نام و نام خانوادگی: سروش مهربان
شماره دانشجویی: 9628099

تمرین تحویلی چهارم داده کاوی

K-means

پیاده‌سازی تابع k-means به صورت زیر می‌باشد:

```
def k_means(X, k, n):
    """
    :param X: The matrix of input data, each row is a data point and each
column is a data feature
    :param k: the number of desired clusters
    :param n: the number of iteration that we want the k-means to run on the
data
    :return: Vector idx such that idx[i] is the index of the center assigned
to example data point i
    """
    centers = initialize_centers(X, k)
    for iteration in range(n):
        idx = find_closest_centers(X, centers)
        centers = compute_means(X, idx, k)

    return idx
```

همانطور که در دستورکار آمده است، در ابتدا ورودی هایی که دریافت میکند عبارتند از:

- ماتریس X که هرسطر یک داده و هر ستون یک ویژگی می‌باشد.
- عدد k که تعداد خوشه ها را نشان می‌دهد.
- عدد n که تعداد پیمایش هارا نشان می‌دهد.

برای پیاده سازی تابع مشابه شبه‌کد داده شده، ابتدا مراکز اولیه تولید خواهند شد. سپس در هر پیمایش ابتدا داده ها به خوشه اختصاص می‌یابند و در آرایه idx ریخته میشود. در این آرایه هر عنصر نشان میدهد داده متناظر به چه خوشه ای منتسب شده است. در ادامه خوشه ها آپدیت میشوند و این روند n بار تکرار میشود.

حال به پیاده سازی تابع اول که مقداردهی اولیه برای مراکز خوشه می‌باشد می‌پردازیم:

```
def initialize_centers(X, k):
    """
    Initialize random centers
    `centers` is a matrix with K rows, each row is one center and each column
is a feature
    """
    x_min, y_min = X.min(axis=0)
    x_max, y_max = X.max(axis=0)

    x_values = np.random.uniform(low=x_min, high=x_max, size=(k, 1))
    y_values = np.random.uniform(low=y_min, high=y_max, size=(k, 1))

    centers = np.hstack((x_values, y_values))
    return centers
```

در این تابع، خروجی یک ماتریس می‌باشد که k سطر (هرسطر برای یک مرکز) و با توجه به دیتاست ورودی ۲ تا ستون دارد که هر

ستون مقدار ویژگی که مختصات x و y می‌باشد را نشان میدهد. برای ساخت این ماتریس ابتدا یک بردار برای پیاده سازی مختصات x مراکز به صورت تصادفی یکنواخت و یک بردار برای مختصات y مراکز به صورت تصادفی یکنواخت می‌سازیم. به گونه ای که ابتدا مینیمم و ماکسیمم مختصات داده ها را حساب میکنیم و سپس بردار تصادفیx و y را با توجه به این مقادیر به صورت رندم یکنواخت تولید میکنیم. درآخر ماتریس مراکز را با قرار دادن این ۲ ستون در کنار هم درست میکنیم.

```
def find_closest_centers(X, centers):
    idx = []
    for entry in X:
        distances = [euclidean_distance(entry, center) for center in centers]
        idx.append(np.argmin(distances))
    return np.array(idx)
```

در تابع اختصاص داده ها به مراکز، به ازای هر سطر (entry) در ماتریس داده ها، فاصله اقلیدسی داده را نسبت به تمامی مراکز می‌سنجیم و در distances قرار می‌دهیم. در آخر به کمک argmin، ایندکس آن آرایه که کمترین فاصله را دارد (شماره مرکز) را در idx قرار میدهیم و باز میگردانیم.

```
def euclidean_distance(entry, center):
    return np.linalg.norm(entry - center)
```

برای محاسبه فاصله اقلیدسی نیز از np.linalg.norm کمک گرفته شده است.

```
def compute_means(X, idx, k):
    centers = []
    for i in range(k):
        cluster_entries = X[idx == i]
        if len(cluster_entries) == 0:
            new_center = initialize_centers(X, k=1).tolist()
        else:
            new_center = cluster_entries.mean(axis=0)
        centers.append(new_center)
    return np.array(centers)
```

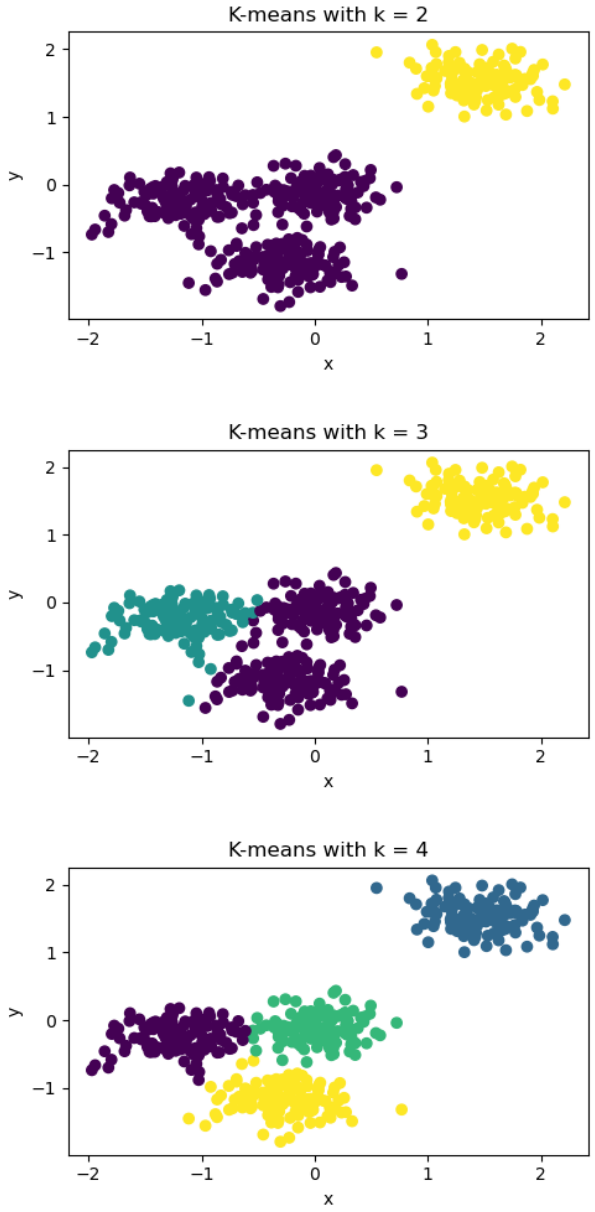
در آخر برای آپدیت کردن مراکز نیز به ازای هر k مرکز، ابتدا از داده هایی که در ماتریس X آن هایی که متعلق به خوشه i ام هستند را انتخاب میکنیم و در cluster_entries قرار می‌دهیم. سپس اگر این تعداد برابر با صفر باشد، یعنی خوشه ای بدون داده است و در این صورت به کمک initialize_centers با در نظر گرفتن k=1، یک مرکز جدید به صورت تصادفی ایجاد میکنیم. در غیر این صورت میانگین مختصاتx و y داده هارا گرفته و معادل با مختصات new_center قرار میدهیم و به لیست مراکز اضافه میکنیم. در انتها نیز آرایه numpy از این مراکز را باز میگردانیم.

```
if __name__ == '__main__':
    X = pd.read_csv('Dataset1.csv').to_numpy()


    fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(5, 10))
    for i, k in enumerate(range(2, 5)):
        idx = k_means(X, k, n=15)
        axes[i].scatter(X[:, 0], X[:, 1], c=idx)
        axes[i].set(title=f"K-means with k = {k}", xlabel='x', ylabel='y')
    plt.tight_layout(h_pad=3)
    plt.show()
```

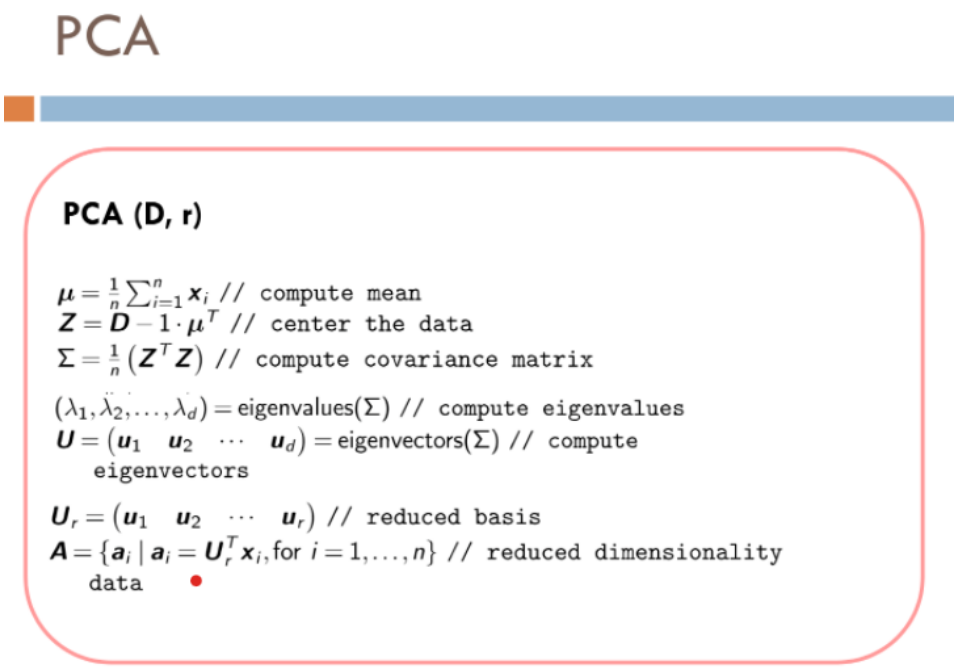
برای اجرای برنامه، ابتدا محتویات Dataset1.csv را میخوانیم و به کمک to_numpy به آرایه نامپای تبدیل میکنیم و در X قرار میدهیم. سپس با توجه به این که در دستورکار گفته شده است برای k=2,3,4 رسم شود، یک نمودار حاوی ۳ تا ax که همگی در

یک ستون قرار دارند میسازیم. در ادامه در یک حلقه به ازای k از ۲ تا ۴، به کمک scatter. داده هارا رسم میکنیم و برای رنگ آن ها نیز از idx که خروجی k_means می باشد (با ۱۵ پیمایش)، استفاده میکنیم. پس از اجرای کد خروجی مورد نظر برابر است با:



PCA

طبق اسلاید درس، الگوریتم PCA به صورت زیر می‌باشد:



لذا هنگام پیاده سازی نیز به نحو فوق عمل میکنیم:

```
def pca(D, r):
    n = D.shape[0]

    miu = D.mean(axis=0)
    Z = D - miu
    cov_matrix = (1 / n) * (Z.T @ Z)
    eigen_values, eigen_vectors = np.linalg.eig(cov_matrix)
    Ur = eigen_vectors[:, :r]

    A = (Ur.T @ D.T).T
    return A
```

با توجه به تابع فوق، ابتدا تعداد سطر ماتریس D که تعداد داده ها می‌باشند را به کمک shape[0]. استخراج میکنیم و در n قرار می‌دهیم. سپس μ را با میانگین گرفتن روی سطر ها انجام می‌دهیم. ماتریسZ را با کم کردن μ از ماتریسD می‌سازیم تا داده‌ها روی مرکز قرار بگیرند. ماتریس کواریانس را طبق رابطه داده شده در اسلاید محاسبه میکنیم و در ادامه به کمک np.lianlg.eig، مقادیر و بردارهای ویژه ماتریس کواریانس را بدست می‌آوریم. با توجه به این که خروجی متد eig، مقادیر ویژه را از بزرگ به کوچک قرار میدهد و بردار های ویژه نظیر آن را نیز در ماتریسeigen_vectors برمیگرداند، تنها کافیست r تای اول (در دستور کار n_components اول) از بردارهای ویژه را انتخاب و در ماتریس U_r قرار دهیم.

با توجه به این که دیتاست ما 150 تا سطر و ۴ تا ویژگی دارد، ماتریس D به صورت 150x4 می‌باشد. بردارهای ویژه نیز هرکدام 4x1 می‌باشند که اگر r=2تای آن را کنار هم قرار دهیم، ماتریس U_r به صورت 4x2 می‌باشد. در انتهای اسلاید برای محاسبه ماتریس A، بدین صورت عمل شده است که ترانهاده ماترسی U_r روی بردار x_i ضرب شده است. اما در ماتریس D، بردارها در سطرها قرار گرفته اند. پس به کمک D.T ترانهاده ماتریس را محاسبه میکنیم که بردارها در ستون قرار گرفته باشند. پس D.T به صورت 4x150 میباشد و Ur.T نیز 2x4. پس حاصل ضرب آن ها 2x150 میباشد که آن را نیز اگر ترانهاده کنیم 150x2 حاصل میشود که ماتریس A می‌باشد. این ماتریس حاوی ۱۵۰ سطر دارد که هرکدام یک داده را نمایش میدهند و ۲ ستون دارد که هرکدام، principal components مورد انتخابی می‌باشند. پس این ماتریس را باز میگردانیم.

```
if __name__ == '__main__':
    df = pd.read_excel("dataset2.xlsx")
    data = df.drop('class', axis=1).to_numpy()
    classes = df['class'].to_numpy()

    scaler = StandardScaler()
    scaler.fit(data)
    D = scaler.transform(data)

    A = pca(D, r=2)
    X_train, X_test, y_train, y_test = train_test_split(A, classes,
test_size=0.2)

    clf = SVC(kernel='rbf')
    clf.fit(X_train, y_train)

    train_accuracy = clf.score(X_train, y_train)
    test_accuracy = clf.score(X_test, y_test)

    print(f"Train accuracy: {train_accuracy * 100:.2f}%")
    print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

در قسمت اصلی کد، ابتدا دیتاست را خوانده و در دیتافریم df قرار میدهیم. سپس تمامی ستون ها به جز ستون class را به صورت ndarray داخل data قرار میدهیم و ستون class نیز به صورت ndarray در classes قرار میدهیم. به کمک StandardScaler، تمامی داده‌ها را scale میکنیم که در یک بازه بین [1 ,-1] قرار بگیرند. در آخر به کمک تابع pca ساخته شده ماتریس A را حاصل میکنیم.

برای تست عملکرد، به کمک train_test_split به صورت نسبت ۸۰ به ۲۰، داده های آموزشی و تست را درست میکنیم. به کمک SVM و با تابع کرنل rbf، مدل را آموزش میدهیم و به کمک score. نیز accuracy های مورد نظر را بدست می آوریم و گزارش میکنیم. یک نمونه accuracy بدست آمده پس از خروجی گرفتن برابر است با:

```
Train accuracy: 93.33%
Test accuracy: 90.00%
```