

Санкт-Петербургский государственный университет
Прикладная математика, информатика и искусственный интеллект

Отчёт по учебной практике 1 (научно-исследовательской работе)
Методы машинного обучения на примере регрессии

Выполнил:

Гуноев Адам Асланбекович, 
группа 22.Б06-мм

Научный руководитель:

Кандидат физ.-мат. наук, доцент
Шпилёв Пётр Валерьевич
Кафедра статического моделирования

Работа выполнена
в полном объёме


Отметка о зачете:

<< Работа выполнена на хорошем уровне и может быть зачтена с оценкой В >>

Санкт-Петербург

2022

Отзыв на учебную практику 1 (научно – исследовательскую работу)
студента 1-го курса бакалавриата Гуноева Адама Асланбековича

Работа посвящена изучению методов машинного обучения на примере задачи регрессии. В рамках данной работы студент самостоятельно ознакомился с классическим подходом к построению коэффициентов регрессии (основанным на решении нормального уравнения), а также, с альтернативными подходами, основанными на использовании градиентного спуска и популярными алгоритмами машинного обучения (метод опорных векторов, дерево решений, случайный лес, регрессия с голосованием). В работе на языке Python реализованы данные подходы к построению регрессий для реального набора данных.

Работа написана достаточно аккуратно, поставленная задача реализована практически в полном объеме. Считаю, что работа может быть зачтена с оценкой В.

29.12.22



Петр Валерьевич Шпилев

Оглавление

Оглавление	2
Введение	3
Основная часть	4
Выбор данных	4
Импортирование модулей	4
Анализ набора данных	4
Обучение и проверка моделей	6
Линейная регрессия	7
Полиномиальная регрессия	8
Полиномиальная регрессия со стохастическим градиентным спуском	9
Метод опорных векторов	10
Дерево решений	11
Ансамблевое обучение	14
Адаптивный бустинг	14
Случайный лес	15
Регрессия с голосованием	16
Понижение размерности	17
Заключение	19
Список источников	20
Приложение	21
Код программы	21

Введение

Была поставлена задача исследования работы различных методов машинного обучения в задаче регрессии, сравнения их результатов, процессов обучения и совместимости с данными.

Основная часть

Вся работа выполнена с использованием языка программирования Python и Scikit-learn — бесплатной библиотеки машинного обучения.

Выбор данных

Для поиска материала был выбран сайт «Kaggle.com» — социальная сеть для специалистов по обработке данных и машинному обучению. Для проведения работы взят набор данных зависимости уровня счастья населения страны от некоторых признаков, таких как «ВВП на душу населения», «уровень восприятия коррупции» и др.

Импортирование модулей

Первым делом импортируем все необходимые модули:

```
import os
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
```

Анализ набора данных

Рассмотрим, что представляют из себя данные:

	Country	happiness_score	gdp_per_capita	family	health	freedom	generosity	government_trust	dystopia_residual	continent	Year	social_supp
0	Norway	7.5370	1.616463	1.533524	0.796667	0.635423	0.362012	0.315964	2.277027	Europe	2015	0.000
1	Denmark	7.5220	1.482383	1.551122	0.792566	0.626007	0.355280	0.400770	2.313707	Europe	2015	0.000
2	Iceland	7.5040	1.480633	1.610574	0.833552	0.627163	0.475540	0.153527	2.322715	Europe	2015	0.000
3	Switzerland	7.4940	1.564980	1.516912	0.858131	0.620071	0.290549	0.367007	2.276716	Europe	2015	0.000
4	Finland	7.4690	1.443572	1.540247	0.809158	0.617951	0.245483	0.382612	2.430182	Europe	2015	0.000
...
787	Botswana	3.4789	0.997549	0.000000	0.494102	0.509089	0.033407	0.101786	0.257241	Africa	2020	1.085
788	Tanzania	3.4762	0.457163	0.000000	0.442678	0.509343	0.271541	0.203881	0.718963	Africa	2020	0.872
789	Rwanda	3.3123	0.343243	0.000000	0.572383	0.604088	0.235705	0.485542	0.548445	Africa	2020	0.522
790	Zimbabwe	3.2992	0.425564	0.000000	0.375038	0.377405	0.151349	0.080929	0.841031	Africa	2020	1.047
791	Afghanistan	2.5669	0.300706	0.000000	0.266052	0.000000	0.135235	0.001226	1.507236	Asia	2020	0.356

Планируется обучить модели так, чтобы они умели прогнозировать значение показателя счастья («*happiness_score*»). Соответственно, такие атрибуты как «Year», «Country», «continent» не имеют смысла, поскольку не являются определяющими признаками для уровня счастья населения. Поэтому от них стоит избавиться.

Рассмотрим корреляции значений относительно показателя счастья:

```
# Корреляция значений относительно показателя счастья

data.corr()["happiness_score"].sort_values(ascending=False)

happiness_score      1.000000
gdp_per_capita       0.793267
health                0.753534
cpi_score              0.693001
freedom                 0.544284
government_trust      0.455477
social_support         0.192633
dystopia_residual     0.174161
generosity              0.155419
family                  0.154946
```

Как можно увидеть, наибольшее влияние на результат будут оказывать значения атрибутов «*gdp_per_capita*», «*health*», «*cpi_score*», «*freedom*» и «*government_trust*».

Далее приведём все значения в диапазон [0, 1] с помощью функции *MinMaxScaler()*. Сделать это нужно потому, что алгоритмы МО лучше всего выполняются на данных с одинаковым масштабом.

```
# Нормализация значений (от 0 до 1)

MMS = MinMaxScaler(copy=True).fit(data)
data = MMS.transform(data)
data = pd.DataFrame(data, columns = ['happiness_score', 'gdp_per_capita',
                                      'family', 'health', 'freedom', 'generosity',
                                      'government_trust', 'dystopia_residual',
                                      'social_support', 'cpi_score'])

X = data.drop('happiness_score', axis=1)
y = data['happiness_score'].copy()
```

Теперь разделим набор данных на обучающий и тестовый используя `train_test_split()`:

```
# Деление на тренировочный и тестовый наборы
train_data, test_data, train_labels, test_labels = train_test_split(X, y, test_size=0.2)
```

Обучение и проверка моделей

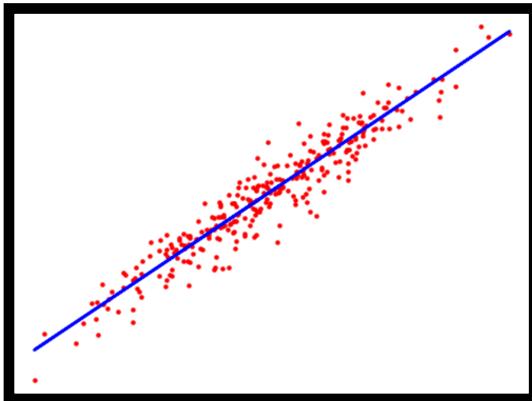
Далее будем обучать регрессионные модели, проверять их обучаемость, точность прогнозов, а так же рассмотрим **кривые обучения** некоторых из них. (Кривая обучения — график изменения величины ошибки в процессе обучения модели). Точность будем измерять ручной проверкой на обучающих и тестовых данных, а также методом **кросс-валидации** (или **перекрестной проверки**). Он заключается в произвольном разбиении обучающего набора на некоторое количество поднаборов, на которых происходит обучение, кроме одного, на котором оценивается модель. Данный алгоритм происходит несколько раз.

