Регрессия

Наумов Д.А., доц. каф. КТ

Экспертные системы и искусственный интеллект, 2020

Содержание лекции

1. Машинное обечение и библиотека Scikit-Learning
2. Визуализация данных
3. Линейная регрессия
4. Комбинация базисных функций
5. Разложение ошибки на смещение и разброс
6. Переобучение. Недообучение
7. Линейный классификатор и логистическая регрессия
8. Регуляризация

Машинное обучение

Машинное обучение, Mashine Learning, ML

часть сферы искусственного интеллекта, средство построения математических моделей для исследования данных.

Задачи ML:

- этап обучения - определение настраиваемых параметров модели, которые нужно "приспособить"для отражения наблюдаемых данных;
- использование модели для предсказания и понимания различных аспектов данных новых наблюдений.

Категории машинного обучения

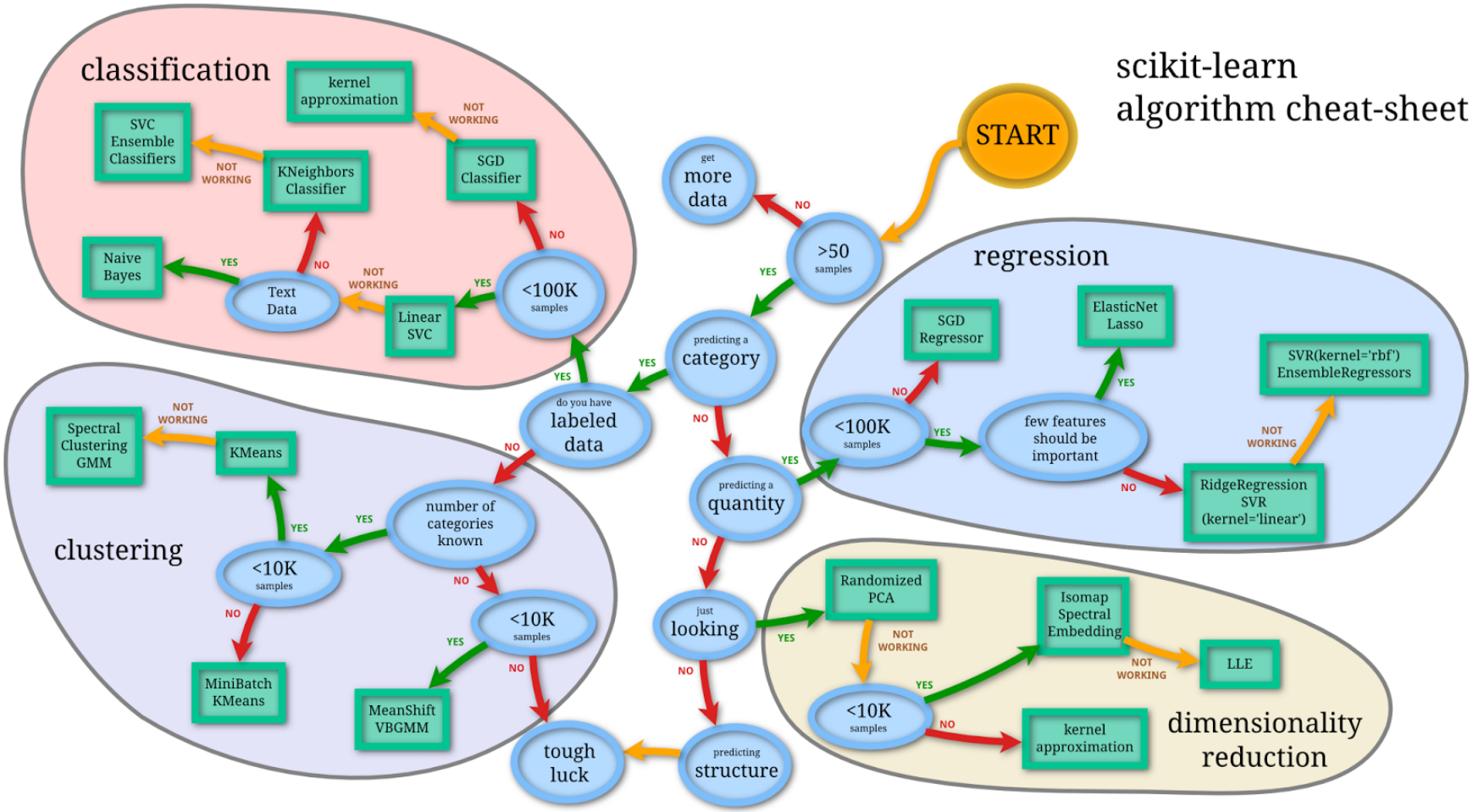
- обучение с учителем (supervised learning) - классификация, регрессия;
- обучение без учителя (supervised learning) - кластеризация, понижение размерности.

Библиотека Scikit-Learn

Библиотека Scikit-Learn:

- пакет, предоставляющий эффективные версии множества распространенных алгоритмов;
- единообразный API, позволяющий легко перейти от одной модели к другой.
  - представление данных (data representation);
  - применение API статистического оценивания (API Estimator).

Библиотека Scikit-Learn



Основное представление - таблица

двумерная сетка данных, в которой строки представляют отдельные элементы набора данных, а столбцы — атрибуты, связанные с каждым из этих элементов.

```
In[1]: import seaborn as sns
       iris = sns.load_dataset('iris')
       iris.head()

Out[1]:    sepal_length  sepal_width  petal_length  petal_width species
        0           5.1          3.5           1.4          0.2  setosa
        1           4.9          3.0           1.4          0.2  setosa
        2           4.7          3.2           1.3          0.2  setosa
        3           4.6          3.1           1.5          0.2  setosa
        4           5.0          3.6           1.4          0.2  setosa
```

- строки - выборка (samples), соответствуют объектам;
- столбцы - признаки (features), соотвествуют свойствами объектов.

Принятые обозначения:

- X[n_samples, n_features] - матрица признаков (features matrix); варианты представления: массив NumPy, Pandas.DataFrame, разреженная матрица SciPy.
- y[n_samples] - целевой массив принзнаков; величину, значения которой мы хотим предсказать на основе имеющихся данных. Варианты представления: массив NumPy, Pandas.Series.

Seaborn

Seaborn

высокоуровневое API на базе библиотеки matplotlib, содержащее дефолтные настройки оформления графиков и сложные типы визуализации, которые в matplotlib потребовали бы большого количество кода.

Импортируем необходимые библиотеки и загрузим данные о продажах и оценках видео-игр из Kaggle Datasets.

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('code/regression/video_games_sales.csv')
df.info()
```

Data

Оставим только те записи, в которых нет пропусков, с помощью метода dropna.

```
df = df.dropna()
print(df.shape)
```

Посмотрим на несколько первых записей с помощью метода head.

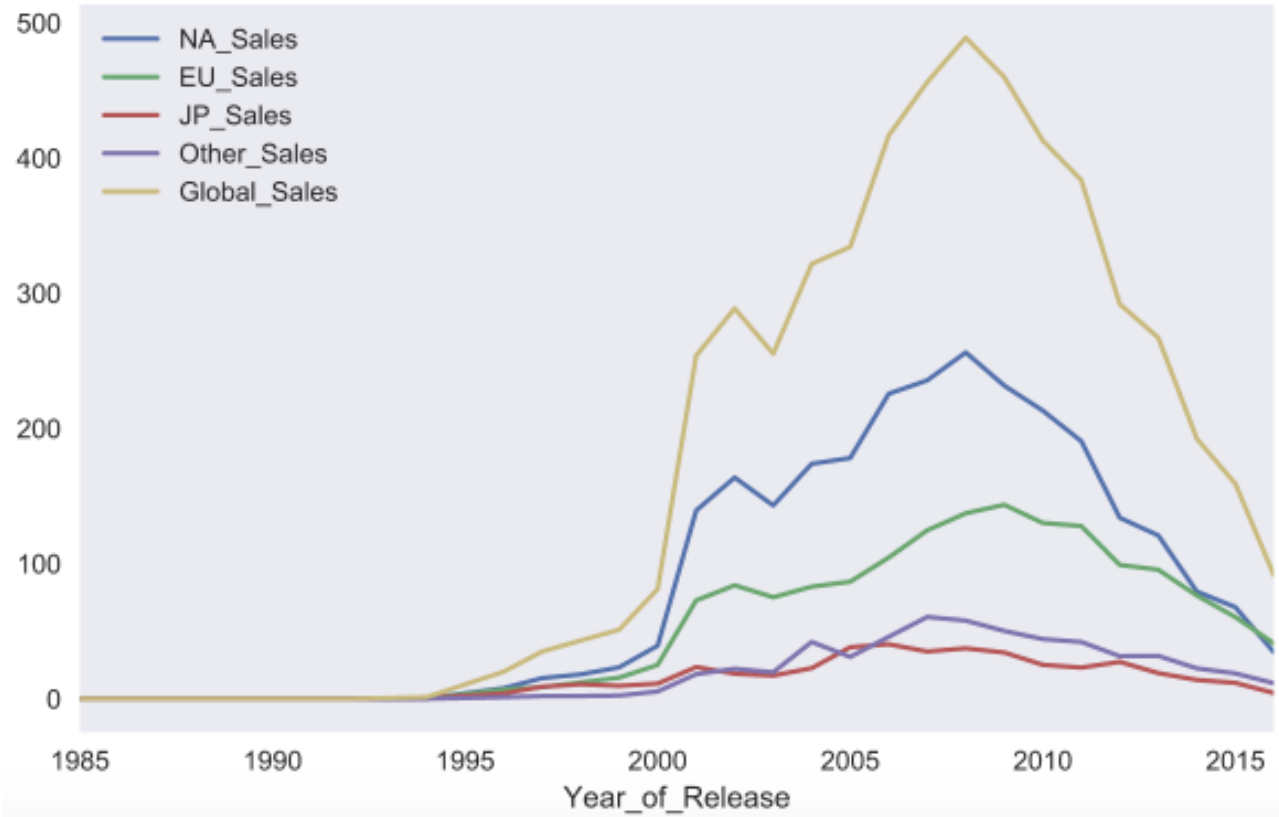
```
useful_cols = ['Name', 'Platform', 'Year_of_Release', 'Genre',
               'Global_Sales', 'Critic_Score', 'Critic_Count',
               'User_Score', 'User_Count', 'Rating'
              ]
df[useful_cols].head()
```

	Name	Platform	Year_of_Release	Genre	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Rating
0	Wii Sports	Wii	2006	Sports	82.53	76.0	51.0	8.0	322.0	E
2	Mario Kart Wii	Wii	2008	Racing	35.52	82.0	73.0	8.3	709.0	E
3	Wii Sports Resort	Wii	2009	Sports	32.77	80.0	73.0	8.0	192.0	E
6	New Super Mario Bros.	DS	2006	Platform	29.80	89.0	65.0	8.5	431.0	E
7	Wii Play	Wii	2006	Misc	28.92	58.0	41.0	6.6	129.0	E

Plot

Воспользуемся функцией plot - построим график продаж видео игр в различных странах в зависимости от года.

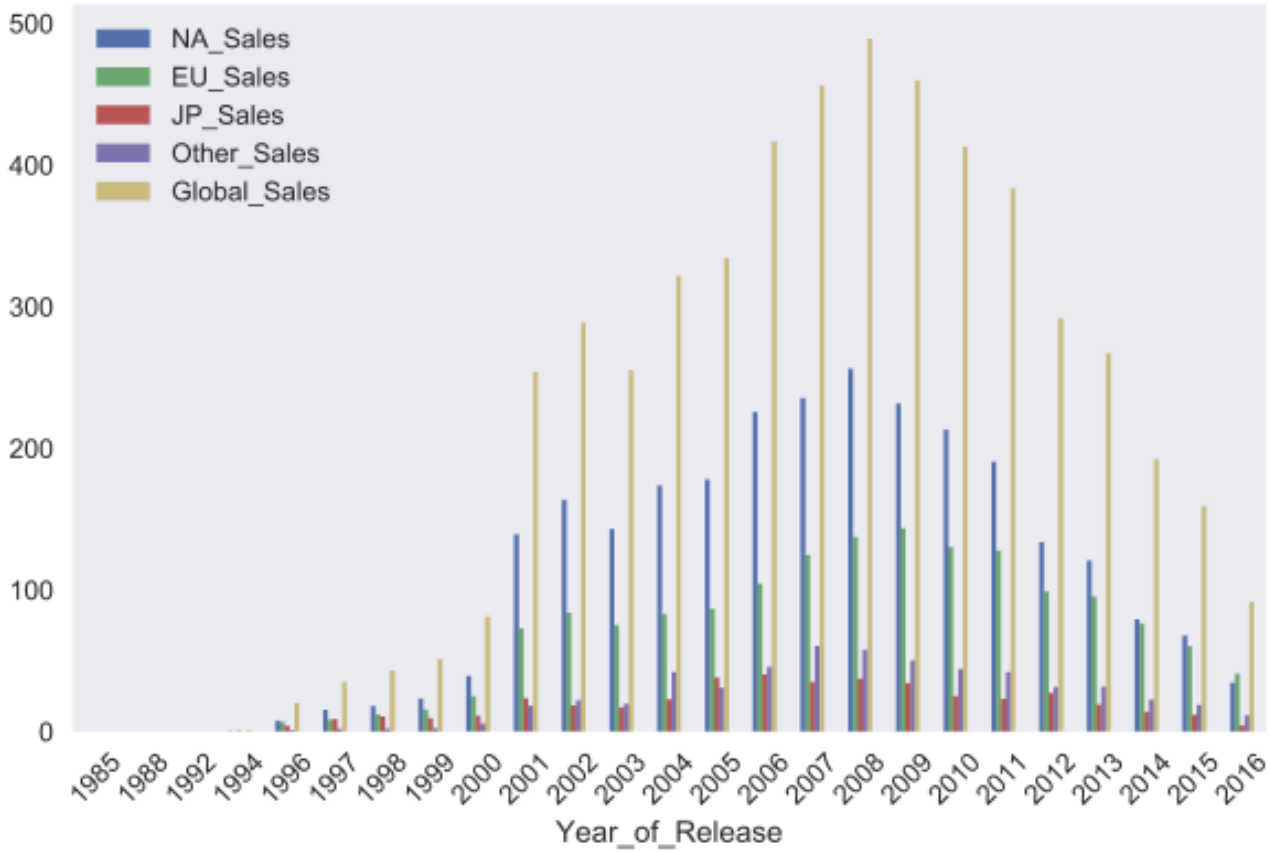
```
sales_df = df[[x for x in df.columns if 'Sales' in x] + ['Year_
sales_df.groupby('Year_of_Release').sum().plot()
```



Plot

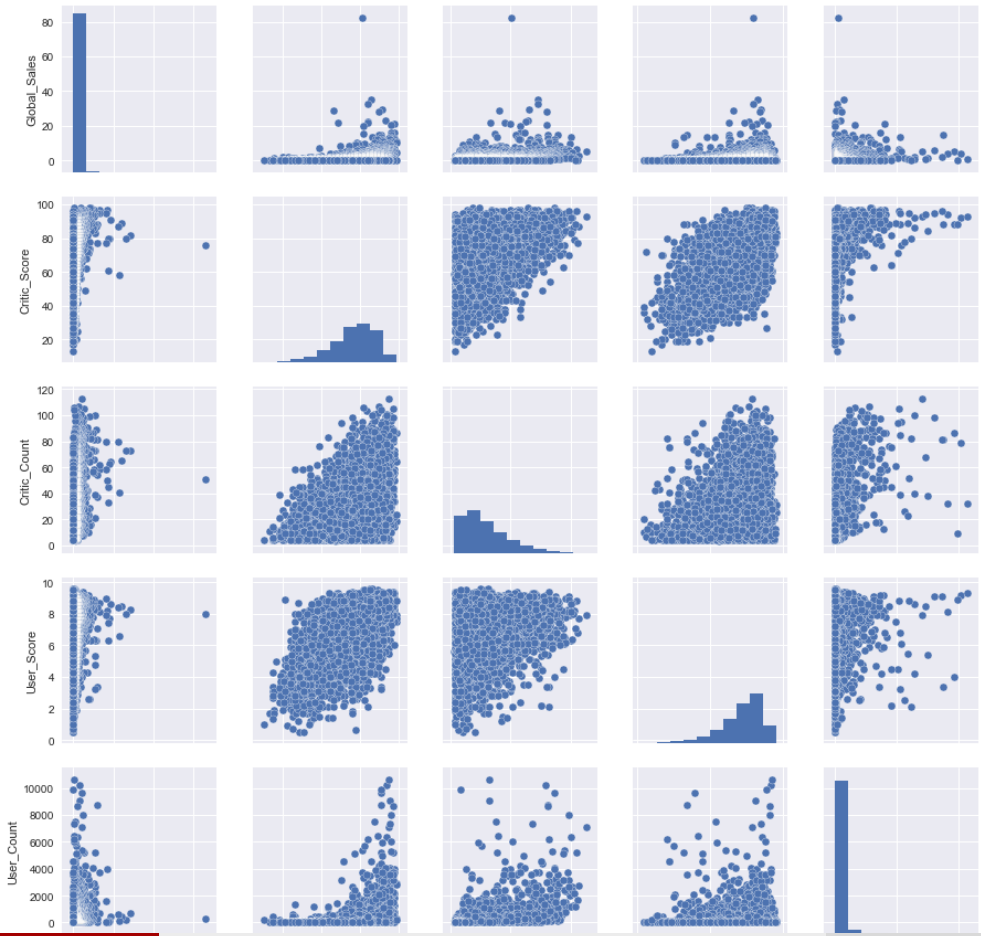
С помощью параметра kind можно изменить тип графика, например, на bar chart.

