음악(오디오) 파일 클러스터링 절차

1. 오디오 데이터 로드
2. 특징 추출 (Feature Extraction):

- **Chromagram**: 12개의 음계(크로마)를 분석하는 데 사용되는 크로마 특징.
- **MFCCs (Mel-frequency cepstral coefficients)**: 오디오의 음향 특징.
- **Spectral Features**: 주파수 대역에서 에너지 분포와 관련된 특징. (예: Spectral Centroid, Spectral Bandwidth)
- **Zero Crossing Rate**: 신호의 변화 속도에 대한 정보.

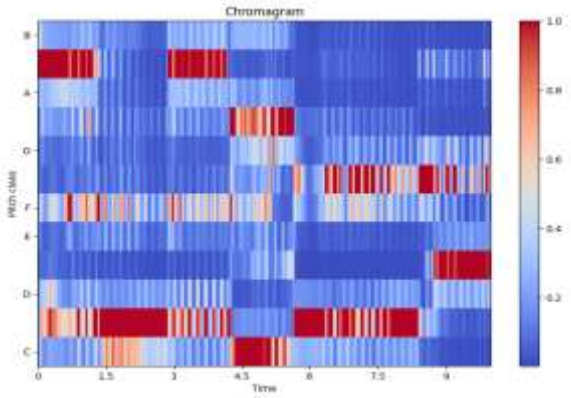
3. 데이터 정규화 (Normalization)

4. 차원 축소 (Dimensionality Reduction): 오디오데이터의 특징은 고차원 데이터이므로, **PCA(Principal Component Analysis)**방법으로 차원을 축소하여 클러스터링을 용이하게 함.

5. 클러스터링 (Clustering): 차원 축소된 데이터를 사용하여 음악 파일들을 군집화.

실습 I. 데이터 정제 및 시각화

```python
import os
import glob
directory = r"C:/Users/dahae/machine learning/ch7_code/7장/music"
full_path_wav = glob.glob(os.path.join(directory, "*.wav"))

wav_file = full_path_wav[0]

import librosa
waveform, sample_rate = librosa.load(wav_file, sr=None,
                                     duration=10) # 10s까지 오디오 파일에서 10초 분량의 신호를 
# resampling - 1/2 with respect to original sample rate
resample_rate = sample_rate /2
resampled = librosa.resample(waveform, orig_sr=sample_rate, #오디오의 속도를 절반으로 절감
                             target_sr=resample_rate)
# 프레임 길이 100ms 설정: sr*2205
chromagram = librosa.feature.chroma_stft(y=resampled, n_fft=2205,
                                         sr = resample_rate, hop_length=512)#크로마그램은 시간

import matplotlib.pyplot as plt
plt.figure(figsize=(10,7))
librosa.display.specshow(chromagram, x_axis='time',
                         y_axis='chroma', cmap='coolwarm')
plt.colorbar()
plt.title('Chromagram')
plt.show()

# 크로마그램 특징벡터 - 히스토그램
import numpy as np
chromagram_vectors = chromagram.T
print(chromagram_vectors)
pitch_class_energy = np.sum(chromagram_vectors, axis=0)
print("----------------------------")
print(pitch_class_energy)
plt.figure(figsize=(10, 7))
pitch_classes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
plt.bar(pitch_classes, pitch_class_energy, color='skyblue')
plt.xlabel('Pitch Class')
plt.ylabel('Count')
plt.title('Histogram of Pitch Classes')
plt.show()

# 히스토그램 정규화
pitch_class_energy = np.sum(chromagram_vectors, axis=0)
normalized_pitch_class_energy = pitch_class_energy / np.sum(pitch_class_energy)
plt.figure(figsize=(10, 7))
pitch_classes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
plt.bar(pitch_classes, normalized_pitch_class_energy, color='skyblue')
plt.xlabel('Pitch Class')
plt.ylabel('Probability')
plt.title('Normalized Histogram of Pitch Classes')
plt.show()
```



