دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی برق

گزارش‌کار آزمایش چهارم

آزمایشگاه مقدمه ای بر هوش محاسباتی

پیاده سازی K-Means Clustering

نگارش

ارشیا اسمعیل طهرانی

علی بابالو

پویا ابراهیمی

استاد راهنما

سرکار خانم موسوی

آذر ماه 1401

چکیده

در این آزمایش به پیاده سازی K-Means Clustering پرداخته شده است.

فهرست مطالب

صفحه

پیش گزارش

2-1- مفهوم الگوریتم با نظارت و بدون نظارت

در یادگیری نظارت شده، قانون یادگیری با مجموعه‌ای از مثال‌ها (مجموعه آموزشی) رفتار مناسب شبکه ارائه می‌شود:

$$\{\mathbf{p}_1,\mathbf{t}_1\}\ ,\ \{\mathbf{p}_2,\mathbf{t}_2\}\ ,\ \dots\ ,\ \{\mathbf{p}_Q,\mathbf{t}_Q\}\ ,$$

که در آن pq یک ورودی به شبکه و tq خروجی صحیح (هدف) مربوطه است. همانطور که ورودی ها به شبکه اعمال می شوند، خروجی های شبکه با اهداف مقایسه می شوند. سپس از قانون یادگیری برای تنظیم وزن ها و بایاس های شبکه استفاده می شود تا خروجی های شبکه به اهداف نزدیک تر شود.

اما در یادگیری بدون نظارت، وزن ها و Bias ها فقط در پاسخ به ورودی های شبکه اصلاح می شوند. هیچ خروجی هدفی در دسترس نیست. در نگاه اول ممکن است این غیرعملی به نظر برسد. اگر نمی دانید که قرار است چه کاری انجام دهد، چگونه می توانید یک شبکه را آموزش دهید؟ اکثر این الگوریتم ها نوعی عملیات خوشه بندی را انجام می دهند. آنها یاد می گیرند که الگوهای ورودی را در تعداد محدودی از کلاس ها طبقه بندی کنند. این به ویژه در کاربردهایی مانند کوانتیزاسیون برداری مفید است.

2-2- بیان مزیت های الگوریتم های با نظارت نسبت به الگوریتم های بدون نظارت و بر عکس

2-3-1- مزایای Supervised learning

- این به شما امکان می دهد در مورد تعریف برچسب ها بسیار دقیق باشید. به عبارت دیگر، می‌توانید الگوریتم را برای تشخیص کلاس‌های مختلف آموزش دهید، جایی که می‌توانید یک مرز تصمیم‌گیری ایده‌آل تعیین کنید.

- شما می توانید تعداد کلاس هایی را که می خواهید داشته باشید تعیین کنید.
- داده های ورودی بسیار شناخته شده و دارای برچسب هستند.
- نتایج حاصل از روش نظارت شده در مقایسه با نتایج حاصل از تکنیک‌های بدون نظارت یادگیری ماشین دقیق‌تر و قابل اعتمادتر هستند. این عمدتا به این دلیل است که داده های ورودی در الگوریتم نظارت شده به خوبی شناخته شده و دارای برچسب هستند. این یک تفاوت کلیدی بین یادگیری تحت نظارت و بدون نظارت است.
- پاسخ‌های موجود در تجزیه و تحلیل و خروجی الگوریتم شما احتمالاً مشخص است، زیرا همه کلاس‌های استفاده شده شناخته شده هستند.

2-3-2- معایب Supervised learning

- یادگیری تحت نظارت در مقایسه با روش بدون نظارت می تواند روشی پیچیده باشد. دلیل اصلی این است که شما باید به خوبی درک کنید و ورودی ها را در یادگیری نظارت شده برچسب گذاری کنید.
- این در Real – time انجام نمی شود در حالی که یادگیری بدون نظارت در مورد Real – Time است. این نیز یک تفاوت عمده بین یادگیری تحت نظارت و بدون نظارت است. یادگیری ماشین نظارت شده از تجزیه و تحلیل خطی استفاده می کند.
- برای آموزش به زمان محاسباتی زیادی نیاز است.
- اگر داده‌های بزرگ و در حال رشد پویا وجود دارد، و برچسب‌هایی که قوانین را از قبل تعریف کرده اند غیر قابل اطمینان اند. این می تواند یک چالش واقعی باشد.

2-3-3- مزایای Unsupervised learning

- پیچیدگی کمتر در مقایسه با یادگیری تحت نظارت. برخلاف الگوریتم های نظارت شده، در یادگیری بدون نظارت، هیچ کس نیازی به درک و سپس برچسب گذاری ورودی داده ها ندارد. این امر یادگیری بدون نظارت را پیچیده تر می کند و توضیح می دهد که چرا بسیاری از افراد تکنیک های بدون نظارت را ترجیح می دهند.
- در زمان واقعی انجام می شود به طوری که تمام داده های ورودی در حضور فراگیران تجزیه و تحلیل و برچسب گذاری می شود. این به آنها کمک می کند تا مدل های مختلف یادگیری و مرتب سازی داده های خام را به خوبی درک کنند.

- دریافت داده‌های بدون برچسب اغلب آسان‌تر است - از رایانه نسبت به داده‌های برچسب‌گذاری شده، که نیاز به مداخله شخص دارند. این نیز یک تفاوت کلیدی بین یادگیری تحت نظارت و بدون نظارت است.

2-3-4- معایب Unsupervised learning

- شما نمی توانید در مورد تعریف مرتب سازی داده ها و خروجی خیلی دقیق صحبت کنید. این به این دلیل است که داده های مورد استفاده در یادگیری بدون نظارت برچسب گذاری شده و شناخته شده نیستند. این وظیفه ماشین است که داده های خام را قبل از تعیین الگوهای پنهان برچسب گذاری و گروه بندی کند.
- دقت کمتر نتایج این نیز به این دلیل است که داده های ورودی از قبل توسط افراد مشخص نیست و برچسب گذاری نشده است، به این معنی که ماشین باید این کار را به تنهایی انجام دهد.
- نتایج تجزیه و تحلیل را نمی توان مشخص کرد. هیچ دانش قبلی در مورد روش بدون نظارت یادگیری ماشین وجود ندارد. علاوه بر این، تعداد کلاس ها نیز مشخص نیست. منجر به عدم توانایی در تعیین نتایج حاصل از تجزیه و تحلیل می شود.

