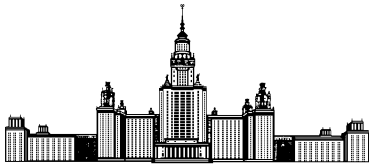
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Математических Методов Прогнозирования

КУРСОВАЯ РАБОТА СТУДЕНТА 317 ГРУППЫ

«Оптимизация гиперпараметров в алгоритмах машинного обучения»
«Hyperparameter optimization for machine learning algorithms»

Выполнил:

студент 3 курса 317 группы

Михеев Борис Михайлович

Научный руководитель:

д.ф-м.н., профессор

Дьяконов Александр Геннадьевич

Москва, 2022

Содержание

Аннотация

В данной работе проводится подробный обзор существующих подходов к оптимизации гиперпараметров в алгоритмах машинного обучения. Проведен сравнительный анализ наиболее распространенных методов, рассмотрены их особенности, преимущества и недостатки, а также проведены вычислительные эксперименты с библиотеками `optuna` и `scikit-optimize` с целью более детального исследования.

1 Введение

В настоящее время существует большое количество разнообразных алгоритмов машинного обучения. Они решают различные задачи и применяются во многих сферах человеческой деятельности. Чаще всего они параметризованы двумя наборами параметров: настраиваемыми во время обучения по данным (например, веса линейной модели или нейронной сети), и **гиперпараметрами**, задаваемыми до начала обучения и определяющими сложность и структуру итоговой модели (коэффициент регуляризации, число моделей в ансамбле и т. д.). Наличие гиперпараметров позволяет управлять процессом обучения и адаптировать алгоритм к различным данным и задачам. Однако они не могут быть настроены во время обучения. Возникает потребность в правильном их задании для достижения наилучшего качества и эффективности обучаемой модели.

В некоторых случаях выбор гиперпараметров может осуществляться вручную, по различным эмпирическим правилам или по результатам перебора всевозможных сочетаний их значений, например, с помощью поиска по сетке. Данные подходы достаточно просты, однако они крайне неэффективны в случае большого числа параметров, свойственного многим современным алгоритмам машинного обучения. Приобретает важность задача автоматического определения оптимальных гиперпараметров с целью снизить затраты времени и ресурсов в процессе настройки моделей, а также повысить их качество и обобщающую способность.

Оптимизация гиперпараметров сталкивается с рядом проблем. Ее цель заключается в нахождении параметров, минимизирующих или максимизирующих определенную метрику, характеризующую качество модели (например, потери или точность на валидации). Для этого потребуется многократное ее вычисление при различных значениях гиперпараметров, что может быть долго и дорого для сложных моделей или больших данных. Она может быть весьма сложно устроена, и как правило она не обладает свойствами гладкости и выпуклости, нет возможности вычислить ее градиент, к ней неприменимы известные градиентные методы оптимизации. Также гиперпараметров может быть много, и они могут быть различных типов, иметь раз-

личные диапазоны значений, зависеть друг от друга и в разной степени влиять на итоговое качество.

В настоящее время существует множество подходов к оптимизации гиперпараметров. В данной работе рассматриваются наиболее популярные из них, проводится их сравнительный анализ, а также проводится исследование реализаций некоторых методов применительно к различным моделям машинного обучения и типам данных.

2 Постановка задачи

Пусть $\mathcal{A}$ – алгоритм машинного обучения с N гиперпараметрами. Λ_n – область значений n-ого их них. $\Lambda = \Lambda_1 \times \cdots \times \Lambda_N$ – пространство конфигураций гиперпараметров, $\lambda \in \Lambda$ – вектор их значений . D_{train}, D_{valid} – обучающие и валидационные данные из некоторого распределения $\mathcal{D}$. $\mathcal{L}(D_{valid}, \mathcal{A})$ – функция потерь на валидационных данных. $\mathcal{F}(\lambda, \mathcal{A}, D_{train}, D_{valid}, \mathcal{L})$ – целевая функция, измеряющая потери модели, полученной алгоритмом $\mathcal{A}$ с гиперпараметрами λ при обучении на данных D_{train} и оцененной на валидационных данных D_{valid}.

Цель поиска гиперпараметров – найти оптимальный вектор их значений

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}} \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathcal{D}} \mathcal{F}(\lambda, \mathcal{A}, D_{train}, D_{valid}, \mathcal{L}).$$

Можно рассмотреть и иную постановку, где целевой функцией является, например, точность модели, и ее необходимо максимизировать. В любом случае имеется оптимизационная задача. На практике есть потребность не только минимизировать потери или максимизировать качество, но и сократить при этом время процесса поиска гиперпараметров, количество вычислительных операций и т. д., и в общем случае оптимизация гиперпараметров является многокритериальной задачей[1]. Сосредоточимся на базовой постановке с одним оптимизируемым критерием.

Данная задача имеет особенности и сталкивается с рядом проблем:

- Пространство поиска Λ может быть достаточно сложным. Гиперпараметры могут быть вещественными (темп обучения, коэффициент регуляризации), целочисленными (число базовых алгоритмов ансамбля, число слоев сети) или категориальными (вид препроцессинга, тип ядра для SVM). Между ними также могут существовать зависимости, например, если число слоев нейронной сети меньше n, то параметры n-ого слоя не имеют смысла. Помимо всего прочего было показано, что во многих случаях лишь часть гиперпараметров существенно влияет на целевую функцию, и отдельной задачей является определение наиболее важных из них[2]. Не всегда ясно, как задавать диапазон значений параметров.

- Целевая функция обычно имеет сложное устройство, не обладает свойствами гладкости, выпуклости или дифференцируемости, может иметь много локальных оптимумов, что делает применение градиентных методов оптимизации невозможным. Как правило неизвестна ее зависимость от гиперпараметров. По сути все, что с ней можно делать – это вычислять ее в определенных точках, к другой информации доступа нет.

- Вычисление целевой функции требует обучения модели с разными гиперпараметрами и оценки ее на валидационных данных. Этот процесс может занимать очень много времени. Также продолжительность обучения и тестирования может существенно зависеть от конкретных параметров (размер ансамбля, градиентный шаг). Возникает потребность в методах оптимизации гиперпараметров, совершающих как можно меньше вычислений целевой функции.

3 Обзор подходов к оптимизации гиперпараметров

Существует большое число соответствующих методов, как достаточно простых, так и использующих более сложные идеи. Как уже было отмечено, в общем случае устройство оптимизируемой функции в данной задаче неизвестно, и есть возможность лишь вычислять ее значения на входах. Можно только строить предположения о ней и подбирать оптимальные значения гиперпараметров определенным способом. Такие функции называют «черными ящиками» (black-box functions), и существует широкий спектр методов их оптимизации[3][4]. Предпочтение отдается алгоритмам глобальной оптимизации, не требующим использования производных. Условно их можно разделить на следующие группы:

- **Простейшие методы**, не использующие никакой информации и не делающие предположений о задаче или функции, сводящиеся по сути к простому перебору. Таковыми являются поиск по сетке и случайный поиск.

- **Метаэвристические методы**, использующие различные эвристики для поиска возможных решений задачи, оптимальных или близких к оптимальным. При этом они не используют предположения об оптимизируемой функции, в некотором смысле тоже реализуют случайный поиск, но подчиненный некоторым правилам или концепции. Их основная цель – наиболее эффективно исследовать пространство поиска на предмет близких к оптимуму значений гиперпараметров. Они не гарантируют нахождения глобально оптимального решения, однако могут быть применены к широкому кругу задач и показывают неплохие результаты на практике[5][6][7]. Ярким и распространенным примером таких методов являются популяционные алгоритмы[8], в которых поддерживается и итеративно обновляется набор возможных решений. Обновления происходят согласно некоторым правилам с элементами случайности.

- **Последовательные методы, основывающиеся на моделях** (Sequential Model-Based Optimization, SMBO). В них присутствуют две основных компоненты. Первая – вероятностная (т. н. суррогатная) модель, которая обучается по известным значениям целевой функции в наборе точек и аппроксимирует

ее. Вычисления этой модели как правило значительно менее затратны в сравнении с целевой функцией. Вторая – функция выбора (или функция выгоды, acquisition function) для определения следующей точки. Она использует предсказательное распределение вероятностной модели, оценивает перспективность точек-кандидатов, соблюдая баланс между уточнением рассмотренных регионов конфигурационного пространства и исследованием новых областей. Ярким примером такого подхода является байесовская оптимизация и различные ее модификации.[9][10]

Рассмотрим наиболее популярные методы более детально.

3.1 Простейшие методы

3.1.1 Поиск по сетке

Поиск по сетке является простым и широко используемым методом подбора гиперпараметров. Пространство поиска в нем задается дискретной сеткой – декартовым произведением конечных наборов значений гиперпараметров, задаваемых пользователем. Алгоритм вычисляет целевую функцию на всех наборах сетки и выдает набор гиперпараметров, соответствующий наилучшему среди найденных значений. Он прост в реализации и может выполняться параллельно, т. к. точки сетки оцениваются независимо. Однако он крайне неэффективен в случае большого числа гиперпараметров и высокой частоты дискретизации шкал их значений, т. к. число комбинаций, и, соответственно, вычислений целевой функции будет расти экспоненциально с ростом размерности пространства переменных и точности разбиения сетки. При этом данный алгоритм способен находить достаточно успешные конфигурации в случае низкой размерности конфигурационного пространства[1][2]. Также он не использует информацию о предыдущих успешных результатах. Это может приводить к рассмотрению большого числа неоптимальных конфигураций. Существует его модернизация – contracting grid search[11]. В начале поиск идет по сетке с грубым разбиением, далее строится более дискретизированная сетка меньшего размера с центром в наилучшей найденной комбинации, и уже по этой сетке вновь запускается поиск.

3.1.2 Случайный поиск

Данный алгоритм выбирает значения гиперпараметров независимо из заданных равномерных (в простейшем случае) распределений каждого из них. Процесс прекращается по достижении определенного числа итераций или по исчерпании некоторго бюджета (запаса времени, памяти и т. д.). Он также прост в реализации и распараллеливании, на практике находит более удачные конфигурации, чем поиск по сетке, часто используется в качестве бейзлайна в задачах оптимизации гиперпараметров[2]. Однако данный метод тоже не учитывает историю вычислений и информацию о потенциально оптимальных регионах, аналогично требует многократного вычисления целевой функции.

Стоит отметить, что случайный поиск более эффективен, чем поиск по сетке в конфигурационных пространствах большой размерности. На практике целевая функция часто имеет низкую эффективную размерность, т. е. ее зависимость от некоторого подмножества гиперпараметров значительно сильнее, чем от всех остальных.

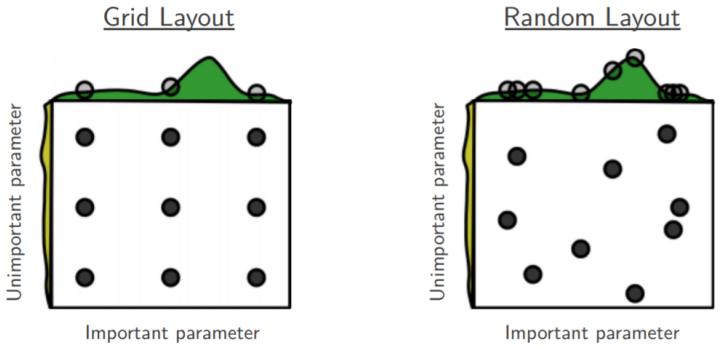


Рис. 1: Исслюстрация из статьи[2], демонстрирующая различия между поиском по сетке и случайным поиском.

На рис. 1 приведен пример из статьи[2], демонстрирующий покрытие конфигурационного пространства алгоритмами случайного поиска и поиска по сетке для целевой функции $f(x, y)$ с низкой эффективной размерностью. Ее значение при различных x меняется сильно, но почти не реагирует на изменения y, т. е. по сути она зависит от своих аргументов независимо, и тогда $f(x, y)$ можно представить как сумму двух функций одного аргумента: $f(x, y) \approx f_1(x) + f_2(y)$. Сетка задает равномерное покрытие пространства, но проекции точек на оси координат расположены с доволь-

но большим шагом, и в данном случае проекции на ось x попали в области с малым значением $f_1(x)$ и, соответственно, $f(x, y)$, упустив из рассмотрения перспективный регион поиска. Точки, выбираемые случайным поиском, распределены менее упорядоченно, но при этом более плотно покрывают проекциями координатные оси, и на рис. 1 они попали очень близко к оптимуму. Это позволяет случайному поиску рассмотреть большее число регионов в конфигурационном пространстве и повысить вероятность нахождения наилучшего решения. Однако в некоторых случаях он может очень долго сходиться к оптимальным точкам или вовсе не достигнуть их в силу случайности. Стоит отметить, что в случае дискретных переменных в рассмотренном примере покрытия алгоритмов будут идентичны.

