Exercises for the course
Machine Learning for Data Science
Winter Semester 2022/23

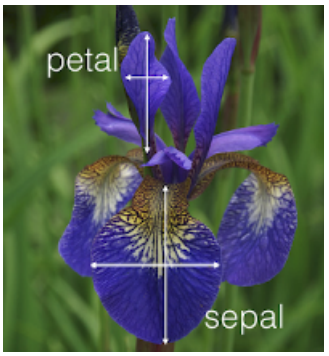
G. Montavon
Institute of Computer Science
Department of Mathematics and Computer Science
Freie Universität Berlin

Exercise Sheet 4 (programming part)

In [1]:
```
import numpy
import scipy
import sklearn
import sklearn.datasets
import sklearn.decomposition
import torchvision
import matplotlib
%matplotlib inline
from matplotlib import pyplot as plt
```

Exercise 3 (10 + 10 P)

In this exercise we perform a clustering of the Iris dataset, a famous dataset from Fisher (1936) which one can access from sklearn. Each instance of the dataset is an iris plant which comes with four measurements (1. sepal length in cm, 2. sepal width in cm, 3. petal length in cm, 4. petal width in cm). What these measurements correspond to is depicted in the image below.



In addition to these measurements, the dataset also includes for each instance the type of iris plant (iris setosa, iris versicolour, iris virginica) which we treat here as metadata. Overall, the Iris dataset has 150 instances, and can be stored as an array of size 150 x 4. The following cell loads the dataset and performs some standardization.

In [2]:
```
dataset = sklearn.datasets.load_iris()

X = dataset['data']

X = X - X.mean(axis=0)
X = X / X.std()
```

We focus on one of the simplest clustering methods, which is to find the *connected components* of a graph associated to the dataset. We consider a graph where two nodes are connected if the distance between the corresponding instances (after dataset normalization) is below some threshold value `delta`.

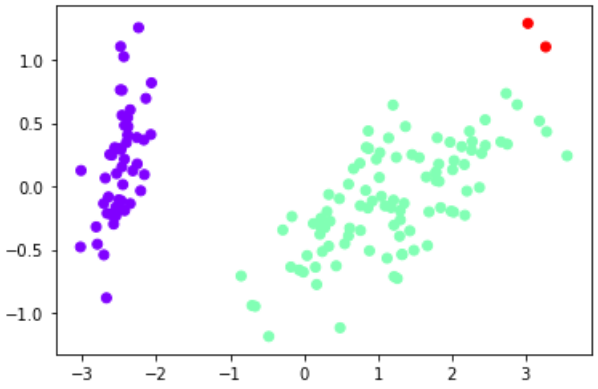
(a) Implement this simple clustering method. *(Hint: you can make use of the method `scipy.sparse.csgraph.connected_components` to find the connected components associated to a particular adjacency matrix.)*

In [3]:
```
class ConnectedComponentsClustering:

    def __init__(self,delta):
        self.delta = delta

    def fit_predict(self,X):

        # ----------------------------------------
        # TODO: Replace by your code
        # ----------------------------------------
        import solution
        n_components, Y = solution.cc_fit_predict(self,X)
        # ----------------------------------------

        return n_components, Y
```

The algorithm is now applied to the Iris dataset with a particular threshold value `delta = 0.725` . Results are visualized in a PCA plot, where instances are color-coded according to their cluster.

In [4]:
```
delta = 0.725

n_components, Y = ConnectedComponentsClustering(delta).fit_predict(X)

Z = sklearn.decomposition.PCA(2).fit_transform(X)
plt.scatter(*Z.T,c=Y,cmap='rainbow')
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x7f0a7449d340>
```



We observe three clusters. Two clusters contain the bulk of the data, and another cluster is very sparsely populated and appears to contain two outliers. Clusters are consistent with what a human can see in the PCA plot.

(b) We now would like to compare the taxonomy derived from the clustering above with the actual iris types available as meta-data. *Write* code that produces a correspondence table linking the clustering to the classification into iris types.

In [5]:
```
T = dataset['target']
names = dataset['target_names']

# ----------------------------------------
# TODO: Replace by your code
# ----------------------------------------
import solution
solution.correspondence(names,T,n_components,Y)
# ----------------------------------------
```

```
            cluster_0  cluster_1  cluster_2
setosa             50          0          0
versicolor          0         50          0
virginica           0         48          2
```

We observe a one-to-one correspondence bewteen cluster 0 and the type iris setosa. The two other iris types (versicolor and virginica) are grouped into a single cluster. A last cluster contain two specific instances of the type virginica. These two instances appear to be somewhat distinct from other instances of the same type.

