Санкт-Петербургский государственный университет
Математическое моделирование, программирование и искусственный интеллект
Динамические системы, эволюционные уравнения, экстремальные задачи и математическая кибернетика

Реутская Анастасия Александровна

Предсказание хаотической динамики методами машинного обучения

Выпускная квалификационная работа

Научный руководитель:
к. ф.-м. н., ст. науч. сотр. Р. Н. Мокаев

Рецензент:
гл. технический директор ООО
«ЮниДата» А. В. Цырюльников

Санкт-Петербург
2022

Saint Petersburg State University
Mathematical Modeling, Programming and Artificial Intelligence
Dynamical Systems, Evolutionary Equations, Extreme Problems and Mathematical Cybernetics

Reutskaya Anastasia Aleksandrovna

A machine learning-based chaotic dynamics prediction

Graduation Project

Scientific Supervisor:
Senior Researcher R. N. Mokaev

Reviewer:
Chief Technical Director of UniData LLC
A. V. Tsyriulnikov

Saint Petersburg
2022

Оглавление

Введение

Динамический подход к описанию различных систем известен со времен Ньютона. Он является основой анализа большинства классических явлений в физике и других естественных науках: сначала строится соответствующая математическая модель в виде динамических уравнений, а далее тем или иным способом изучаются их решения, которые можно сопоставить с данными экспериментов. Развитие этих идей, а также представление о том, что состояние модели в любой момент времени однозначно определяется начальными условиями, привели исследователей к понятию динамической системы.

Хотя динамическая система и является некоторой математической абстракцией, данная парадигма стала продуктивным инструментом при описании многих реальных явлений. Наибольший успех в этом направлении был получен академиком А. А. Андроновым в 30-е годы XX века, которым была создана строгая теория автоколебаний двумерных систем [1, 3]. Следующей целью исследователей стало изучению возможности распространения этой теория на многомерные системы. Однако, несмотря на значительные открытия в данной области, до 60-х годов XX столетия не было понятно, насколько сложными могут быть движения в таких системах.

В 1963 г. американский метеоролог Э. Лоренц экспериментально показал принципиальное существование предельного режима (аттрактора) нового типа в гладких многомерных динамических системах [32]. Полученный Лоренцем аттрактор внешне представляет собой сложную геометрическую форму. С точки зрения численного моделирования, малая неточность в начальных данных в таких аттракторах приводит к разбеганию траекторий во времени. Такие предельные режимы называются хаотическими. Именно из-за чувствительности к выбору начальных данных предсказание хаотических траекторий существенно затруднено.

Хаотическое поведение наблюдается в самых разных системах — электрические схемы, лазеры, химические реакции, динамика жидкостей и магнитно-механических устройств, метеорология, движение спутников солнечной системы, динамика потенциалов в нейронах и молекулярных колебаниях, и во многих других. В настоящее время теория хаоса активно применяется, например, в медицине при изучении эпилепсии для предсказаний приступов, учитывая первоначальное состояние организма.

Динамический хаос встречается во множестве прикладных систем — от моделирования движения тектонических плит до изучения экономических процессов. Поэтому задача прогнозирования хаоса является крайне важной и актуальной. С другой стороны, в связи с чувствительностью к выбору начальных данных, долгосрочное прогнозирование в системах с хаотическим поведением становится сложной и нетривиальной задачей. Поэтому возможности традиционных методов исследования динамических систем, таких как, например, математическое моделирование, оказались сильно ограничены в исследовании хаоса.

Активное развитие машинного обучение с конца XX века дало толчок исследованию хаотической динамики и позволило выработать новый подход к решению этой задачи. Методы машинного обучения хорошо подходят для предсказания хаотической динамики, так как это универсальные методы аппроксимации функций, которые могут отображать любую нелинейную функцию без априорных предположений о данных. С помощью статистических методов алгоритмы машинного обучения позволяют классифицировать данные, строить прогнозы и выделять важные особенности в работе различных систем.

При этом критической проблемой в применении методов машинного обучения является переобучение. Это связано с тем, что модель машинного обучения улавливает не только полезную информацию, содержащуюся во входных данных, но и нежелательный шум. Это приводит к низкому уровню обобщающей способности. Данные вне выборки — это те данные, которые не используются при обучении модели. И эффективность методов с точки зрения обобщения для данных вне выборки всегда ниже, чем у обучающих данных. Таким образом, для достижения высокой точности предсказания в каждой конкретной задаче важно определить оптимальные параметры модели машинного обучения. Для этой цели используются различные алгоритмы оптимизации параметров модели.

От эффективности оптимизации параметров модели машинного обучения зависит практическая применимость метода машинного обучения. Существуют различные методы оптимизации параметров. Мы рассмотрели такие популярные алгоритмы оптимизации как алгоритм кукушки (CS) [11], алгоритм серых волков (GWO) [12], метод роя частиц (PSO) [4, 5], методы Пауэлла [2] и Джая-Пауэлла [18], алгоритм хаотической оп-

тимизации (COA) [17], а также была разработана собственная модификацию алгоритма Пауэлла.

В данной работе мы проанализировали эффективность различных комбинаций алгоритмов машинного обучения и методов оптимизации параметров для предсказания хаотической динамики системы Лоренца и хаоса в модели лазеров с насыщающимся поглотителем. Также был разработан программный пакет на языке Python, позволяющий комбинировать методы машинного обучения SVR и XGBoost с описанными и предложенным алгоритмами оптимизации параметров.

Цели и задачи работы

Первой целью данной работы является исследование и развитие существующих алгоритмов машинного обучения и методов оптимизации параметров.

Второй целью является создание программного пакета на языке Python для комбинирования изученных и разработанных методов и применения их к различным системам.

Третьей целью является применение полученных методов для предсказания хаоса в системе Лоренца и в модели лазеров с насыщающимся поглотителем. Исходя из результатов экспериментов проанализируем эффективность, точность и обобщающую способность методов.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Изучить методы машинного обучения: регрессия опорных векторов и экстремальный градиентный бустинг
2. Изучить и реализовать некоторые известные алгоритмы оптимизации параметров
3. Разработать и реализовать модификацию алгоритма Пауэлла
4. Применить комбинации изученных методов машинного обучения и алгоритмов оптимизации параметров к системе Лоренца
5. Проанализировать эффективности и точность моделей машинного обучения для системы Лоренца
6. Применить полученные комбинации методов для предсказания хаотической динамики в модели лазеров с насыщающимся поглотителем
7. Проанализировать обобщающую способность полученных моделей машинного обучения в сравнении с результатами экспериментов для системы Лоренца

Алгоритмы машинного обучения

Машинное обучение — обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных прогнозировать поведение системы на основе закономерностей в существующих данных о ее поведении. Данное направление находится на стыке математической статистики, методов оптимизации и классических математических дисциплин, но имеет также и собственную специфику, связанную с проблемами вычислительной эффективности и переобучения.

Опишем общую постановку задачи машинного обучения. Дано конечное множество *прецедентов* (объектов, ситуаций), по каждому из которых собраны некоторые данные (описание). Совокупность всех имеющихся описаний прецедентов называется *обучающей выборкой*. Требуется по этим частным данным выявить общие зависимости, закономерности, взаимосвязи, присущие не только этой конкретной выборке, но вообще всем прецедентам, в том числе тем, которые ещё не наблюдались.

Для решения задачи обучения по прецедентам в первую очередь фиксируется модель восстанавливаемой зависимости. Затем вводится функционал качества (функция ошибок), значение которого показывает, насколько хорошо модель описывает наблюдаемые данные. Алгоритм обучения ищет такой набор параметров модели, при котором функционал качества на заданной обучающей выборке принимает оптимальное значение. Процесс обучение модели по выборке данных в большинстве случаев сводится к применению численных методов оптимизации.

Таким образом, алгоритм машинного обучения состоит из трех основных частей:

1. Процесс принятия решений.

   Взяв за основу некоторые входные данные, алгоритм выдает оценку в отношении наличия закономерности в данных.

2. Функция ошибок.

   Функция ошибок служит для оценки прогноза по модели. При наличии тренировочных данных функция ошибок сравнивает их с предсказаниями, чтобы оценить точность модели.

3. Процесс оптимизации модели.

   Оптимальные параметры модели машинного обучения повышают точность и уменьшают время работы алгоритма.

В данной работе мы применяем метод регрессии опорных векторов и метод экстремального градиентного бустинга. Рассмотрим структуры этих методов более подробно.

2.1. Регрессия метода опорных векторов (SVR)

Метод опорных векторов (SVM) — один из самых популярных методов машинного обучения, который применяется для решения задач классификации и регрессии. Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом. Алгоритм основан на предположении, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора.

Предсказание хаотических временных рядов можно отнести к задачам регрессии метода опорных векторов (support vector regression, SVR). Основная идея SVR состоит в том, чтобы отобразить данные в пространстве объектов более высокой размерности с помощью нелинейного отображения $\Phi(\cdot)$ и выполнить линейную регрессию в этом пространстве [28]. Таким образом, с помощью регрессии мы получаем оценку функции на основе заданного набора данных $\{x_i, y_i : i = 1, 2, \ldots, N\}$, где $x_i \in R^n$ – входящий вектор , а $y_i \in R$ – искомые значения. SVM аппроксимирует функцию вида

$$f(x) = w^T \cdot \Phi(x) + b, \quad \Phi : R^n \to F, \;\; w \in F \tag{2.1}$$

Здесь $\{\Phi_i(x)\}_{i=1}^N$ – точки пространства параметров, а $\{w_i\}_{k=1}^N$ и b – коэффициенты. Их можно оценить через минимизацию функции риска:

$$R(C) = C\frac{1}{l}\sum_{i=1}^{N} L_\epsilon(y_i, f(x_i)) + \frac{1}{2}\|w\|^2, \tag{2.2}$$

где $L_\epsilon(y_i, f(x_i))$ – функция потерь, измеряющая ошибки аппроксимации между ожидаемым выходом y_i и предсказанным выходом $f(x_i)$, а C – константа регуляризации, определяющая компромисс между ошибкой обучения и производительностью обобщения. Второе слагаемое $\frac{1}{2}\|w\|^2$ используется в качестве измерения плоскостности функ-

ции. Если ввести в данное уравнение фактор релаксации ξ, ξ^*, то получим следующую функцию:

$$minJ(w,\xi,\xi^*) = \frac{1}{2}w^T W + C\sum_{i=1}^{N}(\xi+\xi^*), \text{ где } \begin{cases} y_i - w^T\Phi(x_i) - b \le \epsilon + \xi_i, \\ w^T\Phi(x_i) + b - y_i \le \epsilon + \xi_i^*, \\ \xi, \xi^* \ge 0. \end{cases} \tag{2.3}$$

