Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Майстренко Марат Алексеевич

Параллельная сегментация точечных изображений деревьев: сравнительный анализ и оптимизация методов машинного обучения для эффективного разделения кроны и листвы

КУРСОВАЯ РАБОТА

Научный руководитель:
И.М. Никольский

Москва, 2025

Аннотация

Курсовая работа посвящена исследованию методов параллельной сегментации облаков точек LiDAR сканирования деревьев с применением алгоритмов машинного обучения. Работа включает введение, три главы, заключение, список использованных источников (14 наименований) и приложения. Основной объём работы изложен на 52 страницах и содержит 14 таблиц, 13 рисунков и 6 графиков.

В первой главе рассмотрены теоретические основы работы LiDAR, форматы данных LAS и LAZ, а также методы сегментации облаков точек. Во второй главе представлена методология исследования, включая предобработку данных, применение алгоритмов машинного обучения (KNN, Decision Tree, Random Forest, Stacking, Blending) и анализ методов распараллеливания вычислений (NumPy, Joblib, Scikit-learn). Третья глава содержит результаты сравнительного анализа методов, оценку точности сегментации и визуализацию данных.

Цель работы — разработка рекомендаций по оптимизации и ускорению обработки облаков точек за счёт распараллеливания алгоритмов. Результаты исследования демонстрируют значительное ускорение вычислений при использовании комбинации Joblib и Scikit-learn, что позволяет эффективно обрабатывать большие объёмы данных.

Ключевые слова: LiDAR, облако точек, сегментация, машинное обучение, распараллеливание, Random Forest, KNN, Joblib, Scikit-learn.

Оглавление

Введение

Современные технологии лазерного сканирования, в частности LiDAR (Light Detection and Ranging), находят широкое применение в задачах построения трёхмерных моделей объектов и анализа окружающей среды. Одной из актуальных задач является сегментация облаков точек, полученных с помощью LiDAR, для выделения различных структур — например, кроны и листвы деревьев. Объёмы данных, генерируемых такими сканерами, чрезвычайно велики, что обуславливает высокие требования к скорости и эффективности обработки информации.

Актуальность темы исследования обусловлена необходимостью разработки и оптимизации методов машинного обучения для автоматической сегментации облаков точек. Особое значение имеет распараллеливание вычислений, позволяющее значительно сократить время обработки больших наборов данных.

Целью данной работы является исследование и сравнение эффективности различных методов машинного обучения для сегментации облаков точек LiDAR, а также анализ методов распараллеливания вычислений для ускорения обработки данных.

Объектом исследования являются облака точек LiDAR-сканирования деревьев, а предметом исследования — методы машинного обучения и распараллеливания для их сегментации.

1 Теоретические основы сегментации облаков точек

Облако точек - набор точек данных в трёхмерной системе координат. Это не полноценный объект, а неупорядоченный и неструктурированный массив, для обработки которого требуются значительные вычислительные ресурсы.

Каждая точка представляет собой одно пространственное измерение на поверхности какого-то объекта (например, здания), и в совокупности облако точек представляет собой всю его внешнюю поверхность.

Кроме координат в каждой точке может храниться атрибутивная информация об интенсивности, RGB цвет и времени. Облака точек возникают во многих задачах из реального трёхмерного мира и позволяют нам понять, как он выглядит. Обработка таких данных важна в строительстве, архитектуре, археологии, картографии, землеустройстве и т.п.

1.1 Понятие и принципы работы LiDAR

LiDAR (Light Detection and Ranging) — технология, использующая лазерное излучение для точного измерения расстояний до объектов и их пространственного расположения. Данный метод является одним из самых популярных способ получения облаков точек. [1]

Принцип работы LiDAR состоит в следующем: посылается луч, он отражается от поверхности и возвращается на детектор. Расстояние вычисляется исходя из засеченного времени и скорости света.

Данная технология позволяет вычислить интенсивность для каждой точки, которая характеризуется мощностью отраженного сигнала. Например, у стекла или воды мощность отражения чуть больше нуля, когда у дерева или камня приближается к единице. [2]

Важное преимущество LiDAR - возможность получения отклика на расстоянии свыше 200 метров с сохранением высокой точности измерений.

Сканирование происходит за счёт генерации лазерного импульса, замера времени и изменения поляризации. Скорость сканирования варьируется от задачи. Есть два вида LiDAR-сканеров:

1) Сканеры, осуществляющие точечное сканирование. В один момент времени аппаратура может отправлять и получать только один сигнал. Сканирование требует большое количество временных затрат, но имеет в финале высокую точность.
2) Многолучевые, матричные LiDAR-сканеры. Использование нескольких лазерных источников позволяет уменьшить время, затрачиваемое для сканирования, за счёт уменьшения плотности точек в финальном облаке. [3]

Для задач, связанных со сканированием объекта с широким диапазоном обзора, лучше подойдёт многолучевой сканер, когда точечный LiDAR подходит больше для получения детальных трёхмерных моделей отдельных вещей.

1.2 Форматы данных сканирования LiDAR (LAS и LAZ)

LAS (LiDAR Aerial Survey) – бинарный формат хранения и обмена данных, получаемых LiDAR сканером. Стандарт был разработан ASPRS в 2003 году.

В файле может храниться множество атрибутов. Одни из них: цвет, интенсивность отклика, GPS-время, класс точки. Формат данных имеет в файле заголовок с метаданными: информация о системе координат, время съемки, использующиеся атрибуты, программа в которой был создан файл. Благодаря этой «шапке» есть возможность добавлять новые признаки для точки.

Формат LAZ (сокращение от LASzip) – формат для хранения данных, разработанный в 2007 году. LAZ используется для сжатия данных LAS файла на 80-90% без потери информации.

Стандарты LAS и LAZ имеют высокую совместимость с различными программными средствами, используемыми для анализа и визуализации облаков точек. С помощью них обеспечивается эффективный обмен и обработка без необходимости преобразования данных.

1.3 Методы сегментации облаков точек

Сегментация — это процедура классификации объектов в однородные группы согласно их схожим характеристикам. **Кластеризация** — процесс поиска сходства с целью объединения в группы. Кластеризация находит связь между точками данных, чтобы их можно было сегментировать. [4]

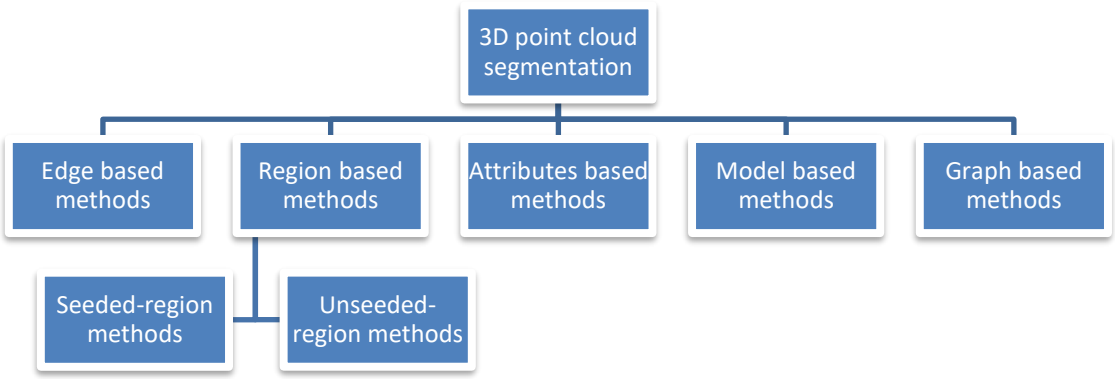

Рис.1
структурная классификация методов сегментации облаков точек

Edge-based methods основаны на поиске точек, имеющих резкое изменении в одной из характеристики точек. Граница выделяется по принципу резкого различия определенного атрибута, что делает невозможным его использования в данных с шумом и с неравномерной плотностью точек.

Region-based methods – методы, основанные на принципе объединения близлежащих точек, имеющих сходные свойства, для получения и поиска различных изолированных областей. Данные методы делятся на два множества:

1) *Seeded (bottom-up) method* зависит от изначально выбранных точек, от которых далее идёт сегментация пространства. Получить границу таким способом проблематично, так как присутствует большая чувствительность к порогам совместимости.

2) На вход *Unseeded (top – down) method* подается все облако точек. Далее он начинает самостоятельно анализировать и сегментировать на область. Данный метод требует предварительных знаний и легко приводит к избыточной сегментации.

При *Attributes-based method* вычисляются дополнительные атрибуты, основываюсь на которых происходит дальнейшая сегментация облака точек. При многомерном пространстве признаков задача сегментации становится крайне трудоемкой, зачастую в этом случае выделяют главные компоненты (например, с помощью PCA) для работы алгоритма. Достоинством данного метода является неподверженность влиянию шума.

Model-based methods основываются на поиске геометрических примитивов для группировки точек. Эти методы работают быстро и устойчивы к выбросам, но могут давать неточные результаты при обработке данных из разных источников.

Graph-based methods исходят из принципа, что каждая точка – это вершина графа. Все вершины попарно соединяются ребрами, основываясь на выбранной нами метрики расстояния, которая определяет степень различия между точками данных. В результате работы алгоритма мы имеем граф, разделенный на связные компоненты. Метод хорошо сегментирует сложные сцены, включающие шум или неравномерную плотность. Но алгоритмы, основанные на графах, требуют больших вычислительных, временных затрат и тонкой настройки пороговых значений. [5, 6, 7]

Понимание сильных и слабых сторон каждого метода позволяет исследователям и инженерам выбирать наиболее подходящий подход для конкретной задачи исходя из ряда факторов, включая специфику решаемой задачи, характеристики входных данных, требуемую точность результатов и доступные временные и вычислительные ресурсы.

2 Постановка задачи

Целью данной работы является исследование эффективности методов параллельной сегментации облаков точек LiDAR-сканирования деревьев с применением алгоритмов машинного обучения для оптимизации скорости и качества обработки данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Изучить теоретические основы работы LiDAR-сканеров, форматов хранения данных и методов сегментации облаков точек;
2. Провести предобработку исходных данных LiDAR-сканирования;
3. Изучить специфику работы методов машинного обучения для сегментации облаков точек, включая K-Nearest Neighbors (KNN), Decision Tree, Random Forest, Stacking и Blending;
4. Исследовать и применить методы распараллеливания вычислений с использованием библиотек NumPy, Joblib и Scikit-learn;
5. Изучить способы сравнения и метрики оценка точности методов машинного обучения для сегментации облаков точек;

6. Провести сравнительный анализ точности сегментации и скорости работы различных методов;
7. Разработать рекомендации по выбору оптимальных подходов для эффективной сегментации и ускорения обработки больших объемов данных.

3 Методология исследования

В рамках исследования проведена работа по предобработке данных и применению различных алгоритмов классификации и сегментации для получения наилучших результатов. Исходные данные, полученные LiDAR сканированием, в формате LAS или LAZ.

