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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №8 по дисциплине "Методы машинного обучения"

Тема Эволюционные алгоритмы

Студент Варламова Е. А.

Группа ИУ7-23М

Оценка (баллы) _____

Преподаватели Солодовников Владимир Игоревич

Москва — 2024 г.

СОДЕРЖАНИЕ

1 | Теоретическая часть

Генетические и эволюционные алгоритмы представляют собой набор методов оптимизации, вдохновленных принципами естественного отбора и генетики. Они широко используются для решения сложных задач оптимизации, в моделировании, в финансовой математике, инженерии, компьютерной графике и дизайне.

Целью данной лабораторной работы является создание программы, которая с использованием генетического алгоритма оптимизации аппроксимирует заданную функцию полиномом. Для этого необходимо решить следующие задачи:

— описать предлагаемый генетический алгоритм (выбранные функции мутаций, скрещивания и пр.);

— привести особенности реализации ПО, решающего поставленную задачу;

— провести сравнение результатов работы генетического алгоритма с результатами, полученными методом наименьших квадратов;

— исследовать зависимость ошибки аппроксимации от количества эпох и степени полинома.

1.1 Постановка задачи

Феномен Рунге – это эффект нежелательных осцилляций, возникающий при использовании полиномов высоких степеней для интерполяции.

Функция:

$$y(x) = \frac{1}{1 + 25x^2}, x \in [-2, 2] \tag{1.1}$$

Обучающая выборка:

$$S_l : x_i = \frac{4(i-1)}{l-1} - 2, i = 1, \ldots, l \tag{1.2}$$

Контрольная выборка:

$$S_k : x_i = \frac{4(i-0.5)}{l-1} - 2, i = 1, \ldots, l-1. \tag{1.3}$$

Для оптимальной степени полинома, найденной в ЛР1 (феномен Рунге), найти значения коэффициентов с использованием эволюционного алгоритма.

Визуализировать процесс обучения по эпохам. Сравнить найденные значения с результатами, полученными методом наименьших квадратов.

1.2 Генетический алгоритм

Генетический алгоритм является метаэвристическим методом оптимизации, вдохновленным естественным отбором и генетикой. Он используется для решения задач оптимизации и поиска, особенно в случаях, когда пространство поиска сложно или многомерно.

Опишем основные понятия, используемые в генетическом алгоритме.

— Популяция: Генетический алгоритм работает с популяцией индивидуумов (потенциальных решений). Популяция состоит из множества особей, представляющих потенциальные решения задачи оптимизации.

— Генотип: Генотип представляет генетическое кодирование индивидуума в терминах хромосом, генов и аллелей (фактически – размерность решения).

— Функция приспособленности: Каждый индивидуум оценивается по тому, насколько хорошо он соответствует целевой функции или заданной цели. При работе с оптимизационными задачами, функция приспособленности разрабатывается для оценки того, насколько хорошо решение соответствует желаемому результату.

— Количество поколений: определяет время жизни популяции и представляет собой количество итераций прохода по основным этапам алгоритма.

Опишем основные этапы генетического алгоритма с указанием функций, используемых в них в данной работе:

1. Инициализация: На этапе инициализации создается начальная популяция индивидуумов. Индивидуумы обычно инициализируются случайным образом или с помощью некоторых эвристик, зависящих от конкретной задачи.

2. Оценка приспособленности: Каждый индивидуум из начальной популяции оценивается с помощью функции приспособленности. Функция приспособленности используется для определения того, насколько хорошо решение соответствует цели оптимизации.

В данной работе используется функция:

$$fitness = \frac{1}{\frac{1}{N}\sum_{i=1}^{N}(y_i - y_i')^2}, \tag{1.4}$$

где

— $fitness$ – оценка того, насколько решение подходит цели оптимизации; обратно пропорционально значению ошибки;

— y_i, y_i' $i \in [1; N]$ – координаты точек обучающей выборки и предсказанные координаты точек обучающей выборки с помощью подобранных коэффициентов полинома.

3. Отбор (селекция): На этом этапе выбираются индивидуумы, которые будут иметь право размножаться и передавать свои гены в следующее поколение. Чем выше приспособленность индивидуума, тем выше вероятность его отбора.

