IMPERIAL

ELEC60014 - GROUP CONSULTANCY PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

MFTECH Patient Voice AI Documentation

Author:
Liam Browne,
Jongmin Choi,
Anlan Qiu,
Luke Torpey,
Edward Xiao,
Daisy Yu

Supervisor:
Dr. Cong Ling

Client:
Dr. Miguel Hernandez-Silveira

June 28, 2024

Contents

Chapter 1

Introduction

1.1 Project Overview

This project focuses on evaluating the feasibility of using AI algorithms to assess patients' health by analysing their voice and processing their speech.

As a solution, the team developed machine learning algorithms that can process existing audio recordings from a given dataset, interpret sound and voice tone, understand conversations, and analyse sentiment.

Furthermore, as an end product, the team built a web application encompassing the automatic examination process plus the display of statistical results and suggestions to the patient as a showcase of the solution.

All the code implementations can be found at the project's GitHub repository [1]. Record of meetings held with the Client and the Supervisor can be found in Appendix B and Appendix C.

1.2 Project Context

This project is part of a larger MF Technology AI teleconsultation project, which aims to use audiovisual information from patient video calls to derive vital signs and additional health indicators through audio processing and deep learning.

1.3 Target Audience

The target audience comprises medical professionals working in doctors' offices, hospitals, and clinics worldwide, who are interested in remotely examining patients and capturing vital health data, gaining further insights into patients' health; as well as the general public, who lack timely preliminary diagnoses advising them to seek professional medical care.

The service is aimed to be made widely utilised in the medical field, with the creation of a web application as an online platform for accessing the service.

1.4 Purpose of Document

This document aims to clarify the detailed design and functional specifications inside this project, as well as resources used and materials sourced by this project.

[1] `https://github.com/torpeyl/Group-16-Project-MFTECH-Patient-Voice-AI`

Chapter 2

System Overview

The high-level system pipeline consists of six stages, as shown in Figure 2.1.

The first stage receives a voice recording of the patient describing the symptoms as the input data.

During the next three stages, the three paralleled processing paths work independently to extract and examine different aspects of the patient's voice input. The individual workflow of each processing path is described in Section 4.1.

Finally, in the fifth stage, the overall evaluator takes in all outputs from the previous stage and integrates them into a numerical score indicating the degree of urgency for the patient to receive treatment, which is to be outputted in the next and final stage along with the detailed statistics.

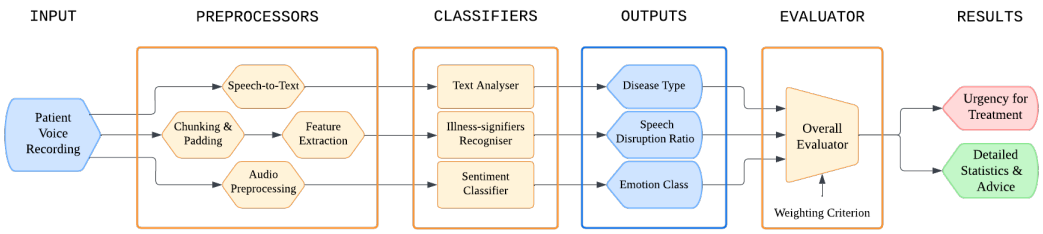


Figure 2.1: Diagram of the high-level pipeline of the system.

Chapter 3

Functional Specifications

3.1 Patient Data Upload

Patients can conveniently upload or record an audio file on the website, describing their current symptoms in their own voice.

Along with this, the patient is required to fill in specific personal details, which will be sent to the doctor. These details include the patient's Name, Age, Level of Pain on a scale from 1 to 10, Gender, and Drinking/Smoking Status. This information would only be used to provide extra information to doctors. It is not meant as a model input.

Once all the necessary information is provided, the patient will press the 'Submit' button, initiating the processing of the data upload. The system will then analyse the information and display the results for the patient to review.

3.2 Result Interpretation

After the evaluation, the result of the analysis will be presented in a PDF file, offering both overall conclusions on consultation and treatment necessity and a detailed breakdown of the patient's health condition. This document includes several key metrics to provide comprehensive insights into the patient's health. An example of the result document can be found in Appendix A.

One of the primary metrics is the Speech Disruption Ratio, which indicates the extent to which the patient's speech has been disrupted by symptoms such as coughing or sneezing. The PDF will also highlight the Most Probable Disease that the patient might have, based on the audio recording, along with the associated probabilities for each potential disease. Additionally, the Sentiment Score will reflect the emotional tone of the speech, where a higher score signifies a more negative or sad emotion in the patient's voice.

In the conclusion section, the diagnosis report will provide a recommendation on whether further consultation or treatment is necessary. This recommendation will be supported by a Treatment Score, which assesses the urgency or need for additional medical attention.

For the patient's convenience, there is an option to have these results sent directly to their email.

Chapter 4

Technical Specifications

4.1 Classifier Models

4.1.1 Text analyser

Symptom-Disease Dataset

The text analyser has been trained on a symptom-disease dataset[1] (Prajapati, 2023). This contained 7043 patient-symptom/disease-label pairs, and 1080 types of diseases labels. Another symptom-diagnosis dataset [2] (Barman, Karim & Sharma, 2023) was also considered. This dataset contained 1065 patient-symptom/disease-label pairs and 22 types of diseases labels. Despite a high test accuracy of 0.9670, 22 types of disease labels would be insufficient, as a user could potentially have a disease that the model doesn't account for.

Algorithm

The analyser takes a patient's description of their symptoms, ideally one to a few sentences, as input, and returns a predicted disease as output.

The architecture is a BERT model (Devlin et al., 2018), with a fully-connected dense layer connected to the output. The dense output layer has the same number of neurons as the number of disease labels. A visualisation of the model structure is shown in Figure 4.1.

After creating the model, the pre-trained BERT weights were then fine-tuned on the chosen symptom-disease dataset. The data had been split into tokens using the `AutoTokenizer` before used as inputs to the model.

