CME 466: Design of a Digital System

Winter 2024

Machine Learning – Clustering [part 1]

KHAN A. WAHID, PhD, PEng, SMIEEE

Professor
Electrical and Computer Engineering
University of Saskatchewan
Email: khan.wahid@usask.ca

1. Clustering in Unsupervised Learning

- Clustering is a form of unsupervised learning, where the labels are not available. The goal is to group the data into meaningful clusters. This technique is important in data mining.

- Each clustering algorithm in scikit-learn comes in two variants:
  - ***a class***, that implements the *fit* method to learn the clusters on the dataset, and
  - ***a function***, that returns an array of integer labels corresponding to the different clusters[1].

- For the class, the labels over the training data can be found in the *labels_* attribute. Unlike classification, there is no model training (or fitting) stage in clustering.

1.1 K-means clustering algorithm

- The K-means algorithm clusters the data by trying to separate samples in *n* groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. The center of the cluster is called centroids.

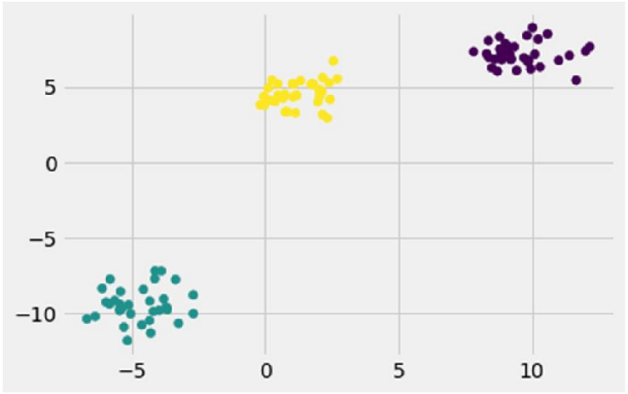
- This algorithm requires the number of clusters to be specified at the start. It scales well to large numbers of samples and has been used across a large range of applications in many different fields.

- We will generate synthetic isotropic data with known labels for this experiment.

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

[1] https://scikit-learn.org/stable/modules/clustering.html

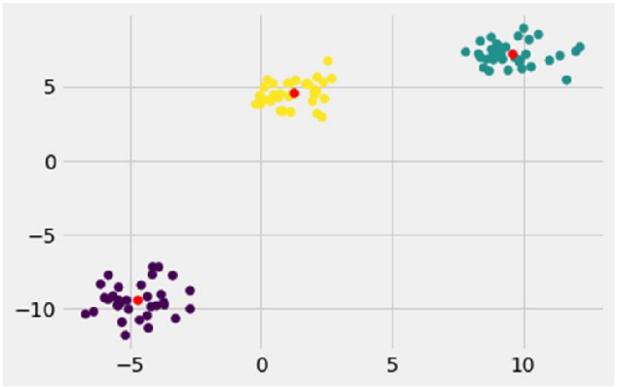
```
import pandas as pd
import seaborn as sns
from sklearn import metrics

from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=100, centers=3,
n_features=2, random_state=813)
print(X.data.shape, y.data.shape)
plt.scatter(X[:, 0], X[:, 1], c=y);
```



- Now we assume that we do not know how many classes are there in the dataset, and will use K-means algorithm to predict or group the datasets. Let us see what happens.

- Remember, the algorithm requires the number of clusters to be specified at the start. So, we use *n_clusters = 3*.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
cluster_labels = kmeans.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels);

print(cluster_labels)
cc = kmeans.cluster_centers_
print(cc)
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels);
plt.scatter(cc[:,0], cc[:, 1], c='r', marker='+');
```



- As we can see from the results, the K-means algorithm successfully grouped the datasets into three classes with the three centroids marked in red.

1.2 Evaluation metrics in clustering algorithm

- For supervised learning, such as regression or classification, where the model predicts some outcome, we can easily infer some form of accuracy, confusion matrix or other metrics.

- However, in a clustering algorithm, we do not aim for any target. Therefore, we cannot use previously discussed metrics and need to look for other types of measurement for performance evaluation.