Существует модификация случайного поиска – взвешенный случайный поиск[12]. Он основывается на предположении, что значения некоторых из гиперпараметров, приведшие к хорошему результату, стоит протестировать с другими значениями остальных. Стандартный случайный поиск на каждой итерации генерирует новые значения для каждого параметра. В модифицированном алгоритме значения гиперпараметров изменяются с некоторой вероятностью p. Для каждого параметра i после некоторого числа итераций k_i вместо того, чтобы всегда генерировать новое значение, алгоритм генерирует его с вероятностью p_i или заменяет на значение этого параметра, соответствующее лучшему известному значению целевой функции с вероятностью $1-p_i$. Основная идея такого метода в том, что если функция уже прооптимизирована по некоторому числу переменных, то эффективнее будет остальным переменным присвоить лучшие известные значения, чем генерировать случайные. Это также позволяет избегать застревания в локальных оптимумах. На практике данный алгоритм превосходит стандартную версию при одинаковом числе рассматриваемых комбинаций[12].

В некотором смысле развитием идеи случайного поиска является алгоритм Hyperband[13], использующий концепцию многоруких бандитов[14]. Конфигурации параметров в нем выбираются случайно, но изначально им выделяются лишь части данных для обучения, и для наиболее успешных из них объем обучающей выборки адаптивно увеличивается.

Также существуют варианты случайного поиска, использующие различные стратегии раннего останова и сэмплирования точек[15][16].

3.2 Метаэвристические алгоритмы

3.2.1 Генетический алгоритм

Генетический алгоритм является характерным примером эволюционной стратегии. Он осуществляет оптимизацию, используя концепцию естественного отбора. Создается и поддерживается популяция возможных решений задачи. Согласно используемой терминологии, вектор гиперпараметров называется генотипом или хромосомой, его координаты – генами. Функцией приспособленности обычно называют оптимизируемую функцию в случае максимизации, в противном случае она берется с обратным знаком. Начальная популяция как правило задается случайно. Затем моделируется итеративный эволюционный процесс. Каждая итерация соответствует одному поколению и состоит из следующих основных стадий: скрещивание, мутации, селекция по значению функции приспособленности и создание нового поколения. Процесс продолжается до выполнения критерия остановки, как правило превышения числа итераций или количества вычислений целевой функции.

Выбор родителей для скрещивания может производиться различными способами. Обычно в них вероятность выбора особи прямопропорциональна ее приспособленности. Часто используется турнирная селекция: отбирается фиксированное число особей, и в качестве родителя выбирается особь с наилучшей приспособленностью с некоторой вероятностью p. Если она не была выбрана, то родителем становится вторая по приспособленности особь, и т. д. Процесс продолжается, пока не наберется нужное число пар родителей.

Следующей стадией является скрещивание. Оно может быть проведено множеством способов, но требуется, чтобы потомки тем или иным образом унаследовали черты обоих родителей. Один из способов – разбить родительские хромосомы на k частей, и хромосому потомка составлять из родительских участков, выбираемых в случайном порядке.

Далее к определенной доле популяции применяется операция мутации. Она также может быть осуществлена различными способами, часто прибавлением случайного числа из нормального распределения с нулевым средним ко всем генам или к их части. Применение мутаций приводит к повышению разнообразия популяции, позволяет не застревать в локальных оптимумах и рассматривать области пространства, в которых не побывали предки. Однако если применять слишком сильные мутации, то алгоритм станет близок к случайному поиску. Можно в таком случае пошагово уменьшать дисперсию распределения, из которого берутся слагаемые для мутаций. Также это может помочь алгоритму лучше сойтись к оптимуму по мере приближения к нему[7].

После оценивается приспособленность популяции, производится селекция наилучших особей и формируется новое поколение. Можно поддерживать всегда заданное число наиболее приспособленных особей в популяции и передавать их хромосомы вместе с потомками в следующее поколение (т. н. концепция элитизма). Обычно это приводит к улучшению сходимости алгоритма[7].

Генетический алгоритм достаточно прост, может быть легко выполнен параллельно, т. к. оценка особей и выполнение скрещиваний и мутаций происходят независимо. Однако он требует много вычислений оптимизируемой функции. В нем присутствует достаточно много параметров для варьирования (тип и вероятность мутаций, тип селекции и т. д.), что позволяет адаптироваться к различным задачам.

Интересным примером применения генетического алгоритма является задача поиска архитектуры нейронной сети[17]. Параметры архитектуры сети можно закодировать вектором-генотипом. В статье описана проблема «перестановок»: две архитектуры, описанные разными векторами состояний, могут быть эквиваленты (рис. 2). Применение генетического алгоритма и операций скрещивания и мутаций позволяет лучше исследовать конфигурационное пространство и найти оптимальную архитектуру среди «перестановок».

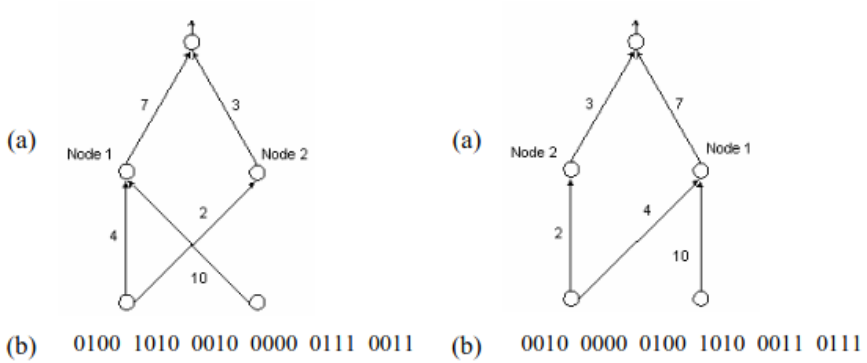


Рис. 2: Иллюстрация из статьи[17], демонстрирующая т. н. проблему «перестановок». Две архитектуры кодируются различными векторами параметров, однако на деле являются эквивалентными (используется двоичное кодирование весов).

Одним из часто применяемых и эффективных вариантов генетического алгоритма является Nondominated Sorting Genetic Algorithm II (NSGA-II)[18]. Его главными особенностями являются использование стратегии элитизма, т. е. передачи наилучших особей в следующее поколение, и возможность обобщения на многокритериальные оптимизационные задачи. На каждом шаге алгоритма генерируется популяция потомков того же размера N, что и родительская, и из общей популяции в следующее поколение отбирается N особей с наибольшей приспособленностью. NSGA-II успешно справляется с различными оптимизационными задачами, особенно многокритериальными, превосходит другие реализиции генетического алгоритма, имеет лучшую сходимость и разнообразие популяции по сравнению с ними[18].

3.2.2 Оптимизация роя частиц

Алгоритм оптимизации роя частиц (Particle Swarm Optimization, PSO)[19] использует концепцию т. н. роевого ителлекта. Имеется множество возможных решений задачи, называемых частицами. Их эволюция происходит итеративно. Ключевая идея состоит в том, что частицы могут обмениваться информацией и благодаря этому ускорить процесс поиска решения, т. е. популяция будет «сообща» группироваться в области оптимума. Пусть $f : \mathbb{R}^n \to \mathbb{R}$ - минимизируемая целевая функция, S – число

частиц в рое, x_i^k – положение i-ой частицы на k-ой итерации, v_i^k – вектор ее скорости, p_i^k – наилучшее ее известное положение с точки зрения значения f, g^k – наилучшее положение роя, т. е. точка с наименьшим известным на данный момент значением f. Начальное положение частиц выбирается из многомерного равномерного распределения, соответствующего рассматриваемым диапазонам значений гиперпараметров, т. е. для j-ой координаты вектора x_i^0: $(x_i^0)_j \sim U(low_j, high_j)$ $\forall j = \overline{1,n}$, $[low_j, high_j]$ – диапазон значений j-ой координаты. Далее инициализируется лучшее известное положение: $p_i^0 = x_i^0$. Начальное наилучшее положение роя g задается в общем случае как значение f в произвольной точке, затем, $\forall i = \overline{1,S}$, если $f(p_i^0) < f(g)$, то $g = p_i^0$. Инициализируются значения скоростей: $(v_i^0)_j \sim U(low_j - high_j, high_j - low_j)$ $\forall j = \overline{1,n}$. Затем запускается итерационный процесс до достижения максимального числа итераций или до выполнения иного критерия. Рассмотрим шаг k. Для каждой частицы генерируются векторы r_p, r_g с координатами из $U(0,1)$, обновляется значение скорости: $v_i^{k+1} = wv_i^k + \phi_p r_p \times (p_i^k - x_i^k) + \phi_g r_g \times (g - x_i^k)$, где $\times$ – покоординатное умножение, w, ϕ_p, ϕ_g – задаваемые параметры. ϕ_p, ϕ_g также называют когнитивным и социальным весами. Частица смещается на вектор скорости: $x_i^{k+1} = x_i^k + v_i^{k+1}$. Если $f(x_i^{k+1}) < f(p_i^k)$, то $p_i^{k+1} = x_i^{k+1}$, и если $f(p_i^{k+1}) < f(g)$, то $g = p_i^{k+1}$. После завершения процесса g – наилучшее найденное решение. В формуле пересчета скорости учитывается как локальный фактор (близость к собственному лучшему положению), так и глобальный (близость к лучшему положению среди всех частиц).

Данный метод прост, легко распараллеливается, но требует множества вычислений целевой функции. Существует большое число его усовершенствований[19], во многом базирующихся на изменении формулы пересчета скоростей и задании w, ϕ_p, ϕ_g особым способом. Таким образом, можно варьировать влияние соседних частиц или учитывать информацию о прошлом направлении движения для ускорения сходимости и избежания застревания в локальных оптимумах. Также применяются различные стратегии сэмплирования начальной популяции. PSO используется в различных задачах, в том числе и в обучении глубоких нейронных сетей, и показывает неплохие результаты[5].

3.2.3 CMA-ES

CMA-ES является эффективным и наиболее продвинутым эволюционным алгоритмом оптимизации. Полное описание метода приведено в соответствующих статьях[20][8]. Рассмотрим ключевые моменты и идеи.

В определенном смысле его стадии схожи со стадиями генетического алгоритма. Основная идея – по результатам каждого поколения увеличивать или уменьшать пространство поиска в направлении оптимума функции для следующих поколений, изменяя форму распределения точек и его положение путем изменения ковариационной матрицы и вектора средних соответственно.

Пусть $f : \mathbb{R}^n \to \mathbb{R}$ – минимизируемая целевая функция. Задается размер популяции λ, обычно $\lambda \geq 4$. $m^{(g)} \in \mathbb{R}^n$, $C^{(g)} \in \mathbb{R}^{n \times n}$ – вектор средних и ковариационная матрица для g-ого поколения, $\sigma^{(g)}$ – т. н. размер шага. $m^{(0)}$ и $\sigma^{(0)}$ выбираются в зависимости от задачи, $C^{(0)} = I$, p_σ, $p_c \in \mathbb{R}^n$ – т. н. эволюционные пути, инициализируются нулевыми векторами θ.

Далее на стадии мутации для поколения с номером g потомки генерируются из многомерного нормального распределения:

$$x_k^{(g+1)} \sim \mathcal{N}\left(m^{(g)}, (\sigma^{(g)})^2 C^{(g)}\right) \quad \forall k = \overline{1, \lambda}.$$

На стадии селекции выбирается $\mu \leq \lambda$ родительских особей $x_{i:\lambda}^{(g+1)}$, $i = \overline{1, \mu}$ с наименьшим значением f, т. е. $f(x_{1:\lambda}^{(g+1)}) \leq \cdots \leq f(x_{\mu:\lambda}^{(g+1)})$. Далее производится рекомбинация, для следующего поколения $g + 1$ вычисляется вектор средних:

$$m^{(g+1)} = m^{(g)} + c_m \sum_{i=1}^{\mu} w_i \left(x_{i:\lambda}^{(g+1)} - m^{(g)}\right),$$

$$\sum_{i=1}^{\mu} w_i = 1, \quad w_1 \geq \cdots \geq w_\mu > 0,$$

где $w_i, i = \overline{1, \mu}$ – весовые коэффициенты, $c_m \leq 1$ – темп обучения. Множители w_i могут быть выбраны различными способами, и свобода их задания позволяет применять различные механизмы селекции, в разной степени учитывать $x_{i:\lambda}^{(g+1)}$. Например,

при $c_m = 1$, $w_i = \frac{1}{\mu}$, $i = \overline{1,\mu}$ особи учитываются одинаково, вектор средних $m^{(g+1)}$ будет просто средним арифметическим решений $x_{i:\lambda}^{(g+1)}$. Для задания w_i авторами алгоритма используется т. н. эффективная масса отбора $\mu_{eff} = (\sum_{i=1}^{\mu} w_i^2)^{-1}$. Из определения w_i вытекает, что $1 \leq \mu_{eff} \leq \mu$, и $\mu_{eff} = \mu$ для $w_i = \frac{1}{\mu}$, $i = \overline{1,\mu}$. В определенном смысле μ_{eff} можно интерпретировать как число наиболее учитываемых родительских особей $x_{i:\lambda}^{(g+1)}$ при вычислении $m^{(g+1)}$. В оригинальной статье[20] утверждается, что выбор w_i таких, что $\mu_{eff} \approx \frac{\lambda}{4}$, является наиболее эффективным. Также в ней утверждается, что $c_m < 1$ позволяет адаптировать алгоритм к зашумленным функциям. В большинстве случаев выбирается $c_m = 1$.

