3 Data Preprocessing

Data Science - Capstone Project Submission

- Student Name: **James Toop**
- Student Pace: **Self Paced**
- Scheduled project review date/time: **29th October 2021 @ 21:30 BST**
- Instructor name: **Jeff Herman / James Irving**
- Blog URL: **https://toopster.github.io/ (https://toopster.github.io/)**

IMPORTANT NOTE:

This section presents code and instructions for preprocessing each dataset for training the models.

The datasets and transformed JSON files have not been included in the GitHub repository with this notebook and will need to be downloaded and stored in the local repository for the code to run correctly.

The code in the notebook (2_data_acquisition.ipynb) entitled `2_data_acquisition.ipynb` contains code for downloading the datasets.

To ensure ease of use, however, it is also possible to download the raw and transformed datasets using this link (https://drive.google.com/file/d/11IKYIZiwEQJ-pp0G1bJPHXLJLj8uKPqW/view?usp=sharing).

In [1]:

```
# Import required libraries and modules for data preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from pydub import AudioSegment
import wave
import soundfile as sf
import librosa, librosa.display
import IPython.display as ipd

import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
from tensorflow.keras import models

import os
import pathlib
from pathlib import Path
import shutil
import collections
import nltk

import shared_functions.preprocessing as preprocess
```

3.1 Stage 1 - Transforming the Ultrasuite dataset

3.1.1 Isolating utterances from the original audio samples and labelling

Unlike the Speech Commands dataset, each audio sample in it's raw format contains multiple utterances that are spoken by both the Speech Therapist and the child subject.

The labels for each `.wav` file have been stored in a separate `.lab` file together with timestamps for the start and end of each utterance. The following is an example for the audio clip previewed earlier:

```
54700000 60900000 CAR
88800000 94800000 GIRL
109500000 112999999 MOON
126100000 136400000 KNIFE
```

In order to get the audio samples from the Ultrasuite datasets into the appropriate format for the Deep Learning models, we will need to splice the raw audio samples according to these timestamps.

In [2]:

```
def ultrasuite_word_labels(src_dataset, src_file):
    '''
    Extracts the labels from a single *.lab file into a single DataFrame

        Params:
            src_dataset (str): Dataset name of the target *.lab file
            src_file (str): Filename of the target *.lab file

        Returns:
            word_labels_df (pandas.core.frame.DataFrame):
            DataFrame containing the labels (utterances), timestamps, speaker
            and session from the *.lab file
    '''
    filepath = 'data/ultrasuite/labels-uxtd-uxssd-upx/'
    filepath = filepath + src_dataset + '/word_labels/lab/' + src_file

    columns = ['start_time', 'end_time', 'utterance']
    word_labels_df = pd.DataFrame()
    word_labels_df = pd.read_csv(filepath,
                                 sep=" ",
                                 header=None,
                                 names=columns)

    # Extract the speaker, session and speech data from the filename
    word_labels_df['dataset'] = src_dataset
    word_labels_df['speaker'] = src_file[0:3]
    if len(src_file[4:-9]) == 0:
        word_labels_df['session'] = None
    else:
        word_labels_df['session'] = src_file[4:-9]
    word_labels_df['speech_waveform'] = src_file[-8:-4]

    # Tidy up data formatting and correct time based units
    word_labels_df['utterance'] = word_labels_df['utterance'].str.lower()
    word_labels_df['start_time'] = pd.to_timedelta(word_labels_df['start_time']
                                                   * 100)
    word_labels_df['end_time'] = pd.to_timedelta(word_labels_df['end_time']
                                                 * 100)

    return word_labels_df
```

In [3]:

```
# Quick test to check function works for a single labels file
upx_01F_df = ultrasuite_word_labels('upx', '01F-BL1-005A.lab')
upx_01F_df.head()
```

Out[3]:

	start_time	end_time	utterance	dataset	speaker	session	speech_waveform
0	00:00:00	00:00:00.620000	teeth	upx	01F	BL1	005A
1	00:00:03.860000	00:00:04.650000	watch	upx	01F	BL1	005A
2	00:00:06.160000	00:00:06.780000	orange	upx	01F	BL1	005A
3	00:00:09.050000	00:00:09.980000	school	upx	01F	BL1	005A

