CS522_assignment2_group1

October 23, 2022

1 Imports

[1]:
```
import scipy.io.wavfile as wavfile
from scipy import signal
import numpy as np
from scipy.fftpack import fft, ifft
import cv2
import random
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
```

2 Data preperation

[2]:
```
# Defining variables used for reading the data

classes = ["Alarm", "Blender", "Microwave", "Music", "Silence", "Vacuum"]
samples_per_class = 20
```

2.1 Reading in all data

[3]:
```
# Data is stored in a two dimensional list with first dimension beeing the␣
 ↪class and the second dimension the samples
data = [[] for _ in range(len(classes))]

for cls_number, cls_name in enumerate(classes):
    path = f"Data/{cls_name}/{cls_name}_"
    for i in range(samples_per_class):
        fs,y=wavfile.read(path + f"{i}.wav")
        data[cls_number].append([fs, y])
```

```
C:\Users\Timmy\AppData\Local\Temp\ipykernel_14960\824154626.py:7:
WavFileWarning: Chunk (non-data) not understood, skipping it.
```

```
fs,y=wavfile.read(path + f"{i}.wav")
```

2.2 Removing all frequencies above 10 kHz

For preprocessing the data all frequencies above 10 kHz are removed. It is assumed that these higher frequencies are created by noise and therefore do not hold any information about the data. A comparison between the original data and the preprocessed showed that it still sounds the same and looks similar in the time domain, so it is assumed to be safe to remove these frequencies.

[4]:
```
for cls_number in range(len(data)):
    for i in range(len(data[cls_number])):
        audio_data = data[cls_number][i][1]
        T = 1/data[cls_number][i][0]
        N = len(audio_data)
        max_val = 1.0/(2.0*T)
        num_vals = N//2

        yf = fft(audio_data)

        xf = np.linspace(0.0, max_val, num_vals)

        i_over_10kHz = np.argmax(xf > 10000)
        yf[i_over_10kHz: len(yf) - i_over_10kHz] = 0.0
        data[cls_number][i][1] = ifft(yf).real
```

2.3 Windowing

Chosen approach: Single window

2.3.1 Single window

By reading in the data samples as a whole, a single window is already used. So therefore no further implementation is needed for the single window approach.

2.3.2 Multiple windows (running this cell decreases the testing accuracy of the SVM)

As shown by the window size performance comparison in the "Data visualization" segtion, the single window approach gives the best performance for the SVM. Random Forest has a testing accuracy of 100% with all tested window sizes. But because of the better accuracy for the SVM, the single window approach is chosen and this cell should not be executed.

[ ]:
```
window_size = 10  # in seconds
overlap = 0.5  # in per cent

data_windowed = [[] for _ in range(len(data))]

for cls_number in range(len(data)):
    for i in range(len(data[cls_number])):
        sample_rate = data[cls_number][i][0]
```

```
        indexes_per_window = window_size * sample_rate
        end_index = indexes_per_window
        while end_index < len(data[cls_number][i][1]):
            data_windowed[cls_number].append([sample_rate,␣
 ↪data[cls_number][i][1][end_index - indexes_per_window:end_index]])
            end_index += int((1 - overlap) * indexes_per_window)

data = data_windowed
```

2.4 Creating spectrograms of the data

[5]:
```
FFT_SIZE=1024
data_spectrograms = [[] for _ in range(len(data))]
for cls_number in range(len(data)):
    for i in range(len(data[cls_number])):
        f,t,pxx = signal.spectrogram(data[cls_number][i][1], nperseg=FFT_SIZE,␣
 ↪fs=data[cls_number][i][0], noverlap=FFT_SIZE/2)
        data_spectrograms[cls_number].append([f, t, pxx])
```

3 Feature engineering

3.1 Binning

[6]:
```
num_freq_bins=5
num_time_bins=5

data_spectrograms_binned = [[] for _ in range(len(data))]

for cls_number in range(len(data_spectrograms)):
    for i in range(len(data_spectrograms[cls_number])):
        resized_pxx = cv2.
 ↪resize(data_spectrograms[cls_number][i][2],(num_time_bins,num_freq_bins))
        data_spectrograms_binned[cls_number].
 ↪append([data_spectrograms[cls_number][i][0],␣
 ↪data_spectrograms[cls_number][i][1], resized_pxx])
```

3.2 Feature extraction

Chosen features:

- All bins from the binned spectrogram (25 features)
- Three frequencies with the highest magnitude (3 features)
- Mean magnitude of the frequencies (1 feature)
- Median magnitude of the frequencies (1 feature)
- Variance of the magnitudes of the frequencies (1 feature)

[7]:
```
def extract_frequency_features(cls_number: int, i: int) -> list:
    audio_data = data[cls_number][i][1]
    T = 1/data[cls_number][i][0]
    N = len(audio_data)
    max_val = 1.0/(2.0*T)
    num_vals = N//2

    yf_all = fft(audio_data)
    yf = 2.0/N * np.abs(yf_all[0:num_vals])

    xf = np.linspace(0.0, max_val, num_vals)

    sorted_frequencies = np.argsort(yf)

    return [xf[sorted_frequencies[-1]], xf[sorted_frequencies[-2]],␣
 ↪xf[sorted_frequencies[-3]], np.mean(yf), np.median(yf), np.var(yf)]


data_features = [[] for _ in range(len(data))]

for cls_number in range(len(data_spectrograms_binned)):
    for i in range(len(data_spectrograms_binned[cls_number])):
        features = data_spectrograms_binned[cls_number][i][2].reshape((-1,)).
 ↪tolist()
        features.extend(extract_frequency_features(cls_number, i))
        data_features[cls_number].append(np.asarray(features))
```

3.3 Creating labels

[8]:
```
data_list = []
labels_list = []

for cls_number in range(len(data_features)):
    for i in range(len(data_features[cls_number])):
        data_list.append(data_features[cls_number][i])
        labels_list.append(cls_number)
```

3.4 Normalization

Neccesarry for the SVM to achieve 100% testing accuracy, for the Random Forest it does not matters if the data is normalized or not.

