Chapter 12
K-Means

12.1 The Problem

So far, we have covered supervised learning algorithms for which the classes are predefined and the class of each instance of the training dataset is known in advance. The problem was to find a model that correctly classifies instances into their appropriate classes with a minimal cost (i.e., minimal error rate).

The problem in unsupervised learning is different; we have a dataset, but neither the classes nor the way to classify each instance in the training dataset is known in advance, i.e., the samples in the dataset are not labeled. For example, one might have data about residents of a city and want to cluster them geographically based on their assumed support for a candidate for election, or a medical image and want to cluster the pixel to perform image segmentation into similar regions, or a dataset about houses' characteristics (e.g., number of bedrooms, number of bathrooms, area, price, postal code) and want to cluster them by their value. Hospitals would be interested in uncovering clusters of patients who are high users of their services. Businesses might want to perform customer segmentation, i.e., to cluster customers based on some criteria, such as their purchases and their website activities; then, the businesses can recommend products for customers in the same clusters [1].

Our aim is to build a model that uncovers the clusters of data latent in the dataset based on feature similarities and classifies the instances into these clusters with a minimal error rate. How to label these clusters is the job of the data analyst.

C. El Morr et al., *Machine Learning for Practical Decision Making*, International Series in Operations Research & Management Science 334,
https://doi.org/10.1007/978-3-031-16990-8_12

12.2 A Practical Example

Let us consider the iris database and discount our knowledge of the class attribute. The instances are now not labeled, and we are in front of a clustering problem. We would like to use *K*-means to cluster the data into *n* clusters/categories. In this problem, we know that there are three clusters (*Iris setosa*, *Iris versicolor*, and *Iris virginica*); however, usually, *n* is suggested by a domain expert who understands the reality the data describes.

The dataset contains the length and width of the sepal and petal of the three types of irises. The dataset is labeled (i.e., class attribute); we can either delete the label or, if we are using Weka, we can ask the *K*-means algorithm to ignore the label. Figure 12.1 shows how to delete the label in Weka, while Fig. 12.6 shows how to ignore it when executing *K*-means in Weka; as usual, in the lab, you will be using Python and R.

We can start by exploring the dataset to have an understanding of the data trends and the problem. Figure 12.2 shows the histograms for each attribute by class. The dataset contains 50 instances for each type of iris. The *petallength* and *petalwidth*

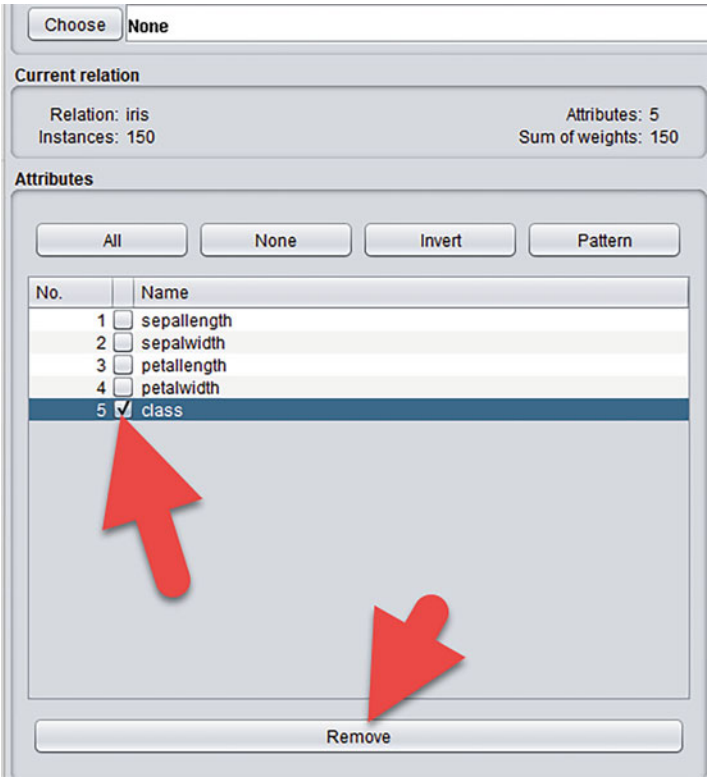


Fig. 12.1 Remove the class attribute

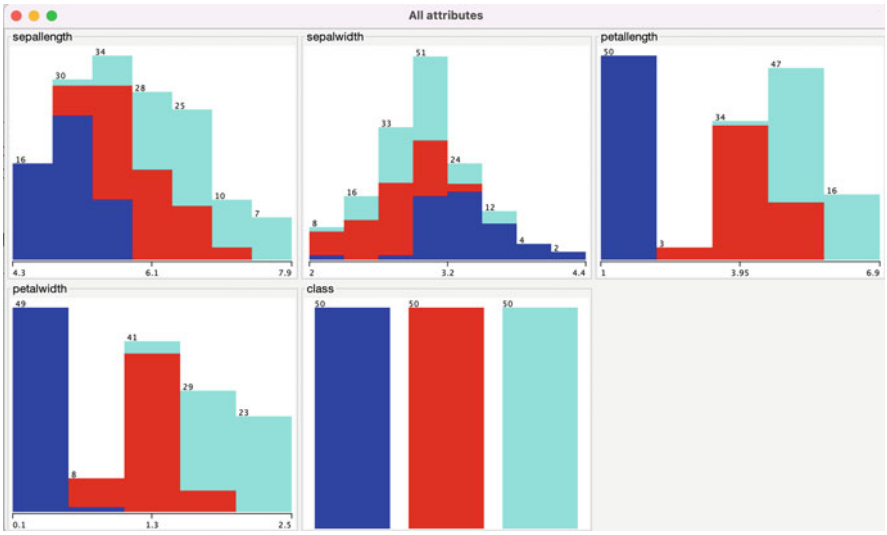


Fig. 12.2 Histograms for the different dataset features

histograms indicate that the petal length and width of one of the iris types (*Iris setosa*) are clearly distinct from the other two types, and hence these two attributes will help us identify *Iris setosa*. *Iris versicolor* and *Iris virginica* have common petal length and width, and the sepal length and width do not provide enough information to distinguish a separate class, as all three types of flowers have common values for these two attributes. We can already conclude that we will encounter errors in clustering, especially between *Iris versicolor* and *Iris virginica*.

When we explore the graph plots for the dataset features (Fig. 12.3), the scatterplot shows the feature on the *X*-axis (i.e., in the columns) against all other features on the *Y*-axis (i.e., in the rows). Again, we notice that the *Iris setosa* is separated from the other two species in each of the scatterplots. *Iris versicolor* and *Iris virginica* have many of their instances intermixed and are hard to distinguish in all the scatterplots (e.g., sepal width vs. sepal length).

