Event Detection in DCASE

April 5, 2022

Francisco Javier Sáez Maldonado

[1]:
```
from appsa_pr1 import *
import numpy as np
from os import listdir
from os.path import isfile, join

%load_ext autoreload
%autoreload 2
```

1 Audio segment representation and annotations

1.1 Wave shapes and audio features

We begin by plotting a random signal of the validation set. We fix the numpy random seed so that the same signal appears in every execution (I have noticed that the signal may change with different versions of `NumPy`). If we change the seed, the plotted signal would change. Also, recall that the `plot_waveform` function returns the loaded signal, which we can use to extract useful information.

[2]:
```
np.random.seed(123)
path = DATASET_PATH + AUDIO_SUBPATH
audio_names = [f for f in listdir(path) if isfile(join(path, f))]
choice = np.random.choice(audio_names,1)[0]
print(choice)
```

```
Y_HDolAn2_us_60.000_70.000.wav
```

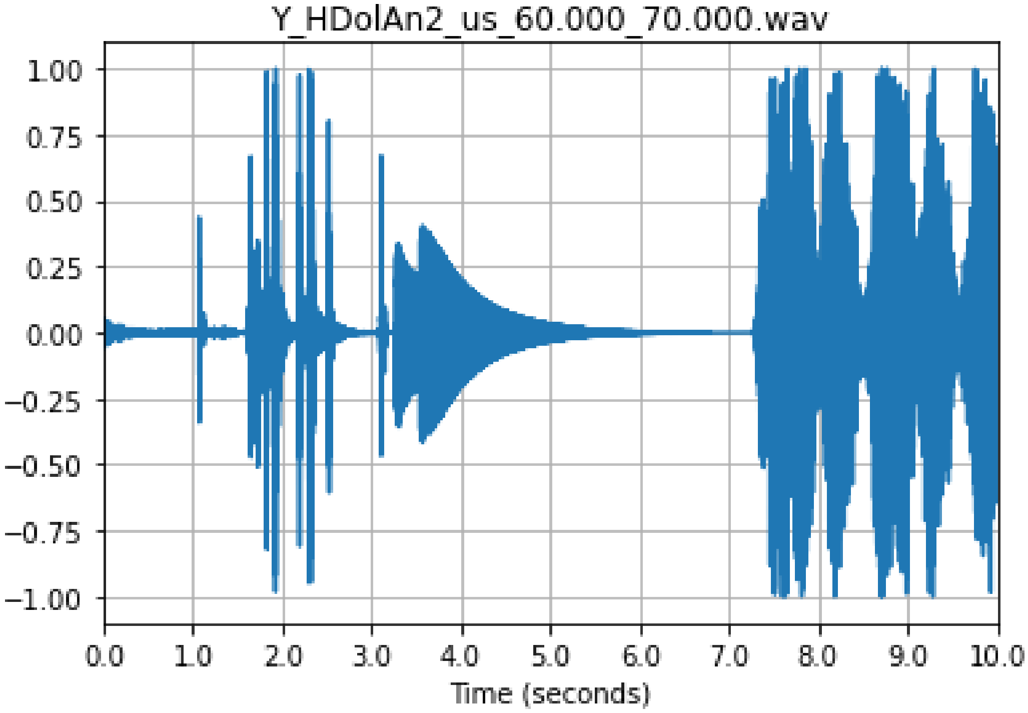
When we listen to this audio, we can recognize a few events:

- In the first 3 seconds, a knock on certain type of surface sounds.
- After that, a door bell rings
- Lastly, around second 7, a few people say "Little pigs, little pigs".

[3]:
```
signal = plot_waveform(choice)
print("Num samples     = {}".format(signal[0].size))
print("Sample rate f_s = {}".format(signal[1]))
print("Total duration  = {}".format(signal[0].size / signal[1]))
```

```
Num samples     = 441000
Sample rate f_s = 44100
```

```
Total duration  = 10.0
```



- What is the sample rate f_s ?