[9]:
```
scaler = StandardScaler()
data_list = scaler.fit_transform(data_list)
```

3.5 Create training and test data

[10]:
```
xtrain, xtest, ytrain, ytest = train_test_split(data_list, labels_list,␣
↪test_size=0.30, random_state=42)
```

4 ML models

Two machine learning algorithms have been tested for this assignment. Firstly a SVM with a linear kernel, which has a testing accuracy of 100%. Also Random Forest has been implemented and achieves a testing accuracy of 100% as well. In order for the SVM to achieve 100%, the data has to be normalized. For the Random Forest model it does not matter and it achieves 100% with and without normalization.

4.1 Random forest

[11]:
```
clf = RandomForestClassifier()
clf.fit(xtrain, ytrain)
cv_scores = cross_val_score(clf, xtrain, ytrain, cv=10)
print('Average Cross Validation Score from Training:', cv_scores.mean(),␣
↪sep='\n', end='\n\n\n')

ypred = clf.predict(xtest)
cm = confusion_matrix(ytest, ypred)
cr = classification_report(ytest, ypred)

print('Confusion Matrix:', cm, sep='\n', end='\n\n\n')
print('Test Statistics:', cr, sep='\n', end='\n\n\n')

print('Testing Accuracy:', accuracy_score(ytest, ypred))
```

```
Average Cross Validation Score from Training:
1.0


Confusion Matrix:
[[7 0 0 0 0 0]
 [0 5 0 0 0 0]
 [0 0 7 0 0 0]
 [0 0 0 7 0 0]
 [0 0 0 0 5 0]
 [0 0 0 0 0 5]]


Test Statistics:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00         5
```

```
           2       1.00      1.00      1.00         7
           3       1.00      1.00      1.00         7
           4       1.00      1.00      1.00         5
           5       1.00      1.00      1.00         5

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

```
Testing Accuracy: 1.0
```

4.2 SVM

[12]:
```
clf = SVC(kernel="linear")
clf.fit(xtrain, ytrain)
cv_scores = cross_val_score(clf, xtrain, ytrain, cv=10)
print('Average Cross Validation Score from Training:', cv_scores.mean(),␣
 ↪sep='\n', end='\n\n\n')

ypred = clf.predict(xtest)
cm = confusion_matrix(ytest, ypred)
cr = classification_report(ytest, ypred)

print('Confusion Matrix:', cm, sep='\n', end='\n\n\n')
print('Test Statistics:', cr, sep='\n', end='\n\n\n')

print('Testing Accuracy:', accuracy_score(ytest, ypred))
```

```
Average Cross Validation Score from Training:
0.9416666666666667


Confusion Matrix:
[[7 0 0 0 0 0]
 [0 5 0 0 0 0]
 [0 0 7 0 0 0]
 [0 0 0 7 0 0]
 [0 0 0 0 5 0]
 [0 0 0 0 0 5]]


Test Statistics:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00         5
```

```
           2       1.00      1.00      1.00         7
           3       1.00      1.00      1.00         7
           4       1.00      1.00      1.00         5
           5       1.00      1.00      1.00         5

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36


Testing Accuracy: 1.0
```

5 Live demo

Run all cells above except:

- Multiple windows
- Normalization
- SVM

And then run the following cell

[ ]:
```
import pyaudio

def fft_preparations(data_points: np.ndarray, sample_rate: int = 44100) -> np.
 ↪ndarray:
    T = 1/sample_rate
    N = len(data_points)
    max_val = 1.0/(2.0*T)
    num_vals = N//2

    yf = fft(data_points)

    xf = np.linspace(0.0, max_val, num_vals)

    i_over_10kHz = np.argmax(xf > 10000)
    yf[i_over_10kHz: len(yf) - i_over_10kHz] = 0.0

    yf_normalized = 2.0/N * np.abs(yf[0:num_vals])
    sorted_frequencies = np.argsort(yf_normalized)

    return ifft(yf).real, [xf[sorted_frequencies[-1]],␣
 ↪xf[sorted_frequencies[-2]], xf[sorted_frequencies[-3]], np.
 ↪mean(yf_normalized), np.median(yf_normalized), np.var(yf_normalized)]

def spectrograms_binned(data_points: np.ndarray, sample_rate: int = 44100) ->␣
 ↪np.ndarray:
```

```
    _,_,pxx = signal.spectrogram(data_points, nperseg=1024, fs=sample_rate,␣
 ↪noverlap=512)
    return cv2.resize(pxx,(5,5))

def generate_features(data_points: np.ndarray, sample_rate: int = 44100) -> np.
 ↪ndarray:
    data_points_new, features_fft = fft_preparations(data_points, sample_rate)
    features = spectrograms_binned(data_points_new, sample_rate).reshape((-1,)).
 ↪tolist()
    features.extend(features_fft)
    return np.asarray([features])


