Reconnaissance vocale et transcription (STT) avec Vosk

En recherchant un programme de transcription de l'audio à l'écrit (**STT** pour “Speak To Text”) je suis tombé sur un article intéressant.

Parmi les différents programmes proposés sur l'article il a fallu trier ceux qu'on pouvait piloter depuis Python ne garder que ceux qui offraient des modèles entraînés avec la langue française. On aurait très bien pu entraîner notre propre modèle si on avait eu des heures d'extraits audio sous le coude (avec transcription écrite à la main d'humain) et des flopées de GigaFlops.

Comme ce n'est pas le cas, mon choix s'est arrêté sur VOSK

Installation

Je n'ai pas réussi à installer **VOSK** avec une version de Python inférieure à la 3.8 donc il est conseillé de mettre à jour votre interpréteur Python si besoin.

Il vous faudra également **PIP**, le récupérateur de packets Python.

```
$ apt install python3-pip
```

C'est une bonne pratique de créer un environnement virtuel avant d'installer les packets nécessaires :

```
$ python3.8 -m venv env
$ source env/bin/activate
$ pip3 install vosk
```

Dans le cas où vous voulez pouvoir utiliser un micro (pour de la transcription en temps réel par exemple) il faudra également installer la librairie `sounddevice`

```
$ pip3 install sounddevice
```

Bien, mais de base VOSK est vierge de tout apprentissage. Il faut donc lui fournir un modèle pré-entraîné sur la langue de votre choix.

Utilisation de modèles pré-entraînés

Quelques modèles sont proposés à l'adresse suivante : https://alphacephei.com/vosk/models

J'ai eu l'occasion de tester deux modèles :

https://alphacephei.com/vosk/models/vosk-model-small-fr-pguyot-0.3.zip Très léger (<50 Mo) mais peu précis. Conseillé pour les machines peu performantes, téléphonie mobile ou Raspberry Pi.

https://alphacephei.com/vosk/models/vosk-model-fr-0.6-linto-2.2.0.zip Plus gros (1,5 Go) mais bien meilleures performances. C'est le modèle qui a été utilisé lors de la résidence “Artificialité Insolente” sur l'installation “Nathalie”.

Les modèles sont à décompresser dans le dossier `model`

Sur la page https://github.com/alphacep/vosk-api/tree/master/python/example on peut trouver plusieurs scripts Python pour interroger le modèle de plusieurs façons.

Transcription d'un fichier audio vers fichier texte

Utilisation avec le modèle vosk-model-en-us-0.22

Bizarrement le script plante avec ce modèle, après ce dernier message:

`LOG (VoskAPI:ReadDataFiles():model.cc:307) Loading RNNLM model from model/vosk-model-en-us-0.22/rnnlm/final.raw`

Une solution (pas super satisfaisante) consiste a supprimer le dossier `rnnlm`...

Transcription depuis un microphone

Pour utiliser le script çi-dessous, exécutez-le d'abord avec l'argument `-l` pour obtenir la liste des périphériques audio connectés à votre machine :

```
$ python3 test_microphone.py -l
```

Ensuite (ou n correspond au numéro de l'interface audio récupérée précédemment) :

```
$ python3 test_microphone.py -d n
```

test_microphone.py (cliquer pour afficher le code)