2-3- توضیح الگوریتم دسته بندی K – Means

الگوریتم چگونه کار می کند؟ ، فرض کنید centroid ها به شما داده شده است. شما به راحتی می توانید تمام نمونه های موجود در مجموعه داده را با اختصاص دادن هر یک از آنها به خوشه ای که مرکز آن نزدیک ترین است، برچسب گذاری کنید. برعکس، اگر همه برچسب‌های نمونه به شما داده شود، می‌توانید به راحتی مرکز هر خوشه را با محاسبه میانگین نمونه‌های آن خوشه پیدا کنید. اما به شما نه برچسب و نه مرکز داده شده است، پس چگونه می توانید ادامه دهید؟

- با قرار دادن سانتروئیدها به طور تصادفی شروع کنید (به عنوان مثال، با انتخاب k نمونه به طور تصادفی از مجموعه داده و استفاده از مکان آنها به عنوان مرکز).
- سپس نمونه‌ها را برچسب بزنید، مرکزها را به‌روزرسانی کنید،
- نمونه‌ها را برچسب بزنید، مرکزها را به‌روزرسانی کنید،
- و به همین ترتیب تا زمانی که مرکزها از حرکت بازایستند.

این الگوریتم تضمین شده است که در تعداد محدودی از مراحل (معمولاً بسیار کوچک) همگرا شود. این به این دلیل اســـت که میانگین مجذور فاصـــله بین نمونه‌ها و نزدیک‌ترین مرکز آنها فقط می‌تواند در هر مرحله پایین بیاید و از آنجایی که نمی‌تواند منفی باشد، تضمین می‌شود که همگرا شود.

اگرچه الگوریتم تضمین شده است که همگرا شود، اما ممکن است به راه حل مناسب همگرا نشود (یعنی ممکن اســت به یک بهینه محلی همگرا شــود): اینکه آیا این الگوریتم به مقدار دهی اولیه مرکز بســتگی دارد.

الگوریتم K – Means یک الگوریتم بازگشتی است که با یک فرض اولیه درباره مراکز دسته ها آغاز می شود در هر مرحله از اجرای الگوریتم مراحل زیر اجرا می شود:

1) پیدا کردن دسته متناظر با تمام نقاط
2) تازه سازی مراکز دسته ها

در مرحله اول اجرای الگوریتم فاصله تمام نقاط تا مراکز دسته ها محاسبه می شود و سپس هر داده متعلق به دسته ای که کمترین فاصله با آن را دارد می شود.

بعد از پیدا کردن دسته های متناظر با هر داده ، در مرحله دوم میانگین داده های متعلق به یک دسته به عنوان مرکز دسته در نظر گرفته می شود.

مراحل۱ و۲ تا زمانی که مراکز دسته ها تغییر نکند و یا تغییرات خیلی کمی داشته باشه ادامه می یابد.

بعد از ثابت شدن مراکز دسته ها الگوریتم همگرا می شود.

2-4- کاربرد های الگوریتم K – Means

خوشه بندی در طیف گسترده ای از برنامه ها استفاده می شود، از جمله:

2-3-5- تقسیم بندی مشتریان

شما می توانید مشتریان خود را بر اساس خرید و فعالیت آنها در وب سایت خود دسته بندی کنید. این برای درک اینکه مشــتریان شــما چه کســانی هســتند و به چه چیزی نیاز دارند مفید اســت، بنابراین می توانید محصـــولات و کمپین های بازاریابی خود را با هر بخش تطبیق دهید. برای مثال، تقســـیم‌بندی

مشتری می‌تواند در سیستم‌های توصیه‌گر برای پیشنهاد محتوایی که سایر کاربران در همان خوشه از آن لذت می‌برند، مفید باشد.

2-3-6- تحلیل داده ها

هنگامی که یک مجموعه داده جدید را تجزیه و تحلیل می کنید، اجرای یک الگوریتم خوشـــه بندی و سپس تجزیه و تحلیل هر خوشه به طور جداگانه می تواند مفید باشد.

2-3-7- کاهش ابعاد

هنگامی که یک مجموعه داده خوشـــه‌بندی شـــد، معمولاً می‌توان وابســـتگی هر نمونه را با هر خوشـــه اندازه‌گیری کرد. وابسـتگی هر معیاری اسـت که نشـان می دهد یک نمونه چقدر در یک خوشـه قرار می گیرد. سـپس بردار ویژگی هر نمونه x را می توان با بردار قرابت های خوشـه ای آن جایگزین کرد. اگر k خوشه وجود داشته باشد، این بردار k بعدی است. بردار جدید معمولاً ابعاد بسیار کمتری نسبت به بردار ویژگی اصلی دارد، اما می تواند اطلاعات کافی را برای پردازش بیشتر حفظ کند.

2-3-8- مهندسی ویژگی

پیوندهای خوشـه اغلب می توانند به عنوان ویژگی های اضـافی مفید باشـند. به عنوان مثال، K - Means برای افزودن ویژگی‌های وابسته به خوشه جغرافیایی به مجموعه داده مسکن کالیفرنیا استفاده می‌شود، و آنها به عملکرد بهتر کمک کردند.

2-3-9- تشخیص ناهنجاری (Outliers)

هر نمونه ای که تمایل کمی به همه خوشـه ها داشـته باشـد، احتمالاً یک ناهنجاری اسـت. به عنوان مثال، اگر کاربران وب سایت خود را بر اساس رفتار آنها دسته بندی کرده باشید، می توانید کاربرانی را که رفتار غیرعادی دارند، مانند تعداد غیرعادی درخواست در ثانیه شناسایی کنید.

