Assignment 1 Lab Report

November 20, 2024

Name	Christian Niederreiter
Matriculation Number	K0726258

1 Vehicle Categorization

1.1 Data Recording

1.1.1 Setting

Data was recorded at the bus stop "Linz/Donau JKU Universität (Altenberger Straße)" for buses to the main station (the bus stop is located at the northwest sidewalk of the street).

Recording Device The recording device was a smartphone (iPhone 8, iOS 16.7.10), held in the hands while sitting on the bus stop bench, power connector and microphone facing down.

Date and Time October 30, 2024 at 13:57

1.1.2 Resulting Video

Duration The resulting video has a duration of 13 minutes and 23 seconds.

Audio Channels The resulting video has only a single audio channel (mono).

1.1.3 Labeling

The majority of vehicles were passenger cars of various sizes (only 1 motorcycle and 1 bus). For this reason, I decided to use the following categories and labels for labeling:

Vehicle Size Class	Label	Description
light	**s** for *small*	lighter than compact cars
medium	**m** for *medium*	compact cars
heavy	**h** for *heavy*	sport utility cars, vans, buses,...

(a compact car belongs to the "C-segment" in the European car classification)

Some vehicles turned out to be ambiguous in terms of their size class when I tried to label them. Thus several labels are known to be questionable.

1.2 Pre-Processing

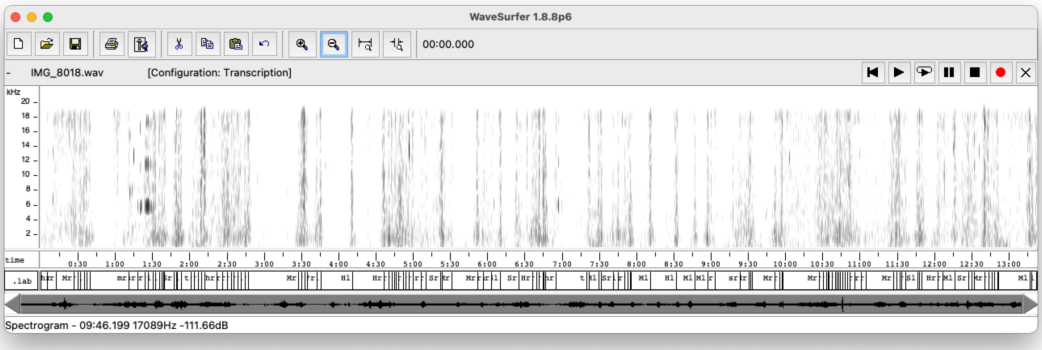
1.2.1 Extracting WAV

The video file was converted to a WAV file using `ffmpeg`:

```
$ ffmpeg -i IMG_8018.mov IMG_8018.wav
```

1.2.2 Noise Reduction

The following screenshot of WaveSurfer suggests that the vehicle sound spectrum covers the entire frequency range up to 20 kHz.



Due to this finding, in order to prevent information loss, I decided to leave out any noise-reducing low or high pass filtering and keep the original audio for feature extraction.

1.3 Segmentation

1.3.1 Tools

To extract individudal vehicle samples from the WAV file, they were labeled using WaveSurfer and IINA video player:

Tool	Purpose
WaveSurfer	Labeling
IINA video player	determine split seconds of vehicle exactly in front of recording device

1.3.2 Procedure

For the sake of quickness, single-letter labels were used:

- first letter: vehicle size s, m or h (case insensitive)
- second letter (irrelevant for current task): l or r (coming from left or right)

First lines of the label file (IMG_8018.lab):

```
0.0000000 5.4965836 hr
5.4965836 12.0748003 hr
12.0748003 26.8034101 Mr
26.8034101 29.9975989 Mr
29.9975989 32.1085410 Hl
32.1085410 34.8722087 Sl
34.8722087 36.8581609 Hl
36.8581609 39.4551752 Ml
39.4551752 70.8415518 mr
70.8415518 77.8274389 sr
77.8274389 83.7417866 Sr
83.7417866 86.5197378 Sr
86.5197378 90.6866645 Ml
...
```

Meanings of the columns:

Column	Meaning
Column 1	begin time of the label in seconds
Column 2	end time of the label in seconds
Column 3	label

The end time was chosen to be the time when the vehicle was nearest to the recording device (exactly behind the smartphone). Sound samples were taken from the end time, surrounding the end point in time +/- half sample duration. The begin time was ignored. Each sample has the same length.

