Wprowadzenie do programowania w języku Python

Klasteryzacja danych

Inżynieria i Analiza Danych

Bartłomiej Draguła
Adrian Dereń
2023-01-22

1. Temat projektu

Klasteryzacja to metoda uczenia maszynowego polegająca na dzieleniu danych na grupy (klastry) na podstawie ich podobieństwa. Celem klasteryzacji jest znalezienie naturalnych grup w danych, tak aby elementy w danej grupie były jak najbardziej podobne do siebie, a elementy z różnych grup były jak najbardziej różne od siebie. Klasteryzacja jest używana w wielu dziedzinach, takich jak marketing, biologia, medycyna czy jak w naszym przypadku – informatyka.

2. Metody klasteryzacji

Klasteryzację możemy wykonać na wiele sposobów, korzystając z różnych metod. Poniżej opiszemy kilka najważniejszych z nich:

K-means - to jedna z najczęściej używanych metod klasteryzacji. Polega ona na określeniu K centroidów, które reprezentują klastry, a następnie przypisywaniu każdego punktu danych do najbliższego centroidu.

Hierarchiczna - ta metoda polega na tworzeniu hierarchii klastrów poprzez ich agregację lub dzielenie.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) - ta metoda polega na klasteryzacji punktów danych na podstawie ich gęstości. Klastry są definiowane jako obszary o wysokiej gęstości punktów, a punkty poza tymi obszarami są uznawane za szum.

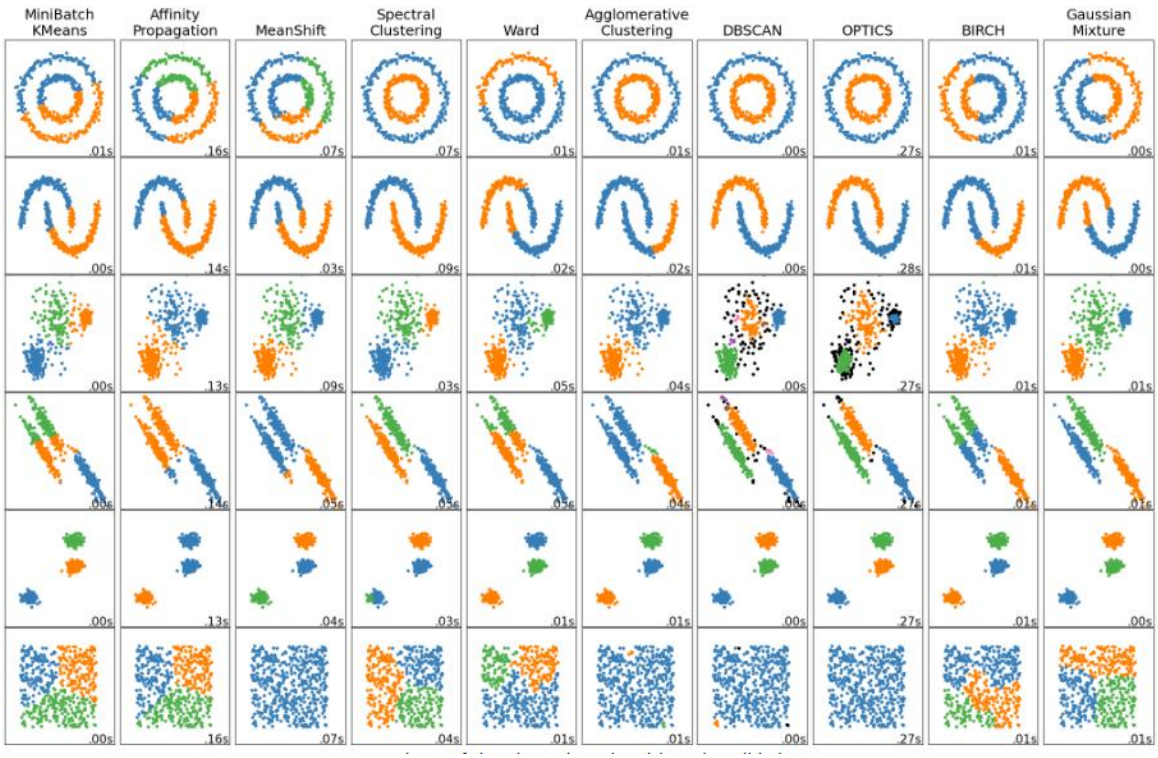
GMM (Gaussian Mixture Model) - ta metoda polega na modelowaniu klastrów jako rozkładów Gaussa, co pozwala na uwzględnienie nieregularności kształtu klastrów.