Given these observations, let us apply the *K*-means algorithm to the dataset and explore the result (Fig. 12.4). Click on the algorithm's name to open the list of its parameters; since we know that we are seeking to detect three clusters, we will change the numClusters (i.e., *K*) parameter to 3 (Fig. 12.5); the default distance used is Euclidean, which fits our problem (i.e., length and width). Finally, we choose to ignore the Class parameter, as the label is not part of the features that we would use in a clustering problem (Fig. 12.6), and we run the algorithm.

The resulting window (Fig. 12.7) displays important information. We can notice that the algorithm converged after three iterations. The clusters are enumerated from 0 to $K - 1$, and Weka will display the centroid positions for each iteration (e.g., Cluster 0: 6.1, 2.9, 4.7, 1.4); the sum of the square error that we are trying to minimize is 6.99, and the number of instances that were assigned to the three clusters

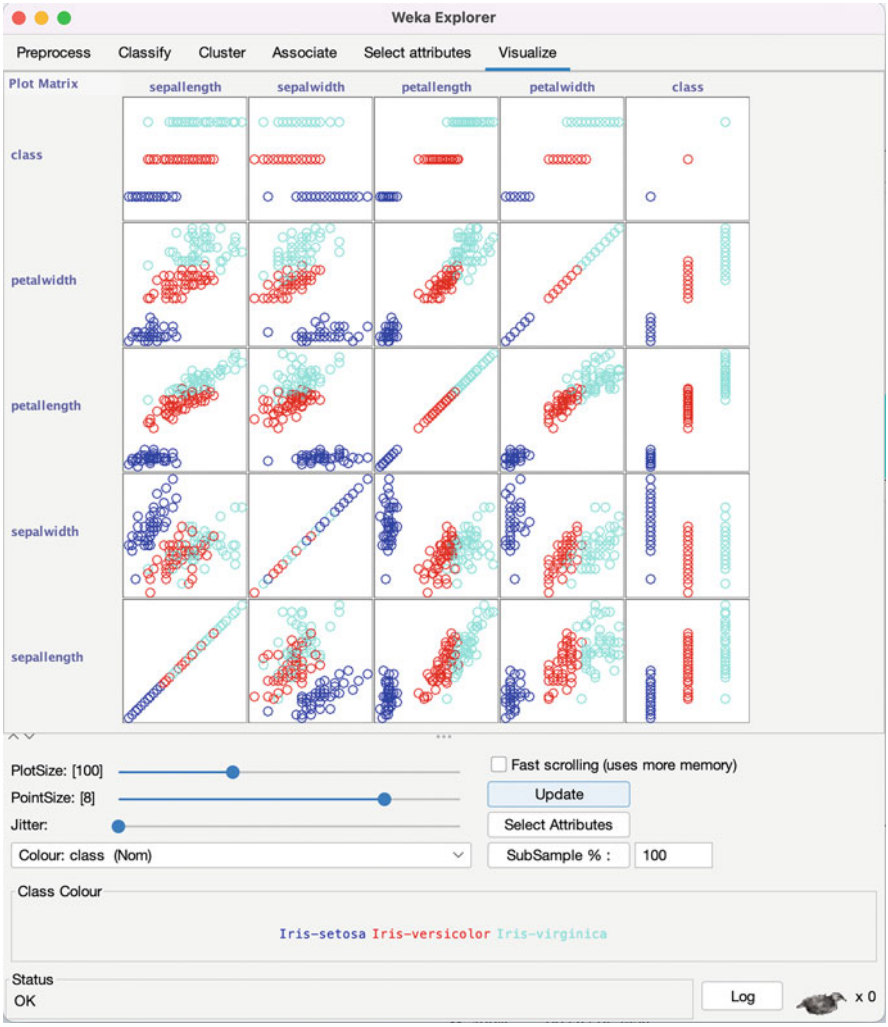


Fig. 12.3 Scatterplots for the dataset features

is 61, 50, and 39; since we know the labels, in this particular example, we know that there were instances assigned to the wrong cluster, but this is already expected given the instances' intermixing that we noticed during the data visualization phase.

Since the dataset contains the labels, we can ask *K*-means to consider the labels instead of ignoring them to let us know how many instances were assigned to the wrong cluster. In Weka, we can do so by choosing the "classes to clusters evaluation" and selecting the class from the list of features (Fig. 12.8). Clicking on Start this time shows an extra output (Fig. 12.9), the incorrectly clustered instances. We notice that the *K*-means algorithm incorrectly clustered 17 instances: 14 *Iris virginica* were

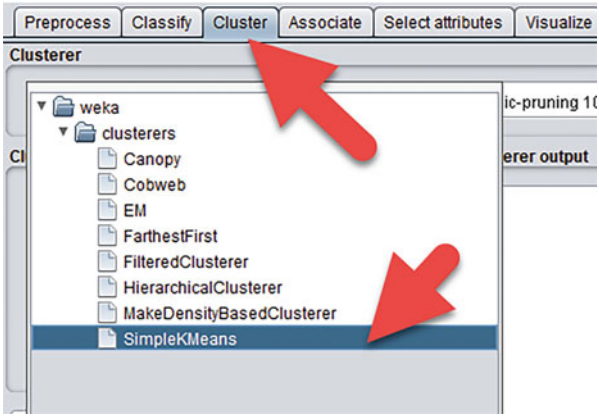


Fig. 12.4 Choose the SimpleKMeans algorithm from the Cluster tab in Weka

clustered with the *Iris versicolor*, and three *Iris versicolor* were clustered with the *Iris virginica*; the 50 instances of *Iris setosa* were all correctly clustered. As expected, the errors were related to the distinction between the *Iris versicolor* and *Iris virginica* species.

We can always visualize the cluster assignments by right-clicking on the name of the algorithm in the Result window and clicking on "Visualize Cluster Assignments" (Fig. 12.10). A new window opens and displays a scatterplot for data on the *X*- and *Y*-axes. To display a scatterplot showing the classes on the *Y*-axis, click on the Y dropdown list and choose the Cluster attribute. The scatterplot shows us how the instances were assigned, and we can see clearly that cluster 1 is distinct, 13 instances are assigned to cluster 0 while they clearly belong to cluster 2, and three instances are assigned to class 0 while they seem to belong to class 2 (Fig. 12.11).

We can check the cluster to which each instance was assigned by using a filter called AddCluster (Fig. 12.12).

We can click on the filter to choose the clustering algorithm and set its parameters. In our case, we would like to set *K* to 3 (Fig. 12.13).

When we apply the filter, a new attribute called "cluster" is created; it indicates the cluster associated with each instance. To display the data, it is enough to click on the Edit button (Fig. 12.14).

12.3 The Algorithm

The *K*-means algorithm involves the following steps:

- Specify the number of clusters *K*.
- Initialize the centroids by randomly selecting *K* samples from the training dataset.
- Assign samples to the clusters based on the closest centroid. The closeness is determined using a distance (e.g., Euclidean, Manhattan).

