ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет информатики, математики и компьютерных наук

**Программа подготовки бакалавров по направлению
Компьютерные науки и технологии**

Мухаметшин Руслан Рашидович

КУРСОВАЯ РАБОТА

Интерактивный конструктор моделей искусственного интеллекта с поддержкой мультимодальных задач.
Создание пайплайнов для реализации Classic ML

Научный руководитель
старший преподаватель НИУ
ВШЭ - НН

Саратовцев Артём Романович

Нижний Новгород, 2025г.

Структура работы

1. Введение

Современный этап развития машинного обучения характеризуется активным внедрением технологий в различные сферы человеческой деятельности: от медицины и финансов до промышленности и развлечений. Однако разработка и обучение ML моделей остаются сложными задачами, требующими глубоких знаний в программировании, математике и обработке данных. Это создаёт барьер для специалистов, не обладающих техническим бэкграундом, но заинтересованных в применении ML для решения своих задач.

В этом контексте особую актуальность приобретают No-code платформы и AutoML решения, которые автоматизируют ключевые этапы работы с ML: от предобработки данных до выбора оптимальной модели и её настройки. Такие системы не только ускоряют разработку, но и делают технологии ИИ доступными для широкого круга пользователей.

Целью данной работы является разработка интерактивного конструктора ML моделей, а также реализация и сравнение алгоритмов AutoML, предназначенных для подбора гиперпараметров моделей.

Задачи проекта:

1. Провести анализ существующих алгоритмов AutoML для подбора гиперпараметров.
2. Изучить существующие AutoML библиотеки.
3. Реализовать и сравнить несколько популярных HPO алгоритмов
4. Реализовать удобный интерфейс для обучения классических ML моделей с использованием библиотеки scikit-learn.
5. Реализовать систему трекинга экспериментов для отслеживания результатов моделей.

2. Теоретические основы методов машинного обучения и AutoML

2.1. Основные концепции машинного обучения

Машинное обучение — это область исследований, посвящённая разработке алгоритмов, способных на основе анализа данных выявлять закономерности и принимать решения без явного программирования. В основе большинства подходов лежит идея поиска функции f, которая сопоставляет входным данным x соответствующие значения y. При наличии обучающей выборки

$$\{(x_i, y_i)\}_{i=1}^{N},$$

задача сводится к поиску такой модели α, что

$$\alpha(x_i) \approx y_i, \quad i = 1, \ldots, N.$$

В качестве примера можно рассмотреть линейную модель

$$\alpha(x) = w^\top x + b,$$

где w — вектор весов, а b — смещение.

Ключевой этап в обучении моделей связан с минимизацией функции потерь $\mathcal{L}(y, \hat{y})$, которая измеряет расхождения между истинными значениями y и предсказаниями модели $\hat{y}$. Для задачи регрессии часто используется среднеквадратичная ошибка:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$

При этом обучение моделей включает итеративную оптимизацию параметров, где такие алгоритмы, как градиентный спуск, позволяют находить минимумы функций потерь. Итеративное обновление параметров можно записать следующим образом:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}(\theta),$$

где θ представляет набор параметров модели, η — скорость обучения, а $\nabla_\theta \mathcal{L}$ обозначает градиент функции потерь.

2.2. Почему AutoML? Предпосылки и мотивация

Ранее, для настройки гиперпараметров моделей использовались методы, требующие значительного участия эксперта. Ручной подбор параметров, например, с

применением *Grid Search* или *Random Search*, заключался в последовательном тестировании заранее заданных вариантов. Такой подход нередко был сопряжён с большими вычислительными затратами и высокой зависимостью от интуиции специалиста. Кроме того, человеческий фактор мог приводить к субъективным решениям, что, в свою очередь, влияло на качество итоговых моделей.

С увеличением сложности моделей и расширением пространства поиска гиперпараметров стало понятно, что традиционные методы не всегда способны обеспечить оптимальные результаты в разумные сроки. Эти ограничения стимулировали разработку новых подходов, в числе которых особое место заняла автоматизация процессов подбора параметров. Именно здесь концепция AutoML демонстрирует свою актуальность: автоматизация позволяет не только упростить настройку моделей, но и значительно повысить их эффективность за счёт интеграции алгоритмов, способных анализировать пространство гиперпараметров и выбирать наиболее перспективные направления для поиска.

Таким образом, переход от ручных методов подбора к системам AutoML определяется стремлением к снижению временных и вычислительных затрат, уменьшению влияния субъективного выбора и повышению качества итоговых моделей. В дальнейшем будут рассмотрены конкретные алгоритмы автоматизированного поиска оптимальных гиперпараметров.

2.3. Автоматизированное машинное обучение (AutoML)

Автоматизированное машинное обучение (AutoML) представляет собой совокупность подходов и инструментов, направленных на минимизацию ручного вмешательства эксперта в процессе разработки модели. Основные задачи AutoML включают:

1. Автоматизацию этапов предобработки и отбора признаков.
2. Автоматический подбор архитектуры модели.
3. Оптимизацию гиперпараметров.

В дальнейшем, в этой работе будет рассматриваться 3 пункт - подбор гиперпараметров (или HPO - Hyper Paramater Optimization). Среди классических подходов по поиску оптимальных гиперпараметров выделяются:

- **Grid Search** – перебор по заранее заданной сетке значений гиперпараметров.

- **Random Search** – случайный выбор комбинаций параметров, что часто оказывается более эффективным при ограниченных вычислительных ресурсах.

- **Bayesian optimization** – метод, который использует вероятностную модель (например, гауссовский процесс) для оценки целевой функции и выбора следующих точек исследования. Формально, поиск оптимальных гиперпараметров можно записать как задачу:

$$x^* = \arg\max_{x \in \mathcal{X}} \alpha(x),$$

  где $\alpha(x)$ — функция приобретения, отражающая полезность проверки данной точки, а $\mathcal{X}$ представляет пространство гиперпараметров.

- **Evolutionary algorithms** – подходы, заимствованные из теории эволюции, где генерация новых кандидатов основана на операциях скрещивания, мутации и отбора, что позволяет эффективно исследовать большое пространство гиперпараметров.

Автоматизация позволяет не только уменьшить временные затраты на поиск оптимальных параметров, но и делает технологии машинного обучения доступными для специалистов с различным уровнем подготовки. В современных информационных системах AutoML становится неотъемлемой частью реализации сложных аналитических пайплайнов.

3. HPO алгоритмы

3.1. Формальная постановка задачи

Постановка задачи оптимизации гиперпараметров (HPO) заключается в поиске вектора $\boldsymbol{\lambda} \in \Lambda$, для которого функция качества модели $Q(\boldsymbol{\lambda})$ достигает оптимального значения. Формально это записывается как

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda} \in \Lambda} \mathcal{L}(\boldsymbol{\lambda}),$$

где $\mathcal{L}(\boldsymbol{\lambda})$ — функция потерь на пространстве гиперпараметров Λ [1].

Важные особенности задачи HPO:

- **Вычислительная сложность.** При полном переборе (Grid Search) или случайном поиске (Random Search) время выполнения растёт экспоненциально от числа гиперпараметров [2].
- **Баланс исследования и эксплуатации.** Современные методы, такие как байесовская оптимизация [3], эффективно сочетают исследование нового пространства гиперпараметров и использование накопленных данных. Так называемые : *exploration* - исследование тех областей, в которых у нас мало семплов на текущей итерации, что даёт нам возможность с меньшей вероятностью пропустить оптимальное значение и *exploitation* - выбирать больше семплов в областях, которые мы достаточно неплохо изучили и где, как мы считаем, с большой вероятностью находится оптимум. [4]

Подходы к решению данной задачи могут значительно различаться как по методологии, так и по используемым вычислительным ресурсам.

3.2. GridSearch и Random Search

Два базовых метода для подбора гиперпараметров, которые часто используются как отправная точка для более сложных алгоритмов, это *Grid Search* и *Random Search*.

