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
```

```
300 def csv_file(list_of_files,similarity_matrix):
301     #""" Interpreting the Output data as a CSV file ,
302     #where each element represent the percentage matching between the file
303     #corresponding to a row and column"""
304     """
305     Arguments     :
306         list_of_files    :list of source code file names
307         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
308     Functionality:
309         Plotting the output similarity_matrix and saving it as an csv file
310 
311     """
312     f=similarity_matrix.tolist()
313     files=['filenames']+list_of_files
314     for x in range(len(list_of_files)):
315         f[x]=[list_of_files[x]]+f[x]
316     f=[files]+f
317     with open("media/result.csv", "w+") as myCsv:
318         csvWriter = csv.writer(myCsv, delimiter=',')
319         csvWriter.writerows(f)
320     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
```

```
171 def preprocessing(list_of_paths,list_of_files):
172     """
173     Core logic is based on the Bag_of_Words
174     and TF-IDF Strategy( Term frequency and Inverse Document Frequency )
175 
176     Arguments    :
177         list_of_paths    :list of source code paths
178         list_of_files    :it consists of list of file names
179     Functionality:
180         It finds the count of each word after removing comments and replacing macros and passes this vector to tf_idf function.
181 
182     """
183     word_count_across_documents={}
184     """ Maintains the word count of each element in a document """
185 
186     word_count_in_each_file=[]
187 
188     for files in list_of_paths:
189 
190         filename='media/'+files
191         temp={}
192         myfile=open(filename,"r")
193         content=myfile.read()
194         myfile.close()
195 
196         if(files[-4:]=='.cpp'):
197             content=remove_redundant_functions(content)
198             content=remove_macros(content)
199             content=remove_comments(content)
200         if(files[-5:]=='.java'):
201             content=remove_comments(content)
202         elif(files[-3:]=='.py'):
203             content=remove_comments_pythonfile(content)
204         sym=[",",";","{","}",")","(","[","]","+","-","*","/","%","|","&","^","!","=","<",">","?","'",'"','#','.']
205         for i in sym:
206             content=content.replace(i," "+i+" ")
207 
208         if(files[-4:]=='.cpp' or files[-5:]=='.java'):
209             content=content.replace("while","for")
210             content=content.replace("switch","if")
211             content=content.replace("case","else if")
212             content=content.replace("default","else")
213             content=content.replace("unsigned long long int","double")
214             content=content.replace("unsigned long long","double")
215             content=content.replace("long long int","double")
216             content=content.replace("long long","double")
217             content=content.replace("float","double")
218             content=content.replace("int","double")
219             content=content.replace("for","double")
220             content=content.replace("+ +","+ = 1")
221             content=content.replace("- -","- = 1")
222             content=content.replace("< <","<<")
223             content=content.replace('> >',">>")
224             cont=""
225             for i in content.split('\n'):
226                 if(i=='\n'):
227                     continue
228                 i=i.strip()
229                 if(len(i)==0 or i[0]=='#'):
230                     continue
231                 else:
232                     cont=cont+' '+i
233             content=cont
234 
235         elif(files[-3:]=='.py'):
236             content=content.replace('while','for')
237             content=content.replace('switch','if')
238             content=content.replace('case','elif')
239             content=content.replace('default','else')
240             content=content.replace('do','')
241             content=re.sub(':|\'|\"','',content)
242         List_of_words=content.split()
243 
244         for i in List_of_words:
245             temp[i]=temp.get(i,0)+1
246             word_count_across_documents[i]=word_count_across_documents.get(i,0)+1
247 
248         word_count_in_each_file.append(temp)
249     tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files)
250 
```

- 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 '''
```

```
156 def remove_comments_pythonfile(file_content):
157     """
158     Arguments:
159         file_content: string storing the source code in python
160     Return type: updated string
161     Functionality:
162         All commments in the code are replaced.
163     Logic Used:
164         Using Regex detect substrings starting with # and ending with \n
165         Similarly detect substrings starting with ''' and ending with '''
166     """
167     pattern=re.compile('#.*?$|\'\'\'.*?\'\'\'|\"\"\".*?\"\"\"',re.DOTALL|re.MULTILINE)
168     content=re.sub(pattern,'',file_content)
169     return content
170
```

