13 - Clustering

Data Science Practicum 2021/22, Lesson 13

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Machine Learning

Unsupervised Learning

Exercises

Assignment

References

Machine Learning

Unsupervised Learning

- Data-driven
- Uses unlabeled data
- Goal: Structure recognition
- Types
  - Clustering (divide by similarity)
    - E.g. Targeted Marketing
  - Dimensionality Reduction
    - E.g. compacting data

Document	f1	f2	f3
It was the best of times	1	2	1
it was the worst of times	3	3	3
it was the age of wisdom	4	1	2
it was the age of foolishness	2	2	1

Supervised Learning

- Task-driven
- Uses pre-categorized data
- Goal: predictive modeling
- Types
  - Classification: labels are discrete classes
  - Regression: labels are continuous variables

Document	f1	f2	f3	Label
It was the best of times	1	2	1	C
it was the worst of times	3	3	3	S
it was the age of wisdom	4	1	2	C
it was the age of foolishness	2	2	1	?

Table of Contents

Machine Learning

Unsupervised Learning

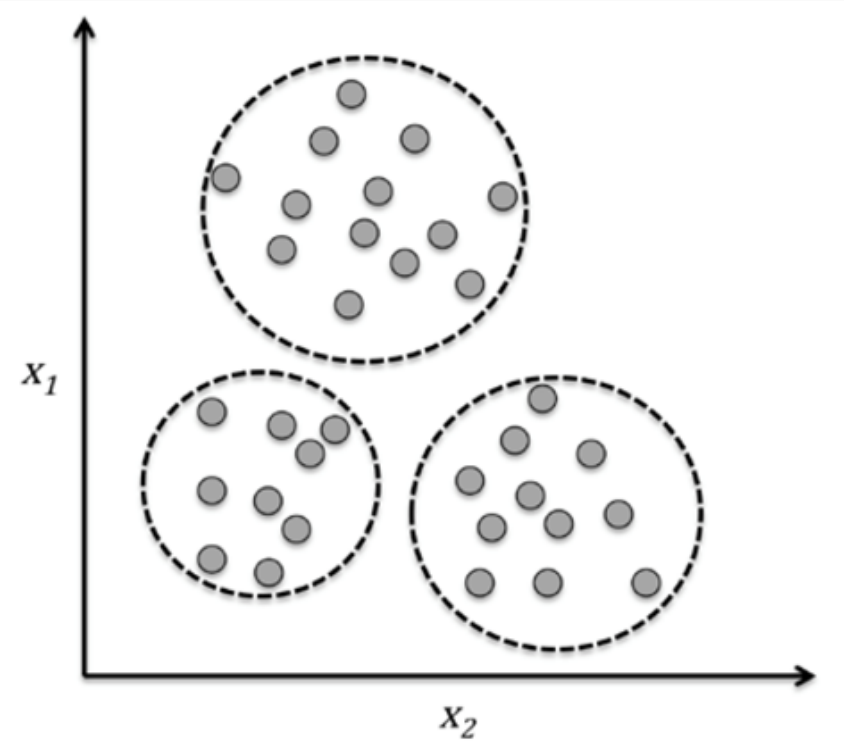
Exercises

Assignment

References

Clustering

- Prototype-based: each cluster is represented by a prototype, which can either be:
  - the centroid (average) of similar points with continuous features
  - the medoid (the most representative or most frequently occurring point) in the case of categorical features.
- Example: K-Means
- Hierarchical



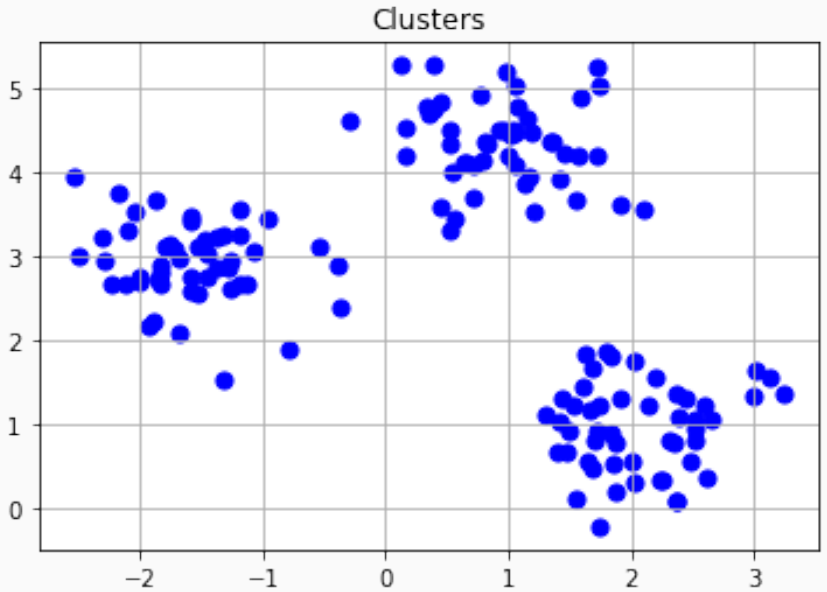
Clusters

- make_blobs: create multiclass datasets by allocating each class one or more normally-distributed clusters of points
  - see also related function make_classification

