**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. 6307 |  | Давыдова Н. П. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2021

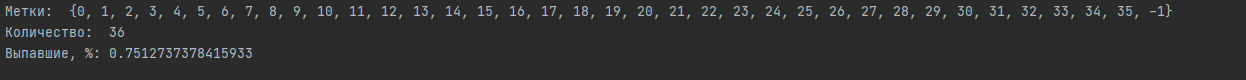
Цель:

Ознакомиться с методами кластеризации модуля Sklearn

DBSCAN

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

|  |
| --- |
| clustering = DBSCAN().fit(scaled\_data)  print('Метки: ', set(clustering.labels\_))  print('Количество: ',len(set(clustering.labels\_)) - 1)  print('Выпавшие, %:', list(clustering.labels\_).count(-1) / len(list(clustering.labels\_))) |



Опишите все параметры, которые принимает DBSCAN:

**EPS *float, по умолчанию=0,5***

Максимальное расстояние между двумя выборками для одной из них должно рассматриваться как в окрестности другой. Это не является максимальным ограничением на расстояния точек внутри кластера. Это самый важный параметр DBSCAN, который необходимо выбрать соответствующим образом для вашего набора данных и функции расстояния.

**min\_samples *int, default=5***

Число выборок (или общий вес) в окрестности точки , которая должна рассматриваться как основная точка. Это включает в себя и саму точку.

**Метрическая *строка, или вызываемая, по умолчанию=’евклидова’***

Метрика, используемая при вычислении расстояния между экземплярами в массиве объектов. Если метрика является строкой или вызываемой, она должна быть одной из опций, разрешенных **[sklearn.metrics.pairwise\_distances](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html" \l "sklearn.metrics.pairwise_distances" \o "sklearn.metrics.pairwise_distances)**для ее параметра metric. Если метрика "предварительно вычислена", то X предполагается матрицей расстояний и должна быть квадратной. X может быть [глоссарием](https://scikit-learn.org/stable/glossary.html#term-sparse-graph), и в этом случае только “ненулевые” элементы могут считаться соседями для DBSCAN.

*Новое в версии 0.17:*метрика *предварительно* вычисляется для приема предварительно вычисленной разреженной матрицы.

**metric\_params *dict, по умолчанию=нет***

Дополнительные аргументы ключевого слова для метрической функции.

*New in version 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, по умолчанию=30***

Размер листа передается в BallTree или cKDTree. Это может повлиять на скорость построения и запроса, а также на объем памяти, необходимый для хранения дерева. Оптимальное значение зависит от характера задачи.

**P *float, по умолчанию=None***

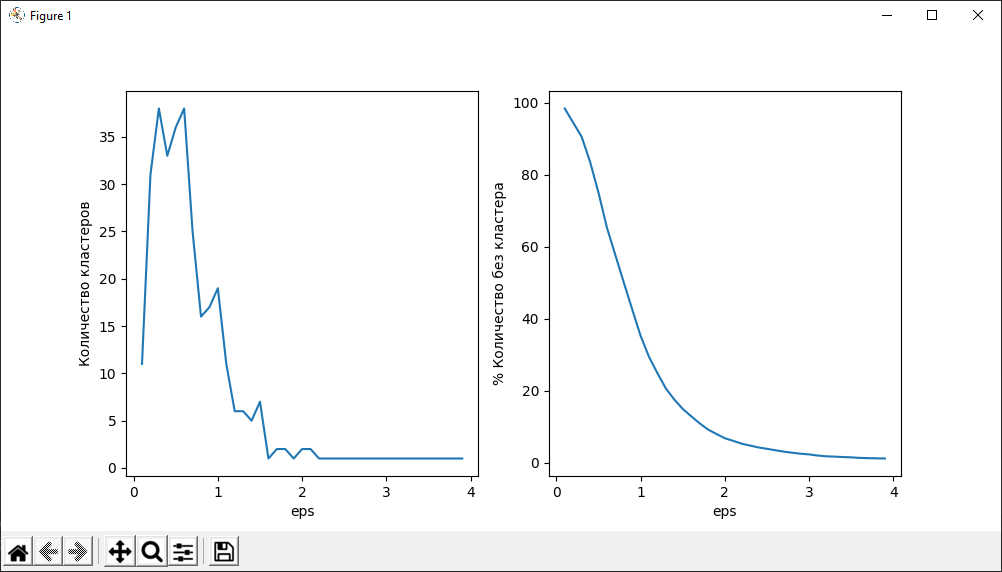
Мощность метрики Минковского, которая будет использоваться для вычисления расстояния между точками. Если нет, то p=2(эквивалентно Евклидову расстоянию).