1.3.3 Audio File Segmentation

The following shell script was used to perform the audio file segmentation according to the labels:

```
#!/bin/bash
path="$1"
duration_seconds="$2"
filename_extension="${path##*.}"
while read -r ignored label_seconds label
do
  # $start_seconds = $label_seconds - ($duration_seconds * 0.5):
  start_seconds=$( echo -e "$label_seconds\n$duration_seconds\n0.5\n*\n-\np\nq" | dc )
  ffmpeg -i "$path" -ss "0$start_seconds" -t "0$duration_seconds" \
    -c copy "${label}_${label_seconds}.${filename_extension}"
  echo
done
```

A duration of 1 second turned out to deliver the best results. Thus the samples used for further processing have an audio length of 1 second.

1.4 Feature Extraction

1.4.1 MFCCs

Mel-frequency cepstral coefficients (MFCCs) were used as basis for the features. The function `librosa.feature.mfcc` (part of the Python library "librosa") was used to compute the MFCCs.

MFCC Computation Configuration The following configuration settings were varied to explore their impacts on the classification correctness:

Configuration Setting	Description	Observations
`n_fft`	length of the FFT window	deviating from the default value (2048) leads to worse classification
`hop_length`	number of samples between successive frames	deviating from the default value (512) leads to worse classification

1.4.2 Reducing the number of values

Each MFCC vector may consist of several tens of values, depending on the lengths of the audio samples. Sticking to the function default configuration of 20 MFCC result vectors per audio sample, this may lead to hundreds of values per sample. To reduce this amount of values, each MFCC vector was collapsed to single values, each of them describing a statistical property of the vector. Examples of the 16 statistical properties are *max*, *min*, *average*, *variance*,… (see `STATS_FUNCTIONS` in the code in the appendix).

1.4.3 Attribute Notation

Example Attribute	Meaning
`c1_average`	average of the first MFCC vector
`c20_var`	variance of the 20th MFCC vector
…	…

1.4.4 Choosing the most informative attributes

From the MFCC statistics mentioned above, the most informative ones were used for training. On the WEKA tool's *Weka Explorer / Select attributes* page, the attribute evaluator called **InfoGainAttributeEval** (default settings) was used to get an idea about the worth of the attributes by measuring the information gain with respect to the class.

1.5 Classification Results

When mixing the directions and taking vehicles from the left and from the right as training samples, correctly classified vehicles are roughly 50%, which is similar to a wild guess. The ambiguity problems mentioned in *Data Recording / Labeling* are a likely explanation for such bad results, which are already reflected by the low number of attributes suggested by **InfoGainAttributeEval**

(1 to 2). When limiting the training samples to vehicles *coming from right only*, the result gets slightly better. I put that result as an example here:

See the file `classification_results.small_medium_heavy.rightonly.txt` for detailed result logs.

Top 2 attributes of the **InfoGainAttributeEval** result:

```
Ranked attributes:
 0.168    248 c16_stddev
 0.168    247 c16_var
```

All others have a value of 0 and still using them led to worse classifications.

Choosing those two attributes lead to the following classification results of four different classifiers:

1.5.1 IBk (k Nearest Neighbors)

IBk performs worst.

```
Correctly Classified Instances          35               42.6829 %
Incorrectly Classified Instances        47               57.3171 %
```

1.5.2 NaiveBayes

```
Correctly Classified Instances          45               54.878  %
Incorrectly Classified Instances        37               45.122  %
```

1.5.3 J48 (Decision Tree)

```
Correctly Classified Instances          45               54.878  %
Incorrectly Classified Instances        37               45.122  %
```

1.5.4 MultilayerPerceptron

MultilayerPerceptron performs best with 57% correctly classified instances. Still the confusion matrix reveals heavy vehicles were never recognized which renders classification into 3 classes useless.

```
Time taken to build model: 0.04 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          47               57.3171 %
Incorrectly Classified Instances        35               42.6829 %
Kappa statistic                          0.2247
Mean absolute error                      0.3974
Root mean squared error                  0.4512
Relative absolute error                 93.5426 %
Root relative squared error             97.9443 %
Total Number of Instances               82
```

```
=== Detailed Accuracy By Class ===

     TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
     0,391    0,034    0,818      0,391    0,529      0,471    0,588     0,524     s
     0,974    0,767    0,535      0,974    0,691      0,303    0,501     0,447     m
     0,000    0,000    ?          0,000    ?          ?        0,500     0,256     h
WAvg 0,573    0,375    ?          0,573    ?          ?        0,525     0,422

=== Confusion Matrix ===

  a  b  c   <-- classified as
  9 14  0 |  a = s
  1 38  0 |  b = m
  1 19  0 |  c = h
```

