Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ_____

КАФЕДРА _____СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Кластеризация данных LiDAR

Студент ___ИУ5-61Б__ _____ ____**Смыслов Д. О.**__
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ _____**Канев А. И.**___
(Подпись, дата) (И.О.Фамилия)

Москва, 2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ___ИУ5___
(Индекс)

_____ _____
(И.О.Фамилия)

« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме _____Кластеризация данных LiDAR_____

_____

_____

Студент группы ____ИУ5-61Б____

_____Смыслов Дмитрий Олегович_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

_____Исследовательская_____

Источник тематики (кафедра, предприятие, НИР) _____НИР_____

График выполнения НИР: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Техническое задание _Исследовать использование методов машинного обучения для решения задачи сегментации деревьев из облака точек

_____

_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на _30_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____

_____

Дата выдачи задания «07» февраля 2023 г.

Руководитель НИР _____ **Канев А.И.**
(Подпись, дата) (И.О.Фамилия)

Студент _____ **Смыслов Д.О.**
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

LiDAR – технология, позволяющая измерять расстояния в пространстве путем излучения света и замера времени возвращения этого отраженного света на приемник. Сканирующие лидары применяются в системах машинного зрения для формирования двумерной и трехмерной картин окружающего мира, «атмосферные» лидары используются для анализа свойств прозрачных сред, поглощающих и рассеивающих свет, а доплеровские лидары применяются для отслеживания перемещения воздушных потоков в различных слоях атмосферы.

Результатом съемки лидара обычно является массив трехмерных точек – облако точек, каждая из которых принадлежит какому-либо объекту и их элементам. Далее эти точки необходимо проанализировать – например сгруппировать их по принадлежности какому-нибудь объекту или классу объектов. Такая задача называется сегментацией. Имея в дальнейшем обработанные данные, их можно использовать в других задач – для построения моделей и анализа выделенных объектов. Имеется большое количество сфер, где данную технологию можно применить, и сегментация и выделение деревьев с целью их дальнейшего анализа – одна из таких задач.

Целью данной работы является исследование применения методов машинного обучения для сегментации деревьев из облака точек. В данной работе задачами являются применение методов машинного обучения для сегментации деревьев из двух облаков точек – одной малой размерности с четырьмя деревьями, второе – большей размерности, охватывающее часть леса, а также анализ и оценка результатов сегментации в том числе с помощью соответствующих метрик.

1 Постановка задачи

В качестве исходных данных имеется множество трехмерных точек, которое получено в результате съемки лидаром нескольких деревьев или небольшой части леса, включающей пару десятков деревьев. Необходимо выполнить сегментацию деревьев из облака точек, то есть необходимо отнести точки, относящиеся к одному и тому же дереву, к одному классу.

Решение данной задачи может быть полезным для извлечения новых полезных данных, анализ которых позволит усовершенствовать способы ведения лесного хозяйства и повысить его эффективность. Проведение измерений вручную, например подсчет деревьев на участке и исследование их характеристик, может быть трудоемким, может занимать много времени или может быть затруднено погодными условиями. Использование технологии LiDAR является эффективным решением этой проблемы, так как технология LiDAR активно развивается, удобна для получения больших объемов данных и легко автоматизируема [3] [5].

За многими объектами, в том числе и за деревьями, полезно, а иногда и необходимо, наблюдать в любое время года и при любой погоде. Но в таких случаях велика вероятность сильного зашумления данных, полученных с помощью съемки лидаром при таких условиях, как снег, дождь и туман. Существуют алгоритмы, способные относительно четко выделять эти шумы и соответственно их фильтровать, тем самым очищая данные [2].

Задача сегментации деревьев из облака точек может быть решена различными методами. Например, в статье [1] для сегментации используется метод, основанный на графах. Точки из облака группируются в воксели заданного размера, а узлы графа определяются средним положением всех точек в вокселе. Метод предполагает обход облака точек по графу: отслеживаются пути от каждого узла ствола до каждого узла в графе. Каждый узел дерева, который был достигнут по некоторому пути, соотносится узлу ствола, который породил этот путь. После того, как все узлы соотнесены соответствующим

узлам ствола, сегментация распространяется на все точки, принадлежащие соответствующим вокселям.

Еще один подход к сегментации деревьев из облака точек – метод Layer Stacking [4]. Облако точек разбивается на слои, например по высоте, в каждом из которых применяется метод кластеризации K-means. После этого в каждом слое строятся полигоны Тиссена. Затем, при наложении друг на друга полигонов различных слоев, образуются области плотного перекрытия, которые свидетельствуют о наличии отдельного дерева в области перекрытия. Качество сегментации можно оценить с использованием таких метрик, как precision, recall и F1 score, в случае если данные размечены.

2 Описание данных, используемых методов и метрик

Исходные данные представляют собой массив трехмерных точек (Рисунок 2.1), полученных в результате съемки лидаром расположенных довольно близко друг к другу (наблюдается пересечение крон) четырех деревьев (вариант 1). При этом точки, принадлежащие стволу, распределены плотнее, чем точки, принадлежащие листве деревьев, что может сказаться на результатах сегментации, в особенности на листве.

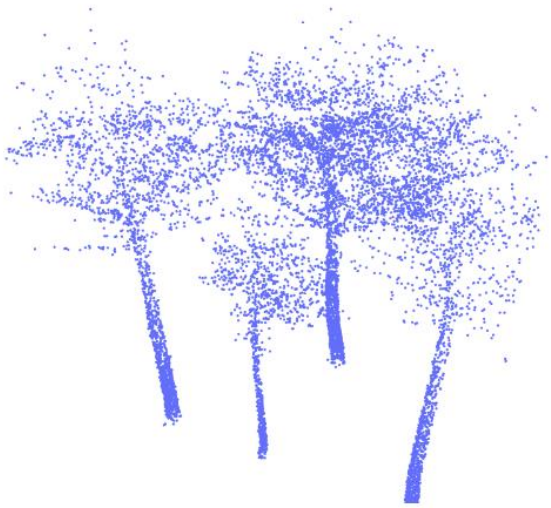

Рисунок 2.1 – облако точек

В качестве метода сегментации (кластеризации) будет применятся DBSCAN – основанный на плотности алгоритм метрической кластеризации с

присутствием шума. У данного алгоритма есть 2 гиперпараметра, которые необходимо задать перед его запуском – радиус рассматриваемых ε-окрестностей точек eps и минимальное количество соседей min_samples. Точка будет считаться корневой (то есть принадлежать центру кластера), если в ее окрестности радиуса eps будет находиться не менее min_samples соседей. Алгоритм выбирает из исходного массива какую-либо корневую точку и помещает ее соседей в список обхода. Если сосед также является корневым, то и его соседи включаются в список обхода, иначе – нет, и точка считается границей кластера. Остальные точки считаются шумом и являются плохо достижимыми для всех кластеров. Таким образом происходит образование кластеров точек, которое будет продолжаться до тех пор, пока обход не пройдет по всем точкам.

Использование DBSCAN для данных большой размерности неэффективно, так как его расход оперативной памяти в среднем асимптотически не линеен, а в худшем случае и вовсе квадратичен. Наши данные таковыми не являются, поэтому DBSCAN подойдет для их исследования. Так как данные никак не размечены, то качество кластеризации будет оцениваться «на глаз» а также с помощью популярной для такого случая метрики Silhouette Score. Метрика Silhouette Score является внутренней метрикой оценки качества кластеризации. Эта метрика подойдет для работы с нашими данными, так как она не требует априорной информации о наборе данных и производит оценку только лишь на основе результатов кластеризации.

3 Сегментация деревьев из облака точек

С использованием метода DBSCAN и функций для представления набора точек в формате pcd, сегментации, графического отображения облака точек и присваивания точкам цветов была произведена сегментация облака точек и его отображение для параметров eps = 0,5 и min_samples = 206 (Рисунок 3.1). Результат оказался неудовлетворительным, так как крона одного дерева

зацепила крону другого дерева, и таким образом одно из деревьев было неверно сегментировано. Также возникает подозрение, что ствол этого дерева где-то по середине как бы разрывается, и не может быть сегментирован верно.

