Axis-Parallel Subspace Clustering (20 points)

In [2]:

```
from time import time
import numpy as np
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
import pandas as pd
```

In [109]:

```
n_classes = 3
data = pd.read_csv("iris_csv.csv")
```

In [95]:

```
from collections import Counter

def most_frequent(List):
    b = Counter(List)
    return b.most_common(1)
```

a. (6 points)

Use the k-means algorithm with k = 3 to cluster the Iris dataset, over the whole 4- dimensional input space. How well do the clusters match the actual labels?

In [160]:

```
a_data = data.copy()

classes = a_data['class'].copy()

X_train = a_data.drop('class', axis=1)


kmeans = KMeans(init='k-means++', n_clusters=n_classes, n_init=10)
kmeans.fit(X_train)

y_pred = kmeans.fit_predict(X_train)

label1 = most_frequent(y_pred[:50])
label2 = most_frequent(y_pred[50:100])
label3 = most_frequent(y_pred[100:150])

for i in range(classes.size):
    if classes[i] == "Iris-setosa":
        classes[i] = label1[0][0]
    if classes[i] == "Iris-versicolor":
        classes[i] = label2[0][0]
    if classes[i] == "Iris-virginica":
        classes[i] = label3[0][0]

errors = 0
for i in range(y_pred.size):
    if y_pred[i] != classes[i]:
        errors = errors + 1

print('Accuracy is:', ((y_pred.size - errors)/y_pred.size)*100, '%')
```

```
Accuracy is: 89.33333333333333 %
```

The clusters are clustered around 89% correct. Because K-Means is not a classifier but a cluster algorithm, the cluster labels that k-means returns are in a quite random order so we check witch label is most common for each part of the dataset, representing each class of the flower. We then use the most common label as the correct label for the type of flower. In this exersice we can trust this gives the right accuarcy.

β. (4 points)

Project the dataset axis-parallel onto the dimensions Sepal.Length and Sepal.Width (hint: probably the easiest way of doing this is by simply throwing away the other columns). Then, run the k-means algorithm with k = 3 again on the projected dataset. Do the results improve? Do they deteriorate?

In [161]:

```
b_data = data.copy()
B_train = b_data.drop(['petallength','petalwidth','class'], axis=1)

classes = b_data['class'].copy()

b_n_classes = 3

kmeans = KMeans(n_clusters=b_n_classes, n_init=10)
kmeans.fit(B_train)

y_pred = kmeans.fit_predict(B_train)

label1 = most_frequent(y_pred[:50])
label2 = most_frequent(y_pred[50:100])
label3 = most_frequent(y_pred[100:150])

for i in range(classes.size):
    if classes[i] == "Iris-setosa":
        classes[i] = label1[0][0]
    if classes[i] == "Iris-versicolor":
        classes[i] = label2[0][0]
    if classes[i] == "Iris-virginica":
        classes[i] = label3[0][0]

errors = 0
for i in range(y_pred.size):
    if y_pred[i] != classes[i]:
        errors = errors + 1

print('Accuracy is:', ((y_pred.size - errors)/y_pred.size)*100, '%')
```

```
Accuracy is: 82.0 %
```

Here we get an accuracy of 82% which is worse than before.

γ. (10 points)

Repeat the previous exercise with each possible two-dimensional axis-parallel subspace (i.e.: five more times). In which subspace does the clustering mimic the true labels of the dataset most closely? Which type of Iris most often ends up in a wrong cluster? Which records are particularly difficult to cluster?

