Student Notebook - Lecture 11

In this lecture, we will investigate different methods for clustering time series:

- Aggregating the data
- Using distance metrics that can handle vectors (e.g. Euclidean distance)
- Using dynamic time warping

We will use spectral clustering for all experiments. Furthermore, we will again use a synthetic data set to explore the characteristics of the different approaches.

Using our synthetic data, we are interested in exploring procrastination. For this purpose, we will cluster the data of 30 high-school students based on their usage of an academic learning platform. The dataset contains the number of hours per biweek of the year that each student spent on the platform.

The dataset is described by the following columns:

- student id: unique student identifier-
- biweek of the year: number of the biweek of the school year. Biweek 0 refers to the first two wekeks of the school year.
- hours: number of hours the student spent on the platform for that particular biweek-
- student type: expert tagging of student behavior, where (1) is procrastinators, (2) regular students, and (3) precrastinators. We will use the expert label as ground truth for the clustering.

```
#Important imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tslearn.metrics import cdist_dtw
from sklearn.preprocessing import StandardScaler

from scipy.spatial import distance
from scipy.sparse.csgraph import laplacian
from scipy import linalg
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.metrics import silhouette_score
from sklearn.metrics.pairwise import pairwise_kernels
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import spectral_embedding

# Data directory
DATA_DIR = "./../../data/"
```

```
df = pd.read_csv('{}/hours_biweek_students.csv'.format(DATA_DIR))
df.head()
```

```
   student_id  biweek_of_year      hours  student_type
0           0               0  39.915507             3
1           0               1  32.356082             3
2           0               2  17.456692             3
3           0               3  16.012725             3
4           0               4  15.859812             3
```

In a first step, we extract a time series (of biweeks) for each student.

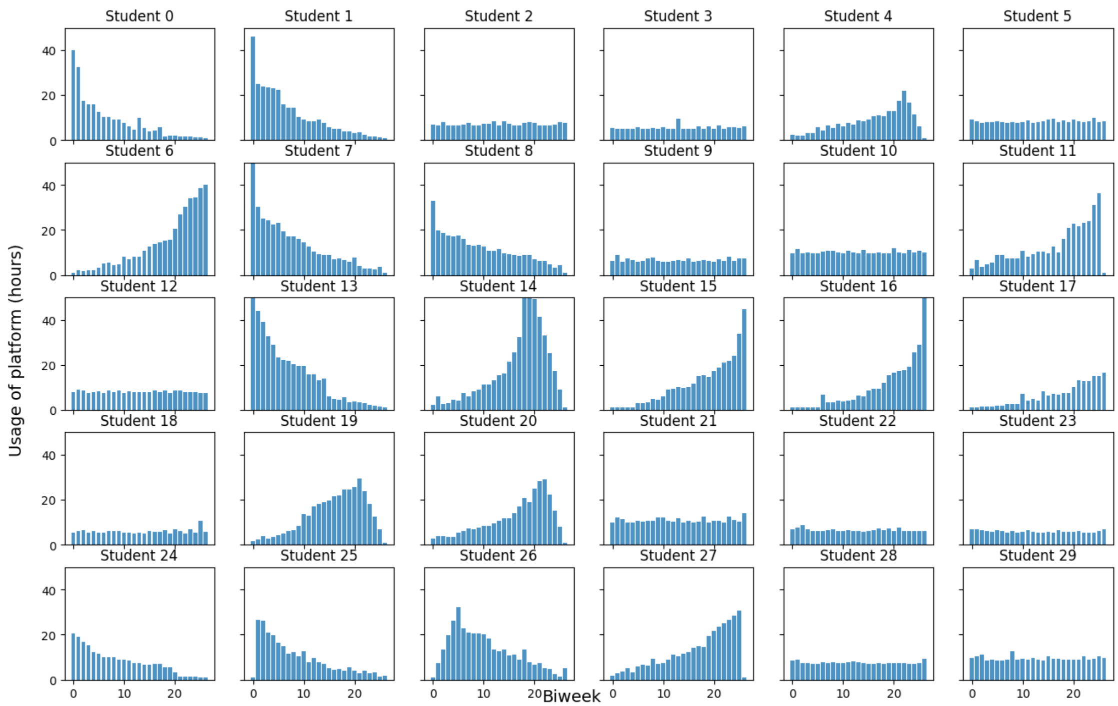
```
def get_time_series(df):
    """
    reshapes DataFrame from long to wide and returns an np.array
    :param df: pd.DataFrame with data in long format
    :return: np.array with reshaped data
    """
    df_array = (df.sort_values(['student_id', 'biweek_of_year'],
ascending=True)
                .groupby('student_id')
                .agg({'hours': lambda x: list(x)}))

    data = np.asarray(df_array.hours.values.tolist())
    return data


data = get_time_series(df)
data.shape
```

```
(30, 27)
```

