Water Leak Detection System

SRIDEVI NARAYAN

Computing for AI & Machine Learning
CCE AUG – DEC 2022
IISC, Bangalore

1 Summary

The **water leak detection system** aims at identifying water leaks in households. It encompasses an *end-to-end system from constantly monitoring the pipes, collecting samples at some intervals,* ***analysing the samples to detect leaks*** and provide notifications. The focus of this project is to analyse the samples to detect leaks.

Since the end-to-end system is not in scope, audio samples had to be collected manually. About 7 features were extracted from these audio samples and fed into a 3-layer neural network.

The samples were split into (80%-20%), (70%-30%) and (60%-40%) train-test partitions. The training was done on the training set and then evaluated. on the test set. The experiments were carries out with epochs from 20 to 100 – a total of 9 iterations. During validation, an a maximum accuracy of 75% was achieved.

The trained neural network was then used to predict the leaks using the test set again. The precision and recall metrics were used to analyse. A precision of **75% and 90%.** and achieved and a recall of approx. 50% was achieved. This meant that the false negatives were higher for all cases except with a test-slit (80%-20%)

This is not sufficient to deploy the system. Further work must be carried out to use more data samples, improve the metrics, drop some features or even check a different algorithm.

2 Project Description and Scope

2.1 Problem Statement

At homes, we find that water is wasted. Mainly attributable to

- Water leaks due wear and tear of the plumbing
- Or taps are not closed properly.
- Or overflow

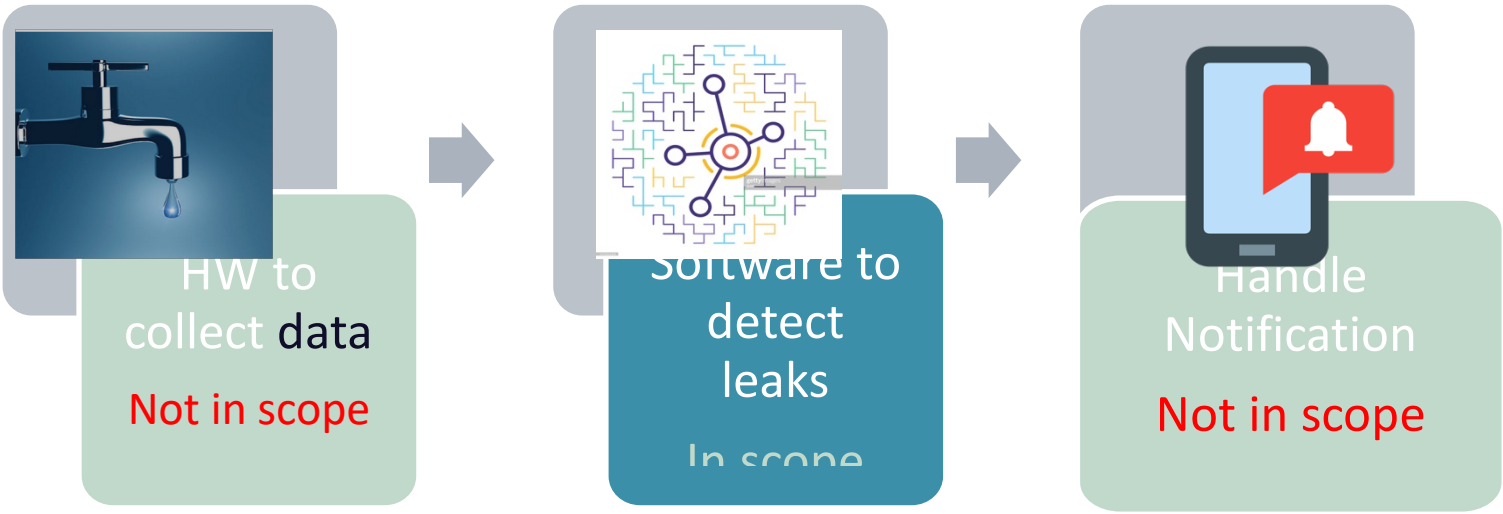
2.2 Proposed Solution

Water Leak Detection System
A non-invasive system is envisaged which can provide notifications on detecting a leak. The HW is supposed to clamp onto water pipes and monitor it continuously. Data is sent to the main system for further analysis. If a leak is detected, notifications are sent to the concerned people (configurable) for further action.