Для кривых обучения зададим функцию:

```
# Кривые обучения
def plot_learning_curves(model):
    X_train = train_data.values
    X_val = test_data.values
    y_train = train_labels.values
    y_val = test_labels.values
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
        val_errors.append(mean_squared_error(y_val_predict, y_val))
    plt.plot(np.sqrt(train_errors), "r-+", linewidth=1, label="train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
```

Линейная регрессия

Начнём с самой простой модели - линейной регрессии. Простыми словами, она демонстрирует зависимость в виде графика прямой:



Код выглядит следующим образом:

```
# Линейная регрессия

# Обучение без кросс-валидации
print("Без кросс-валидации\n")
LR = LinearRegression()
LR.fit(train_data, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
LR_predictions = LR.predict(train_data)
print("Прогнозы: ", LR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
LR_rmse = np.sqrt(mean_squared_error(train_labels, LR_predictions))
print("Корень средней квадратичной ошибки: ", LR_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
LR_predictions = LR.predict(test_data)
print("Прогнозы: ", LR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
LR_rmse = np.sqrt(mean_squared_error(test_labels, LR_predictions))
print("Корень средней квадратичной ошибки: ", LR_rmse, '\n')

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print("Кросс-валидация:")
score = cross_val_score(LR, train_data, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), '\n\n')

print("\tКривые обучения:")
plot_learning_curves(LR)
```

Благодаря использованию библиотеки Scikit-learn, код для разных моделей будет практически идентичен, поэтому в дальнейшем вместо целого кода будут указаны лишь отличия.

Рассмотрим вывод линейной регрессии:

```
Без кросс-валидации

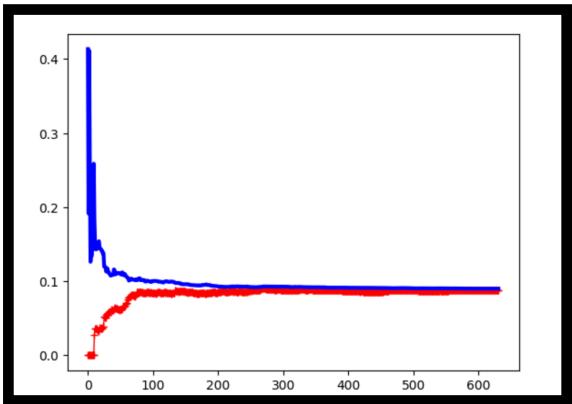
Проверка на тренировочном наборе:

Прогнозы: [0.181282  0.74621477 0.56529315 0.37885393 0.53511023]
Ожидания: [0.3050669547951684, 0.6659925770499675, 0.514155449084116, 0.5011827902470916, 0.33215690216128374]
Корень средней квадратичной ошибки: 0.08669319734192645

Проверка на тестовом наборе:

Прогнозы: [0.52806625 0.34632495 0.54673495 0.98657868 0.37406323]
Ожидания: [0.4838223417780984, 0.13970392483893418, 0.5402914928420763, 0.931359483918661, 0.14220304896138475]
Корень средней квадратичной ошибки: 0.09454038107301722

Кросс-валидация:
Средняя оценка: 0.0883987541725284
```



(Замечание: Красной линией представлено изменение ошибки на обучающих данных, синей - на тестовых)

Как можно наблюдать, ошибка модели довольно велика: 0.088. Кривые обучения сходятся в одной точке. Очевиден вывод, что линейная регрессия — слишком простая модель.

Полиномиальная регрессия

Рассмотрим следующий метод — полиномиальная регрессия. Отличие от линейной регрессии состоит в том, что обучение линейной модели происходит на полиномиальных данных. Они преобразуются следующим образом:

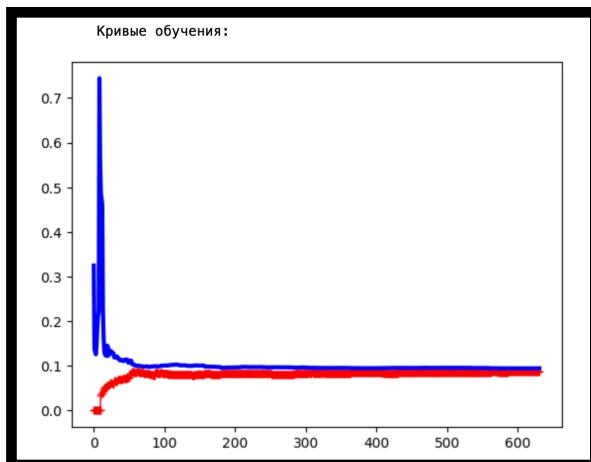
```
# Создание полиномиальных признаков
poly_features = PolynomialFeatures(degree=2, include_bias=False)
train_data_poly = poly_features.fit_transform(train_data)
test_data_poly = poly_features.fit_transform(test_data)
```

Рассмотрим вывод:

```
Проверка на тренировочном наборе:
Прогнозы: [0.23742641 0.67157829 0.49006137 0.53257988 0.41716435]
Ожидания: [0.3050669547951684, 0.6659925770499675, 0.514155449084116, 0.5011827902470916, 0.33215690216128374]
Корень средней квадратичной ошибки: 0.054649250849835386

Проверка на тестовом наборе:
Прогнозы: [0.59190546 0.28773506 0.49423509 0.95000834 0.16076483]
Ожидания: [0.4838223417780984, 0.13970392483893418, 0.5402914928420763, 0.931359483918661, 0.14220304896138475]
Корень средней квадратичной ошибки: 0.07069841149140882

Кросс-валидация:
Средняя оценка: 0.05977533533354824
```



(Замечание: При установке параметра $\text{degree} \geq 3$, модель переобучается)

Заметно значительное улучшение результата: средняя ошибка 0.059 против 0.088 у линейной регрессии. Кривые обучения строят тот же рисунок.

Полиномиальная регрессия со стохастическим градиентным спуском

Теперь применим на полиномиальных данных метод обучения со **стохастическим градиентным спуском**. Сначала про сам метод:

Градиентный спуск - численный метод нахождения локального минимума или максимума функции ошибки с помощью движения вдоль градиента (направления движения графика). Обычно считается как сумма градиентов, вызванных каждым элементом обучения. При стохастическом градиентном спуске значение градиента аппроксимируются градиентом функции, вычисленном только на одном элементе обучения.

Используем функцию SGDRegressor() со следующими параметрами:

```
SGDR = SGDRegressor(max_iter=99999999, penalty='l2',
                     eta0=0.025, power_t=0, tol=0.03,
                     loss="squared_error")
```

(Данные параметры были подобраны вручную как наилучшие, после множественных сравнений комбинаций)

