**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Машинное обучение»**

Тема: Кластеризация (DBSCAN, OPTICS)

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Кирьянов Д.И. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2021

**Цель работы.**

Ознакомится с методами кластеризации модуля Sklearn.

**Ход работы.**

**Загрузка данных**

Был загружен исходный набор данных, убрав столбец с метками и откинув наблюдения с пропущенными значениями. Результат работы представлен на рисунке 1.

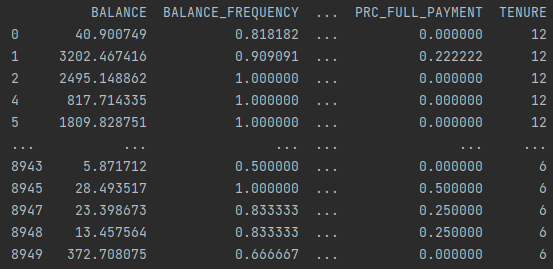


Рисунок 1 – Загруженные данные.

**DBSCAN**

Проведена кластеризацию методов DBSCAN при параметрах по умолчанию. Выведены метки кластеров, количество кластеров, а также процент наблюдений, которые кластеризовать не удалось.

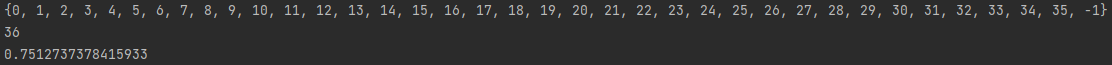


Рисунок 2 – Информация о кластерах

Параметры DBSCAN:

eps – максимальное расстояние между двумя элементами.

min\_samples – число элементов в окрестности точки, чтобы считать ее основной.

metric – метрика для расчета расстояния между элементами.

metric\_params – дополнительные параметры для метрики.

algorithm – алгоритм, который будет использоваться для вычисления точечных расстояний и поиска ближайших соседей.

leaf\_size – может повлиять на скорость конструкции и запроса, а также на, требуемую для построения дерева, память.

p – степень метрики Миньковского, которая будет использоваться для вычисления расстояния между точками.

n\_jobs – количество процессов для распараллеливания.

*Parameters  
----------  
eps : float, default=0.5  
 The maximum distance between two samples for one to be considered  
 as in the neighborhood of the other. This is not a maximum bound  
 on the distances of points within a cluster. This is the most  
 important DBSCAN parameter to choose appropriately for your data set  
 and distance function.  
  
min\_samples : int, default=5  
 The number of samples (or total weight) in a neighborhood for a point  
 to be considered as a core point. This includes the point itself.  
  
metric : str, or callable, default='euclidean'  
 The metric to use when calculating distance between instances in a  
 feature array. If metric is a string or callable, it must be one of  
 the options allowed by :func:`sklearn.metrics.pairwise\_distances` for  
 its metric parameter.  
 If metric is "precomputed", X is assumed to be a distance matrix and  
 must be square. X may be a :term:`Glossary <sparse graph>`, in which  
 case only "nonzero" elements may be considered neighbors for DBSCAN.  
  
 .. versionadded:: 0.17  
 metric \*precomputed\* to accept precomputed sparse matrix.  
  
metric\_params : dict, default=None  
 Additional keyword arguments for the metric function.  
  
 .. versionadded:: 0.19  
  
algorithm : {'auto', 'ball\_tree', 'kd\_tree', 'brute'}, default='auto'  
 The algorithm to be used by the NearestNeighbors module  
 to compute pointwise distances and find nearest neighbors.  
 See NearestNeighbors module documentation for details.  
  
leaf\_size : int, default=30  
 Leaf size passed to BallTree or cKDTree. This can affect the speed  
 of the construction and query, as well as the memory required  
 to store the tree. The optimal value depends  
 on the nature of the problem.  
  
p : float, default=None  
 The power of the Minkowski metric to be used to calculate distance  
 between points. If None, then ``p=2`` (equivalent to the Euclidean  
 distance).  
  
n\_jobs : int, default=None  
 The number of parallel jobs to run.  
 ``None`` means 1 unless in a :obj:`joblib.parallel\_backend` context.  
 ``-1`` means using all processors. See :term:`Glossary <n\_jobs>`  
 for more details.*

Построен график количества кластеров и процента не кластеризованных наблюдений в зависимости от максимальной рассматриваемой дистанции между наблюдениями.

