Uge 4 Øvelser

Øvelse 1

For this assignment, the group is to experiment with PCA for dimensional reduction and visualisation.

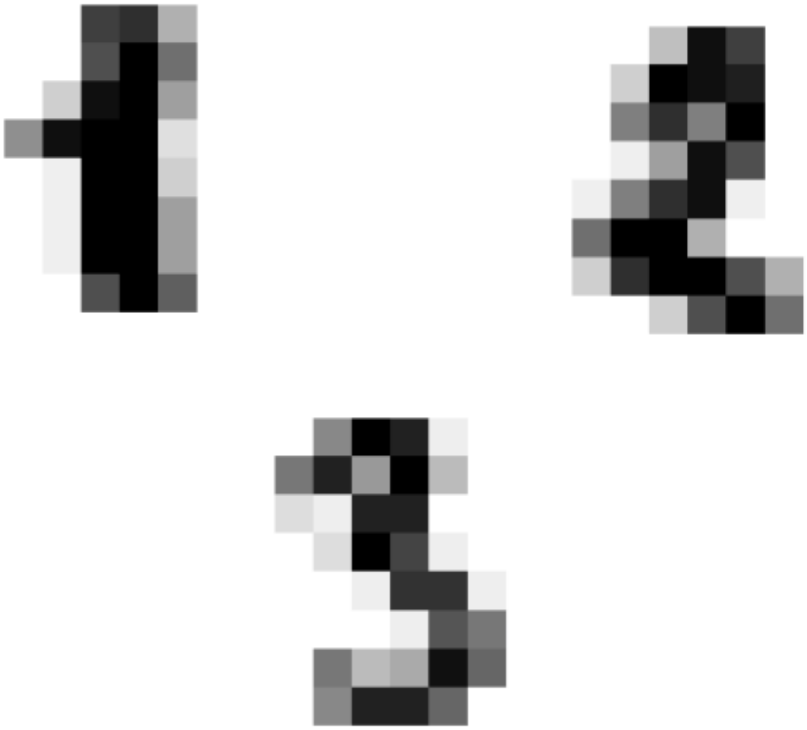
a)

For this first part, we are to read the data(handwritten digits), and plot some example pictures:

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

import matplotlib
import matplotlib.pyplot as plt

# Load digit data and split into training and test data
digits = load_digits()
X = digits.data
y = digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)
```

With the data loaded, some example pictures can be produced:

```
#%% Plot single digit
def plot_digit(digit, cmpsize=64):
    s = int(np.sqrt(cmpsize))
    image = digit.reshape(s, s)
    plt.imshow(image, cmap = matplotlib.cm.binary, interpolation="nearest")
    plt.axis("off")
    plt.show()

digit_index = 5
plot_digit(X[digit_index])
print(y[digit_index])
```



Example pictures of the digits 1, 2 and 3

From the pictures it is seen that the resulution is quite bad, though the digits shape can vaguely be used to determine the digit.

b)

In order to plot in the 2D space, the digits would have to be reduced to only 2 dimensions, meaning to find the vectors projection onto a surface placed in the $N_{digitResolution}$ space. In preperation for other parts of the assignment, and out of curiosity, dimensional reduction is performed with more than only 2 dimensions, and stored in a list of reduced sets:

```
#%% Create PCAs with different number of components
def create_pca(data, cmps):
    pca = PCA(n_components=cmps)
    pca.fit(data)
    return pca

dimensions_sq = [n**2 for n in range(1,7)] # 4, 9, 16 ...
dimensions_2n = [2**n for n in range(1,7)] # 2, 4, 8 ...
dimensions = sorted(set(dimensions_sq + dimensions_2n))