2 Driving Direction Recognition

2.1 Data Recording

see *Data Recording* of *Vehicle Categorization* (same procedure)

2.2 Segmentation

2.2.1 Tools

see *Segmentation* of *Vehicle Categorization*

2.2.2 Procedure

For the sake of quickness, single-letter labels were used:

- first letter (irrelevant for current task): vehicle size s, m or h (case insensitive)
- second letter: l or r (coming from left or right)

First lines of the label file (IMG_8018.lab):

```
0.0000000 5.4965836 hr
5.4965836 12.0748003 hr
12.0748003 26.8034101 Mr
26.8034101 29.9975989 Mr
29.9975989 32.1085410 Hl
32.1085410 34.8722087 Sl
34.8722087 36.8581609 Hl
...
```

For further information, see *Segmentation* of *Vehicle Categorization*.

2.2.3 Audio File Segmentation

see *Segmentation* of *Vehicle Categorization*

2.3 Feature Extraction

see *Feature Extraction* of *Vehicle Categorization*

2.4 Classification Results

See the file `classification_results.left-to-right_right-to-left.txt` for detailed result logs.

67 top attributes of the **InfoGainAttributeEval** result:

```
Ranked attributes:
 0.4366     12 c1_quartile2
 0.4229     13 c1_quartile3
 0.4229      6 c1_average
 0.3844      1 c1_max
 0.3335     11 c1_quartile1
 0.2995     50 c4_min
 0.2909     14 c1_iqr1-2
 0.2673      2 c1_min
...
 0.0887    303 c19_iqr2-3
 0.0873    187 c12_quartile1
 0.0857    212 c14_maxPos
 0.0813    209 c14_max
 0.0767    196 c13_maxPos
```

Sticking to all those 67 attributes without further investigation is not recommended. This is emphasized by my finding that leaving out several attributes has almost no influence on the classification quality.

2.4.1 IBk (k Nearest Neighbors)

```
Correctly Classified Instances         118               89.3939 %
Incorrectly Classified Instances        14               10.6061 %
```

2.4.2 NaiveBayes

```
Correctly Classified Instances         116               87.8788 %
Incorrectly Classified Instances        16               12.1212 %
```

2.4.3 J48 (Decision Tree)

J48 performs worst.

```
Correctly Classified Instances         114               86.3636 %
Incorrectly Classified Instances        18               13.6364 %
```

2.4.4 MultilayerPerceptron

Once again, **MultilayerPerceptron** performs best.

```
Time taken to build model: 2.44 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         123               93.1818 %
Incorrectly Classified Instances         9                6.8182 %
Kappa statistic                          0.8568
Mean absolute error                      0.0734
Root mean squared error                  0.2389
Relative absolute error                 15.5703 %
Root relative squared error             49.2487 %
Total Number of Instances              132

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,940     0,073     0,887       0,940   0,913      0,858    0,974     0,966     l
               0,927     0,060     0,962       0,927   0,944      0,858    0,974     0,982     r
WAvg           0,932     0,065     0,934       0,932   0,932      0,858    0,974     0,976

=== Confusion Matrix ===

  a  b   <-- classified as
 47  3 |  a = l
  6 76 |  b = r
```

3 Lessons Learned

3.1 Labeling

During my work on the assignment, a new idea about labeling arose:

During labeling, image information was required for identifying a vehicle's class and the vehicle's position at a certain point in time. Since the labeling tool of my choice was WaveSurfer, I needed a second tool (IINA video player) to get image information in conjunction with precise time information. Synchronizing both tools manually was very cumbersome. Thus I thought about using a video editor for labeling instead of WaveSurfer:

3.1.1 Video Editor ShotCut

https://www.shotcut.org

ShotCut is a video editor that supports markers which could serve as labels. They can be quickly added to the video timeline. After saving the project file, they could be extracted from the XML structure of the project file and used instead of the labels generated by WaveSurfer.