The load function from `librosa` returns both the audio timeseries `y` and the sample rate `f_s`. In this case,the **sample rate** of 44100 samples per second.

- What is the duration of the audio file in seconds?

We can compute it by dividing the total number of samples by the sample rate, obtaining a duration of 10 seconds.

- Include the obtained figure in this document

We can see the obtained figure above.

1.1.1 Mel-spectrogram

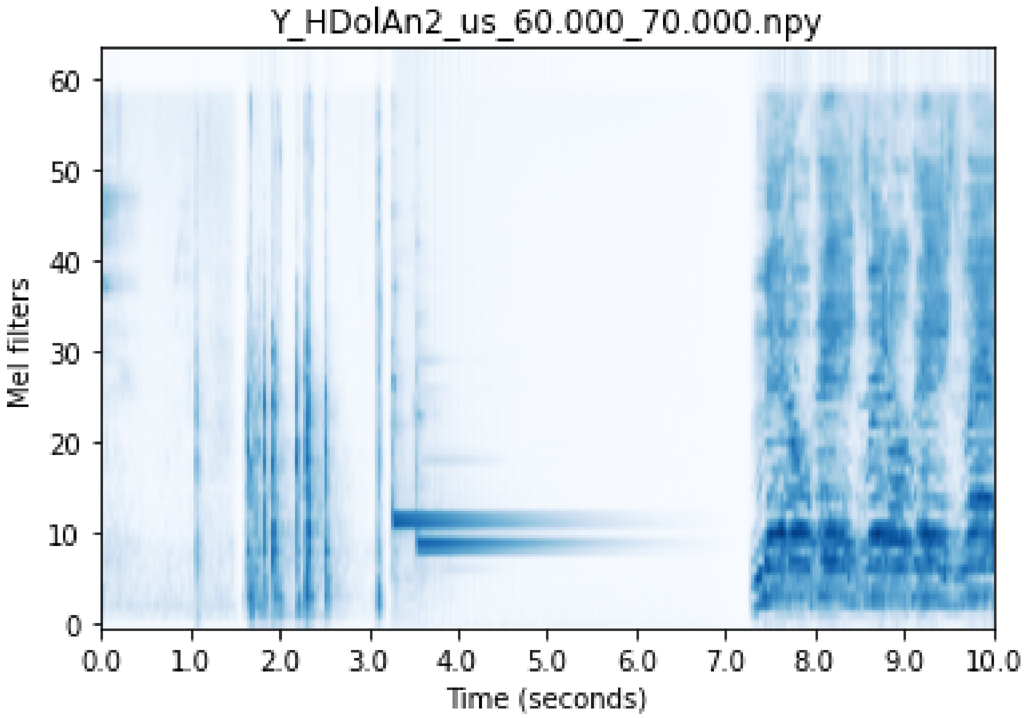
Let us plot the **mel-spectrogram**. This is obtained applying the *Fast Fourier transform* using overlapping windowed segments of the audio signal, and the converting the y axis (frequency) to a **mel-scale**. We use the same audio used in the previous case (we replace in the filename the last three characters for the ones needed for the `NumPy` extension).

We include the figure and some variables that will help us on the following questions.

```
[4]: choice_feats_filename = choice[:-3] + 'npy'
     mel = plot_melgram(choice_feats_filename)
```

```python
print("Mel shape : {}".format(mel.shape))
print("Seconds   : {}".format(mel.shape[0]/86))
print("Total size: {}".format(np.prod(mel.shape)))
```

```
Mel shape : (864, 64)
Seconds   : 10.046511627906977
Total size: 55296
```



- How many temporal frames does the mel-spectrogram representation contains?

Inspecting the shape of the mel-spectrogram, we found that it has 864 frames.

- How many mel filters are represented in the obtained mel-spectrogram?

Again, inspecting the shape we found that we are representing 64 filters.

