5. ML project design with unsupervised learning (k-means clustering)

M.A.Z. Chowdhury and M.A. Oehlschlaeger

Department of Mechanical, Aerospace and Nuclear Engineering
Rensselaer Polytechnic Institute
Troy, New York

chowdm@rpi.edu
oehlsm@rpi.edu

"If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake."
- Yann LeCun, Touring Award Laureate, Chief AI Scientist at Meta

MANE 4962 and 6962

End-to-end ML Project Design

1. Frame the problem and look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.

There are other ways to design ML projects but this is a good starting point. Every step is described as a checklist in Appendix B of reference Aurélien Géron's book.

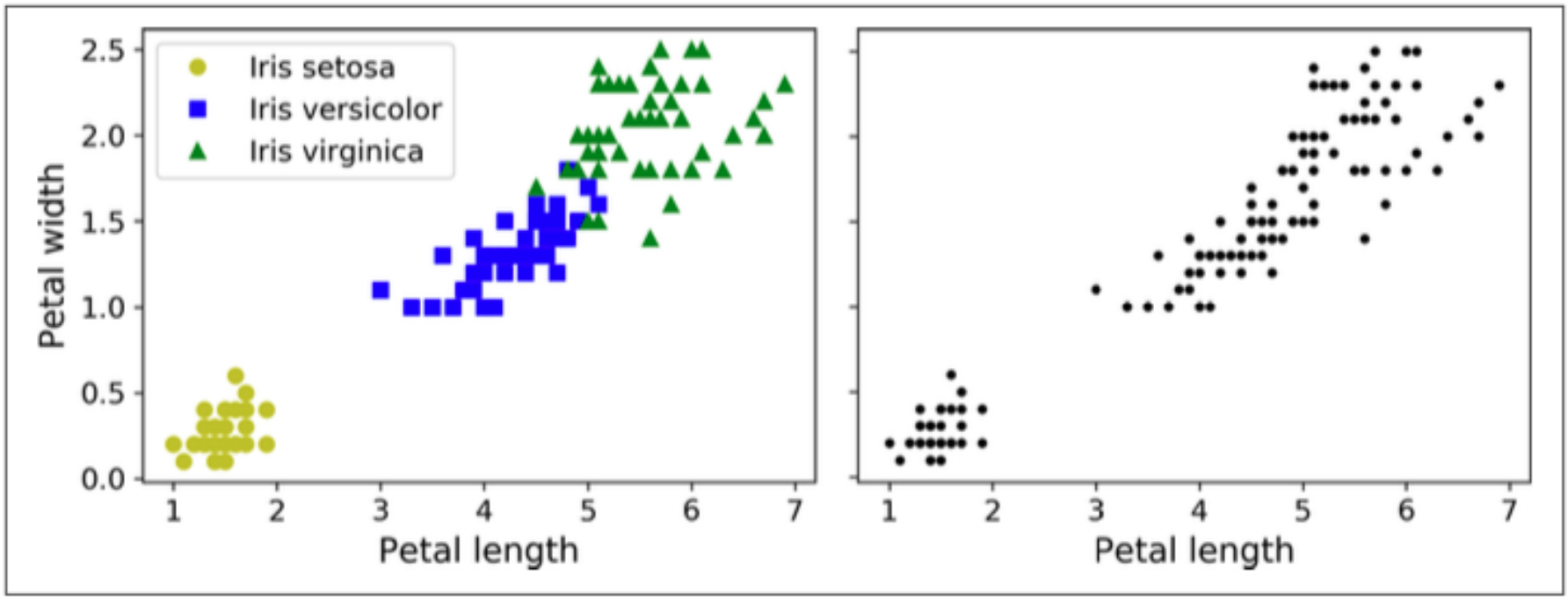
Autonomous drone to identify different species

1. We want to make an autonomous drone which is going to identify three different species of iris.
2. It is going to measure the length and width of the sepals and petals.
3. Use the the features from Iris dataset to design a simple unsupervised machine learning model.
4. This means the data will not have labels/targets. So model must discover patterns in the data.
5. We know the data comes from 3 different species though.

The cake analogy

If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake.
- Yann LeCun

Clustering



Similar to classification, each instance gets assigned to a group. However, unlike classification, clustering is an unsupervised task. Because the data is not labeled, so the unsupervised learner must tag the data points with a label by understanding the pattern in the data.

