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
```

```
def csv_file(list_of_files,similarity_matrix):
    #""" Interpreting the Output data as a CSV file ,
    #where each element represent the percentage matching between the file
    #corresponding to a row and column"""
    """
    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

    """
    f=similarity_matrix.tolist()
    files=['filenames']+list_of_files
    for x in range(len(list_of_files)):
        f[x]=[list_of_files[x]]+f[x]
    f=[files]+f
    with open("media/result.csv", "w+") as myCsv:
        csvWriter = csv.writer(myCsv, delimiter=',')
        csvWriter.writerows(f)
    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
```

```python
def 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 function.

    """
    word_count_across_documents={}
    """ Maintains the word count of each element in a document """

    word_count_in_each_file=[]

    for files in list_of_paths:

        filename='media/'+files
        temp={}
        myfile=open(filename,"r")
        content=myfile.read()
        myfile.close()

        if(files[-4:]=='.cpp'):
            content=remove_redundant_functions(content)
            content=remove_macros(content)
            content=remove_comments(content)
        if(files[-5:]=='.java'):
            content=remove_comments(content)
        elif(files[-3:]=='.py'):
            content=remove_comments_pythonfile(content)
        sym=[",",";","{","}",")","(","[","]","+","-","*","/","%","|","&","^","!","=","<",">","?","'",'"','#','.']
        for i in sym:
            content=content.replace(i," "+i+" ")

        if(files[-4:]=='.cpp' or files[-5:]=='.java'):
            content=content.replace("while","for")
            content=content.replace("switch","if")
            content=content.replace("case","else if")
            content=content.replace("default","else")
            content=content.replace("unsigned long long int","double")
            content=content.replace("unsigned long long","double")
            content=content.replace("long long int","double")
            content=content.replace("long long","double")
            content=content.replace("float","double")
            content=content.replace("int","double")
            content=content.replace("for","double")
            content=content.replace("+ +","+ = 1")
            content=content.replace("- -","- = 1")
            content=content.replace("< <","<<")
            content=content.replace('> >',">>")
            cont=""
            for i in content.split('\n'):
                if(i=='\n'):
                    continue
                i=i.strip()
                if(len(i)==0 or i[0]=='#'):
                    continue
                else:
                    cont=cont+' '+i
            content=cont

        elif(files[-3:]=='.py'):
            content=content.replace('while','for')
            content=content.replace('switch','if')
            content=content.replace('case','elif')
            content=content.replace('default','else')
            content=content.replace('do','')
            content=re.sub(':|\'|\"','',content)
        List_of_words=content.split()

        for i in List_of_words:
            temp[i]=temp.get(i,0)+1
            word_count_across_documents[i]=word_count_across_documents.get(i,0)+1

        word_count_in_each_file.append(temp)
    tf_idf(word_count_in_each_file,word_count_across_documents,list_of_paths,list_of_files)

```

◆ remove_comments()

def models.remove_comments (string)

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

```

◆ 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 '''
```

```
def 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 '''
    """
    pattern=re.compile('#.*?$|\'\'\'.*?\'\'\'|\"\"\".*?\"\"\"',re.DOTALL|re.MULTILINE)
    content=re.sub(pattern,'',file_content)
    return content

```

◆ 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
```

```python
def 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
    """
    temp = open("temp.cpp", "w")
    temp.write(content)
    temp.close()
    pre = subprocess.getstatusoutput("g++ -E temp.cpp")
    prep=pre[1].split('using namespace std;')[-1]
    content=prep

    m=re.findall('typedef .+ .+',content)
    """finding macros"""

    content=re.sub('typedef .+ .+','',content)
    """removing macros definitions"""

    for i in range(len(m)):
        """replacing macros"""
        j=m[i].split()
        if(j[-1]==';'):
            last_word=j[-2]
        else:
            last_word=j[-1]
            if(last_word[-1]==';'):
                last_word=last_word[:-1]
            string=""
            for i in range(1,len(j)-1):
                string+=j[i]+' '
            content=content.replace(last_word,string)
    return content