CHUNK=8000
FORMAT=pyaudio.paInt16
CHANNELS=1
RATE=44100

p=pyaudio.PyAudio()
stream=p.
 ↪open(format=FORMAT,channels=CHANNELS,rate=RATE,input=True,frames_per_buffer=CHUNK)
print("start classifying")

try:
    while True:
        data_chunk=stream.read(CHUNK)
        data_points = np.frombuffer(data_chunk, dtype=np.int16)
        features = generate_features(data_points, RATE)
        predicted_class = clf.predict(features)
        print(classes[predicted_class[0]])
except KeyboardInterrupt:
    pass

print("classifying stopped")
stream.stop_stream()
stream.close()
p.terminate()
```

6 Data visualization

Tipps for adressing the audio data in the data list:

- First index defines class of data (see classes list at the top)
- Second index is the number of the sample
- Third index decides between sample rate and actual data (0 = sample rate; 1 = audio data)

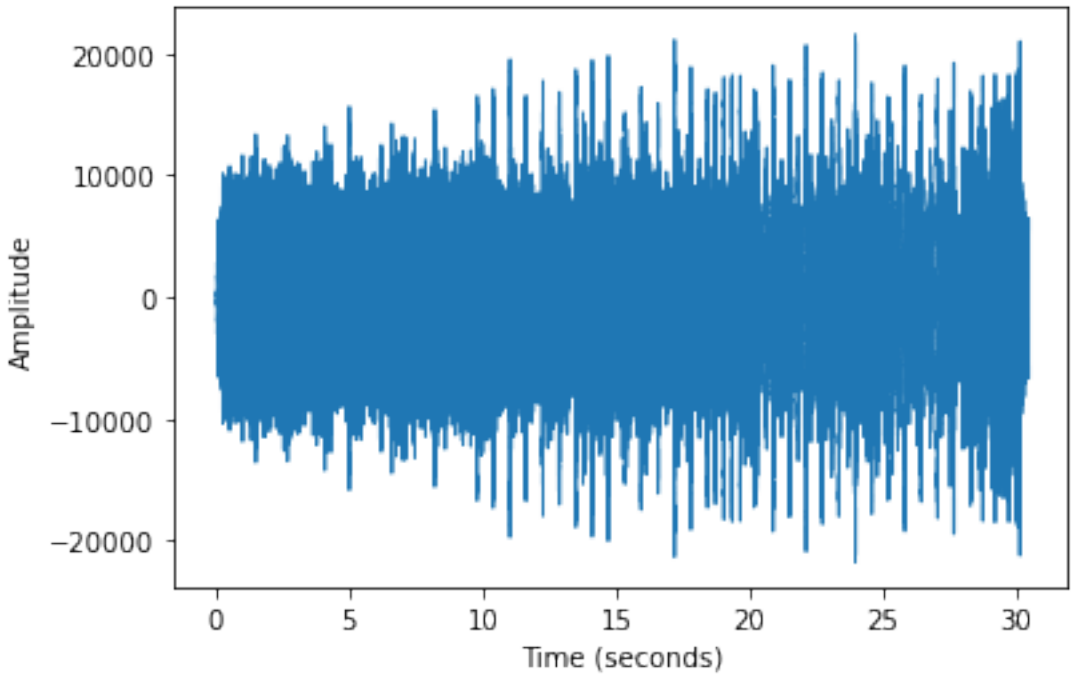
6.1 Play audio

[13]:
```
from IPython.display import Audio

Audio(data=data[3][9][1], rate=data[3][9][0])
```

[13]:
```
<IPython.lib.display.Audio object>
```

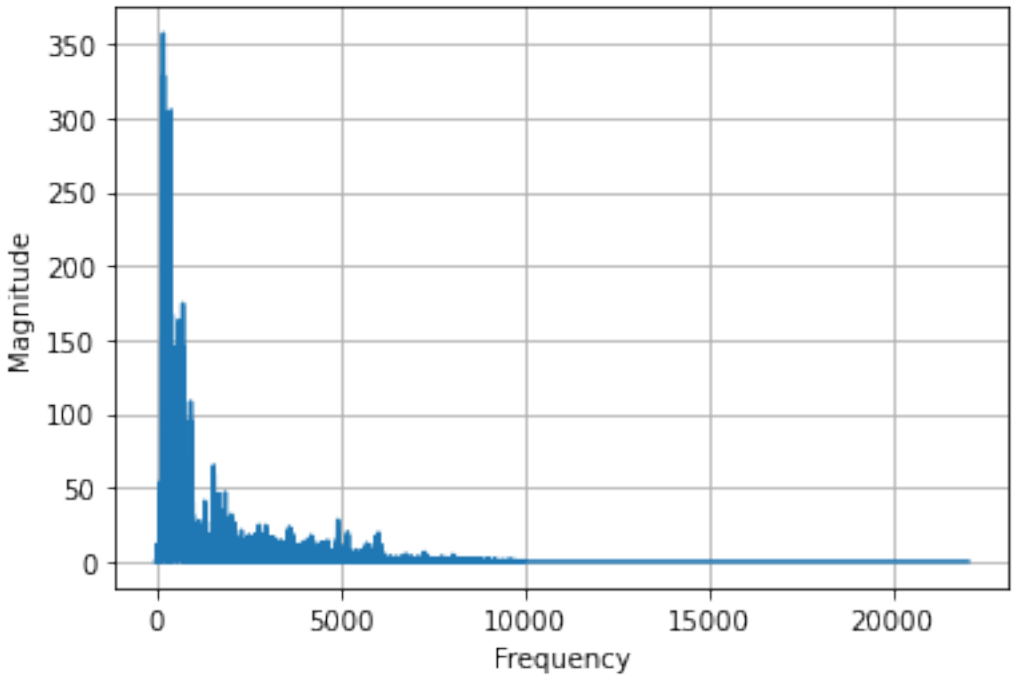
6.2 Plot audio in time domain

[14]:
```
import librosa
import matplotlib.pyplot as plt
import librosa.display