In [4]:

```
def extract_segments(y, sr, segments, dataset):
    '''
    Extracts audio segments from the source *.wav file based on timestamps
    contained within the associated *.lab file

        Params:
            y (str): Path to input file
            sr (int): Sample Rate
            segments (DataFrame): DataFrame containing timestamps, labels,
                                  speaker and session data
            dataset (str): Specific ultrasuite dataset to process can be
                           'upx', 'uxtd' or 'uxssd'
    '''
    # Compute segment regions in number of samples
    starts = np.floor(segments.start_time.dt.total_seconds() * sr).astype(int)
    ends = np.ceil(segments.end_time.dt.total_seconds() * sr).astype(int)

    isolated_directory = 'data/ultrasuite_isolated/' + dataset + '/'

    if not os.path.isdir(isolated_directory):
        os.makedirs(isolated_directory.strip('/'))

    i = 0
    # Slice the audio into segments
    for start, end in zip(starts, ends):
        audio_seg = y[start:end]
        print('extracting audio segment:', len(audio_seg), 'samples')

        # Set the file path for the spliced audio file
        file_path = isolated_directory + str(segments.speaker[i]) + '/'
        if segments.session[i] != None:
            file_path = file_path + str(segments.session[i]) + '/'
        file_path = file_path + str(segments.speech_waveform[i]) + '/'

        if not os.path.isdir(file_path):
            os.makedirs(file_path.strip('/'))

        file_name = file_path + str(segments.utterance[i]) + '.wav'

        sf.write(file_name, audio_seg, sr)
        i += 1
```

In [5]:

```
def process_ultrasuite_wav_files(src_dataset, src_speaker, src_session):
    '''
    Processes and extracts audio segments for all Ultrasuite *.wav files

        Params:
            src_dataset (str): Ultrasuite dataset to process can be
                               'upx', 'uxtd' or 'uxssd'
            src_speaker (str): Speaker to process
            src_session (str): Session to process

    '''
    directory = 'data/ultrasuite/core-'
    directory = directory + src_dataset + '/core/' + src_speaker + '/'

    # Set the target directory based on session if available
    if src_session != False:
         directory = directory + src_session + '/'

    # Loop through files in directory, splice and rename based on labels
    for filename in os.listdir(directory):

        if not filename[-5:-4] == 'E' or filename[-5:-4] == 'D':
            # Fetch the corresponding word labels and load into a DataFrame
            # Handle errors for when no labels exist
            # Labels only available for high quality samples
            try:
                if src_session != False:
                    labels_filename = src_speaker + '-' + src_session + '-'
                    labels_filename = labels_filename + filename[-8:-4]
                else:
                    labels_filename = src_speaker + '-' + filename[-8:-4]

                labels_filename = labels_filename + '.lab'

                labels_df = ultrasuite_word_labels(src_dataset,
                                                   labels_filename)

                wav_path = directory + filename
                y, sr = librosa.load(wav_path, sr=16000)
                extract_segments(y,
                                 sr,
                                 labels_df,
                                 src_dataset)

            except IOError:
                if src_session != False:
                    print('\n',
                          src_speaker,
                          '-',
                          src_session,
                          '-',
                          filename[-8:-4],
                          '.lab not found \n')
                else:
                    print('\n',
                          src_speaker,
                          '-',
                          filename[-8:-4],
                          '.lab not found \n')
```

In [6]:

```
def process_all_wav_files(datasets):
    '''
    Processes and extracts audio segments for all Ultrasuite *.wav files

        Params:
            datasets (list): Ultrasuite dataset to process can be any or all
                             of 'upx', 'uxtd', 'uxssd'
    '''
    # Loop through the datasets
    for dataset in datasets:
        current_dataset_dir = 'data/ultrasuite/core-' + dataset + '/core/'
        speakers = os.listdir(current_dataset_dir)

        # Loop through the speakers
        for speaker in speakers:
            current_speaker_dir = 'data/ultrasuite/core-' + dataset + '/core/'
            current_speaker_dir = current_speaker_dir + speaker + '/'
            sessions = os.listdir(current_speaker_dir)

            # If multiple therapy sessions loop through and process files
            for session in sessions:
                if os.path.isdir(os.path.join(current_speaker_dir, session)):
                    process_ultrasuite_wav_files(dataset, speaker, session)
                else:
                    process_ultrasuite_wav_files(dataset, speaker, False)
```