В исследовании использованы несколько методов машинного обучения, таких как K-Nearest Neighbors (KNN), Decision Tree, Random Forest, а также более сложные методы ансамблирования, включая Stacking и Blending. Каждый из методов будет оцениваться по точности сегментации, а также по скорости обработки данных. Особое внимание уделяется распараллеливанию методов, что позволит эффективно работать с большими объемами данных и ускорить процесс обучения моделей.

Таким образом, цель исследования заключается в том, чтобы выбрать наиболее подходящие методы для сегментации облаков точек, обеспечить оптимальную точность предсказаний и минимизировать затраты времени на обработку и обучение моделей с помощью распараллеливания алгоритмов.

3.1 Описание исходных данных

В качестве данных, на которых проводилось тестирование методов сегментации и их распараллеленных вариантов, был выбран датасет, содержащий 9 облаков точек деревьев, полученных с помощью наземного лазерного сканирования (TLS) (в формате LAZ версии 1.4). Облака точек были вручную помечены точками листьев и древесины (0 = дерево, 1 = лист). Облака точек имеют дополнительные атрибуты (отклонение, коэффициент отражения, Амплитуда, GPS-время, PointSourceId, количество возвратов). При работе с датасетом использовались координаты x, y, z и истинная классификация.

3.2 Предобработка данных

Ввиду большого объема файлов, значительного количества точек и ограниченных вычислительных ресурсов тестирование методов машинного обучения осуществлялось с уменьшенным в 10 раз количеством точек в облаке, что, тем не менее, оставалось репрезентативным для оценки качества моделей благодаря высокой плотности облака точек. Финальные результаты были получены на полных данных. [8]

Для обучения некоторых моделей и последующего предсказания использовались дополнительные признаки, вычисляемые отдельно для каждой точки. Сначала для каждой точки составлялся список из k=20 её соседей, используя евклидово расстояние. Затем для найденных соседей вычислялась **ковариационная матрица** распределения координат точек в трёхмерном пространстве:

$$C = \frac{1}{N}\sum_{j=1}^{N}(\mathbf{x_j} - \bar{\mathbf{x}})(\mathbf{x_j} - \bar{\mathbf{x}})^T$$

где $\mathbf{x}_j$ - координаты j-ой соседней точки, $\bar{\mathbf{x}}$ - среднее значение координат соседей, а N — их количество.

Эта матрица описывает локальную структуру точечного облака вокруг рассматриваемой точки, и её собственные значения $\lambda_0 \leq \lambda_1 \leq \lambda_2$ используются для расчёта дополнительных признаков:

- **Линейный индекс** $L_\lambda = \frac{\lambda_2 - \lambda_1}{\lambda_2}$ - характеризует степень вытянутости локального распределения точек вдоль главного направления
- **Планарный индекс** $P_\lambda = \frac{\lambda_1 - \lambda_0}{\lambda_2}$ - отражает степень приближённости формы локального облака точек к плоскости
- **Рассеянный индекс** $S_\lambda = \frac{\lambda_0}{\lambda_2}$ - показывает степень равномерности распределения точек в трёхмерном пространстве
- **Кривизна** $C_\lambda = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$ **-** описывает степень локальной шероховатости поверхности.
- **Отношение собственных значений** $R_\lambda = \frac{\lambda_1}{\lambda_2}$ - оценивает структуры локального распределения точек, в частности, степени их анизотропии.

Все вычисленные признаки использовались в дальнейшем анализе данных и моделировании.

3.3 Применяемые методы машинного обучения

Различные способы сегментации облаков точек реализуются с использованием передовых методов машинного обучения с учителем. Для этого в качестве исходных данных используются заранее размеченные файлы, которые подаются на вход алгоритмам. В процессе обучения модель анализирует структурные особенности и закономерности в данных, что позволяет ей извлекать и классифицировать различные элементы облака точек. После завершения обучения алгоритм становится способным автоматически сегментировать новые данные.

Объединение различных моделей машинного обучения позволяет нам выиграть в точности и во времени, так как поддается распараллеливанию. Общая идея ансамблей заключается в том, чтобы построить базовые (слабые) алгоритмы (модели)

$$\{b_i(x)|b_i: x \to R\}_{i=1}^{M},$$

в идеале - независимые, но хотя бы существенно отличающиеся друг от друга. Далее агрегировать их прогнозы в ансамбль

$$a(x) = F\big(b_1(x), \dots, b_M(x)\big),$$

где $F: R^M \to Y$ функция агрегации или мета-алгоритм.

Большим плюсом ансамблевых методов является уникальная предсказательная способность, которая позволяет описывать неординарные зависимости, не поддающиеся к описанию отдельной модели данного типа. Причем качество ансамблевого метода будет гораздо выше качества базовой модели.

3.3.1 K-Nearest Neighbors (KNN)

KNN – один из наиболее распространенных алгоритмов классификации в машинном обучении. Данный метод машинного обучения не требует, как такого «обучения», а просто сохраняет размеченные тренировочные данные. Процесс классификации начинается при появлении новых неразмеченных данных. [9]

Алгоритм можно разделить на 3 этапа:

1. Вычисление декартова расстояния по переменным целевого объекта из m-мерного пространства признаков до каждого из объектов обучающей выборки.

2. Отбор k объектов обучающей выборки, расстояния до которых минимальны

3. Получение класс объекта на основе наиболее часто встречающегося среди k ближайших соседей

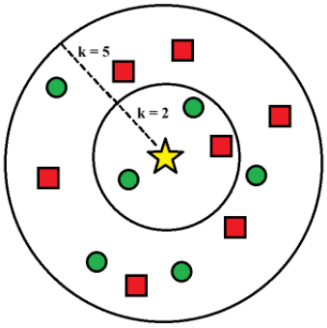

Рис.2
Визуализация работы KNN в 2D случае

3.3.2 Decision Tree

При работе обучении модели используется сегментирование пространства признаков на области, основанное на наборе правил, представимых в виде дерева. Этот процесс в некотором смысле согласуется с естественным для человека процессом принятия решений.

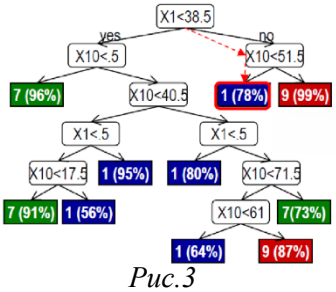

Рис.3
Визуализация построенного дерева с категориальным откликом

Модель используется в задачах регрессии и классификации, отклик может быть категориальным или непрерывным. Дерево решений – граф (древовидная структура), в котором внутренние узлы (условия на атрибуты) содержат в себе ограничение на множество признаков R. В листьях дерева лежат метки классов или значение целевой переменной, при этом области для листьев не содержат внутри других областей

Построение дерева в 2 фазы:

- *рост* – Изначально в корне находится вся выборка, которая в дальнейшнем разбивается на множество листьев по выбранным атрибутам и пороговым значениям.
- *отсечение* – На данном этапе выявляются и удаляются ветви, приводящих к переобучению или относящиеся к выбросам.

Отсечение дерева происходит за счёт:

- достижения целевой глубины
- нехватки числа объектов в узле относительно определенного значения
- уменьшения неопределённости отклика в узле относительно порога $\phi(t) \leq \varepsilon$
- недостаточного изменения неопределённости при разбиении узла $\Delta\phi(t) \leq \delta$

Функция неопределенности отклика $\phi(t)$ зависит от вероятностей классов для объектов, попавших в узел t. Зачастую её выражают с помощью:

1) Класификационной ошибки. Данная функция измеряет ожидаемое число ошибок при максимально вероятной классе. Пусть вероятность наиболее вероятного класса в узле - p_{max}, соотвественно $\phi(t) = 1 - p_{max}$.

2) Критерия Джини. Он измеряет вероятность ошибки при случайном угадывании класса. Пусть y – предсказание для точки, $\hat{y}$ – истинная классификация, а p_i - вероятность i-класса. Тогда вероятность ошибки при выборе класса равна

$$p(\hat{y} \neq y) = \sum_{i=1}^{k} p(y \neq \hat{y} | y = i) = \sum_{i=1}^{k} (1 - p_i) p_i.$$

Данная величина будет максимальной, когда классы равновероятны, и минимальной, когда все объекты принадлежат одному классу.

3) Энтропийного критерия. Он измеряет среднюю информацию, необходимую для предсказания класса объекта. Чем выше энтропия — тем более хаотично распределены классы. Функция выражается формулой

$$\phi(t) = -\sum_{i=1}^{k} p_i \log_2 p_i$$

Функция неопределенности может использоваться как и при росте, так и при отсечении дерева. Но важно, чтобы критерий на этих двух этапах был разный.

Отдельным плюсом решающего дерева является явная и легкая интерпретируемость модели. Decision tree также может использоваться в качестве *суррогатной* модели, когда на прогнозах более сложного и неинтерпретируемого алгоритма строится решающее дерево, с помощью которого легче проводить оценку согласованности и апроксимации исходной модели. На реальных откликах такое дерево не построить.

Также преимуществом данного метода машинного обучения является его скорость работы и нетребовательность к большим вычислительным мощностям, что позволяет его использовать в ансамблевых моделях. [10]

3.3.3 Random Forest

Одним из примеров ансамблевых моделей является Random Forest. Алгоритм работает по принципу Bagging (bootstrap aggregation) ансамбля.

Сначала генерируется M выборок с возвращением, на которых независимо обучается M решающих деревьев. Получение финального прогноза происходит за счёт агрегации предсказаний базовых моделей простым усреднением

$$a_{bag}(x) = \frac{1}{M}\sum_{i=1}^{M} b_i(x).$$

Ключевое преимущество бэггинга заключается в том, что обучающие выборки базовых моделей не полностью совпадают, что позволяет оценить качество всего ансамбля. Поскольку каждое дерево обучается на своей bootstrap-выборке, для каждого из них остаются примеры, не попавшие в тренировочный набор – OOB (out of bag) примеры, которые используются для оценки качества модели без необходимости выделять отдельную валидационную выборку.

Для каждой точки x считается OOB прогноз:

$$a_{OOB}(x) = \frac{1}{|\{i: x \in OOB_i\}|}\sum_{i: x \in OOB_i} b_i(x).$$

Далее вычисляется OOB ошибка, выражающаяся через усреднение по всем значениям функции ошибки истинного отклика от прогноза. Пусть L – наша функция ошибки, принимающая на вход истинную классификацию y_i и OOB прогноз $a_{OOB_i}(x_i)$, тогда OOB оценка для всего ансамбля на обучающей выборке Z*:*

$$OOB = \frac{1}{l} \sum_{i:x_i \epsilon Z} L\left(a_{OOB_i}(x_i), y_i\right)$$

Достаточное количество гиперпараметров (глубина и количество деревьев, число признаков, подающихся на вход) позволяют эффективно бороться с переобучением в Random Forest. [11, 12]

3.3.4 Stacking

Стекинг - метод ансамблирования, который агрегирует предсказания нескольких базовых моделей с помощью метамодели. В отличие от простого усреднения или голосования, стекинг обучает дополнительную модель (мета-алгоритм), которая учится делать финальное предсказание на основе выходов базовых алгоритмов. Например, в качестве базовых алгоритмов может быть выбран KNN и Решающее Дерево, а в качестве мета-алгоритма – линейная регрессия.