Результат:

```
Без кросс-валидации

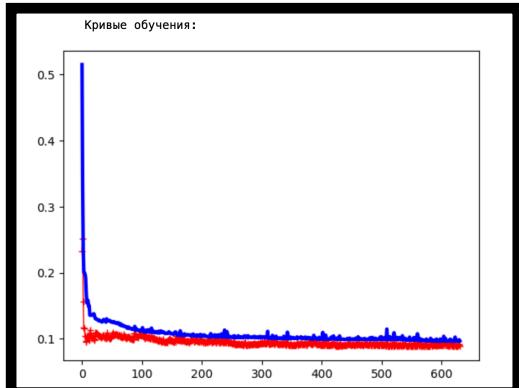
Проверка на тренировочном наборе:

Прогнозы: [0.21270442 0.6821295 0.49885271 0.45504345 0.47757319]
Ожидания: [0.3050669547951684, 0.6659925770499675, 0.514155449084116, 0.5011827902470916, 0.33215690216128374]
Корень средней квадратичной ошибки: 0.06374707149706836

Проверка на тестовом наборе:

Прогнозы: [0.55819098 0.31508581 0.51901279 0.93620519 0.27013415]
Ожидания: [0.4838223417780984, 0.13970392483893418, 0.5402914928420763, 0.931359483918661, 0.14220304896138475]
Корень средней квадратичной ошибки: 0.07721350364134708

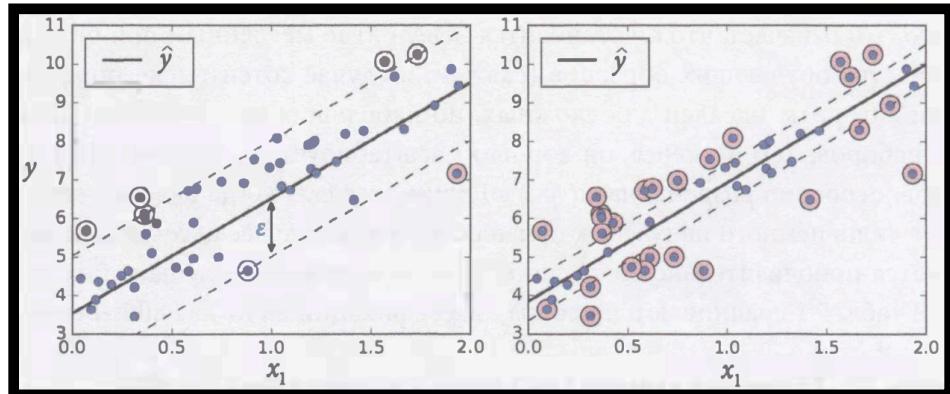
Кросс-валидация:
Средняя оценка: 0.06759403964966618
```



Модель работает хуже обычной полиномиальной регрессии. На кривых обучения виден характерный для градиентного спуска рисунок. Также видно, что ошибка медленно, но ровно приближается к 0. Учитывая эффективность градиентного спуска, можно предположить, что модели не хватает данных (весь набор состоит из 792 примера) для более эффективного обучения.

Метод опорных векторов

Идея данного метода заключается в построении линии или гиперплоскости, умещающей как можно большее количество образцов на себе в пределах зазора (« ϵ » на рисунке ниже).



Посмотрим, как он будет работать с нашими данными. Используем SVR() со следующими параметрами: [kernel="poly", degree=2, C=100, epsilon=0.1].

```
Проверка на тренировочном наборе:  
Прогнозы: [0.26776893 0.64607431 0.48851905 0.47248088 0.43254892]  
Ожидания: [0.3050669547951684, 0.6659925770499675, 0.514155449084116, 0.5011827902470916, 0.33215690216128374]  
Корень средней квадратичной ошибки: 0.06554802278768182  
  
Проверка на тестовом наборе:  
Прогнозы: [0.62994436 0.29252902 0.49118055 0.86609798 0.07377803]  
Ожидания: [0.4838223417780984, 0.13970392483893418, 0.5402914928420763, 0.931359483918661, 0.14220304896138475]  
Корень средней квадратичной ошибки: 0.08540167549198921  
  
Кросс-валидация:  
Средняя оценка: 0.07009344788771184
```

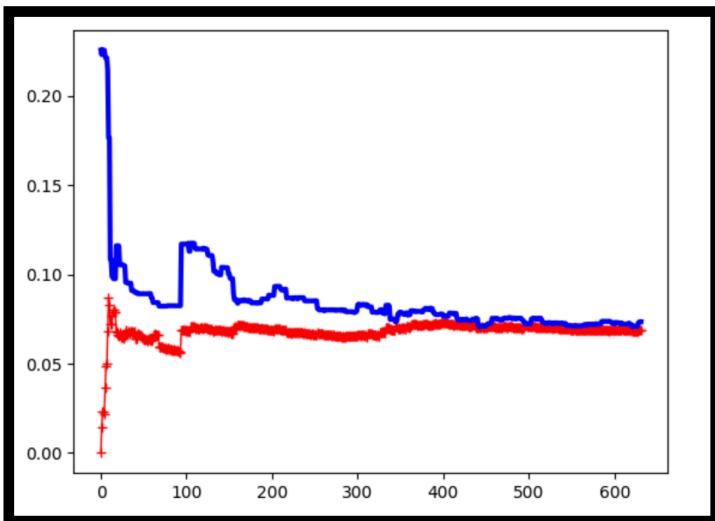
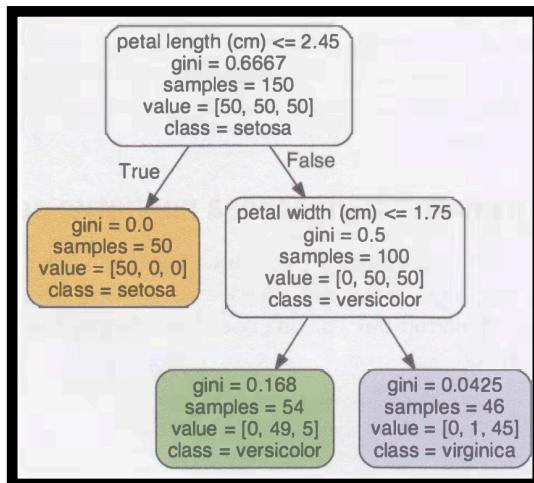


График модели нестабильный, но кривые сходятся на одном уровне. Ошибка так же относительно велика.

Дерево решений

Теперь исследуем дерево решений. Оно представляет собой иерархическую древовидную структуру, состоящую из правила вида «Если ..., то ...». За счет обучающего множества правила генерируются автоматически в процессе обучения.



Пример
дерева
решений

Эта модель очень мощная, поэтому можно сразу предположить, что она будет легко переобучаться. Используем DecisionTreeRegressor().

Рассмотрим вывод:

```
Без кросс-валидации

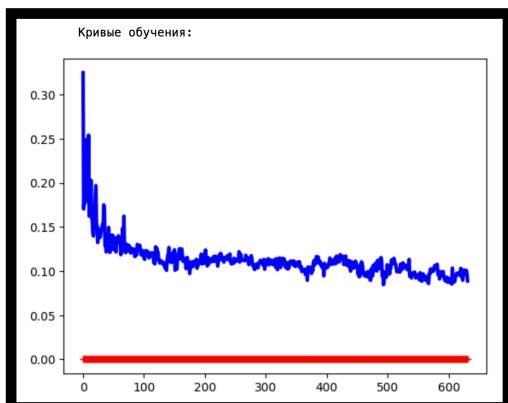
Проверка на тренировочном наборе:

Прогнозы: [0.30506695 0.66599258 0.51415545 0.50118279 0.3321569]
Ожидания: [0.3050669547951684, 0.6659925770499675, 0.514155449084116, 0.5011827902470916, 0.33215690216128374]
Корень средней квадратичной ошибки: 5.08288248238951e-10

Проверка на тестовом наборе:

Прогнозы: [0.7381243 0.37527185 0.50080124 0.90428859 0.14634285]
Ожидания: [0.4838223417780984, 0.13970392483893418, 0.5402914928420763, 0.931359483918661, 0.14220304896138475]
Корень средней квадратичной ошибки: 0.09371895028243522

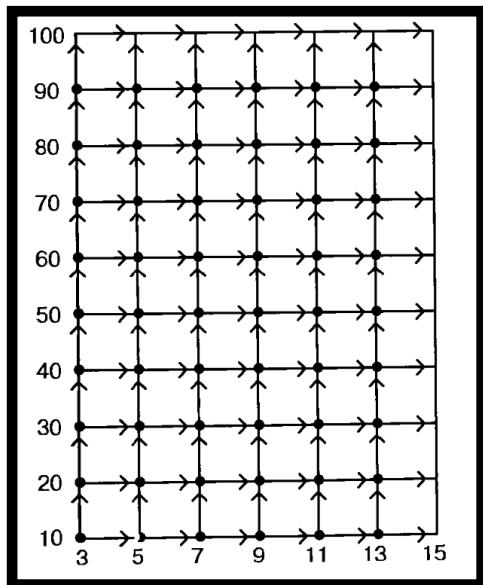
Кросс-валидация:
Средняя оценка: 0.09399752320310606
```



Что и ожидалось: модель настолько мощная, что ошибка на обучающем наборе с самого начала неизменно равна 0.