2.3 Scope

The scope of the project is limited to analyzing the audio files using machine learning techniques. In addition, collection of data in terms of manually recorded sample files is included in the scope.

The volume of data feeding into the whole system is expected to be pretty high. The scalability, reliability and robustness of this system is not included in t



3 Data Collection and Preparation

The audio recording on the iphone was used to collect data. Different types of 10-sec sounds were recorded, on different days and different times so that the ambient noise was not the same. Some recordings that could be mistaken for a leak (like a train chugging or the pressure cooker letting out steam in bursts) were also include. These files were in .m4a format. This was converted to .wav format (using a trial version of smartconverter).
Each individual sound file was played, and manually classified. The classification was incorporated into the file name.

A total of 59 samples were collected. The distribution of the samples is given in (Figure 1)

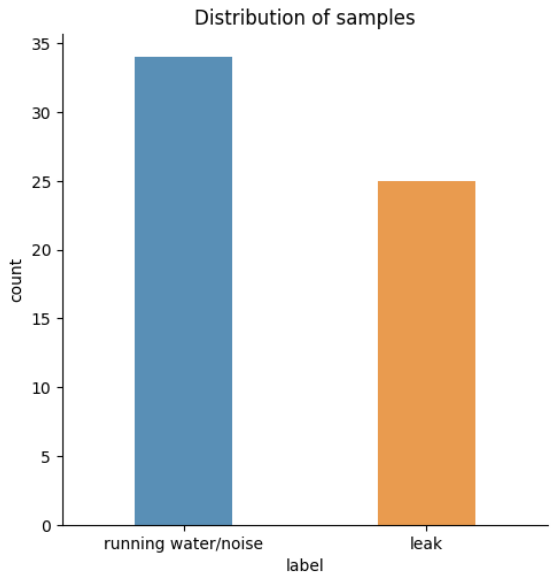


Figure 1 Distribution of Audio samples

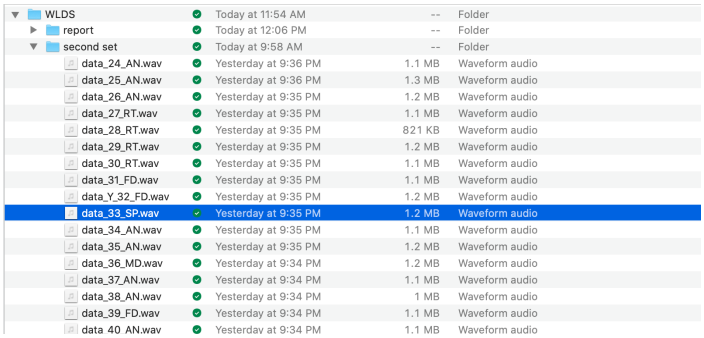


Figure 2 File Naming Convention

The following convention was used(Figure 2):
Consider the file "data_33_SP.wav"

Component	Example	Explanation
Data_	"data_33_SP.wav"	**Prefix for all data files**
nn	"data_33_SP.wav"	Sequence number of the sample, separated by underscore
XX	"data_33_SP.wav	These two characters help classify the file: 'RW', 'RT', 'HW' – running water, running tap or water from a height respectively. These are all classified as 'run' 'FD', 'MD', 'SD', 'SP', 'HD' – fast drip, medium drip, slow drip, spray and drip from a height are all classified as a 'leak'

		'WN', 'AN' – White noise and ambient noise are classified as ‘noise’
Ext	“data_33_SP.wav	The type of sound file. The .m4a were converted to .wav files for ease of use.

3.1 Feature Selection

From the literature, it was found that the following features contributed most in separating out the sounds in the audio file. The library ‘librosa’ has been used extensively to extract features.

