Homework 7 - Unsupervised Learning

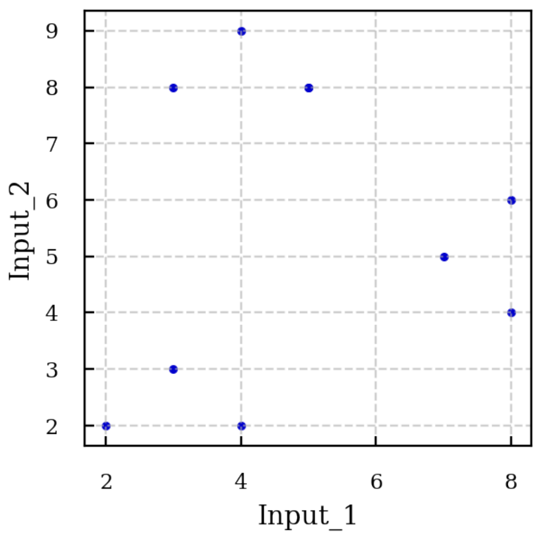
March 15, 2020

Student: Dimitrios Tanoglidis

[1]:
```
#Import stuff
import numpy as np
import pandas as pd
import itertools
from sklearn import linear_model
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import rcParams
#import seaborn as sns
rcParams['font.family'] = 'serif'

# Adjust rc parameters to make plots pretty
def plot_pretty(dpi=200, fontsize=8):

    import matplotlib.pyplot as plt

    plt.rc("savefig", dpi=dpi)
    #plt.rc('text', usetex=True)
    plt.rc('font', size=fontsize)
    plt.rc('xtick', direction='in')
    plt.rc('ytick', direction='in')
    plt.rc('xtick.major', pad=10)
    plt.rc('xtick.minor', pad=5)
    plt.rc('ytick.major', pad=10)
    plt.rc('ytick.minor', pad=5)
    plt.rc('lines', dotted_pattern = [0.5, 1.1])

    return
plot_pretty()
```

1 k-Means Clustering ”By Hand”

[2]:
```
input_1 = np.array([5,8,7,8,3,4,2,3,4,5])
input_2 = np.array([8,6,5,4,3,2,2,8,9,8])

plt.figure(figsize=(3.,3.))
plt.scatter(input_1,input_2, s=5.0, color='mediumblue')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('Input_1', fontsize=10)
plt.ylabel('Input_2', fontsize=10)
plt.show()
```

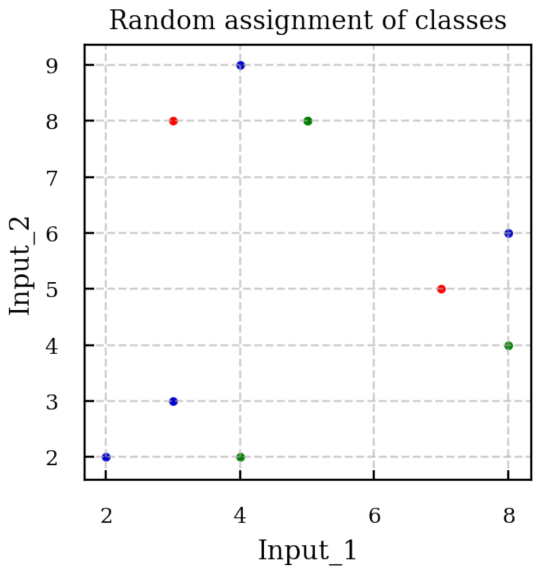


1.1 Initialize

To initialize I will create a random array with the size that of the sample (that is, 10 in our case) and values between 0-2 in each one of the entries.

[3]:
```
n_size = 10 #Size of the sample
k = 3 #k for the k-means
init_class = np.random.randint(0,k,n_size) # Get random guesses for the classes

# Plot the random guesses
plt.figure(figsize=(3.,3.))
plt.title('Random assignment of classes')
plt.scatter(input_1[init_class==0],input_2[init_class==0], s=5.0,␣
↪color='mediumblue')
plt.scatter(input_1[init_class==1],input_2[init_class==1], s=5.0, color='red')
```

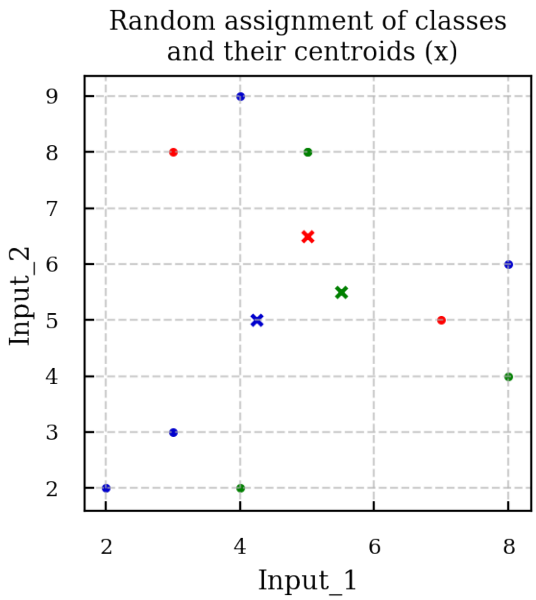
```python
plt.scatter(input_1[init_class==2],input_2[init_class==2], s=5.0, color='green')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('Input_1', fontsize=10)
plt.ylabel('Input_2', fontsize=10)
plt.show()
```



1.2 Implement k-Means clustering iteratively

[4]:
```python
# Start by the first iteration
lab_old = np.copy(init_class) #Old labels

# Calculate centroids of the three clusters

# Cluster 0
in_1_cl0 = input_1[lab_old==0]
in_2_cl0 = input_2[lab_old==0]
m_1_cl0 = np.mean(in_1_cl0)
m_2_cl0 = np.mean(in_2_cl0)
# Cluster 1
in_1_cl1 = input_1[lab_old==1]
in_2_cl1 = input_2[lab_old==1]
m_1_cl1 = np.mean(in_1_cl1)
m_2_cl1 = np.mean(in_2_cl1)
# Cluster 2
in_1_cl2 = input_1[lab_old==2]
in_2_cl2 = input_2[lab_old==2]
```

```
m_1_cl2 = np.mean(in_1_cl2)
m_2_cl2 = np.mean(in_2_cl2)

# Let's print the centroids to see them
# Plot the random guesses
plt.figure(figsize=(3.,3.))
plt.title('Random assignment of classes\n and their centroids (x)')
plt.scatter(input_1[init_class==0],input_2[init_class==0], s=5.0,␣
 ↪color='mediumblue')
plt.scatter(input_1[init_class==1],input_2[init_class==1], s=5.0, color='red')
plt.scatter(input_1[init_class==2],input_2[init_class==2], s=5.0, color='green')

plt.scatter(m_1_cl0,m_2_cl0, s=16.0, marker='x', color='mediumblue')
plt.scatter(m_1_cl1,m_2_cl1, s=16.0, marker='x', color='red')
plt.scatter(m_1_cl2,m_2_cl2, s=16.0, marker='x', color='green')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('Input_1', fontsize=10)
plt.ylabel('Input_2', fontsize=10)
plt.show()
```