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt

X, y = make_blobs(n_samples=150,
                  n_features=2,
                  centers=3,
                  cluster_std=0.5,
                  shuffle=True,
                  random_state=0)

plt.scatter(X[:,0],X[:,1],c="blue", marker="o", s=50)
plt.title("Clusters")
plt.grid()
plt.show()
```



K-Means

- Two important terms:
  - Squared Euclidian distance
  - Within-cluster SSE (Sum of Squared Error)

K-Means

- Two important terms:
  - Squared Euclidian distance
  - Within-cluster SSE (Sum of Squared Error)

1. Randomly pick **k** centroids as initial cluster centroids
2. Other samples: assign them to the closest centroid (based on SED)
3. Calculate new centroids for the clusters
4. Repeat 2 and 3 until
   - the assignment of samples does not change
   - a max of iterations has been reached

fit(), transform(), fit_transform() and predict()

- Most objects in sklearn are
  - **Transformers** are for pre-processing before modeling. Examples: The Imputer class (like SimpleImputer for filling in missing values), FeatureSelection classes
  - **Models** are used to make predictions. Examples: Linear Regression model, Decision Tree model, Random Forest model etc.
    - You will usually pre-process your data (with transformers) before putting it in a model.

fit(), transform(), fit_transform() and predict()

- Most objects in sklearn are
  - **Transformers** are for pre-processing before modeling. Examples: The Imputer class (like SimpleImputer for filling in missing values), FeatureSelection classes
  - **Models** are used to make predictions. Examples: Linear Regression model, Decision Tree model, Random Forest model etc.
    - You will usually pre-process your data (with transformers) before putting it in a model.

- Transformers
  - **fit()** - It is used for calculating the initial filling of parameters on the training data (like mean of the column values) and saves them as an internal objects state
  - **transform()** - Use the above calculated values and return modified training data
  - **fit_transform()** - It joins above two steps. Internally, it just calls first fit() and then transform() on the same data.

fit(), transform(), fit_transform() and predict()

- Most objects in sklearn are
  - **Transformers** are for pre-processing before modeling. Examples: The Imputer class (like SimpleImputer for filling in missing values), FeatureSelection classes
  - **Models** are used to make predictions. Examples: Linear Regression model, Decision Tree model, Random Forest model etc.
    - You will usually pre-process your data (with transformers) before putting it in a model.

- Transformers
  - **fit()** - It is used for calculating the initial filling of parameters on the training data (like mean of the column values) and saves them as an internal objects state
  - **transform()** - Use the above calculated values and return modified training data
  - **fit_transform()** - It joins above two steps. Internally, it just calls first fit() and then transform() on the same data.

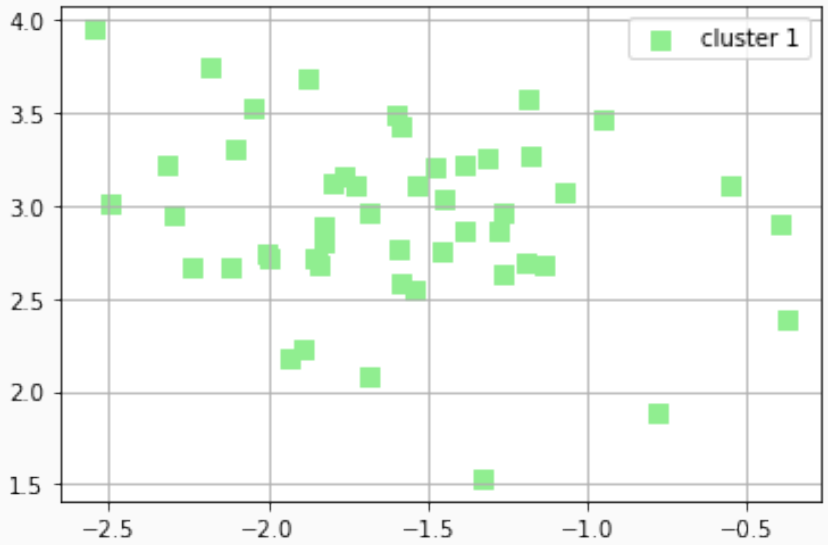
- Models:
  - **fit()** - It calculates the parameters/weights on training data (e.g. parameters returned by coef() in case of Linear Regression) and saves them as an internal objects state.
  - **predict()** - Use the above calculated weights on test data to make the predictions
  - **fit_predict()** - It joins above two steps. Internally, it just calls first fit() and then predict() on the same data
  - **transform()** - Cannot be used
  - **fit_transform()** - Cannot be used

K-Means

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters = 3,
            init ="random",
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)

plt.scatter(X[y_km == 0, 0], X[y_km==0, 1], s=50, c="lightgreen", marker="s", label="cluster 1")
plt.legend()
plt.grid()
plt.show()
```