plt.figure()
librosa.display.waveshow(np.asarray(data[3][9][1], dtype=float),␣
 ↪sr=data[3][9][0])
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.show()
```



6.3 Plot audio in frequency domain

[15]:
```
from scipy.fftpack import fft
import matplotlib.pyplot as plt

def fft_method(audio, sampling_rate):
    T = 1/sampling_rate
    N = len(audio)
    max_val = 1.0/(2.0*T)
    num_vals = N//2

    yf_all = fft(audio)

    xf = np.linspace(0.0, max_val, num_vals)
    yf = 2.0/N * np.abs(yf_all[0:num_vals])

    return xf, yf


xf, yf = fft_method(data[3][9][1], data[3][9][0])

plt.plot(xf, yf)
plt.grid()
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.show()
```

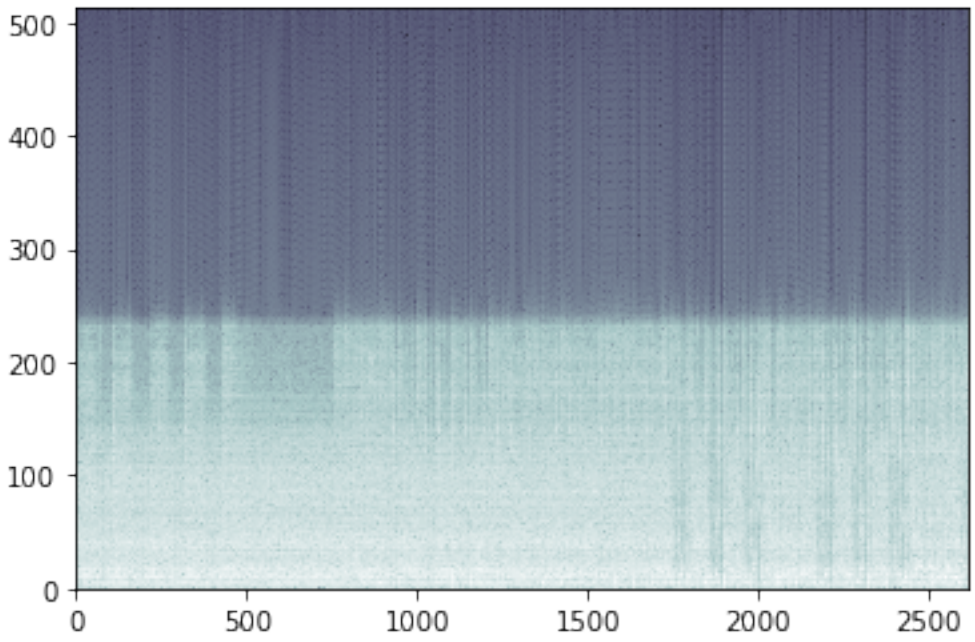


6.4 Plot spectrograms

[16]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
cmap=plt.cm.bone
cmap.set_under(color='k', alpha=None)
plt.pcolormesh(np.log10(data_spectrograms[3][9][2]),cmap=cmap)
```

```
C:\Users\Timmy\AppData\Local\Temp\ipykernel_14960\3681133312.py:6:
MatplotlibDeprecationWarning: You are modifying the state of a globally
registered colormap. This has been deprecated since 3.3 and in 3.6, you will not
be able to modify a registered colormap in-place. To remove this warning, you
can make a copy of the colormap first. cmap = mpl.cm.get_cmap("bone").copy()
  cmap.set_under(color='k', alpha=None)
```

[16]:
```
<matplotlib.collections.QuadMesh at 0x162874b6eb0>
```

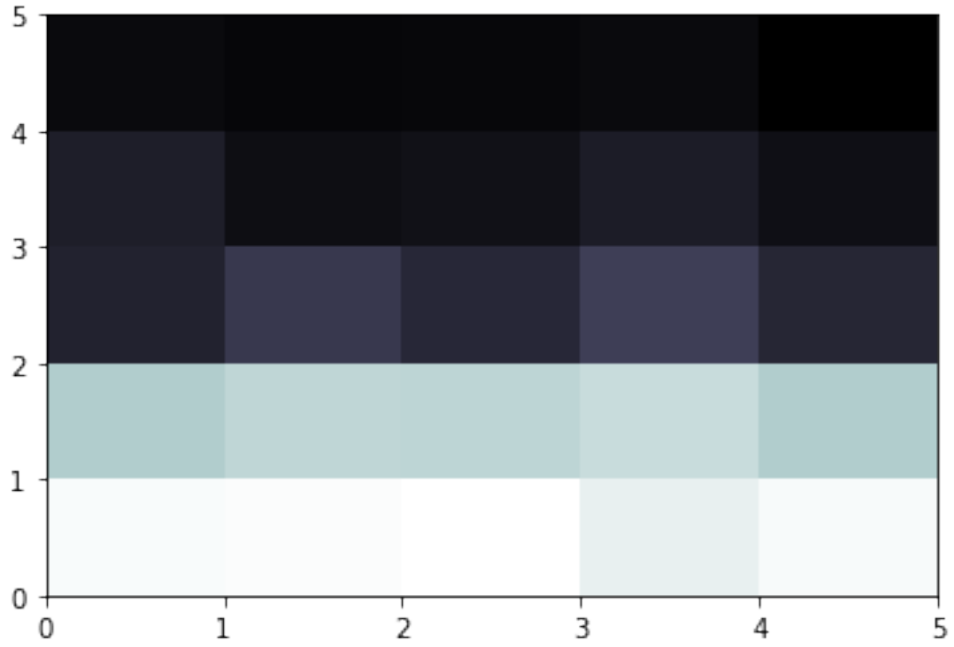


6.5 Plot binned spectrograms

[17]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
cmap=plt.cm.bone
cmap.set_under(color='k', alpha=None)
plt.pcolormesh(np.log10(data_spectrograms_binned[3][9][2]),cmap=cmap)
```