- Compare the waveform size with the mel-spectrogram size in term of the number of samples in each representation

We have seen that the original signal has 441000 samples and the mel-spectrogram has $864 \cdot 64 = 55296$ elements, which is a little bit more of 1/10 of the original signal, reducing the size significantly.

1.2 Event annotation

In this section we will load the annotations of the considered example and we will make some comments about them. Let us begin by loading them. We also assume that in the events with `NaN` annotations (no annotations for this event), we can use 0.0 in all the cases.

[5]:

```
anno_path = DATASET_PATH + META_SUBPATH + META_FILE

import pandas as pd
df = pd.read_csv(anno_path, sep = "\t", header = 0)
df = df.fillna(0)
print(df.head)
```

```
<bound method NDFrame.head of                               filename  onset
offset     event_label
0     Y00pbt6aJV8Y_350.000_360.000.wav  0.000   9.971  Vacuum_cleaner
1       Y00pK0GMmE9s_70.000_80.000.wav  0.000  10.000  Vacuum_cleaner
2       Y02sD1KJeoGA_50.000_60.000.wav  0.000  10.000          Frying
3       Y0bjUq9XMMmQ_30.000_40.000.wav  0.000  10.000          Frying
4       Y0cH_NlhhMAs_30.000_40.000.wav  1.710   6.005             Cat
…                                …    …     …             …
4246  Yb8GxUkjLSUY_628.000_638.000.wav  4.772   5.228          Speech
4247  Yb8GxUkjLSUY_628.000_638.000.wav  5.606   6.360          Speech
4248  Yb8GxUkjLSUY_628.000_638.000.wav  7.644   8.220          Speech
4249  Yb8GxUkjLSUY_628.000_638.000.wav  8.524   9.391          Speech
4250    Y86owBlJa8f0_24.000_34.000.wav  0.000   0.000               0

[4251 rows x 4 columns]>
```

We can now select the annotations for our audio

[6]:

```
choice_annotations = df.loc[df['filename'] == choice]
print(choice_annotations)
```

```
                           filename  onset  offset         event_label
530  Y_HDolAn2_us_60.000_70.000.wav  3.224   5.706  Alarm_bell_ringing
531  Y_HDolAn2_us_60.000_70.000.wav  7.253   9.983              Speech
```

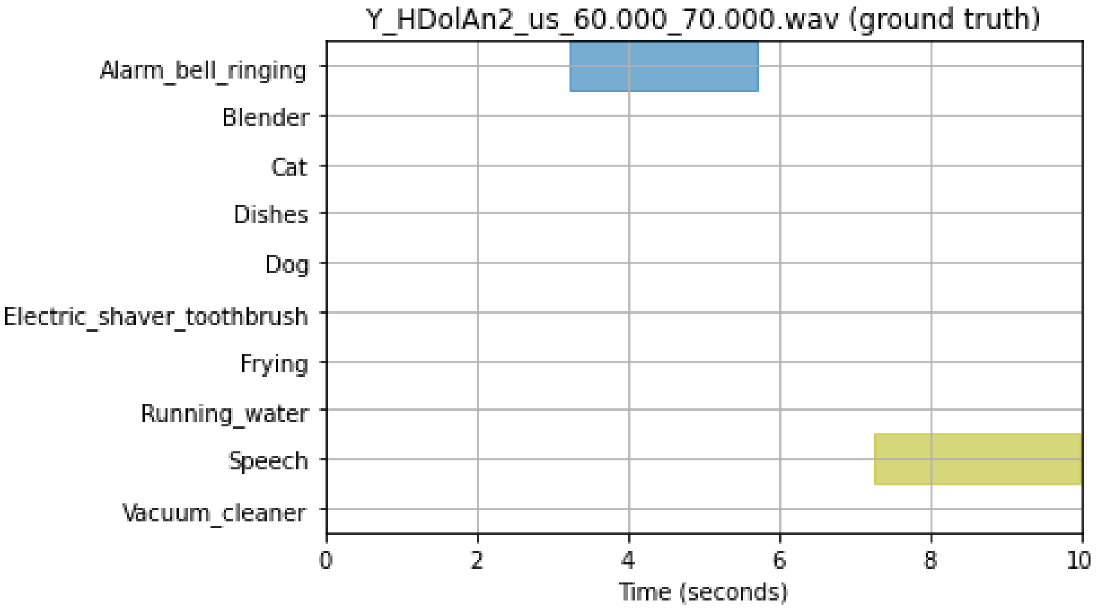
As we can see, there are two events in our audio:

- One in the interval of time $[3.224, 5.706]$ that is an `Alarm_bell_ringing`
- The second one is in the interval of time $[7.253, 9.983]$ that is `Speech`.

Let us plot the annotations:

[7]:

```
plot_labels(choice)
```



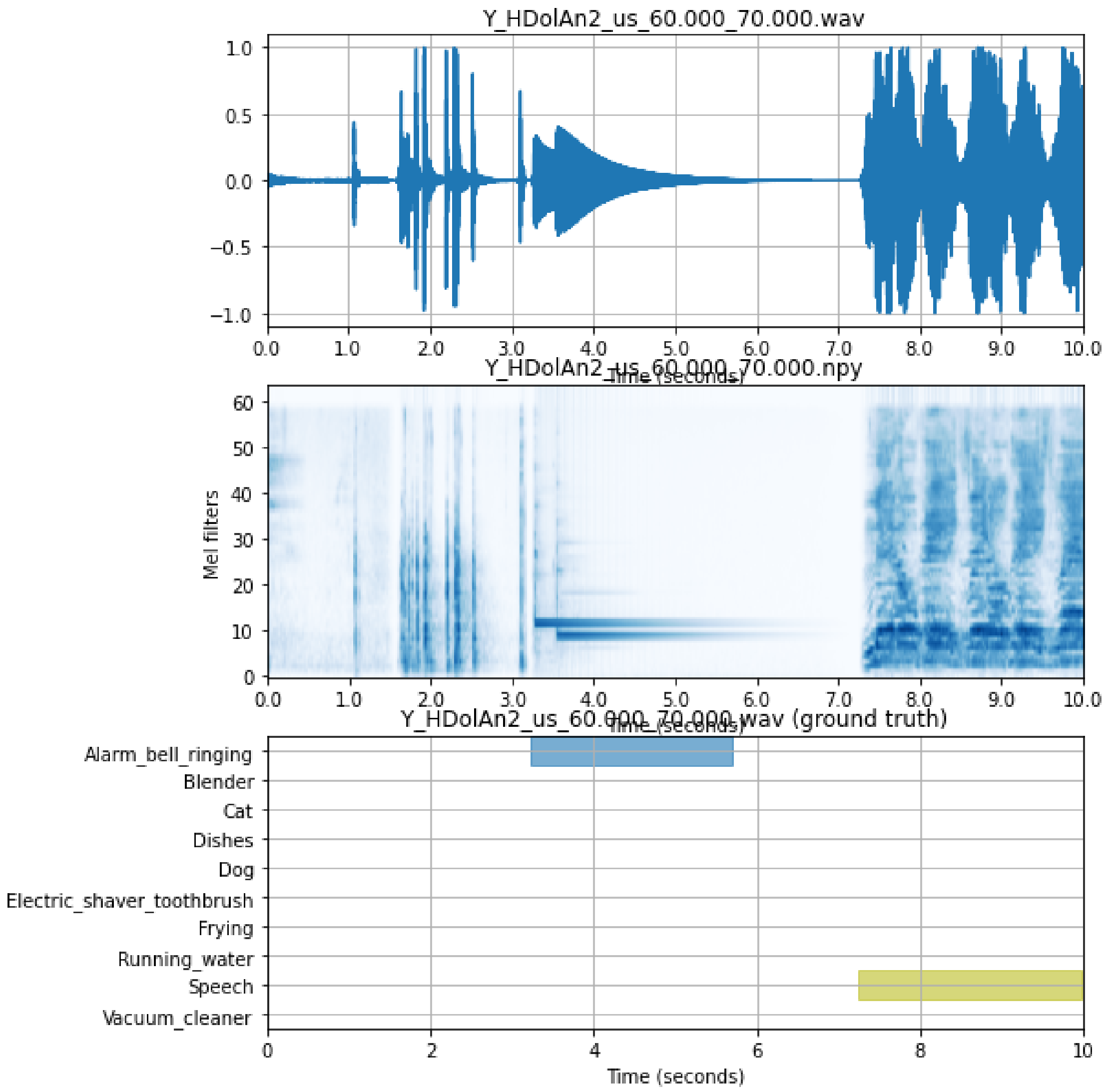
- Is there overlapping between the audio events?