The model was trained using sparse categorical cross-entropy as the loss function and the AdamW algorithm (Loshchilov & Hutter, 2019) as the optimiser. Hyperparameter search has been performed to find the optimal learning rate (0.0001), the optimal loss-function and the optimal optimiser. The early-stopping callback was used, and hence the model was trained for 10 epochs before the validation loss was assumed to have stopped improving. The `ReduceLROnPlateau` callback was also used to attempt to improve convergence, however it had no significant effect.

Design Choices

BERT was chosen as the base model, because of its existing understanding of language. BERT utilises the encoder part of the transformer architecture (Vaswani et al., 2023), and transformers use attention layers, which understand the connection between words in a sentence. Hence, after

[1] `https://huggingface.co/datasets/duxprajapati/symptom-disease-dataset`
[2] `https://huggingface.co/datasets/gretelai/symptom_to_diagnosis`

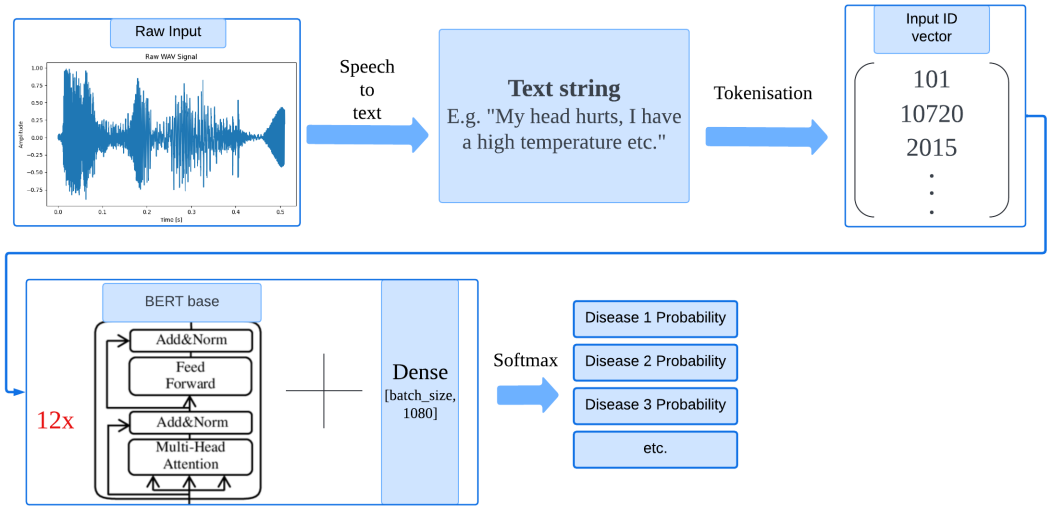


Figure 4.1: Visualisation of the architecture of the text analyser.

a small number of training epochs, BERT has a much higher validation accuracy than models that don't use attention, due to its preexisting knowledge.

Performance

The train-validation-test split was 66% for training, 14% for validation, and 20% for testing. The final test accuracy was 0.8190. The low accuracy was potentially due to the limited data for diseases. Some diseases only contained one data sample, which was used in the test data set. To better make use of all the training data, perhaps k-fold cross validation should have been used.

4.1.2 Illness-Signifiers Recogniser

Cough-Speech-Sneeze Dataset

The dataset that the Client has provided to the team was VOICED (Cesari et al., 2018), which only includes 208 samples of vowel pronunciations, insufficient to meet the Client's requirements. Thus, the team decided to seek an alternative dataset.

The cough-speech-sneeze dataset[3] (Amiriparian et al., 2017) was used for training the illness-signifiers recogniser. The dataset contains sufficient samples of human speech (1031), coughing (2871), and sneezing (1031) collected from YouTube, as well as silence clips (6).

The model was designed and trained to distinguish among these 4 classes (speech, coughing, sneezing, silence) for a very short clip.

Algorithm

The classifier takes the patient's voice recording, split or pad it into chunks of a fixed size with the pre-processor, and extracts features (spectrogram) from the chunks, which are then taken by the model as inputs to produce framewise illness-signifier recognition.

Inspired by a previous work on speech command recognition (Andrade et al., 2018), the model's architecture is designed to compose of two serial 2D convolution layers (LeCun, Bengio & Hinton, 2015) for extracting local relations; two serial bidirectional LSTMs (Hochreiter & Schmidhuber, 1997) for capturing long-term dependencies, followed by a max-pooling procedure to aggregate features across the span of the recording; and finally, three fully-connected layers for

[3] `https://audeering.github.io/datasets/datasets/cough-speech-sneeze.html`

classification. The final dense layer contains the same number of output neurons as the number of classes. A visualisation of the model structure is shown in Figure 4.2.

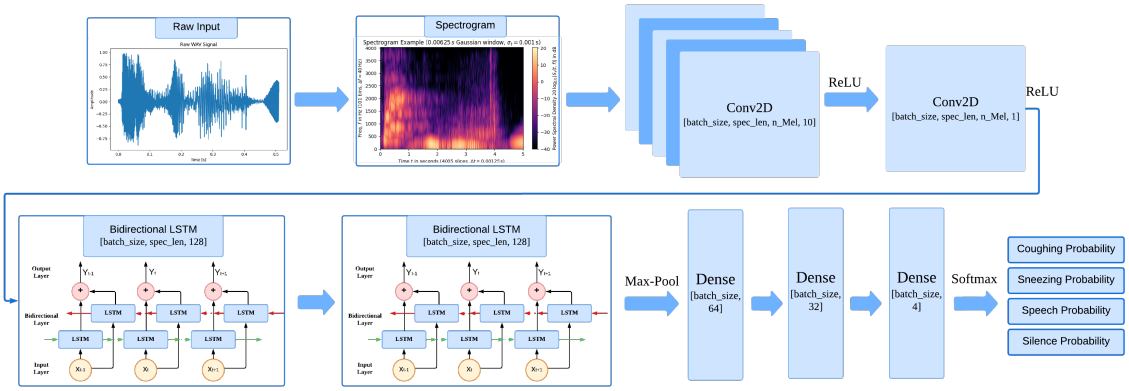


Figure 4.2: Visualisation of the architecture of the illness-signifiers recogniser.

