Pattern Recognition Winter Term 2019

Institute of Neural Information Processing

PD Dr F. Schwenker

Assignment 2 (Submission until November 12, 2019)

Exercise 1 (10 points): Continuous Naive Bayes Classifier [Python]

In this part of the exercise, we want to apply the naive Bayes classifier to a continuous feature. We are building a small example application which classifies regions in audio signals as either silence or voice. That is, we want to map each position in the audio signal to one of the two classes. As a feature, we use the energy of the audio signal which is defined as the quadratic sum over a window of size $w > 0$

$$E(t) = \sum_{i=t}^{\min(t+w,N)} s(i)^2 \tag{1}$$

with t as the current position in the signal and N as the number of samples. This is a one-dimensional feature which we can calculate at every position in the signal. A possible solution for this exercise is shown in Figure 1.

1. What is the level of measurement (scale) of the energy feature?
2. Start by recording two short sentences. Don't make your audio recordings too long, a few seconds are enough. One file will be used to train and the other to test the classifier.
3. Now, we can start and import the audio files.
   a) To load the files in your Python program, you can use the `scipy.io.wavfile.read()` function. Store also the sample rates as we need them later. Note: when you recorded in stereo, use only the first channel.
   b) Normalize the values of your audio signals to the $[-1; 1]$ range. For this, you need to handle negative and positive values separately and divide by the absolute minimum and maximum possible value, respectively. The documentation lists these extrema values for different data types.
4. Calculate the energy according to Equation 1 for both files. Use a window size of $\tilde{w} = 10$ ms. Note that the unit of the window size w in Equation 1 is in samples (and not milliseconds); so, the value w you need to use depends on the sampling rate of your recorded audio signals.
5. Training phase: use the first audio file in the following subtasks.
   a) Manually label the regions of silence in your first audio file. You can use any external tool for this task.
   b) Store the information of the labelling process in your code e.g. in a boolean array which has the same length as your signal (1 = silence):

```
# Manually mark silence regions in the training file
silence = np.zeros(len(signalTrain), bool) # Logical map
silence[0:6000] = 1 # Alternatively, apply a threshold
silence[12100:14900] = 1
silence[19100:23000] = 1
silence[27500:32700] = 1
silence[35000:len(signalTrain)] = 1
```

c) Calculate the prior-probabilities $P(Silence)$ and $P(Voice)$.

d) Since we have now continuous features and don't know the precise distribution from which they arise, we cannot calculate the likelihoods $p(x|\omega_i)$ directly (as it was the case in the fruit classifier). We will rather assume that the energy values from each class are normally distributed and infer the parameters form the data itself.

   Build two distributions, one for the energy values from the silence and one for the energy values in the voice regions of your audio file. For this, create `scipy.stats.norm` objects with the required parameters mean μ and standard deviation σ calculated from the corresponding energy values (there are functions in numpy for this job).

6. Testing phase: use the second audio file in the following subtasks.

   a) Classify each energy value from the test file as silence or voice and store the result (e.g. again in a boolean array). To calculate $p(x|\omega_i)$, you can use the `pdf` method of the `norm` object.

   b) Store also the scaled likelihoods used in the classification process, i.e. $p(x|Silence) \cdot P(Silence)$ and $p(x|Voice) \cdot P(Voice)$, in additional arrays so that you can plot them in the next step.

7. Plot the energy values and show the labelled respectively classified classes for both audio files. Additionally, plot the normal distributions from the training phase and the scaled likelihoods values used in the classification process over the signal range.

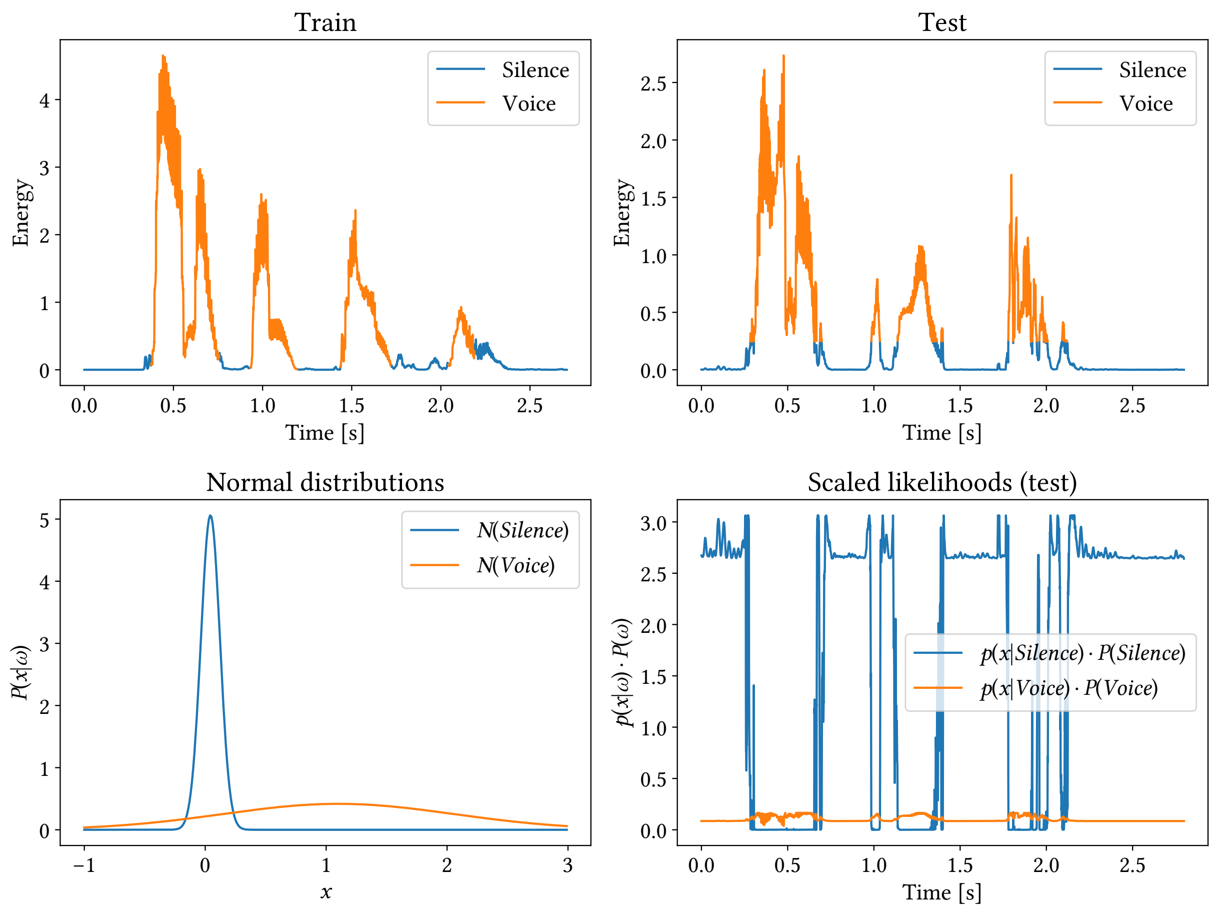

Figure 1: Possible solution for the silence/voice-classifier. Note that the plots depend highly on the used audio files and hence serve only as a general hint. The coloured functions (plots in the first row) are created with the help of NumPy's `MaskedArray` class.