elif word in ishdict:

sentiment\_value \*= 0.5

elif word in insufficientdict:

sentiment\_value \*= 0.25

elif word in inversedict:

sentiment\_value \*= -1

return sentiment\_value

# (2) Function of transforming negative score to positive score for comparing:

# Formate each combination of poscount and the negcount to positive numbers so that they will easily be

# compared.

# Example: [5, -2] → [7, 0]; [-4, 8] → [0, 12];

def transform\_to\_positive\_num(poscount, negcount):

pos\_count = 0

neg\_count = 0

if poscount < 0 and negcount >= 0:

neg\_count += negcount - poscount

pos\_count = 0

elif negcount < 0 and poscount >= 0:

pos\_count = poscount - negcount

neg\_count = 0

elif poscount < 0 and negcount < 0:

neg\_count = -poscount

pos\_count = -negcount

else:

pos\_count = poscount

neg\_count = negcount

return [pos\_count, neg\_count]

# (3) Function of calculating review's every sentence sentiment score:

# Return the list containing positive value and negative value of the whole sentence

def sumup\_sentence\_sentiment\_score(score\_list):

score\_array = np.array(score\_list)

Pos = np.sum(score\_array[:,0])

Neg = np.sum(score\_array[:,1])

return [Pos, Neg]

# (4) Function for single review's positive and negative score:

# (a). Seperate the whole sentence into several meaning group by "and", "or","but" and punctuation.

# (b). In each meaning group, searching for adjective and adverb will be taken if finding a sentiment word,

# and searching range is from the word behind the last sentiment word to the current sentiment word.

# (c). Return the final sentiment score by using variable review\_sentiment\_score.

def single\_review\_sentiment\_score(seg\_sent):

single\_review\_senti\_score = []

word\_posit = 1 # word position counter for whole seg\_sent

group\_start\_bit = 0 # start position for each meaning group

start\_bit = 0 # sentiment word position

stop\_bit = 0 # word position counter for slice of seg\_sent[a:i]

poscount = 0 # count a positive score

negcount = 0 # count a negative score

pos\_flag = 0 # if there is a positive word, the pos\_flag will equal to 1, else equal to 0

neg\_flag = 0 # if there is a negative word, the neg\_flag will equal to 1, else equal to 0

for word in seg\_sent:

# seperate into different meaning group

if word == "and" or word == "or" or word in list(punctuation):

start\_bit = stop\_bit = group\_start\_bit

for x in seg\_sent[ group\_start\_bit : word\_posit ]:

stop\_bit += 1

if x in posdict:

poscount += 1

pos\_flag = 1

neg\_flag = 0

for w in seg\_sent[start\_bit:word\_posit]:

poscount = match(w, poscount)

start\_bit = stop\_bit + 1

elif x in negdict:

negcount += 1

pos\_flag = 0

neg\_flag = 1

for w in seg\_sent[start\_bit:word\_posit]:

negcount = match(w, negcount)

start\_bit = stop\_bit + 1

elif stop\_bit == word\_posit:

if pos\_flag == 1:

for w in seg\_sent[start\_bit:word\_posit]:

poscount = match(w, poscount)

elif neg\_flag == 1:

for w in seg\_sent[start\_bit:word\_posit]:

negcount = match(w, negcount)

group\_start\_bit = word\_posit + 1

pos\_flag = 0

neg\_flag = 0

# if one meaning group has negative words, it means positive sentiment even if having positive words

if negcount > 0:

poscount = 0

# arriving at the end of the seg\_sent

elif word\_posit == len(seg\_sent):

start\_bit = stop\_bit = group\_start\_bit

for x in seg\_sent[ group\_start\_bit : word\_posit ]:

stop\_bit += 1

if x in posdict:

poscount += 1

pos\_flag = 1

neg\_flag = 0

for w in seg\_sent[start\_bit:word\_posit]:

poscount = match(w, poscount)

start\_bit = stop\_bit + 1

elif x in negdict:

negcount += 1

pos\_flag = 0

neg\_flag = 1

for w in seg\_sent[start\_bit:word\_posit]:

negcount = match(w, negcount)

start\_bit = stop\_bit + 1

elif stop\_bit == word\_posit:

if pos\_flag == 1:

for w in seg\_sent[start\_bit:word\_posit]:

poscount = match(w, poscount)

elif neg\_flag == 1:

for w in seg\_sent[start\_bit:word\_posit]:

negcount = match(w, negcount)