test_microphone.py

```
#!/usr/bin/env python3

import argparse
import os
import queue
import sounddevice as sd
import vosk
import sys

q = queue.Queue()


def int_or_str(text):
    """Helper function for argument parsing."""
    try:
        return int(text)
    except ValueError:
        return text

def callback(indata, frames, time, status):
    """This is called (from a separate thread) for each audio block."""
    if status:
        print(status, file=sys.stderr)
    q.put(bytes(indata))

parser = argparse.ArgumentParser(add_help=False)
parser.add_argument(
    '-l', '--list-devices', action='store_true',
    help='show list of audio devices and exit')
args, remaining = parser.parse_known_args()
if args.list_devices:
    print(sd.query_devices())
    parser.exit(0)
parser = argparse.ArgumentParser(
    description=__doc__,
    formatter_class=argparse.RawDescriptionHelpFormatter,
    parents=[parser])
parser.add_argument(
    '-f', '--filename', type=str, metavar='FILENAME',
    help='text file to store transcriptions')
parser.add_argument(
    '-m', '--model', type=str, metavar='MODEL_PATH',
    help='Path to the model')
parser.add_argument(
    '-d', '--device', type=int_or_str,
    help='input device (numeric ID or substring)')
parser.add_argument(
    '-r', '--samplerate', type=int, help='sampling rate')
args = parser.parse_args(remaining)

try:
    if args.model is None:
        args.model = "model"
    if not os.path.exists(args.model):
        print ("Please download a model for your language from https://alphacephei.com/vosk/models")
        print ("and unpack as 'model' in the current folder.")
        parser.exit(0)
    if args.samplerate is None:
        device_info = sd.query_devices(args.device, 'input')
        # soundfile expects an int, sounddevice provides a float:
```

```
            args.samplerate = int(device_info['default_samplerate'])

        model = vosk.Model(args.model)

        if args.filename:
            dump_fn = open(args.filename, "a")
        else:
            dump_fn = None

        with sd.RawInputStream(samplerate=args.samplerate, blocksize = 1024, device=args.device, dtype='int16',
                               channels=1, latency='high', callback=callback):
                print('#' * 80)
                print('Press Ctrl+C to stop the recording')
                print('#' * 80)

                rec = vosk.KaldiRecognizer(model, args.samplerate)
                while True:
                    data = q.get()
                    if rec.AcceptWaveform(data):
                        r = eval(rec.Result())
                        t = r["text"]
                        if t:
                            print(t)
                            if dump_fn is not None and len(t) > 5:
                                dump_fn.write(t+'\n')

    except KeyboardInterrupt:
        print('\nDone')
        parser.exit(0)
    except Exception as e:
        parser.exit(type(e).__name__ + ': ' + str(e))
```

Entraînement d'un nouveau modèle linguistique et acoustique

Tutoriaux

https://towardsdatascience.com/how-to-start-with-kaldi-and-speech-recognition-a9b7670ffff6

http://kaldi-asr.org/doc/kaldi_for_dummies.html

http://kaldi-asr.org/doc/data_prep.html

https://www.eleanorchodroff.com/tutorial/kaldi/training-acoustic-models.html

https://web.stanford.edu/class/cs224s/assignments/a3/

Installation de Kaldi et initialisation du projet

Kaldi est un kit d'outils pour la création de modèles linguistiques. Les modèles sont ensuite utilisés par VOSK pour faciliter leur utilisation pour la reconnaissance vocale.

Les instruction pour l'installation sont dans le fichier `tools/INSTALL`

Cloner le répo de Kaldi : https://github.com/kaldi-asr/kaldi

```
$ git clone https://github.com/kaldi-asr/kaldi
```

Vérifier les dépendances :

```
$ ./tools/extras/check_dependencies.sh
```

Installation des outils nécessaires à Kaldi :

```
$ cd tools
$ make
```

Installation de Intel Math Kernel Library (optimisation des opération d'algèbre linéaire) :

```
$ sudo ./tools/extra/install_mkl.sh
```

Installation de SRILM (outil pour la création de modèles de langages)

```
$ ./tools/install_srilm.sh
```

Installation de kaldi :

```
$ cd src
$ ./configure
$ make -j clean depend
$ make -j <NCPU>    # où <NCPU> est le nombre de coeurs de processeurs à utiliser pour la compilation
```

Créer un nouveau dossier pour le projet dans le dossier `egs` (`mycorpus` dans l'exemple ci-dessous)