Решить данную проблему поможет метод **регуляризации**. Грубо говоря - упрощение модели путём изменения её гиперпараметров. Но ручной перебор гиперпараметров займет слишком много времени (как это было с градиентным спуском), поэтому воспользуемся методом **решётчатого поиска** с помощью функции GridSearchCV().

Принцип работы решётчатого поиска: Диапазоны переменных задают решетку комбинаций их значений, затем оценивается эффективность каждой такой комбинации.



Пример сетки
решётчатого поиска

Код:

```
# Гиперпараметры
hyperparams = {'max_depth': [22, 23, 24, 25], 'min_samples_leaf': [2, 3, 4, 5],
               'min_samples_split': [2, 4, 6, 8], 'max_features': [6, 7, 8, 9],
               'max_leaf_nodes': [25, 26, 27, 28]}

# Обучение модели
DTR = DecisionTreeRegressor()
GS_DTR = GridSearchCV(DTR, hyperparams, cv=10, scoring='neg_mean_squared_error')
GS_DTR.fit(train_data, train_labels)
print("Лучший результат: ", np.sqrt(-GS_DTR.best_score_))
```

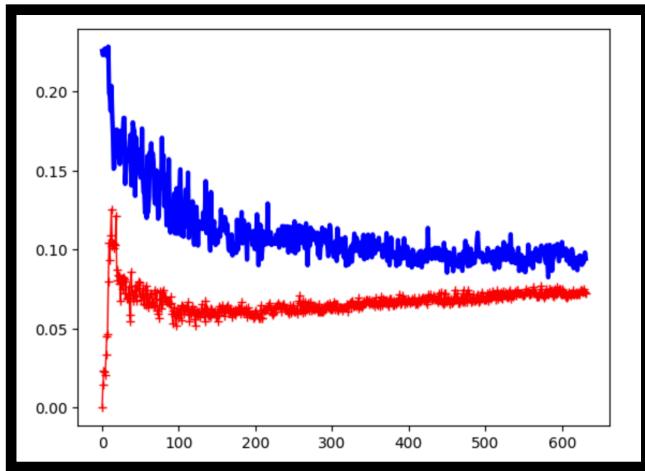
Затем обучаем модель по тому же коду, но на этот раз не нужен блок с кросс-валидацией, т.к. она встроена в GridSearchCV(). Рассмотрим результат:

```
Лучший результат: 0.09310817680646348
Проверка на тренировочном наборе:

Прогнозы: [0.41698013 0.51141994 0.39026731 0.49871439 0.38373029]
Ожидания: [0.527509624189013, 0.5563165221980702, 0.49946582281609225, 0.5038536284730906, 0.5153000780130861]
Корень средней квадратичной ошибки: 0.07270665315922994

Проверка на тестовом наборе:

Прогнозы: [0.63998689 0.45059076 0.29251089 0.70408936 0.4065747]
Ожидания: [0.6965355290630153, 0.3334923153602843, 0.20910755281820587, 0.5628028436040677, 0.4352512630265198]
Корень средней квадратичной ошибки: 0.09021390579803824
```



(Замечание: Решётчатый поиск занимает некоторое время (зависит от количества предложенных значений), соответственно общее время для обучения модели увеличивается)

Очевидно, модель работает лучше. Ошибка велика, зато решена проблема переобучения. По графику видно, что линии не сошлись в одной точке, но двигались друг к другу. Модель недообучена. Возможно ей не хватило имеющихся данных.

С помощью строки «GS_DTR.best_estimator_» можем вывести лучшее сочетание гиперпараметров:

```
▼ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=25, max_features=7, max_leaf_nodes=27,
min_samples_leaf=5, min_samples_split=8)
```

Ансамблевое обучение

Адаптивный бустинг

Продолжая работу с деревом решений, применим один из методов ансамблевого обучения (т.е. с обучением нескольких моделей) — **адаптивный бустинг**. Его суть заключается в последовательном, опирающимся на ошибки прошлых итераций, обучении одинаковых моделей и их объединении в «ансамбль».

Для этого используем AdaBoostRegressor() со следующими параметрами:

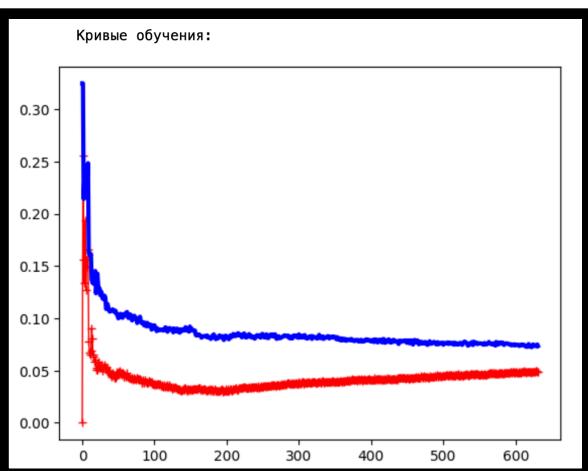
```
ABR = AdaBoostRegressor(GS_DTR.best_estimator_, n_estimators=100, learning_rate=0.1, loss='square')
ABR.fit(train_data, train_labels)
```

Результат:

```
Проверка на тренировочном наборе:
Прогнозы: [0.25310236 0.65879809 0.44405547 0.48894113 0.38847342]
Ожидания: [0.3050669547951684, 0.6659925770499675, 0.514155449084116, 0.5011827902470916, 0.33215690216128374]
Корень средней квадратичной ошибки: 0.04895174814042977

Проверка на тестовом наборе:
Прогнозы: [0.63082894 0.27102154 0.4812122 0.88540785 0.21092627]
Ожидания: [0.4838223417780984, 0.13970392483893418, 0.5402914928420763, 0.931359483918661, 0.14220304896138475]
Корень средней квадратичной ошибки: 0.07271461667399857

Кросс-валидация:
Средняя оценка: 0.07124060869389451
```



(**Замечание:** так как при ансамблевом обучении применяются несколько моделей, времени на обучение требует больше)

Результат улучшился, но ошибка всё равно велика. Кривые обучения стали более ровными. Модель так же недообучена.

Случайный лес

Данный метод представляет из себя ансамбль деревьев решений. Проверим, как он работает на наших данных.