Grid Search заключается в следующем: исходное пространство гиперпараметров дискретизируется с помощью заранее определённой сетки. Пусть для

гиперпараметра λ_i задан набор значений $\{v_{i,1}, v_{i,2}, \ldots, v_{i,k_i}\}$. Тогда общая конфигурация параметров определяется декартовым произведением всех наборов. Таким образом, осуществляется полный перебор всех возможных комбинаций:

$$\Lambda_{\text{grid}} = \{(v_{1,j_1}, v_{2,j_2}, \ldots, v_{n,j_n}) \mid 1 \leq j_i \leq k_i\}.$$

Этот метод гарантирует нахождение оптимума при условии достаточной плотности сетки, однако его основным недостатком является резко возрастающее число конфигураций при увеличении размерности. Особенно остро эта проблема проявляется, когда лишь небольшое подмножество гиперпараметров действительно оказывает существенное влияние на производительность модели, в то время как Grid Search тратит ресурсы на перебор неважных параметров.

Random Search предлагает альтернативный подход, при котором выборка гиперпараметров осуществляется случайным образом из заданного пространства Λ. При таком подходе конфигурация параметров $\boldsymbol{\lambda}$ выбирается согласно заранее определённому распределению (например, равномерному для непрерывных параметров или дискретному равномерному для категориальных). Это позволяет с большей вероятностью «случайно» попасть в область с лучшим значением целевой функции, особенно если эффективное пространство поиска имеет низкую внутреннюю размерность [2]. На практике данный метод часто оказывается более эффективным при ограниченных вычислительных ресурсах, поскольку позволяет исследовать пространство более равномерно, не тратя время на проверки «неинтересных» комбинаций вдоль осей неважных гиперпараметров.

[4] Есть ещё одно довольно интересное объяснение, почему Random Search работает хорошо. Рассмотрим случай, когда у нас конечная сетка гиперпараметров (каждому гиперпараметру сопоставлено конечное число значений).

Для того чтобы с вероятностью не менее 95% хотя бы один из случайно выбранных наборов гиперпараметров оказался среди лучших 5%, необходимо, чтобы количество таких случайных выборов n удовлетворяло следующему неравенству:

$$1 - (1 - 0.05)^n \geq 0.95$$

Решая это неравенство, получаем $n \approx 59$. Это означает, что всего 59 случайных испытаний достаточно, чтобы с высокой вероятностью найти конфигурацию из топ-5% лучших, независимо от общего числа конфигураций в сетке, которое может быть огромным.

Оба метода обладают своими достоинствами и недостатками. Grid Search может быть полезен для очень маленьких пространств поиска, или когда требу-

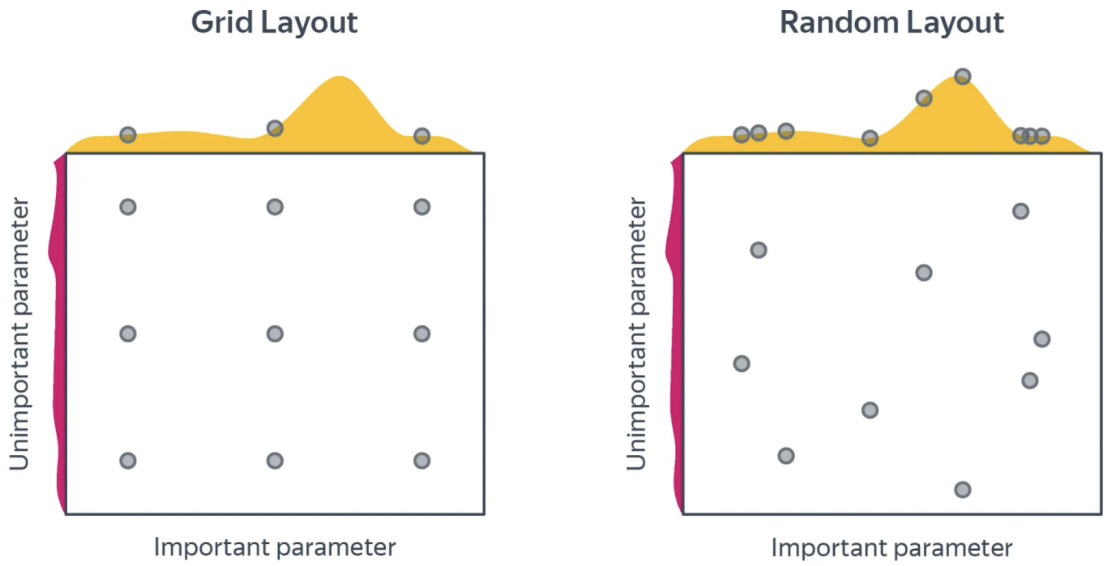


Рис. 1: Иллюстрация сравнения Grid Search и Random Search. Зелёным отмечена область оптимальных значений. Random Search с большей вероятностью находит точки в этой области при одинаковом количестве вычислений. Источник: Учебник яндекса по ML

ется гарантированно найти лучший вариант в дискретной сетке. Random Search часто является хорошей отправной точкой для большинства задач HPO из-за его эффективности и простоты.

3.3. Bayesian Optimization

Bayesian Optimization (байесовская оптимизация) представляет собой последовательную стратегию глобальной оптимизации «чёрного ящика», которая не предполагает фиксированной формы целевой функции и оптимизирует её на основе прошлых наблюдений [5]. Обычно в качестве вероятностной модели (суррогатной модели) целевой функции используется гауссовский процесс (GP), который моделирует обобщающую производительность алгоритма как выборку из GP и позволяет эффективно выбирать следующий набор гиперпараметров на основе приобретательной функции [6]. Гауссовский процесс определяется функцией среднего $m(\boldsymbol{\lambda})$ и ковариационной функцией (ядром) $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, которая описывает сходство между различными конфигурациями гиперпараметров.

Основной алгоритм байесовской оптимизации можно свести к следующим шагам:

1. **Инициализация:** Запускается несколько экспериментов (обычно выбранных случайно или с использованием сетки) с различными значениями ги-

перпараметров для получения начальных значений $\{(\boldsymbol{\lambda}_i, \mathcal{L}(\boldsymbol{\lambda}_i))\}_{i=1}^{n_0}$.

2. **Построение суррогатной модели:** На основе имеющихся наблюдений $D_t = \{(\boldsymbol{\lambda}_i, \mathcal{L}(\boldsymbol{\lambda}_i))\}_{i=1}^{t}$ строится вероятностная модель (например, подгоняется гауссовский процесс) для аппроксимации целевой функции $\mathcal{L}(\boldsymbol{\lambda})$. Эта модель даёт не только точечную оценку, но и меру неопределённости предсказания.
3. **Оптимизация функции приобретения:** Определяется функция приобретения $\alpha(\boldsymbol{\lambda} \mid D_t)$, которая количественно оценивает «полезность» вычисления $\mathcal{L}(\boldsymbol{\lambda})$ в новой точке $\boldsymbol{\lambda}$. Популярные функции приобретения включают Expected Improvement (EI), Probability of Improvement (PI) и Upper Confidence Bound (UCB). Функция приобретения отражает баланс между исследованием (exploration) областей с высокой неопределённостью и эксплуатацией (exploitation) областей, где модель предсказывает хорошие значения.
4. **Выбор следующей точки:** Выбирается следующая конфигурация гиперпараметров $\boldsymbol{\lambda}_{t+1}$ путём максимизации функции приобретения:

$$\boldsymbol{\lambda}_{t+1} = \arg\max_{\boldsymbol{\lambda} \in \Lambda} \alpha(\boldsymbol{\lambda} \mid D_t).$$

   Эта задача оптимизации сама по себе может быть сложной, но обычно решается стандартными методами оптимизации, поскольку функция приобретения обычно дешевле для вычисления, чем исходная целевая функция $\mathcal{L}(\boldsymbol{\lambda})$.
5. **Вычисление целевой функции и обновление модели:** Проводится эксперимент с новой конфигурацией $\boldsymbol{\lambda}_{t+1}$, получается значение $\mathcal{L}(\boldsymbol{\lambda}_{t+1})$. Новая пара $(\boldsymbol{\lambda}_{t+1}, \mathcal{L}(\boldsymbol{\lambda}_{t+1}))$ добавляется к набору наблюдений D_t, и модель пересчитывается (шаг 2). Шаги 2-5 повторяются до исчерпания бюджета вычислений.

Преимущества байесовской оптимизации включают её эффективность по количеству вызовов целевой функции, что делает её подходящей для дорогих в вычислении функций $\mathcal{L}$. Однако она может быть вычислительно затратной при большом количестве итераций из-за необходимости обновления и оптимизации суррогатной модели, а также её производительность сильно зависит от выбора ядра GP и функции приобретения.