Чтобы избежать переобучения, на этапе обучения стекинга применяется кросс-валидация: предсказания базовых моделей получаются не на тех же данных, на которых они обучались, а на отложенных фолдах. Эти предсказания (так называемые мета-признаки) используются для обучения метамодели. Это позволяет эффективно использовать исходную выборку даже при небольшом её объёме.

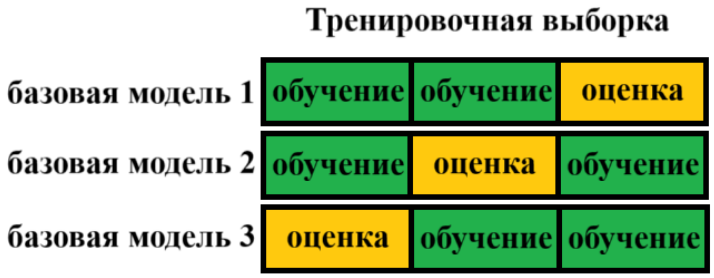

Рис.4
Пример распределения обучающей и проверочной выборки для случая из 3 базовых моделей при кросс-валидации

Кроме того, на вход агрегирующего алгоритма могут подаваться не только предсказания базовых моделей, но и признаки из исходных данных — это зависит от конкретной задачи и позволяет повысить точность предсказания. Стекинг особенно эффективен при небольшом объёме данных и разнообразии моделей.

3.3.5 Blending

Blending - метод ансамблирования, похожий на стекинг, но отличающийся способом формирования обучающих данных для метамодели. В блэндинге исходная обучающая выборка делится на две части:

- **Training set (70–80%)** — используется для обучения базовых моделей.
- **Holdout set (20–30%)** — на нём базовые модели делают предсказания, используемые как входные данные (мета-признаки) для метамодели.

Агрегирующий алгоритм обучается только на holdout-наборе, где предсказания базовых моделей были получены на данных, которые они ранее не видели. Это снижает риск переобучения метамодели.

В отличие от стекинга, где применяется кросс-валидация и используются предсказания со всех фолдов, blending проще в реализации, так как не требует циклического переобучения базовых моделей. Однако он менее эффективен при небольшом объёме данных, так как часть выборки (holdout) не используется при обучении базовых моделей. Поэтому blending чаще применяют, когда данных достаточно много.

Дополнительно, в blending можно подавать на вход метамодели не только предсказания базовых алгоритмов, но и исходные признаки. [13]

3.4 Распараллеливание вычислений

Сегментация облаков точек методами машинного обучения требует значительных вычислительных и временных ресурсов. При больших объёмах данных использование одного потока обработки может привести к существенным временным затратам. Распараллеливание вычислений позволяет уменьшить время выполнения программы и эффективно задействовать ресурсы современного оборудования.

Все последующие замеры эффективности распараллеливания вычислений будут проводить при следующих условиях вычисления:

Процессор: 12th Gen Intel(R) Core (TM) i5-1240P 1.70 GHz

Ядра: 12 (4 Performance-cores + 8 Efficient-cores)

Потоки: 16 (Hyper-Threading работает только на Performance-ядрах)

Оперативная память: 16 гб

Версия Python: 3.12.9

Версия NumPy: 2.2.2

Версия Joblib: 1.4.2

Версия Sklearn: 1.4.1

Исследования времени выполнения будет проводиться при работе с 9 laz-файлами. Пусть, для примера, количество соседей точки, необходимых для вычисления ковариационной матрицы равно 20 (далее мы найдём оптимальное для нашей задачи значение). Полученные результаты представлены в таблице ниже:

Имя laz-файла	Объем файла	Количество точек
1.laz	78.23 мб	6 382 773 шт
2.laz	127.32 мб	10 627 158 шт
3.laz	35.47 мб	2 611 685 шт
4.laz	19.73 мб	1 710 699 шт
5.laz	112.19 мб	9 707 837 шт
6.laz	39.63 мб	3 936 054 шт
7.laz	15.76 мб	1 322 666 шт
8.laz	67.93 мб	5 852 254 шт
9.laz	6.33 мб	479 034 шт

Таблица 1. Характеристики laz-файлов, используемых для исследования изменения времени

3.4.1 Встроенное распараллеливание от Numpy

Операции с матрицами (умножение, суммирование, транспонирование и т.п.), выполняемые с помощью библиотеки NumPy, подвергаются автоматическому распараллеливанию на уровне C/Fortran-библиотек. NumPy реализует распараллеливание операций, используя интерфейс BLAS (Basic Linear Algebra Subprograms) и LAPACK (Linear Algebra PACKage) через C.

Распараллеливание от NumPy можно отключить, явно указав это, изменив переменные окружения, в коде:

```
import os
os.environ["OMP_NUM_THREADS"] = "1"        # Для OpenMP
os.environ["MKL_NUM_THREADS"] = "1"        # Для Intel MKL
os.environ["NUMEXPR_NUM_THREADS"] = "1"    # Для NumExpr
os.environ["OPENBLAS_NUM_THREADS"] = "1"   # Для OpenBLAS
```

Для оценки эффективности параллелизма проведены замеры времени выполнения вычислений дополнительных признаков для каждой точки на одном файле (9.laz) при различном количестве потоков (1, 2, 4, 8, 12, 16).

Количество потоков	1	2	4	8	12	16
Время выполнения	7.65 сек	7.65 сек	8.04 сек	7.75 сек	7.79 сек	7.67 сек

Таблица 2. Сравнение времени работы NumPy в случае разного количества доступных потоков для 9.laz

Для оценки масштабируемости времени вычисления дополнительных признаков для точки от размера файла было проведено тестирование и замер времени выполнения на нескольких файлах, используя максимальное количество доступных потоков.

Файлы	**Однопоточная**	**Многопоточная**
1.laz	152.55 с	153.52 с
2.laz	303.12 с	304.29 с
3.laz	78.31 с	79.15 с
4.laz	61.55 с	62.44 с
5.laz	209.39 с	208.04 с
6.laz	123.18 с	124.23 с
7.laz	48.41 с	47.93 с
8.laz	157.54 с	155.04 с
9.laz	7.65 с	7.67 с

Таблица 3. Сравнение времени работы NumPy в однопоточном и многопоточном случае

Результаты тестирования показывают, что использование многопоточности в NumPy не дает значительного ускорения (разница менее 1%), а в некоторых случаях даже увеличивает время выполнения.

Это объясняется особенностями решаемой задачи: при вычислении дополнительных признаков для каждой точки мы работаем с малыми ковариационными матрицами с 3 исходными признаками (координаты точки), построенными на основе 20 соседних точек.

Из малости размера матриц операции не получают существенного выигрыша от параллелизма из-за высоких накладных расходов на организацию многопоточности - время, затрачиваемое на распределение задач между потоками и сбор результатов, становится сопоставимым или превышающим время самих вычислений.

Частый доступ к памяти при обработке множества небольших матриц создаёт узкое место, что сводит на нет преимущества параллельных вычислений. В итоге встроенный параллелизм NumPy не эффективен в контексте нашей задачи, где преобладают многочисленные мелкие матричные операции над небольшими наборами данных.

3.4.2 Распараллеливание с помощью joblib

Библиотека Joblib позволяет распараллеливать и ускорять выполнение операций. Есть возможность использовать потоки или процессы в зависимости от задачи. С минимальными усилиями и изменениями в коде можно добиться большого прироста в ускорении благодаря данной библиотеке.

Для распараллеливания вычисления дополнительных признаков достаточно добавить следующий фрагмент кода:

```
from joblib import Parallel, delayed

…

features = Parallel(n_jobs=-1)(delayed(compute_features_for_point)(i, points, indices) for i in range(len(points)))
```

В данном фрагменте кода с помощью конструкции `Parallel(n_jobs=-1)` выполняется распараллеливание вычислений на все доступные ядра процессора. Функция `compute_features_for_point`, принимающая на вход номер точки `i`, массив точек `points` и номера точек соседей `indices` и обернутая в `delayed`, вызывается для каждой точки из массива `points`, что позволяет независимо и одновременно вычислить признаки `features`. Такой подход значительно ускоряет обработку больших объемов данных, требующих малые вычисления.

Для демонстрации масштабируемости библиотеки Joblib относительно количество задействованных потоков было проведено сравнение времени вычисления дополнительных признаков с применением связки NumPy + Joblib, варьируя количество доступных потоков от 1 до 16.

Количество потоков	1	2	4	8	12	16
Время выполнения	4.87 сек	2.41 сек	1.98 сек	2.73 сек	6.72 сек	2.72 сек

Таблица 4. Сравнение времени работы joblib в случае разного количества доступных потоков для 9.laz

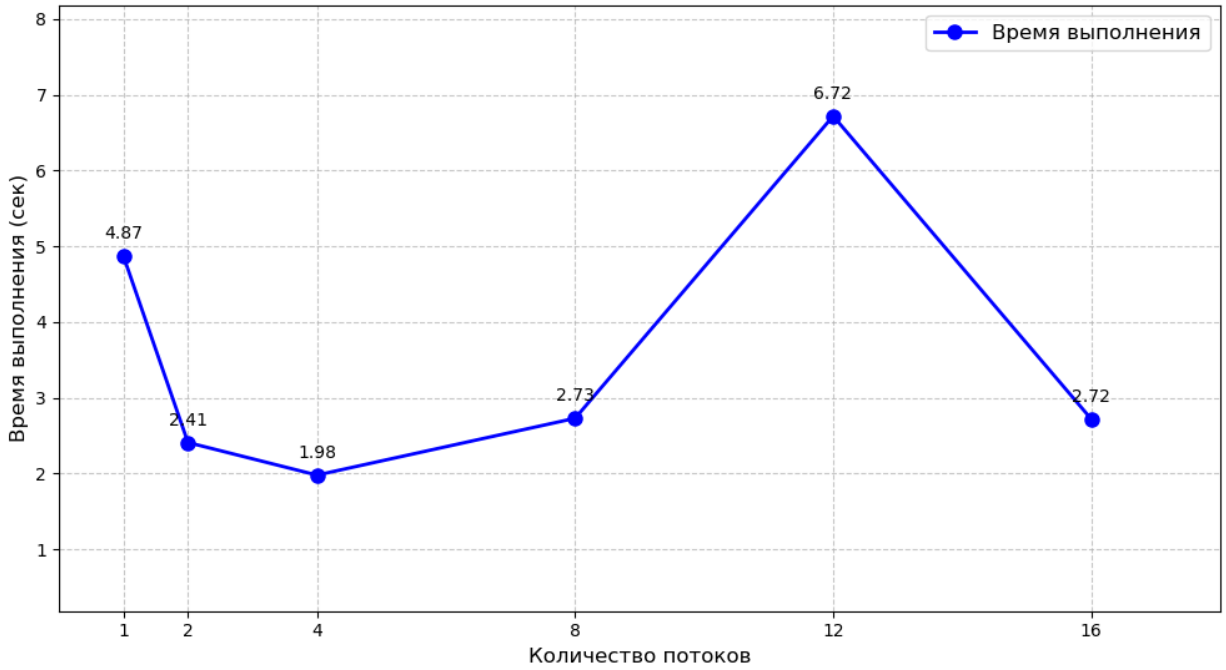

График 1. Зависимость времени выполнения от количества потоков (Joblib). Файл: 9.laz

Результаты исследования показывают, что оптимальное количество потоков для распараллеливания составляет 4.