   В данной работе используется метод селекции «стохастически равный отбор», так как он гарантирует отбор не только индивида с большой приспособленностью, но и других особей с меньшей приспособленностью, что обеспечивает более равномерный отбор и сохраняет разнообразие популяции.

4. Скрещивание: Выбранные родительские индивидуумы комбинируют свои генотипы с использованием операторов скрещивания (кроссинговера), создавая потомство, которое, в свою очередь, составит новое поколение.

   В данной работе используется оператор скрещивания «одноточечное скрещивание», так как он позволяет частично сохранить структуру геномов родителей, при этом эффективно объединив генетическую информациию.

5. Мутация: На этом этапе случайным образом изменяются гены у некоторых индивидуумов популяции. Мутация помогает сохранить генетическое разнообразие в популяции, предотвращая сходимость к локальным оптимумам.

   В данной работе используется случайная мутация, что позволяет достичь более разноообразных результатов. Процент генов, которые подвергаются мутации – 40%.

6. Оценка нового поколения: Новое поколение индивидуумов оценивается с использованием функции приспособленности. Этот этап определяет, насколько успешно индивидуумы нового поколения решают задачу оптимизации.

7. Выбор лучших особей для формировнаия нового поколения: Выбераются лучшие особи из нового поколения для создания новой популяции. При этом в новую популяцию также могут быть помещены лучшие представители старого поколения.

   В данной работе лучшие родители помещаются в новую популяцию.

8. Завершение: Работа генетического алгоритма может завершиться по истечении определенного количества поколений, достижении желаемого критерия останова, либо при достижении определенной структуры популяции.

2 | Практическая часть

2.1 Выбор средств разработки

В качестве языка программирования был использован язык Python, поскольку этот язык кроссплатформенный и для него разработано огромное количество библиотек и модулей, решающих разнообразные задачи.

В частности, имеются библиотеки, включающие в себя генетический алгоритм.

В данной работе была использована библиотека «pygad» [1], в которой реализован генетический алгоритм.

Для создания графиков была выбрана библиотека matplotlib [2], доступная на языке Python, так как она предоставляет удобный интерфейс для работы с данными и их визуализации.

2.2 Исследование ПО

В листинге 2.1 представлен код, решающий задачу аппрксимации полиномом с помощью генетического алгоритма.

Листинг 2.1: Код аппроксимации

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pygad
from sklearn.model_selection import train_test_split
import imageio.v2 as imageio
import os
def true_function(x):
    return 1 / (1 + 25 * x**2)

l = 21
X_train = np.array([4 * (i - 1) / (l - 1) - 2 for i in range(1, l + 1)]).
    reshape(-1, 1)
```

```
X_control = np.array([4 * (i - 0.5) / (l - 1) - 2 for i in range(1, l)]).
    reshape(-1, 1)
y_train = true_function(X_train)
y_control = true_function(X_control)

def fit_polynomial_regression(X, y, degree):
    poly_features = PolynomialFeatures(degree=degree)
    X_poly = poly_features.fit_transform(X)
    model = LinearRegression()
    model.fit(X_poly, y)
    return model, poly_features


def calculate_error(model, poly_features, X, y):
    X_poly = poly_features.transform(X)
    y_pred = model.predict(X_poly)
    return mean_squared_error(y, y_pred), y_pred


def find_best_deg():
    degrees = np.arange(1, 21)
    train_errors = []
    control_errors = []

    for degree in degrees:
        model, poly_features = fit_polynomial_regression(X_train, y_train,
            degree)
        train_error, y_p_t = calculate_error(model, poly_features, X_train,
            y_train)
        control_error, y_p_c = calculate_error(model, poly_features,
            X_control, y_control)
        train_errors.append(train_error)
        control_errors.append(control_error)
        if degree == 16 or degree == 20 or degree == 13 or degree == 10 or
            degree == 5:
            plt.plot(X_train, y_train, label = "train")
            plt.plot(X_control, y_control, label = "control")
            plt.plot(X_train, y_p_t, label = "predicted train")
            plt.plot(X_control, y_p_c, label = "predicted control")
            plt.legend()
            plt.title('                ')
            plt.savefig(str(degree) + ".png")
            plt.clf()

    plt.plot(degrees, train_errors, label='
        ')
    plt.plot(degrees, control_errors, label='
                        ')
    plt.xlabel('                                    ')
    plt.ylabel('                                  ')
```