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:
    This is based on preprocessing done by g++ compiler
    This function assumes the existence of "using namespace std;" as a substring in the source code string
    g++ -E file.cpp produces the preprocessed code which does not contain any comments or macro
    The substring following "using namespace std;" is extracted
    Using Regex to detect the typedef 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         This is based on preprocessing done by g++ compiler
122         This function assumes the existence of "using namespace std;" as a substring in the source code string
123         g++ -E file.cpp produces the preprocessed code which does not contain any comments or macro
124         The substring following "using namespace std;" is extracted
125         Using Regex to detect the typedef macros
126         Replace them using replace() function
127     """
128     temp = open("temp.cpp", "w")
129     temp.write(content)
130     temp.close()
131     pre = subprocess.getstatusoutput("g++ -E temp.cpp")
132     prep=pre[1].split('using namespace std;')[-1]
133     content=prep
134 
135     m=re.findall('typedef .+ .+',content)
136     """finding macros"""
137 
138     content=re.sub('typedef .+ .+','',content)
139     """removing macros definitions"""
140 
141     for i in range(len(m)):
142         """replacing macros"""
143         j=m[i].split()
144         if(j[-1]==';'):
145             last_word=j[-2]
146         else:
147             last_word=j[-1]
148             if(last_word[-1]==';'):
149                 last_word=last_word[:-1]
150             string=""
151             for i in range(1,len(j)-1):
152                 string+=j[i]+' '
153             content=content.replace(last_word,string)
154     return content
155 
```

- 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','double']
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
```

```
321 def similarity(s,t):
322
323     """
324     Arguments     :
325         s    :(sorted)list of numbers
326         t    :(sorted)list of numbers
327     Return type   :
328         returns a number in the range(0,1)
329     Functionality:
330         Evaluates the cosine product of the two vectors
331     """
332     x=np.zeros(abs(s.size-t.size))
333     '''
334     if(s.size>t.size):
335         t=np.concatenate((x,t))
336     else:
337         s=np.concatenate((x,s))
338     '''
339     x=min(s.size,t.size)
340     s=s[-x:]
341     t=t[-x:]
342
343     #s=(s-np.mean(s))/np.std(s)
344     #t=(t-np.mean(t))/np.std(t)
345     return 1-np.linalg.norm(s-t)/(np.linalg.norm(s)+np.linalg.norm(t))
346     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.
```

```
251 def tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files):
252     """ Arguments     :
253         list_of_paths              :list of source code files
254         list_of_files                  :It consists data of all the Users who have been SignedUp
255         word_count_in_each_file    :Frequency of word corresponding to each file as an array of dictionary
256         word_count_across_documents:Frequency of each word across as files corresponding to a particular assignment as dictionary
257     Functionality:
258         It computes tf_idf vector corresponding to each file.
259         The tf_idf function is somewhat different from the original one
260         If we use the bag of words strategy then similarity is determined mostly by the variables which have maximum count in a file.
261         But similarity should depend more on core logic loke number of functions,operators loops etc.
262         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
263         Words which have low frequency in a file than average frequency across all files are given +ve weightage
264         Words which have high frequency in a file than average frequency across all files are given -ve weightage
265         Uniqueness is determined by high weightage words.
266     """
267     similarity_matrix=np.zeros((len(list_of_paths),len(list_of_paths)))
268     tf_idf_vec=[]
269     for i in range(len(list_of_paths)):
270         temp=[]
271         for j in word_count_in_each_file[i]:
272             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
273         temp.sort()
274         tf_idf_vec.append(temp)
275 
276     for i in range(len(list_of_paths)):
277         similarity_matrix[i,i]=0
278         for j in range(i+1,len(list_of_paths)):
279             similarity_matrix[i,j]=similarity(np.array(tf_idf_vec[i]),np.array(tf_idf_vec[j]))
280             similarity_matrix[j,i]=similarity_matrix[i,j]
281 
282     txt_file(similarity_matrix,list_of_paths,list_of_files)
283 
```

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
```

```
284 def txt_file(similarity_matrix,list_of_paths,list_of_files):
285     """
286     Arguments     :
287         list_of_paths     :list of source code file names
288         similarity_matrix:2-dimensional matrix representing mutual similarity between each pair of files
289         list_of_files         :It consists data of all the Users who have been SignedUp
290     Functionality:
291         Displaying the Percentage matching among files in text format and saving it as a csv file """
292     result=open("media/result.txt","w")
293     res=""
294     for i in range(len(list_of_paths)):
295         for j in range(i+1,len(list_of_paths)):
296             res+="similarity between "+ list_of_files[i]+" and "+list_of_files[j]+" = "+str(similarity_matrix[i][j])+"\n"
297     result.write(res)
298     csv_file(list_of_files,similarity_matrix)
299 
```

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(similarity_matrix,cmap=cmap,vmin=0,vmax=1,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",fontsize=50/(len(list_of_files)+1))
101             ax.text(i,j, int(similarity_matrix[i,j]*100),ha="center", va="center", color="k",fontsize=50/(len(list_of_files)+1))
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
```