**n\_jobs *int, default=None***

Количество выполняемых параллельных заданий. None означает 1, если только в **[joblib.parallel\_backend](https://joblib.readthedocs.io/en/latest/parallel.html" \l "joblib.parallel_backend" \o "(в joblib v1. 1. 0. dev0))**контексте. -1 означает использование всех процессоров. Более подробную информацию смотрите в глоссарии

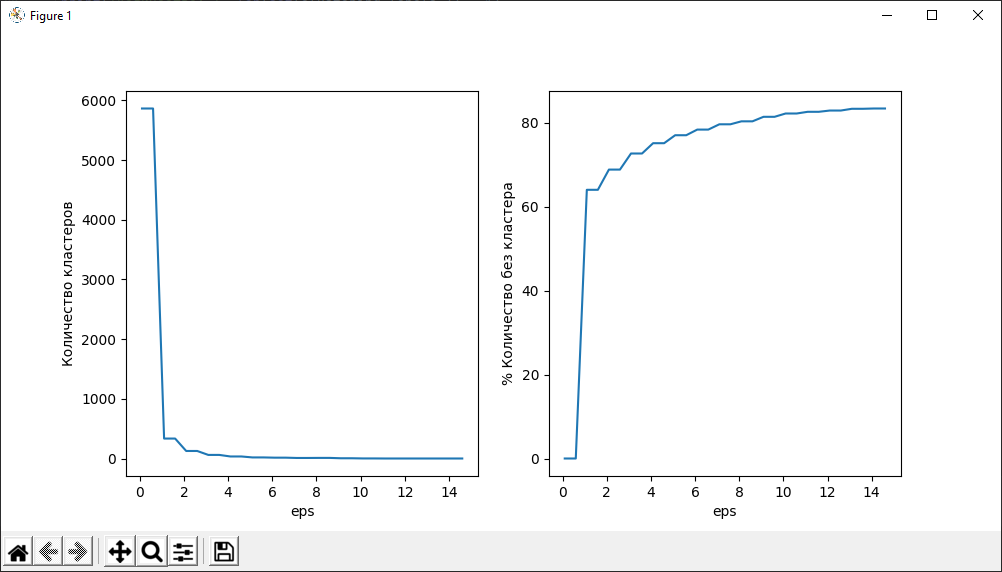
4. Постройте график количества кластеров и процента не кластеризованных наблюдений в зависимости от максимальной рассматриваемой дистанции(eps,оно жн максимальное) между наблюдениями. Минимальное значение количества точек образующих, кластер оставить по умолчанию

|  |
| --- |
| eps\_range = np.arange(0.1, 4, 0.1)  clust\_number = []  clust\_percent\_dropped = []  for eps in eps\_range:  clustering = DBSCAN(eps=eps).fit(scaled\_data)  clust\_number.append(len(set(clustering.labels\_)) - 1)  clust\_percent\_dropped.append(100 \* list(clustering.labels\_).count(-1) / len(list(clustering.labels\_)))  fig, ax = plt.subplots(1, 2, figsize=(10, 5))  ax[0].plot(eps\_range, clust\_number)  ax[0].set\_ylabel('Количество кластеров')  ax[0].set\_xlabel('eps')  ax[1].plot(eps\_range, clust\_percent\_dropped)  ax[1].set\_ylabel('% Количество без кластера')  ax[1].set\_xlabel('eps')  plt.show() |