Aurélien Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow

Applications of k-means and clustering

k-means is also known as Lloyd's or Lloyd-Forgy algorithym.

- Customer segmentation
- Data analysis
- Dimensionality reduction
- Anomaly or outlier detection
- Semi-supervised learning
- Search engines (very popular for images)
- Segment an image

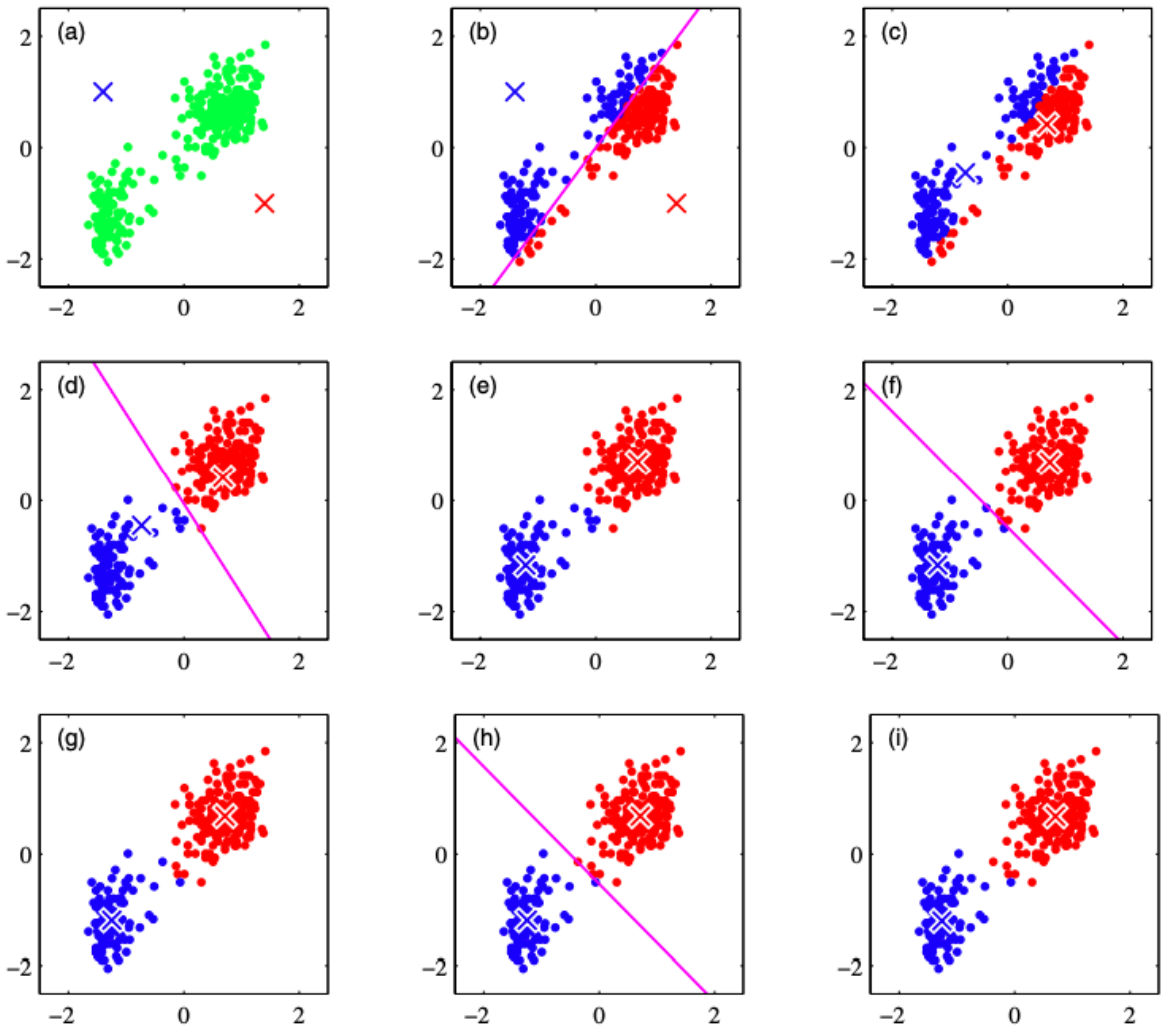
k-means algorithm

1. Initialize k centroids (means), denoted as $\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_k}$.
2. Repeat until convergence:
   For each data point $\boldsymbol{x_i}$: Compute the distance $d(\boldsymbol{x_i}, \boldsymbol{c_j})$ between $\boldsymbol{x_i}$ and each centroid $\boldsymbol{c_j}$ using a distance metric, i.e., L2 norm or Euclidean distance.
3. Assign $\boldsymbol{x_i}$ to the cluster whose centroid is closest to it (i.e. the cluster for which the distance is smallest) For each cluster j: Recalculate the centroid $\boldsymbol{c_j}$ as the mean of all the data points $\boldsymbol{x_i}$ assigned to it:

$$\boldsymbol{c_j} = \frac{1}{|S_j|} \sum_{\boldsymbol{x_i} \in S_j} \boldsymbol{x_i}$$

   Here, $|S_j|$ is the number of points in cluster j, and $\sum_{\boldsymbol{x_i} \in S_j}$ is the sum of all the data points $\boldsymbol{x_i}$ assigned to cluster j.
4. Output the clusters and centroids.

How clusters are assigned?



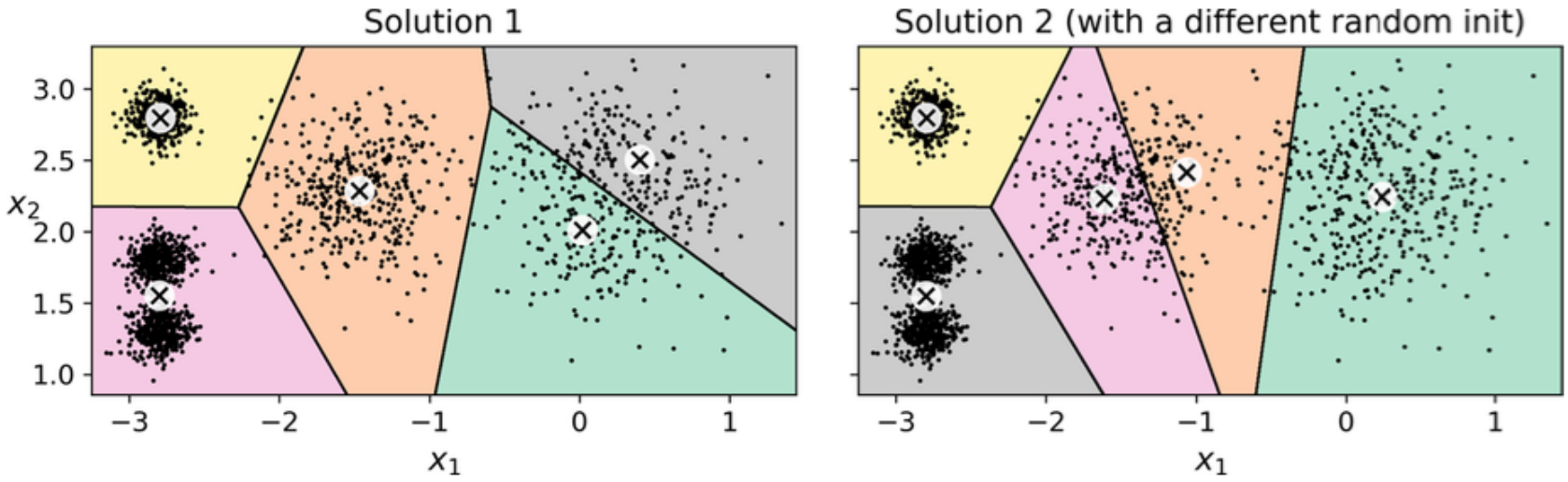
C. Bishop, Pattern Recognition and Machine learning

A simple implementation of k-means

```python
class KMeans:
    def __init__(self, k):
        self.k = k
        self.cluster_labels = None
    def fit(self, X):
        self.centroids = X[np.random.choice(X.shape[0], self.k, replace=False), :]
        self.cluster_labels = np.arange(self.k)
        while True:
            distances = np.array([np.linalg.norm(X - centroid, axis=1) for centroid in self.centroids])
            self.clusters = np.argmin(distances, axis=0)
            new_centroids = np.array([X[self.clusters == i, :].mean(axis=0) for i in range(self.k)])
            # check convergence
            if np.array_equal(new_centroids, self.centroids):
                break
            else:
                self.centroids = new_centroids
    def predict(self, X):
        distances = np.array([np.linalg.norm(X - centroid, axis=1) for centroid in self.centroids])
        return self.cluster_labels[np.argmin(distances, axis=0)]
```

The initialization and computation of distances can be improved.