The model was trained using sparse categorical cross-entropy as the loss function and the Adam algorithm (Kingma & Ba, 2017) as the optimiser. The training process implemented early stopping with patience of 10 epochs and maximum duration of 50 epochs. The learning rate was adjusted adaptively with initial learning rate of 0.001 and decay of 0.4 every 10 epochs. The batch size was 32, the train-validation-test split ratio was 8:1:1. The chunk size of each sample is 1.5 seconds after pre-processing, which has been determined from 10-fold cross-validation.

Design Choices

Alternative model types have been considered. Gaussian Mixture Models (GMMs), as a representative of unsupervised learning, has been implemented and tested on the same dataset with the same train-test data split and with GeMAPS features (Eyben et al., 2016) extracted using OpenSMILE (Eyben, Wöllmer & Schuller, 2010). However, after hyperparameter tuning, a test accuracy of 0.88051 with recall rates lower than precision rates for all classes has been considered inadequate and allowed room for improvements.

The selected model structure (CNNs+LSTMs) has been trained with different features extracted from voice recordings (spectrograms, MFCCs), and a modified structure only with LSTMs has been trained with OpenSMILE (GeMAPS) features (the CNNs for local feature extraction is not necessary as the feature vector is a stack of possibly correlated but individual prosodic and spectral features). A comparison of their performances is shown in Table 4.1. Among them, the one with spectrograms outperformed the others regarding both accuracy and speed, hence being used in the final implementation.

Audio feature	Model structure	Train accuracy	Test accuracy
Mel-spectrograms	CNNs + LSTMs	0.98811	0.97034
MFCCs	CNNs + LSTMs	0.97260	0.96654
MFCCs (with deltas & double deltas)	CNNs + LSTMs	0.97336	0.96654
OpenSMILE (GeMAPS) features	LSTMs	0.89901	0.84980

Table 4.1: Comparison of performances of models with different features.

Performance

During 10-fold cross validation, the final model achieved an average accuracy of 0.9876 on the train set and 0.9666 on the test set.

4.1.3 Sentiment Classifier

SAVEE Dataset

The sentiment classifier was trained on the Surrey Audio Visual Expressed Emotion (SAVEE) dataset[4] (Jackson & haq, 2011), which records four English speakers talking whilst expressing several different types of emotion.

However, because it only contains 480 recordings, augmentation techniques such as pitch shifting, time stretching and noise injection were used to enhance the dataset.

Algorithm

The classifier processes short audio speech samples to assess the probability of a patient exhibiting various emotions such as happiness, sadness, fear or surprise. The model utilises these probabilities to provide an estimation of the patient's sentiment, indicating the overall positivity or negativity of their emotional state.

The architecture is a GraphSAGE (SAmple and aggreGatE) model[5] (Hamilton, Ying & Leskovec, 2017) which uses 4 SAGEConv layers to process node features (audio features) and capturing relational information between nodes (audio segments). Batch Normalisation is applied after every SAGEConv layer to stabilise and accelerate the training process. A fraction of random node features are dropped during training to prevent overfitting. A global mean-pooling is used to aggregate node features across the graph to show a condensed and uniform representation of the graph. Finally, a linear layer with sigmoid activation predicts various emotions based on aggregate features, then further computing an illness score. A visualisation of the model structure is shown in Figure 4.3.

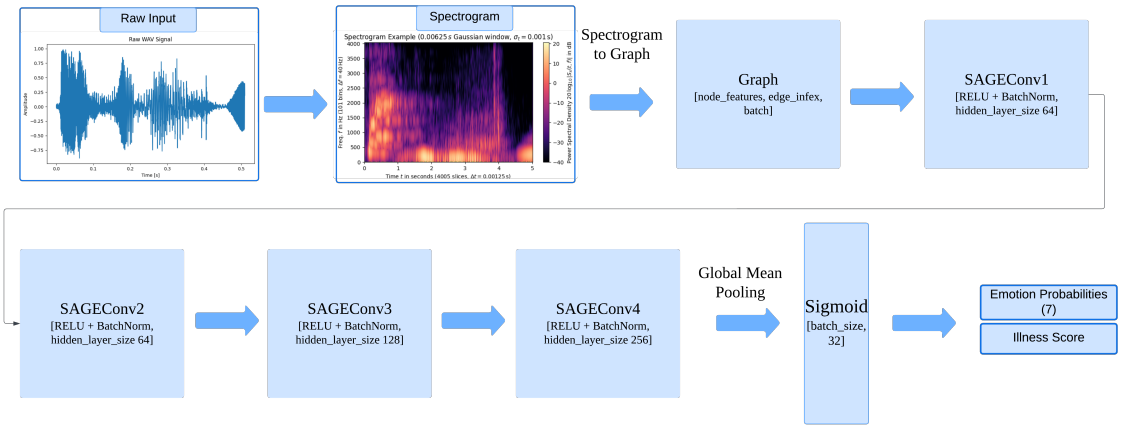


Figure 4.3: Visualisation of the architecture of the sentiment classifier.

The model training employed cross-entropy loss for emotion prediction and mean squared error (MSE) for illness scoring. A training loop with early stopping based on validation loss is used to prevent overfitting and ensure optimal model generalisation.

Design Choices

Several types of architectures, such as Convolutional Neural Network (CNN) (LeCun, Bengio & Hinton, 2015), Graph Convolutional Network (GCN) (Kipf & Welling, 2017) or Graph Attention Network (GAT) (Veličković et al., 2018), were considered and tested. GraphSAGE produced the best results.