3.4. Evolutionary Algorithm

Эволюционные алгоритмы (ЭА) представляют собой класс методов стохастической оптимизации, вдохновлённых естественным отбором и генетикой. Они

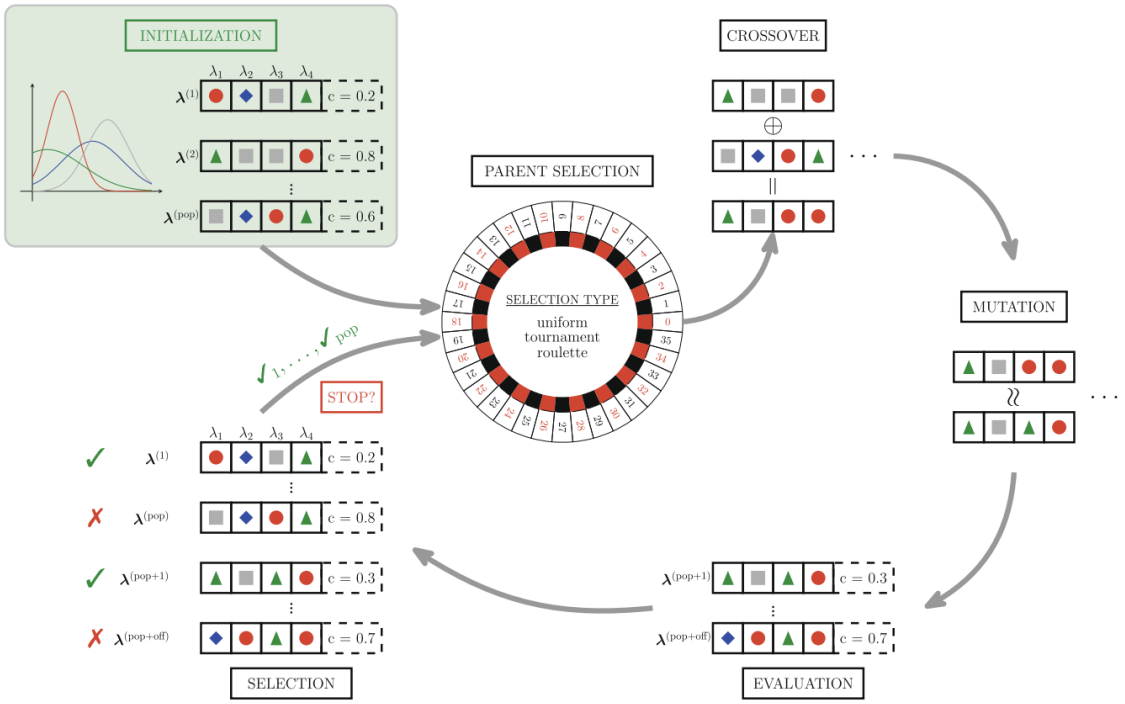


Рис. 2: Иллюстрация работы байесовской оптимизации. Суррогатная модель (например, гауссовский процесс) аппроксимирует целевую функцию, а функция приобретения помогает выбрать следующую точку для оценки. Источник: Учебник Яндекса по ML

оперируют популяцией потенциальных решений (индивидов), итеративно улучшая их с помощью генетических операторов, таких как отбор, скрещивание (рекомбинация) и мутация [7]. В контексте HPO каждый индивид в популяции соответствует вектору гиперпараметров $\boldsymbol{\lambda}$, а его «пригодность» (fitness) оценивается функцией потерь $\mathcal{L}(\boldsymbol{\lambda})$ (или её отрицанием, если мы максимизируем метрику качества).

Ключевые этапы эволюционного алгоритма для HPO можно сформулировать следующим образом:

1. **Инициализация популяции:** Генерируется начальная популяция

$$P^{(0)} = \{\boldsymbol{\lambda}_1^{(0)}, \boldsymbol{\lambda}_2^{(0)}, \ldots, \boldsymbol{\lambda}_N^{(0)}\},$$

   где N — размер популяции. Индивиды могут быть сгенерированы случайно в пределах допустимых диапазонов гиперпараметров.
2. **Оценка пригодности:** Для каждого индивида $\boldsymbol{\lambda}_i^{(t)}$ в текущем поколении $P^{(t)}$ вычисляется значение функции потерь $\mathcal{L}(\boldsymbol{\lambda}_i^{(t)})$, которое определяет его пригодность.
3. **Селекция (Отбор):** Индивиды выбираются из текущей популяции для

формирования родительского пула на основе их пригодности. Более приспособленные индивиды имеют больше шансов быть выбранными. Распространённые методы селекции включают турнирный отбор, отбор по принципу «колеса рулетки» и ранговый отбор.

4. **Генетические операторы (Скрещивание и Мутация):**
   - **Скрещивание (crossover/recombination):** Родительские пары, выбранные на предыдущем шаге, используются для создания одного или нескольких потомков. Оператор скрещивания комбинирует части генотипов (векторов гиперпараметров) родителей. Например, для числовых параметров это может быть арифметическое скрещивание (взвешенная сумма) или одноточечный/многоточечный обмен значениями.
   - **Мутация:** К генотипам потомков (а иногда и всех индивидов популяции) с некоторой вероятностью применяются небольшие случайные изменения. Это вносит разнообразие в популяцию и помогает избежать преждевременной сходимости к локальным оптимумам. Для числовых параметров мутация может заключаться в добавлении случайного значения из гауссовского распределения.
5. **Формирование нового поколения:** Новая популяция $P^{(t+1)}$ формируется из потомков и, возможно, части лучших индивидов из предыдущего поколения (элитизм). Существуют различные стратегии замещения (например, генерационная, стационарная).
6. **Критерий остановки:** Цикл (шаги 2-5) повторяется до тех пор, пока не будет выполнен критерий остановки, например, достижение заданного числа поколений (итераций), исчерпание вычислительного бюджета, или если улучшение лучшего решения в популяции становится незначительным.

Преимуществом ЭА является их универсальность (не требуют градиентной информации о целевой функции), способность работать с различными типами гиперпараметров (числовыми, категориальными, условными) и хорошая масштабируемость на многомерных и сложных пространствах поиска. Они также по своей природе подходят для параллельных вычислений, так как оценка пригодности индивидов может производиться независимо. Основным недостатком может быть высокая вычислительная стоимость, так как для сходимости может потребоваться большое количество оценок функции потерь (т.е. обучений моделей). Настройка самих параметров ЭА (размер популяции, вероятности скрещивания и мутации, тип операторов) также может быть нетривиальной задачей.

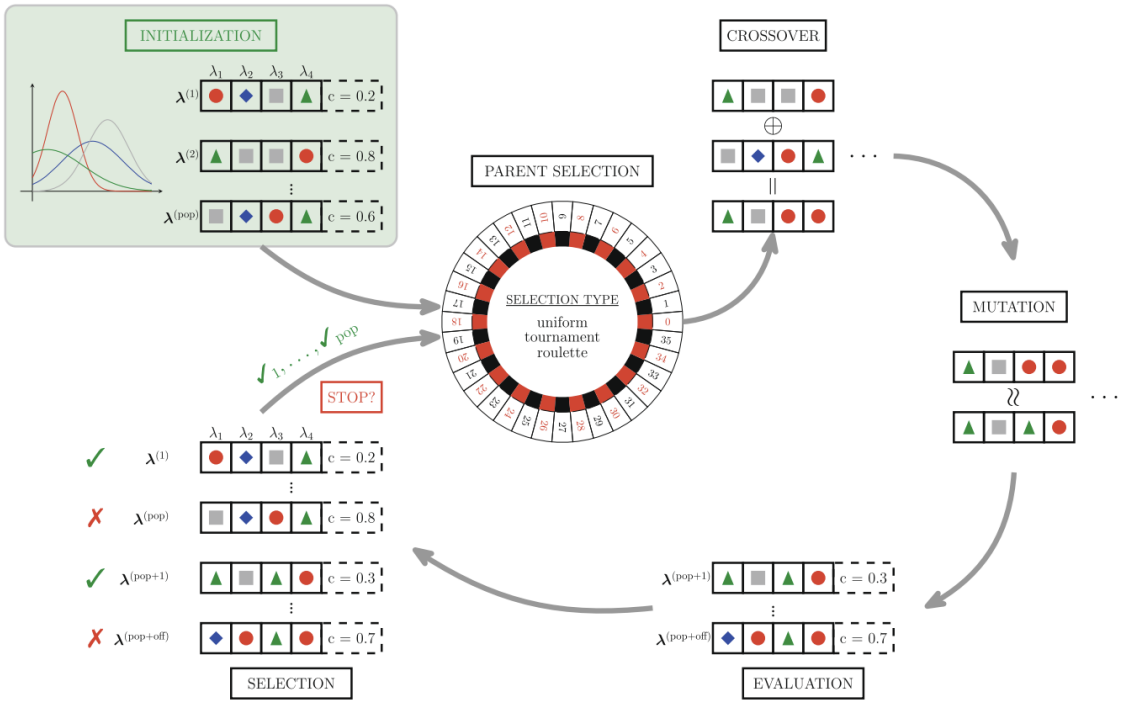


Рис. 3: Общая схема эволюционного алгоритма. Источник: Bischl et al. (2023), "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges WIREs Data Mining and Knowledge Discovery.

