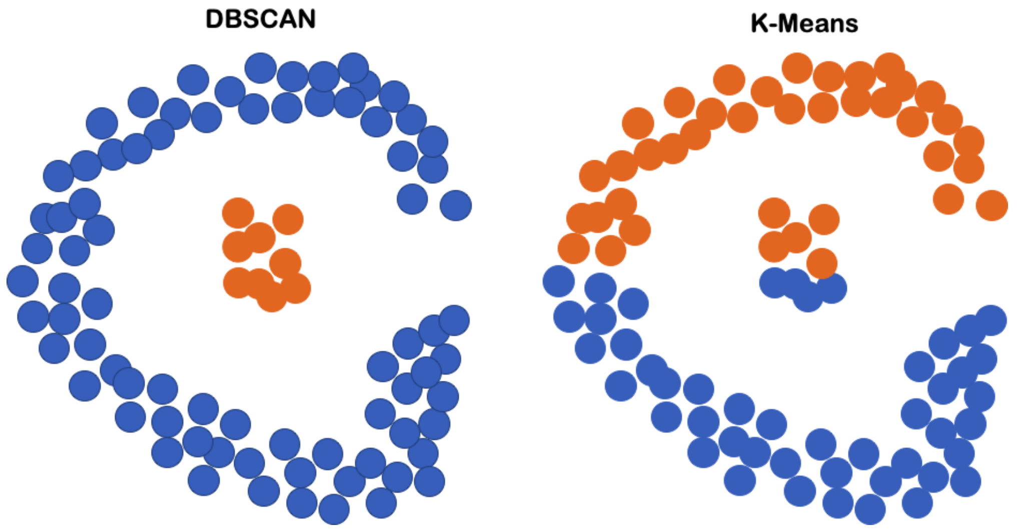
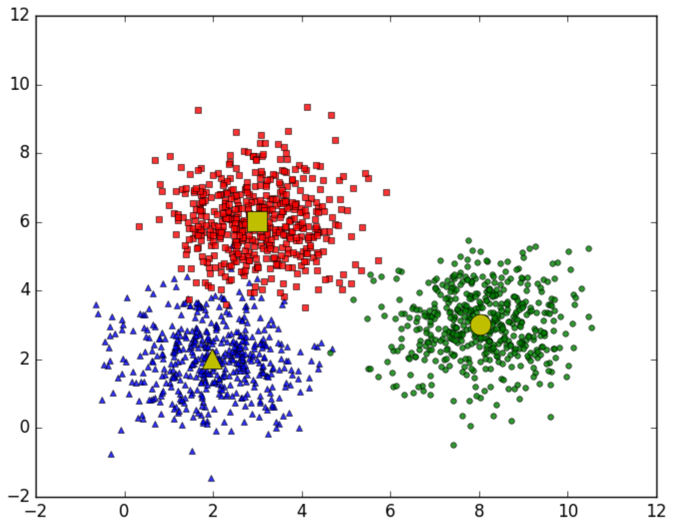
LESSON 7: K-MEANS CLUSTERING

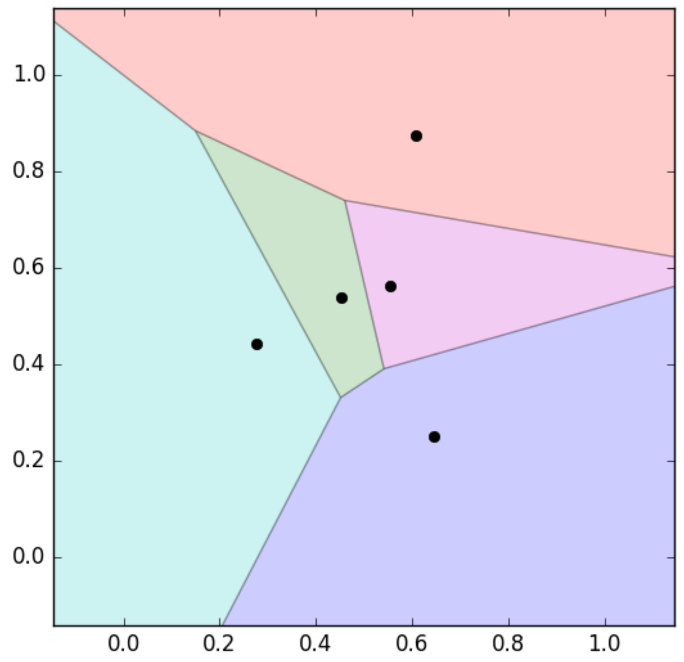


This lecture was refered by machinelearningcoban.com

1. K-means clustering introduction



With the dataset contains lots of data points without label, we have to group these data points into K clusters and we expect that samples in the same cluster will have similar features. The algorithm to solve this problem is called ***K-MEANS CLUSTERING***.