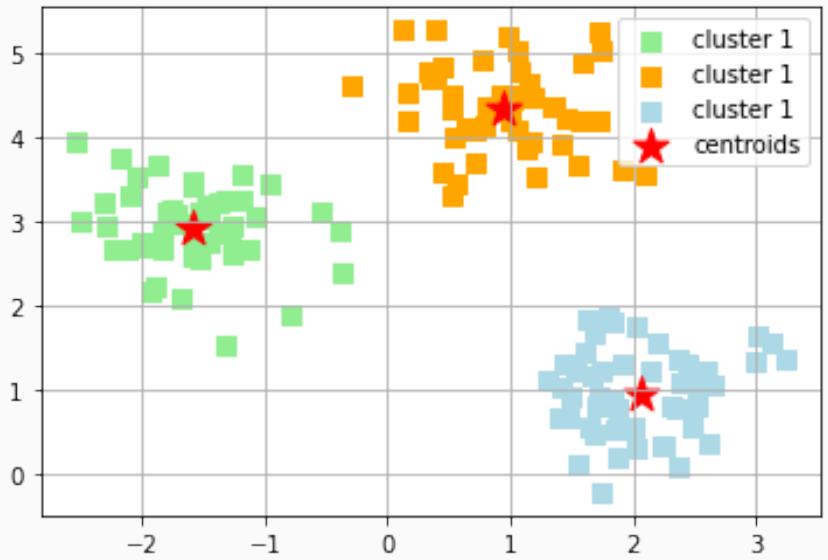


K-Means

```
plt.scatter(X[y_km == 0, 0], X[y_km==0, 1], s=50, c="lightgreen", marker="s", label="cluster 1")
plt.scatter(X[y_km == 1, 0], X[y_km==1, 1], s=50, c="orange", marker="s", label="cluster 1")
plt.scatter(X[y_km == 2, 0], X[y_km==2, 1], s=50, c="lightblue", marker="s", label="cluster 1")

plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1], s=250, marker="*", c="red", label="centroids")

plt.legend()
plt.grid()
plt.show()
```

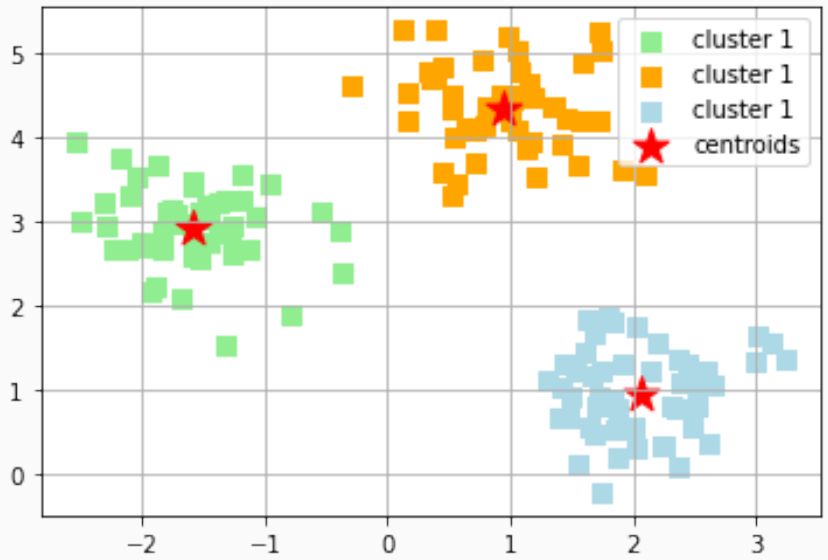


K-Means

```
plt.scatter(X[y_km == 0, 0], X[y_km==0, 1], s=50, c="lightgreen", marker="s", label="cluster 1")
plt.scatter(X[y_km == 1, 0], X[y_km==1, 1], s=50, c="orange", marker="s", label="cluster 1")
plt.scatter(X[y_km == 2, 0], X[y_km==2, 1], s=50, c="lightblue", marker="s", label="cluster 1")

plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1], s=250, marker="*", c="red", label="centroids")

plt.legend()
plt.grid()
plt.show()
```



- within-cluster vs between-cluster variance

How to choose the number of clusters?

- Elbow method:
  - takes into account within-cluster variance only
- Silhouette method:
  - takes into account within-cluster and between-cluster variance