4. Дополнительные HPO алгоритмы

Помимо классических методов, существует ряд более продвинутых алгоритмов оптимизации гиперпараметров, которые стремятся повысить эффективность поиска и лучше управлять вычислительными ресурсами.

4.1. Hyperband

Hyperband [8] — это алгоритм оптимизации гиперпараметров, основанный на принципе многоуровневого отбора и раннего прекращения неэффективных конфигураций. Он адаптирует идею «successive halving» (последовательного сокращения вдвое) для эффективного распределения вычислительного бюджета между множеством случайно выбранных конфигураций. Вместо того чтобы полностью обучать каждую конфигурацию, Hyperband быстро отсеивает плохо работающие варианты на ранних стадиях.

Основная идея Hyperband заключается в запуске нескольких «кронштейнов» (brackets) алгоритма Successive Halving. Каждый кронштейн соответствует разному компромиссу между количеством исследуемых конфигураций (n) и

ресурсом (r), выделяемым каждой из них. Алгоритм Successive Halving (SHA) работает следующим образом:

1. Начать с n конфигураций, выделить каждой r_0 ресурсов (например, эпох обучения, подвыборка данных).
2. Оценить все n конфигураций.
3. Сохранить лучшую долю $1/\eta$ конфигураций.
4. Увеличить ресурс для сохраненных конфигураций в η раз.
5. Повторять шаги 2-4 до тех пор, пока не останется одна конфигурация или не будет достигнут максимальный ресурс.

Hyperband запускает SHA с различными начальными значениями n (и, соответственно, r_0), чтобы гарантировать, что как агрессивное раннее прекращение (много конфигураций, мало ресурсов на каждую), так и более консервативный подход (мало конфигураций, много ресурсов на каждую) будут исследованы.

Формально, Hyperband работает так:

1. Определение минимального ресурса r_{min} (например, 1 эпоха) и максимального R (например, общее количество эпох, доступных для одной конфигурации).
2. Вычисление максимального числа «раундов сокращения» в SHA для одного кронштейна: $s_{\max} = \lfloor \log_\eta(R/r_{min}) \rfloor$. Параметр $\eta > 1$ (обычно 3 или e) контролирует долю отбрасываемых конфигураций на каждом шаге SHA.
3. Для каждого значения $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ (каждое s определяет новый «кронштейн»):
   3.1. Определяется количество конфигураций $n_s = \lceil \frac{B}{R_{total,s}} \cdot \frac{s_{\max}+1}{s+1} \cdot \eta^s \rceil$ (примерно), которые будут запущены в этом кронштейне, где B – общий бюджет, $R_{total,s}$ – общий ресурс на кронштейн s. Более простой способ определить n_s: $n_s = \lceil \frac{(s_{max}+1)\eta^s}{s+1} \rceil$. Каждой из этих n_s конфигураций изначально выделяется $r_s = R \cdot \eta^{-s}$ ресурсов.
   3.2. Запускается алгоритм Successive Halving: на каждом шаге $i = 0, \ldots, s$, из $\approx \lfloor n_s/\eta^i \rfloor$ оставшихся конфигураций отбирается $\approx \lfloor n_s/\eta^{i+1} \rfloor$ лучших, и им выделяется $r_s \cdot \eta^{i+1}$ ресурсов. Остальные отбрасываются.

Hyperband теоретически обоснован в рамках задачи многорукого бандита с ограниченным бюджетом и демонстрирует хорошие эмпирические результаты, особенно когда кривые обучения моделей быстро расходятся.

Преимущества:

- Значительное сокращение вычислительных затрат за счёт раннего прекращения слабых конфигураций [9], что позволяет исследовать большее количество конфигураций в рамках одного и того же бюджета.
- Теоретические гарантии производительности при определённых предположениях.
- Простота реализации и хорошая масштабируемость для параллельных вычислений.
- Не требует построения суррогатной модели, что избавляет от связанных с этим накладных расходов.

Недостатки:

- Чувствительность к выбору параметров R, r_{min} и η. Неправильный выбор может привести к слишком раннему отсеву хороших конфигураций или, наоборот, к недостаточной экономии ресурсов.
- Случайный выбор начальных конфигураций в каждом кронштейне означает, что Hyperband не использует информацию из предыдущих оценок для выбора новых конфигураций (в отличие от байесовской оптимизации).
- Может быть неэффективен, если кривые обучения различных конфигураций сходятся очень медленно или пересекаются на поздних стадиях обучения (т.е. конфигурация, плохая на малом ресурсе, может стать лучшей на большом).

4.1.1. BOHB: Combining Hyperband with Bayesian Optimization

Для устранения недостатка случайного выбора конфигураций в Hyperband был предложен алгоритм BOHB (Bayesian Optimization and Hyperband) [10]. BOHB заменяет случайный выбор конфигураций на каждом этапе Hyperband на модель-ориентированный подход, использующий байесовскую оптимизацию (чаще всего TPE или гауссовские процессы с ядрами, учитывающими бюджеты).

- На ранних этапах, когда данных мало, BOHB может полагаться на случайный поиск или использовать априорные знания.

- По мере накопления оценок для различных конфигураций на разных бюджетах, строится суррогатная модель (например, TPE, где плотности $l(\lambda)$ и $g(\lambda)$ строятся на основе конфигураций, оцененных на определенном бюджете).

- Эта модель используется для выбора новых конфигураций, которые, как ожидается, будут хорошо работать на текущем бюджете в рамках данного кронштейна Hyperband. [10]

BOHB стремится объединить лучшие стороны обоих подходов: эффективное распределение ресурсов Hyperband и интеллектуальный поиск байесовской оптимизации. Это часто приводит к нахождению лучших конфигураций быстрее, чем любой из этих методов по отдельности, особенно для задач с дорогим вычислением целевой функции.

4.2. Sequential Model-Based Algorithm Configuration (SMAC)

SMAC (Sequential Model-Based Algorithm Configuration) [11, 12] — это метод байесовской оптимизации, который использует модели на основе случайных лесов (Random Forests) для аппроксимации функции потерь. Случайные леса хорошо подходят для этой задачи, поскольку они естественным образом обрабатывают категориальные и условные гиперпараметры, а также могут предоставлять оценку неопределенности предсказаний.

Основной цикл SMAC:

1. **Инициализация:** Проводятся n_0 испытаний с конфигурациями, выбранными случайно или с помощью методов латинского гиперкуба, чтобы собрать начальный набор данных $\{(\lambda_i, \mathcal{L}(\lambda_i))\}$.
2. **Построение суррогатной модели:** На текущем наборе наблюдений $D_t = \{(\lambda_i, \mathcal{L}(\lambda_i))\}_{i=1}^{t}$ обучается модель случайного леса $\mathcal{M}$, которая предсказывает $\hat{\mathcal{L}}(\lambda)$ и её неопределённость (например, дисперсию предсказаний деревьев в лесу).
3. **Оптимизация функции приобретения:** Вычисляется функция приобретения, обычно Expected Improvement (EI), которая использует предсказания $\hat{\mathcal{L}}(\lambda)$ и их неопределённость от случайного леса.

$$EI(\lambda) = E_{\mathcal{M}}[\max(0, \ell_{\min} - \mathcal{L}(\lambda))],$$

где $\ell_{\min}$ — наилучшее наблюдаемое на данный момент значение функции потерь. EI максимизируется для нахождения следующей наиболее перспективной конфигурации λ_{t+1}. Оптимизация EI обычно выполняется с помощью локальных поисковых процедур или эволюционных алгоритмов.