Следующая стадия – преобразование ковариационной матрицы. Формула пересчета учитывает три составляющие – текущую ковариационную матрицу, ковариацию между поколениями, и ковариационную матрицу, оцененную по новым отобранным особям (первое, второе и третье слагаемое соответственно):

$$C^{(g+1)} = \left(1 - c_1 - c_\mu \sum_{i=1}^{\lambda} w_i\right) C^{(g)} + c_1 p_c^{(g+1)} p_c^{(g+1)T} + \frac{c_\mu}{(\sigma^{(g)})^2} \sum_{i=1}^{\lambda} w_i \left(x_{i:\lambda}^{(g+1)} - m^{(q)}\right) \left(x_{i:\lambda}^{(g+1)} - m^{(q)}\right)^T.$$

Таким образом, ковариационная матрица обновляется с учетом информации обо всей популяции и о связи между поколениями. Рассмотрим формулу подробнее. c_1, c_μ – весовые слагаемые, по сути темпы обучения. Вектор $p_c^{(g+1)}$ – т. н. эволюционный путь. Эволюционным путем называют последовательность успешных шагов $\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}$, т. е. смещений распределения точек. Его можно рассмотреть как сумму последовательных шагов с экспоненциальным сглаживанием:

$$p_c^{(g+1)} = (1 - c_c)\, p_c^{(g)} + \sqrt{c_c(2 - c_c)\mu_{eff}} \frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}},$$

где $c_c \leq 1$ – весовое слагаемое, в определенном смысле коэффициент забывания прошлого пути, определяет влияние предыдущих успешных сдвигов распределений. $\sqrt{c_c(2 - c_c)\mu_{eff}}$ – константа нормализации для эволюционного пути. При $c_c = 1$, $\mu_{eff} = 1$ в силу определений μ_{eff} и w_i получим $p_c^{(g+1)} = \frac{x_{1:\lambda}^{(g+1)} - m^{(g)}}{\sigma^{(g)}}$. Также если $p_c^{(g)} \sim \frac{x_{i:\lambda}^{(g+1)} - m^{(g)}}{\sigma^{(g)}} \sim \mathcal{N}(0, C^{(g)})$, $c_m = 1$, то т. к. $(1 - c_c)^2 + \left(\sqrt{c_c(2 - c_c)}\right)^2 = 1$, $\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}} \sim \frac{c_m}{\sigma^{(g)}} \sum_{i=1}^{\mu} w_i \left(x_{i:\lambda}^{(g+1)} - m^{(g)}\right) \sim \sum_{i=1}^{\mu} w_i \mathcal{N}\left(0, C^{(g)}\right) \sim \frac{1}{\sqrt{\mu_{eff}}} \mathcal{N}\left(0, C^{(g)}\right)$, то и $p_c^{(g+1)} \sim \mathcal{N}\left(0, C^{(g)}\right)$. Т. к. при инициализации $p_c^{(0)} = \theta$, то использование такой константы нормализации приводит к тому, что $p_c^{(g+1)} \sim \mathcal{N}(0, C^{(g)})$, что логично, ведь

смещения распределения вообще говоря равноправны по всем направлениям. Эволюционный путь хранит информацию о взаимосвязи между отдельными шагами, и это может значительно улучшить процесс поиска.

Алгоритм также контролирует размер шага σ на стадии мутации. Он отвечает за скорость перемещения популяции по пространству поиска. Для его настройки также можно использовать идею эволюционного пути. Рассмотрим иллюстрацию из статьи[20]:

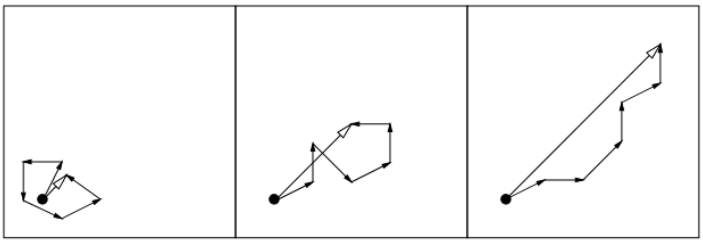


Рис. 3: Иллюстрация из статьи[20], демонстрирующая различные возможные эволюционные пути в алгоритме CMA-ES в случае двумерного пространства поиска. Оси соответствуют значениям гиперпараметров. Центр распределения популяции (черная точка) может перемещаться к оптимуму различными способами в зависимости от его положения.

На рис. 3 длины каждого шага по отдельности сравнимы друг с другом. Длина эволюционного пути сильно отличается, и ее можно использовать для контролирования размера шага. Когда эволюционный путь мал, как на левом изображении рис. 3, то отдельные шаги компенсируют друг друга, грубо говоря они антикоррелированы. В таком случае стоит уменьшить размер шага, т. к. вероятно популяция уже подошла к оптимуму, и требуются болеее аккуратные шаги для его достижения. Если же эволюционный путь большой, как на правом изображении рис. 3, отдельные шаги указывают на сходные направления, они в некоторой степени взаимосвязаны друг с другом. Одно и то же расстояние в таком случае можно преодолеть меньшим количеством, но более длинных шагов в одних и тех же направлениях. В предельном случае, когда последовательные шаги имеют одинаковое направление, они могут быть заменены любым из увеличенных одиночных шагов. Следовательно, размер шага должен быть увеличен для ускорения процесса, т. к. вероятно в такой ситуации

популяция находится далеко от оптимума. На среднем изображении рис. 3 показана промежуточная ситуация.

Для корректирования шага строится вектор $p_\sigma^{(g+1)}$, называемый вектором сопряженного эволюционного пути:

$$p_\sigma^{(g+1)} = (1 - c_\sigma)\, p_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}}\, \left(C^{(g)}\right)^{-\frac{1}{2}} \frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}},$$

где $c_\sigma < 1$ – также весовое слагаемое, $p_\sigma^{(0)} = \theta$. Константа нормализации $\sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}}$ и c_σ выбираются из тех же соображений, что и в случае $p_c^{(g+1)}$. В $p_\sigma^{(g+1)}$ также учитывается информация о совершенных шагах. Изменение $\sigma^{(g+1)}$ должно зависеть от длины эволюционного пути, но не его направления. Для этого в оригинальной статье применяется домножение $\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}$ на матрицу $\left(C^{(g)}\right)^{-\frac{1}{2}}$ слева. Утверждается, что это приводит к равномерному масштабированию $\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}$ относительно главных компонент распределения $\mathcal{N}\left(0, C^{(g)}\right)$, и таким образом в $p_\sigma^{(g+1)}$ учитывается лишь длина эволюционного пути[20]. В итоге длина шага корректируется следующим образом:

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||p_\sigma^{(g+1)}||}{\mathbb{E}||\mathcal{N}(0, I)||} - 1\right)\right),$$

где $d_\sigma \approx 1$, по сути контролирует изменение шага, $\mathbb{E}||\mathcal{N}(0, I)||$ – матожидание нормы вектора из $\mathcal{N}(0, I)$. Обоснование данных формул приведено в оригинальной статье[20]. При таком изменении шага, если $||p_\sigma^{(g+1)}|| > \mathbb{E}||\mathcal{N}(0, I)||$, т. е. если длина эволюционного пути достаточно велика, то $\sigma^{(g+1)}$ увеличивается. В противном случае, когда длина эволюционного пути мала, и, соответственно, $||p_\sigma^{(g+1)}|| < \mathbb{E}||\mathcal{N}(0, I)||$, то шаг $\sigma^{(g+1)}$ уменьшается. Это позволяет популяции быстрее и эффективнее двигаться в сторону оптимума.

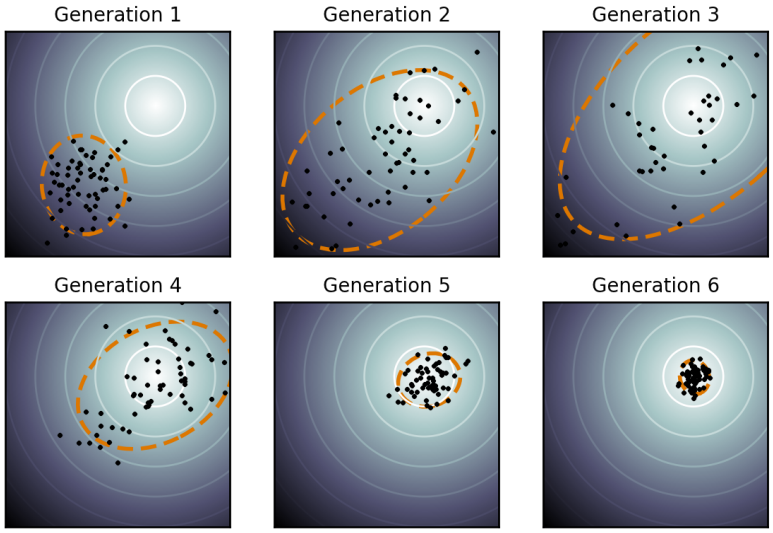


Рис. 4: Пример работы алгоритма CMA-ES для модельной двумерной задачи. Распределение точек постепенно сдвигается и стягивается в направлении оптимума. Источник: [21]

На рис. 4 показан пример работы алгоритма для двумерной задачи. В данном случае рассматривается лишь 2 гиперпараметра, их значения откладываются по координатным осям. В этом двумерном пространстве изображены концентрические линии уровня оптимизируемой функции, в их центре располагается оптимум. Точками изображены особи популяции – возможные решения задачи, пунктиром обозначено распределение, из которого сэмплируются новые конфигурации. Благодаря итеративным изменениям среднего и матрицы ковариации этого распределения, популяция эффективно сдвигается в сторону оптимума и затем стягивается вокруг него для нахождения как можно лучшего решения.

Описанные действия происходят на каждой последующей итерации. Процесс прекращается обычно по исчерпании лимита времени, оценок f или числа итераций. На практике CMA-ES демонстрирует высокую эффективность в различных задачах[22]. Однако он по-прежнему требует достаточного количества вычислений черного ящика на каждой итерации. У алгоритма довольно много параметров, позволяющих контролировать размер популяции, изменение шага, влияние прошлого и текущих поколений. Существуют различные рекомендации по их заданию[20]. Также

алгоритм в определенном смысле учитывает некоторую связь между поколениями, т. е. по сути историю вычислений, через формулу пересчета ковариационной матрицы.

3.3 Последовательные методы, основывающиеся на моделях

3.3.1 Байесовская оптимизация

Байесовская оптимизация[10][9] является ярким примером SMBO-алгоритма. На практике при оптимизации важно не только делать точечный прогноз, но и оценивать его неопределенность. Также в многомерных задачах вычисления целевой функции дорогие, и она может быть зашумленной. Возникает потребность снизить количество ее подсчетов и более разумно выбирать максимально перспективные регионы в пространстве поиска на основе оценки неопределенности, в которых могут находиться локальные или глобальные оптимумы. С этим хорошо справляется байесовская оптимизация – мощный итерационный алгоритм глобальной оптимизации «черных ящиков», в том числе и зашумленных. Согласно SMBO-модели, алгоритм состоит из двух компонент: вероятностной модели, приближающей целевую функцию, и функции выбора (функции выгоды, acquisition function), определяющей следующую точку для подсчета целевой функции. В качестве вероятностной модели в стандартной версии байесовской оптимизации используется гауссовский процесс, т. к. он является богатой вероятностной моделью с удобными свойствами и рассчетными формулами[23].

Алгоритм устроен следующим образом. Имеется выборка точек (инициализируемая обычно случайно), значения целевой функции $f : \mathbb{R}^n \to \mathbb{R}$ в которых нам известны. По ней обучается вероятностная модель для приближения f. Затем вычисляется функция выгоды для определения улучшений от проведенной оптимизации и выбора следующей точки. В выбранной точке вычисляется целевая функция, пара «точка-значение» добавляется в выборку, вероятностная модель перестраивается по новой выборке и процесс повторяется до выполнения критерия останова (обычно время выполнения или число итераций).