5. Постройте график количества кластеров и процента не кластеризованных наблюдений в зависимости от минимального значения количества точек, образующих кластер. Максимальную рассматриваемую дистанцию между наблюдениями оставьте по умолчанию

|  |
| --- |
| min\_sample\_range = np.arange(0.1, 15, 0.5)  clust\_number = []  clust\_percent\_dropped = []  for min\_sample in min\_sample\_range:  clustering = DBSCAN(min\_samples=min\_sample).fit(scaled\_data)  clust\_number.append(len(set(clustering.labels\_)) - 1)  clust\_percent\_dropped.append(100 \* list(clustering.labels\_).count(-1) / len(list(clustering.labels\_)))  fig, ax = plt.subplots(1, 2, figsize=(10, 5))  ax[0].plot(min\_sample\_range, clust\_number)  ax[0].set\_ylabel('Количество кластеров')  ax[0].set\_xlabel('eps')  ax[1].plot(min\_sample\_range, clust\_percent\_dropped)  ax[1].set\_ylabel('% Количество без кластера')  ax[1].set\_xlabel('eps')  plt.show() |



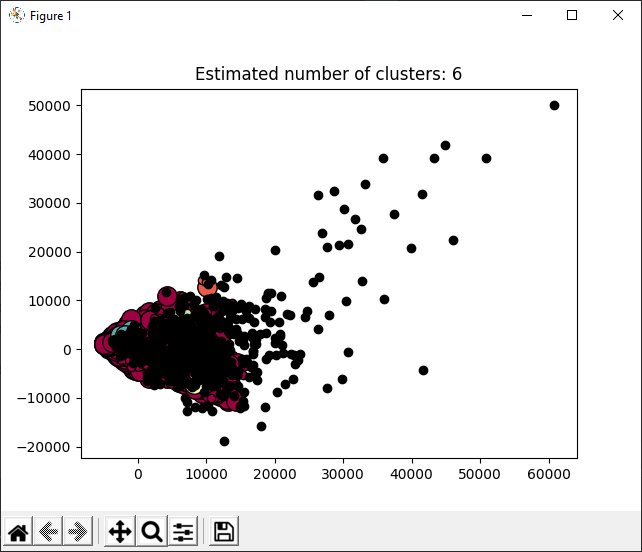
6. Определите значения параметров, при котором количество кластеров получается от 5 до 7, и процент не кластеризованных наблюдений не превышает 12%.

|  |
| --- |
| min\_sample\_range = np.arange(1, 4, 0.5)  eps\_range = np.arange(1, 4, 0.2)  clust\_number = []  clust\_percent\_dropped = []  for eps in min\_sample\_range:  for min\_sample in min\_sample\_range:  clustering = DBSCAN(eps = eps, min\_samples=min\_sample).fit(scaled\_data)  clust\_number.append(len(set(clustering.labels\_)) - 1)  clust\_percent\_dropped.append(100 \* list(clustering.labels\_).count(-1) / len(list(clustering.labels\_)))  if 5 <= clust\_number[len(clust\_number)-1] <= 7 and clust\_percent\_dropped[len(clust\_percent\_dropped)-1] <= 12:  print(eps, min\_sample, clust\_number[len(clust\_number)-1], clust\_percent\_dropped[len(clust\_percent\_dropped)-1]) |

|  |
| --- |
| eps= 2.0 min\_sample= 2.5 clust\_number= 6 clust\_percent\_dropped= 6.287633163501621  eps= 2.0 min\_sample= 3.0 clust\_number= 6 clust\_percent\_dropped= 6.287633163501621  eps= 2.599996 min\_sample= 2.5 clust\_number= 5 clust\_percent\_dropped= 3.0917091245947197  eps= 2.599996 min\_sample= 3.0 clust\_number= 5 clust\_percent\_dropped= 3.0917091245947197  eps= 3.0 min\_sample= 2.5 clust\_number= 5 clust\_percent\_dropped= 1.9569245020842982  eps= 3.0 min\_sample= 3.0 clust\_number= 5 clust\_percent\_dropped= 1.9569245020842982 |