Bad initialization and addressing suboptimal solutions



Suboptimal solutions from k-means due to unlucky centroid initializations.
Solution: Use centroid initialization methods

Kmeans++ (Arthur and Vassilvitskii) selects centroids distant from one another. This improvement makes the K-Means algorithm much less likely to converge to a suboptimal solution. Kmeans++ is used by default for Scikit's initialization.

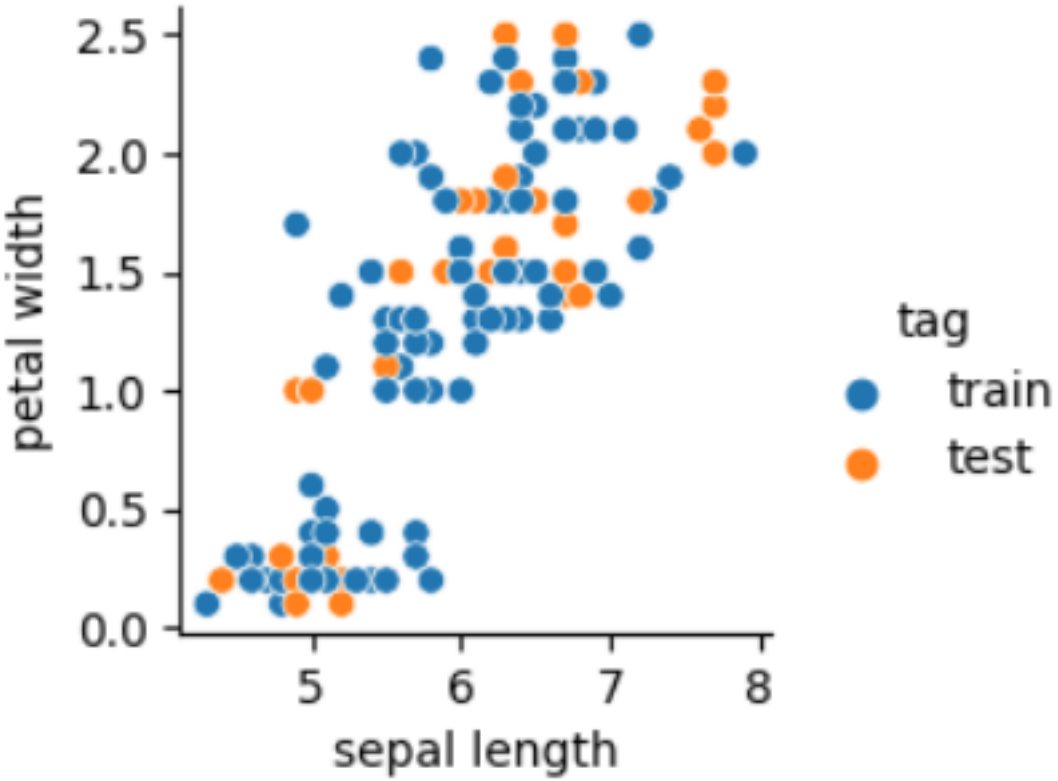
Aurélien Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow

Scikit's implementation of k-means

```
from sklearn.cluster import KMeans
skmodel = KMeans(
    n_clusters=3,
    init='k-means++',
    n_init='auto',
    max_iter=300,
    tol=0.0001,
    verbose=0,
    random_state=None,
    copy_x=True,
    algorithm='lloyd',
)
skmodel.fit(X_train)
```

