курс «Прикладные задачи анализа данных»

Подготовка данных (Data Preprocessing)

Александр Дьяконов

План

Фундаментальные свойства данных

Виды данных

Предобработка данных (все этапы)

Очистка данных (Data Cleaning),сокращение данных (Data Reduction), трансформация данных (Data Transformation), интеграция данных (Data Integration).

Данные

**More data beats clever algorithms,
but better data beats more data
(P. Norvig)**

computer scientists	data scientists
algorithm-driven	data-driven
организовывают законы	открывают законы
ошибки – катастрофа	ошибки – естественны
их нанимают, чтобы они делали код	их нанимают, чтобы они находили закономерности

Нулевой этап решения задачи АД

Фундаментальные свойства данных

Доступность (Accessibility)
Актуальность (Timeliness)
Ценность (Value added)
Истинность (Believability)

Остальные свойства – потом

Сбор данных (Data Collection)

На что смотреть:

- **размеры, размерность, число элементарных порций (объектов), разреженность, разрешение**, полнота

- **семантика данных, идентификация отдельных элементов и порций данных** (id объектов, связи между таблицами и т.п.)

- **структура данных, режим доступа к данным (online / offline), способ доступа, источник данных (source)**

Виды данных

- признаковые описания (матрица объект-признак)
- измерения
  - одномерные сигналы (ряды, звук и т.п.), последовательности, тексты
  - изображения
  - видео
- метрические данные
- данные в специальных форматах
  - графы
  - XML-файлы
  - пространственно-временные
  - сырые логи
  - и т.п.

Источники данных

Proprietary data sources	часто нельзя получить доступ
Government data sets	Data.gov
Academic data sets	при написании публикаций
Web search	«Scraping», есть лимиты и условия использования
Sensor data	Относительно дёшевы, но специфичны

+ Ваши данные – самые ценные

https://toolbox.google.com/datasetsearch

Свойства данных

Свойства данных	Что мешает этому свойству	Причины нарушения свойства	Средство борьбы
Корректность (точность, Accuracy) м.б. Истинность	Аномалии (выбросы + шум), «некорректности»	Погрешность приборов, ошибки при заполнении	Очистка данных (Data Cleaning)
Полнота (Completeness) Недостаточность	Пропуски м.б. разреженность объектов << признаков	Недоступность данных, ошибки при заполнении, сбои при записи	Очистка данных (Data Cleaning) Сокращение данных (Data Reduction)
Непротиворечивость (согласованность, Consistency)	«противоречия»	Различные источники данных	Интеграция (Data Integration)
Безызбыточность	Дубликаты Шум Излишняя дискретизация	Особенности интеграции, ошибки при заполнении	Сокращение данных (Data Reduction) Трансформация (Data Transformation)
Ясность Interpretability	«неясности»	Плохие хранение и подготовка д.	Трансформация (Data Transformation)
Структурированность Однородность	Сырые данные	Нет признаковых описаний Признаки в разных шкалах	Генерация признаков (Feature engineering) Трансформация (Data Transformation)

Предобработка данных (Data Preprocessing / Preparation)

– замена, модификация или удаление частей набора данных с целью повышения непротиворечивости, полноты, корректности и ясности набора данных, а также уменьшения избыточности

процесс преобразования данных в форму, удобную для анализа

Выполняется на полном наборе данных

(и на контрольных объектах тоже)

Тонкость: не допустить утечки

(информации, не доступной при функционировании модели)

Что бывает в данных

	дата	пол	образование	сумма	платёжная строка	число просрочек	?????	x_m
0	12/01/2017	1	высшее	5000.0	0000	0	0	0.00000
1	13/01/2017	1	высшее	2500.0	0000	1	1	1.00000
2	13/01/2017	1	высшее	2500.0	001000	1	1	1.00000
3	13/01/2017	0		13675.0	111	3	3	0.00000
4	25/01/2017	0		NaN	0	0	0	0.00000
5		1	начальное	NaN	00	0	0	0.00000
6	02/02/2017	1	среднее	1000.0		0	0	0.00000
7	01/01/0001	13/01/2017	среднее	0.0		-7	-7	-0.00001