Рассмотрим алгоритм подробнее. Предполагается, что целевая функция $f(x) \sim \mathcal{GP}\left(m(x), k(x, x')\right)$, где $m(x)$, $k(x, x')$ – среднее и ковариационная функция гауссовского процесса $\mathcal{GP}$. Ковариационная функция задает форму траектории случайного процесса, среднее – его положение. Задав эти параметры, мы по сути задаем априорное распределение в пространстве функций. Для упрощения теоретических выкладок примем $m(x) \equiv 0$. На практике такой выбор оказывается целесообразным, т. к. информации в данных и свойств гауссовского процесса как правило оказывается достаточно, чтобы должным образом приблизить целевую функцию[23]. Также такой выбор среднего приемлем в силу отсутствия информации об истинной f. В случае произвольного $m(x)$, в т. ч. и для нестационарных процессов, соответствующие выкладки не будут сильно отличаться[23]. Итак, далее для предсказания нам необходимо апостериорное распределение значений f при условии $D_t = \{x_i, f(x_i)\}$ – выборки точек и известных значений f в них на итерации t. Пусть x_{t+1} – некоторая новая произвольная точка. По формуле Байеса:

$$p\left(f(x_{t+1})|D_t, x_{t+1}\right) = \frac{p\left(D_t, \{x_{t+1}, f(x_{t+1}\}\right)}{p(D_t)}.$$

В числителе и знаменателе – плотности многомерных нормальных распределений согласно определению гауссовского процесса. Таким образом, по известным формулам[23] можно получить:

$$p\left(f(x_{t+1})|D_t, x_{t+1}\right) = \mathcal{N}\left(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1})\right),$$

где

$$\mu_t(x_{t+1}) = k^T K^{-1} f_{1,t},$$

$$\sigma_t^2(x_{t+1}) = k(x_{t+1}, x_{t+1}) - k^T K^{-1} k,$$

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_t) \\ \vdots & \ddots & \vdots \\ k(x_t, x_1) & \cdots & k(x_t, x_t) \end{bmatrix},$$

$$k = \left(k(x_{t+1}, x_1), \cdots, k(x_{t+1}, x_t)\right)^T, \quad f_{1,t} = \left(f(x_1), \cdots, f(x_t)\right)^T.$$

Далее для обучения гауссовсокого процесса максимизируется правдоподобие $p\left(f(x_{t+1})|D_t, x_{t+1}\right)$ по параметрам. В общем случае это будут гиперпараметры ковариационной функции и функции среднего. Т. к. изначально было принято $m(x) \equiv 0$, то рассмотрим лишь параметры $k\left(x, x'\right)$.

Ковариационную функцию (или ядерную функцию) можно выбирать различными способами[24], и это определяет форму и гладкость траекторий гауссовсого процесса, а также позволяет вносить априорную информацию. Часто используемым вариантом является RBF-ядро:

$$k\left(x, x'\right) = A \exp\left(-\gamma||x - x'||^2\right) + \sigma^2.$$

A - параметр амплитуды, γ контролирует гладкость, σ^2 – шумовое слагаемое. При таком ядре изменения по всем размерностям равноправны. Для учета различий важности гиперпараметров можно сконструировать ядро следующего вида[23]:

$$k\left(x, x'\right) = \exp\left(-\frac{1}{2}\left(x - x'\right)^T diag\left(\theta\right)^{-2}\left(x - x'\right)\right).$$

В таком случае если, например, θ_i мало, то ядро не будет сильно зависеть от изменений i-ого параметра. Другой часто используемый вариант – ядро Матерна[25] с параметром гладкости ν:

$$k\left(x, x'\right) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{l}||x - x'||\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l}||x - x'||\right),$$

где K_ν – функция Бесселя порядка ν[26], Γ – гамма-функция, l – параметр длины. Наличие этих параметров позволяет получать множество разнообразных траекторий с различными свойствами.

После задания модели функции и ее неопределенности необходимо определить способ выбора новых точек для дальнейшего поиска оптимума. При этом есть потребность одновременно как уточнять места, где уже были наблюдения с низким значением целевой функции, т. е. там может быть оптимум, так и исследовать новые регионы с большой вероятностью нахождения минимума. В литературе две эти компоненты алгоритма называются exploitation и exploration соответственно, по сути

уточнение и разведка. Для соблюдения баланса между этими целями и, соответственно, более оптимального выбора следующей точки вводится т. н. функция выгоды $a(x)$(acquisition function). Ее устройство таково, что значения этой функции высоко, если значение f в точке потенциально мало, если велика неопределенность прогноза, или если оба условия выполнены одновременно в достаточной степени.

Введение функции выгоды $a(x)$ позволяет в какой-то степени заменить в общем случае нерешаемую точно задачу $x^* = \underset{x \in \mathcal{X}}{\operatorname{argmin}} f(x)$ на задачу $x_{t+1} = \underset{x \in \mathcal{X}}{\operatorname{argmax}} a(x)$. Она более простая, функция $a(x)$ вычисляется проще, чем f, т. к. в нее входят параметры гауссовского процесса. Также она может выбираться дифференцируемой, но часто при этом она имеет множество локальных оптимумов.

Существуют различные варианты выбора $a(x)$. Наиболее интуитивной стратегией является максимизировать вероятность улучшения по сравнению с лучшим текущим значением. В таком случае в качестве функции выгоды берется Probability of Improvement (PI):

$$PI(x) = \mathbb{P}\left(f(x) < f(x_{best})\right) = \Phi\left(\frac{f(x_{best}) - \mu(x)}{\sigma(x)}\right),$$

где x_{best} – текущая наилучшая точка, Φ – функция стандартного нормального распределения, $\mu(x)$, $\sigma^2(x)$ – среднее и дисперсия апостериорного распределения значений f. При такой функции точки, для которых с большой вероятностью $f(x)$ немного меньше, чем $f(x_{best})$, будут приоритетнее, чем точки с явно меньшим значением, но меньшей уверенностью, т. е. не учитывается фактор уточнения. Можно ввести настраиваемый параметр $\varepsilon \geq 0$:

$$PI(x) = \mathbb{P}\left(f(x) < f(x_{best}) - \varepsilon\right) = \Phi\left(\frac{f(x_{best}) - \mu(x) - \varepsilon}{\sigma(x)}\right).$$

Параметр ε можно задавать по-разному, контролируя соотношения между уточнением и разведкой[27][28].

Можно максимизировать и ожидаемую величину улучшения, используя Expected Improvemnt (EI):

$$EI(x) = \mathbb{E}\left(f(x) < f(x_{best})\right).$$

Данная функция в отличие от PI(x) не будет отдавать предпочтение малым, но достаточно вероятным улучшениям по сравнению с более крупными. EI(x) также является довольно логичным и интуитивным выбором, часто применяется на практике. Помимо рассмотренных функций существуют и другие приемы, позволяющие выбирать между уточнением и разведкой[29].

В конечном счете для выбора следующей точки-кандидата проводится максимизация $a(x)$. Эта задача может быть решена различными способами и с небольшими затратами, в том числе и градиентными методами. Выбрав x_{t+1}, вычисляется значение $f(x_{t+1})$, пара $\{x_{t+1}, f(x_{t+1})\}$ добавляется в выборку, и процесс повторяется: обучается суррогатная модель, ищется максимум функции выгоды, пополняется обучающая выборка, и так до завершения. Иллюстрацию работы можно увидеть на рис. 5.

Байесовская оптимизация с гауссовскими процессами является эффективным методом оптимизации и успешно применяется во многих областях машинного обучения[30]. Метод требует меньшее количество вычислений исходного черного ящика по сравнению с рассмотренными ранее алгоритмами, работает с более простой суррогатной моделью, однако имеет кубическую сложность от объема выборки из-за обращения матрицы K на каждой итерации, а также выполняется последовательно.

Ко всему прочему алгоритм имеет одну особенность: используемые в нем гауссовские процессы плохо работают с категориальными переменными. На практике же они часто встречаются в моделях машинного обучения одновременно с непрерывными вещественными гиперпараметрами. В различных модификациях байесовской оптимизации данная проблема часто решается использованием другой суррогатной модели.

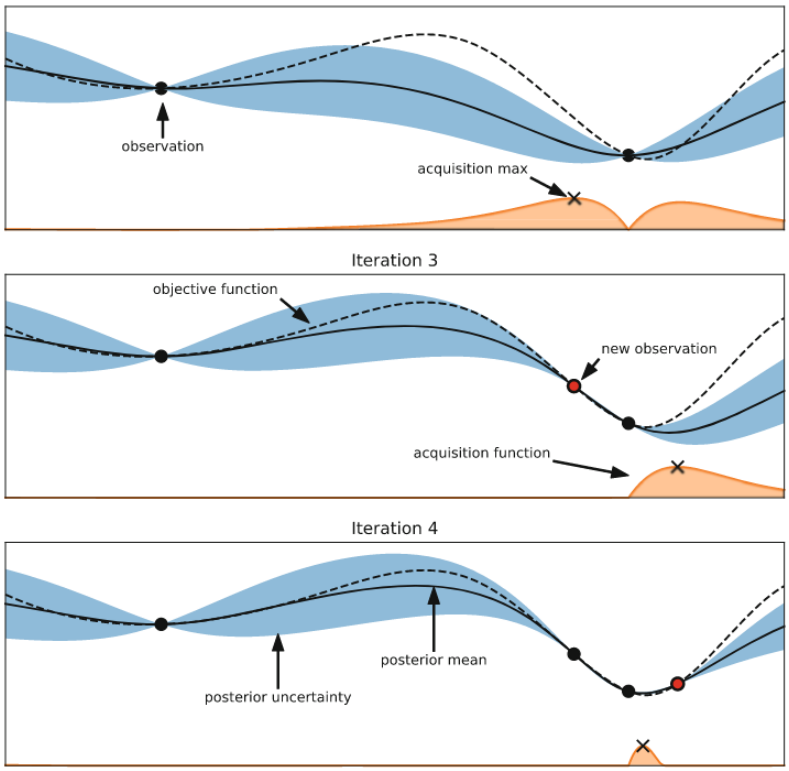


Рис. 5: Иллюстрация из [1], демонстрация работы алгоритма байесовской оптимизации. Сплошная линия - траектория гауссовского процесса, пунктирная - целевая функция. Итеративно обучая вероятностную модель и пополняя выборку при помощи функции выгоды, алгоритм ищет оптимум и при этом стремится рассмотреть новые регионы пространства, уменьшить неопределенность прогноза.

Однако можно адаптировать ядерную функцию для смеси вещественных и категориальных гиперпараметров, например, как предлагается в данной статье[31]:

$$k_{mixed}\left(x, x'\right) = \exp\left(\sum_{l \in \mathcal{P}_{cont}} \left(-\lambda_l (x_l - x'_l)^2\right)\right) + \sum_{l \in \mathcal{P}_{cat}} \left(-\lambda_l \left(1 - \delta\left(x_l, x'_l\right)\right)\right),$$

где $\mathcal{P}_{cont}$ – множество непрерывных переменных, $\mathcal{P}_{cat}$ – множество категориальных переменных, λ_l – параметры ядра, $\delta\left(x_l, x'_l\right) = 1$, если $x_l = x'_l$, иначе $\delta(x_l, x'_l) = 0$. Это модификация RBF-ядра, где для категориальных переменных вместо квадрата нормы используется растояние Хэмминга. В статье доказывается, что данная функция является ядром.

В одной из модификаций байесовской оптимизации – алгоритме Sequential Model-based Algorithm Configuration (SMAC)[31], обозначенная проблема решается заменой гауссовского процесса на случайный лес для регрессии, который может работать с обоими типами переменных. По аналогии со стандартным алгоритмом случайный лес обучается на имеющейся выборке, вычисляются эмпирические среднее и дисперсия прогнозов, с их помощью ищется максимум функции выгоды и происходит пополнение выборки. В качестве функции выбора используется Expected Improvement. Большинство альтернативных вариантов, пригодных для стандартного алгоритма, непригодны для SMAC, т. к. они во многом используют свойства гауссовских процессов. Также в статье утверждается, что алгоритм может учитывать взаимосвязи между гиперпараметрами и решать многокритериальные задачи оптимизации[31].

Существуют и другие модернизации алгоритма, с использованием иных суррогатных моделей или с возможностью производить параллельное выполнение[1][10][29].