Affinity Propagation - ta metoda polega na wymianie informacji między punktami danych, aby znaleźć reprezentantów klastrów (tak zwanych "ekspertów").

Spectral Clustering - ta metoda polega na tworzeniu macierzy podobieństwa między punktami danych, a następnie redukcji jej do niższej wymiarowości i klasteryzacji w tej przestrzeni.

Te metody są różne w zależności od danych i celów klasteryzacji i nie ma jednego uniwersalnego rozwiązania, które będzie działać dla każdego przypadku.

Podstawowe metody klasteryzacji dla różnych zbiorów danych:



Wybór odpowiedniej metody klasteryzacji zależy od kilku czynników, takich jak:

Rodzaj danych - niektóre metody działają lepiej z danymi numerycznymi, inne z danymi tekstowymi lub obrazami.

Liczba klastrów - niektóre metody wymagają podania liczby klastrów na początku, podczas gdy inne same ją określają.

Kształt klastrów - niektóre metody lepiej radzą sobie z klastrami o regularnym kształcie, inne z klastrami o nieregularnym kształcie.

Skalowalność - niektóre metody radzą sobie lepiej z dużymi ilościami danych, inne z małymi ilościami.

Zależności między punktami - niektóre metody uwzględniają zależności między punktami, inne nie.

Wymiarowość danych - niektóre metody radzą sobie lepiej z wysokowymiarowymi danymi, inne z niskowymiarowymi.

Wymagane założenia - niektóre metody wymagają założeń, np. metoda K-means wymaga założenia, że klastry są kuliste, metoda GMM zakłada, że dane pochodzą z rozkładów normalnych

Dobranie odpowiedniej metody klasteryzacji wymaga dokładnego zrozumienia danych oraz celów klasteryzacji, a czasami jest to proces eksperymentalny. Ważne jest aby przetestować kilka różnych metod i porównać ich wyniki, aby wybrać najlepszą dla konkretnych danych i oczekiwanych celów.

3. Klasteryzacja przy użyciu języka Python

W języku Python istnieje kilka bibliotek, które udostępniają różne metody klasteryzacji:

scikit-learn - jest to popularna biblioteka uczenia maszynowego w Pythonie, która zawiera implementacje wielu popularnych metod klasteryzacji, takich jak K-means, Hierarchiczna, DBSCAN, GMM, Spectral Clustering, Affinity Propagation.

Pycluster - ta biblioteka zawiera implementacje metod klasteryzacji, takich jak K-means, Hierarchiczna, DBSCAN.

Clusterpy - ta biblioteka zawiera implementacje metod klasteryzacji, takich jak K-means, Hierarchiczna, DBSCAN.

fastcluster - ta biblioteka zawiera szybkie implementacje metod klasteryzacji hierarchicznej.

HDBSCAN - ta biblioteka jest implementacją DBSCAN, który jest skalowalny i pozwala na tworzenie klastrów o różnych rozmiarach i kształtach.

hdbscan - ta biblioteka jest implementacją metody HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) pozwala ona na tworzenie nieregularnych klastrów.

OPTICS - ta biblioteka jest implementacją metody OPTICS (Ordering Points to Identify the Clustering Structure) pozwala ona na tworzenie klastrów o różnych rozmiarach i kształtach.