[4] https://www.kaggle.com/datasets/barelydedicated/savee-database
[5] https://arxiv.org/abs/1706.02216

Batch Normalisation and Dropout are both used to help the model perform better on generalised test data and make the model more robust. Batch Normalisation normalises inputs to a layer for each mini-batch and helps to accelerate training and improve the stability of the neural network. This means faster convergence of the training process and helps maintain consistent input distributions across layers, making the model more robust to variations in training data. Dropout randomly removes a fraction of nodes during training, so the network doesn't become to reliant on on specific nodes and thus reducing overfitting.

```
# Hidden Layers
for i in range(1, num_layers):
    self.convs.append(SAGEConv(hidden_channels[i-1], hidden_channels[i]))
    self.batch_norms.append(torch.nn.BatchNorm1d(hidden_channels[i]))
```

Figure 4.4: Batch Norm Hidden Layers

Pre-emphasis is first applied, so that the high-frequency components of the signal are boosted. Then cube root compression is applied compressing the overall amplitude range, but the relative balance introduced by pre-emphasis is retained. The high frequencies remain emphasised, but with a reduced dynamic range.

```
def pre_emphasis_librosa(self, signal):
    return librosa.effects.preemphasis(signal, coef=self.pre_emphasis_coefficient)

def cubic_root_compression(self, signal):
    return np.sign(signal) * np.abs(signal)**(2/3)
```

Figure 4.5: Pre-Emphasis Cube Root Compression

Time stretching, Pitch Shifting and Noise injection techniques are used to increase the amount of training data and make the data more diverse and comprehensive.

```
def augment_data(self, signal, sr):
    augmented_signals = []
    # Time Stretching
    stretched = librosa.effects.time_stretch(signal, rate=1.1)
    augmented_signals.append(stretched)
    # Pitch Shifting
    pitch_shifted = librosa.effects.pitch_shift(signal, sr=sr, n_steps=2)
    augmented_signals.append(pitch_shifted)
    # Noise Injection
    noise = np.random.randn(len(signal))
    signal_noise = signal + 0.005 * noise
    augmented_signals.append(signal_noise)
    return augmented_signals
```

Figure 4.6: Data Augmentation

Global mean pooling is used to aggregate node features from a graph to produce a single, fixed-size representation for the entire graph.

```
# Global pooling
x = global_mean_pool(x, batch)
```

Figure 4.7: Global Pooling

Below shows the best parameters found when doing hyper-parameter tuning on learning rate, hidden layer dimensions, number of layers, batch size and dropout fraction.

```
params = {
    'lr': 0.001,
    'hidden_dim': [64, 64, 128, 256],
    'num_layers': 4,
    'batch_size': 16,
    'num_epochs': 100,
    'dropout': 0.1
}
```

Figure 4.8: Best Parameters

Performance

The final model achieved an average accuracy of 0.8255, precision of 8.4134, recall of 8.2923 and F1 score 8.3524 when predicting scores for each emotion during k-fold cross validation, which involved setting aside a separate test set to prevent overfitting. Furthermore it is worth noting that, despite having high test accuracy, the model could have benefited more from a larger and more diverse dataset.

4.1.4 Overall Evaluator

A function has been written to act as the overall evaluator. It takes the following inputs:

- the sentiment score outputted by the Sentiment Classifier;
- the speech-disruption ratio calculated using outputs from the Illness-Signifiers Recogniser;
- the disease that is most probable according to the Disease Classifier;
- the probability of that disease outputted by the Disease Classifier.

The overall evaluator outputs two Boolean values:

- whether further consultation is needed;
- whether further treatment is needed.

Consultation

Firstly, a list of extremely serious diseases, for which further consultation would always be recommended, has been identified. Their mapping indices have been saved in a JSON file. If the index of a detected disease is contained in this list, consultation is automatically required.

For the others, a different approach is taken. If the probability of the disease is lower than a "consultation threshold" defined as 0.5, then further consultation is needed. Otherwise, no consultation is needed. This is because, if the probability of the most likely disease is low, the model is unsure of what disease the patient has, and is not confident enough in its final prediction, hence it would be the best for the patient to determine their disease through further consultation with a professional doctor.

The above determination process is shown in Figure 4.9.

Treatment

A treatment score between 0 and 1 is calculated, and if this score is higher than a treatment threshold of 0.5, the treatment Boolean value is *True*, and vice versa. The treatment score is

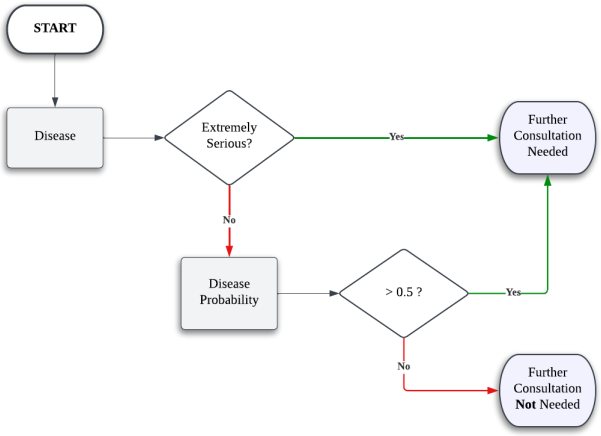


Figure 4.9: The logic for determining whether further consultation is needed.

calculated through a weighted average between the sentiment score and the disruption ratio. The weightings are determined as follows: There exists a list of diseases that deeply relate to speech disruptions. There also exists a list of diseases that heavily correspond to emotional responses.

- If the disease is in the disruption and emotional response lists, the weightings will be 0.5 for both the sentiment score and the disruption ratio.
- If the disease is in the disruptions list but not the emotional response list, the weighting will be 0.7 for the disruption ratio and 0.3 for the sentiment score.
- Similarly, if the disease is in the emotional response list, but not the disruption list, the weighting will be 0.7 for the sentiment score and 0.3 for the disruption ratio.
- Finally, if the disease is not on either list, the weighting will be 0.3 for both the sentiment score and the disruption ratio.

The above determination process is shown in Figure 4.10.

