DOI 10.58351/2949-2041.2024.9.4.005

Рогаль Сергей Александрович,
Шибанов Вячеслав Сергеевич, Студенты,
Сахалинский государственный университет, Южно-Сахалинск

Научный руководитель **Осипов Геннадий Сергеевич,**
д.т.н., профессор кафедры информатики,
Сахалинский государственный университет, Южно-Сахалинск

СРАВНИТЕЛЬНЫЙ АНАЛИЗ БАЗОВЫХ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ, РЕАЛИЗОВАННЫХ НА ЯЗЫКЕ PYTHON И В СИСТЕМЕ WOLFRAM MATHEMATICA

Аннотация: Представлены результаты аналитического сравнительного исследования базовых методов машинного обучения для построения регрессионных моделей в двух средах: язык Python и система символьной математики Wolfram Mathematica.

Ключевые слова: методы машинного обучения, модели регрессии, нейронные сети, Python, Wolfram Mathematica.

Введение

На данный момент существует множество инструментов для реализации методов машинного обучения и применения их на практике. Поэтому важно с особым вниманием подходить к выбору инструмента разработки.

Целью настоящего исследования являлся сравнительный анализ базовых методов машинного обучения, реализованных в библиотеках языка Python [1] (scikit-learn [2] и PyTorch [3]) и в системе Wolfram Mathematica [4] для решения задачи синтеза регрессионных моделей оценки стоимости квартир на основании экспертной базы данных [5].

В работе было проведено аналитическое сравнение следующих методов машинного обучения [6]:

1. линейная регрессия;
2. случайный лес;
3. нейронная сеть.

Сравнение методов производилось по времени обучения модели и квадратному корню из среднеквадратичной ошибки (RMSE).

Построение регрессионных моделей на языке Python

Первоначальным этапом использования методов машинного обучения является подключение необходимых библиотек и считывание исходных данных (обучающей выборки, data set). На рисунке 1 представлены соответствующие операторы на языке Python.

Подключение библиотек

```
import pandas as pd #библиотека для работы с таблицами
import torch        #библиотека для создания нейронных сетей
import numpy as np  #билблиотека для работы с матрицами
```

Загрузка датасета

```
df = pd.read_excel("./Test.xlsx")
df = df.drop('ID', axis=1)
df.head()
```

Рис. 1 Подключение библиотек и считывание исходных данных

Фрагмент обучающей выборки [5] приведен на рисунке 2.

Район	Тип планировки	Количество комнат	Крайний этаж	Общая площадь	Жилая площадь	Площадь кухни	Агенство	Состояние	Стоимость
Орджоникидзевский	Улучшенная	2	Нет	51.0	30.0	9.0	Нет	удовл	1520
Правобережный	Нестандартная	1	Нет	36.2	17.0	7.0	Нет	хор	1350
Правобережный	хрущевка	2	Нет	44.0	30.0	6.0	Да	удовл	1270
Ленинский	сталинка	1	Да	30.0	19.0	6.0	Нет	удовл	750
Орджоникидзевский	Улучшенная	2	Нет	50.0	30.0	9.0	Нет	удовл	1550

Рис 2 Фрагмент обучающей выборки

Перед построением моделей методами машинного обучения исходные данные приведены к цифровому формату (см. рисунок 3)

```
#импорт инструмента для преобразования категориальных признаков в числовой эквивалент
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
#преобразуем столбцы с категориальными признаками
for column in ['Район', 'Тип планировки','Крайний этаж', 'Агенство', 'Состояние']:
    new_data = encoder.fit_transform(df[column].values)
    df[column] = new_data
xDf = df[df.columns[:-1]]
yDf = df["Стоимость"]
X = xDf.values
y = yDf.values
df.head()
```

Рис. 3 Преобразование категориальных признаков

Результат соответствующих преобразований приведен на рисунке 4.