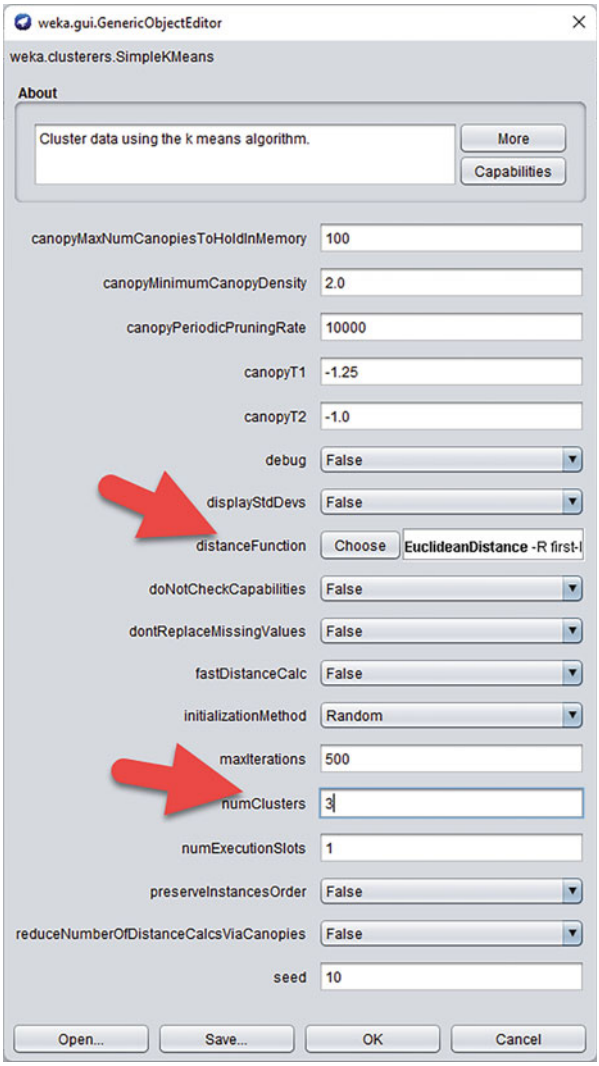


Fig. 12.5 Set the numClusters (i.e., *K*) parameters of the *K*-means algorithm to 3

– Update the centroid for each cluster by recalculating the center point for each cluster. The latter is the mean of the cluster's samples, hence the name *K*-means.
– Repeat assigning samples and updating centroids until model convergence, i.e., there is little change in the centroids, or a certain number of iterations is completed.

After convergence, the model is represented by the centroids. Each new sample is assigned to the closest centroid; that is, each new sample is assigned to one of the clusters 1 to *K* [2].

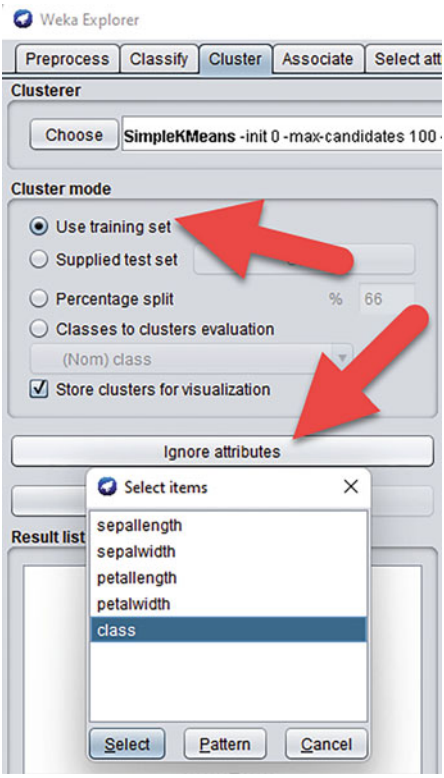


Fig. 12.6 Choose to ignore the class label before executing *K*-means

We can represent the dataset with a matrix of N instances with n features, the instances being the rows and the features, the columns of the matrix.

$$X = \begin{bmatrix} x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_n^{(N)} \end{bmatrix}$$

Each vector $x^{(i)} = \left[x_1^{(i)}\ x_2^{(i)} \ldots x_n^{(i)}\right]^T$ and $X = [x^{(1)}\ x^{(2)} \ldots x^{(N)}]^T$.

The K-means algorithm computes the distance between each vector $x^{(i)}$ and the centroids of the cluster $\mu_k\ k = 1, \ldots, K$. If N_k is the number of instances in the cluster k, then the centroid μ_k is calculated as follows:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N_k} x^{(i)}$$

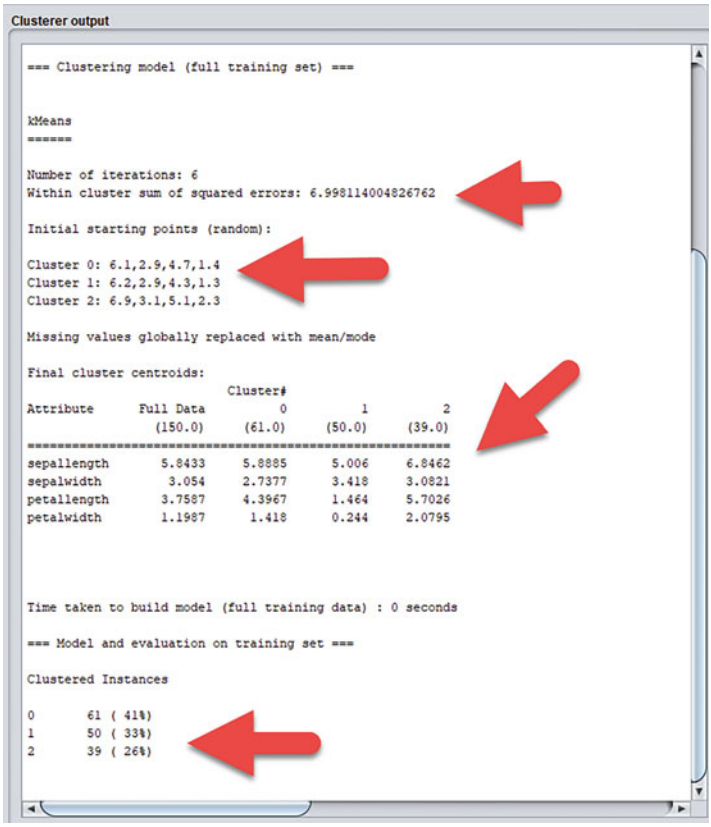


Fig. 12.7 *K*-means clustering result

Using the Euclidean distance (note that other distances can be used) to measure similarities between instances [3], the distance between an instance μ_k and a centroid μ_k can be computed as follows:

$$d_{ik} = \sqrt{\sum_{i=1}^{n} \left(x_j^{(i)} - \mu_{kj}\right)^2}$$

If we use the matrix notation, we can rewrite the same as follows:

$$x^{(i)} = \left(\left[x^{(i)} - \mu_k\right]^T \left[x^{(i)} - \mu_k\right]\right)^{1/2} = \left\|x^{(i)} - \mu_k\right\|$$