Введя множители Лагранжа и используя ограничения оптимальности, решение можно описать в следующей форме:

$$f(x) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)K(x_i, x) + b, \tag{2.4}$$

где $\alpha_i, \alpha_i^*, i = 1..N$ – множители Лагранжа, удовлетворяющие $\alpha_i \times \alpha_i^* = 0$, где $\alpha_i, \alpha_i^* > 0$. Они получаются путем максимизации следующей формулы с введенным фактором релаксации:

$$maxL(\alpha,\alpha^*) = -\frac{1}{2}\sum_{i=1}^{N}(\alpha_i-\alpha_i^*)^2K(x_i,x)+(1-\epsilon)\sum_{i=1}^{N}y_i(\alpha_i-\alpha_i^*), \text{ где } \begin{cases} \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0, \\ 0 \le \alpha_i, \alpha_i^* \le C, i = 1..N. \end{cases} \tag{2.5}$$

Согласно регрессии метода опорных векторов, большинство α_i, α_i^* занулятся. Таким образом, окончательная формулировка получена путем решения упомянутой выше задачи оптимизации, а регрессия метода опорных векторов имеет следующий вид:

$$f(x) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)K(x_i, x) + b. \tag{2.6}$$

Несмотря на то, что нелинейная функция $\Phi(\cdot)$ обычно неизвестна, все вычисления, связанные с ней могут быть сведены к форме $K(x_i, x) = \Phi(x_i)^T \cdot \Phi(x)$. Тогда все, что нам нужно сделать, сводится к выбору соответствующей функции ядра. Преимущества радиально-базисной функции ядра (RBF) делают ее универсальной для метода опорных векторов. Эта функция может быть применена к любой системе при правильном выборе параметров.

2.2. Экстремальный градиентный бустинг (XGBoost)

Алгоритм XGBoost – это алгоритм машинного обучения, который был представлен Тяньци Ченом в 2014 году [21]. Когда мы пытаемся предсказать целевую переменную с помощью любого алгоритма машинного обучения, главные причины отличий реальной и предсказанной переменной — это шумы (искажение данных), дисперсия (чувствительность к небольшим изменениям входных данных) и смещение (потеря взаимосвязи между входным вектором и выходным состоянием). XGBoost включает в себя ансамбль (ensemble learning) и вариант градиентного бустинга, основанный на деревьях решений. Они представляют собой логические схемы, позволяющие получить окончательное решение о классификации объекта после ответов на иерархически организованную систему вопросов. Ансамбль — это набор предсказателей, то есть некоторых алгоритмов оптимизации, которые вместе дают ответ (например, среднее по всем результатам предсказателей). Преимущество ансамбля заключается в том, что несколько предсказателей, которые пытаюсь получить одну и ту же переменную, дадут более точный результат, нежели одиночный предсказатель. Также ансамбль позволяет уменьшить дисперсию и смещение.

Бустинг – это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно. То есть бустинг последовательно объединяет слабых учеников (результаты предсказания), чтобы получить сильного конечного ученика. Каждый базовый алгоритм обучения учится у своего предыдущего базового ученика и уменьшает его ошибку. В итоге, конечный предсказатель проявляет минимальную дисперсию и смещение в процессе обучения. В целом, алгоритм XGBoost объединяет несколько базовых предсказателей (деревьев решений) для построения более надежной агрегированной модели. Чтобы предсказать конечный результат, алгоритм XGBoost объединяет веса листьев со всех деревьев. Следовательно, для любого заданного набора данных с m объектами для прогнозирования выходных данных используется следующее уравнение:

$$y_i^* = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \{f(x) = w_q(x)\}, \ (q \in R^m \to T, \ w \in R^T), \tag{2.7}$$

где K – количество аддитивных функций обучения, T – количество листьев в дереве, q – древовидная структура, а w – вес.

Аддитивное обучение − это стратегия, используемая для оптимизации деревьев; она соответствует добавлению одного дерева за раз при оптимизации цели. Неправильная настройка параметров деревьев решений − например, в отношении количества деревьев, их глубины или количества итераций − может привести к переобучению. Что делает алгоритм XGBoost достаточным, так это его замечательная способность к регуляризации. XGBoost ограничивает модели за счет интеграции $L1$- и $L2$-регуляризации, что позволяет избежать переобучения. Функция $L(\phi)$ используется на каждой итерации с целью оптимизации градиентного бустинга и определяется следующим уравнением:

$$L(\phi) = \sum_i l(y_i^*, y_i) + \sum_k \Omega(f_k), \tag{2.8}$$

где l − функция потерь, которая вычисляет разницу между прогнозируемым и фактическим значением y, а $\Omega(f_k)$ − функция, которая ограничивает сложность модели. Приведенное ниже уравнение (2.9) определяет формулу $\Omega(f)$, где коэффициенты (γ и λ) используются для управления моделью предсказания.

$$\Omega(f) = \gamma T = \frac{1}{2}\lambda ||w||^2 \tag{2.9}$$

Как правило, предсказатели продолжают добавляться до тех пор, пока в целевую функцию не будет внесено никаких улучшений. Точность алгоритма XGBoost достигается тем, что он фокусируется на исправлении ошибок, допущенных другими предсказателями. Таким образом, на каждой итерации вычисляются ошибки и обновляются веса для улучшения окончательного прогноза. Поэтому благодаря своей высокой производительности, высокой скорости выполнения и способности решать крупномасштабные задачи параллельно с небольшими вычислительными требованиями, XGBoost стал очень популярным алгоритмом для решения регрессионных, классификационных и различных других задач [22, 23].

Алгоритмы оптимизации параметров

От эффективности оптимизации параметров модели машинного обучения зависит практическая применимость метода машинного обучения. В эпоху больших данных многие классические алгоритмы оптимизации становятся неприменимы, так как в этом случае требуется решать задачи оптимизации функций за время меньшее, чем необходимо для вычисления значения функции в одной точке. Этим требованиям можно удовлетворить в случае грамотного комбинирования известных подходов в оптимизации с учётом конкретной специфики решаемой задачи. В нашем случае важной спецификой рассматриваемых задач является хаотическая динамика.

Рассмотрим $n-$мерную хаотическую систему, описанную следующим уравнением:

$$\dot{x} = f(x, x_0, \varphi_0), \tag{3.1}$$

где $x = (x_1, x_2, \ldots, x_n) \in R^n$ представляет собой вектор состояния исходной системы, x_0 описывает ее начальное состояние, а $\varphi_0 = (\varphi_1^0, \varphi_2^0, \ldots, \varphi_n^0)$ − набор исходных параметров системы.

Зная структуру хаотической системы (3.1), можем описать соответствующую оценочную систему:

$$\dot{y} = f(y, x_0, \tilde{\varphi_0}), \tag{3.2}$$

где $y = (y_1, y_2, \ldots, y_n) \in R^n$ является вектором состояния оцениваемой системы, а $\tilde{\varphi_0} = (\tilde{\varphi_1^0}, \tilde{\varphi_2^0}, \ldots, \tilde{\varphi_n^0})$ представляет собой набор оцененных параметров.

Оптимизация параметров проводится через оценку параметров хаотической системы. Полученные оптимальные параметры могут свести к минимуму разрыв в поведении между исходной системой и оцениваемой [18]. Для достижения этих параметров $\tilde{\varphi}_1^0, \tilde{\varphi}_2^0, \ldots, \tilde{\varphi}_n^0$ требуется минимизация функции потерь

$$minJ(\tilde{\varphi}) = \frac{1}{K} \sum_{k=1}^{K} \|x_k - y_k\|. \tag{3.3}$$

На Рис. 3.1 показан основной принцип оптимизации параметров модели хаотической системы. Здесь x_k и y_k, где $(k = 1, 2, \ldots, K)$ − векторы состояния исходной и оцениваемой систем соответственно, которые наблюдаются в момент времени k. K −

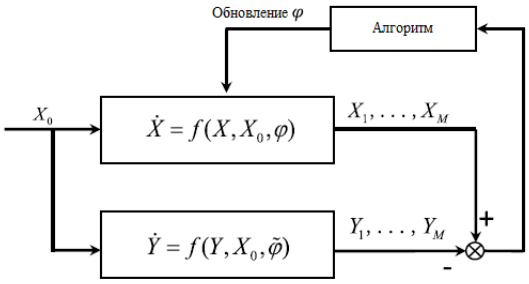

Рис. 3.1. Принцип оптимизации оценки параметров для хаотической системы

общее количество векторов состояния, используемых при оценке. Сложность оценки параметров модели машинного обучения для хаотической системы связана с нестабильностью ее фазового пространства. Еще одной сложностью является чувствительность таких систем к начальному состоянию. Классические алгоритмы оптимизации на хаотических системах легко попадают в локальный оптимум, что мешает поиску решения.

Рассмотрим некоторые популярные алгоритмы оптимизации параметров.

3.1. Алгоритм кукушки

Алгоритм кукушки (Cuckoo search) — один из алгоритмов роевого интеллекта, разработанный Янг Синьшэ и Суашем Дебом в 2009 году [11]. Этот алгоритм вдохновлен гнездовым паразитизмом некоторых видов кукушек, что подкладывают свои яйца в гнезда других птиц. Некоторые из владельцев гнезд могут вступить в прямой конфликт с кукушками, что врываются к ним. Например, если владелец гнезда обнаружит, что яйца не его, то он или выбросит эти чужие яйца или просто покинет гнездо создаст новое где-то в другом месте.

Каждое яйцо в гнезде представляет собой решение, а яйцо кукушки — новое решение. Цель заключается в использовании новых и потенциально лучших (кукушкиных) решений, чтобы заменить менее хорошее решения в гнездах.

Алгоритм кукушки основан на трех базовых правилах:

- Кукушка закладывает по одному яйцу в гнездо за раз, и сбрасывает его в случайно выбранном гнезде;
- Лучшие гнезда с высоким качеством яиц ($\Leftrightarrow$ с лучшими решениями) переносятся в следующие поколения;

- Количество гнезд фиксировано, и хозяин может обнаружить кукушкино яйцо с вероятностью $P_a \in [0, 1]$. В этом случае хозяин может выбросить яйцо из гнезда или покинуть это гнездо и построить новое.

Исходя из определенных правил, опишем схему алгоритма:

1. **Инициализация**

   Инициализируем популяцию $S = (s_i, i \in [1 : |S|])$ из $|S|$ хозяйских гнёзд и кукушку, то есть определяем начальные значения компонентов векторов $X_i; i \in [1 : |S|]$, и вектор начального положения кукушки X_c.

2. **Полёт Леви**

   Находим новое положение кукушки X_c, с помощью полётов Леви: $X_c(t+1) = X_c(t) + V \otimes Levy(\lambda)$, где $V = rand(0,1) * (X_c - X_i)$, а $\otimes -$ матричное произведение. Полёт Леви обеспечивает случайное блуждание, а длина случайного шага берется из распределения Леви $Levy(\lambda)$ $u = t^{-\lambda}, (1 < \lambda \leq 3)$.