Użycie tych bibliotek jest łatwe, ponieważ zwykle wymagają jedynie podania danych i parametrów dla wybranej metody klasteryzacji.

W każdej z tych bibliotek, metody klasteryzacji są dostępne jako obiekty, które należy utworzyć, a następnie użyć metod fit i predict.

4. Cel pracy

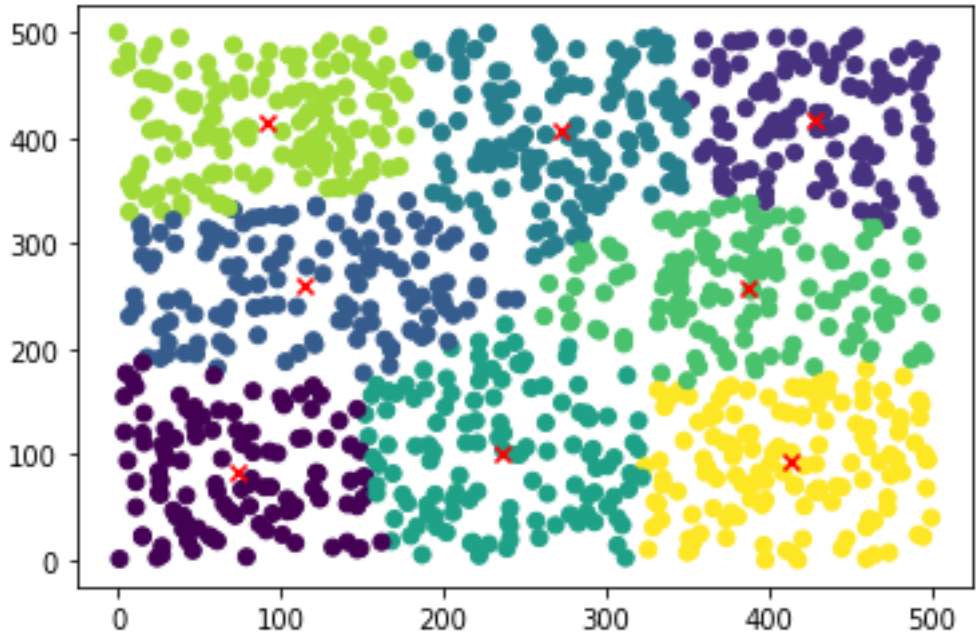
Naszym zadaniem było wygenerowanie losowych punktów oraz centroidów, będących środkowymi punktami każdego z klastrów. Punkty reprezentować mają urządzenia z dostępem do Internetu, natomiast punkty centralne odpowiadają routerom. Następnie naszym celem było sklasteryzowanie utworzonych punktów na podstawie różnych parametrów.

5. Klasteryzacja za pomocą metody K-Means

W pierwszym kroku, po zapoznaniu się z działaniem metod klasteryzacji oraz ich składnią przystąpiliśmy do pracy. Wygenerowaliśmy tysiąc punktów o losowych współrzędnych x oraz y, znajdujących się w przedziale od 0 do 500. Następnie zapisaliśmy je w uprzednio utworzonej pustej liście. Zaimportowaliśmy bibliotekę, potrzebną do klasteryzacji metodą K-Means. Następnie przy jej wykorzystaniu podzieliliśmy punkty na osiem klastrów oraz wyznaczyliśmy środki każdego z nich, które zostały zaznaczone za pomocą czerwonego znaku x.

```python
import random

points = []
for i in range(1000):
    x = random.randint(0, 500)
    y = random.randint(0, 500)
    points.append((x, y))

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=8)
kmeans.fit(points)

clusters = kmeans.predict(points)
cluster_centers = kmeans.cluster_centers_

import matplotlib.pyplot as plt

plt.scatter([p[0] for p in points], [p[1] for p in points], c=clusters)

plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x')
plt.show()
```

Otrzymany rezultat:



Jak łatwo można zauważyć, każdy z klastrów zaznaczony jest innym kolorem. Dzięki temu możemy dokładnie odróżniać je od siebie. Ponadto zostały również wyznaczone oraz zaznaczone środki każdego z klastrów. Punkty zostały sklasteryzowane na podstawie odległości, zatem sposobu, który w wielu przypadkach daje naprawdę świetny rezultat.