We then plot the time series data for each student. The three student types are visually very well separable.

```
def plot_students(data):
    """
    Plot the students time-series
    :param data: np.array with students' time-series
    :return:
    """
    students, biweeks = data.shape
    fig, axs = plt.subplots(5, 6, figsize=(16, 10), sharex=True,
                            sharey=True, facecolor='w', edgecolor='k')
    axs = axs.ravel()
    for i in range(students):
        axs[i].bar(range(biweeks), data[i], alpha=0.8)
        axs[i].set_ylim([0, 50])
        axs[i].set_title('Student {0}'.format(i))
    fig.text(0.5, 0.09, 'Biweek', va='center', ha='center',
fontsize=14)
```

```
    fig.text(0.09, 0.5, 'Usage of platform (hours)', va='center',
ha='center', rotation='vertical', fontsize=14)


plot_students(data)
```



Next, we implement some helper functions needed to perform spectral clustering. Specifically, we provide the following functions:

- get_adjacency: computes the adjacency matrix W from a pairwise similarity matrix S
- spectral_clustering: performs spectral clustering for a given number of clusters k, based on an adjacency matrix W
- get_heuristics_spectral: performs spectral clustering for k=2,...,n clusters and computes the Silhouette score and eigengap heuristic for each k
- plot_metrics: visualizes the heuristics for the number of clusters

```
def get_adjacency(S, connectivity='full'):
    """
    Computes the adjacency matrix
    :param S: np array of similarity matrix
    :param connectivity: type of connectivity
    :return: adjacency matrix
    """

    if(connectivity=='full'):
        adjacency = S
    elif(connectivity=='epsilon'):
        epsilon = 0.5
        adjacency = np.where(S > epsilon, 1, 0)
```

```python
    else:
        raise RuntimeError('Method not supported')

    return adjacency

def spectral_clustering(W, n_clusters, random_state=111):
    """
    Spectral clustering
    :param W: np array of adjacency matrix
    :param n_clusters: number of clusters
    :return: tuple (kmeans, proj_X, eigenvals_sorted)
        WHERE
        kmeans scikit learn clustering object
        proj_X is np array of transformed data points
        eigenvals_sorted is np array with ordered eigenvalues

    """
    # Compute eigengap heuristic
    L = laplacian(W, normed=True)
    eigenvals, _ = linalg.eig(L)
    eigenvals = np.real(eigenvals)
    eigenvals_sorted = eigenvals[np.argsort(eigenvals)]

    # Create embedding
    random_state = np.random.RandomState(random_state)
    proj_X = spectral_embedding(W, n_components=n_clusters,
                                random_state=random_state,
                                drop_first=False)

    # Cluster the points using k-means clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state =
random_state)
    kmeans.fit(proj_X)

    return kmeans, proj_X, eigenvals_sorted

def plot_metrics(n_clusters_list, metric_dictionary):
    """
    Plots metric dictionary (auxilary function)
    [Optional]

    :param n_clusters_list: List of number of clusters to explore
    :param metric_dictionary:
    """
    fig = plt.figure(figsize=(12, 10), dpi=80)
    i = 1

    for metric in metric_dictionary.keys():
        plt.subplot(3, 2, i)
```