[5]:
```
# Now update cluster assignment
lab_new = np.zeros(10) # Array for the new labels

for i in range(10):
    # Coordinates of the point i
    coo_1 = input_1[i]
```

```
    coo_2 = input_2[i]

    # Calculate the distances from the three centroids
    dist_0 = (coo_1-m_1_cl0)**2.0 + (coo_2-m_2_cl0)**2 # Distance from 0
    dist_1 = (coo_1-m_1_cl1)**2.0 + (coo_2-m_2_cl1)**2 # Distance from 1
    dist_2 = (coo_1-m_1_cl2)**2.0 + (coo_2-m_2_cl2)**2 # Distance from 2

    dists = np.asarray([dist_0,dist_1,dist_2])
    inds = np.argsort(dists) # Get the indices of the sorted array of indices

    lab_new[i] = int(inds[0]) # The new label is the index that corresponds to␣
 ↪the minimal distance



lab_new = lab_new.astype(int)
```

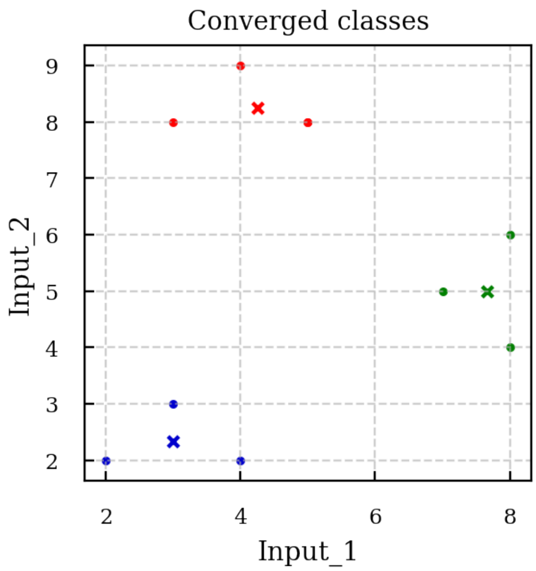
Now we can repeat the above procedure iteratively, till convergence.

[6]:
```
#while ((lab_new-lab_old).any!=0): # Repeat till all the labels do not change␣
 ↪any more
for j in range(100):
    lab_old = np.copy(lab_new)

    # Calculate centroids of the three clusters
    # Cluster 0
    in_1_cl0 = input_1[lab_old==0]
    in_2_cl0 = input_2[lab_old==0]
    m_1_cl0 = np.mean(in_1_cl0)
    m_2_cl0 = np.mean(in_2_cl0)
    # Cluster 1
    in_1_cl1 = input_1[lab_old==1]
    in_2_cl1 = input_2[lab_old==1]
    m_1_cl1 = np.mean(in_1_cl1)
    m_2_cl1 = np.mean(in_2_cl1)
    # Cluster 2
    in_1_cl2 = input_1[lab_old==2]
    in_2_cl2 = input_2[lab_old==2]
    m_1_cl2 = np.mean(in_1_cl2)
    m_2_cl2 = np.mean(in_2_cl2)

    # Now update cluster assignment
    lab_new = np.zeros(10) # Array for the new labels

    for i in range(10):
        # Coordinates of the point i
        coo_1 = input_1[i]
        coo_2 = input_2[i]
```

```
        # Calculate the distances from the three centroids
        dist_0 = (coo_1-m_1_cl0)**2.0 + (coo_2-m_2_cl0)**2 # Distance from 0
        dist_1 = (coo_1-m_1_cl1)**2.0 + (coo_2-m_2_cl1)**2 # Distance from 1
        dist_2 = (coo_1-m_1_cl2)**2.0 + (coo_2-m_2_cl2)**2 # Distance from 2

        dists = np.asarray([dist_0,dist_1,dist_2])
        inds = np.argsort(dists) # Get the indices of the sorted array of indices

        lab_new[i] = int(inds[0]) # The new label is the index that corresponds␣
↪to the minimal distance


    lab_new = lab_new.astype(int)
```

1.3 Final clusters

[7]:
```
plt.figure(figsize=(3.,3.))
plt.title('Converged classes')
plt.scatter(input_1[lab_new==0],input_2[lab_new==0], s=5.0, color='mediumblue')
plt.scatter(input_1[lab_new==1],input_2[lab_new==1], s=5.0, color='red')
plt.scatter(input_1[lab_new==2],input_2[lab_new==2], s=5.0, color='green')

plt.scatter(m_1_cl0,m_2_cl0, s=16.0, marker='x', color='mediumblue')
plt.scatter(m_1_cl1,m_2_cl1, s=16.0, marker='x', color='red')
plt.scatter(m_1_cl2,m_2_cl2, s=16.0, marker='x', color='green')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('Input_1', fontsize=10)
plt.ylabel('Input_2', fontsize=10)
plt.show()
```



We see that after convergence, the assigned clusters are the ”expected” ones. With ”x” we show the centroids.