4. **Оценка выбранной конфигурации:** Новая конфигурация λ_{t+1} оценивается (т.е. обучается модель и вычисляется $\mathcal{L}(\lambda_{t+1})$).
5. **Обновление данных и модели:** Новое наблюдение $(\lambda_{t+1}, \mathcal{L}(\lambda_{t+1}))$ добавляется в D_t, и цикл повторяется с шага 2.
6. **Интенсификация (опционально):** SMAC также включает механизм «интенсификации». Вместо того чтобы всегда полностью оценивать каждую выбранную конфигурацию, SMAC может сравнивать несколько перспективных кандидатов (например, текущего лучшего «инкумбента» и несколько новых, выбранных с помощью EI) на возрастающих бюджетах (например, на больших подвыборках данных или с большим числом итераций обучения). Это позволяет быстрее отсеивать конфигурации, которые плохо работают, даже если EI их выбрал. [11, 12]

SMAC особенно эффективен для конфигурации алгоритмов с большим количеством категориальных или условных параметров, что часто встречается при настройке сложных решателей или алгоритмов машинного обучения.

Преимущества:

- Эффективная работа с категориальными и условными гиперпараметрами благодаря использованию случайных лесов [13].
- Устойчивость к шуму в оценках функции потерь.
- Механизм интенсификации позволяет эффективно распределять бюджет между множеством конфигураций.
- Широко используется и хорошо зарекомендовал себя, например, в библиотеке auto-sklearn.

Недостатки:

- Построение и обновление модели случайного леса может быть вычислительно затратным при очень большом числе наблюдений (тысячи и более).
- Оптимизация функции приобретения над сложным пространством конфигураций может быть нетривиальной.
- Производительность может зависеть от настроек самого SMAC (например, параметров случайного леса).

4.3. Tree-structured Parzen Estimator (TPE)

Алгоритм TPE (Tree-structured Parzen Estimator) [14, 15] является ещё одним популярным методом байесовской оптимизации. Вместо того чтобы моделировать $p(\mathcal{L} \mid \lambda)$ (вероятность оценки при заданных гиперпараметрах), как это делают методы на основе гауссовских процессов, TPE моделирует $p(\lambda \mid \mathcal{L})$ (вероятность гиперпараметров при заданной оценке).

Конкретно, TPE разделяет наблюдаемые точки $(\lambda, \mathcal{L}(\lambda))$ на две группы на основе некоторого порога ℓ^*. Обычно ℓ^* выбирается как γ-квантиль наблюдаемых значений $\mathcal{L}(\lambda)$ (например, $\gamma = 0.15$ или 0.25, что означает, что лучшие 15-25% точек считаются «хорошими»). Затем TPE строит две непараметрические оценки плотности вероятности:

- $l(\lambda) = p(\lambda \mid \mathcal{L}(\lambda) < \ell^*)$ — плотность для «хороших» конфигураций.
- $g(\lambda) = p(\lambda \mid \mathcal{L}(\lambda) \geq \ell^*)$ — плотность для «плохих» конфигураций.

Эти плотности обычно моделируются с использованием ядерных оценок Парзена (Kernel Density Estimators, KDE). Для обработки условных гиперпараметров используется древовидная структура KDE, где параметры в дереве активируются или деактивируются в зависимости от значений их родительских параметров.

Шаг алгоритма TPE:

1. Собрать начальный набор наблюдений (например, с помощью случайного поиска).
2. На каждой итерации:
   (a) Выбрать порог ℓ^* на основе текущих наблюдений (например, γ-квантиль значений $\mathcal{L}$).
   (b) Разделить наблюдения на две группы: $D_{\text{good}} = \{\lambda_i \mid \mathcal{L}(\lambda_i) < \ell^*\}$ и $D_{\text{bad}} = \{\lambda_i \mid \mathcal{L}(\lambda_i) \geq \ell^*\}$.
   (c) Построить модели плотностей $l(\lambda)$ на основе D_{good} и $g(\lambda)$ на основе D_{bad} с использованием KDE.
   (d) Найти новую конфигурацию λ_{new}, которая максимизирует критерий Expected Improvement (EI). В TPE это обычно сводится к максимизации отношения $l(\lambda)/g(\lambda)$. Интуитивно, мы ищем точки, которые имеют высокую вероятность по модели $l(\lambda)$ и низкую вероятность по модели $g(\lambda)$. На практике, для выбора λ_{new} генерируется множество

кандидатов сэмплированием из $l(\lambda)$, а затем выбирается тот, который максимизирует $l(\lambda)/g(\lambda)$.

3. Оценить $\mathcal{L}(\lambda_{\text{new}})$, добавить $(\lambda_{\text{new}}, \mathcal{L}(\lambda_{\text{new}}))$ к набору наблюдений и повторить.

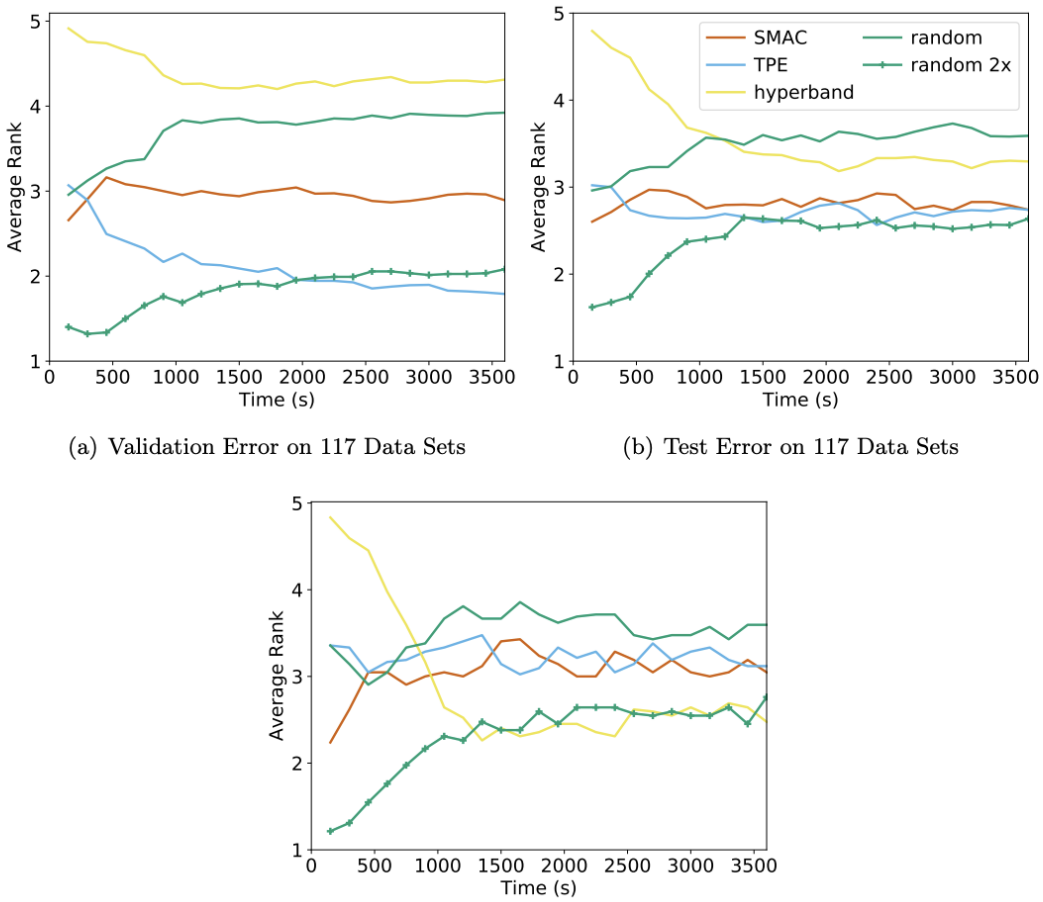
Преимущества:

- Естественная и эффективная работа с условными пространствами гиперпараметров благодаря древовидной структуре оценщиков Парзена [14].
- Часто более эффективен, чем GP-байесовская оптимизация, при большом количестве одновременных вычислений (параллелизм), так как выбор нескольких кандидатов проще.
- Меньшая вычислительная сложность на каждой итерации по сравнению с некоторыми реализациями GP, особенно при большом количестве уже сделанных оценок.
- Широко используется в таких библиотеках, как Hyperopt.