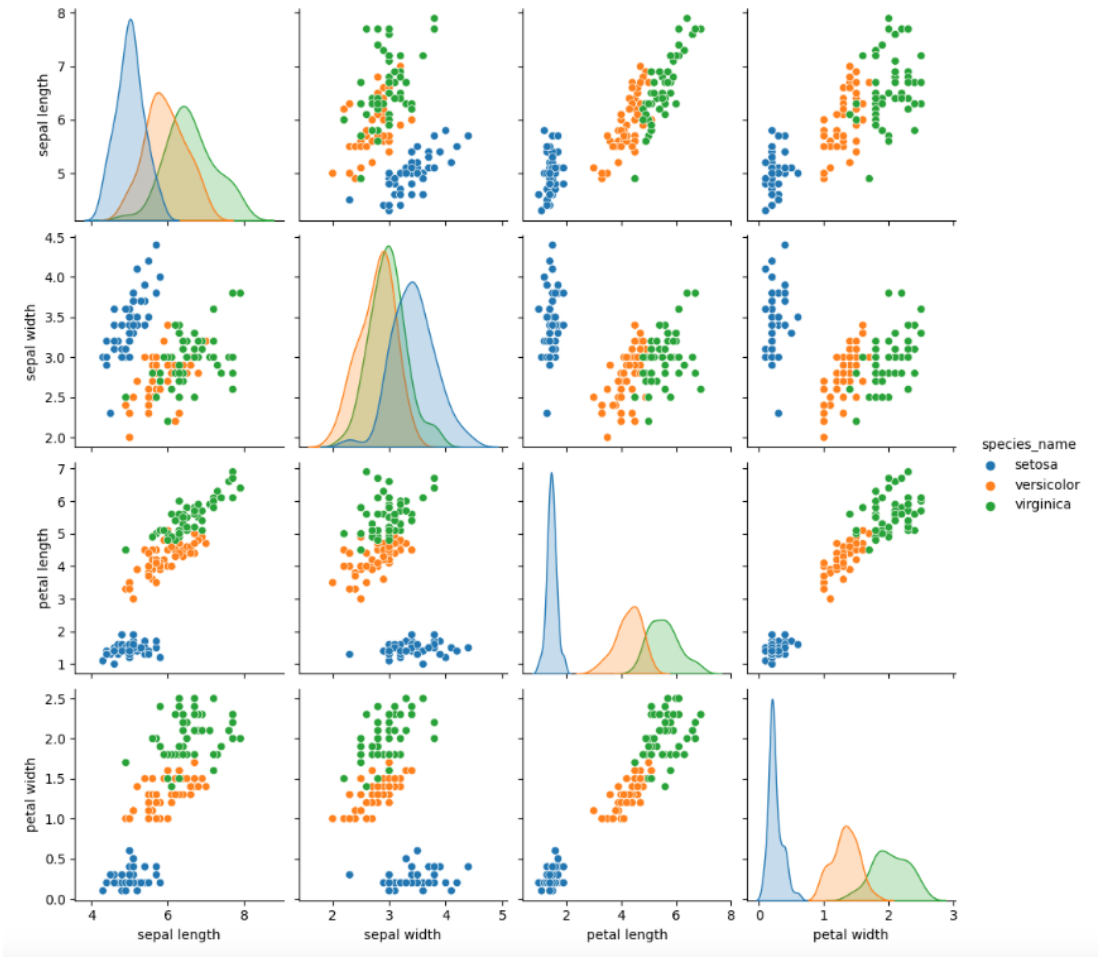
We can also compare against Scikit's implementation.

