Laboratorium 1

Pakiet SpeechRecognition dla Pythona

Instalacja i działanie były testowane dla systemu operacyjnego Linux (NAME="Linux Mint", VERSION="19.2 (Tina)", ID=linuxmint, ID_LIKE=ubuntu) w środowisku wirtualnym Anaconda (conda 4.7.10) postawionego jako maszyna wirtualna (VMware) w Windows 10.

Utworzenie i aktywacja środowiska wirtualnego

```
$ conda create python=3.6 -n VR
$ conda activate VR
```

Instalacja pakietu SpeechRecognition w środowisku wirtualnym

```
(VR)$ pip install SpeechRecognition
```

Test rozpoznawania mowy z pliku wav

```
(VR)$ python
>>> import speech_recognition as sr
>>> sr.__version__
'3.8.1'
>>> r = sr.Recognizer()
>>> harvard = sr.AudioFile('harvard.wav')
>>> with harvard as source:
...     audio = r.record(source)
...
>>> r.recognize_google(audio)
'the stale smell of old beer lingers it takes heat to bring out the odor a cold dip
restores health and zest a salt pickle taste fine with ham tacos al Pastore are my
favorite a zestful food is be hot cross bun'
```

Test rozpoznawania mowy rejestrowanej przez mikrofon

Aby rozpoznawać mowę rejestrowaną przez mikrofon niezbędne jest doinstalowanie dodatkowych pakietów dla pythona. Po wyjściu z pythona i ze środowiska wirtualnego (niekoniecznie) wykonujemy polecenie:

```
$ sudo apt-get install python-pyaudio python3-pyaudio
```

Pakiety pyaudio i portaudio należy również zainstalować w środowisku wirtualnym:

```
$ conda activate VR
(VR) $ conda install nwani::portaudio nwani::pyaudio
```

UWAGA:
W zasadzie w środowisku conda powinna wystarczyć standardowa instalacja pakietów pyaudio i portaudio tzn.:

```
pip install pyaudio
```

lub

```
conda install pyaudio
```

ale wg. stanu na 07.10.2019 instalacje te mają bugi i działają instalacje z repozytorium nwani. Bugi są w albo bibliotece _portaudio*.so albo w bibliotece libportaudio*.so,

```
(VR)$ python
>>> import pyaudio
>>> pyaudio.pa.__file__
'/home/zbislaw/.conda/envs/VR/lib/python3.6/site-packages/_portaudio.cpython-36m-
x86_64-linux-gnu.so'
```

Po zainstalowaniu bibliotek pyaudio i portaudio rozpoznawanie mowy rejestrowanej mikrofonem powinno już działać:

```
$ python -m speech_recognition
        A moment of silence, please...
        Set minimum energy threshold to 1007.5191959469204
        Say something!
        Got it! Now to recognize it...
        You said speech recognition works
        Say something!
```

Powinien również działać następujący skrypt, rozpoznający mowę z mikrofonu i z pliku audio:

```
import speech_recognition as sr
r = sr.Recognizer()
mic = sr.Microphone(device_index=0)
with mic as source:
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)
v = r.recognize_google(audio)
print(v)

hello = sr.AudioFile('hello.wav')
with hello as source1:
        audio1 = r.record(source1)
u = r.recognize_google(audio1)
print(u)
```

Po zapisaniu skryptu w pliku listen.py, należy go uruchomić z linii poleceń:

```
$ python listen.py
```

Listę aktywnych mikrofonów można otrzymać, wykonując polecenie:

```
>>> sr.Microphone.list_microphone_names()
```

Mikrofon do odsłuchiwania wybiera się, ustalając w oparciu o ww. listę odpowiedni indeks w linii kodu:

```
>>> mic = sr.Microphone(device_index=0)
```

Dokumentacja pakietu SpeechRecognition znajduje się pod adresem:
https://github.com/Uberi/speech_recognition/blob/master/reference/library-reference.rst
https://pypi.org/project/SpeechRecognition/1.3.0/