```
C:\Users\Timmy\AppData\Local\Temp\ipykernel_14960\3151999976.py:6:
MatplotlibDeprecationWarning: You are modifying the state of a globally
registered colormap. This has been deprecated since 3.3 and in 3.6, you will not
be able to modify a registered colormap in-place. To remove this warning, you
can make a copy of the colormap first. cmap = mpl.cm.get_cmap("bone").copy()
  cmap.set_under(color='k', alpha=None)
```

[17]:
```
<matplotlib.collections.QuadMesh at 0x162a1803790>
```



6.6 Plot binned spectrograms of all classes

[18]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
cmap=plt.cm.bone
```

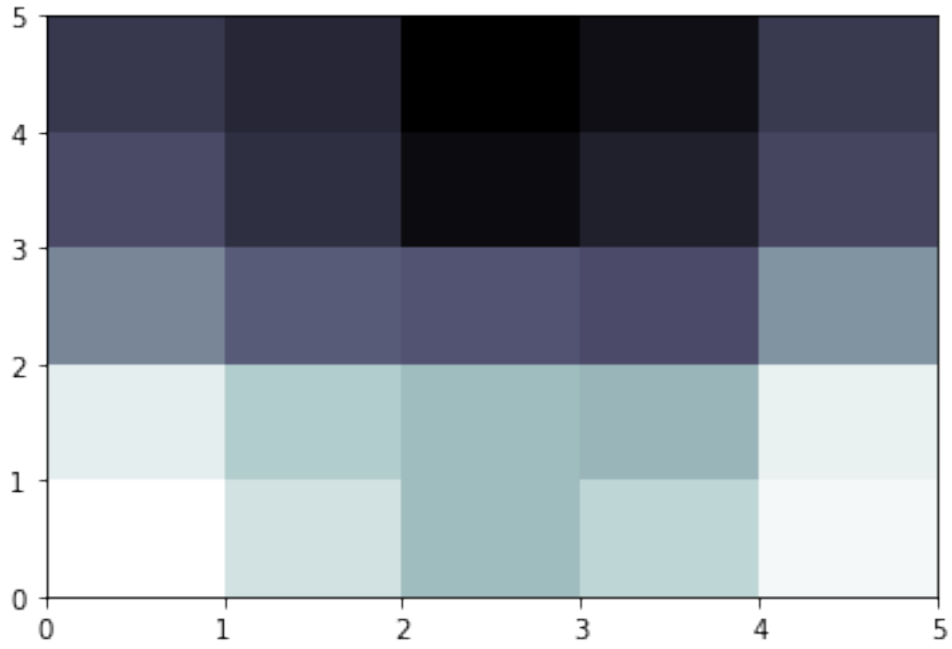
```
cmap.set_under(color='k', alpha=None)
```

```
C:\Users\Timmy\AppData\Local\Temp\ipykernel_14960\3670946234.py:6:
MatplotlibDeprecationWarning: You are modifying the state of a globally
registered colormap. This has been deprecated since 3.3 and in 3.6, you will not
be able to modify a registered colormap in-place. To remove this warning, you
can make a copy of the colormap first. cmap = mpl.cm.get_cmap("bone").copy()
  cmap.set_under(color='k', alpha=None)
```

6.6.1 Alarm

[19]:
```
plt.pcolormesh(np.log10(data_spectrograms_binned[0][0][2]),cmap=cmap)
```

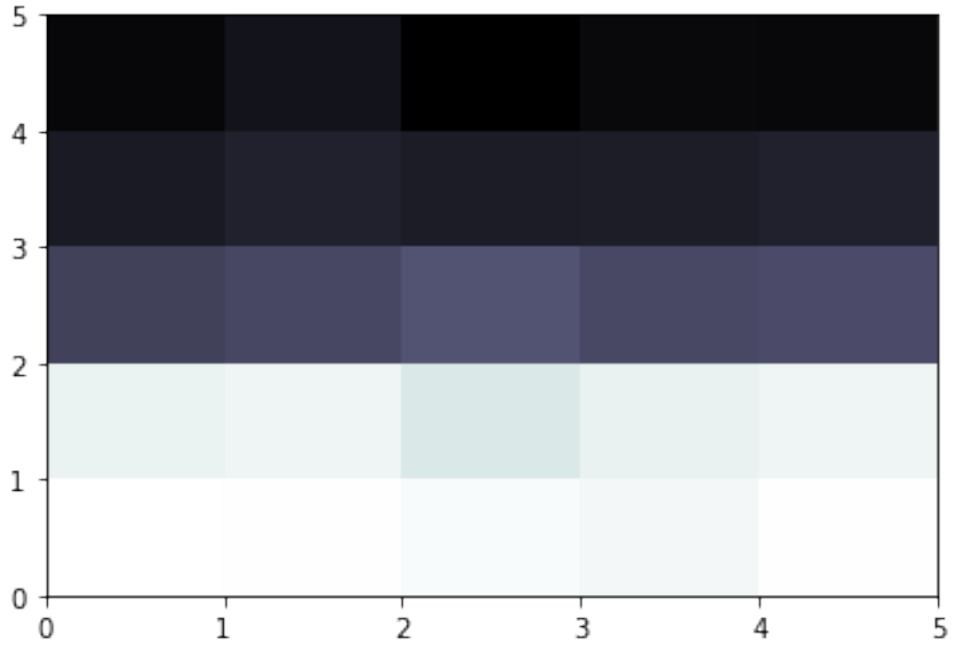
[19]:
```
<matplotlib.collections.QuadMesh at 0x162874f4460>
```