We already saw in the previous description with the time intervals that there is no overlapping between the tags. The plot of the labels confirms this.

- Compare the waveform and the mel-spectrogram with the annotation, and associate the annotated audio events with the different parts of the representations

To achieve this, it we will plot the three previous graphs stacked:

[8]:
```
fig,ax = plt.subplots(3,1,figsize = (8,10))
_ = plot_waveform(choice, ax[0])
_ = plot_melgram(choice_feats_filename, ax[1])
plot_labels(choice, ax[2])
```



We can appreciate that there is a direct correspondence between the audio waveform, the mel-spectrogram, the labels and the audio. As we can see, when the alarm bell rings, the mel-spectrogram turns blue in the filters $[9 - 13]$ approximately. Also, there is a big perturbation in the waveform. The same occurs when the people start talking (as we mentioned in the description of the listened audio), there is a huge change in the waveform plot, the mel-spectrogram turns blue and the label starts, all in approximately the same vertical point.

2 Acoustic event dectection using a pre-trained model

2.1 Scores and metrics

Firstly, we will execute the validation using the pretrained model. The results are redirected to the file "pretrained_results.txt" (included in the zip file), and we will comment those results in this document.

[1]:

```
!python TestModel.py --model_path=pretrained_model.p > pretrained_results.txt
```

We are asked to comment the **event based metrics**. In particular, we will focus on the following ones:

- Class-wise average metrics (macro-average)
- Class-wise metrics

The F_1 score is the main metric for the evaluation of acoustic detection systems. It is expressed as follows:

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN},$$

where TP are the *true positives*, FP are the *false positives* and FN are the false negatives. It is expressed a percentage, so it is usually in the range $[0, 100]\%$. In fact, the F_1 score can be seen as a **weighted average** of the **precision** and the **recall**. Let us dig deeper on these concepts.

The precision is written as

$$\text{precision} = \frac{TP}{TP + FP},$$

which essentially indicates the percentage (in the $[0, 1]$ scale) of the positives that are true positives. Clearly, if the precision is close to zero, it means that our model is classifying as positive examples that are not positive.

The recall is written as:

$$\text{recall} = \frac{TP}{TP + FN},$$

indicating the percentage of total positives that have been classified as positive. Again, we would like to reduce of false negatives (FN) to obtain higher recalls.

Using precision and recall, the F_1 score can be expressed as follows:

$$F_1 = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Usually, the performance is computed as the average of the F_1 scores in each of the tested categories. Let us show the obtained results in terms of **class-wise average metrics**:

Measure	Metric	Score
F-measure		
	F1 score	22.56 %
	Precision	22.75 %
	Recall	24.83 %
Error rate		
	Error rate (ER)	1.86
	Deletion Rate	0.75
	Insertion Rate	1.10

To obtain these results, the **deletion/insertion rate** are considered. We found in the metrics of the DCASE challenge that these have the following definition:

- A **substitution** is an event in system output that has **correct temporal position** but **incorrect class label**
- The **insertions** are events in the system output that are not correct nor substituted. Informally, we could say that the model detected *something not existent.*
- The **deletions** are labeled events that are not correct nor substituted. That is, labeled events that our model has not captured.