In 2D space, we can see that the region of each cluster is a polygon with a line border. This line border is the mid perpendicular of a line connect 2 centers.

Assume that we have N data point $X = [x_1, x_2, \dots, x_N]$ and K < N is the number of clusters. We have to find $M = [m_1, m_2, \dots, m_K]$ which is the center (or representatives) of K clusters and $Y = [y_1, y_2, \dots, y_N]$ which is the label of N data points.

Label y_i of a sample is encoded into one-hot type, which means $y_i = [y_{i1}, y_{i2}, \dots, y_{iK}]$ and $y_{ij} = 1$ if x_i is predicted to belong to cluster j.

So, we have

$$y_{ik} \in \{0, 1\}$$

$$\sum_{k=1}^{K} y_{ik} = 1$$

2. Loss function and Optimizer for K-means clustering

The main target of clustering technique is to minimize the distance between each data point and its center.

If data point x_i belong to cluster m_k, so $y_{ik} = 1$ and $y_{ij} = 0$ with $\forall j \neq k$

$$D(x_i, m_k) = (x_i - m_k)^2$$
$$D(x_i, m_k) = \sum_{j=1}^{K} y_{ij}(x_i - m_j)^2$$

We have the loss function for the whole dataset

$$\mathcal{L}(Y, M) = MSE(Y, M) = \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij}(x_i - m_j)^2$$

With two variables Y and M, to optimize the loss function, we fix one variable and optimize another and vice versa. Specifically, we solve two problems: Fixed M, optimize Y and Fix Y, optimize M respectively.

Fixed M, optimize Y

For each data point x_i,

$$y_i = arg \min_{y_i} \mathcal{L}(y_i)$$
$$= arg \min_{y_i} \sum_{j=1}^{K} y_{ij}(x_i - m_j)^2$$

We need to find only one y_i for each x_i so this problem is solved by assign the label of each x_i as the **nearest center** of it.

Fixed Y, optimize M

For each existing cluster,

$$
\begin{aligned}
m_j &= arg\min_{m_j} \mathcal{L}(m_j) \\
&= arg\min_{m_j} \sum_{i=1}^{N} y_{ij}(x_i - m_j)^2
\end{aligned}
$$

We need to find only one m_j for each existing cluster

Calculate derivative of $\mathcal{L}(m_j)$ with m_j and solve the derivative function

$$
\frac{\partial \mathcal{L}(m_j)}{\partial m_j} = 2\sum_{i=1}^{N} y_{ij}(m_j - x_i) = 0
$$

$$
m_j \sum_{i=1}^{N} y_{ij} = \sum_{i=1}^{N} y_{ij}x_i
$$

$$
m_j = \frac{\sum_{i=1}^{N} y_{ij}x_i}{\sum_{i=1}^{N} y_{ij}}
$$

$$
m_j = \frac{\text{Sum of all data points in cluster} j}{\text{Number of data points in cluster} j}
$$

That's why we call this algorithm ***k-means clustering***.

3. The algorithm

Input: Dataset contains N samples, K clusters.

Output: N label y for each data sample, K center m for each cluster.

Step 1: Randomly choose K data points as initialized cluster center.

Step 2: Assign label for each data point by nearest center.

Step 3:

- If the results of ***Step 2*** is same as the previous iteration:
    - Stop the algorithm

- Else:
    - Continue the next step.

Step 4: Calculate the new cluster center by the mean of data point in this cluster.

Step 5: Go back to ***Step 2***.