Для оценки эффективности библиотеки Joblib проведено сравнение времени расчёта дополнительных признаков между реализацией на чистом NumPy и комбинацией NumPy + Joblib, установив количество доступных потоков равным 4, как самое оптимальное число потоков в контексте задачи.

Файлы	**NumPy**	**NumPy + Joblib**
1.laz	152.55 с	44.39 с
2.laz	303.12 с	75.08 с
3.laz	78.31 с	16.75 с
4.laz	61.55 с	12.37 с
5.laz	209.39 с	71.18 с
6.laz	123.18 с	31.14 с
7.laz	48.41 с	9.94 с
8.laz	157.54 с	38.84 с
9.laz	4.87 с	1.98 с

Таблица 5. Сравнение времени работы без и с использование Jobllib (кол-во потоков=4) на всех файлах

Анализ результатов показывает значительное снижение времени вычислений при использовании связки NumPy + Joblib по сравнению с одним только NumPy. В среднем ускорение составляет в 3–4 раза, это подтверждает эффективность параллельного вычисления с помощью joblib в контексте нашей задачи.

3.4.3 Встроенное распараллеливание от Sklearn

В библиотеке Scikit-learn многие алгоритмы машинного обучения поддерживают встроенное распараллеливание через параметр n_jobs. Это позволяет задействовать несколько ядер CPU для ускорения обучения и предсказания без дополнительных настроек.

В контексте нашей задачи ансамблевый метод RandomForestClassifier и метрический алгоритм KNeighborsClassifier поддерживают данный параметр, что позволяет распараллелить вычисления и уменьшить время выполнения.

Для поиска оптимального значения n_jobs было проведено исследование, при котором варьировалось значение параметра от 1 до 16 и замерялось время обучения и предсказания ансамблевого метода RandomForestClassifier на файле 9.laz при 50 решающих деревьях в ансамбле.

Параметр n_jobs	1	2	4	8	12	16
Время обучения	375.34 сек	214.86 сек	126.32 сек	71.98 сек	49.93 сек	40.18 сек
Время предсказания	28.0306 сек	14.0783 сек	9.0627 сек	4.2628 сек	3.8748 сек	3.3571 сек

Таблица 6.Сравнение времени обучения и предсказания в случае разного значения n_jobs для 9.laz

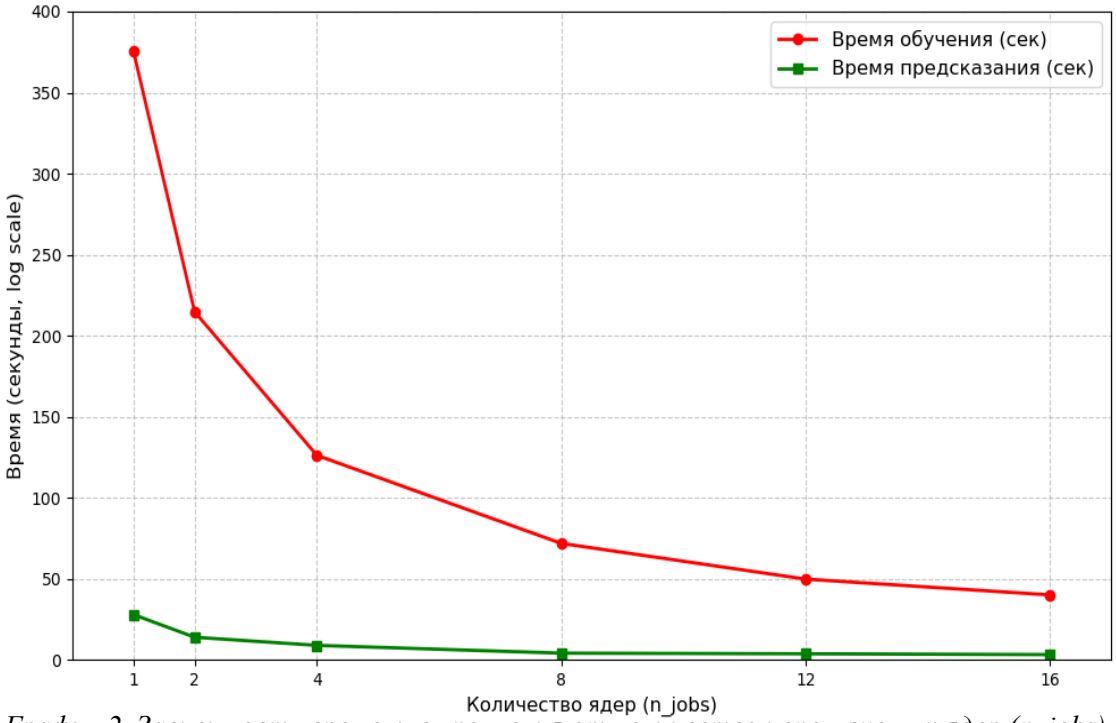

График 2. Зависимость времени выполнения от количество используемых ядер (n_jobs)

Исследование явно демонстрирует, что оптимальное значение параметра n_jobs – максимально возможное 16 потоков. Это обуславливается большим количество входных данных для работы алгоритмов.

Исследуем масштабируемость обучения и предсказания при 16 потоках.

Файлы	**Обучение**		**Предсказание**	
	n_jobs = 1	**n_jobs = 16**	**n_jobs = 1**	**n_jobs = 16**
1.laz	473.12 с	45.22 с	32.1263 с	3.1625 с
2.laz	887.02 с	96.17 с	60.2787 с	6.0004 с
3.laz	148.48 с	13.73 с	13.0629 с	1.2723 с
4.laz	97.22 с	8.82 с	8.0469 с	0.9470 с
5.laz	954.94 с	84.64 с	68.1408 с	7.0181 с
6.laz	238.59 с	23.79 с	15.0785 с	1.4925 с
7.laz	67.05 с	6.77 с	6.9782 с	0.7125 с
8.laz	382.20 с	37.86 с	23.6086 с	1.8632 с
9.laz	23.07 с	2.18 с	2.6496 с	0.1864 с

Таблица 7. Сравнение времени обучения и предсказания модели в случае однопоточного и многопоточного выполнения

Результаты были получены на прореженных файлах, количество точек в которых было уменьшено в 10 раз.

Использование параметра n_jobs=-1 в RandomForest обеспечивает значительное ускорение - время обучения и предсказания сокращается в 10-12 раз. Разница особенно заметна на больших файлах: например, на 5.laz время обучения снизилось с 955 до 85 секунд. Ускорение имеет линейную зависимость от объема данных: выигрыш в производительности становится более значительным с увеличением количества точек.

3.4.4 Сравнение методов распараллеливания

Для оценки эффективности различных подходов к распараллеливанию выполнения программы в задаче вычисления дополнительных признаков и обучения/предсказания модели были проведены сравнительные эксперименты. Результаты тестирования на реальных данных позволили выявить оптимальные сценарии применения каждого подхода, а также определить их влияние на общую производительность вычислительного конвейера. В сравнительной таблице (представлена ниже) в качестве критериев используются следующие метрики:

- время выполнения
- простота интеграции в существующий код
- масштабируемость при увеличении объёма данных.
- гибкость настройки

Метод	Ускорение	Простота использования	Масштабируемость	Гибкость
NumPy	На 0-1 %	Автоматически	Только для больших данных	Средняя
Joblib	В 2-4 раза	Легкая интеграция	Линейная	Универсально
Sklearn	В 10-12 раз	Настройка при инициализации	Линейная	Низкая

Таблица 8. Сравнительная характеристика методов распараллеливания

NumPy обеспечивает автоматическое распараллеливание на уровне линейной алгебры, но его эффективность ограничена типом операций и размером матриц. Подходит для простых случаев без необходимости внесения изменений в код.

Joblib является наиболее универсальным решением, обеспечивает значительное ускорение и легко встраивается в программу, особенно для независимых операций.

Sklearn показывает наибольшее ускорение при работе с алгоритмами машинного обучения. Однако его область применения ограничена встроенными методами библиотеки и не распространяется на произвольные вычисления.

Таким образом, в зависимости от характера задачи и требуемого уровня оптимизации, целесообразно комбинировать различные подходы к распараллеливанию.

Использование NumPy – позволяет «на старте» выиграть от встроенных функций распараллеливания базовых линейных операций.

Joblib - оптимален для независимых пользовательских вычислений независимо от объема данных.

Параметр n_jobs в Sklearn - эффективен при обучении и предсказании с использованием готовых моделей. Совместное применение этих инструментов позволяет построить гибкий и производительный вычислительный конвейер, обрабатывать большие объёмы данных с минимальными временными затратами и ресурсами.

3.5 Метрики оценки моделей

Оценка моделей машинного обучения играет ключевую роль в выборе наилучшего алгоритма для задачи сегментирования облаков точек. Критерии помогают определить эффективность и корректно сравнить различные подходы к бинарной классификации точек.

Изначально все предсказанные объекты разбиваются на 4 группы:

- верно-положительные (TP) — объект в действительности принадлежит предсказанному положительному классу.
- верно-отрицательные (TN) — объект в действительности принадлежит предсказанному отрицательному классу.
- ложно-положительные (FP) — объект в действительности не принадлежит предсказанному положительному классу.
- ложно-отрицательный (FN) — объект в действительности не принадлежит предсказанному отрицательному классу.

Благодаря такому разбиению на группы мы можем непосредственно посчитать метрики качества.

3.5.1 Accuracy

Достоверность (Accuracy) показывает долю правильно классифицированных объектов от общего числа примеров:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Метрика идеально подходит для случаев, когда классы сбалансированы. Например, когда перед нами стоит задача классифицировать столовый прибор перед нами на вилку или ложку.

В задачах с неравными классами данная метрика бесполезна. Например, пусть из 1000 точек LiDAR скана 900 принадлежат классу (положительный класс) кроны, а все остальные (100) – листве (отрицательный класс).

Предположим, что из 900 точек кроны наш классификатор определил верно только 800 (TP = 800, FP = 100). Из 100 точек листвы он верно определил только 70 (TN = 70, FN = 30). Accuracy = 0,87.

