Reconhecimento de Voz para Sistemas de Ensino

Catarina Machado[A81047], Filipe Monteiro[A80229], and João Vilaça[A82339]

Departamento de Informática - Universidade do Minho

Resumo Os sistemas de ensino eletrónicos têm surgido naturalmente com a crescente digitalização da sociedade, a que temos assistido desde meados do século XX, com o início da adoção e proliferação dos computadores digitais. Este processo começou pela digitalização dos materiais de apoio às aulas, em plataformas como a Blackboard, onde apenas vários anos mais tarde, foram introduzidas as aulas remotas. Apesar da lenta evolução, assistimos atualmente a uma revolução deste paradigma, com o aparecimento de assistentes inteligentes de educação, como o Leonardo, que autonomamente guiam os alunos pelo processo de aprendizagem. Neste contexto, surge a necessidade da introdução de ferramentas de reconhecimento de voz, que se traduzem no aumento da usabilidade do serviço, encorajando interações mais naturais e fluídas, humanizando o sistema.

Keywords: Reconhecimento de Voz · Sistemas de Ensino e de Avaliação de Conhecimento · Interação Humano-Computador · Inteligência Artificial · Google Speech-to-Text

1 Introdução

A formação é um instrumento essencial para a promoção de características diferenciadoras em todos os seres humanos, como a capacidade de adaptação e a conquista de novas competências. Agregado à crescente procura por sistemas que de certa forma consigam reproduzir comportamentos humanos, surgem os Sistemas de Ensino e de Avaliação de Conhecimento, que têm como principal objetivo dotar os seus utilizadores de novos conhecimentos, preferencialmente de uma forma intuitiva e simples. Para além disso, estes sistemas complementam esta formação com uma verificação dos conhecimentos adquiridos.

Devido à qualidade que se exige destes Sistemas, verifica-se uma enorme procura por torná-los o mais efetivos possíveis na comunicação utilizador - sistema. O Reconhecimento de Voz vem desta forma colmatar este problema, melhorando realmente a experiência de utilização dos mesmos.

O facto dos utilizadores poderem utilizar a voz para instruções breves ou para a redação de pequenos textos de opinião pode facilitar a utilização da plataforma em vários momentos do nosso quotidiano por tornar o sistema *hands-free*, fazendo assim com que poupemos tempo e, consecutivamente, aumentando a produtividade.

Para além da funcionalidade *voice to text*, a funcionalidade *text to voice* contribui ainda para uma consolidação da aprendizagem caso, por exemplo, a

matéria em questão seja relacionada com idiomas e contribui também para o aumento da acessibilidade da plataforma, até para alunos com necessidades educativas especiais.

2 Trabalho Relacionado

Em termos de trabalho anterior desenvolvido nesta área, numa primeira fase, foi sobretudo essencial estudar projetos que usem sistemas de reconhecimento de voz sejam eles ou não, diretamente relacionados com o processo educativo, que permitam fazer uma introspeção dos seus modelos de tomada de decisão, e dos algoritmos que usam não só para a recolha e processamento de voz mas também em relação à extração de informação desses excertos.

O projeto Jarvis[2] permitiu perceber quais os modelos de decisão normalmente utilizados nesta área, sendo que disponibilizavam um diagrama do mesmo, o que facilitou a sua análise mais extensiva, possibilitando também a inferência de algumas conclusões sobre os estados possíveis do serviço e qual o melhor controlo de fluxo aplicacional a implementar.

Também neste projeto, é de destacar a metodologia de cálculo da ação a executar a partir da análise do que o utilizador pediu.

Após esta análise, procurou-se perceber como, em outras tecnologias educativas com recurso a interação por voz e a inteligência artifical existentes no mercado, é conduzido todo processo de aprendizagem, nomeadamente em termos de controlo de fluxo, em termos de perguntas-respostas (sejam elas aluno -> assistente ou assistente -> aluno) e em relação à avaliação de respostas em função da natural variabilidade semântica.