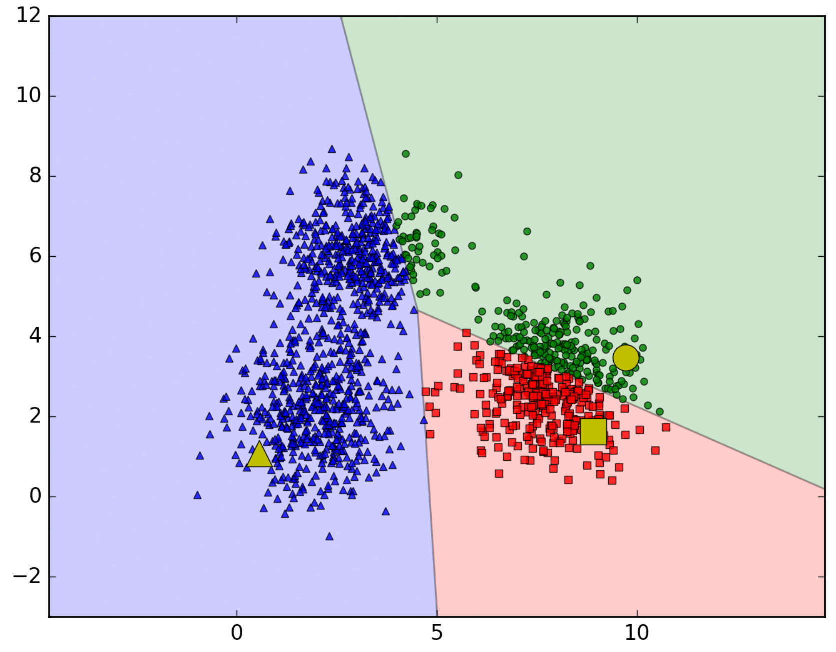
4. Weaknesses of K-means clustering

We need to define number of clusters K

In some cases, we don't know the number of cluster and this is an obstacle while using K-means clustering

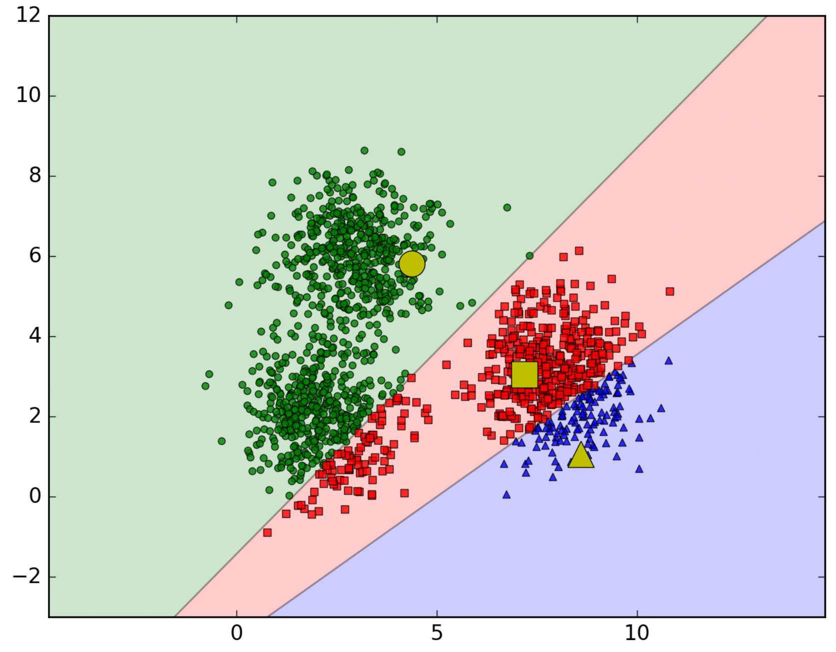
Clustering results is highly depended on initialization