Elbow method

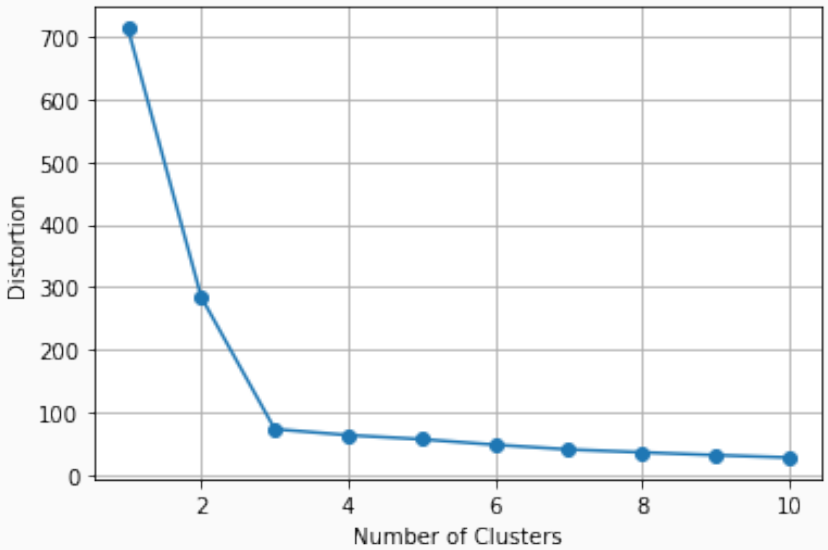
```
distortions = []
for i in range(1,11):
    km = KMeans(n_clusters = i, init ="k-means++", n_init=10, max_iter=300, random_state=0)
    km.fit(X)
    distortions.append(km.inertia_)

distortions
```

```ini
[713.699828943416,
 283.461017802093,
 72.47601670996696,
 62.840617685422224,
 56.288267331024166,
 47.43195201948497,
 39.95659663146229,
 35.322556689201335,
 30.88571211015811,
 26.961804858604854]
```

Elbow method

```
plt.plot(range(1,11),distortions, marker="o")
plt.xlabel("Number of Clusters")
plt.ylabel("Distortion")
plt.grid();
plt.show()
```



- Elbow method is a known heuristic approach in mathematical optimization
- Idea: choose a point where diminishing returns are no longer worth the additional cost

Table of Contents

Machine Learning

Unsupervised Learning

Exercises

Assignment

References

Exercise

- Write the function generate_2D_clusters() that receives the parameters (n_samples,n_centers), generates clusters of 2-dimentional random data, and returns it

Exercise

- Write the function generate_2D_clusters() that receives the parameters (n_samples,n_centers), generates clusters of 2-dimentional random data, and returns it

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt

def gen_2d_cl(n_samples, n_centers):
    n_feat = 2
    X,y = make_blobs(n_samples, n_feat, centers=n_centers, cluster_std=0.5)
    return X,y

X, y= gen_2d_cl( 250,6)
```

Exercise

- Write the function predict_and_plot_2D_clusters() that generates clusters of 2-dimensional random data and plots them together with their centroids

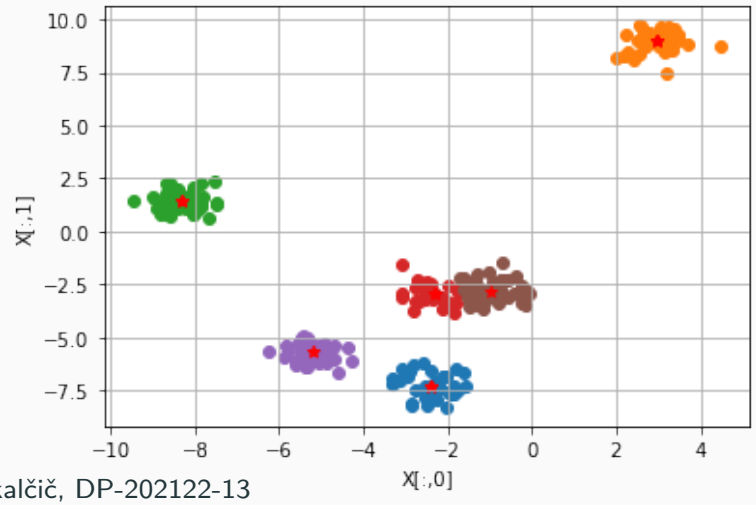
Exercise

- Write the function predict_and_plot_2D_clusters() that generates clusters of 2-dimensional random data and plots them together with their centroids

```
from sklearn.cluster import KMeans

def pred_and_plot(n_samples,n_clusters):
    X, y= gen_2d_cl(n_samples,n_clusters )
    km = KMeans(n_clusters)
    y= km.fit_predict(X)
    for clus in range(n_clusters):
        plt.scatter(X[y == clus,0], X[y == clus,1], label="cluster")
        plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1], label="centroid", marker = "*", color="red")
    plt.xlabel("X[:,0]")
    plt.ylabel("X[:,1]")
    plt.grid()
    plt.show()

pred_and_plot(250,6)
```



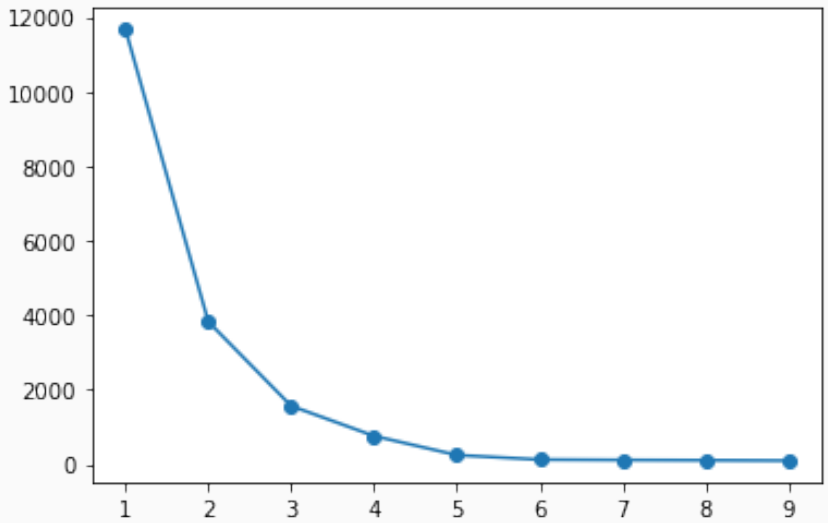
Exercise

- Try the Elbow method on the data from the previous exercises

Exercise

- Try the Elbow method on the data from the previous exercises

