unsupervised-learning-exercise

June 4, 2019

0.1 PCA

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

In [ ]:
```
import numpy as np

        fig, axes = plt.subplots(15, 2, figsize=(10, 20))
        malignant = cancer.data[cancer.target == 0]
        benign = cancer.data[cancer.target == 1]

        ax = axes.ravel()

        for i in range(30):
            _, bins = np.histogram(cancer.data[:, i], bins=50)
            ax[i].hist(malignant[:, i], bins=bins, color=mglearn.cm3(0), alpha=.5)
            ax[i].hist(benign[:, i], bins=bins, color=mglearn.cm3(2), alpha=.5)
            ax[i].set_title(cancer.feature_names[i])
            ax[i].set_yticks(())
        ax[0].set_xlabel("feature size")
        ax[0].set_ylabel("frequency")
        ax[0].legend(["malignant", "benign"], loc="best")
        fig.tight_layout()
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
        from sklearn.datasets import load_breast_cancer
        cancer = load_breast_cancer()
        print(cancer.data.shape)

        scaler = StandardScaler()
        scaler.fit(cancer.data)
        X_scaled = scaler.transform(cancer.data)
```

In [ ]:
```
print(X_scaled[0,:])
```

In [ ]:
```
from sklearn.decomposition import PCA
        #
        pca = PCA(n_components=2)
        #   PCA
```

```
        pca.fit(X_scaled)

        #
        X_pca = pca.transform(X_scaled)
        print("  : {}".format(X_scaled.shape))
        print("  : {}".format(X_pca.shape))
```

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

```
In [ ]: #          .
        #
        import mglearn
        import matplotlib.pyplot as plt
        plt.figure(figsize=(8, 8))
        mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], cancer.target)
        plt.legend(["malignant", "benign"], loc="best")
        plt.gca().set_aspect("equal")
        plt.xlabel("first pca")
        plt.ylabel("second pca")
```

```
In [ ]: print("PCA  : {}".format(pca.components_.shape))
```

```
In [ ]: print("PCA : {}".format(pca.components_))
```

```
In [ ]: plt.matshow(pca.components_, cmap='viridis')
        plt.yticks([0, 1], ["first pca", "second pca"])
        plt.colorbar()
        plt.xticks(range(len(cancer.feature_names)),
                   cancer.feature_names, rotation=60, ha='left')
        plt.xlabel("features")
        plt.ylabel("pca")
```