Tecnologias como Edwin[5] e Cognii[6] são líderes de mercado na área dos assistentes de educação porque se distinguem das competição por serem capazes de receber e processar respostas abertas, que incentivam o pensamento e resolução críticas dos problemas, humanizando em simultâneo o sistema. Entre outras funcionalidades, estes serviços são capazes de oferecer aos utilizadores feedback em tempo real da sua prestação e de personalizar o comportamento do assistente dependendo de cada aluno.

3 O Sistema de Processamento de Voz

O modelo *LHA* foi construído com uma mentalidade de modularidade e facilidade na análise e correcção de problemas dentro deste. Como tal, foi dividido em 2 grandes partes: um processador de áudio, responsável pela tomada de decisão das ações a tomar para determinado *input* e um servidor *web* capaz de receber este *input*, passá-lo para o processador de áudio, e por fim devolver determinada resposta. Este último foi apenas necessário para assegurar a integração para com os outros módulos no qual estará integrado no projecto final.

Figura 1. *Flow* principal do programa.

O utilizador ao seleccionar a opção de voz, o microfone entrará em escuta gravando até ouvir silêncio, enviando para o servidor *web*. Neste, será processado com ajuda do *SpeechRecognizer* e *ActionManager*, devolvendo no fim o tipo de ação que a plataforma *Leonardo* terá de tomar.

Para manter a coesão na divisão das tarefas de cada componente, separámos estes nas seguintes directorias: **Actions**, **ActionManager**, **Speech** e **Web**.

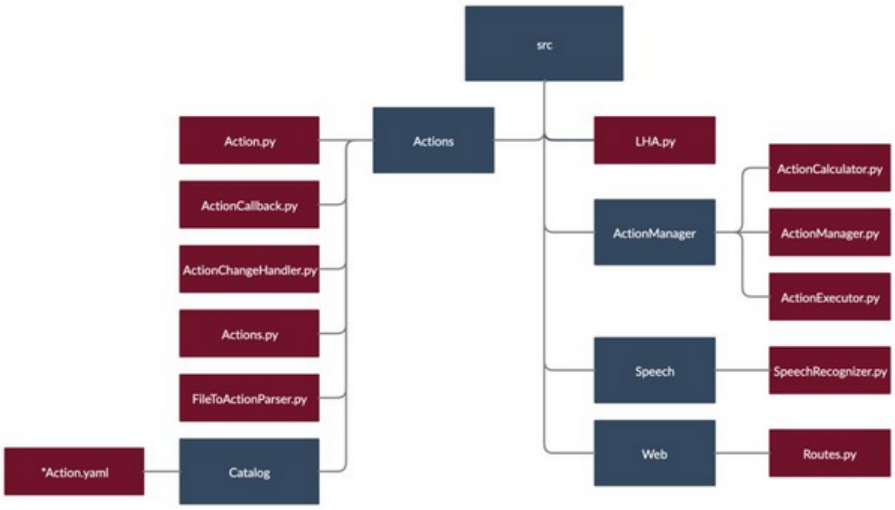

Figura 2. Estrutura de ficheiros do LHA.

Gerado a partir do seguinte diagrama de classes:

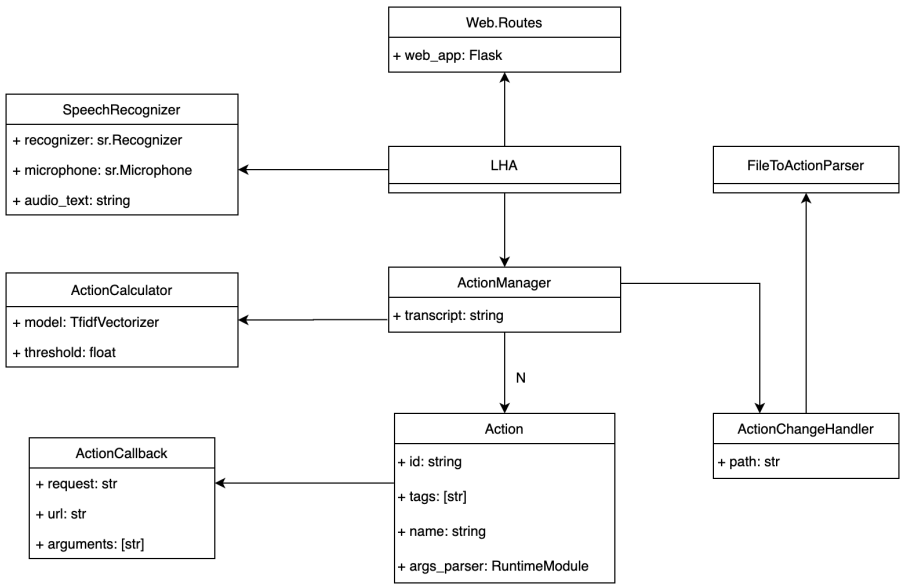

Figura 3. Diagrama de classes do LHA.