```
dist =[]
for i in range(1,10):
    km = KMeans(n_clusters=i, init="k-means++")
    km.fit(X)
    dist.append(km.inertia_)
plt.plot(range(1,10),dist,"o-")
plt.show()
```



Exercise

- load the iris dataset from sklearn

```
from sklearn import cluster, datasets

iris = datasets.load_iris()
X_iris = iris.data
```

- find the number of clusters

Exercise

- load the iris dataset from sklearn

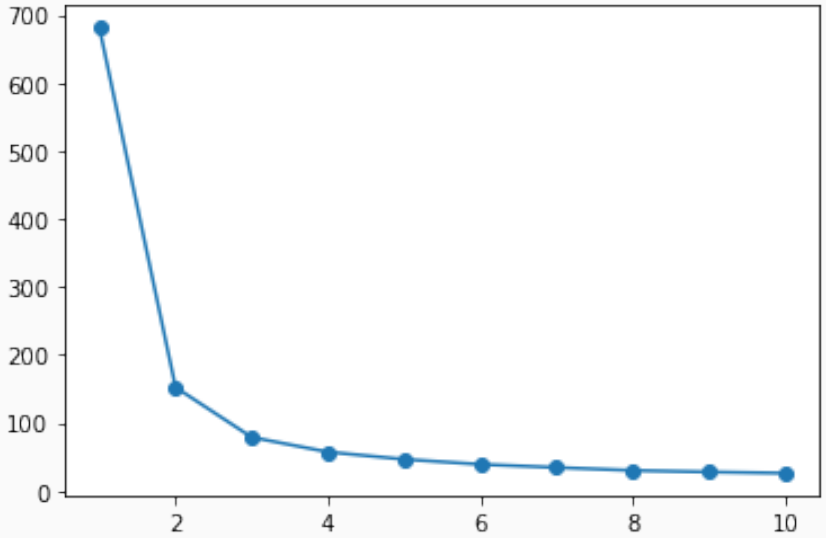
```
from sklearn import cluster, datasets

iris = datasets.load_iris()
X_iris = iris.data
```

- find the number of clusters

```
from matplotlib import pyplot as plt

dist = []
for i in range(1,11):
    km = cluster.KMeans(n_clusters=i, init="k-means++")
    km.fit(X_iris)
    dist.append(km.inertia_)
plt.plot(range(1,11),dist,marker="o")
```



Exercise

- fit the clustering to nClusters $= 3$
- test new items in which cluster they fit

```
import numpy as np
unseen_data = np.array([[5.5, 2.5, 4.5, 1.5],
                        [7.0, 3.0, 6.0, 2.0]])
```

Exercise

- fit the clustering to nClusters $= 3$
- test new items in which cluster they fit

```
import numpy as np
unseen_data = np.array([[5.5, 2.5, 4.5, 1.5],
                        [7.0, 3.0, 6.0, 2.0]])
```

```
k_means = cluster.KMeans(n_clusters=3)
k_means.fit(X_iris)
prediction = k_means.predict(unseen_data)
print("Data point", unseen_data[0], " cluster:", prediction[0])
print("Data point", unseen_data[1], " cluster:", prediction[1])
```

```
Data point [5.5 2.5 4.5 1.5]  cluster: 1
Data point [7. 3. 6. 2.]  cluster: 2
```

Exercise

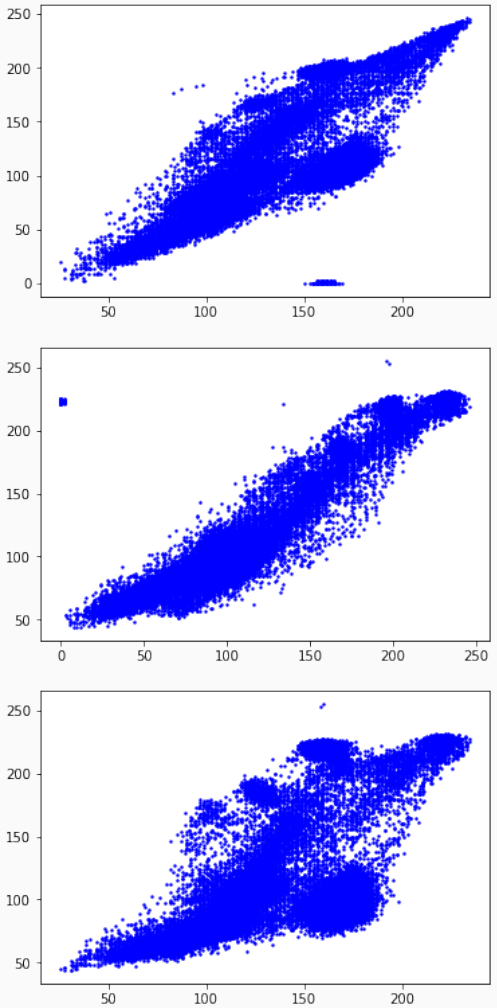
- read the dataset `housec8.txt`
- visualize with scatterplots

Exercise

- read the dataset housec8.txt
- visualize with scatterplots

