mesure_similarity

April 4, 2022

1 Mesure de similarité des Processes Nextflow

Une présentation des différents hypothèses de mesure de similarité entre les processes de Nextflow (présentation des idéés + fonctions pour faire fonctionner)

1.1 *Récupérer les données & Charger les bibliothèques*

[ ]:
```
import json
import pandas as pd
import math

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

[ ]:
```
samba= {}
with open('samba.json') as json_file:
    samba = json.load(json_file)

eager= {}
with open('eager.json') as json_file:
    eager = json.load(json_file)
```

1.2 *Definition des fonctions auxiliaires*

[ ]:
```
#Fonction qui retourne les éléments d'intersection entre 2 ensembles
def intersection(l1, l2):
    return list(set(l1) & set(l2))

#Fonction qui retourne les éléments d'union entre 2 ensembles
def union(l1, l2):
    return list(set(l1 + l2))
```

1.3 *Definition des fonctions de mesure de similarité entre 2 ensembles*

[ ]:
```
#Indice de Jaccard
def jaccard(t1, t2):
    num = len(intersection(t1, t2))
    if(num==0):
        return 0
    denum= len(union(t1, t2))
    return num/denum

#Indice de Sørensen-Dice
def soresen(t1, t2):
    num = 2*len(intersection(t1, t2))
    if(num==0):
        return 0
    denum= len(t1) + len(t2)
    return num/denum

#Overlap coefficient
def overlap(t1, t2):
    num = len(intersection(t1, t2))
    if(num==0):
        return 0
    denum= min(len(t1), len(t2))
    return num/denum
```

1.4 *Definition des fonctions de mesure de similarité entre 2 textes*

[ ]:
```
#Similarité cosinus
#Fonction inspiré par https://studymachinelearning.com/
 ↪cosine-similarity-text-similarity-metric/
def similarite_cosinus(doc_1, doc_2):
    data = [doc_1.lower(), doc_2.lower()]
    count_vectorizer = CountVectorizer()
    vector_matrix = count_vectorizer.fit_transform(data)

    tokens = count_vectorizer.get_feature_names_out()
    vector_matrix.toarray()

    def create_dataframe(matrix, tokens):

        doc_names = [f'doc_{i+1}' for i, _ in enumerate(matrix)]
        df = pd.DataFrame(data=matrix, index=doc_names, columns=tokens)
        return(df)

    create_dataframe(vector_matrix.toarray(),tokens)
```

```
    cosine_similarity_matrix = cosine_similarity(vector_matrix)

    Tfidf_vect = TfidfVectorizer()
    vector_matrix = Tfidf_vect.fit_transform(data)

    tokens = Tfidf_vect.get_feature_names_out()
    create_dataframe(vector_matrix.toarray(),tokens)

    cosine_similarity_matrix = cosine_similarity(vector_matrix)
    return␣
 ↪create_dataframe(cosine_similarity_matrix,['doc_1','doc_2'])['doc_1']['doc_2']

#Indice de Jaccard
def jaccard_texts(s1, s2):
    s1 , s2= s1.lower().split(), s2.lower().split()
    num = len(intersection(s1, s2))
    if(num==0):
        return 0
    denum= len(union(s1, s2))
    return num/denum
```

1.5 Hypothèse 1.1 : Identité entre les processes

Cette fonction vérifie si les deux processes donné en paramètre sont identique ou pas

[ ]:
```
def H_1_1(p1, p2):
    return p1["string_process"].lower() == p2["string_process"].lower()
```

1.5.1 Exemple :

[ ]:
```
H_1_1(eager['unzip_reference'], eager['unzip_reference'])
```

[ ]:
```
True
```

[ ]:
```
H_1_1(eager['unzip_reference'], eager['makeBWAIndex'])
```

[ ]:
```
False
```

1.6 Hypothèse 1.2 : Identité entre les scripts

Cette fonction vérifie si les script des deux processes donné en paramètre sont identique ou pas

[ ]:
```
def H_1_2(p1, p2):
    return p1["string_script"].lower() == p2["string_script"].lower()
```

1.6.1 Exemple :