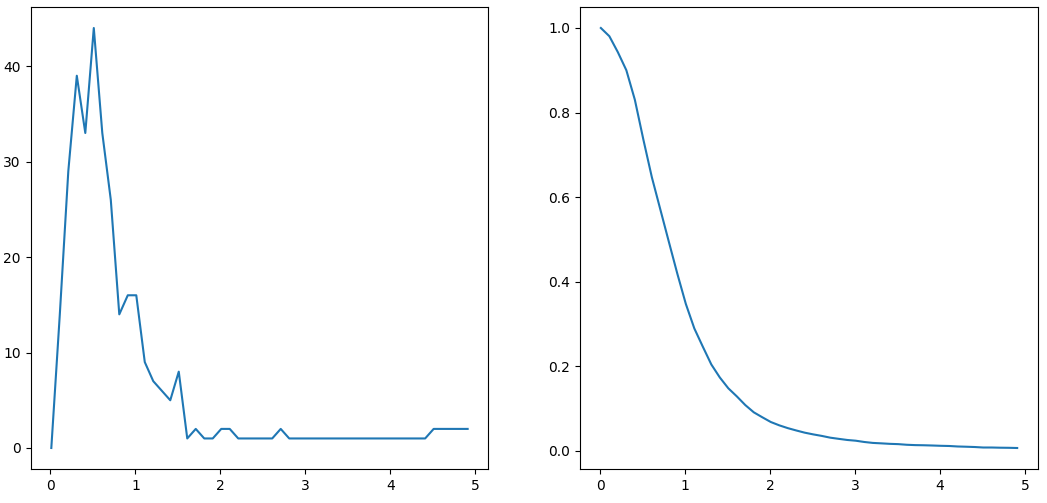


Рисунок 3 – График зависимости кол-ва кластеров и процента не кластеризованных наблюдений от макс. дистанции.

Построен график количества кластеров и процента не кластеризованных наблюдений в зависимости от минимального значения количества точек, образующих кластер.

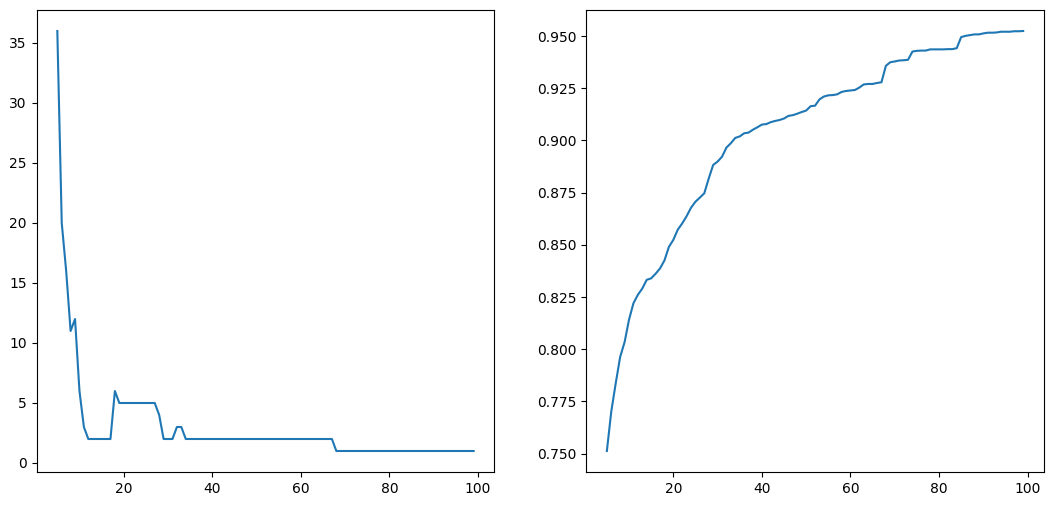


Рисунок 4 – График зависимости кол-ва кластеров и процента не класт. данных от мин. значения кол-ва точек, образующих кластер.

Определены значения минимального кол-ва точек и максимального расстояния при которых кол-во кластеров равно 6, а процент не кластеризованных наблюдений не превышает 12. Полученные величины: eps = 2, min\_samples = 3.

Понижена размерность данных до 2, используя метод главных компонент. Визуализированы результаты кластеризации, полученные в пункте 6.

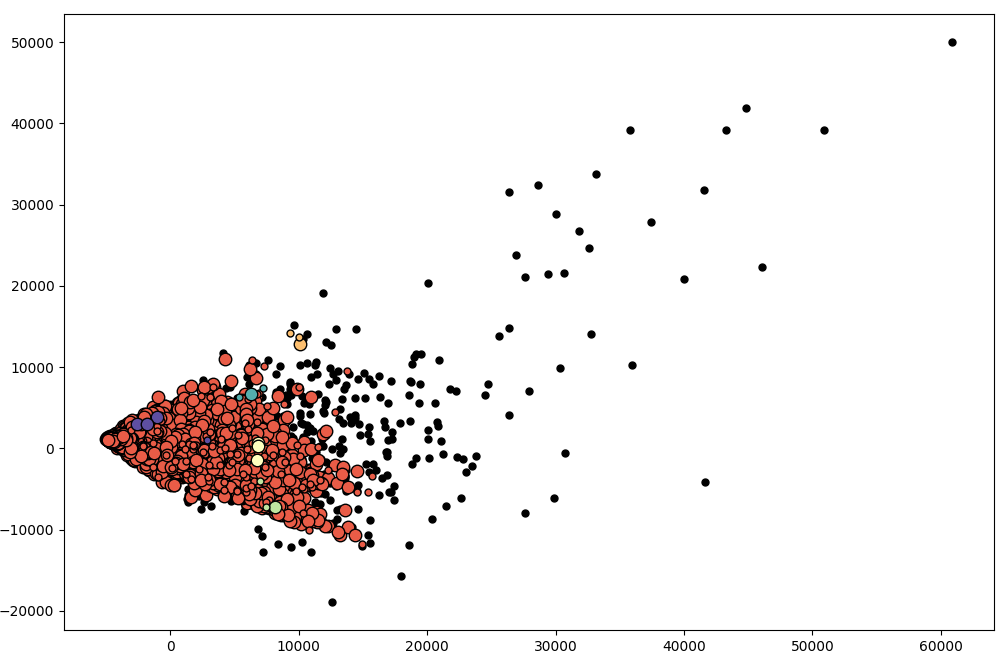


Рисунок 5 – Визуализация результатов кластеризации

**OPTICS**

Параметры OPTICS:

min\_samples – число элементов в окрестности точки, чтобы она считалась основной.