6. Klasteryzacja na podstawie odległości od najbliższego centroidu

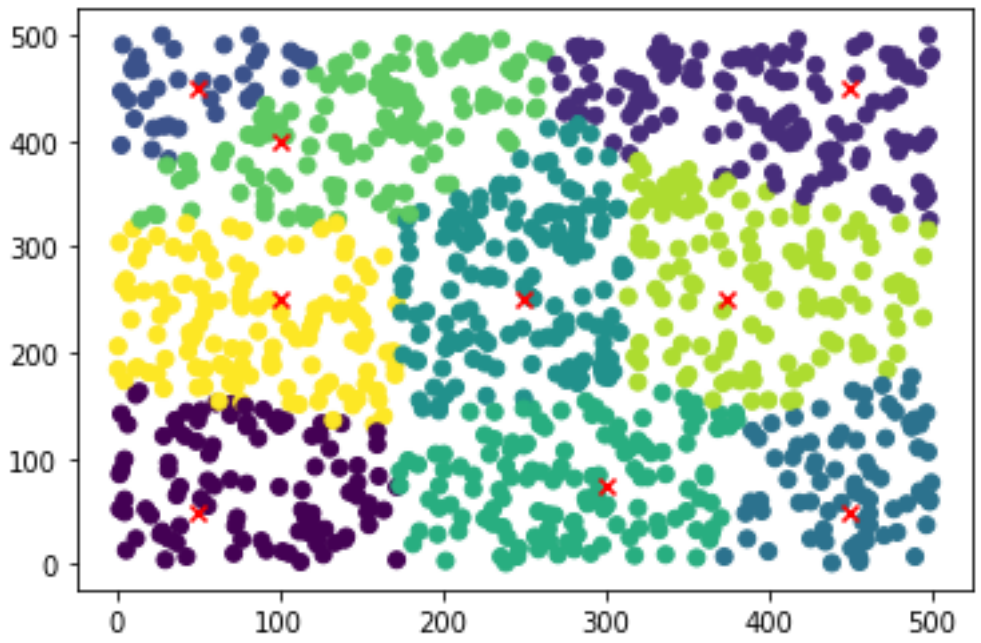
W tym przypadku, nasze zamiary będą bardzo podobne do tych z poprzedniego punktu. Tym razem jednak wystąpią pewne znaczące różnice. Na początku w dokładnie taki sam sposób jak w poprzednim przypadku generujemy tysiąc losowo rozmieszczonych punktów.

```
import random

points = []
for i in range(1000):
    x = random.randint(0, 500)
    y = random.randint(0, 500)
    points.append((x, y))
```

Następnie wyznaczymy środki klastrów – tym razem jednak sami wyznaczamy punkty centralne, przypisując im współrzędne. To właśnie od ilości punktów centralnych zależeć będzie na ile klastrów punkty zostaną podzielone.

```
import numpy as np
from sklearn.metrics import pairwise_distances

center_points = np.array([[50, 50], [450, 450], [50, 450], [450, 50], [250, 250],
                          [300, 75], [100, 400], [375, 250], [100, 250]])
distances = pairwise_distances(points, center_points)
clusters = np.argmin(distances, axis=1)
```

Teraz pozostało tylko przedstawić wszystko na wykresie. Punkty zostały przypisane do klastrów na podstawie odległości, która dzieliła je od punktów centralnych – najbliższy środek centralny określał, w którym klastrze znajdował się będzie punkt.



