Clustering_excercise

May 29, 2019

1 Clustering

```
In [ ]: # %load_ext watermark
        # %watermark -a "Sebastian Raschka" -u -d -v -p numpy,pandas,matplotlib,scipy,sklearn
```

The use of `watermark` is optional. You can install this IPython extension via "`pip install watermark`". For more information, please see: https://github.com/rasbt/watermark.

```
In [ ]: from IPython.display import Image
        %matplotlib inline
```

2 Grouping objects by similarity using k-means

2.1 K-means clustering using scikit-learn

```
In [ ]: from sklearn.datasets import make_blobs

        X, y = make_blobs(n_samples=150,
                          n_features=2,
                          centers=3,
                          cluster_std=0.5,
                          shuffle=True,
                          random_state=0)
```

```
In [ ]: print(X,y)
```

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

        plt.scatter(X[:, 0], X[:, 1],
                    c='white', marker='o', edgecolor='black', s=50)
        plt.grid()
        plt.tight_layout()
        #plt.savefig('images/11_01.png', dpi=300)
        plt.show()
```

```
In [ ]: from sklearn.cluster import KMeans
```

```
        km = KMeans(n_clusters=3,
                    init='random',
                    max_iter = 300,
                    random_state=0)

        y_km = km.fit_predict(X)
```

```
In [ ]: print(y_km)
```

```
In [ ]: X[y_km == 0, 0]
```

```
In [ ]: X[y_km == 1, 0]
```

```
In [ ]: print(km.cluster_centers_)
```

```
In [ ]: plt.scatter(X[y_km == 0, 0],
                    X[y_km == 0, 1],
                    s=50, c='lightgreen',
                    marker='s', edgecolor='black',
                    label='cluster 1')
        plt.scatter(X[y_km == 1, 0],
                    X[y_km == 1, 1],
                    s=50, c='orange',
                    marker='o', edgecolor='black',
                    label='cluster 2')
        plt.scatter(X[y_km == 2, 0],
                    X[y_km == 2, 1],
                    s=50, c='lightblue',
                    marker='v', edgecolor='black',
                    label='cluster 3')
        plt.scatter(km.cluster_centers_[:, 0],
                    km.cluster_centers_[:, 1],
                    s=250, marker='*',
                    c='red', edgecolor='black',
                    label='centroids')
        plt.legend(scatterpoints=1)
        plt.grid()
        plt.tight_layout()
        #plt.savefig('images/11_02.png', dpi=300)
        plt.show()
```

2.2 Using the elbow method to find the optimal number of clusters

```
In [ ]: print('Distortion: %.2f' % km.inertia_)
```

```
In [ ]: # kmeans++ -       ,  kmeans
        # inertia_ :  SSE

        distortions = []
        for i in range(1, 11):
            km = KMeans(n_clusters=i,
                        init='random',
                        random_state=0)
            km.fit(X)
            distortions.append(km.inertia_)
        plt.plot(range(1, 11), distortions, marker='o')
        plt.xlabel('Number of clusters')
        plt.ylabel('Distortion')
        plt.tight_layout()
        #plt.savefig('images/11_03.png', dpi=300)
        plt.show()

In [ ]: from sklearn.cluster import KMeans

        km = KMeans(n_clusters=3,
                    init='k-means++',
                    max_iter = 300,
                    random_state=0)

        y_km = km.fit_predict(X)

In [ ]: plt.scatter(X[y_km == 0, 0],
                    X[y_km == 0, 1],
                    s=50, c='lightgreen',
                    marker='s', edgecolor='black',
                    label='cluster 1')
        plt.scatter(X[y_km == 1, 0],
                    X[y_km == 1, 1],
                    s=50, c='orange',
                    marker='o', edgecolor='black',
                    label='cluster 2')
        plt.scatter(X[y_km == 2, 0],
                    X[y_km == 2, 1],
                    s=50, c='lightblue',
                    marker='v', edgecolor='black',
                    label='cluster 3')
        plt.scatter(km.cluster_centers_[:, 0],
                    km.cluster_centers_[:, 1],
                    s=250, marker='*',
                    c='red', edgecolor='black',
                    label='centroids')
        plt.legend(scatterpoints=1)
        plt.grid()
        plt.tight_layout()
        #plt.savefig('images/11_02.png', dpi=300)
        plt.show()

