Лабораторная работа №9. Кластеризация

Цель лабораторной работы: научится производить кластерный анализ данных на основе метода K-средних.

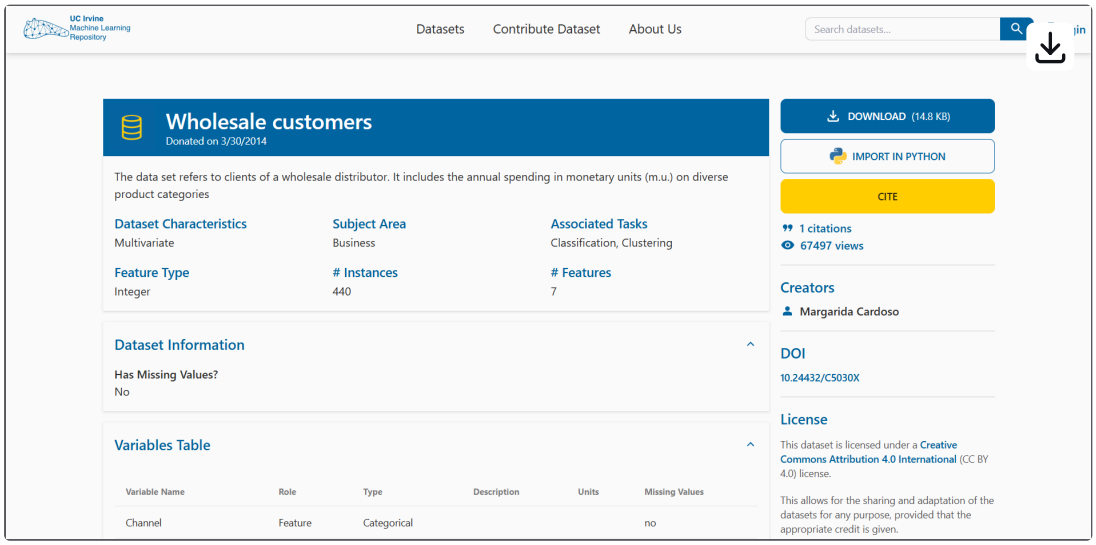
Основные задачи:

- получение навыков рефакторинга кода в проектах машинного обучения;
- изучение принципов определения оптимального количества кластеров в методах кластерного анализа;
- изучение возможностей языка Python для реализации кластерного анализа.

Ход выполнения индивидуального задания:

1. Выбор набора данных для задачи кластеризации

В качестве данных для лабораторной работы был выбран набор под названием **Wholesale customers**, который входит в репозиторий **UCI Machine Learning Repository**.



2. Построение модели кластеризации с использованием метода K-means

2.1 Загрузка, предобработка и визуализация набора данных

Первым делом импортируем все необходимые библиотеки для выполнения лабораторной работы, а также загрузим наш набор данных с помощью **pandas** и выведем первые пять строк:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

data = pd.read_csv('Wholesale_customers_data.csv')

print(data.head())
```

Как результат получаем следующий вывод:

```
   Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicassen
0        2       3  12669  9656     7561     214              2674        1338
1        2       3   7057  9810     9568    1762              3293        1776
2        2       3   6353  8808     7684    2405              3516        7844
3        1       3  13265  1196     4221    6404               507        1788
4        2       3  22615  5410     7198    3915              1777        5185
```

Перед применением метода *k-means* необходимо выполнить некоторую **предобработку данных**. Нам нужно **удалить** из набора все **категориальные признаки**, так как они **не используются** в кластеризации, а также применить **масштабирование** признаков для лучшей работы метода **k-means**:

```
X = data.drop(['Channel', 'Region'], axis=1)

print(X.head())
print("\n----------------------------------------------------------------
--------")

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("\nПромасштабированные признаки:")
print(X_scaled[:5])
```

```
   Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicassen
0  12669  9656     7561     214              2674        1338
1   7057  9810     9568    1762              3293        1776
2   6353  8808     7684    2405              3516        7844
3  13265  1196     4221    6404               507        1788
4  22615  5410     7198    3915              1777        5185