Рисунок 3.1 – результат сегментации при eps = 0,5, min_samples = 206

Таким образом возникает необходимость подбора правильных гиперпараметров. Было решено для каждого eps от 0,2 до 0,65 с шагом 0,05 и каждого min_samples от 2 до 298 с шагом 2 определить количество кластеров и значение метрики силуэта. Был создан датафрейм, который ставит каждой паре eps и min_samples в соответствие количество кластеров (в том числе кластер с выбросами) и значение метрики силуэта (Рисунок 3.2, Рисунок 3.3).

```
%%time

# Подбор параметров
tuned_eps = np.arange(0.2, 0.7, 0.05)
tuned_min_pts = np.arange(2, 300, 2)

# Множество всех возможных пар параметров
tuned_par = [(eps, min_pts) for eps in tuned_eps for min_pts in tuned_min_pts]
size = len(tuned_par)

# Счетчик перебранных пар параметров
cnt = 0
perc = 0

# Количество кластеров и Silhouette Score  как результаты очередной работы алгоритма
num_clusters, silhouette_scores = [], []

# Непосредственный перебор
for eps, min_pts in tuned_par:
    clustering = DBSCAN(eps=eps, min_samples=min_pts,
                        algorithm='ball_tree', n_jobs=4).fit(X)
    num_clusters.append(clustering.labels_.max() + 1)
    sil_score = None
    if 0 < clustering.labels_.max() < 9999:
        sil_score = silhouette_score(X, clustering.labels_)
    silhouette_scores.append(sil_score)
    # оповещаем каждые 5%
    cnt += 1
    factor = size // 20
    if cnt % factor == 0:
        print(f"Перебор выполнен на {cnt * 100 / size} %, всего перебрано {cnt} пар гиперпараметров")
```

Рисунок 3.2 – код перебора параметров и подсчета количества кластеров

```
esp_list = [e for e, p in tuned_par]
min_pts_list = [p for e, p in tuned_par]

zipped_full_test_data = list(zip(esp_list, min_pts_list, num_clusters, silhouette_scores))
test_df = pd.DataFrame(zipped_full_test_data, columns=['eps', 'min_pts', 'Number of clusters', 'Silhouette Score'])

test_df
```

	eps	min_pts	Number of clusters	Silhouette Score
0	0.20	2	16	-0.377282
1	0.20	4	7	-0.168658
2	0.20	6	9	-0.222121
3	0.20	8	10	-0.229045
4	0.20	10	10	-0.419131
...	...	...	...	...
1485	0.65	290	1	NaN
1486	0.65	292	1	NaN
1487	0.65	294	1	NaN
1488	0.65	296	1	NaN
1489	0.65	298	1	NaN

1490 rows × 4 columns

Рисунок 3.3 – создание датафрейма и его содержание

Из всех записей были отобраны только те, в которых количество кластеров оказалось равно 4 (Рисунок 3.4).

```
good_test_df = test_df[test_df['Number of clusters'] == 4]
good_test_df
```

	eps	min_pts	Number of clusters	Silhouette Score
40	0.20	82	4	0.226569
41	0.20	84	4	0.226298
42	0.20	86	4	0.226295
43	0.20	88	4	0.226507
51	0.20	104	4	0.250752
...	...	...	...	...
1174	0.55	264	4	0.153322
1175	0.55	266	4	0.152094
1176	0.55	268	4	0.150884
1177	0.55	270	4	0.151041
1178	0.55	272	4	0.150418

144 rows × 4 columns

Рисунок 3.4 – отфильтрованный датафрейм

Вообще говоря, наиболее значимым в этой задаче является именно параметр eps. При очень малых его значениях, каждая точка будет рассматриваться в такой окрестности, в которую почти ни одна другая точка не попадет. Таким образом будет наблюдаться огромное количество шума. Наоборот, при очень больших значениях данного параметра, в окрестность почти любой точки будут попадать почти все остальные точки. Таким образом минимизируется итоговое количество кластеров. Поэтому, если необходимо уменьшить количество шума – нужно увеличивать этот параметр. Если же нужно увеличить количество кластеров – уменьшать.

В данном случае, при eps=0,2 и min_samples=2, кластеров хоть и нашлось 16 штук, однако преобладает среди них лишь один, что объясняется малым значением min_pts (Рисунок 3.5).

```
new_pcd, colors, max_label, obj_points = segment_pcd(X, pcd, points, 0.2, 2)
print(f"Распознано {max_label} объектов в облаке точек")
```

```
Распознано 16 объектов в облаке точек
```

```
new_pcd, colors = add_color(new_pcd, colors)
draw_plot(points, colors)
```

Рисунок 3.5 – результат сегментации при eps=0,2 и min_samples=2

При увеличении значений параметров eps и min_samples до 0,65 и 298 соответственно, наблюдается единственный кластер, что обусловлено большим значением параметра eps – два соседних дерева рассматриваются по сути как окрестность одной точки (Рисунок 3.6).

```
new_pcd, colors, max_label, obj_points = segment_pcd(X, pcd, points, 0.65, 298)
print(f"Распознано {max_label} объектов в облаке точек")
```

```
Распознано 1 объектов в облаке точек
```

```
new_pcd, colors = add_color(new_pcd, colors)
draw_plot(points, colors)
```

Рисунок 3.6 – результаты сегментации при eps = 0,65, min_samples = 298

При значениях параметров eps=0,2 и min_samples=100 наблюдается большое количество шумных точек, что объясняется малой окрестностью и сильным требованием на необходимое количество точек в данной окрестности (Рисунок 3.7)

```
new_pcd, colors, max_label, obj_points = segment_pcd(X, pcd, points, 0.2, 100)
print(f"Распознано {max_label} объектов в облаке точек")
```

```
Распознано 5 объектов в облаке точек
```

```
new_pcd, colors = add_color(new_pcd, colors)
draw_plot(points, colors)
```

Рисунок 3.7 – результаты сегментации при eps = 0,2 и min_samples = 100

Наилучшие результаты были достигнуты при eps=0,45 и min_samples=150. При заданных параметрах получаем необходимое число кластеров, которые визуально выглядят правильными и похожими на настоящие деревья, при этом количество шума не велико (Рисунок 3.8).

```
new_pcd, colors, max_label, obj_points = segment_pcd(X, pcd, points, 0.45, 150)
print(f"Распознано {max_label} объектов в облаке точек")
```

```
Распознано 4 объектов в облаке точек
```

```
new_pcd, colors = add_color(new_pcd, colors)
draw_plot(points, colors)
```

Рисунок 3.8 – результаты сегментации при eps = 0,45 и min_samples = 150

Как уже упоминалось, Silhouette Score является популярной метрикой, когда нужно оценить кластеризацию на неразмеченных данных. Вопрос заключается в том, подходит ли данная метрика к рассматриваемой предметной области, полезна ли она в принципе и может ли быть использована для объективной оценки и автоматизации подбора параметров алгоритма DBSCAN.

С этой целью ранее были найдены значения метрики Silhouette Score для каждой пары eps и min_samples из выбранного ранее интервала их значений: eps меняется от 0,2 до 0,65 и min_samples – от 2 до 298. В первом случае (Рисунок 3.5) Silhouette Score равен -0.377282, во втором случае (Рисунок 3.6) невозможно вычислить значение метрики, так как получен всего один кластер, в третьем случае (Рисунок 3.7) – 0.03874 и, наконец, для самого успешного случая (Рисунок 3.8) Silhouette Score равен 0.262.