Further explanation about this metrics with its detailed formulas can be found at the original paper A Discriminative Model for Polyphonic Piano Transcription. With this definitions, we can explain the results in a clearer way. The following results have to be remarked:

- We find that the system obtains a F_1 score of 22.56%, which is a **really low value**. Considering that the F_1 is expressed as a function of the precision and recall, and observing that both precision and recall are also very low, this is not a surprise. Since both precision and recall are low, we can say that the model model is not good at classifying **only** positive examples as positives (FP must be high in order to reduce precision), and it also **rejects many true positives**, marking them as negatives (FN must be high to obtain a low recall).
- The error rate is also quite high, exceeding 1 by a wide margin. In a standard machine learning problem, usually the error is in the range $[0, 1]$, and the relation $1 = \text{error} + \text{accuracy}$ is always true. However, this is not true for event detection in audio. The Error Rate formula is expressed as follows: The error rate formula is

  $$ER = \frac{S + D + I}{N},$$

  where S is the number of substitutions, D is the number of deletions, I is the number of insertions and N is the number of events in the ground truth labels. In this case, since we are **inserting** events that may not exist, we may have that $S + D + I > N$, leading to an error rate greater than 1, which happens in the case that we ara analyzing.
- The previous statement is reinforced by the Deletion and Insertion rates, which are quite high, meaning that the model is not detecting events that exist (deletions) and detecting non-existent event (insertions).

Event label	Nref	Nsys	F	Pre	Rec	ER	Del	Ins
Dishes	567	502	13.1%	13.9%	12.3%	1.64	0.88	0.76
Alarm_bell..	420	283	38.7%	48.1%	32.4%	1.03	0.68	0.35
Running_wa..	237	331	28.9%	24.8%	34.6%	1.70	0.65	1.05
Electric_s..	65	100	14.5%	12.0%	18.5%	2.17	0.82	1.35
Speech	1754	1213	35.9%	43.9%	30.3%	1.08	0.70	0.39
Dog	570	431	6.6%	7.7%	5.8%	1.64	0.94	0.70
Blender	96	119	23.3%	21.0%	26.0%	1.72	0.74	0.98
Cat	341	331	20.8%	21.1%	20.5%	1.56	0.79	0.77
Frying	94	376	11.9%	7.4%	29.8%	4.40	0.70	3.70
Vacuum_cle..	92	127	32.0%	27.6%	38.0%	1.62	0.62	1.00