--------------------------------------------------------------------

Промасштабированные признаки:
[[ 0.05293319  0.52356777 -0.04111489 -0.58936716 -0.04356873 -0.06633906]
 [-0.39130197  0.54445767  0.17031835 -0.27013618  0.08640684  0.08915105]
 [-0.44702926  0.40853771 -0.0281571  -0.13753572  0.13323164  2.24329255]
 [ 0.10011141 -0.62401993 -0.3929769   0.6871443  -0.49858822  0.09341105]
 [ 0.84023948 -0.05239645 -0.07935618  0.17385884 -0.23191782  1.29934689]]
```

2.2 Определение оптимального количества кластеров

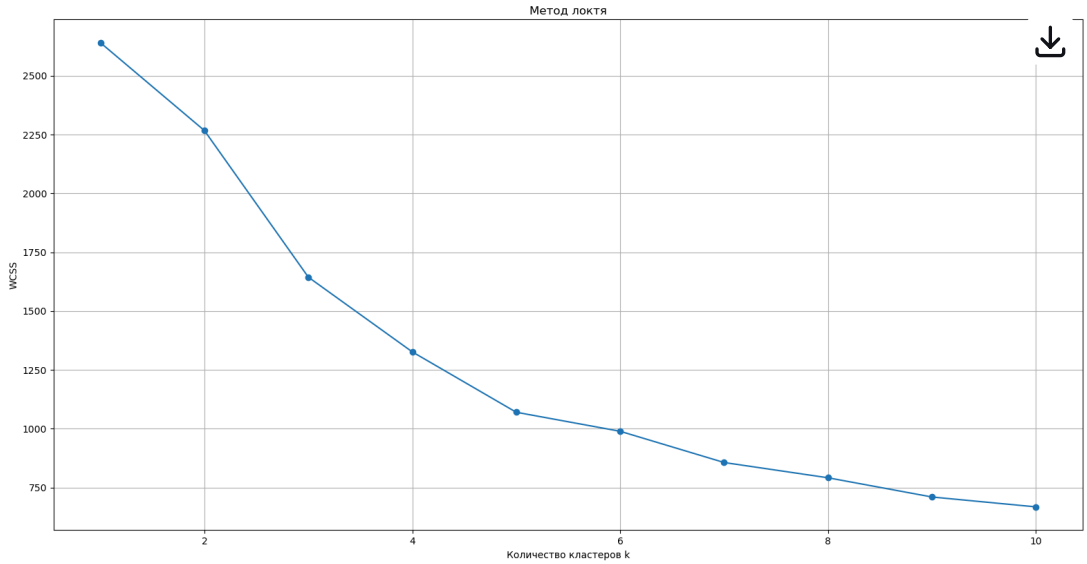
Следующим этапом является **определение оптимального числа кластеров** для нашей модели. Для этого воспользуемся **методом локтя (Elbow Method)** — одним из **самых популярных** методов для определения оптимального количества кластеров в задачах кластеризации.

Чтобы реализовать метод локтя мы запускаем алгоритм **k-means** для разного количества кластеров **k** (от 1 до 10) и для каждого из них вычисляем **сумму квадратов внутрикластерных расстояний (WCSS)**. Это мера разброса точек внутри кластеров — чем меньше, тем более плотные кластеры. Далее строим **график зависимости WCSS** от **k** и ищем “локоть” на графике — точку, после которой уменьшение WCSS становится менее заметным. Это точка и будет обозначать **оптимальное количество кластеров** для модели.

```
k_range = range(1, 11)
wcss = []

for i in k_range:
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state =
42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(k_range, wcss, marker='o')
plt.xlabel('Количество кластеров k')
plt.ylabel('WCSS')
plt.title('Метод локтя')
plt.grid()
```



Смотря на график, можно сказать, что изменение значения WCSS становится менее заметным после **k = 5**. Следовательно, **оптимальным**

количеством кластеров для нашей модели будет **5**.

2.3 Обучение модели кластеризации для оптимального количества кластеров и визуализация результатов

Обучим нашу модель кластеризации на **оптимальном количестве кластеров** и добавим метки кластеров в исходный набор данных:

```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X_scaled)