Таким образом, даже для очень хорошей сегментации деревьев, значение метрики, вообще говоря, может быть не велико, а установить какую-либо четкую границу для приемлемого качества кластеризации не представляется возможным. Более того, высокое значение метрики (даже выше, чем для самого

удачного случая) не гарантирует хорошее качество сегментации. Например, для параметров с лучшим значением метрики: eps=0,55 и min_samples=292, качество сегментации сильно хуже. В том числе три из четырех деревьев сегментированы неправильно, шума больше (Рисунок 3.9, Рисунок 3.10).

```
test_df.sort_values(by='Silhouette Score', ascending=False)
```

	eps	min_pts	Number of clusters	Silhouette Score
1188	0.55	292	7	0.325824
1191	0.55	298	7	0.325604
1189	0.55	294	7	0.325544
1190	0.55	296	7	0.325408
1021	0.50	256	7	0.296033
...	...	...	...	...
1485	0.65	290	1	NaN
1486	0.65	292	1	NaN
1487	0.65	294	1	NaN
1488	0.65	296	1	NaN
1489	0.65	298	1	NaN

1490 rows × 4 columns

Рисунок 3.9 – параметры кластеризации с наилучшим значением метрики

```
new_pcd, colors, max_label, obj_points = segment_pcd(X, pcd, points, 0.55, 292)
print(f"Распознано {max_label} объектов в облаке точек")
```

```
Распознано 7 объектов в облаке точек
```

```
new_pcd, colors = add_color(new_pcd, colors)
draw_plot(points, colors)
```

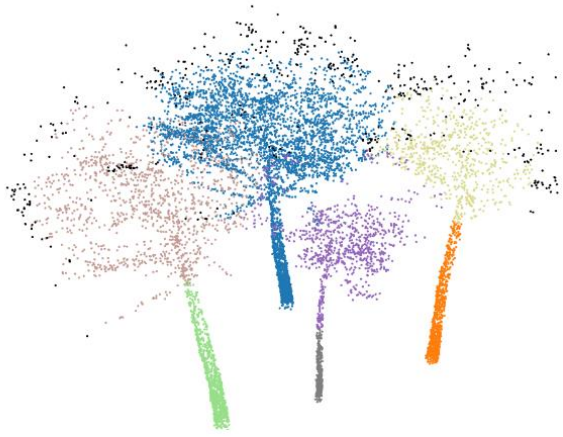

Рисунок 3.10 – сегментация на параметрах с лучшим значением метрики

4 Сегментация облака точек большой размерности

Была произведена сегментация облака точек большой размерности. Исходные данные представляют собой массив трехмерных точек, количество которых равно 46744308 (Рисунок 4.1), полученных в результате съемки лидаром множества деревьев.

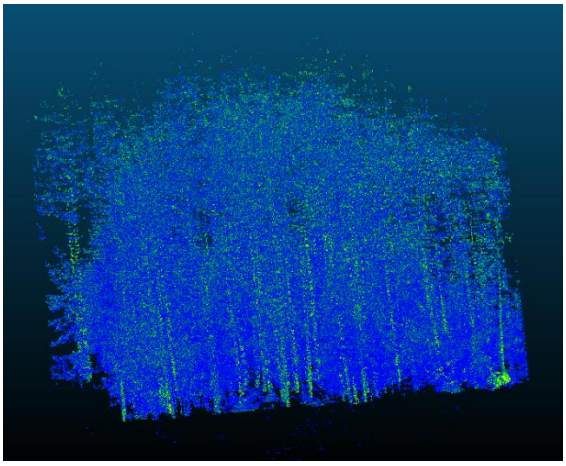

Рисунок 4.1 – облако точек большой размерности

Как уже было упомянуто ранее, DBSCAN расходует огромное количество оперативной памяти и в том числе использует файл подкачки на диске для хранения матрицы расстояний между точками в облаке. Так как исходный файл с облаком занимает чуть больше 1 Гб, то становится очевидно, что без прореживания облака алгоритм просто никогда не завершит работу и в конечном итоге займет всю доступную память и прервет выполнение работы.

Для прореживания такого облака точек полезно посмотреть на распределение точек по высоте. Соответствующий код и его вывод представлены ниже (Рисунок 4.2).

```
# Распределение точек по высоте
sns.displot(data=points[:, 2], kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x1c8380b0490>
```

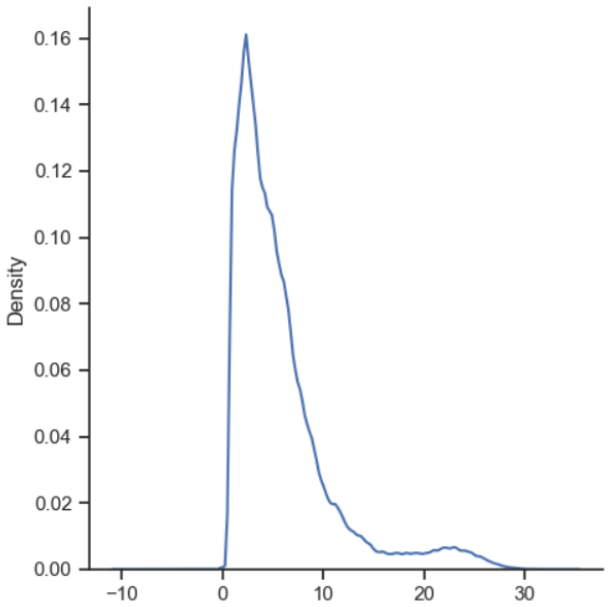

Рисунок 4.2 – анализ плотности распределения точек по высоте

Сразу заметна неравномерность распределения: точек, находящихся ближе к земле, сильно больше, чем точек, относящихся к кронам деревьев и верхней части стволов. Более того, есть точки, располагающиеся явно ниже самих деревьев – земля или прочий шум (Рисунок 4.1). С этой целью можно провести два типа прореживания: сначала неравномерное – чтобы сгладить исходное распределение, затем равномерное – чтобы получить необходимое число точек в облаке.

Вообще говоря, из полученного графика можно получить набор точек, по которым сама кривая и строится, и использовать их для построения функции плотности распределения вероятности того, что точка на данной высоте будет отброшена. Проблема в том, что координаты точек могут не совпадать с исходными, что приводит к необходимости интерполирования функции для огромного количества точек, что в свою очередь приводит к огромным вычислительным и временным затратам. Поэтому была построена императивно (исходя из распределения, «на глаз») кусочно-линейная функция вероятности отбрасывания точки на заданной высоте (Рисунок 4.3).

```
plt.plot([-15, 1, 1, 10, 10, 15, 15, 35],
         [0, 0, 0.995, 0.88, 0.9, 0.4, 0.2, 0.2])
plt.xlabel("Высота точки")
plt.ylabel("Вероятность удаления точки")
```

```
Text(0, 0.5, 'Вероятность удаления точки')
```

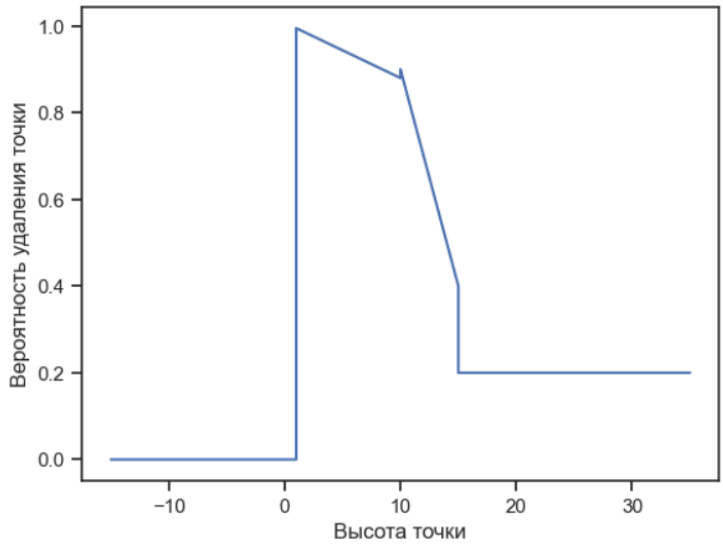

Рисунок 4.3 – график функции вероятности отбрасывания точки

Было решено отбрасывать все точки с высотой менее 1. Точки с высотой более 15 равномерно отбрасываются с довольно малой вероятностью, чтобы оставить нормальную плотность на верхушках стволов и в кронах. Точки с высотой от 1 до 15 отбрасываются с очень большой вероятностью, так как на этот интервал их приходится подавляющее количество. Была реализована соответствующая функция прореживания (Рисунок 4.4).

```
def reduce_pcd(points_):
    ground_treshold = 1
    p, rl, rr, ql, q = 0.995, 0.88, 0.90, 0.4, 0.2
    rand = random.Random()
    reduced_pcd = []
    for point in points_:
        if point[2] < ground_treshold:
            continue
        if point[2] > 15:
            if rand.random() > q:
                reduced_pcd.append(point)
        elif point[2] < 10:
            if rand.random() > p + (point[2] - 0) * (rl - p) / 10:
                reduced_pcd.append(point)
        else:
            if rand.random() > rr + (point[2] - 10) * (ql - rr) / 5:
                reduced_pcd.append(point)
    return np.asarray(reduced_pcd)
```