```
sales_df.groupby('Year_of_Release').sum().plot(kind='bar', rot=
```



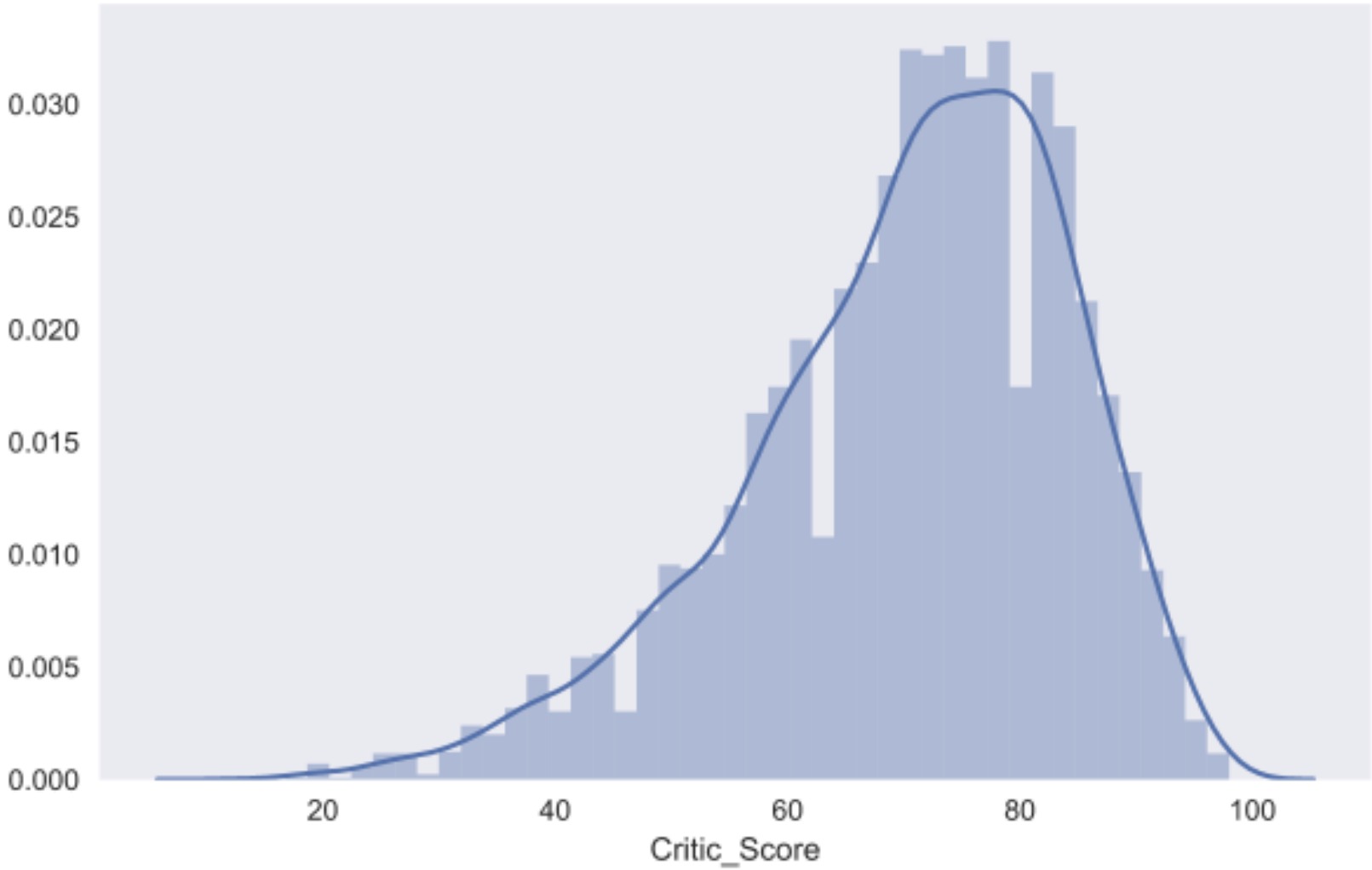
Графики типа pair plot (scatter plot matrix) позволяет посмотреть на одной картинке, как связаны между собой различные признаки.

```
cols = ['Global_Sales', 'Critic_Score', 'Critic_Count', 'User_S
sns_plot = sns.pairplot(df[cols])
sns_plot.savefig('pairplot.png')
```



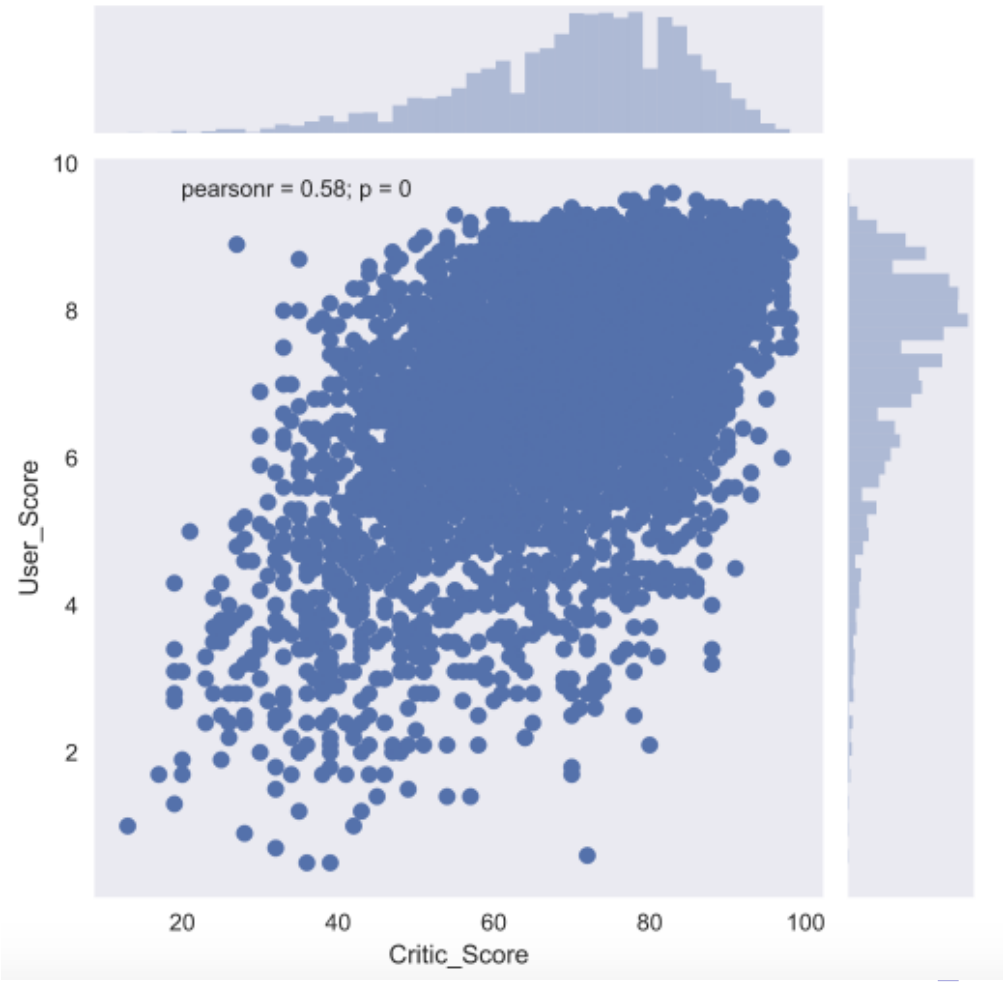
С помощью seaborn можно построить и распределение dist plot. Для примера посмотрим на распределение оценок критиков Critic_Score.