1. **MFCC** : Mel Frequency Cepstrum (MFC) encodes the power spectrum of a sound whereas the Mel-frequency cepstral coefficients [MFCCs] collectively make up the MFC. MFCC coefficients model the spectral energy distribution in a persistent and meaningful way, thus they are most widely feature for audio classification

   y : NumPy array for the audio signal
   sr : sampling rate of the signal y. By default, Librosa gives us a sampling rate of 22050.

1. **spectral_centroid**: The spectral centroid indicates at which frequency the energy of a spectrum is centered upon or in other words It indicates where the ” center of mass” for a sound is located
2. **spectral_bandwidth** : The spectral bandwidth or spectral spread is derived from the spectral centroid. It is the spectral range of interest around the centroid, that is, the variance from the spectral centroid. It has a direct correlation with the perceived timbre. The bandwidth is directly proportional to the energy spread across frequency bands
3. **spectral_rolloff:** It is a measure of the shape of the signal. It represents the frequency at which high frequencies decline to 0.
4. **Onset strength/Spectral flux:,** Spectral flux measures the spectral change between two successive frames
5. **zero_crossing_rate: is** simply the number of times a waveform crosses the horizontal time axis. This feature has been primarily used in recognition of percussive vs pitched sounds, monophonic pitch estimation, voice/unvoiced decision for speech signals, etc.
6. **Percussive part** – Separating the harmonic and percussive parts. With some experiments, found that margin 16 gave the best separation of the dripping. A sample illustration of
   a. the separation of harmonic and percussive components of the audio file is given in (**Figure 4**) and **Figure 3**

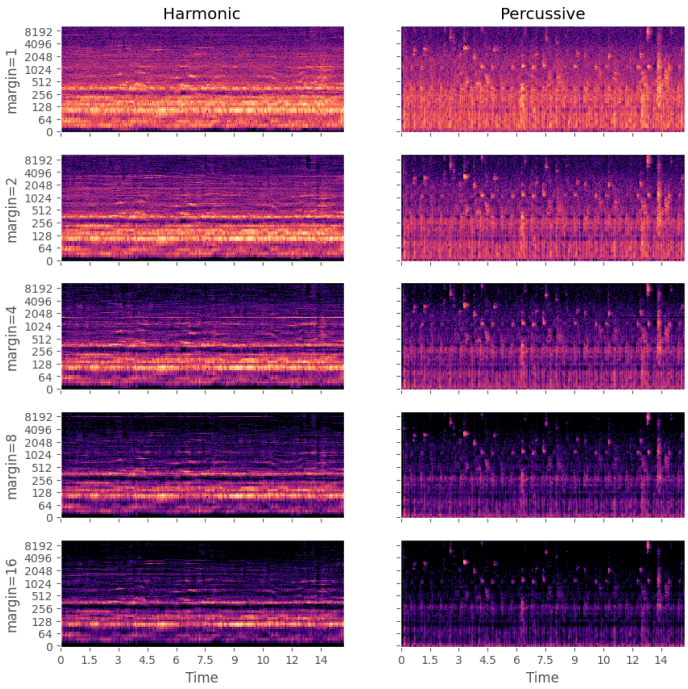


Figure 4Separation of harmonica and percussive components

Figure 3 Audio sample of the percissive component

4 Algorithm Choice

A simple neural network with 3 layers was selected as a first option. Further experiments need to be carried out with other algorithms

```
Parameter and Model Selection
model = Sequential ([
layers.Dense(100, activation='relu'),
layers.Dense(100, activation="relu"),
layers.Dense(2, activation="softmax")
])
model.compile(loss='categorical_crossentropy',
metrics=['accuracy'], optimizer='Adam'
```

- #Nodes were chosen based on literature and some experiments by tinkering with the parameter
- Categorical Crossentropy: This is the preferred loss function that is used for classification problems. The network would be learning to minimize the "distance" between two probability distributions:
    - the first is the output vector which has been scaled by a softmax (all values sum to 1) and
    - the ground truth vector, which is just a probability of 1 for our target class, and a probability of 0 elsewhere.