# if one meaning group has negative words, it means positive sentiment even if having positive words

if negcount > 0:

poscount = 0

word\_posit += 1

single\_review\_senti\_score.append(transform\_to\_positive\_num(poscount, negcount))

review\_sentiment\_score = sumup\_sentence\_sentiment\_score(single\_review\_senti\_score)

return review\_sentiment\_score

# (5) Function for Sentiment analysis and output:

# (a). Processing the sentences input from outside before sentiment analysis.

# (b). Recall the function of single\_review\_sentiment\_score to process sentiment analysis.

def review\_sentiment\_analysis(review):

tweet\_processed=review.lower()

tweet\_processed=tweet\_processed.replace("'",'')

tweet\_processed=tweet\_processed.replace('"','')

for p in list(punctuation):

temp\_str=' '+p

tweet\_processed=tweet\_processed.replace(p,temp\_str)

tweet\_processed=tweet\_processed.replace("1",'')

tweet\_processed=tweet\_processed.replace("5",'')

tweet\_processed=tweet\_processed.replace("\t",'')

tweet\_processed=tweet\_processed.replace("\r",'')

words=tweet\_processed.split(' ')

sentiment\_score = single\_review\_sentiment\_score(words)

if sentiment\_score[0] > sentiment\_score[1]:

return 'positive'

elif sentiment\_score[0] <= sentiment\_score[1]:

return 'negative'

##########################################################################################################

#

# Main function

# (1).Documents reading

#

##########################################################################################################

# list of files needed by the program, and they should be put at the same file path as the file of review\_sentiment\_analysis.py

filelist=['negative.txt','positive.txt','mostdict.txt', 'verydict.txt', 'moredict.txt','ishdict.txt','insufficientdict.txt','inversedict.txt','24kData.txt']

for num in range(0, 9):

filelist[num] = os.path.join(os.getcwd(), filelist[num])

negative\_word = file(filelist[0], "r").read()

negative\_word=negative\_word.replace("\r",'')

negdict = negative\_word.split('\n')

positive\_word = file(filelist[1], "r").read()

positive\_word=positive\_word.replace("\r",'')

posdict = positive\_word.split('\n')

most\_word = file(filelist[2], "r").read()

most\_word=most\_word.replace("\r",'')

mostdict = most\_word.split('\n')

very\_word = file(filelist[3], "r").read()

very\_word=very\_word.replace("\r",'')

verydict = very\_word.split('\n')

more\_word = file(filelist[4], "r").read()

more\_word=more\_word.replace("\r",'')

moredict = more\_word.split('\n')

ish\_word = file(filelist[5], "r").read()

ish\_word=ish\_word.replace("\r",'')

ishdict = ish\_word.split('\n')

insufficient\_word = file(filelist[6], "r").read()

insufficient\_word=insufficient\_word.replace("\r",'')

insufficientdict = insufficient\_word.split('\n')

inverse\_word=file(filelist[7],"r").read()

inverse\_word=inverse\_word.replace("\r",'')

inversedict=inverse\_word.split('\n')

recommend\_contain = file(filelist[8], "r").read()

recommend\_list = recommend\_contain.split('\n')

##########################################################################################################

#

# Main function

# (2).Documents processing

# write the new output data to a new file

#

##########################################################################################################

for i in range(0, len(recommend\_list)):

recommend\_list[i] = review\_sentiment\_analysis(recommend\_list[i]) + ' ' + recommend\_list[i] + "\n"

fp = file(os.path.join(os.getcwd(), "output.txt"), "w")

fp.writelines(recommend\_list)

fp.close()