Each $x^{(i)}$ is considered as belonging to a cluster K if it is closest to it and hence satisfies the following condition:

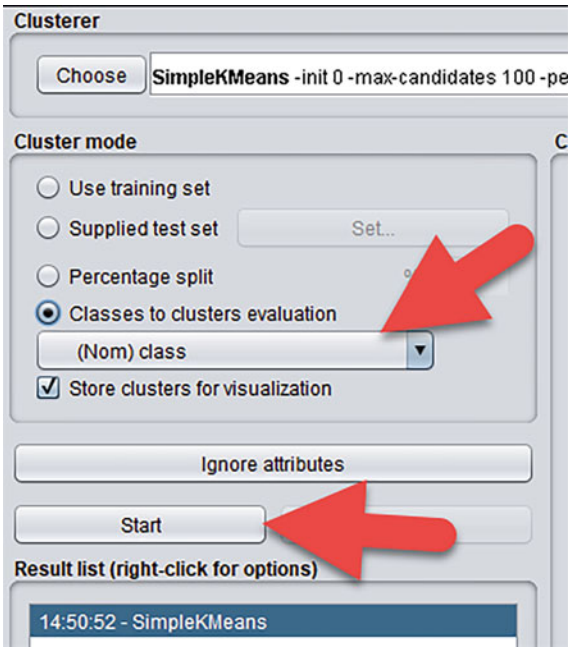


Fig. 12.8 Analyzing the incorrectly clustered instances

$$\left\|x^{(i)} - \mu_k\right\| < \left\|x^{(i)} - \mu_j\right\|; j = 1, \ldots, K, j \neq k$$

The algorithm stops when the change in the number of instances belonging to the clusters is minimal (less than a certain constant) or the clusters' center's location is minimal; the stopping criteria is

either

$$\sum_{k=1}^{K} \left|N_k^{t+1} - N_k^t\right| < \varepsilon$$

or

$$\sum_{k=1}^{K} \left|\mu_k^{t+1} - \mu_k^t\right| < \varepsilon$$

The algorithm can be summarized as follows:

1. Initialize k, and $\mu_1^{(t_0)}$ to $\mu_k^{(t_0)}$, and set the time $t = 1$.
2. Classify the N instances according to the nearest $\mu_k^{(t-1)}$:

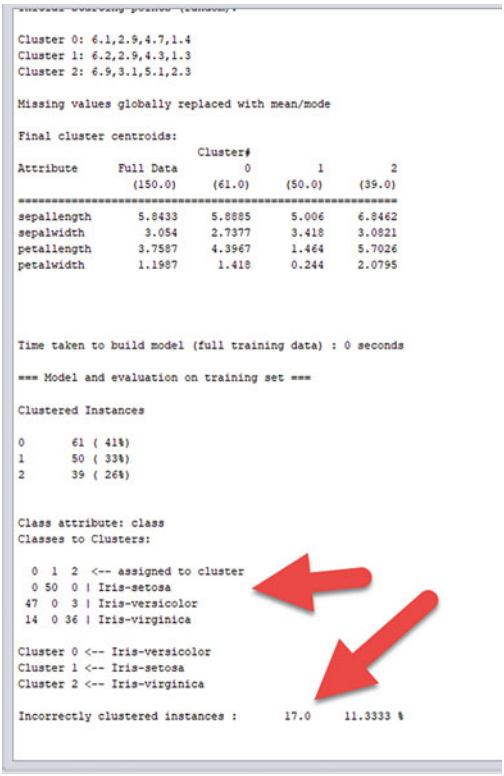


Fig. 12.9 Seventeen instances were incorrectly clustered

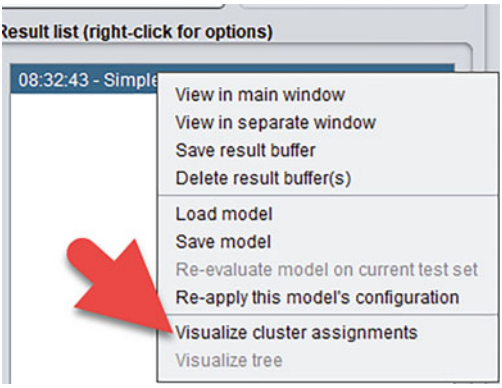


Fig. 12.10 Visualize cluster assignments

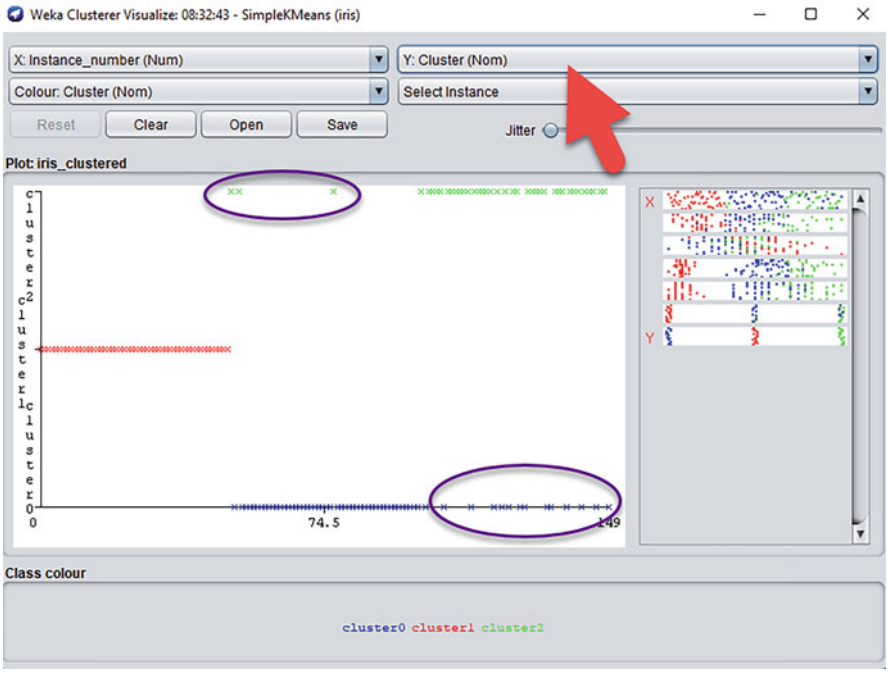
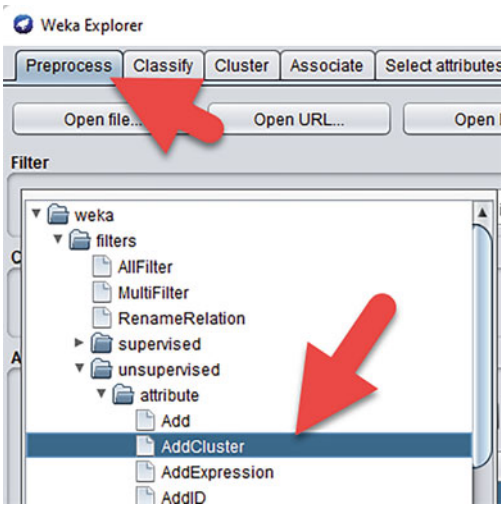