Будем использовать RandomForestRegressor() и решётчатый поиск для подбора гиперпараметров:

```
# Гиперпараметры
hyperparams = [
    {'n_estimators': [2, 4, 10, 20, 50, 100], 'max_features': [2, 4, 6, 8, 9, 12, 16]}
]

# Обучение и оценка
print("Решётчатый поиск и кросс-валидация: \n")
RDR = RandomForestRegressor()
GS_RDR = GridSearchCV(RDR, hyperparams, cv=10, scoring='neg_mean_squared_error')
GS_RDR.fit(train_data, train_labels)
print("Лучший результат: ", np.sqrt(-GS_RDR.best_score_))
```

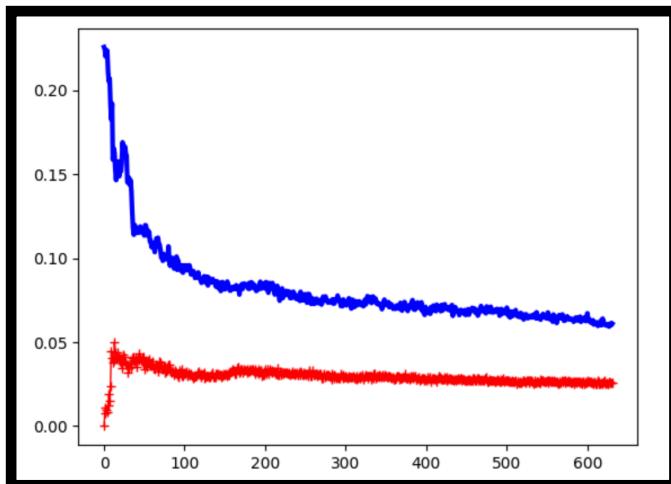
Результаты:

```
Лучший результат: 0.0668542832962067
Проверка на тренировочном наборе:

Прогнозы: [0.53602159 0.56498988 0.48359437 0.50263229 0.50769563]
Ожидания: [0.5275096241890813, 0.5563165221980702, 0.49946582281609225, 0.5038536284730906, 0.5153000780130861]
Корень средней квадратичной ошибки: 0.025821080662785205

Проверка на тестовом наборе:

Прогнозы: [0.67855449 0.37699797 0.31931817 0.62041359 0.38255923]
Ожидания: [0.6965355290630153, 0.3334923153602843, 0.20910755281820587, 0.5628028436040677, 0.4352512630265198]
Корень средней квадратичной ошибки: 0.061877693567205246
```



Наблюдаем новый рисунок. Кривые двигаются параллельно. Модель явно недообучена. Возможно, если бы данных было больше, модель смогла бы оптимально обучиться.

Для сравнения, проверим **рандомизированный поиск** вместо решётчатого. Отличие в том, что вместо перебора всех значений гиперпараметров происходит случайный перебор некоторого установленного количества. Используем RandomizedSearchCV():

```
# Гиперпараметры
hyperparams = [
    {'n_estimators': [i for i in range(1, 200)]}, 'max_features': [i for i in range(1, 50)]}

# Обучение и оценка
print("Рандомизированный поиск и кросс-валидация: \n")
RS_RDR = RandomizedSearchCV(RDR, hyperparams, n_iter=20, cv=10, scoring='neg_mean_squared_error')
RS_RDR.fit(train_data, train_labels)
```

Результат:

```
Лучший результат: 0.06687114135646972
Проверка на тренировочном наборе:

Прогнозы: [0.52295518 0.56433838 0.47722895 0.5049975 0.54416771]
Ожидания: [0.5275096241890813, 0.5563165221980702, 0.49946582281609225, 0.5038536284730906, 0.5153000780130861]
Корень средней квадратичной ошибки: 0.02522180300132976

Проверка на тестовом наборе:

Прогнозы: [0.68269895 0.39414914 0.30276715 0.61621728 0.34923546]
Ожидания: [0.6965355290630153, 0.3334923153602843, 0.20910755281820587, 0.5628028436040677, 0.4352512630265198]
Корень средней квадратичной ошибки: 0.06230465900993596
```

Особых изменений нет.

Регрессия с голосованием

Данный метод заключается в обучении нескольких моделей после которого общий прогноз строится на среднем значении всех их прогнозов.

Попробуем использовать для этого стохастический градиентный спуск, дерево решений с адаптивным бустингом, метод опорных векторов и градиентный бустинг:

```
# Модели
reg1 = SGDRegressor(max_iter=99999999, penalty='l2', eta0=0.025, power_t=0, tol=0.03, loss="squared_error")
reg2 = AdaBoostRegressor(GS_DTR.best_estimator_, n_estimators=100, learning_rate=0.1, loss='square')
reg3 = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
reg4 = GradientBoostingRegressor(max_depth=2, n_estimators=3000, learning_rate=0.01, subsample=0.25)

# Обучение
VR = VotingRegressor(estimators=[('lr', reg1), ('dt', reg2), ('svr', reg3), ('gd', reg4)])
VR.fit(train_data, train_labels)
```

Результат:

Проверка на тренировочном наборе:

Прогнозы: [0.92611878 0.24978819 0.2623451 0.61698811 0.38284074]
Ожидания: [0.957705352733913, 0.21521232590620348, 0.2606165757481857, 0.6534205691290322, 0.3300583696241585]
Корень средней квадратичной ошибки: 0.05127938266138475

Проверка на тестовом наборе:

Прогнозы: [0.89193384 0.52619371 0.59047786 0.71642699 0.57787595]
Ожидания: [0.9590407718469127, 0.4819146001880991, 0.5112938206740878, 0.6302987444921759, 0.5257926567580822]
Корень средней квадратичной ошибки: 0.06663277992350355

Кросс-валидация:

Средняя оценка: 0.06345635923616719

Мы получили средний результат относительно всех других методов: ≈ 0.06

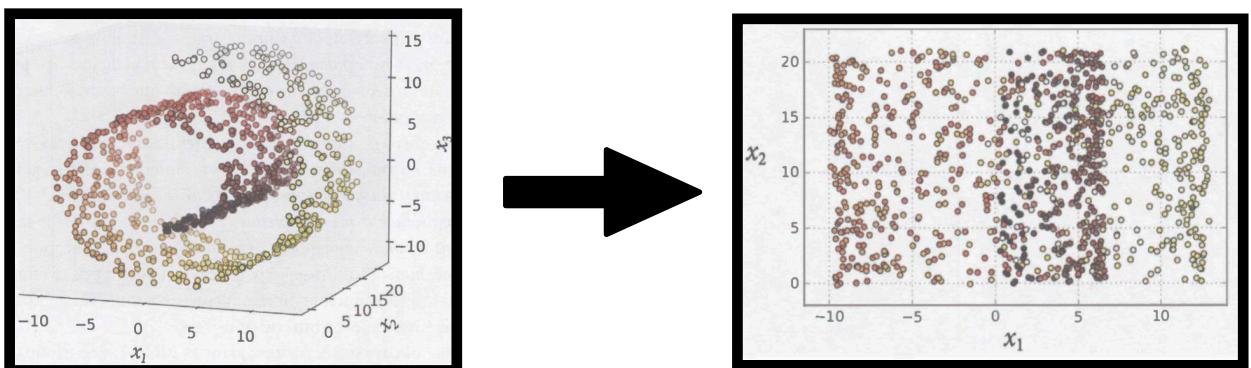
Понижение размерности

Напоследок, попробуем уменьшить размерность данных с сохранением 95% дисперсии — чувствительности к небольшим изменениям в данных. Суть понижения размерности заключается в упрощении представления данных с максимальным сохранением информации. Для этого воспользуемся **методом главных компонент (PCA)**.

```
pca = PCA(n_components=0.95)
train_data_PCA = pca.fit_transform(train_data)
test_data_PCA = pca.fit_transform(test_data)
```

Он представляет собой ортогональное линейное преобразование, которое отображает данные из исходного пространства признаков в новое пространство меньшей размерности

Пример понижения размерности:



Попробуем обучить на этих данных дерево решений с решётчатым поиском гиперпараметров и адаптивным бустингом.