Район	Тип планировки	Количество комнат	Крайний этаж	Общая площадь	Жилая площадь	Площадь кухни	Агенство	Состояние	Стоимость
1	1	2	1	51.0	30.0	9.0	1	2	1520
2	0	1	1	36.2	17.0	7.0	1	3	1350
2	5	2	1	44.0	30.0	6.0	0	2	1270
0	4	1	0	30.0	19.0	6.0	1	2	750
1	1	2	1	50.0	30.0	9.0	1	2	1550

```
f.to_excel('output.xlsx', index=False) #сохранение обработанного датасета
```

Рис. 4 Исходные данные после преобразования

Ключевым моментом подготовительного этапа для обучения моделей является разбиение обучающей выборки на именно обучающее множество и множество данных, которые будут использованы для тестирования модели. Соответствующие преобразования представлены на рисунке 5.

```
#импорт функции для разделения на обучающую и тестовую выборки
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, train_size=0.8,
                                                    random_state=42, shuffle=True)
#преобразование к тензорам
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1, 1)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32).reshape(-1, 1)
print("Размерность тренировочной выборки:", len(X_train))
print("Размерность валидационной выборки:", len(X_test))
```

```
Размерность тренировочной выборки: 1707
Размерность валидационной выборки: 427
```

Рис. 5 Разделение обучающей выборки на обучающее и тестовое множества

Изначально классическими методами машинного обучения являлись линейная регрессия и метод случайного леса. Рисунок 6 иллюстрирует процесс синтеза моделей обучения данными методами.

```
#импорт моделей машинного обучения: линейная регрессия и случайный лес
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import time #библиотека для измерения времени выполнения частей программы

start = time.time()
lr = LinearRegression().fit(X_train, y_train.ravel()) #инициализация и обучение модели линейной регрессии
time_lr = time.time() - start

start = time.time()
rf = RandomForestRegressor().fit(X_train, y_train.ravel()) #инициализация и обучение модели случайного леса
time_rf = time.time() - start

lr_pred = lr.predict(X_test)
rf_pred = rf.predict(X_test)
```

Рис. 6 Обучение моделей линейной регрессии и случайного леса

На рисунке 7 представлено вычисление основных показателей точности построенных моделей (метрик).

```
from sklearn.metrics import mean_squared_error #импорт средней квадратичной ошибки

dict_py = {} # словарь для сохранения метрик

#сохранение метрик (RMSE, время обучения)
dict_py['rmse_lr'] = round(mean_squared_error(y_test, lr_pred)**0.5, 2)
dict_py['time_lr'] = round(time_lr, 5)
dict_py['rmse_rf'] = round(mean_squared_error(y_test, rf_pred)**0.5, 2)
dict_py['time_rf'] = round(time_rf, 5)
print('-' * 5, "RMSE", '-' * 5)
print("LinearRegression:", dict_py['rmse_lr'])
print("RandomForestRegressor: ", dict_py['rmse_rf'])
print("Время обучения")
print("Линейная регрессия", dict_py['time_lr'])
print("Случайный лес", dict_py['time_rf'])
```

```
----- RMSE -----
LinearRegression: 210.0
RandomForestRegressor:  156.03
Время обучения
Линейная регрессия 0.0
Случайный лес 0.33709
```

Рис. 7 Вычисление метрик

Наряду с методами линейной регрессии и случайного леса исследовался нейросетевой вариант модели. На рисунке 8 представлены операторы инициализации многослойной нейронной сети.

```
## Создаем класс нейронной сети
class Net(torch.nn.Module):
    def __init__(self): #инициализация слоев
        super(Net, self).__init__()
        self.fc1 = torch.nn.Linear(9, 9)
        self.ac1 = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(9, 4)
        self.ac2 = torch.nn.ReLU()
        self.fc3 = torch.nn.Linear(4, 1)
        self.ac3 = torch.nn.ReLU()
        self.fc4 = torch.nn.Linear(1, 1)

    def forward(self, x): #функция прогноза
        x = self.fc1(x)
        x = self.ac1(x)
        x = self.fc2(x)
        x = self.ac2(x)
        x = self.fc3(x)
        x = self.ac3(x)
        x = self.fc4(x)
        return x
```

```
model = Net() #инициализация модели
lossFunction = torch.nn.MSELoss()  # задаем функцию потерь mean square error
optimizer = torch.optim.Adam(model.parameters(), lr = 1) # задаем оптимизатор в виде Adam
```