Если мы классифицируем все точки как крону, то Accuracy = 0,9. Значение метрики лучше, хотя, при этом, вторая модель совершенно не обладает никакой предсказательной силой.

3.5.2 Precision

Точность (precision) демонстрирует долю действительно положительных объектов из всех предсказанных положительных:

$$\frac{TP}{TP\ +\ FP}$$

Чем меньше ложноположительных срабатываний допускает модель, тем больше будет её точность. Метрика объективно оценивает качество модели, поскольку не подвержена влиянию дисбаланса классов.

3.5.3 Recall

Полнота (recall) отражает долю правильно найденных положительных объектов среди всех объектов положительного класса:

$$\frac{TP}{TP\ +\ FN}$$

Чем меньше ложно отрицательных срабатываний, тем выше полнота модели. Как и точность, метрика не подвержена влиянию дисбаланса классов.

3.5.4 F1-score

Метрики полноты и точности позволяют удобно оценить модель. F1-score позволяет скомпоновать пару precision-recall в одну метрику:

$$\frac{2}{\frac{1}{precision}+\frac{1}{recall}} = \frac{2 * TP}{TP\ +\frac{(FN+FP)}{2}}$$

Использование среднего гармонического позволяет F1-score достичь максимума при полноте и точности, равными единице, и резко снизиться, когда один из аргументов близок к нулю.

4 Результаты исследования и их анализ

В данной главе представлены результаты исследования, направленного на сравнительный анализ методов сегментации облаков точек LiDAR-сканирования деревьев.

На основе проведённых экспериментов оцениваются качество классификации различных алгоритмов машинного обучения: K-Nearest Neighbors (KNN), Decision Tree, Random Forest, а также ансамблевых методов Stacking и Blending.

Метрика Accuracy не будет использована при оценки качества модели, так как классы точек у laz-файлов не сбалансированны. Для каждой модели были рассчитаны основные метрики качества (Precision, Recall, F1-score), а также проведена визуализация полученных результатов.

4.1 Сравнительный анализ примененных методов

В рамках исследования были протестированы пять различных моделей машинного обучения: K-Nearest Neighbors (KNN), Decision Tree, Random Forest, Stacking и Blending. Каждая модель обучалась на наборе данных с заранее вычисленными локальными признаками точек.

Исходные данные разделяются на тестовую (30%) и обучающую (70%) выборки случайным образом (с использованием random_state=42 для воспроизводимости), при этом пропорции классов сохраняются благодаря параметру stratify=labels.

Выборка для лучшей обучающей и предсказательной способности была предварительно обработана с использованием методов масштабирования: для координат точек (X, Y, Z) применялся MinMaxScaler, приводящий значения к диапазону [0, 1], что важно для сохранения пространственной структуры данных, а для дополнительных признаков использовался StandardScaler, стандартизирующий распределение с математическим ожиданием 0 и дисперсией 1. Это необходимо, так как координаты требуют сохранения относительных расстояний, в то время как остальные характеристики нуждались в приведении к единому масштабу для устойчивой работы алгоритмов

Все вычислительные эксперименты, включая обучение моделей, подбор гиперпараметров и оценку качества, проводились на сервере с двухсокетной системой Intel Xeon Platinum 8168 (48 физических ядер, 96 логических потоков). Это обеспечило высокую производительность и эффективность при обработке больших объёмов данных.

Перед самими экспериментами для правильной оценки качества моделей было найдено подходящее количество соседей точки, используемое при подсчёте ковариационной матрицы и дополнительных признаков. Данное число было найдено варьированием n_neighbors от 10 до 70 с шагом 5. Автоматические значения гиперпараметров библиотеки Scikit-learn были выбраны в качестве параметров алгоритмов машинного обучения Decision Tree и KNN, используемых для сегментации. Были получены следующие результаты тестирования:

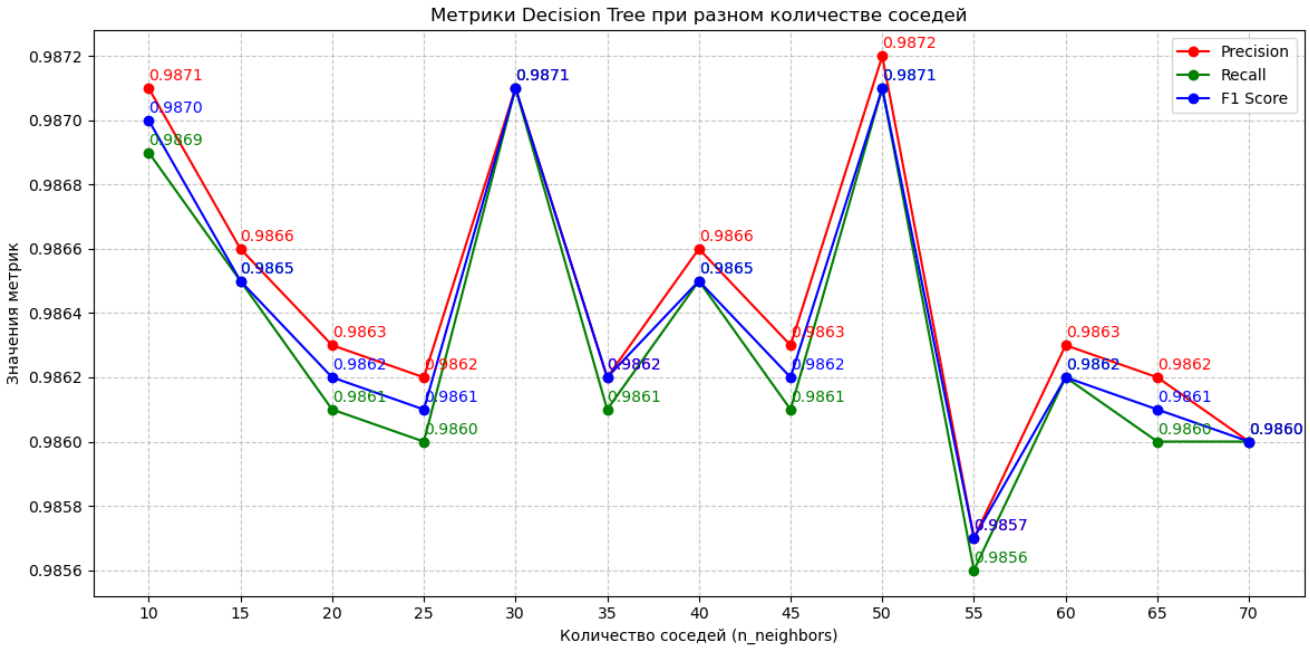

График 4. Зависимость метрик качества Решающего Дерева, полученных на файле 9.laz, от количества используемых соседей при подсчёте дополнительных признаков

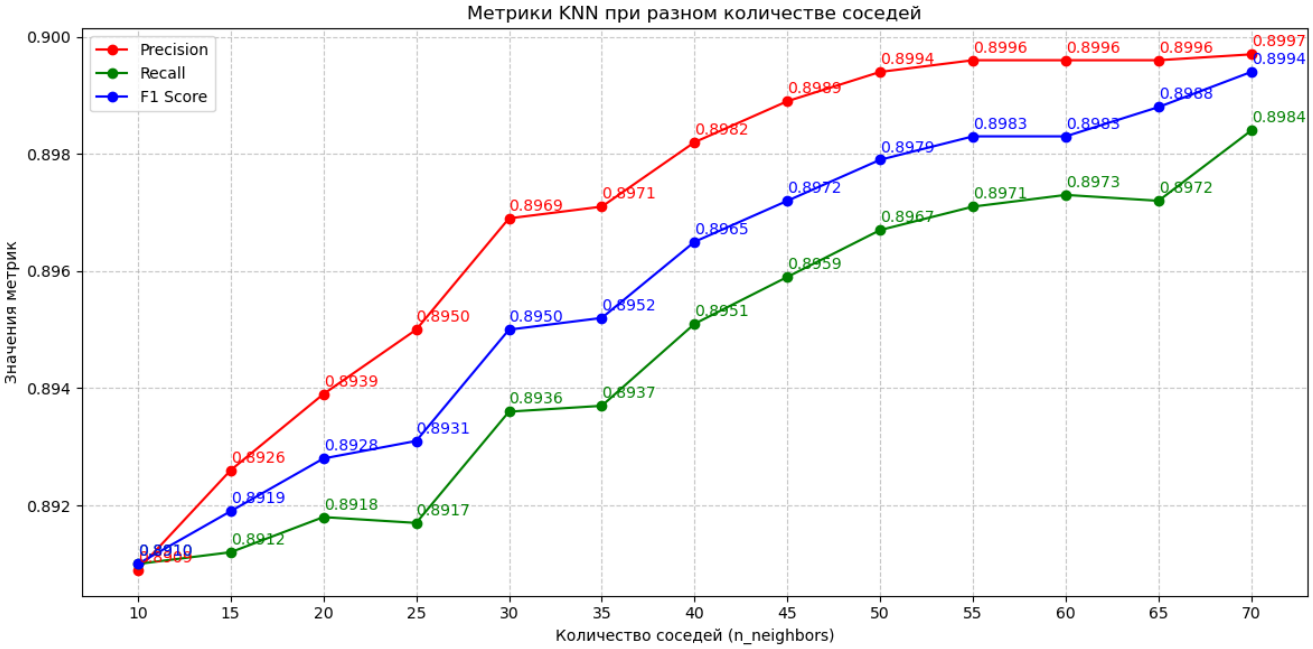

График 3. Зависимость метрик качества KNN, полученных на файле 9.laz, от количества используемых соседей при подсчёте дополнительных признаков

Исследование показывает, что оптимальное количество соседей, используемых при подсчете дополнительных признаков равно 50.

1. K-Nearest Neighbors (KNN)

Метод KNN имеет единственный гиперпараметр – количество ближайших соседей. Наилучшее значение данной переменной заранее определить нельзя. Были проведены тесты на файле 9.laz с варьированием числа ближайших соседей от 2 до 50 с шагом 2. Выбор данного интервала связан с высокой плотностью точек в облаке и важностью локальной структуры дерева (облака точек).