```
[ ]: H_1_2(eager['unzip_reference'], eager['unzip_reference'])
```

```
[ ]: True
```

```
[ ]: H_1_2(eager['unzip_reference'], eager['makeBWAIndex'])
```

```
[ ]: False
```

1.7 Hypothèse 2.1 : Similarité entre les processes

Cette fonction donne la mesure de similarité entre les deux processes donné en paramètre avec la fonction de mesure de similarité de texte aussi donné en paramètre

```
[ ]: def H_2_1(p1, p2, similarity):
         return similarity(p1["string_process"], p2["string_process"])
```

1.7.1 Exemple :

```
[ ]: H_2_1(eager['unzip_reference'], eager['unzip_reference'], jaccard_texts)
```

```
[ ]: 1.0
```

```
[ ]: H_2_1(eager['unzip_reference'], eager['makeBWAIndex'], jaccard_texts)
```

```
[ ]: 0.15873015873015872
```

1.8 Hypothèse 2.2 : Similarité entre les scripts

Cette fonction donne la mesure de similarité entre les scripts des deux processes donné en paramètre avec la fonction de mesure de similarité de texte aussi donné en paramètre

```
[ ]: def H_2_2(p1, p2, similarity):
         return similarity(p1["string_script"], p2["string_script"])
```

1.8.1 Exemple :

```
[ ]: H_2_2(eager['unzip_reference'], eager['unzip_reference'], jaccard_texts)
```

```
[ ]: 1.0
```

```
[ ]: H_2_2(eager['unzip_reference'], eager['makeBWAIndex'], jaccard_texts)
```