4 Appendix

[1]:
```
import numpy as np
from scipy import stats

# inspired by https://github.com/CCCodes/arff-mfcc-scripts

STATS_FUNCTIONS = [
    ('max', lambda _, mfcc: max(mfcc)),
    ('min', lambda _, mfcc: min(mfcc)),
    ('range', lambda d, _: d['max'] - d['min']),
    ('maxPos', lambda d, mfcc: mfcc.tolist().index(d['max'])),
    ('minPos', lambda d, mfcc: mfcc.tolist().index(d['min'])),
    ('average', lambda _, mfcc: np.average(mfcc)),
    ('var', lambda _, mfcc: np.var(mfcc)),
    ('stddev', lambda _, mfcc: np.std(mfcc)),
    ('skewness', lambda _, mfcc: stats.skew(mfcc)),
    ('kurtosis', lambda _, mfcc: stats.kurtosis(mfcc)),
    ('quartile1', lambda _, mfcc: np.percentile(mfcc, 25)),
    ('quartile2', lambda _, mfcc: np.percentile(mfcc, 50)),
    ('quartile3', lambda _, mfcc: np.percentile(mfcc, 75)),
    ('iqr1-2', lambda d, _: d['quartile2'] - d['quartile1']),
    ('iqr2-3', lambda d, _: d['quartile3'] - d['quartile2']),
    ('iqr1-3', lambda d, _: d['quartile3'] - d['quartile1'])
]

def feature_vector_keys(number_of_mfccs):
    for i in range(number_of_mfccs):
        for item in STATS_FUNCTIONS:
            key = f'c{i+1}_{item[0]}'
            yield key
```

[2]:
```
import csv

def selected_attributes_from_file(selected_attributes_file_path):
    # 1 or more spaces between columns
    # third column contains the values
    with open(selected_attributes_file_path, newline='') as␣
 ↪selected_attributes_file:
        csv_rows = csv.reader(selected_attributes_file, delimiter=' ',␣
 ↪skipinitialspace=True)
        for csv_row in csv_rows:
            yield csv_row[2]
```

[3]:
```
import glob
import os
import csv
```

```
CLASSES_SMALL_MEDIUM_HEAVY = ['s', 'm', 'h']
CLASSES_LEFT_RIGHT = ['l', 'r']

def paths_and_small_medium_heavy_labels_from_filenames(wav_files_wildcard):
    paths = glob.glob(wav_files_wildcard)
    for path in paths:
        first_letter_of_basename = os.path.basename(path)[0]
        label = first_letter_of_basename.lower()
        if label == 't': # HINT t = tram; skip tram
            continue
        yield path, label

def paths_and_left_right_labels_from_filenames(wav_files_wildcard):
    paths = glob.glob(wav_files_wildcard)
    for path in paths:
        second_letter_of_basename = os.path.basename(path)[1]
        label = second_letter_of_basename.lower()
        if label == '_': # HINT no left or right given; skip
            continue
        yield path, label
```

[11]:
```
from dataclasses import dataclass

@dataclass
class ArffFileData:
    relation_name: ...
    classes: ...
    number_of_mfccs: ...
    selected_attributes: ...
    feature_vectors: ... # contains list of numeric values, label and WAV file␣
 ↪path

def write_arff_header(opened_arff_file, relation_name, classes,␣
 ↪number_of_mfccs, selected_attributes=None):
    f = opened_arff_file
    print(f'@RELATION {relation_name}', file=f)
    if selected_attributes is None:
        for key in feature_vector_keys(number_of_mfccs):
            print(f'@ATTRIBUTE {key} NUMERIC', file=f)
    else:
        for selected_attribute in selected_attributes:
            print(f'@ATTRIBUTE {selected_attribute} NUMERIC', file=f)
    classes_string = '{' + ','.join(classes) + '}'
    print(f'@ATTRIBUTE class {classes_string}', file=f)
    print(f'@DATA', file=f)
```

```
def write_arff_feature_vector(opened_arff_file, feature_vector, label,␣
 ↪wav_file_path):
    f = opened_arff_file
    feature_vector_without_keys = map(lambda item: f'{item[1]}', feature_vector)
    print(','.join(feature_vector_without_keys) + ',' + label + ',' +␣
 ↪wav_file_path, file=f)

def write_arff(arff_file_path, arff_file_data):
    data = arff_file_data
    with open(arff_file_path, 'w') as f:
        write_arff_header(f,
            data.relation_name,
            data.classes,
            data.number_of_mfccs,
            data.selected_attributes)
        for feature_vector, label, wav_file_path in data.feature_vectors:
            write_arff_feature_vector(f,
                feature_vector,
                label,
                wav_file_path)
```