3.3.2 Tree-structured Parzen Estimator

Алгоритм Tree-structured Parzen Estimator (TPE)[32] использует те же принципы, что и байесовская оптимизация, однако вместо апостериорного распределения $p(f|x)$ моделируется $p(x|f)$. Строятся две различные модели:

$$p(x|f) = \begin{cases} \ell(x), & f < f^* \\ g(x), & f \geq f^* \end{cases}.$$

В свою очередь плотности $\ell(x)$, $g(x)$ строятся по точкам x таким, что $f(x)$ меньше или больше либо равно, чем f^* соотвественно. f^* подбирается так, чтобы оно было γ-квантилем известных значений f: $\mathbb{P}(f < f^*) = \gamma$. γ по сути является гиперпараметром алгоритма TPE, на практике как правило берется $\gamma = 0.25$. $\ell(x)$ и $g(x)$ строятся каждая как смеси гауссиан с центрами в точках из разделенных по квантилю множеств. Такое моделирование базируется на интуитивном предположении о том, что перспективнее выбирать следующую точку из тех, которые соответствуют

низким значениям f. Оно подтверждается при максимизации функции выгоды, в TPE это Expected Improvement. Согласно[32]:

$$EI_{f^*}(x) \propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1}.$$

Таким образом, для максимизации $EI_{f^*}(x)$ стоит брать точки из $\ell(x)$ с большей вероятностью, чем из $g(x)$, т. е. максимизируя $\frac{\ell(x)}{g(x)}$.

TPE превосходит байесовскую оптимизацию с гауссовскими процессами[32] и в целом показывает неплохие результаты в различных экспериментах[32][33]. Однако алгорим не может в полной мере моделировать совместные распределения, т. к. предполагается независимость точек из $\ell(x)$ и $g(x)$. Его реализации присутствуют во многих пакетах для оптимизации гиперпараметров. Существуют и модификации алгоритма, например, для многокритериальных задач[34].

4 Вычислительные эксперименты

Проведем серию экспериментов для сравнения наиболее популярных из описанных в предыдущем разделе методов оптимизации гиперпараметров. Рассмотрим их применительно к различным типам задач, данных и алгоритмов. Предметом исследования является время работы, номер итерации, на которой была найдена лучшая конфигурация гиперпараметров (т. е. как быстро алгоритм находит лучшее решение), качество модели на ней. Все эксперименты и вычисления проведены с использованем языка программирования Python.

4.1 Используемые данные

- Breast Cancer Wisconsin – данные об опухолях молочной железы, разделяющихся на класс злокачественных и доброкачественных. Число объектов - 569, число признаков - 30.
- Pima Indians Diabetes – данные пациентов для диагностики диабета. Присутствуют два класса - болен/здоров. Содержит 768 объектов, описывающихся 9 признаками.
- Ionosphere – данные с радаров, разделенные на два класса – несущие информацию о структуре ионосферы или нет. Сигналы описываются 34 признаками, число объектов - 350.
- California Housing Price – данные о недвижимости, целевая переменная - стоимость жилья. Содержит 20640 строк, 9 признаков.
- CIFAR-10 – цветные изображения размером 32×32, разделенные на 10 классов. Количество изображений - 60000.
- MNIST – черно-белые изображения рукописных цифр размером 28×28. Содержит 70000 объектов, разделенных на 10 классов.

Перечисленные данные, кроме CIFAR-10[35] и MNIST[36], взяты из UCI Machine Learning Repository[37]. На датасете California Housing Prices решается задача регрессии, на остальных данных – задача классификации.

4.2 Рассмотренные методы оптимизации гиперпараметров

Были рассмотрены следующие алгоритмы оптимизации гиперпараметров:

- Поиск по сетке
- Случайный поиск
- Генетический алгоритм NSGA-II
- TPE
- CMA-ES
- Байесовская оптимизация с гауссовским процессом

Реализация байесовской оптимизации использовалась из пакета `scikit-optimize`[38], реализации остальных методов – из пакета `optuna`[39]. Каждый метод оптимизации гиперпараметров применялся к каждой из рассмотренных моделей машинного обучения. Алгоритмы тестируются в стандартной реализации, без модернизаций и при одинаковом числе итераций для общности сравнения и рассмотрения их характерных особенностей. При задании ограничения на время выполнения в определенных случаях некоторые алгоритмы, например, байесовская оптимизация с гауссовским процессом, не успели бы сделать достаточное число шагов для анализа их поведения в целом. Число итераций бралось равным количеству всевозможных комбинаций в поиске по сетке, чтобы все методы рассмотрели одинаковое число точек в пространстве поиска. Это позволяет понять, как выбираются комбинации гиперпараметров на каждой итерации и насколько эффективно, оценить характер и скорость приближения к оптимуму, а также покрытие конфигурационного пространства. Стоит также отметить, что при рассмотрении меньшего числа итераций или с ограничениями по времени результаты некоторых методов были достаточно нестабильными в силу наличия в них случайности.

4.3 Рассмотренные модели машинного обучения

- Метод опорных векторов (SVM) для классификации с RBF-ядром из библиотеки `scikit-learn`. Настраиваемые гиперпараметры – коэффициент регуляризации C, параметр ядра γ. Исследовался на данных Breast Cancer Wisconsin и Pima Indians Diabetes. Значения C и γ рассматривались из логарифмически равномерного распределения на $[10^{-5}, 10^5]$. Разделение на обучающую и тестовую выборку – в соотношении 7/3. При подборе гиперпараметров использовалась кросс-валидация по 5 фолдам. Данное число фолдов является распространенной практической рекомендацией, и эмпирически было показано, что оно является оптимальным с точки зрения баланса смещения и разброса[40].

- Логистическая регрессия с регуляризацией `elastic-net` из `scikit-learn`. Оптимизируемые параметры – коэффициент регуляризации C и коэффициент при компоненте L1 регуляризации ℓ_1 (`l1_ratio`). Рассмотренные данные – датасет Ionosphere. Значения C брались из логарифмически равномерного распределения на $[10^{-4}, 10^4]$, значения ℓ_1 – из равномерного распределения на $[0, 1]$. Разбиение на обучение и тест – 7/3, при подборе гиперпараметров использовалась кросс-валидация по 5 фолдам.

- Градиентный бустинг для регрессии из библиотеки `CatBoost`[41]. Настраиваемые параметры – число деревьев, их максимальная глубина, темп обучения, коэффициент регуляризации функции потерь (`l2_leaf_reg`). Минимизируемая функция ошибки – RMSE. Данные – датасет California Housing Prices. Число деревьев перебиралось из списка значений $5, 10, 50, 100, 200$, максимальная глубина – из диапазона $[3, 13]$, коэффициент регуляризации – из отрезка $[1, 9]$, темп обучения – из отрезка $[10^{-3}, 1]$ по логарифмической шкале. Разбиение на обучение/тест/валидацию – 49/21/30.

- Сверточная сеть архитектуры ResNet18, реализация выполнена с использованием библиотеки `pytorch`. Гиперпараметры – тип функции активации (ReLU или LeakyReLU), тип оптимизатора (SGD или Adam) и темп обучения (из логарифмически равномерного распределения на $[10^{-3}, 10^{-1}]$). Рассматривалась сеть на датасете CIFAR-10. Разбиение обучение/валидация/тест – в соотношении 10/1/1. Количество эпох обучения – 30. Обучение проводилось на GPU Nvidia Tesla P100.

- Многослойный перцептрон из `scikit-learn`. Рассматриваемые параметры – число скрытых слоев (1 или 2) и число нейронов в них (25, 50, 100), тип функции активации (ReLU или гиперболический тангенс), начальное значение темпа обучения (из логарифмически равномерного распределения на $[10^{-3}, 1]$) и его тип (коснтантный или адаптивный, уменьшающийся в 5 раз в случае, если потери не уменьшаются на протяжении двух эпох), число эпох обучения (10 или 20). Данные для исследования – датасет MNIST, разделение данных на обучающие и тестовые – в соотношении 6/1. При подборе гиперпараметров применялась кросс-валидация по 5 фолдам.

Для всех моделей, кроме ResNet18, обучение проходило на CPU. В случае задач классификации максимизировалась точность на валидации, в случае регрессии происходила минимизация ошибки RMSE. Для метода опорных векторов и логистической регрессии, в силу небольших размеров датасетов, для оценки точности модели с найденными гиперпараметрами на тестовых данных обучение проводится 5 раз, и рассматривается средняя точность. Стоит отметить, что в случае градиентного бустинга, сверточной сети и многослойного перцептрона пространство поиска состоит из смеси вещественных, дискретных и категориальных гиперпараметров.

4.4 Результаты экспериментов и анализ

Результаты серии экспериментов с обозначенными алгоритмами и моделями приведены в следующих таблицах.

	Поиск по сетке	Случайный поиск	NSGA-II	TPE	CMA-ES	Байесовская оптимизация
Время работы	34.6 с	32.1 с	30.7 с	27.6 с	**25.6 с**	**2 ч 15 мин 58 с**
Номер лучшей конфигурации	247	160	334	185	**41**	58
Точность лучшей конфигурации, валидация	0.959	0.957	0.959	0.959	0.959	0.959
Средняя точность лучшей конфигурации, тест	0.941	**0.947**	**0.947**	0.941	0.941	0.935

Таблица 1: Метод опорных векторов для классификации, датасет Breast Cancer Wisconsin, 400 итераций для каждого алгоритма. Оптимизируемые параметры: C, γ.

	Поиск по сетке	Случайный поиск	NSGA-II	TPE	CMA-ES	Байесовская оптимизация
Время работы	1 мин 11 с	1 мин 2 с	1 мин 9 с	52.2 с	**35.2** с	**2 ч 12 мин 45 с**
Номер лучшей конфигурации	239	268	**41**	349	183	319
Точность лучшей конфигурации, валидация	0.774	0.782	0.78	0.782	0.782	0.782
Средняя точность лучшей конфигурации, тест	0.722	0.731	0.718	0.731	**0.732**	0.731

Таблица 2: Метод опорных векторов для классификации, датасет Pima Indians Diabetes, 400 итераций для каждого алгоритма. Оптимизируемые параметры: C, γ.

	Поиск по сетке	Случайный поиск	NSGA-II	TPE	CMA-ES	Байесовская оптимизация
Время работы	45.9 с	37.1 с	**32.4** с	33.1 с	33.4 с	**2 ч 16 мин 31 с**
Номер лучшей конфигурации	6	6	250	108	**2**	18
Точность лучшей конфигурации, валидация	0.889	0.897	0.893	0.897	0.889	0.893
Средняя точность лучшей конфигурации, тест	0.857	0.847	0.838	0.847	**0.867**	0.838

Таблица 3: Логистическая регрессия, датасет Ionosphere, 400 итераций для каждого алгоритма. Оптимизируемые параметры: C, ℓ_1.

	Поиск по сетке	Случайный поиск	NSGA-II	TPE	CMA-ES	Байесовская оптимизация
Число итераций	1250	1250	1250	1250	1250	300
Время работы	39 мин 12 с	55 мин 31 с	54 мин 39 с	58 мин 31 с	**7 мин 28 с**	**1 ч 49 мин 42 с**
Номер лучшей конфигурации	**277**	422	527	571	304	280
RMSE лучшей конфигурации, валидация	0.461	0.463	0.446	0.457	0.461	0.46
RMSE лучшей конфигурации, тест	0.459	0.452	0.456	**0.45**	0.455	0.457

Таблица 4: Градиентный бустинг для регрессии, датасет California Housing Prices. Оптимизируемые параметры: число деревьев, их максимальная глубина, темп обучения, коффициент регуляризации функции потерь.

	Поиск по сетке	Случайный поиск	NSGA-II	TPE	CMA-ES	Байесовская оптимизация
Время работы	4 ч 17 мин 36 с	4 ч 17 мин 52 с	4 ч 29 мин 32 с	4 ч 18 мин 16 с	4ч 14 мин 18 с	4 ч 15 мин 10 с
Номер лучшей конфигурации	6	3	5	12	**2**	12
Точность лучшей конфигурации, валидация	0.845	0.843	0.814	0.841	0.844	0.843
Точность лучшей конфигурации, тест	0.836	**0.843**	0.798	0.828	0.84	0.841

Таблица 5: ResNet18, датасет CIFAR-10, 12 итераций для каждого алгоритма. Оптимизируемые параметры - тип функции активации, выбор оптимизатора и темп обучения.

	Поиск по сетке	Случайный поиск	NSGA-II	TPE	CMA-ES	Байесовская оптимизация
Время работы	**3 ч 24 мин 23** с	4 ч 59 мин 23 с	5 ч 44 мин 22 с	5 ч 26 мин 28 с	5 ч 39 мин 26 с	**8 ч 42 мин 10 с**
Номер лучшей конфигурации	108	94	101	171	**39**	75
Точность лучшей конфигурации, валидация	0.974	0.973	0.974	0.975	0.973	0.976
Точность лучшей конфигурации, тест	0.976	0.974	0.977	0.975	0.976	**0.978**

Таблица 6: Многослойный перцептрон, датасет MNIST, 192 итерации для каждого алгоритма. Оптимизируемые параметры - число скрытых слоев и число нейронов в них, тип функции активации и темпа обучения, начальное значение темпа обучения, число эпох обучения.

