import numpy as np

import matplotlib.pyplot as plt

from matplotlib import style

import pandas as pd

style.use('ggplot')

class K\_Means:

def \_\_init\_\_(self, k =3, tolerance = 0.0001, max\_iterations = 500):

self.k = k

self.tolerance = tolerance

self.max\_iterations = max\_iterations

def fit(self, data):

self.centroids = {}

#initialize the centroids, the first 'k' elements in the dataset will be our initial centroids

for i in range(self.k):

self.centroids[i] = data[i]

#begin iterations

for i in range(self.max\_iterations):

self.classes = {}

for i in range(self.k):

self.classes[i] = []

#find the distance between the point and cluster; choose the nearest centroid

for features in data:

distances = [np.linalg.norm(features - self.centroids[centroid]) for centroid in self.centroids]

classification = distances.index(min(distances))

self.classes[classification].append(features)

previous = dict(self.centroids)

#average the cluster datapoints to re-calculate the centroids

for classification in self.classes:

self.centroids[classification] = np.average(self.classes[classification], axis = 0)

isOptimal = True

for centroid in self.centroids:

original\_centroid = previous[centroid]

curr = self.centroids[centroid]

if np.sum((curr - original\_centroid)/original\_centroid \* 100.0) > self.tolerance:

isOptimal = False

#break out of the main loop if the results are optimal, ie. the centroids don't change their positions much(more than our tolerance)

if isOptimal:

break

def pred(self, data):

distances = [np.linalg.norm(data - self.centroids[centroid]) for centroid in self.centroids]

classification = distances.index(min(distances))

return classification

def main():

df = pd.read\_csv("data/students.csv")

df = df[['curriculum', 'athletic']]

dataset = df.astype(float).values.tolist()

X = df.values #returns a numpy array

km = K\_Means(3)

km.fit(X)

# Plotting

colors = 10\*["r", "c", "k", "g", "b"]

for centroid in km.centroids:

plt.scatter(km.centroids[centroid][0], km.centroids[centroid][1], s = 130, marker = "x")

for classification in km.classes:

color = colors[classification]

for features in km.classes[classification]:

plt.scatter(features[0], features[1], color = color,s = 30)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()