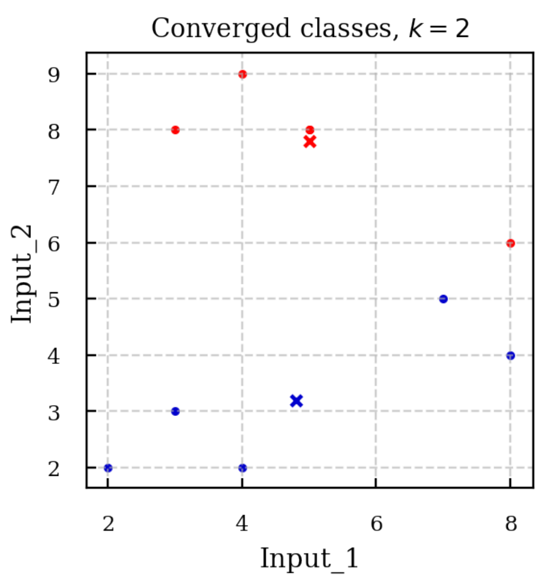
1.4 Repeat for k=2

[8]:
```
n_size = 10 #Size of the sample
k = 2 #k for the k-means
init_class = np.random.randint(0,k,n_size) # Get random guesses for the classes

# Start by the first iteration
lab_old = np.copy(init_class) #Old labels

# Calculate centroids of the two clusters

# Cluster 0
in_1_cl0 = input_1[lab_old==0]
in_2_cl0 = input_2[lab_old==0]
m_1_cl0 = np.mean(in_1_cl0)
m_2_cl0 = np.mean(in_2_cl0)
# Cluster 1
in_1_cl1 = input_1[lab_old==1]
in_2_cl1 = input_2[lab_old==1]
m_1_cl1 = np.mean(in_1_cl1)
m_2_cl1 = np.mean(in_2_cl1)

lab_new = np.zeros(10) # Array for the new labels
```

```python
for i in range(10):
    # Coordinates of the point i
    coo_1 = input_1[i]
    coo_2 = input_2[i]

    # Calculate the distances from the three centroids
    dist_0 = (coo_1-m_1_cl0)**2.0 + (coo_2-m_2_cl0)**2 # Distance from 0
    dist_1 = (coo_1-m_1_cl1)**2.0 + (coo_2-m_2_cl1)**2 # Distance from 1

    dists = np.asarray([dist_0,dist_1,dist_2])
    inds = np.argsort(dists) # Get the indices of the sorted array of indices

    lab_new[i] = int(inds[0]) # The new label is the index that corresponds to␣
 ↪the minimal distance


lab_new = lab_new.astype(int)

for j in range(100):
    lab_old = np.copy(lab_new)

    # Calculate centroids of the two clusters
    # Cluster 0
    in_1_cl0 = input_1[lab_old==0]
    in_2_cl0 = input_2[lab_old==0]
    m_1_cl0 = np.mean(in_1_cl0)
    m_2_cl0 = np.mean(in_2_cl0)
    # Cluster 1
    in_1_cl1 = input_1[lab_old==1]
    in_2_cl1 = input_2[lab_old==1]
    m_1_cl1 = np.mean(in_1_cl1)
    m_2_cl1 = np.mean(in_2_cl1)


    # Now update cluster assignment
    lab_new = np.zeros(10) # Array for the new labels

    for i in range(10):
        # Coordinates of the point i
        coo_1 = input_1[i]
        coo_2 = input_2[i]

        # Calculate the distances from the three centroids
        dist_0 = (coo_1-m_1_cl0)**2.0 + (coo_2-m_2_cl0)**2 # Distance from 0
        dist_1 = (coo_1-m_1_cl1)**2.0 + (coo_2-m_2_cl1)**2 # Distance from 1

        dists = np.asarray([dist_0,dist_1,dist_2])
```

```
        inds = np.argsort(dists) # Get the indices of the sorted array of indices

        lab_new[i] = int(inds[0]) # The new label is the index that corresponds␣
↪to the minimal distance


    lab_new = lab_new.astype(int)
```

[9]:
```
plt.figure(figsize=(3.,3.))
plt.title('Converged classes, $k=2$')
plt.scatter(input_1[lab_new==0],input_2[lab_new==0], s=5.0, color='mediumblue')
plt.scatter(input_1[lab_new==1],input_2[lab_new==1], s=5.0, color='red')


plt.scatter(m_1_cl0,m_2_cl0, s=16.0, marker='x', color='mediumblue')
plt.scatter(m_1_cl1,m_2_cl1, s=16.0, marker='x', color='red')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('Input_1', fontsize=10)
plt.ylabel('Input_2', fontsize=10)
plt.show()
```



1.5 Which one is the best?

Visually it seems that 3 is the best number of clusters. Let's calculate the within cluster sum of squares in the two cases.

Here for ease, I will use the sklearn methods.

[258]:
```
from sklearn.cluster import KMeans
X_inp = np.zeros([10,2])
X_inp[:,0] = input_1
X_inp[:,1] = input_2
```

[259]:
```
kmeanModel_3 = KMeans(n_clusters=3).fit(X_inp)
WCSS_3 = kmeanModel_3.inertia_
print(WCSS_3)
```

```
8.833333333333332
```

[260]:
```
kmeanModel_2 = KMeans(n_clusters=2).fit(X_inp)
WCSS_2 = kmeanModel_2.inertia_
print(WCSS_2)
```

```
46.952380952380956
```

The Within-Cluster-Mean-of-Squares for three clusters is much smaller than when we have two clusters, which confirms our first impression that three clusters better describe the dataset.

2 Application

2.1 Dimension Reduction

Let's read the dataset first.

[13]:
```
df_wiki = pd.read_csv('wiki.csv')
df_wiki.head()
```

[13]:
```
   age  gender  phd  yearsexp  userwiki  pu1  pu2  pu3  peu1  peu2  …  exp5  \
0   40       0    1        14         0    4    4    3     5     5  …     2
1   42       0    1        18         0    2    3    3     4     4  …     4
2   37       0    1        13         0    2    2    2     4     4  …     3
3   40       0    0        13         0    3    3    4     3     3  …     4
4   51       0    0         8         1    4    3    5     5     4  …     4

   domain_Sciences  domain_Health.Sciences  domain_Engineering_Architecture  \
0                1                       0                                0
1                0                       0                                0
2                0                       0                                1
3                0                       0                                1
4                0                       0                                1

   domain_Law_Politics  uoc_position_Associate  uoc_position_Assistant  \
0                    0                       1                       0
```

```
1                         1                           1                            0
2                         0                           0                            1
3                         0                           0                            1
4                         0                           0                            1

   uoc_position_Lecturer  uoc_position_Instructor  uoc_position_Adjunct
0                      0                        0                     0
1                      0                        0                     0
2                      0                        0                     0
3                      0                        0                     0
4                      0                        0                     0

[5 rows x 57 columns]
```

2.1.1 Perform PCA on the dataset

Let's start by standardizing the data.

[44]:
```
from sklearn.preprocessing import StandardScaler
X_fts = df_wiki.values
scaler = StandardScaler() #Define the scaler
X_fts_scl = scaler.fit_transform(X_fts) # Scale the features
```

Although we can use the `sklearn` methods for the PCA, I prefer to do it ”by hand” calulating the Singular Value Decomposition (SVD) of the matrix X.