In [165]:

```
Gamma_data = data.copy()
names = ["sepallength", "sepalwidth", "petallength","petalwidth"]
gamma_n_classes = 3
errorIndex = 0
errorMatrix = np.zeros((150, 8))

for i in range(150):
    errorMatrix[i,7] = i + 1

for i in range(data.shape[1] - 2):
    for j in range(i+1,data.shape[1] - 1):
        classes = Gamma_data['class'].copy()
        temp_data = Gamma_data.copy()
        Gamma_train = temp_data.drop([names[i],names[j],'class'], axis=1)

        kmeans = KMeans(n_clusters=gamma_n_classes, n_init=10)
        kmeans.fit(Gamma_train)

        y_pred = kmeans.fit_predict(Gamma_train)

        label1 = most_frequent(y_pred[:50])
        label2 = most_frequent(y_pred[50:100])
        label3 = most_frequent(y_pred[100:150])
        classesNames = ['', '', '']

        for t in range(classes.size):
            if classes[t] == "Iris-setosa":
                classes[t] = label1[0][0]
                classesNames[label1[0][0]] = "Iris-setosa"
            if classes[t] == "Iris-versicolor":
                classes[t] = label2[0][0]
                classesNames[label2[0][0]] = "Iris-versicolor"
            if classes[t] == "Iris-virginica":
                classes[t] = label3[0][0]
                classesNames[label3[0][0]] = "Iris-virginica"
        errors = [0, 0, 0, 0]



        for r in range(y_pred.size):
            if(r < 50):
                if y_pred[r] != classes[r]:
                    errors[0] = errors[0] + 1
                    errorMatrix[r, errorIndex] = errorMatrix[r, errorIndex] + 1
            if(r >= 50 and r < 100):
                if y_pred[r] != classes[r]:
                    errors[1] = errors[1] + 1
                    errorMatrix[r, errorIndex] = errorMatrix[r, errorIndex] + 1
            if(r >= 100 and r < 150):
                if y_pred[r] != classes[r]:
                    errors[2] = errors[2] + 1
                    errorMatrix[r, errorIndex] = errorMatrix[r, errorIndex] + 1

        errors[3] = errors[0] + errors[1] + errors[2]
        errorIndex = errorIndex + 1

        print('Accuracy for dropped', names[i], 'and', names[j], 'is:', ((y_pred
.size - errors[3])/y_pred.size)*100, '%')
        print('Errors for', classesNames[0], 'is:', errors[0])
        print('Errors for', classesNames[1], 'is:', errors[1])
        print('Errors for', classesNames[2], 'is:', errors[2])
        print('Total errors are:', errors[3])
        print('')

for i in range(150):
    errorMatrix[i, 6] =  errorMatrix[i, 0] + errorMatrix[i, 1] + errorMatrix[i, 
2] + errorMatrix[i, 3] + errorMatrix[i, 4] + errorMatrix[i, 5]

for i in errorMatrix:
    if i[6] >= 4:
        print(i, 'Flower number:', int(i[7]), 'was wrongly clustered', int(i[6
]), 'times.')
```

```
Accuracy for dropped sepallength and sepalwidth is: 96.0 %
Errors for Iris-virginica is: 0
Errors for Iris-setosa is: 2
Errors for Iris-versicolor is: 4
Total errors are: 6

Accuracy for dropped sepallength and petallength is: 92.666666666666
66 %
Errors for Iris-versicolor is: 1
Errors for Iris-setosa is: 4
Errors for Iris-virginica is: 6
Total errors are: 11

Accuracy for dropped sepallength and petalwidth is: 92.6666666666666
6 %
Errors for Iris-virginica is: 0
Errors for Iris-setosa is: 2
Errors for Iris-versicolor is: 9
Total errors are: 11

Accuracy for dropped sepalwidth and petallength is: 82.6666666666666
7 %
Errors for Iris-setosa is: 0
Errors for Iris-versicolor is: 11
Errors for Iris-virginica is: 15
Total errors are: 26

Accuracy for dropped sepalwidth and petalwidth is: 88.0 %
Errors for Iris-versicolor is: 0
Errors for Iris-setosa is: 5
Errors for Iris-virginica is: 13
Total errors are: 18

Accuracy for dropped petallength and petalwidth is: 82.0 %
Errors for Iris-virginica is: 0
Errors for Iris-setosa is: 12
Errors for Iris-versicolor is: 15
Total errors are: 27

[ 1.  1.  1.  1.  1.  1.  6. 78.] Flower number: 78 was wrongly clus
tered 6 times.
[  1.   1.   1.   1.   1.   1.   6. 107.] Flower number: 107 was wro
ngly clustered 6 times.
[  0.   0.   1.   1.   1.   1.   4. 114.] Flower number: 114 was wro
ngly clustered 4 times.
[  1.   1.   1.   1.   1.   1.   6. 120.] Flower number: 120 was wro
ngly clustered 6 times.
[  0.   0.   1.   1.   1.   1.   4. 122.] Flower number: 122 was wro
ngly clustered 4 times.
[  0.   0.   1.   1.   1.   1.   4. 124.] Flower number: 124 was wro
ngly clustered 4 times.
[  1.   0.   1.   1.   1.   1.   5. 127.] Flower number: 127 was wro
ngly clustered 5 times.
[  0.   0.   1.   1.   1.   1.   4. 128.] Flower number: 128 was wro
ngly clustered 4 times.
[  1.   0.   1.   1.   1.   1.   5. 139.] Flower number: 139 was wro
ngly clustered 5 times.
```

We get the best case, subspace, when we drop sepallength and sepalwidth, accuracy is: 96.0%. We can see that the most troublesome iris is the Iris-versicolor, it is always most often wrongly classified, in every subspace. We can also see that flowers number 78, 107 and 120 are wrongly classified in every subspace. Above one can also see other flowers that is pretty hard to classify.

```
In [ ]:
```