Slow convergence



iteration 0: assign points to clusters

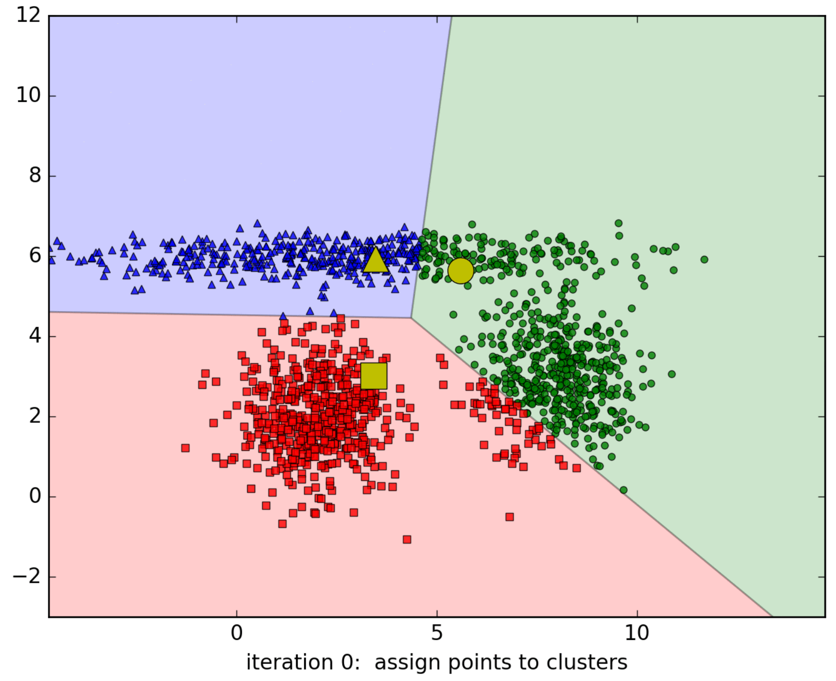
Bad results



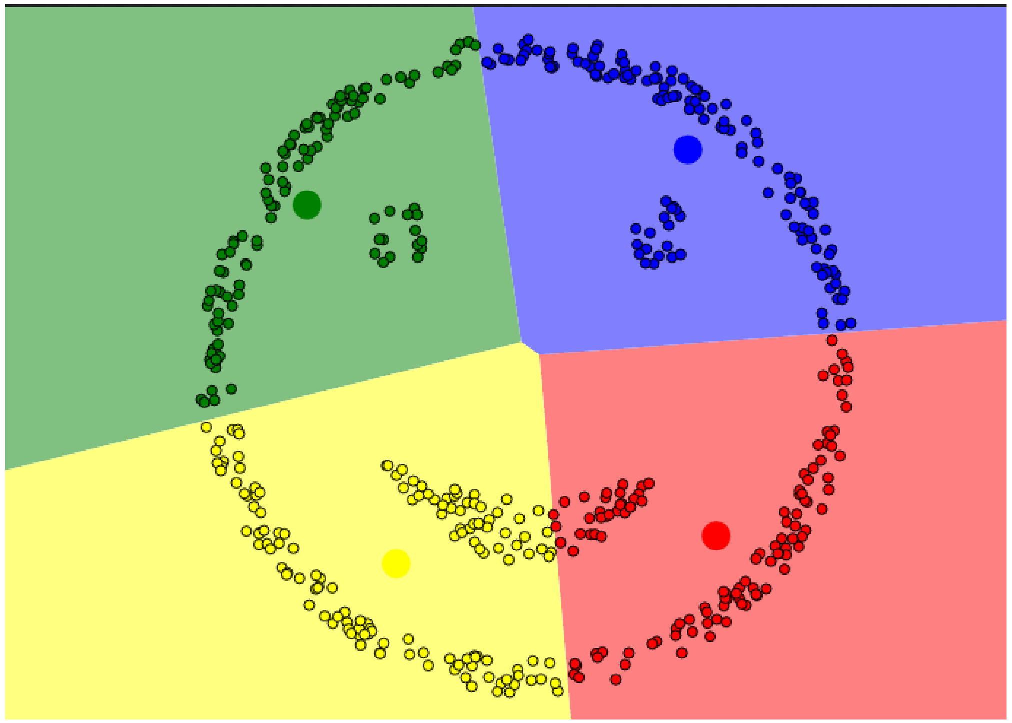
iteration 0: assign points to clusters

We can overcome this problem by running K-means clustering multiple times and choose the best results. Or there are some upgraded versions of K-means clustering like K-means++.

Clusters must be round in shape



K-means clustering doesn't work with non-convex dataset



5. Implementation example

5.1. Prepare library and data

In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

sns.set()
from scipy.spatial.distance import cdist
```

In [2]:
```python
iris_df = sns.load_dataset('iris')
iris_df
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa

	sepal_length	sepal_width	petal_length	petal_width	species
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [3]:
```
df = iris_df.drop(columns=['sepal_length', 'sepal_width'])
df
```