The SVD of X is given by:

$$X = U\Sigma V^T \tag{1}$$

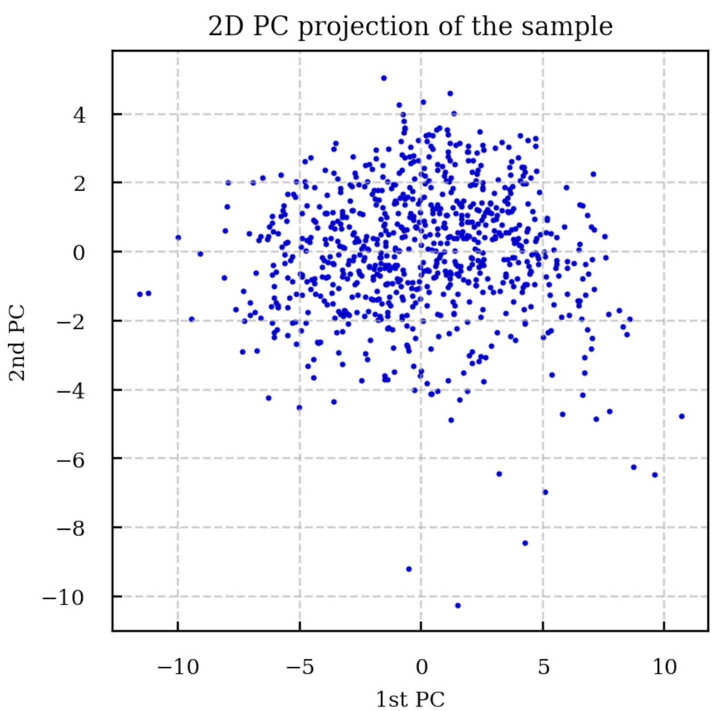
The columns of US are the principal components of X, while Σ is a diagonal matrix. Singular values are related to the eigenvalues of covariance matrix via $\lambda_i = s_i^2/(n-1)$. Eigenvalues λ_i show variances of the respective PCs.

Columns columns of V are principal directions/axes.

[102]:
```
from scipy import linalg
n_len = np.shape(X_fts_scl)[0]

U, Sig, V_h = linalg.svd(X_fts_scl,full_matrices=False)

#PC_s = np.matmul(U,Sig)
V = V_h.T # The columns of V are the principal directions

V_1 = V[:,0] # First principal component
V_2 = V[:,1] # Second princial component
```

```
coords_PCs = np.zeros([n_len,2]) # This matrix will store the coordinates in the␣
↪reduced 2-d space


for i in range(n_len):
    vect = X_fts_scl[i,:]

    # Coord in first p.c.
    coords_PCs[i,0] = np.dot(vect,V_1)
    # Coord in second p.c.
    coords_PCs[i,1] = np.dot(vect, V_2)
```

[103]:
```
plt.figure(figsize = (4.0,4.0))

plt.scatter(coords_PCs[:,0],coords_PCs[:,1], s=1.2, color='mediumblue')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st PC');plt.ylabel('2nd PC')
plt.title('2D PC projection of the sample')

plt.show()
```



The loading vectors are the columns of U.

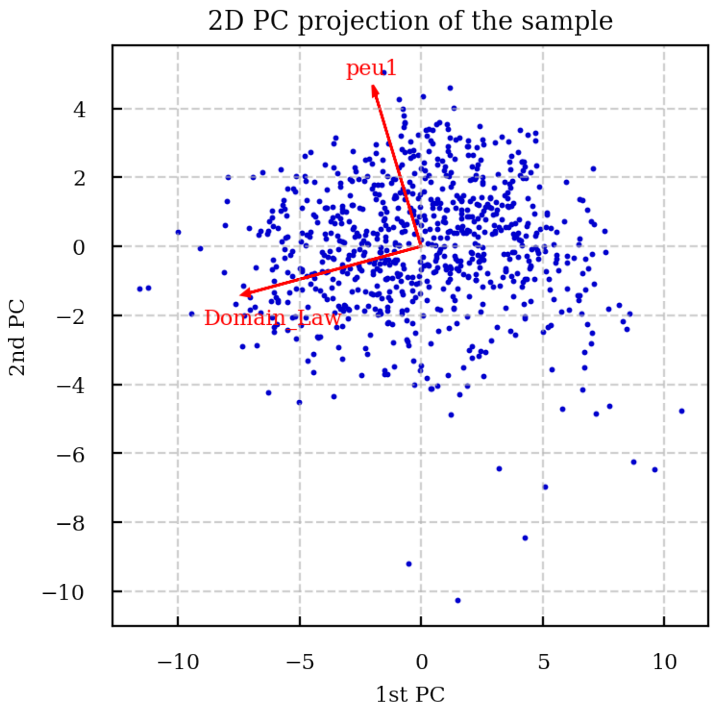
[105]:
```
print(np.shape(U))
```

```
(800, 57)
```

[106]:
```
load_1 = U[:,0] #For projections on the first PC
load_2 = U[:,1] #For projections on the second PC

coords_loads = np.zeros([57,2]) # This matrix will store the coordinates of the␣
↪projections
# in the 2D space

for i in range(57):
    vect = X_fts_scl[:,i]

    # Coord in first p.c.
    coords_loads[i,0] = np.dot(vect,load_1)
    # Coord in second p.c.
    coords_loads[i,1] = np.dot(vect, load_2)
```

[168]:
```
labels = df_wiki.columns
# Get the indices of sorted loads
sort_1 = np.argsort(abs(coords_loads[:,0]))
sort_2 = np.argsort(abs(coords_loads[:,1]))
```

[181]:
```
plt.figure(figsize = (4.0,4.0))

plt.scatter(coords_PCs[:,0],coords_PCs[:,1], s=1.2, color='mediumblue')
plt.arrow(0,0,0.3*coords_loads[sort_1[-1],0],0.
↪3*coords_loads[sort_1[-1],1],color='red',head_width=0.2)
plt.arrow(0,0,0.3*coords_loads[sort_2[-1],0],0.
↪3*coords_loads[sort_2[-1],1],color='red',head_width=0.2)
plt.text(0.38*coords_loads[sort_1[-1],0],0.5*coords_loads[sort_1[-1],1],␣
↪'Domain_Law',color='red')
plt.text(0.5*coords_loads[sort_2[-1],0],0.34*coords_loads[sort_2[-1],1],␣
↪'peu1',color='red')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st PC');plt.ylabel('2nd PC')
plt.title('2D PC projection of the sample')
#plt.legend(loc='lower right', fontsize=10)
plt.show()
```