Рис. 8 Синтез и инициализация нейронной сети

Задание параметров и обучение нейронной сети иллюстрирует рисунок 9.

```
start = time.time()
epochs = 200   # кол-во эпох
batchSize = 512 # размер батча

# Переменные лучшей модели
best_mse = np.inf   # бесконечность
best_weights = None
history = []
size = X.shape[0]

# Процесс тренировки
for epoch in range(epochs):
    model.train() # Переключаем в режим тренировки

    index = 0

    while index * batchSize <= size:
        # Берем батч
        X_batch = X_train[index:index + batchSize]
        y_batch = y_train[index:index + batchSize]

        # обнуление градиента
        optimizer.zero_grad()
        # делаем прогноз
        y_pred = model.forward(X_batch)
        # рассчет функции потерь
        loss = lossFunction(y_pred, y_batch)
        loss.backward()
        # обновление весов
        optimizer.step()
        index += batchSize
```

Рис. 9 Обучение нейронной сети

На рисунке 10 представлены операторы завершения обучения сети и вывода ошибки модели.

```
    # Переключение режима модели
    model.eval()

    y_pred = model(X_test) # прогноз на тестовой выборке
    mse = lossFunction(y_pred, y_test) # получение метрики MSE
    mse = float(mse)
    history.append(mse)
    # нахождение лучшей модели с лучшим показателем MSE
    if mse < best_mse:
        best_mse = mse
        # копирование весов
        best_weights = copy.deepcopy(model.state_dict())

print(f"best mse is {best_mse} -- {round(best_mse**0.5, 2)}")
print('Затраченное время: ', time.time()-start)
# Сохранение лучших весов нейросети и вывод лучшей MSE
model.load_state_dict(best_weights)
dict_py['rmse'] = round(best_mse**0.5, 2)
dict_py['time'] = time.time()-start
```

```
best mse is 55565.31640625 -- 235.72
Затраченное время:  0.3139760494232178
```

Рис. 10 Завершение обучения

Для сопоставления результатов построения моделей методами машинного обучения с Wolfram Mathematica использовались соответствующие метрики (см. рисунок 11).

```
import json #библиотека для работы с файлами типа json

file = open('./metrics.json', 'r') # получаем данные с json-файла от Wolfram
dict_wolf = json.loads(file.read())

print("Python", dict_py)
print("Wolfram", dict_wolf)
```

```
Python {'rmse_lr': 210.0, 'time_lr': 0.0, 'rmse_rf': 156.03, 'time_rf': 0.33709, 'rmse': 235.72, 'time': 0.3139760494232178}
Wolfram {'rmse': 217.78749090610378, 'time': 1.234375, 'rmse_lr': 194.52476766154138, 'time_lr': 1.609375, 'rmse_rf': 179.16509510309174, 'time_rf': 0.3437
5}
```

Рис. 11 Загрузка метрик Wolfram

Построение регрессионных моделей в системе Wolfram Mathematica

Классический ввод исходных данных из файла представлен операторами на рисунке 12.

```
path = StringJoin[NotebookDirectory[], "output.xlsx"];
       соединить с… директория файла блокнота
data = Import[path, "Data"]
       импорт

x = data[[1, 2 ;; -1, 1 ;; -2]];


y = data[[1, 2 ;; -1, -1]];
```

Рис. 12 Ввод исходных данных

Этап разбиения обучающей выборки на обучающее и тестовое множества представлен на рисунке 13.

```
Lx = Length[x];
     длина
trainIds = RandomSample[Range[Lx], IntegerPart[0.80 Lx]];
           случайная вы… диапазон   целая часть
trainingSet = Thread[x[[trainIds]] → y[[trainIds]]];
              нанизать
valIds = Complement[Range[Lx], trainIds];
         дополнение  диапазон
validationSet = Thread[x[[valIds]] → y[[valIds]]];
                нанизать
```