```
import pandas as pd
from matplotlib import pyplot as plt

rgb = pd.read_csv("housec8.txt")
# visualize with scatterplots
plt.scatter(rgb.iloc[:,0], rgb.iloc[:,1], c = "blue", marker = 'o', s = 2)
plt.show()
plt.scatter(rgb.iloc[:,1], rgb.iloc[:,2], c = "blue", marker = 'o', s = 2)
plt.show()
plt.scatter(rgb.iloc[:,0], rgb.iloc[:,2], c = "blue", marker = 'o', s = 2)
plt.show()
```



- there are three columns
- hence we need three scatterplots (one per pair of variables)

Exercise

- use the elbow method to identify the optimal number of clusters

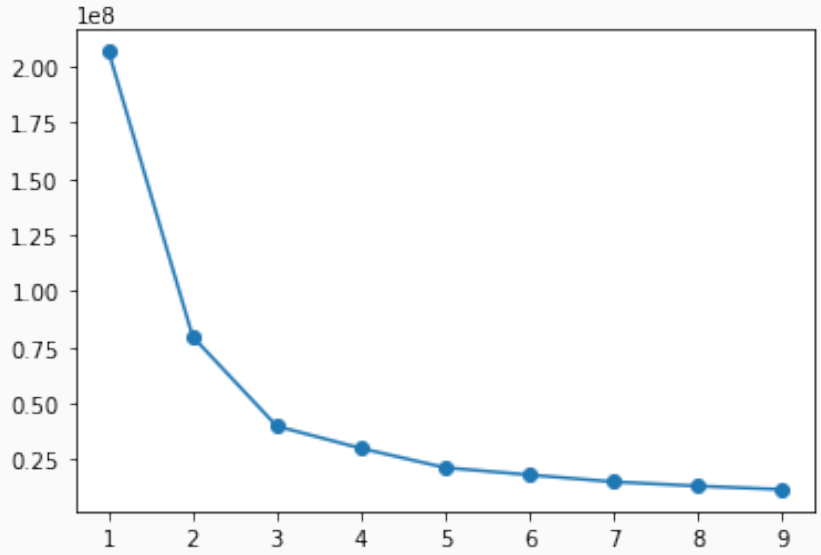
Exercise

- use the elbow method to identify the optimal number of clusters

```
from sklearn import cluster
import numpy as np

dist =[]
for i in range(1,10):
    km = cluster.KMeans(n_clusters=i, init="k-means++")
    km.fit(rgb)
    dist.append(km.inertia_)
plt.plot(range(1,10),dist,"o-")
plt.show()
```

- how many clusters?



Exercise

- choose num of clusters
- perform the clustering

Exercise

- choose num of clusters
- perform the clustering

```
km = cluster.KMeans(n_clusters=5, init="k-means++")
rgb_c = km.fit_predict(rgb.iloc[:,[0,2]])
```

- explore the following parameters of `km`:
  - `km.labels_`
  - `print(km.cluster_centers_)`
  - `print(km.inertia_)` Sum of squared distances of samples to their closest cluster center.
  - the return value of the `km.fit_predict`

Exercise

- choose num of clusters
- perform the clustering

```
km = cluster.KMeans(n_clusters=5, init="k-means++")
rgb_c = km.fit_predict(rgb.iloc[:,[0,2]])
```

- explore the following parameters of `km`:
  - `km.labels_`
  - `print(km.cluster_centers_)`
  - `print(km.inertia_)` Sum of squared distances of samples to their closest cluster center.
  - the return value of the `km.fit_predict`

```
print(km.labels_)
print(km.cluster_centers_)
print(km.inertia_)
print(rgb_c)
```