```
58      plt.title('
                                                ')
59      plt.legend()
60      plt.show()
61
62
63      optimal_degree = degrees[np.argmin(control_errors)]
64      print(f'Optimal polynomial degree for approximation: {optimal_degree}')
65
66  def fitness_func(ga_instance, solution, solution_idx):
67      y_pred = np.polyval(solution, X_train)
68      return 1 / mean_squared_error(y_train, y_pred)
69  def GA(deg):
70      ga_instance = pygad.GA(num_generations=400, num_parents_mating=10,
71                             fitness_func=fitness_func, sol_per_pop=20,
72                             save_best_solutions=True,
73                             num_genes=deg + 1, gene_type=float,
74                             crossover_type = "single_point",
75                             mutation_type = "random",
76                             mutation_percent_genes = 40,
77                             parent_selection_type = "sss",
78                             keep_parents = 1)
79      ga_instance.run()
80      errs_train = []
81      errs_test = []
82      gens = []
83      images = []
84      for generation in range(400):
85          gens.append(generation)
86          best_solution = ga_instance.best_solutions[generation]
87
88          plt.plot(X_train, true_function(X_train), label="
                                                    (                    )")
89          plt.plot(X_train, np.polyval(best_solution, X_train),  label="
                                          -           (                    )")
90
91          plt.plot(X_control, true_function(X_control), label="
                                                    (                  )")
92          plt.plot(X_control, np.polyval(best_solution, X_control), label="
                                          -           (                    )")
93
94          plt.legend()
95          errs_train.append(mean_squared_error(y_train, np.polyval(
               best_solution, X_train)))
96          errs_test.append(mean_squared_error(y_control, np.polyval(
               best_solution, X_control)))
97
98          plt.title(f"           {generation}")
99          plt.savefig("img_{}.png".format(generation))
100         plt.clf()
```

```
101
102         images.append(imageio.imread("img_{}.png".format(generation)))
103         os.remove("img_{}.png".format(generation))
104
105     imageio.mimsave(f'training_history_{deg}.gif', images, duration=50)
106
107     plt.plot(gens, errs_train,  label="                                    "
        )
108     plt.plot(gens, errs_test, label="                                  ")
109     plt.xlabel("          ")
110     plt.ylabel("      ")
111     plt.title("            ")
112     plt.legend()
113     plt.savefig(f"errs_{deg}.png")
114     plt.clf()
115     return errs_train[-1], errs_test[-1]
116 def comp():
117     errs_train = []
118     errs_test = []
119     degs = []
120     for i in range(7, 8):
121         e1, e2 = GA(i)
122         errs_train.append(e1)
123         errs_test.append(e2)
124         degs.append(i)
125
126     plt.plot(degs, errs_train,  label="                                    "
        )
127     plt.plot(degs, errs_test, label="                                  ")
128     plt.xlabel("                              ")
129     plt.ylabel("      ")
130     plt.title("            ")
131     plt.legend()
132     plt.savefig(f"errs_degs.png")
133     plt.clf()
134 def best_comp():
135     ga_instance = pygad.GA(num_generations=400, num_parents_mating=10,
136                            fitness_func=fitness_func, sol_per_pop=20,
137                            save_best_solutions=True,
138                            num_genes=8, gene_type=float,
139                            crossover_type = "single_point",
140                            mutation_type = "random",
141                            mutation_percent_genes = 40,
142                            parent_selection_type = "sss",
143                            keep_parents = 1)
144     ga_instance.run()
145     best_solution = ga_instance.best_solutions[-1]
146
147     model, poly_features = fit_polynomial_regression(X_train, y_train, 10)
148
```