5 Training and Evaluation

The following strategy was adopted for the training

- Both normalised data and un-normalised data have been considered.
- Training was done for the following three 'train test split' scenarios
    - 80%-20%, 70%-30% and 60%-40%
- For each of the above scenarios, training was run by varying the number of epochs – from 20 to 100 in steps of 10. (i.e. about 9 times)
    a. This was done because the training produced very fluctuating results for the validation data. It was noticed that the data for the training was accidentally run on un-normalised data. And the training epochs did not help in stabilising the fitted model
    b. Eventually, it became interesting to compare the learning for normalized and un-normlized data. Hence both the results are depicted further down

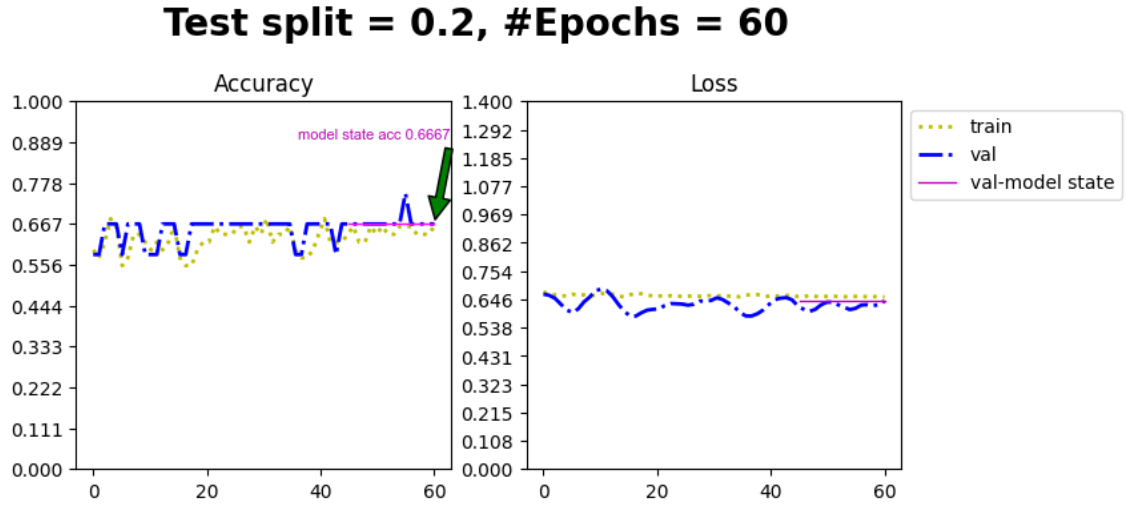


Figure 5 Accuracy and Loss for test split 0.2 and epochs = 60 (Normalised)

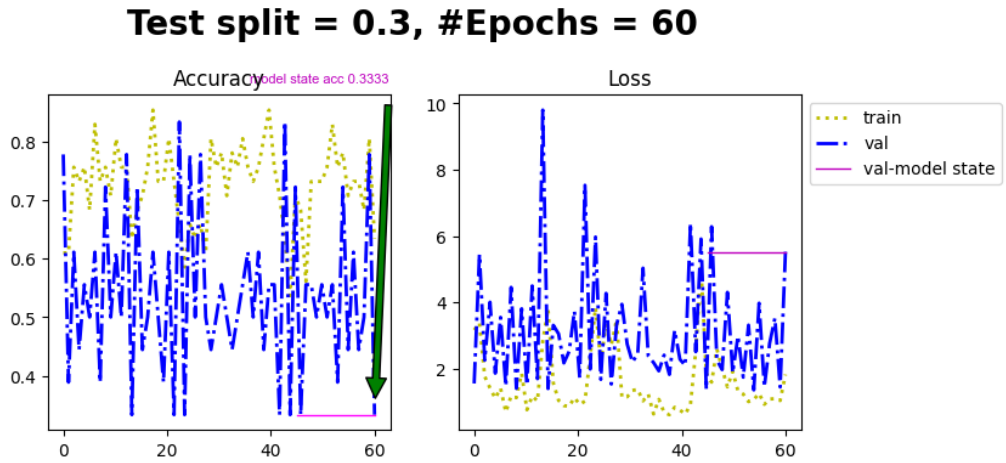


Figure 6 Accuracy and Loss for test split 0.2 and epochs 60 (Unnormalised)