data['y_kmeans'] = y_kmeans
```

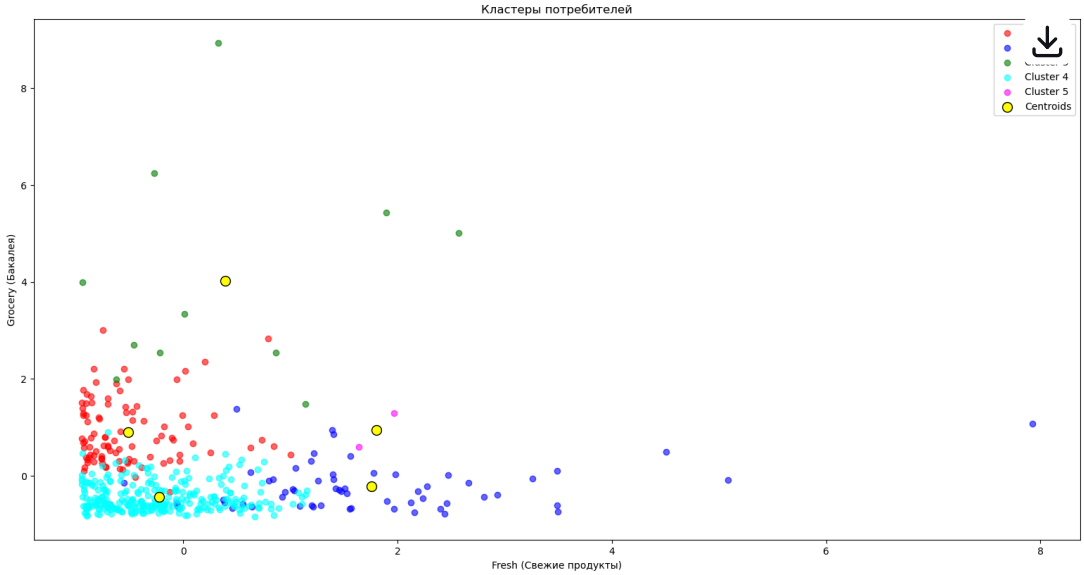
Теперь, чтобы понять насколько хорошо или плохо модель кластеризации выполнила работу, **визуализируем** результаты её работы. Для этого выведем графики некоторых признаков, которые будут разбиты на кластеры:

```
X_vis = X_scaled[:, [0, 2]]

plt.figure(figsize=(10, 6))
colors = ['red', 'blue', 'green', 'cyan', 'magenta']
for i in range(5):
    plt.scatter(X_vis[y_kmeans == i, 0], X_vis[y_kmeans == i, 1],
                c=colors[i], label=f'Cluster {i+1}', alpha=0.6)

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
2],
            s=100, c='yellow', edgecolors='black', label='Centroids')

plt.title('Кластеры потребителей')
plt.xlabel('Fresh (Свежие продукты)')
plt.ylabel('Grocery (Бакалея)')
plt.legend()
```

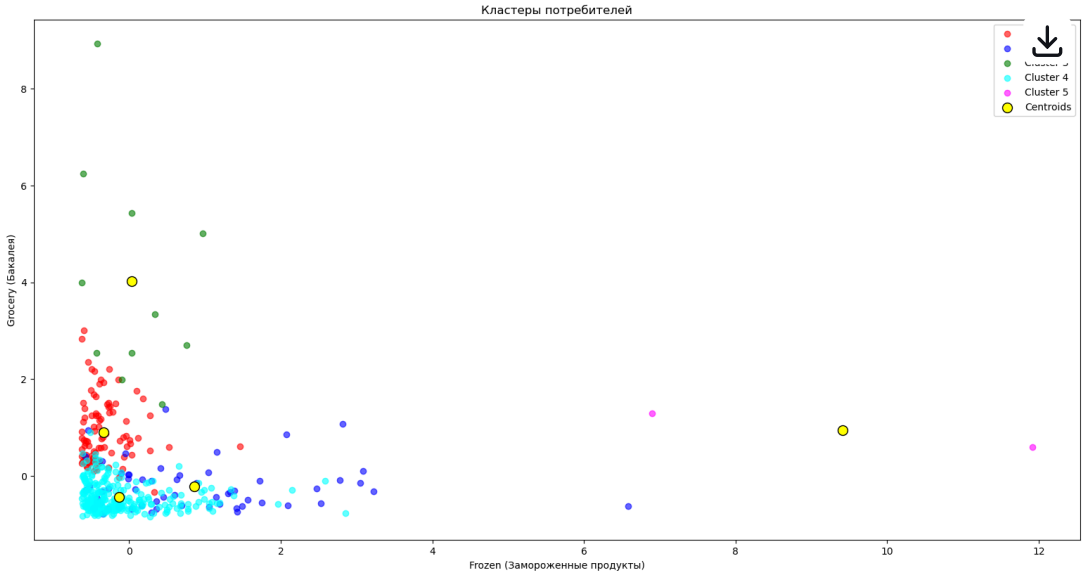