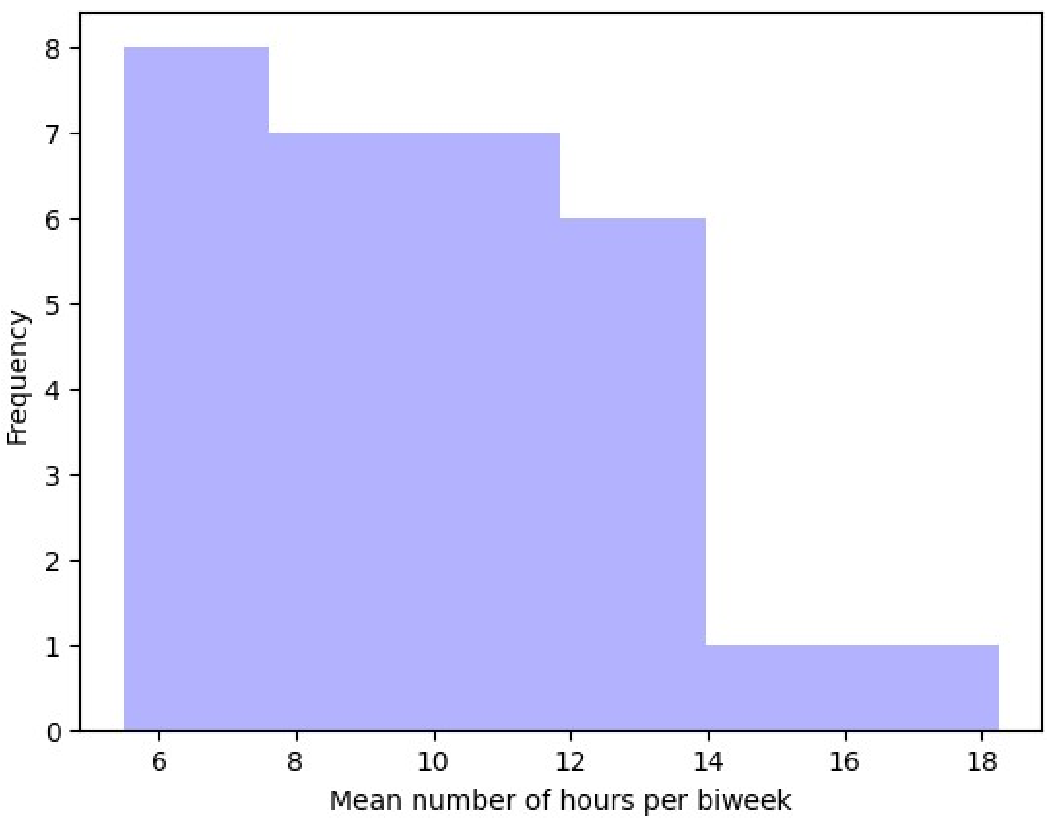
```
        if metric == 'Eigengap':
            clusters = len(n_clusters_list)
            eigenvals_sorted = metric_dictionary[metric]
            plt.scatter(range(1, len(eigenvals_sorted[:clusters * 2])
+ 1), eigenvals_sorted[:clusters * 2])
            plt.xlabel('Eigenvalues')
            plt.xticks(range(1, len(eigenvals_sorted[:clusters * 2]) +
1))
        else:
            plt.plot(n_clusters_list, metric_dictionary[metric], '-o')
            plt.xlabel('Number of clusters')
            plt.xticks(n_clusters_list)
        plt.ylabel(metric)
        i += 1

def get_heuristics_spectral(W, n_clusters_list, plot=True):
    """
    Calculates heuristics for optimal number of clusters with Spectral
Clustering

    :param W: np array of adjacency matrix
    :param n_clusters_list: List of number of clusters to explore
    :plot: bool, plot the metrics if true
    """
    silhouette_list = []
    eigengap_list = []

    df_labels = pd.DataFrame()

    for k in n_clusters_list:

        kmeans, proj_X, eigenvals_sorted = spectral_clustering(W, k)
        y_pred = kmeans.labels_
        df_labels[str(k)] = y_pred

        if k == 1:
            silhouette = np.nan
        else:
            silhouette = silhouette_score(proj_X, y_pred)
        silhouette_list.append(silhouette)


    metric_dictionary = {
                         'Silhouette': silhouette_list,
                         'Eigengap': eigenvals_sorted,
                        }

    if(plot):
        plot_metrics(n_clusters_list, metric_dictionary)
        return df_labels
```

```python
    else:
        return df_labels, metric_dictionary
```

1 - Aggregated Data

The first method we will explore is aggregating features over time. In our example, we will use the mean for the aggreation. We therefore first compute the mean value of our feature (number of hours per biweek) over the whole time series.