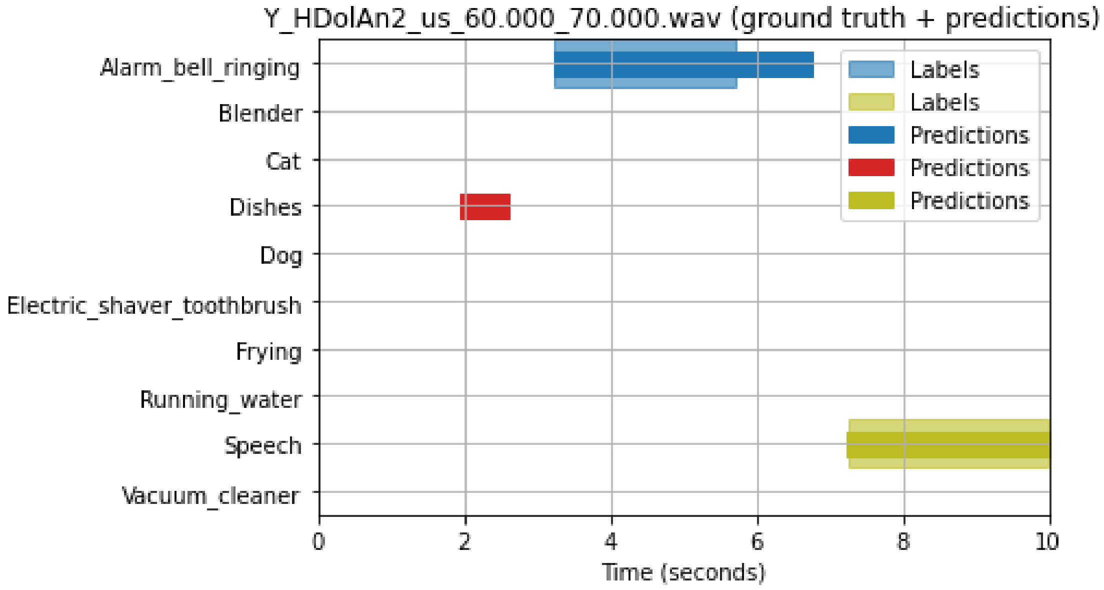
We can see that there is no event that has a higher F_1 score than 38% (a really low value), indicating that the model is not good at capturing any of the acoustic events. Also, we see that all the ERs are above one, meaning that the insertions and deletions are high in all cases. Specially, we notice that the *ER* is 4.4 in the *Frying* event, having a 3.7 insertion rate: the model is inserting many *frying* events that do not exist (or, at least, they were not labeled).

2.2 Predictions

The previous execution also saves the prediction for the models in a TSV file called `validation2019_predictions.tsv`. We would like to compare the predictions of the model with the **ground truth** labels of our audio files. Let us present some of the results obtained by the model and comment these results using the visualization.

We begin with the audio we chose for the first analysis. We have made a slight modification on the `plot_labels_predictions` so that it shows on the legend which events are predictions and which events are ground trugh labels. Different class predictions/labels will be shown in different colors. (We also added a parameter to show the legend in the position we want so that it does not hide the important parts of the plot).

[10]:
```
plot_labels_predictions(choice)
```



As we can observe, the results commented previously apply to our example: the model classifies well the correctly found events, but also creates an insertion about a non existent event: The model creates a *dishes* prediction when there is no label, and also keeps the *alarm bell ringing* longer that the true label. There are two examples of insertions, leading to lower F_1 score.

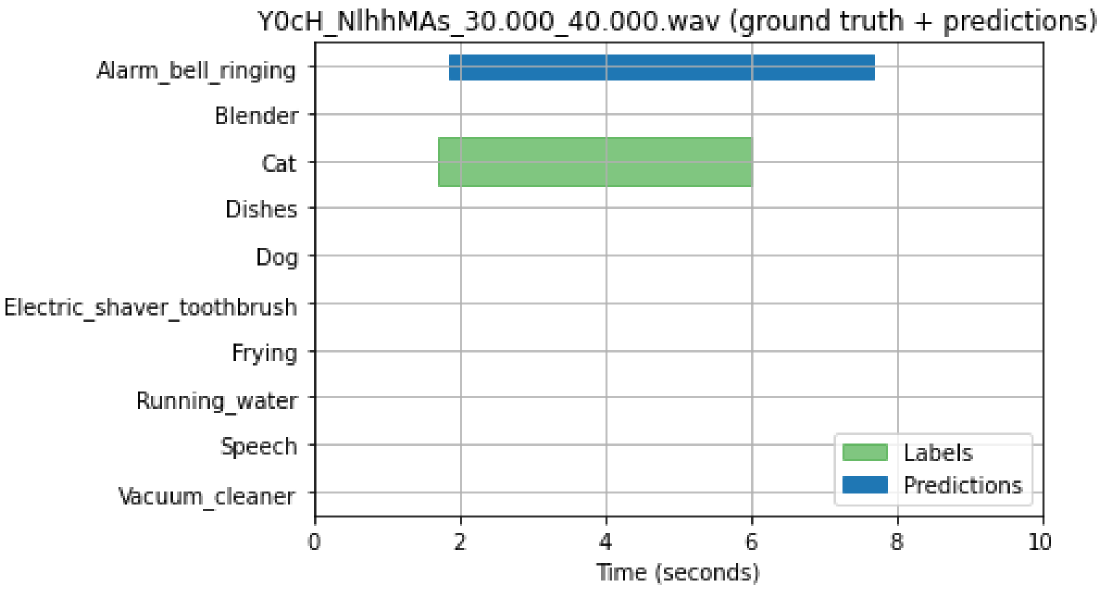
We now select two different audios (the code is ready to change the number of audios and select K new audios) and plot the labels and predictions of our system.

