SGN-16006, AUDITORY SCENE RECOGNITION

Jonas Nikula, Vili Saura

240497, 240264
jonas.nikula@student.tut.fi, vili.saura@student.tut.fi

1. INTRODUCTION

In this laboratory assignment we implement a k-nearest neighbor classifier and try to classify audio data collected in different environments. First we preprocess the data and extract the relevant features, and then we run them through our classifier. We've used Python 3.6 in this assignment.

2. METHODS

2.1. Feature extraction

We load the audio signals data using the Python module soundfile. This command gives us the actual data and the sample rate used. The sample rate in all the data was 8000Hz.

The features we want are the relative energy ratios of 4 frequency bands: $0.0-0.5kHz$, $0.5-1.0kHz$, $1.0-2.0kHz$ and $2.0-4.0kHz$. We get this by first dividing the audio signal into frames with a length of 30ms, and an overlap of 15ms. With the sampling rate used this means that one frame has 240 samples.

These frames are then multiplied, or windowed using the Hanning window function. In figure 1 there's the waveform of the 50th windowed frame.

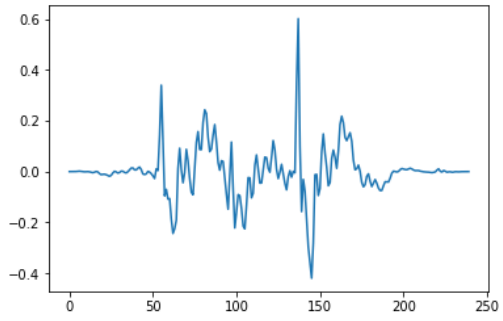

Fig. 1: 50th Hanning-windowed frame

Then we get the discrete fourier transform of the windowed frames, with a bin size of 1024. In figure 2 there's the amplitude spectrum of the 50th frame.

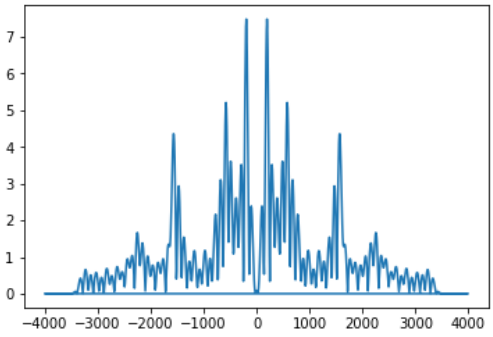

Fig. 2: Amplitude spectrum of the 50th frame

Out of the frame's DFT we extract the values corresponding to the 4 frequency bands. For example, the indices of the frequency band $1-2kHz$ are 128 to 256. Then we calculate the energy ratio of each band. The energy ratio is calculated by dividing the energy of a subband by the total energy of that frame. In figure 3 there's a visualization of the 50th frames extracted features.

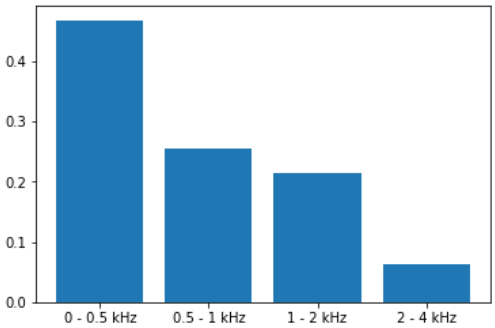

Fig. 3: Features of the 50th frame

Finally, we combine all the frames of a sample by taking the average of all the frames energy ratios.

2.2. k-Nearest neighbor classification

When we have extracted the features, we implement a k-nn classifier as a Python object. Basically, in the training phase

we just copy the data and labels. The real work is done in the prediction function, where we calculate all the distances between the sample whose label we're predicting, and all the training samples. When we have the distances, we take the k-lowest (ie. nearest), and out of those samples we take the most common.

First we classify using only one neighbor. The accuracy was reported by scikit learn as 69%. In figure 4 we can see the confusion matrix. The classifier seems to have problems with label number 7, but all in all the results are pretty good.

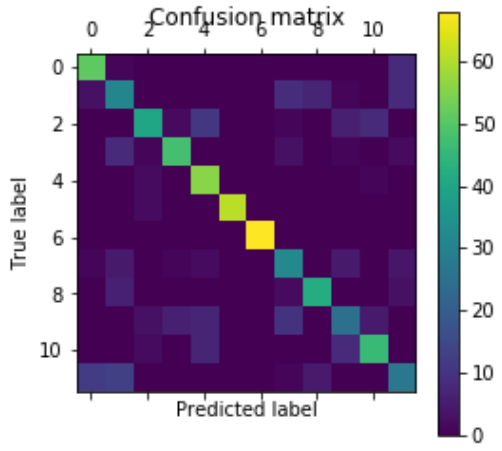

Fig. 4: Confusion matrix of k=1 classifier

Then we run the classifier with 9 neighbors. This time the accyracy was 70%. Figure 5 has the confusion matrix of that classifier. No class sticks out as being very difficult to classify.

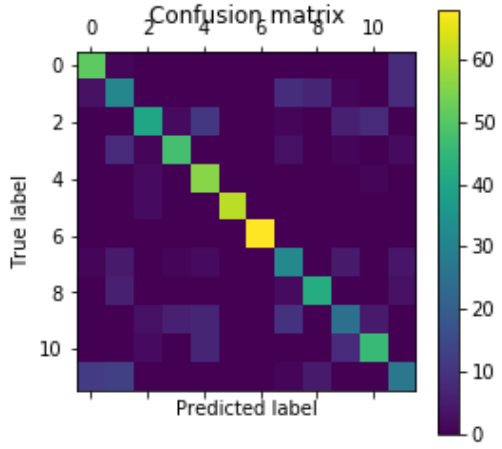

Fig. 5: Confusion matrix of k=9 classifier

3. CONCLUSIONS

In conclusion the classifier was pretty good at classifying different environments, considering how simple the implementation was. In realistic situations nearest neighbor isn't the greatest classifier because a) it isn't that good and b) it takes a long time. It's pretty much the slowest classifier at classifying, because it has to go through the whole training data each time it classifies something.

3.1. Feedback

The instructions were easy enough to follow. I don't know if I learned anything super useful, as framing and nearest neighbor classifiers are pretty basic info. It took us about 4-5 hours to done the whole assignment