In [ ]:

```
# Splice all *.wav files for all datasets
# NOTE: This takes a long time to run
process_datasets = ['upx', 'uxssd', 'uxtd']
process_all_wav_files(process_datasets)
```

3.1.2 Standardising the audio samples and folder structure

In [12]:

```
def pad_silence(target_length, input_filepath, output_filepath):
    '''
    Pad the spliced audio samples with silence so that they are all at least
    1 second in length

        Params:
            target_length (int): Target length of final audio sample in
                                 milliseconds
            input_filepath (str): File path to input / original *.wav file
            output_filepath (str): File path to output / padded *.wav file
    '''
    target_length = target_length
    audio = AudioSegment.from_wav(input_filepath)
    if len(audio) > target_length:
        print(str(input_filepath) ,
              'is longer that 1 second, no padding required.')
        silence = AudioSegment.silent(duration=0)
    else:
        silence = AudioSegment.silent(duration=target_length - len(audio) + 1)

    padded = audio + silence
    padded.export(output_filepath, format='wav')
```

In [13]:

```
def standardise_filing(datasets):
    '''
    Standardise filing structure for isolated samples, padding and renaming
    files in the process

        Params:
            datasets (list): Ultrasuite dataset to process can be any or all
                             of 'upx', 'uxtd', 'uxssd'
    '''
    # Loop through the datasets
    for dataset in datasets:

        isolated_files = Path.cwd() / 'data/ultrasuite_isolated' / dataset

        for isolated_file in isolated_files.glob('**/*'):

            if isolated_file.is_file():

                # Rename the file but don't lose the original references
                filename = isolated_file.stem
                extension = isolated_file.suffix
                sourcedata = dataset
                sourcefile = isolated_file.parent.parts[-1]


                # Handle the different folder structures
                if dataset == 'uxtd':
                    speaker = isolated_file.parent.parts[-2]
                    new_filename = f'{filename}_{dataset}-{speaker}-{sourcefile}

                else:
                    session = isolated_file.parent.parts[-2]
                    speaker = isolated_file.parent.parts[-3]
                    new_filename = f'{filename}_{dataset}-{speaker}-{session}-{s

                # Define the new file path creating directory if it doesn't exis
                new_path = Path.cwd() / 'data/ultrasuite_transformed' / filename

                if not new_path.exists():
                    new_path.mkdir(parents=True, exist_ok=True)

                new_file_path = new_path.joinpath(new_filename)

                # Pad audio sample if required and move to new location
                if extension == '.wav':
                    pad_silence(1000, str(isolated_file), str(new_file_path))
```

In [ ]:

```
# Run the function to standardise the filing for all Ultrasuite datasets
standardise_filing(['upx', 'uxssd', 'uxtd'])
```

3.1.3 Cleansing the dataset

1. Only keep audio samples of actual words using NLTK WordNet as a source corpus
2. Remove audio samples of simple phonetic letters (from the `manual remove` list)

3. Only keep audio samples that have more than 5 different samples

In [15]:

```
# Check to see if the WordNet corpus is available, download if not and import
try:
    nltk.data.find('corpora/wordnet')
except LookupError:
    nltk.download('wordnet')

from nltk.corpus import wordnet as wn

def remove_invalid_samples():
    '''
    Function to remove all 'invalid' audio samples based on predetermined
    criteria
    '''
    transformed_files = 'data/ultrasuite_transformed/'

    manual_remove = ['a']

    for name in sorted(os.listdir(transformed_files)):

        path = os.path.join(transformed_files, name)

        if os.path.isdir(path):
            num_samples = len(os.listdir(path))

            # Remove audio samples of words not listed in NLTK WordNet corpus
            if not wn.synsets(name) or len(name)==1:
                print(name,
                      'is NOT a valid word, removing',
                      num_samples,
                      'samples')
                shutil.rmtree(path)
            # Remove audio samples where there are 5 or less samples
            elif num_samples <= 5:
                print(name,
                      'does NOT have enough samples, removing',
                      num_samples,
                      'samples')
                shutil.rmtree(path)
            # Remove audio samples based on our manually constructed list above
            elif name in manual_remove:
                print(name,
                      'is being manually removed',
                      num_samples,
                      'samples')
                shutil.rmtree(path)
            else:
                print('---')
                print(name,
                      'is a valid word and there are',
                      num_samples,
                      'samples:\n')
```