Fig. 12.11 Cluster visualization

Fig. 12.12 AddCluster filter



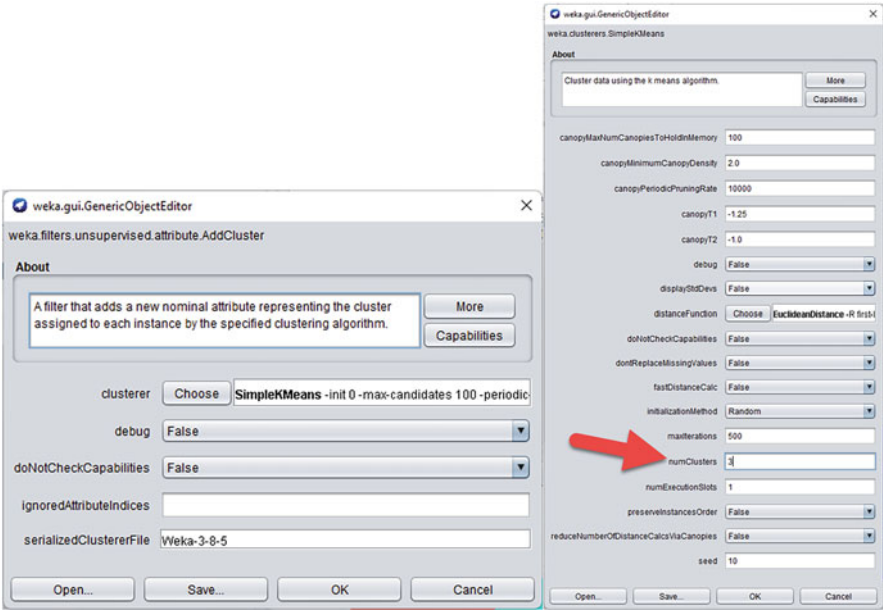


Fig. 12.13 The *K*-means parameters in the AddCluster filter

$$\left\|x^{(i)} - \mu_k^{(t-1)}\right\| < \left\|x^{(i)} - \mu_k^{(t-1)}\right\|; j = 1, \ldots, K, j \neq k$$

3. For each group of instances $N_k^{(t-1)}$ in a cluster K, $k = 1, \ldots K$, recompute:

$$\mu_k = \frac{1}{N_k^{(t-1)}} \sum_{i=1}^{N_k^{(t-1)}} x^{(i)}$$

4. Stop when the stopping criteria are satisfied; otherwise, increment t: $t = t + 1$ and repeat from step 2.
5. Return the result $\mu_1^{(t)}$ to $\mu_k^{(t)}$.

12.4 Inertia

Once the learning is one (using .fit() in Python), the algorithm can always assign a new instance to the closest centroid (using .predict()), which is called a hard clustering.

Alternatively to hard clustering, we can perform soft clustering that assigns scores to the new instance, each score is equivalent to the distance of the new instance to one of the centroids (.transform() in Python). In this way, each data instance can be

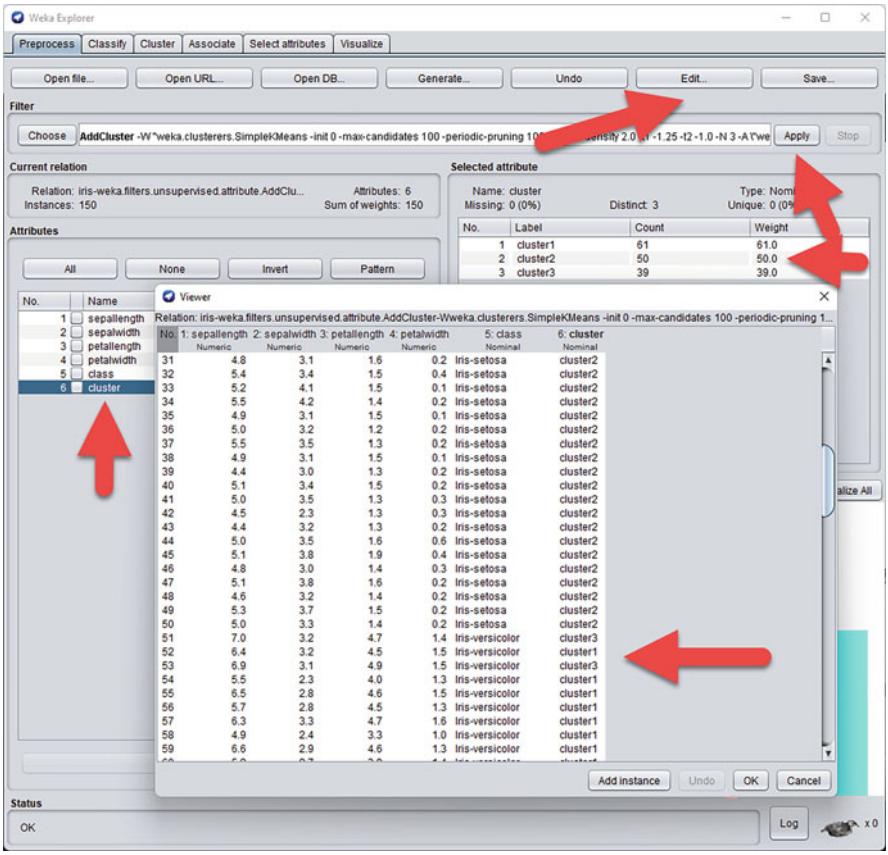


Fig. 12.14 Display dataset values

represented by its position to the K clusters, if K is lower than the dataset dimension, the result of clustering would be a reduction in the data dimensions; so, K-means could be used for dimensionality reduction.

The K-means is sensitive to the initial centroids and can provide you with different solutions if you start with different initial centroids. So how to choose the best of many solutions? What is the cost function to minimize? Since this is an unsupervised learning, we do not have labels to measure the performance of the algorithm, what we have seen in the Weka example above was only for learning purposes. However, there is one performance measurement for K-means called inertia, that consist of the within-cluster sum-of-squares that measures the sum of squared distances of each instance to its centroid. Inertia measures the internal coherence of the clusters; a low inertia means a better clustering solution. We can run K-Means multiple times (in Python this is controlled by K-Means *n_init* hyperparameter), and K-means will use inertia to find an optimal clustering solution.

12.5 Minibatch *K*-Means

The original *K*-means algorithm was proposed by Stuart Lloyd in 1957 [4]. To accelerate the execution of *K*-Means, a faster new algorithm was proposed in 2003 by Charles Elkan [5]; both version can be set by the *K*-Means *algorithm* hyperparameter in Python, the default being Lloyd's. An even faster version of *K*-Means was proposed in 2010 by David Sculley [6] that uses min-batches of the dataset instead of the full dataset at each iteration, it is implemented under the name MiniBatchKMeans in Python.