3. **Подкидывание яйца**

   Случайным образом находим гнездо $s_i : i \in [1 : |S|]$, и, если $f(X_c) > f(X_i)$, заменяем яйцо в этом гнезде на яйцо кукушки, то есть полагаем $X_i = X_c$.

4. **Выбрасывание яиц**

   С вероятностью P_a удаляем из популяции некоторое число худших случайно выбранных гнезд и строим новые гнезда в местах определённых с помощью полётов Леви.

   Если поколение достигло заданного предела, то заканчиваем алгоритм, иначе переходим ко второму шагу.

3.2. Алгоритм серых волков

Рассмотрим еще одного представителя алгоритмов роевого интеллекта − алгоритм серых волков. Алгоритм оптимизации «Серых волков» (Grey Wolf Optimization) [12] − это один из недавних биоинспированных алгоритмов оптимизации, основанный на имитации загонной охоты стаи серых волков, разработанный Сейедали Мирджалили в 2014 году. Данный алгоритм очень хорошо зарекомендовал себя на различных типах функций, как по точности поиска экстремума, так и по скорости сходимости. Главным

преимуществом данного алгоритма является скорость сходимости. Практически уже при 100 итерациях алгоритм находит приемлемое решение на всех тестовых функциях и на любой размерности задачи. Благодаря этому алгоритм хорошо подходит для оптимизации параметров метода машинного обучения для предсказания хаотической динамики.

В основе метода лежит механизм охоты серых волков в природе. Все агенты делятся на четыре иерархических типа: альфа, бета, дельта и омега. Для выполнения оптимизации выполняются три основных этапа, в ходе которых волки ищут добычу, окружают, а затем атакуют её [12]. Математическая реализация заключается в том, что тройке самых приспособленных, то есть близких к искомой точке, экземпляров присваиваются ранги альфа, бета и дельта, по мере убывания приспособленности. Остальные волки получают ранг омега. В основном цикле алгоритма волки окружают свою добычу. Позиция каждого волка в стае на $t+1$ шаге окружения жертвы описывается следующим образом:

$$X_i(t+1) = X_{p,i} - A_i * |C_i * X_{p,i} - X_i(t)|,\ i = 1..N, \tag{3.4}$$

где t – номер итерации, i – номер компоненты N мерного пространства, $X(t)$ – позиция рассматриваемого волка, $X_p(t)$ – позиция рассматриваемой жертвы, которая является оптимальным решением.

$$A_i = 2a(t)rand(0,1) - a(t), C_i = 2rand(0,1), a(t) = 2 - \tfrac{2t}{L}. \tag{3.5}$$

Для поиска глобального оптимума организуются следующая сходящаяся процедура. Пусть волки $\alpha,\ \beta,\ \delta$ характеризуют три лучшие текущие на шаге t позиции. Для каждого волка с ролью ω с учетом известных позиций волков $\alpha,\ \beta,\ \delta$ считаются $X_1,\ X_2,\ X_3$:

$$X_{1,i}^{(k)}(t+1) = X_i^{(\alpha)}(t) - A_i^{(\alpha)}|C_i^{(\alpha)}X_i^{(\alpha)}(t) - X_i^{(k)}(t)|, \tag{3.6}$$

$$X_{2,i}^{(k)}(t+1) = X_i^{(\beta)}(t) - A_i^{(\beta)}|C_i^{(\beta)}X_i^{(\beta)}(t) - X_i^{(k)}(t)|, \tag{3.7}$$

$$X_{3,i}^{(k)}(t+1) = X_i^{(\delta)}(t) - A_i^{(\delta)}|C_i^{(\delta)}X_i^{(\delta)}(t) - X_i^{(k)}(t)|, \tag{3.8}$$

где i – номер компоненты N мерного пространства, k – номер волка с ролью ω. С помощью $X_1,\ X_2,\ X_3$ итоговая позиция любого рассматриваемого волка определяется следующим образом:

$$X(t+1) = \frac{X_1(t+1) + X_2(t+1) + X_3(t+1)}{2} \tag{3.9}$$

На вход в алгоритм серых волков поступает популяция размером M, общее количество итераций L, и N – размерность пространства поиска. Далее выполняются следующие шаги:

1. **Инициализация**

   Случайно инициализируется позиции волков $X_i(i = 1..M)$ в области поиска.

2. **Целевая функция**

   Рассчитывается целевая функция для каждого волка $y_i = f(X_i),\ i \in (1..M)$.

3. **Распределение ролей**

   Выбирается три наилучших решения из $\{y_i,\ i = 1..N\}$ и определяются соответствующие им $X^{(\alpha)},\ X^{(\beta)},\ X^{(\delta)}$.

4. **Новые позиции волков**

   Для каждого волка $i = 1..M$ по всем компонентам $j = 1..N$ рассчитываются $X_1,\ X_2,\ X_3$ по формулам (3.6), рассчитываются компонента вектора новой позиции волка по (3.9).

3.3. Метод роя частиц

Метод роя частиц (Particle swarm optimization) был вдохновлен поведением стаи птиц, так как она представляет собой прекрасный пример коллективного поведения животных. Летая большими группами, птицы почти никогда не сталкиваются в воздухе, это объясняется наличием роевого интеллекта – птицы в стае действуют согласно определенным правилам. Кружа в небе, каждая из птиц следит за своими сородичами и координирует свое движение согласно их положению. А найдя источник пищи, она оповестит их об этом. И этот факт играет одну из ключевых ролей в рассматриваемом методе оптимизации. Причины такого поведения являлись предметом исследования многих

социобиологов. Одним из наиболее популярных объяснений этого феномена является то, что преимущества от такого поведения каждой особи стаи больше, чем такие очевидные недостатки, как необходимость борьбы за найденную пищу с другими особями. Источники пищи обычно расположены случайным образом, поэтому в одиночестве птица вполне может погибнуть, не найдя ни один в течение долгого времени. Однако если все птицы будут делиться с сородичами информацией о находках, то шансы каждой из них на выживание резко повышаются. Таким образом, будучи неэффективной для отдельной особи, такая стратегия является залогом эффективности стаи и вида в целом.

Наблюдение за птицами вдохновило Крейга Рейнольдса на создание в 1986 году компьютерной модели, которую он назвал Boids [9]. Для имитации поведения стаи птиц, Рейнольдс запрограммировал поведение каждой из них в отдельности, а также их взаимодействие. При этом он использовал три простых принципа:

1. каждая птица стремится избежать столкновений с другими птицами,
2. каждая птица движется в том же направлении, что и находящиеся неподалеку птицы,
3. птицы стремятся двигаться на одинаковом расстоянии друг от друга.

Результаты моделирования показали, что несмотря на простоту лежащих в основе программы алгоритмов, поведение компьютерной модели очень похоже на поведение реальной стаи: модели птиц сбиваются в группы, уходят от столкновений и хаотично метаются аналогично настоящим.

Вдохновленные моделью Рейнольдса, в 1995 году Джеймс Кеннеди и Рассел Эберхарт предложили [4] алгоритм для оптимизации непрерывных нелинейных функций, названный ими методом роя частиц. Метод оптимизирует функцию, поддерживая популяцию возможных решений (частиц) и перемещая эти частицы в пространстве решений согласно простой формуле. Перемещения подчиняются принципу наилучшего найденного в этом пространстве положения, которое постоянно изменяется при нахождении частицами более выгодных положений.

Допустим, в рое N частиц размерностью M, а L – максимальное число итераций. Тогда положим $x_i(k)$ – положение i-й частицы роя $i = 1..N$ в k-й момент времени $k = 1..L$. Решение о том, какое положение занять, частица принимает на основе своего

положения в прошлый момент времени и текущей скорости по следующей формуле:

$$x_i(k+1) = x_i(k) + v_i(k+1). \tag{3.10}$$

Выражение для скорости частицы является основой метода роя частиц:

$$v_i(k+1) = c_{in}v_i(k) + c_{cog}rand(0,1)(r_i(k) - x_i(k)) + c_{soc}rand(0,1)(p_i(k) - x_i(k)). \tag{3.11}$$

Здесь первое слагаемое (инерционное) обеспечивает стремление частицы продолжить движение в том же направлении, второе (когнитивное) − стремление к собственному лучшему положению, а третье (социальное) − стремление к лучшему положению, найденному окружением частицы.

Инерционный коэффициент c_{in} позволяет настроить значимость инерционной компоненты. Большая величина c_{in} ускоряет исследование окрестностей частицей на предмет оптимального положения. Поэтому инерционный коэффициент линейно уменьшается с увеличением числа итераций k:

$$c_{in}(k) = (c_{in}^{low} - c_{in}^{up})\frac{k}{N} + c_{in}^{up}. \tag{3.12}$$

Когнитивный и социальный коэффициенты c_{cog} и c_{soc} обычно принимаются постоянными и равными 1.49445 [7].

Описанный подход к выбору инерционного, когнитивного и социального параметров приводит к следующему. В начале поиска частицы активно перемещаются по допустимому пространству за счет превалирования инерционной компоненты. При этом на нее практически не оказывают влияние сведения о лучшем положении, найденном соседями, тогда как собственное лучшее положение имеет некоторое значение. Фактически, поначалу частицы роя исследуют практически независимо. Со временем они начинают все больше принимать во внимание полученный соседями результат, все меньше исследуя новые области в и все больше изучая окрестности собственного и соседних лучших положений. Наконец, к концу времени поиска все частицы переходят к изучению окрестностей общего лучшего положения.

При определении положения частицы $x_i(k+1)$ с помощью формулы (3.11) на каждом шаге необходимо проверить, находится ли она в допустимой области $lowerBound_i \leq x_i \leq u$

Если j-я компонента вектора $x_i^j(k+1)$ выходит за допустимые рамки, ее значение принимается равным граничному, а компонента скорости $v_i^j(k+1)$ обнуляется. Аналогично при расчете скорости (3.11) проверяется, не превышает ли какая-либо ее j-я компонента по величине разности верхней и нижней границ скорости, иначе положение частицы заведомо выйдет за пределы. В таком случае значение скорости принимается максимально допустимой. На первом шаге частицы $x_i(1)$ равномерно распределяются по области с нулевыми скоростями, что позволяет начать изучение всего доступного пространства.

Исходя из этого, алгоритм можно описать следующими шагами:

1. **Инициализация**

   а. Задаются число частиц в рое N и их размерность M, максимальное число итераций L, верхние и нижние границы частиц и скорости, минимальное и максимальное значения инерционного коэффициента c_{in}^{low}, c_{in}^{up},

   б. Начальные положения частиц $x_i(1)$ равномерно распределяются в области.

   в. Начальные скорости $v_i(1)$ принимаются нулевыми.