7. Klasteryzacja na podstawie odległości od najbliższego centroidu oraz parametru prędkość

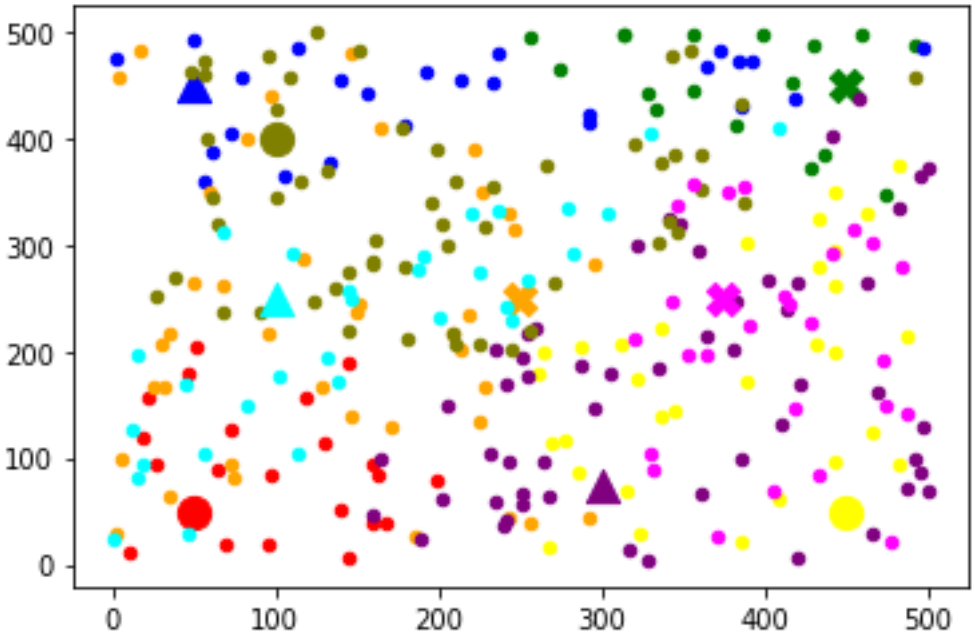
Kolejnym krokiem było sklasteryzowanie punktów, lecz tym razem na podstawie nie tylko odległości. W tym przypadku musieliśmy uwzględnić również dodatkowo przypisany do każdego punktu oraz punktu centralnego parametr. Kod tworzy klasę "Point", która ma atrybuty x, y, color, predkosc. Następnie tworzy tablicę "points" z losowymi współrzędnymi, kolorami i prędkościami. Tworzy również tablicę "centers" z punktami, które będą reprezentować centra klastrów. Następnie kod przypisuje każdy punkt z tablicy "points" do najbliższego centrum z tablicy "centers" na podstawie odległości oraz predkości. Na końcu kod rysuje wykres z punktami z tablicy "points" z grupy wraz z centrami klastrów.

```python
import random
import math
import matplotlib.pyplot as plt

class Point:
    def __init__(self, x, y, color, predkosc):
        self.x = x
        self.y = y
        self.color = color
        self.predkosc = predkosc

    def dystans(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

points = []
colors = ['red', 'green', 'blue', 'yellow', 'orange', 'purple', 'pink', 'magenta', 'aqua']
predkosci = [50, 100, 150]

for i in range(300):
    x = random.randint(0, 500)
    y = random.randint(0, 500)
    color = random.choice(colors)
    predkosc = random.choice(predkosci)
    points.append(Point(x, y, color, predkosc))

centers = [Point(50, 50, 'red', 50),
           Point(450, 450, 'green', 100),
           Point(50, 450, 'blue', 150),
           Point(450, 50, 'yellow', 50),
           Point(250, 250, 'orange', 100),
           Point(300, 75, 'purple', 150),
           Point(100, 400, 'olive', 50),
           Point(375, 250, 'magenta', 100),
           Point(100, 250, 'aqua', 150)]

clusters = {center:[] for center in centers}
```

```python
for point in points:
    nearest_center = min(centers, key = lambda center: point.dystans(center) if center.predkosc == point.predkosc else float('inf'))
    clusters[nearest_center].append(point)

for center, cluster in clusters.items():
    x_coords = [point.x for point in cluster]
    y_coords = [point.y for point in cluster]
    plt.scatter(x_coords, y_coords, color = center.color, s = 20)
    if center.predkosc == 50:
        plt.scatter(center.x, center.y, marker = 'o', color = center.color, s = 150)
    elif center.predkosc == 100:
        plt.scatter(center.x, center.y, marker = 'X', color = center.color, s = 150)
    elif center.predkosc == 150:
        plt.scatter(center.x, center.y, marker = '^', color = center.color, s = 150)

plt.show()
```