12.6 Final Notes: Advantages, Disadvantages, and Best Practices

K-means models are widely used because of their simplicity. They can easily scale and generalize to different data size/shapes and easily adapts to new scenarios. As convergence is key for accuracy and optimal decisions, *K*-means model has shown good convergence results. Choosing the number of centroid (K) is a challenge, and we will see a method to find the optimal K in Sect. 12.10 below.

Like any ML model, *K*-means has its own disadvantages, including the manual choice of K value, which makes it dependent on initial values. One of key issues of *K*-means is the impact of outliers on clusters generation; if the data is not well preprocessed, outliers can have their own clusters. Lastly, highly dimensional data can generate curse of dimensionality problem, thus affecting the convergence of the model. Using feature engineering techniques can mitigate this issue.

When working with *K*-means, consider trying many values for K, as well as many clustering algorithms (e.g., *K*-Means, DBSCAN, and test which setup provides better results. You may also want to take special care to avoid overfitting, mainly you need to increase the value of K to eliminate noises in clusters.

12.7 Key Terms

1. *K*-means
2. Clustering
3. Cluster
4. Centroids
5. Matrix
6. Segmentation
7. Image segmentation
8. Customer segmentation

9. Hard clustering
10. Soft clustering

12.8 Test Your Understanding

1. Give examples of situations from different fields where we can use clustering.
2. How do you decide which cluster a data instance belongs to?
3. How do you decide the centroid of each cluster?
4. How do you determine the centroids of the first clusters?
5. How do you determine the number of clusters K?
6. What is the stopping criterion for K-means?
7. There is a method called "elbow" that allows us to compute the optimum number of clusters in a dataset. Explore this method; you will be using it in the lab.
8. Cite some of the hyperparameters of K-means.
9. What is inertia in K-Means? What is it used for?

12.9 Read More

1. Aris, T. A., Nasir, A. S. A., & Mohamed, Z. (2021). A Robust Segmentation of Malaria Parasites Detection using Fast K-Means and Enhanced K-Means Clustering Algorithms. 2021 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), 128–133. doi: 10.1109/ICSIPA52582.2021.9576799
2. Armstrong, J. J., Zhu, M., Hirdes, J. P., & Stolee, P. (2012). K-means cluster analysis of rehabilitation service users in the Home Health Care System of Ontario: examining the heterogeneity of a complex geriatric population. Arch Phys Med Rehabil, 93(12), 2198–2205. doi: 10.1016/j.apmr.2012.05.026
3. Fahim, A. (2021). K and starting means for K-means algorithm. Journal of Computational Science, 55(Complete). doi: 10.1016/j.jocs.2021.101445
4. Gesicho, M. B., Were, M. C., & Babic, A. (2021). Evaluating performance of health care facilities at meeting HIV-indicator reporting requirements in Kenya: an application of K-means clustering algorithm. BMC Med Inform Decis Mak, 21(1), 6. doi: 10.1186/s12911-020-01367-9
5. Grant, R. W., McCloskey, J., Hatfield, M., Uratsu, C., Ralston, J. D., Bayliss, E., & Kennedy, C. J. (2020). Use of Latent Class Analysis and K-Means Clustering to Identify Complex Patient Profiles. JAMA Netw Open, 3(12), e2029068. doi: 10.1001/jamanetworkopen.2020.29068
6. Jabi, M., Pedersoli, M., Mitiche, A., & Ayed, I. B. (2021). Deep Clustering: On the Link Between Discriminative Models and K-Means. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(6), 1887–1896. doi: 10.1109/TPAMI.2019.2962683

7. Khan, A. R., Khan, S., Harouni, M., Abbasi, R., Iqbal, S., & Mehmood, Z. (2021). Brain tumor segmentation using K-means clustering and deep learning with synthetic data augmentation for classification. Microsc Res Tech, 84(7), 1389–1399. doi: 10.1002/jemt.23694
8. Kwedlo, W., & Lubowicz, M. (2021). Accelerated K-Means Algorithms for Low-Dimensional Data on Parallel Shared-Memory Systems. IEEE Access, 9, 74286–74301. doi: 10.1109/ACCESS.2021.3080821
9. Li, Y., Zhang, Y., Tang, Q., Huang, W., Jiang, Y., & Xia, S.-T. (2021). t-k-means: A ROBUST AND STABLE K-means VARIANT. ICASSP 2021—2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 3120–3124. doi: 10.1109/ICASSP39728.2021.9414687
10. Sinaga, K. P., Hussain, I., & Yang, M.-S. (2021). Entropy K-Means Clustering With Feature Reduction Under Unknown Number of Clusters. IEEE Access, 9, 67736–67751. doi: 10.1109/ACCESS.2021.3077622
11. Tavallali, P., Tavallali, P., & Singhal, M. (2021). K-means tree: an optimal clustering tree for unsupervised learning. The Journal of Supercomputing, 77(5), 5239–5266. doi: 10.1007/s11227-020-03436-2
12. Yuan, R. (2021). An improved K-means clustering algorithm for global earthquake catalogs and earthquake magnitude prediction. Journal of Seismology, 25(3), 1005–1020. doi: 10.1007/s10950-021-09999-8
13. Zukotynski, K., Black, S. E., Kuo, P. H., Bhan, A., Adamo, S., Scott, C. J. M., . . . Gaudet, V. (2021). Exploratory Assessment of K-means Clustering to Classify 18F-Flutemetamol Brain PET as Positive or Negative. Clin Nucl Med, 46(8), 616–620. doi: 10.1097/rlu.0000000000003668

12.10 Lab

12.10.1 Working Example in Python

Download the dataset using the following link: https://www.kaggle.com/datasets/uciml/adult-census-income

This dataset includes demographics and income level (above or less or equal to 50K). It includes the following columns:

- Age: person's age
- Workclass: work class type the person belongs to
- Fnlwgt: final weight estimate for the specified socioeconomic characteristics of the population
- Education: education level
- Education.num: education level as a number
- Marital.status: person's marital status
- Occupation: person's occupation
- Relationship: person's relationship status
- Race: person's race

- Sex: person's gender
- Capital.gain: an increase in the person's profit
- Capital.loss: a decrease in the person's profit
- Hours.per.week: number of working hours per week for the person
- Native.country: the person's original country
- Income: the person's salary

12.10.1.1 Load Person's Demographics

After downloading the adult census dataset, load the dataset (Fig. 12.15).