3.1 Actions

Como o nome indica aqui encontram-se componentes relacionadas diretamente com as ações. *Action.py* define uma estrutura para representar uma ação, sendo o resultado originado através de *FileToActionParser.py* que faz o *parsing* dos ficheiros contendo ações para o sistema. Por questões de otimização de recursos e facilidade na escrita de ações, qualquer mudança na pasta **actions_catalog**, onde se encontram todas as ações disponíveis, em diferentes pastas correspondentes ao idioma, será refletida diretamente no sistema recorrendo ao *ActionChangeHandler.py*, que escuta constantemente mudanças neste diretório. Por fim, *ActionCallback.py* e *Actions.py* são estruturas auxiliares para as ações, sendo a primeira a representação do resultado de uma ação e a última o dicionário completo de ações no sistema, separadas por idioma.

Definição de uma ação Para a criação de ações, resolvemos basear-nos em ficheiros **YAML**, pois é fácil de escrever e *python* já possuía várias bibliotecas capazes de executar *parsing*. Por isso, foi adotada a seguinte estrutura:

```
tags:
  - keyword_1
  - keyword_2
  - keyword_3
  (...)

callback:
  param_1: value_1
  param_2: value_2
  param_3: value_3
  param_4: value_4
  (...)
  arguments:
    - arg1
    - arg2
    (...)

parse_callback_arguments_from_transcript: |
    (Python function to parse any argument from speech)
```

Figura 4. Estrutura das ações, escritas em formato *YAML*.

- **tags**: lista de palavras-chave para reconhecimento da ação. Não pode haver ações com exatamente as mesmas *tags*, senão o modelo não será capaz de decidir a correta. Cada ação tem de conter no mínimo uma palavra-chave única ou uma combinação única de palavras-chaves;
- **callback**: parâmetros necessários por parte de módulos que usam estas ações, seja para atualizações de *front-end* ou outros, como resposta de retorno por parte do servidor. Aqui pode existir um campo **arguments** responsável por indicar o valor de determinada argumento usado no áudio - por exemplo, se pedirmos para selecionar a opção 4, ele terá de perceber qual a opção a escolher, retornando este *4* e indicando a que este corresponde -, sendo que, para existir, precisa do seguinte campo;
- **parse_callback_arguments_from_transcript**: função *python* responsável por fazer *parsing* do texto de *input*, para retirar argumentos (novamente, no caso anterior, seria a "(...) opção 4").

Dentro da pasta do catálogo de ações, estas serão colocadas de acordo com o idioma em questão, em diferentes pastas contendo como nome o código de cada idioma - por exemplo, português de Portugal seria *pt-PT*, inglês britânico *en-UK*, etc.

3.2 ActionManager

Aqui todos os componentes são responsáveis pela tomada de decisão da ação a tomar para determinado *input*. Para isso foram criados 3 componentes distintos: *ActionManager.py*, responsável geral pela da tomada de decisão; *ActionCalculator.py*, onde é feito o cálculo da ação a tomar, utilizando a biblioteca de *Machine-Learning* **Scikit-learn**, mais precisamente o *TfidVectorizer*; *ActionExecutor.py*, responsável por preparar o *output* da ação decidida.

3.3 Speech

Componente responsável pela conversão de áudio-texto, utilizando o serviço **Speech-to-Text** da *Google* (podendo ser fácilmente adaptado para usar qualquer outro serviço para além do atual).