W celu sprawdzenia oraz łatwiejszego odczytu wyników, zmieniliśmy markery punktów centralnych. Jeśli router oferuje prędkość łącza równą 50, to punkt centralny przybierze postać koła. W przypadku prędkości równej 100, będzie to pogrubiony x, natomiast jeśli wartość ta wyniesie 150, to punkt centralny stanie się trójkątem.

Punkty należące do konkretnego klastra, tak jak w poprzednich przypadkach mają kolor punktu centralnego, dzięki czemu łatwo je od siebie odróżnić.



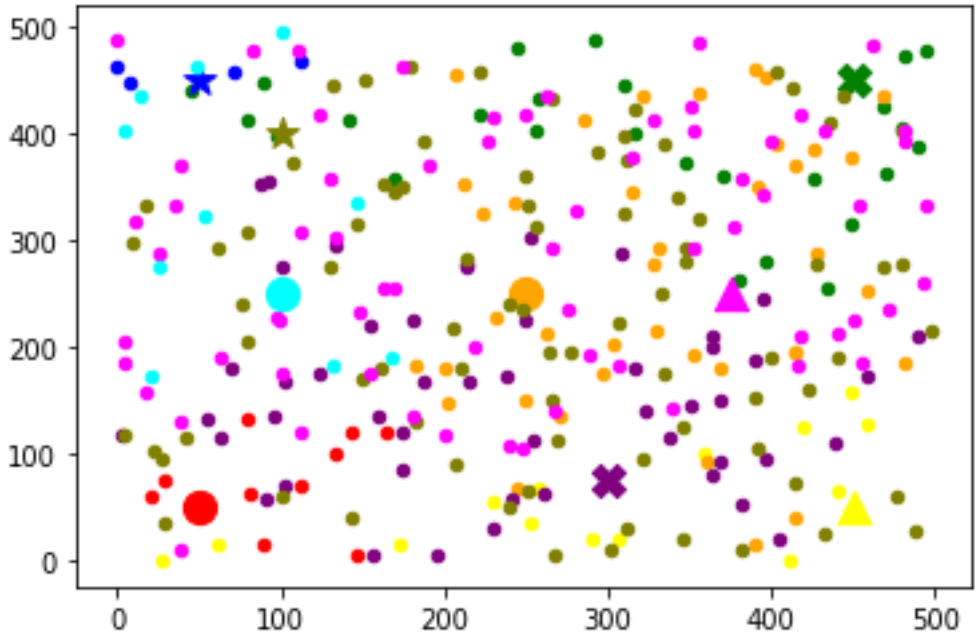
8. Klasteryzacja na podstawie odległości od najbliższego centroidu oraz parametru ping