```
X_vis = X_scaled[:, [3, 2]]
```

```
plt.figure(figsize=(10, 6))
colors = ['red', 'blue', 'green', 'cyan', 'magenta']
for i in range(5):
    plt.scatter(X_vis[y_kmeans == i, 0], X_vis[y_kmeans == i, 1],
                c=colors[i], label=f'Cluster {i+1}', alpha=0.6)

plt.scatter(kmeans.cluster_centers_[:, 3], kmeans.cluster_centers_[:,
2],
            s=100, c='yellow', edgecolors='black', label='Centroids')

plt.title('Кластеры потребителей')
plt.xlabel('Frozen (Замороженные продукты)')
plt.ylabel('Grocery (Бакалея)')
plt.legend()
```

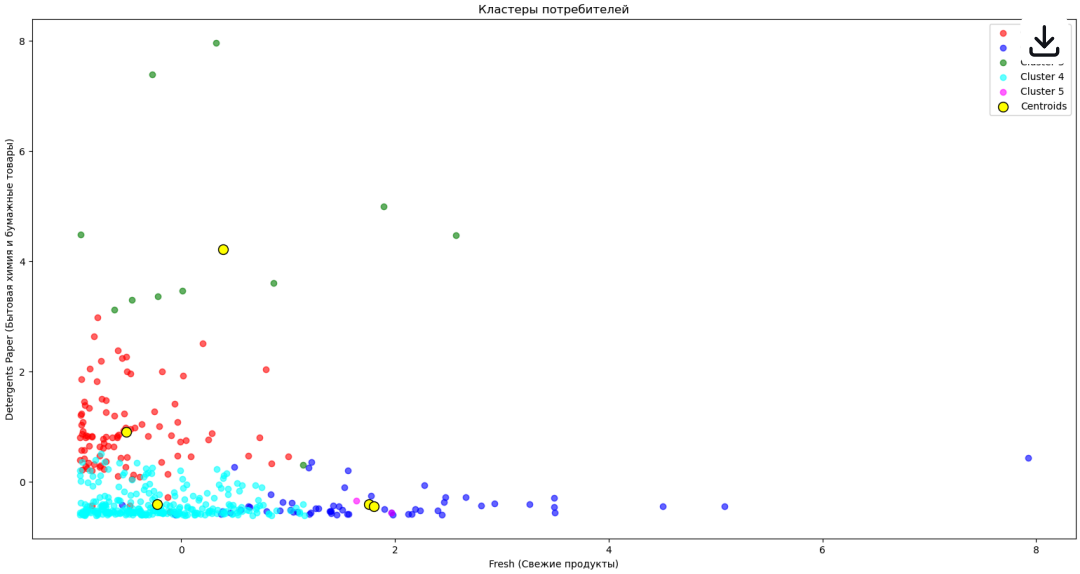



```
X_vis = X_scaled[:, [0, 4]]

plt.figure(figsize=(10, 6))
colors = ['red', 'blue', 'green', 'cyan', 'magenta']
for i in range(5):
    plt.scatter(X_vis[y_kmeans == i, 0], X_vis[y_kmeans == i, 1],
                c=colors[i], label=f'Cluster {i+1}', alpha=0.6)

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
4],
            s=100, c='yellow', edgecolors='black', label='Centroids')