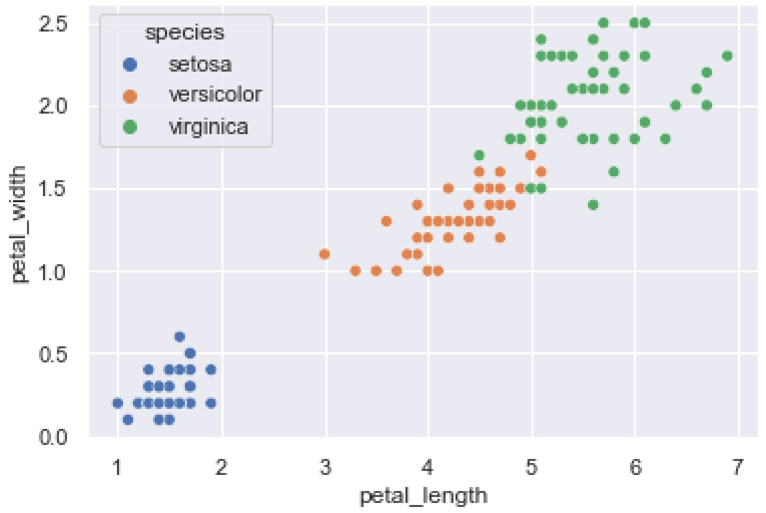
Out[3]:

	petal_length	petal_width	species
0	1.4	0.2	setosa
1	1.4	0.2	setosa
2	1.3	0.2	setosa
3	1.5	0.2	setosa
4	1.4	0.2	setosa
...	...	...	...
145	5.2	2.3	virginica
146	5.0	1.9	virginica
147	5.2	2.0	virginica
148	5.4	2.3	virginica
149	5.1	1.8	virginica

	petal_length	petal_width	species

150 rows × 3 columns

```
In [4]: sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='species')
        plt.show()
```



```
In [5]: X = np.array(df.iloc[:, :2])
        X.shape
```

```
Out[5]: (150, 2)
```

```
In [6]: X
```

```
Out[6]: array([[1.4, 0.2],
               [1.4, 0.2],
               [1.3, 0.2],
               [1.5, 0.2],
               [1.4, 0.2],
               [1.7, 0.4],
               [1.4, 0.3],
               [1.5, 0.2],
               [1.4, 0.2],
               [1.5, 0.1],
               [1.5, 0.2],
       [1.6, 0.2],
       [1.4, 0.1],
       [1.1, 0.1],
       [1.2, 0.2],
       [1.5, 0.4],
       [1.3, 0.4],
       [1.4, 0.3],
       [1.7, 0.3],
       [1.5, 0.3],
       [1.7, 0.2],
       [1.5, 0.4],
       [1. , 0.2],
       [1.7, 0.5],
       [1.9, 0.2],
       [1.6, 0.2],
       [1.6, 0.4],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.6, 0.2],
       [1.6, 0.2],
       [1.5, 0.4],
       [1.5, 0.1],
       [1.4, 0.2],
       [1.5, 0.2],
       [1.2, 0.2],
       [1.3, 0.2],
       [1.4, 0.1],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.3, 0.3],
       [1.3, 0.3],
       [1.3, 0.2],
       [1.6, 0.6],
       [1.9, 0.4],
       [1.4, 0.3],
       [1.6, 0.2],
       [1.4, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [4.7, 1.4],
       [4.5, 1.5],
       [4.9, 1.5],
       [4. , 1.3],
       [4.6, 1.5],
       [4.5, 1.3],
       [4.7, 1.6],
       [3.3, 1. ],
       [4.6, 1.3],
       [3.9, 1.4],
       [3.5, 1. ],
       [4.2, 1.5],
       [4. , 1. ],
       [4.7, 1.4],
       [3.6, 1.3],
       [4.4, 1.4],
       [4.5, 1.5],
       [4.1, 1. ],
       [4.5, 1.5],
       [3.9, 1.1],
       [4.8, 1.8],
       [4. , 1.3],
       [4.9, 1.5],
       [4.7, 1.2],
       [4.3, 1.3],
       [4.4, 1.4],
       [4.8, 1.4],
       [5. , 1.7],
       [4.5, 1.5],
       [3.5, 1. ],
       [3.8, 1.1],
       [3.7, 1. ],
       [3.9, 1.2],
       [5.1, 1.6],
       [4.5, 1.5],
       [4.5, 1.6],
       [4.7, 1.5],
       [4.4, 1.3],
       [4.1, 1.3],
       [4. , 1.3],
       [4.4, 1.2],
       [4.6, 1.4],
       [4. , 1.2],
       [3.3, 1. ],
       [4.2, 1.3],
       [4.2, 1.2],
       [4.2, 1.3],
       [4.3, 1.3],
       [3. , 1.1],
       [4.1, 1.3],
       [6. , 2.5],
       [5.1, 1.9],
       [5.9, 2.1],
       [5.6, 1.8],
       [5.8, 2.2],
       [6.6, 2.1],
       [4.5, 1.7],
       [6.3, 1.8],
       [5.8, 1.8],
       [6.1, 2.5],
       [5.1, 2. ],
       [5.3, 1.9],
       [5.5, 2.1],
       [5. , 2. ],
       [5.1, 2.4],
       [5.3, 2.3],
       [5.5, 1.8],
       [6.7, 2.2],
       [6.9, 2.3],
       [5. , 1.5],
       [5.7, 2.3],
       [4.9, 2. ],
       [6.7, 2. ],
       [4.9, 1.8],
       [5.7, 2.1],
       [6. , 1.8],
       [4.8, 1.8],
       [4.9, 1.8],
       [5.6, 2.1],
       [5.8, 1.6],
       [6.1, 1.9],
       [6.4, 2. ],
       [5.6, 2.2],
       [5.1, 1.5],
       [5.6, 1.4],
       [6.1, 2.3],
       [5.6, 2.4],
       [5.5, 1.8],
       [4.8, 1.8],
       [5.4, 2.1],
       [5.6, 2.4],
       [5.1, 2.3],
       [5.1, 1.9],
       [5.9, 2.3],
       [5.7, 2.5],
       [5.2, 2.3],
       [5. , 1.9],
       [5.2, 2. ],
       [5.4, 2.3],
       [5.1, 1.8]])
```