2-3-10-یادگیری نیمه نظارتی

اگر فقط چند برچسب دارید، می‌توانید خوشه‌بندی را انجام دهید و برچسب‌ها را در همه نمونه‌های یک خوشه منتشر کنید. این تکنیک می‌تواند تعداد برچسب‌های موجود برای الگوریتم یادگیری تحت نظارت بعدی را تا حد زیادی افزایش دهد و در نتیجه عملکرد آن را بهبود بخشد.

2-3-11-موتورهای جستجو

برخی از موتورهای جستجو به شما امکان می دهند تصاویری را جستجو کنید که شبیه به یک تصویر مرجع هستند. برای ساختن چنین سیستمی، ابتدا باید یک الگوریتم خوشه بندی را برای تمام تصاویر موجود در پایگاه داده خود اعمال کنید. تصاویر مشابه به یک خوشه ختم می شوند. سپس هنگامی که یک کاربر یک تصویر مرجع ارائه می دهد، تنها کاری که باید انجام دهید این است که از مدل خوشه بندی آموزش دیده برای یافتن خوشه این تصویر استفاده کنید و سپس می توانید به سادگی تمام تصاویر را از این خوشه برگردانید.

2-3-12-تقسیم بندی تصویر

با خوشه بندی پیکسل ها بر اساس رنگ آنها و سپس جایگزینی رنگ هر پیکسل با رنگ متوسط خوشه آن، می توان تعداد رنگ های مختلف در یک تصویر را به میزان قابل توجهی کاهش داد. بخش‌بندی تصویر در بسیاری از سیستم‌های تشخیص و ردیابی اشیا استفاده می‌شود، زیرا تشخیص کانتور هر شی را آسان‌تر می‌کند.

شرح آزمایش

در این قسمت به پیاده سازی K – Means در محیط پایتون نیز پرداخته شده است.

2-5- محیط Python

2-3-13-کلاس KMeans

در ابتدای امر کتابخلنه های مورد نیاز را import می‌کنیم. و seed برای random را یک مقدار دلخواه انتخاب می‌کنیم تا با هر بار اجرای برنامه مقادیر مختلف random نیز گرفته نشود!

```python
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import cv2
import os

np.random.seed(42)
```

شکل ۱) ایمپورت کردن کتابخانه ها

در ابتدا در تابع کانستراکتور یا سازنده کلاس مقادیر مورد نیاز را مقدار دهی اولیه می‌کنیم؛

```python
class KMeans:
    def __init__(self, K=5, max_iters=100, plot=False):
        self.K = K
        self.max_iters = max_iters
        self.plot = plot
        self.mean = []
        self.num_samples, self.num_features = 0, 0

        # list of sample indices for each cluster
        self.clusters = [[] for _ in range(self.K)]
        # the centers (mean feature vector) for each cluster
        self.centroids = []
```

شکل ۲) کانستراکتور کلاس K-Means

- K برای تعداد کلاستر های ورودی
- Max_iters برای حداکثر iteration هایی که طی شود
- Plot برای اینکه آیا step by step چاپ بکند یا خیر
- Mean لیستی است که در آن ها میانگین فاصله داده های از centroid محاسبه شده و ریخته می‌شود.
- Num_samples و num_features به ترتیب شماره sample ها و شماره feature ها هستند.

- Clusters، لیستی از لیست ها است که در آن اندیس های عضور های داخل هر یک از کلاستر ها، داخل هر یک از المان های آن ریخته می‌شود که تعداد المان های آن، پرواضح است که برابر با تعداد کلاستر های می‌باشد.
- Centroids لیستی است که در آن centroid ها ریخته می‌شود.

سپس به مقدار دهی کردن اولیه centroid هایمان می‌پردازیم؛

```python
def _initialize_centroids(self, X):

    # 1
    # for _ in range(self.K):
    #     centroid = X[np.random.choice(range(self.num_samples))]
    #     self.centroids.append(centroid)

    # 2
        # for _ in range(self.K):
        #     centroid = X[np.random.choice(self.num_samples, replace=False)]
        #     self.centroids.append(centroid)

    # 3 - MAIN
    random_sample_idxs = np.random.choice(self.num_samples, self.K, replace=False)
    self.centroids = [self.X[idx] for idx in random_sample_idxs]
```

شکل۳) مقدار دهی اولیه به مراکز دسته ها

- روش اول و دوم؛ پس از پیاده سازی این روش ها که در کد آورده شده است، مشخص شد که در خروجی برخی از میانگین هایمان نیز Nan ظاهر می‌شود. پس از بررسی فراوان مشخص شد که دلیل این امر، این بود که چون در هر iteration حلقه، از sample ها به صورت random حتی با وجود غیر تکراری بودن، نمونه برداری می‌شود، و با نمونه برداری در هر حلقه، بررسی نمی‌شود که آیا مقدار برداشته شده، با مقادیر برداشته شده در حلقه های قبل یکسان می‌باشد یا خیر، آن گاه در برخی از k ها و مخصوصا در k های بالاتر نیز، مقادیر تکراری برای centroid ها وجود می‌داشت و وجود این مقادیر تکراری معادلات و محاسبات را بر هم می‌زد و باعث ظهور مقادیر Nan در میانگین ها می‌شد!
- روش سوم؛ بنابراین تصمیم بر این شد که ابتدا و فقط یک بار بدون تکرار مقادیر تصادفی ای را از num_sample هایمان انتخاب و نمونه برداری کنیم(اندیس های آن ها انتخاب می‌شوند)، و سپس، با یک list comprehension، مقادیر centroid ها را از آن ها برداریم!