max\_eps – максимальное расстояние между элементами, чтобы они считались соседними.

metric – метрика для расчета расстояния между элементами.

p – параметр для метрики Миньковского.

metric\_params – дополнительные параметры для метрики.

cluster\_method – метод извлечения кластеров

eps – расстояние между элементами, чтобы они считались соседними. По умолчанию соответствует max\_eps.

xi – определяет минимальную крутизну графика достижимости, которая составляет границу кластера.

predecessor\_correction – коррекция кластеров в соответствии с предшественниками.

min\_cluster\_size – минимальное количество элементов в кластере OPTICS.

algorithm – алгоритм для поиска ближайших соседей.

leaf\_size – может повлиять на скорость конструкции и запроса, а также на, требуемую для построения дерева, память.

memory – используется для кеширования вывода вычисления дерева.

n\_jobs – количество параллельных процессов для поиска соседей.

*Parameters  
----------  
min\_samples : int > 1 or float between 0 and 1, default=5  
 The number of samples in a neighborhood for a point to be considered as  
 a core point. Also, up and down steep regions can't have more than  
 ``min\_samples`` consecutive non-steep points. Expressed as an absolute  
 number or a fraction of the number of samples (rounded to be at least  
 2).  
  
max\_eps : float, default=np.inf  
 The maximum distance between two samples for one to be considered as  
 in the neighborhood of the other. Default value of ``np.inf`` will  
 identify clusters across all scales; reducing ``max\_eps`` will result  
 in shorter run times.  
  
metric : str or callable, default='minkowski'  
 Metric to use for distance computation. Any metric from scikit-learn  
 or scipy.spatial.distance can be used.  
  
 If metric is a callable function, it is called on each  
 pair of instances (rows) and the resulting value recorded. The callable  
 should take two arrays as input and return one value indicating the  
 distance between them. This works for Scipy's metrics, but is less  
 efficient than passing the metric name as a string. If metric is  
 "precomputed", X is assumed to be a distance matrix and must be square.  
  
 Valid values for metric are:  
  
 - from scikit-learn: ['cityblock', 'cosine', 'euclidean', 'l1', 'l2',  
 'manhattan']  
  
 - from scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev',  
 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski',  
 'mahalanobis', 'minkowski', 'rogerstanimoto', 'russellrao',  
 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean',  
 'yule']  
  
 See the documentation for scipy.spatial.distance for details on these  
 metrics.  
  
p : int, default=2  
 Parameter for the Minkowski metric from  
 :class:`~sklearn.metrics.pairwise\_distances`. When p = 1, this is  
 equivalent to using manhattan\_distance (l1), and euclidean\_distance  
 (l2) for p = 2. For arbitrary p, minkowski\_distance (l\_p) is used.  
  
metric\_params : dict, default=None  
 Additional keyword arguments for the metric function.  
  
cluster\_method : str, default='xi'  
 The extraction method used to extract clusters using the calculated  
 reachability and ordering. Possible values are "xi" and "dbscan".  
  
eps : float, default=None  
 The maximum distance between two samples for one to be considered as  
 in the neighborhood of the other. By default it assumes the same value  
 as ``max\_eps``.  
 Used only when ``cluster\_method='dbscan'``.  
  
xi : float between 0 and 1, default=0.05  
 Determines the minimum steepness on the reachability plot that  
 constitutes a cluster boundary. For example, an upwards point in the  
 reachability plot is defined by the ratio from one point to its  
 successor being at most 1-xi.  
 Used only when ``cluster\_method='xi'``.  
  
predecessor\_correction : bool, default=True  
 Correct clusters according to the predecessors calculated by OPTICS  
 [2]\_. This parameter has minimal effect on most datasets.  
 Used only when ``cluster\_method='xi'``.  
  
min\_cluster\_size : int > 1 or float between 0 and 1, default=None  
 Minimum number of samples in an OPTICS cluster, expressed as an  
 absolute number or a fraction of the number of samples (rounded to be  
 at least 2). If ``None``, the value of ``min\_samples`` is used instead.  
 Used only when ``cluster\_method='xi'``.  
  
algorithm : {'auto', 'ball\_tree', 'kd\_tree', 'brute'}, default='auto'  
 Algorithm used to compute the nearest neighbors:  
  
 - 'ball\_tree' will use :class:`BallTree`  
 - 'kd\_tree' will use :class:`KDTree`  
 - 'brute' will use a brute-force search.  
 - 'auto' will attempt to decide the most appropriate algorithm  
 based on the values passed to :meth:`fit` method. (default)  
  
 Note: fitting on sparse input will override the setting of  
 this parameter, using brute force.  
  