Рисунок 4.4 – функция неравномерного прореживания точек по высоте

Результат работы функции и итоговое распределение точек представлены ниже (Рисунок 4.5). Стоит отметить, что в целом распределение было выровнено, хотя и неидеально, и не совсем автоматически.

```
test_reduce_pcd = reduce_pcd(points)
test_reduce_pcd.shape
```

```
(5411689, 3)
```

```
# Распределение облака точек сниженной размерности
sns.displot(data=test_reduce_pcd[:, 2], kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x1c8380f3280>
```

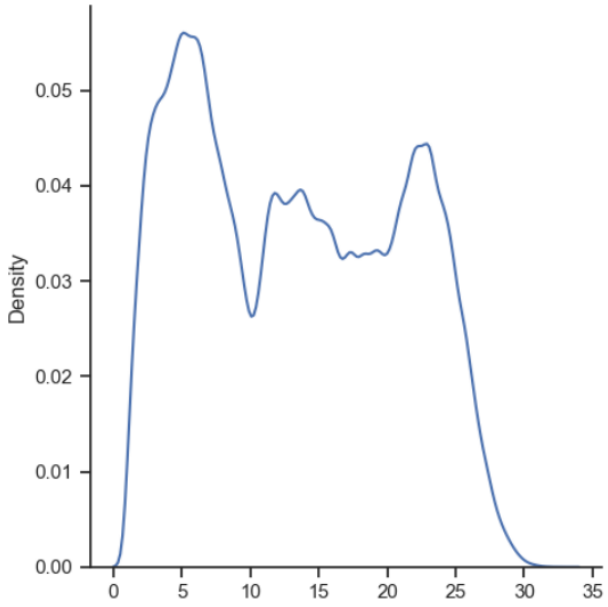

Рисунок 4.5 – результирующее распределение точек по высоте после прореживания

По итогу было получено 5411689 точек, что все еще слишком много для работы алгоритма. Так как теперь распределение выровнялось, стало возможным применение равномерного прореживания до необходимого числа точек – до 200000. Функция равномерного прореживания и результаты её работы представлены ниже (Рисунок 4.6, Рисунок 4.7, Рисунок 4.8).

```
# до ~ num_points точек
def down_pcd(data, num_points):
    if data.shape[0] > num_points:
        factor = data.shape[0] // num_points
    else:
        factor = 1
    down_data = data[::factor]
    return down_data
```

Рисунок 4.6 – функция равномерного прореживания точек в облаке

```
res_points = down_pcd(test_reduce_pcd, 200_000)
```

```
o3d.visualization.draw_geometries([np_to_pcd(res_points)])
```

```
res_points.shape
```

```
(202840, 3)
```

Рисунок 4.7 – применение прореживания и итоговое число точек в облаке

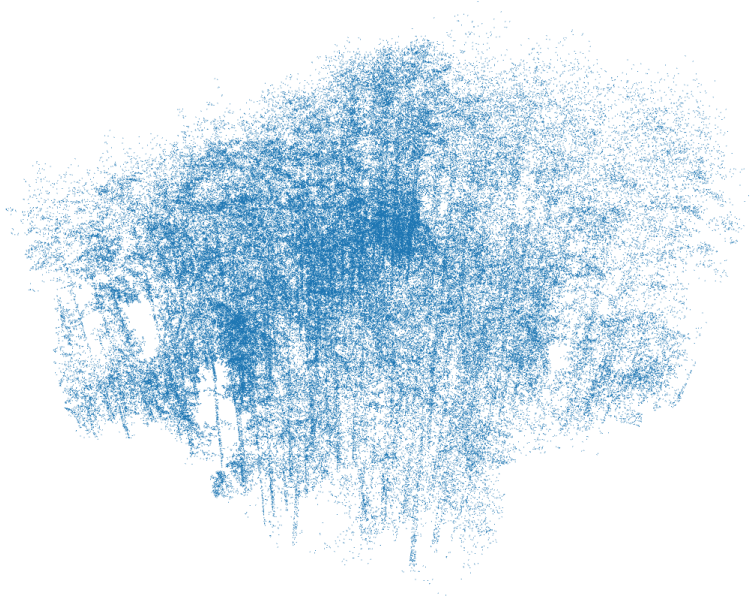

Рисунок 4.8 – результирующее облако точек

Использование найденных ранее параметров eps=0,45 и min_samples=150, оптимальных для сегментации предыдущего облака точек, содержащего в себе 4 дерева, оказалось неэффективным для сегментации данного облака точек: все точки были определены как шум (Рисунок 4.9).

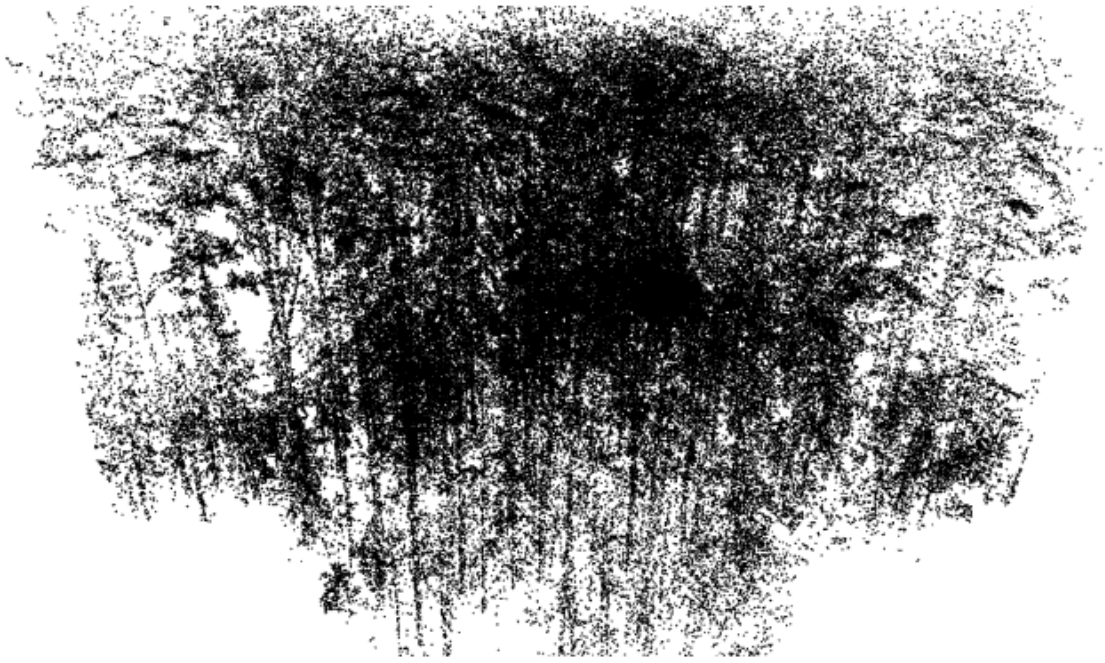

Рисунок 4.9 – результат сегментации при eps = 0,45 и min_samples = 150

Из полученного результата следует, что значения параметров используемого алгоритма не являются универсальными даже в рамках одной задачи – в нашем случае сегментации деревьев. Поэтому, как и ранее был выполнен небольшой перебор параметров и анализ результатов сегментации при их соответствующих значениях.

