README.md

Fichier config.yml pour le pipeline de traitement UDP

Structure

Le fichier doit contenir les champs suivants :

- `directories` (dictionnaire)
- `udp` (dictionnaire)
- `audio` (dictionnaire)
- `processes` (dictionnaire)

`Directories`

Cette partie doit être sous la forme :

```yaml
directories:
  temp_dir: String
  save_dir: String
```

où `temp_dir` est le dossier temporaire d'enregistrement des fichiers audio, et `save_dir` celui de la sauvegarde des fichiers audio intéressants. Ces chemins peuvent être relatifs au script Python principal ou absolu.

`udp`

Cette partie doit être sous la forme :

```yaml
udp:
  ip_address: String
  send_port: Int
  recv_port: Int
  id_length: Int
  timeout: Int
```

avec :

- `ip_address` l'adresse de réception des données UDP (le plus simple est de mettre `"0.0.0.0"`) ;
- `send_port` le port utilisé pour envoyer des messages depuis le serveur ;
- `recv_port` le port utilisé pour recevoir les données audio ;
- `id_length` la taille, en octet, de l'ID qui doit se trouver en début de chaque paquet UDP ;
- `timeout` la durée, en secondes, avant de considérer qu'un client est déconnecté si le serveur ne reçoit plus de données.

`audio`

Cette partie doit être sous la forme :

```yaml
audio:
  rate: Int
```

```
    sample_width: Int
    channels: Int
    file_duration: Float
```

avec :

- `rate` la fréquence d'échantillonnage, en Hz ;
- `sample_width` la taille, en octet, d'un échantillon audio ;
- `channels` le nombre de channels audio ;
- `file_duration` la durée, en secondes, des fichiers audio enregistrés.

On observera que le nombre d'octets nécessaires avant d'enregistrer un fichier est le produit des quatre valeurs.

processes

Cette partie est composée d'une liste de process qui doivent avoir les champs suivants :

- `name` (String)
- `position` (String)
- `type` (String)
- `model` (String)
- `config` (dictionnaire)
- `log` (null | String | CustomString)
- `actions` (dictionnaire)

name

Nom du process, nécessaire pour être référencé depuis d'autres. Il est obligatoire et doit être unique.

position

Position dans la chaîne. Champ ignoré s'il ne vaut pas `'input'`. Dans ce cas, ce process sera celui d'entrée. Il doit toujours y avoir exactement un process d'entrée dans une chaîne.

type

Type de process, il détermine le format du résultat. Les valeurs possibles pour l'instant sont :

- `'anomaly'` : renvoie un booléen qui indique si une anomalie est détectée ;
- `'classification'` : renvoie une chaîne de caractère qui contient la ou les classes détectées. Si plusieurs sont détectées, elles sont séparées par une virugle `,`.

model

Chemin du modèle, qui doit être compatible avec *tflite_runtime*. Il est soit relatif au script Python principal soit absolu.

config

Ce champ doit être sous la forme :

```
config:
    input_shape: List[Int]
    preprocess: null | String
    threshold: Float
    labels: String
```

avec :

- `input_shape` une liste d'entier représentant la forme du tenseur d'entrée du modèle ;
- `preprocess` le nom du pré-processing à appliquer aux données avant de les envoyer au modèle. Seuls deux possibilités sont acceptées pour l'instant :
  - `null` (sans guillemets) pour ne pas en avoir ;
  - `'mfcc'` pour appliquer une transformation MFCC.
- `threshold` (seulement pour les modèles de type `'anomaly'`) le seuil, entre 0 et 1, à partir duquel on considère qu'on a une anomalie ;
- `labels` (seulement pour les modèles de type `'classification'`) le chemin d'accès au fichier contenant les classes du modèle de classification. Celui-ci doit avoir une classe par ligne précédée de son indice, sous la forme `<indice>,"nom de la classe"`.
- `minimum_confidence` (seulement pour les modèles de type `'classification'`) (facultative) la valeur minimum requise, entre 0 et 1, pour que le résultat soit renvoyé. Valeur par défaut : 0.
- `count` (seulement pour les modèles de type `'classification'`) (facultative) le nombre maximum de résultat renvoyés par exécution du modèle. Il peut être combiné à `minimum_confidence` pour choisir un certain nombre de résultats au dessus d'un seuil. Valeur par défaut : le nombre de classes (attention : si ni `minimum_confidence` ni `count` ne sont spécifiés, alors toutes les classes seront renvoyées).