Recréer l'arborescence ci-dessous à partir du dossier `mycorpus` (*les lignes rouges pointillées sont des liens symboliques*) :



```
cd mycorpus
ln -s ../wsj/s5/steps .
ln -s ../wsj/s5/utils .
ln -s ../../src .
cp ../wsj/s5/path.sh .
```

Traitement des fichiers son

Conversion en wav mono 16 bits et avec une fréquence d'échantillonnage de 16000 Hz

```
$ ffmpeg -i in.mp3 -acodec pcm_s16le -ac 1 -ar 16000 out.wav
```

Détection des silences et des non silences avec Python

https://librosa.org/

Création des fichiers du dossier 'data/train'

Les fichiers essentiels à la création d'un modèle kaldi sont : `wav.scp`, `utt2spk`, `spk2utt` et `text`.

Fichier 'text'

```
s5# head -3 data/train/text
sw02001-A_000098-001156 HI UM YEAH I'D LIKE TO TALK ABOUT HOW YOU DRESS FOR WORK AND
sw02001-A_001980-002131 UM-HUM
sw02001-A_002736-002893 AND IS
```

The first element on each line is the `utterance-id`, which is an arbitrary text string, but if you have speaker information in your setup, you should make the `speaker-id` a prefix of the utterance id; this is important for reasons relating to the sorting of these files. The rest of the line is the transcription of each sentence. You don't have to make sure that all words in this file are in your vocabulary; out of vocabulary words will get mapped to a word specified in the file data/lang/oov.txt.

It needs to be the case that when you sort both the `utt2spk` and `spk2utt` files, the orders "agree", e.g. the list of speaker-ids extracted from the `utt2spk` file is the same as the string sorted order. The easiest way to make this happen is to make the speaker-ids a prefix of the utter. Although, in this particular example we have used an underscore to separate the "speaker" and "utterance" parts of the utterance-id, in general it is probably safer to use a dash ("-"). This is because it has a lower ASCII value; if the speaker-ids vary in length, in certain cases the speaker-ids and their corresponding utterance ids can end up being sorted in different orders when using the standard "C"-style ordering on strings, which will lead to a crash. Another important file is `wav.scp`. In the Switchboard example,

Fichier 'wav.scp'

```
s5# head -3 data/train/wav.scp
sw02001-A /home/dpovey/kaldi-trunk/tools/sph2pipe_v2.5/sph2pipe -f wav -p -c 1 /export/corpora3/LDC/LDC97S62/swb1/sw02001.sph |
sw02001-B /home/dpovey/kaldi-trunk/tools/sph2pipe_v2.5/sph2pipe -f wav -p -c 2 /export/corpora3/LDC/LDC97S62/swb1/sw02001.sph |
```

The format of this file is

`<recording-id> <extended-filename>`

where the "extended-filename" may be an actual filename, or as in this case, a command that extracts a wav-format file. The pipe symbol on the end of the extended-filename specifies that it is to be interpreted as a pipe. We will explain what `recording-id` is below, but we would first like to point out that if the `segments` file does not exist, the first token on each

line of wav.scp file is just the utterance id. The files in wav.scp must be single-channel (mono); if the underlying wav files have multiple channels, then a sox command must be used in the wav.scp to extract a particular channel.

Fichier 'segments'

In the Switchboard setup we have the segments file, so we'll discuss this next.

s5# head -3 data/train/segments

```
sw02001-A_000098-001156 sw02001-A 0.98 11.56
sw02001-A_001980-002131 sw02001-A 19.8 21.31
sw02001-A_002736-002893 sw02001-A 27.36 28.93
```

The format of the segments file is:

```
<utterance-id> <recording-id> <segment-begin> <segment-end>
```

where the segment-begin and segment-end are measured in seconds. These specify time offsets into a recording. The recording-id is the same identifier as is used in the wav.scp file– again, this is an arbitrary identifier that you can choose.