При относительно больших значениях параметров eps=2 и min_samples=100 результатом сегментации является единственный кластер с малым количеством шума. Тогда, исходя из предыдущих рассуждений, приходим к выводу, что eps нужно уменьшать, так как окрестность одной точки захватывает соседние деревья. Вместе с уменьшением eps также необходимо уменьшать и min_samples, чтобы шум не начал преобладать в результатах сегментации (Рисунок 4.10).

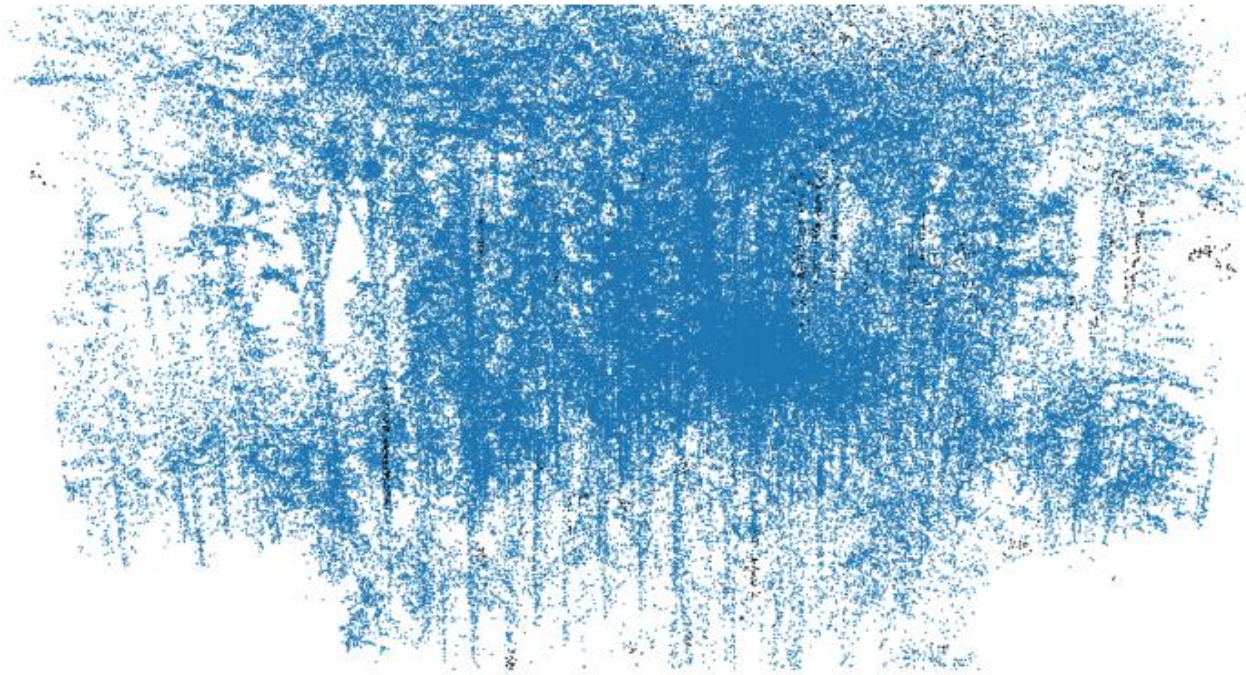

Рисунок 4.10 – результат сегментации при eps = 2 и min_samples = 100

При малых eps и min_samples почти все точки определяются как шум, за исключением некоторых редких сегментов. При малых eps не имеет смысла увеличивать значение min_samples, тогда необходимо увеличивать параметр

eps, который в конечном счете выдаст лучший результат, приняв значение из интервала от 0,3 до 0,9 (Рисунок 4.11, Рисунок 4.12).