3.4 Web

Devido à natureza da integração do módulo de voz no projeto mãe, onde terá de comunicar com outros módulos, foi necessário criar um servidor para onde se fazem pedidos de em áudio para uma ação, sendo retornada o resultado desta. Para isto, aqui encontram-se as componentes relacionadas com o servidor, como a definição de caminhos que este suporta, sendo esta utilizando a biblioteca **Flask**.

4 Casos de Aplicação

No momento em que o módulo é iniciado, mais precisamente o servidor *web*, este começa por fazer uma busca de todas as ações existentes no catálogo, processando e importando-as para o sistema (podendo ser atualizadas em tempo-real não interrompendo o módulo). No momento em que um pedido chega contendo um *blob* (ficheiro) de áudio, este continua para o *SpeechRecognizer* onde é processado recorrendo à biblioteca *Speech Recognizer* [7], sendo enviado para a *Google* o áudio e recebendo como resposta o texto correspondente. Após a conversão este é passado para o *ActionManager* onde será decidido que tipo de ação corresponde o *input*.

```
tags:
  - fixa
  - fixar
  - define
  - definir
  - opção

callback:
  request: POST
  url: http://localhost:3000/answer/next
  arguments:
    - answer

parse_callback_arguments_from_transcript: |
  import re
  extensiveNumbers: [(int, str)] = [
      (1, "primeira"),
      (2, "segunda"),
      (3, "terceira"),
      (4, "quarta"),
      (5, "quinta"),
      (6, "sexta"),
      (7, "sétima"),
      (8, "oitava"),
      (9, "nona"),
  ]
  def parse(transcript: str):
    numbers: [int] = [int(s) for s in re.findall(r'\b\d+\b', transcript)]
    if len(numbers) > 0:
        return [numbers[0]]
    for (number, extensiveNumber) in extensiveNumbers:
        if extensiveNumber in transcript:
            return [number]
    raise ValueError('No number present in transcript "{}"'.format(transcript))
```

Figura 5. Exemplo de uma ação que seleciona uma opção e avança para a próxima pergunta.

No momento da escolha da ação correspondente ao *input* o programa terá em atenção dois aspetos fundamentais: o idioma escolhido e as palavras do texto. Foi necessário separar os idiomas para que o modelo de *Machine Learning* tivesse menos dados com que trabalhar (facilitando os cálculos), mas também porque, naturalmente, se estamos a trabalhar numa língua só no momento do pedido, teremos de manter a consistência. Quanto às palavras, ao invés de serem separadas como inicialmente teria sido feito, optámos por usar a frase completa como *input* do modelo, pois assim os vetores de aproximação de palavras que constituem o modelo de decisão convergem de forma mais correta.

Continuando sobre o modelo de decisão mencionado em cima, mais concretamente o **TfidVectorizer** do *Scikit-learn*[9], este transforma palavras para um espaço matricial de várias dimensões (*features*) onde palavras mais próximas encontram-se naturalmente mais próximas umas das outras. Assim, na presença de uma frase contendo certas palavras-chaves, o modelo irá tentar colocar esta na sua posição ideal de aproximação tendo em conta todas as palavras com que o modelo foi treinado, obtendo assim a ação mais próxima da frase (através de uma função de aproximação, neste caso, **Cosine Similarity**[10]). Dado que é impossível haver uma precisão muito alta devido a possíveis palavras irrelevantes na frase de *input* ou até mesmo a existência de palavras-chave repetidas em várias ações, a precisão do modelo não é 100%, ou seja, não consegue ter completa garantia que está correto. Por isso foi necessário ajustar o grau de sensibilidade sobre a certeza da ação, não podendo ser demasiado elevado (pois iria falhar em demasia) nem demasiado baixo (iria errar em demasia), tendo ficado nos 0,4.

