DSP Final Project : Music Type Classification

105061211 陳祈瑋
105060012 張育菘

Wavelet Denoising Visualization

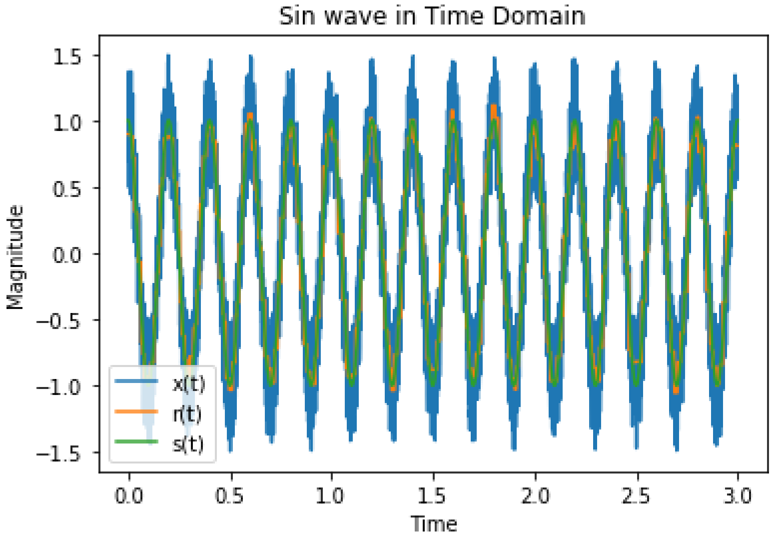
Create a perfect sine wave s(t), and then add some noise to produce x(t). In order to denosie, we use "denoise_wavelet" in skimage.restoration to reconstruct the original signal, obtaining r(t). These three signals are shown in time domain and frequency domain below.

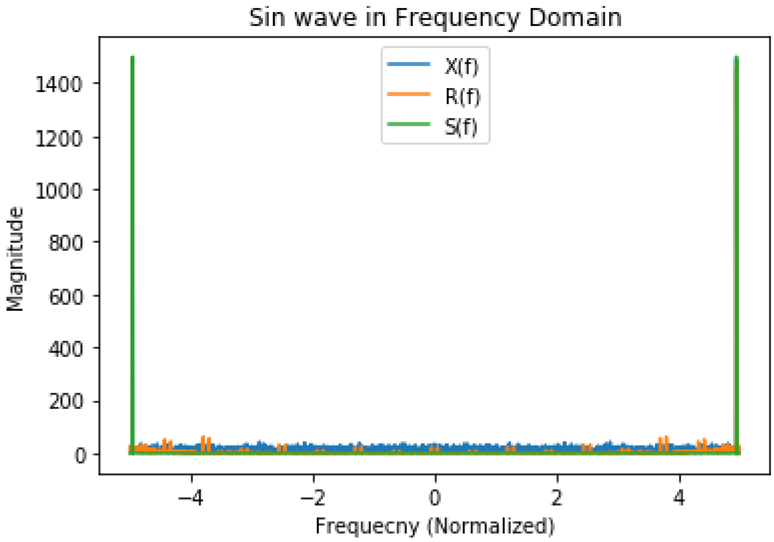
In [2]:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from skimage.restoration import (denoise_wavelet, estimate_sigma)
from scipy.fftpack import fft,ifft

# Denoising
res = 1000  # Resolution
fs = 5    # Frequency
t = 3      # Duration
n = np.arange(t*res)
s = np.cos(2*np.pi*fs*(n/res))

w = np.random.rand(t*res)-0.5
x = s + w
r = denoise_wavelet(x, wavelet='db1', mode='soft', method='BayesShrink', rescale_sigma=
'True')

# Time domain
fig, ax = plt.subplots()
plt.plot(n/(res), x)
plt.plot(n/(res), r)
plt.plot(n/(res), s)
ax.set_xlabel('Time')
ax.set_ylabel('Magnitude')
ax.set_title('Sin wave in Time Domain')
ax.legend(['x(t)','r(t)','s(t)'],loc="best")