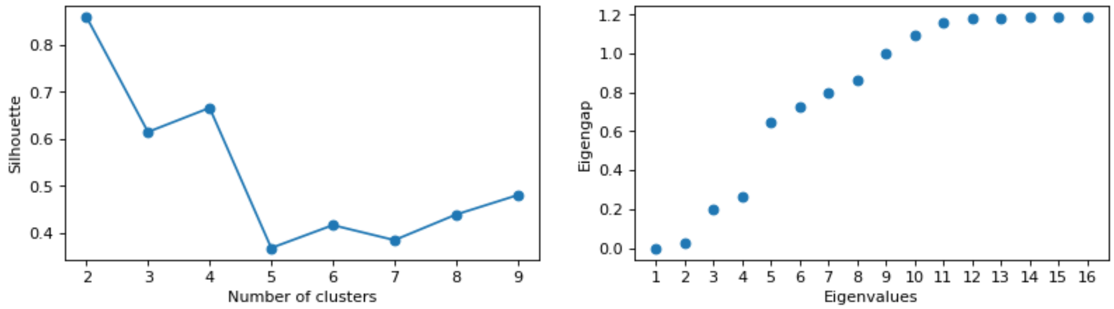
```python
# compute the average of the feature over the whole time series
aggregated_data = np.mean(data, axis = 1)

# plot the histogram of the feature for all students
plt.hist(aggregated_data, bins = 6, alpha = 0.3, color = 'blue')
plt.xlabel('Mean number of hours per biweek')
plt.ylabel('Frequency');
```



We then again build a similarity matrix and a similarity graph and perform spectral clustering for k=2,...10 clusters. We visualize the Silhouette score and the eigengap heuristic.

```python
S = pairwise_kernels(aggregated_data.reshape(-1,1), metric='rbf', 
gamma=1)

W =  get_adjacency(S)
```

```python
n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```



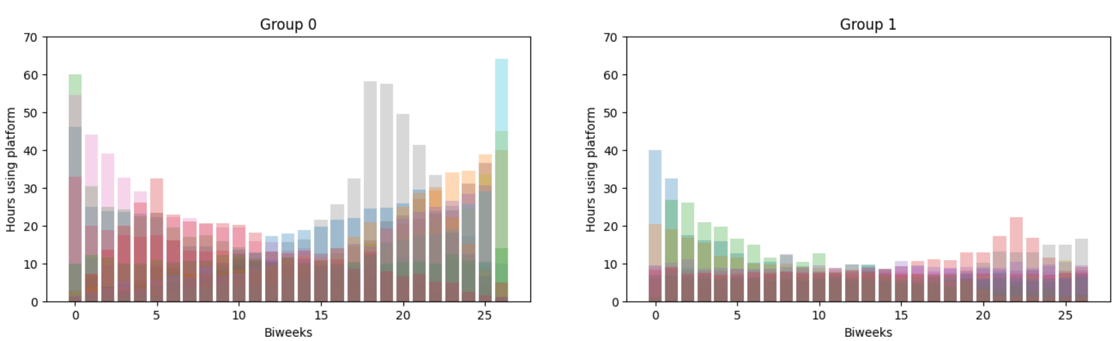
Next, we want to visualize the time series of the students in the different clusters. We implement a function `view_clusters`, which visualizes the average behavior for each cluster. We also implement a function `plot_students_group`, which visualizes the time series of the students in each group.