Рис. 13 Разделение на обучающее и тестовое множества

На рисунке 14 приведен фрагмент текста, обеспечивающего процессы построения моделей линейной регрессии и случайного леса, а также подсчета их ошибок.

```
TimeLR = First[Timing[LR = Predict[trainingSet, Method → "LinearRegression"]]]
         первый затраченное… предсказать              метод
TimeRF = First[Timing[RF = Predict[trainingSet, Method → "RandomForest"]]]
         первый затраченное… предсказать              метод
ξlr = LR[x[[valIds]]];
ξrf = RF[x[[valIds]]];
m = Length[x[[valIds]]];
    длина
RMSELR = √(∑_{i=1}^{m} (y[[valIds]][[i]] - ξlr[[i]])^2 / m)
RMSERF = √(∑_{i=1}^{m} (y[[valIds]][[i]] - ξrf[[i]])^2 / m)
```

Рис. 14 Пример построения моделей и оценки погрешности

Инициализация многослойной нейронной сети представлена соответствующими операторами на рисунке 15.

```
net = NetInitialize @ NetChain[{
      инициализироват… нейронная сеть
    LinearLayer["Input" → Dx, "Output" → Dx,
    линейный слой
     "Weights" → DiagonalMatrix[1 / ArrayReduce[Max, x, 1]],
                 диагональная матрица сокращение … максимум
     LearningRateMultipliers → 0.001],
     множители оценок обучения
    ParametricRampLayer[],
    параметрический слой уклона
    LinearLayer["Input" → Dx, "Output" → 4],
    линейный слой
    ParametricRampLayer[],
    параметрический слой уклона
    LinearLayer["Input" → 4, "Output" → 1],
    линейный слой
    ParametricRampLayer[],
    параметрический слой уклона
    LinearLayer["Input" → 1, "Output" → "Real", "Weights" → Max[y]]}]
    линейный слой                                  максимум
```

Рис 15 Инициализация нейронной сети

Топология построенной сети представлена на рисунке 16.

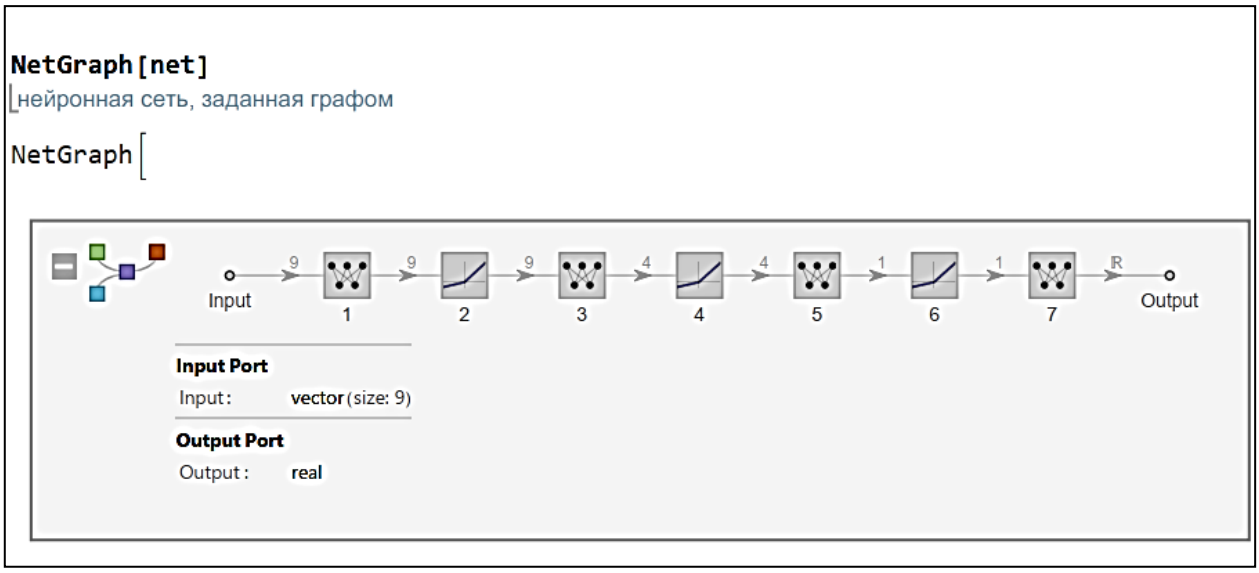

Рис. 16 Структура нейросети

Для сравнения результатов эксперимента в двух средах осуществлялся экспорт данных (см. рисунок 17).

```
Export[StringJoin[NotebookDirectory[], "metrics.json"],
 {"rmse" → RMSE, "time" → First[time], "rmse_lr" → RMSELR,
  "time_lr" → TimeLR, "rmse_rf" → RMSERF, "time_rf" → TimeRF}]
```