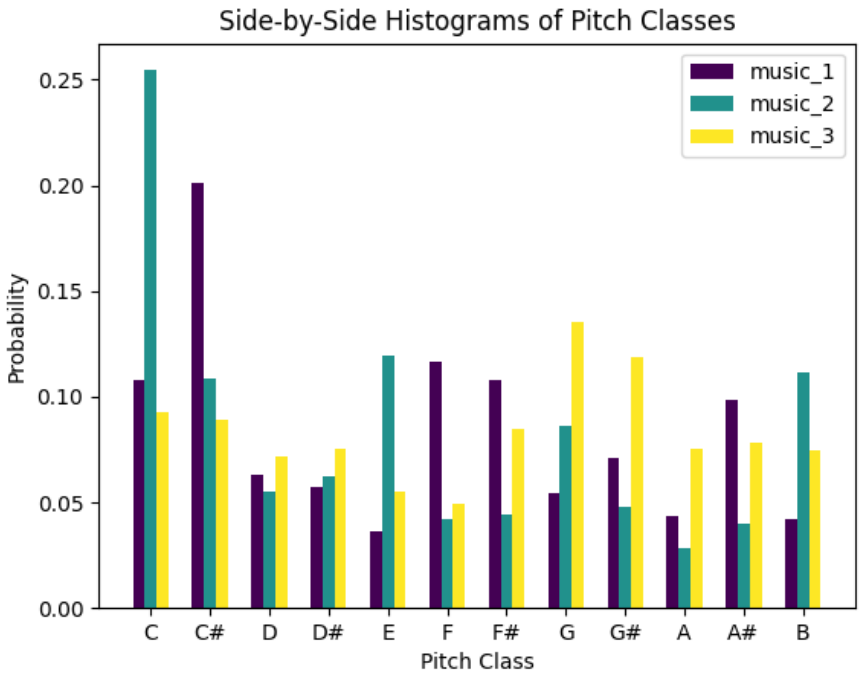
Chromagram 시각화

Chromagram feature 추출

Chromagram feature 정규화

실습 2. 다양한 소리의 특징 벡터 - 결과 그래프

```python

import os
import glob
directory = r"C:/Users/dahae/machine learning/ch7_code/7장/music"
full_path_wav = glob.glob(os.path.join(directory, "*.wav"))

wav_file = full_path_wav[0:3]

import librosa
import numpy as np

hop_length = 512
# 크로마그램과 리샘플, 히스토그램
def compute_normalized_histogram(wav_file, hop_length=hop_length):
    waveform, sample_rate = librosa.load(wav_file, sr=None,
                                         duration=10) # 10s까지
    resample_rate = sample_rate /2
    resampled = librosa.resample(waveform, orig_sr=sample_rate,
                                 target_sr=resample_rate)
    chromagram = librosa.feature.chroma_stft(y=resampled, n_fft=2205,
                                             sr = resample_rate, hop_length=512)
    chromagram_vectors = chromagram.T
    pitch_class_energy = np.sum(chromagram_vectors, axis=0)
    normalized_pitch_class_energy = pitch_class_energy / np.sum(pitch_class_energy)
    return normalized_pitch_class_energy

pitch_classes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
# 막대바의 폭
bar_width = 0.2
bar_positions = np.arange(len(pitch_classes))

import matplotlib.pyplot as plt
# matplotlib의 컬러맵에서 자동으로 색상 생성
colors = plt.cm.viridis(np.linspace(0, 1, len(wav_file)))
for i, (wav_file, color) in enumerate(zip(wav_file, colors)):
    normalized_histogram = compute_normalized_histogram(wav_file)
    positions = bar_positions + i * bar_width
    plt.bar(positions, normalized_histogram, width=bar_width,
            color=color, label=f'music_{i+1}')
plt.xlabel('Pitch Class')
plt.ylabel('Probability')
plt.title('Side-by-Side Histograms of Pitch Classes')
plt.xticks(bar_positions + bar_width, pitch_classes)
plt.legend(loc='upper right')
plt.show()
```