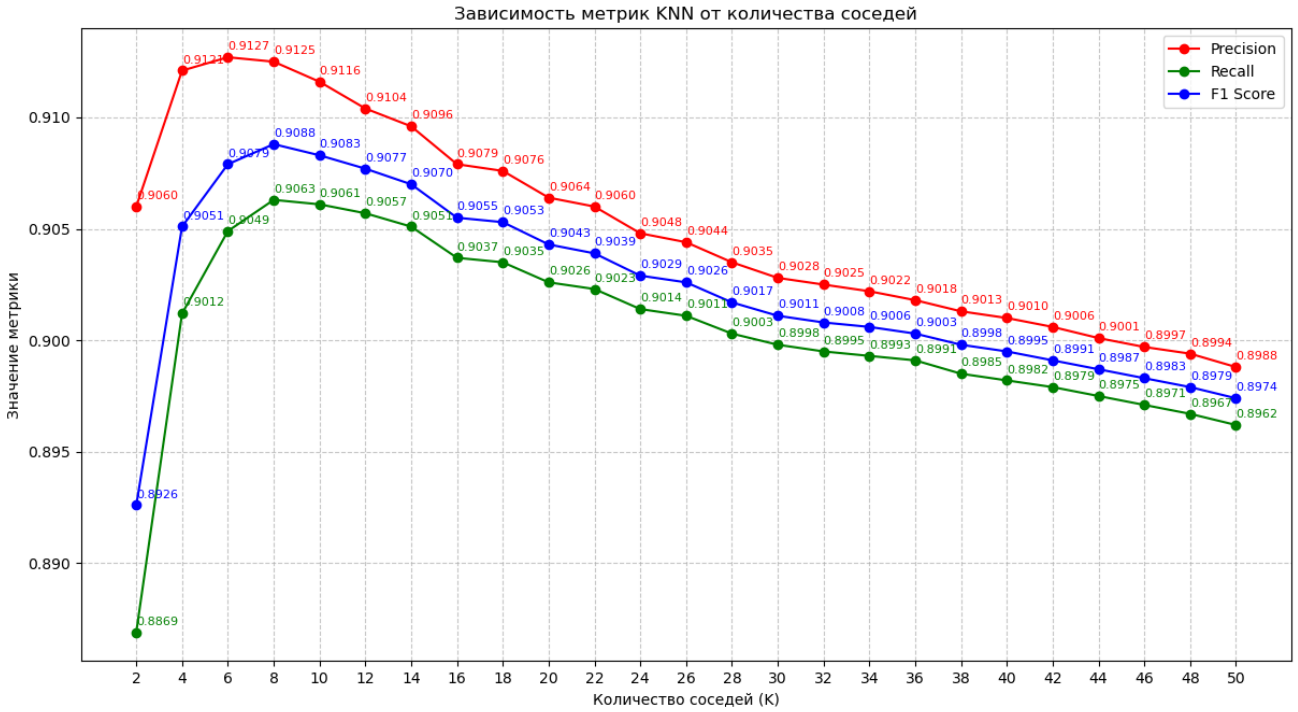

График 5. Зависимость значений метрик качества от количества соседей

Исследование показывает, что наибольшие значения метрик качества (Precision, Recall, F1-score) достигаются при значении гиперпараметра n_neighbors равном 8.

Используя данное значение для количества соседей в методе машинного обучения KNN были получены следующие метрики качества для всех файлов:

Файлы	Точность			Время предсказания (30 % в train)
	Precision	Recall	F1-score	
1.laz	0.8412	0.8182	0.8286	171.4649 сек.
2.laz	0.8440	0.8126	0.8273	352.8468 сек.
3.laz	0.8971	0.8843	0.8903	67.8651 сек.
4.laz	0.9181	0.9049	0.9108	38.9190 сек.
5.laz	0.9287	0.8940	0.9102	303.9809 сек.
6.laz	0.9316	0.9227	0.9269	96.9415 сек.
7.laz	0.9203	0.9071	0.9131	28.2298 сек.
8.laz	0.9504	0.9362	0.9428	152.7610 сек.
9.laz	0.9125	0.9063	0.9088	6.2093 сек.

Таблица 9. Результаты метрик качества и времени алгоритма KNN на файлах в датасете

2. Решающее дерево (Decision Tree)

В методе решения задачи с помощью решающего дерева мы будем использовать критерий Джини для оценки качества разделения классов, поскольку он позволяет более эффективно (по сравнению с классификационной ошибкой) и менее вычислительно затратно (по сравнению с энтропийный критерием) отслеживать ошибку. Были получены следующие результаты:

Файлы	Точность			Время	
	Precision	Recall	F1-score	Обучения	Предсказания
1.laz	0.9825	0.9823	0.9824	103.8032 сек.	0.4371 сек.
2.laz	0.9802	0.9806	0.9804	180.9451 сек.	5.0195 сек.
3.laz	0.9872	0.9873	0.9872	36.6573 сек.	0.1253 сек.
4.laz	0.9773	0.9775	0.9774	23.5695 сек.	0.0847 сек.
5.laz	0.9878	0.9881	0.9879	146.4735 сек.	0.4414 сек.
6.laz	0.9868	0.9870	0.9869	50.6510 сек.	0.1826 сек.
7.laz	0.9866	0.9868	0.9867	14.9511 сек.	0.0543 сек.
8.laz	0.9900	0.9899	0.9900	87.4916 сек.	0.2618 сек.
9.laz	0.9868	0.9867	0.9867	4.7439 сек.	0.0186сек.

Таблица 10. Результаты метрик качества и времени алгоритма Решающее Дерево на файлах в датасете

3. Случайный лес (Random Forest)

Ансамблевый метод Случайный Лес позволяет выбрать количество базовых моделей в своём алгоритме. Это число играет важную роль в предсказании, качестве и в оценке времени обучения. Подходящее количество базовых моделей нельзя вычислить заранее. Были проведены тесты на файле 9.laz с варьированием числа базовых моделей в ансамбле от 10 до 100 с шагом 5 и получены результаты:

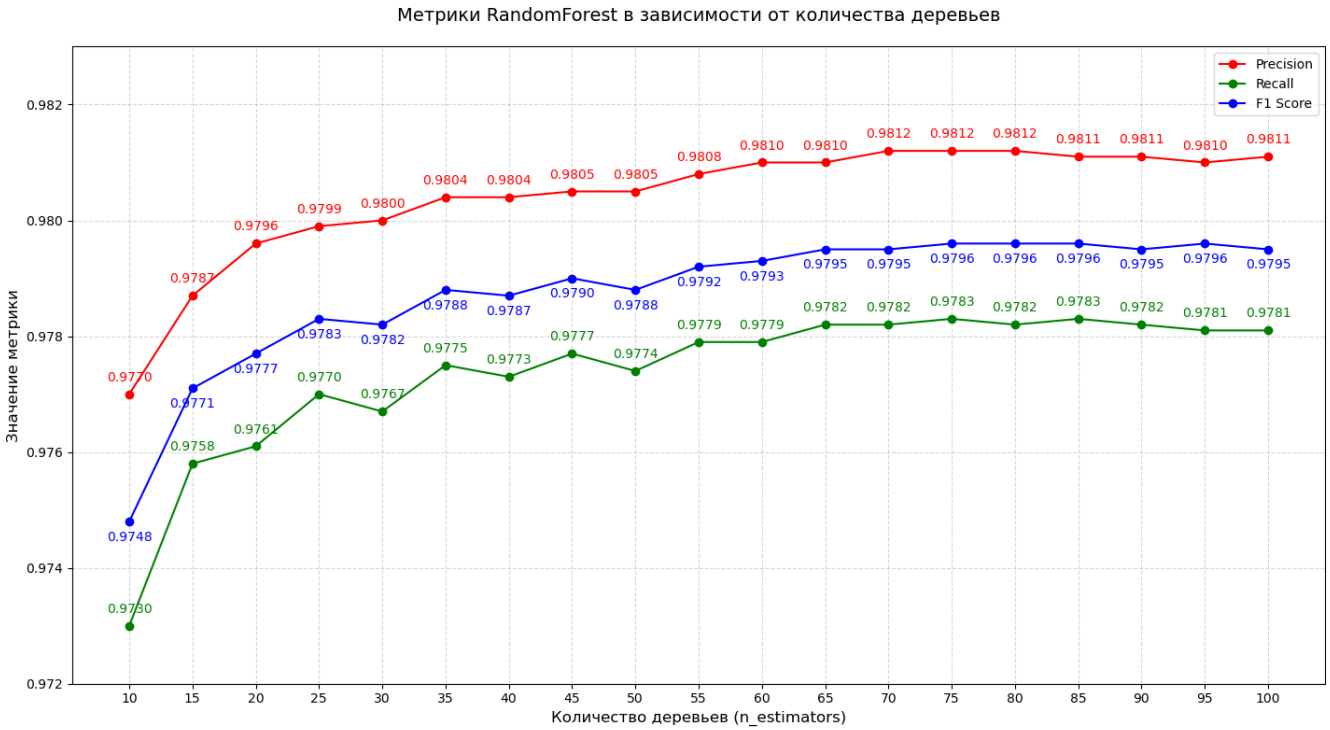

График 6. Зависимость метрик качества, полученных на файле 9.laz, от количества базовых моделей в ансамбле

Исследование показывает, что наибольшие значения метрик качества (Precision, Recall, F1-score) достигаются при значении гиперпараметра n_estimators равном 75.

Используя данное значение в качестве количества базовых моделей в ансамбле, были получены следующие метрики качества и замеры времени на обучение и предсказание для всех файлов в датасете.

Файлы	Точность			Время	
	Precision	Recall	F1-score	Обучения	Предсказания
1.laz	0.9745	0.9581	0.9659	314.8111 сек.	10.5086 сек.
2.laz	0.9762	0.9498	0.9625	544.6821 сек.	14.4280 сек.
3.laz	0.9816	0.9729	0.9771	118.3681 сек.	2.4953 сек.
4.laz	0.9701	0.9603	0.9649	77.5773 сек.	1.5944 сек.
5.laz	0.9839	0.9689	0.9762	476.1727 сек.	10.4776 сек.
6.laz	0.9794	0.9695	0.9743	181.4611 сек.	3.8621сек.
7.laz	0.9810	0.9731	0.9768	50.5387 сек.	1.0809 сек.
8.laz	0.9859	0.9755	0.9804	286.8266 сек.	5.9447 сек.
9.laz	0.9812	0.9783	0.9796	14.2010 сек.	0.3686 сек.

Таблица 11. Результаты метрик качества и времени алгоритма Случайный Лес на файлах в датасете

4. Stacking

В качестве базовых моделей в стэкинге были выбраны алгоритм Решающее Дерево и KNN. В качестве функции, агрегирующей результаты предсказаний, были протестированы два варианта: метод Логистической Регрессии с параметром максимального количества итераций max_iter = 1000 для лучшей сходимости, а также метод Градиентного Бустига с параметром learning_rate=0.1 для оценки их эффективности. Параметр n_neighbors в методе KNN выбран равным 48, исходя из выше проведенных исследований.

Метрики	Precision	Recall	F1-score	Время обучения	Время предсказания
Логистическая Регрессия	0.9868	0.9867	0.9867	20.1907 сек.	5.3091 сек.
Градиентный Бустинг	0.9868	0.9867	0.9867	25.2147 сек.	5.1652 сек.

Таблица 12. Сравнение метрик качества и времени агрегирующих алгоритмов (Логистическая Регрессия и Градиентный Бустинг) в методе Stacking

В результате тестирования в качестве агрегирующего алгоритма для ансамблевого метода Stacking был выбран Градиентный бустинг, так как он показал лучшие результаты по времени обучения и предсказания при равных значениях метрик качества. Были получены значения основных метрик (точности, полноты и F1-меры), а также временные затраты на обучение и предсказание для всех файлов в датасете, с использованием выбранных базовых и агрегирующей модели.