Что бывает в данных

	дата	пол	образование	сумма	платёжная строка	число просрочек	?????	x_m
0	12/01/2017	1	высшее	5000.0	0000	0	0	0.00000
1	13/01/2017	1	высшее	2500.0	0000	1	1	1.00000
2	13/01/2017	1	высшее	2500.0	001000	1	1	1.00000
3	13/01/2017	0		13675.0	111	3	3	0.00000
4	25/01/2017	0		NaN	0	0	0	0.00000
5		1	начальное	NaN	00	0	0	0.00000
6	02/02/2017	1	среднее	1000.0		0	0	0.00000
7	01/01/0001 13/01/2017		среднее	0.0		-7	-7	-0.00001

неясность

дубликаты

некорректность

ошибка

нечисловой признак

пропуски

дубликаты

выброс

РАЗДЕЛЫ Предобработки данных

Очистка данных (Data Cleaning)

Обнаружение (и удаление / замена) …

- **аномалий / выбросов** (Anomaly Detection) **+ отдельная лекция**
- **пропусков** (Missing Data Imputation)
- **шумов** (Noise Identification)
- **некорректных значений** (Correct Bad Data / Filter Incorrect Data)

Сокращение данных (Data Reduction)

- **Сэмплирование** (Sampling)
- **Сокращение размерности** (Dimensionality reduction)
- **Отбор признаков** (Feature subset selection)
- **Отбор объектов** (Instance Selection)
  - **удаление дубликатов**

РАЗДЕЛЫ Предобработки данных

Трансформация данных (Data Transformation)

- Переименование признаков, объектов, значений признаков, преобразование типов
- Кодирование значений категориальных переменных + отдельная лекция
- Дискретизация (Discretization / Binning)
- Нормализация (Normalization)
- Сглаживание (Smoothing)
- Создание признаков (Feature creation) + отдельная лекция
- Агрегирование (Aggregation)
- Обобщение (Generalization)
- Деформация значений

Интеграция данных (Data Integration)

- Объединение данных из разных источников

Переименования

Названия переменных (и их значения ?!) должны быть интуитивны

(они используются в том числе при передачи данных коллегам, презентации результатов и т.п.)

	X5XX.	X5XV.	price($)	date
0	200$	Jan.1.2018	200	2018-01-01
1	150$	Feb.13.2017	150	2017-02-13
2	7000$		7000	NaT
3	110$	Jun.13.1996	110	1996-06-13

Преобразования типов данных

Нужно использовать типы, которые поддерживает Ваша среда программирования

	X5XX.	X5XV.	price($)	date
0	200$	Jan.1.2018	200	2018-01-01
1	150$	Feb.13.2017	150	2017-02-13
2	7000$		7000	NaT
3	110$	Jun.13.1996	110	1996-06-13

```
df.rename(columns={'X5XX.': 'price($)'}, inplace=True)
df['price($)'] = df['price($)'].apply(lambda x: int(x[:-1]))
# быстрее?!
df['price($)'] = df['price($)'].apply(lambda x: x.replace('$', '')).astype(int)
df['date'] = pd.to_datetime(df['X5XV.'], errors='coerce')
```

случай из практики: 3/4/2001 = «3 апреля 2001» или «4 марта 2001»

(а бывает, что на разных частях данных по-разному)

Кодировки

Как правило, компьютер работает с числами ⇒
категории представляем числами (векторами)
(дальше подробнее)

	ans	weather	ans_coded	weather_coded
0	yes	warm	1	0
1	no	cool	0	1
2	yes	cold	1	2
3	no	warm	0	0

```
dct = {'yes': 1, 'no': 0}
df['ans_coded'] = df['ans'].map(dct)
# быстрее?!
df['weather_coded'] = df.weather.factorize()[0]
```