In [ ]:

```
# Remove invalid audio samples from the transformed dataset
remove_invalid_samples()
```

3.1.4 Subsetting the dataset

In [17]:

```
# Get the audio sample file information for the Ultrasuite dataset
ultrasuite_filestats = preprocess.get_filestats('data/ultrasuite_transformed')
ultrasuite_filestats.head()
```

Out[17]:

	sample_utterance	sample_filename	sample_duration	sample_samplerate
0	parch	parch_upx-05M-BL2-017A.wav	1.000938	16000
1	parch	parch_upx-05M-Mid-016A.wav	1.001000	16000
2	parch	parch_upx-05M-BL3-016A.wav	1.000875	16000
3	parch	parch_upx-05M-BL4-016A.wav	1.000875	16000
4	parch	parch_upx-05M-Maint-016A.wav	1.000875	16000

In [18]:

```
# Summarise the number of samples for each utterance
us_summary = (ultrasuite_filestats.groupby(['sample_utterance'])
                                  .size()
                                  .reset_index(name='count')
                                  .sort_values('count', ascending=False))
us_summary.head(35)
```

Out[18]:

	sample_utterance	count
366	helicopter	292
700	say	290
961	watch	235
249	elephant	233
322	got	229
705	scissors	222
946	umbrella	222
274	fishing	222
814	spider	217
397	in	211
314	gloves	210
892	thank	204
84	bridge	198
290	frog	178
958	was	171
744	sheep	166
980	yellow	163
323	gown	162
237	ear	159
538	on	154
75	boy	148
282	four	146
412	ken	143
542	or	142
704	school	142
984	zebra	141
908	times	135
505	monkey	135
906	tiger	133
548	pack	132

	sample_utterance	count
275	five	130
884	teeth	128
905	tie	123
106	cab	123
176	crab	122

In [19]:

```
def copy_keywords(num_keywords, keywords):
    '''
    Copys the 'top' keywords based on number of samples to a new folder

        Params:
            num_keywords (int): Number of 'top' keywords to copy based on
                                number of samples available
            keywords (DataFrame): DataFrame containing keywords sorted by
                                  number of samples
    '''
    src_directory = 'data/ultrasuite_transformed/'
    top_directory = 'data/ultrasuite_top' + str(num_keywords) + '/'

    sorted_keywords = keywords.reset_index()

    if not os.path.isdir(top_directory):
        os.makedirs(top_directory.strip('/'))

    i = 0
    while (i < num_keywords):
        src_folder = src_directory + sorted_keywords.sample_utterance[i]
        dest_folder = top_directory + sorted_keywords.sample_utterance[i]

        if not os.path.isdir(dest_folder):
            shutil.copytree(src_folder, dest_folder)

            print(sorted_keywords.sample_utterance[i], 'copied')
        else:
            print(sorted_keywords.sample_utterance[i], 'already exists')
        i += 1
```

In [20]:

```
1 copy_keywords(35, us_summary)
```

```
helicopter copied
say copied
watch copied
elephant copied
got copied
scissors copied
umbrella copied
fishing copied
spider copied
in copied
gloves copied
thank copied
bridge copied
frog copied
was copied
sheep copied
yellow copied
gown copied
ear copied
on copied
boy copied
four copied
ken copied
or copied
school copied
zebra copied
times copied
monkey copied
tiger copied
pack copied
five copied
teeth copied
tie copied
cab copied
crab copied
```