```

◆ 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
```

```
def 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
    """
    type_list=['int','void','char','string','double']
    t=""
    for type in type_list:
        L=re.findall(type+"\s*[a-z0-9_]\s\([a-z0-9_ \n\t,\r\f\v]\)\s\{",content)
        for y in L:
            y=y.replace("(","\(")
            y=y.replace(")","\)")
            x=re.search(y,content)
            first=x.span()[0]
            last=x.span()[1]
            count=1
            while(count!=0):
                if(content[last]=='{'):
                    count+=1
                if(content[last]=='}'):
                    count-=1
                last+=1
            t=content[0:first]+content[last:]
        temp = open("temp.cpp", "w")
        temp.write(t)
        temp.close()
        g = subprocess.getstatusoutput("g++ temp.cpp")
        if(g[1]==""):
            content=t
    return content

```

◆ 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
```

```python
def 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
    """
    x=np.zeros(abs(s.size-t.size))
    '''
    if(s.size>t.size):
        t=np.concatenate((x,t))
    else:
        s=np.concatenate((x,s))
    '''
    x=min(s.size,t.size)
    s=s[-x:]
    t=t[-x:]

    #s=(s-np.mean(s))/np.std(s)
    #t=(t-np.mean(t))/np.std(t)
    return 1-np.linalg.norm(s-t)/(np.linalg.norm(s)+np.linalg.norm(t))
    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 dictic
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 ir
    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.
```

```
def 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 dictionary
    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 a file.
        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/(freq of x i
        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.
    """
    similarity_matrix=np.zeros((len(list_of_paths),len(list_of_paths)))
    tf_idf_vec=[]
    for i in range(len(list_of_paths)):
        temp=[]
        for j in word_count_in_each_file[i]:
            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
        temp.sort()
        tf_idf_vec.append(temp)

    for i in range(len(list_of_paths)):
        similarity_matrix[i,i]=0
        for j in range(i+1,len(list_of_paths)):
            similarity_matrix[i,j]=similarity(np.array(tf_idf_vec[i]),np.array(tf_idf_vec[j]))
            similarity_matrix[j,i]=similarity_matrix[i,j]

    txt_file(similarity_matrix,list_of_paths,list_of_files)

```

◆ 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
```

```
def 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 """
    result=open("media/result.txt","w")
    res=""
    for i in range(len(list_of_paths)):
        for j in range(i+1,len(list_of_paths)):
            res+="similarity between "+ list_of_files[i]+" and "+list_of_files[j]+" = "+str(similarity_matrix[i][j])+"\n"
    result.write(res)
    csv_file(list_of_files,similarity_matrix)

```

◆ 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
```

```python
def 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 """
    #x=range(len(list_of_files))
    #y=range(len(list_of_files))
    #xx,yy=np.meshgrid(x,y)
    #z=similarity_matrix[xx,yy]
    #z=np.round(z*100)
    cmap = matplotlib.colors.LinearSegmentedColormap.from_list("", ["green","yellow","red"])
    fig=plt.figure()
    ax=fig.add_subplot(111)
    im=ax.matshow(similarity_matrix,cmap=cmap,vmin=0,vmax=1,origin='lower')
    for i in range(len(list_of_files)):
        for j in range(i+1,len(list_of_files)):
            ax.text(j, i, int(similarity_matrix[i,j]*100),ha="center", va="center", color="k",fontsize=50/(len(list_of_files)+1))
            ax.text(i,j, int(similarity_matrix[i,j]*100),ha="center", va="center", color="k",fontsize=50/(len(list_of_files)+1))
    fig.colorbar(im,shrink=0.5)
    ax.set_xticks(range(len(list_of_files)))
    ax.set_yticks(range(len(list_of_files)))
    ax.set_xticklabels(list_of_files,rotation=90)
    ax.set_yticklabels(list_of_files)
    random=np.random.randint(1,100)
    path='result.png'
    plt.savefig('media/'+path)

```