`log`

Indique si des logs doivent être produits sur les résultats du modèle, et si oui sous quelle forme. Ce champ est facultatif, et peut avoir plusieurs valeurs :

- `null` (sans les guillemets) : ne va pas log, équivalent à ne pas mettre le champ ;
- `<CustomString>` : ligne qui sera loggée ;
- `'default'` : équivalent à mettre la chaîne `"Process '%n' -> Result : '%r'"`

`actions`

Ce champ contient les actions à effectuer en fonction des résultats obtenus par le process. Selon sous quelles conditions on veut réaliser les actions, on les trie dans trois catégories, sous la forme :

```yaml
actions:
  on_result:
    - ...
  on_not_result:
    - ...
  always:
    - ...
```

avec :

- `on_result` : les actions dans ce champ seront associées à un résultat et exécutées si le modèle renvoie ce résultat ;
- `on_not_result` : les actions dans ce champ seront associées à un résultat et exécutées si le modèle renvoie un résultat différent ;
- `always` : les actions dans ce champ seront toujours exécutées.

Les deux premières possibilités ont une structure semblables, mais ce n'est pas le cas de la troisième.

`on_result et on_not_result`

Les actions doivent être indiquées sous cette forme :

```yaml
on_result:
  - <condition 1>:
    action: String
```

```
    arg1: value1
    ...
  - <condition 2>:
    action: String
    arg1: value1
    ...
  - ...
```

Les conditions sont un ensemble de sous-conditions séparées par des points-virgules `;` sous l'une des trois formes suivantes :

```
<Résultat>
<Résultat><Operateur><Pourcentage>%
<Résultat><Operateur><Nombre>
```

avec :

- `<Résultat>` le résultat sur lequel on effectue une condition, sous la forme d'une chaîne de caractère en minuscule et sans caractères spéciaux. Si il n'y a que ce champ, alors une comparaison de string sera effectuée;
- `<Opérateur>` un élément de la liste :
  - `>`
  - `>=`
  - `==`
  - `<=`
  - `<`
- `<Pourcentage>` un entier ou un float qui est le pourcentage de ce résultat dans les résultats totaux ;
- `<Nombre>` un entier qui est le nombre d'occurence de ce résultat dans les résultats totaux.

Note 1 : les modèles renvoient toujours des résultats sous la forme de tableaux de string, et c'est sur ces tableaux que les pourcentages et les comptes sont effectués. Si on effectue une comparaison de string, on va concaténer le tableau en séparant les éléments par des virgules `,`.

Note 2 : si on se trouve dans une partie `on_not_results` les conditions sont toutes inversées.

Note 3 : les point-virgules agissent comme des opérateurs ET.

Exemples de conditions

- `silence` : le résultat sous forme concaténée est exactement `'silence'` ;
- `false>=50%` : il y a au moins 50% de résultats `'false'` dans les résultats totaux ;
- `cri>=2;silence<5` : il y au moins 2 `'cri'` et strictement moins de 5 `'silence'` dans les résultats totaux ;
- `voix<=5;voix>=1` : il y a entre 1 et 5 (inclus) `'voix'` dans les résultats totaux ;
- `ronflements>0` : équivalent à tester si un des résultats est `'ronflement'` ;
- `toux>=100%` : équivalent à vérifier que l'intégralité des résultats est `'toux'`.

`always`

Les actions doivent être indiquées sous cette forme (l'indentation est importante) :

```
always:
  - action: String
    arg1: value1
    ...
  - action: String
    arg1: value1
    ...
```

Les types d'action

- `null` (sans guillemets) : ne fait rien. Pas d'argument supplémentaire ;
- `'save'` : enregistre les données audio. Arguments :
  - `directory` (facultatif) : dossier dans lequel enregistrer l'audio. Il peut avoir les valeurs suivantes :
    - `<CustomString>` ;
    - `'default'` : va enregistrer dans un dossier nommé selon l'ID du client dans le dossier d'enregistrement défini au début du fichier. Cette valeur est utilisée si cet argument est omis.
  - `filename` (facultatif) : nom du fichier audio (format WAV). Attention si un fichier du même nom existe dans le dossier, il sera remplacé. Il peut avoir les valeurs suivantes :
    - `<CustomString>` ;
    - `'default'` : équivaut à la CustomString `"%d-%n-%r.wav"` . Cette valeur est utilisée si cet argument est omis.
- `'log'` : log la ligne spécifiée. Arguments :
  - `line` (facultatif) : ligne à logger. Elle peut avoir les valeurs suivantes :
    - `<CustomString>` ;
    - `'default'` : équivaut à la CustomString `"Process '%n' -> '%r'"` . Cette valeur est utilisée si cet argument est omis.
- `'next'` : indique le process suivant dans la chaîne. Arguments :
  - `target` (obligatoire) : nom du process suivant ;
  - `input` (obligatoire) : indique quelles données doivent être envoyées au process suivant. Les valeurs possibles sont :
    - `'same'` : les données envoyées sont les mêmes que celles reçues ;
    - `'result'` : les données reçues sont le résultat du process en cours.
- `'output'` : renvoie les données. Cette action est ignorée si une action `'next'` est déclenchée. Pas d'arguments supplémentaires.