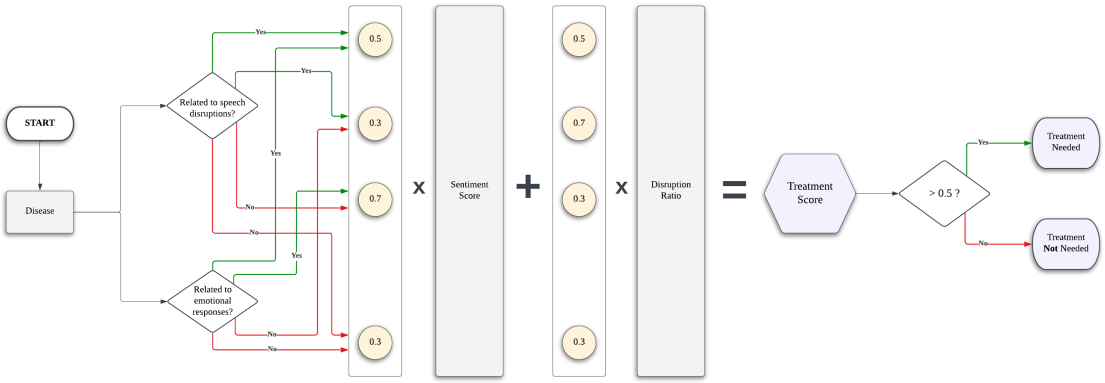


Figure 4.10: The logic for determining whether further treatment is needed.

4.2 Web Application

In order to demonstrate to the audience at the hackbooth what the project is all about, meanwhile enabling future applications of the product, the team has decided to develop an interactive website.

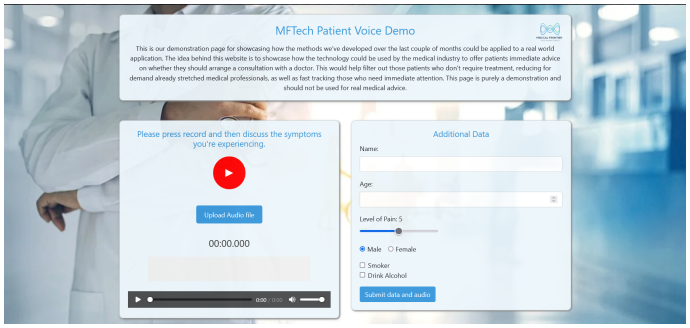


Figure 4.11: User interface of the Patient Voice AI

A Python script has been used as the back-end which served the front-end which is implemented with HTML and JavaScript.

4.2.1 Front-End

The front end for the main page consists of a section for recording audio data and a section for gathering key information about the patient such as age, and whether the patient smokes. This adds to the picture that the back-end software is trying to build, and in the case that the symptoms a patient is describing are ambiguous or very common, these lifestyle questions may aid in working out the patient's condition. A image of the landing page can be seen in Figure 4.11.

Initially, the front-end was implemented using vanilla HTML, CSS and JavaScript. However, to ensure the web-application remained flexible for a variety of devices, the Bootstrap library was used. This enabled the web page to remain dynamic, i.e. it resizes depending on the device it is being viewed on. Furthermore, when designing the result page, it was decided to display the PDF document with the patient's results on it. This was chosen over displaying the results in HTML/CSS, since the patient can see the PDF before it is sent to the patient's email.

When integrating the front-end with the back-end, compatibility between the languages used need to be ensured. Hence, Jinja2, a Python library was used to inject Python code in the HTML templates. Thus, variables and dictionaries constructed in Python could be directly communicated to the HTML templates on the back-end and returned to the front-end using GET requests.

Ensuring the anonymity and privacy integrity of the patient was centre to the front-end's design. To accommodate this, it has been ensured that the deep-learning models only receive the voice recording of the patient, i.e. no personal details that could be used to identify the patient. As a result, if the models become deployed on an external cloud service, the cloud service would only receive anonymous data. That being said, the doctor and patient will still be able to see data that corresponds to the patient's identity.

4.2.2 Back-End

Functional web-applications require communication with a server that strings together all aspects of the application. These aspects include, but are not limited to, user verification, data processing and endpoint management. To achieve this functionality, a web-application's front-end communicates with a server via HTML requests to exchange information. For instance, an HTML GET request would retrieve data from a server, while a POST request would send data to it. This data exchange virtually enables a web-application to have the functionality of a server. Furthermore, it has the advantage of presenting the results in an appealing manner to the user on the front-end. The server and any processes related to it tend to be referred to as the 'back-end'.

A back-end can be constructed using various programming languages, runtime-environments and frameworks. Initially, NodeJS was considered for this project's back-end. NodeJS is a JavaScript runtime-environment catered towards back-end development, whose asynchronous nature results in fast-loading web-applications. Furthermore, it allows developers to use a single language, i.e.

JavaScript, for both the front-end and back-end. Despite these advantages, the purpose of this project's website was to retrieve user information from the front-end, send it to various deep-learning models on the back-end and then send the respective results back to the front-end. Thus, since the deep-learning models were constructed in Python, using NodeJS for the back-end would have still required the execution of a Python script. Hence, the team decided to develop a Python-based back-end.

Flask, a Python framework, was utilised to facilitate the development of the back-end. The web-application had two main endpoints. Firstly, the landing page. The landing page was the page the user would initially visit. It included a form for additional patient details and an option to record or upload a voice recording. After pressing the landing page's submit button, a POST request would be sent to the back-end with the provided patient data, including the voice recording. An sample of this endpoint can be seen in Figure 4.12 The voice recording would then be sent to the models, each run sequentially. Due to the sequential running of the models, it would take an estimated 30 seconds before one received a result. Hence, a loading screen was implemented, preventing the user from submitting multiple, identical, POST requests to the back-end. Once the results had been processed, the back-end would send the results back to the front-end as a PDF document using a GET request. Furthermore, an updated HTML/CSS template would be sent with the results to ensure the results are displayed on a new page.

Despite creating a functional web-application, there are a considerable number of suggested improvements that would enhance the application's functionality. First and foremost, while recording on the website worked, the team were unable to append the recording to the form due to it being in an unfamiliar .ogg file format. Furthermore, running the models in parallel using mutliple GPUs would reduce the loading time to reach the results page.

```
@app.route('/convert_wav', methods=['POST'])
def convert_wav():

    data.clear()

    data["name"] = request.form['name']
    data["age"] = request.form['age']
    data["pain_level"] = request.form['points']
    data["birth_gender"] = request.form['inlineRadioOptions']
    data["smoker"] = 'Y' if request.form.get('smoker') == 'Y' else 'N'
    data["drinker"] = 'Y' if request.form.get('drinker') == 'Y' else 'N'
    file = request.files['file']
```

Figure 4.12: Sample code used for the POST request's endpoint.