12.10.1.2 Data Visualization and Cleaning

You can explore the data visually; we will content with one plot (Fig. 12.16).

```
# Imports Required Libraries
import numpy as np
import pandas as pd # data processing
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.cluster import KMeans
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix, roc_curve,accuracy_score,roc_auc_score,classification_report,auc
from sklearn.metrics import f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

%matplotlib inline

# Load persons Dataset
df = pd.read_csv('adult.csv')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64
 11  capital.loss    32561 non-null  int64
 12  hours.per.week  32561 non-null  int64
 13  native.country  32561 non-null  object
```

Fig. 12.15 Load persons' census income dataset into pandas

```
#Data Visualisation
f, ax = plt.subplots(figsize=(7, 5))
sns.countplot(x='income', data=df)
_ = plt.title('(Person\'s income <= 50k) vs (Persons Income > 50k)')
_ = plt.xlabel('Person\'s Income')
_ = plt.ylabel('Frequency')
```

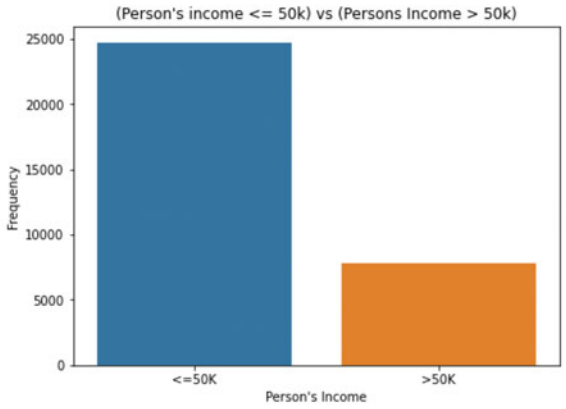


Fig. 12.16 Visualizing persons' census income data in histogram

```
df["workclass"] = df["workclass"].replace('?','unknown')
df["occupation"] = df["occupation"].replace('?','unknown')

#drop two irrelevant features
df =df.drop(["native.country"], axis =1)
```

Fig. 12.17 Replacing "?" with "unknown" and dropping the "native.country"

12.10.1.3 Data preprocessing

"Workclass" and "occupation" features that contain interrogation marks, and we will start by replacing the "?" with the word "unknown," while the "native.country" feature is irrelevant for our work and we will drop it altogether (Fig. 12.17).

Many features (workclass, marital.status, occupation, relationship, race, sex) as well as the target (income) are categorical data and hence need one-hot-encoding. The features that are one-hot encoded will be split into many new "dummy" features, if we are working with a linear regression algorithm that would be a problem as the values of any of the features can be deduced if we know the values of all other ones (if all features other than the dropped one are zero, then for sure the dropped feature is 1, and if any one of the other features is 1 then for sure the dropped feature value is zero). This means that one of the features is always correlated with all others. With *K*-means we do not need to do so. We show how to proceed with splitting the categorical features into new features in Fig. 12.18.

```
# get the dummy variables from te workclass feature
dummies = pd.get_dummies(df.workclass)

# Concatenate the dummies to the original dataframe
merged = pd.concat([df, dummies], axis='columns')

# drop the workclass features as it is not needed anymore
df=merged.drop(columns=['workclass'])

# print the dataframe info for verification
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   age               32561 non-null  int64
 1   fnlwgt            32561 non-null  int64
 2   education         32561 non-null  object
 3   education.num     32561 non-null  int64
 4   marital.status    32561 non-null  object
 5   occupation        32561 non-null  object
 6   relationship      32561 non-null  object
 7   race              32561 non-null  object
 8   sex               32561 non-null  object
 9   capital.gain      32561 non-null  int64
 10  capital.loss      32561 non-null  int64
 11  hours.per.week    32561 non-null  int64
 12  native.country    32561 non-null  object
 13  income            32561 non-null  object
 14  Federal-gov       32561 non-null  uint8
 15  Local-gov         32561 non-null  uint8
 16  Never-worked      32561 non-null  uint8
 17  Private           32561 non-null  uint8
 18  Self-emp-inc      32561 non-null  uint8
 19  Self-emp-not-inc  32561 non-null  uint8
 20  State-gov         32561 non-null  uint8
 21  Without-pay       32561 non-null  uint8
dtypes: int64(6), object(8), uint8(8)
memory usage: 3.7+ MB
```

Fig. 12.18 One-hot encoding workclass categorical variable

The *education* feature in ordinal in nature, so we will change the string values to numeric ordered values (Fig. 12.19).

```
# code the label/outcome to 0 and 1
df["income"] = df["income"].replace('<=50K', 0)
df["income"] = df["income"].replace('>50K', 1)
```

```
df["education"] = df["education"].replace('10th', 1)
df["education"] = df["education"].replace('11th', 2)
df["education"] = df["education"].replace('12th', 3)
df["education"] = df["education"].replace('1st-4th', 4)
df["education"] = df["education"].replace('5th-6th', 5)
df["education"] = df["education"].replace('7th-8th', 6)
df["education"] = df["education"].replace('9th', 7)
df["education"] = df["education"].replace('Assoc-acdm', 8)
df["education"] = df["education"].replace('Assoc-voc', 9)
df["education"] = df["education"].replace('Bachelors', 10)
df["education"] = df["education"].replace('Some-college', 11)
df["education"] = df["education"].replace('Prof-school', 12)
df["education"] = df["education"].replace('Masters', 13)
df["education"] = df["education"].replace('HS-grad', 14)
df["education"] = df["education"].replace('Doctorate', 15)
df["education"] = df["education"].replace('Preschool', 16)

df.info()
```

Fig. 12.19 Mapping ordinal categorical values into numeric values

```
# Prepare the feature vector x and the class vector y
#x = df.values # INCORRECT : need to drop income
x = df.drop(['income'], axis=1).values
y = df['income'].values

#scaling the whole dataset
scl = MinMaxScaler() #is it better thn StandardScaler
x = scl.fit_transform(x)
```

Fig. 12.20 Preparing the data for analysis

12.10.1.4 Choosing Features and Scaling Data

The feature and the target vectors are prepared and data is normalized (check the results if you do not normalize the data, how would the clustering be affected) (Fig. 12.20).

12.10.1.5 Finding the Best *K* for the *K*-Means Model

To find the best *K* for the *K*-means we will proceed manually using a method called the "the elbow" method and also we will sue the usual grid search cross-validation method.

```
# finding the best K: Elbow method
sse = [] #n array to store the sum of square error
number_clusters = range(1,7)
for i in range(1, 7):
    km = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    km.fit(x)
    sse.append(km.inertia_) # append the intertia: intertia=the sum of squared distance of samples to their closest cluster center
    print('k={}, SSE={}'.format(i, km.inertia_))

plt.plot(range(1,7), sse)
plt.title('Optimizing Using The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Cost')
plt.show()
```

```
k=4, SSE=67865.22238230314
k=5, SSE=65021.58351581503
k=6, SSE=61272.519285730545
```