7. Понизьте размерность данных до 2 при используя метод главных компонент. Визуализируйте результаты кластеризации полученные в пункте 6 (метки должны быть получены на данных до уменьшения размерности.

|  |
| --- |
| pca\_data = PCA(n\_components=2).fit\_transform(data)  db = DBSCAN(eps=2, min\_samples=3).fit(scaled\_data)  core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)  core\_samples\_mask[db.core\_sample\_indices\_] = True  labels = db.labels\_  # Number of clusters in labels, ignoring noise if present.  n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)  n\_noise\_ = list(labels).count(-1)  # Black removed and is used for noise instead.  unique\_labels = set(labels)  colors = [plt.cm.Spectral(each)  for each in np.linspace(0, 1, len(unique\_labels))]  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 = pca\_data[class\_member\_mask & core\_samples\_mask]  plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),  markeredgecolor='k', markersize=14)  xy = pca\_data[class\_member\_mask & ~core\_samples\_mask]  plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),  markeredgecolor='k', markersize=6)  plt.show() |



1. Опишите параметры метода OPTICS, а также какими атрибутами он обладает

**min\_samples *int > 1 или float между 0 и 1, по умолчанию=5***

Число выборок в окрестности точки, рассматриваемой как основная точка. Кроме того, вверх и вниз по крутым склонам не может быть больше min\_samples последовательные не крутые точки. Выражается как абсолютное число или доля от числа выборок (округляется не менее чем на 2).

**max\_eps *float, default=np.inf***

Максимальное расстояние между двумя образцами для того, чтобы один рассматривался как находящийся по соседству с другим. Значение по умолчанию np.infбудет идентифицировать кластеры во всех масштабах; сокращение max\_epsприведет к сокращению времени выполнения.

**Метрика *str или callable, по умолчанию=’minkowski’***

Метрика, используемая для вычисления расстояния. Можно использовать любую метрику из scikit-learn или scipy.spatial.distance.

Если метрика является вызываемой функцией, то она вызывается для каждой пары экземпляров (строк) и записывается результирующее значение. Вызываемый объект должен принимать два массива в качестве входных данных и возвращать одно значение, указывающее расстояние между ними. Это работает для метрик Scipy, но менее эффективно, чем передача имени метрики в виде строки. Если метрика “предварительно вычислена”, то X считается матрицей расстояний и должна быть квадратной.

Допустимыми значениями для метрики являются:

* из 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 **[pairwise\_distances](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html" \l "sklearn.metrics.pairwise_distances" \o "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***

Определяет минимальную крутизну на графике достижимости , образующем границу кластера. Например, восходящая точка на графике достижимости определяется отношением от одной точки до ее преемника, составляющим не более 1-xi. Используется только когда cluster\_method='xi'.

**preferencor\_correction *bool, default=True***

Правильные кластеры в соответствии с предшественниками, вычисленными оптикой [[2]](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html#r2c55e37003fe-2). Этот параметр оказывает минимальное влияние на большинство наборов данных. Используется только когда cluster\_method='xi'.

**min\_cluster\_size *int > 1 или float между 0 и 1, по умолчанию=нет***

Минимальное количество выборок в оптическом кластере, выраженное в абсолютном числе или доле от числа выборок (округляется не менее чем на 2). Если Nonemin\_samplesвместо этого используется значение of. Используется только тогда, когда cluster\_method='xi'.