Рис. 17 Экспорт данных в json-файл

Визуализация результатов моделирования и их сравнения представлены в виде диаграммы, приведенной на рисунке 18.

```
import matplotlib.pyplot as plt
#визуализация метрик
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
categories = ['pyNN', 'wolNN', 'pyLR', 'wolLR', 'pyRF', 'wolRF']

ax.bar(categories, [dict_py['rmse'], dict_wolf['rmse'], dict_py['rmse_lr'], dict_wolf['rmse_lr'], dict_py['rmse_rf'], dict_wolf['rmse_rf']], color=['red',
ax.set_title("RMSE моделей")
```

```
Text(0.5, 1.0, 'RMSE моделей')
```

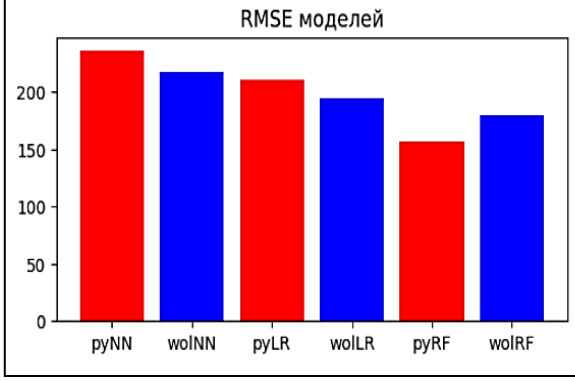

Рис. 18 Визуализация результатов

Выводы и заключение

Основные результаты сравнения выбранных методов обучения представлены в таблице

Таблица

Основные результаты исследования

№	Метод обучения	RMSE	
		Wolfram	**Python**
1.	Линейная регрессия	195	210
2.	Случайный лес	179	156
3.	Нейронная сеть	218	236

Из таблицы видно, что результаты тестирования по ошибке RMSE практически идентичны. Отметим, что продолжительность обучения моделей на языке Python (при выбранных параметрах настройки) меньше, чем в среде Wolfram Mathematica.

Из полученных результатов можно сделать вывод, что оба инструмента показывают практически идентичный результат при создании моделей машинного обучения. Поэтому использование обоих инструментов является целесообразным для решения задачи регрессии. Кроме того, следует учитывать преимущества и особенности каждой системы и применять их для решения задач.

Список литературы:

1. Python documentation — URL.: https://www.python.org/doc (Дата обращения 11.04.2024).
2.
3. Scikit-learn documentation — URL.: https://scikit-learn.org/stable/user_guide.html2. (Дата обращения 11.04.2024).
4.
5. PyTorch documentation — URL.: https://pytorch.org/docs (Дата обращения 11.04.2024).
6.
7. Wolfram Mathematica, URL: https://www.wolfram.com/mathematica/ (Дата обращения 11.04.2024).
8. Loginom — аналитическая low-code платформа, URL: https://loginom.ru/platform, (Дата обращения 11.04.2024).
9. Витковская, П. В., Рогаль С.А., Шибанов В.С. Введение в сравнительный анализ методов машинного обучения для решения задачи прогнозирования // МОЛОДЫЕ ИССЛЕДОВАТЕЛИ в ОТВЕТ на СОВРЕМЕННЫЕ ВЫЗОВЫ: сборник статей II Международного научно-исследовательского конкурса, Петрозаводск, 09 ноября 2022 года. – Петрозаводск: Международный центр научного партнерства «Новая Наука», 2022. – С. 164-170. – EDN ENCXXD.