4.2.3 Integration

Integration was crucial for the coherent interaction between the web-application and deep-learning models. Implementing a Python back-end meant the models could be called from the main file, i.e. `app.py`. That being said, due to the models being developed in independent environments, and therefore using varying library versions, a version for each library that supported all models must be found. In retrospect, this could have been avoided by agreeing on library versions before commencing the development. Following the successful execution of each model using a function, the user data from the front-end was integrated. This pipeline involved sending the voice recording to the back-end in the POST request. Once received, the recording would be saved locally to the server. Subsequently, the file path where the recording was saved would then be inputted into each model, allowing them to retrieve the recording locally. Each models' result would then be assigned to a variable of type String, which would then be stored in a dictionary. Storing all the data in a dictionary meant only the dictionary's name had to be passed as an input for functions, rather than each data in the dictionary. The code for this pipeline can be seen in Figure 4.13.

Once the result models had been retrieved on the back-end, a data-to-word function would be called. This function automatically generated a Microsoft Word document, provided a pre-existing template, using the unique patient data. The data was injected using the Jinja2 library.

Furthermore, to increase accessibility on the patient's side, the word document was converted to a PDF and sent to the patient's email.

```
if not os.path.exists('downloads'):
    os.makedirs('downloads')

filename = 'recordings.wav'
upload_path = f"./downloads/{filename}"
file.save(upload_path)

#Code for Anlan's Model
result_disease_model, index, prob = disease_model(upload_path)
data["result_disease_model"] = result_disease_model
data["disease_index"] = index
data["disease_prob"] = prob

#Code for Daisy's Model
result_cough_model, SDR = cough_model(upload_path)
result_cough_model = result_cough_model[8:]
data["result_cough_model"] = result_cough_model
data["SDR"] = SDR

#Code for Ed's Model
result_sentiment_model = sentiment_model(upload_path)
data["result_sentiment_model"] = result_sentiment_model

#Evaluation
consult_bool, treatment_bool, treatment_score = eval(data)
data['consult_bool'] = consult_bool
data['treatment_bool'] = treatment_bool
data['treatment_score'] = treatment_score

#Data --> .docx --> .pdf
toDoc(data)

return render_template('results.html', data=data)
```

Figure 4.13: Sample code used for model-integration on the back-end.

Chapter 5

Materials and Resources

Resources and Services:

1. Google Colab

Essential Python Libraries:

for audio processing:

1. Librosa 0.10.2
2. SpeechRecognition 3.10.4
3. TensorFlow I/O 0.31.0

for models:

1. PyTorch 2.3.1
2. Tensorflow 2.15.0
3. Keras 2.15.0
4. Transformers 4.41.2
5. SciPy 1.13.1
6. NumPy 1.26.4
7. Matplotlib 3.9.0

for datasets:

1. Datasets 2.20.0

The remaining Python libraries and their versions used in this project are shown in Appendix D.

Bibliography

Amiriparian, S., Pugachevskiy, S., Cummins, N., Hantke, S., Pohjalainen, J., Keren, G. & Schuller, B. (2017) CAST a database: Rapid targeted large-scale big data acquisition via small-world modelling of social media platforms. *2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII)*, 340–345. Available from: DOI:10.1109/ACII.2017.8273622.

Andrade, D. C. de, Leo, S., Viana, M. L. D. S. & Bernkopf, C. (2018) A neural attention model for speech command recognition. Available from: https://arxiv.org/abs/1808.08929 arXiv: 1808.08929 [eess.AS].

Barman, N. R., Karim, F. & Sharma, K. (2023) Symptom2Disease. https://www.kaggle.com/datasets/niyarrbarman/symptom2disease. Version 1.0. Uploaded to Hugging Face by Gretel.ai.

Cesari, U., De Pietro, G., Marciano, E., Niri, C., Sannino, G. & Verde, L. (2018) A new database of healthy and pathological voices. *Computers & Electrical Engineering*. 68, 310–321. Available from: https://doi.org/10.1016/j.compeleceng.2018.04.008

Devlin, J., Chang, M., Lee, K. & Toutanova, K. (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. abs/1810.04805. Available from: http://arxiv.org/abs/1810.04805 arXiv: 1810.04805.

Eyben, F., Scherer, K. R., Schuller, B. W., Sundberg, J., André, E., Busso, C., Devillers, L. Y., Epps, J., Laukka, P., Narayanan, S. S. & Truong, K. P. (2016) The Geneva Minimalistic Acoustic Parameter Set (GeMAPS) for Voice Research and Affective Computing. *IEEE Transactions on Affective Computing*. 7 (2), 190–202. Available from: DOI:10.1109/TAFFC.2015.2457417.

Eyben, F., Wöllmer, M. & Schuller, B. (October 2010) openSMILE - The Munich Versatile and Fast Open-Source Audio Feature Extractor. *Proc. ACM Multimedia (MM)*. Florence, Italy: ACM, 1459–1462.

Hamilton, W. L., Ying, R. & Leskovec, J. (2017) Inductive Representation Learning on Large Graphs. arXiv: 1706.02216 [cs.SI]. Available from: DOI:10.48550/arXiv.1706.02216.

Hochreiter, S. & Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Computation*. 9 (8), 1735–1780.

Jackson, P. & haq, S. ul (April 2011) Surrey Audio-Visual Expressed Emotion (SAVEE) database. Version 1.0. Available from: https://www.surrey.ac.uk/surrey-audiovisual-expressed-emotion-savee-database

Kingma, D. P. & Ba, J. (2017) Adam: A Method for Stochastic Optimization. Available from: https://arxiv.org/abs/1412.6980 arXiv: 1412.6980 [cs.LG].

Kipf, T. N. & Welling, M. (2017) Semi-Supervised Classification with Graph Convolutional Networks. Available from: https://arxiv.org/abs/1609.02907 arXiv: 1609.02907 [cs.LG].

LeCun, Y., Bengio, Y. & Hinton, G. (2015) Deep learning. *Nature*. 521, 436–444. Available from: DOI:10.1038/nature14539.

Loshchilov, I. & Hutter, F. (2019) Decoupled Weight Decay Regularization. Available from: https://arxiv.org/abs/1711.05101 arXiv: 1711.05101 [cs.LG].