6.6.2 Blender

[22]:
```
plt.pcolormesh(np.log10(data_spectrograms_binned[1][0][2]),cmap=cmap)
```

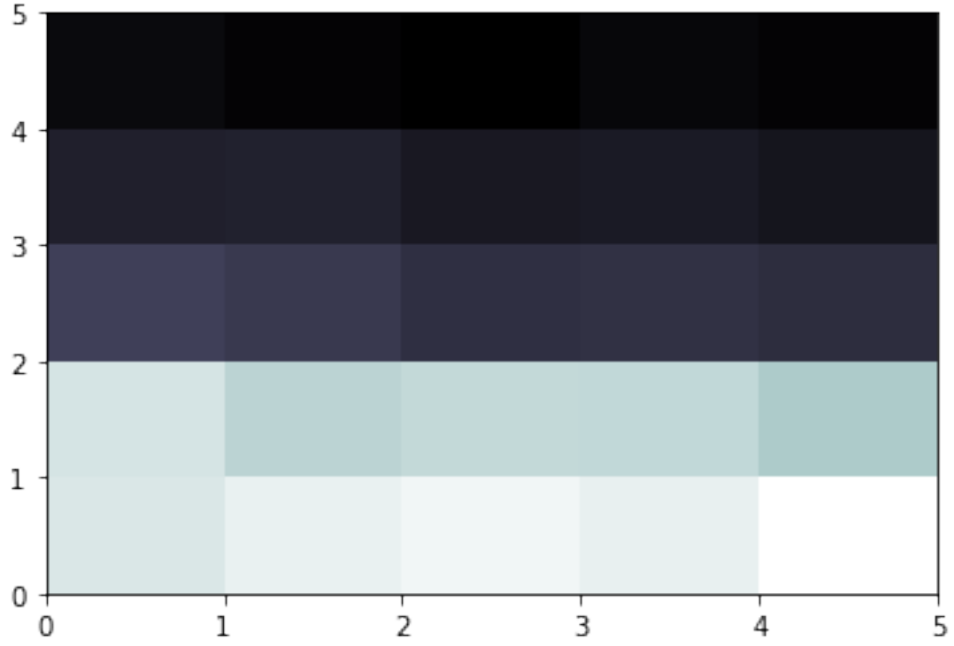
[22]:
```
<matplotlib.collections.QuadMesh at 0x16287612670>
```



6.6.3 Microwave

[21]:
```
plt.pcolormesh(np.log10(data_spectrograms_binned[2][0][2]),cmap=cmap)
```

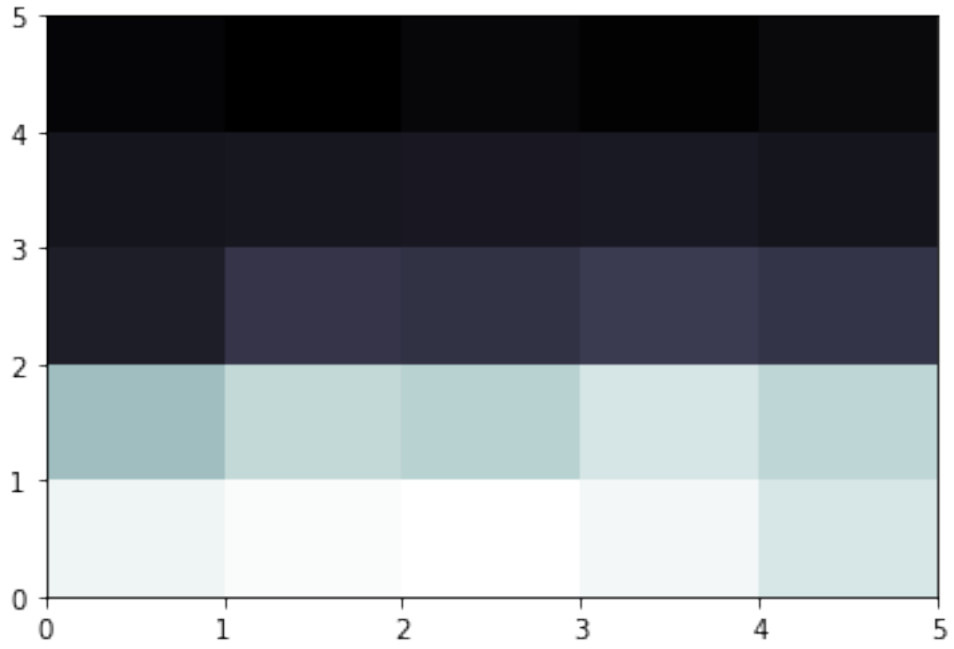
[21]:
```
<matplotlib.collections.QuadMesh at 0x162875b0a00>
```



6.6.4 Music

[23]:
```
plt.pcolormesh(np.log10(data_spectrograms_binned[3][0][2]),cmap=cmap)
```