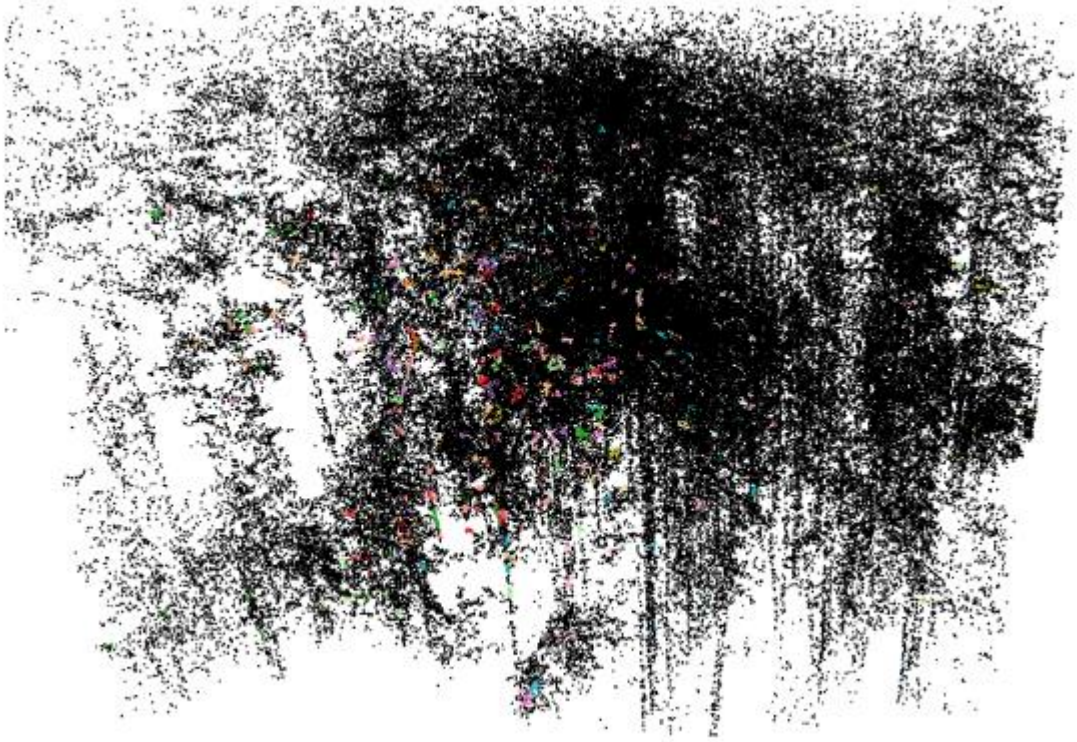

Рисунок 4.11 – результат сегментации при eps = 0,2 и min_samples = 10

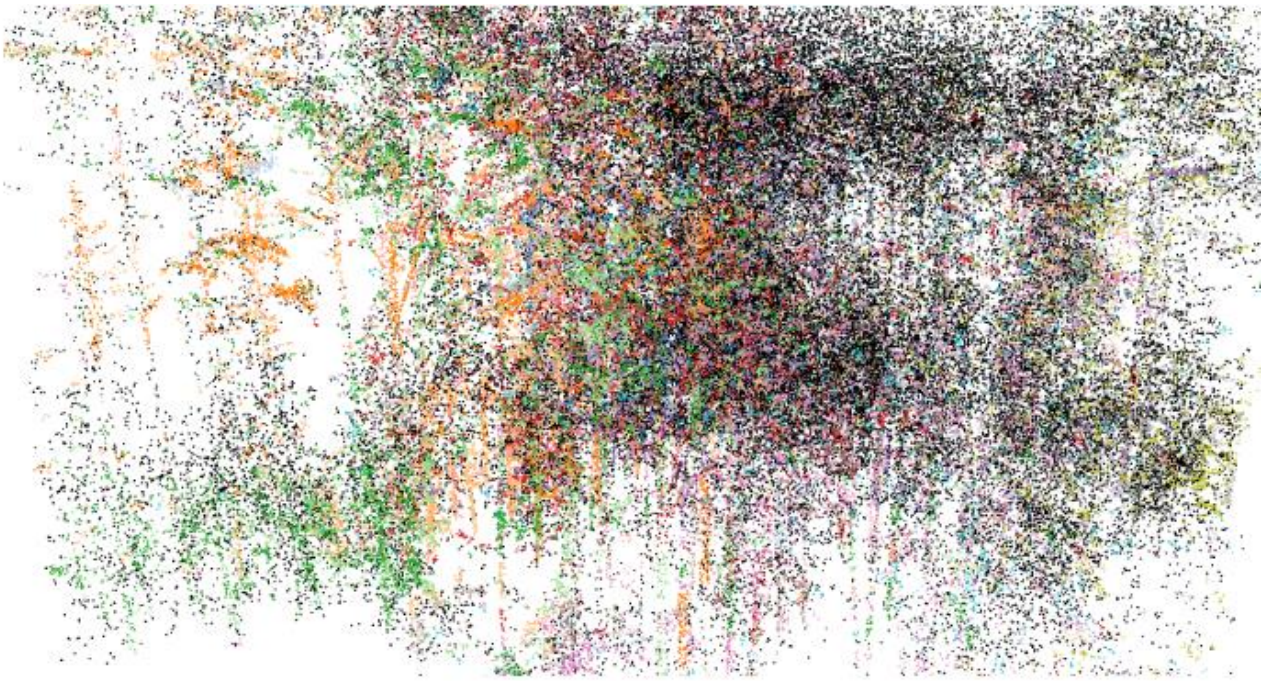

Рисунок 4.12 – результат сегментации при eps = 0,2 и min_samples = 3

При средних значениях параметров eps и min_samples ситуация, к сожалению, не улучшается: при eps=0,6 и min_samples=50 большинство точек все также определяется как шум, что приводит к необходимости повторного уменьшения параметра min_samples. Однако, уменьшение значение min_samples до 25 хоть и приводит к уменьшению шума и увеличению количества сегментированных точек, однако некоторые соседние деревья начинают сливаться в один кластер (Рисунок 4.13, Рисунок 4.14).

Рисунок 4.13 – результат сегментации при eps = 0,6 и min_samples = 50

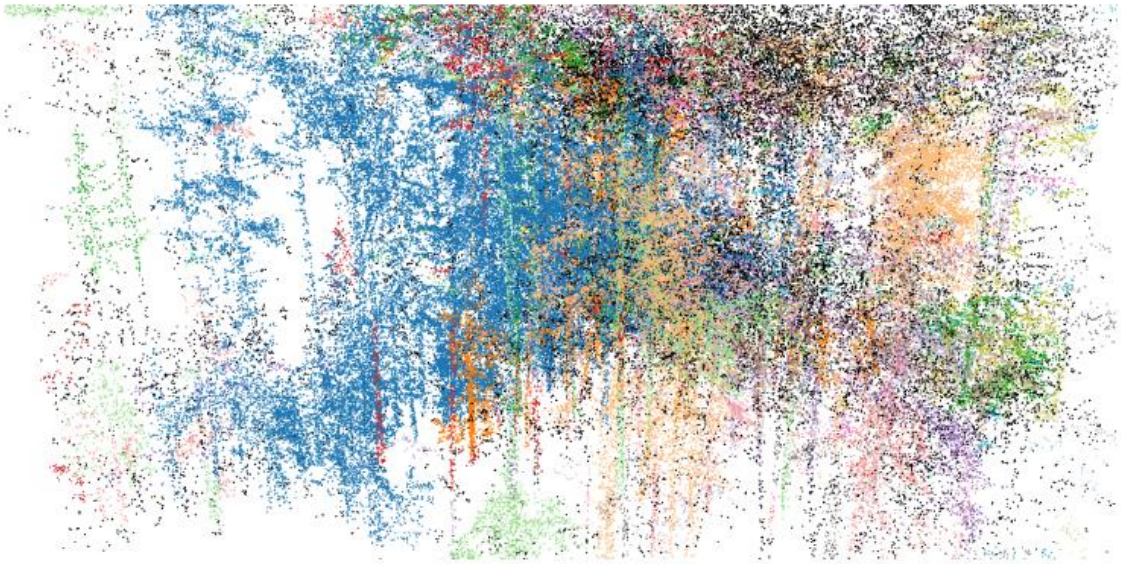

Рисунок 4.14 – результат сегментации при eps = 0,6 и min_samples = 25

При значениях параметров eps=0,45 и min_samples=8 решить проблемы с шумом и слиянием соседних деревьев в один кластер также не удалось (Рисунок 4.15).

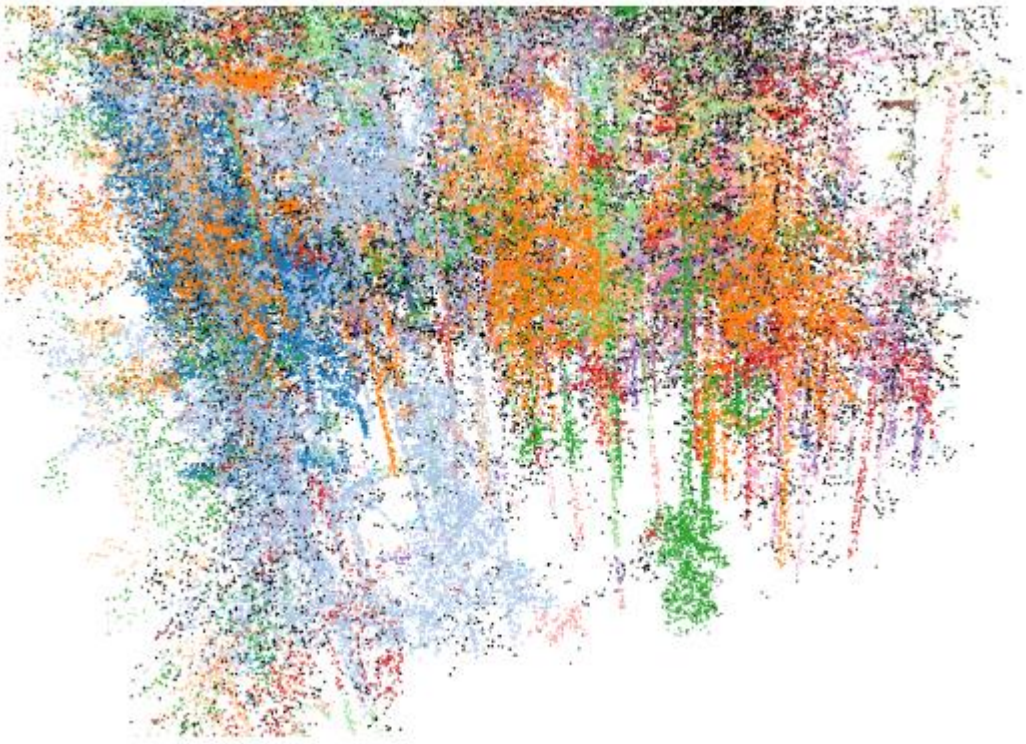

Рисунок 4.15 – результат сегментации при eps = 0,45 и min_samples = 8

При значениях параметров eps=0,75 и min_samples=25 также не удалось разрешить обозначенные выше две проблемы. Шума все также много, из чего следует необходимость уменьшить min_samples, из чего в свою очередь следуют необходимость уменьшить и eps, чтобы соседние не сливались в один кластер. Но тогда возвращаемся к предыдущим случаям (Рисунок 4.16).