leaf\_size : int, default=30  
 Leaf size passed to :class:`BallTree` or :class:`KDTree`. This can  
 affect the speed of the construction and query, as well as the memory  
 required to store the tree. The optimal value depends on the  
 nature of the problem.  
  
memory : str or object with the joblib.Memory interface, default=None  
 Used to cache the output of the computation of the tree.  
 By default, no caching is done. If a string is given, it is the  
 path to the caching directory.  
  
n\_jobs : int, default=None  
 The number of parallel jobs to run for neighbors search.  
 ``None`` means 1 unless in a :obj:`joblib.parallel\_backend` context.  
 ``-1`` means using all processors. See :term:`Glossary <n\_jobs>`  
 for more details.*

Атрибуты OPTICS:

labels\_ - метки кластера для каждой точки в датасете, заданной для fit().

reachability\_ - расстояния достижимости для каждой выборки, индексированные по порядку объектов.

ordering\_ - упорядоченный список выборочных индексов кластера.

core\_distances\_ - расстояние, на котором каждая выборка становится центральной точкой, индексируется по порядку объектов.

predecessor\_ - точка, из которой была получена выборка, индексируется по порядку объектов.

cluster\_hierarchy\_ - список кластеров вида [начало, конец] в каждой строке, включая все индексы.

n\_features\_in\_ - количество деталей, видимых во время посадки.

feature\_names\_in\_ - названия особенностей, замеченных во время посадки. Определяется только, если X имеет имена функций, являющихся строками.

*Attributes  
----------  
labels\_ : ndarray of shape (n\_samples,)  
 Cluster labels for each point in the dataset given to fit().  
 Noisy samples and points which are not included in a leaf cluster  
 of ``cluster\_hierarchy\_`` are labeled as -1.  
  
reachability\_ : ndarray of shape (n\_samples,)  
 Reachability distances per sample, indexed by object order. Use  
 ``clust.reachability\_[clust.ordering\_]`` to access in cluster order.  
  
ordering\_ : ndarray of shape (n\_samples,)  
 The cluster ordered list of sample indices.  
  
core\_distances\_ : ndarray of shape (n\_samples,)  
 Distance at which each sample becomes a core point, indexed by object  
 order. Points which will never be core have a distance of inf. Use  
 ``clust.core\_distances\_[clust.ordering\_]`` to access in cluster order.  
  
predecessor\_ : ndarray of shape (n\_samples,)  
 Point that a sample was reached from, indexed by object order.  
 Seed points have a predecessor of -1.  
  
cluster\_hierarchy\_ : ndarray of shape (n\_clusters, 2)  
 The list of clusters in the form of ``[start, end]`` in each row, with  
 all indices inclusive. The clusters are ordered according to  
 ``(end, -start)`` (ascending) so that larger clusters encompassing  
 smaller clusters come after those smaller ones. Since ``labels\_`` does  
 not reflect the hierarchy, usually  
 ``len(cluster\_hierarchy\_) > np.unique(optics.labels\_)``. Please also  
 note that these indices are of the ``ordering\_``, i.e.  
 ``X[ordering\_][start:end + 1]`` form a cluster.  
 Only available when ``cluster\_method='xi'``.  
  
n\_features\_in\_ : int  
 Number of features seen during :term:`fit`.  
  
 .. versionadded:: 0.24  
  
feature\_names\_in\_ : ndarray of shape (`n\_features\_in\_`,)  
 Names of features seen during :term:`fit`. Defined only when `X`  
 has feature names that are all strings.*

Найдены такие параметры метода OPTICS, при которых получается результат близкий к результату DBSCAN из пункта 6. Полученные величины: eps = 2, min\_samples = 3, cluster\_method = dbscan.

Визуализирован полученный результат, а также построен график достижимости.

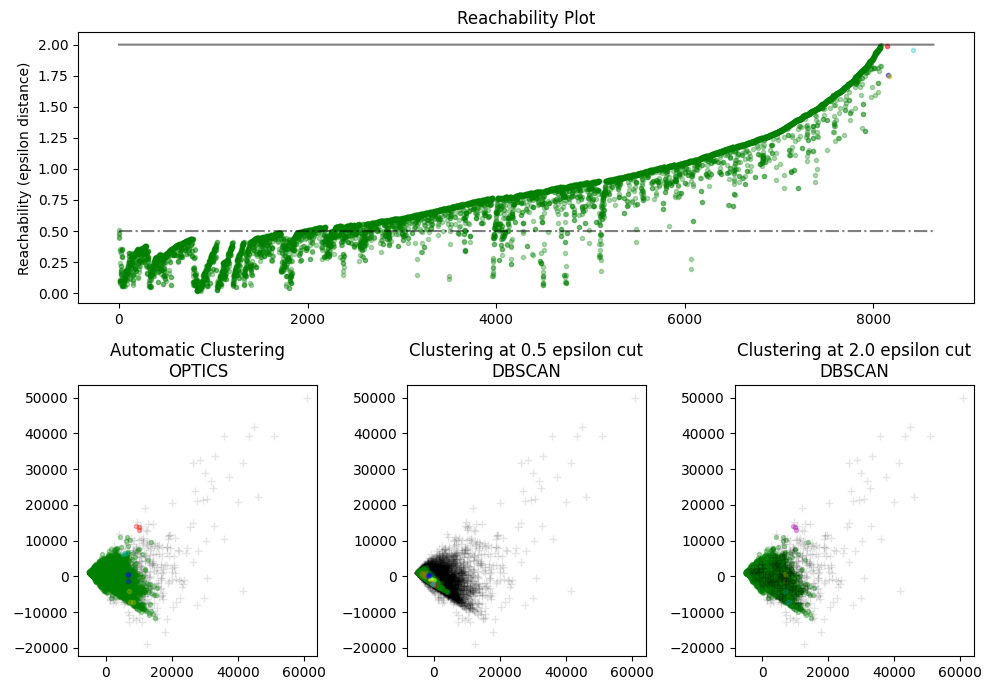


Рисунок 6 – Визуализированный результат.