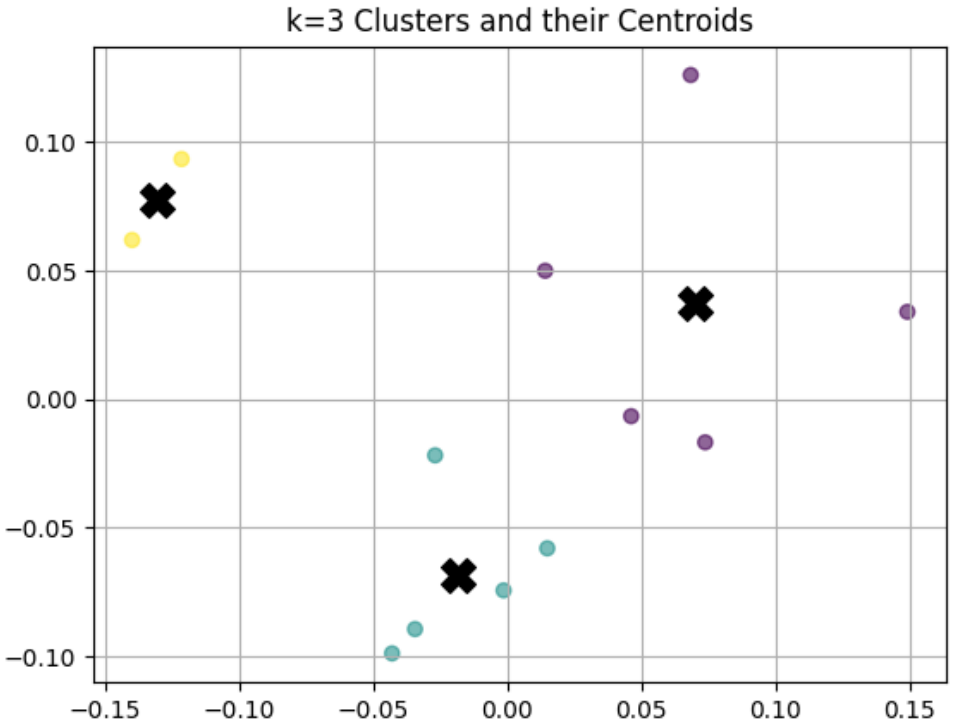
실습 3. Kmeans 클러스터링 코드

```python
import os
import glob
directory = r"C:/Users/dahae/machine learning/ch7_code/7장/music"
full_path_wav = glob.glob(os.path.join(directory, "*.wav"))

wav_file = full_path_wav[:] #모두 선택

import librosa
import numpy as np

hop_length = 512
# 크로마그램과 리샘플, 히스토그램
def compute_normalized_histogram(wav_file, hop_length=hop_length):
    waveform, sample_rate = librosa.load(wav_file, sr=None,
                                         duration=10) # 10s까지
    resample_rate = sample_rate /2
    resampled = librosa.resample(waveform, orig_sr=sample_rate,
                                 target_sr=resample_rate)
    chromagram = librosa.feature.chroma_stft(y=resampled, n_fft=2205,
                                             sr = resample_rate, hop_length=512)
    chromagram_vectors = chromagram.T
    pitch_class_energy = np.sum(chromagram_vectors, axis=0)
    normalized_pitch_class_energy = pitch_class_energy / np.sum(pitch_class_energy)
    return normalized_pitch_class_energy

pitch_classes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
histograms = np.array([compute_normalized_histogram(wav_file)
                       for wav_file in full_path_wav])
input_dim = histograms.shape[1] # 특징벡터 수
```

```python
from sklearn.cluster import KMeans
n_kClusters = 3 #클러스터 개수(k수)
kmeans = KMeans(n_clusters=n_kClusters,init='k-means++',max_iter=5000,
                n_init=10,random_state=42)
history=kmeans.fit_predict(histograms)

cluster_labels = kmeans.labels_
centroids = kmeans.cluster_centers_

for i, label in enumerate(cluster_labels):
    print(os.path.basename(wav_file[i])+f": {label+1}")

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
pca = PCA(n_components=2) # 2D 그래프
reduced_features  = pca.fit_transform(histograms)
reduced_centroids = pca.transform(centroids)
unique_labels = np.unique(cluster_labels) #라벨 확인
plt.scatter([reduced_features[:, 0]], [reduced_features[:, 1]], c=cluster_labels, alpha=0.6)
plt.scatter([reduced_centroids[:, 0]], [reduced_centroids[:, 1]], c='black', marker='X',s=200)
plt.title('k='+f'{n_kClusters}'+' Clusters and their Centroids')
plt.grid()
plt.show()
```