```
[3 3 3 ... 1 1 1]
[[117.88697819  99.40548287]
 [182.51250788 212.78599958]
 [166.37910314  97.82941704]
 [ 86.46341118  69.74734607]
 [137.59743178 161.18041734]]
12612610.237390246
[3 3 3 ... 1 1 1]
```

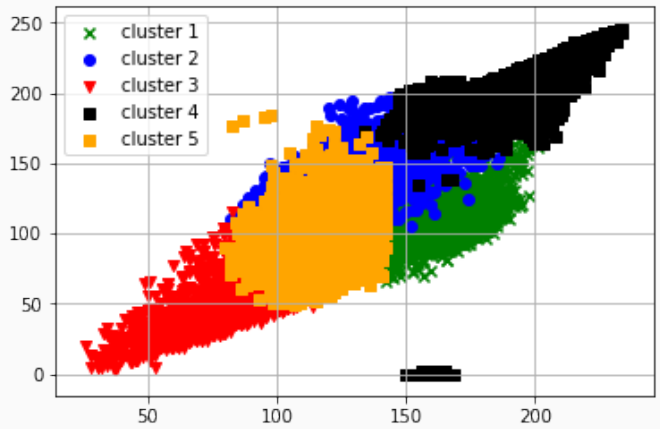
Exercise

- visualize the clusters with three scatterplots:
  - R-G
  - R-B
  - G-B
- use different colors for the clusters

Exercise

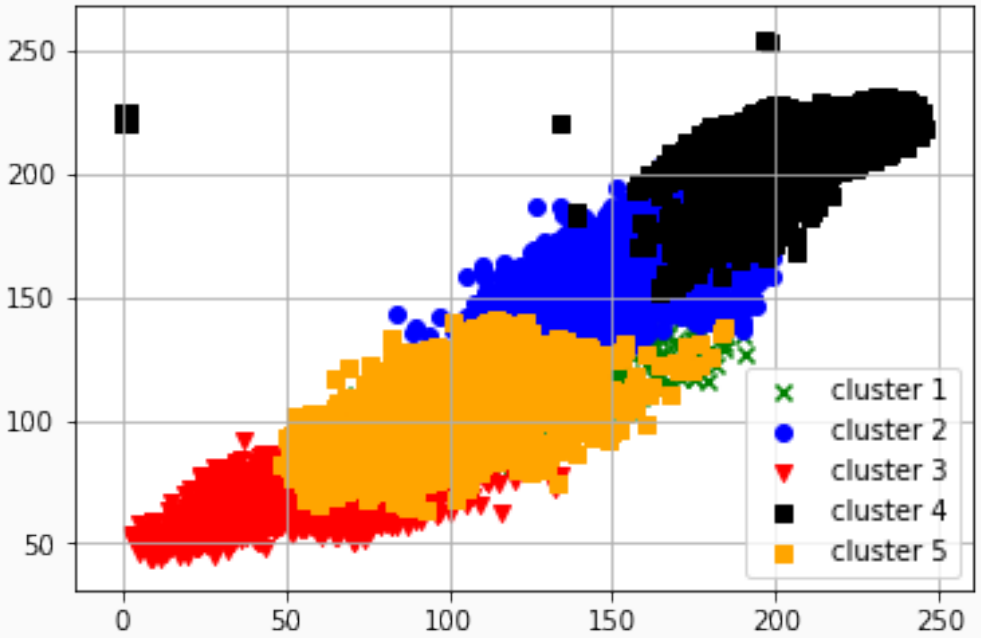
- visualize the clusters with three scatterplots:
  - R-G
  - R-B
  - G-B
- use different colors for the clusters

```
plt.scatter(rgb.iloc[rgb_c ==0, 0],rgb.iloc[rgb_c ==0, 1], c = "green", marker = "x", label = "cluster 1")
plt.scatter(rgb.iloc[rgb_c ==1, 0],rgb.iloc[rgb_c ==1, 1], c = "blue", marker = "o", label = "cluster 2")
plt.scatter(rgb.iloc[rgb_c ==2, 0],rgb.iloc[rgb_c ==2, 1], c = "red", marker = "v", label = "cluster 3")
plt.scatter(rgb.iloc[rgb_c ==3, 0],rgb.iloc[rgb_c ==3, 1], c = "black", marker = "s", label = "cluster 4")
plt.scatter(rgb.iloc[rgb_c ==4, 0],rgb.iloc[rgb_c ==4, 1], c = "orange", marker = "s", label = "cluster 5")
plt.legend()
plt.grid()
plt.show()
```



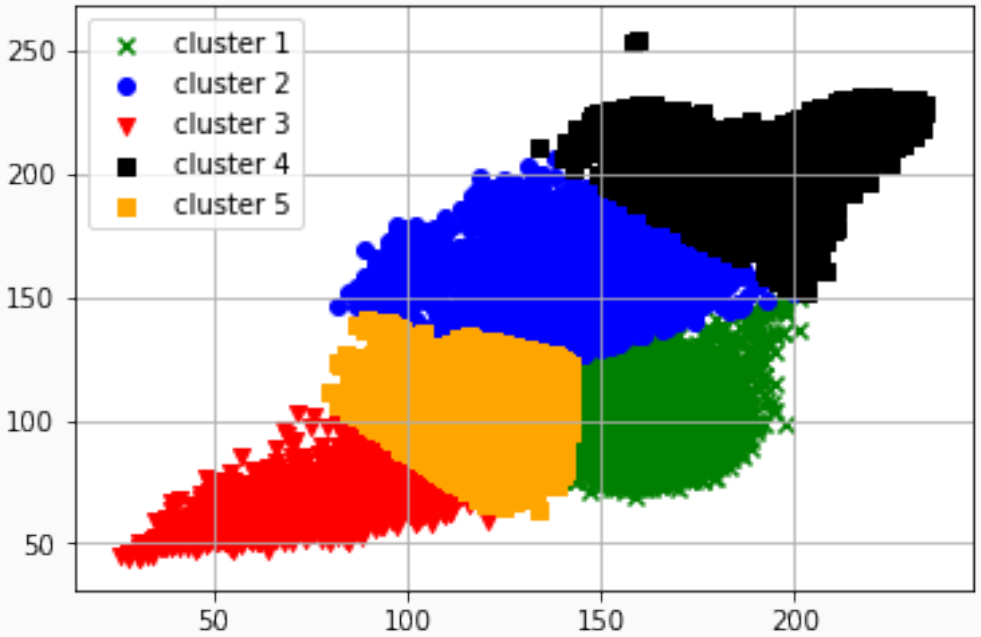
Exercise