# Frequency domain
S = fft(s)
X = fft(x)
R = fft(r)
fig, ax = plt.subplots()

plt.plot((2*(n/(res*t))-1)*fs, abs(X))
plt.plot((2*(n/(res*t))-1)*fs, abs(R))
plt.plot((2*(n/(res*t))-1)*fs, abs(S))
ax.set_xlabel('Frequecny (Normalized)')
ax.set_ylabel('Magnitude')
ax.set_title('Sin wave in Frequency Domain')
ax.legend(['X(f)','R(f)','S(f)'],loc="best")
```

Out[2]:

```
<matplotlib.legend.Legend at 0x20f96152908>
```





Parameter Setup

In [3]:

```
# Parameter Setup
N_type = 5
N_slice = 3
N_data = 100
N_clip = N_type*N_slice*N_data

duration = 30

oct_diff = np.array([0, 200, 400, 800, 1600, 3200, 6400, 11025])  # Freq Normalized int
erval
oct_alpha = 0.02;  # tunable

N_freq_intvl = (len(oct_diff)-1);
feature = np.zeros((N_clip, N_freq_intvl*2))
style = np.zeros((N_clip, 1));
```

Visualize one music clip

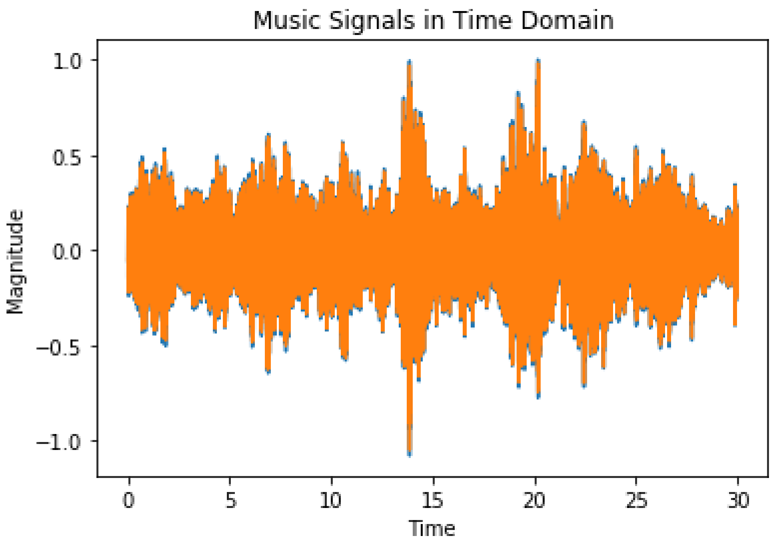
We plot a selected music clip which is classical type to understand the effect before and after applying "denoise_wavelet" in time domain.

In [4]:

```
from scipy.io import wavfile
from skimage.restoration import (denoise_wavelet, estimate_sigma)

# Visualization of one music clip
path = "Data/genres_original/classical/classical.00000.wav"
fs, y = wavfile.read(path)
N_y = y.shape[0]

y = y/max(y)

y_de = denoise_wavelet(y, wavelet='db1', mode='soft', method='BayesShrink', rescale_sig
ma='True')

# Time domain
fig, ax = plt.subplots()
plt.plot(np.arange(N_y)/N_y*duration, y)
ax.set_xlabel('Time')
ax.set_ylabel('Magnitude')
ax.set_title('Music Signals in Time Domain')
# fig, ax = plt.subplots()
plt.plot(np.arange(N_y)/N_y*duration, y_de)
ax.set_xlabel('Time')
ax.set_ylabel('Magnitude')
ax.set_title('Music Signals in Time Domain')
```

Out[4]:

```
Text(0.5, 1.0, 'Music Signals in Time Domain')
```



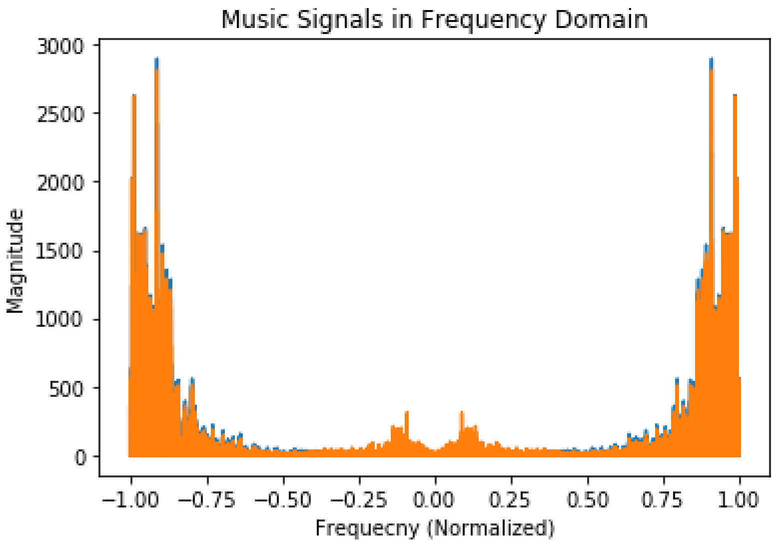
We plot a selected music clip which is classical type to understand the effect before and after applying "denoise_wavelet" in frequency domain.

In [5]:

```
# Frequency domain
Y = fft(y)
Y_de = fft(y_de)
fig, ax = plt.subplots()
plt.plot(2*(np.arange(N_y)/N_y)-1, abs(Y))
ax.set_xlabel('Frequecny (Normalized)')
ax.set_ylabel('Magnitude')
ax.set_title('Music Signals in Frequency Domain')
# fig, ax = plt.subplots()
plt.plot(2*(np.arange(N_y)/N_y)-1, abs(Y_de))
ax.set_xlabel('Frequecny (Normalized)')
ax.set_ylabel('Magnitude')
ax.set_title('Music Signals in Frequency Domain')
```

Out[5]:

```
Text(0.5, 1.0, 'Music Signals in Frequency Domain')
```



Octave-Scale filter

"oct_feature" divides the frequency domain into seven intervals, and then extract the peak, valley and their difference in each interval as the music input features.

In [6]:

```
def oct_feature(start, stop, Y, alpha):
    N = Y.shape[0]
    Y_filter = Y[int(np.floor(N*start)):int(np.floor(N*stop))]
    N_filter = Y_filter.shape[0]
    Y_sort = np.sort(abs(Y_filter))
    N_avg = int(np.floor(N_filter*alpha))
    peak = np.sum(Y_sort[N_filter-N_avg:N_filter])/N_avg
    valley = np.sum(Y_sort[0:N_avg])/N_avg
    sc = peak - valley
    return peak, valley, sc
```

Data preprocessing

Use for loop to preprocessing the data, such as slicing, FFT, wavelet denoising and feature extraction etc. As a result, we obtain parameter 'feature' as input features, and parameter 'style' as labels.

In [7]:

```
data_type = ["classical", "country", "blues", "hiphop", "pop"]

# Sliced to 5 clip
# data_diff = np.array([0, 5, 10, 15, 20, 25, 30])/30    # Time Normalized interval
# Y_avg = np.zeros((N_type, int(N_y/3)))   # Time Normalized interval

# Sliced to 3 clip
data_diff = np.array([0, 10, 20, 30])/30    # Time Normalized interval
Y_avg = np.zeros((N_type, int(N_y/3)))   # Time Normalized interval