```
sns.distplot(df.Critic_Score)
```



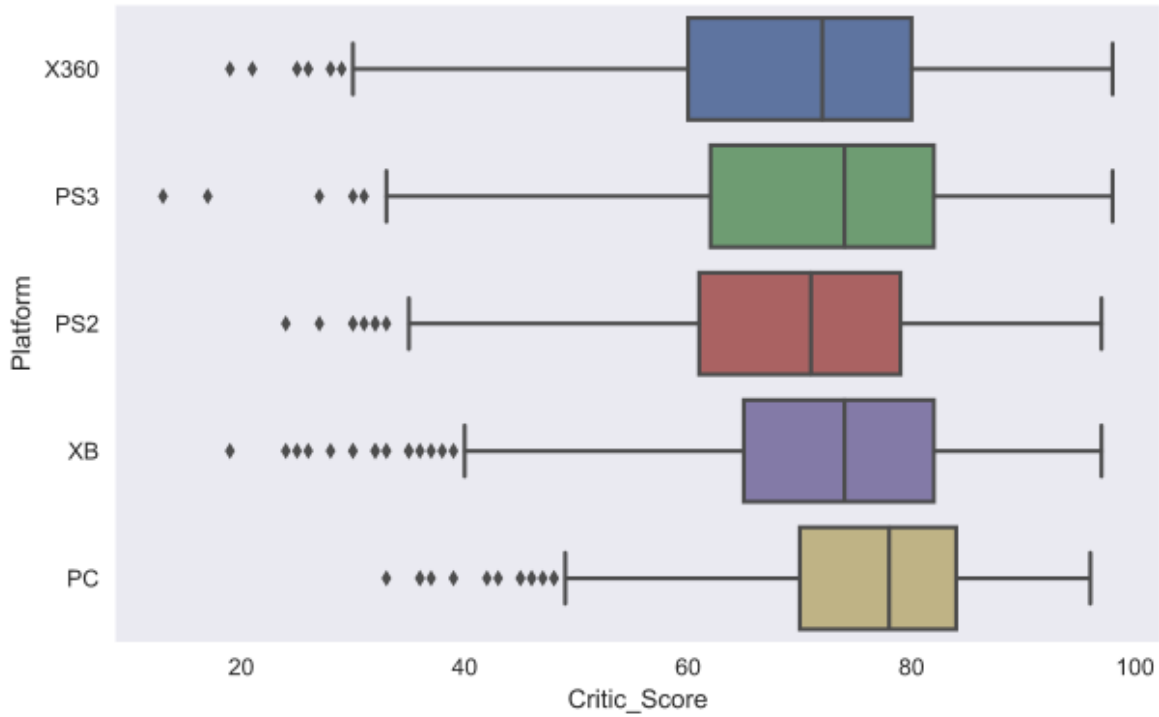
Для того, чтобы подробнее посмотреть на взаимосвязь двух численных признаков, есть joint plot — гибрид scatter plot и histogram.

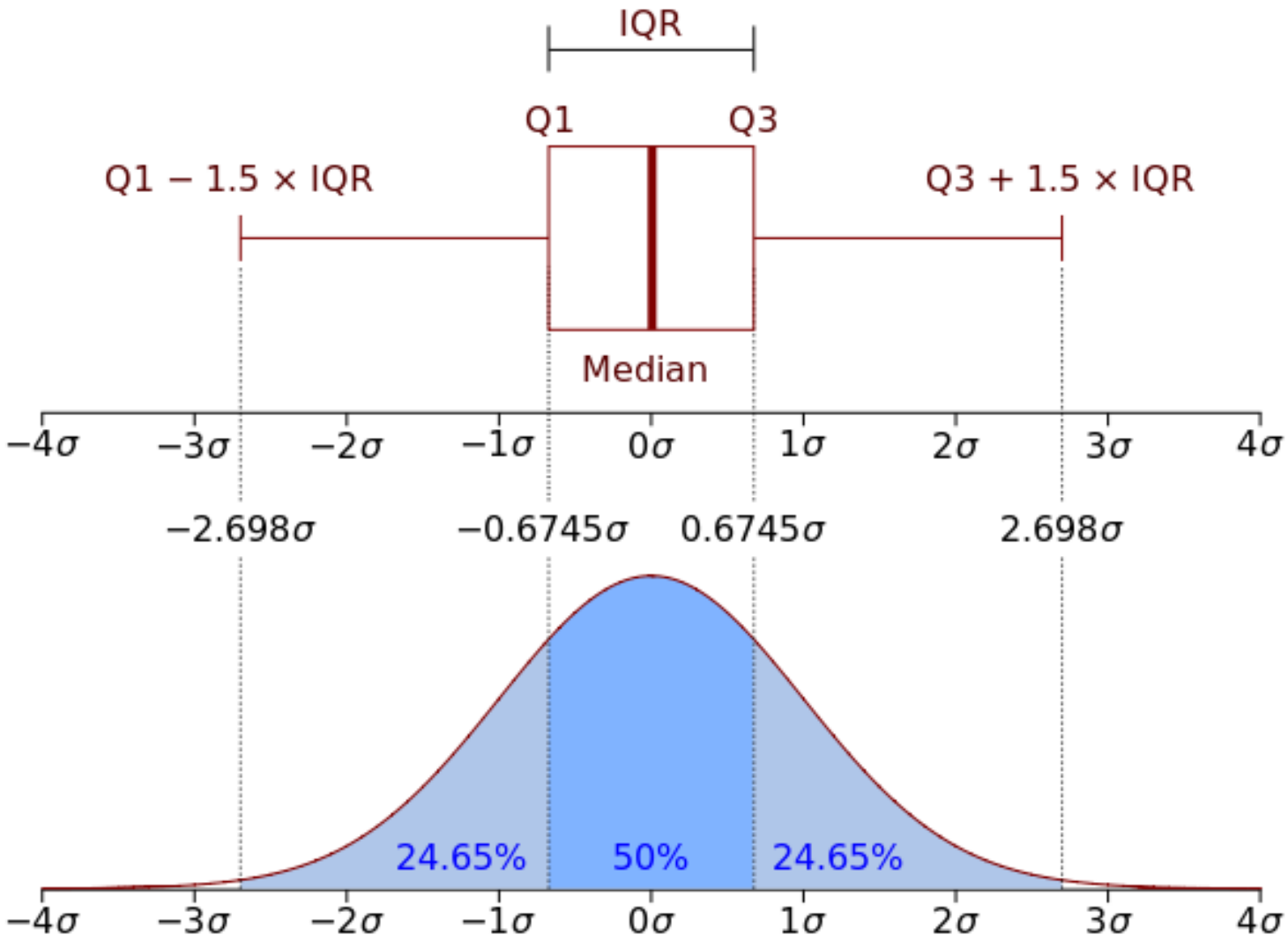
```
cols = ['Critic_Score', 'User_Score']
sns.jointplot(df[cols])
```



Сравним оценки игр от критиков для топ-5 крупнейших игровых платформ при помощи графиков box_plot.

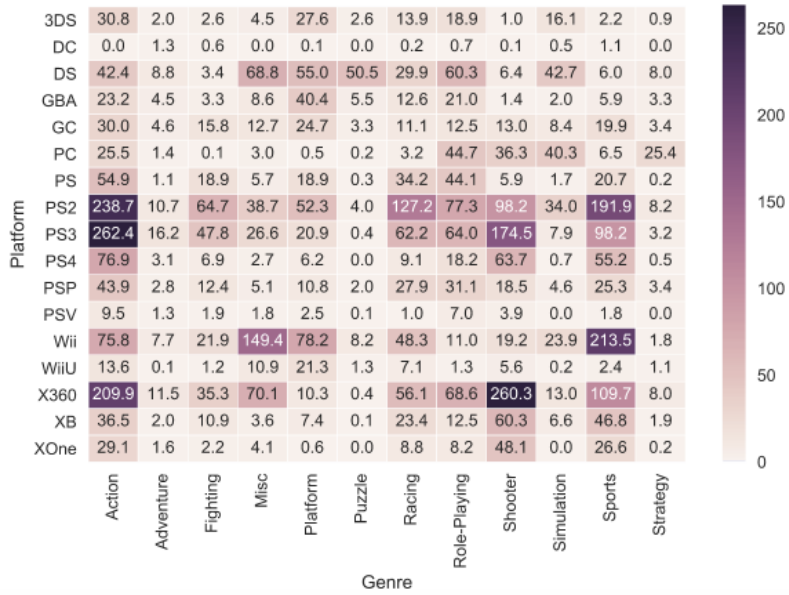
```
top_platforms = df.Platform.value_counts().sort_values(
    ascending = False).head(5).index.values
sns.boxplot(y="Platform", x="Critic_Score",
    data=df[df.Platform.isin(top_platforms)], orient="h")
```




IQR
Q1
Q3
Q1 − 1.5 × IQR
Q3 + 1.5 × IQR
Median
−4σ
−3σ
−2σ
−1σ
0σ
1σ
2σ
3σ
4σ
−2.698σ
−0.6745σ
0.6745σ
2.698σ
24.65%
50%
24.65%

Heat map позволяет посмотреть на распределение какого-то численного признака по двум категориальным. Визуализируем суммарные продажи игр по жанрам и игровым платформам.

```
platform_genre_sales = df.pivot_table(
                          index='Platform',
                          columns='Genre',
                          values='Global_Sales',
                          aggfunc=sum).fillna(0).applymap(float)
sns.heatmap(platform_genre_sales, annot=True, fmt=".1f", linew
```



API Scikit-Learn

Использование API статистического оценивания:

1. Выбор класса модели с помощью импорта соответствующего класса оценивателя из библиотеки Scikit-Learn.
2. Выбор гиперпараметров модели путем создания экземпляра этого класса с соответствующими значениями.
3. Компоновка данных в матрицу признаков и целевой вектор в соответствии с описанным выше.
4. Обучение модели на своих данных посредством вызова метода fit() экземпляра модели.
5. Применение модели к новым данным (в случае машинного обучения с учителем метки для неизвестных данных обычно предсказывают с помощью метода predict()).

Линейная регрессия

модель зависимости объясняемой переменной y от объясняющих ее факторов, причем функция зависимости является линейной:

$$y = w_0 + \sum_{i=0}^{n} w_i x_i$$

Зададим модель следующим образом:

$$\vec{y} = X\vec{w} + \epsilon$$

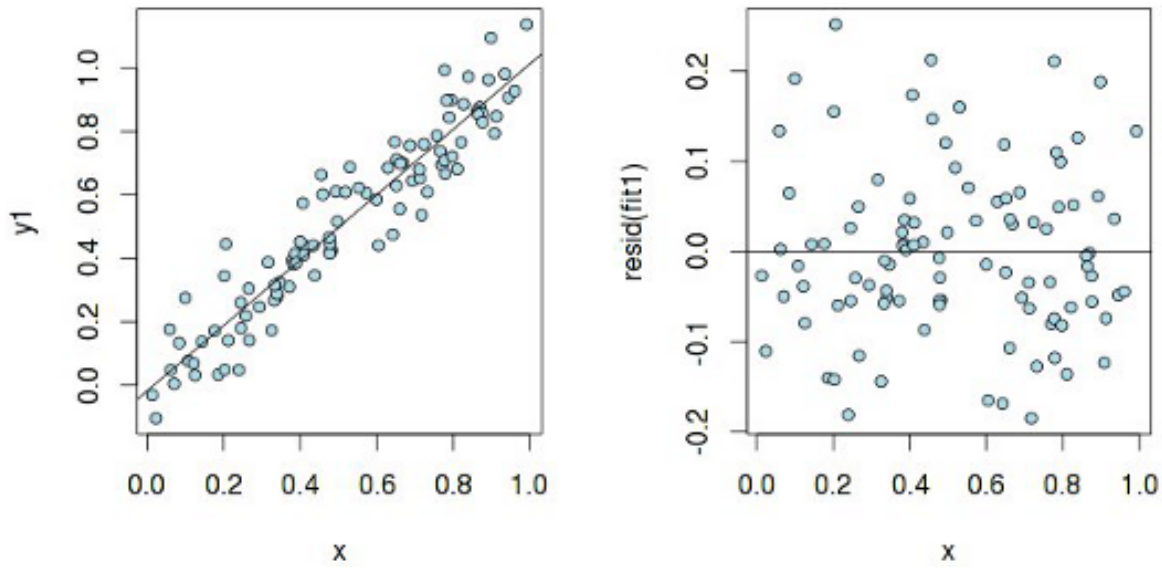
- $\vec{y} \in R^n$ - объясняемая (или целевая) переменная;
- w - вектор параметров модели (веса модели);
- X - матрица наблюдений и признаков размерности n строк на $m + 1$ столбцов (включая фиктивную единичную колонку слева);
- ϵ - случайная переменная, соответствующая случайной, непрогнозируемой ошибке модели.

Линейная регрессия

Ограничения на модель:

- матожидание случайных ошибок равно нулю;
- дисперсия случайных ошибок одинакова и конечна (гомоскедастичность);
- случайные ошибки не скоррелированы.

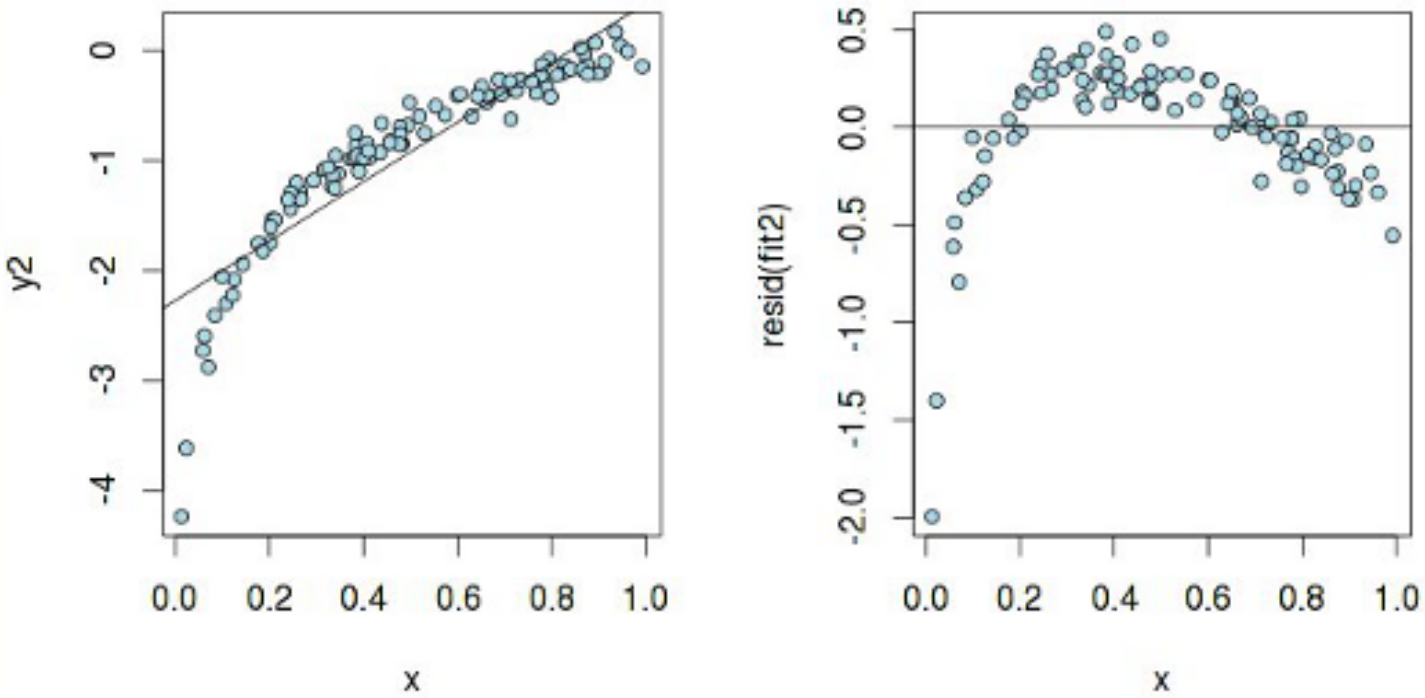
Пример графика остатков в случае простой линейной зависимости:



Остатки равномерно распределены относительно горизонтальной оси.

Линейная регрессия

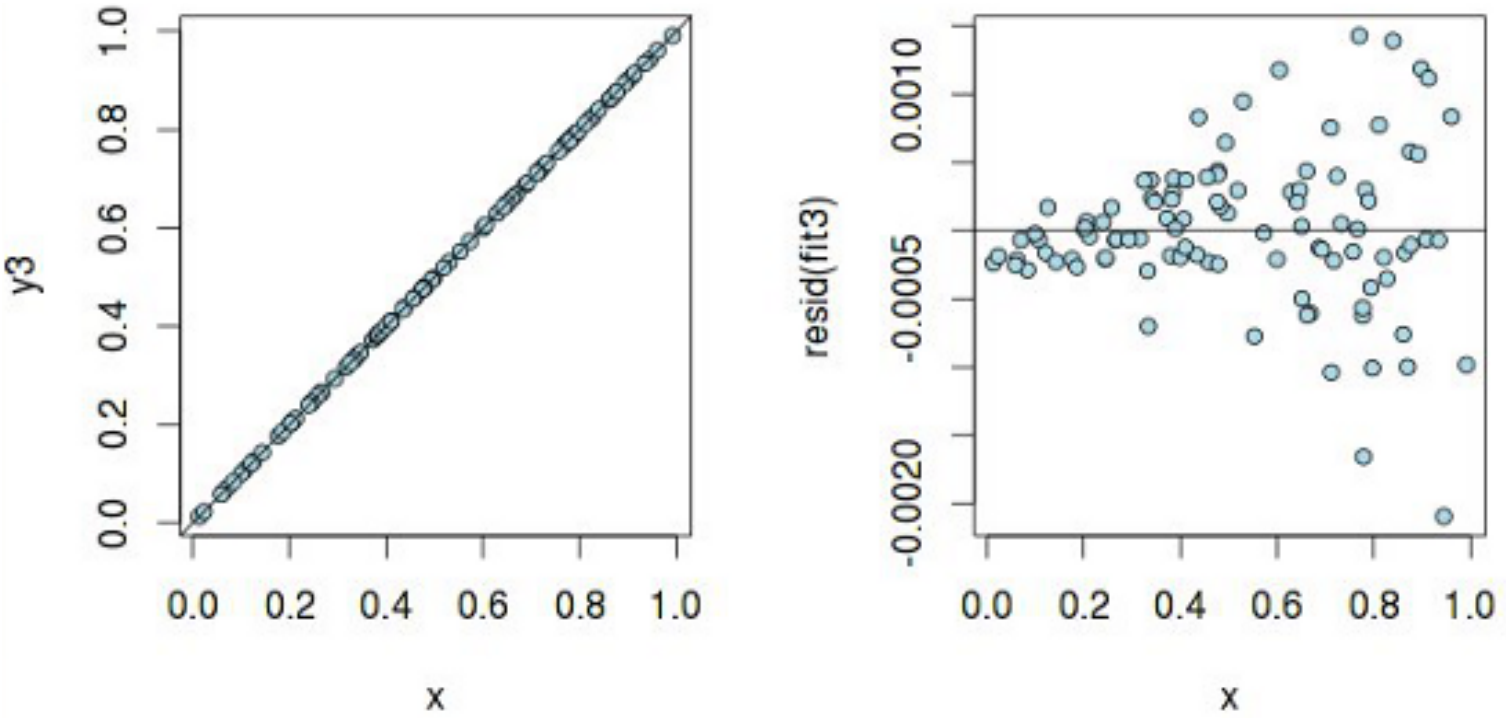
Исследуем график, но построенный для линейной модели (для данных, которые на самом деле не является линейными):



По графику $y(x)$ бы можно предположить линейную зависимость, но у остатков есть паттерн, а значит, чистая линейная регрессия тут не пройдет.

Линейная регрессия

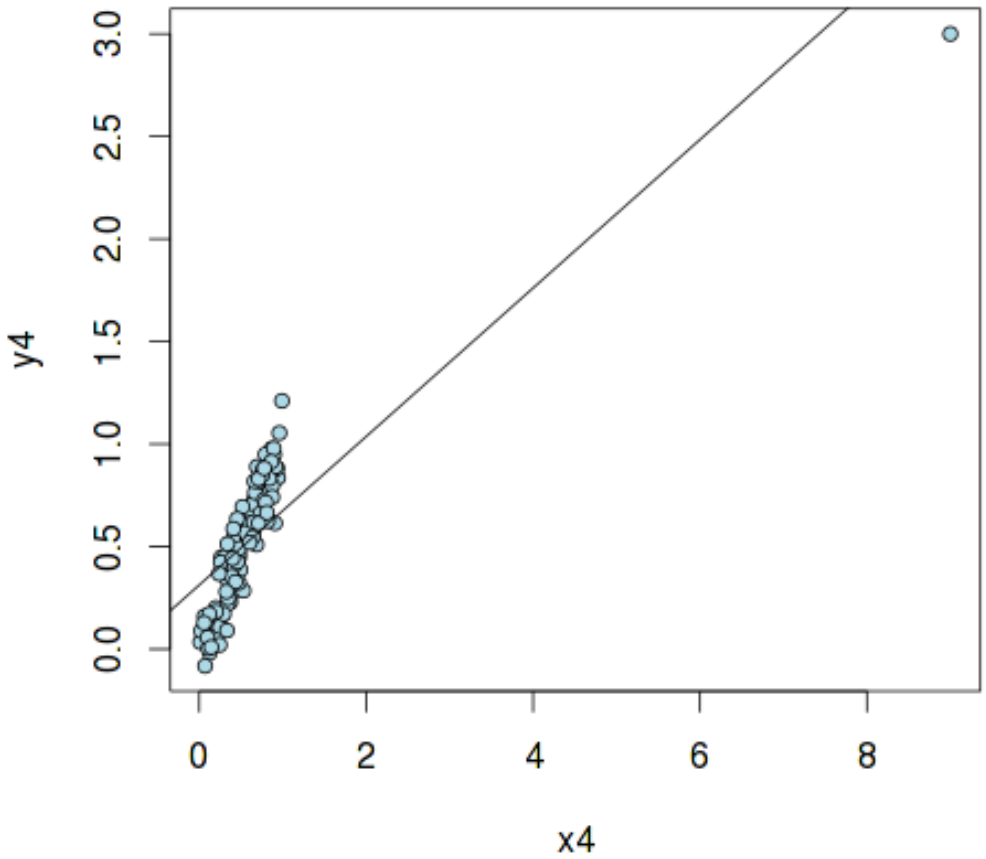
Пример не гетероскедастичности:



Линейная модель с такими "раздувающимися"остатками не будет корректна.

Линейная регрессия

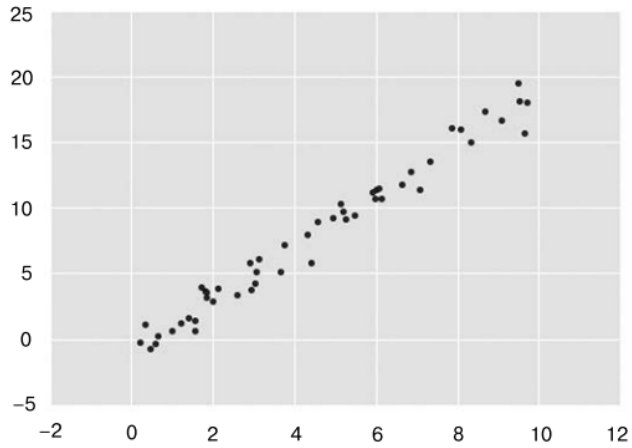
Пример выброса (резко выделяющегося значения):



Выброс который может сильно исказить результаты и привести к ошибочным выводам.

Пример: линейная регрессия

Построим простую линейную регрессию - часто встречающийся случай подбора аппроксимирующей прямой для данных вида (x, y):




```
import matplotlib.pyplot as plt
import numpy as np
rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)
plt.scatter(x, y)
```

Пример: линейная регрессия

1. Выбор класса модели

Каждый класс модели в Scikit-Learn представлен соответствующим классом языка Python.

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

2. Выбор гиперпараметров модели

- Хотим ли мы выполнить подбор сдвига прямой?
- Хотим ли мы нормализовать модель?
- Хотим ли мы сделать модель более гибкой, выполнив предварительную обработку признаков?
- Какая степень регуляризации должна быть у нашей модели?
- Сколько компонент модели мы хотели бы использовать?

Пример: линейная регрессия

2. Выбор гиперпараметров модели

Создадим экземпляр класса LinearRegression и укажем с помощью гиперпараметра fit_intercept, что нам бы хотелось выполнить подбор точки пересечения с осью координат:

```
model = LinearRegression(fit_intercept=True)
```

API Scikit-Learn разделяет выбор модели и применение модели к данным.

3. Формирование из данных матриц признаков и целевого вектора

Измененим форму одномерного массива X, чтобы привести ее к размерности [n_samples, n_features]:

```
X = x[:, np.newaxis]
X.shape
```

Пример: линейная регрессия

4. Обучение модели на данных

Применим модель к данным с помощью метода fit() модели:

```
model.fit(X, y)
```

Команда fit() вызывает выполнение множества вычислений, в зависимости от модели, и сохранение результатов этих вычислений в атрибутах модели, доступных для просмотра пользователем:

```
print('coef = ', model.coef_)
print('intercept = ', model.intercept_)
```

Эти два параметра представляют собой угловой коэффициент и точку пересечения с осью координат для простой линейной аппроксимации наших данных.

Пример: линейная регрессия

5. Предсказание новых значений

После обучения модели главная задача машинного обучения с учителем заключается в вычислении с ее помощью значений для новых данных, не являющихся частью обучающей последовательности:

```
xfit = np.linspace(-1, 11)
```

Как и ранее, эти x-значения требуется преобразовать в матрицу признаков [n_samples, n_features], после чего можно подать их на вход модели:

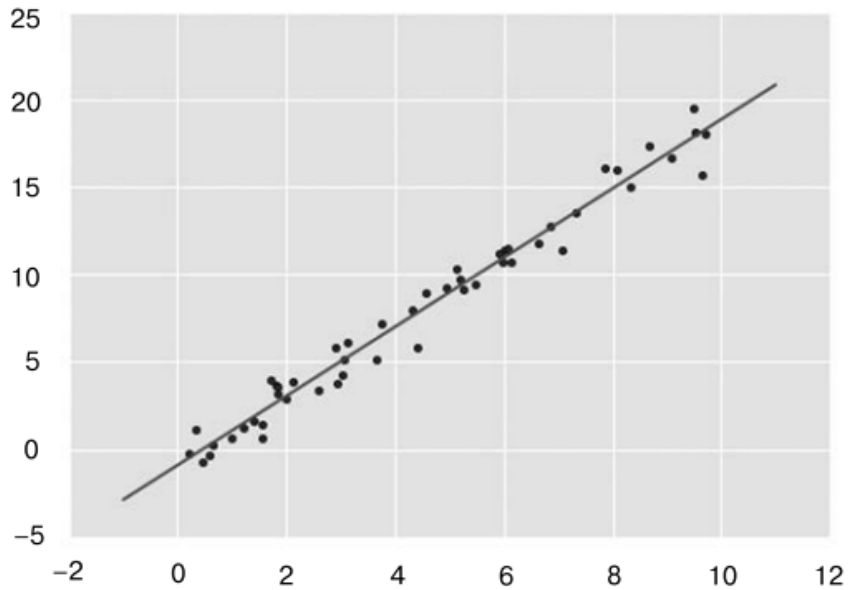
```
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

Пример: линейная регрессия

5. Предсказание новых значений

Визуализируем результаты, нарисовав сначала график исходных данных, а затем обученную модель:

```
plt.scatter(x, y)
plt.plot(xfit, yfit)
```



Комбинация базисных функций

позволяет приспособить линейную регрессию к нелинейным отношениям между переменными, путем преобразования данных в соответствии базисными функциями.

За основу берется многомерная линейную модель:

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 ...$$

x_1, x_2, x_3 получаются на основе имеющегося одномерного входного значения $x_n = F_n(x)$.
Например, если $x_n = x^n$, то модель превращается в полиномиальную регрессию:

$$y = a_0 + a_1 x^1 + a_2 x^2 + a_3 x^3 ...$$

- модель по-прежнему остается линейной - коэффициенты a_n не умножаются и не делятся друг на друга;
- мы взяли одномерные значения x и выполнили проекцию их на более многомерное пространство.

Полиномиальные базисные функции

Полиномиальная проекция встроена в библиотеку Scikit-Learn в виде преобразователя PolynomialFeatures:

```
from sklearn.preprocessing import PolynomialFeatures
x = np.array([2, 3, 4])
poly = PolynomialFeatures(3, include_bias=False)
poly.fit_transform(x[:, None])
```

Преобразователь превратит одномерный массив в трехмерный путем возведения каждого из значений в степень:

```
array([[ 2.,  4.,  8.],
[ 3.,  9.,  27.],
[ 4.,  16.,  64.]])
```

Полиномиальные базисные функции

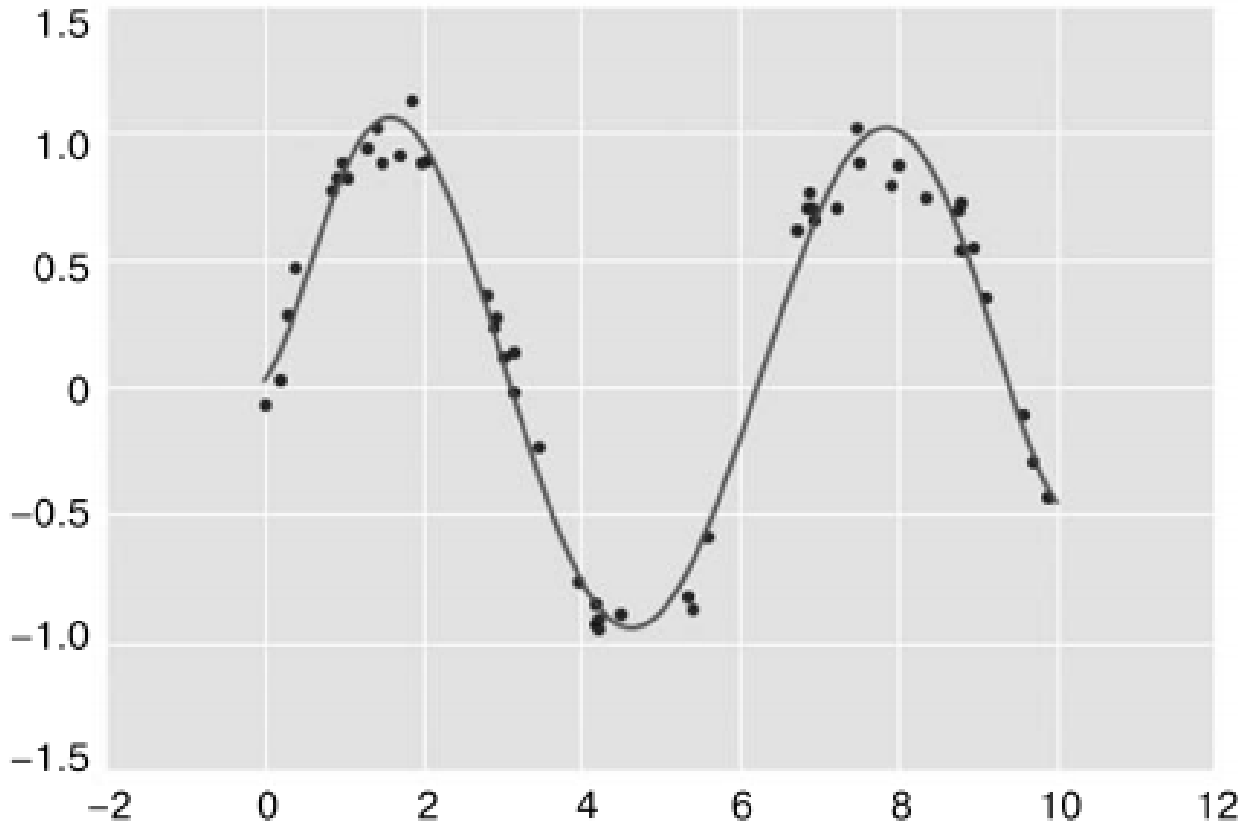
Создадим полиномиальную модель седьмого порядка:

```
from sklearn.pipeline import make_pipeline
poly_model = make_pipeline(PolynomialFeatures(7), LinearRegress
```

После такого преобразования можно воспользоваться линейной моделью для подбора намного более сложных зависимостей между величинами x и y:

```
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = np.sin(x) + 0.1 * rng.randn(50)
poly_model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 10, 1000)
yfit = poly_model.predict(xfit[:, np.newaxis])
plt.scatter(x, y)
plt.plot(xfit, yfit);
```

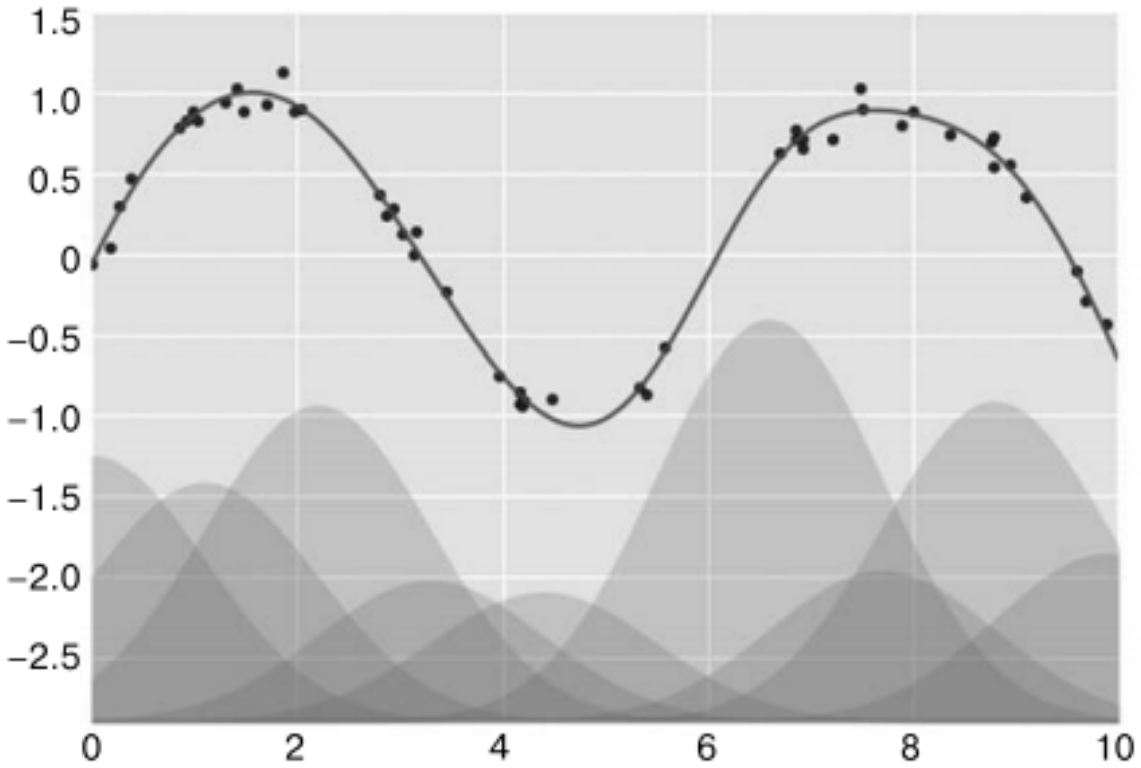
Полиномиальные базисные функции



С помощью линейной модели, используя полиномиальные базисные функции седьмого порядка, получена аппроксимацию этих нелинейных данных.

Гауссовы базисные функции

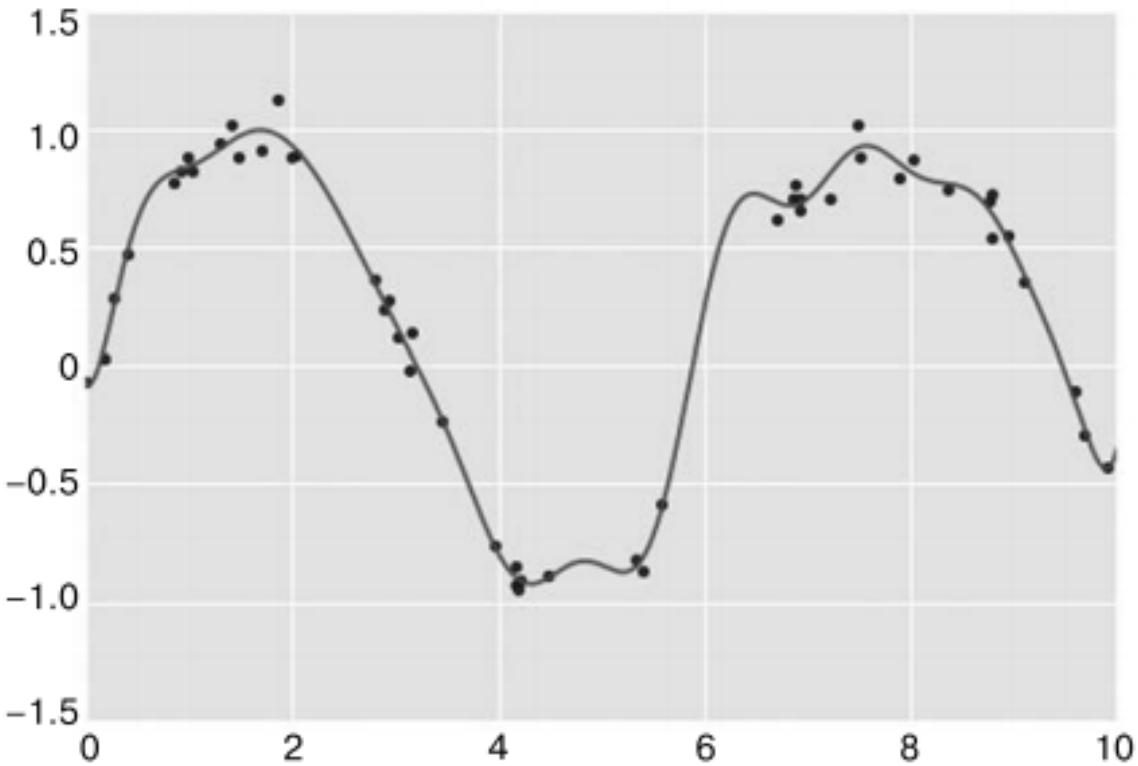
Один из полезных паттернов - обучение модели, представляющей собой сумму не полиномиальных, а Гауссовых базисных функций:



Затененные области - нормированные базисные функции, дающие при сложении аппроксимирующую данные гладкую кривую.

Гауссовы базисные функции

Гауссовы базисные функции не встроены в библиотеку Scikit-Learn, но можуj написать для их создания пользовательский преобразователь, как показано в примере (gauss.py):



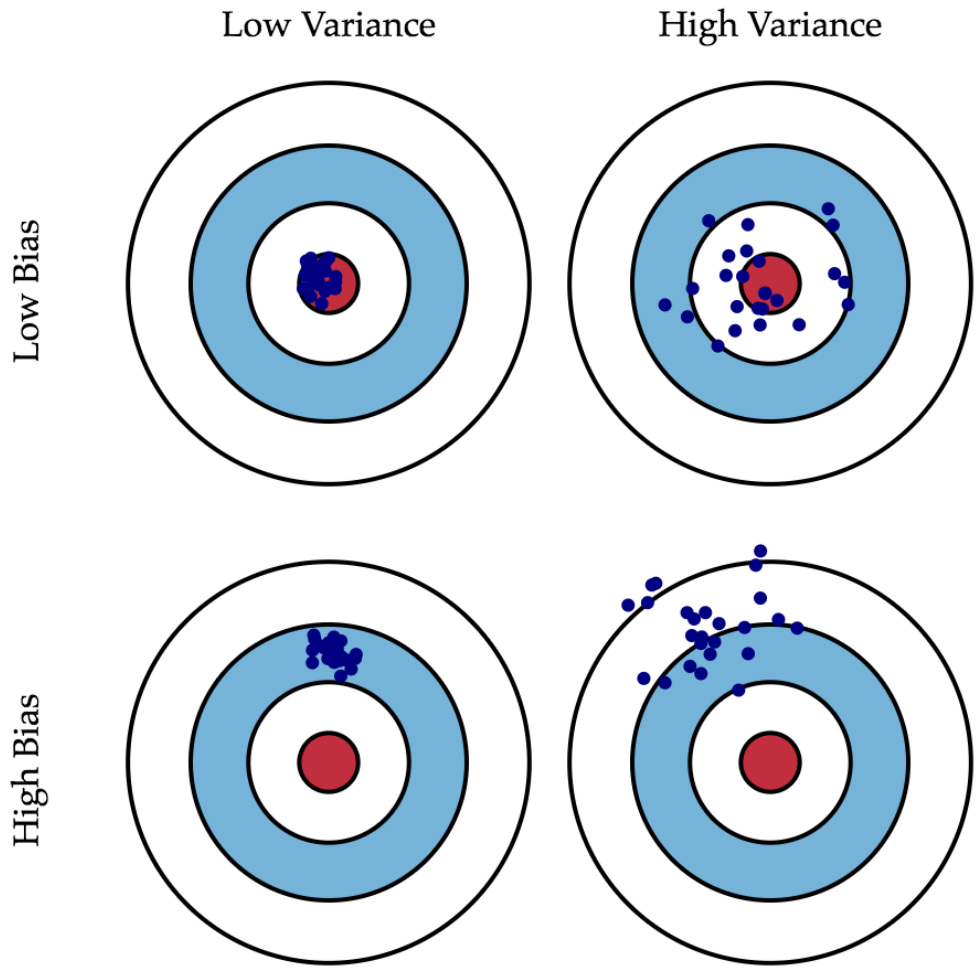
Разложение ошибки на смещение и разброс (Bias-variance decomposition)

Ошибка прогноза любой модели вида $y = f(\vec{x} + \epsilon)$ складывается из:

- квадрата смещения $Bias(\hat{f})$ - средняя ошибка по всевозможным наборам данных;
- дисперсии $Var(\hat{f})$ – вариативность ошибки, то, на сколько ошибка будет отличаться, если обучать модель на разных наборах данных;
- неустранимой ошибки σ^2.