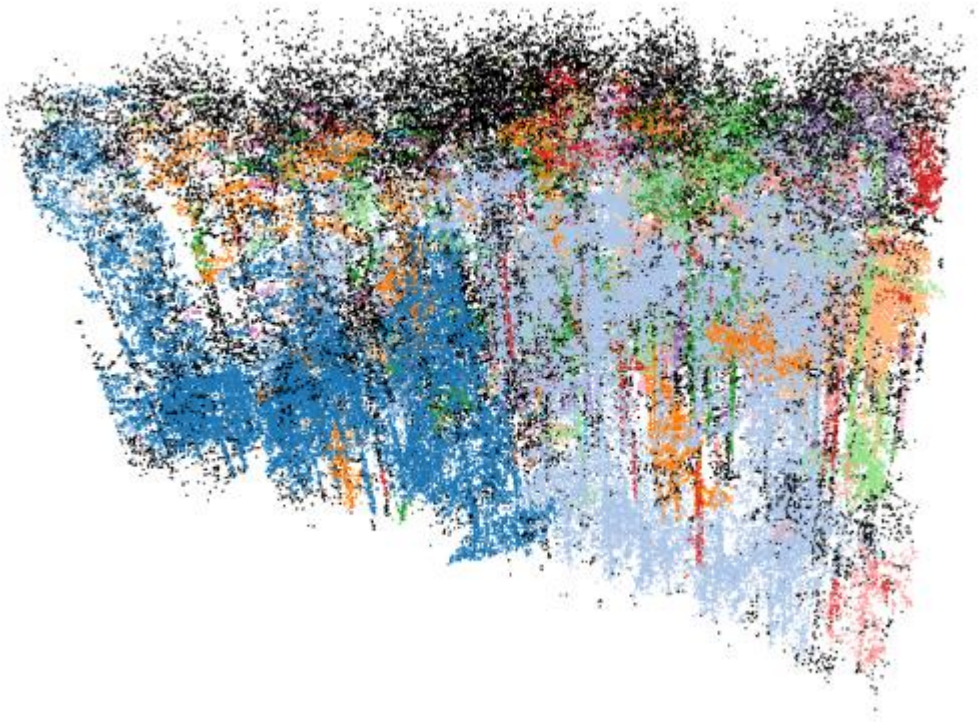

Рисунок 4.16 – результат сегментации при eps = 0,75 и min_samples = 25

Было решено воспользоваться методом для нахождения оптимального значения eps для проведения сегментации. Метод заключается в нахождении “локтя” на графике, который можно построить с использованием библиотеки kneed, и определении eps, соответствующего этому “локтю”. Было построено несколько графиков, каждый из которых строился с учетом среднего расстояния между ближайшими n соседями. Полученные значения «локтя» равны соответственно 0,75, 1, 1,49 (Рисунок 4.17 – Рисунок 4.20).

```
from kneed import KneeLocator
from sklearn.neighbors import NearestNeighbors

n_neighbors = 10
nearest_neighbors =  NearestNeighbors(n_neighbors=n_neighbors + 1)
neighbors = nearest_neighbors.fit(X)
distances, indices = neighbors.kneighbors(X)
distances = np.sort(distances[:, n_neighbors], axis=0)

i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex',
                   direction='increasing', interp_method='polynomial')
fig = plt.figure(figsize=(20, 10))
knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")

print(distances[knee.knee])
```

Рисунок 4.17 – код построения и нахождения «локтя»

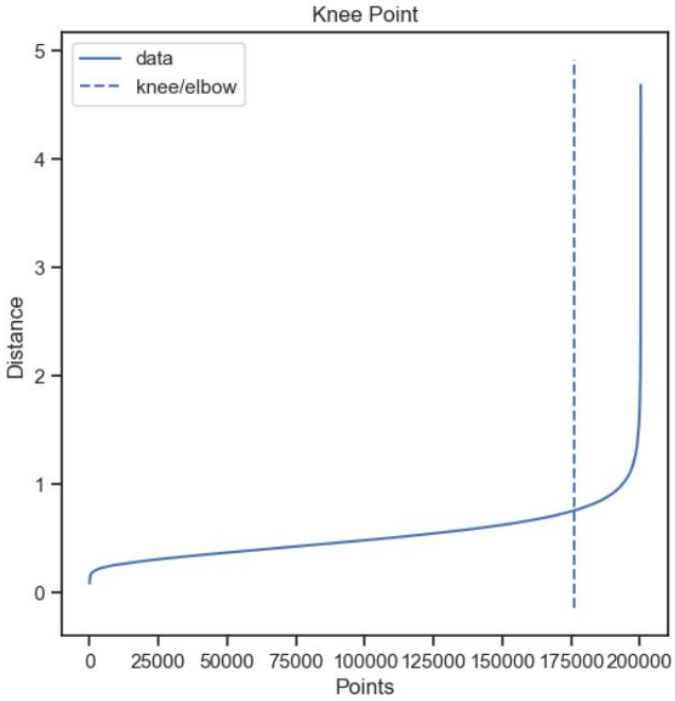

Рисунок 4.18 – график для n = 10

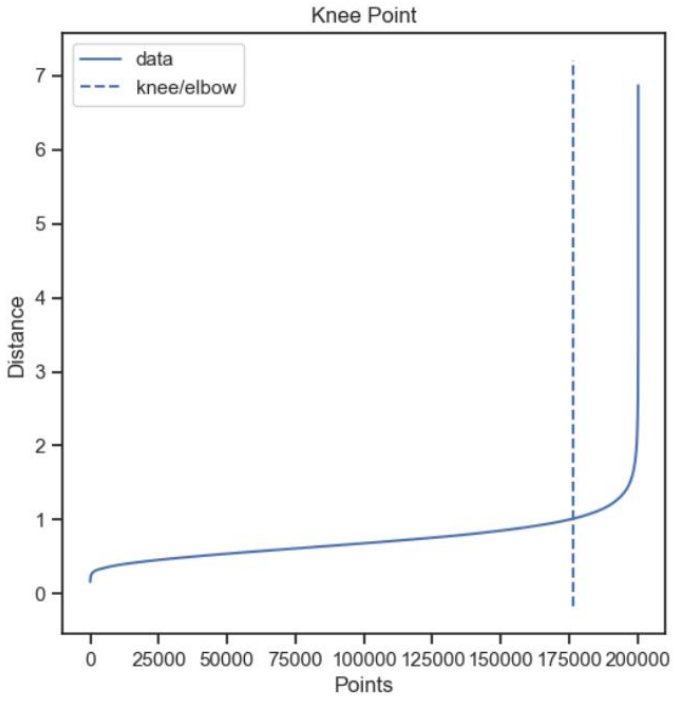

Рисунок 4.19 – график для n = 25

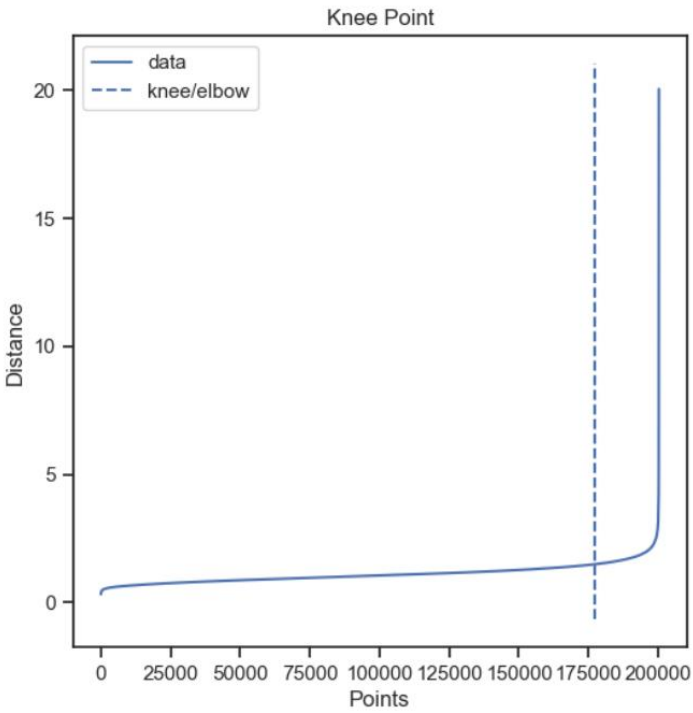

Рисунок 4.20 – график для n = 50

Для второго и третьего найденных выше значений eps было подобрано наиболее удачное значение второго параметра min_samples (Рисунок 4.21, Рисунок 4.22).