**алгоритм*{'auto’,' ball\_tree’, ‘kd\_tree’, ‘brute’}, по умолчанию=’auto’***

Алгоритм, используемый для вычисления ближайших соседей:

* ‘ball\_tree’ будет использовать BallTree
* ‘kd\_tree’ будет использовать KDTree
* ‘грубый’ будет использовать поиск грубой силы.
* ‘auto’ попытается выбрать наиболее подходящий алгоритм на основе значений, переданных **[fit](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html" \l "sklearn.cluster.OPTICS.fit" \o "склеарн.кластер.Оптика.подгонка)**методу. (по умолчанию)

Примечание: установка на разреженном входе переопределит настройку этого параметра, используя грубую силу.

**leaf\_size *int, по умолчанию=30***

Размер листа передается в BallTreeОр KDTree. Это может повлиять на скорость построения и запроса, а также на объем памяти , необходимый для хранения дерева. Оптимальное значение зависит от характера задачи.

**n\_jobs *int, default=None***

Число параллельных заданий для выполнения поиска соседей. None означает 1, если только в **[joblib.parallel\_backend](https://joblib.readthedocs.io/en/latest/parallel.html" \l "joblib.parallel_backend" \o "(в joblib v1. 1. 0. dev0))** контексте. -1 означает использование всех процессоров. Более подробную информацию смотрите в глоссарии.

2. Найдите такие параметры метода OPTICS (\*max\_eps \*и min\_samples) при которых, чтобы получить результаты близкие к результатам DBSCAN из пункта 6

|  |
| --- |
| for eps in eps\_range:  for min\_sample in min\_sample\_range:  clustering = DBSCAN(eps = eps, min\_samples=min\_sample).fit(scaled\_data)  clust\_number.append(len(set(clustering.labels\_)) - 1)  clust\_percent\_dropped.append(100 \* list(clustering.labels\_).count(-1) / len(list(clustering.labels\_)))  if 5 <= clust\_number[len(clust\_number)-1] <= 7 and clust\_percent\_dropped[len(clust\_percent\_dropped)-1] <= 12:  print("eps= ", eps," min\_sample= ", min\_sample," clust\_number= ", clust\_number[len(clust\_number)-1]," clust\_percent\_dropped= ", clust\_percent\_dropped[len(clust\_percent\_dropped)-1]) |

|  |
| --- |
| eps= 1.9999999999999998 min\_sample= 2.5 clust\_number= 6 clust\_percent\_dropped= 6.287633163501621  eps= 1.9999999999999998 min\_sample= 3.0 clust\_number= 6 clust\_percent\_dropped= 6.287633163501621  eps= 2.5999999999999996 min\_sample= 2.5 clust\_number= 5 clust\_percent\_dropped= 3.0917091245947197  eps= 2.5999999999999996 min\_sample= 3.0 clust\_number= 5 clust\_percent\_dropped= 3.0917091245947197  eps= 2.9999999999999996 min\_sample= 2.5 clust\_number= 5 clust\_percent\_dropped= 1.9569245020842982  eps= 2.9999999999999996 min\_sample= 3.0 clust\_number= 5 clust\_percent\_dropped= 1.9569245020842982 |

3. Визуализируйте полученный результат, а также постройте график достижимости (reachable plot)

|  |
| --- |
| pca\_data = PCA(n\_components=2).fit\_transform(data)  np.random.seed(0)  n\_points\_per\_cluster = 250  clust = OPTICS(min\_samples=50, xi=.05, min\_cluster\_size=.05)  # Run the fit  clust.fit(scaled\_data)  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.']  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.plot(pca\_data[clust.labels\_ == -1, 0], pca\_data[clust.labels\_ == -1, 1], 'k+', alpha=0.1)  ax2.set\_title('Automatic Clustering\nOPTICS')  # DBSCAN at 0.5  colors = ['g', 'greenyellow', 'olive', 'r', 'b', 'c']  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.plot(pca\_data[labels\_050 == -1, 0], pca\_data[labels\_050 == -1, 1], 'k+', alpha=0.1)  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() |