```
149     train_error_poly, _ = calculate_error(model, poly_features, X_train,
            y_train)
150     control_error_poly, _ = calculate_error(model, poly_features, X_control,
             y_control)
151     train_error_ga = mean_squared_error(y_train, np.polyval(best_solution,
            X_train))
152     control_error_ga = mean_squared_error(y_control, np.polyval(
            best_solution, X_control))
153     print("\hline                    & {:10.3f} & {:10.3f} \\\\".format(
            train_error_poly, train_error_ga))
154     print("\hline                   & {:10.3f} & {:10.3f} \\\\".format(
            control_error_poly, control_error_ga))
155 comp()
156 best_comp()
```

С помощью разработанного ПО были построены зависимости ошибки аппроксимации от степени полинома:

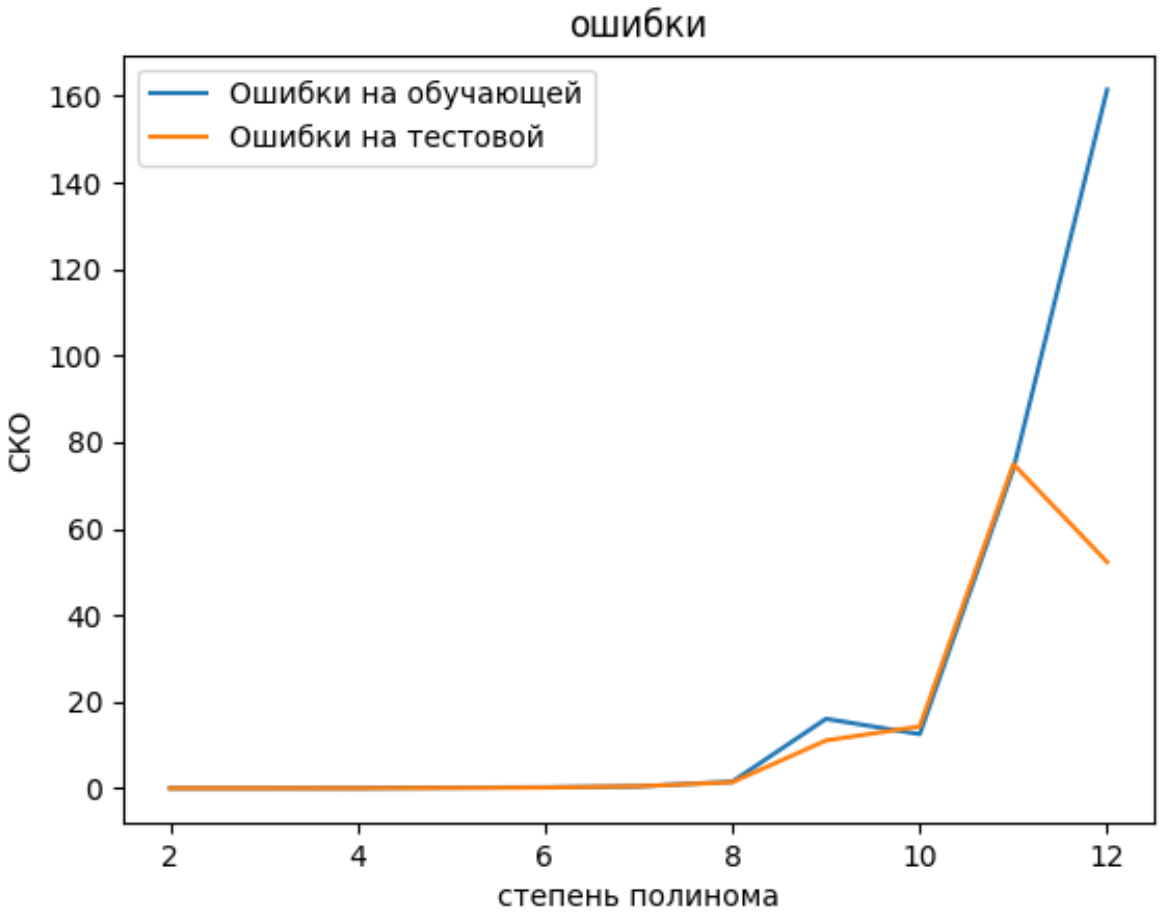

Рис. 2.1: ошибки аппроксимации от степени полинома

Видно, что полиномы степени 7-8 лучше всего аппроксимируют заданную функцию при использовании генетического алгоритма.