The following conclusions were arrived at:

1. The average of the validation accuracy for normalized and un-normalised was comparable. But this did not reflect how well the model performed for prediction. Hence the average accuracy as a metric was dropped and is not illustrated here.
2. The model state accuracy reflected the state of the model at a particular instant.
3. The validation accuracy and loss for the model was stable for the normalized data. But not for the un-normalised data. It is the pink line in the figures above.(**Figure 5** and **Figure 6**) It reflected a snapshot at the point where the training stopped. If we stopped at a different point, this metric could potentially give a very different result. Hence no conclusions could be drawn from it.
4. The behaviour of the metric for each epoch was considered. This also reiterated that the training was more stable for normalized data, and where we stop did not vary the result +/- 10%. Whereas where the training stopped for un-normalised data was very important. The accuracy in Figure 6 varied from 20% to 80%. This is also reflected in Figure 7
5. Again, the loss metric was also very stable for normalized Figure 5. Figure 6 shows that the loss was higher for un-normalized data as compared to normalized data. Further, the loss function was worse for validation set than for training set for unnormalized data.
6. The loss was very low for all epochs and all test-sets for normalized data. For un-normalised data, the loss also fluctuated and it mattered how many epochs you trained for.
7. In conclusion, normalization is very important for training and prediction.

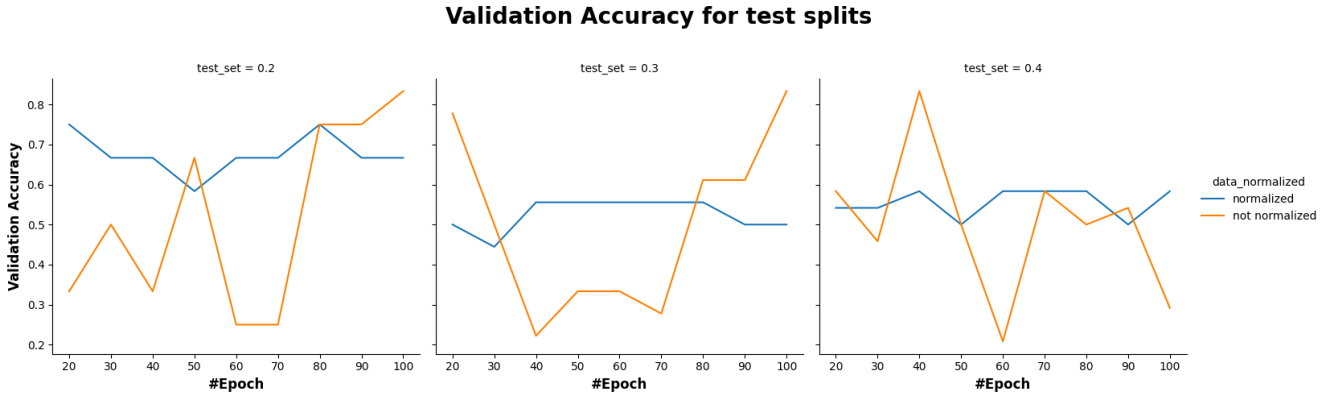


Figure 7 Consolidated Accuracy for all experiments

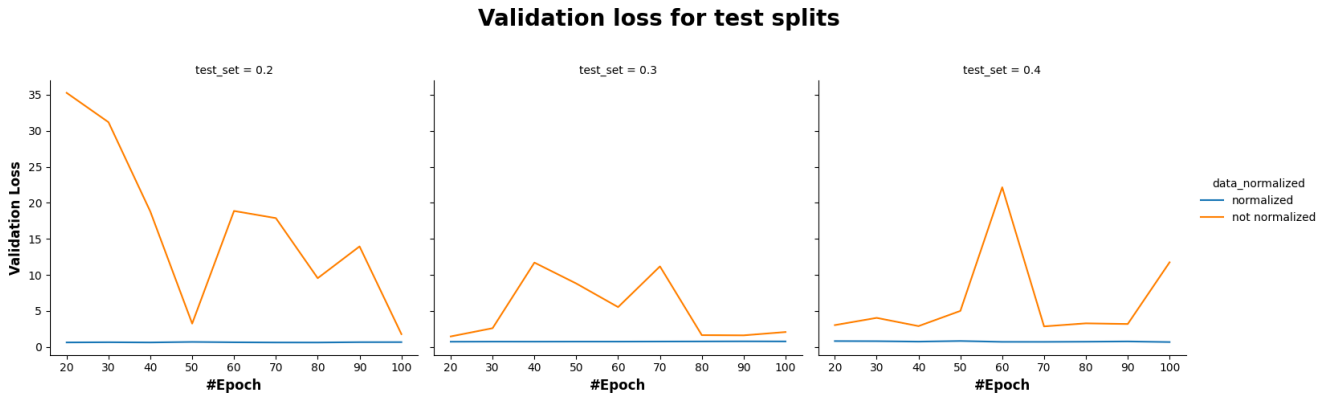


Figure 8 Consolidated Loss for all experiments

6 Prediction

The test-set was used with *model.predict()*_. This was done for all the scenarios and epochs mentioned earlier.

The following two metrics are analysed for the prediction.

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Where, *tp* are the total positives, *fp* are false positives and *fn* are the false negatives

1. The precision for normalized data is consistently between 75% and 90%. (Figure 9) This implies that the false positives are lower.
2. The recall for normalized data is lower and between 45%-50% for most experiments but higher for test split with 0.2. (Figure 10). This implies that the false negatives are higher for all cases except test-split with 0.2

The plots for the same are given below as well as a subset of the tabular data (Figure 11):

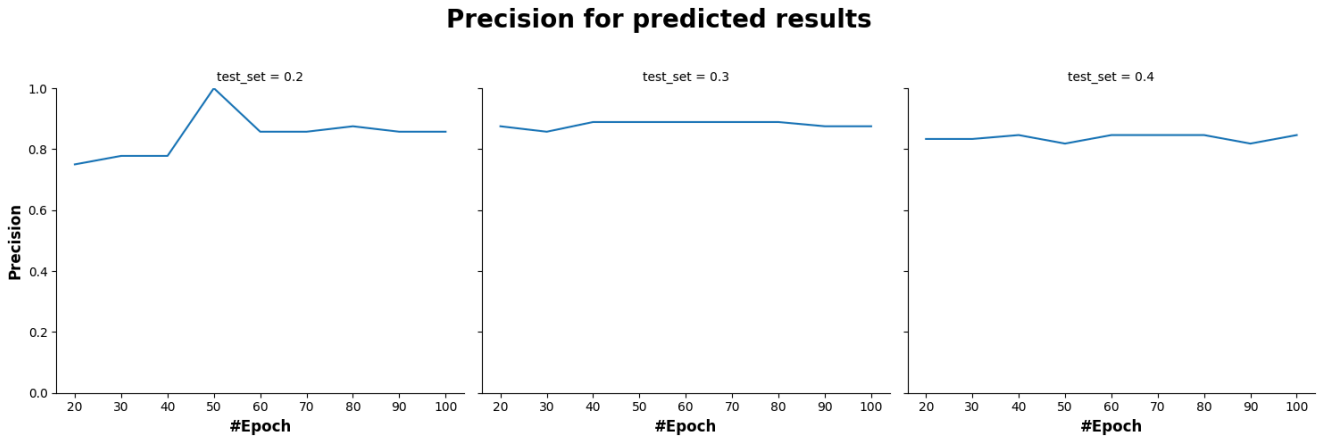
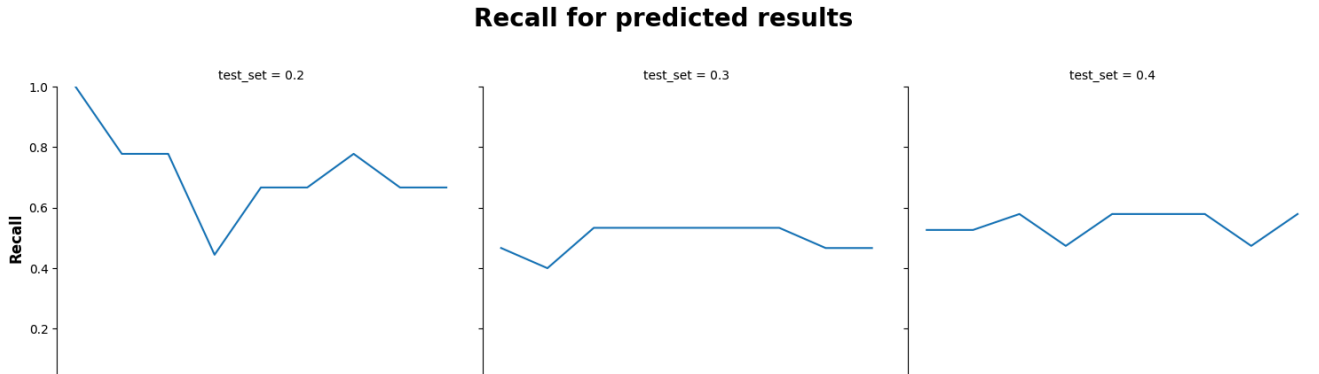