```
print(metrics.accuracy_score(y, cluster_labels))
print(metrics.confusion_matrix(y,cluster_labels))
```

- The most common measurement for cluster performance is the distinctness or uniqueness of the clusters created. This is because the goal for clustering is to create clusters that are as unique as possible. We will use two methods:

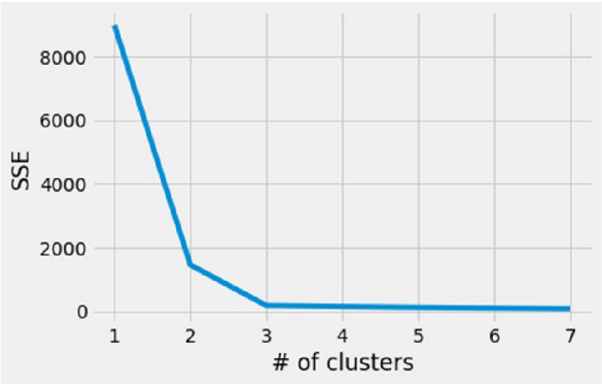
    - **Elbow method:** It is a heuristic method used in determining the number of clusters in a data set. It uses a graphical representation to find the optimal 'K' in a K-means clustering algorithm.

    - It starts by finding the sum of the square error (SSE) of the distance between points in a cluster and the cluster centroid. It then uses the "elbow" or "knee of a curve" as a cutoff point, which determines the optimal value of ‘K’.

- **Silhouette score[2]:** It is the mean Silhouette Coefficient[3] for all clusters, which is calculated using the mean intra-cluster distance and the mean nearest-cluster distance. This score is between -1 and 1, where the higher the score the more unique and distinct the clusters are.

- A score of near +1 indicates that the sample is far away from the neighboring clusters.
- A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters (i.e., overlapping clusters).
- A negative value indicates that those samples might have been assigned to the wrong cluster.

```
# The number of iterations required to converge
kmeans.n_iter_

# lowest SSE sum of squared error value
kmeans.inertia_

# sum of squared error (SSE) values for each k
sse = []
for k in range(1, 8):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)

plt.plot(range(1, 8), sse)
plt.xticks(range(1, 8))
plt.xlabel("# of clusters")
```



[2] https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
[3] https://www.educative.io/answers/what-is-silhouette-score

```
plt.ylabel("SSE")
from sklearn.metrics import silhouette_score
silhouette_score(X, kmeans.labels_, metric='euclidean')
```

1.3 Clustering algorithm using an unknown dataset

- In this section, we will experiment with an unknown dataset and try to understand it better. Let us use the "Palmer Penguins" dataset[4]. Size measurements, clutch observations, and blood isotope ratios for 344 adult penguins were observed on islands near Palmer Station, Antarctica and reported here.

- Download the "penguins.xls" file from the course site. Read the dataset in pandas dataframe. Do not look for the number of species. Since we do not know how many penguin species are there in the dataset, we will use clustering algorithm to figure it out.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import metrics

# Create a dataframe
peng = pd.read_csv('penguins.xls')
peng.head()

peng.info()
print(peng.shape)
```

[4] https://www.kaggle.com/code/parulpandey/penguin-dataset-the-new-iris

1.4 Pre-processing of the dataset (imputing or filling in missing values)

- The problem of using a real dataset is that they are often not well curated. As a result, there could be missing parameters (or entries) in the dataset, which will cause program errors during fitting into a clustering or even classification algorithm. Therefore, we will need to pre-process the dataset.

- Let us first look for missing entries and fill them in with similar entries. It is known as **imputation**[5]. You can impute by a constant value (like 0), the mean value of each feature, a kNN prediction, or using an iterative process. We will use "mean" in this example for imputation.

```
# check for missing values
peng.isna().sum()
          species              0
          island               0
          bill_length_mm       2
          bill_depth_mm        2
          flipper_length_mm    2
          body_mass_g          2
          sex                 11
```

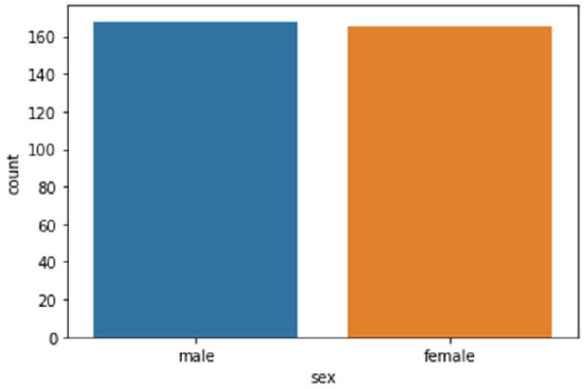
```
# filling NaN
peng["bill_length_mm"] = peng["bill_length_mm"].fillna(value =
peng["bill_length_mm"].mean())
peng["bill_depth_mm"] = peng["bill_depth_mm"].fillna(value =
peng["bill_depth_mm"].mean())
peng["flipper_length_mm"] = peng["flipper_length_mm"].fillna(value =
peng["flipper_length_mm"].mean())
peng["body_mass_g"] = peng["body_mass_g"].fillna(value =
peng["body_mass_g"].mean())
```