При этом ранее при использовании полиномиальной регрессии было получено, что наилучшим образом заданную функцию аппроксимирует полином 10 степени. Абсолютные значения ошибок для полинома 10 степени, полученного с помощью полиномиальной регрессии, и для полинома 7 степени, полученного с помощью генетического алгоритма, приведены в таблице:

Таблица 2.1: Ошибки моделей

Выборка	Полиномиальная регрессия	Генетический алгоритм)
обучающая	0.008	1.369
тестовая	0.008	0.975

Видно, что полиномиальная регрессия точнее аппроксимировала заданную функцию полиномом по сравнению с генетическим алгоритмом.

Также с помощью разработанного ПО были исследованы зависимости ошибки аппроксимации от количества эпох, а также визуализированы полиномы после каждой эпохи. На рисунках 2.2-2.4 приведены ошибки аппроксимации в зависимости от количества эпох для полиномов 2, 7 и 12 степени. На рисунках 2.5-2.7 визуализированы итоговые полиномы (после последней, то есть 400 эпохи) при аппроксимации полиномами степени 2, 7, 12.

Видно, что количество эпох со временем начинает влиять незначительно на ошибку аппроксимации, поэтому увеличение количества эпох не повысит точность аппроксимации.

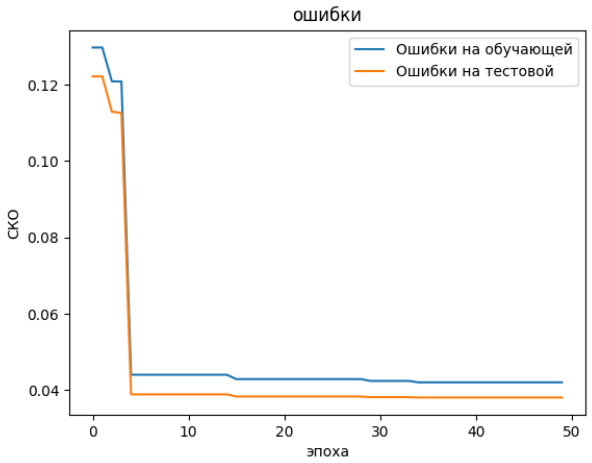

Рис. 2.2: ошибки аппроксимации от количества эпох для полинома 2 степени

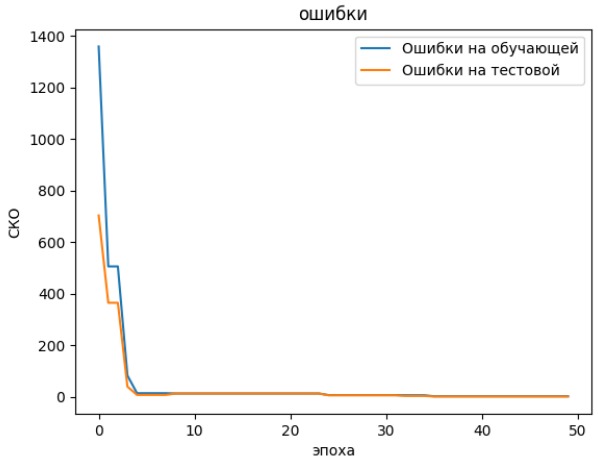

Рис. 2.3: ошибки аппроксимации от количества эпох для полинома 7 степени

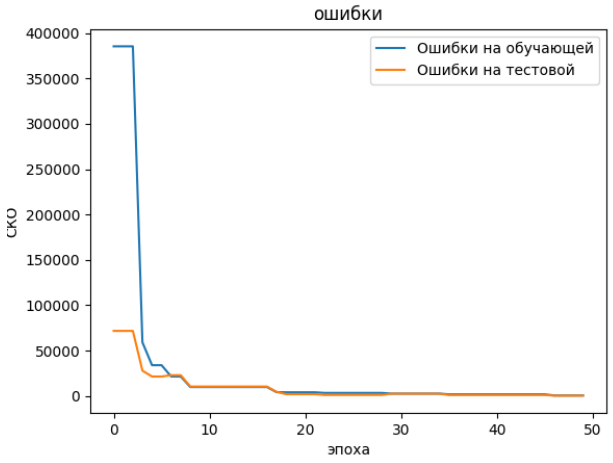

Рис. 2.4: ошибки аппроксимации от количества эпох для полинома 12 степени

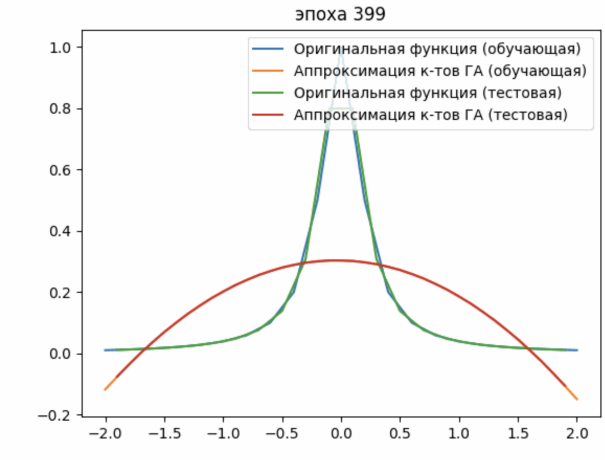

Рис. 2.5: полином после последней эпохи для аппроксимации полиномом 2 степени

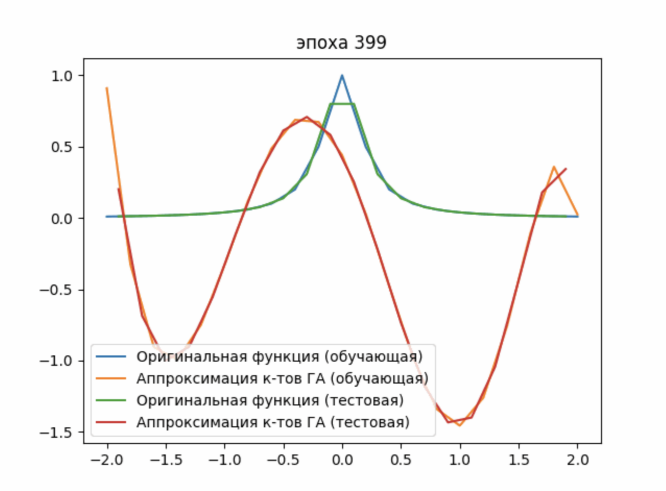

Рис. 2.6: полином после последней эпохи для аппроксимации полиномом 7 степени

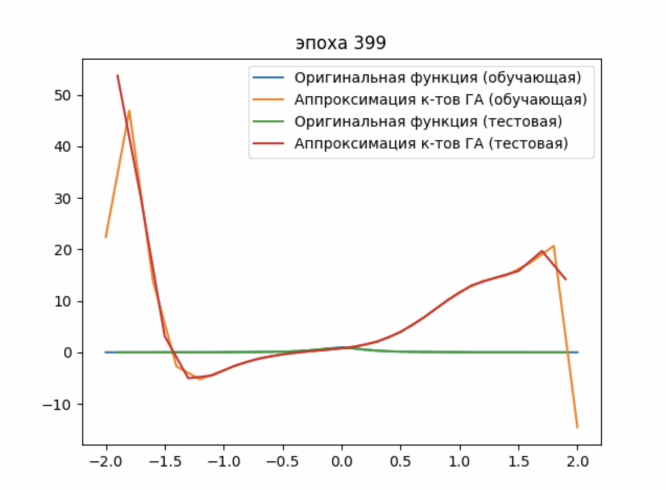

Рис. 2.7: полином после последней эпохи для аппроксимации полиномом 12 степени

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Gad A. F.* PyGAD: An Intuitive Genetic Algorithm Python Library. — 2021. — arXiv: 2106.06158 [cs.NE].
2. Библиотека визуализации данных matplotlib [Электронный ресурс]. — Режим доступа: URL: https://matplotlib.org (дата обращения: 13.12.2023).