Exemple d'actions

```yaml
actions:
  on_result:
  - false>=100%:
      action: null
  - true>=0:
      action: next
      input: same
      target: yamnet
```

Dans ce cas, deux actions ont été déclarées lorsqu'un résultat se produit. La première, vide, est déclenchée lorsqu'on obtient que des résultats `false` . On pourrait l'omettre, mais cela permet d'indiquer clairement qu'il ne se passe rien dans ce cas. La seconde est déclenchée lorsqu'on obtient au moins un résultat `true` et consiste à continuer la chaîne avec le process `'yamnet'` , en lui envoyant les mêmes données d'entrée (`input: same`).

```yaml
actions:
  on_not_result:
    - silence>=100%:
        action: log
        line: default
  always:
    - action: output
    - action: save
      filename: '%d;%r;%a.wav'
      directory: default
```

Dans ce cas-là, trois actions ont été définies dans deux catégories. Tout d'abord, si on n'obtient pas 100% de `'silence'` (= si on a au moins une valeur différente), on va logger une ligne sous la forme `"Process '%n' -> '%r'"` . Ensuite, peu importe on le résultat, on va indiquer d'une part que ce process doit renvoyer son résultat (`action: output`) et d'autre part qu'il doit enregistrer l'audio dans le répertoire par défaut avec le nom `'%d;%r;%a.wav'` .

Types customs

CustomString

Chaîne de caractère utilisée dans un process dont certaines sous-chaînes vont être remplacées par des valeurs spéciales :

- `%n` : nom du process ;
- `%r` : résultat du process ;
- `%d` : date au format `2021_07_26-14_29_43` ;
- `%c` : ID du client lié au process ;
- `%a` : nombre aléatoire à quatre chiffres.

Exemples de fichier config.yaml

```yaml
---
directories:
  temp_dir: 'temp'
  save_dir: 'recordings'

udp:
  ip_address: "0.0.0.0"
  send_port: 7001
  recv_port: 7000
  id_length: 10
  timeout: 15

audio:
  rate: 16000 # Hz
  sample_width: 2 # bytes
  channels: 1
  file_duration: 1 # seconds

processes:
  - name: anomalies
    position: input
    type: anomaly
    model: 'models/cae16k.tflite'
    config:
      threshold: 0.01
      input_shape: [1, 16000, 1]
      preprocess: null
    log: default
    actions:
      on_result:
        - false>=100%:
            action: null
        - true>=0:
            action: next
            input: same
            target: yamnet

  - name: yamnet
    type: classification
    model: 'models/yamnet.tflite'
```

```
    config:
      labels: 'models/labels/yamnet.csv'
      input_shape: [16000]
      preprocess: null
    log: null
    actions:
      on_not_result:
        - silence>=100%:
            action: log
            line: default
      always:
        - action: output
        - action: save
          filename: '%d;%r;%a.wav'
          directory: default
```

```
---
directories:
  temp_dir: 'temp'
  save_dir: 'recordings'

udp:
  ip_address: "0.0.0.0"
  send_port: 7001
  recv_port: 7000
  id_length: 10
  timeout: 15

audio:
  rate: 16000 # Hz
  sample_width: 2 # bytes
  channels: 1
  file_duration: 1 # seconds

processes:
  - name: anomalies
    position: input
    type: anomaly
    model: 'models/mfcc16-32.tflite'
    config:
      threshold: 0.01
      input_shape: [1, 16000, 1]
      preprocess: mfcc
    log: default
    actions:
      on_result:
        - false>=100%:
            action: null
        - true>=0:
            action: next
            input: same
            target: yamnet

  - name: yamnet
    type: classification
    model: 'models/yamnet.tflite'
    config:
      labels: 'models/labels/yamnet.csv'
      input_shape: [16000]
      preprocess: null
    log: null
    actions:
      on_not_result:
        - silence>=100%:
```

```
      action: log
      line: default
always:
  - action: output
  - action: save
    filename: '%d;%r;%a.wav'
    directory: default
```

Les deux sont très semblables, à l'exception du process `anomalies` : dans le second exemple, on utilise un modèle qui nécessite un pré-processing.