در ادامه در متد get_cluster_labels_، هر sample ای، لندیس کلاستری را که عضو آن است، می‌گیرد!

```python
def _get_cluster_labels(self, clusters):
    # each sample will get the label of the cluster it was assigned to
    labels = np.empty(self.num_samples)

    for cluster_idx, cluster in enumerate(clusters):
        for sample_index in cluster:
            labels[sample_index] = cluster_idx
    return labels
```

شکل۴) دسته بندی کلاستر ها

- توجه شود که np.empty، یک array جدیدی را با type و shape ورودی آن یعنی num_samples را می‌گیرد!
- در کلاستر های enumerate می‌کند، همچنین در سمپل های هر کلاستر می‌چرخد، و سمپل مخصوص هر کلاستر را، اندیس آن کلاستر را بهش می‌دهد، و در نهایت label ها نیز return می‌شوند!

سپس در _create_clusters، لیستی از لیست ها که لیست های داخل به تعداد k هستند، ایجاد شده؛

```python
def _create_clusters(self, centroids):
    # Assign the samples to the closest centroids to create clusters
    clusters = [[] for _ in range(self.K)]
    for idx, point in enumerate(self.X):
        closest_centroid = np.argmin(np.sqrt(np.sum((point - centroids) ** 2, axis=1)))
        clusters[closest_centroid].append(idx)
    return clusters
```

شکل۵) تابع برای دسته بندی کردن ورودی

- فاصله هر point را با مراکز کلاستر ها مقایسه می‌کند و کمترین آن فاصله را پیدا کرده. سپس، اندیس آن point را در عضو clusters که اندیس آن فاصله min را دارد، افزوده و در نهایت clusters را return می‌کند!

در متد _new_centroids، میانگین داده های عضو هر کلاستر محاسبه و مشخص می‌شود که کدام عضو است، و centroid جدید را برابر آن عضو قرار می‌دهد.

```python
def _new_centroids(self, clusters):
    # assign mean value of clusters to centroids
    centroids = np.zeros((self.K, self.num_features))
    for idx, cluster in enumerate(clusters):
        cluster_mean = np.mean(self.X[cluster], axis=0)
        centroids[idx] = cluster_mean
    return centroids
```

شکل۶) تابع برای تغییر مراکز

- به این گونه عمل می‌کند که ابتدا آرایه ای جدید با shape و type آرگومان داخلی اش تولید می‌شود که تعداد سطر های آن k یا تعداد کلاستر ها، و تعداد ستون های آن تعداد feature ها می‌باشد.
- در clusters، enumerate می‌کند که با این کار اندیس هر عضو هم در می‌آید، میانگین داده های هر کلاستر به صورت سطری نیز حساب می‌شود و مقدار لیبلcentroid، برابر با مقداد label آن اندیس نیز قرار می‌گیرد.

در ادامه در متد fit، مقادیر X، Num_samples، Num_features، مقدار دهی می‌شوند،

```
def fit(self, X):
    self.X = X
    self.num_samples, self.num_features = X.shape
    # print(f"num_samples: {self.num_samples}")
    # print(f"range um_samples: {range(self.num_samples)}")

    # initialize
    self._initialize_centroids(X)

    # Optimize clusters
    for _ in range(self.max_iters):
        # Assign samples to closest centroids (create clusters)
        self.clusters = self._create_clusters(self.centroids)

        if self.plot:
            self._plot()

        # Calculate new centroids from the clusters
        centroids_old = self.centroids
        self.centroids = self._new_centroids(self.clusters)

        # check if clusters have changed
        diff = self.centroids - centroids_old
        if not diff.any():
            break

        if self.plot:
            self._plot()

    # Classify samples as the index of their clusters
    return self._get_cluster_labels(self.clusters)
```

شکل۷) predict/fit کردن ورودی

- با فراخوانی متد _initialize_centroids، centroid ها نیز مقدار دهی اولیه می‌شوند.
- در loop نیز با هر بار epoch، به واسطه ی centroid های مرحله قبلی و یک سری operation، کلاستر های ساخته می‌شوند.
- کلاستر های با _creat_cluster نیز ساخته شده؛
- اگر آرگومان plot، True بود، در هر epoch، نمودار های plot می‌شوند.
- Centroids_old و centroids های جدید نیز مقدار دهی می‌شوند،

- در diff نیز اختلاف آن ها محاسبه می‌شود،
- اگر همه آن ها 0 شده بود، حلقه نیز break می‌کند،
- در نهایت، سمپل ها بر اساس اندیس کلاسترشان، دسته بندی و مشخص می‌شوند و در نهایت return می‌شوند.

در متد _plot نیز محل قرار گیری point ها و centroid هر کلاستر نیز رسم می‌شود، و اگر flag مربوط به plot در آرگومان ورودی کلاس نیز True باشد، در هر حلقه نیز رسم می‌شود؛

```
def _plot(self):
    fig, ax = plt.subplots(figsize=(12, 8))

    for i, index in enumerate(self.clusters):
        point = self.X[index].T
        ax.scatter(*point)

    for point in self.centroids:
        ax.scatter(*point, marker="x", color="black", linewidth=2)

    plt.show()
```

شکل۸) پلات کردن نقطه ها و نشان دادن مرکز هر دسته

```
  def _plot_(self, X, Y):
      fig = px.scatter(X[:, 0], X[:, 1], color=Y)
      fig.show()
```

شکل۹) در متد _plot_ به نحوی دیگر داده ها نمایش داده می‌شوند

در ادامه، تابع ذیل، نقاط و دسته بندی متناظر با آن ها را از ورودی دریافت می‌کند، سپس برای هر دسته میانگین فاصله نقاط متعلق به دسته مورد نظر را تا مرکز دسته محاسبه می‌کند که به این عدد خطای هر دسته گفته می شود.

```
 def _mean(self, X):
     dist_t = []
     for i in range(len(self.centroids)):
         dist = []
         for j in self.clusters[i]:
             dis = np.sqrt(np.sum((X[j] - self.centroids[i])**2))
             dist.append(dis)
         dist_t.append(dist)
         self.mean.append(np.mean(dist_t[i]))
     return self.mean
```

شکل۱۰) محاسبه میانگین فاصله هر نقطه از مرکز دسته اش

- در dist، میانگین فاصله نقاط هر کلاستر تا مرکز آن کلاستر مشخص شده است،
- در dist_t، dist ها ذخیره شده است
- ابتدا در dis نیز فاصله هر نقطه تا مرکز مشخص شده، سپس در dist نیز append می‌شود و در نهایت dist در dist_t نیز append می‌شود. و در آخر میانگین dist_t ها در mean نیز append شده و mean بازگردانده می‌شود.