[5] https://scikit-learn.org/stable/auto_examples/impute/plot_missing_values.html#sphx-glr-auto-examples-impute-plot-missing-values-py

- For numerical features like, bill_length_mm, bill_depth_mm, and so on, it would be easier to fill them with the mean value. However, for non-numeric entries, like sex, we can first see whether there is any imbalance and then fill in the missing entries with the lesser type to make the dataset more balanced.

```
#check sex of penguin
sns.countplot(x = "sex", data = peng)
peng['sex'].value_counts()

peng["sex"] = peng["sex"].fillna("female")
peng.isna().sum()
```

```
species              0
island               0
bill_length_mm       0
bill_depth_mm        0
flipper_length_mm    0
body_mass_g          0
sex                  0
```

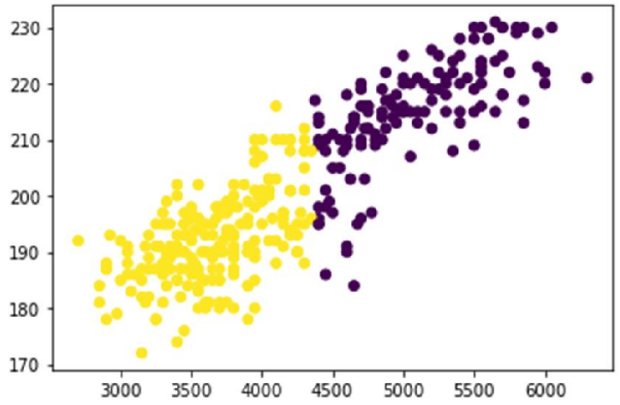


- We can now see that the dataset does not have any missing entries, and therefore is ready to be used for fitting.

- We will use numeric features only from the dataset and apply K-means algorithm with *n_clusters* = 2.

```
# convert df to numpy
Xori = peng.to_numpy()
print(Xori.shape)
# use numeric features for clustering
X = Xori [:, [2,3,4,5]]
y = peng.species
print(X.shape, y.shape)
```

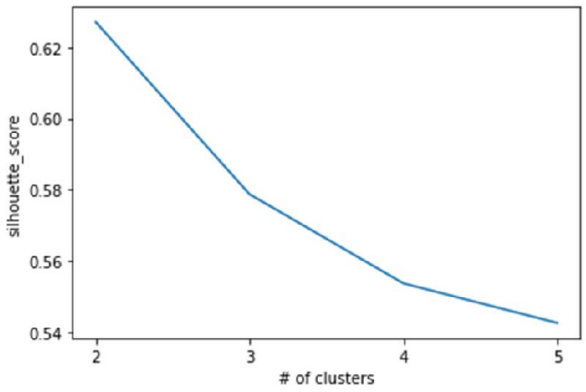
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
cluster_labels = kmeans.predict(X)
plt.scatter(X[:, 3], X[:, 2], c=cluster_labels);

from sklearn.metrics import silhouette_score
silhouette_score(X, kmeans.labels_,
metric='euclidean')
```



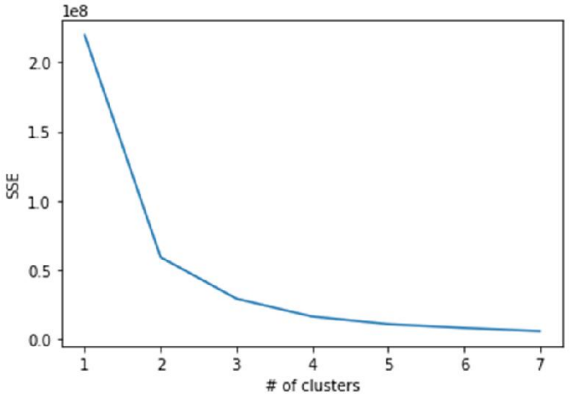
- As stated before, the weakness of K-means clustering is that we do not know how many clusters we need by just running the model. We need to test ranges of values and make a decision on the best value of *k*. We typically make a decision using the Elbow method or silhouette score to determine the optimal number of clusters, where we are not overfitting the data with too many clusters, and neither underfitting with too few.

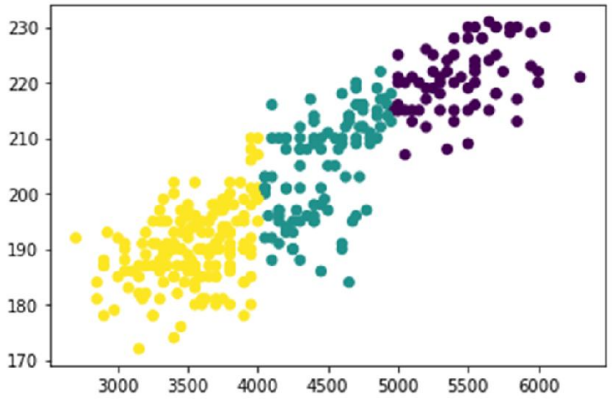
- We create the below loop to test and store different model results so that we can make a decision on the best number of clusters.

