Examples of SingularityNET AI Service Assemblages

Nil Geisweiller

November 4, 2022

1 Introduction

This document goes over a list of potential AI service assemblages that could be used as test cases for the next phase. Nearly all AI services involved already exist on the SingularityNET platform.

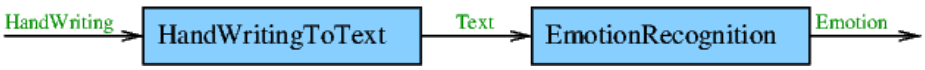
2 AI Services Assemblages

Each assemblage is shown with:

1. a flowchart, arrows representing the flow of information between AI services represented as blue boxes;
2. a type signature using types provided by an imaginary possible future version of the AI-DSL, for now expressed in Idris;
3. optionally an implementation of such assemblage, again expressed in Idris.

2.1 Recognize Emotion of Hand Written Text

Let's begin with an assemblage of two services to recognize the emotion of hand written text. Given an image of handwritten text, it first turns it into a string of text, then recognize an emotion from that.




```
handwritingEmotionRecognition : HandWriting -> Emotion
handwritingEmotionRecognition = emotionRecognition . handWritingToText
```

The type `HandWriting` could be specialized type of image and `Emotion` could be a category, or possibly categorical distribution, of emotions. The composition operator `.` can be used for the implementation given the simplicity of the assemblage.

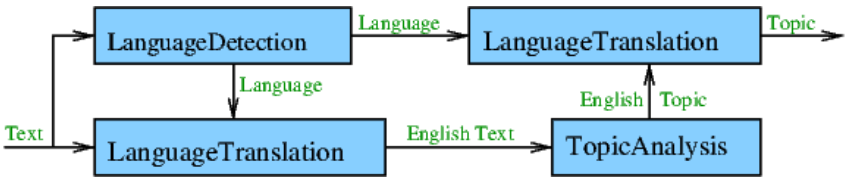
2.2 Topic Analysis of Speech


```
speechTopicAnalysis : Speech -> Topic
speechTopicAnalysis = topicAnalysis . speechToText
```

`Speech` could be a special type of sound and `Topic` a special type of text, or maybe category, or categorical distribution, of topics.

2.3 Topic Analysis of Text from Any Language

This assemblage is simply doing topic analysis but for any language by translating the input text into English and the output topic back into the original language.




```
anyLanguageTopicAnalyzer : Text -> (l : Language ** Topic l)
anyLanguageTopicAnalyzer = ?h
anyLanguageTopicAnalyzer' : Text l -> Topic l
anyLanguageTopicAnalyzer' = ?h
```

We offer two definitions:

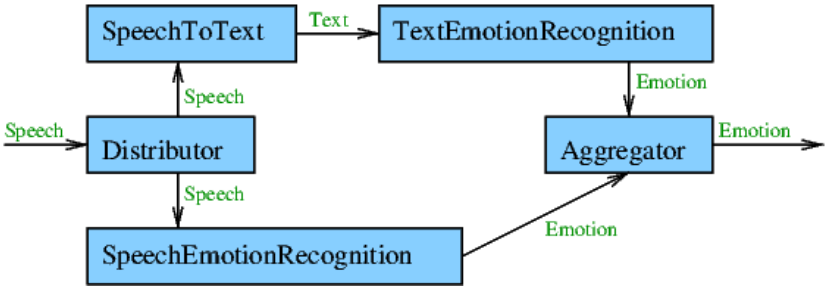
1. `anyLanguageTopicAnalyzer`: corresponding exactly to its flowchart;
2. `anyLanguageTopicAnalyzer'`: corresponding to that flowchart without language detection.

There is an interesting use of dependent types here. In the first definition, `anyLanguageTopicAnalyzer`, a dependent pair is used to return the recognized language then passed to `Topic`, here a parameterized type, to specify its language. In the second definition, `anyLanguageTopicAnalyzer'`, a type variable is used to express the guaranty that the language of the output topic is the same as the one of the text.

2.4 Recognize Emotion from Speech

That assemblage combines a Speech Emotion Recognition service with a sub-assemblage of speech-to-text and text-emotion-recognition services. The idea is

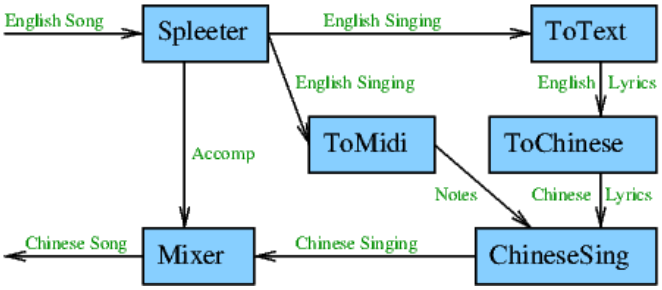
that such assemblage would improve the performance of emotion recognition by combining different services and aggregating their results.




```
speechEmotionRecognizer : Speech -> Emotion
speechEmotionRecognizer = ?h
```

2.5 Turn an English Song into a Chinese Song

This assemblage aims at turning an English song into the same song but sang in Chinese.




```
englishSongToChineseSong : Song English -> Song Chinese
englishSongToChineseSong x = let (enSing, accomp) = Spleeter x
                                 enLyrics = toText enSing
                                 chLyrics = toChinese enLyrics
                                 notes = toMidi enSing
                                 chSing = chineseSing notes chLyrics
                             in accomp + chSing
Spleeter : Song l -> (Singing l, Intrumental)
Spleeter song = ?h
```

Note the use of the parameterized type `Song` that could be a specialized type of audio, a song in a given language. The implementation involves the `let` construct. There are certainly ways to replace that by something purely compositional, something we may want to consider to make the AI-DSL language more composition friendly. In addition we provide an example of type signature of the AI service `Spleeter`, using dependent types again.

All services involved in that assemblage with the exception of `toMidi` exist on SingularityNET. To create `toMidi` we have however found a number of open source projects that could potentially be used

- https://basicpitch.spotify.com (Apache license);
- https://github.com/NFJones/audio-to-midi (MIT license);
- https://aubio.org/ (GPL license);
- https://wave2mid.sourceforge.net/index-en.html (unknown license);
- https://github.com/justinsalamon/audio_to_midi_melodia (unknown license);
- https://github.com/bill317996/Audio-to-midi (MIT license);
- https://github.com/emredjan/audio-to-midi (unknown license)
- https://github.com/sbaeunker/audioToMidiConverter (GPL license)

3 conclusion

The type signatures are fairly simple in these examples, they could of course be made more sophisticated, or additional properties could be added. These aspects are important and have been previously explored to some degree. But they will likely not be the main focus of the next phase, so are left out or simplified for now. The purpose of the next phase will be to build a rudimentary, yet working from end-to-end, AI-DSL prototype.