Skrypt do przetestowania

```
import random
import time
import speech_recognition as sr


def recognize_speech_from_mic(recognizer, microphone):
    """Transcribe speech from recorded from `microphone`.

    Returns a dictionary with three keys:
    "success": a boolean indicating whether or not the API request was
               successful
    "error":   `None` if no error occured, otherwise a string containing
               an error message if the API could not be reached or
               speech was unrecognizable
    "transcription": `None` if speech could not be transcribed,
               otherwise a string containing the transcribed text
    """
    # check that recognizer and microphone arguments are appropriate type
    if not isinstance(recognizer, sr.Recognizer):
        raise TypeError("`recognizer` must be `Recognizer` instance")

    if not isinstance(microphone, sr.Microphone):
        raise TypeError("`microphone` must be `Microphone` instance")
    # adjust the recognizer sensitivity to ambient noise and record audio
    # from the microphone
    with microphone as source:
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)

    # set up the response object
    response = {
        "success": True,
        "error": None,
        "transcription": None
    }

    # try recognizing the speech in the recording
    # if a RequestError or UnknownValueError exception is caught,
    #     update the response object accordingly
    try:
        response["transcription"] = recognizer.recognize_google(audio)
    except sr.RequestError:
        # API was unreachable or unresponsive
        response["success"] = False
        response["error"] = "API unavailable"
    except sr.UnknownValueError:
        # speech was unintelligible
        response["error"] = "Unable to recognize speech"

    return response


if __name__ == "__main__":
    # set the list of words, maxnumber of guesses, and prompt limit
    WORDS = ["apple", "banana", "grape", "orange", "mango", "lemon"]
    NUM_GUESSES = 3
    PROMPT_LIMIT = 5

    # create recognizer and mic instances
    recognizer = sr.Recognizer()
    microphone = sr.Microphone()

    # get a random word from the list
    word = random.choice(WORDS)

    # format the instructions string
    instructions = (
        "I'm thinking of one of these words:\n"
        "{words}\n"
        "You have {n} tries to guess which one.\n"
    ).format(words=', '.join(WORDS), n=NUM_GUESSES)

    # show instructions and wait 3 seconds before starting the game
    print(instructions)
    time.sleep(3)

    for i in range(NUM_GUESSES):
        # get the guess from the user
        # if a transcription is returned, break out of the loop and
        #     continue
        # if no transcription returned and API request failed, break
        #     loop and continue
        # if API request succeeded but no transcription was returned,
        #     re-prompt the user to say their guess again. Do this up
        #     to PROMPT_LIMIT times
        for j in range(PROMPT_LIMIT):
            print('Guess {}. Speak!'.format(i+1))
            guess = recognize_speech_from_mic(recognizer, microphone)
            if guess["transcription"]:
                break
            if not guess["success"]:
                break
            print("I didn't catch that. What did you say?\n")

        # if there was an error, stop the game
        if guess["error"]:
            print("ERROR: {}".format(guess["error"]))
            break

        # show the user the transcription
        print("You said: {}".format(guess["transcription"]))

        # determine if guess is correct and if any attempts remain
        guess_is_correct = guess["transcription"].lower() == word.lower()
        user_has_more_attempts = i < NUM_GUESSES - 1

        # determine if the user has won the game
        # if not, repeat the loop if user has more attempts
        # if no attempts left, the user loses the game
        if guess_is_correct:
            print("Correct! You win!".format(word))
            break
        elif user_has_more_attempts:
            print("Incorrect. Try again.\n")
        else:
            print("Sorry, you lose!\nI was thinking of '{}'.".format(word))
            break
```

Zadanie
Zaprojektować system sterujący, wykorzystujący API Google
(np. włączanie/wyłączanie diod w Raspberry Pi)

Laboratorium 2
Prezentacja projektów