Kmeans 클러스터링

실습 3. Kmeans 클러스터링-특징벡터



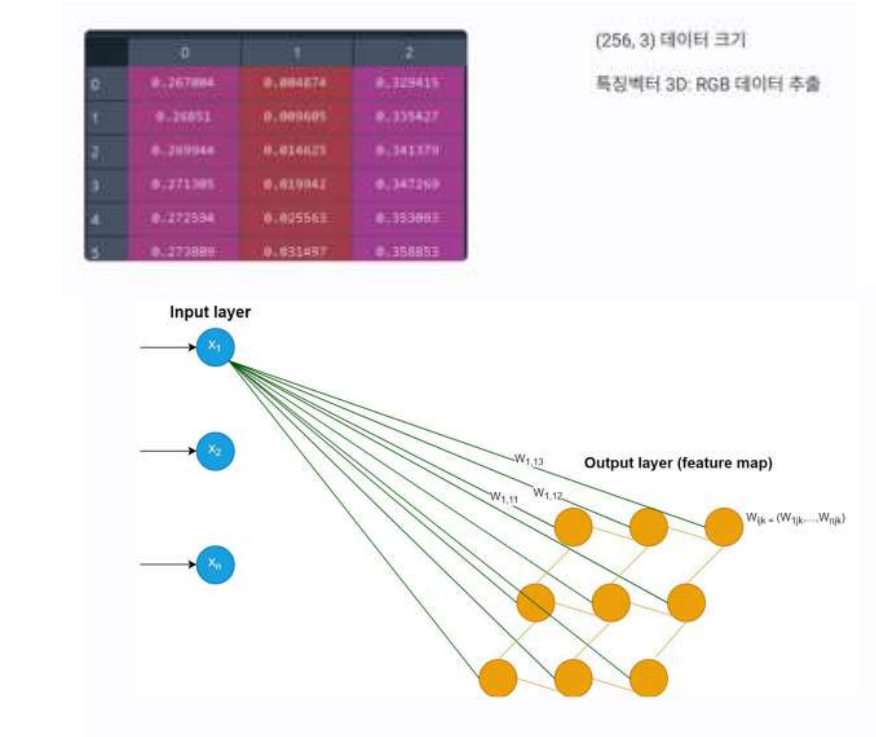
실습 4. SOM_Colormap

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from minisom import MiniSom

# 'viridis' 컬러맵에서 256개의 RGB 데이터를 추출
colormap = cm.get_cmap('viridis', 256)

# RGB 데이터를 추출하고 0-1 사이로 정규화
colormap_data = colormap(np.arange(256))[:, :3]

# SOM 모델 설정 (10x10 그리드, 입력 차원 3 (RGB))
som = MiniSom(x=10, y=10, input_len=3, sigma=1.0, learning_rate=0.5)

# SOM 초기화 및 학습
som.random_weights_init(colormap_data)
som.train_random(colormap_data, 1000)  # 1000회 반복 학습

# 결과 시각화
plt.figure()
plt.bone()
plt.pcolor(som.distance_map().T)
plt.colorbar()