Figure 9 Precision



	data_normalized	test_set	tr_epoch	pred	label	TP	TN	FP	FN	Precision	Recall
0	normalized	0.2	20	[1,1,1,1,1,1,1,1,1,1,1]	[1,1,0,1,1,0,1,1,0,1,1]	9	0	3	0	0.75	1.0
1	normalized	0.2	30	[1,1,1,1,1,1,0,1,0,1,1,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	7	1	2	2	0.7777777777777780	0.7777777777777780
2	normalized	0.2	40	[1,1,1,1,1,1,0,1,0,1,1,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	7	1	2	2	0.7777777777777780	0.7777777777777780
3	normalized	0.2	50	[0,1,0,1,0,0,0,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	4	3	0	5	1.0	0.4444444444444440
4	normalized	0.2	60	[0,1,1,1,1,0,1,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	6	2	1	3	0.8571428571428570	0.6666666666666670
5	normalized	0.2	70	[0,1,1,1,1,0,1,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	6	2	1	3	0.8571428571428570	0.6666666666666670
6	normalized	0.2	80	[1,1,1,1,1,0,1,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	7	2	1	2	0.875	0.7777777777777780
7	normalized	0.2	90	[0,1,1,1,1,0,1,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	6	2	1	3	0.8571428571428570	0.6666666666666670
8	normalized	0.2	100	[0,1,1,1,1,0,1,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	6	2	1	3	0.8571428571428570	0.6666666666666670
9	normalized	0.3	20	[0,1,1,1,1,0,0,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	7	2	1	8	0.875	0.4666666666666670
10	normalized	0.3	30	[0,1,1,1,0,0,0,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	6	2	1	9	0.8571428571428570	0.4
11	normalized	0.3	40	[0,1,1,1,1,0,1,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	8	2	1	7	0.8888888888888890	0.5333333333333330
12	normalized	0.3	50	[0,1,1,1,1,0,1,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	8	2	1	7	0.8888888888888890	0.5333333333333330
13	normalized	0.3	60	[0,1,1,1,1,0,1,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	8	2	1	7	0.8888888888888890	0.5333333333333330
14	normalized	0.3	70	[0,1,1,1,1,0,1,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	8	2	1	7	0.8888888888888890	0.5333333333333330
15	normalized	0.3	80	[0,1,1,1,1,0,1,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	8	2	1	7	0.8888888888888890	0.5333333333333330
16	normalized	0.3	90	[0,1,1,1,0,0,1,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	7	2	1	8	0.875	0.4666666666666670
17	normalized	0.3	100	[0,1,1,1,0,0,1,0,1,0,0,1,1,0,0,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1]	7	2	1	8	0.875	0.4666666666666670
18	normalized	0.4	20	[0,1,1,1,0,0,0,0,1,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	10	3	2	9	0.8333333333333330	0.5263157894736840
19	normalized	0.4	30	[0,1,1,1,0,0,0,0,1,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	10	3	2	9	0.8333333333333330	0.5263157894736840
20	normalized	0.4	40	[0,1,1,1,0,0,1,0,1,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	11	3	2	8	0.8461538461538460	0.5789473684210530
21	normalized	0.4	50	[0,1,1,1,0,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	9	3	2	10	0.8181818181818180	0.47368421052631600
22	normalized	0.4	60	[0,1,1,1,0,0,1,0,1,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	11	3	2	8	0.8461538461538460	0.5789473684210530
23	normalized	0.4	70	[0,1,1,1,0,0,1,0,1,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	11	3	2	8	0.8461538461538460	0.5789473684210530
24	normalized	0.4	80	[0,1,1,1,0,0,1,0,1,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	11	3	2	8	0.8461538461538460	0.5789473684210530
25	normalized	0.4	90	[0,1,1,1,0,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	9	3	2	10	0.8181818181818180	0.47368421052631600
26	normalized	0.4	100	[0,1,1,1,0,0,1,0,1,0,0,1,1,0,0,0,0,1,1,0,1,1,1,1]	[1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,0,1,1]	11	3	2	8	0.8461538461538460	0.5789473684210530
27	not normalized	0.2	20	[0,0,0,0,0,0,0,0,1,0,0,0]	[1,1,0,1,1,0,1,1,1,0,1,1]	1	3	0	8	1.0	0.1111111111111110
28	not normalized	0.2	30	[0,0,0,1,0,0,0,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	3	3	0	6	1.0	0.3333333333333330
29	not normalized	0.2	40	[0,1,0,0,0,0,0,0,0,0,0,0]	[1,1,0,1,1,0,1,1,1,0,1,1]	1	3	0	8	1.0	0.1111111111111110
30	not normalized	0.2	50	[0,1,1,1,1,0,1,0,1,0,0,1]	[1,1,0,1,1,0,1,1,1,0,1,1]	6	2	1	3	0.8571428571428570	0.6666666666666670
31	not normalized	0.2	60	[0,0,0,0,0,0,0,0,0,0,0,0]	[1,1,0,1,1,0,1,1,1,0,1,1]	0	3	0	9	-1.0	0.0