   г. Задается матрица размера $M \times N$ лучших когда-либо найденных положений для каждой частицы r. Элементы матрицы инициализируются начальными положениями $x(1)$.

   д. Задается матрица размера $M \times N$ лучших когда-либо найденных положений для окрестности какой-либо частицы ρ. Элементы матрицы инициализируются начальными положениями $x(1)$.

   е. Задается вектор лучших значений целевой функции Y^{best} большими числами.

   ж. Задается лучшее значение целевой функции, найденное всем роем y^{best}

2. **Для** $k = 1..L$

3. **Вычисление целевой функции**

   Для $i = 1..N$ вычислим целевую функцию y_i для текущей частицы x_i.

4. **Обновление лучшего значения целевой функции и соответствующего положения для i-й частицы**

   если $y_i < y^{best}$, то $y^{best} = y_i$, $r_i = x_i$

5. **Обновление лучшего значения целевой функции и соответствующего положения для всего роя**

   если $y_i < Y_i^{best}$, то $y^{best} = y_i$, $x^{best} = x_i$

6. **Сброс соответствующих положений в окрестности i-й частицы**

   $\rho_i = x_i$

7. **Определение скорости**

   Скорость рассчитывается по (3.11) И проверяется на вхождение в границы

8. **Обновление положения частицы**

   Положение частицы рассчитывается по (3.10) и проверяется на нахождение в области поиска

3.4. Алгоритм хаотической оптимизации

Алгоритм хаотической оптимизации (chaos optimization algorithm, COA) разработали китайские ученые университета Чжэнчжоу — Ху Юсия и Чжан Хунтао в 2012 году [17]. Алгоритм является эффективным и удобным способом глобальной оптимизации параметров модели машинного обучения. Для повышения эффективности и точности поиска алгоритм используется в изменяющемся масштабе, чтобы динамически уменьшать диапазоны поиска во время работы алгоритма. В данной работе мы используем COA для оптимизации ключевых параметров SVR: C, ε, g. Параметр C — константа регуляризации, определяющая компромисс между ошибкой обучения и производительностью обобщения. Константа отвечает за баланс между удержанием полосы SVM как можно более широкой и ограничением количества нарушений зазора (т.е. появления экземпляров, которые оказываются посредине полосы или даже на неправильной стороне). g — параметр функции ядра радиального базиса, а ε — параметр используемый в процессе регрессии, им управляется ширина полосы.

3.4.1. Реконструкция фазового пространства

Прогнозирование хаотических временных рядов основано на теории реконструкции фазового пространства в соответствии с теоремой вложения Такенса [19]. Реконструированное фазовое пространство — это m-мерное метрическое пространство, в которое встроен временной ряд. Мы имеем хаотический временной ряд $\{x(t)\}, t = 1, \ldots, N$,

теперь необходимо выбрать размерность вложения m, время задержки τ. Тогда фазовое пространство может быть описано следующим образом:

$$X(t) = (x(t), x(t-\tau), ..., x(t-(m-1)\tau)), X(t) \in R^m, t = 1, .., M$$

Где $X(t)$ − вектор или точка в фазовом пространстве построения, $M = N - (m-1)\tau$ − количество точек в реконструированном фазовом пространстве. Если размер вложения достаточно велик, фазовое пространство гомеоморфно пространству состояний, создавшему временной ряд, то есть фазовое пространство содержит ту же информацию, что и исходное пространство состояний. Существует отображение $f(\cdot)$, удовлетворяющее следующему уравнению:

$$x(t+T) = f(X(t))$$

Локальная модель обычно используется для прогнозирования следующего шага, в котором $T > 0$ является шагом прямого прогнозирования. $f(\cdot)$ − модель прогнозирования для хаотических временных рядов.

Если взять $f(\cdot)$ в качестве регрессии опорного вектора и $T = 1$, получим модель прогнозирования для хаотических временных рядов на основе SVM [17]:

$$\hat{x}(t+1) = \sum_{i=1}^{M} (\alpha_i - \alpha_i^*) K(X(t), X(i)) + b,$$

где $\hat{x}(t+1)$ − значение одного шага прогнозирования. Аналогично могут быть выражены следующие точки в фазовом пространстве:

$$\hat{X}(t+1) = (\hat{x}(t+1), x(t+1-\tau), ..., x(t+1-(m-2)\tau)).$$

3.4.2. Оптимизация параметров

Хаотические системы обладают особыми свойствами, такими как эргодичность, стохастичность и зависимость чувствительности от начальных данных.

К алгоритму хаотической оптимизации применяется одномерная логистическое отображение, математическая модель которого имеет следующий вид:

$$Z_{k+1} = \mu Z_k (1 - Z_k), \quad Z_k \in (0, 1), k = 1, 2, \ldots \tag{3.13}$$

Где Z_k – значение переменной Z на k-й итерации, а μ – так называемый бифуркационный параметр системы ($0 \leq \mu \leq 4$). Если $3.5699456 \cdots \leq \mu \leq 4$, логистическое отображение работает в хаотическом состоянии, то есть без стационарного решения, а система представляет собой полное отображение интервала $[0, 1]$.

Алгоритм хаотической оптимизации [17] состоит из следующих шагов:

1. **Инициализация.**

   Генерируем Z_i^k, $k = 1, \ldots N$, где N – длина временного ряда. Определяем x_0 как начальное оптимальное решение x^*

   $$x_i^0 = a_i + (b_i - a_i)Z_i^k, \;\; i = 1, \ldots, 3$$

   где k – случайное значение из множества $1, \ldots, N$, a, b – границы искомых параметров C, ε, g для алгоритма SVR. Начальное значение целевой функции f^* определяется через x^*, а именно $f^* = f(x^*) = f(x_0)$. Отсюда определяется критерий остановки: достижение максимальной итерации M, максимальная допустимая среднеквадратичная ошибка J_{end} или количество раз, когда достигается оптимальное решение A. Определяем K как количество итераций и количество раз достижения оптимального решения $time = 0$.

2. **Преобразование диапазонов поиска.**

   Предсказание параметров модели машинного обучения:

   $$x_i^k = a_i + (b_i - a_i)Z_i^k, \;\; i = 1, \ldots, 3$$

   Следовательно, диапазон поиска параметра x_i^k будет изменен с $[-1, 1]$ на $[a_i, b_i]$.

3. **Обновление параметров поиска**

   Применяем метод SVR с параметрами, предсказанными в x_i^k и находим среднеквадратичную ошибку предсказания

   $$f = RMSE = \sqrt{\sum_{i=1}^{N}(y(i) - \hat{y}(i))^2/N}$$

   где $y(i)$ и $\hat{y}(i)$ – соответственно фактические и прогнозируемые значения хаотических временных рядов, N – количество тестовых точек. В нашем алгоритме

фактическими значениями являются точки из реконструированного фазового пространства. А прогнозируемые значения − результат предсказания с помощью SVR с параметрами x_i^k.

Тогда целевая функция $f(x_i^k) = f(x_i(K))$. Если $f(x_i(K)) < f^*$, тогда $f^* = f(x_i(K))$, $x^* = x_i(K)$, $time = time + 1$. Если количество раз достижения оптимального решения $time$ достигло $A = 10$, пропускаем все итерации и переходим к следующему шагу.

4. **Обновление диапазонов и критерий остановки**

   Если $K > M$ или $f^* < J_{end}$, оптимальные параметры найдены. Иначе, переходим к следующей итерации: $K = K + 1$ и обновляем диапазоны поиска:

$$a_i' = x_i^* - \frac{1}{K+1}(b_i - a_i), \quad b_i' = x_i^* + \frac{1}{K+1}(b_i - a_i)$$

   Чтобы уменьшить диапазон поиска, если $a_i' > a_i$, то $a_i = a_i'$; если $b_i' < b_i$, то $b_i = b_i'$. Переходим к шагу 2.

5. **Результат**

   Оптимальными параметрами является x^*, среднеквадратичная ошибка параметров − f^*.

3.5. Алгоритм Пауэлла

Метод Пауэлла, а точнее метод сопряженного направления Пауэлла − это алгоритм, предложенный Майклом Дж. Д. Пауэллом в 1964 году [2] для нахождения локального минимума функции. Причем функция не обязательно должна быть дифференцируемой, и никакие производные не берутся. Метод ориентирован на поиск оптимальных значений в многомерном пространстве решений, рассматривается как алгоритм локальной оптимизации без специфичных для алгоритма параметров. Это метод поиска, использующий сопряженное направление для ускорения скорости сходимости во время оптимизации. Между тем, алгоритм Пауэлла вместо случайных скачков использует стратегию двумерного поиска для минимизации целевой функции, которая может быть применена к задаче недифференцируемой оптимизации. Пусть x_0 − вектор начального состояния, а f − целевая функция. Инициализируем направления u_i

базисными векторами $u_i = e_i, i = 1, \ldots, n$. Алгоритм Пауэлла [2] состоит из следующих шагов:

1. **Инициализация**

   Определяем начальную точку $P_0 = x_i$ и допустимую ошибку ε.

2. **Минимизация**

   Находим значение $\gamma = \gamma_i$ которое минимизирует $f(P_{i-1} + \gamma_i \times u_i)$ и задаем $P_i = P_{i-1} + \gamma_i \times u_i$ для $i = 1, \ldots, n$.

3. **Обновление базисных векторов**

   Определяем $u[i+1]$, $i = 1, \ldots, n$;

   $u_n = P_n - P_0$;

   $i = i + 1$.

4. **Минимизация**

   Находим значение $\gamma = \gamma_{min}$ которое минимизирует $f(P_{i-1} + \gamma_{min} \times u_n)$ и задаем $x_i = P_0 + \gamma_{min} \times u_n$ для $i = 1, \ldots, n$.

5. **Повторение**

   Повторяем с 1 по 5 шаги пока целевая функция больше ε.

Начальные вектора алгоритма Пауэлла u_i полностью заменяются набором новых сопряженных векторов после n итераций.

3.6. Алгоритм Джая-Пауэлла

Алгоритм Джая − это алгоритм адаптивной оптимизации, который помогает потенциальным решениям приблизиться к текущему оптимуму и отойти от наихудшего решения на протяжении итераций. Основными характеристиками алгоритма Джая является то, что он не содержит специфичных для алгоритма параметров. Так как такие параметры могут значительно повлиять на производительность оптимизации и должны быть хорошо настроены. В алгоритме Джая необходимо заранее задать только размер популяции и количество итераций. Алгоритм Джая-Пауэлла, предложенный Ли Чжуаном и коллегами в 2020 году [18] состоит в том, что оптимальное решение, полученное алгоритмом Джая, также минимизируется алгоритмом Пауэлла.

1. **Определение параметров**

   Установите размер совокупности, диапазон параметров и максимальное количество итераций.