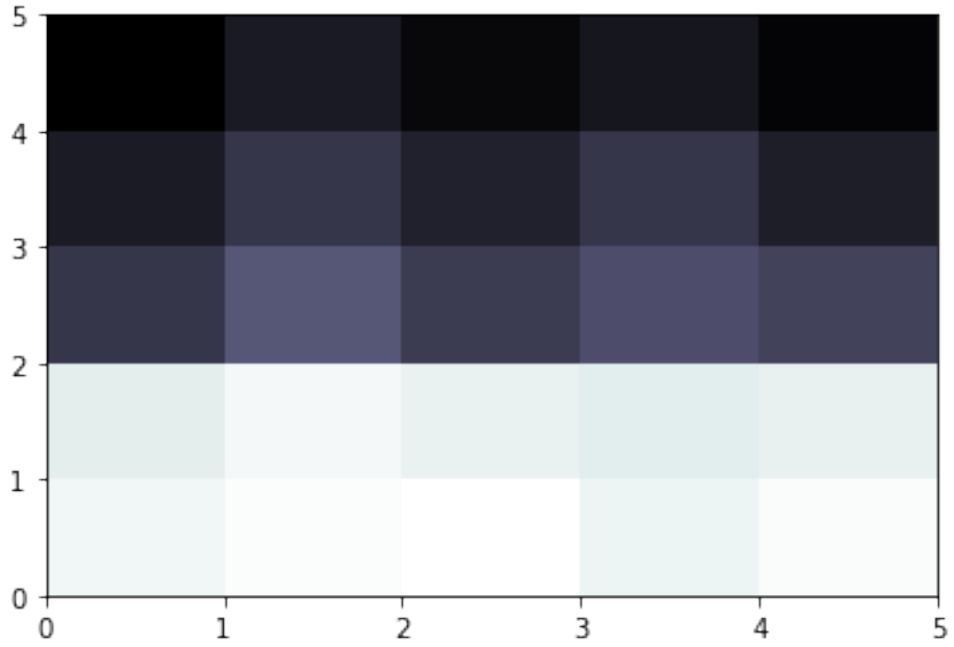
[23]:
```
<matplotlib.collections.QuadMesh at 0x1628766f310>
```



6.6.5 Silence

[24]:
```
plt.pcolormesh(np.log10(data_spectrograms_binned[4][0][2]),cmap=cmap)
```

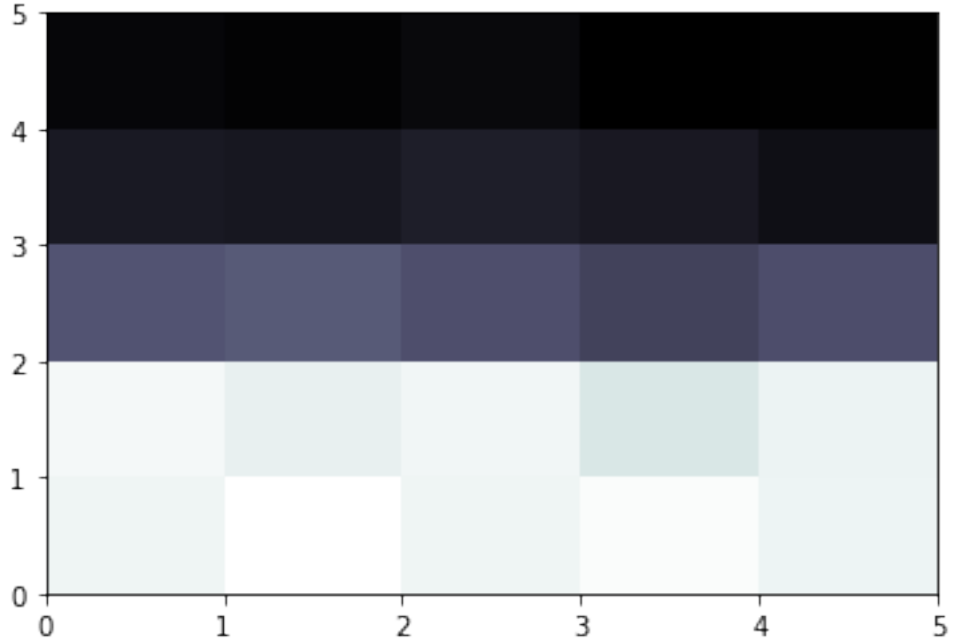
[24]:
```
<matplotlib.collections.QuadMesh at 0x162876c4d30>
```



6.6.6 Vacuum

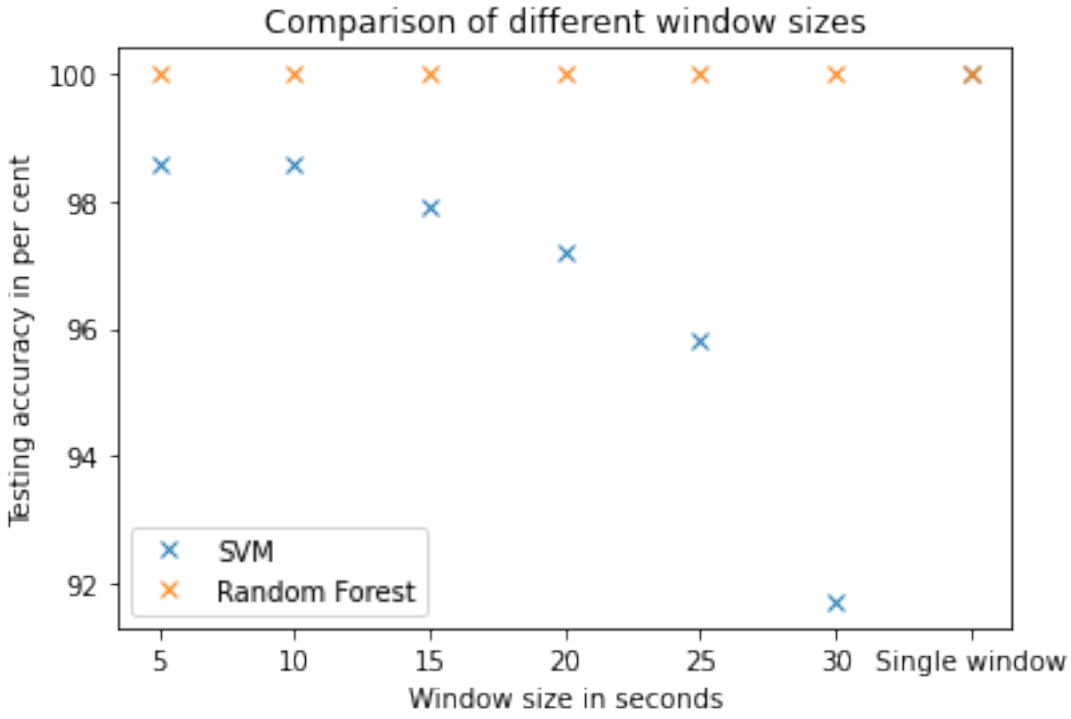
[25]:
```
plt.pcolormesh(np.log10(data_spectrograms_binned[5][0][2]),cmap=cmap)
```