5.2. Implement from scratch

In [7]:
```
class MyKMeans():

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

    def init_centers(self, X):
        self.centers = X[np.random.choice(
            X.shape[0],
            self.num_clusters,
            replace=False
        )]

    def assign_clusters(self, X):
        # calculate pairwise distances between data points and centers
        D = cdist(X, self.centers)
        # return index of the closest center
        return np.argmin(D, axis=1)

    def update_centers(self, X, labels):
        centers = np.zeros((self.num_clusters, X.shape[1]))
        for k in range(self.num_clusters):
            # collect all points assigned to the k-th cluster
            Xk = X[labels == k, :]
            # take average
            centers[k,:] = np.mean(Xk, axis = 0)
        self.centers = centers

    def has_converged(self, prev_centers):
        # return True if two sets of centers are the same
        return (set([tuple(a) for a in prev_centers]) == \
                set([tuple(a) for a in self.centers]))

    def __call__(self, X):
        self.init_centers(X)
        centers = [self.centers]
        cluster_preds = []

        while True:
            cluster_preds.append(self.assign_clusters(X))
            self.update_centers(X, cluster_preds[-1])
            if self.has_converged(centers[-1]):
                break
            centers.append(self.centers)
```

```python
        return centers, cluster_preds
```

In [8]:
```python
my_kmeans = MyKMeans(num_clusters=3)
```

In [9]:
```python
centers, cluster_preds = my_kmeans(X)
```

In [10]:
```python
centers[-1]
```

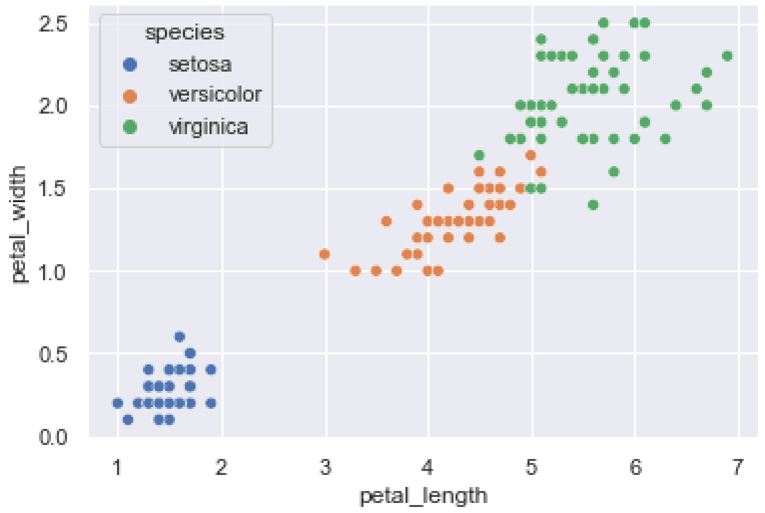
Out[10]:
```
array([[5.62608696, 2.04782609],
       [1.462     , 0.246     ],
       [4.29259259, 1.35925926]])
```