Результат:

```
Проверка на тренировочном наборе:
```

```
Прогнозы: [0.24998217 0.65700059 0.49591742 0.41999574 0.37522098]
Ожидания: [0.3050669547951684, 0.6659925770499675, 0.514155449084116, 0.5011827902470916, 0.33215690216128374]
Корень средней квадратичной ошибки: 0.04845910280898717
```

```
Проверка на тестовом наборе:
```

```
Прогнозы: [0.69836433 0.34841508 0.51159634 0.8949887 0.66403942]
Ожидания: [0.4838223417780984, 0.13970392483893418, 0.5402914928420763, 0.931359483918661, 0.14220304896138475]
Корень средней квадратичной ошибки: 0.12745661180712248
```

Модель переобучается: ошибка на тестовом наборе довольно велика — 0.12.

Теперь попробуем обучить на этих данных регрессию с голосованием:

```
Проверка на тренировочном наборе:
```

```
Прогнозы: [0.88154232 0.22432556 0.31823575 0.57287131 0.35503769]
Ожидания: [0.957705352733913, 0.21521232590620348, 0.2606165757481857, 0.6534205691290322, 0.3300583696241585]
Корень средней квадратичной ошибки: 0.05988078702264582
```

```
Проверка на тестовом наборе:
```

```
Прогнозы: [0.88357206 0.43914857 0.55704671 0.70435783 0.60354019]
Ожидания: [0.9590407718469127, 0.4819146001880991, 0.5112938206740878, 0.6302987444921759, 0.5257926567580822]
Корень средней квадратичной ошибки: 0.08743809484659634
```

```
Кросс-валидация:
```

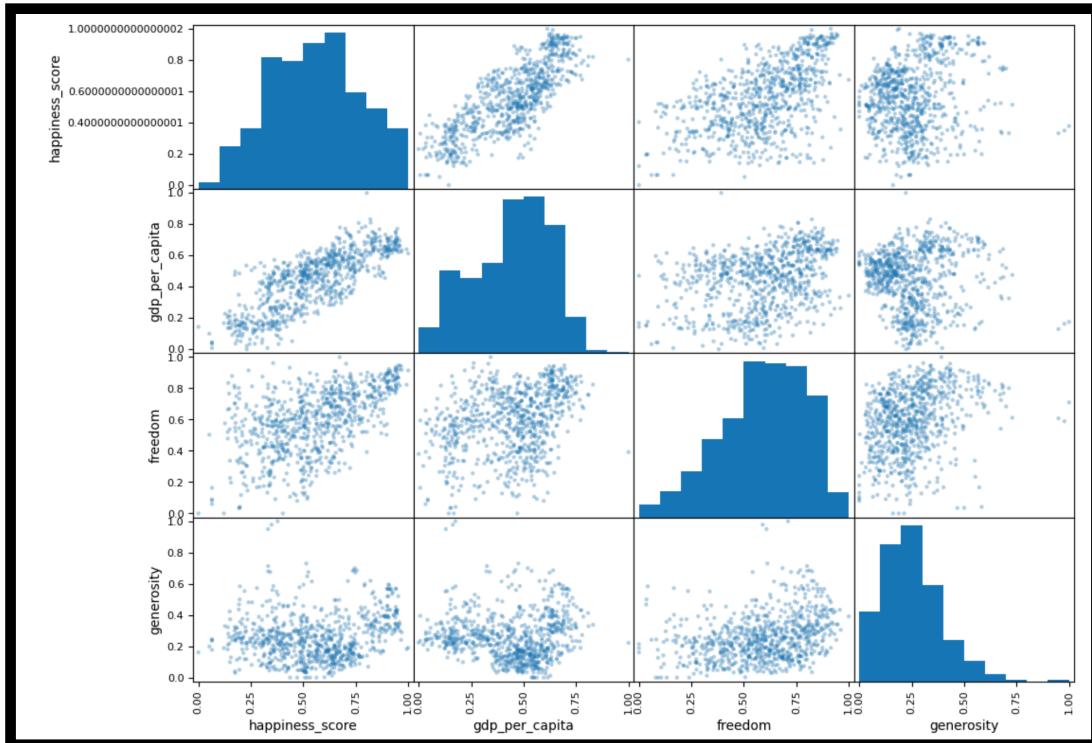
```
Средняя оценка: 0.0725491075508879
```

Уменьшение размерности ухудшило результат.

Заключение

Перед подведением итогов, взглянем, что представляют из себя зависимости показателя «`happiness_score`» относительно некоторых атрибутов с помощью `scatter_matrix()`:

```
# График зависимости показателя счастья от некоторых других
scatter_matrix(data[["happiness_score", "gdp_per_capita",
                      "freedom", "generosity"]], figsize=(12, 9), alpha=0.35)
```



Как видим, параметры образуют разные рисунки. Например, «`gdp_per_capita`» имеет вполне линейную форму, а «`freedom`» очень сильно разбросан. К тому же, сам набор состоит всего из 792 примеров, что довольно мало.

На основе этих фактов можно предположить, что данных хватает для обучения (хоть и неточного) простых моделей, по типу линейной регрессии, но недостаточно для полноценного обучения более мощных, таких как дерево решений, такими же методами мы не пользовались.

Несмотря на это, можно сделать вывод, что правильное использование и сочетание разных подходов (регуляризация моделей, ансамблевое обучение) улучшает результаты, так как некоторые модели недообучались, но явно стремились к хорошему показателю ошибки.

Список источников

1. [kaggle.com](https://www.kaggle.com) - социальная сеть для специалистов по обработке данных и машинному обучению
2. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow - O. Жерон

Приложение

Код программы

```
import os
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error

# Загрузка данных
data = pd.read_csv("datasets/happiness/data.csv")
data

# Удаление лишних атрибутов
data = data.drop("Year", axis=1)
data = data.drop("Country", axis=1)
data = data.drop("continent", axis=1)

# Корреляция значений относительно показателя счастья
data.corr()["happiness_score"].sort_values(ascending=False)

# Нормализация значений (от 0 до 1)

MMS = MinMaxScaler(copy=True).fit(data)
data = MMS.transform(data)
data = pd.DataFrame(data, columns = ['happiness_score', 'gdp_per_capita',
                                      'family', 'health', 'freedom', 'generosity',
                                      'government_trust', 'dystopia_residual',
                                      'social_support', 'cpi_score'])

X = data.drop('happiness_score', axis=1)
y = data['happiness_score'].copy()

# Деление на тренировочный и тестовый наборы
train_data, test_data, train_labels, test_labels = train_test_split(X, y, test_size=0.2)

# График зависимости показателя счастья от некоторых других
scatter_matrix(data[["happiness_score", "gdp_per_capita",
                      "freedom", "generosity"]], figsize=(12, 9), alpha=0.35)

# Кривые обучения
def plot_learning_curves(model):
    X_train = train_data.values
    X_val = test_data.values
    y_train = train_labels.values
    y_val = test_labels.values
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
        val_errors.append(mean_squared_error(y_val_predict, y_val))

    plt.figure(figsize=(12, 6))
    plt.plot(train_errors, label='Training Error')
    plt.plot(val_errors, label='Validation Error')
    plt.legend()
    plt.title('Learning Curves')
    plt.xlabel('Number of training samples')
    plt.ylabel('Mean Squared Error')

    plt.show()
```

```

plt.plot(np.sqrt(train_errors), "r-", linewidth=1, label="train")
plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")