در متد mean_total_ نیز، متد mean_ فراخوانی می‌شـــود و در نهایت با میانگین گیری از خروجی آن یعنی mean(mean) نیز، میانگین خطای دســـته ها به عنوان خطای الگوریتم محاســـبه و به عنوان خروجی باز گردانده می‌شود.

```
def _mean_total(self, X):
    mean = self._mean(X)
    print("____________________Check____________________")
    print(f"mean:\n{mean}")
    print(f"Length of mean:\n{len(mean)}")
    return np.mean(mean)
```

شکل۱۱) محاسبه خطای نهایی که برابر میانگین همه میانگین فواصل از مرکز دسته است

در نهایت در متد error_plot، میزان خطای clustering را به ازای تعداد دسته ها یا تعداد کلاستر های مختلف که از 1 تا 15 می‌باشد، محاسبه می‌شود، در اصل الگوریتم کلاسترینگ به ازای k های محتلف تا 15 پیاده سازی و اجرا می‌شود و خطای آن ها نیز در لیستی ذخیر شده و در نهایت plot می‌شوند.

```
def error_plot(X, max_cluster=15):
    error = [0]

    for i in range(max_cluster):
        k = KMeans(K=i+1, max_iters=100, plot=False)
        k.fit(X)
        error.append(k._mean_total(X))
    plt.plot([*range(0,max_cluster+1,1)], error)
```

شکل۱۲) پلات کردن خطا بر حسب k

- عضو اول لیست error با 0 پر شده است چرا که برای plot شدن، مقادیر ارور باید با مقدار 0 جمع شوند و الگوریتم پیش برود.
- سپس در حلقه ای برای هر iteration و برای هر k ای نیز KMeans اجرا شده و خطای آن الگوریتم که پیش تر توسط متد mean_total_ نیز محاسبه شده بود، به لیست error نیز append شده و در نهایت در span مشخصی نیز plot می‌شود.

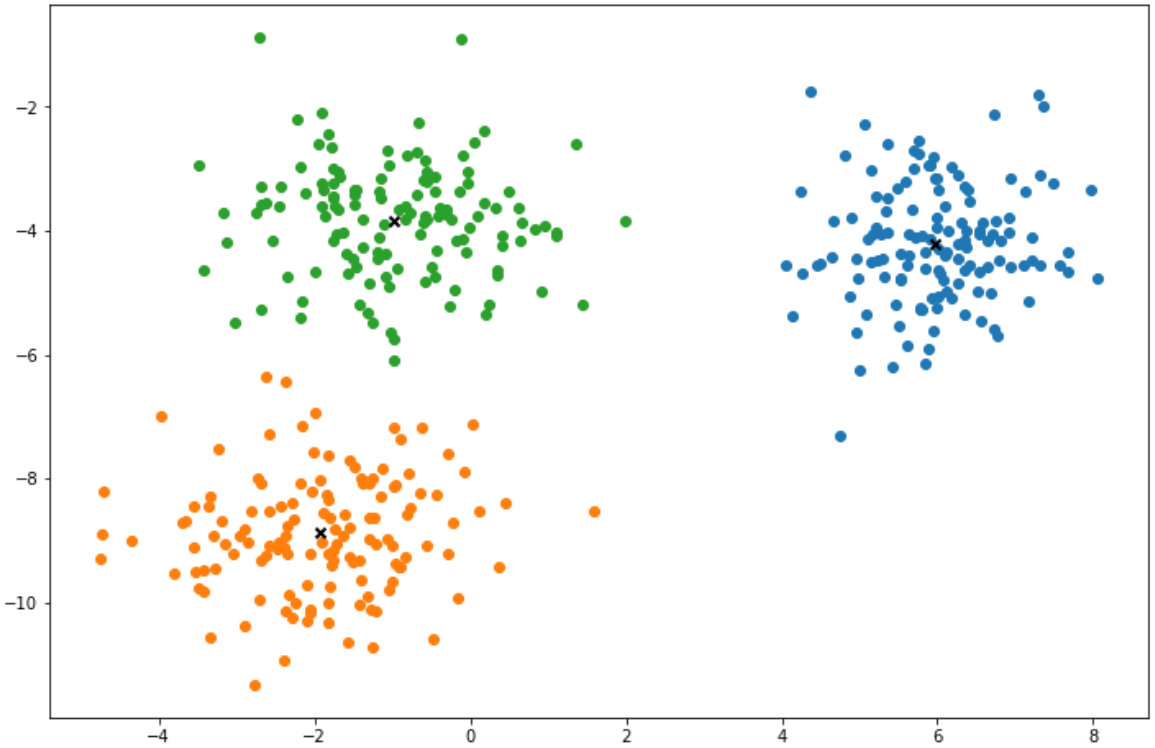
Train کردن مدل

Test

```
In [23]: # Testing
if __name__ == "__main__":
    from sklearn.datasets import make_blobs

    X, y = make_blobs(
        centers=3, n_samples=400, n_features=2, shuffle=True, random_state=40
    )
    print(X.shape),

    clusters = len(np.unique(y))
    print(clusters)

    k = KMeans(K=clusters, max_iters=150, plot=True)
    y_pred = k.fit(X)

    # plot
    print("____________________final plot____________________")
    k._plot_(X, k._get_cluster_labels(k.clusters))
    k._plot()

    # ploting Error
    print("____________________Error plot____________________")
    error_plot(X, max_cluster=15)
    plt.xlabel('K')
    plt.ylabel('Error')
```

شکل۱۳) اموزش الگوریتم Kmeans

برای آموزش الگوریتم K-Means از تابع make_blobs در کتابخانه sklearn استفاده می‌کنیم که شبکه را با استفاده از این دیتاست Train می‌کنیم. که در شکل پایین خروجی الگوریتم را مشاهده می‌کنیم که به درستی ۳ مرکز دسته هارا پیدا کرده است.