[5]:
```
from collections import OrderedDict

def compute_stats(mfcc):
    d = OrderedDict()
    for key, f in STATS_FUNCTIONS:
        d[key] = f(d, mfcc)
    return d

def create_feature_vector(mfccs, selected_attributes=None):
    collected_features = OrderedDict()
    for i, mfcc in enumerate(mfccs):
        mfcc_stats = compute_stats(mfcc)
        for item in mfcc_stats.items():
            key = f'c{i+1}_{item[0]}'
            collected_features[key] = item[1] #f'{key}={item[1]}'
    if selected_attributes is None:
        for key in feature_vector_keys(len(mfccs)):
            yield key, collected_features[key]
    else:
        for selected_attribute in selected_attributes:
            yield selected_attribute, collected_features[selected_attribute]
```

[14]:
```
# ASSIGNMENT 1, 1) SMALL, MEDIUM, HEAVY

import librosa
```

```
def recording_to_small_medium_heavy_arff_file():
    relation_name = 'vehicle_audio_small_medium_heavy_1000_ms_rightonly'
    classes = CLASSES_SMALL_MEDIUM_HEAVY
    number_of_mfccs = 20
    selected_attributes = None
    selected_attributes =␣
 ↪list(selected_attributes_from_file('selected_attributes.7.txt'))
    paths_with_labels =␣
 ↪paths_and_small_medium_heavy_labels_from_filenames('parts/1000ms_rightonly/*.
 ↪wav')
    def mfcc_function(data, sampling_rate):
        return librosa.feature.mfcc(
            y=data,
            sr=sampling_rate,
            n_mfcc=number_of_mfccs)
            #n_fft=20000)
            #hop_length=8192)
    def generate_feature_vectors_with_labels():
        for path, label in paths_with_labels:
            y, sr = librosa.load(path)
            mfccs = mfcc_function(y, sr)
            feature_vector = create_feature_vector(mfccs, selected_attributes)
            yield feature_vector, label, path
    feature_vectors = generate_feature_vectors_with_labels()
    data = ArffFileData(relation_name, classes, number_of_mfccs,␣
 ↪selected_attributes, feature_vectors)
    if selected_attributes is None:
        write_arff(f'{relation_name}.arff', data)
    else:
        write_arff(f'{relation_name}.selected_attributes.arff', data)

recording_to_small_medium_heavy_arff_file()
```

[17]:
```
# ASSIGNMENT 1, 2) LEFT, RIGHT

import librosa

def recording_to_left_right_arff_file():
    relation_name = 'vehicle_audio_left_right_1000ms'
    classes = CLASSES_LEFT_RIGHT
    number_of_mfccs = 20
    selected_attributes = None
    selected_attributes =␣
 ↪list(selected_attributes_from_file('selected_attributes_left_right.txt'))
    paths_with_labels = paths_and_left_right_labels_from_filenames('parts/
 ↪1000ms/*.wav')
    def mfcc_function(data, sampling_rate):
```

```
        return librosa.feature.mfcc(
            y=data,
            sr=sampling_rate,
            n_mfcc=number_of_mfccs)
            #n_fft=20000)
            #hop_length=8192)
    def generate_feature_vectors_with_labels():
        for path, label in paths_with_labels:
            y, sr = librosa.load(path)
            mfccs = mfcc_function(y, sr)
            feature_vector = create_feature_vector(mfccs, selected_attributes)
            yield feature_vector, label, path
    feature_vectors = generate_feature_vectors_with_labels()
    data = ArffFileData(relation_name, classes, number_of_mfccs,␣
 ↪selected_attributes, feature_vectors)
    if selected_attributes is None:
        write_arff(f'{relation_name}.arff', data)
    else:
        write_arff(f'{relation_name}.selected_attributes.arff', data)

recording_to_left_right_arff_file()
```

[ ]: