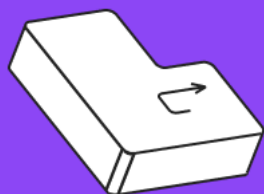




Линейная регрессия

Машинное обучение



Оглавление

Введение	2
Термины, используемые в лекции	2
Линейная регрессия	3
Обучаем модель линейной регрессии	
Функция потерь	
Регуляризация	5
Домашнее задание	6
Что можно почитать еще?	6
Используемая литература	6

Введение

На этом занятии мы рассмотрим алгоритм линейной регрессии. Данный алгоритм существует давно и хорошо известен. Также, известны подходы в работе с ним.

Мы рассмотрим с вами обучение модели линейной регрессии в Google Collab, а затем усовершенствуем модель. В большинстве случаев линейной регрессии недостаточно просто обучения, требуется еще и регуляризация. Мы разберемся с этим понятием.

Термины, используемые в лекции

Задача регрессии – поиск отображения множества объектов в множество признаков.

МНК (Метод наименьших квадратов) – один из способов нахождения точки минимума для линейной регрессии.

Линейная регрессия – линейный алгоритм для предсказания значений в диапазоне от минус бесконечность до плюс бесконечность.

Регуляризация – дополнительное ограничение на вектор весов.

Линейная регрессия.

На этом занятии мы рассмотрим алгоритм линейной регрессии. Данный алгоритм существует давно и хорошо известен. Также, известны подходы в работе с ним.

Мы начнем с того, что посмотрим на процесс создания модели машинного обучения сверху, и разберем основные этапы разработки. Потом перейдем к задаче регрессии и разберем, как работает линейная регрессия, какими параметрами она обладает. Затем, на практике, мы рассмотрим несколько примеров обучения модели линейной регрессии.

На прошлом занятии вы познакомились с большим количеством определений, которые применяются в машинном обучении. Давайте наведем в порядок в полученных знаниях и сформируем четкий алгоритм, которого мы будем придерживаться при решении задач. Такой алгоритм применяется на практике при решении прикладных задач. Его еще называют пайплайн (pipeline) разработки.

Первый, и наверно самый важный этап – это сбор и подготовка данных. От данных напрямую зависит, какой получится наша модель. В этот этап входит так же и сходит проектирование признаков, про которое мы говорили на прошлом занятии. Так же мы должны понимать, что мы ищем, и на какой вопрос должны ответить собранные данные. На ближайших занятиях мы будем исходить из того, что наши данные идеальны, но на практике такое встречается редко. Поэтому на одном из следующих занятий мы обязательно разберем EDA первичный анализ данных и проблемы, которые есть у сырых данных.

После того как данные собраны, мы переходим к обучению модели. На этом этапе нужно выбрать алгоритм, который поможет решить задачу. Как правило, имеющиеся данные определяют алгоритм.

После обучения нам нужно понимать, как работает наша модель, насколько верные предсказания она делает. В этом нам помогут различные метрики, например средняя квадратичная ошибка (MSE) или средняя абсолютная ошибка (MAE). Метрик много, и чтобы выбрать правильные, нужно отталкиваться от обучающего алгоритма.

Когда мы убедились, что модель обучена хорошо, наступает следующий этап — оптимизация. На этом этапе разработчики обычно возвращаются по пайплайну назад, и смотрят как можно улучшить модель, начиная с обучающих данных.



Вот так выглядит полный цикл разработки. С каждым из пунктов мы познакомимся более детально на одном из следующих занятий. Тем не менее мы будем придерживаться именно такого алгоритма, даже если будем пропускать некоторые пункты.

На прошлом занятии мы разбирали конкретную задачу прогнозирования стоимости жилплощади за одну ночь.

Давайте рассмотрим первый алгоритм, который позволяет решать задачи связанные с численным прогнозированием — этот алгоритм называется линейная регрессия. Примерами линейной регрессии является предсказание стоимости домов, погоды на следующей день и прочие задачи, где цель — предсказать число.

Разберемся на примере. Пусть у нас будет некое множество объектов k , имеющее d признаков.

И второе множество — множество y .



Модель машинного обучения

Множество объектов k , каждый объект имеет d признаков

Множество объектов y

Допустим, это будут ноутбуки. То есть у нас есть некоторое количество ноутбуков k , и у каждого из этих ноутбуков есть число признаков d . Допустим это будет количество ядер, диагональ экрана и частота процессора. Целевая переменная – цена ноутбука.

Мы считаем, что между признаками и целевой переменной есть какая-то зависимость. Нам нужно определить эту зависимость.

Мы предполагаем, что такую зависимость можно отобразить в виде формулы, которую вы видите на слайде:

Получается, для того, чтобы найти зависимость некой целевой переменной от каких-то признаков, нужно произвести сумму всех признаков, умноженных на веса. А что такое веса? Это коэффициенты, которые и получаются в результате обучения модели!

Давайте рассмотрим пример, в котором целевая переменная зависит только от одного признака, то есть имеет вид:

То есть у нас получилось обычное уравнение прямой. В таком виде w_1 это наклон нашей линии, а w_0 – сдвиг относительно оси y . Это самый простой случай регрессии. График такой функции будет выглядеть как прямая, проведенная через облако точек, соответствующих объектам. Конечно, в жизни редко встречается

такая зависимость. Почти всегда целевая переменная зависит от нескольких признаков.

На самом деле, в наше уравнение мы можем добавлять дополнительные слагаемые для получения более сложной регрессии. Аналогично, мы можем работать и с другими видами функций, а в случае более высоких размерностей, на графике вместо прямой будет гиперплоскость с аналогичным смыслом

У нас есть еще w_0 , который называется свободный член или bias. Давайте избавимся от него. Мы введем некий фиктивный признак, который для всех элементов будет равен единице. Если это так, то уравнение регрессии приобретает вид как на слайде. Получается, чтобы найти зависимость, нужно посчитать скалярное произведение весов на признаки объекта. Замечательно!

Так же, для удобства расчетов математическую форму записи задачи регрессии обычно представляют в матричном виде.

Такой вид удобен, но только для математических расчетов. На практике, если мы создаем модель, он не так полезен. Мы не будем выводить формулу, так как это за рамками нашего курса.

Решение уравнения регрессии с помощью МНК

Для решения задачи линейной регрессии (и не только) нужно уменьшать функцию потерь, это мы выяснили с вами на прошлом занятии. Вот только как это сделать?

На самом деле, есть очень много вариантов решения данной задачи. Мы можем решить геометрически, можем решить с помощью вероятностных подходов, можем решить численно верно. Минуя вывод формул, могу сказать, что данные решения нас не удовлетворяют. Именно поэтому мы рассмотрим с вами метод приближенного численного решения – метод наименьших квадратов.

Посмотрим как это будет выглядеть на практике.

Пусть у нас есть пять точек и два вектора предсказаний – y_1 и y_2 – от двух разных моделей.

```
import numpy as np
```

```
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 2, 3, 4, 5])
y1 = np.array([1, 2, 3, 4, 5])
y2 = np.array([1, 1, 4, 9, 0])
```

Отообразим их на графике:

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.scatter(x, y, s=30)
plt.plot(x, y1, 'g')
plt.plot(x, y2, 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Очевидно, что зеленая линия, которая соответствует y_1 , имеет лучшую модель. Однако посчитаем ошибку по приведенной ранее формуле

```
err1 = sum(y1-y)
err2 = sum(y2-y)

print('ошибка err1:', err1)
print('ошибка err2:', err2)
```

```
ошибка err1: 0
ошибка err2: 0
```

Смотрите что мы получили. Первая модель предсказала вектор абсолютно верно, и ошибка равна нулю. Но вторая модель предсказала вектор неверно, а ошибка все равно равна нулю. Это произошло из-за взаимоуничтожения ошибок -1, 1 и -5, 5 (если произвести в уме вычисления то так и получается). Такой результат нас не устает, поэтому давайте возводить результат в квадрат:

```
err1 = sum((y1-y)**2)
err2 = sum((y2-y)**2)

print('ошибка err1:', err1)
print('ошибка err2:', err2)
```

```
ошибка err1: 0
ошибка err2: 52
```

Тогда мы будем минимизировать квадратичную функцию потерь. Поэтому метод называется методом наименьших квадратов или МНК.

<https://colab.research.google.com/drive/1p2Ugr5XeswFHoIhVh6OwtPFx4Q92IrPf?usp=ssharing>

Обучение модели линейной регрессии

Практика.

https://colab.research.google.com/drive/1T0vLWLzQK4Yn5aZS0Z_c4_s12diUztdA?usp=ssharing

Зная способы уменьшения функции потерь, мы сможем создать модель машинного обучения. Давайте посмотрим, как это будет выглядеть на практике.

Обучение модели с помощью sklearn

```
%matplotlib inline  
  
import matplotlib.pyplot as plt  
  
import numpy as np  
  
import pandas as pd
```

Пример 1

Пусть нам заданы 1000 точек — это матрица X, и есть целевая функция Y.

```
np.random.seed(0) # для воспроизведения результатов  
  
x = np.random.rand(1000, 1)  
  
y = 1 + 3 * x + np.random.randn(1000, 1)  
  
d = {'x': x.reshape(1,1000)[0], 'y': y.reshape(1,1000)[0]}  
df = pd.DataFrame(data=d)  
df.head()
```

x	y
---	---


```
0 0.548814 2.544743
1 0.715189 3.164847
2 0.602763 4.657881
3 0.544883 2.420483
4 0.423655 1.771948
```

```
plt.scatter(df['x'], df['y'], s=10)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
[]
```

Мы хотим найти зависимость с помощью линейной регрессии.

Разделим исходные данные на обучающую и тестовую выборки в соотношении 70% и 30%.

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(df, train_size=0.7, random_state=123) # random_state
нужен для воспроизводимости результата
```

Для построения самой модели импортируем класс LinearRegression из sklearn.linear_model.

```
from sklearn.linear_model import LinearRegression
```

Теперь у нас есть всё необходимое для построения модели.

Создадим экземпляр класса `LinearRegression`, который будет содержать в себе модель регрессии.

```
model = LinearRegression()
```

Чтобы обучить модель, необходимо вызвать метод `.fit()` на объект `model`. В качестве аргументов здесь передаются матрица X и целевая функция Y .

Обучать модель будем на обучающей выборке.

```
model.fit(train[['x']], train[['y']])
```

```
LinearRegression()
```

Поздравляем, модель готова!

Можно сделать всё проще, в одну строчку. Эта операция делает то же самое.

```
model = LinearRegression().fit(train[['x']], train[['y']])
```

У полученного объекта `model` есть атрибуты: `.intercept_` — представляет собой коэффициент w_0 , `.coef_` — представляют собой w_1, \dots, w_k .

```
print('w_0:', model.intercept_)
```

```
print('w_1:', model.coef_)
```

```
w_0: [1.08258227]
```

```
w_1: [[2.84716323]]
```

Полученная модель имеет вид $y^* = 1.08 + 2.85 \cdot x_1$

Напомним, что исходная модель — $y = 1 + 3 \cdot x + \text{np.random.randn}(1000, 1)$

Теперь можно использовать модель для прогноза значений. Для этого будем использовать метод `.predict()`. В качестве аргументов для данного метода требуется матрица X .

```
y_predict_train = model.predict(train[['x']])
```

```
print(y_predict_train[:, 0][:10])
```

```
[1.78985595 1.45860173 2.99601208 2.15810342 3.66369375 1.16785573
```

```
1.80559558 1.45979619 1.11593418 2.08447907]
```

Проверим этот расчёт по найденной формуле.

```
y_predict_train = model.intercept_ + model.coef_ * train[['x']]
```

```
print(y_predict_train[:10])
```

x

```
498 1.789856
```

```
243 1.458602
```

```
314 2.996012
```

202 2.158103

300 3.663694

682 1.167856

230 1.805596

306 1.459796

166 1.115934

620 2.084479

Посмотрим на графике, какую модель получили.

```
plt.plot(train['x'], y_predict_train, linewidth=4, c='g')
```

```
plt.scatter(train['x'], train['y'], s=10, c='b')
```

```
plt.scatter(test['x'], test['y'], s=10, c='r')
```

```
plt.xlabel('x')
```

```
plt.ylabel('y')
```

```
plt.show()
```

```
[]
```

Чтобы получить результаты модели по тестовой выборке, выполним следующее:

```
y_predict_test = model.predict(test[['x']])
```

Оценим качество модели с помощью MSE.

```
from sklearn.metrics import mean_squared_error
```

```
train_mse = mean_squared_error(train[['y']], y_predict_train)
```

```
test_mse = mean_squared_error(test[['y']], y_predict_test)

print("Train MSE: {}".format(train_mse))

print("Test MSE: {}".format(test_mse))
```

Train MSE: 0.9599829425512107

Test MSE: 0.8715242272540316

Оценим также качество модели с помощью MAE.

```
from sklearn.metrics import mean_absolute_error

train_mae = mean_absolute_error(train[['y']], y_predict_train)

test_mae = mean_absolute_error(test[['y']], y_predict_test)

print("Train MAE: {}".format(train_mae))

print("Test MAE: {}".format(test_mae))
```

Train MAE: 0.7845720071057997

Test MAE: 0.7261670560834567

Именно эти метрики чаще всего используются в задаче регрессии.

Остановимся на них подробнее.

Для примера посмотрим на задачу прогноза цены на автомобиль. Допустим, у нас нет ни одного параметра об автомобилях, есть только сами значения целевой функции. То есть прогнозировать будем модель следующего вида:

$$y^* = w_0.$$

По факту это означает, что будем прогнозировать для всех объектов одно число — константу.

Возникает вопрос, какое число нужно выбрать, если мы будем использовать оценку ошибки MAE? А если будем использовать ошибку MSE?

Итак, наилучшее качество такой модели при использовании метрики MAE

будет достигаться при медиане $w_0 = \text{median}(y)$.

А при использовании метрики MSE наилучшее качество такой модели будет достигаться при среднем значении $w_0 = \text{mean}(y)$.

Таким образом, можно отметить, что в метрике MAE по сравнению с MSE

существенно меньший вклад в ошибку будут вносить примеры, сильно удалённые от ответов модели. Так как используется модуль расстояния, а не квадрат.

И функция потерь MAE уместна в случаях, когда в данных большое количество выбросов в целевой функции.

Перейдём к следующему примеру.

Пример 2

Пусть нам также заданы 1000 точек — это матрица X , и есть целевая функция Y . Однако теперь сама по себе искомая зависимость нелинейная.

```
np.random.seed(0) # для воспроизведения результатов
```

```
x = np.random.rand(1000, 1)
```

```
y = 1 + 3 * np.log(x) + np.random.randn(1000, 1)
```

```
d = {'x': x.reshape(1,1000)[0], 'y': y.reshape(1,1000)[0]}
```

```
df = pd.DataFrame(data=d)
```

```
df.head()
```

```
      x      y
```

```
0 0.548814 -0.901687
```

```
1 0.715189  0.013656
```

```
2 0.602763 1.330900
3 0.544883 -1.035718
4 0.423655 -2.075526
```

```
plt.scatter(df['x'], df['y'], s=10)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
[]
```

Построим модель линейной регрессии, используя уже известные нам команды.

```
train, test = train_test_split(df, train_size=0.7, random_state=123) # random_state
нужен для воспроизводимости результата
```

```
model = LinearRegression()
model.fit(train[['x']], train[['y']])
```

```
LinearRegression()
```

```
y_predict_train = model.predict(train[['x']])
y_predict_test = model.predict(test[['x']])
```

```
plt.scatter(train['x'], train['y'], s=10, c='b')
plt.scatter(test['x'], test['y'], s=10, c='r')
```

```
plt.scatter(train['x'], y_predict_train, s=10, c='g')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.show()
```

```
[]
```

Очевидно, получилось не самое лучшее решение. Посмотрим на качество модели.

```
train_mse = mean_squared_error(train[['y']], y_predict_train)  
test_mse = mean_squared_error(test[['y']], y_predict_test)  
print("Train MSE: {}".format(train_mse))  
print("Test MSE: {}".format(test_mse))
```

```
Train MSE: 3.137684915760023
```

```
Test MSE: 1.9666504105743103
```

```
from sklearn.metrics import mean_absolute_error
```

```
train_mae = mean_absolute_error(train[['y']], y_predict_train)  
test_mae = mean_absolute_error(test[['y']], y_predict_test)  
print("Train MAE: {}".format(train_mae))  
print("Test MAE: {}".format(test_mae))
```

```
Train MAE: 1.2394688443948811
```

```
Test MAE: 1.0749333352063293
```


Как вы знаете, задача машинного обучения — найти наиболее оптимальную модель. Попробуем улучшить модель, добавив переменную логарифм от переменной x .

```
df['x2'] = df['x'].apply(lambda x: np.log(x))
```

```
train, test = train_test_split(df, train_size=0.7, random_state=123) # random_state  
нужен для воспроизводимости результата
```

Обратите внимание, здесь и далее будем использовать уже две переменных:
 x и $x2$.

```
model = LinearRegression()  
model.fit(train[['x', 'x2']], train[['y']])
```

```
LinearRegression()
```

```
y_predict_train = model.predict(train[['x', 'x2']])  
y_predict_test = model.predict(test[['x', 'x2']])
```

```
plt.scatter(train['x'], train['y'], s=10, c='b')  
plt.scatter(test['x'], test['y'], s=10, c='r')  
plt.scatter(train['x'], y_predict_train, s=10, c='g')  
plt.xlabel('x')  
plt.ylabel('y')
```

```
plt.show()
```

```
[]
```

График уже выглядит лучше. Оценим качество.

```
train_mse = mean_squared_error(train[['y']], y_predict_train)
```

```
test_mse = mean_squared_error(test[['y']], y_predict_test)
```

```
print("Train MSE: {}".format(train_mse))
```

```
print("Test MSE: {}".format(test_mse))
```

```
Train MSE: 0.9500342328699919
```

```
Test MSE: 0.8666177085273008
```

```
from sklearn.metrics import mean_absolute_error
```

```
train_mae = mean_absolute_error(train[['y']], y_predict_train)
```

```
test_mae = mean_absolute_error(test[['y']], y_predict_test)
```

```
print("Train MAE: {}".format(train_mae))
```

```
print("Test MAE: {}".format(test_mae))
```

```
Train MAE: 0.7802681943269093
```

```
Test MAE: 0.7283304269248566
```

Действительно, добавление переменной логарифма от x позволило улучшить качество модели, то есть снизить ошибку.

Проблема переобучения

Переобучение – это случай, когда значение Функции потери (Loss Function) действительно малó, но Модель (Model) Машинного обучения (ML) ненадежна. Это связано с тем, что модель «слишком много учится» на обучающем наборе данных.

Когда мы входим в сферу ML, появляются двусмысленные термины: Переобучение, Недообучение (Underfitting) и Дилемма смещения-дисперсии (Bias-Variance Trade-off). Эти концепции лежат в основе Машинного обучения в целом. Почему нам вообще должно быть до этого дело?

Возможно, модели машинного обучения преследуют одну единственную цель: хорошо *обобщать*.

Обобщение (генерализация) – это способность модели давать разумные предсказания на основе входных данных, которых она никогда раньше не видела.

Обычные программы не могут этого сделать, так как они могут выдавать выходные данные только алгоритмически, то есть на основании вручную определенных опций (например, если зарплата человека меньше определенного порога, банковский алгоритм не предлагает кредит в приложении). Производительность модели, а также ее полезность в целом во многом зависят от ее обобщающей способности. Если модель хорошо обобщает, она служит своей цели. Существует множество методов оценки такой производительности.

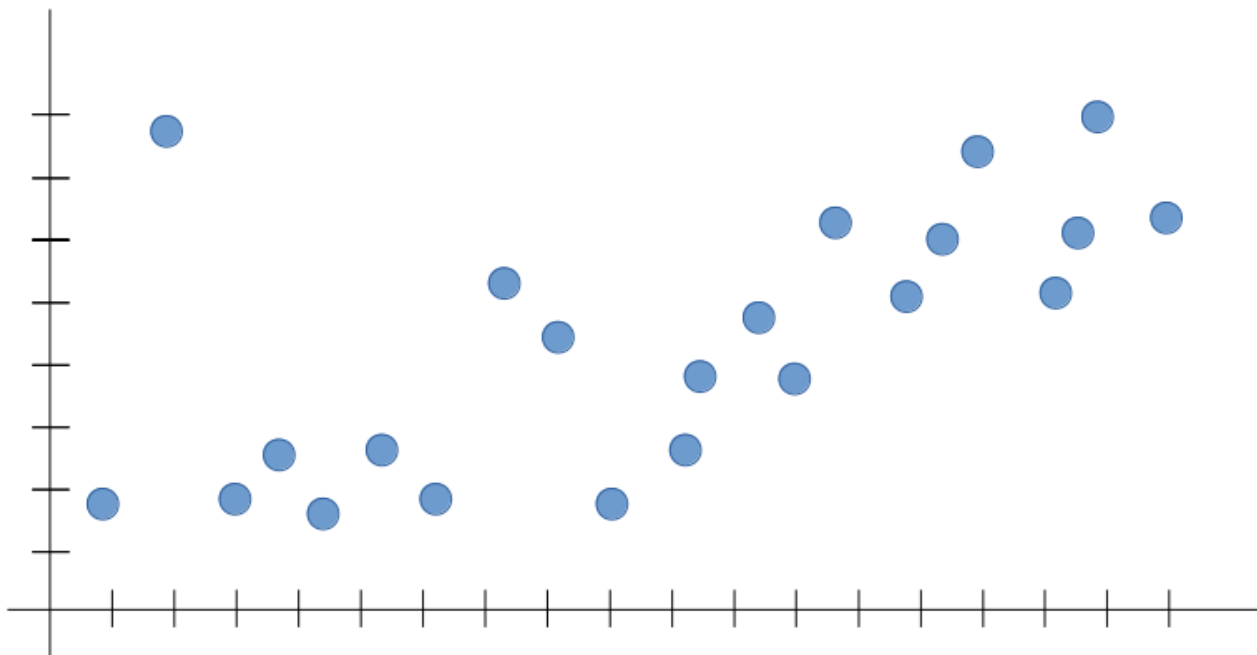
Основываясь на этой идее, пере и недообучение относятся к недостаткам, от которых может пострадать предсказательная способность модели. «Насколько плохи» ее прогнозы – это степень близости ее к пере или недообучению.

Модель, которая хорошо обобщает, не является ни переобученной, ни недообученной.

Возможно, это пока не имеет большого смысла, но мне нужно, чтобы Вы запомнили это предложение на протяжении всей статьи, так как это общая картина темы.

Пример.

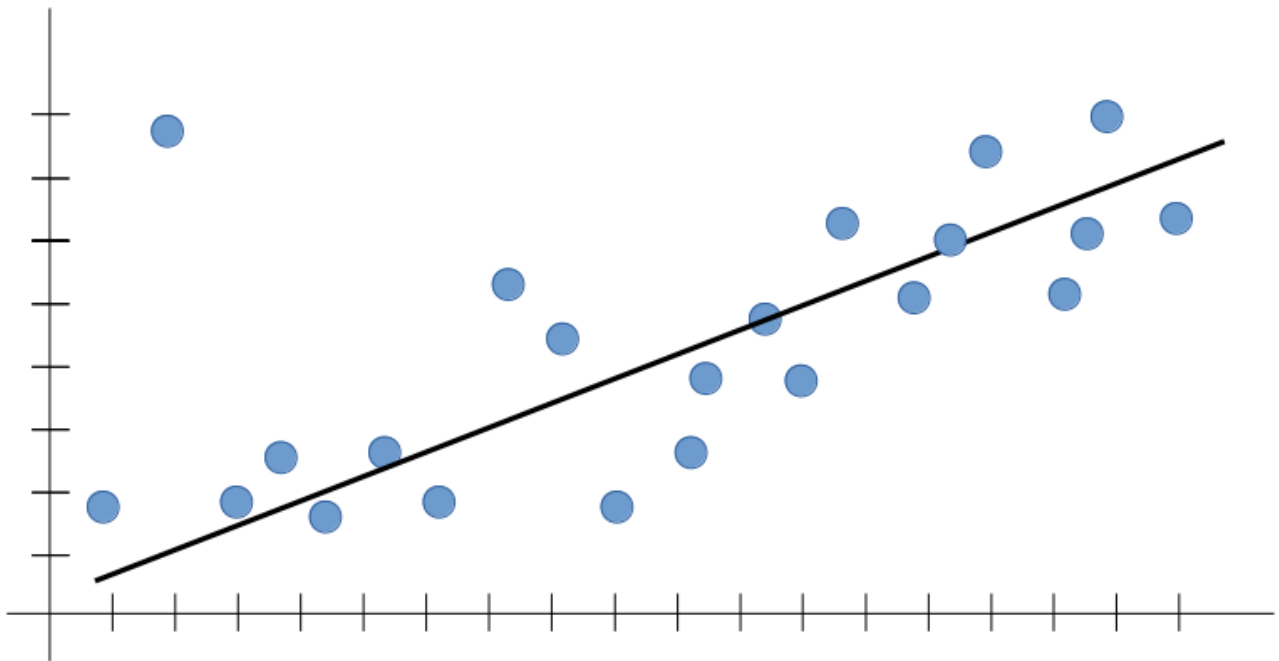
Допустим, мы пытаемся построить модель машинного обучения для следующего набора данных:



Ось X – это Предикторы (Predictor Variable) – например, площадь дома, а ось Y – Целевая переменная (Target Variable) – стоимость дома. Если у Вас есть опыт обучения модели, Вы, вероятно, знаете, что есть несколько Алгоритмов (Algorithm), однако для простоты в нашем примере выберем одномерную Линейную регрессию (Linear Regression).

Этап обучения

Обучение модели линейной регрессии в нашем примере сводится к минимизации общего расстояния (т.е. стоимости) между линией, которую мы пытаемся проложить, и фактическими точками Наблюдений (Observation). Проходит несколько итераций, пока мы не найдем оптимальную конфигурацию нашей линии. Это именно то место, где происходит пере и недообучение. Мы хотим, чтобы наша модель следовала примерно такой линии:

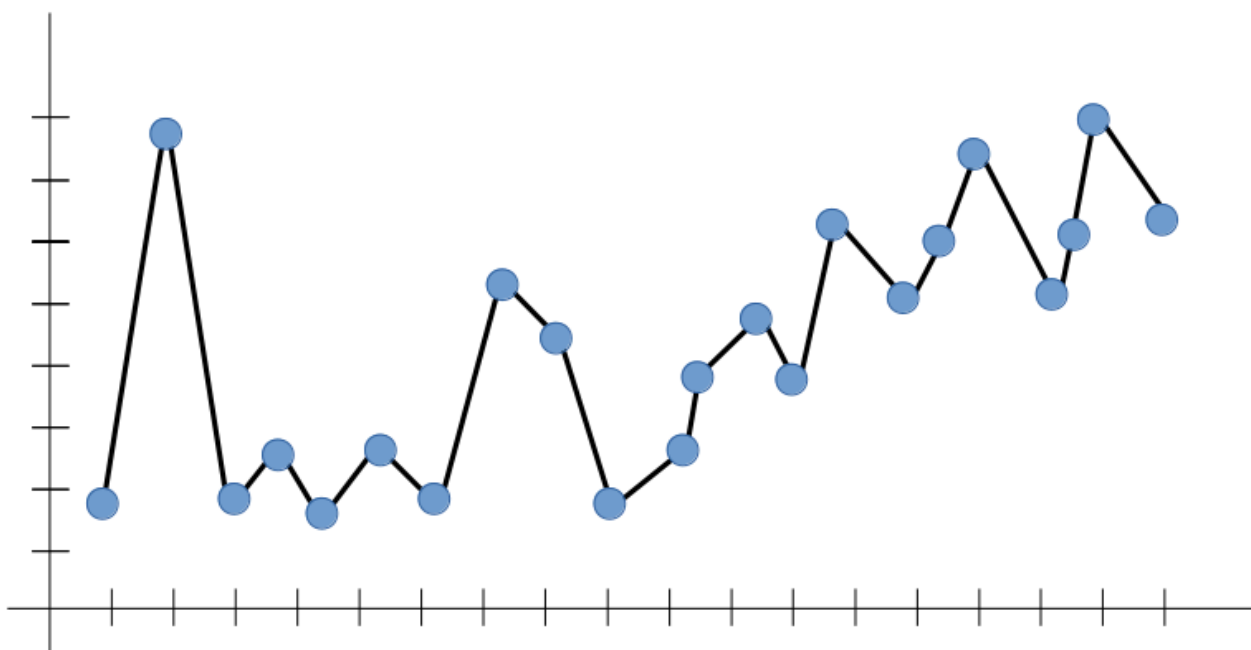


Несмотря на то, что общие потери не являются минимальными (т.е. существует лучшая конфигурация, в которой линия может давать меньшее расстояние до точек данных), линия выше очень хорошо вписывается в тенденцию, что делает модель надежной. Допустим, мы хотим знать значение Y при неизвестном доселе модели значении X (т. е. обобщить). Линия, изображенная на графике выше, может дать очень точный прогноз для нового X , поскольку с точки зрения машинного обучения ожидается, что результаты будут следовать тенденции, наблюдаемой в обучающем наборе.

Переобучение.

Когда мы запускаем обучение нашего алгоритма на Датасете (Dataset), мы стремимся уменьшить потери (т.е. расстояния от каждой точки до линии) с увеличением количества итераций. Длительное выполнение этого обучающего алгоритма приводит к минимальным общим затратам. Однако это означает, что линия будет вписываться во все точки, включая Шум (Noise), улавливая вторичные закономерности, которые не требуются модели.

Возвращаясь к нашему примеру, если мы оставим алгоритм обучения запущенным на долгое время, он, в конце концов, подгонит строку следующим образом:



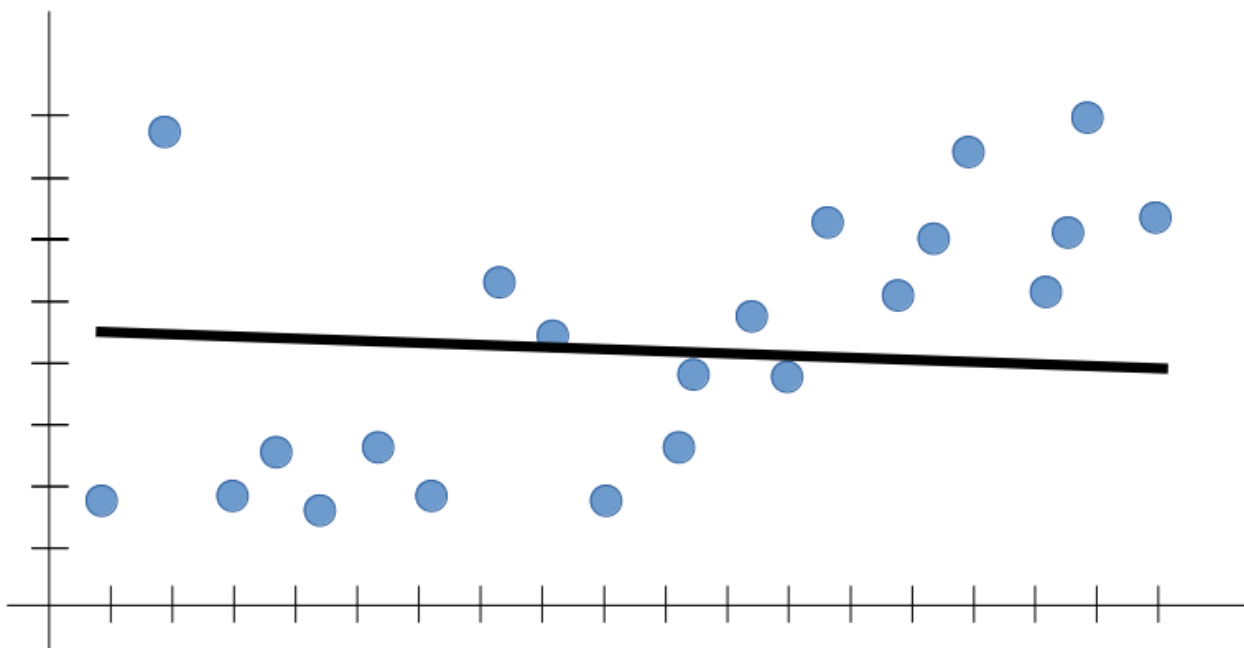
Выглядит хорошо, правда? Да, но насколько это надежно? Не совсем.

Суть такого алгоритма, как линейная регрессия, состоит в том, чтобы захватить доминирующий тренд и подогнать нашу линию к нему. На рисунке выше алгоритм уловил все тенденции, но не доминирующую. Если мы хотим протестировать модель на входных данных, которые выходят за пределы имеющихся у нас строк (т.е. обобщить), как бы эта линия выглядела? На самом деле нет возможности сказать. Следовательно, результаты ненадежны. Если модель не улавливает доминирующую тенденцию, которую мы все видим (в нашем случае с положительным увеличением), она не может предсказать вероятный результат для входных данных, которых никогда раньше не видела, что противоречит цели машинного обучения с самого начала!

Переобучение – это случай, когда общие потери-затраты действительно невелики, но обобщение модели ненадежно. Это связано с тем, что модель «слишком много учится» на обучающем наборе. Это может показаться абсурдным, но переобучение, или высокая Дисперсия (Variance), приводит к большему количеству плохих, чем хороших результатов. Какая польза от модели, которая очень хорошо усвоила данные обучения, но все еще не может делать надежные прогнозы для новых входных данных?

Недообучение.

Мы хотим, чтобы модель совершенствовалась на обучающих данных, но не хотим, чтобы она училась слишком многому (то есть слишком много паттернов). Одним из решений может быть досрочное прекращение тренировки. Однако это может привести к тому, что модель не сможет найти достаточно шаблонов в обучающих данных и, возможно, даже не сможет уловить доминирующую тенденцию. Этот случай называется недообучением:



Это случай, когда модель «недостаточно усвоила» обучающие данные, что приводит к низкому уровню обобщения и ненадежным прогнозам.

Как вы, вероятно, и ожидали, такое большое Смещение (Bias) так же плохо для модели, как и переобучение. При большом смещении модель может не обладать достаточной гибкостью с точки зрения подгонки линии, что приводит к чрезмерной упрощенности.

Дилемма смещения-дисперсии.

Итак, какова правильная мера? Этот компромисс является наиболее важным аспектом обучения модели машинного обучения. Как мы уже говорили, модели выполняют свою задачу, если хорошо обобщают, а это связано с двумя нежелательными исходами – большим смещением и высокой дисперсией. Это и есть Дилемма смещения-дисперсии. Ответственность за определение того, страдает ли модель от одного из них, полностью лежит на разработчике модели.

Выводы

Линейная регрессия — базовый, фундаментальный линейный алгоритм, который берет свое начало из статистики. Алгоритм отлично интерпретируется — это значит что он понятный, и легко проверяемый. Легкость и простота это еще и главный минус линейной регрессии, потому что в реальной жизни редко можно заметить линейную зависимость (например цены на ноутбук зависят от его мощности, а цены на автомобиль от количества лошадиных сил). Как правило, наша жизнь полна как раз не линейными алгоритмами, которые зависят от каждого сделанного шага.

С линейной регрессией связано много способов оптимизации. Так как мы рассматриваем с вами численные данные, взятые из таблиц, часто такие данные неполные. Чуть позже мы будем говорить про оптимизацию моделей. Но это не единственная сложность работы с линейной регрессией — часто ей нужна регуляризация — дополнительные ограничения, связанные с разными свойствами алгоритмов. Тем не менее это интересный алгоритм, и я вас поздравляю с первой практикой в машинном обучении!

Дополнительный материалы

За рамками видео осталась очень важная тема, которая касается линейной регрессии. Хотя это и простой для понимания алгоритм, когда вы будете им пользоваться и обучать модель, вы можете встретиться с трудностями. Давайте разберемся с какими.

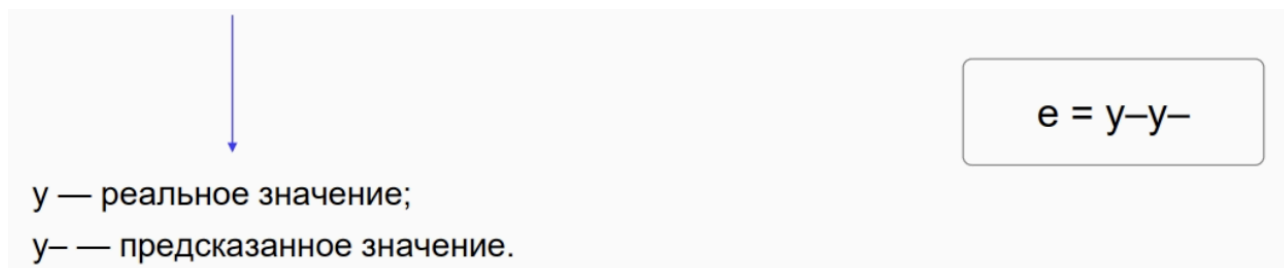
Зачастую, чем проще алгоритм, тем больше, скажем так, “настроек” он имеет. Это применимо к линейной регрессии. Это простой алгоритм, который даже не совсем относится к машинному обучению, а скорее к статистике. В силу того, что этот алгоритм легко интерпретируем, то есть его результаты легко прочесть и понять, он не так хорошо описывает реальное положение дел. Например, стоимость квартиры действительно зависит от количества комнат и удаленности от метро. Но сколько еще факторов влияют на стоимость? Скажем честно — много. И если все эти характеристики мы добавим в модель линейной регрессии, скорее всего без дополнительных настроек у нас получится не самая качественная модель. И это если не учитывать переобучение.

Давайте по порядку разберемся, как мы можем усовершенствовать алгоритм линейной регрессии для его корректной работы с любыми линейными данными. Но для начала разберемся, как понять что наша модель получилась качественной, и насколько хорошо она справляется со своей задачей.

Анализ результатов линейной регрессии

Допустим, мы построили модель и получили какую то формулу. Как же нам понять, получилась ли модель качественно?

Нужно сравнивать предсказанное значение с реальными данными. То есть то, чтобы было перед обучением, и то предсказание, которое получилось.



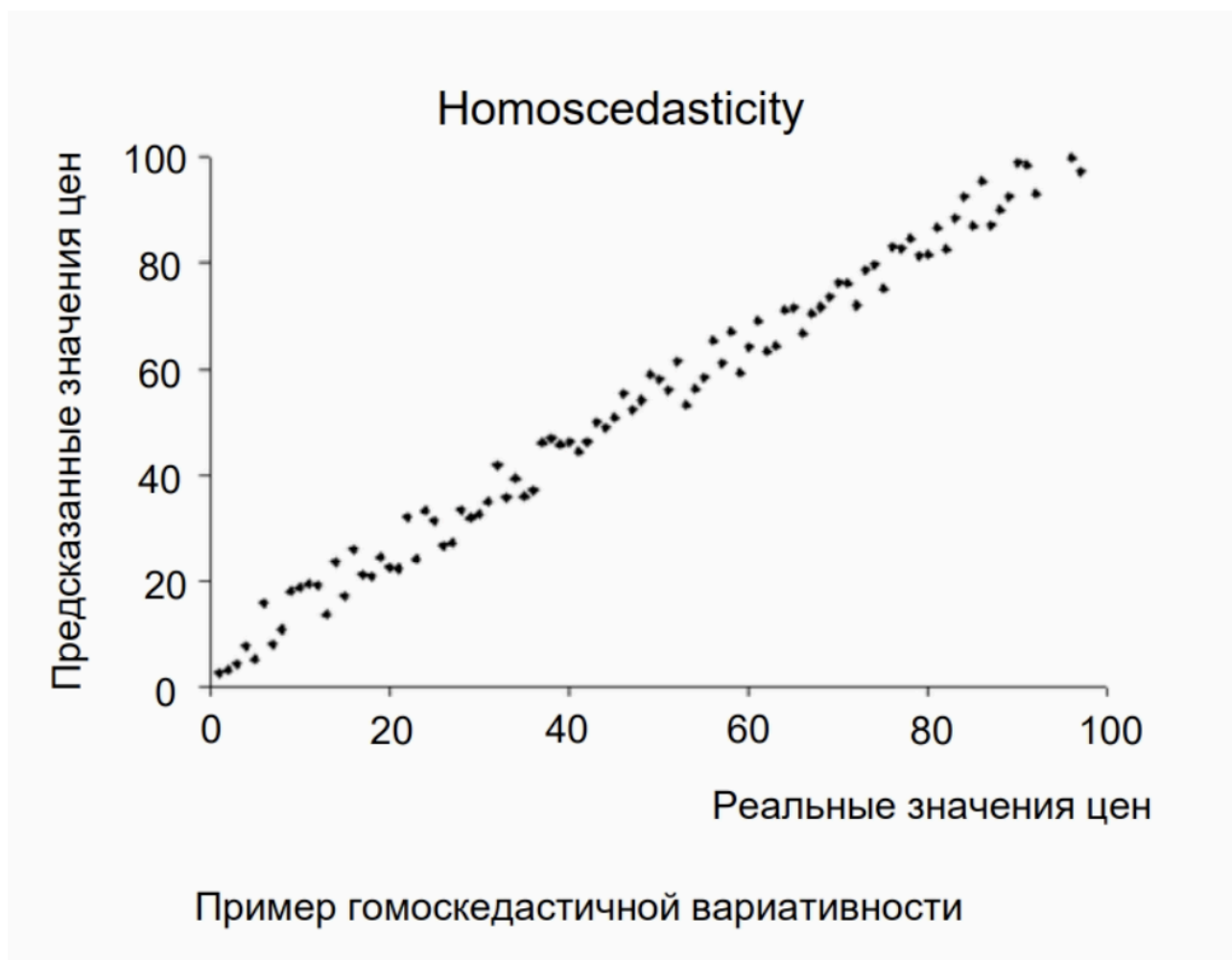
y — реальное значение;
 $y-$ — предсказанное значение.

$$e = y - y-$$

Разность между этими значениями будет называться остатками регрессии.

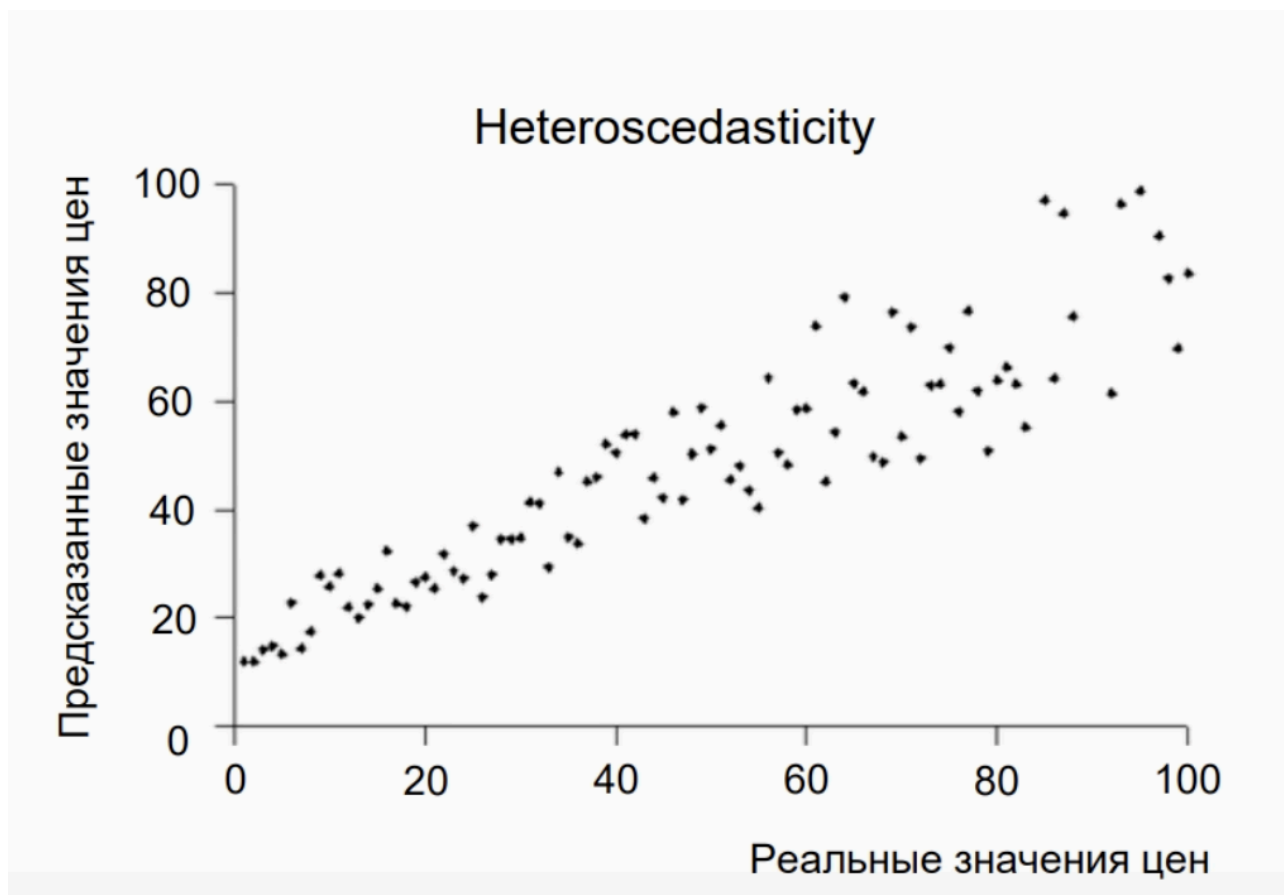