شکل۱۴) خروجی الگوریتم Kmeans

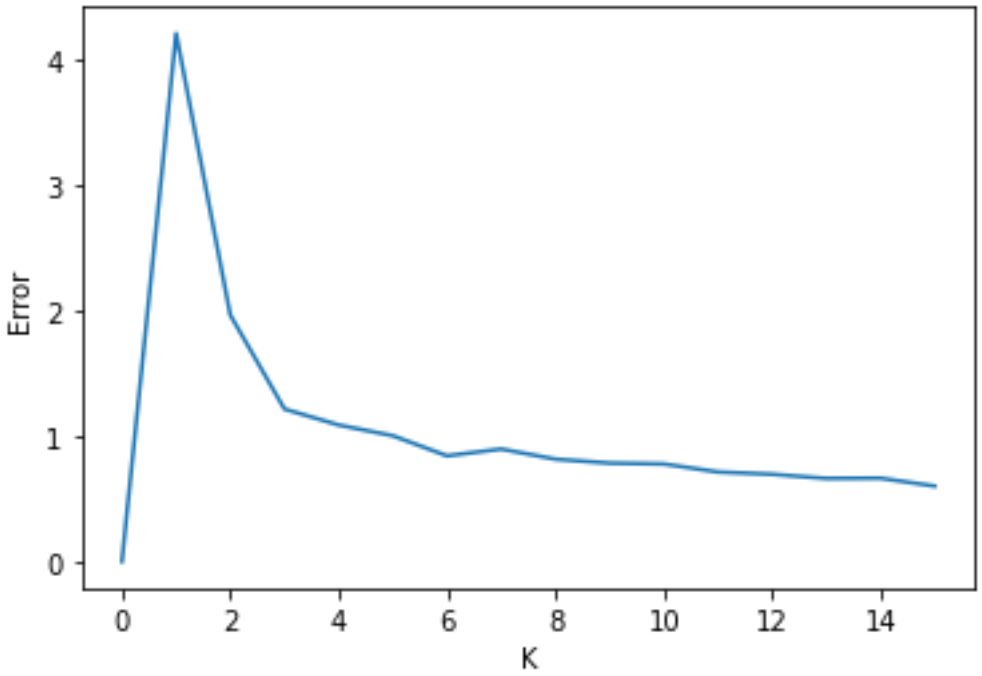
ارزیابی الگوریتم

```
    def _mean(self, X):
        dist_t = []
        for i in range(len(self.centroids)):
            dist = []
            for j in self.clusters[i]:
                dis = np.sqrt(np.sum((X[j] - self.centroids[i])**2))
                dist.append(dis)
            dist_t.append(dist)
            self.mean.append(np.mean(dist_t[i]))
        return self.mean

    def _mean_total(self, X):
        mean = self._mean(X)
        print("____________________Check____________________")
        print(f"mean:\n{mean}")
        print(f"Length of mean:\n{len(mean)}")
        return np.mean(mean)



def error_plot(X, max_cluster=15):
    error = [0]

    for i in range(max_cluster):
        k = KMeans(K=i+1, max_iters=100, plot=False)
        k.fit(X)
        error.append(k._mean_total(X))
    plt.plot([*range(0,max_cluster+1,1)], error)
```

شکل ۱۵) ارزیابی الگوریتم

برای ارزیابی الگوریتم یک تابع error_plot تعریف میکنیم که از دوتابع _mean, _mean_total استفاده می‌کند که اولی میانگین فاصله نقاط کلاستر از مرکز آن را حساب می‌کند و دیگری میانگین همه میانگین هارا حساب می‌کند.

در تابع error_plot یک لیست error تعریف میکنیم که ایندکس اول آن برابر ۰ است زیرا الگوریتم kmeans برای k = 0 تعریف نشده است پس برای پلات کردن آن از نقطه (۰-۰) شروع می‌کنیم. این تابع یک ورودی max_cluster میگیرد که ماکسیمم تعداد کلاستر های می‌خواهیم برسی کنیم می‌باشد. سپس یک حلقه در رنج ماکسیمم تعداد تشکیل می‌دهیم که در هر حلقه کلاس kmeans را با K های مختلف ایجاد می‌کند و دیتا را بر روی آن fit می‌کند سپس ارور الگوریتم را بر لیست error، append میکند. در نهایت خطا را بر اساس مقدار k (تعداد دسته ها) پلات می‌کنیم که در تصویر پایین خروجی این تابع مشخص است.



شکل۱۶) نمودار خطا برحسب k

کاهش حجم عکس با استفاده از K-Means:

همانطور که در دستور کار توضیح داده شده است، یکی از کاربرد های این الگوریتم کاهش حجم عکس است گه در ادامه آن را توضیح خواهیم داد.