# Don't be sliced
# data_diff = np.array([0, 30])/30
# Y_avg = np.zeros((N_type, N_y))

i_num = 0
for i in data_type:
    print("Now processing " + i)
    for j in range(N_data):
        if j < 10:
            num = "0"+str(j)
        else:
            num = str(j)
        path = "Data/genres_original/"+i+"/"+i+".000"+num+".wav"
        [fs, y] = wavfile.read(path)
        if y.shape[0] < N_y :
            z = np.zeros((N_y - y.shape[0]))
            y = np.concatenate((y, z), axis=0)
        for k in range(N_slice):
            y_slice = y[int(np.floor(N_y*data_diff[k])):int(np.floor(N_y*data_diff[k+1
]))]
            y_slice = y_slice/max(y_slice)
            y_de = denoise_wavelet(y_slice, wavelet='db1', mode='soft', method='BayesSh
rink', rescale_sigma='True')
            #Y = fft(y_slice)
            Y = fft(y_de)
            N_y_slice = Y.shape[0]
            Y_avg[i_num,:] = Y_avg[i_num,:] + abs(Y[:])
            # Octave-Scale Filter
            oct_diff_norm = oct_diff/(fs/2);  # Normalized interval
            sc = np.zeros((1,N_freq_intvl))
            valley = np.zeros((1,N_freq_intvl))
            peak = np.zeros((1,N_freq_intvl))
            for m in range(oct_diff.shape[0]-1):
                [peak[0,m], valley[0,m], sc[0,m]] = oct_feature(oct_diff_norm[m], oct_d
iff_norm[m+1], Y[int(np.floor(N_y_slice/2)):N_y_slice], oct_alpha)
            feature[i_num*N_data*N_slice+j*N_slice+k,0:N_freq_intvl] = sc
            feature[i_num*N_data*N_slice+j*N_slice+k,N_freq_intvl:2*N_freq_intvl] = val
ley
            style[i_num*N_data*N_slice+j*N_slice+k,0] = i_num