```

```
# Линейная регрессия
```

```
# Обучение без кросс-валидации
print("Без кросс-валидации\n")
LR = LinearRegression()
LR.fit(train_data, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
LR_predictions = LR.predict(train_data)
print("Прогнозы: ", LR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
LR_rmse = np.sqrt(mean_squared_error(train_labels, LR_predictions))
print("Корень средней квадратичной ошибки: ", LR_rmse, "\n")

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
LR_predictions = LR.predict(test_data)
print("Прогнозы: ", LR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
LR_rmse = np.sqrt(mean_squared_error(test_labels, LR_predictions))
print("Корень средней квадратичной ошибки: ", LR_rmse, "\n")

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация')
score = cross_val_score(LR, train_data, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), "\n\n")

print("\tКривые обучения:")
plot_learning_curves(LR)


```

```
# Полиномиальная регрессия
```

```
# Создание полиномиальных признаков
poly_features = PolynomialFeatures(degree=2, include_bias=False)
train_data_poly = poly_features.fit_transform(train_data)
test_data_poly = poly_features.fit_transform(test_data)

# Обучение
LR_poly = LinearRegression()
LR_poly.fit(train_data_poly, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
LR_poly_predictions = LR_poly.predict(train_data_poly)
print("Прогнозы: ", LR_poly_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
LR_poly_rmse = np.sqrt(mean_squared_error(train_labels, LR_poly_predictions))
print("Корень средней квадратичной ошибки: ", LR_poly_rmse, "\n")

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
LR_poly_predictions = LR_poly.predict(test_data_poly)
print("Прогнозы: ", LR_poly_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
LR_poly_rmse = np.sqrt(mean_squared_error(test_labels, LR_poly_predictions))
print("Корень средней квадратичной ошибки: ", LR_poly_rmse, "\n")

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация')
score = cross_val_score(LR, train_data_poly, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), "\n\n")

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(LR_poly)


```

```
# Полиномиальная регрессия со стохастическим градиентным спуском
```

```
# Обучение без кросс-валидации
```

```

print("Без кросс-валидации\n")
SGDR = SGDRegressor(max_iter=99999999, penalty='l2',
                     eta0=0.025, power_t=0, tol=0.03,
                     loss="squared_error"
                    )
SGDR.fit(train_data_poly, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
SGDR_predictions = SGDR.predict(train_data_poly)
print("Прогнозы: ", SGDR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
SGDR_rmse = np.sqrt(mean_squared_error(train_labels, SGDR_predictions))
print("Корень средней квадратичной ошибки: ", SGDR_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
SGDR_predictions = SGDR.predict(test_data_poly)
print("Прогнозы: ", SGDR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
SGDR_rmse = np.sqrt(mean_squared_error(test_labels, SGDR_predictions))
print("Корень средней квадратичной ошибки: ", SGDR_rmse, '\n')

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация')
score = cross_val_score(SGDR, train_data_poly, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), '\n\n')

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(SGDR)


---


# Метод опорных векторов

# Обучение
svr = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svr.fit(train_data, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
svr_predictions = svr.predict(train_data)
print("Прогнозы: ", svr_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
svr_rmse = np.sqrt(mean_squared_error(train_labels, svr_predictions))
print("Корень средней квадратичной ошибки: ", svr_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
svr_predictions = svr.predict(test_data)
print("Прогнозы: ", svr_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
svr_rmse = np.sqrt(mean_squared_error(test_labels, svr_predictions))
print("Корень средней квадратичной ошибки: ", svr_rmse, '\n')

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация')
score = cross_val_score(svr, train_data, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), '\n')

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(svr)


---


# Дерево решений

# Обучение без кросс-валидации
print("Без кросс-валидации\n")
DTR = DecisionTreeRegressor()
DTR.fit(train_data, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
DTR_predictions = DTR.predict(train_data)

```

```

print("Прогнозы: ", DTR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
DTR_rmse = np.sqrt(mean_squared_error(train_labels, DTR_predictions))
print("Корень средней квадратичной ошибки: ", DTR_rmse, "\n")

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
DTR_predictions = DTR.predict(test_data)
print("Прогнозы: ", DTR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
DTR_rmse = np.sqrt(mean_squared_error(test_labels, DTR_predictions))
print("Корень средней квадратичной ошибки: ", DTR_rmse, "\n")

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация')
score = cross_val_score(DTR, train_data, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), "\n\n")

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(DTR)



---


# Дерево решений, регуляризованное с использованием решётчатого поиска (вместе с кросс-валидацией)

# Гиперпараметры
hyperparams = [{'max_depth': [22, 23, 24, 25], 'min_samples_leaf': [2, 3, 4, 5],
                'min_samples_split': [2, 4, 6, 8], 'max_features': [6, 7, 8, 9],
                'max_leaf_nodes': [25, 26, 27, 28]}]

# Обучение модели
DTR = DecisionTreeRegressor()
GS_DTR = GridSearchCV(DTR, hyperparams, cv=10, scoring='neg_mean_squared_error')
GS_DTR.fit(train_data, train_labels)
print("Лучший результат: ", np.sqrt(-GS_DTR.best_score_))

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
DTR_predictions = GS_DTR.best_estimator_.predict(train_data)
print("Прогнозы: ", DTR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
DTR_rmse = np.sqrt(mean_squared_error(train_labels, DTR_predictions))
print("Корень средней квадратичной ошибки: ", DTR_rmse, "\n")

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
DTR_predictions = GS_DTR.best_estimator_.predict(test_data)
print("Прогнозы: ", DTR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
DTR_rmse = np.sqrt(mean_squared_error(test_labels, DTR_predictions))
print("Корень средней квадратичной ошибки: ", DTR_rmse, "\n\n")

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(GS_DTR.best_estimator_)
GS_DTR.best_estimator_



---


# Дерево принятия решений с ансамблевым обучением (Адаптивным бустингом)
# после регуляризации

# Обучение модели
ABR = AdaBoostRegressor(GS_DTR.best_estimator_, n_estimators=100, learning_rate=0.1, loss='square')
ABR.fit(train_data, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
ABR_predictions = ABR.predict(train_data)
print("Прогнозы: ", ABR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
ABR_rmse = np.sqrt(mean_squared_error(train_labels, ABR_predictions))
print("Корень средней квадратичной ошибки: ", ABR_rmse, "\n")

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")