```python
def view_clusters(data, labels, ylim = 70, xlabel= 'Biweeks'):
    """
    visualize the different time-series of students belonging to each
cluster.
    :param data: np.array with students' time-series
    :param labels: np.array predicted labels from clustering model
    :return:
    """
    _, biweeks = data.shape
    clusters = np.unique(labels).shape[0]
    fig, axs = plt.subplots(1, clusters, figsize=(16, 4),
facecolor='w', edgecolor='k')
    axs = axs.ravel()

    for i in range(clusters):
        students_cluster = data[labels == i]
        number_students = students_cluster.shape[0]
        for student in range(number_students):
            axs[i].bar(range(biweeks), students_cluster[student],
alpha=0.3)

        axs[i].set_ylim([0, ylim])
        axs[i].set_title('Group {0}'.format(i))
        axs[i].set_ylabel('Hours using platform')
        axs[i].set_xlabel(xlabel)

def plot_students_group(data, labels):
    """
    Plot the students time-series
    :param data: np.array with students' time-series
    :param labels: pd.Series indicating the labels of the students
    :return:
```

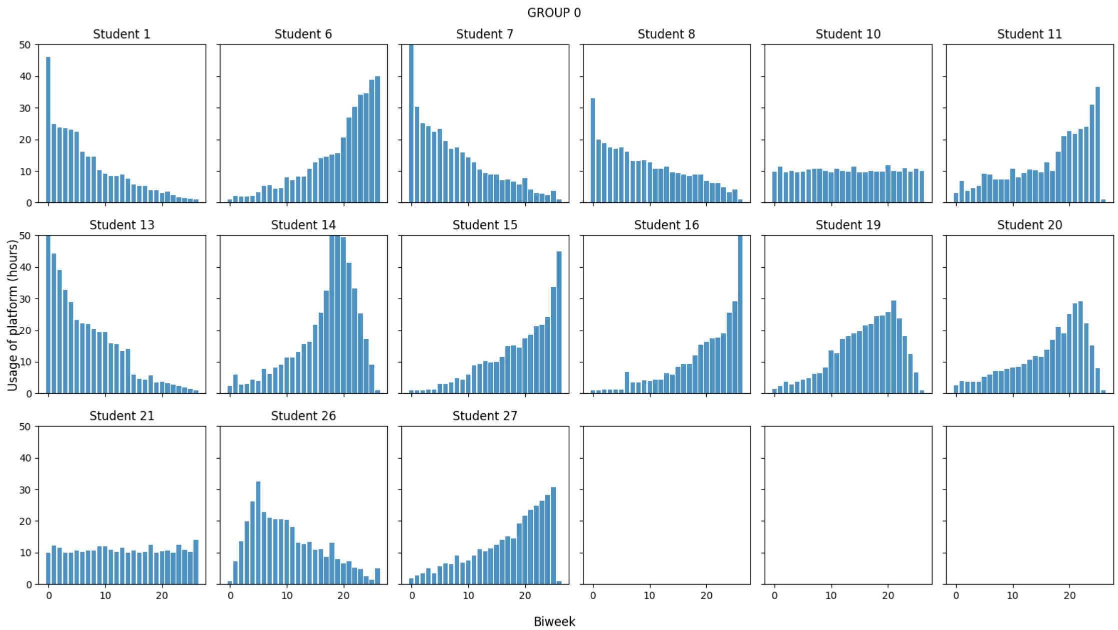
```
    """
    for group in np.unique(labels):
        subdata = data[labels==group]
        subindex = labels[labels==group].index
        students, biweeks = subdata.shape

        rows = int(np.ceil(students/6))
        fig, axs = plt.subplots(rows, 6, figsize=(16, rows*3),
sharex=True,
                                sharey=True, facecolor='w', edgecolor='k')

        axs = axs.ravel()
        for i in range(students):
            axs[i].bar(range(biweeks), subdata[i], alpha=0.8)
            axs[i].set_ylim([0, 50])
            axs[i].set_title('Student {0}'.format(subindex[i]))