```
# find silhouette score for each k
sse = []
for k in range(2, 6):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X)
    sc = silhouette_score(X, kmeans.labels_,
metric='euclidean')
    sse.append(sc)

plt.plot(range(2, 6), sse)
```



```
plt.xticks(range(2, 6))
plt.xlabel("# of clusters")
plt.ylabel("silhouette_score")
plt.show()

# sum of squared error (SSE) values for each k
sse = []
for k in range(1, 8):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)

plt.plot(range(1, 8), sse)
plt.xticks(range(1, 8))
plt.xlabel("# of clusters")
plt.ylabel("SSE")
plt.show()
```



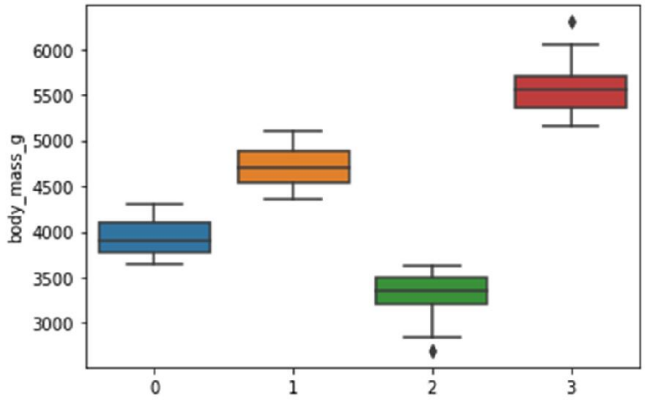


- What conclusion can we draw from the analysis? Are there 2, 3 or 4 species?
- It is often difficult to conclusively determine how many classes (or species) are there. There is another technique we can apply.

- We can plot the cluster labels (determined by the K-means algorithm) with features like body mass, bill length or others, and investigate if we can observe any visible differences that will assist us determine the right value of *k*.

```
sns.boxplot(x = kmeans.labels_, y = peng['body_mass_g']);
sns.boxplot(x = kmeans.labels_, y = peng['bill_length_mm']);
```

- Finally, it can be reasonably determined that there are three species in the dataset. We can now use the confusion matrix to see how well the clustering algorithm performed in grouping the data points.




```
# plot confusion matirx to check K-Means model vs the actual species
conf = pd.DataFrame({'labels': cluster_labels, 'species': y})
cm = pd.crosstab(conf['labels'], conf['species'])
print(cm)
```

species	Adelie	Chinstrap	Gentoo
labels			
0	0	0	68
1	36	15	55
2	116	53	1

- We can see that, species “Adelie” is somewhat well clustered, however, the other two are poorly grouped. The question in now, how can we improve the performance?

1.5 Pre-processing of the dataset (using Scaling)

- There are many scaling algorithms in scikit-learn. We will use one of them, called *StandardScaler*[6]. It standardizes the data by removing the mean and scaling to unit variance. As a result, the range of the feature values is shrunk.

$$X_{scaled} = \frac{X - \mu}{\sigma}$$

[6] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

- After StandardScaler is applied, the mean of the dataset becomes 0 and the standard deviation becomes 1. In the link below, you can find the effect of different scalers on data using scikit-lrearn.

https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py

```
# testing different datasets
X = [ [90, 85, 82, 59, 39], [41, 40, 65, 79, 83]]
print(X)
#use StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
print(X)
print(scaler.mean_)
print(scaler.var_)
```

- Apply the standard scaler on the penguin dataset and see if the performance improves or not. It is seen from the scatter plot that the dataset is normalized.

- Even though the silhouette_score is lowered (0.4438398792000198), the confusion matrix confirms that, the all species are somewhat well grouped. Therefore, scaling the dataset did help in clustering it.

species	Adelie	Chinstrap	Gentoo
labels			
0	0	0	123
1	25	63	1
2	127	5	0