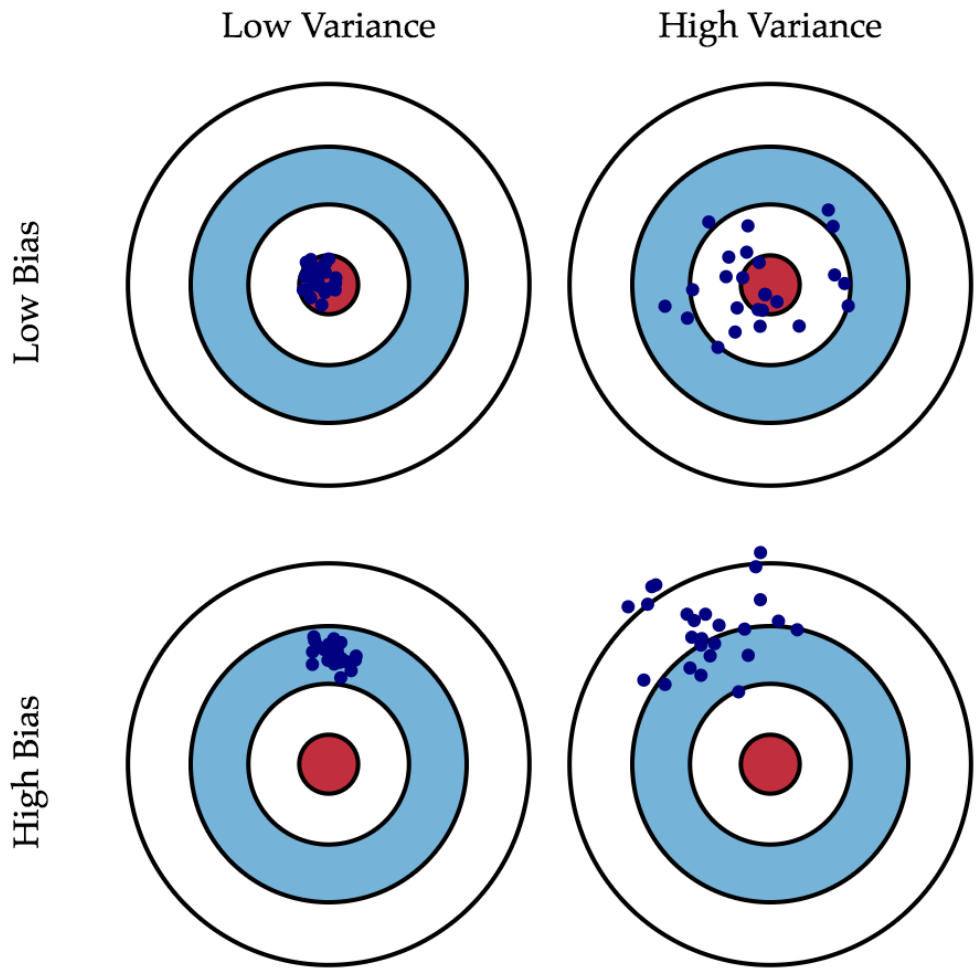
В идеале, конечно же, хотелось бы свести на нет оба этих слагаемых (левый верхний квадрат рисунка), но на практике часто приходится балансировать между смещенными и нестабильными оценками (высокая дисперсия).

Bias-variance decomposition

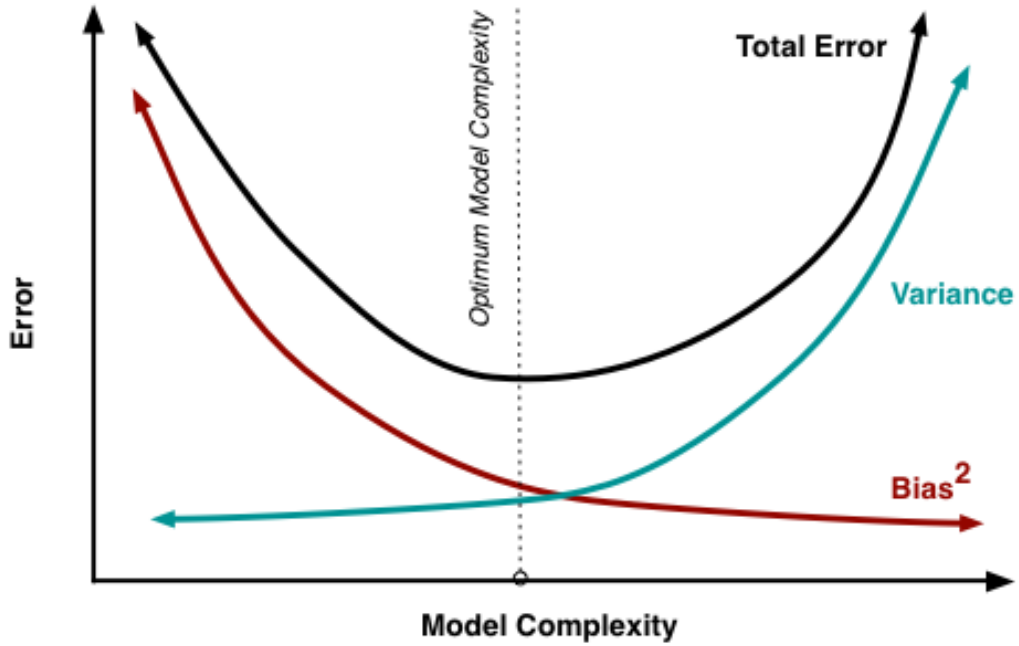


При увеличении сложности модели (например, при увеличении количества свободных параметров) увеличивается дисперсия (разброс)

Bias-variance decomposition



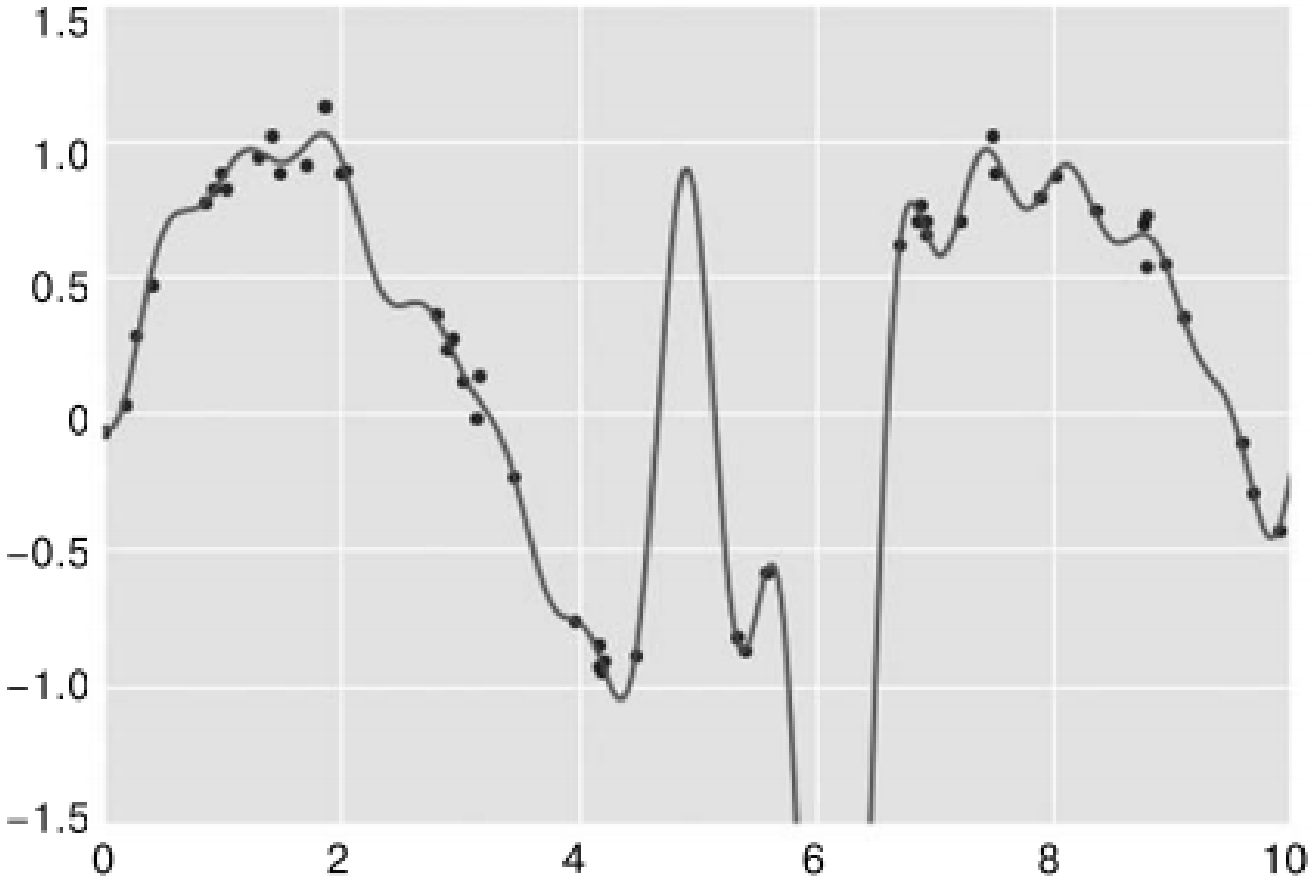
Переобучение. Недообучение



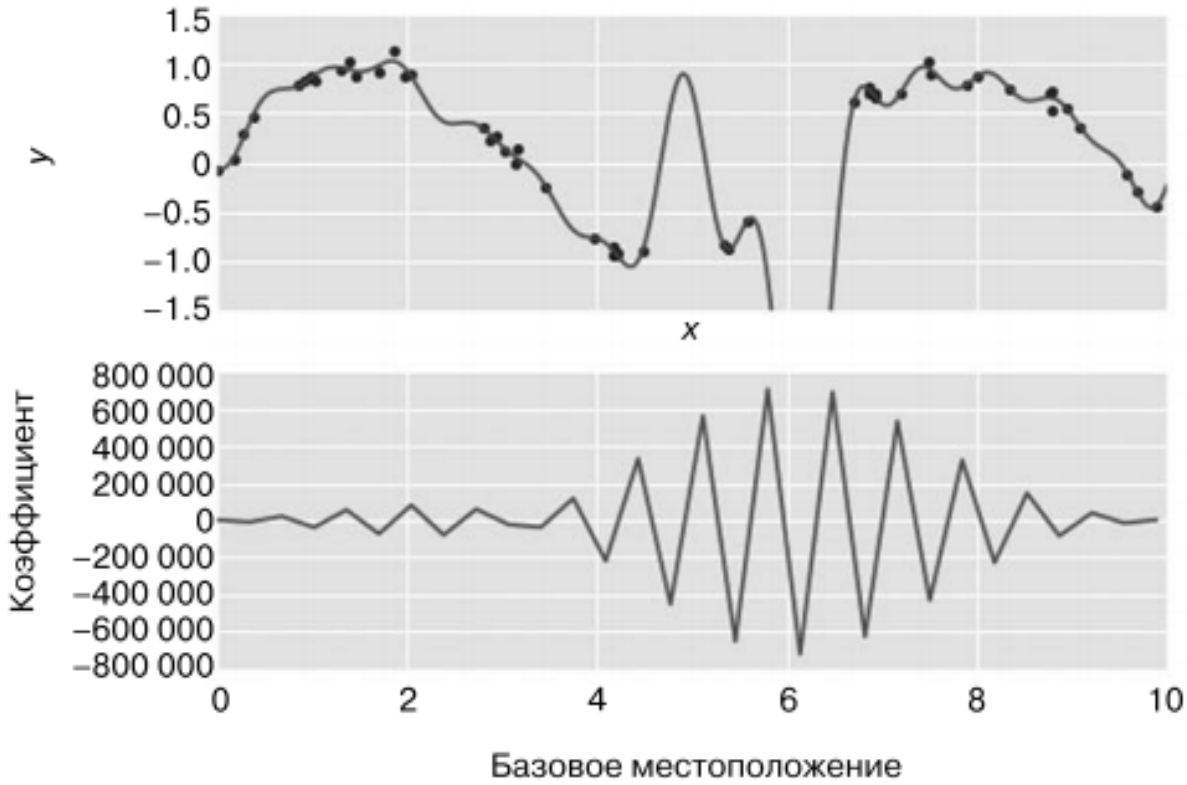
- В сложной модели тренировочный набор данных полностью запоминается вместо обобщения, небольшие изменения приводят к неожиданным результатам (**переобучение**).
- Если же модель слабая, то она не в состоянии выучить закономерность, в результате выучивается что-то другое, смещенное относительно правильного решения (**недообучение**).

Переобучение на Гауссовой модели

Если выбрать слишком много Гауссовых базисных функций, мы в итоге получим не слишком хорошие результаты.



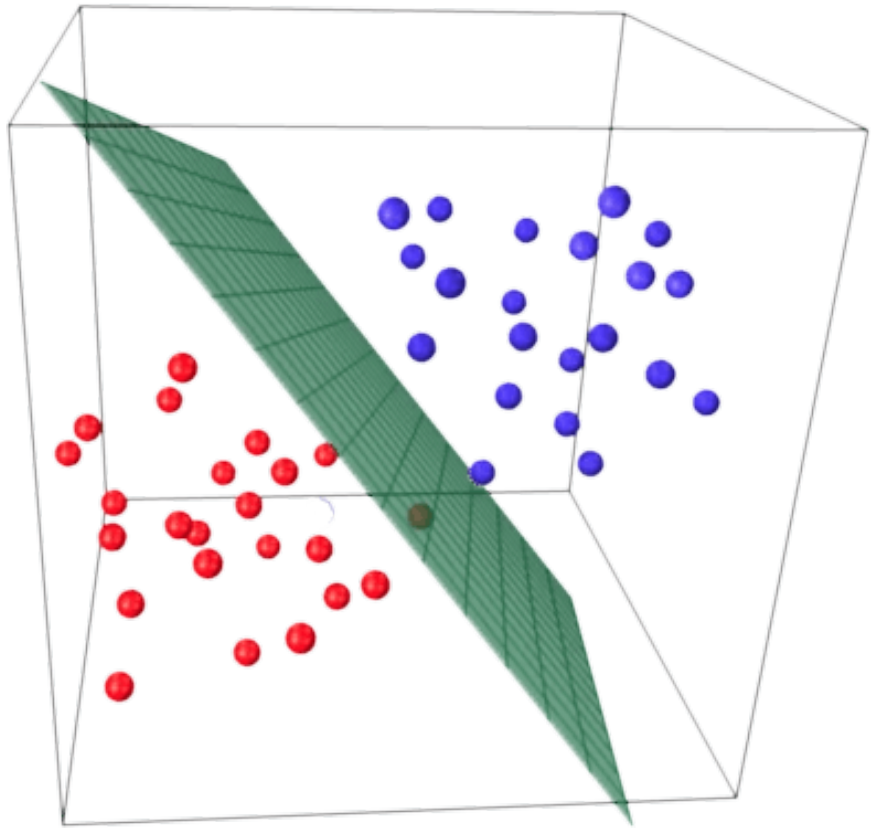
В результате проекции данных на 30-мерный базис модель оказалась слишком уж гибкой и стремится к экстремальным значениям в промежутках между точками, в которых она ограничена данными.



Причину этого можно понять, построив график коэффициентов Гауссовых базисных функций в соответствии с координатой x.

Линейный классификатор и логистическая регрессия

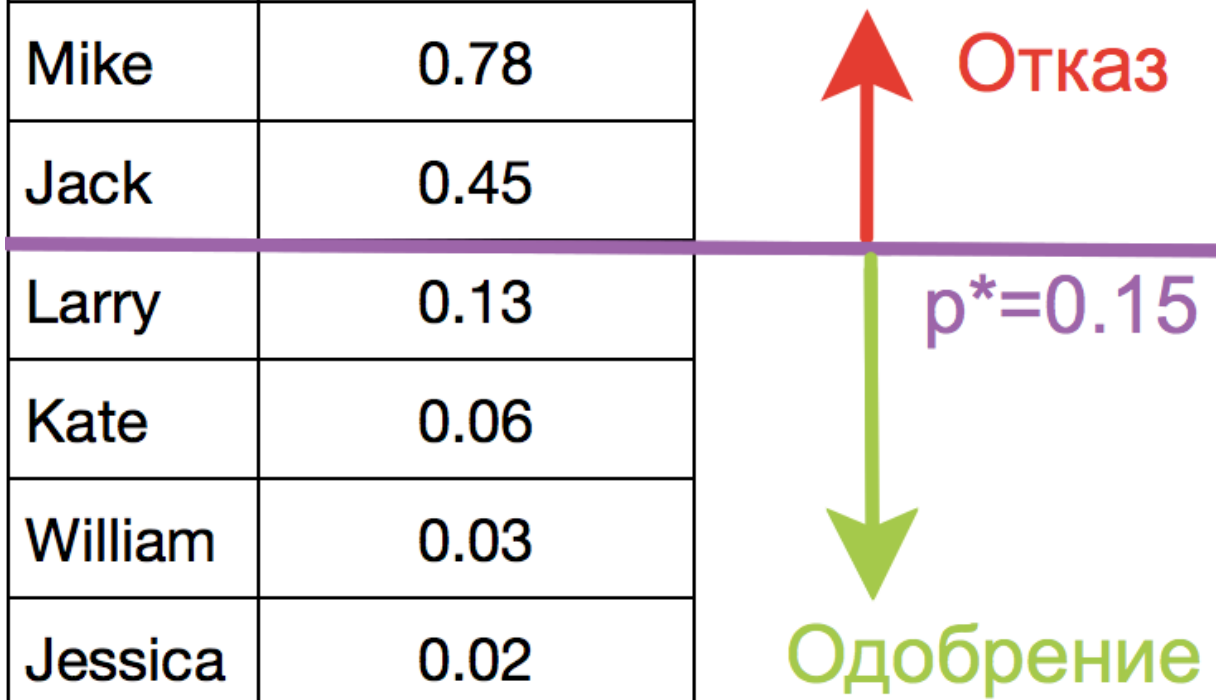
Основная идея линейного классификатора заключается в том, что признаковое пространство может быть разделено гиперплоскостью на два полупространства, в каждом из которых прогнозируется одно из двух значений целевого класса.



Логистическая регрессия

Логистическая регрессия - задача бинарной классификации, причем метки целевого класса "+1"(положительные примеры) и 1"(отрицательные примеры).

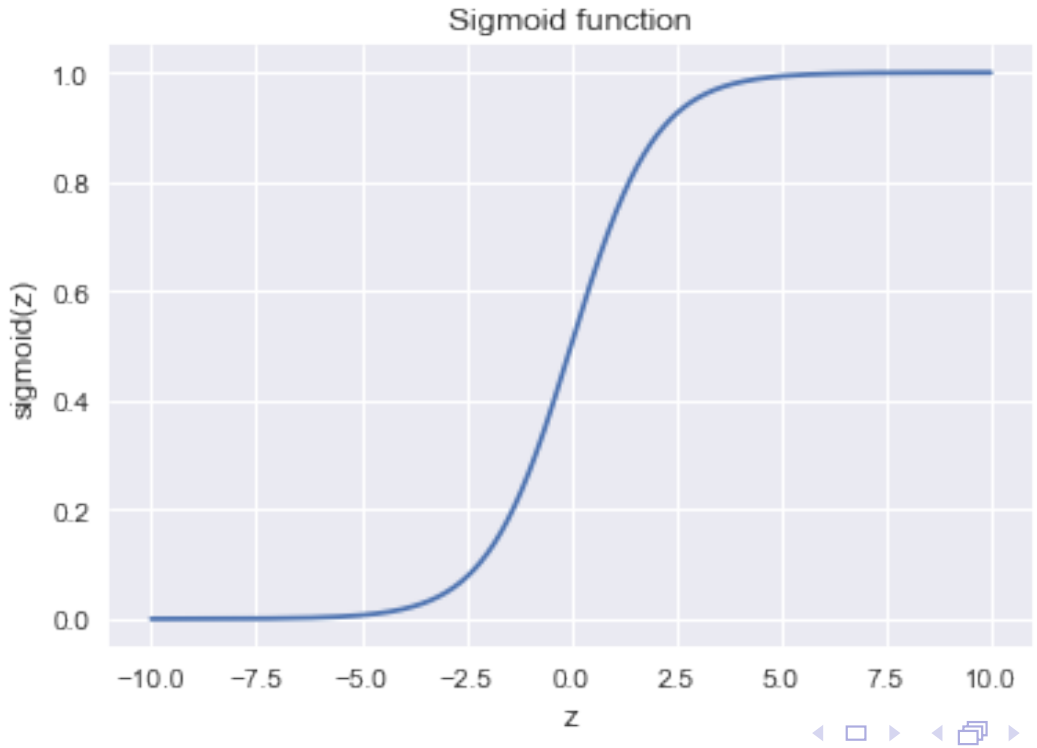
Клиент	Вероятность невозврата
Mike	0.78
Jack	0.45
Larry	0.13
Kate	0.06
William	0.03
Jessica	0.02



Логистическая регрессия

Для преобразования линейного прогноза $b(\vec{x}) = \vec{w}^T\vec{x} \in R$ в вероятность отнесения к классу необходима функция $f : R \to [0, 1]$.
В модели логистической регрессии берется функция

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Регуляризация

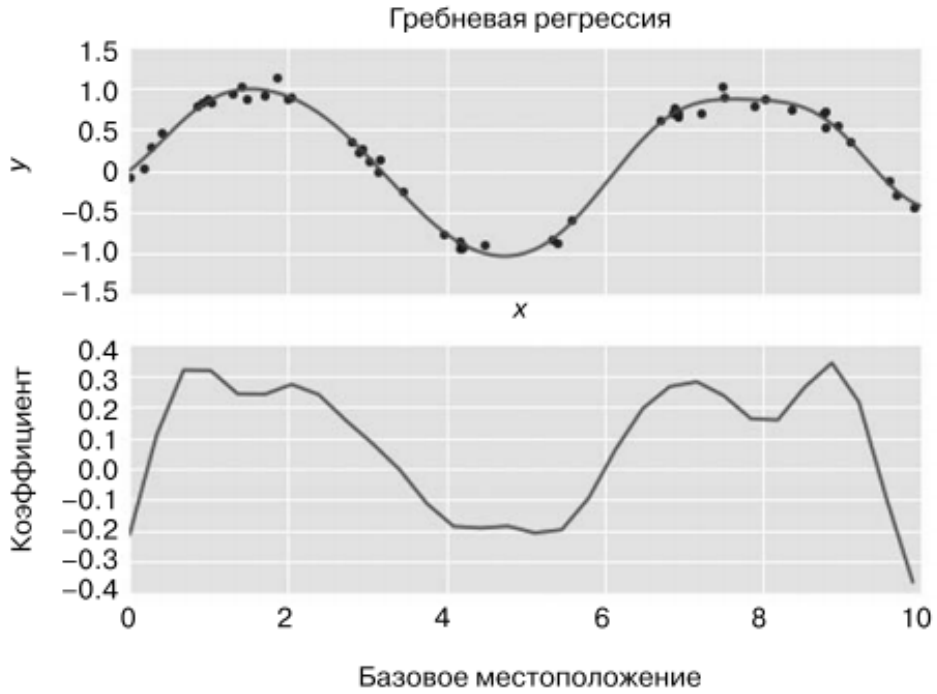
В задаче классификации, не умея напрямую минимизировать число ошибок, мы минимизируем некоторую ее верхнюю оценку, задаваемую **логистической функции потерь**.

$$\hat{w} = \arg\min_{\vec{w}} J(X, \vec{y}, \vec{w}) = \arg\min_{\vec{w}} \left(C \sum_{i=1}^{\ell} \log(1 + \exp^{-y_i \vec{w}^T \vec{x_i}}) + |\vec{w}|^2\right)$$

L2-регуляризация модели встроена в библиотеку Scikit-Learn в виде оценивателя Ridge.

```
from sklearn.linear_model import Ridge
model = make_pipeline(GaussianFeatures(30), Ridge(alpha=0.1))
basis_plot(model, title='Ridge Regression')
```

Параметр α служит для управления сложностью получаемой в итоге модели.



- В предельном случае $\alpha \to 0$ мы получаем результат, соответствующий стандартной линейной регрессии;
- в предельном случае $\alpha \to \infty$ будет происходить подавление любого отклика модели.

Пример. Регуляризация логистической регрессии

- Посмотрим, как регуляризация влияет на качество классификации на наборе данных по тестированию микрочипов из курса Andrew Ng по машинному обучению.
- Будем использовать логистическую регрессию с полиномиальными признаками и варьировать параметр регуляризации C.
- Сначала посмотрим, как регуляризация влияет на разделяющую границу классификатора, интуитивно распознаем переобучение и недообучение.
- Потом численно установим близкий к оптимальному параметр регуляризации с помощью кросс-валидации (cross-validation) и перебора по сетке (GridSearch).

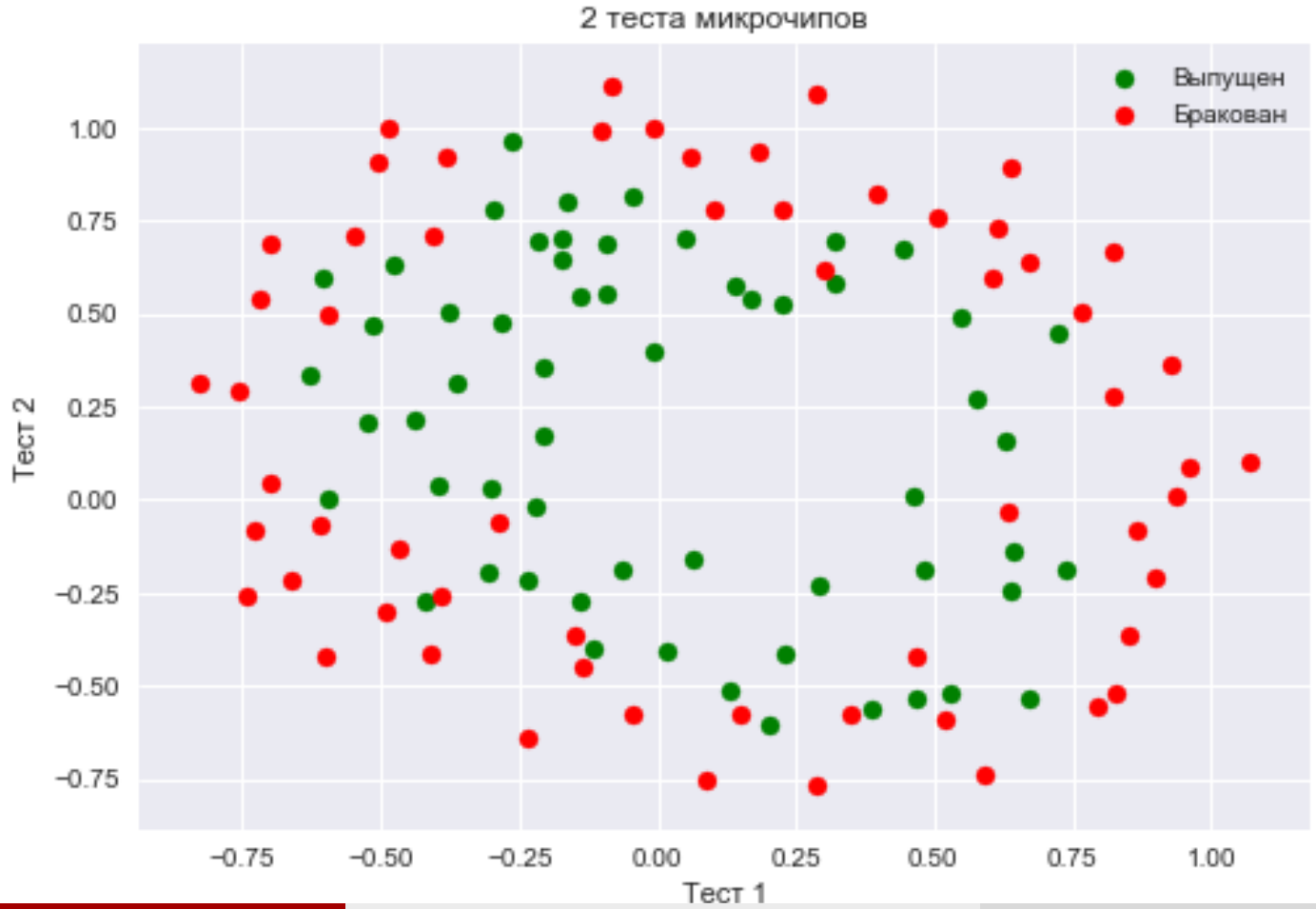
Пример. Регуляризация логистической регрессии

	test1	test2	released
0	0.051267	0.69956	1
1	-0.092742	0.68494	1
2	-0.213710	0.69225	1
3	-0.375000	0.50219	1
4	-0.513250	0.46564	1

	test1	test2	released
113	-0.720620	0.538740	0
114	-0.593890	0.494880	0
115	-0.484450	0.999270	0
116	-0.006336	0.999270	0
117	0.632650	-0.030612	0