Исследована работа метода OPTICS с использованием различных метрик (выбрано не менее 5 метрик).

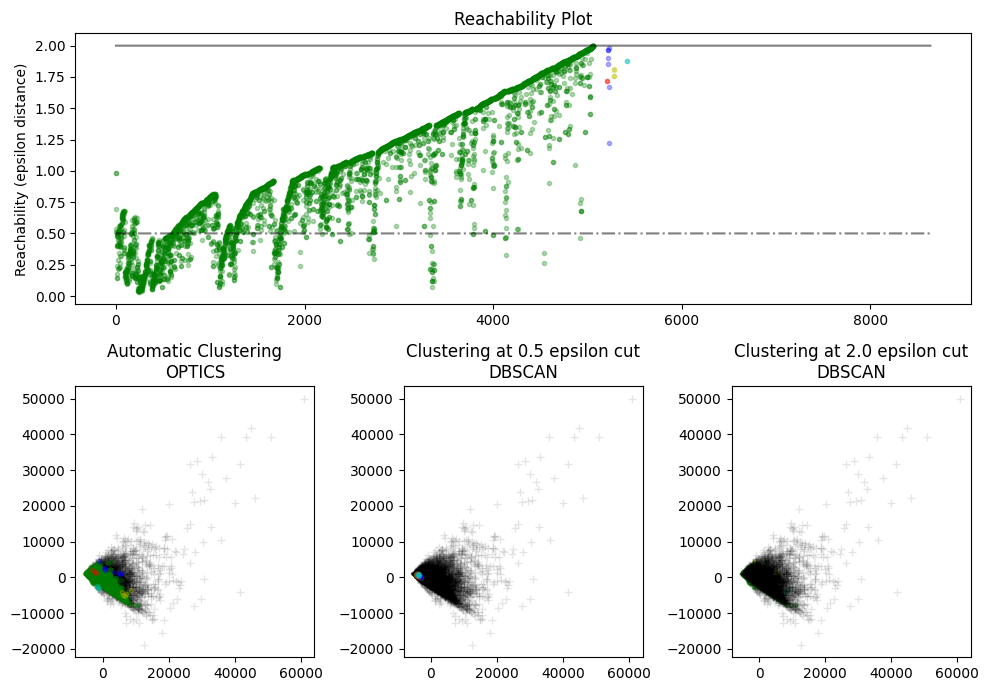


Рисунок 7 – График достижимости при использовании метрики “cityblock”

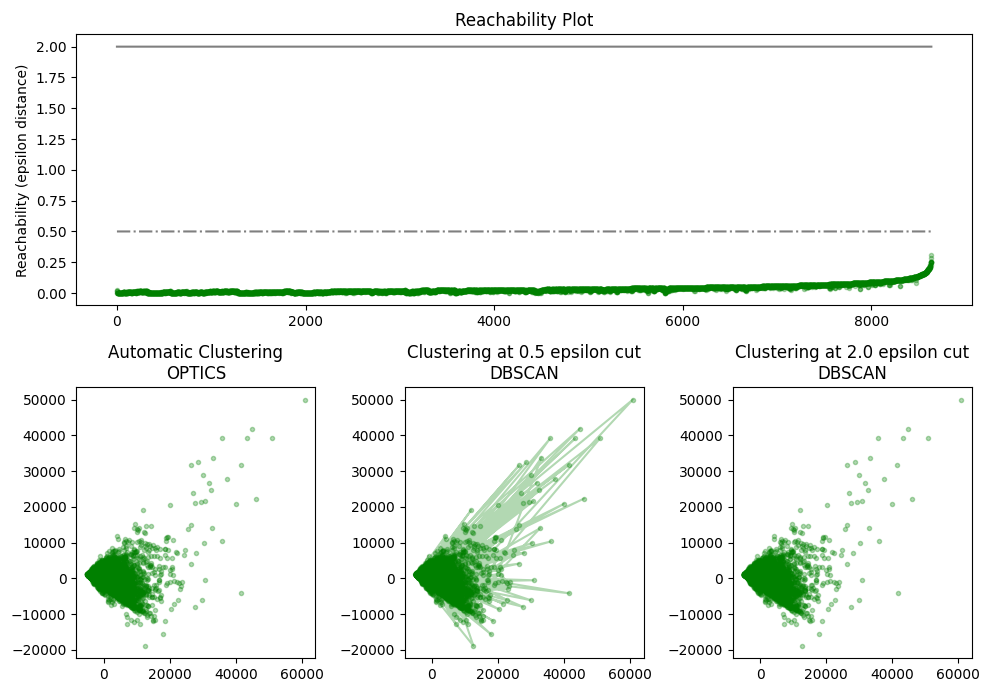


Рисунок 8 – График достижимости при использовании метрики “cosine”