        fig.suptitle('GROUP {}'.format(group))
        fig.supxlabel('Biweek')
        fig.supylabel('Usage of platform (hours)')
        plt.tight_layout()
        plt.show()
```

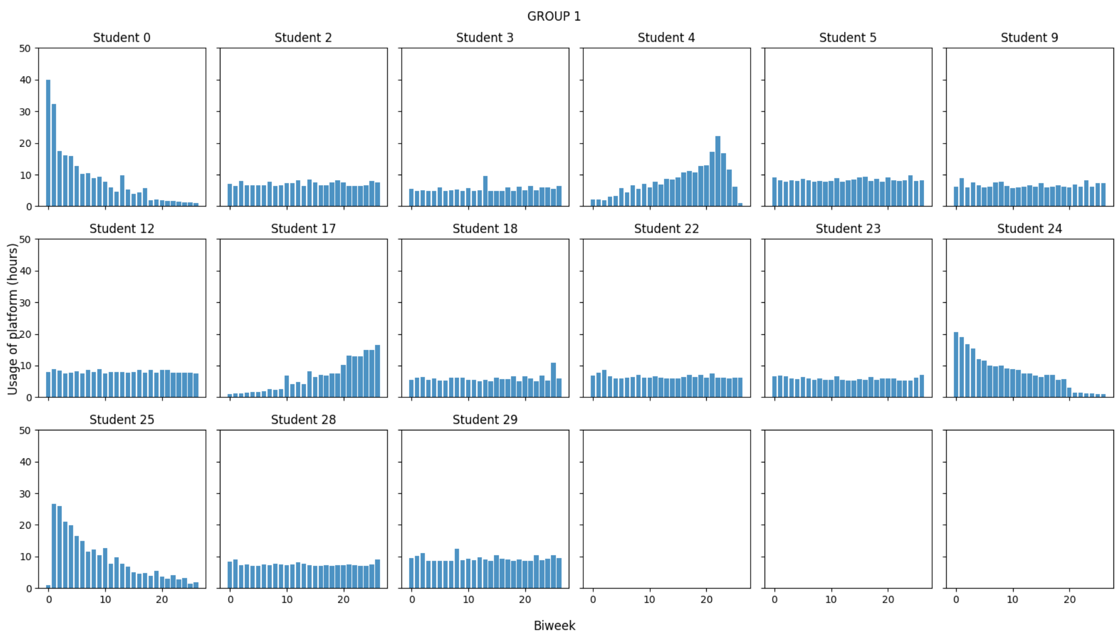
Both the Silhouette score and the eigengap heuristic suggest that the optimal number of clusters is 2. We visualize the mean behavior as well as the time series data of the students in each group.

```
k = 2
view_clusters(data, df_labels[str(k)])
```




```
plot_students_group(data, df_labels[str(k)])
```





Your Turn - Task 1

Discuss your observations and send them to us through the SpeakUp Chat (or through this notebook):

- Can you interpret the obtained clusters?
- Is the approach able to retrieve the procrastination patterns? If not, why not?

```
# Notebook option
answer = """
Can you interpret the obtained clusters?
"""
```

```
send(answer, 11)

answer = """
Is the approach able to retrieve the procrastination patterns? If not,
why not?
"""
send(answer, 12)
```

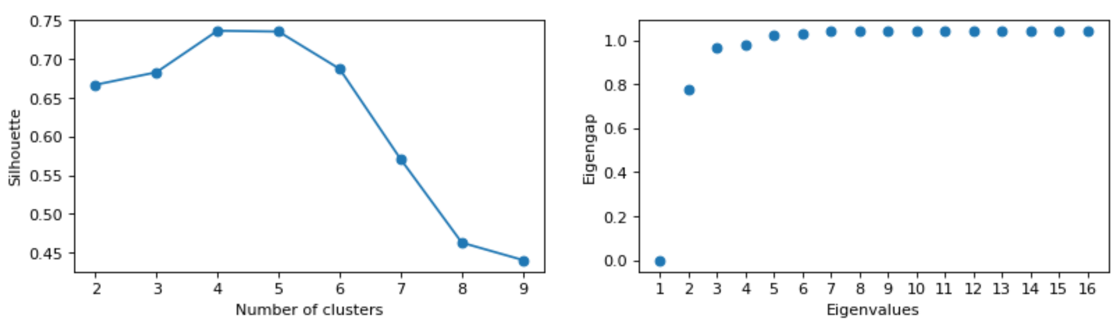
2 - Assuming fixed time intervals

Given the fact that all the students have the same number of biweeks (worth a year), the time series of each student has the same length. We can therefore simply use the Euclidean distance to compute the pairwise distances. In order to avoid clustering by the absolute number of hours and capture students individual differences over the semester (i.e. students who work more at the beginning of the semester and then less over the course of the semester) we normalize the data for each student (i.e. within the student's time series).

