models Namespace Reference

Classes

class add_data

Functions

def remove_redundant_functions (content)
def remove_comments (string)
def visualizer (list_of_files, similarity_matrix)
def remove_macros (content)
def remove_comments_pythonfile (file_content)
def preprocessing (list_of_paths, list_of_files)
def tf_idf (word_count_in_each_file, word_count_across_documents, list_of_paths, list_of_files)
def txt_file (similarity_matrix, list_of_paths, list_of_files)
def csv_file (list_of_files, similarity_matrix)
def similarity (s, t)

Function Documentation

csv_file()

def models.csv_file (list_of_files,
similarity_matrix
)

```
Arguments     :
    list_of_files    :list of source code file names
    similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
Functionality:
    Plotting the output similarity_matrix and saving it as an csv file
```

```
296 def csv_file(list_of_files,similarity_matrix):
297     #""" Interpreting the Output data as a CSV file ,
298     #where each element represent the percentage matching between the file
299     #corresponding to a row and column"""
300     """
301     Arguments     :
302         list_of_files    :list of source code file names
303         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
304     Functionality:
305         Plotting the output similarity_matrix and saving it as an csv file
306 
307     """
308     f=similarity_matrix.tolist()
309     files=['filenames']+list_of_files
310     for x in range(len(list_of_files)):
311         f[x]=[list_of_files[x]]+f[x]
312     f=[files]+f
313     with open("media/result.csv", "w+") as myCsv:
314         csvWriter = csv.writer(myCsv, delimiter=',')
315         csvWriter.writerows(f)
316     visualizer(list_of_files,similarity_matrix)
```

preprocessing()

def models.preprocessing (list_of_paths,
list_of_files
)

```
Core logic is based on the Bag_of_Words
and TF-IDF Strategy( Term frequency and Inverse Document Frequency )

Arguments    :
    list_of_paths    :list of source code paths
    list_of_files    :it consists of list of file names
Functionality:
    It finds the count of each word after removing comments and replacing macros and passes this vector to tf_idf functio
```

```
167 def preprocessing(list_of_paths,list_of_files):
168     """
169     Core logic is based on the Bag_of_Words
170     and TF-IDF Strategy( Term frequency and Inverse Document Frequency )
171 
172     Arguments    :
173         list_of_paths    :list of source code paths
174         list_of_files    :it consists of list of file names
175     Functionality:
176         It finds the count of each word after removing comments and replacing macros and passes this vector to tf_idf function.
177 
178     """
179     word_count_across_documents={}
180     """ Maintains the word count of each element in a document """
181 
182     word_count_in_each_file=[]
183 
184     for files in list_of_paths:
185 
186         filename='media/'+files
187         temp={}
188         myfile=open(filename,"r")
189         content=myfile.read()
190         myfile.close()
191 
192         if(files[-4:]=='.cpp'):
193             content=remove_redundant_functions(content)
194             content=remove_macros(content)
195             content=remove_comments(content)
196         if(files[-5:]=='.java'):
197             content=remove_comments(content)
198         elif(files[-3:]=='.py'):
199             content=remove_comments_pythonfile(content)
200         sym=[",",";","{","}",")","(","[","]","+","-","*","/","%","|","&","^","!","=","<",">","?","'",'"','#','.']
201         for i in sym:
202             content=content.replace(i," "+i+" ")
203 
204         if(files[-4:]=='.cpp' or files[-5:]=='.java'):
205             content=content.replace("while","for")
206             content=content.replace("switch","if")
207             content=content.replace("case","else if")
208             content=content.replace("default","else")
209             content=content.replace("unsigned long long int","double")
210             content=content.replace("unsigned long long","double")
211             content=content.replace("long long int","double")
212             content=content.replace("long long","double")
213             content=content.replace("float","double")
214             content=content.replace("int","double")
215             content=content.replace("for","double")
216             content=content.replace("+ +","+ = 1")
217             content=content.replace("- -","- = 1")
218             content=content.replace("< <","<<")
219             content=content.replace('> >',">>")
220             cont=""
221             for i in content.split('\n'):
222                 if(i=='\n'):
223                     continue
224                 i=i.strip()
225                 if(len(i)==0 or i[0]=='#'):
226                     continue
227                 else:
228                     cont=cont+' '+i
229             content=cont
230 
231         elif(files[-3:]=='.py'):
232             content=content.replace('while','for')
233             content=content.replace('switch','if')
234             content=content.replace('case','elif')
235             content=content.replace('default','else')
236             content=content.replace('do','')
237             content=re.sub(':|\'|\"','',content)
238         List_of_words=content.split()
239 
240         for i in List_of_words:
241             temp[i]=temp.get(i,0)+1
242             word_count_across_documents[i]=word_count_across_documents.get(i,0)+1
243 
244         word_count_in_each_file.append(temp)
245     tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files)
246 
```