[182]:
```
print(labels[sort_1])
print(labels[sort_2])
```

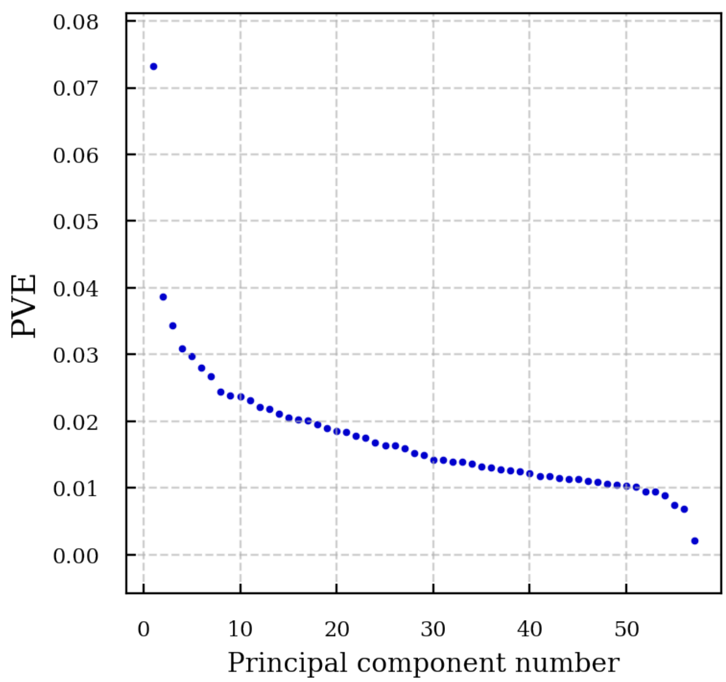
```
Index([u'uoc_position_Instructor', u'uoc_position_Assistant',
       u'uoc_position_Adjunct', u'uoc_position_Associate',
       u'domain_Health.Sciences', u'uoc_position_Lecturer', u'age',
       u'domain_Sciences', u'phd', u'yearsexp', u'gender',
       u'domain_Engineering_Architecture', u'qu4', u'peu1', u'jr2', u'im2',
       u'jr1', u'userwiki', u'inc3', u'inc4', u'domain_Law_Politics', u'inc2',
       u'exp4', u'peu3', u'pf1', u'pf2', u'inc1', u'pf3', u'exp5', u'peu2',
       u'vis2', u'sa2', u'sa3', u'sa1', u'enj2', u'exp3', u'enj1', u'use2',
       u'qu3', u'im1', u'im3', u'qu2', u'vis1', u'vis3', u'qu1', u'use1',
       u'qu5', u'pu2', u'pu1', u'exp2', u'use5', u'exp1', u'pu3', u'use4',
       u'use3', u'bi1', u'bi2'],
      dtype='object')
Index([u'uoc_position_Assistant', u'uoc_position_Instructor',
       u'uoc_position_Adjunct', u'pu1', u'qu5', u'uoc_position_Associate',
       u'domain_Sciences', u'domain_Law_Politics', u'domain_Health.Sciences',
       u'pu2', u'pf2', u'uoc_position_Lecturer', u'vis1', u'pu3', u'exp2',
       u'use5', u'phd', u'qu3', u'qu1', u'im3', u'vis2', u'bi1', u'im2',
       u'yearsexp', u'qu2', u'peu3', u'exp1', u'exp5', u'bi2', u'age', u'pf3',
       u'qu4', u'jr2', u'im1', u'pf1', u'exp3', u'userwiki', u'jr1', u'gender',
       u'enj1', u'use3', u'use4', u'domain_Engineering_Architecture', u'vis3',
       u'use1', u'inc2', u'inc4', u'use2', u'inc3', u'peu2', u'sa2', u'enj2',
```

```
 u'exp4', u'sa1', u'sa3', u'inc1', u'peu1'],
dtype='object')
```

2.1.2 PVE and cumulative PVE

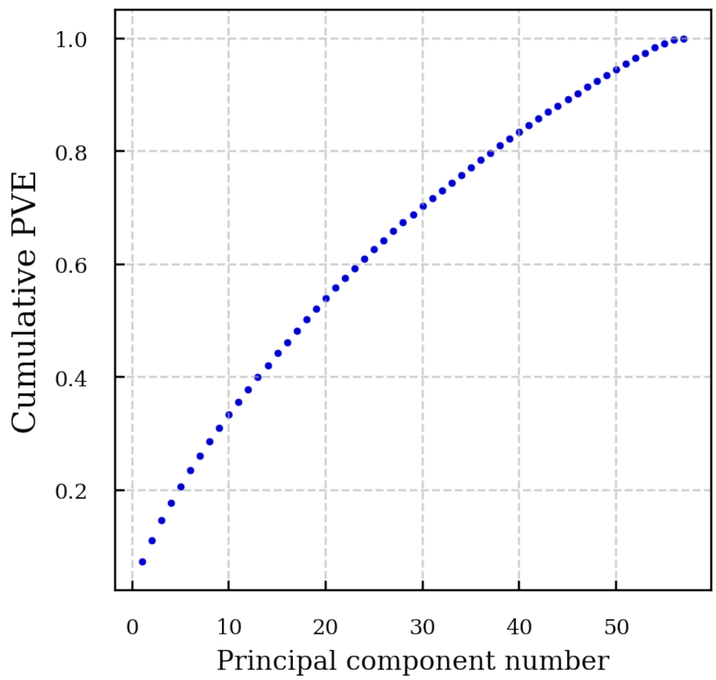
First I'll calculate the PVA. According to what we said, PVE is $\sigma_i^2/\sum_i \sigma_i^2$, where σ^2 the entries of matrix Σ.

[196]:
```
pve = Sig/np.sum(Sig)
num_com = np.arange(1,58)



plt.figure(figsize = (4.0,4.0))
plt.grid(ls='--', alpha=0.6)
plt.scatter(num_com,pve,s=3,color='mediumblue')
plt.xlabel('Principal component number',fontsize=10)
plt.ylabel('PVE',fontsize=12)
plt.show()
```



[197]:
```
cum_pve = np.cumsum(Sig)/np.sum(Sig)
plt.figure(figsize = (4.0,4.0))
plt.scatter(num_com,cum_pve,s=3,color='mediumblue')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('Principal component number',fontsize=10)
```

```
plt.ylabel('Cumulative PVE',fontsize=12)
plt.show()
```