[25]:
```
<matplotlib.collections.QuadMesh at 0x1628771f9a0>
```



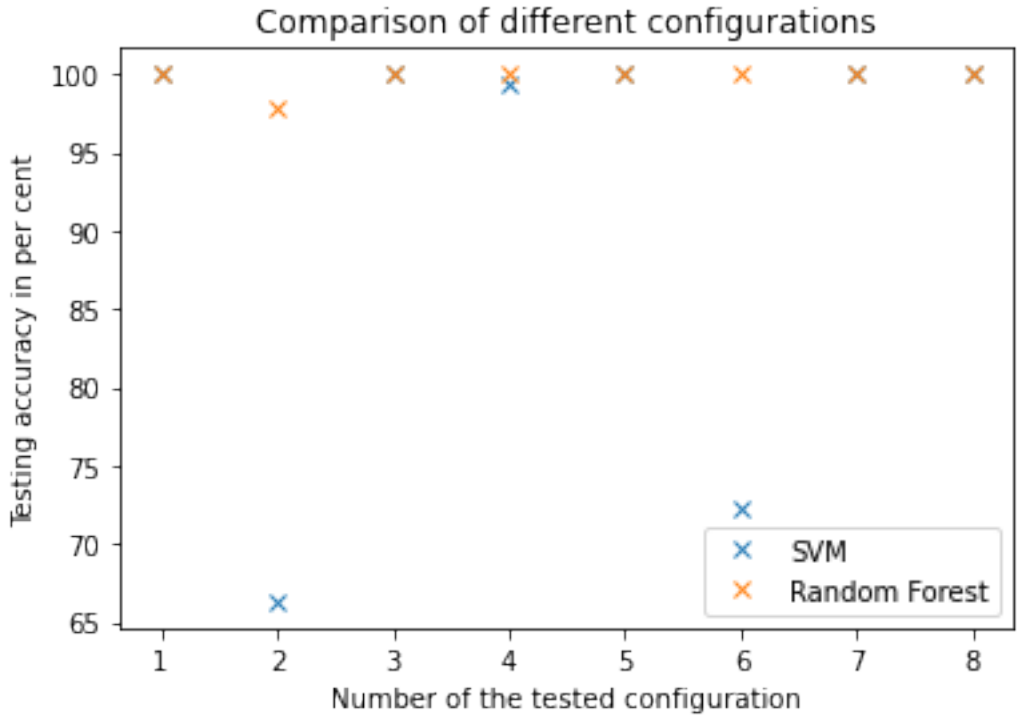
6.7 Window size comparison

[26]:
```
# Overlap of 50%

import matplotlib.pyplot as plt

window_sizes = [5, 10, 15, 20, 25, 30, 'Single window']
results_svm = [98.6, 98.6, 97.9, 97.2, 95.8, 91.7, 100.0]
results_rf = [100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0]

plt.plot(window_sizes, results_svm, "x", label="SVM")
plt.plot(window_sizes, results_rf, "x", label="Random Forest")
plt.xlabel("Window size in seconds")
plt.ylabel("Testing accuracy in per cent")
plt.title("Comparison of different window sizes")
plt.legend()
plt.show()
```



6.8 Comparison feature combinations and window sizes

[27]:
```
import matplotlib.pyplot as plt

x_labels = ['Multiple windows with spectrogram', 'Multiple windows with binned␣
 ↪spectrogram',
           'Multiple windows with spectrogram and other features',
            'Multiple windows with binned spectrogram and other features',
           'Single window with spectrogram', 'Single window with binned␣
 ↪spectrogram',
           'Single window with spectrogram and other features',
            'Single window with binned spectrogram and other features']
x_values = np.arange(8) + 1

svm_results = [100.0, 66.3, 100.0, 99.4, 100.0, 72.2, 100.0, 100.0]
rf_results = [100.0, 97.8, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0]

plt.plot(x_values, svm_results, "x", label="SVM")
plt.plot(x_values, rf_results, "x", label="Random Forest")
plt.xlabel("Number of the tested configuration")
plt.ylabel("Testing accuracy in per cent")
plt.title("Comparison of different configurations")
plt.legend()
plt.show()
```



Explanation of the meanings of the configuration number:

- 1: Multiple windows with only the spectrogram data as features
- 2: Multiple windows with only the binned spectrogram as features
- 3: Multiple windows with the spectrogram data and the other features
- 4: Multiple windows with the binned spectrogram data and the other features
- 5: Single window with only the spectrogram data as features
- 6: Single window with only the binned spectrogram as features
- 7: Single window with the spectrogram data and the other features
- 8: Single window with the binned spectrogram data and the other features

For all multiple window configurations a window size of 10 seconds and 50% overlap have been chosen