- remove_comments()

def models.remove_comments (string)

```
66 def remove_comments(string):
67     pattern = r"(\".*?\"|\'.*?\')|(/\*.*?\*/|//[^\r\n]*$)"
68     # first group captures quoted strings (double or single)
69     # second group captures comments (//single-line or /* multi-line */)
70     regex = re.compile(pattern, re.MULTILINE|re.DOTALL)
71     def _replacer(match):
72         # if the 2nd group (capturing comments) is not None,
73         # it means we have captured a non-quoted (real) comment string.
74         if match.group(2) is not None:
75             return "" # so we will return empty to remove the comment
76         else: # otherwise, we will return the 1st group
77             return match.group(1) # captured quoted-string
78     return regex.sub(_replacer, string)
79
```

remove_comments_pythonfile()

def models.remove_comments_pythonfile (file_content)

```
Arguments:
    file_content: string storing the source code in python
Return type: updated string
Functionality:
    All commments in the code are replaced.
Logic Used:
    Using Regex detect substrings starting with # and ending with \n
    Similarly detect substrings starting with ''' and ending with '''
```

```
152 def remove_comments_pythonfile(file_content):
153     """
154     Arguments:
155         file_content: string storing the source code in python
156     Return type: updated string
157     Functionality:
158         All commments in the code are replaced.
159     Logic Used:
160         Using Regex detect substrings starting with # and ending with \n
161         Similarly detect substrings starting with ''' and ending with '''
162     """
163     pattern=re.compile('#.*?$|\'\'\'.*?\'\'\'|\"\"\".*?\"\"\"',re.DOTALL|re.MULTILINE)
164     content=re.sub(pattern,'',file_content)
165     return content
166
```

remove_macros()

def models.remove_macros (content)

```
Arguments:
    content: string storing the source code
Return type: updated string
Functionality:
    All macros in the code are replaced.
Logic Used:
    Using Regex detect the macros
    Replace them using replace() function
```

```
111 def remove_macros(content):
112
113
114     """
115     Arguments:
116         content: string storing the source code
117     Return type: updated string
118     Functionality:
119         All macros in the code are replaced.
120     Logic Used:
121         Using Regex detect the macros
122         Replace them using replace() function
123     """
124     temp = open("temp.cpp", "w")
125     temp.write(content)
126     temp.close()
127     pre = subprocess.getstatusoutput("g++ -E temp.cpp")
128     prep=pre[1].split('using namespace std;')[-1]
129     content=prep
130
131     m=re.findall('typedef .+ .+',content)
132     """finding macros"""
133
134     content=re.sub('typedef .+ .+','',content)
135     """removing macros definitions"""
136
137     for i in range(len(m)):
138         """replacing macros"""
139         j=m[i].split()
140         if(j[-1]==';'):
141             last_word=j[-2]
142         else:
143             last_word=j[-1]
144             if(last_word[-1]==';'):
145                 last_word=last_word[:-1]
146             string=""
147             for i in range(1,len(j)-1):
148                 string+=j[i]+' '
149             content=content.replace(last_word,string)
150     return content
151
```