2. **Инициализация**

   Произвольно инициализируйте популяцию с определенным диапазоном параметров.

3. **Лучшие и худшие решения**

   Определите лучшие и худшие решения в текущем пространстве решений.

4. **Обновление популяции**

   Обновите текущее решение на основе лучших и худших решений в соответствии с уравнением

$$O'_{m,p,n} = O_{m,p,n} + r_{1,p,n}(O_{m,best,n} - |O_{m,p,n}|) - r_{2,p,n}(O_{m,worst,n} - |O_{m,p,n}|)$$

5. **Целевая функция**

   Оцените значение целевой функции для каждого обновленного решения. Затем обновленное решение будет принято если значение целевой функции нового решения не превосходит целевой функции существующего. Иначе решение будет отклонено.

6. **Повторение**

   Если достигнута максимальная итерация, оптимизация завершена. Иначе переходим к шагу 3.

7. **Алгоритм Пауэлла**

   Применяем алгоритм Пауэлла к оптимальному решению.

3.7. Модификация алгоритма Пауэлла: GWO-Powell

Выше мы рассмотрели алгоритм Пауэлла, использующий понятие сопряженности векторов, которые определяют направление поиска на смежных итерациях. Несмотря на важное преимущество данного метода — безусловность оптимизации, данный метод имеет особенность, которая в случае поиска параметров для метода хаотической

оптимизации является скорее недостатком — метод наиболее эффективен для функций, имеющий квадратичный вид. В таком случае он обеспечивает сходимость за число шагов, не более размерности задачи. Поэтому существуют различные модификации метода Пауэлла для оптимизации его применения к остальным функциям. Так, мы уже рассмотрели одну из модификаций — алгоритм Джая-Пауэлла.

В данной работе предлагается еще одна модификация алгоритма Пауэлла — скрещивание метода Пауэлла и алгоритма серых волков. Выбор алгоритма серых волков обоснован высокой скоростью сходимости. В свою очередь, эта комбинация алгоритмов компенсирует недостаток исследовательской составляющей в методе GWO.

Модификация состоит из следующих этапов:

1. **Определение параметров**

   Задается размер популяции, диапазон параметров и максимальное количество итераций для обоих алгоритмов.

2. **Метод серых волков**

   Находим решение с помощью метода серых волков.

3. **Метод Пауэлла**

   Используем данное решение как начальную точку поиска для метода Пауэлла.

Подробное описание методов приведены выше в работе.

Данная модификация позволяет использовать преимущества обоих методов, компенсируя их недостатки. Таким образом повышается эффективность и точность предсказание параметров хаотической динамики.

Эксперименты

Чтобы проверить эффективность предложенных методов машинного обучения и алгоритмов оптимизации параметров, мы применим комбинаций этих методов и алгоритмов к системе Лоренца и к модели лазеров с насыщающимся поглотителем.

Эксперимент состоит из следующих шагов:

1. **Инициализация**

   - Определяем нижние и верхние границы параметров метода машинного обучения
   - Задаем целевую функцию как среднеквадратичную ошибку предсказания методом машинного обучения с текущим значением параметров в алгоритме оптимизации $RMSE = \sqrt{\sum_{i=1}^{N}(y(i) - \hat{y}(i))^2/N}$, где y - реальное значение, а $\hat{y}$ - результат предсказания.

2. **Оптимизация параметров**

   Вызываем алгоритм оптимизации параметров с заданными границами параметров, количеством итераций

3. **Обучение**

   Производим обучение модели с параметрами, найденными методом оптимизации, на тренировочном наборе данных

4. **Предсказание**

   Получаем предсказание хаотической динамики для тестового набора данных и рассчитываем среднеквадратичную ошибку

4.1. Система Лоренца

Система Лоренца была введена Эдвардом Лоренцем, метеорологом Массачусетского технологического института в 1963 году [32]. Он сформулировал сильно упрощенную модель конвективной жидкости. Эта простая модель для некоторых значений парамет-

ров имеет хаотическое поведение. Система задается следующим образом:

$$\begin{cases} \dot{x} = a(y - x), \\ \dot{y} = -xz + cz - y, \\ \dot{z} = xy - bz, \end{cases} \tag{4.1}$$

где положим $a = 10,\ b = \dfrac{8}{3},\ c = 28$. При данном значении параметров у этой динамической системы наблюдается хаотическое траекторий, которые притягиваются к странному аттрактору (Рис. 4.1).

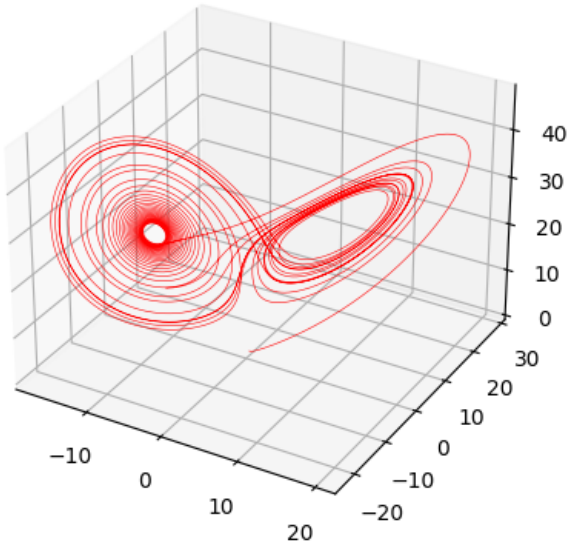

Рис. 4.1. Фазовый портрет аттрактора Лоренца для значений параметров $a = 10,\ b = \dfrac{8}{3},\ c = 28$

С помощью различных методов оптимизации параметров для алгоритмов SVR и XGBoost мы хотим предсказывать хаотическое поведение траектории системы $x,\ y,\ z$, которые выглядят следующим образом:

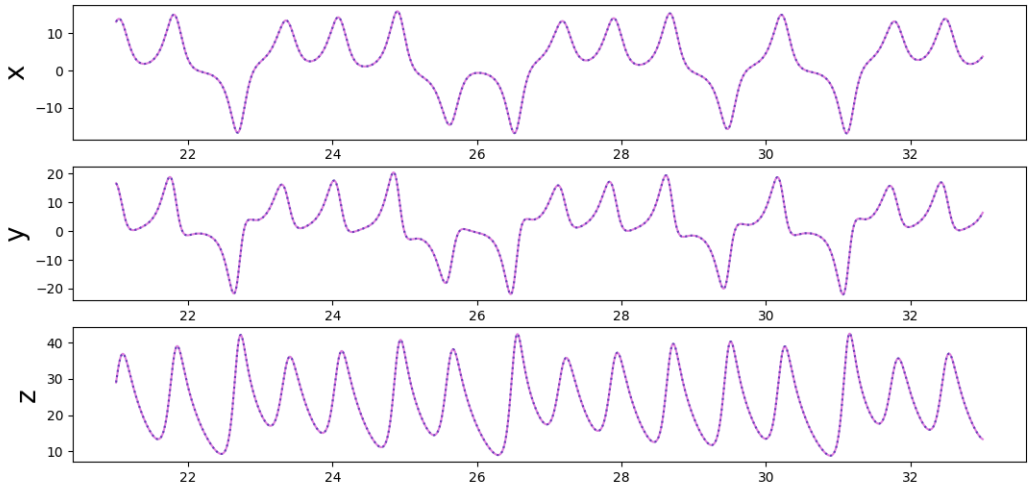

Рис. 4.2. Траекторий системы Лоренца

4.2. Хаос в модели лазеров

Динамика лазера с насыщающимся поглотителем находится в поле зрения исследователей, начиная с 1965 года [37, 38]. Также известно [39, 40], что в режиме «сверхсветового» распространения электромагнитного поля в усиливающей среде его амплитуда может иметь хаотический характер.

Простейшая модель базируется [36] на использовании уравнений одномодового лазера для амплитуды поля электромагнитной волны лазера x, поляризации активной среды y и разности концентраций частиц z на верхнем и нижнем рабочих уровнях энергии активной среды лазера:

$$\begin{cases} \dot{x} = -a(1 + \dfrac{\alpha}{1 + \eta x^2})x + ay, \\ \dot{y} = -y + xz, \\ \dot{z} = b(r - z) - xy, \end{cases} \tag{4.2}$$

где $a,\ b,\ r$ – параметры системы, а слагаемое $\dfrac{a\alpha}{1 + \eta x^2}$, описывает безынерционное насыщающееся поглощение. Режим стационарной генерации нормального лазера теряет устойчивость, если параметр возбуждения r превышает критическое значение $r^* = \dfrac{a(a + b + 3)}{a - b - 1}$. Хаотические пульсации наблюдаются при значениях параметров $a = 3,\ b = 1,\ \eta = 1000,\ r^* = 21,\ r = 22.$

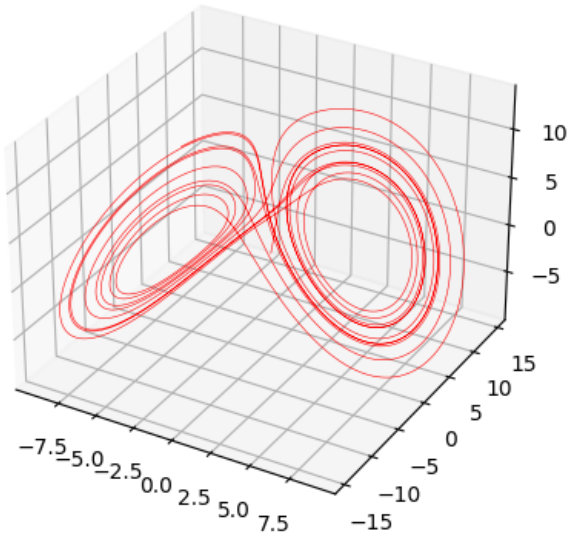

Рис. 4.3. Хаотическая динамика модели лазера с насыщающимся поглотителем

Мы проанализируем эффективность предсказания хаотических траекторий модели лазера с насыщающимся поглотителем (Рис. 4.4) с помощью различных методов оптимизации параметров для алгоритмов SVR и XGBoost:

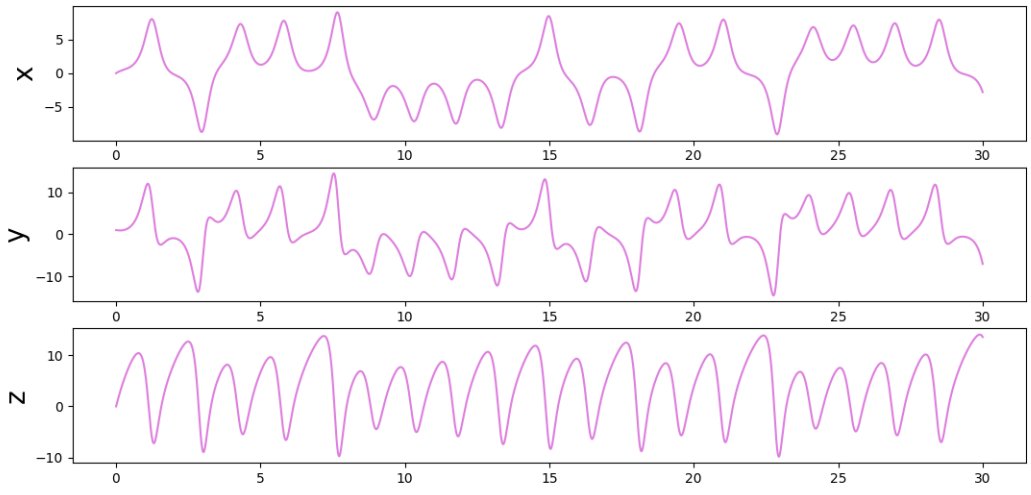

Рис. 4.4. Траекторий хаотической динамики модели лазера с насыщающимся поглотителем

Результаты

В этой работе основное внимание уделяется предсказанию хаотической динамики методами машинного обучения. Машинное обучение включает в себя огромное множество различных методов, несмотря на то, что эта научная область появилась относительно недавно. Поэтому очень важно правильно выбрать метод для конкретной задачи. Мы рассмотрели популярные методы предсказания параметров и алгоритмы машинного обучения. В данном разделе мы проанализируем эффективность этих методов по результатам проведенных экспериментов.