```

```

ABR_predictions = ABR.predict(test_data)
print("Прогнозы: ", ABR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
ABR_rmse = np.sqrt(mean_squared_error(test_labels, ABR_predictions))
print("Корень средней квадратичной ошибки: ", ABR_rmse, '\n\n')

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация')
score = cross_val_score(ABR, train_data, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), '\n\n')

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(ABR)



---


# Метод случайного леса с решётчатым поиском (вместе с кросс-валидацией)

# Гиперпараметры
hyperparams = [
    {'n_estimators': [2, 4, 10, 20, 50, 100], 'max_features': [2, 4, 6, 8, 9, 12, 16]}
]

# Обучение и оценка
print("Решётчатый поиск и кросс-валидация: \n")
RDR = RandomForestRegressor()
GS_RDR = GridSearchCV(RDR, hyperparams, cv=10, scoring='neg_mean_squared_error')
GS_RDR.fit(train_data, train_labels)
print("Лучший результат: ", np.sqrt(-GS_RDR.best_score_))

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
GS_RDR_predictions = GS_RDR.best_estimator_.predict(train_data)
print("Прогнозы: ", GS_RDR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
GS_RDR_rmse = np.sqrt(mean_squared_error(train_labels, GS_RDR_predictions))
print("Корень средней квадратичной ошибки: ", GS_RDR_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
GS_RDR_predictions = GS_RDR.best_estimator_.predict(test_data)
print("Прогнозы: ", GS_RDR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
GS_RDR_rmse = np.sqrt(mean_squared_error(test_labels, GS_RDR_predictions))
print("Корень средней квадратичной ошибки: ", GS_RDR_rmse, '\n\n')

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(GS_RDR.best_estimator_)



---


# Метод случайного леса с рандомизированным поиском (вместе с кросс-валидацией)

# Гиперпараметры
hyperparams = [
    {'n_estimators': [i for i in range(1, 200)], 'max_features': [i for i in range(1, 50)]}
]

# Обучение и оценка
print("Рандомизированный поиск и кросс-валидация: \n")
RDR = RandomForestRegressor()
RS_RDR = RandomizedSearchCV(RDR, hyperparams, n_iter=20, cv=10, scoring='neg_mean_squared_error')
RS_RDR.fit(train_data, train_labels)
print("Лучший результат: ", np.sqrt(-RS_RDR.best_score_))

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
RS_RDR_predictions = RS_RDR.best_estimator_.predict(train_data)
print("Прогнозы: ", RS_RDR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
RS_RDR_rmse = np.sqrt(mean_squared_error(train_labels, RS_RDR_predictions))
print("Корень средней квадратичной ошибки: ", RS_RDR_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
RS_RDR_predictions = RS_RDR.best_estimator_.predict(test_data)

```

```

print("Прогнозы: ", RS_RDR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
RS_RDR_rmse = np.sqrt(mean_squared_error(test_labels, RS_RDR_predictions))
print("Корень средней квадратичной ошибки: ", RS_RDR_rmse, '\n\n')

# График процесса обучения
print("\tКривые обучения:")
plot_learning_curves(RS_RDR.best_estimator_)



---


# Градиентный бустинг

# Обучение
GB = GradientBoostingRegressor(max_depth=2, n_estimators=3000, learning_rate=0.01, subsample=0.25)
GB.fit(train_data, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
GB_predictions = GB.predict(train_data)
print("Прогнозы: ", GB_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
GB_rmse = np.sqrt(mean_squared_error(train_labels, GB_predictions))
print("Корень средней квадратичной ошибки: ", GB_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
GB_predictions = GB.predict(test_data)
print("Прогнозы: ", GB_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
GB_rmse = np.sqrt(mean_squared_error(test_labels, GB_predictions))
print("Корень средней квадратичной ошибки: ", GB_rmse, '\n')

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация')
score = cross_val_score(GB, train_data, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), '\n')



---


# Уменьшение размерности с сохранением 95% дисперсии данных

pca = PCA(n_components=0.95)
train_data_PCA = pca.fit_transform(train_data)
test_data_PCA = pca.fit_transform(test_data)



---


# Дерево принятия решений с ансамблевым обучением (Адаптивным бустингом)
# и решётчатым поиском на данных с уменьшенной размерностью

# Гиперпараметры
hyperparams = [ {'max_depth': [18, 19, 20, 21], 'min_samples_leaf': [2, 3, 4, 5, 6],
                 'min_samples_split': [2, 3, 4], 'max_features': [4, 5, 6],
                 'max_leaf_nodes': [38, 39, 40]}]

# Обучение
DTR = DecisionTreeRegressor()
GS_DTR_PCA = GridSearchCV(DTR, hyperparams, cv=10)
GS_DTR_PCA.fit(train_data_PCA, train_labels)
ABR = AdaBoostRegressor(GS_DTR_PCA.best_estimator_, n_estimators=100, learning_rate=0.01, loss='square')
ABR.fit(train_data_PCA, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
ABR_predictions = ABR.predict(train_data_PCA)
print("Прогнозы: ", ABR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
ABR_rmse = np.sqrt(mean_squared_error(train_labels, ABR_predictions))
print("Корень средней квадратичной ошибки: ", ABR_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
ABR_predictions = ABR.predict(test_data_PCA)
print("Прогнозы: ", ABR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
ABR_rmse = np.sqrt(mean_squared_error(test_labels, ABR_predictions))
print("Корень средней квадратичной ошибки: ", ABR_rmse, '\n')



---


# Регрессия с голосованием с использованием лучших моделей

```

```

# Модели
reg1 = SGDRegressor(max_iter=99999999, penalty='l2', eta0=0.025, power_t=0, tol=0.03, loss="squared_error")
reg2 = AdaBoostRegressor(GS_DTR.best_estimator_, n_estimators=100, learning_rate=0.1, loss='square')
reg3 = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
reg4 = GradientBoostingRegressor(max_depth=2, n_estimators=3000, learning_rate=0.01, subsample=0.25)

# Обучение
VR = VotingRegressor(estimators=[('lr', reg1), ('dt', reg2), ('svr', reg3), ('gd', reg4)])
VR.fit(train_data, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
VR_predictions = VR.predict(train_data)
print("Прогнозы: ", VR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
VR_rmse = np.sqrt(mean_squared_error(train_labels, VR_predictions))
print("Корень средней квадратичной ошибки: ", VR_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
VR_predictions = VR.predict(test_data)
print("Прогнозы: ", VR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
VR_rmse = np.sqrt(mean_squared_error(test_labels, VR_predictions))
print("Корень средней квадратичной ошибки: ", VR_rmse, '\n')

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация:')
score = cross_val_score(VR, train_data, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), '\n')

# Регрессия с голосованием на данных
# с уменьшенной размерностью


---


# Модели
reg1 = SGDRegressor(max_iter=99999999, penalty='l2', eta0=0.025, power_t=0, tol=0.03, loss="squared_error")
reg2 = AdaBoostRegressor(GS_DTR.best_estimator_, n_estimators=100, learning_rate=0.1, loss='square')
reg3 = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
reg4 = GradientBoostingRegressor(max_depth=2, n_estimators=3000, learning_rate=0.01, subsample=0.25)

# Обучение
VR = VotingRegressor(estimators=[('sgdr', reg1), ('dt', reg2), ('svr', reg3), ('gd', reg4)])
VR.fit(train_data_PCA, train_labels)

# Оценка модели на тренировочном наборе с помощью RMSE
print("Проверка на тренировочном наборе:\n")
VR_predictions = VR.predict(train_data_PCA)
print("Прогнозы: ", VR_predictions[:5])
print("Ожидания: ", list(train_labels.values)[:5])
VR_rmse = np.sqrt(mean_squared_error(train_labels, VR_predictions))
print("Корень средней квадратичной ошибки: ", VR_rmse, '\n')

# Оценка модели на тестовом наборе с помощью RMSE
print("Проверка на тестовом наборе:\n")
VR_predictions = VR.predict(test_data_PCA)
print("Прогнозы: ", VR_predictions[:5])
print("Ожидания: ", list(test_labels.values)[:5])
VR_rmse = np.sqrt(mean_squared_error(test_labels, VR_predictions))
print("Корень средней квадратичной ошибки: ", VR_rmse, '\n')

# Обучение и оценка с кросс-валидацией на тренировочном наборе
print('Кросс-валидация:')
score = cross_val_score(VR, train_data_PCA, train_labels, scoring="neg_mean_squared_error", cv=10)
score = np.sqrt(-score)
print("Средняя оценка: ", score.mean(), '\n')

```