[191]:
```
print(pve[0]+pve[1])
```

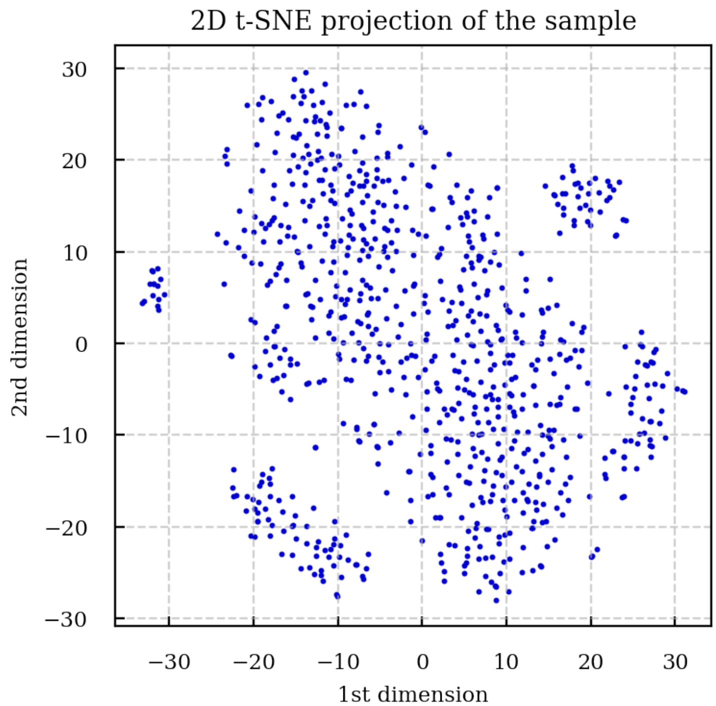
```
0.11190230444466974
```

About 11% of the variance is explained by the first two principal components.

2.1.3 t-SNE

[200]:
```
from sklearn.manifold import TSNE
X_tsne_2 = TSNE(n_components=2).fit_transform(X_fts_scl)
```

[201]:
```
plt.figure(figsize = (4.0,4.0))

plt.scatter(X_tsne_2[:,0],X_tsne_2[:,1], s=1.2, color='mediumblue')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st dimension');plt.ylabel('2nd dimension')
plt.title('2D t-SNE projection of the sample')

plt.show()
```



2.2 Clustering

2.2.1 k-means and PCA

Perform k-means with $k = 2, 3, 4$ and then plot observations on the first and second PCs color-coded based on the cluster membership.

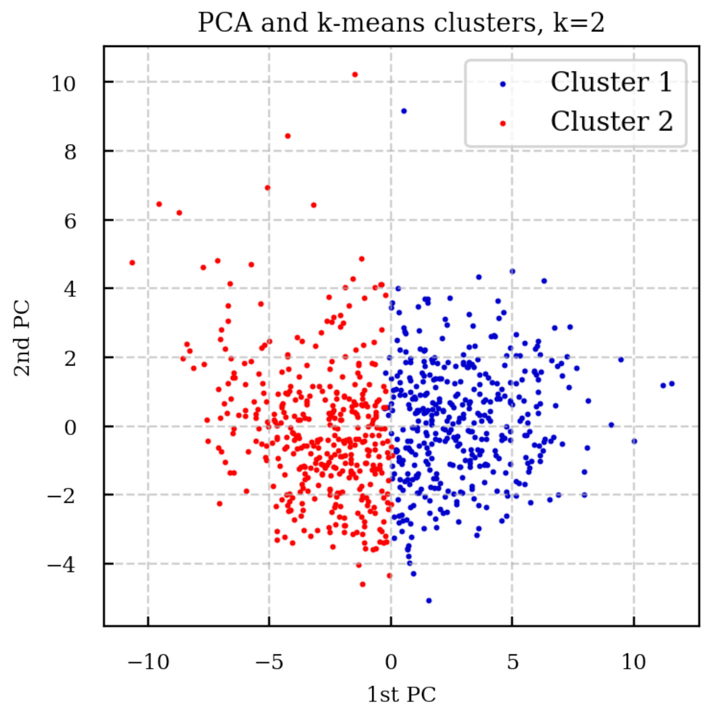
k=2

[202]:
```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

[205]:
```
kmeans_2 = KMeans(n_clusters=2, random_state=0).fit(X_fts_scl) #Fit k-means
# Find cluster memeber labels
km_labels_2  = kmeans_2.labels_
# Get the first two PCs
X_PCA_2 = PCA(n_components=2).fit_transform(X_fts_scl)
```

[209]:
```
ft_1 = X_PCA_2[:,0]
ft_2 = X_PCA_2[:,1]
plt.figure(figsize = (4.0,4.0))
```

```
plt.scatter(ft_1[km_labels_2==0],ft_2[km_labels_2==0], s=1.2,␣
 ↪color='mediumblue', label='Cluster 1')
plt.scatter(ft_1[km_labels_2==1],ft_2[km_labels_2==1], s=1.2, color='red',␣
 ↪label='Cluster 2')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st PC');plt.ylabel('2nd PC')
plt.title('PCA and k-means clusters, k=2')

plt.legend(frameon=True, loc='upper right', fontsize=10)
plt.show()
```

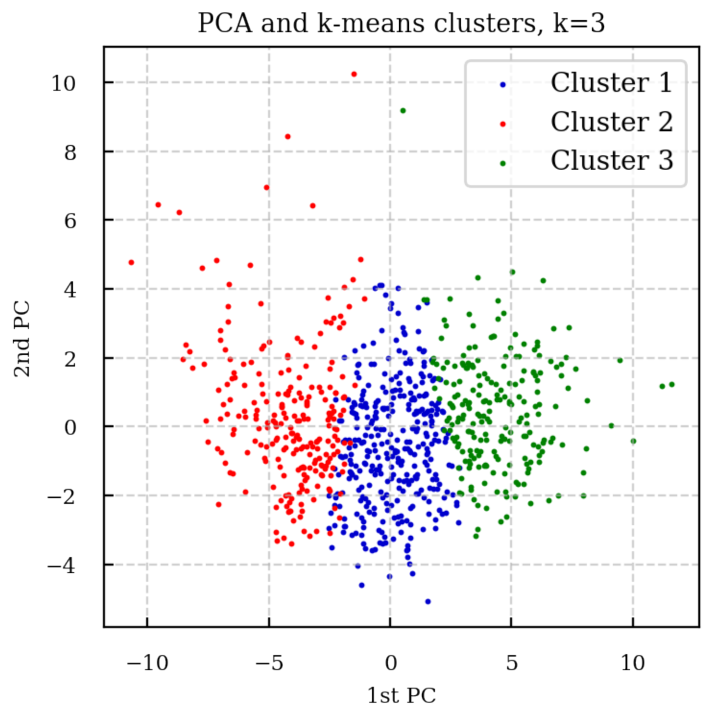


k=3

[210]:
```
kmeans_3 = KMeans(n_clusters=3, random_state=0).fit(X_fts_scl) #Fit k-means
# Find cluster memeber labels
km_labels_3  = kmeans_3.labels_
# Get the first two PCs
X_PCA_3 = PCA(n_components=2).fit_transform(X_fts_scl)
```

[213]:
```
ft_1 = X_PCA_3[:,0]
ft_2 = X_PCA_3[:,1]
plt.figure(figsize = (4.0,4.0))
```

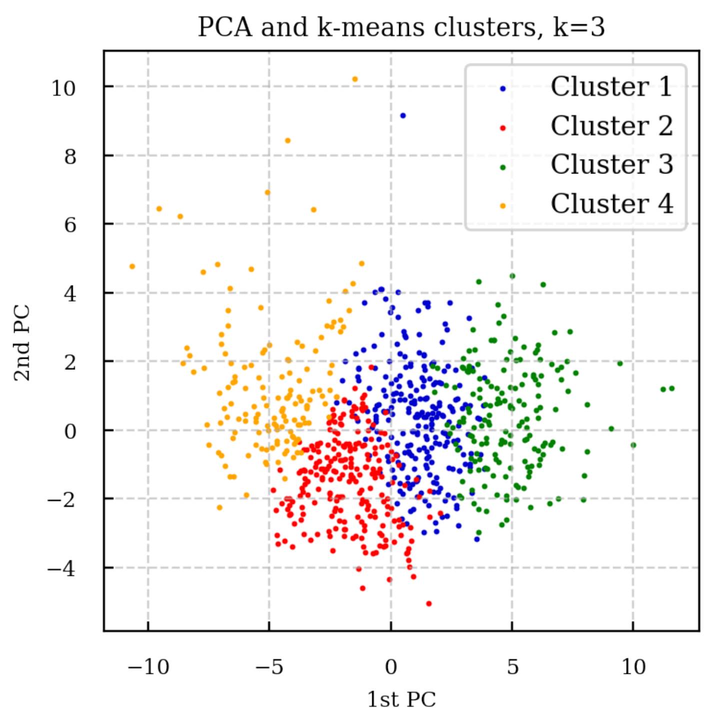
```
plt.scatter(ft_1[km_labels_3==0],ft_2[km_labels_3==0], s=1.2,␣
 ↪color='mediumblue', label='Cluster 1')
plt.scatter(ft_1[km_labels_3==1],ft_2[km_labels_3==1], s=1.2, color='red',␣
 ↪label='Cluster 2')
plt.scatter(ft_1[km_labels_3==2],ft_2[km_labels_3==2], s=1.2, color='green',␣
 ↪label='Cluster 3')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st PC');plt.ylabel('2nd PC')
plt.title('PCA and k-means clusters, k=3')