    Y_avg[i_num,:] = Y_avg[i_num,:]/(N_data*N_slice)
    i_num = i_num + 1
```

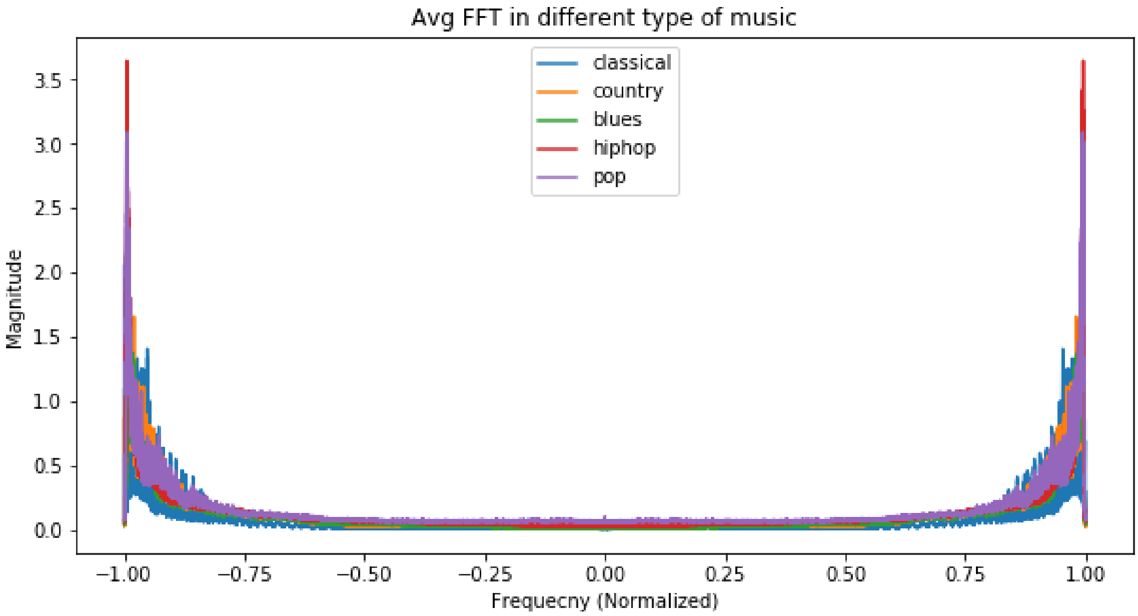
```
Now processing classical
Now processing country
Now processing blues
Now processing hiphop
Now processing pop
```

Average Spectrums in different type of music

There are 500 clip of music in each types. To analysis the spectrums, we average the FFT in each type. As a result, we can see that different type of music correspond to different spectrum configurations.

In [8]:

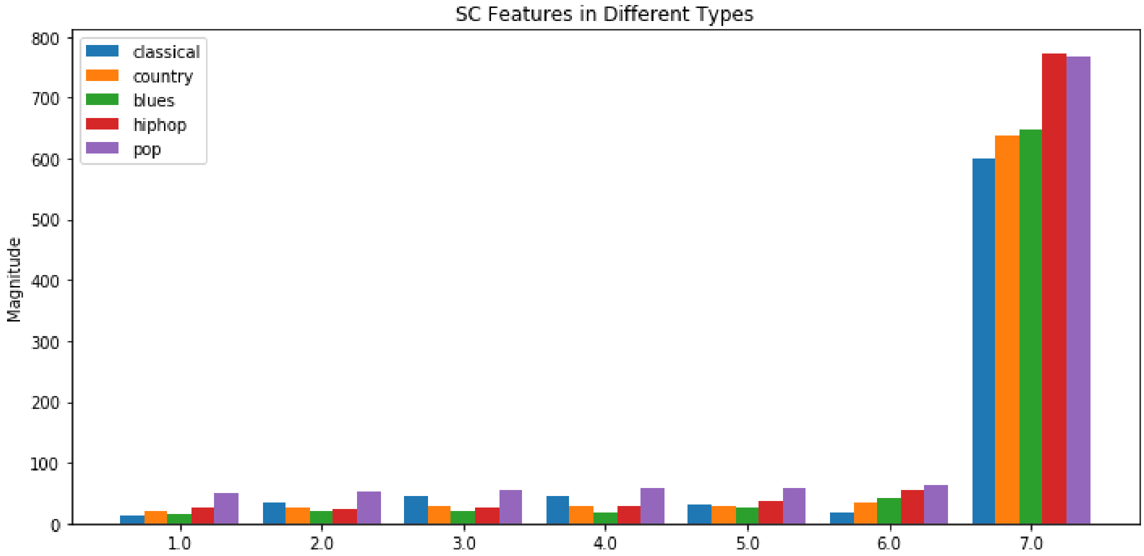
```
fig, ax = plt.subplots(figsize=(10,5))
for i in range(N_type):
    plt.plot(2*(np.arange(Y_avg[i,:].shape[0])/Y_avg[i,:].shape[0])-1, Y_avg[i,:]/(N_da
ta*N_slice))
    ax.set_xlabel('Frequecny (Normalized)')
    ax.set_ylabel('Magnitude')
    ax.set_title('Avg FFT in different type of music')
    ax.legend(data_type,loc="best")
```

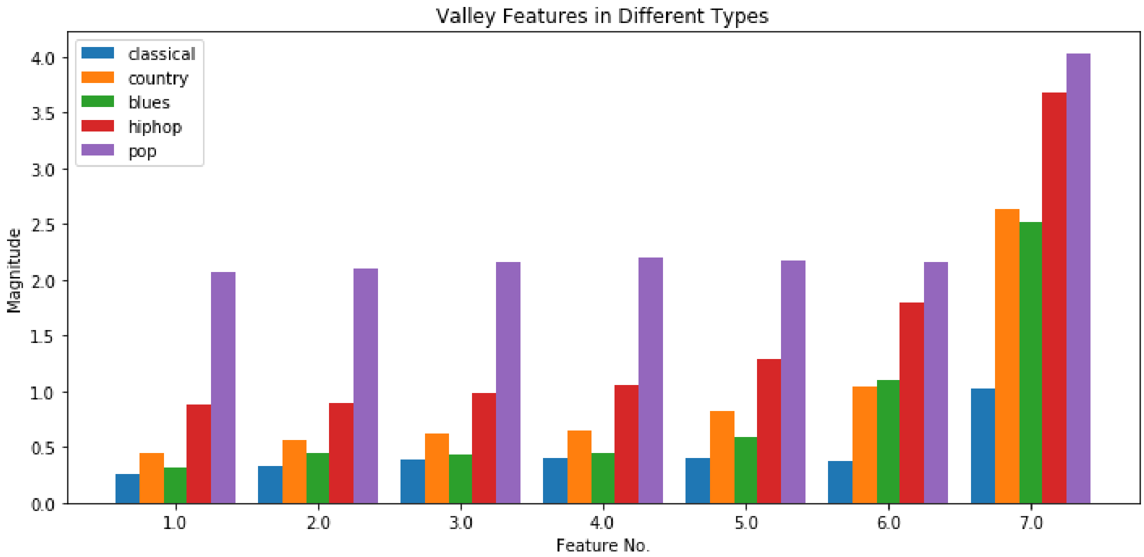


Features Visulization

In [10]:

```
feature = np.array(feature)

size = (feature.shape[1])/2
labels = np.arange(size)+1;
type1 = feature[0:N_slice*N_data,:]
type2 = feature[N_slice*N_data:2*N_slice*N_data,:]
type3 = feature[2*N_slice*N_data:3*N_slice*N_data,:]
type4 = feature[3*N_slice*N_data:4*N_slice*N_data,:]
type5 = feature[4*N_slice*N_data:5*N_slice*N_data,:]

x = np.arange(len(labels))*3  # the label locations
width = 0.5  # the width of the bars

fig, ax = plt.subplots(figsize=(10,5))
rects1 = ax.bar(x - 2*width, np.mean(type1[:,0:7], axis=0), width, label= data_type[0])
rects2 = ax.bar(x - width  , np.mean(type2[:,0:7], axis=0), width, label= data_type[1])
rects3 = ax.bar(x          , np.mean(type3[:,0:7], axis=0), width, label= data_type[2])
rects4 = ax.bar(x + width  , np.mean(type4[:,0:7], axis=0), width, label= data_type[3])
rects5 = ax.bar(x + 2*width, np.mean(type5[:,0:7], axis=0), width, label= data_type[4])

# Add some text for labels, title and custom x-axis tick labels, etc.

ax.set_ylabel('Magnitude')
ax.set_title('SC Features in Different Types')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

fig.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(10,5))
rects1 = ax.bar(x - 2*width, np.mean(type1[:,7:14], axis=0), width, label= data_type[0
])
rects2 = ax.bar(x - width  , np.mean(type2[:,7:14], axis=0), width, label= data_type[1
])
rects3 = ax.bar(x          , np.mean(type3[:,7:14], axis=0), width, label= data_type[2
])
rects4 = ax.bar(x + width  , np.mean(type4[:,7:14], axis=0), width, label= data_type[3
])
rects5 = ax.bar(x + 2*width, np.mean(type5[:,7:14], axis=0), width, label= data_type[4
])

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Feature No.')
ax.set_ylabel('Magnitude')
ax.set_title('Valley Features in Different Types')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

fig.tight_layout()
plt.show()
```





SVM Model Training

In [21]:

```
X = feature
y = style

# Training and testing data splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
=1, stratify = y)

# Standardization
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

# SVM model
from sklearn.svm import SVC
svm = SVC(kernel='rbf', C=1000.0, random_state=1)
svm.fit(X_train_std, y_train.ravel())
print('SVM Accuracy: %.2f %%' %(svm.score(X_test_std, y_test.ravel())*100))
```

```
SVM Accuracy: 78.33 %
```

confusion matrix

Use confusion matrix to understand which type is tend to be mis-classified.

In [22]:

```
from sklearn.metrics import confusion_matrix
y_pred = svm.predict(X_test_std)
c = confusion_matrix(y_test.ravel(), y_pred)/int(y_test.shape[0]/N_type)*100
print(c)
```

```
[[90.          6.66666667  3.33333333  0.          0.        ]
 [11.66666667 78.33333333  6.66666667  1.66666667  1.66666667]
 [ 0.         16.66666667 76.66666667  6.66666667  0.        ]
 [ 0.         13.33333333  5.         76.66666667  5.        ]
 [ 5.         16.66666667  0.          8.33333333 70.        ]]
```