In [11]:
```
# GIGA chad xD

pic = cv2.imread("./assets/giga_chad.png").astype(np.int32)

# pic_ = pic.reshape(pic.shape[0] * pic.shape[1],3)
pic_ = pic.reshape(-1, 3)
compressed_pic = pic_

k = KMeans(K=2, max_iters=100, plot=False)
k.fit(pic_)

for i in range(len(k.centroids)):
    compressed_pic[k.clusters[i]] = k.centroids[i]

compressed_pic = np.clip(compressed_pic.astype('uint8'), 0, 255)
compressed_pic = compressed_pic.reshape(pic.shape[0], pic.shape[1], 3)

cv2.imwrite(os.path.join("./assets", "chad_compressed.png"), compressed_pic)
```

Out[11]: True

In [7]:
```
# Ladybug

pic = cv2.imread("./assets/ladybug.png").astype(np.int32)

# pic = pic.reshape(pic.shape[0] * pic.shape[1],3)
pic_ = pic.reshape(-1, 3)
compressed_pic = pic_

k = KMeans(K=16, max_iters=100, plot=False)
k.fit(pic_)

for i in range(len(k.centroids)):
    compressed_pic[k.clusters[i]] = k.centroids[i]

compressed_pic = np.clip(compressed_pic.astype('uint8'), 0, 255)
compressed_pic = compressed_pic.reshape(pic.shape[0], pic.shape[1], 3)

cv2.imwrite(os.path.join("./assets", "ladybug_k16.png"), compressed_pic)
```

شکل ۱۷) تصویر مربوط به

در ابتدا عکس را به وسیله تابع imread() در کتابخانه OpenCv می‌خوانیم که این تابع یک ماتریس به ابعاد (طول*عرض*۳) میدهد که عدد ۳ مربوط به RGB هر پیکسل می‌باشد. الگوریتم K-Means ما یک ورودی ۲ بعدی نیاز دارد که برای اینکه ما ماتریس ۳بعدیمان را به ۲ بعد تبدیل کنیم از تابع reshape استفاده میکنیم که سایز ماتریس را به تعداد کل پیکسل ها (طول*عرض) *۳ تبدیل بکند.

سپس یک کپی از ماتریسمان می‌گیریم و آن را در ماتریس کامپرس شده ذخیره می‌کنیم.

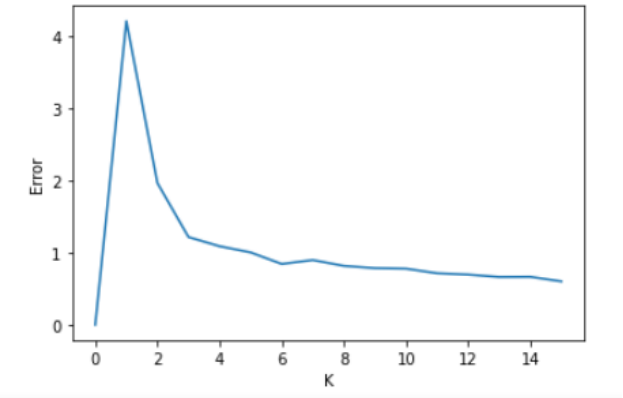
آنگاه کلاس Kmeans را با K دلخواه – مثلا ۱۶ – صدا می‌کنیم و آن را با ماتریس ۲بعدیمان fit می‌کنیم. سپس بعد از fit شدن دیتا هایمان، مقدار RGB های دیتا های هر کلاستر را برابر با مقدار RGB مرکز آن دسته قرار می‌هیم و به این صورت حجم تصویرمان کاهش می‌یابد، که این کار را در اینجا

با استفاده از یک حلقه انجام دادیم. در انتها نیز این ماتریس ۲بعدی تصویر کاهش یافته را به ابعاد اصلی آن بر میگردانیم و با استفاده از تابع ()imwrite ذخیره می‌کنیم.

شکل ۱۸) تصویر اصلی و کاهش یافته با K = 2 از ارنست خالیموف ملقب به Giga-Chad

تمارین

۱ – همانطور که در نمودار خطا مشاهده کردیم با اضافه شدن مقادیر K بصورت میانگین مقدار خطا کاهش می‌یابد اما بعد از حدی از K این کاهش خطا بسیار کم می‌شود بصورتی که تقریبا تاثیری روی دقت الگوریتم ندارد و فقط حجم محاسبات را زیاد می‌کند. برای پیدا کردن نقطه بهینه الگرویتم باید نقطه elbow خطا را پیدا کنیم و آن را مقدار بهینه K برای استفاده از الگوریتم بیان کنیم.



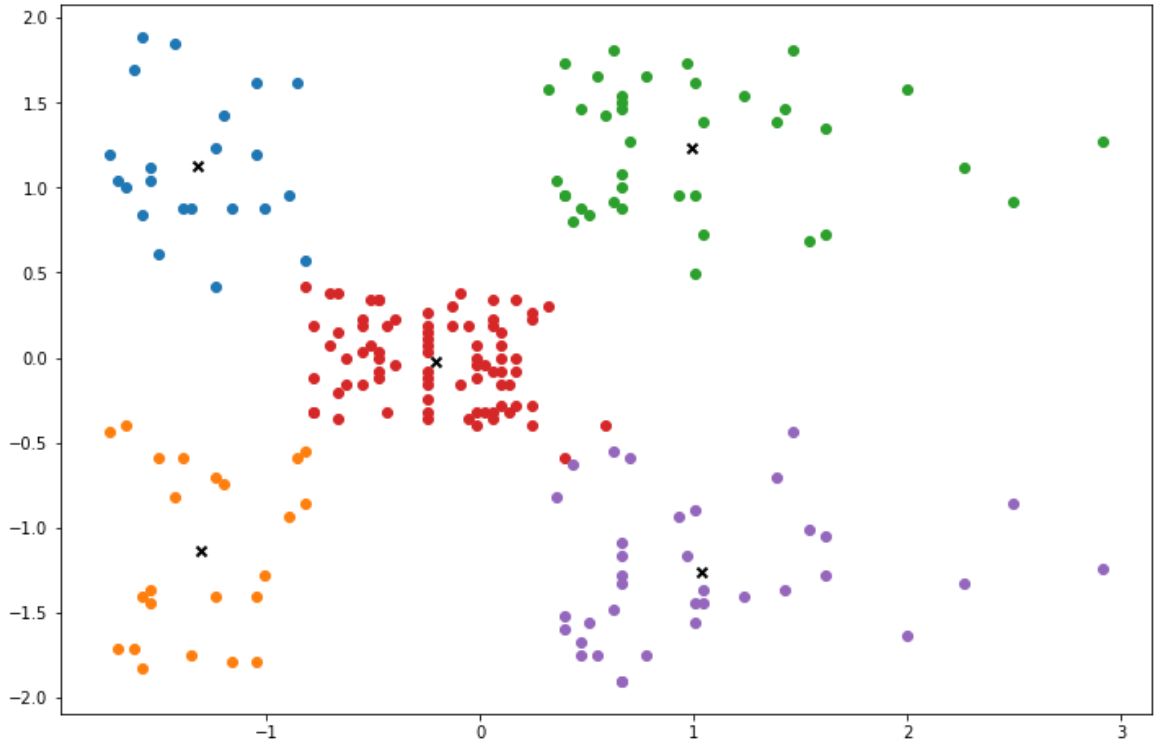
همانطور که در تصویر مشاهده میکنیم، در k=8 به بعد تقریبا خطا تغییری نمی‌کند و می‌توان گفت به نقطه elbow رسیدیم البته برای پیدا کردن آن نقطه نیز باید از الگوریتم خاصی استفاده کنیم که چون در دستور کار گفته نشده بود از زدن ان پرهیز کردیم و صرفا به بیان نقطه elbow اکتفا می‌کنیم.