Недостатки:

- Производительность может зависеть от выбора квантиля γ и параметров KDE.
- Менее точная аппроксимация функции потерь в областях с малым количеством наблюдений по сравнению с GP.
- Как и другие методы байесовской оптимизации, может потребовать значительного числа начальных случайных проб для построения адекватных начальных моделей плотностей.

А вот как выглядит сравнение всех 3 вышеперечисленных алгоритмов:



(a) Validation Error on 117 Data Sets

(b) Test Error on 117 Data Sets

(c) Test Error on 21 Data Sets

Figure 5: Average rank across all data sets for each searcher. For each data set, the searchers are ranked according to the average validation/test error across 20 trials.

Рис. 4: Схематичное сравнение производительности различных HPO алгоритмов, включая Hyperband, SMAC, TPE, в зависимости от вычислительного бюджета. Источник: Адаптировано из Hyperband: A novel bandit-based approach to hyperparameter optimization [8] и других сравнительных исследований.

5. Обзор популярных AutoML-библиотек

В этом разделе рассматриваются ключевые библиотеки AutoML, которые широко применяются для автоматизации процесса подбора моделей и гиперпараметров, инкапсулируя многие из рассмотренных выше HPO алгоритмов.

5.1. auto-sklearn

auto-sklearn — это AutoML система с открытым исходным кодом, построенная на основе популярной библиотеки машинного обучения scikit-learn [12]. Она ав-

томатизирует выбор алгоритма машинного обучения и настройку гиперпараметров.

- **Основные HPO методы:** Использует байесовскую оптимизацию, в частности SMAC (Sequential Model-Based Algorithm Configuration), для эффективного поиска в пространстве конфигураций.
- **Meta-Learning:** Применяет мета-обучение для инициализации процесса оптимизации. Анализируя метаданные из множества предыдущих задач (например, с платформы OpenML), auto-sklearn определяет, какие конфигурации работали хорошо на похожих наборах данных, и использует их в качестве отправных точек для байесовской оптимизации. Это значительно ускоряет поиск.
- **Ансамблирование:** Автоматически создает ансамбль из лучших найденных моделей с использованием метода ансамблирования по выбору (ensemble selection), предложенного Caruana et al. Это часто приводит к более робастным и точным предсказаниям, чем любая отдельная модель.
- **Предобработка данных:** Включает автоматический выбор и настройку шагов предобработки данных (например, масштабирование, кодирование категориальных признаков, обработка пропусков).
- **Ограничения:** Может быть требовательной к вычислительным ресурсам, особенно на больших наборах данных. В основном ориентирована на табличные данные и задачи классификации и регрессии.

5.2. TPOT

TPOT (Tree-based Pipeline Optimization Tool) — это AutoML система на основе Python, которая оптимизирует пайплайны машинного обучения с использованием генетического программирования [16].

- **Основные HPO методы:** Использует генетические алгоритмы для эволюционного поиска оптимальной последовательности операций предобработки данных и моделей машинного обучения. Пайплайны представляются в виде деревьев.
- **Структура пайплайнов:** Каждый индивид в популяции генетического программирования представляет собой полный пайплайн, включающий шаги отбора признаков, предобработки, преобразования данных и моделирования.
- **Генетические операторы:** Применяет стандартные генетические операторы, такие как мутация (случайное изменение оператора или его парамет-

ра в дереве пайплайна), скрещивание (обмен поддеревьями между двумя родительскими пайплайнами) и селекция (выбор лучших пайплайнов для следующего поколения).
- **Гибкость:** Позволяет пользователям определять собственные компоненты пайплайна и метрики оптимизации. Поддерживает многоцелевую оптимизацию (например, баланс между точностью и сложностью модели).
- **Вывод кода:** Может экспортировать найденный оптимальный пайплайн в виде Python-кода на scikit-learn.

5.3. H2O AutoML

H2O AutoML — это компонент платформы H2O.ai, предназначенный для автоматизации процесса обучения моделей машинного обучения. H2O — это распределенная in-memory платформа с открытым исходным кодом.

- **Поддерживаемые модели:** Автоматически обучает и настраивает широкий спектр моделей, включая градиентный бустинг (GBM), случайные леса (DRF), глубокие нейронные сети, обобщенные линейные модели (GLM) и XGBoost.
- **HPO и стратегии:** Использует случайный поиск по сетке (Random Grid Search) для HPO, дополненный стратегиями ранней остановки и стеккинга.
- **Стеккинг ансамблей (Stacked Ensembles):** Автоматически создает два типа ансамблей: один из всех обученных моделей и один из лучших моделей каждого алгоритма. В качестве мета-ученика (meta-learner) обычно используется GLM.
- **Масштабируемость и интерфейсы:** Работает на Java-бэкенде, что обеспечивает хорошую производительность и масштабируемость для больших наборов данных. Предоставляет API для R, Python, Scala, а также веб-интерфейс H2O Flow для интерактивной работы и мониторинга.
- **Leaderboard:** В процессе работы H2O AutoML формирует таблицу лидеров (leaderboard), где модели ранжируются по заданной метрике качества, что позволяет пользователю отслеживать прогресс и выбирать лучшую модель.

5.4. AutoKeras

AutoKeras — это AutoML-библиотека на основе Keras, ориентированная на задачи глубокого обучения. Она автоматизирует процесс поиска архитектуры нейронной сети (Neural Architecture Search, NAS) и настройки гиперпараметров [17].

- **Neural Architecture Search (NAS):** Основная функция AutoKeras — автоматический поиск оптимальной архитектуры нейронной сети для заданной задачи (например, классификации изображений, текста, структурированных данных). Использует эффективные методы NAS, такие как сетевой морфизм или байесовская оптимизация, для исследования пространства архитектур.
- **Простота использования:** Предоставляет высокоуровневый API, схожий с scikit-learn, что делает его доступным для пользователей без глубоких знаний в проектировании нейронных сетей.
- **Гибкость:** Поддерживает различные типы задач (ImageClassifier, TextClassifier, StructuredDataClassifier и т.д.) и позволяет настраивать параметры поиска.
- **Расширяемость:** Построен на TensorFlow и Keras, что обеспечивает совместимость с экосистемой этих фреймворков.
- **Управление ресурсами:** Позволяет контролировать время поиска и другие вычислительные ресурсы.

5.5. Optuna

Optuna — это универсальный фреймворк для оптимизации гиперпараметров, который отличается гибкостью и современным Python-ориентированным API [18].

- **Define-by-Run API:** Ключевая особенность Optuna — это «define-by-run» стиль определения пространства поиска. Это означает, что пространство поиска гиперпараметров может быть задано динамически во время выполнения целевой функции с использованием стандартных конструкций Python (циклы, условные операторы). Это обеспечивает большую гибкость для сложных, условных пространств поиска.
- **Алгоритмы сэмплирования (Samplers):** Поддерживает различные алгоритмы сэмплирования, включая TPE (Tree-structured Parzen Estimator, используется по умолчанию), случайный поиск, сеточный поиск, CMA-ES

(Covariance Matrix Adaptation Evolution Strategy) для сложных непрерывных пространств.
- **Алгоритмы отсечения (Pruners):** Реализует механизмы раннего отсечения (pruning) для неперспективных испытаний. Включает такие прунеры, как MedianPruner, PercentilePruner, HyperbandPruner, которые могут остановить обучение конфигурации, если её промежуточные результаты хуже, чем у других.
- **Визуализация и анализ:** Предоставляет инструменты для визуализации процесса оптимизации, такие как графики истории оптимизации, важности гиперпараметров, контурные графики и графики параллельных координат. Имеется веб-интерфейс (Dashboard).
- **Интеграция:** Легко интегрируется с большинством популярных библиотек машинного обучения (scikit-learn, PyTorch, TensorFlow, XGBoost, LightGBM и др.).

5.6. Ray Tune

Ray Tune — это масштабируемая библиотека для оптимизации гиперпараметров, построенная поверх Ray, фреймворка для распределенных вычислений на Python [19].