На основе полученных результатов проведем сравнительный анализ. Поиск по сетке, несмотря на примитивность, в сравнении с более продвинутыми алгоритмами демонстрирует близкие к ним результаты. В задачах классификации точность моделей с найденными им гиперпараметрами достаточно высока, в некоторых случаях она является одной из лучших (табл. 3, табл. 6). Однако в задаче регрессии для градиентного бустинга (табл. 4) полученное решение приводит к худшему значению RMSE. Алгоритм работает достаточно долго, рассматривает большое число

точек до достижения наилучшей, хотя в некоторых случаях оптимальная конфигурация находится довольно быстро (табл. 3). Во многом это зависит от задачи и от заданной сетки значений гиперпараметров. Метод способен достигать качественных результатов и в случае более сложных моделей, данных и пространств поиска (табл. 5, табл. 6), однако он будет вычислительно неэффективен и сильно ограничен фиксированными значениями гиперпараметров для оценки.

Случайный поиск близок к поиску по сетке во многих экспериментах, однако имеет более нестабильные результаты в силу случайности. В экспериментах для SVM (табл. 1) и ResNet18 (табл. 5) метод нашел гиперпараметры, соответствующие наилучшей точности на тестовой выборке, и за сравнительно небольшое число итераций. Однако в случае с градиентным бустингом (табл. 4) его время работы заметно выше, чем у того же поиска по сетке с тем же числом итераций. Вероятно на каком-то шаге им были просэмплированы значения гиперпараметров, приводящие к долгому обучению (например, большая максимальная глубина деревьев). В целом получаемые алгоритмом результаты нестабильны, что неоднократно наблюдалось в ходе проведения экспериментов, время его работы, номер лучшей найденной точки и соответствующее качество на тесте сильно варьировались.

NSGA-II в целом схож со случайным поиском, как по времени работы, так и по итоговому качеству. Его результаты также нестабильны в силу наличия случайности на стадии мутаций, просматривается много конфигураций до достижения лучшей. В некоторых случаях, например, для ResNet18 (табл. 5) или SVM (табл. 2), качество с найденными параметрами оказывается значительно хуже, чем для остальных методов, но при этом в экспериментах с многослойным перцептроном (табл. 6) его результаты – одни из лучших. Также присутствие в нем сортировки особей популяции может заметно увеличивать время работы.

CMA-ES можно назвать наилучшим среди алгоритмов по совокупности итогового качества и времени работы. Он одинаково хорошо работает с пространствами и моделями разной сложности. Например, для градиентного бустинга (табл. 4) CMA-ES имеет рекордно низкое время работы и хорошее качество на тесте и валидации, а в экспериментах с ResNet18 (табл. 5) он часто быстрее всего находил наилучшую

конфигурацию. В нем также есть стадия сортировки решений, но она выполняется не для всей популяции, а лишь для ее части, что увеличивает время работы незначительно.

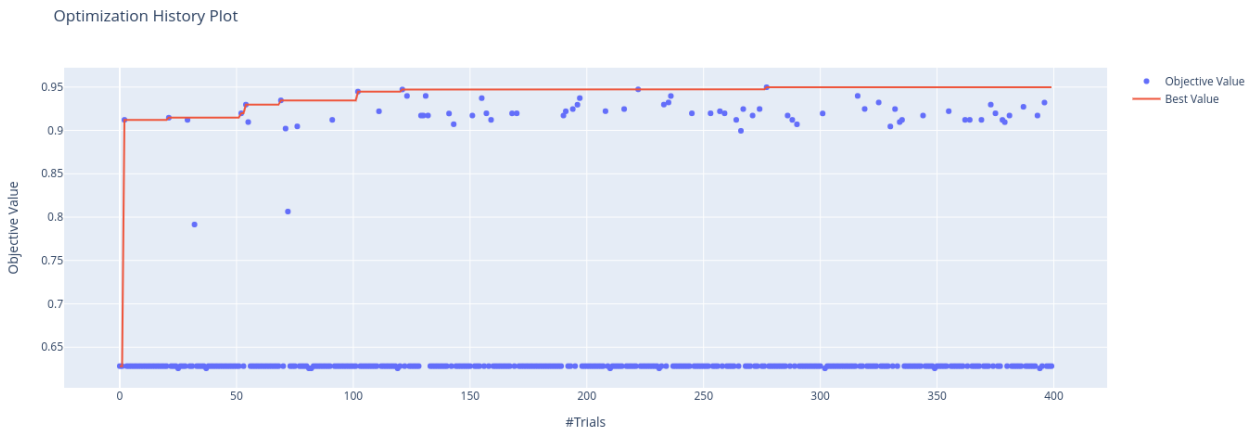
Байесовская оптимизация с гауссовским процессом имеет очень большое время работы в силу последовательного выполнения и свойств гауссовского процесса. В экспериментах для градиентного бустинга (табл. 4) было сокращено число ее итераций до 300 с целью экономии времени. Однако хорошие решения находятся методом за малое число итераций, и их итогове качество является одним из самых высоких. Хоть алгоритм и требует меньше дорогостоящих вычислений целевой функции, в силу последовательности он все равно очень долго работает. Однако в случае более сложных моделей и данных его время работы сопоставимо с остальными методами, что можно наблюдать в экспериментах с ResNet18 (табл. 5). При этом и получаемое качество достаточно высокое.

Алгоритм TPE работает значительно быстрее классической байесовской оптимизации, т. к. вместо последовательного обучения гауссовского процесса смеси гауссиан $l(x)$ и $g(x)$ строятся по сути параллельно. Время его работы сравнимо со случайным поиском и поиском по сетке, а получаемые им значения гиперпараметров оказываются весьма эффективными.

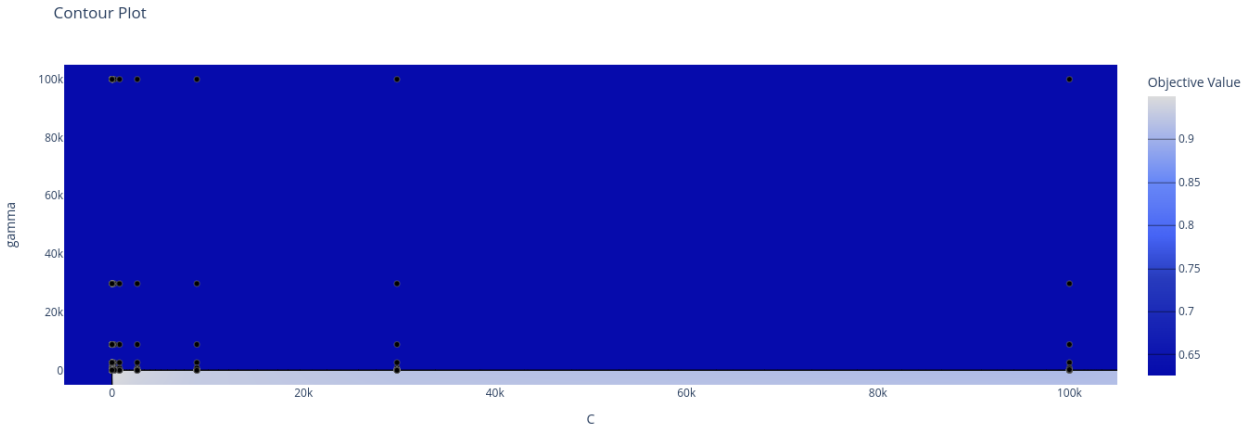
Можно сказать, что результаты рассмотренных алгоритмов различаются не очень сильно, и качество моделей с найденными гиперпараметрами на тестовой выборке является довольно высоким. Стоит отметить, что для получаемых конфигураций параметров не наблюдалось эффекта переобучения, точность и RMSE лучших решений на валидации и тесте довольно близки. Схожие результаты получались и при повторных проведениях данных экспериментов.

Для еще более подробного анализа алгоритмов рассмотрим, как меняется точность на валидации с каждым шагом при оценке новых конфигураций, и на сколько она будет отклоняться от лучшего на данный момент значения, т. е. насколько эффективно метод исследует множество решений. Также рассмотрим, как располагаются выбираемые методом решения в пространстве поиска, как они покрывают его регионы и как близко они подходят к области оптимума целевой функции.

Для наглядности рассмотрим соответствующие графики для экспериментов с методом опорных векторов, где конфигурационное пространство двумерно и признаки вещественны (оси значений гиперпараметров C и γ).



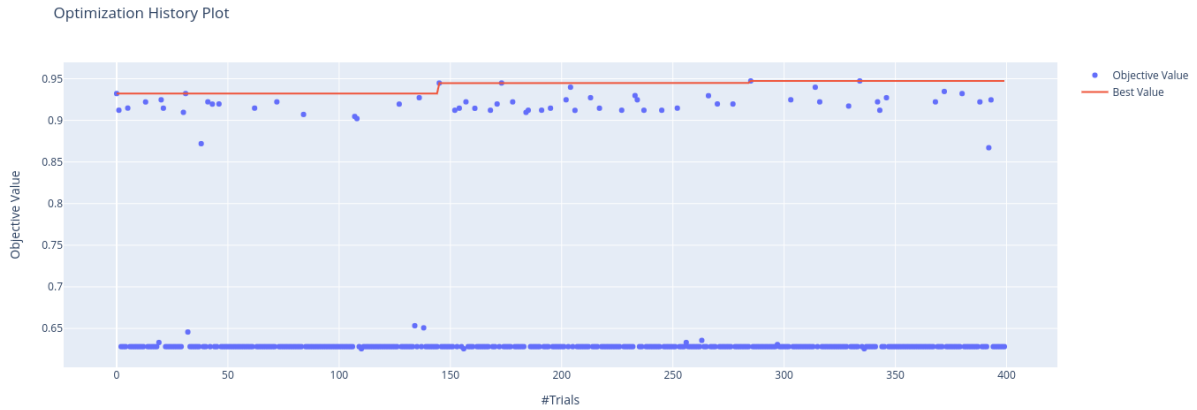
(a) Зависимость качества модели на валидации от номера итерации алгоритма.



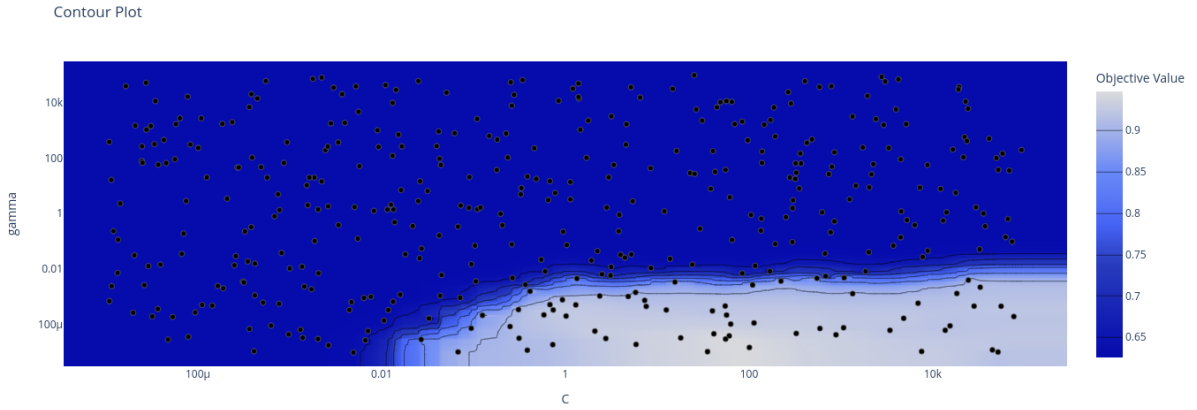
(b) Рассматриваемые алгоритмом конфигурации гиперпараметров (черные точки) в осях значений C и γ и соответствующие значения целевой функции (отмечены цветом).

Рис. 6: Поиск по сетке для SVM, датасет Breast Cancer Wisconsin.

Начнем с поиска по сетке. На рис. 6a видно, что хоть и точка с довольно высоким качеством находится достаточно рано, до следующего улучшения алгоритм рассмотрит очень много неперспективных кандидатов с низкой точностью. Из рис. 6b видно, что сетка задает очень редкое и неэффективное покрытие пространства, однако в ее узлах все же оказались точки, попавшие в область с высоким значением точности.