plt.figure(figsize=(7, 7))
for i, x in enumerate(colormap_data):
    w = som.winner(x)
    plt.text(w[0] + 0.5, w[1] + 0.5, str(i), ha='center', va='center',
             bbox=dict(facecolor=x, alpha=0.5, edgecolor='black', boxstyle='round,pad=0.3'))
plt.xlim([0, som.get_weights().shape[0]])
plt.ylim([0, som.get_weights().shape[1]])
plt.grid()
plt.title('SOM colormap',pad=10)
plt.show()
```

입력 데이터 차원

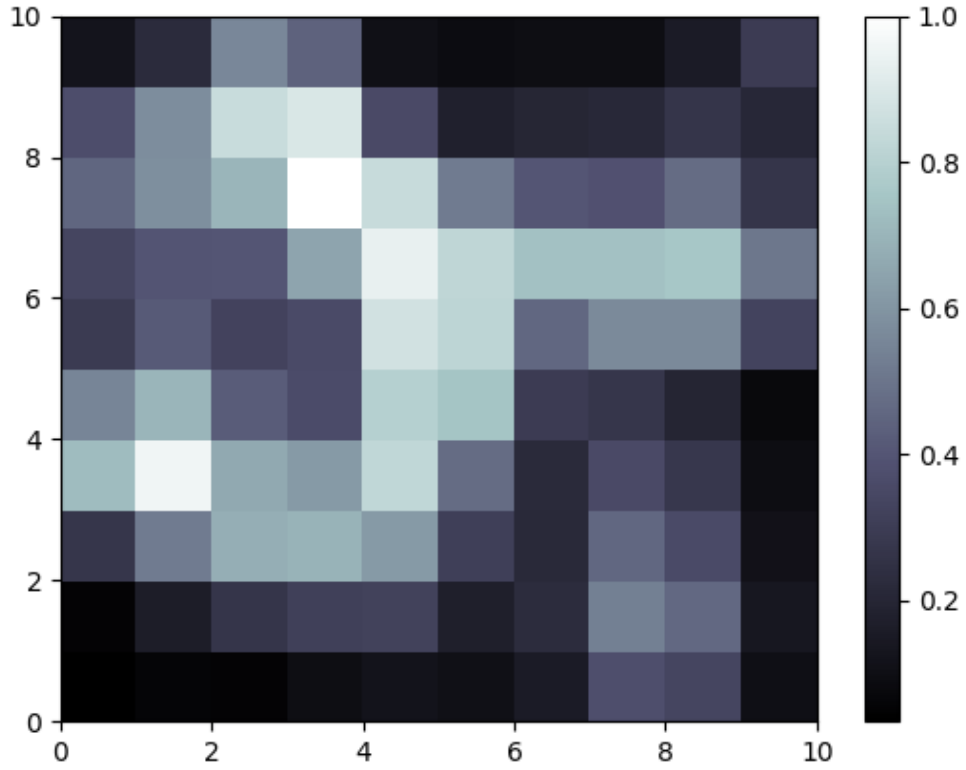
SOM

Viridis

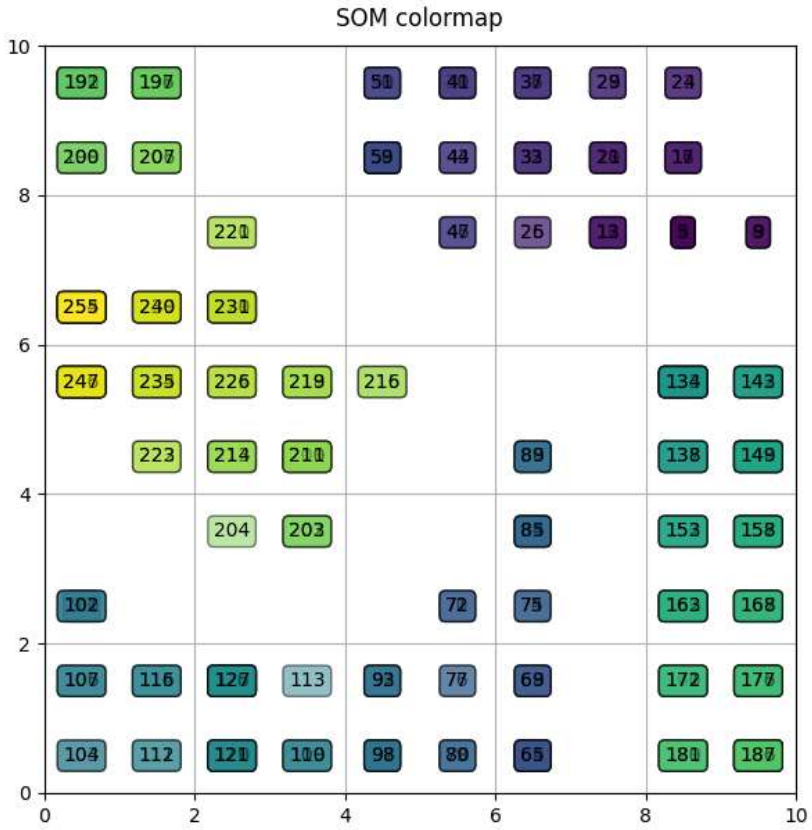


데이터 패턴을 학습하여
유사한 데이터가 인접한 위치에 매핑되도록 함

실습 4. SOM_Colormap 결과



밝은 색상은 클러스터 경계를 나타내고, 어두운 부분은 유사한 데이터가 밀집된 영역



유사한 색상이 가까운 위치에 매핑

실습 4. SOM_iris