plt.legend(frameon=True, loc='upper right', fontsize=10)
plt.show()
```



k=4

[214]:
```
kmeans_4 = KMeans(n_clusters=4, random_state=0).fit(X_fts_scl) #Fit k-means
# Find cluster memeber labels
km_labels_4  = kmeans_4.labels_
# Get the first two PCs
X_PCA_4 = PCA(n_components=2).fit_transform(X_fts_scl)
```

[216]:
```
ft_1 = X_PCA_4[:,0]
ft_2 = X_PCA_4[:,1]
plt.figure(figsize = (4.0,4.0))

plt.scatter(ft_1[km_labels_4==0],ft_2[km_labels_4==0], s=1.2,␣
 ↪color='mediumblue', label='Cluster 1')
plt.scatter(ft_1[km_labels_4==1],ft_2[km_labels_4==1], s=1.2, color='red',␣
 ↪label='Cluster 2')
plt.scatter(ft_1[km_labels_4==2],ft_2[km_labels_4==2], s=1.2, color='green',␣
 ↪label='Cluster 3')
plt.scatter(ft_1[km_labels_4==3],ft_2[km_labels_4==3], s=1.2, color='orange',␣
 ↪label='Cluster 4')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st PC');plt.ylabel('2nd PC')
plt.title('PCA and k-means clusters, k=3')

plt.legend(frameon=True, loc='upper right', fontsize=10)
plt.show()
```

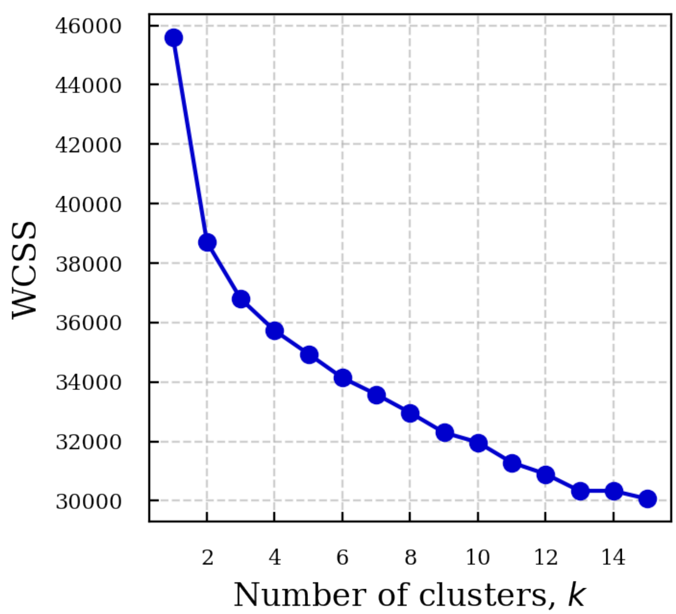


We see that, at least visually, the assignment of clusters seems arbitrary; I.e. the cluster do not seem to correspond to what a human would define as 2/3/4 separate clusters. It is just a slicing of the dataset.

2.2.2 Elbow method

[243]:
```
from scipy.spatial.distance import cdist

# k means determine k
WCSS = [] # Array to put within cluster sum of squares
K = range(1,16)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X_fts_scl)
    WCSS_loc = kmeanModel.inertia_
    WCSS.append(WCSS_loc)
```

[244]:
```
plt.figure(figsize = (3.5,3.5))

plt.plot(K,WCSS, marker='o', color='mediumblue')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('Number of clusters, $k$',fontsize=12)
plt.ylabel('WCSS',fontsize=12)
plt.show()
```



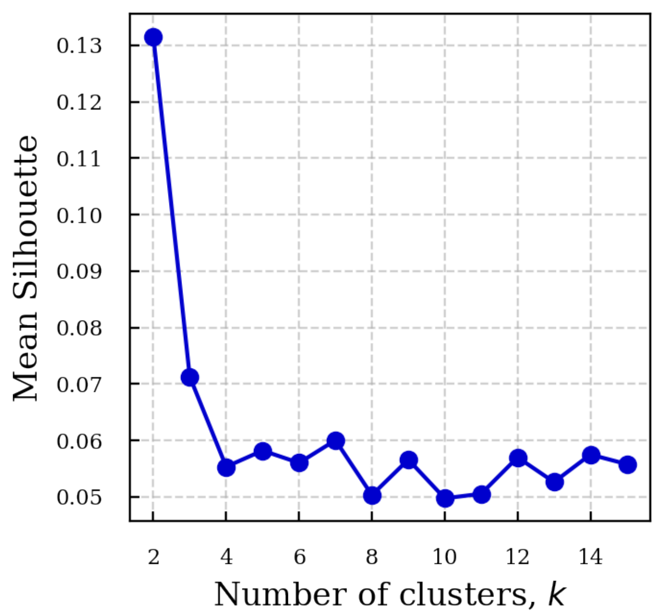
There is no clear ”elbow” in the above plot. Maybe $k = 2$, but not conclusive. Someone may say that $k = 8$ can also be the elbow.

2.2.3 Average Silhouette

[245]:
```
from sklearn.metrics import silhouette_score
```

[246]:
```
# k means determine k
aver_silh = [] # average silhouette scores
K = range(2,16)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X_fts_scl)
    labels_km = kmeanModel.labels_
    silh_score = silhouette_score(X_fts_scl,labels_km)
    aver_silh.append(silh_score)
```

[247]:
```
plt.figure(figsize = (3.5,3.5))