Kod, tworzy klasę o nazwie "Point", która posiada atrybuty x, y, color i ping. Następnie generuje 300 punktów z losowymi współrzędnymi x, y, kolorami i wartościami ping. Potem tworzy listę punktów centralnych i przypisuje im zakresy wartości ping. Następnie dla każdego punktu znajduje najbliższy punkt centralny, którego zakres wartości ping zawiera wartość ping danego punktu. Punkty są następnie przypisywane do klastrów na podstawie najbliższego punktu centralnego. Klastry są wykreślane na wykresie, gdzie każdy punkt jest kolorowany na podstawie przypisanego klastra, a punkty centralne są tak jak w poprzednim przypadku oznaczone znakami specjalnymi na podstawie zakresu wartości ping. Na końcu jest wywoływana funkcja plt.show(), aby wyświetlić wykres.

```python
import random
import math
import matplotlib.pyplot as plt

class Point:
    def __init__(self, x, y, color, ping):
        self.x = x
        self.y = y
        self.color = color
        self.ping = ping

    def dystans(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

points = []
colors = ['red', 'green', 'blue', 'yellow', 'orange', 'purple', 'pink', 'magenta', 'aqua']
pingi = range(10,151)

for i in range(300):
    x = random.randint(0, 500)
    y = random.randint(0, 500)
    color = random.choice(colors)
    ping = random.choice(pingi)
    points.append(Point(x, y, color, ping))

centers = [Point(50, 50, 'red', range(10, 40)),
           Point(450, 450, 'green', range(40, 80)),
           Point(50, 450, 'blue', range(80, 120)),
           Point(450, 50, 'yellow', range(120, 151)),
           Point(250, 250, 'orange', range(10, 40)),
           Point(300, 75, 'purple', range(40, 80)),
           Point(100, 400, 'olive', range(80, 120)),
           Point(375, 250, 'magenta', range(120, 151)),
           Point(100, 250, 'aqua', range(10, 40))]

clusters = {center:[] for center in centers}
```

```python
for point in points:
    nearest_center = min(centers, key = lambda center: point.dystans(center) if point.ping in center.ping else float('inf'))
    clusters[nearest_center].append(point)

for center, cluster in clusters.items():
    x_coords = [point.x for point in cluster]
    y_coords = [point.y for point in cluster]
    plt.scatter(x_coords, y_coords, color = center.color, s = 20)
    if center.ping == range(10,40):
        plt.scatter(center.x, center.y, marker = 'o', color = center.color, s = 150)
    elif center.ping == range(40,80):
        plt.scatter(center.x, center.y, marker = 'X', color = center.color, s = 150)
    elif center.ping == range(80,120):
        plt.scatter(center.x, center.y, marker = '*', color = center.color, s = 150)
    elif center.ping == range(120,151):
        plt.scatter(center.x, center.y, marker = '^', color = center.color, s = 150)

plt.show()
```

Wynik działania powyższego skryptu:



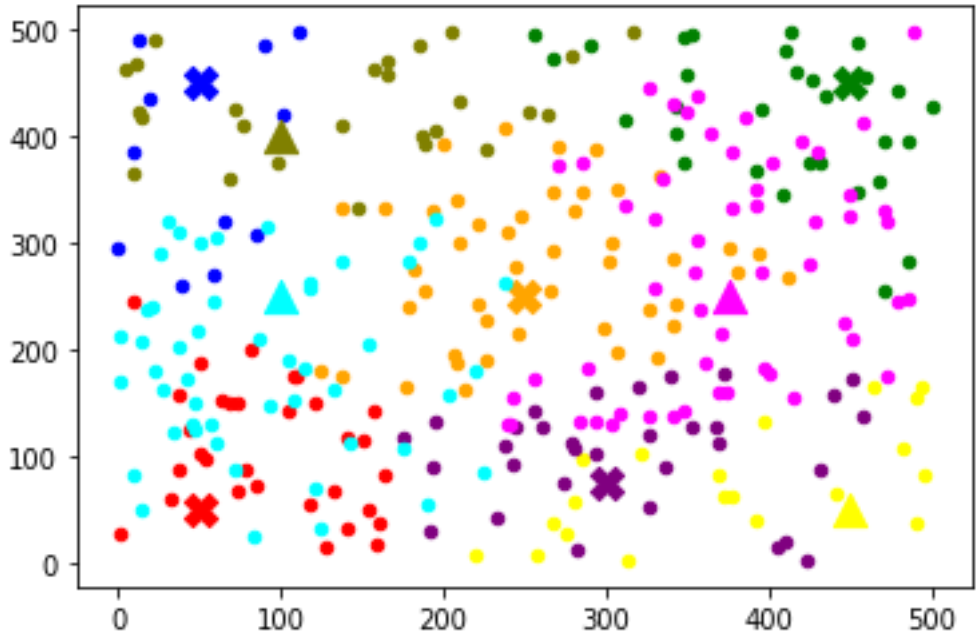
9. Klasteryzacja na podstawie odległości od najbliższego centroidu oraz parametru priorytet