```
plt.scatter(rgb.iloc[rgb_c ==0, 1],rgb.iloc[rgb_c ==0, 2], c = "green", marker = "x", label = "cluster 1")
plt.scatter(rgb.iloc[rgb_c ==1, 1],rgb.iloc[rgb_c ==1, 2], c = "blue", marker = "o", label = "cluster 2")
plt.scatter(rgb.iloc[rgb_c ==2, 1],rgb.iloc[rgb_c ==2, 2], c = "red", marker = "v", label = "cluster 3")
plt.scatter(rgb.iloc[rgb_c ==3, 1],rgb.iloc[rgb_c ==3, 2], c = "black", marker = "s", label = "cluster 4")
plt.scatter(rgb.iloc[rgb_c ==4, 1],rgb.iloc[rgb_c ==4, 2], c = "orange", marker = "s", label = "cluster 5")
plt.legend()
plt.grid()
plt.show()
```



Exercise

```
plt.scatter(rgb.iloc[rgb_c ==0, 0],rgb.iloc[rgb_c ==0, 2], c = "green", marker = "x", label = "cluster 1")
plt.scatter(rgb.iloc[rgb_c ==1, 0],rgb.iloc[rgb_c ==1, 2], c = "blue", marker = "o", label = "cluster 2")
plt.scatter(rgb.iloc[rgb_c ==2, 0],rgb.iloc[rgb_c ==2, 2], c = "red", marker = "v", label = "cluster 3")
plt.scatter(rgb.iloc[rgb_c ==3, 0],rgb.iloc[rgb_c ==3, 2], c = "black", marker = "s", label = "cluster 4")
plt.scatter(rgb.iloc[rgb_c ==4, 0],rgb.iloc[rgb_c ==4, 2], c = "orange", marker = "s", label = "cluster 5")
plt.legend()
plt.grid()
plt.show()
```



Exercise

- there are many datapoints in this dataset
- use a smaller subset

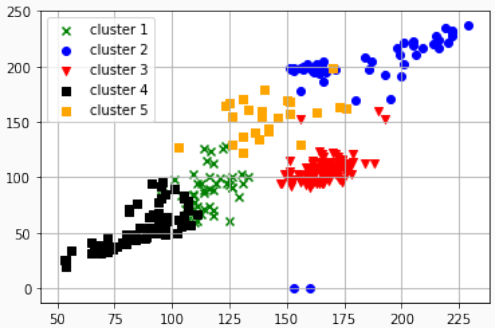
```
idx = np.random.choice(np.arange(len(rgb)), num_of_samples)
```

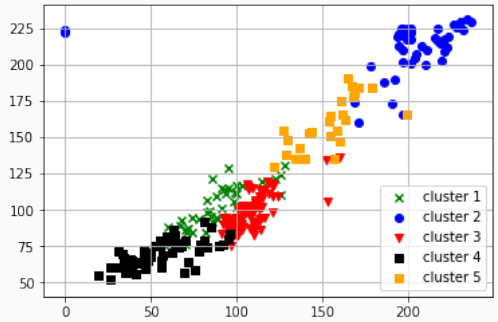
Exercise

- there are many datapoints in this dataset
- use a smaller subset

```
idx = np.random.choice(np.arange(len(rgb)), num_of_samples)
```

```
idx = np.random.choice(np.arange(len(rgb)), 300)
rgb = rgb.iloc[idx,:]
rgb_c = rgb_c[idx]
```





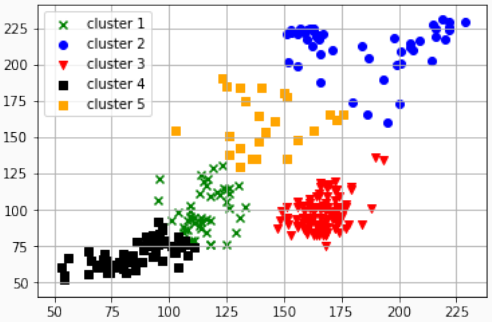


Table of Contents

Machine Learning

Unsupervised Learning

Exercises

Assignment

References

Assignment

- dataset: face.csv
- perform and visualize clustering
  - initial inspection (scatterplots)
  - choice of number of clusters (elbow method)
  - visualize clusters only with the second and third variable

Table of Contents

Machine Learning

Unsupervised Learning

Exercises

Assignment

References

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://medium.com/@dkatzman_3920/supervised-vs-unsupervised-learning-and-use-cases-for-each-8b9cc3ebd301
- https://www.linkedin.com/pulse/difference-between-fit-transform-fittransform-predict-aman-agarwal/