In [ ]: # kmeans++ -      ,   kmeans
```

```
# inertia_ :  SSE

distortions = []
for i in range(1, 11):
    km = KMeans(n_clusters=i,
                init='k-means++',
                random_state=0)
    km.fit(X)
    distortions.append(km.inertia_)
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
#plt.savefig('images/11_03.png', dpi=300)
plt.show()
```

2.3 silhouette

-
- a -   ,
- b -   ,
- silhouette = (b-a)/max(b,a)

```
In [ ]: import numpy as np
        from matplotlib import cm
        from sklearn.metrics import silhouette_samples

        km = KMeans(n_clusters=3,
                    init='k-means++',
                    random_state=0)
        y_km = km.fit_predict(X)
        print(X[:5])
        print(y_km[:5])

        cluster_labels = np.unique(y_km)
        print(cluster_labels)
        n_clusters = cluster_labels.shape[0]
        print(n_clusters)
        silhouette_vals = silhouette_samples(X, y_km, metric='euclidean')
        print(silhouette_vals[:5])
        print(len(silhouette_vals))

        y_ax_lower, y_ax_upper = 0, 0
        yticks = []
```

```
In [ ]: np.mean(silhouette_samples(X, y_km, metric='euclidean'))
```

```
In [ ]: plt.plot(range(len(silhouette_vals)), sorted(silhouette_vals))
In [ ]: clist = ['g','b','y']
        for i, c in enumerate(cluster_labels):
            c_silhouette_vals = silhouette_vals[y_km == c]
            c_silhouette_vals.sort()
            y_ax_upper += len(c_silhouette_vals)
        #     color = cm.jet(float(i) / n_clusters)
            plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.0,
                     edgecolor='none', color=clist[i])

            yticks.append((y_ax_lower + y_ax_upper) / 2.)
            y_ax_lower += len(c_silhouette_vals)

        silhouette_avg = np.mean(silhouette_vals)
        plt.axvline(silhouette_avg, color="red", linestyle="--")

        plt.yticks(yticks, cluster_labels + 1)
        plt.ylabel('Cluster')
        plt.xlabel('Silhouette coefficient')

        plt.tight_layout()
        #plt.savefig('images/11_04.png', dpi=300)
        plt.show()
```

Comparison to “bad” clustering:

```
In [ ]: km = KMeans(n_clusters=2,
                    init='k-means++',
                    n_init=10,
                    max_iter=300,
                    tol=1e-04,
                    random_state=0)
        y_km = km.fit_predict(X)

        plt.scatter(X[y_km == 0, 0],
                    X[y_km == 0, 1],
                    s=50,
                    c='lightgreen',
                    edgecolor='black',
                    marker='s',
                    label='cluster 1')
        plt.scatter(X[y_km == 1, 0],
                    X[y_km == 1, 1],
                    s=50,
                    c='orange',
                    edgecolor='black',
                    marker='o',
                    label='cluster 2')
        plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
                    s=250, marker='*', c='red')#, label='centroids')
        plt.legend()
        plt.grid()
        plt.tight_layout()
        #plt.savefig('images/11_05.png', dpi=300)
        plt.show()
```

```
In [ ]: cluster_labels = np.unique(y_km)
        n_clusters = cluster_labels.shape[0]
        silhouette_vals = silhouette_samples(X, y_km, metric='euclidean')
        y_ax_lower, y_ax_upper = 0, 0
        yticks = []
        colorlist=['g','b']
        for i, c in enumerate(cluster_labels):
            c_silhouette_vals = silhouette_vals[y_km == c]
            c_silhouette_vals.sort()
            y_ax_upper += len(c_silhouette_vals)
        #     color = cm.jet(float(i) / n_clusters)b
            plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.0,
                     edgecolor='none', color=colorlist[i])

            yticks.append((y_ax_lower + y_ax_upper) / 2.)
            y_ax_lower += len(c_silhouette_vals)

        silhouette_avg = np.mean(silhouette_vals)
        plt.axvline(silhouette_avg, color="red", linestyle="--")

        plt.yticks(yticks, cluster_labels + 1)
        plt.ylabel('Cluster')
        plt.xlabel('Silhouette coefficient')

        plt.tight_layout()
        #plt.savefig('images/11_06.png', dpi=300)
        plt.show()
```