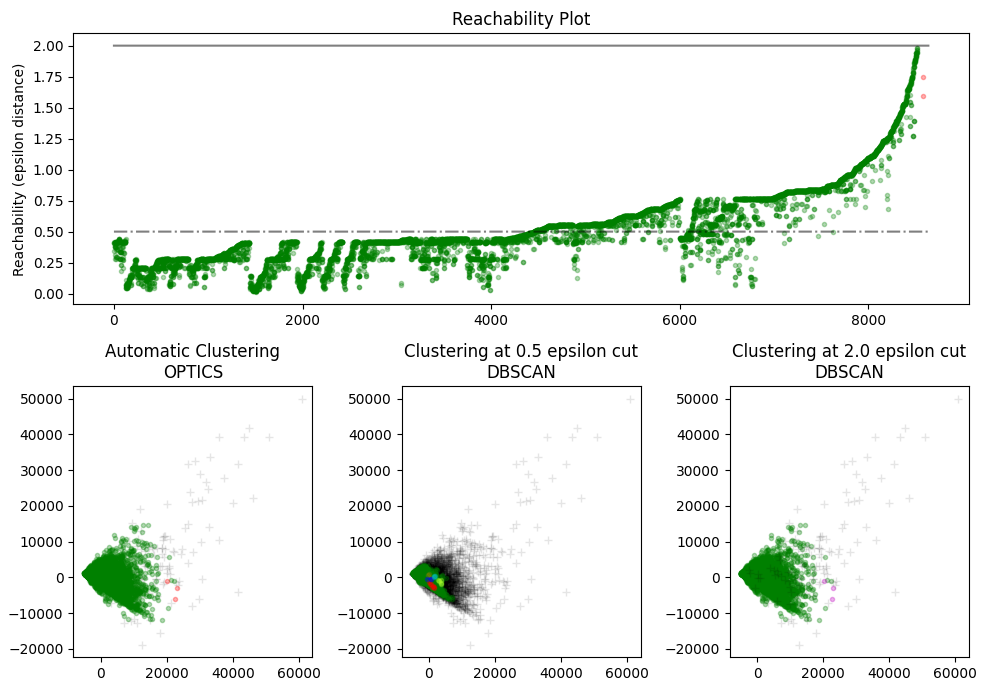


Рисунок 9 – График достижимости при использовании метрики “chebyshev”

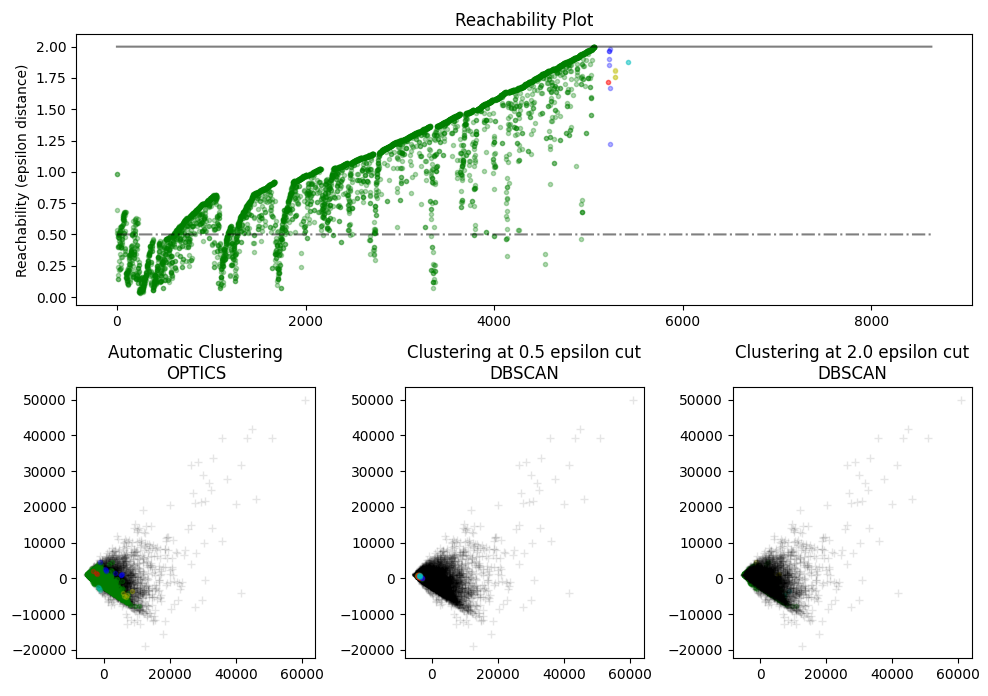


Рисунок 10 – График достижимости при использовании метрики “l1”