Файлы	**Точность**			**Время**	
	Precision	**Recall**	**F1-score**	**Обучения**	**Предсказания**
1.laz	0.9825	0.9823	0.9824	296.4775 сек.	163.6777 сек.
2.laz	0.9802	0.9806	0.9804	489.9349 сек.	332.4043 сек.
3.laz	0.9872	0.9873	0.9872	105.0167 сек.	60.9537 сек.
4.laz	0.9773	0.9775	0.9774	68.1715 сек.	33.8397 сек.
5.laz	0.9878	0.9881	0.9879	438.9355 сек.	290.7244 сек.
6.laz	0.9868	0.9870	0.9869	143.1728 сек.	87.6637 сек.
7.laz	0.9866	0.9868	0.9867	43.9828 сек.	23.5752 сек.
8.laz	0.9900	0.9899	0.9900	251.1161 сек.	137.7955 сек.
9.laz	0.9868	0.9867	0.9867	15.0649 сек.	3.9731 сек.

Таблица 13.Результаты метрик качества и времени алгоритма Stacking (базовые модели: KNN и Решающее Дерево; Агрегирующий алгоритм: Логистическая Регрессия) на файлах в датасете

5. Blending

В качестве базовых моделей в блэндинге были выбраны алгоритм Решающее Дерево, Логистическая регрессия и KNN (n_neighbors=48). Были получены результаты метрик качества и временных затрат на обучение и предсказание для всех файлов в датасете.

Файлы	Точность			Время	
	Precision	Recall	F1-score	Обучения	Предсказания
1.laz	0.9617	0.9108	0.9327	106.4864 сек.	163.4758 сек.
2.laz	0.9708	0.8506	0.8994	185.3918 сек.	338.3534 сек.
3.laz	0.9350	0.9330	0.9470	37.7827 сек.	59.6611 сек.
4.laz	0.9587	0.9365	0.9459	24.5986 сек.	33.5513 сек.
5.laz	0.9801	0.9108	0.9415	150.0246 сек.	297.2092 сек.
6.laz	0.9736	0.9521	0.9619	54.7636 сек.	84.5116 сек.
7.laz	0.9662	0.9411	0.9519	15.2558 сек.	24.8102 сек.
8.laz	0.9774	0.9570	0.9663	90.1313 сек.	135.4850 сек.
9.laz	0.9661	0.9628	0.9643	5.1693 сек.	4.6335 сек.

Таблица 14.Результаты метрик качества и времени алгоритма Blending (базовые модели: KNN, Решающее Дерево, Логистическая Регрессия) на файлах в датасете

4.2 Оценка точности сегментации

Наивысшие значения метрик качества были получены с помощь алгоритма Решающее Дерево. Достойные значения качества на файлах продемонстрировал Stacking, но время его обучения и предсказания на больших файлах на порядок выше, чем у метода Решающее дерево.

Лучше всего по соотношению "качество — время работы" показал себя алгоритм Решающее дерево. Он дает хороший прогноз, при этом затратив на предсказание минимальное время.

4.3 Визуализация результатов

Визуализация результатов позволяет наглядно оценить качество работы алгоритма сегментации.

Сравнение проводится между исходными данными с правильной классификацией точек из облака и предсказаниями, полученными с помощью алгоритма Решающее Дерево.

Построение визуализаций было сделано с помощью библиотеки Matplotlib. На одном графике отображаются два варианта сегментации: эталонный и предсказанный. Это позволяет сразу увидеть, где алгоритм справился хорошо, а где допустил ошибки — например, если часть кроны дерева ошибочно отнесена к листве. [14]