Poniższy kod tworzy klasę Point, która przechowuje dane dotyczące punktu (współrzędne x i y, kolor oraz priorytet). Następnie generuje losowo 300 punktów i przypisuje im współrzędne z zakresu od 0 do 500 oraz losowy kolor i priorytet. Następnie tworzy 9 punktów - centroidów z różnymi kolorami i priorytetami. W kolejnym kroku dla każdego punktu znajduje najbliższy centroid i przypisuje go do odpowiedniego klastra, który posiada taki sam priorytet. Wyjątek stanowi jednak priorytet punktu równy 0. W takim przypadku, taki punkt zostaje sklasteryzowany wyłącznie na podstawie odległości do najbliższego punktu centralnego. Na końcu rysowany jest jeszcze wykres punktów i centroidów, gdzie punkty są koloru centroidu do którego zostały przypisane, a centroidy są oznaczone specjalnymi znakami w zależności od wartości ich priorytetu.

```python
import random
import math
import matplotlib.pyplot as plt

class Point:
    def __init__(self, x, y, color, priorytet):
        self.x = x
        self.y = y
        self.color = color
        self.priorytet = priorytet

    def dystans(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

points = []
colors = ['red', 'green', 'blue', 'yellow', 'orange', 'purple', 'pink', 'magenta', 'aqua']
priorytety = [0, 1, 2]

for i in range(300):
    x = random.randint(0, 500)
    y = random.randint(0, 500)
    color = random.choice(colors)
    priorytet = random.choice(priorytety)
    points.append(Point(x, y, color, priorytet))

centers = [Point(50, 50, 'red', 1),
           Point(450, 450, 'green', 1),
           Point(50, 450, 'blue', 1),
           Point(450, 50, 'yellow', 2),
           Point(250, 250, 'orange', 1),
           Point(300, 75, 'purple', 1),
           Point(100, 400, 'olive', 2),
           Point(375, 250, 'magenta', 2),
           Point(100, 250, 'aqua', 2)]

clusters = {center:[] for center in centers}
```

```python
for point in points:
    if point.priorytet == 0:
        nearest_center = min(centers, key = lambda center: point.dystans(center))
    else:
        nearest_center = min(centers, key = lambda center: point.dystans(center) if center.priorytet == point.priorytet else float('inf'))
    clusters[nearest_center].append(point)

for center, cluster in clusters.items():
    x_coords = [point.x for point in cluster]
    y_coords = [point.y for point in cluster]
    plt.scatter(x_coords, y_coords, color = center.color, s = 20)
    if center.priorytet == 1:
        plt.scatter(center.x, center.y, marker = 'X', color = center.color, s = 150)
    elif center.priorytet == 2:
        plt.scatter(center.x, center.y, marker = '^', color = center.color, s = 150)

plt.show()
```

Wynik działania powyższego skryptu:



10. Podsumowanie

Klasteryzacja to metoda uczenia nienadzorowanego, która polega na dzieleniu danych na grupy (klastery) o podobnych cechach. Celem klasteryzacji jest znalezienie naturalnych grup w danych, tak aby elementy w jednej grupie były podobne do siebie, a elementy w różnych grupach różniły się od siebie. Podsumowując, klasteryzacja to metoda, która może być przydatna w wielu różnych dziedzinach, takich jak biznes, marketing, medycyna, nauki społeczne czy informatyka. Może być używana do segmentacji rynku, identyfikacji anomalii, zrozumienia skupisk danych i wielu innych zastosowań.