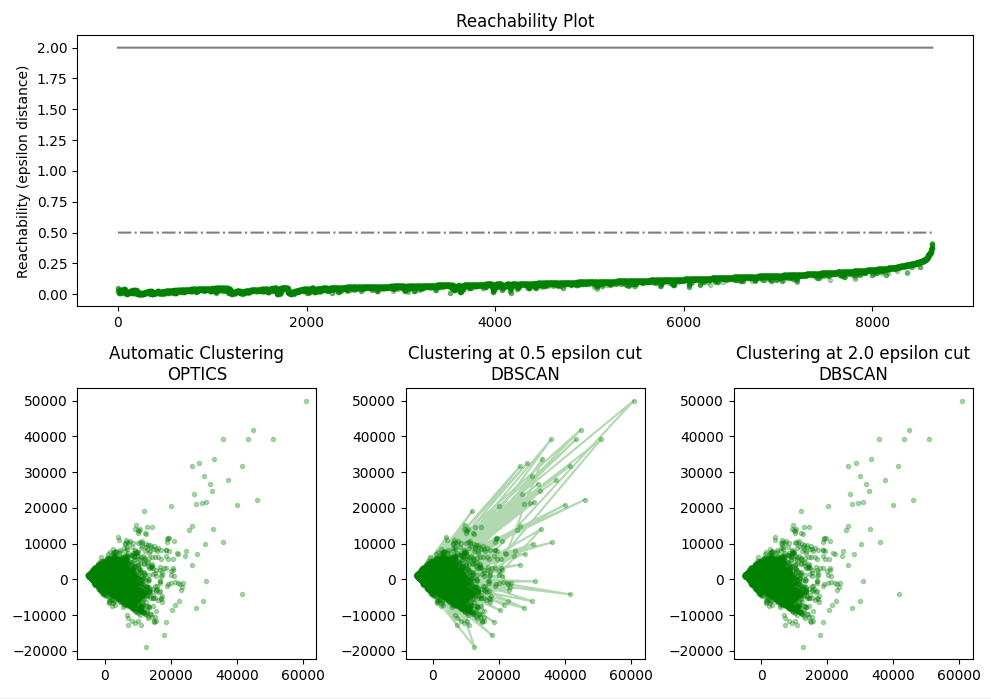


Рисунок 11 – График достижимости при использовании метрики “braycurtis”

**Вывод**

Был получен опыт работы с методами кластеризации.

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

import pandas as pd  
import numpy as np  
from sklearn.cluster import KMeans, DBSCAN, OPTICS, cluster\_optics\_dbscan  
from sklearn import preprocessing  
import matplotlib.pyplot as plt  
import matplotlib.gridspec as gridspec  
from sklearn.decomposition import PCA  
  
  
data = pd.read\_csv('CC GENERAL.csv').iloc[:,1:].dropna()  
print(data)  
  
k\_means = KMeans(init='k-means++', n\_clusters=3, n\_init=15)  
no\_labeled\_data = data[1:]  
k\_means.fit(no\_labeled\_data)  
data = np.array(data, dtype='float')  
min\_max\_scaler = preprocessing.StandardScaler()  
scaled\_data = min\_max\_scaler.fit\_transform(data)  
  
clustering = DBSCAN().fit(scaled\_data)  
print(set(clustering.labels\_))  
print(len(set(clustering.labels\_)) - 1)  
print(list(clustering.labels\_).count(-1) / len(list(clustering.labels\_)))  
  
eps\_ = np.arange(0.01, 5.0, 0.1)  
info = []  
for eps in eps\_:  
 clustering = DBSCAN(eps=eps).fit(scaled\_data)  
 labels\_set = set(clustering.labels\_)  
 info.append([len(labels\_set) - 1, list(clustering.labels\_).count(-1) / len(list(clustering.labels\_))])  
  
info = np.array(info)  
fig, ax = plt.subplots(1, 2, figsize=(13,6))  
ax[0].plot(eps\_, info[:,0])  
ax[1].plot(eps\_, info[:,1])  
plt.show()  
samples = np.arange(5, 100, 1)  
info = []  
for sample in samples:  
 clustering = DBSCAN(min\_samples=sample).fit(scaled\_data)  
 labels\_set = set(clustering.labels\_)  
 info.append([len(labels\_set) - 1, list(clustering.labels\_).count(-1) / len(list(clustering.labels\_))])  
info = np.array(info)  
fig, ax = plt.subplots(1, 2, figsize=(13,6))  
ax[0].plot(samples, info[:,0])  
ax[1].plot(samples, info[:,1])  
plt.show()  
samples = np.arange(1, 4, 1)  
eps\_ = np.arange(1.5, 2.5, 0.1)  
info = {}  
for sample in samples:  
 for eps in eps\_:  
 clustering = DBSCAN(eps=eps ,min\_samples=sample, n\_jobs=-1).fit(scaled\_data)  
 labels\_set = set(clustering.labels\_)  
 info[(sample, eps)]= [len(labels\_set) - 1, list(clustering.labels\_).count(-1) / len(list(clustering.labels\_))]  
for key, value in info.items():  
 if value[0]>=5 and value[0]<=7 and value[1]<=0.12:  
 print(key, value)  
  
clustering = DBSCAN(eps=2, min\_samples=3, n\_jobs=-1).fit(scaled\_data)  
pca = PCA(n\_components=2)  
reduced\_data = pca.fit\_transform(data)  
pca.explained\_variance\_ratio\_  
core\_samples\_mask = np.zeros\_like(clustering.labels\_, dtype=bool)  
core\_samples\_mask[clustering.core\_sample\_indices\_] = True  
labels = clustering.labels\_  
  