```
In [ ]: # <>    silhouette  .
```

```
In [ ]:
```

3 Hierachical Clustering - Agglomerative Clustering

1.
2.
3.
4. .
5.

- ward - .

- average -   .
- complete -   .

```
In [ ]: import mglearn
```

```
In [ ]: import matplotlib.font_manager as fm
        path = 'C:\\Windows\\Fonts\\ALGER.TTF'
        fontprop = fm.FontProperties(fname=path, size=18)

        mglearn.plots.plot_agglomerative_algorithm()
```

```
In [ ]: from sklearn.cluster import AgglomerativeClustering
        from sklearn.datasets import make_blobs
        import matplotlib.pyplot as plt
        %matplotlib inline

        # import matplotlib.font_manager as fm
        # path = 'C:\\Windows\\Fonts\\ALGER.TTF'
        # fontprop = fm.FontProperties(fname=path, size=18)

        X, y = make_blobs(random_state=1)

        agg = AgglomerativeClustering(n_clusters=3)
        assignment = agg.fit_predict(X)
        print(assignment[:20])

        mglearn.discrete_scatter(X[:, 0],X[:,1], assignment)
        plt.legend([' 0', ' 1', ' 2'], loc='best')
        plt.xlabel(' 0')
        plt.ylabel(' 1')
```

```
In [ ]: mglearn.plots.plot_agglomerative()
```

```
In [ ]: from scipy.cluster.hierarchy import dendrogram, ward

        linkage_array = ward(X) #

        row_dendr = dendrogram(linkage_array, # dendrogram
                               labels=y,
                               )
        plt.tight_layout()
        plt.ylabel('Euclidean distance')
        plt.show()
```

```
In [ ]:
```

```
In [ ]:
```

3.1 Grouping clusters in bottom-up fashion

```
In [ ]: from IPython.display import Image
        Image(filename='./images/11_05.png', width=400)
```

```
In [ ]: import pandas as pd
        import numpy as np

        np.random.seed(123)

        variables = ['X', 'Y', 'Z']
        labels = ['ID_0', 'ID_1', 'ID_2', 'ID_3', 'ID_4']

        X = np.random.random_sample([5, 3])*10
        df = pd.DataFrame(X, columns=variables, index=labels)
        df
```

3.2 Applying agglomerative clustering via scikit-learn

```
In [ ]: from sklearn.cluster import AgglomerativeClustering

        ac = AgglomerativeClustering(n_clusters=3,
                                     affinity='euclidean',
                                     linkage='complete')
        labels = ac.fit_predict(X)
        print('Cluster labels: %s' % labels)
```

```
In [ ]: ac = AgglomerativeClustering(n_clusters=2,
                                     affinity='euclidean',
                                     linkage='complete')
        labels = ac.fit_predict(X)
        print('Cluster labels: %s' % labels)
```

4 Locating regions of high density via DBSCAN

4.1

- - e
- (MinPts) e
- - e (MinPts) , e
- - .
- , e ()
-
- - , ,
- - ()
- random , ,

```
In [ ]: Image(filename='images/11_13.png', width=500)
In [ ]: from sklearn.datasets import make_moons

        X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
        plt.scatter(X[:, 0], X[:, 1])
        plt.tight_layout()
        #plt.savefig('images/11_14.png', dpi=300)
        plt.show()
```

```
In [ ]: # <>   clustering algorithm       ,     .
```

```
In [ ]:
```

```
In [ ]:
```

K-means and hierarchical clustering:

```
In [ ]:
```

Density-based clustering:

5 Summary

...

Readers may ignore the next cell.

```
In [ ]: ! python ../.convert_notebook_to_script.py --input ch11.ipynb --output ch11.py
```