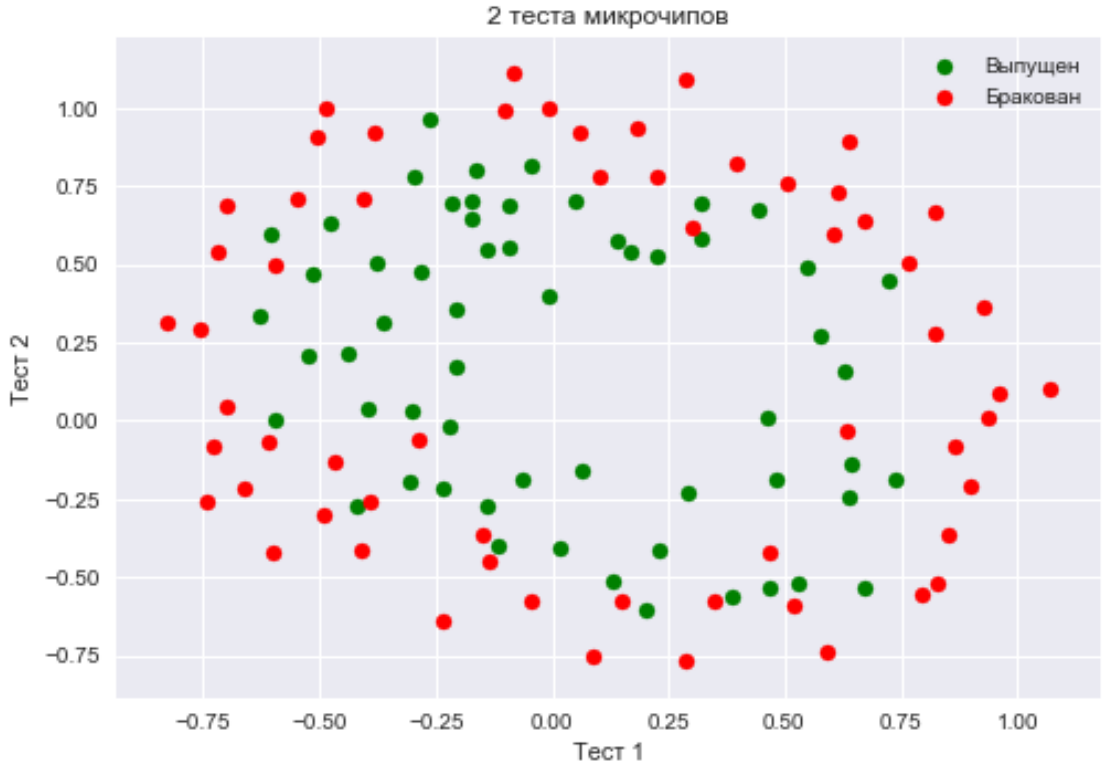
Пример. Регуляризация логистической регрессии

Сохраним обучающую выборку и метки целевого класса в отдельных массивах NumPy. Отобразим данные. Красный цвет соответствует бракованным чипам, зеленый – нормальным.

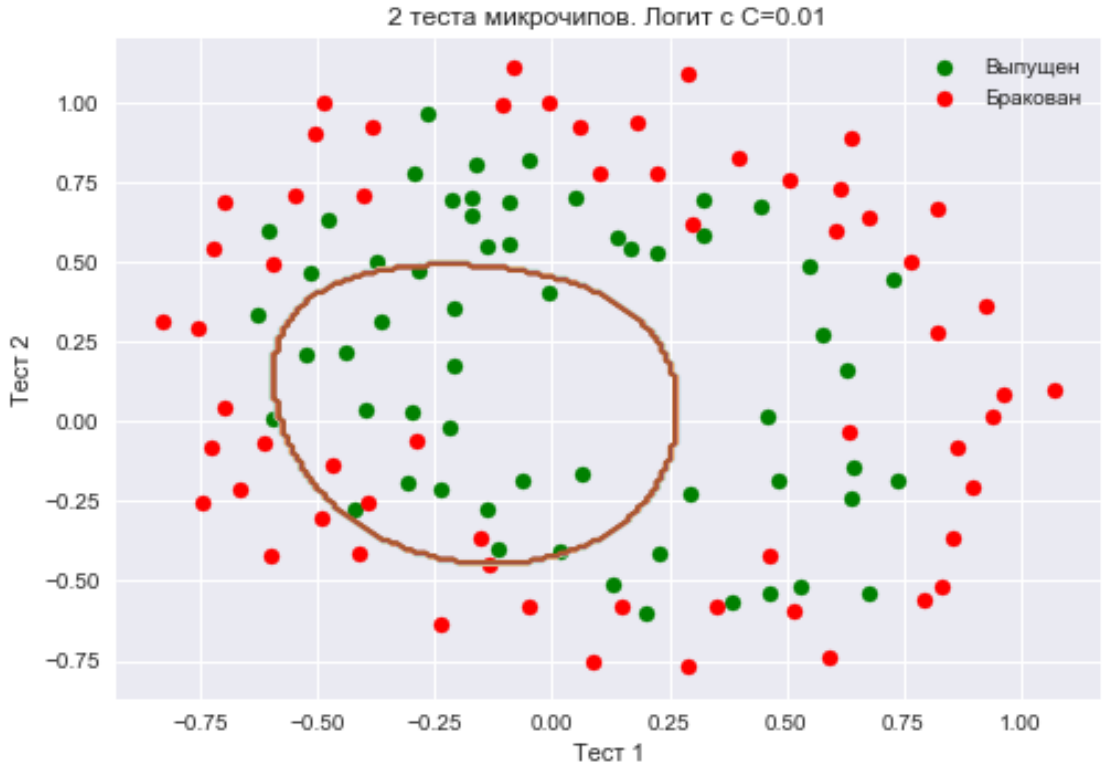


Пример. Регуляризация логистической регрессии

Сохраним обучающую выборку и метки целевого класса в отдельных массивах NumPy. Отобразим данные. Красный цвет соответствует бракованным чипам, зеленый – нормальным.

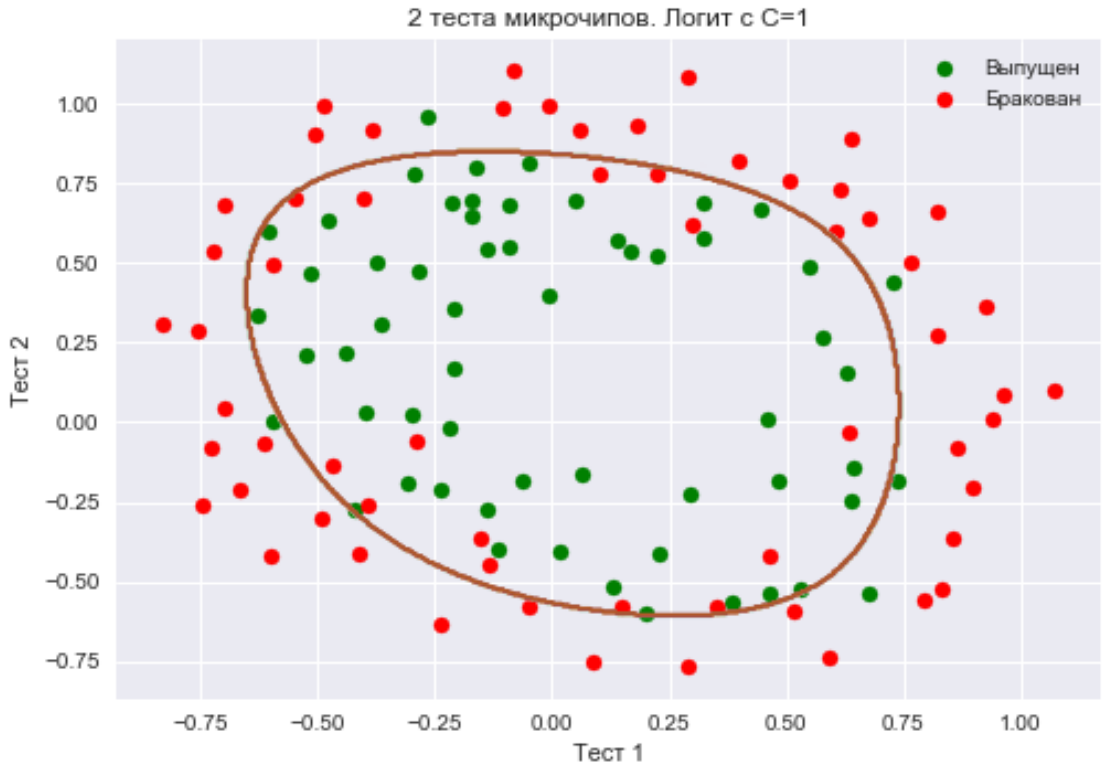


Обучим логистическую регрессию с параметром регуляризации $C = 10^{-2}$ с полиномами степени 7.



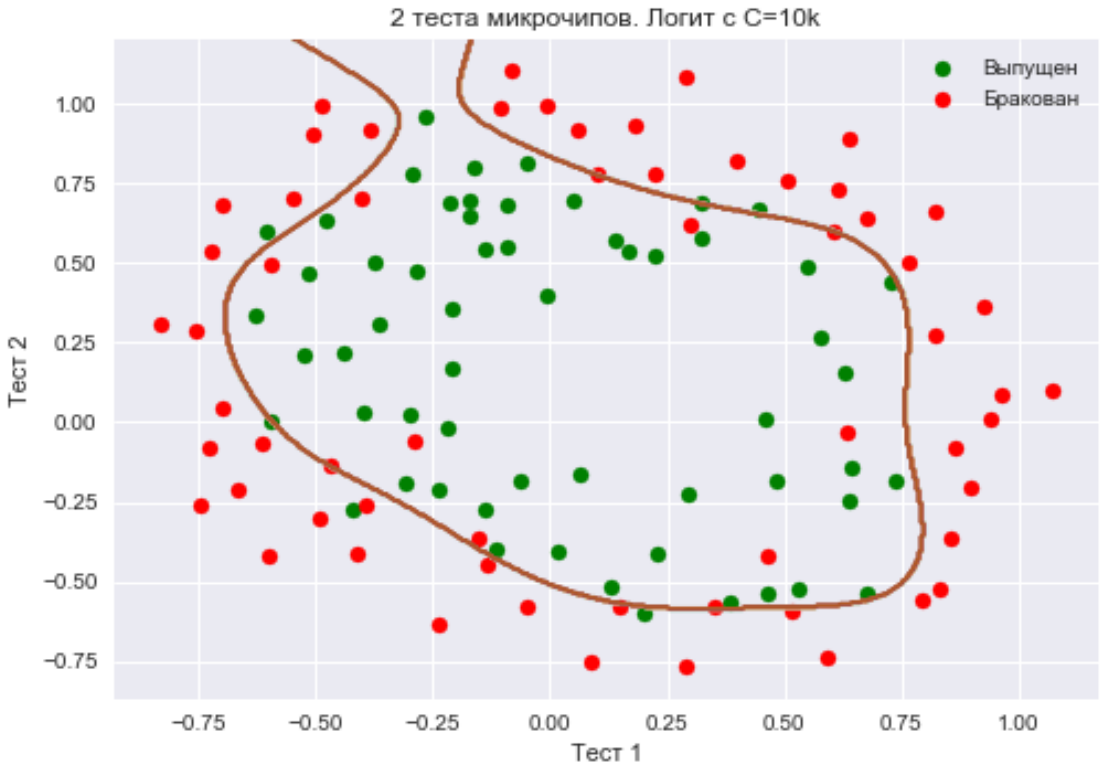
Регуляризация оказалась слишком сильной, и модель "недообучилась". Доля правильных ответов классификатора на обучающей выборке оказалась равной 0.627.

Увеличим C до 1. Тем самым мы ослабляем регуляризацию, теперь в решении значения весов логистической регрессии могут оказаться больше (по модулю), чем в прошлом случае.



Теперь доля правильных ответов классификатора на обучающей выборке – 0.831.

Еще увеличим – до 10 тысяч. Теперь регуляризации явно недостаточно, и мы наблюдаем переобучение.



Доля правильных ответов классификатора на обучающей выборке – 0.873.

Выводы:

- чем больше параметр C, тем более сложные зависимости в данных может восстанавливать модель;
- если регуляризация слишком сильная (малые значения C), модель окажется недообученной (1 случай) и недостаточно "штрафуется"за ошибки;
- если регуляризация слишком слабая (большие значения C), модель слишком "боится"ошибиться на объектах обучающей выборки, поэтому окажется переобученной (3 случай);
- то, какое значение C выбрать, сама логистическая регрессия "не поймет"(или еще говорят "не выучит"), то есть это не может быть определено решением оптимизационной задачи, которой является логистическая регрессия (в отличие от весов w).