Exercise 4 (10 + 10 + 10 P)

In this exercise, we consider clustering of fashion items from the FashionMNIST dataset. The FashionMNIST dataset consists of 60000 fashion items, each of which coming as a 28 x 28 grayscale image. For the purpose of limiting computations, we consider a

subset of 1000 instances from this dataset, and thus, extract a dataset of size 1000 x 784 (the 784 dimensions correspond to representing images as a flat vector)

In [6]:
```
X = torchvision.datasets.FashionMNIST('.',download=True).data.numpy()

numpy.random.seed(0)
R = numpy.random.permutation(len(X))[:1000]
X = X[R].reshape(-1,784)

X = X / 255.0
```

The first 50 images of our dataset are visualized below:

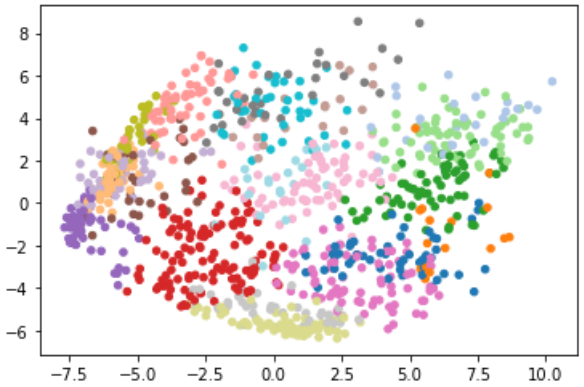
In [7]:
```
plt.figure(figsize=(10,5))
plt.axis('off')
plt.imshow(X[:50].reshape(5,10,28,28).transpose(0,2,1,3).reshape(5*28,10*28),cmap='gray',vmin=
plt.show()
```

(a) Implement a rudimentary version of K-Means, where the cluster centers are initialized to the first few examples of the dataset, and where the number of iterations is fixed to 100.

In [8]:
```
import sklearn.cluster

class KMeans:

    def __init__(self,n_clusters):
        self.n_clusters = n_clusters

    def fit(self,X):

        # ----------------------------------------
        # TODO: Replace by your code
        # ----------------------------------------
        import solution
        solution.kmeans_fit(self,X)
        # ----------------------------------------

    def predict(self,X):

        D = scipy.spatial.distance.cdist(X,self.cluster_centers_)
        return numpy.argmin(D,axis=1)

model = KMeans(20)
model.fit(X)
```

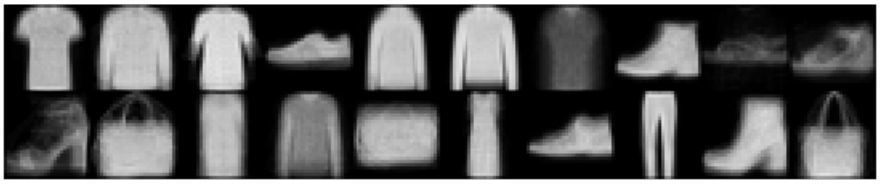
The outcome of your clustering procedure can be visualized in the following PCA plot color-coded by cluster membership.

In [9]:
```
numpy.random.seed(0)
Y = model.predict(X)
Z = sklearn.decomposition.PCA(n_components=2).fit_transform(X)
plt.scatter(*Z.T,c=Y,s=20,cmap='tab20')
plt.show()
```



(b) We now would like to get insights into what are prototypical fashion items in our dataset. For this, we consider the cluster centers learned by k-means. *Extract* these cluster centers and *visualize* them as images.

In [10]:
```
# ----------------------------------------
# TODO: Replace by your code
# ----------------------------------------
import solution
solution.view_cluster_centers(model)
# ----------------------------------------
```



(c) Lastly, we would like to know how well these cluster centers describe our data. Using the formulas presented in the lecture, compute the total variance, the within-cluster variance, the between-cluster variance, and the percentage of explained variance.

In [11]:
```
# ----------------------------------------
# TODO: Replace by your code
# ----------------------------------------
import solution
solution.analyze_variance(X,model)
# ----------------------------------------
```

```
Total variance             68.120
Between-cluster variance   41.579
Within-cluster variance    26.541

Explained variance (%)     61.04%
```

We observe that a bit more than half of the variance in the data is captured by the clustering, which is quite good, considering that the data is high-dimensional. Note that a higher explained variance could be obtained by incorporating more clusters, however that would come at the cost of making the data description potentially too complex for the end user.