Fichier 'utt2spk'

The last file you need to create yourself is the utt2spk file. This says, for each utterance, which speaker spoke it.

```
s5# head -3 data/train/utt2spk
sw02001-A_000098-001156 2001-A
sw02001-A_001980-002131 2001-A
sw02001-A_002736-002893 2001-A
```

The format is

```
<utterance-id> <speaker-id>
```

Note that the speaker-ids don't need to correspond in any very accurate sense to the names of actual speakers– they simply need to represent a reasonable guess. In this case we assume each conversation side (each side of the telephone conversation) corresponds to a single speaker. This is not entirely true – sometimes one person may hand the phone to another person, or the same person may be speaking in multiple calls – but it's good enough for our purposes. If you have no information at all about the speaker identities, you can just make the speaker-ids the same as the utterance-ids , so the format of the file would be just <utterance-id> <utterance-id>. We have made the previous sentence bold because we have encountered people creating a “global” speaker-id. This is a bad idea because it makes cepstral mean normalization ineffective in training (since it's applied globally), and because it will create problems when you use utils/split_data_dir.sh to split your data into pieces.

Fichier 'reco2file_and_channel' (optionnel)

The file reco2file_and_channel is only used when scoring (measuring error rates) with NIST's sclite tool:

```
s5# head -3 data/train/reco2file_and_channel
sw02001-A sw02001 A
sw02001-B sw02001 B
sw02005-A sw02005 A
```

The format is:

```
<recording-id> <filename> <recording-side (A or B)>
```

The filename is typically the name of the .sph file, without the suffix, but in general it's whatever identifier you have in your stm file. The recording side is a concept that relates to telephone conversations where there are two channels, and if not, it's probably safe to use “A”. If you don't have an stm file or you have no idea what this is all about, then you don't need the “reco2file_and_channel” file.

Fichier 'spk2gender' (optionnel)

There is another file that exists in some setups; it is used only occasionally and not in the Kaldi system build. We show what it looks like in the Resource Management (RM) setup:

```
s5# head -3 ../../rm/s5/data/train/spk2gender
adg0 f
ahh0 m
ajp0 m
```

This file maps from speaker-id to either “m” or “f” depending on the speaker gender.

Une fois tous les fichiers crées, lancer la commande :

```
$ utils/fix_data_dir.sh data/train
```

Fichiers du dossier 'data/local/lang'

Fichiers de donnés liés au langage. On doit fournir : `lexicon.txt`, `nonsilence_phones.txt`, `optional_silence.txt`, `silence_phones.txt` et `extra_questions.txt` (optionnel).

Fichier 'lexicon.txt'

Le fichier `lexicon.txt` contient la liste de chaque mot du corpus (en majuscule), suivi de sa prononciation phonétique.

Exemple:

```
WORD W ER D
LEXICON L EH K S IH K AH N
```

The pronunciation alphabet must be based on the same phonemes you wish to use for your acoustic models. You must also include lexical entries for each “silence” or “out of vocabulary” phone model you wish to train.

https://en.wikipedia.org/wiki/ARPABET

On peut s'aider de cet outil en ligne pour la construction du fichier `lexicon.txt` (mais se limite à la prononciation anglaise) : http://www.speech.cs.cmu.edu/tools/lextool.html

Liens pour la phonétique de la langue bretonne :

- https://en.wikipedia.org/wiki/Help:IPA/Breton
- https://en.m.wikiversity.org/wiki/Breton/Breton_pronunciation
- http://www.kervarker.org/en/courseintro_03_noid.html
- https://en.wikipedia.org/wiki/Breton_language#Geographic_distribution_and_dialects

Fichier 'nonsilence_phones.txt'

Liste de tous les phonèmes utilisés dans notre corpus

Une fois les fichiers crées, lancer la commande :

```
$ utils/prepare_lang.sh data/local/dict '<UNK>' data/local/lang data/lang
```

A propos des mots inconnus

This is an explanation of how Kaldi deals with unknown words (words not in the vocabulary); we are putting it on the “data preparation” page for lack of a more obvious location.