```python
from minisom import MiniSom
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
data = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
target = iris['target']
# iris-setosa == 0, iris-versicolor == 1, iris-virginica == 2
# 각 열에 대해 개별적으로 정규화 (열 단위 최댓값으로 나누기)
data = data.apply(lambda x: x / x.max(), axis=0)
data = np.array(data)

som = MiniSom(7, 7, 4, sigma=.1, learning_rate=0.0001)
som.random_weights_init(data)
som.train_random(data, 1000)  # random training
data.shape

plt.figure()
plt.bone()
plt.pcolor(som.distance_map().T)
plt.colorbar()
plt.show()

markers = ['o', 'o', 'o']
colors = ['r', 'g', 'b']
plt.figure(figsize=(7, 7))
for cnt, xx in enumerate(data):
    w = som.winner(xx)  # 해당 데이터의 BMU
    plt.plot(w[0]+.1+0.005*cnt, w[1]+.5, markers[target[cnt]], markerfacecolor='None',
             markeredgecolor=colors[target[cnt]], markersize=8, markeredgewidth=2)
plt.title('Setosa (Red), Versicolor (Green), Virginica (Blue)', pad=20)
plt.xlim([0, som.get_weights().shape[0]])
plt.ylim([0, som.get_weights().shape[1]])
plt.grid()
plt.show()
```

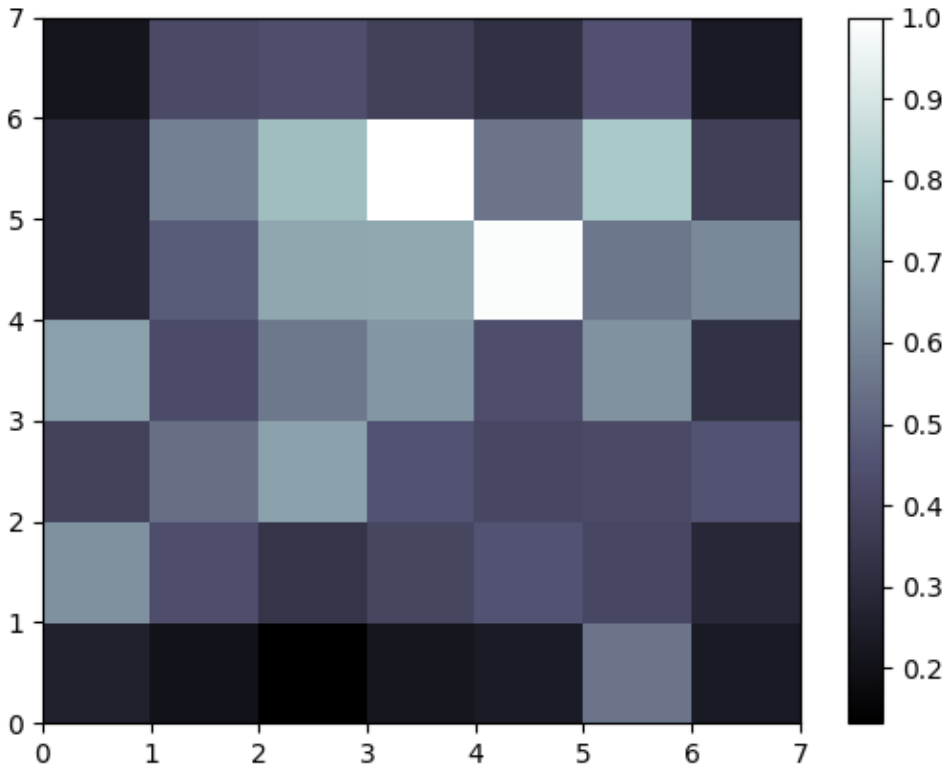
SOM

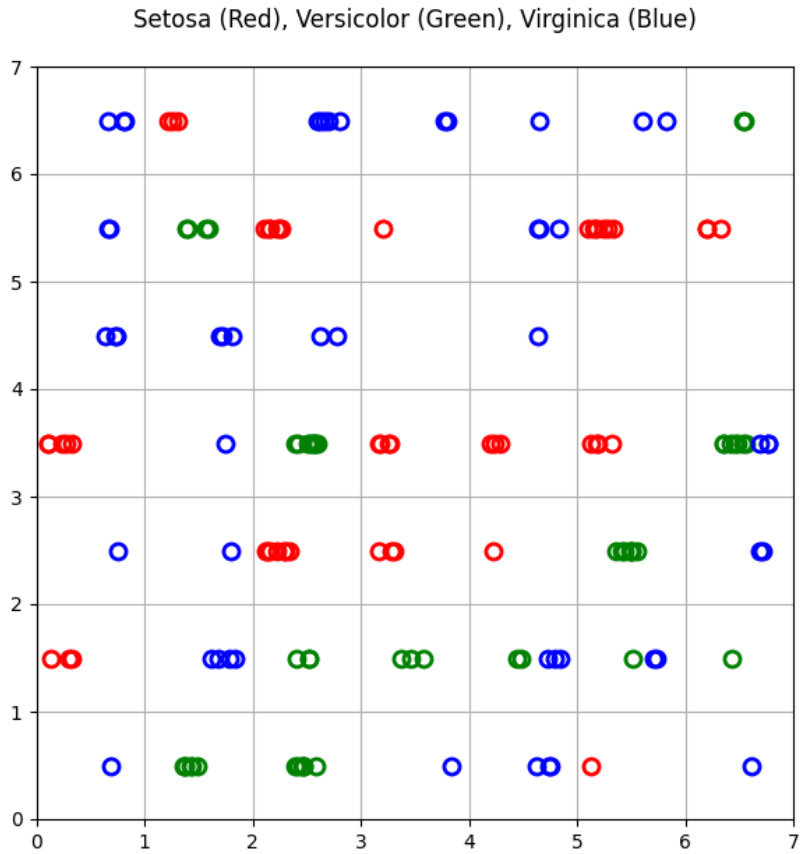