- remove_redundant_functions()

def models.remove_redundant_functions (content)

```
Arguments:
    content: string storing the source code
Return type: updated string
Functionality:
    redundant functions in a source code are removed
Logic Used:
     Write a minimal parser that can identify functions.
     It just needs to detect the start and ending line of a function.
     Programmatically comment out the first function, save to a temp file.
     Try to compile the file by invoking the complier.
     Check if there are compile errors, if yes, the function is called, if not, it is unused.
     Continue with the next function.
     Reference:https://stackoverflow.com/questions/33209302/removal-of-unused-or-redundant-code
```

```
24 def remove_redundant_functions(content):
25     """
26     Arguments:
27         content: string storing the source code
28     Return type: updated string
29     Functionality:
30         redundant functions in a source code are removed
31     Logic Used:
32          Write a minimal parser that can identify functions.
33          It just needs to detect the start and ending line of a function.
34          Programmatically comment out the first function, save to a temp file.
35          Try to compile the file by invoking the complier.
36          Check if there are compile errors, if yes, the function is called, if not, it is unused.
37          Continue with the next function.
38          Reference:https://stackoverflow.com/questions/33209302/removal-of-unused-or-redundant-code
39     """
40     type_list=['int','void','char','string']
41     t=""
42     for type in type_list:
43         L=re.findall(type+"\s*[a-z0-9_]\s\([a-z0-9_ \n\t,\r\f\v]\)\s\{",content)
44         for y in L:
45             y=y.replace("(","\(")
46             y=y.replace(")","\)")
47             x=re.search(y,content)
48             first=x.span()[0]
49             last=x.span()[1]
50             count=1
51             while(count!=0):
52                 if(content[last]=='{'):
53                     count+=1
54                 if(content[last]=='}'):
55                     count-=1
56                 last+=1
57             t=content[0:first]+content[last:]
58         temp = open("temp.cpp", "w")
59         temp.write(t)
60         temp.close()
61         g = subprocess.getstatusoutput("g++ temp.cpp")
62         if(g[1]==""):
63             content=t
64     return content
65
```

- similarity()

def models.similarity (s,
t
)

```
Arguments    :
    s   :(sorted)list of numbers
    t   :(sorted)list of numbers
Return type  :
    returns a number in the range(0,1)
Functionality:
    Evaluates the cosine product of the two vectors
```

```
317 def similarity(s,t):
318
319     """
320     Arguments    :
321         s   :(sorted)list of numbers
322         t   :(sorted)list of numbers
323     Return type  :
324         returns a number in the range(0,1)
325     Functionality:
326         Evaluates the cosine product of the two vectors
327     """
328     x=np.zeros(abs(s.size-t.size))
329     '''
330     if(s.size>t.size):
331         t=np.concatenate((x,t))
332     else:
333         s=np.concatenate((x,s))
334     '''
335     x=min(s.size,t.size)
336     s=s[-x:]
337     t=t[-x:]
338
339     #s=(s-np.mean(s))/np.std(s)
340     #t=(t-np.mean(t))/np.std(t)
341     return 1-np.linalg.norm(s-t)/(np.linalg.norm(s)+np.linalg.norm(t))
342     return np.dot(s,t)/(np.linalg.norm(s)*np.linalg.norm(t))
```

- tf_idf()

def models.tf_idf (word_count_in_each_file,
word_count_across_documents,
list_of_paths,
list_of_files
)

```
 Arguments     :
    list_of_paths              :list of source code files
    list_of_files                  :It consists data of all the Users who have been SignedUp
    word_count_in_each_file    :Frequency of word corresponding to each file as an array of dictionary
    word_count_across_documents:Frequency of each word across as files corresponding to a particular assignment as dictio
Functionality:
    It computes tf_idf vector corresponding to each file.
    The tf_idf function is somewhat different from the original one
    If we use the bag of words strategy then similarity is determined mostly by the variables which have maximum count in
    But similarity should depend more on core logic loke number of functions,operators loops etc.
    The weight added for each word say 'x' in file 'f' is log(freq of x across all files corresponding to assignment/(fre
    Words which have low frequency in a file than average frequency across all files are given +ve weightage
    Words which have high frequency in a file than average frequency across all files are given -ve weightage
    Uniqueness is determined by high weightage words.
```

```
247 def tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files):
248     """ Arguments     :
249         list_of_paths              :list of source code files
250         list_of_files                  :It consists data of all the Users who have been SignedUp
251         word_count_in_each_file    :Frequency of word corresponding to each file as an array of dictionary
252         word_count_across_documents:Frequency of each word across as files corresponding to a particular assignment as dictionary
253     Functionality:
254         It computes tf_idf vector corresponding to each file.
255         The tf_idf function is somewhat different from the original one
256         If we use the bag of words strategy then similarity is determined mostly by the variables which have maximum count in a file.
257         But similarity should depend more on core logic loke number of functions,operators loops etc.
258         The weight added for each word say 'x' in file 'f' is log(freq of x across all files corresponding to assignment/(freq of x i
259         Words which have low frequency in a file than average frequency across all files are given +ve weightage
260         Words which have high frequency in a file than average frequency across all files are given -ve weightage
261         Uniqueness is determined by high weightage words.
262     """
263     similarity_matrix=np.zeros((len(list_of_paths),len(list_of_paths)))
264     tf_idf_vec=[]
265     for i in range(len(list_of_paths)):
266         temp=[]
267         for j in word_count_in_each_file[i]:
268             temp.append(word_count_in_each_file[i].get(j)*((math.log(word_count_across_documents.get(j)/word_count_in_each_file[i].ge
269         temp.sort()
270         tf_idf_vec.append(temp)
271 
272     for i in range(len(list_of_paths)):
273         similarity_matrix[i,i]=0
274         for j in range(i+1,len(list_of_paths)):
275             similarity_matrix[i,j]=similarity(np.array(tf_idf_vec[i]),np.array(tf_idf_vec[j]))
276             similarity_matrix[j,i]=similarity_matrix[i,j]
277 
278     txt_file(similarity_matrix,list_of_paths,list_of_files)
279 
```