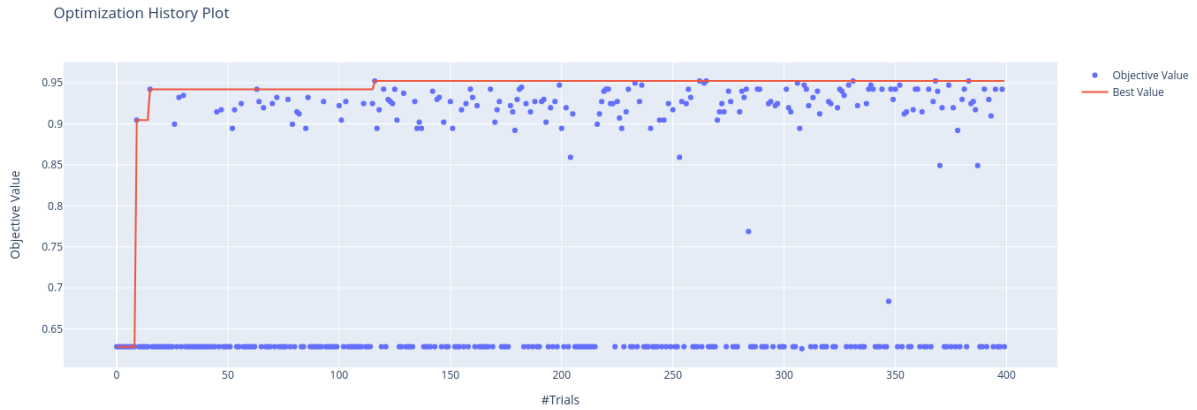
(a) Зависимость качества модели на валидации от номера итерации алгоритма.



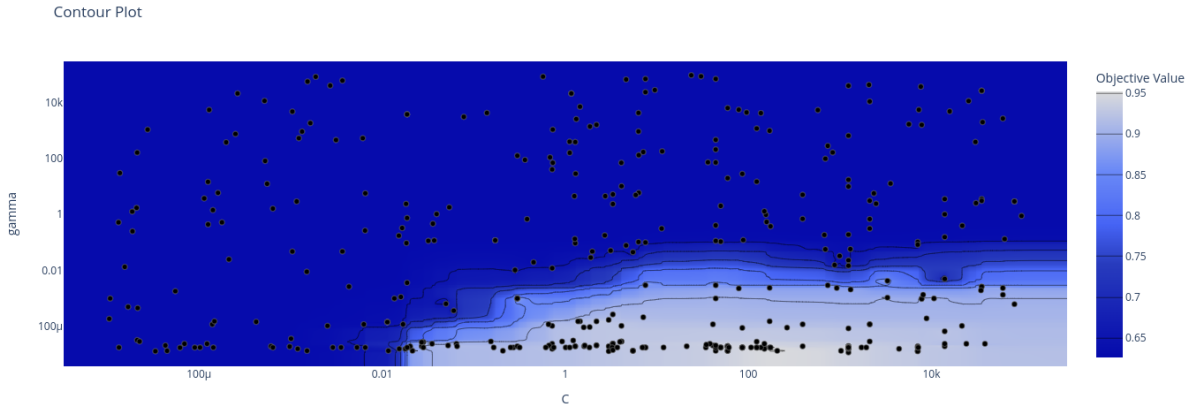
(b) Рассматриваемые алгоритмом конфигурации гиперпараметров (черные точки) в осях значений C и γ и соответствующие значения целевой функции (отмечены цветом).

Рис. 7: Случайный поиск для SVM, датасет Breast Cancer Wisconsin.

Случайный поиск с самого начала выбрал точки с высоким значением точности (рис. 7a), однако он так же, как и поиск по сетке, в процессе нахождения улучшения вычисляет целевую функцию в множестве неперспективных точек. При этом сэмплируемые им конфигурации покрывают почти все пространство значений гиперпараметров (рис. 7b), что повышает вероятность нахождения оптимума. Однако в данном случае подавляющее большинство рассмотренных точек находится в областях, соответствующих низкой точности.



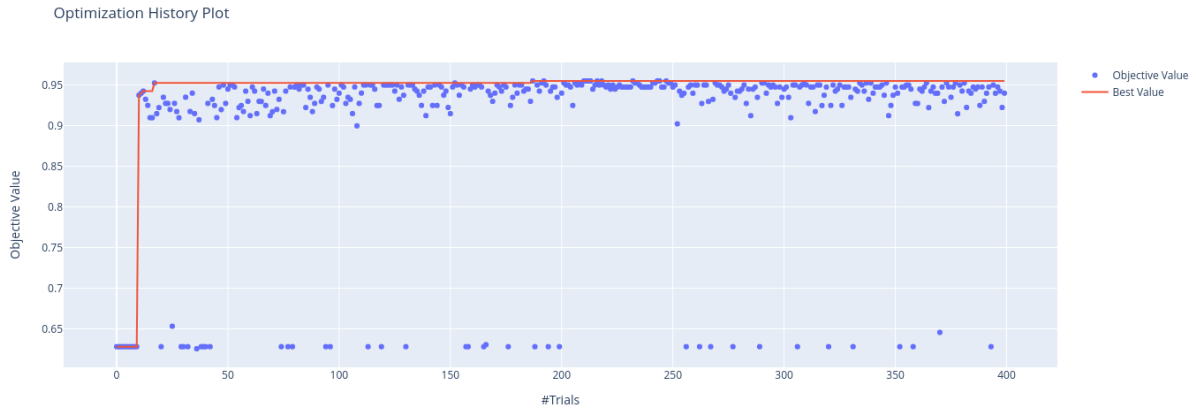
(a) Зависимость качества модели на валидации от номера итерации алгоритма.



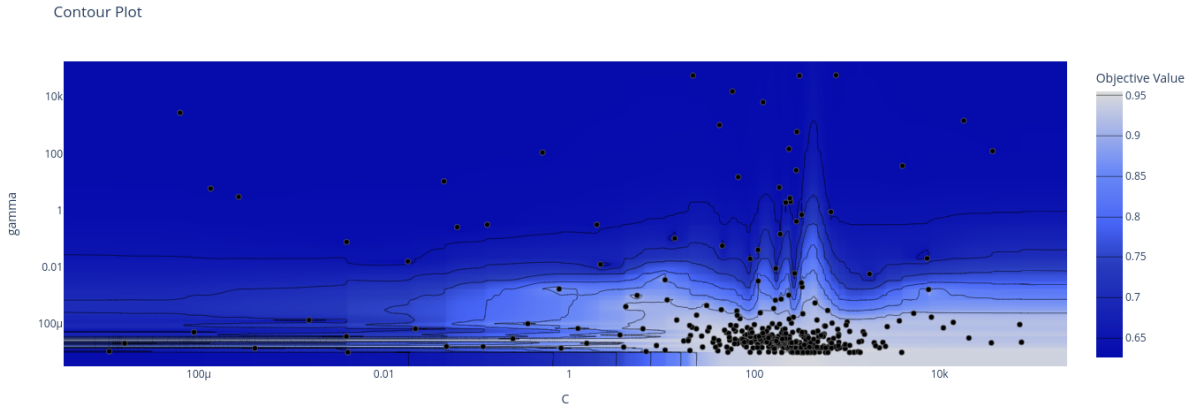
(b) Рассматриваемые алгоритмом конфигурации гиперпараметров (черные точки) в осях значений C и γ и соответствующие значения целевой функции (отмечены цветом).

Рис. 8: Генетический алгоритм NSGA-II для SVM, датасет Breast Cancer Wisconsin.

Для NSGA-II картина схожа со случайным поиском, но выбираемые решения распределены более упорядоченно, небольшими вытянутыми скоплениями, и их плотность растет по мере приближения к области оптимума. Также на рис. 8a можно заметить, что точки сэмплируются эффективнее, чем для предыдущих двух методов, рассматривается большее число перспективных комбинаций с высоким значением оптимизируемой точности. Таким образом, используемая в методе концепция элитизма действительно способствует лучшей сходимости к оптимуму. Но все же рассматриваемые комбинации распределены по пространству также достаточно хаотично.



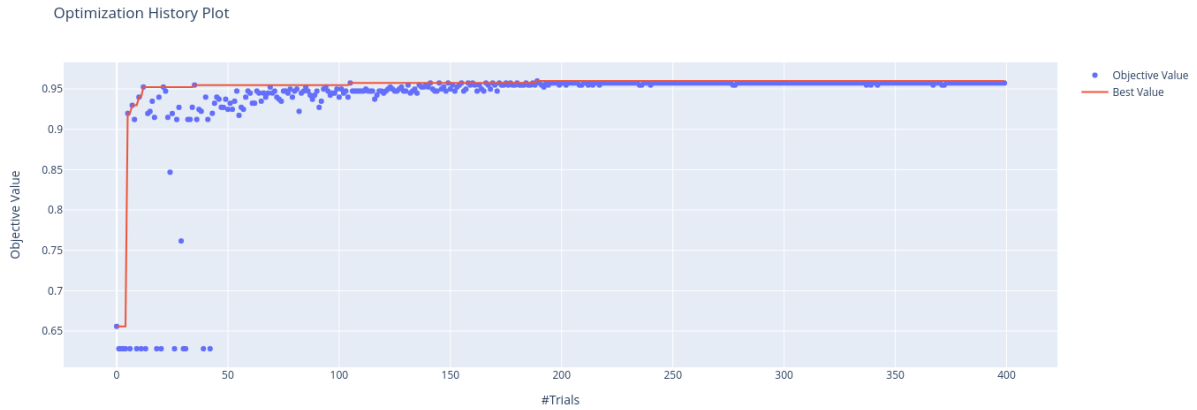
(a) Зависимость качества модели на валидации от номера итерации алгоритма.



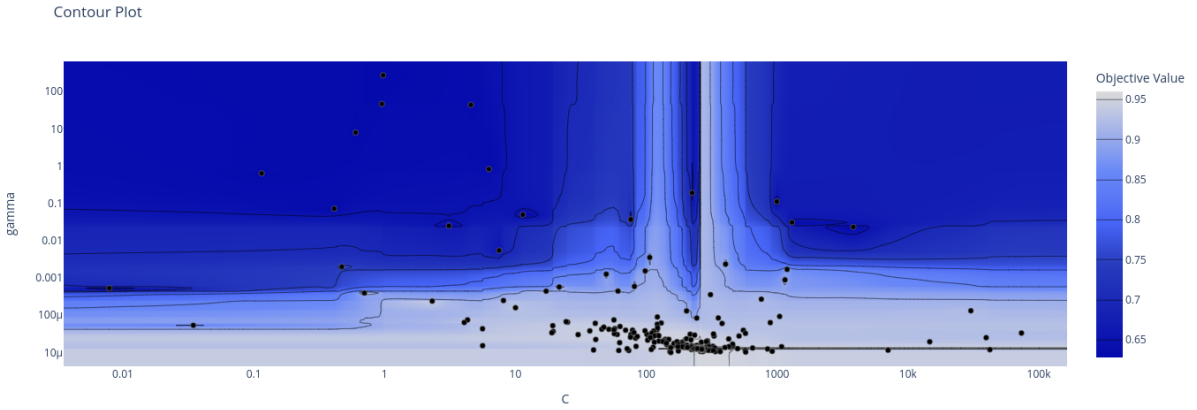
(b) Рассматриваемые алгоритмом конфигурации гиперпараметров (черные точки) в осях значений C и γ и соответствующие значения целевой функции (отмечены цветом).

Рис. 9: TPE для SVM, датасет Breast Cancer Wisconsin.

Куда лучше обстоит ситуация для алгоритма TPE. Разбивая на каждой стадии точки по квантилю значений целевой функции, он преимущественно сэмплирует новые конфигурации из области с большой вероятностью улучшения качества. Действительно, на рис. 9a видно, что постепенно находя все лучшие решения, алгоритм практически не будет рассматривать невыгодных кандидатов с меньшей точностью. На рис. 9b видно, как выбираемые TPE точки агрегируются в области с высоким значением целевой функции, что улучшает процесс поиска максимума.



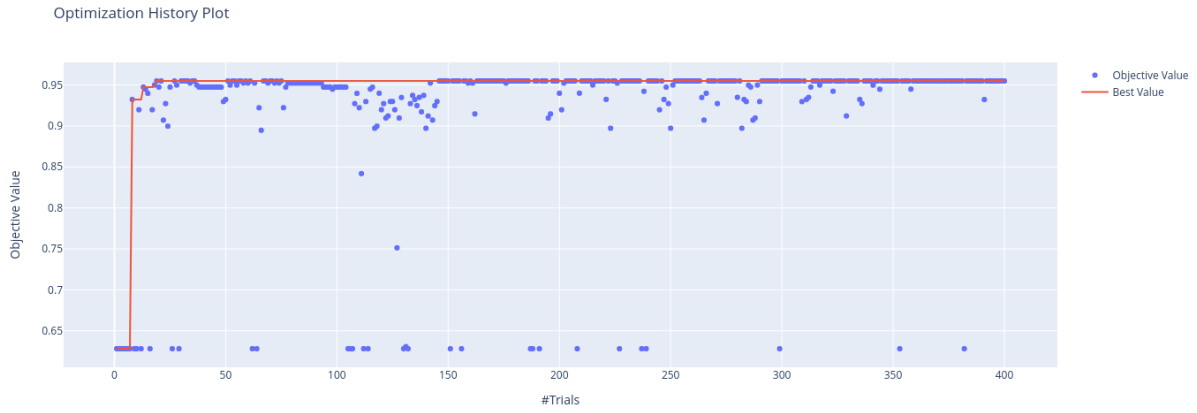
(a) Зависимость качества модели на валидации от номера итерации алгоритма.