Prajapati, D. (2023) symptom-disease-dataset. https://huggingface.co/datasets/duxprajapati/symptom-disease-dataset. Version 1.0. Hugging Face.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2023) Attention Is All You Need. Available from: https://arxiv.org/abs/1706.03762 arXiv: 1706.03762 [cs.CL].

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. & Bengio, Y. (2018) Graph Attention Networks. Available from: https://arxiv.org/abs/1710.10903 arXiv: 1710.10903 [stat.ML].

Appendix A

Example Diagnosis Result

MEDICAL FRONTIER TECHNOLOGY

Imperial College Project – Group 16

MFTech Patient Diagnosis

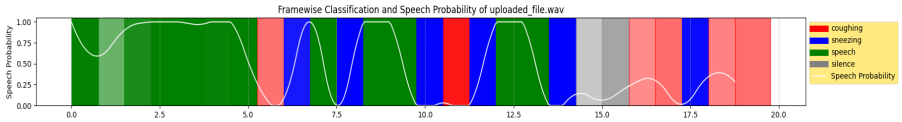
Patient Details:

Date: 25/06/24
Name: Ellie
Age: 23
Pain Level: 5
Smoker: Y
Drinker: N

Symptoms Transcript:

"*I've been feeling super sick recently I threw up last night I've been feeling nauseous and my head really hurts*"

Model Results:



Speech-Disruption Ratio: 0.5384615384615384

Most-Probable Diseases (Disease: Probability):

- Drug Reaction: 68.6 %
- Malaria: 7.6 %
- Dengue: 4.3 %

Sentiment Score: 0.369

Conclusion:

Further Consultation Required: N

Treatment Required: Y

Treatment Score: 0.6845

Appendix B

Meeting Minutes with Client

MFTECH Patient Voice AI Project

Minutes for Tuesday, 30th April, 2024

Present: Dr. Miguel Hernandez-Silveira (Client), Liam Browne, Jongmin Choi, Anlan Qiu, Luke Torpey, Edward Xiao, Daisy Yu

Meeting Purpose

The project team meets with Client for a full briefing on the project, context, and expectations.

Agenda Items

Motivation

Virtual consultations can allow doctors to examine patients vitals signs data such as temperature, blood pressure or heart rate. Multi-model AI may help provide more valuable insights into patients health by determining patterns or discrepancies in the patients voice. m

Project Goal

As conveyed by Client:

1. To analyse the patients' tone of voice for determining the possibility of illness.
2. To spot patterns in voices which may contain implications for potential illness.
3. To understand the content of the patients' utterance, and extract relevant keywords.

The Italian Dataset (VOICED)

The project team examined the provided dataset[1] and suggested that the recordings did not contain any verbal content except pronunciations of vowels.

The Client expressed that an algorithm could then be developed to simply identify patterns in the voice contained in the dataset and determine probability of illness.

The Client approved of searching for alternative datasets to use if necessary.

Available Resources

The Client pointed out that the project team could consider making use of various existing resources, including AWS (Amazon Web Services), processing resources on Google Colab, algorithms and models on HuggingFace, etc.

Expectations

The Client emphasised the focus on extracting meaningful information from the voice, rather than development of complicated algorithms.

[1] https://physionet.org/content/voiced/1.0.0/

Actions

1. Create a WhatsAPP groupchat with client to allow easier communication.
2. Research and discuss feasibility of VOICED dataset.
3. Research ideas and techniques to extract useful information from medical dialogue.

Next Meeting: Tuesday, 7th May, 2024

MFTECH Patient Voice AI Project

Minutes for Tuesday, 7th May, 2024

Present: Dr. Miguel Hernandez-Silveira (Client), Liam Browne, Jongmin Choi, Anlan Qiu, Luke Torpey, Edward Xiao, Daisy Yu

Meeting Purpose

The project team meets with Client to discuss details on choosing alternative datasets, as the team has found the provided dataset insufficient to meet Client's need.

Agenda Items

Alternative Datasets

The project team presented to Client with two alternative datasets:

1. the VietMed[1] dataset, which contains actual Vietnamese conversations that allows natural language processing, as Client has wished for.
2. the Saarbruecken[2] dataset, which is similar to the provided VOICED[3] dataset but provides an extra 2000 voice samples.

The Client proposed that the team could split the tasks among the group members and explore all the above-mentioned datasets before selecting one.

The Client requested for a detailed working plan outlining how the project is to be proceeded, expected to be received before the following week.

Audio Processing Methods

The Client suggested that Mel-spectrograms could be utilised as a representation of auditory signals. The team appreciated this valuable suggestion and expressed that it would be attempted along with other feature extraction methods.

Actions

1. Split into two groups of three and research architectures or methods to extract information from the VietMed or Saarbruecken dataset.
2. Research ways of representing audio signals in a way compatible with ML models.

Next Meeting: Tuesday, 21st May, 2024

[1] https://huggingface.co/datasets/leduckhai/VietMed
[2] http://stimmdb.coli.uni-saarland.de/index.php4#target
[3] https://physionet.org/content/voiced/1.0.0/

MFTECH Patient Voice AI Project

Minutes for Tuesday, 21st May, 2024

Present: Dr. Miguel Hernandez-Silveira (Client), Liam Browne, Anlan Qiu, Luke Torpey, Daisy Yu

Absent: Jongmin Choi, Edward Xiao

Meeting Purpose

The project team meets with Client to update on the progress since last meeting and request for feedback and clarification on several related subjects.

Agenda Items

Progress and Feedback on Text Analysis

The project team showed Client the results obtained from fine-tuning the BERT[1] model on a new dataset, symptom-disease-dataset[2].

The Client was impressed with the performance of the model and provided constructive suggestions to further improving the results and developing the pipeline:

1. Perform hyperparameter tuning using model training APIs[3] with a validation set of data to find the best set of parameters (and obtain a rationale).
2. Switch the training optimiser to $AdamW$[4] and increasing the learning rate to speed up the training process.
3. Plot the performance of the model on training set and validation set over epochs to find the optimal training duration and prevent overfitting.
4. Add more explanatory text as comments to the code to improve readability.
5. Chain the text-analyser model with a speech-to-text API (e.g. on HuggingFace[5] or by OpenAI[6]) to complete the pipeline.
6. Evaluate the functionality of the pipeline by testing it with self-recorded audio samples containing descriptions of symptoms.