plt.plot(K,aver_silh, marker='o', color='mediumblue')
plt.grid(ls='--', alpha=0.6)
plt.ylabel('Mean Silhouette',fontsize=12)
plt.xlabel('Number of clusters, $k$',fontsize=12)
plt.show()
```



This gives $k = 2$ as the best number of clusters.

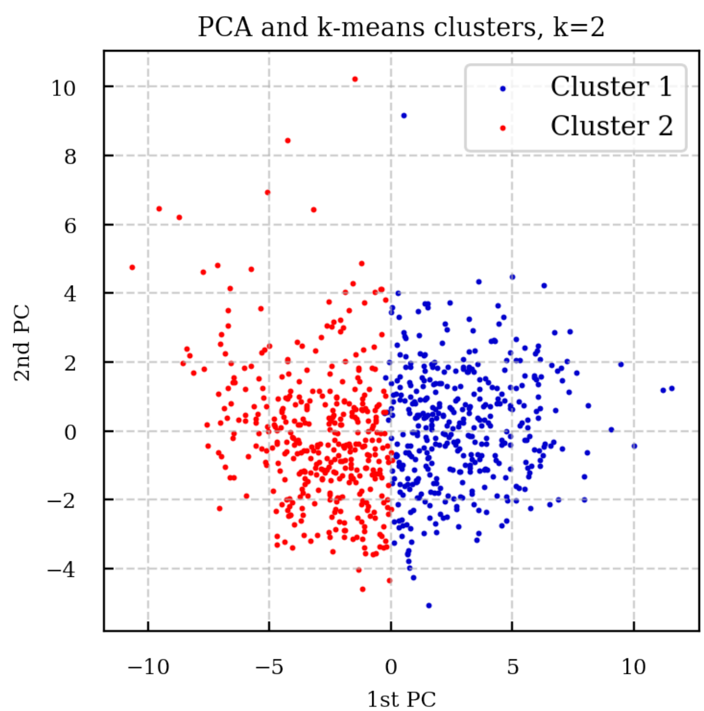
2.2.4 Optimal clustering.

For $k = 2$.

Let's try first with PCA (same as above)

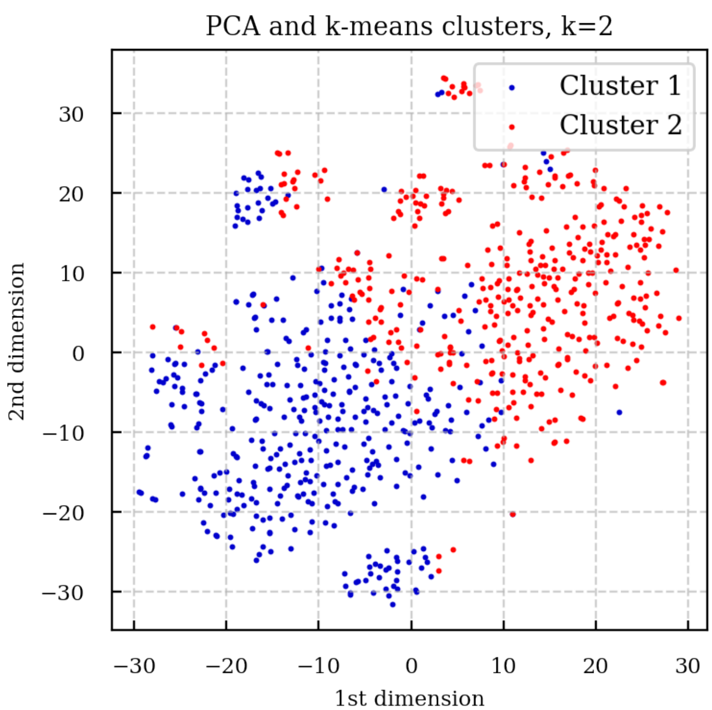
[248]:
```
kmeans_2 = KMeans(n_clusters=2, random_state=0).fit(X_fts_scl) #Fit k-means
# Find cluster memeber labels
km_labels_2  = kmeans_2.labels_
# Get the first two PCs
X_PCA_2 = PCA(n_components=2).fit_transform(X_fts_scl)
```

[249]:
```
ft_1 = X_PCA_2[:,0]
ft_2 = X_PCA_2[:,1]
plt.figure(figsize = (4.0,4.0))

plt.scatter(ft_1[km_labels_2==0],ft_2[km_labels_2==0], s=1.2,␣
↪color='mediumblue', label='Cluster 1')
plt.scatter(ft_1[km_labels_2==1],ft_2[km_labels_2==1], s=1.2, color='red',␣
↪label='Cluster 2')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st PC');plt.ylabel('2nd PC')
plt.title('PCA and k-means clusters, k=2')

plt.legend(frameon=True, loc='upper right', fontsize=10)
plt.show()
```



And now t-SNE

[252]:
```
X_tsne_2 = TSNE(n_components=2).fit_transform(X_fts_scl)
ft_1 = X_tsne_2[:,0]
ft_2 = X_tsne_2[:,1]
plt.figure(figsize = (4.0,4.0))

plt.scatter(ft_1[km_labels_2==0],ft_2[km_labels_2==0], s=1.2,␣
 ↪color='mediumblue', label='Cluster 1')
plt.scatter(ft_1[km_labels_2==1],ft_2[km_labels_2==1], s=1.2, color='red',␣
 ↪label='Cluster 2')

plt.grid(ls='--', alpha=0.6)
plt.xlabel('1st dimension');plt.ylabel('2nd dimension')
plt.title('PCA and k-means clusters, k=2')

plt.legend(frameon=True, loc='upper right', fontsize=10)
plt.show()
```



We see that in the t-SNE case the two clusters are better separated compared to the PCA case one.