pca = {n: create_pca(X_train, n) for n in dimensions}

X_reduced = {n: pca[n].fit_transform(X_train) for n in dimensions}

print(f"Created {len(dimensions)} pca's of training data with dimensions
{dimensions}")
```

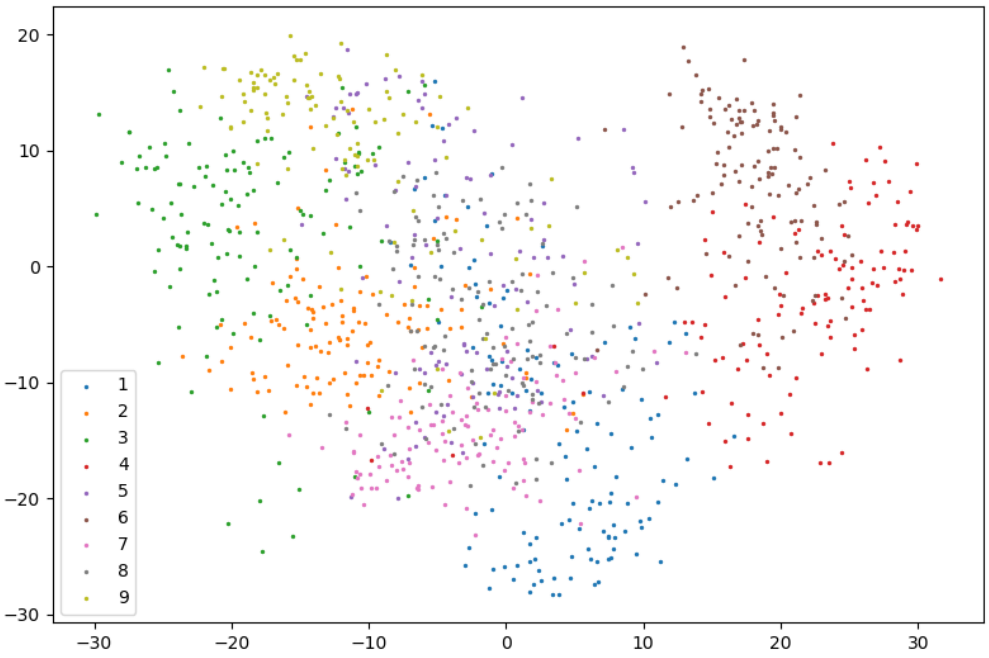
Which outputs:

```
Created 10 pca's of training data with dimensions [1, 2, 4, 8, 9, 16, 25, 32, 36,
64]
```

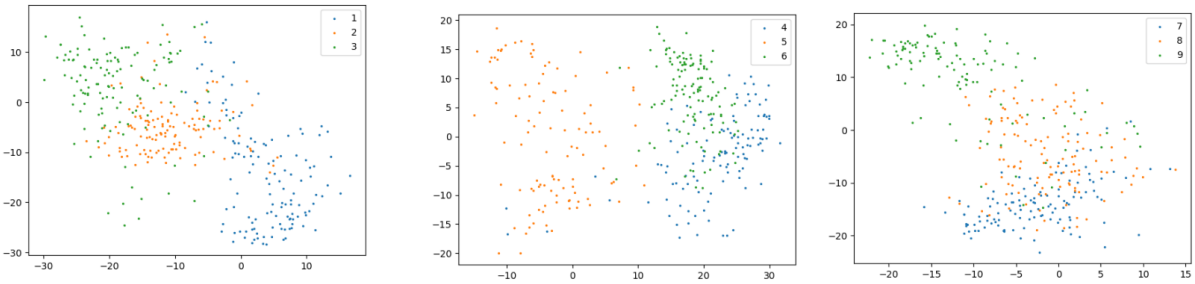
With the above, a scatterplot can be produced. One could extract only two columns from one of the PCA's with more than 2 dimensions, but with the assumption that the variance is maximised of the PCA when perform the fit with only 2 components, this is the one chosen:

```
#%% Split data into which digit it "should" represent
def mkdict1(x_reduced): # Method 1
    xdict={i:[] for i in range(10)}
    for i, x in enumerate(x_reduced):
        xdict[y_train[i]].append(x)
    return {i:np.array(xdict[i]) for i in range(10)}

def mkdict2(x_reduced): # Method 2
    return {
        digit: np.array([x for xi, x in enumerate(x_reduced) if y_train[xi] ==
digit])
        for digit in range(10)
    }

#%% Time the two different mothods of creating dict
from timeit import timeit
r1 = timeit(lambda: mkdict1(X_reduced[2]), number=100)
r2 = timeit(lambda: mkdict2(X_reduced[2]), number=100)
print(f"mkdict1 = {r1}")
print(f"mkdict2 = {r2}")
print("the fastest is {}".format("mkdict1" if r1 < r2 else "mkdict2"))

#%% Plot the two-component representation of digits and color by 'actual' digit
npdict = mkdict1(X_reduced[2])
def plot_two_component_digits(digits):
    plt.figure()
    for dnum in digits:
        plt.scatter(npdict[dnum][:,0], npdict[dnum][:,1], s=2, label=str(dnum))
    # plt.axis('equal')
    plt.legend()
    plt.show()

plot_two_component_digits([1, 2, 3])
plot_two_component_digits([4, 5, 6])
plot_two_component_digits([7, 8, 9])
plot_two_component_digits([1,2,3,4,5,6,7,8,9])
```

The above code yields the plots as seen below:



Scatter plot with all digits in 2D



Scatter plots easily readable pr. digit

From which it is seen that event after dimension reduction, the digits are still somewhat distinguishiable when looked at in sets. When plotted all together, it becomes hard to disinguish them. This makes somewhat sense, as digits can be seen as combination of smaller patterns, e.g both 9 and 1 consists of a vertical line, and as such, when compressing these, we loose resolution, and they will become more similar.