В таблице 5.1 представлены результаты сравнения среднеквадратичной ошибки предсказания хаотической динамики в системе Лоренца методами машинного обучения SVR и XGBoost с рассмотренными в работе алгоритмами оптимизации параметров.

	SVR	XGB
CS	0.00420411	0.00091628
GWO	0.00095896	0.00071578
PSO	0.0008838	0.00083473
COA	0.00090722	0.00756321
Powell	0.00089928	0.00071695
Jaya-Powell	0.00046758	0.00072361
GWO-Powell	0.00063655	0.00059784

Таблица 5.1. Среднеквадратичная ошибка предсказания хаотической динамики в системе Лоренца методами машинного обучения с алгоритмами оптимизации параметров.

Исходя из получившихся данных видно, что более точным методом машинного обучения для предсказания хаотической динамики является алгоритм экстремального градиентного бустинга, так как по всем методам оптимизации параметров среднеквадратичная ошибка получается меньше, чем у метода опорных векторов. При более детальном рассмотрении можно также отметить, что алгоритм кукушки при оптимизации параметров для метода опорных векторов малоэффективен, так как дает ошибку на порядок больше остальных рассмотренных методов. Лучшие результаты оптимизации параметров для метода SVR получаются в случае использования модификации алгоритма Пауэлла − Jaya-Powell и GWO-Powell. Алгоритм хаотической оптимизации

параметров для метода экстремального градиентного бустинга не так эффективен, как для метода опорных векторов. В этом случае среднеквадратичная ошибка отличается на порядок. Метод роя частиц, в целом, можно называть универсальным алгоритмом оптимизации параметров, так как в случае обоих методов машинного обучения среднеквадратичная ошибка схожа.

Проверим выводы об эффективности методов сделанные на основе результатов предсказания хаотической динамики системы Лоренца, применив те же комбинации методов машинного обучения и алгоритмов оптимизации параметров к модели лазера с насыщающимся поглотителем и проанализируем получившиеся результаты.

	SVR	XGB
CS	0.00340198	0.00164526
GWO	0.00006284	0.00005579
PSO	0.00084052	0.00063171
COA	0.00034149	0.00122632
Powell	0.00049553	0.00041838
Jaya-Powell	0.00089929	0.00063412
GWO-Powell	0.00005855	0.00002199

Таблица 5.2. Среднеквадратичная ошибка предсказания хаотической динамики в модели лазера с насыщающимся поглотителем методами машинного обучения с алгоритмами оптимизации параметров.

По результатам предсказания хаоса в модели лазера, показанных в таблице 5.2, сделаем выводы об обобщающей способности комбинаций методов машинного обучения и алгоритмов оптимизации, сравнив полученные среднеквадратичные ошибки с результатами экспериментов с системой Лоренца. Так, алгоритм кукушки при оптимизации параметров в обоих случаях даёт ошибку на порядок хуже, чем была для системы Лоренца, откуда делаем вывод о низкой обобщающей способности такой комбинации. Аналогично алгоритм оптимизации хаоса в комбинации с экстремальным градиентным бустингом оказался плохо обобщаем несмотря на высокую точность в случае комбинации с регрессией опорных векторов. Отличные результаты в свою очередь были получены для алгоритмов серых волков и модификации метода Пауэлла алгоритмом серых волков — в случае этих алгоритмов среднеквадратичная ошибка при предсказании ха-

оса модели лазера на порядок меньше предсказания хаотической динамики Лоренца, что свидетельствует о высокой обобщаемости. Остальные методы − метод роя частиц, Пауэлла и Джая-Пауэлла, показали стабильный результат. То есть применение их для предсказания хаотической динамики возможно, влияние шумов на модель минимально.

Заключение

В представленной работе были исследованы методы машинного обучения — регрессия опорных векторов и экстремальный градиентный бустинг, и алгоритмы оптимизации их параметров для предсказания хаотической динамики — алгоритм кукушки, алгоритм серых волков, метод роя частиц, алгоритм хаотической оптимизации, алгоритм Пауэлла и алгоритм Джая-Пауэлла. Основные результаты работы заключаются в следующем:

1. Предложена модификация алгоритма Пауэлла с помощью алгоритма серых волков. По результатам экспериментов данная модификация оказалась самой эффективной и обобщаемой из всех рассматриваемых алгоритмов оптимизации.

2. Реализован программный пакет для применения комбинаций изученных и разработанных методов машинного обучения и алгоритмов оптимизации их параметров.

В работе также показано, что, несмотря на универсальность и производительность методов машинного обучения для предсказания хаотической динамики, огромную роль при построении модели машинного обучения играет выбор алгоритма оптимизации параметров. Полученные в работе результаты позволяют реализовать эффективную модель машинного обучения в задачах предсказания хаотической динамики. В рамках дальнейшего исследования планируется создание новых модификаций алгоритмов оптимизации параметров.

Список литературы

1. Андронов А.А., Хайкин С.Э. Теория колебаний. М.–Л.: ОНТИ, 1937.
2. M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives (англ.) // The Computer Journal. — 1964. — 1 January (vol. 7). — P. 155–162.
3. Андронов А.А., Витт А.А., Хайкин С.Э. Теория колебаний. 2-е изд. М.: Физматлит, 1959.
4. J. Kennedy, R. Eberhart, Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks IV: 1942-1948.
5. Y. Shi, R. Eberhart, A modified particle swarm optimizer. Proceedings of IEEE International Conference on Evolutionary Computation: 69-73.
6. Kennedy, J.; Eberhart, R.C., Swarm Intelligence — Morgan Kaufmann, 2001.
7. Hu X., Eberhart R. Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization, 6th World Multiconference Syst. Cybern. Informatics. Orlando, Florida, USA, 2002. P. 203–206.
8. Poli, R. Analysis of the publications on the applications of particle swarm optimisation, Journal of Artificial Evolution and Applications — 2008.
9. C. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model Computer Graphics, 21(4), стр. 25–34, 1987.
10. Hu X., Eberhart R. Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization // 6th World Multiconference Syst. Cybern. Informatics. Orlando, Florida, USA, 2002. P. 203–206.
11. X.-S. Yang; S. Deb (December 2009). Cuckoo search via Lévy flights. World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). IEEE Publications. pp. 210–214.
12. Mirjalili S., Mirjalili S. M., Lewis A., Grey wolf optimizer, Advances in Engineering Software. – 2014. – vol. 69, pp. 46–61. 2.
13. Long W., Cai S., Jiao J., Tang M., An efficient and robust grey wolf optimizer algorithm for large-scale numerical optimization, Soft Computing. – 2019. – vol. 3.
14. Д.В. Ежов, О. В. Коваленко, Параллельный алгоритм оптимизации непрерывных функций большой размерности на основе гибридизации алгоритма оптимизации «се-

рых волков», Научный сервис в сети Интернет: труды XXII Всероссийской научной конференции (21-25 сентября 2020 г., онлайн), стр. 255-267.

15. В.А. Частикова, С.А. Жерлицын, Исследование алгоритма серых волков, Научные труды КубГТУ, № 16, 2016.
16. Cortes C, Vapnik V N, "Support Vector Networks Machine Learning, Vol.20, no.3, pp273-295, September 1995.
17. Yuxia Hu, Hongtao Zhang, "Chaos Optimization Method of SVM Parameters Selection for Chaotic Time Series Forecasting Physics Procedia, 2012, vol.25, pp. 588-594
18. L. Zhuang et al., "Parameter Estimation of Lorenz Chaotic System Based on a Hybrid Jaya-Powell Algorithm,"in IEEE Access, vol. 8, pp. 20514-20522, 2020
19. F. Takens, in: D. Rand, L.-S. Young (Eds.), Dynamical Systems and Turbulence, Warwick, 1980, Springer, Berlin, 1981, pp366.
20. Arcomano, T., Szunyogh, I., Pathak, J., Wikner, A., Hunt, B. R., & Ott, E. (2020). A machine learning-based global atmospheric forecast model. Geophysical Research Letters, 47.
21. T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 785–794.
22. S.-H. Wang, H.-T. Li, E.-J. Chang, A.-Y. A. Wu, Entropy-assisted emotion recognition of valence and arousal using xgboost classifier, in: IFIP International Conference on Artificial Intelligence Applications and Innovations, Springer, 2018, pp. 249–260.
23. A.B. Parsa, A. Movahedi, H. Taghipour, S. Derrible, A.K. Mohammadian, Toward safer highways, application of xgboost and shap for real-time accident detection and feature analysis, Accident Analysis & Prevention 136 (2020) 105405.
24. Q. Huang, W. Xie, “The prediction of short-term exchange rate based on the phase space reconstruction and kalman”, Computer Applications and Software, vol.25, no.8, pp79-80, August 2008.
25. Z. Zhang, Y. Sun, “A new approach of STLF based on combination of phase space reconstruction theory and optimal recursive neural networks,” Electric Power, vol.37, no. 1, pp19-23, January 2004.
26. Z. Shan, C. Lin, “Chaos Prediction of Wave Hydrodynamic Pressure Signals Based on Local Support Vectors Machine,” Journal of System Simulation, 2008, vol.20, no.23,

pp6470-6472, December 2008.