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	4.9	3.0	1.4	0.2	setosa
1	4.7	3.2	1.3	0.2	setosa
2	4.6	3.1	1.5	0.2	setosa
3	6.4	3.2	4.5	1.5	versicolor
4	6.9	3.1	4.9	1.5	versicolor
5	5.5	2.3	4.0	1.3	versicolor
6	7.1	3.0	5.9	2.1	virginica
7	6.3	2.9	5.6	1.8	virginica
8	7.6	3.0	6.6	2.1	virginica

	0	1	2	3
0	0.64557	0.795455	0.202899	0.08
1	0.620253	0.681818	0.202899	0.08
2	0.594937	0.727273	0.188406	0.08
3	0.582278	0.704545	0.217391	0.08
4	0.632911	0.818182	0.202899	0.08
5	0.683544	0.886364	0.246377	0.16
6	0.582278	0.772727	0.202899	0.12
7	0.632911	0.772727	0.217391	0.08
8	0.556962	0.659091	0.202899	0.08
9	0.620253	0.704545	0.217391	0.04
10	0.683544	0.840909	0.217391	0.08
11	0.607595	0.772727	0.231884	0.08
12	0.607595	0.681818	0.202899	0.04
13	0.544304	0.681818	0.15942	0.04

- **150개의 샘플**: 각 샘플은 하나의 아이리스
- **4개의 특징 (특성)**:
  i. **sepal length** (꽃받침 길이, cm)
  ii. **sepal width** (꽃받침 너비, cm)
  iii. **petal length** (꽃잎 길이, cm)
  iv. **petal width** (꽃잎 너비, cm)
- **3개의 클래스 (라벨)**:
  i. **Setosa** (세토사)
  ii. **Versicolor** (버시컬러)
  iii. **Virginica** (버지니카)

실습 4. SOM_iris 결과





밝은 색상은 클러스터 경계를 나타내고, 어두운 부분은 유사한 데이터가 밀집된 영역