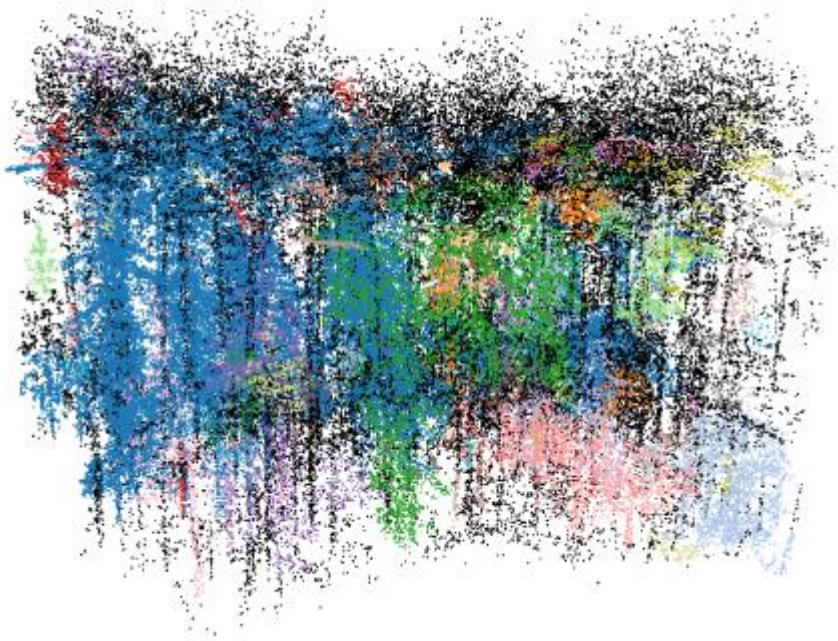

Рисунок 4.21 – результат сегментации при eps = 1 и min_samples = 50

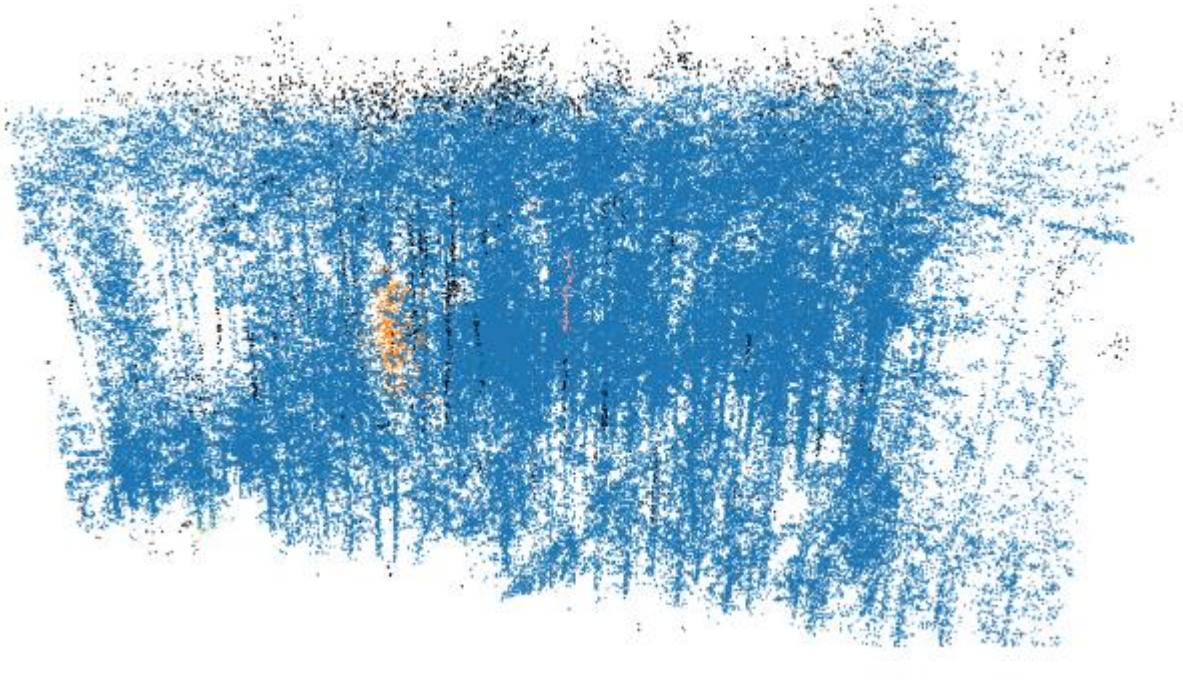

Рисунок 4.22 – результат сегментации при eps = 1,49 и min_samples = 78

ЗАКЛЮЧЕНИЕ

В результате выполнения научно-исследовательской работы были выполнены следующие задачи:

1. С использованием метода машинного обучения DBSCAN была произведена сегментация деревьев из облака точек относительно малой размерности. В результате сегментации было получено четыре кластера, соответствующие отдельным деревьям.
2. Была произведена оценка качества кластеризации с использованием метрики Silhouette Score. Невысокое значение метрики = 0,262 может быть обусловлено тем, что деревья являются вытянутыми, протяженными объектами, поэтому среднее расстояние от точки одного кластера до других точек этого же кластера может оказаться больше, чем для более типичных кластеров, что вносит вклад в значение метрики Silhouette Score. Было выяснено, что данная метрика не является надежной и подходящей для данной задачи, но возможна ее доработка до необходимых условий.
3. С использованием метода машинного обучения DBSCAN была произведена сегментация деревьев из облака точек большой размерности. Облако точек было прорежено последовательно двумя способами. Было достигнуто почти равномерное распределение точек по высоте, однако вклад кустов и низких деревьев мало изменился, что подводит к необходимости дальнейшей очистки облака точек. Был достигнут результат частичной сегментации данного облака точек на деревья. Основная проблема в том, что деревья расположены близко и могут иметь тонкий ствол, в связи с чем плотностные алгоритмы, в том числе и исследуемый, не дают требуемых результатов.
4. Был использован графический метод определения оптимального значения параметра eps для сегментации облака точек. Причиной, по которой не удалось достичь полной сегментации облака точек, может быть нарушение совершаемого при использовании такого метода

допущения об одинаковой размерности и средней плотности кластеров.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Fred Westling, James Underwood, Mitch Bryson. Graph-based methods for analyzing orchard tree structure using noisy point cloud data: [Электронный ресурс]. // arXiv.org. 2023. Дата обновления: 02.02.2021. URL: https://arxiv.org/abs/2009.13727 (Дата обращения: 29.03.2023).

2. Chu Chen, Yanqi Ma, Bingcheng Dong, Junjie Cao. DMNR: Unsupervised De-noising of Point Clouds Corrupted by Airborne Particles: [Электронный ресурс]. // arXiv.org. 2023. Дата обновления: 10.05.2023. URL: https://arxiv.org/abs/2305.05991 (Дата обращения: 29.05.2023).

3. Yuchen Bai, Jean-Baptiste Durand, Florence Forbes, Grégoire Vincent. Semantic segmentation of sparse irregular point clouds for leaf/wood discrimination: [Электронный ресурс]. // arXiv.org. 2023. Дата обновления: 26.05.2023. URL: https://arxiv.org/abs/2305.16963 (Дата обращения: 1.06.2023).

4. Kaisen Ma, Zhenxiong Chen, Liyong Fu, Wanli Tian, Fugen Jiang, Jing Yi, Zhi Du, Hua Sun. Performance and Sensitivity of Individual Tree Segmentation Methods for UAV-LiDAR in Multiple Forest Types: [Электронный ресурс]. // mdpi.com. 2023. Дата обновления: 10.01.2022. URL: https://www.mdpi.com/2072-4292/14/2/298 (Дата обращения: 29.03.2023).

5. Feiyu Wang, Mitch Bryson. Tree Segmentation and Parameter Measurement from Point Clouds Using Deep and Handcrafted Features: [Электронный ресурс]. // researchgate.net. 2023. Дата обновления: 16.02.2023. URL: https://www.researchgate.net/publication/368612705_Tree_Segmentation_and_Parameter_Measurement_from_Point_Clouds_Using_Deep_and_Handcrafted_Features (Дата обращения: 29.03.2023).