У остатков регрессии есть несколько особенностей, поэтому давайте введем несколько новых терминов. Чтобы построенная модель была корректной, необходимо чтобы остатки регрессии были гомоскедастичны. **Гомоскедастичность** — свойство, обозначающее постоянство дисперсии некоторой последовательности случайных величин.

Напомню, что дисперсия — отклонение точек относительно прямой. В нашем случае это значит, что дисперсия остатков регрессии должна быть однородной, стабильной для всех наблюдаемых объектов и во все моменты измерения.



Например, на графике отражены реальные значения цен на автомобили, и предсказанные. Обратите внимание, что с увеличением реальной цены на автомобиль, растет и предсказанная цена, при этом ошибки имеют одинаковую дисперсию.

Если бы ошибки имели бы разнородную дисперсию, например при повышении цен на автомобиль, ошибка бы тоже росла, то у графика был бы следующий вид:



То есть дисперсия ошибки — вариативность — растет с увеличением цены. Такое явление противоположно гомоскедастичности, и носит название гетероскедастичность.

Гомоскедастичность — свойство, обозначающее постоянство дисперсии некоторой последовательности случайных величин.

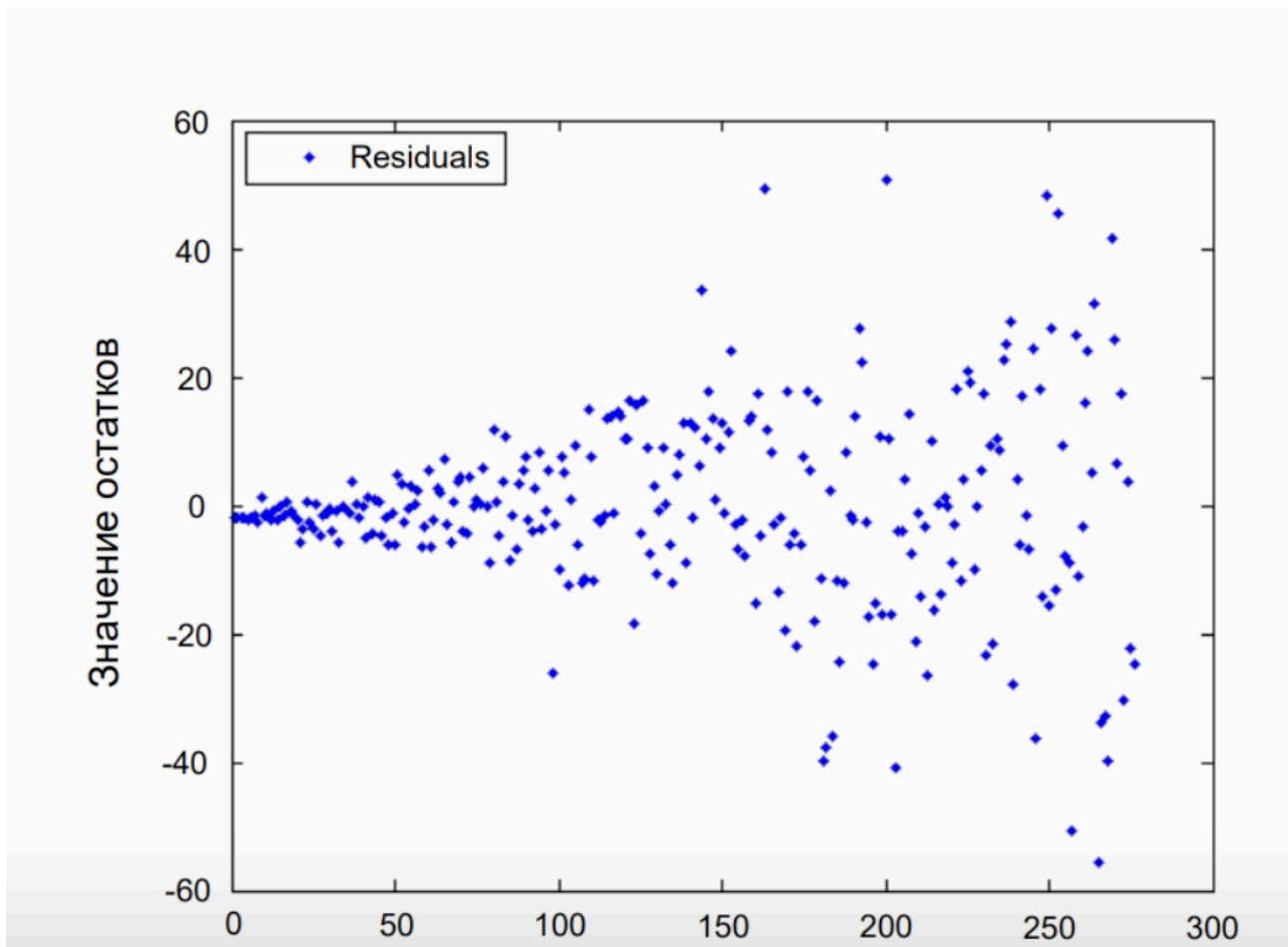
Гетероскедастичность — наоборот, неоднородность наблюдений, выражающуюся в неодинаковой, непостоянной дисперсии случайной ошибки регрессионной модели.

То есть:

Если выбранная регрессионная модель хорошо описывает истинную зависимость, то остатки должны быть независимыми нормально распределёнными случайными величинами с нулевым средним и в их значениях должен отсутствовать тренд.

То есть график остатков позволяет определить, насколько адекватной модель получилась.

Отображение остатков регрессии и зрительная оценка позволяют оценить модель, посмотреть, нормально ли распределены остатки регрессии, и не получилась ли у нас гетероскедастичность.



Что можно почитать еще?

1. <https://gb.ru/blog/mashinoe-obuchenie/> - введение в машинное обучение
2. <https://habr.com/ru/companies/ods/articles/322076/> - введение
3. <https://www.youtube.com/playlist?list=PLk4h7dmY2eYHHTyfLyrl7HmP-H3mMAW08> - курс лекций по машинному обучению и математике в машинном обучении

Используемая литература

1. Бринк Х., Ричардс Дж., Феверолф М. - Машинное обучение, 2017 год
2. Андрей Бурков — Машинное обучение без лишних слов, 2020 год
3. Андреас Мюллер, Сара Гвидо — Введение в машинное обучение с помощью Python, 2017 год
4. Джереми Уатт, Реза Борхани, Ангелос Катсаггелос - Машинное обучение: основы, алгоритмы и практика применения