txt_file()

def models.txt_file (similarity_matrix,
list_of_paths,
list_of_files
)

```
Arguments     :
    list_of_paths     :list of source code file names
    similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
    list_of_files         :It consists data of all the Users who have been SignedUp
Functionality:
    Displaying the Percentage matching among files in text format and saving it as a csv file
```

```
280 def txt_file(similarity_matrix,list_of_paths,list_of_files):
281     """
282     Arguments     :
283         list_of_paths     :list of source code file names
284         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
285         list_of_files         :It consists data of all the Users who have been SignedUp
286     Functionality:
287         Displaying the Percentage matching among files in text format and saving it as a csv file """
288     result=open("media/result.txt","w")
289     res=""
290     for i in range(len(list_of_paths)):
291         for j in range(i+1,len(list_of_paths)):
292             res+="similarity between "+ list_of_files[i]+" and "+list_of_files[j]+" = "+str(similarity_matrix[i][j])+"\n"
293     result.write(res)
294     csv_file(list_of_files,similarity_matrix)
295 
```

visualizer()

def models.visualizer (list_of_files,
similarity_matrix
)

```
Arguments     :
    list_of_files    :list of source code file names
    similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
Return type   :
    Path of the saved image
Functionality:
    Plotting the output similarity_matrix and saving it as an image
```

```
 80 def visualizer(list_of_files,similarity_matrix):
 81     """
 82     Arguments     :
 83         list_of_files    :list of source code file names
 84         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
 85     Return type   :
 86         Path of the saved image
 87     Functionality:
 88         Plotting the output similarity_matrix and saving it as an image """
 89     x=range(len(list_of_files))
 90     y=range(len(list_of_files))
 91     xx,yy=np.meshgrid(x,y)
 92     z=similarity_matrix[xx,yy]
 93     z=np.round(z*100)
 94     cmap = matplotlib.colors.LinearSegmentedColormap.from_list("", ["green","yellow","red"])
 95     fig=plt.figure()
 96     ax=fig.add_subplot(111)
 97     im=ax.matshow(z,cmap=cmap,vmin=0,vmax=100,origin='lower')
 98     for i in range(len(list_of_files)):
 99         for j in range(i+1,len(list_of_files)):
100             ax.text(j, i, int(similarity_matrix[i,j]*100),ha="center", va="center", color="k")
101             ax.text(i,j, int(similarity_matrix[i,j]*100),ha="center", va="center", color="k")
102     fig.colorbar(im,shrink=0.5)
103     ax.set_xticks(range(len(list_of_files)))
104     ax.set_yticks(range(len(list_of_files)))
105     ax.set_xticklabels(list_of_files,rotation=90)
106     ax.set_yticklabels(list_of_files)
107     random=np.random.randint(1,100)
108     path='result.png'
109     plt.savefig('media/'+path)
110
```