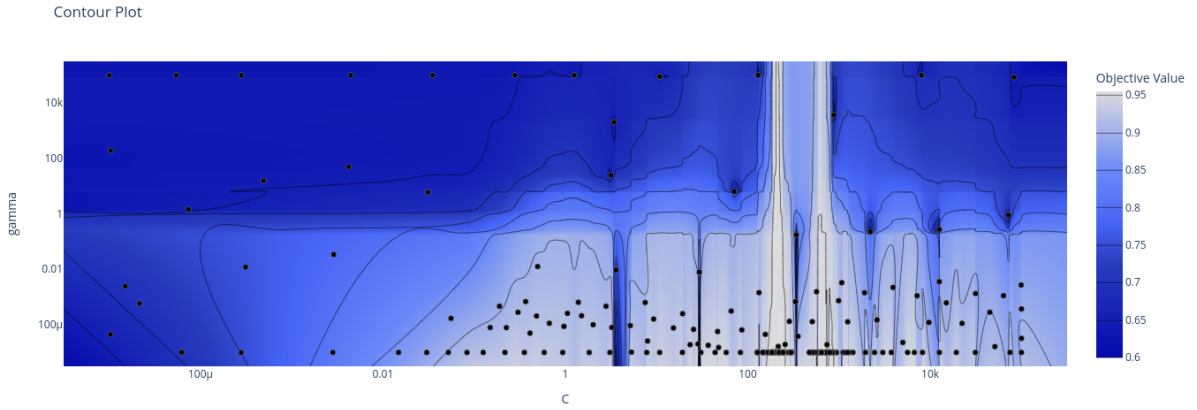
(b) Рассматриваемые алгоритмом конфигурации гиперпараметров (черные точки) в осях значений C и γ и соответствующие значения целевой функции (отмечены цветом).

Рис. 10: CMA-ES для SVM, датасет Breast Cancer Wisconsin.

Еще лучше просматриваемые решения агрегируются в области оптимума в случае алгоритма CMA-ES, что можно наблюдать на рис. 10a, рис. 10b. Разброс решений относительно лучшего качества сужается с числом итераций, и облако точек хорошо концентрируется в районе максимума функции. Это наглядно показывает эффективность идеи и принципа работы алгоритма, заключающегося в последовательном смещении и деформации распределения популяции решений в сторону оптимума.



(a) Зависимость качества модели на валидации от номера итерации алгоритма.



(b) Рассматриваемые алгоритмом конфигурации гиперпараметров (черные точки) в осях значений C и γ и соответствующие значения целевой функции (отмечены цветом).

Рис. 11: Байесовская оптимизация с гауссовскими процессами для SVM, датасет Breast Cancer Wisconsin.

Для байесовской оптимизации с гауссовским процессом точки также постепенно выбираются все ближе к оптимуму, благодаря применению функции выгоды. Причем действительно соблюден баланс между разведкой и уточнением: были просмотрены различные регионы всего пространства, но плотность точек растет по мере приближения к максимуму (рис. 11b). То же можно сказать и о TPE как о модификации байесовской оптимизации. Однако там составляющая разведки удовлетворяется в меньшей степени в силу наличия более приоритетного для сэмплирования распределения $l(x)$. Стоит отметить своеобразное расположение рассмотренных точек: по «периметру» и «радиально» к точке максимума.

Приведенные графики характерны для различных случаев оптимизации гиперпараметров и хорошо отражают особенности, механизмы и поведение рассмотренных алгоритмов. В остальных проведенных экспериментах ситуация схожа, но менее наглядна в силу наличия более 2 измерений. Иллюстрации, данные и код реализации всех проведенных экспериментов доступны в репозитории[42].

Итак, рассмотрев принципы и устройство наиболее распространенных алгоритмов оптимизации гиперпараметров, проведя эксперименты и анализ их результатов, можно выделить ключевые достоинства и недостатки каждого из них:

- **Поиск по сетке**: простой в реализации алгоритм, хорошо работает при небольшом количестве гиперпараметров, однако становится неэффективен с ростом их числа и требует множества дорогостоящих вычислений целевой функции.
- **Случайный поиск**: также прост, способен рассматривать большое число регионов пространства поиска в силу случайности, однако не гарантирует сходимость и аналогично требует многократных вычислений черного ящика, имеет нестабильное поведение.
- **NSGA-II**: в определенном смысле схож со случайным поиском, но использование концепции элитизма позволяет ему генерировать более перспективные решения на каждой итерации и улучшить сходимость популяции к оптимуму.
- **CMA-ES**: мощный и эффективный алгоритм, может хорошо локализовывать оптимальные регионы, но требует возможно и не такого большого, как для предыдущих двух алгоритмов, но все же достаточного количества вычислений оптимизируемой функции на стадии селекции. Также имеет ряд задаваемых параметров, позволяющих лучше адаптировать алгоритм под задачу.

- **Байесовская оптимизация с гауссовским процессом**: хорошо справляется с задачами оптимизации черных ящиков, требует относительно малое количество их вычислений. Позволяет чередовать разведку новых регионов поиска с уточнением уже известных на предмет нахождения оптимума областей. Однако алгоритм выполняется последовательно и имеет кубическую сложность.

- **TPE**: как модификация имеет все те же свойства, что и стандартная байесовская оптимизация, однако работает быстрее в силу использования более простой вероятностной модели. При этом составляющая разведки в нем проигрывает исследованию, т. к. имеется более приоритетное распределение для сэмплирования точек. Также хуже моделируются совместные распределения.

5 Заключение

В данной работе был проведен подробный обзор существующих подходов к задаче оптимизации гиперпараметров в алгоритмах машинного обучения. Был осуществлен их сравнительный анализ. Также было проведено экспериментальное сравнение наиболее популярных методов применительно к различным задачам, данным, моделям машинного обучения и различным их гиперпараметрам. В серии экспериментов наилучшие результаты показали алгоритмы CMA-ES и TPE, как с точки зрения получаемого качества итоговой модели, так и с точки зрения временных и вычислительных затрат. Также на практике подтвердилась широкая область их применения.

Список литературы

1. *Feurer M.*, *Hutter F.* Hyperparameter optimization // Automated Machine Learning: Methods, Systems, Challenges, ser. The Springer Series on Challenges in Machine Learning. — 2019. — P. 3–33.
2. *Bergstra J.*, *Bengio Y.* Random Search for Hyper-Parameter Optimization // J. Mach. Learn. Res. — 2012. — Vol. 13. — P. 281–305.
3. *D. R. Jones M. S.*, *Welch W. J.* Efficient global optimization of expensive black-box functions // Journal of Global Optimization. — 1998. — No. 13. — P. 455–492.
4. *Mayer T.* Efficient global optimization : analysis, generalizations and extensions. — 2003.
5. *Qolomany B.* [et al.]. Parameters optimization of deep learning models using Particle swarm optimization // 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC). — 2017. — P. 1285–1290.
6. *Aszemi N. M.*, *Dominic P. D. D.* Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms // International Journal of Advanced Computer Science and Applications. — 2019.
7. *Tani L.*, *Rand D.*, *Veelken C.*, *Kadastik M.* Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics // arXiv: High Energy Physics - Experiment. — 2020.
8. *Zhang J.* Derivative-Free Global Optimization Algorithms: Population based Methods and Random Search Approaches // ArXiv. — 2019. — Vol. abs/1904.09368.
9. *Snoek J.*, *Larochelle H.*, *Adams R. P.* Practical Bayesian Optimization of Machine Learning Algorithms. — 2012.
10. *Shahriari B.* [et al.]. Taking the Human Out of the Loop: A Review of Bayesian Optimization // Proceedings of the IEEE. — 2016. — Vol. 104. — P. 148–175.
11. *Hesterman J. Y.* [et al.]. Maximum-Likelihood Estimation With a Contracting-Grid Search Algorithm // IEEE Transactions on Nuclear Science. — 2010. — Vol. 57. — P. 1077–1084.

12. *Florea A.*, *Andonie R.* Weighted Random Search for Hyperparameter Optimization // ArXiv. — 2019. — Vol. abs/2004.01628.

13. *Li L.* [et al.]. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization // J. Mach. Learn. Res. — 2017. — Vol. 18. — 185:1–185:52.

14. *Slivkins A.* Introduction to Multi-Armed Bandits // ArXiv. — 2019. — Vol. abs/1904.07272.

15. *Florea A.*, *Andonie R.* A Dynamic Early Stopping Criterion for Random Search in SVM Hyperparameter Optimization. — 2018.

16. *Dorea C. C. Y.*, *Gonçalves C. R.* Alternative sampling strategy for a random optimization algorithm // Journal of Optimization Theory and Applications. — 1993. — Vol. 78. — P. 401–407.

17. *García-Martínez*. Finding Optimal Neural Network Architecture Using Genetic Algorithms. — 2007.

18. *Deb K.*, *Agrawal S.*, *Pratap A.*, *Meyarivan T.* A fast and elitist multiobjective genetic algorithm: NSGA-II // IEEE Trans. Evol. Comput. — 2002. — Vol. 6. — P. 182–197.

19. *Wang D.*, *Tan D.*, *Liu L.* Particle swarm optimization algorithm: an overview // Soft Computing. — 2018. — Vol. 22. — P. 387–408.

20. *Hansen N.* The CMA Evolution Strategy: A Tutorial // ArXiv. — 2016. — Vol. abs/1604.00772.

21. *Lubben J.* Applying a Mixed-integer Evoutionary Strategy for the Configuration and Parameterization of a CMA-ES. — 2018.

22. *Loshchilov I.*, *Hutter F.* CMA-ES for Hyperparameter Optimization of Deep Neural Networks // ArXiv. — 2016. — Vol. abs/1604.07269.

23. *Rasmussen C. E.*, *Williams C. K. I.* Gaussian Processes for Machine Learning. — 2009.

24. *Genton M. G.* Classes of Kernels for Machine Learning: A Statistics Perspective // J. Mach. Learn. Res. — 2001. — Vol. 2. — P. 299–312.

25. *Matérn B.* Spatial variation : Stochastic models and their application to some problems in forest surveys and other sampling investigations //. — 1960.

26. *Khosravian-Arab H.*, *Dehghan M.*, *Eslahchi M. R.* Generalized Bessel functions: Theory and their applications // Mathematical Methods in the Applied Sciences. — 2017. — Vol. 40. — P. 6389–6410.

27. *Lizotte D. J.*, *Wang T.*, *Bowling M.*, *Schuurmans D.* Automatic Gait Optimization with Gaussian Process Regression. — 2007.

28. *Lizotte D. J.* Practical bayesian optimization. — 2008.

29. *Brochu E.*, *Cora V. M.*, *Freitas N. de.* A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning // ArXiv. — 2010. — Vol. abs/1012.2599.

30. *Snoek J.*, *Larochelle H.*, *Adams R. P.* Practical Bayesian Optimization of Machine Learning Algorithms. — 2012.

31. *Hutter F.*, *Hoos H. H.*, *Leyton-Brown K.* Sequential Model-Based Optimization for General Algorithm Configuration. — 2011.

32. *Bergstra J.*, *Bardenet R.*, *Bengio Y.*, *Kégl B.* Algorithms for Hyper-Parameter Optimization. — 2011.

33. *Bergstra J.*, *Yamins D.*, *Cox D. D.* Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. — 2013.

34. *Ozaki Y.*, *Tanigaki Y.*, *Watanabe S.*, *Onishi M.* Multiobjective tree-structured parzen estimator for computationally expensive optimization problems // Proceedings of the 2020 Genetic and Evolutionary Computation Conference. — 2020.

35. *Krizhevsky A.*, *Hinton G.* Learning multiple layers of features from tiny images // Master's thesis, Department of Computer Science, University of Toronto. — 2009.

36. *LeCun Y.*, *Cortes C.* The mnist database of handwritten digits. — 2005.

37. *Aggarwal C. C.* [et al.]. [7] A. Asuncion and D. J. Newman. UCI Machine Learning Repository. — 2008.

38. Scikit-optimize package. — URL: `https://scikit-optimize.github.io/stable/`.

39. *Akiba T.* [et al.]. Optuna: A Next-generation Hyperparameter Optimization Framework // Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. — 2019.

40. *James G. M.*, *Witten D. M.*, *Hastie T. J.*, *Tibshirani R.* An introduction to statistical learning. — 2013.

41. *Ostroumova L.* [et al.]. CatBoost: unbiased boosting with categorical features. — 2018.

42. *Miheev B.* Hyperparameter optimization in machine learning, work repository. — URL: `https://github.com/ezzbreezn/research_seminar`.