Splitting the data



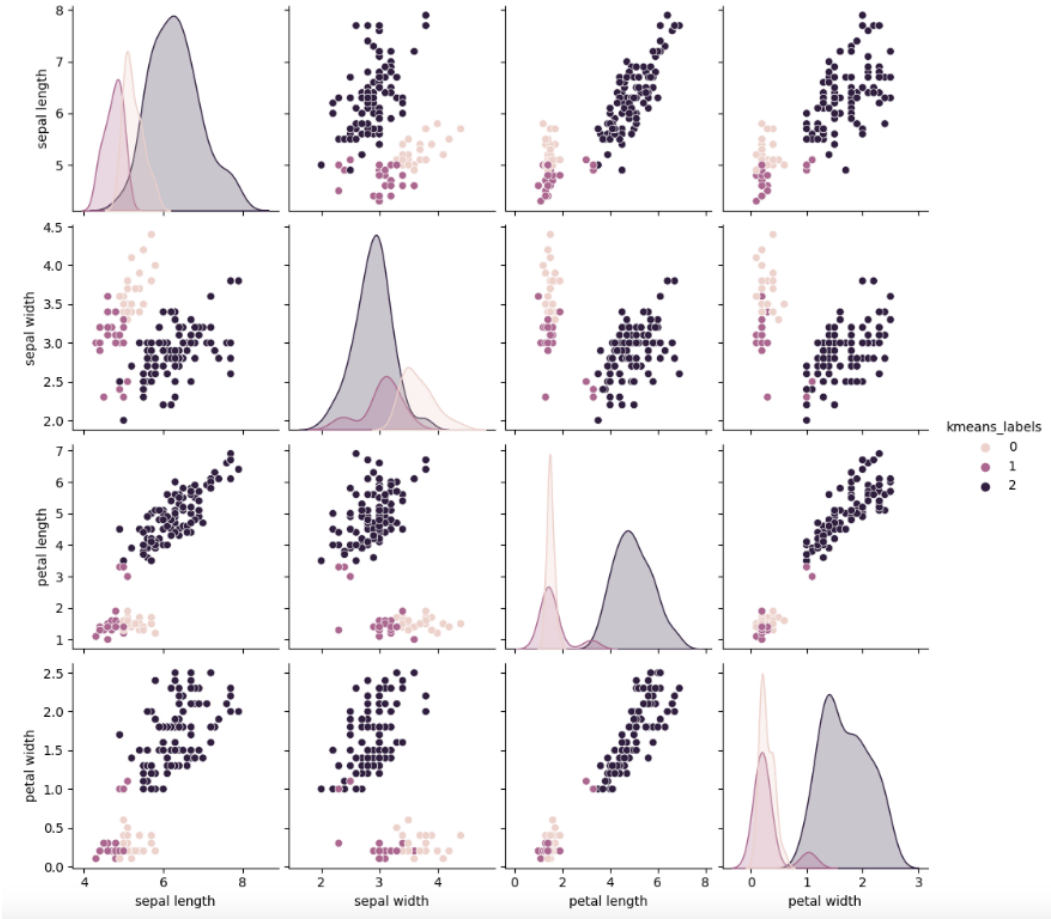
Should pick points from everywhere in the dataset.

Iris pairplot



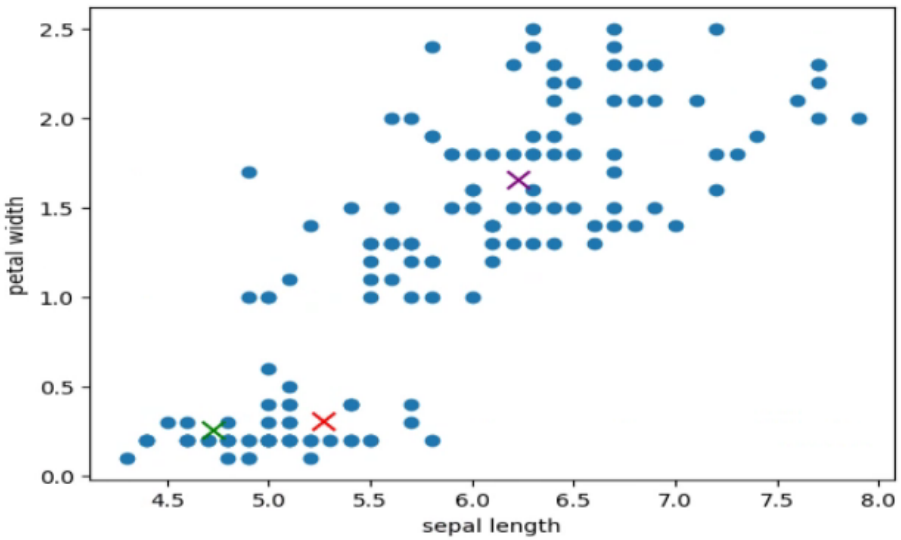
This is the true data.