n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)  
n\_noise\_ = list(labels).count(-1)  
# unique\_labels = set(labels)  
unique\_labels = set(labels)  
unique\_labels.remove(-1)  
unique\_labels = [-1, \*list(unique\_labels)]  
colors = [plt.cm.Spectral(each)  
 for each in np.linspace(0, 1, len(unique\_labels))]  
plt.figure(figsize=(12, 8))  
for k, col in zip(unique\_labels, colors):  
 if k == -1:  
 # Black used for noise.  
 col = [0, 0, 0, 1]  
  
 class\_member\_mask = (labels == k)  
  
 xy = reduced\_data[class\_member\_mask & core\_samples\_mask]  
 plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),  
 markeredgecolor='k', markersize=9)  
  
 xy = reduced\_data[class\_member\_mask & ~core\_samples\_mask]  
 plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),  
 markeredgecolor='k', markersize=5)  
plt.show()  
clustering = OPTICS(max\_eps=2, min\_samples=3, cluster\_method='dbscan').fit(scaled\_data)  
print(set(clustering.labels\_))  
print(len(set(clustering.labels\_)) -1)  
print(100 \* list(clustering.labels\_).count(-1) / len(list(clustering.labels\_)))  
  
pca\_data = PCA(n\_components=2).fit\_transform(data)  
def plot\_optics(clust):  
 labels\_050 = cluster\_optics\_dbscan(reachability=clust.reachability\_,  
 core\_distances=clust.core\_distances\_,  
 ordering=clust.ordering\_, eps=0.5)  
 labels\_200 = cluster\_optics\_dbscan(reachability=clust.reachability\_,  
 core\_distances=clust.core\_distances\_,  
 ordering=clust.ordering\_, eps=2)  
  
 space = np.arange(len(scaled\_data))  
 reachability = clust.reachability\_[clust.ordering\_]  
 labels = clust.labels\_[clust.ordering\_]  
  
 plt.figure(figsize=(10, 7))  
 G = gridspec.GridSpec(2, 3)  
 ax1 = plt.subplot(G[0, :])  
 ax2 = plt.subplot(G[1, 0])  
 ax3 = plt.subplot(G[1, 1])  
 ax4 = plt.subplot(G[1, 2])  
  
 # Reachability plot  
 colors = ['g.', 'r.', 'b.', 'y.', 'c.']  
 for klass, color in zip(range(0, 5), colors):  
 Xk = space[labels == klass]  
 Rk = reachability[labels == klass]  
 ax1.plot(Xk, Rk, color, alpha=0.3)  
 ax1.plot(space[labels == -1], reachability[labels == -1], 'k.', alpha=0.3)  
 ax1.plot(space, np.full\_like(space, 2., dtype=float), 'k-', alpha=0.5)  
 ax1.plot(space, np.full\_like(space, 0.5, dtype=float), 'k-.', alpha=0.5)  
 ax1.set\_ylabel('Reachability (epsilon distance)')  
 ax1.set\_title('Reachability Plot')  
  
 # OPTICS  
 colors = ['g.', 'r.', 'b.', 'y.', 'c.']  
 ax2.plot(pca\_data[clust.labels\_ == -1, 0], pca\_data[clust.labels\_ == -1, 1], 'k+', alpha=0.1)  
 for klass, color in zip(range(0, 5), colors):  
 Xk = pca\_data[clust.labels\_ == klass]  
 ax2.plot(Xk[:, 0], Xk[:, 1], color, alpha=0.3)  
 ax2.set\_title('Automatic Clustering\nOPTICS')  
  
 # DBSCAN at 0.5  
 colors = ['g', 'greenyellow', 'olive', 'r', 'b', 'c']  
 ax3.plot(pca\_data[labels\_050 == -1, 0], pca\_data[labels\_050 == -1, 1], 'k+', alpha=0.1)  
 for klass, color in zip(range(0, 6), colors):  
 Xk = pca\_data[labels\_050 == klass]  
 ax3.plot(Xk[:, 0], Xk[:, 1], color, alpha=0.3, marker='.')  
 ax3.set\_title('Clustering at 0.5 epsilon cut\nDBSCAN')  
  
 # DBSCAN at 2.  
 colors = ['g.', 'm.', 'y.', 'c.']  
 for klass, color in zip(range(0, 4), colors):  
 Xk = pca\_data[labels\_200 == klass]  
 ax4.plot(Xk[:, 0], Xk[:, 1], color, alpha=0.3)  
 ax4.plot(pca\_data[labels\_200 == -1, 0], pca\_data[labels\_200 == -1, 1], 'k+', alpha=0.1)  
 ax4.set\_title('Clustering at 2.0 epsilon cut\nDBSCAN')  
  
 plt.tight\_layout()  
 plt.show()  
  
plot\_optics(clustering)  
  
metrics = ['cityblock', 'cosine', 'chebyshev', 'l1', 'braycurtis']  
  
for metric in metrics:  
 clustering = OPTICS(max\_eps=2, min\_samples=3, cluster\_method="dbscan", metric=metric).fit(scaled\_data)  
 plot\_optics(clustering)