iris

```
In [ ]: import pandas as pd
        from matplotlib import pyplot as plt
        import seaborn as sns
```

```
In [ ]: from sklearn.datasets import load_iris
        iris = load_iris()
        N = 10  #  10
        X = iris.data[:N, :2]  #

        plt.plot(X.T, 'o--')
        plt.xticks(range(4), ["Length", "Width"])
        plt.xlim(-0.5, 2)
        plt.ylim(2.5, 6)
        plt.title("Iris")
        plt.legend(["Sample {}".format(i + 1) for i in range(N)])
        plt.show()
```

```
In [ ]: plt.figure(figsize=(8, 8))
        ax = sns.scatterplot(0, 1, data=pd.DataFrame(X), s=100, \
                             color=".2", marker="s")
        for i in range(N):
            ax.text(X[i, 0] - 0.05, X[i, 1] + 0.03, "Sample {}".\
                    format(i + 1))
        plt.xlabel("Length")
        plt.ylabel("Width")
        plt.title("Iris (2D)")
        plt.axis("equal")
        plt.show()

In [ ]: pca1 = PCA(n_components=1)
        X_low = pca1.fit_transform(X)
        X2 = pca1.inverse_transform(X_low)

In [ ]: plt.figure(figsize=(7, 7))
        ax = sns.scatterplot(0, 1, data=pd.DataFrame(X), s=100, color=".2", \
                             marker="s")
        for i in range(N):
            d = 0.03 if X[i, 1] > X2[i, 1] else -0.04
            ax.text(X[i, 0] - 0.065, X[i, 1] + d, "{}".format(i))
            plt.plot([X[i, 0], X2[i, 0]], [X[i, 1], X2[i, 1]], "k--")
        plt.plot(X2[:, 0], X2[:, 1], "o-", markersize=10)
        plt.plot(X[:, 0].mean(), X[:, 1].mean(), markersize=10, marker="D")
        plt.axvline(X[:, 0].mean(), c='r')
        plt.axhline(X[:, 1].mean(), c='r')
        plt.grid(False)
        plt.xlabel("Length")
        plt.ylabel("Width")
        plt.title("Iris Dim. Reduction")
        plt.axis("equal")
        plt.show()

In [ ]:

In [ ]: X = iris.data[:, [2,3]] #2,3
        Y = iris.target

In [ ]: NUM = 100
        select = np.random.permutation(150)
        Xtr, Ytr = X[select[:NUM]], Y[select[:NUM]]
        Xte, Yte = X[select[NUM:]], Y[select[NUM:]]
        print(Xtr.shape)
        print(Xte.shape)

In [ ]: from sklearn.decomposition import PCA
        pca = PCA(n_components=2)
        pca.fit(Xtr)
```

```
In [ ]: def prn_pca(pca):
          print('Components, Ratio, Eigen Value, Singular Value')
          for c,r,e,s in zip(pca.components_, pca.explained_variance_ratio_, \
                             pca.explained_variance_, pca.singular_values_):
            print('%s, %.3f, %.3f, %.3f' % (c, r, e, s))

In [ ]: prn_pca(pca)

In [ ]: plt.scatter(Xtr[:, 0], Xtr[:, 1], c=Ytr, cmap='YlGnBu')

In [ ]: Xtr2 = pca.transform(Xtr)
        plt.scatter(Xtr2[:, 0], Xtr2[:, 1], c=Ytr, cmap='YlGnBu')

In [ ]:
```

face

```
In [ ]: from sklearn.datasets import fetch_lfw_people
        people = fetch_lfw_people(min_faces_per_person=50,resize=0.7)
        image_shape = people.images[0].shape

        fig, axes = plt.subplots(2, 5, figsize=(15, 8),
                                 subplot_kw={'xticks': (), 'yticks': ()})
        for target, image, ax in zip(people.target, people.images, axes.ravel()):
            ax.imshow(image)
            ax.set_title(people.target_names[target])

In [ ]: print(image_shape)

In [ ]: print(people.images[0])

In [ ]: plt.imshow(people.images[0])

In [ ]:

In [ ]: people.target[0:10], people.target_names[people.target[0:10]]

In [ ]: print("people.images.shape: {}".format(people.images.shape))
        print(" : {}".format(len(people.target_names)))

In [ ]: #
        counts = np.bincount(people.target)
        #
        for i, (count, name) in enumerate(zip(counts, people.target_names)):
            print("{0:25} {1:3}".format(name, count), end='   ')
            if (i + 1) % 3 == 0:
                print()
```

```
In [ ]: mask = np.zeros(people.target.shape, dtype=np.bool)
        for target in np.unique(people.target):
            mask[np.where(people.target == target)[0][:50]] = 1

        X_people = people.data[mask]
        y_people = people.target[mask]

        # 0~255      0~1   .
        # () MinMaxScaler    .
        X_people = X_people / 255.

In [ ]: from sklearn.neighbors import KNeighborsClassifier
        #
        X_train, X_test, y_train, y_test = train_test_split(
            X_people, y_people, stratify=y_people, random_state=0)
        #      KNeighborsClassifier
        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train, y_train)
        print("1-    : {:.2f}".\
              format(knn.score(X_test, y_test)))

In [ ]: mglearn.plots.plot_pca_whitening()

In [ ]: pca = PCA(n_components=100, whiten=True, random_state=0).fit(X_train)
        X_train_pca = pca.transform(X_train)
        X_test_pca = pca.transform(X_test)

        print("X_train_pca.shape: {}".format(X_train_pca.shape))

In [ ]: knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(X_train_pca, y_train)
        print("  : {:.2f}".format(knn.score(X_test_pca, y_test)))

In [ ]: print("pca.components_.shape: {}".format(pca.components_.shape))
```