- **Масштабируемость:** Основное преимущество Tune — лёгкое и эффективное масштабирование HPO на многоядерные процессоры и кластеры. Ray обеспечивает параллельное выполнение испытаний и управление ресурсами.
- **Широкий набор алгоритмов:** Поддерживает большое количество современных HPO алгоритмов и планировщиков, включая ASHA (Asynchronous Successive Halving Algorithm), HyperBand, BOHB, Population-Based Training (PBT), SMAC, TPE и другие.
- **Интеграция с ML фреймворками:** Легко интегрируется с популярными фреймворками глубокого обучения (PyTorch, TensorFlow, Keras) и классического ML (scikit-learn, XGBoost, LightGBM).
- **Гибкость:** Позволяет определять сложные пространства поиска, пользовательские метрики и использовать различные стратегии планирования и ранней остановки.
- **Отказоустойчивость и логирование:** Встроенные средства для логирования экспериментов (например, с TensorBoard) и обеспечения отказоустойчивости при распределенных запусках.

5.7. FLAML

FLAML (A Fast and Lightweight AutoML Library) — это легковесная Python-библиотека, которая быстро находит точные модели машинного обучения с минимальными вычислительными затратами [20].

- **Экономичность:** Основной фокус FLAML — нахождение качественных моделей при строгих ограничениях на время или вычислительные ресурсы. Это достигается за счет новых, экономичных методов оптимизации.
- **CFO и BlendSearch:** Использует новые алгоритмы HPO, такие как CFO (Cost-Frugal Optimization), который учитывает стоимость оценки различных конфигураций, и BlendSearch, который динамически сочетает достоинства экономичных методов и методов, требующих большего бюджета.
- **Универсальность:** Поддерживает как классические задачи машинного обучения (классификация, регрессия), так и задачи временных рядов и обучения с подкреплением.
- **Простота использования:** Предлагает простой API, похожий на scikit-learn (‘fit‘ и ‘predict‘).
- **Настраиваемость:** Позволяет пользователям задавать собственные метрики, модели и пространства поиска.

6. Сравнение и анализ собственных алгоритмов HPO

В этом разделе представлен сравнительный анализ методов оптимизации гиперпараметров (таких как Grid Search, Random Search, Bayesian Optimization, Evolutionary Strategy) для двух классических задач машинного обучения — регрессии и классификации. В качестве моделей выбраны: метод ближайших соседей, случайный лес и градиентный бустинг. Для каждого алгоритма подбора гиперпараметров оцениваются ключевые показатели: точность решения на тестовой выборке и затраченное время работы (в секундах).

6.1. Используемые датасеты

Будут использоваться два публичных датасета, полностью подготовленных для прямого использования в scikit-learn:

- **Diabetes** (регрессия): 442 наблюдения, 10 числовых признаков, целевая переменная — прогрессирование диабета через один год после исходного обследованияя.
- **Wine Quality (red)** (многоклассовая классификация): 1 599 наблюдений, 11 числовых признаков (физико-химические свойства вина), целевая переменная — оценка качества (от 3 до 8 баллов)

6.2. Экспериментальная настройка

- Модели и гиперпараметры (в скобках показана сетка для Grid Search):
  - **RandomForestClassifier & RandomForestRegressor:**
    - n_estimators от 50 до 200 (50, 100, 150, 200)
    - max_depth от 3 до 20 (3, 5, 10, 20)
    - min_samples_split от 2 до 10 (2, 5, 10)
    - min_samples_leaf от 1 до 4 (1, 2, 4)
    - min_impurity_decrease от 0.0 до 1.0 (0.0, 0.1, 0.5, 1.0)
  - **SVC & SVR:**
    - C от 0.1 до 10.0 (0.1, 1.0, 5.0, 10.0)
    - gamma от 0.01 до 1.0 (только для SVC) (0.01, 0.1, 0.5, 1.0)
    - epsilon от 0.01 до 0.2 (только для SVR) (0.01, 0.05, 0.1, 0.2)
  - **GradientBoostingClassifier & GradientBoostingRegressor:**

    - n_estimators от 50 до 200 (50, 100, 150, 200)
    - max_depth от 3 до 10 (3, 6, 10)
    - learning_rate от 0.01 до 0.3 (0.01, 0.1, 0.2, 0.3)
- Каждый алгоритм HPO выполняет 10, 25 и 50 запусков.

6.3. Результаты для задачи регрессии

Модель	**Алгоритм HPO**	**CV MSE**	**Время, с**
RandomForest	GridSearch	3207.9489	48.73
	RandomSearch	3247.1172	1.20
	BayesianOpt	3239.3289	1.68
	EvoStrategy	3234.2308	0.94
SVR	GridSearch	3038.1746	0.08
	RandomSearch	3048.3338	0.06
	BayesianOpt	3038.2845	0.55
	EvoStrategy	3048.3338	0.08
GradientBoosting	GridSearch	3273.8906	4.28
	RandomSearch	3587.1108	0.80
	BayesianOpt	3584.3281	1.89
	EvoStrategy	3410.7006	0.71

Таблица 1: Сравнение алгоритмов HPO на задаче регрессии (10 итераций)

Модель	Алгоритм HPO	CV MSE	Время, с
RandomForest	GridSearch	3207.9489	42.68
	RandomSearch	3231.6922	2.23
	BayesianOpt	3223.7244	4.59
	EvoStrategy	3225.5594	1.85
SVR	GridSearch	3038.1746	0.07
	RandomSearch	3044.2853	0.09
	BayesianOpt	3038.2486	1.41
	EvoStrategy	3038.3249	0.14
GradientBoosting	GridSearch	3273.8906	3.96
	RandomSearch	3381.8119	2.03
	BayesianOpt	3356.5876	4.62
	EvoStrategy	3368.6216	0.81

Таблица 2: Сравнение алгоритмов HPO на задаче регрессии (25 итераций)

Модель	Алгоритм HPO	CV MSE	Время, с
RandomForest	GridSearch	3207.9489	42.11
	RandomSearch	3231.6922	3.88
	BayesianOpt	3224.2415	9.28
	EvoStrategy	3222.1817	3.08
SVR	GridSearch	3038.1746	0.07
	RandomSearch	3041.0603	0.18
	BayesianOpt	3038.2486	3.53
	EvoStrategy	3038.2754	0.20
GradientBoosting	GridSearch	3273.8906	3.90
	RandomSearch	3365.1996	4.14
	BayesianOpt	3309.9584	8.97
	EvoStrategy	3276.3471	1.25

Таблица 3: Сравнение алгоритмов HPO на задаче регрессии (50 итераций)

6.4. Вывод по задаче регрессии

В задаче регрессии на датасете Diabetes наиболее стабильные и при этом быстрые результаты показали эволюционная стратегия и байесовская оптимизация. При 10 итерациях эволюционная стратегия обеспечила CV MSE около 3234 (на

порядка 1.4 % хуже GridSearch), но отработала почти в 50 раз быстрее (0.94 с против 48.7 с). При увеличении числа итераций до 50 эволюционная стратегия продолжила снижать CV MSE (3222), оставаясь в 13–15 раз быстрее классического GridSearch. Байесовская оптимизация при больших бюджетах итераций слегка превосходит эволюцию по точности (CV MSE 3309 при 50 итерациях у GBoost и 3224 у RF), но отрабатывает дольше. Классический GridSearch, хотя и демонстрирует наилучшие или близкие к ним результаты по точности, оказывается непрактичным по времени. SVR-модель в целом мало выигрывает от HPO с точки зрения MSE, все методы дают практически идентичные значения (около 3038), поэтому выбор стоит делать по критерию скорости — тут RandomSearch или эволюция выглядят предпочтительнее.