۵ – دیتا ست Mall-Customer

ابتدا این دیتاست را بصورت یک دیتافریم می‌خوانیم و برای چک کردن اینکه کدام مشتری بیشتر راغب به خریدن کالا است را پیدا کنیم فقط نیاز به درامد سالانه و spending score مشتری داریم به این صورت که در نظر ما مشتری ای را می‌توان به خرید محصول ترغیب کرد که درامد بالایی داشته باشد و احتمال خرید آن فرد نیز بالا باشد زیرا اگر مشتری درامد کم داشته باشد ولی راغب به خرید هم باشد احتمال خرید آن مشتری کم است و بالعکس. پس مقادیر ۲ ستون مربوطه را در یک دیتافریم قرار می‌دهیم که از آنها استفاده کنیم.

بعد از خواندن دیتاست و شافل کردن آن، آنر نورمالایز میکنیم که روش نرمالایز کردن را در تمارین قبلی توضیح داده ایم از تکرار مکررات پرهیز میکنیم.

سپس دیتا نرمالایز شده را با یک کلاس KMeans با K برابر ۵ fit می‌کنیم که خروجی را در تصویر زیر مشاهده میکنیم.



شکل۱۹) دسته های مربوط به دیتاست customer-mall

همانطور که گفته شد مشتری احتمال خرید دارد که درامد و spending-score اش بالا باشد، که در اینجا ما مشتری های دسته سبز رنگ مدنظر ما هستند حال برای اینکه دسته مورد نظر را مشخص کنیم بدین شکل عمل می‌کنیم. چون دیتا ها نرمالایز شده اند ما میگوییم هر دسته ای که مرکز آن مجموع درامد و احتمال خریدش از باقی مراکز بیشتر باشد آن کلاستر مد نظر ماست. پس ما یک لیست به اسم lst_sum تشکیل میدهیم که لیستی از مجموع درامد و احتمال خرید هر یک از مراکز است سپس ما ایندکس بزرگترین مجموع را با استفاده از تابع argmax بدست می‌آوریم که این ایندکس همان مرکز مدنظر ماست سپس در متغیر flag آن کلاستر مربوط به مرکزی که مشتریان راغب به خرید را تشکیل می‌دهند، ذخیره می‌کنیم. در انتها نیز آن کلاستر را پلات می‌کنیم.

In [24]:
```
lst_sum = np.sum(k1.centroids, axis=1)
lst_max_idx = np.argmax(np.asarray(lst_sum))
print(lst_sum)
print(lst_max_idx)
k1.centroids[lst_max_idx]
k1.clusters[lst_max_idx]
print(ds[k1.clusters[lst_max_idx]])
flag = ds[k1.clusters[lst_max_idx]]
```

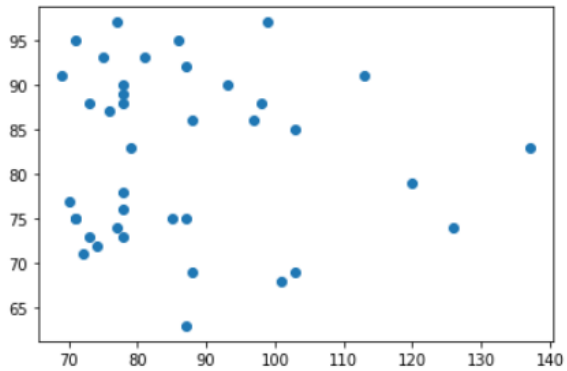
```
[-0.1968734  -2.43836518  2.2255011  -0.22723748 -0.2278314 ]
2
[[ 78  78]
 [ 77  74]
 [ 73  73]
 [ 75  93]
 [ 74  72]
 [ 87  63]
 [ 71  95]
 [137  83]
 [ 98  88]
 [126  74]
 [ 87  75]
 [ 88  69]
 [ 78  89]
 [ 73  88]
 [ 72  71]
 [ 86  95]
 [ 78  88]
 [ 76  87]
 [103  85]
 [ 71  75]
 [ 78  73]
 [ 71  75]
 [ 97  86]
 [113  91]
 [101  68]
 [ 85  75]
 [ 99  97]
 [ 81  93]
```

شکل ۲۰) تصویر مربوط به پیدا کردن مشتریان مدنظر

Plotting

In [25]:
```
index = []
value = []
for x,y in flag:
    index.append(x)
    value.append(y)
"""
    Selected customers
"""
plt.scatter(index, value)
```

Out[25]:
```
<matplotlib.collections.PathCollection at 0x1afa5bbe1f0>
```



شکل ۲۱) مشتریانی که احتمال خرید بالایی دارند که همان دسته سبز رنگ پلات قبلی هستند