plt.title('Кластеры потребителей')
plt.xlabel('Fresh (Свежие продукты)')
plt.ylabel('Detergents Paper (Бытовая химия и бумажные товары)')
plt.legend()
```



Как можно заметить, графики могут содержать в себе только 2 признака. Такие графики не являются очень показательными, так как они могут упускать важные детали. Наш набор данных обладает 6 разными признаками (Fresh, Milk, Grocery, Frozen, Detergents_Paper, Delicassen). Но построить 6D-плоскость, где мы можем учитывать каждый признак, нам не представляется возможным. Именно для такого случая существует **PCA (Principal Component Analysis)** — метод уменьшения размерности данных.

PCA берёт все признаки набора данных и сжимает их в 2 главных компонента, которые сохраняют максимум информации. Уже с помощью PCA мы можем построить график кластеризации для множества признаков в 2D-плоскости.

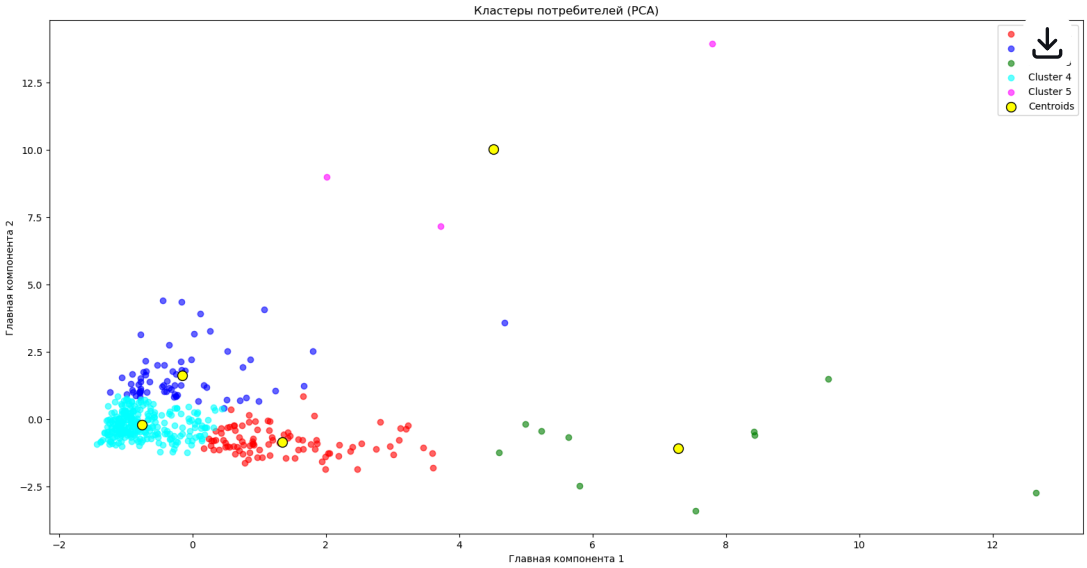
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X_pca)

plt.figure(figsize=(10, 6))
colors = ['red', 'blue', 'green', 'cyan', 'magenta']
for i in range(5):
    plt.scatter(X_pca[y_kmeans == i, 0], X_pca[y_kmeans == i, 1],
                c=colors[i], label=f'Cluster {i+1}', alpha=0.6)

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1],
            s=100, c='yellow', edgecolors='black', label='Centroids')

plt.title('Кластеры потребителей (PCA)')
plt.xlabel('Главная компонента 1')
plt.ylabel('Главная компонента 2')
plt.legend()
```



Контрольные вопросы

1. **Кластерный анализ** — это метод машинного обучения, используемый для **группировки** объектов на основе их схожести. В результате данные делятся на **кластеры**, внутри которых элементы **максимально похожи**, а между кластерами — **различаются**.
2. Известные методы кластерного анализа:
   - **K-Means** – делит данные на kkk кластеров, минимизируя расстояние до центроидов.
   - **Hierarchical Clustering (иерархическая кластеризация)** – строит древовидную структуру кластеров.
   - **DBSCAN** – группирует плотные области данных, игнорируя выбросы.
   - **Mean-Shift** – использует метод "сдвига среднего" для поиска кластеров.
   - **Gaussian Mixture Model (GMM)** – основан на вероятностном разбиении данных.
3. Классы и функции Python для кластерного анализа:
   - `KMeans` – метод K-средних
   - `AgglomerativeClustering` – иерархическая кластеризация
   - `DBSCAN` – метод DBSCAN
   - `MeanShift` – метод сдвига среднего
   - `GaussianMixture` – вероятностная модель GMM
   - `sklearn.metrics.silhouette_score` – оценка качества кластеризации
   - `sklearn.decomposition.PCA` – для снижения размерности перед кластеризацией
4. **Метод локтя (Elbow Method)**: Строится график зависимости суммы квадратов расстояний до центроидов (**WCSS**) от числа кластеров. Оптимальное значение — точка "изгиба" графика.

Коэффициент силуэта (Silhouette Score): Оценивает, насколько хорошо объекты внутри одного кластера схожи и насколько сильно отличаются от других кластеров. Чем выше, тем лучше.

Метод Гауссовых смесей (GMM) и BIC/AIC: Оценка вероятностной модели для выбора числа кластеров.

5. Эти методы решают принципиально разные задачи:
   - **Регрессия** (например, линейная регрессия) предсказывает **непрерывные значения** (цена, температура). Пример: прогнозирование стоимости дома на основе его характеристик.
   - **Классификация** (например, логистическая регрессия или SVM) предсказывает **дискретные метки** (классы). Пример: определение, является ли email спамом.
   - **Кластеризация** (например, K-means) **не использует метки** и выявляет скрытые структуры данных. Пример: разделение клиентов на группы по purchasing behavior.