In [11]:
```python
cluster_preds[-1]
```

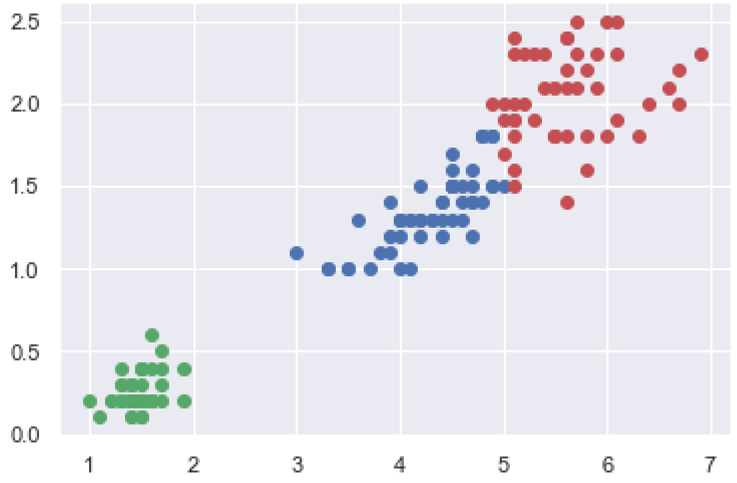
Out[11]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [12]:
```python
def visualize_clusters(X, cluster_preds):
    colors = ['r', 'g', 'b', 'k', 'o']
    for x, cluster in zip(X, cluster_preds):
        plt.plot(x[0], x[1], f'{colors[cluster]}o')
    plt.show()
```

In [13]:
```python
sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='species')
plt.show()
```



In [14]:
```
visualize_clusters(X, cluster_preds[-1])
```



In [15]:
```
my_kmeans = MyKMeans(num_clusters=4)
```

In [16]:
```
centers, cluster_preds = my_kmeans(X)
```

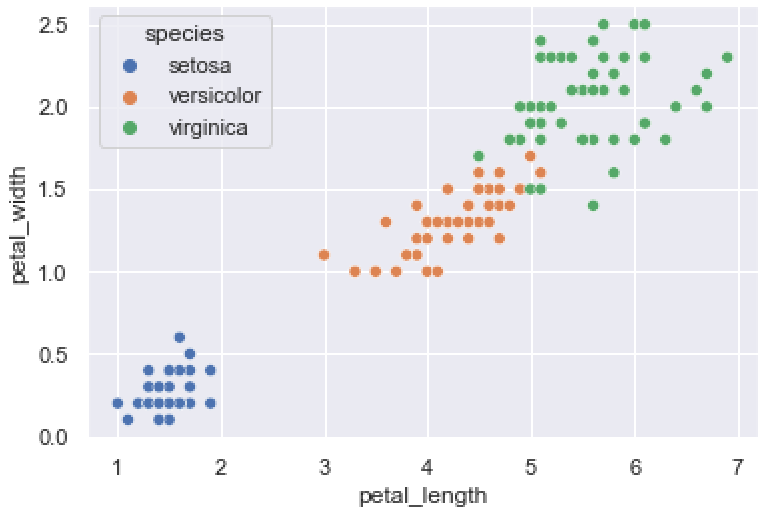
In [17]:
```
centers[-1]
```

```
Out[17]: array([[4.15348837, 1.28837209],
                [5.1       , 1.84705882],
                [6.02608696, 2.14782609],
                [1.462     , 0.246     ]])
```

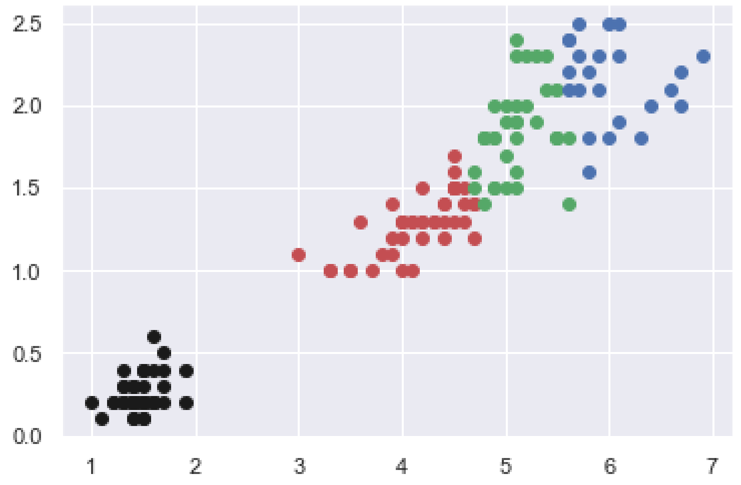
```
In [18]: cluster_preds[-1]
```

```
Out[18]: array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                3, 3, 3, 3, 3, 3, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2, 1, 2, 2, 0, 2, 2, 2,
                1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
                2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1])
```

```
In [19]: sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='species')
         plt.show()
```



```
In [20]: visualize_clusters(X, cluster_preds[-1])
```