27. G. Wu, P. Zhou, “Prediction Based on Phrase Construction and Its Application in Weather Forecast,” Chinese journal of nature, vol. 21, no. 2, pp107-110, April 999.
28. Cortes C, Vapnik V N, “Support Vector Networks,” Machine Learning, Vol.20, no.3, pp273-295, September 1995.
29. C. Hsu, C. Chang , C. Lin, “A practical guide to support vector classification”, http://www.csie.ntu.edu.twcjlin/papem/guide/guide.pdf, 2003
30. X. Yuan, Y. Wang, “Parameter selection of support vector machine for functionapproximation based on chaos optimization”, Journal of Systems Engineering and Electronics, Vol.19, no.1, pp191-97, January 2008.
31. Q. Liu, P. Que, C. Fei, S. Song, “Model selection for SVM using mutative scale chaos optimization algorithm”, Journal of Shanghai University, Vol. 10, no. 6,pp531-534,June 2006.
32. E. N. Lorenz, "Deterministic nonperiodic flow J. Atmos. Sci., vol. 20, no. 2, pp. 130-141, 1963.
33. Matsumoto T., Chua L., Komuro M. The Double Scroll, IEEE "Transactions on Circuits and Systems". 1985. V. 32 (8). P. 797–818.
34. B. Peng, B. Liu, F.-Y. Zhang, and L. Wang, "Differential evolution algorithm-based parameter estimation for chaotic systems Chaos, Solitons Fractals, vol. 39, no. 5, pp. 2110-2118, Mar. 2009.
35. Y. Tang and X. Guan, "Parameter estimation for time-delay chaotic system by particle swarm optimization,” Chaos, Solitons Fractals, vol. 40, no. 3, pp. 1391-1398, May 2009.
36. А. Н. Ораевский, Квантовая электроника, 1999, том 26, номер 3, 214–216
37. Ривлин Л.А. ЖЭТФ, 47, 624 (1965).
38. Воропаев Н.Д., Ораевский А.Н. Изв. вузов. Сер. Радиофизика, 8, 409 (1965).
39. Ораевский А.Н., Бенди Д.К. Письма в ЖЭТФ, 59, 319 (1994).
40. Ораевский А.Н., Бенди Д.К. Квантовая электроника, 24, 331 (1994).

Программный пакет методов машинного обучения и алгоритмов оптимизации параметров

Листинг А.1. **cs.py – алгоритм кукушки**

```
import math
import random
import numpy


def get_cuckoos(nest, best, lb, ub, n, dim):
    tempnest = numpy.array(nest)
    beta = 3 / 2
    sigma = (math.gamma(1 + beta) * math.sin(math.pi * beta / 2) / (
                math.gamma((1 + beta) / 2) * beta * 2 ** ((beta - 1)
   / 2))) ** (1 / beta)

    for j in range(0, n):
        s = nest[j, :]
        u = numpy.random.randn(len(s)) * sigma
        v = numpy.random.randn(len(s))
        step = u / abs(v) ** (1 / beta)
        stepsize = 0.01 * (step * (s - best))
        s = s + stepsize * numpy.random.randn(len(s))

        for k in range(dim):
            tempnest[j, k] = numpy.clip(s[k], lb[k], ub[k])

    return tempnest


def get_best_nest(nest, newnest, fitness, n, dim):
    tempnest = numpy.copy(nest)

```

```
29      for j in range(0, n):
30          fnew = rmse_func(newnest[j, :])
31          if fnew == 1000:
32              continue
33          if fnew <= fitness[j]:
34              fitness[j] = fnew
35              tempnest[j, :] = newnest[j, :]
36
37      fmin = min(fitness)
38      K = numpy.argmin(fitness)
39      bestlocal = tempnest[K, :]
40
41      return fmin, bestlocal, tempnest, fitness
42
43
44  def empty_nests(nest, pa, n, dim):
45      K = numpy.random.uniform(0, 1, (n, dim)) > pa
46      stepsize = random.random() * (nest[numpy.random.permutation(n),
    :] - nest[numpy.random.permutation(n), :])
47      tempnest = nest + stepsize * K
48
49      return tempnest
50
51
52  def CS(lb, ub, dim, n, N_IterTotal):
53      pa = 0.25
54      if not isinstance(lb, list):
55          lb = [lb] * dim
56      if not isinstance(ub, list):
57          ub = [ub] * dim
58
59      nest = numpy.zeros((n, dim))
60      for i in range(dim):
```

```
61          nest[:, i] = numpy.random.uniform(0, 1, n) * (ub[i] - lb[i])
    + lb[i]
62
63      new_nest = numpy.copy(nest)
64      fitness = numpy.zeros(n)
65      fitness.fill(float("inf"))
66
67      print("CS is optimizing")
68
69      fmin, bestnest, nest, fitness = get_best_nest(nest, new_nest,
    fitness, n, dim)
70      for iter in range(0, N_IterTotal):
71          new_nest = get_cuckoos(nest, bestnest, lb, ub, n, dim)
72          fnew, best, nest, fitness = get_best_nest(nest, new_nest,
    fitness, n, dim)
73          new_nest = empty_nests(new_nest, pa, n, dim)
74          fnew, best, nest, fitness = get_best_nest(nest, new_nest,
    fitness, n, dim)
75
76          if fnew < fmin:
77              fmin = fnew
78              bestnest = best
79
80          print(['At iteration ' + str(iter) + ' the best fitness is '
    + str(fmin)])
81
82      return best
```

Листинг А.2. **gwo.py – алгоритм серых волков**

```
1  import random
2  import numpy
3
4
5  def GWO(lb, ub, dim, SearchAgents_no, Max_iter):
```

```
    Alpha_pos = numpy.zeros(dim)
    Alpha_score = float("inf")

    Beta_pos = numpy.zeros(dim)
    Beta_score = float("inf")

    Delta_pos = numpy.zeros(dim)
    Delta_score = float("inf")

    if not isinstance(lb, list):
        lb = [lb] * dim
    if not isinstance(ub, list):
        ub = [ub] * dim

    Positions = numpy.zeros((SearchAgents_no, dim))
    for i in range(dim):
        Positions[:, i] = numpy.random.uniform(0, 1, SearchAgents_no)
    * (ub[i] - lb[i]) + lb[i]

    Convergence_curve = numpy.zeros(Max_iter)
    print("GWO is optimizing")

    for l in range(0, Max_iter):
        for i in range(0, SearchAgents_no):
            for j in range(dim):
                Positions[i, j] = numpy.clip(Positions[i, j], lb[j],
    ub[j])

            fitness = rmse_func(Positions[i, :])

            if fitness < Alpha_score:
                Alpha_score = fitness
                Alpha_pos = Positions[i, :].copy()

```

```
            if Alpha_score < fitness < Beta_score:
                Beta_score = fitness
                Beta_pos = Positions[i, :].copy()

            if Alpha_score < fitness < Delta_score and fitness > Beta_score:
                Delta_score = fitness
                Delta_pos = Positions[i, :].copy()

        a = 2 - l * ((2) / Max_iter)

        for i in range(0, SearchAgents_no):
            for j in range(0, dim):
                r1 = random.random()
                r2 = random.random()

                A1 = 2 * a * r1 - a
                C1 = 2 * r2

                D_alpha = abs(C1 * Alpha_pos[j] - Positions[i, j])
                X1 = Alpha_pos[j] - A1 * D_alpha

                r1 = random.random()
                r2 = random.random()

                A2 = 2 * a * r1 - a
                C2 = 2 * r2

                D_beta = abs(C2 * Beta_pos[j] - Positions[i, j])
                X2 = Beta_pos[j] - A2 * D_beta

                r1 = random.random()
                r2 = random.random()

```

```
                A3 = 2 * a * r1 - a
                C3 = 2 * r2

                D_delta = abs(C3 * Delta_pos[j] - Positions[i, j])
                X3 = Delta_pos[j] - A3 * D_delta

                Positions[i, j] = (X1 + X2 + X3) / 3

        Convergence_curve[l] = Alpha_score

        print(['At iteration ' + str(l) + ' the best fitness is ' +
    str(Alpha_score)])

    return Alpha_pos
```

Листинг А.3. **pso.py – метод роя частиц**

```
import random
import numpy


def PSO(lb, ub, dim, PopSize, iters):
    Vmax = 6
    wMax = 0.9
    wMin = 0.2
    c1 = 2
    c2 = 2
    if not isinstance(lb, list):
        lb = [lb] * dim
    if not isinstance(ub, list):
        ub = [ub] * dim

    vel = numpy.zeros((PopSize, dim))
    pos = numpy.zeros((PopSize, dim))

```

```
    pBestScore = numpy.zeros(PopSize)
    pBestScore.fill(float("inf"))

    pBest = numpy.zeros((PopSize, dim))

    gBest = numpy.zeros(dim)
    gBestScore = float("inf")

    for i in range(dim):
        pos[:, i] = numpy.random.uniform(0, 1, PopSize) * (ub[i] - lb
[i]) + lb[i]

    print("PSO is optimizing")

    for l in range(0, iters):
        for i in range(0, PopSize):
            for j in range(dim):
                pos[i, j] = numpy.clip(pos[i, j], lb[j], ub[j])

            fitness = rmse_func(pos[i, :])

            if pBestScore[i] > fitness:
                pBestScore[i] = fitness
                pBest[i, :] = pos[i, :].copy()

            if gBestScore > fitness:
                gBestScore = fitness
                gBest = pos[i, :].copy()

        w = wMax - l * ((wMax - wMin) / iters)

        for i in range(0, PopSize):
            for j in range(0, dim):
                r1 = random.random()
```

```
52                 r2 = random.random()
53                 vel[i, j] = w * vel[i, j] + c1 * r1 * (pBest[i, j] -
    pos[i, j]) + c2 * r2 * (gBest[j] - pos[i, j])
54
55                 if vel[i, j] > Vmax:
56                     vel[i, j] = Vmax
57
58                 if vel[i, j] < -Vmax:
59                     vel[i, j] = -Vmax
60
61                 pos[i, j] = pos[i, j] + vel[i, j]
62
63         print(['At iteration ' + str(l + 1) + ' the best fitness is '
    + str(gBestScore)])
64
65     return gBest
```