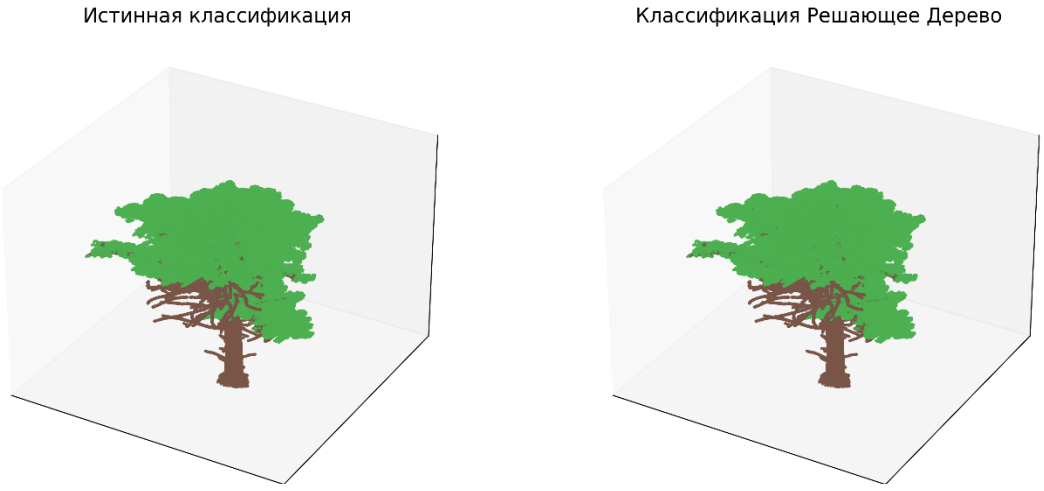

Рисунок 5. Визуализация истинной и предсказанной сегментации файла 1.laz

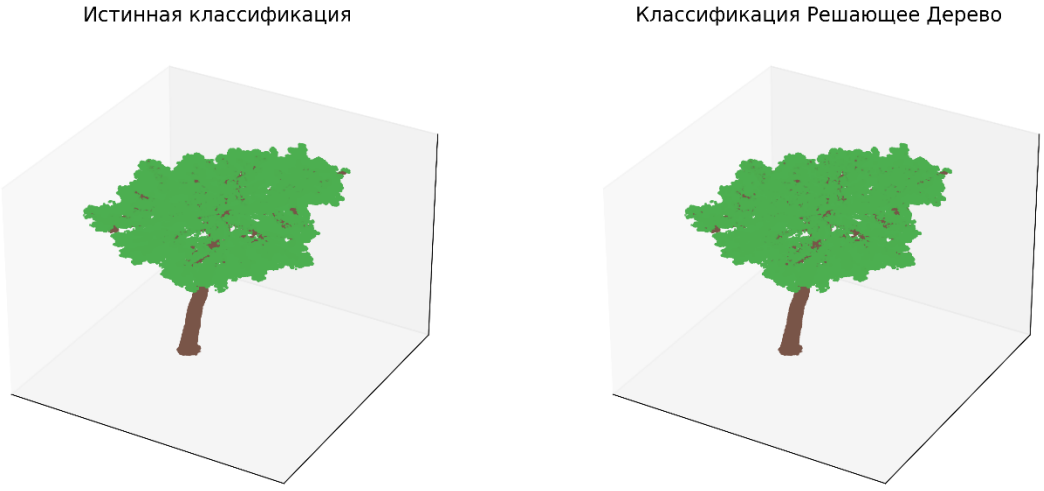

Рисунок 6. Визуализация истинной и предсказанной сегментации файла 2.laz

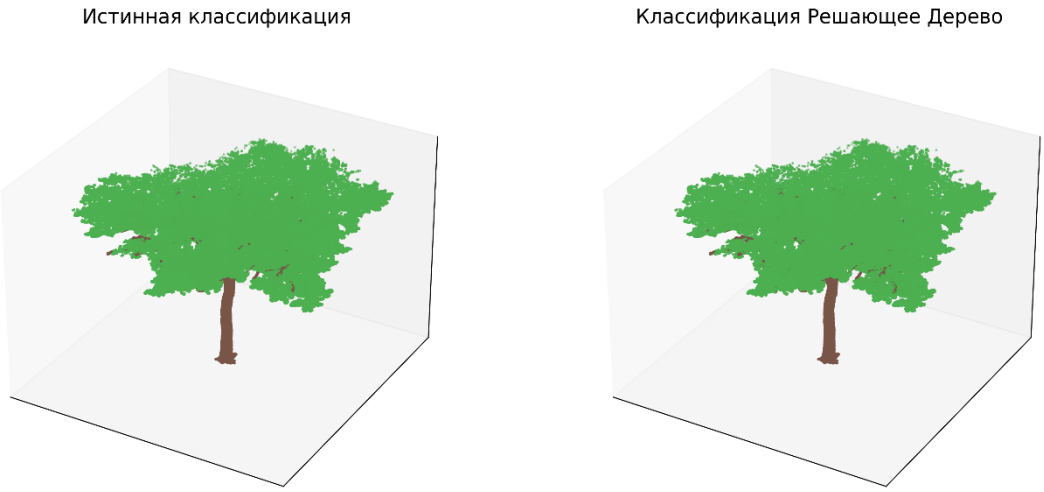

Рисунок 12. Визуализация истинной и предсказанной сегментации файла 6.laz

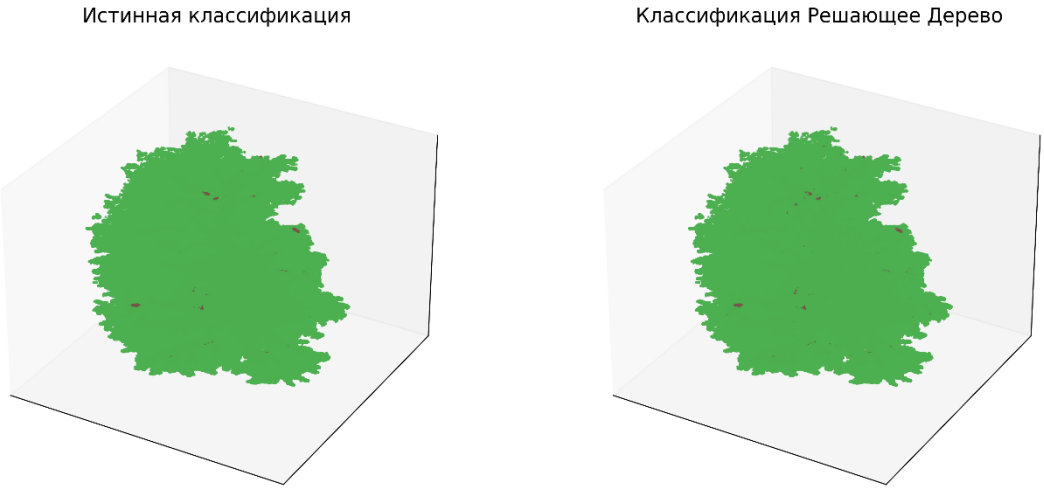

Рисунок 11. Визуализация истинной и предсказанной сегментации файла 7.laz

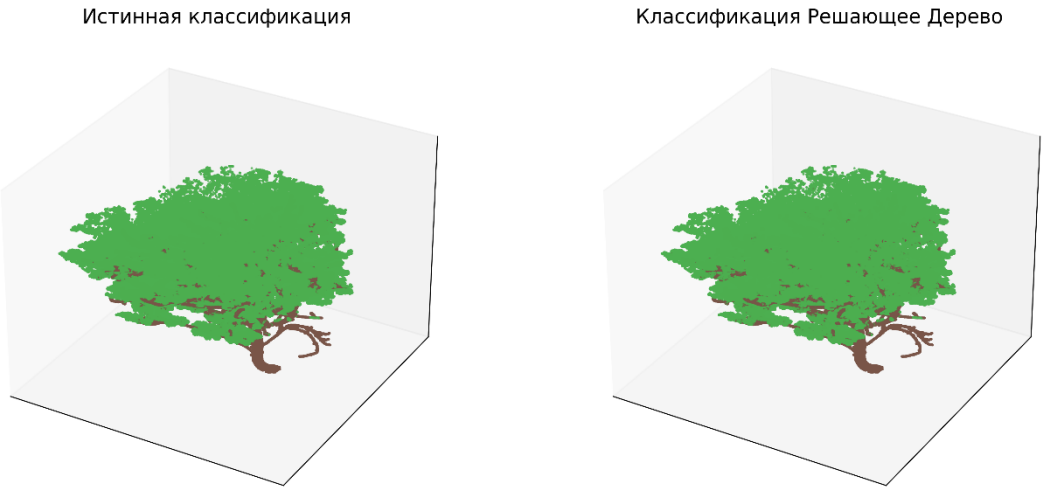

Рисунок 10. Визуализация истинной и предсказанной сегментации файла 8.laz

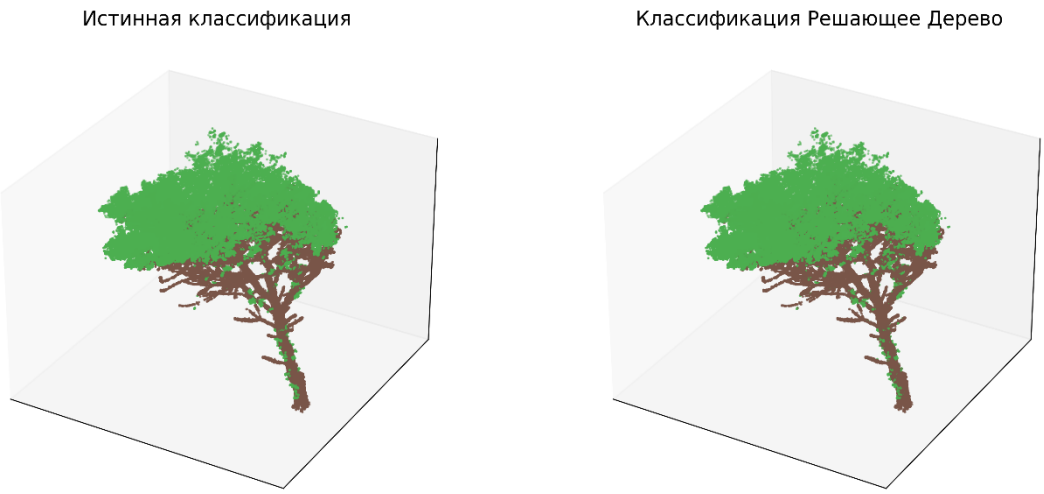

Рисунок 13. Визуализация истинной и предсказанной сегментации файла 9.laz

5 Выводы

В ходе работы были изучены методы машинного обучения (KNN, Решающее Дерево, Случайный Лес, Stacking и Blending), метрики оценки качества (Accuracy, Precision, Recall, F1-score) и способы работы с Laz и Las файлами.

В задаче сегментации облаков точек деревьев, полученных лазерным сканированием, на крону и листву неприемлемо использовать метрику Accuracy, так как присутствует большой дисбаланс классов.

Метрики качеств всех моделей имеют близкие к нулю значения при использовании дополнительных признаков, вычислимых с помощью ковариационной матрицы k соседей, для каждой точки. В качестве дополнительных признаков используются: линейный индекс ($L_\lambda = \frac{\lambda_2 - \lambda_1}{\lambda_2}$), планарный индекс ($P_\lambda = \frac{\lambda_1 - \lambda_0}{\lambda_2}$), рассеянный индекс ($S_\lambda = \frac{\lambda_0}{\lambda_2}$),

кривизна ($C_\lambda = \frac{\lambda_0}{\lambda_0+\lambda_1+\lambda_2}$) и отношение собственных значений ($R_\lambda = \frac{\lambda_1}{\lambda_2}$) Где $\lambda_0 \leq \lambda_1 \leq \lambda_2$ – собственные значения ковариационной матрицы.

Автоматическое распараллеливание матричных операций от NumPy приносит малое ускорение (на 0-1%) в контексте данной задачи по сравнению с однопоточной версией.

Распараллеливание вычислений дополнительных признаков с помощью библиотеки Joblib вносит большой вклад в эффективность программы. Эмпирическим путём было выведено оптимальное число потоков для распараллеливания равное 4 на процессоре 12th Gen Intel(R) Core (TM) i5-1240P 1.70 GHz с оперативной памятью 16 гб. Достигается ускорение в 4 раза.

С помощью распараллеливания обучения и предсказания ансамблевого метода RandomForestClassifier и метрического алгоритма KNeighborsClassifier достигается ускорение в 10-12 раз на процессоре 12th Gen Intel(R) Core (TM) i5-1240P 1.70 GHz с оперативной памятью 16 гб. Ограничения на количество используемых потоков нет, значение гиперпараметра n_jobs устанавливается равное -1.

Исследовательским способом было выведено, что наилучшим агрегирующим алгоритмом в Stacking-е является Логистическая Регрессия по сравнению с Градиентным Бустингом.

Варьированием значений найдено оптимальное количество соседей для подсчёта ковариационной матрицы, используемой при вычислении дополнительных признаков точек, равное 50.

Варьированием значений найдено оптимальное количество соседей для классифицирования точек методом KNN равное 8.

Варьированием значений найдено оптимальное количество базовых моделей в ансамбле деревьев для классифицирования точек методом Случайный Лес равное 75.

Наилучшим методом для данной задачи является Решающее Дерево с критерием Джини в качестве функции неопределенности. Обучение проводится на предварительно обработанной выборке данных. Координаты точек приводятся к интервалу [0, 1] с сохранением их геометрии с помощью MinMaxScaler из библиотеки sklearn.preprocessing, для дополнительных признаков использовался StandardScaler, стандартизирующий распределение с математическим ожиданием 0 и дисперсией 1, из этой же библиотеки.

Анализ результатов показывает, что алгоритм Решающее Дерево стабильно демонстрирует высокие значения метрик качества (Precision, Recall и F1-score выше 0.97) на всех файлах датасета. Время обучения значительно варьируется в зависимости от размера и сложности файла, однако время предсказания остаётся крайне малым (менее 0.5 секунд для всех файлов), что подтверждает высокую эффективность метода для практического применения.

Были достигнуты поставленные задачи, исследована эффективность методов параллельной сегментации облаков точек LiDAR-сканирования деревьев с применением алгоритмов машинного обучения для оптимизации скорости и качества обработки данных.

6 Основные результаты

1 Распараллеливание вычислений

Определены эффективные способы ускорения вычислений: Joblib даёт прирост в 4 раза, Sklearn (n_jobs = -1) — до 12 раз при обучении моделей.

2 Классификация точек облаков

С помощью вычислительных экспериментов выявлены наилучшие значения гипер-параметров для всех исследуемых моделей.

3 Сравнение параметров и оптимизация гиперпараметров

Проведено сравнение моделей с оптимальным значением параметров. Показано, что лучшая модель – Решающее дерево.

Заключение

Проведённое исследование позволило всесторонне оценить эффективность методов машинного обучения для задачи сегментации облаков точек LiDAR-сканирования деревьев. Были протестированы как классические алгоритмы (KNN, Решающее Дерево, Случайный Лес), так и ансамблевые подходы (Stacking и Blending), что обеспечило широкий спектр сравнительного анализа. Особое внимание уделялось вопросам распараллеливания вычислений для повышения скорости обработки больших объёмов данных. Эксперименты подтвердили, что оптимизация вычислений с помощью библиотеки Joblib и встроенного распараллеливания Scikit-learn обеспечивает значительное сокращение времени выполнения. Визуализация результатов продемонстрировала высокую наглядность и качество сегментации, что позволяет рекомендовать предложенные методы для решения практических задач. Итогом работы стало формирование набора рекомендаций по выбору эффективных подходов для сегментации и ускорения обработки облаков точек, что открывает перспективы для дальнейших исследований и внедрения в смежных областях.

Литература

1. Демидов В.Э. Применение воздушного лазерного сканирования для картирования рельефа, поиска следов антропогенного воздействия и изучения растительного покрова на территории Приокско-Террасного государственного природного биосферного заповедника // Труды Мордовского государственного природного заповедника им. П.Г. Смидовича. — 2021.
2. Чермошенцев А.Ю., Сухотин В.К. Обзор методов сегментации точек лазерных отражений, полученных по данным лазерного сканирования. — 2020.
3. Велижев А.Б., Шаповалов Р.В., Потапов Д., Третьяк Л., Конышин А. Автоматическая сегментация облаков точек на основе элементов поверхности. — 2009.
4. Сырых А.С., Медведев С.Н. Сравнительный анализ алгоритмов реконструкции поверхности из облака точек // Международный научно-исследовательский журнал. — 2023.
5. Matveev A., et al. Geometric Attention for Prediction of Differential Properties in 3D Point Clouds // Artificial Neural Networks in Pattern Recognition. IAPR Workshop. — Cham: Springer, 2020.
6. Anh Nguyen, Bac Le 3D Point Cloud Segmentation: A survey — 2021
7. Zhang W., et al. An easy-to-use airborne LiDAR data filtering method based on cloth simulation // Remote Sensing. — 2016.
8. Мокеев П.Е. Инфраструктура для сравнения алгоритмов сегментации плоскостей из облаков точек, полученных с лидаров. — 2022.

9. Коэльо Л.П., Ричарт В. Построение систем машинного обучения на языке Python. — Москва: ДМК Пресс, 2016.
10. Китов В.В. Онлайн-учебник по машинному и глубокому обучению [Электронный ресурс]. — Режим доступа: https://deepmachinelearning.ru/ (дата обращения: 2025-04-26).
11. Рашка С., Лю Ю., Мирджалили В. Машинное обучение с PyTorch и Scikit-Learn. — Москва: ДМК Пресс, 2022.
12. Дайзенрот М.П., Альдо Ф.А., Сунь О.Ч. Математика в машинном обучении. — Москва: ДМК Пресс, 2022.
13. Alberto M.E., Hannah W., Lukas W., Jose C.C., Bernhard H. Deep learning with simulated laser scanning data for 3D point cloud classification. — 2024.
14. Python. Визуализация данных: Matplotlib, Seaborn, Mayavi. — 2020.