5.3. Use `sklearn`

In [21]:
```
from sklearn.cluster import KMeans
```

In [22]:
```
sklearn_kmeans = KMeans(n_clusters=3, random_state=1511)
sklearn_kmeans
```

Out[22]:
```
KMeans(n_clusters=3, random_state=1511)
```

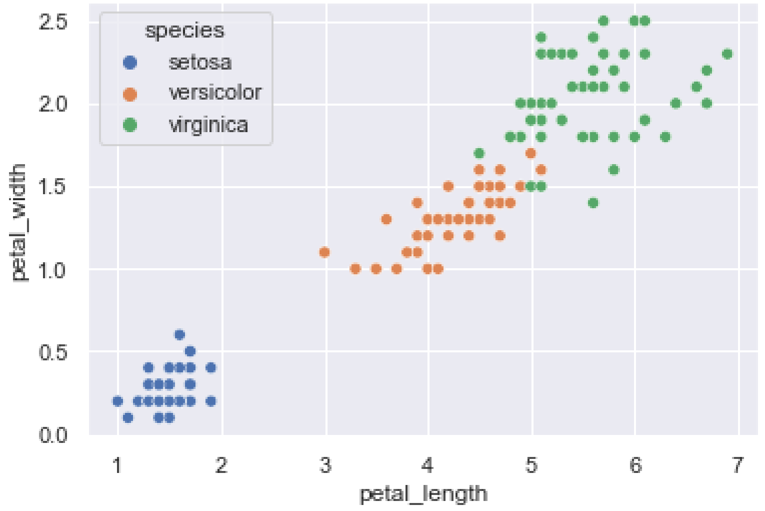
In [23]:
```
sklearn_kmeans.fit(X)
```

Out[23]:
```
KMeans(n_clusters=3, random_state=1511)
```

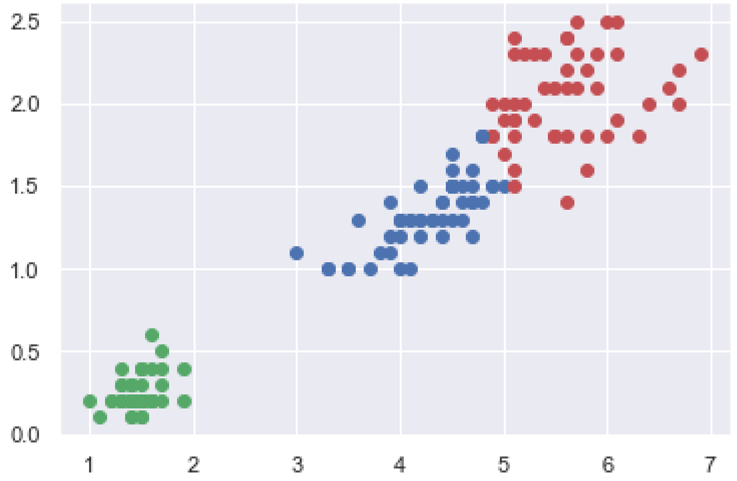
In [24]:
```
sklearn_kmeans.labels_
```

Out[24]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [25]:
```
sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='species')
plt.show()
```



```
In [26]: visualize_clusters(X, sklearn_kmeans.labels_)
```



```
In [27]: sklearn_kmeans.cluster_centers_
```

```
Out[27]: array([[5.59583333, 2.0375    ],
                [1.462     , 0.246     ],
                [4.26923077, 1.34230769]])
```

6. Homework

6.1. Exercise 1:

Load penguins dataset from seaborn, use `bill_length_mm` and `bill_depth_mm` to build a k-means clustering model to cluster `Adelie` species, `Chinstrap` species and `Gentoo` species.

6.2. Exercise 2:

Similar to Exercise 1, but now using `bill_length_mm` , `bill_depth_mm` , `flipper_length_mm` and `body_mass_g` .

In [ ]:

```

```