6.5. Результаты для задачи классификации

Модель	**Алгоритм HPO**	**CV Accuracy**	**Время, с**
RandomForest	GridSearch	0.6638	33.68
	RandomSearch	0.5426	0.58
	BayesianOpt	0.5403	1.28
	EvoStrategy	0.5426	0.53
SVC	GridSearch	0.5629	0.61
	RandomSearch	0.5528	0.51
	BayesianOpt	0.5574	1.03
	EvoStrategy	0.5528	0.50
GradientBoosting	GridSearch	0.6552	71.99
	RandomSearch	0.6560	12.95
	BayesianOpt	0.6615	27.55
	EvoStrategy	0.6583	17.76

Таблица 4: Сравнение алгоритмов HPO на задаче классификации (10 итераций)

Модель	Алгоритм HPO	CV Accuracy	Время, с
RandomForest	GridSearch	0.6638	30.78
	RandomSearch	0.5653	1.31
	BayesianOpt	0.5426	2.91
	EvoStrategy	0.5426	0.79
SVC	GridSearch	0.5629	0.63
	RandomSearch	0.5528	1.14
	BayesianOpt	0.5629	2.82
	EvoStrategy	0.5528	0.91
GradientBoosting	GridSearch	0.6552	70.54
	RandomSearch	0.6615	38.66
	BayesianOpt	0.6615	70.32
	EvoStrategy	0.6599	28.19

Таблица 5: Сравнение алгоритмов HPO на задаче классификации (25 итераций)

Модель	Алгоритм HPO	CV Accuracy	Время, с
RandomForest	GridSearch	0.6638	31.93
	RandomSearch	0.5653	2.25
	BayesianOpt	0.5895	6.17
	EvoStrategy	0.6482	1.56
SVC	GridSearch	0.5629	0.61
	RandomSearch	0.5621	1.93
	BayesianOpt	0.5629	5.58
	EvoStrategy	0.5543	1.17
GradientBoosting	GridSearch	0.6552	69.46
	RandomSearch	0.6615	72.83
	BayesianOpt	0.6615	147.08
	EvoStrategy	0.6630	60.23

Таблица 6: Сравнение алгоритмов HPO на задаче классификации (50 итераций)

6.6. Вывод по задаче классификации

В многоклассовой классификации на Wine Quality лучшие показатели по CV Accuracy и финальной Accuracy показывает градиентный бустинг, особенно при использовании байесовской оптимизации и эволюционной стратегии. Уже при

10 итерациях BayesianOpt повысил CV Accuracy до 0.6615 (против 0.6552 у GridSearch) и отработал примерно в 2.5 раза быстрее. При 50 итерациях эволюционная стратегия добилась CV Accuracy 0.6630 и финальной точности 0.664, при этом время работы было в 1.1 раз меньше, чем у GridSearch (60 с против 69 с). У RandomForest и SVC выгода от HPO по сравнению с GridSearch оказалась незначительной: CV Accuracy менялась мало, зато время существенно сокращалось у RandomSearch и эволюции. Таким образом, для классификации оптимальным балансом точности и скорости является применение эволюционной стратегии или байесовской оптимизации на GradientBoostingClassifier.

6.7. Общий вывод сравнения алгоритмов

Анализ показывает, что выбор метода оптимизации гиперпараметров зависит от компромисса между качеством модели и временными затратами. GridSearch остаётся надёжным способом достижения максимального качества, но его высокая вычислительная стоимость ограничивает применимость в ресурсозатратных сценариях. RandomSearch обеспечивает быструю настройку, однако нередко приносит заметное ухудшение метрик. Байесовская оптимизация часто сочетает хорошее качество и умеренное время работы, но её эффективность с увеличением числа итераций снижается из-за роста вычислительных затрат. Эволюционная стратегия демонстрирует наиболее гибкий баланс: она близка к лучшим метрикам качества (особенно для деревяистых моделей) и при этом требует значительно меньше времени по сравнению с GridSearch и BayesianOptimization.

7. Заключение

В ходе выполнения курсовой работы были разработаны и исследованы алгоритмы автоматическиго подбора гиперпараметров (HPO) для классических методов машинного обучения. Были решены следующие ключевые задачи:

1. Проведён анализ существующих подходов к решении задачи HPO, включая Grid Search, Random Search, байесовскую оптимизацию и эволюционные алгоритмы. Были также рассмотрены более современные методы, такие как Hyperband, SMAC и TPE, и их гибриды (например, BOHB).
2. Реализованы и сравнены четыре основных алгоритма подбора гиперпараметров (Grid Search, Random Search, Bayesian Optimization, Evolutionary Strategy) на задачах регрессии (Diabetes) и классификации (Wine Quality) с использованием трёх различных моделей машинного обучения (Random Forest, SVM, Gradient Boosting).
3. Систематизированы результаты экспериментов: представлены метрики качества (CV MSE, Test MSE, CV Accuracy, Test Accuracy) и время выполнения для различных бюджетов HPO (10, 25 и 50 итераций для итеративных методов).
4. Проведен анализ результатов проведённых измерений. Сформулированы общие выводы и практические рекомендации по выбору метода HPO.
5. Создана платформа, обеспечивающая удобное обучение как классических ML-моделей с пользовательскими гиперпараметрами, так и автоматизированных HPO-алгоритмов.
6. Разработана система трекинга для мониторинга и хранения эксперементов, а также их метрик.

Проведённые эксперименты показали, что не существует универсально лучшего HPO-алгоритма. Выбор зависит от множества факторов, включая сложность пространства поиска, стоимость оценки одной конфигурации, доступные вычислительные ресурсы и требуемый уровень точности. Эволюционные стратегии и байесовская оптимизация часто обеспечивают оптимальный баланс между качеством моделей и временем работы, значительно опережая по скорости классический Grid Search при сопоставимом или даже лучшем уровне точности, особенно при увеличении бюджета итераций. Random Search остается ценным инструментом для быстрого начального исследования или при очень строгих временных ограничениях.

Список литературы

[1] B. Bischl и др. “Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges”. В: *WIREs Data Mining and Knowledge Discovery* 13.2 (2023), e1484. DOI: 10.1002/widm.1484. URL: https://doi.org/10.1002/widm.1484.

[2] James Bergstra и Yoshua Bengio. “Random Search for Hyper-Parameter Optimization”. В: *Journal of Machine Learning Research*. 2012.

[3] Jasper Snoek, Hugo Larochelle и Ryan P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. В: *Advances in Neural Information Processing Systems*. 2012.

[4] Yandex School of Data Analysis. *Machine Learning Handbook*. https://education.yandex.ru/handbook/ml. 2025.

[5] *Bayesian optimization*. https://en.wikipedia.org/wiki/Bayesian_optimization.

[6] Jasper Snoek, Hugo Larochelle и Ryan P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. В: *Advances in Neural Information Processing Systems*. 2012. URL: https://arxiv.org/abs/1206.2944.

[7] A. E. Eiben и J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin, Heidelberg: Springer, 2003. ISBN: 978-3-540-40184-3.

[8] Lisha Li и др. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. В: *International Conference on Learning Representations*. 2017.

[9] Kevin Jamieson и Ameet Talwalkar. “Non-stochastic Best Arm Identification and Hyperparameter Optimization”. В: *Artificial Intelligence and Statistics*. 2017, с. 240—248.

[10] Stefan Falkner, Aaron Klein и Frank Hutter. “BOHB: Robust and Efficient Hyperparameter Optimization at Scale”. В: *Proceedings of the 35th International Conference on Machine Learning* (2018), с. 1436—1445.

[11] Frank Hutter, Holger H Hoos и Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. В: *Learning and Intelligent Optimization*. Springer. 2011, с. 507—523.

[12] Matthias Feurer и др. “Efficient and Robust Automated Machine Learning”. В: *Advances in Neural Information Processing Systems*. 2014, с. 2951—2959.

[13] Katharina Eggensperger и др. “Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters”. В: *Journal of Machine Learning Research* 14 (2013), с. 285—323.

[14] James Bergstra и др. “Algorithms for Hyper-Parameter Optimization”. В: *Neural Information Processing Systems*. 2011, с. 2546—2554.

[15] James Bergstra, Daniel Yamins и David D Cox. “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures”. В: *International Conference on Machine Learning*. 2013, с. 115—123.

[16] Randal S. Olson и др. “TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning”. В: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’16 Companion. New York, NY, USA: Association for Computing Machinery, 2016, с. 151—152. DOI: 10.1145/2908961.2908964.

[17] Haifeng Jin, Qingquan Song и Xia Hu. “Auto-Keras: An Efficient Neural Architecture Search System”. В: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. New York, NY, USA: Association for Computing Machinery, 2019, с. 1946—1956. DOI: 10.1145/3292500.3330648.

[18] Takuya Akiba и др. “Optuna: A Next-generation Hyperparameter Optimization Framework”. В: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. New York, NY, USA: Association for Computing Machinery, 2019, с. 2623—2631. DOI: 10.1145/3292500.3330701.

[19] Richard Liaw и др. “Tune: A Research Platform for Distributed Hyperparameter Tuning”. В: *arXiv preprint arXiv:1807.05118* (2018).

[20] Chi Wang и др. “FLAML: A Fast and Lightweight AutoML Library”. В: *Proceedings of the Fourth Conference on Machine Learning and Systems*. MLSys ’21. 2021.