Листинг А.4. **coa.py – алгоритм хаотической оптимизации**

```
1  import sys
2  import numpy as np
3  from sklearn.svm import SVR
4
5  N = 1000
6  m = 4
7  mu = 3.6
8  tau = 17
9  k = np.random.randint(0, N)
10 M = N - (m - 1) * tau
11 T = 50
12 dt = 0.001
13 x0 = np.array([0, 1, 0])
14 t = np.arange(0.001, T, dt)
15 a = np.array([2 ** -5, 2 ** -12, 2 ** -15])
16 b = np.array([2 ** 15, 2 ** 0, 2 ** 5])
```

```
A = 10
Z = np.zeros(N)


def coa(rmse, x, Z):
    X = reconstruction(x)
    sys.setrecursionlimit(2000)
    init_Z(N - 1)
    times = 0
    _x = a + (b - a) * Z[np.random.randint(N)]
    predicted_x = predict(_x)
    _f = rmse(np.stack(X[train:])[:, 0], predicted_x)
    K = 0
    while _f > 0.001 and K < N:
        x_p = a + (b - a) * Z[K]

        predicted_x = predict(x_p)
        f = rmse(np.stack(X[train:])[:, 0], predicted_x)
        if f <= _f:
            _f = f
            _x = x_p
            times += 1
        if times > A:
            times = 0
            continue

        K += 1
        _a = _x - (b - a) / (K + 1)
        _b = _x + (b - a) / (K + 1)
        for j in range(len(a)):
            if _a[j] > a[j]:
                a[j] = _a[j]
            if _b[j] < b[j]:
                b[j] = _b[j]
```

```
    return _x


def reconstruction(x):
    return [x[i: i + (m - 1) * tau + 1: tau, 0][::-1] for i in range(
   M)]


def init_Z(k):
    if k == 0:
        Z[k] = 0.001
    else:
        Z_previous = init_Z(k - 1)
        Z[k] = mu * Z_previous * (1 - Z_previous)
    return Z[k]


def predict(params):
    svr_rbf = SVR(kernel='rbf', C=params[0], gamma=params[1], epsilon
   =params[2])
    svr_rbf.fit(np.stack(X[:train]), np.stack(X[:train])[:, 0])
    return svr_rbf.predict(np.stack(X[train:]))
```

Листинг А.5. **powell.py – метод Пауэлла**

```
import numpy as np


def goldenSectionSearch(f, x, u, a, c, b, absolutePrecision=1e-4):
    phi = ((1 + 5 ** 0.5) / 2)
    resphi = 2 - phi

    if abs(a - b) < absolutePrecision:
        return (a + b)/2
    d = c + resphi*(b - c)
```

```
    if f(d, x, u) < f(c, x, u):
        return goldenSectionSearch(f, x, u, c, d, b,
    absolutePrecision)
    else:
        return goldenSectionSearch(f, x, u, d, c, a,
    absolutePrecision)


def powell(F, x, lower_b, upper_b):
    def f(_gamma, _x, _v):
        return F(_x + _gamma * _v)

    n = len(x)
    df = np.zeros(n)
    u = np.identity(n)
    p = np.zeros((n, n))
    _f = np.zeros(n)
    fMin = 1
    for j in range(N):
        p0 = x
        for k in range(n):
            if F(x) < 0.001:
                return x

            for i in range(n):
                v = u[i]
                if i == 0:
                    gamma = goldenSectionSearch(f, p0, v, lower_b[i],
    x[i], upper_b[i])
                    fMin = F(p0 + gamma * v)
                    p[i] = p0 + gamma * v
                else:
                    gamma = goldenSectionSearch(f, p[i - 1], v,
    lower_b[i], x[i], upper_b[i])
```

```
                    fMin = F(p[i - 1] + gamma * v)
                    p[i] = p[i - 1] + gamma * v

            if k == 0:
                df[k] = abs(F(p[k]) - F(p0))
            else:
                df[k] = abs(F(p[k]) - F(p[k-1]))

            r = np.argmax(df)
            df_max = df[r]

            f0 = F(p0)
            _f[k] = F(p[k])
            f_E = F(2 * p[n - 1] - p[k])
            f_n = F(p[n - 1])

            if f_E > f0 or 2*(f0 - 2*f_n - f_E) * (f0 - f_n - df_max)
**2 >= 0.5 * df_max * (f0 - f_E)**2:
                _u = p[n - 1] - p[k]
                gamma = goldenSectionSearch(f, p0, _u, lower_b[k], p0
[k], upper_b[k])
                fMin = F(p0 + gamma * _u)
                for i in range(r, n - 1):
                    u[i] = u[i + 1]
                u[n - 1] = _u
                x = p[n - 1] + gamma * _u
                p0 = x
            else:
                x = p[n - 1]
                p0 = x

        print(['At iteration ' + str(j) + ' the best fitness is ' +
str(fMin)])

```

```
72      return x
```

Листинг А.6. **jaya-powell.py – метод Джая-Пауэлла**

```
1  import random
2  import numpy as np
3  import powell
4
5
6  def jayaPowell(lb, ub, dim, SearchAgents_no, Max_iter):
7      x = jaya(lb, ub, dim, SearchAgents_no, Max_iter)
8      minimized_x = powell(rmse_func, x, lb, ub)
9      return minimized_x
10
11
12 def jaya(lb, ub, dim, SearchAgents_no, Max_iter):
13     Best_pos = np.zeros(dim)
14     Best_score = float("inf")
15
16     Worst_pos = np.zeros(dim)
17     Worst_score = float(0)
18
19     fitness_matrix = np.zeros(SearchAgents_no)
20
21     if not isinstance(lb, list):
22         lb = [lb] * dim
23     if not isinstance(ub, list):
24         ub = [ub] * dim
25
26     Positions = np.zeros((SearchAgents_no, dim))
27     for i in range(dim):
28         Positions[:, i] = np.random.uniform(0, 1, SearchAgents_no) *
    (ub[i] - lb[i]) + lb[i]
29
30     for i in range(0, SearchAgents_no):
```

```
        for j in range(dim):
            Positions[i, j] = np.clip(Positions[i, j], lb[j], ub[j])

        fitness = rmse_func(Positions[i])
        fitness_matrix[i] = fitness

        if fitness < Best_score:
            Best_score = fitness
            Best_pos = Positions[i]

        if fitness > Worst_score:
            Worst_score = fitness
            Worst_pos = Positions[i]

    print("JAYA is optimizing")

    for l in range(0, Max_iter):
        for i in range(0, SearchAgents_no):
            New_Position = np.zeros(dim)
            for j in range(0, dim):

                r1 = random.random()
                r2 = random.random()

                New_Position[j] = Positions[i][j] + r1 * (Best_pos[j]
- abs(Positions[i, j])) - r2 * (
                        Worst_pos[j] - abs(Positions[i, j]))

                if New_Position[j] > ub[j]:
                    New_Position[j] = ub[j]
                if New_Position[j] < lb[j]:
                    New_Position[j] = lb[j]

            new_fitness = rmse_func(New_Position)
```

```
64              current_fit = fitness_matrix[i]
65
66              if new_fitness < current_fit:
67                  Positions[i] = New_Position
68                  fitness_matrix[i] = new_fitness
69
70          for i in range(SearchAgents_no):
71              if fitness_matrix[i] < Best_score:
72                  Best_score = fitness_matrix[i]
73                  Best_pos = Positions[i, :].copy()
74
75              if fitness_matrix[i] > Worst_score:
76                  Worst_score = fitness_matrix[i]
77                  Worst_pos = Positions[i, :].copy()
78
79          print(['At iteration ' + str(l) + ' the best fitness is ' +
    str(Best_score)])
80
81      return Best_pos
```

Листинг А.7. **gwo-powell.py – разработанная модификация метода Пауэлла с помощью алгоритма серых волков**

```
1  import gwo
2  import powell
3
4
5  def gwoPowell(lb, ub, dim, SearchAgents_no, Max_iter):
6      x_init = gwo(lb, ub, dim, SearchAgents_no, Max_iter)
7      x_opt = powell(rmse_func, x_init, lb, ub)
8      return x_opt
```

Листинг А.8. **rmse_func_svr.py – метод вычисления среднеквадратичной ошибке для модели машинного обучения на основе регрессии опорных векторов**

```
1  from sklearn.metrics import mean_squared_error
2  from sklearn.svm import SVR
```

```
3
4
5  def rmse_func(_x):
6      # print(x)
7      svr_rbf = SVR(kernel='rbf', C=_x[0], gamma=_x[1], epsilon=_x[2])
8      svr_rbf.fit(x_train, y_train)
9      prediction = svr_rbf.predict(x_test)
10     f = mean_squared_error(y_test, prediction, squared=False)
11
12     return f
```

Листинг А.9. **rmse_func_xgb.py – метод вычисления среднеквадратичной ошибке для модели машинного обучения на основе экстремального градиентного бустинга**

```
1  import xgboost as xgb
2  from sklearn.metrics import mean_squared_error
3
4  def rmse_func(_x):
5      lambda1 = float(_x[len(_x) - 1])
6      alpha = float(_x[len(_x) - 2])
7      bylevel = float(_x[len(_x) - 3])
8      bytree = float(_x[len(_x) - 4])
9      subsample1 = float(_x[len(_x) - 5])
10     minChWeight = int(_x[len(_x) - 6])
11     gamma1 = float(_x[len(_x) - 7])
12     NoEstimator = int(_x[len(_x) - 8])
13     learnRate = float(_x[len(_x) - 9])
14     maxDepth = int(_x[len(_x) - 10])
15
16     xgbbp = xgb.XGBRegressor(max_depth=maxDepth, n_estimators=
       NoEstimator, learning_rate=learnRate,
17                              gamma=gamma1, min_child_weight=
       minChWeight, subsample=subsample1,
18                              colsample_bytree=bytree,
       colsample_bylevel=bylevel,
```

```
                                reg_alpha=alpha, reg_lambda=lambda1,
    silent=True, verbosity=0, verbose=True)

     xgbbp.fit(x_train, y_train)
     prediction = xgbbp.predict(x_test)
     f = mean_squared_error(y_test, prediction, squared=False)

     return f
```

Листинг А.10. **lorenz.py – траектории системы Лоренца**

```
import numpy as np
from scipy import integrate


def lorenz(x, dx, sigma=10., rho=28., beta=8 / 3.):
    dx = np.zeros((3))
    dx[0] = sigma * (x[1] - x[0])
    dx[1] = x[0] * (rho - x[2]) - x[1]
    dx[2] = x[0] * x[1] - beta * x[2]

    return dx


def lorenz_x(x0, T, dt, sigma=10., rho=28., beta=8 / 3.):
    return integrate.odeint(lorenz, x0, np.arange(0.01, T, dt), args
    =(sigma, rho, beta))
```

Листинг А.11. **laser.py – траектории модели лазеров с насыщающимся поглотителем**

```
import numpy as np
from scipy import integrate


def laser(x, dx, a=3., b=1., eta=1000., r=22.):
    dx = np.zeros(3)
```

```
7       dx[0] = -a * (1 + a / (1 + eta * x[0] * x[0])) * x[0] + a * x[1]
8       dx[1] = -x[1] + x[0] * x[2]
9       dx[2] = b * (r - x[2]) - x[0] * x[1]
10      return dx
11
12
13  def laser_x(x0, T, dt, a=3., b=1., eta=1000., r=22.):
14      return integrate.odeint(laser, x0, np.arange(0.01, T, dt), args=(
        a, b, eta, r))
```