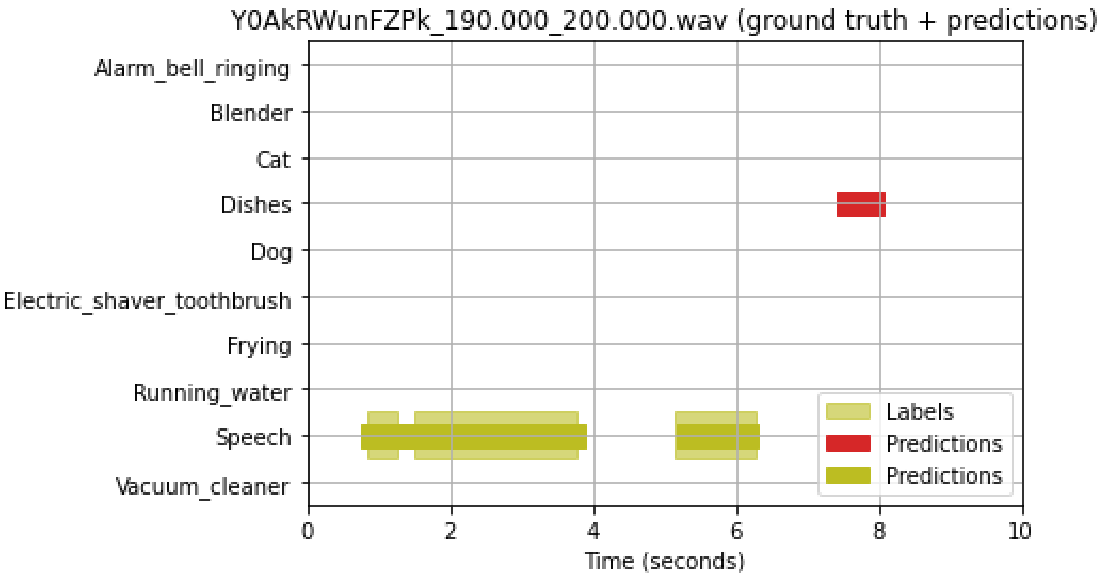
[11]:
```
np.random.seed(234)
new_choices = np.random.choice(audio_names,2)
```

```
print("New choice filenames are: {}, {}".format(new_choices[0], new_choices[1]))
```

```
New choice filenames are: Y0cH_NlhhMAs_30.000_40.000.wav,
Y0AkRWunFZPk_190.000_200.000.wav
```

```
[12]: for file in new_choices:
          plot_labels_predictions(file, legend_location = 'lower right')
```





As we can see, in the **first** presented audio there are two mistakes:

- Firstly, the model is predicting a wrong label (it should be *cat*, it is predicting *alarm_bell_ringing*).
- Secondly, the prediction lasts longer than the actual label, leading to an *insertion*.

In the **second** audio, we see that the system is predicting the *speech* well, but it is inserting a *dishes* event that it is non existent.

3 Conclusions

In this assignment we have learned to plot signals, mel-spectrograms, labels and predictions from a model. We have tested a specific DCASE system that tries to predict certain events in audios, discovering that the model creates many *insertions and deletions* in the audios, leading to low F_1 score.