Figure 11 Prediction Results

7 Next steps

1. Correlate and analyse ‘Accuracy’ in the metrics against precision and recall to increase these metrics

2. Model experiments- both algorithms and parameters. Focus on improving accuracy and recall. In addition, try out image identification and classification for the spectral representation of the features, especially mfcc.

3. Complete system with HW and notifications included.

8 Tools and Libraries used

- PYTHON - Jupiter NB
- LIBROSA
- PANDAS
- MATPLOTLIB
- NUMPY
- TENSORFLOW/ KERAS
- SEABORN
- GITHUB

9 References

[1.] How I Understood: What features to consider while training audio files?
https://towardsdatascience.com/how-i-understood-what-features-to-consider-while-training-audio-files-eedfb6e9002b

[2.] Audio Data Processing— Feature Extraction — Essential Science & Concepts behind them
https://medium.com/analytics-vidhya/audio-data-processing-feature-extraction-science-concepts-behind-them-be97fbd587d8

[3.] Audio Data Analysis Using Deep Learning with Python (Part 1)
Nagesh Singh Chauhan
https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html

[4.] Environmental sound recognition: a survey
sachin chachada and c.-c. jay kuo
https://www.cambridge.org/core/services/aop-cambridge-core/content/view/S2048770314000122

[5.] Machine Learning for Audio Classification
Willies Ogola
https://www.section.io/engineering-education/machine-learning-for-audio-classification/

[6.] Classification of Audio Samples with LogisticMonitoring: Classification of Audio Samples with Logistic Regression, K-Nearest Neighbor, Random Forest and SupportRegression, K-Nearest Neighbor, Random Forest and Support
Vector Machine
Prakhar Amlathe, Utah State University
https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=8156&context=etd

[7.] The Best Metric to Measure Accuracy of Classification Models
https://clevertap.com/blog/the-best-metric-to-measure-accuracy-of-classification-models/

[8.] Precision and recall
https://en.wikipedia.org/wiki/Precision_and_recall