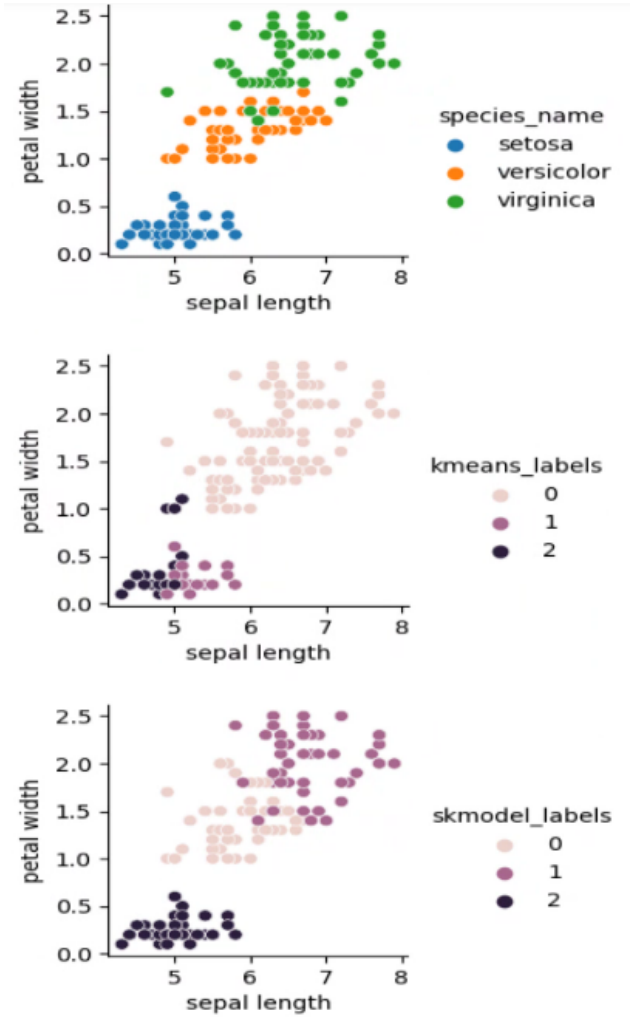
Predictions from simple KMeans



Remember the zeros, ones and twos here do not correspond to the class labels.

Comparison of the models on iris data





- Since the data is 4 dimensional for true visualization we need to create pairplots of the 4 features and plot centers in every dimension to get a true idea of the centers.

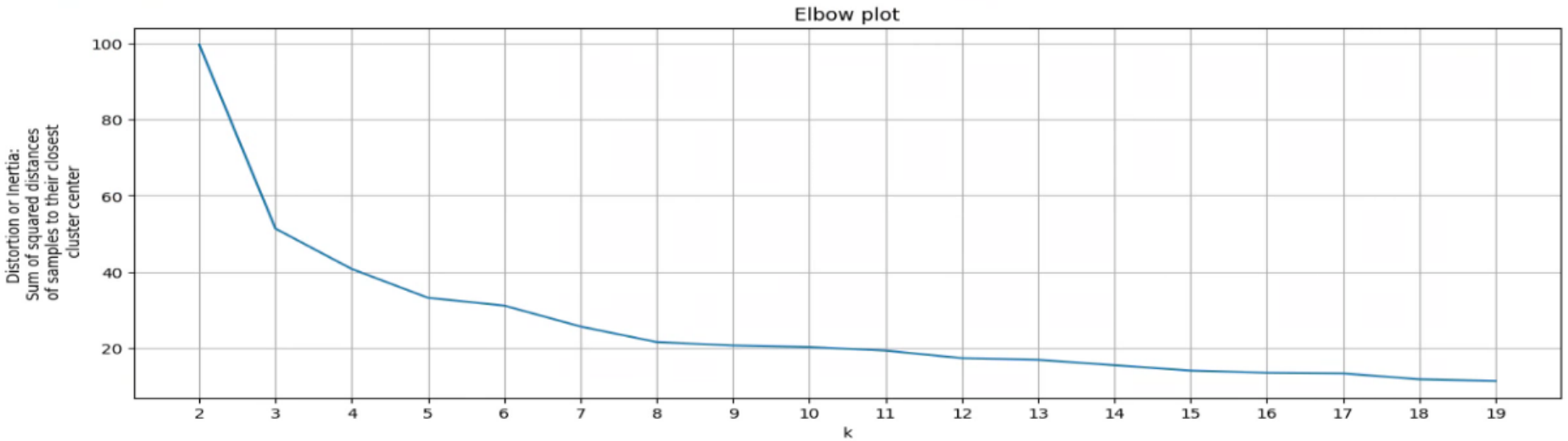
Improvements of k-means

Accelerated K-means (Elkan) accelerates the algorithm by exploiting the triangle inequality and by keeping track of lower and upper bounds for distances between examples and centroids. This is the algorithm the Scikit uses by default.

Mini-batch K-means (Sculley) uses mini-batches instead of the entire dataset to move the centroids just slightly at each iteration. Three or four times speed-up and uses memory efficiently. MiniBatchKMeans is available in Scikit too.