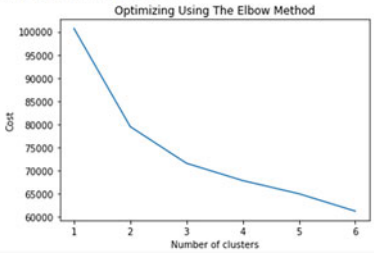


Fig. 12.21 Finding the optimal K using elbow method

```
# finding the best K: GridSearch
from sklearn.model_selection import GridSearchCV
from sklearn import metrics

clusters = KMeans(n_clusters=2,random_state=42)

params = {
            'n_clusters': [2, 3, 5, 7,9,11,13,15]
            }
gridcv = GridSearchCV(clusters, params, cv = 10, scoring='accuracy')

#fit the model
gridcv = gridcv.fit(x,y)
gridcv
```

```
GridSearchCV(cv=10, estimator=KMeans(n_clusters=2, random_state=42),
             param_grid={'n_clusters': [2, 3, 5, 7, 9, 11, 13, 15]},
             scoring='accuracy')
```

Fig. 12.22 Optimizing K-Means hyperparameters using grid search cross-validation

The elbow method consists of drawing for each possible K the inertia cost function. (Fig. 12.21), we can notice that cost declines significantly at the beginning from $K = 1$ to $K = 2$, afterwards the decline is less pronounced. It seems like $K = 2$ is the optimal number of clusters. This indeed reflects the actual data, where we have two clusters: people who earn more than 50k and others who earn 50K or less.

Using grid search cross-validation, we can also find the K for the optimal model (Fig. 12.22). It is important to note that the GridSearch varied the K parameter between 2 and 15 and resulted in $K = 2$ for the optimal K-Means.

```
optimalmodel=gridcv.best_estimator_
print ('best estimator=',optimalmodel)
y_pred=optimalmodel.predict(x)
y_pred
```

```
best estimator= KMeans(n_clusters=2, random_state=42)
array([0, 0, 0, ..., 1, 0, 0])
```

```
# trasnform assigns for each instance a score per cluster (in our case, two scores)
y_pred=optimalmodel.transform(x)
y_pred
```

```
array([[2.47161261, 3.13142783],
       [2.14949519, 2.87394702],
       [2.74825922, 3.339599  ],
       ...,
       [2.07611226, 1.09214504],
       [1.91374206, 2.69238109],
       [1.6420748 , 2.31297143]])
```

```
clusters=optimalmodel.cluster_centers_
clusters
```

```
array([[ 2.47044713e-01,  1.21690293e-01,  6.44801470e-01,
         5.94630756e-01,  5.88518078e-03,  1.43223689e-02,
         3.76464983e-01,  2.74528155e-02,  6.37965996e-02,
         3.63957781e-04,  7.29059429e-01,  1.70020278e-02,
         5.12660531e-02,  4.07112775e-02,  4.67945718e-04,
```

Fig. 12.23 Optimizing *K*-means model using grid search cross-validation

Then you can predict the clusters of the feature vector x using the optimal model found by GridSearch. As mentioned above the .predict() will perform hard clustering, and you can display for each data instance the cluster to which it is assigned. However, .transform() will show perform soft clustering and, in our case, assigns for each datapoint two scores representing the distance between the instance and the two centroids. Finally, .labels_ and .cluster_centers_ allow you to display the labels of each point as well as the coordinates of clusters' centers (Fig. 12.23).

You can think of plotting the datapoints and the centers. You can try that in Sect. 12.10.2.

12.10.2 Do It Yourself

12.10.2.1 The Iris Dataset Revisited

In this section, create a *K*-means model using different values for *K* (try $K = 2$, 5, 10, 15, 20, and 25) using the iris dataset.

You can download the Iris dataset using the following code

```
import numpy as np
import pandas as pd
from sklearn import datasets
iris =datasets.load_iris()
```

Note the variable "iris" is an array and not a data frame; you should be able by now to know how to convert an array to a data frame, but that is not necessary. To learn more about other available data set, click on the following link: https://scikit-learn.org/stable/datasets.html.

You can always now the features names using "feature_names" and the target name using "target_names."

```
iris.feature_names
iris.target_names
```

If you want to convert data to a data frame you can write the following code

```
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
columns= iris['feature_names'] + ['class'])
df.info() # display the data frame information
df # display the first few rows
```

Note: other than the elbow method explained above, there is the silhouette score that allows you to uncover the best K value. Read about the silhouette_score to choose the best K for the Isis dataset.

12.10.2.2 *K*-Means for Dimension Reduction

Download the digits dataset using the following code

```
from sklearn.datasets import load_digits
x, y = load_digits(return_X_y=True)
```

1. Phase 1: basic multiclass classification using logistic regression
   (a) Split the data into training and testing dataset.
   (b) Use logistic regression for multiclass classification (i.e., multi_class="ovr") and consider max_iter=5000.
   (c) Fit the model to the training dataset.
   (d) Compute the algorithm score using the testing dataset.

2. Phase 2: seek enhancement using *K*-Means for preprocessing
   (a) Create a pipeline for *K*-Means followed by logistic regression. Since the digits are handwritten and can be present in many ways, choose the number of clusters for *K*-Means much larger than 10, try 50. The logistic regression parameters are the same as in phase 1.
   (b) Fit the pipeline on the training dataset.
   (c) Compute the pipeline score no the testing dataset.
   (d) Compared to the previous score, was this one better or worse? Discuss the possible reasons behind the new score.
3. Phase 3: search for an optimal *K*-Means and logistic regression.
   (a) Find through GridSearch the optimal model for the pipeline. Finetune only one hyperparameter for *K*-Mean: the number of clusters; vary is between 2 and 100. Note that the execution might take around 20 min depending on your computer configuration.
   (b) Fit the pipeline and check the new score.
   (c) Compared to the previous score, was this one better or worse? Discuss the possible reasons behind the new score.
   (d) What was the best parameter found by GridSearch?

12.10.3 Do More Yourself

Create *K*-means models to solve the problems presented by the following datasets:

- https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities
- https://archive.ics.uci.edu/ml/datasets/Health+News+in+Twitter
- https://archive.ics.uci.edu/ml/datasets/YouTube+Multiview+Video+Games+Dataset

References

1. A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (O'Reilly Media, Sebastopol, CA, 2019)
2. Y. Liu, *Python Machine Learning by Example: Build Intelligent Systems Using Python, TensorFlow 2, PyTorch, and Scikit-Learn, 3rd Edition (Kindle Edition)* (Packt, 2020)
3. M. Gopal, *Applied Machine Learning* (McGraw-Hill Education, New York, 2018)
4. S.P. Lloyd, Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–136 (1982). https://doi.org/10.1109/TIT.1982.1056489
5. C. Elkan, Using the triangle inequality to accelerate k-means. Presented at the Proceedings of the Twentieth International Conference on International Conference on Machine Learning, Washington, DC, USA (2003)
6. D. Sculley, Web-scale k-means clustering. Presented at the Proceedings of the 19th International Conference on World Wide Web, Raleigh, North Carolina, USA (2010) [Online]. Available: https://doi.org/10.1145/1772690.1772862