The team explained that the current work is only meant to be a simplified demonstration, and the suggestions by Client would be taken into account in further development of the model.

[1] https://huggingface.co/google-bert/bert-base-uncased
[2] https://huggingface.co/datasets/duxprajapati/symptom-disease-dataset
[3] https://keras.io/api/models/model_training_apis/
[4] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/AdamW
[5] https://huggingface.co/docs/transformers/model_doc/speech_to_text
[6] https://platform.openai.com/docs/guides/speech-to-text/timestamps

Clarification on Voice Tone/Sentiment Analysis

The project team updated Client with the progress on classification of vocal track diseases based on voice analysis of vowel-pronunciation samples.

The Client explained that rather than directly classifying a specific type of disease from the voice, the focus should be placed more on detecting the overall health condition indicated by the voice, by analysing the sentiment or tone of voice and identifying voice disruptions such as crying, sneezing, coughing, broken voices, etc., resulted from stress or emotions. The analysis should aim to whether the patient requires further consultation and treatment.

The Client provided a list of articles about research in the similar area to further clarify the goal:

1. *Do I Sound Sick? - The Lancet*[7]
2. *Listening For Illness: Your Voice May Soon Help Detect Health Problems - Worldcrunch*[8]
3. *Voice for Health: The Use of Vocal Biomarkers from Research to Clinical Practice - PMC*[9]

The team understood the requirement by Client, and suggested that new datasets need to be used and the current methods might need to be adjusted.

Actions

1. Look for more suitable datasets relevant to what the Client wants.
2. Redistribute roles to work on Sentiment analysis, Keyword analysis and non-vocal expressions analysis (coughing, sneezing etc.)

Next Meeting: To be determined

[7] https://www.thelancet.com/action/showPdf?pii=S2589-7500%2821%2900182-5
[8] https://worldcrunch.com/tech-science/voice-ai-healthcare
[9] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8138221/

Appendix C

Meeting Minutes with Supervisor

MFTECH Patient Voice AI Project

Minutes for Friday, 3rd May, 2024

Present: Dr. Cong Ling (Imperial Project Supervisor), Liam Browne, Jongmin Choi, Anlan Qiu, Luke Torpey, Edward Xiao, Daisy Yu

Meeting Purpose

The project team meets with Imperial Project Supervisor to discuss project details, meeting with client and plan of action.

Agenda Items

Project Details

The project team discussed details of the project, including motivation, expectations and were given advice for success on the project.

Alternative Datasets

The project team discussed and notified the Supervisor about the feasibility of the dataset; believing that it could be beneficial to search for a different dataset.

The Supervisor suggested that the team contact the Client to communicate the concerns about the feasibility of the dataset, so that the team could search for a better, more robust dataset.

Next Meeting: Undetermined

Appendix D

Python Library Dependencies

absl-py 2.1.0
aiohttp 3.9.5
aiosignal 1.3.1
appscript 1.2.5
astunparse 1.6.3
attrs 23.2.0
audioread 3.0.1
Babel 2.15.0
blinker 1.8.2
cachetools 5.3.3
certifi 2024.6.2
cffi 1.16.0
charset-normalizer 3.3.2
click 8.1.7
contourpy 1.2.1
cycler 0.12.1
decorator 5.1.1
docx2pdf 0.1.8
docxcompose 1.4.0
docxtpl 0.17.0
filelock 3.15.3
Flask 3.0.3
Flask-Mail 0.10.0
flatbuffers 24.3.25
fonttools 4.53.0
frozenlist 1.4.1
fsspec 2024.6.0
gast 0.5.4
google-auth 2.30.0
google-auth-oauthlib 1.2.0
google-pasta 0.2.0
grpcio 1.64.1
h5py 3.11.0
huggingface-hub 0.23.4
idna 3.7
itsdangerous 2.2.0
Jinja2 3.1.4
joblib 1.4.2
kiwisolver 1.4.5
lazy_loader 0.4
libclang 18.1.1
librosa 0.10.2
llvmlite 0.43.0
lxml 5.2.2
Markdown 3.6
markdown-it-py 3.0.0
MarkupSafe 2.1.5
mdurl 0.1.2
ml-dtypes 0.2.0
mpmath 1.3.0
msgpack 1.0.8
multidict 6.0.5
namex 0.0.8

networkx 3.3

numba 0.60.0

oauthlib 3.2.2

opencv-contrib-python 4.10.0.84

opt-einsum 3.3.0

optree 0.11.0

packaging 24.1

pandas 2.2.2

pillow 10.3.0

platformdirs 4.2.2

pooch 1.8.2

protobuf 4.25.3

psutil 6.0.0

pyasn1 0.6.0

pyasn1_modules 0.4.0

pycparser 2.22

pydub 0.25.1

Pygments 2.18.0

pyparsing 3.1.2

python-dateutil 2.9.0.post0

python-docx 1.1.2

pytz 2024.1

PyYAML 6.0.1

regex 2024.5.15

requests 2.32.3

requests-oauthlib 2.0.0

rich 13.7.1

rsa 4.9

safetensors 0.4.3

scikit-learn 1.5.0

six 1.16.0

soundfile 0.12.1

soxr 0.3.7

SpeechRecognition 3.10.4

sympy 1.12.1

tensorboard 2.15.2

tensorboard-data-server 0.7.2

tensorflow-estimator 2.15.0

tensorflow-io 0.37.0

tensorflow-io-gcs-filesystem 0.37.0

tensorflow-macos 2.15.0

termcolor 2.4.0

threadpoolctl 3.5.0

tokenizers 0.19.1

torch_geometric 2.5.3

tqdm 4.66.4

typing_extensions 4.12.2

tzdata 2024.1

urllib3 2.2.2

Werkzeug 3.0.3

wfdb 4.1.2

wrapt 1.14.1

yarl 1.9.4