```
X = data
norms = np.linalg.norm(X, axis=1)
data_normalized = X / norms[:, np.newaxis]
```

We then again perform a spectral clustering and visualize the heuristics.

```
S = pairwise_kernels(data_normalized, metric='rbf', gamma=1)
W =  get_adjacency(S)

n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```



Your Turn - Task 2

Discuss your observations and send them to us through the SpeakUp Chat (or through this notebook):

- What is the optimal number of clusters k*?
- Can you interpret the obtained clusters?Visualize the average cluster behavior as well as the time series per student of feach group for k*.

Hint: make use of the functions `view_clusters` and `plot_students_group`.

```
# Notebook option
answer = """
What is the optimal number of clusters k*?
"""
send(answer, 21)

answer = """
Can you interpret the obtained clusters?Visualize the average cluster
behavior as well as the time series per student of feach group for k*.
"""
send(answer, 22)

# YOUR VISUALIZATION CODE HERE

send(plt, 23)
```

3- Dynamic Time Warping

Dynamic time warping allows us to align to sequences in an optimal way by choosing a window size w larger than 0.

We first implement a distance function for computing the dynamic time warping distance for a fixed window size w.

```
def get_distance_matrix(X, metric='euclidean', window=2):
    """
    calculates distance matrix given a metric
    :param X: np.array with students' time-series
    :param metric: str distance metric to compute
    :param window: int for DTW
    :return: np.array with distance matrix
    """
    norms = np.linalg.norm(X, axis=1)
    data_normalized = X / norms[:, np.newaxis]

    if metric == 'dtw':
        distance_matrix = cdist_dtw(data_normalized,
                                    global_constraint='sakoe_chiba',
                                    sakoe_chiba_radius=window)
    else:
        distance_vector = distance.pdist(data_normalized, metric)
        distance_matrix = distance.squareform(distance_vector)
    return distance_matrix
```

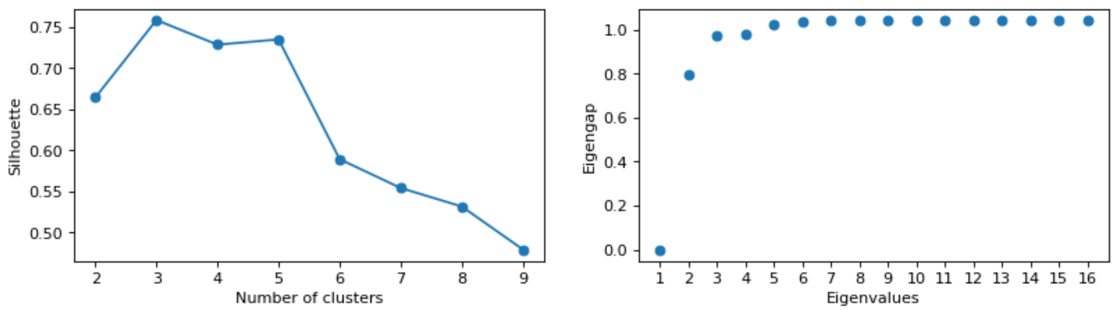
We then also implement a function that computes the similarity matrix for us based on the pairwise distances.

```
def get_affinity_matrix(D, gamma=1):
    """
    calculates affinity matrix from distance matrix
    :param D: np.array distance matrix
```

```
    :param gamma: float coefficient for Gaussian Kernel
    :return:
    """
    S = np.exp(-gamma * D ** 2)
    return S
```