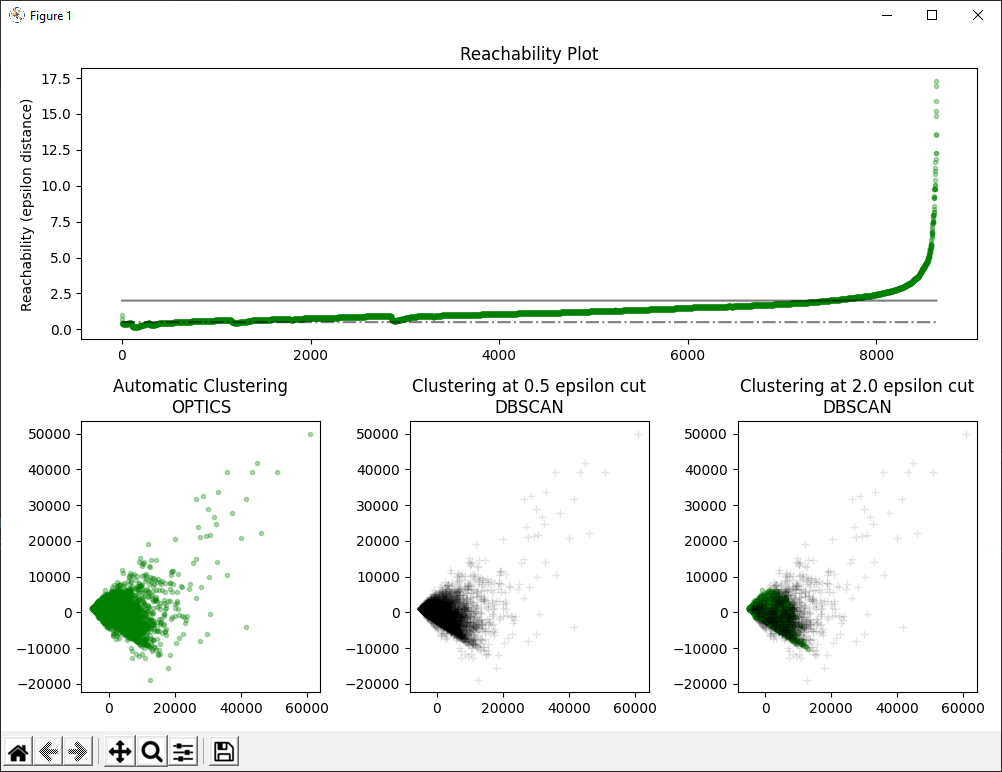
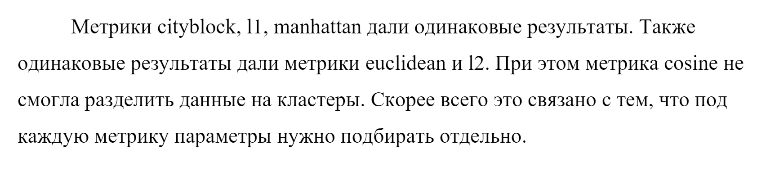


График достижимости

4. Исследуйте работу метода OPTICS с использованием различных метрик (выберите не менее 5 метрик)



|  |
| --- |
| metrics = ["cosine", "euclidean", "cityblock", "manhattan", "l1"]  for metric in metrics:  clustering = OPTICS(max\_eps=2, min\_samples=3, cluster\_method='dbscan', metric=metric).fit(scaled\_data)  clust\_number = len(set(clustering.labels\_)) - 1  clust\_percent\_dropped = list(clustering.labels\_).count(-1) / len(list(clustering.labels\_)) \* 100  print("Метрика: ", metric, " Количество кластеров: ", clust\_number," Безкластерные: ", clust\_percent\_dropped) |

|  |
| --- |
| Метрика: cosine Кластеров: 0 Безкластерных: 0.0  Метрика: euclidean Кластеров: 6 Безкластерных: 6.310792033348772  Метрика: cityblock Кластеров: 55 Безкластерных: 39.49745252431681  Метрика: manhattan Кластеров: 55 Безкластерных: 39.49745252431681  Метрика: l1 Кластеров: 55 Безкластерных: 39.49745252431681 |