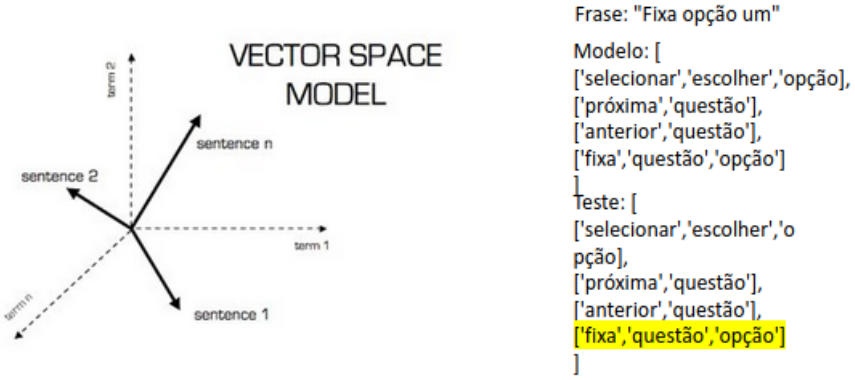

Figura 6. Demonstração simplificada do processo de treino e escolha da ação perante a frase "Fixa opção 1". Do lado esquerdo encontra-se o exemplo de abstração do modelo, onde cada eixo corresponde a uma *feature*, e do lado direito o exemplo em cima descrito e a solução que este devolverá.

Como referido anteriormente, apesar de na figura aparecerem 2 ações com a palavra "opção"contida na frase de *input*, o modelo consegue distinguir estas pelas outras palavras que as constituem. Neste caso, ele colocaria a frase mais perto da ação em amarelo.

5 Conclusões e Trabalho Futuro

A criação desta ferramenta de reconhecimento de voz e de processamento de ações a executar, e a sua consequente integração no assistente de educação Leonardo, representa uma aumento considerável da facilidade com que o utilizador consegue interagir com o sistema, sendo muito mais natural e fluída, aproximando o Leonardo de um tutor humano.

É de destacar a grande versatilidade em termos de gestão de ações que este sistema apresenta, sendo estas criadas, editadas e removidas através de ficheiros YAML, cuja mudança é sempre detectada em tempo real pelo programa, o que permite com imensa facilidade adaptar e evoluir as capacidades do sistema sem necessidade de fazer 'deploys'.

Ainda assim, nesta fase é difícil fazer a gestão dos argumentos que precisam de ser extraídos das transcrições, uma vez que para cada ação definida este processo tem de ser feito através do desenvolvimento de uma função em Python que recebe como argumento um 'string', que representa a transcrição, e que devolve um 'array' de argumentos.

No futuro, as melhorias a este serviço serão em torno da limitação apresentada anteriormente, pois esta impede que este sistema seja capaz de suportar a compreensão de respostas abertas, porque não existe nenhum mecanismo que permita extrair e analisar o conhecimento presente em determinada frase. Ainda que, este género de funcionalidade implicará sempre a perda de alguma autonomia e independência do serviço, tendo que haver uma maior relação contextual entre a interpretação de voz e o conteúdo de que ela advém, perdendo-se a capacidade de este sistema poder ser usado em outros projetos tal como está.

Referências

1. Adachi, Fumihiro & Isotani, Ryosuke & Hanazawa, Ken. (2014). Voice recognition system, voice recognition method, and program for voice recognition.
2. Jarvis, Python AI assistant, https://github.com/ggeop/Python-ai-assistant. Last accessed 6 Jul 2020
3. Leon, open-source personal assistant, https://github.com/leon-ai/leon. Last accessed 6 Jul 2020
4. Dittrich, Toby & Star, Sequoia. (2018). Introducing Voice Recognition into Higher Education. 10.4995/HEAD18.2018.8080.
5. Edwin, AI-powered English tutor, https://edwin.ai/. Last accessed 6 Jul 2020
6. Cognii, Artificial Intelligence for Education and Training, https://www.cognii.com/. Last accessed 6 Jul 2020
7. Speech Recognizer, Library for performing speech recognition, with support for several engines and APIs, online and offline, https://pypi.org/project/SpeechRecognition/.

8. Wong, S. & Raghavan, Vijay. (1984). Vector Space Model of Information Retrieval - A Reevaluation.. Communications of The ACM - CACM. 167-185.
9. Scikit-learn, A set of python modules for machine learning and data mining, https://pypi.org/project/scikit-learn/.
10. Cosine Similarity, Data Mining: Concepts and Techniques, Jiawei Han, Micheline Kamber and Jian Pei, 3ªEdição, https://www.sciencedirect.com/topics/computer-science/cosine-similarity