In many setups, <unk> or something similar will be present in the LM as long as the data that you used to train the LM had words that were not in the vocabulary you used to train the LM, because language modeling toolkits tend to map those all to a single special world, usually called <unk> or <UNK>. You can look at the arpa file to figure out what it's called; it will

usually be one of those two.

During training, if there are words in the text file in your data directory that are not in the words.txt in the lang directory that you are using, Kaldi will map them to a special word that's specified in the lang directory in the file data/lang/oov.txt; it will usually be either <unk>, <UNK> or maybe <SPOKEN_NOISE>. This word will have been chosen by the user (i.e., you), and supplied to prepare_lang.sh as a command-line argument. If this word has nonzero probability in the language model (which you can test by looking at the arpa file), then it will be possible for Kaldi to recognize this word in test time. This will often be the case if you call this word <unk>, because as we mentioned above, language modeling toolkits will often use this spelling for `unknown word` (which is a special word that all out-of-vocabulary words get mapped to). Decoding output will always be limited to the intersection of the words in the language model with the words in the lexicon.txt (or whatever file format you supplied the lexicon in, e.g. lexicop.txt); these words will all be present in the words.txt in your lang directory. So if Kaldi's "unknown word" doesn't match the LM's "unknown word", you will simply never decode this word. In any case, even when allowed to be decoded, this word typically won't be output very often and in practice it doesn't tend to have much impact on WERs.

Of course a single phone isn't a very good, or accurate, model of OOV words. In some Kaldi setups we have example scripts with names local/run_unk_model.sh: e.g., see the file tedlium/s5_r2/local/run_unk_model.sh. These scripts replace the unk phone with a phone-level LM on phones. They make it possible to get access to the sequence of phones in a hypothesized unknown word. Note: unknown words should be considered an "advanced topic" in speech recognition and we discourage beginners from looking into this topic too closely.

Modèles basés sur un réseau neuronal profond

- https://kaldi-asr.org/doc/dnn.html
- http://www.cs.cmu.edu/~ymiao/kaldipdnn.html

Utilisation du modèle post entraînement

https://medium.com/@nithinraok_/decoding-an-audio-file-using-a-pre-trained-model-with-kaldi-c1d7d2fe3dc5

Entrainement d'un modèle compatible VOSK

Un tutorial concis et complet que j'aurais aimé découvrir plus tôt : https://github.com/matteo-39/vosk-build-model

L'utilisation de VOSK simplifie énormément le décodage d'un fichier son avec un modèle Kaldi. Il faut toutefois noter que VOSK n'accepte que les modèles d'un format particulier.

D'après la page https://alphacephei.com/vosk/models il est recommandé d'utiliser la recette `mini-librispeech`, présent sous le dossier "egs" du dossier d'installation de kaldi. Il faudra modifier les scripts `cmd.sh` et `run.sh` pour les adapter à votre configuration et à vos données.

Il faudra également remplacer le dernier script exécuté par `run.sh` : `local/chain2/run_tdnn.sh`, par le script fourni sur la page de VOSK : https://github.com/kaldi-asr/kaldi/blob/master/egs/mini_librispeech/s5/local/chain/tuning/run_tdnn_1j.sh. Ce script nécessitera également des modification pour l'adapter à votre situation. (réduction de nombre de jobs en parallèle (option $nj) et désactivation de l'utilisation du GPU dans mon cas)

Troubleshooting

- Ne pas laisser de caractères spéciaux dans le nom des fichiers de données du corpus, ni d'espaces dans le noms des dossiers (un '&' dans le nom d'une archive sonore peut faire planter la phase d'entrainement du RN)

Article extrait de : http://www.lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse : **http://www.lesporteslogiques.net/wiki/ressource/logiciel/vosk/start**
Article mis à jour: **2022/03/06 23:06**