```
[ ]: 0.05
```

1.9 Hypothèse 2.3 : Similarité entre les noms des processes

Cette fonction donne la mesure de similarité entre les noms des deux processes donné en paramètre avec la fonction de mesure de similarité de texte aussi donné en paramètre

[ ]:
```
def H_2_3(p1, p2, similarity):
    return similarity(p1['name_process'], p2['name_process'])
```

1.9.1 Exemple :

[ ]:
```
H_2_3(eager['unzip_reference'], eager['unzip_reference'], jaccard_texts)
```

[ ]:
```
1.0
```

[ ]:
```
H_2_3(eager['unzip_reference'], eager['makeBWAIndex'], jaccard_texts)
```

[ ]:
```
0
```

1.10 Hypothèse 3 : Identité des outils bio.tools

Cette fonction vérifie si les outils utilisés dans les deux processes sont identiques ou pas

[ ]:
```
def H_3(p1, p2):
    return p1['tools'] == p2['tools']
```

1.10.1 Exemple :

[ ]:
```
H_3(eager['unzip_reference'], eager['unzip_reference'])
```

[ ]:
```
True
```

[ ]:
```
H_3(eager['unzip_reference'], eager['makeBWAIndex'])
```

[ ]:
```
False
```

1.11 Hypothèse 4.1 : Similarité entre les outils bio.tools

Cette fonction mesure la similarité entre l'ensembles des outils utilisés pour les deux processes donné en paramètre, avec la fontion de mesure de similarité d'ensembles aussi donné en paramètre

[ ]:
```
def H_4_1(p1, p2, similarity):
    return similarity(p1['tools'], p2['tools'])
```

1.11.1 Exemple :

[ ]:
```
H_4_1(eager['bowtie2'], eager['bowtie2'], jaccard)
```

[ ]: 1.0

[ ]:
```
H_4_1(eager['bowtie2'], eager['circularmapper'], jaccard)
```

[ ]: 0.16666666666666666

1.12 Hypothèse 4.2 : Similarité entre les outils bio.tools + Nombre de lignes des scripts

Cette fonction calcule un score qui revient la moyenne entre la mesure la similarité entre l'ensembles des outils utilisés pour les deux processes donné en paramètre (fonction défini au dessus) et la mesure de similarité entre le nombre de lignes utilisé dans les scripts. La fontion de mesure de similarité d'ensembles est donné en paramètre

Pour calculer ce score, on effectue le calcule suivant :

$$score = \begin{cases} \frac{similarity(p_1^{(tools)},\ p_2^{(tools)}) + (1 - \frac{|p_1^{(len\ script)} - p_2^{(len\ script)}|}{max(p_1^{(len\ script)},\ p_2^{(len\ script)})})}{2} & si\ similarity(p_1^{(tools)},\ p_2^{(tools)}) \neq 0 \\ 0 & sinon \end{cases}$$

[ ]:
```
def score_length_script(p1, p2):
    num = abs(p1['nb_lignes_script'] - p2['nb_lignes_script'])
    denum = max(p1['nb_lignes_script'], p2['nb_lignes_script'])
    return 1 - num/denum
```

[ ]:
```
def H_4_2(p1, p2, similarity):
    if(similarity(p1['tools'], p2['tools'])!=0):
        return (similarity(p1['tools'], p2['tools']) + score_length_script(p1,␣
 ↪p2))/2
    return 0
```

1.12.1 Exemple :

[ ]:
```
H_4_2(eager['bowtie2'], eager['bowtie2'], jaccard)
```

[ ]: 1.0

[ ]:
```
H_4_2(eager['bowtie2'], eager['circularmapper'], jaccard)
```

[ ]: 0.28144654088050314

1.13 Hypothèse 5.1 : Similarité entre nb d'inputs

Fonction qui mesure la similarité entre le nombre d'input des deux processes donné comme paramètres

[ ]:
```
def H_5_1(p1, p2):
    num = abs(p1['nb_inputs'] - p2['nb_inputs'])
    denum = max(max(p1['nb_inputs'], p2['nb_inputs']), 1)
    return 1 - num/denum
```

1.13.1 Exemple :

[ ]:
```
H_5_1(eager['bowtie2'], eager['bowtie2'])
```

[ ]:
```
1.0
```

[ ]:
```
H_5_1(eager['bowtie2'], eager['circularmapper'])
```

[ ]:
```
0.5
```

1.14 Hypothèse 5.2 : Similarité entre nb d'output

Fonction qui mesure la similarité entre le nombre d'output des deux processes donné comme paramètres

[ ]:
```
def H_5_2(p1, p2):
    num = abs(p1['nb_inputs'] - p2['nb_inputs'])
    denum = max(max(p1['nb_inputs'], p2['nb_inputs']), 1)
    return 1 - num/denum
```

1.14.1 Exemple :

[ ]:
```
H_5_2(eager['bowtie2'], eager['bowtie2'])
```

[ ]:
```
1.0
```

[ ]:
```
H_5_2(eager['bowtie2'], eager['circularmapper'])
```

[ ]:
```
0.5
```

1.15 Mesure euclidienne

On défini la fonction suivante qui calcule un score de similarité "globale" entre les deux processes donné en paramètres. Pour calculer ce score, on calule d'abord la distance euclidienne normalisé associé aux hypothèses choisi, qu'on transforme ensuite en un score de similarité

[ ]:
```
def mesure_eclidienne(p1, p2):
    similarity_text = jaccard_texts
    similarity_set = jaccard
    h_2_1 = True
```

```
    h_2_2 = True
    h_2_3 = False
    h_4_1 = True
    h_4_2 = True
    h_5_1 = False
    h_5_2 = False

    num = 0
    num += h_2_1 * (1 - H_2_1(p1, p2, similarity_text))**2
    num += h_2_2 * (1 - H_2_2(p1, p2, similarity_text))**2
    num += h_2_3 * (1 - H_2_3(p1, p2, similarity_text))**2

    num+= h_4_1 * (1 - H_4_1(p1, p2, similarity_set))**2
    num+= h_4_2 * (1 - H_4_2(p1, p2, similarity_set))**2

    num+= h_5_1 *(1 - H_5_1(p1, p2))**2
    num+= h_5_2 *(1 - H_5_2(p1, p2))**2

    denum = math.sqrt(h_2_1+h_2_2+h_2_3+h_4_1+h_4_2+h_5_1+h_5_2)

    return 1 - (math.sqrt(num)/denum)
```

1.15.1 Exemple :

[ ]:
```
mesure_eclidienne(eager['bowtie2'], eager['bowtie2'])
```

[ ]:
```
1.0
```

[ ]:
```
mesure_eclidienne(eager['bowtie2'], eager['circularmapper'])
```

[ ]:
```
0.22315755404237836
```

[ ]:
```

```