Choosing a good k (Elbow plot)

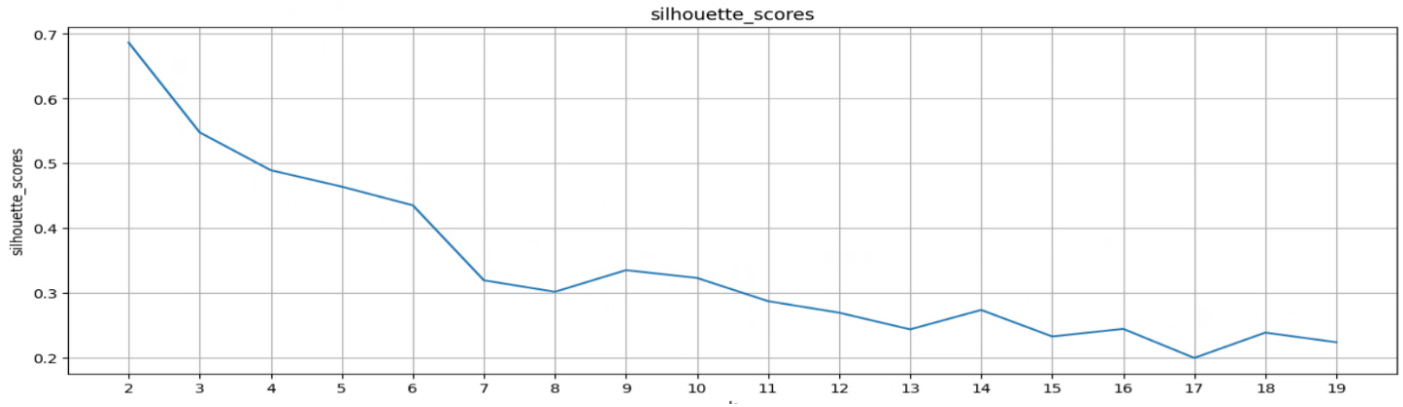
- A priori knowledge about problem or application domain.
  - There are two kinds of people in the world: k = 2 (good and great)
- Search for a good k
  - Use elbow plot (Inertia vs k)
  - Inertia = $\sum_{i=0}^{n}(||x_i - \mu_j||^2)$ for $\mu_j \in C$
  - Try different values of k and evaluate quality of results
  - Run hierarchical clustering on subset of data.



Plot distortion/intertia against k and choose its value where the plot starts to "bend". Can't pick lowest inertia because increasing k will keep decreasing inertia.

Choosing a good k (Silhouette score)

- Computational expensive but a better method than elbow.
  - Defined as mean silhouette coefficient over all the samples.
- For every sample the silhouette coefficient is equal to $\frac{(b \breve{} a)}{max(a,b)}$, where a is the mean distance to the other instances in the same cluster and b is the mean nearest-cluster distance.
- The silhouette coefficient can vary between –1 and +1.
- +1 means that the sample is well inside its own cluster and far from other clusters.
- 0 means that it is close to a cluster boundary.
- –1 means it may have been assigned to the wrong cluster.



Anything above 0.5 is a good choice.

Advantages and Disadvantages

- **Advantages:**
  - Convergence is guranteed (proof beyond scope). Although it might converge too quickly and produce unrealistic clusters.
  - Applicable to wide variety of problems, such as sorting, labeling, anomaly detection, density estimation, dimensionality reduction, image segmentation
  - Fast and scalable.
- **Disadvantages:**
  - Need to know or estimate k.
  - Can get stuck in local minima
  - **Not perfect, need a lot of correction to implement.**
  - Does not perform well with clusters of different densities (elliptical and sparse clusters).

The CIFAR-10 image dataset

```
1   from keras.datasets import cifar10
2   from keras.utils.np_utils import to_categorical
3   import matplotlib.pyplot as plt
4
5   # First time you run this it will download the data
6   (X_train, y_train), (X_test, y_test) = cifar10.load_data()
7
8   cifar_classes = ['airplane', 'automobile', 'bird', 'cat',
9                    'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
10  print('Example training images and their labels: ' + str([x[0] for x in y_train[0:5]]))
11  print('Corresponding classes for the labels: ' + str([cifar_classes[x[0]] for x in y_train[0:5]]))
12
13  f, axarr = plt.subplots(1, 5)
14  f.set_size_inches(16, 6)
15
16  for i in range(5):
17      img = X_train[i]
18      axarr[i].imshow(img)
19  plt.show()
```

Go to this page and learn about this dataset at https://www.cs.toronto.edu/~kriz/cifar.html