We then compute pairwise distances using a window size of 6. Subsequently, we compute the similarity matrix and the adjacency matrix and then again perform spectral clustering and visualize the cluster heuristics.

```
D = get_distance_matrix(data, metric='dtw', window=6)

S = get_affinity_matrix(D)
W = get_adjacency(S)

n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```



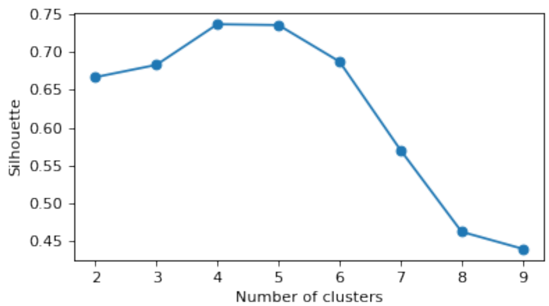
Your Turn - Task 3

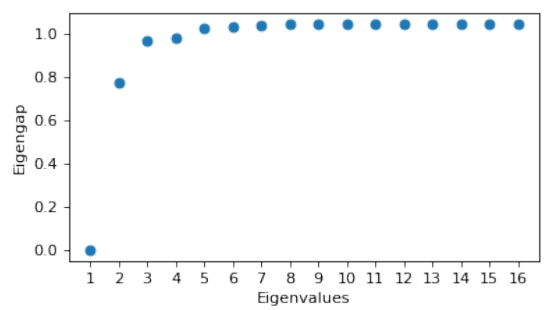
Investigate the clustering results when using DTW (with w = 6). Discuss your observations and send them to us through the SpeakUp Chat (or through this notebook):

- What is the optimal number of clusters k*?
- Can you interpret the obtained cluster? Visualize the average cluster behavior as well as the time series per student of feach group for k*.
- How do the results change for w = 0 and w = 27?

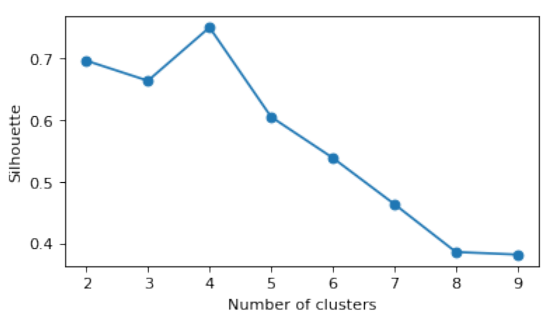
```
# Notebook option
answer = """
What is the optimal number of clusters k*?
"""
send(answer, 31)

answer = """
Can you interpret the obtained cluster? Visualize the average cluster
behavior as well as the time series per student of feach group for k*.
"""
send(answer, 32)

answer = """
```

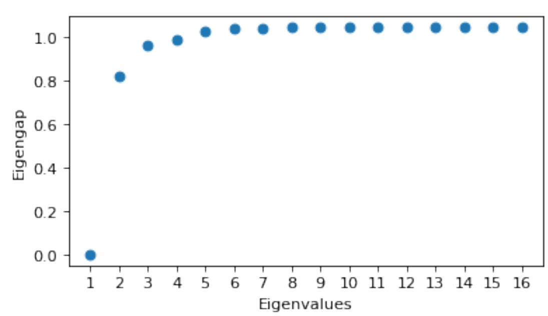
```
How do the results change for w = 0 and w = 27?
"""
send(answer, 33)

# YOUR VISUALIZATION CODE HERE
send(plt, 34)

# windows size 0
D = get_distance_matrix(data, metric='dtw', window=0)
S = get_affinity_matrix(D)
W = get_adjacency(S)
n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```






```
# YOUR VISUALIZATION CODE HERE
send(plt, 35)

# windows size 27
D = get_distance_matrix(data, metric='dtw', window=27)
S = get_affinity_matrix(D)
W = get_adjacency(S)
n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```






```
# YOUR VISUALIZATION CODE HERE
send(plt, 36)
```

Your Turn - Task 4

Discuss the clustering results and send us your observations:

- What is the optimal number of clusters?

- Can you interpret the obtained clusters? Hint: you can use the function `view_clusters` for visualization

```
answer = """
What is the optimal number of clusters?
"""
send(answer, 41)

answer = """
Can you interpret the obtained clusters?
"""
send(answer, 42)

#YOUR VISUALIZATION CODE HERE
send(plt, 43)
```