3.2 Stage 2 - Feature extraction

In [21]:

```
# Function to extract features to use in the models and store in JSON file
def preprocess_dataset(dataset_path,
                       json_path,
                       feature,
                       num_samples,
                       num_mfcc=13,
                       n_fft=2048,
                       hop_length=512):
    '''
    Code adapted from Deep Learning Audio Application from Design
    to Deployment - Valerio Velardo - The Sound of AI

    Extract Mel Spectrograms and MFCCs to use in the models and store in JSON
    file

        Params:
            dataset_path (str):
            Path to dataset containing audio samples
            feature (str):
            Specific feature requested, accepts either 'MFCCs' or 'mel_specs'
            json_path (str):
            Output path to JSON file
            num_samples (int):
            num_mfcc (int):
            n_fft (int):
            hop_length (int):
    '''
    # Dictionary to temporarily store mapping, labels, MFCCs and filenames
    if feature == 'mel_specs':
        data = {
            'mapping': [],
            'labels': [],
            'mel_specs': [],
            'files': []
        }
    else:
         data = {
            'mapping': [],
            'labels': [],
            'MFCCs': [],
            'files': []
        }

    # Loop through all sub directories
    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):

        # Ensure we're at sub-folder level
        if dirpath is not dataset_path:

            # Save label in the mapping
            label = dirpath.split('/')[-1]
            data['mapping'].append(label)
            print("\nProcessing: '{}'".format(label))

            # Process all audio files in the sub directory and store features
            for f in filenames:
                file_path = os.path.join(dirpath, f)

                # Load audio file and slice it to ensure length consistency
```

```
                signal, sample_rate = librosa.load(file_path)

                # Drop audio files with less than pre-decided number of samples
                if len(signal) >= num_samples:

                    # Ensure consistency of the length of the signal
                    signal = signal[:num_samples]

                    # Extract MFCCs
                    if feature == 'mel_specs':
                        mel_specs = librosa.feature.melspectrogram(signal,
                                                                   sample_rate,
                                                                   n_fft=n_fft,
                                                                   hop_length=ho

                        data['mel_specs'].append(mel_specs.T.tolist())

                    else:
                        MFCCs = librosa.feature.mfcc(signal,
                                                     sample_rate,
                                                     n_mfcc=num_mfcc,
                                                     n_fft=n_fft,
                                                     hop_length=hop_length)

                        data['MFCCs'].append(MFCCs.T.tolist())

                    # Append data in dictionary
                    data['labels'].append(i-1)
                    data['files'].append(file_path)
                    print("{}: {}".format(file_path, i-1))

    # Save data in JSON file for re-using later
    with open(json_path, 'w') as file_path:
        json.dump(data, file_path, indent=4)
```

In [22]:

```
# Set the parameters for the Speech Commands dataset for preprocessing
sc_dataset_path = 'data/speech_commands_v0.02'
sc_json_path = 'speech_commands_data.json'
num_samples = 22050
```

In [ ]:

```
# Preprocess the Speech Commands dataset extracting MFCCs
preprocess_dataset(sc_dataset_path, sc_json_path, 'MFCCs', num_samples)
```

In [24]:

```
# Set the parameters for the Ultrasuite dataset for preprocessing
us_dataset_path = 'data/ultrasuite_top35'
us_json_path = 'ultrasuite_top35_data.json'
num_samples = 22050
```

In [ ]:

```
# Preprocess the Ultrasuite dataset extracting MFCCs
preprocess_dataset(us_dataset_path,
                   us_json_path,
                   'MFCCs',
                   num_samples)
```

In [25]:

```
# Set the parameters for the Ultrasuite dataset for preprocessing
# based on Mel Spectrograms
us_dataset_path = 'data/ultrasuite_top35'
us_melspec_json_path = 'ultrasuite_top35_data_melspec.json'
num_samples = 22050
```

In [ ]:

```
# Preprocess the Ultrasuite dataset extract Mel Spectrograms
preprocess_dataset(us_dataset_path,
                   us_melspec_json_path,
                   'mel_specs',
                   num_samples)
```

Sources / Code adapted from:

* Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow - Aurélien Géron (https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/)

* Deep Learning Audio Application from Design to Deployment - Valerio Velardo - The Sound of AI (https://github.com/musikalkemist/Deep-Learning-Audio-Application-From-Design-to-Deployment)