Корректировка значений

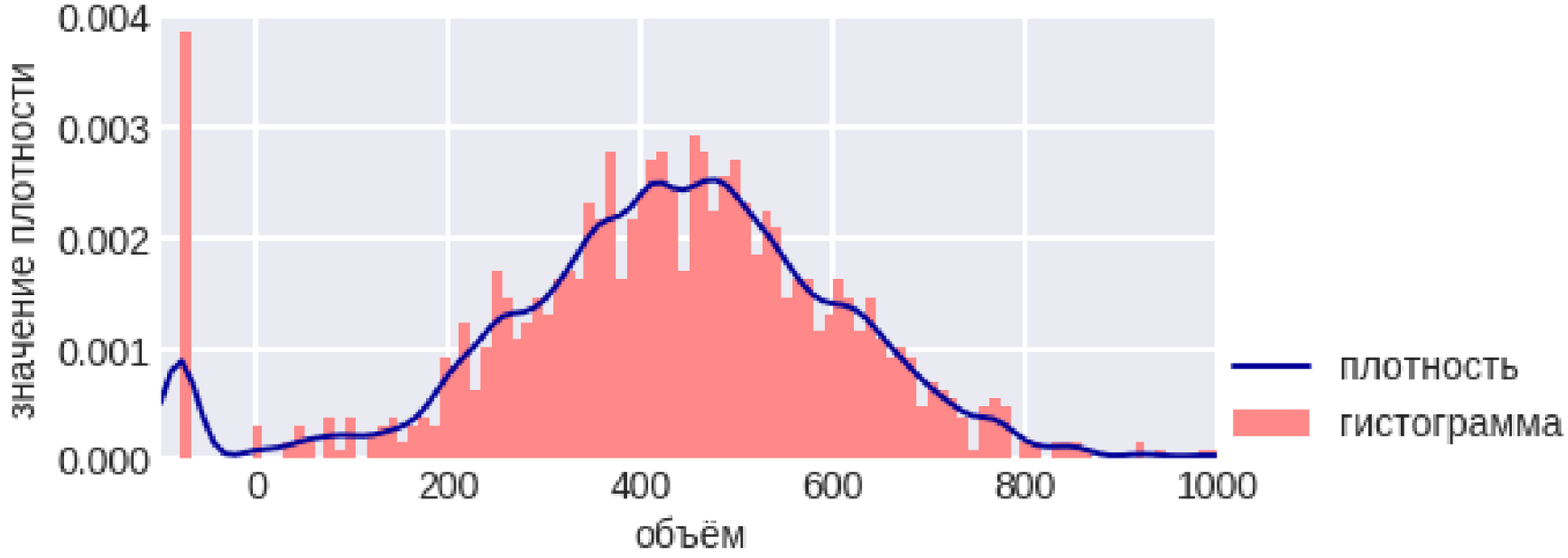
	время	давление	в.давл.	н.давл.	время
0	23:10	120/80	120	80	2018-09-13 23:10:00
1	10 часов	120/70	120	70	2018-09-13 10:00:00
2	7:40	110/70	110	70	2018-09-13 07:40:00

```
tmp = df['давление'].str.split('/')
df['в.давл.'] = tmp.apply(lambda x: x[0])
df['н.давл.'] = tmp.apply(lambda x: x[1])
# быстрее
df[['в.давл.', 'н.давл.']] = pd.DataFrame(df['давление'].str.split('/',
                    1).tolist(), columns = ['Давление_в','Давление_н'])
# тоже быстрее
df[['в.давл.', 'н.давл.']] = df['Давление'].str.split('/', expand=True)
# очень быстро, если нет ошибок в данных
st = '/'.join(df['Давление'])
df[['в.давл.', 'н.давл.']] = pd.DataFrame(np.array(st.split('/')).reshape(-1, 2))
```

Д/З Как лучше? Относится к любому фрагменту кода

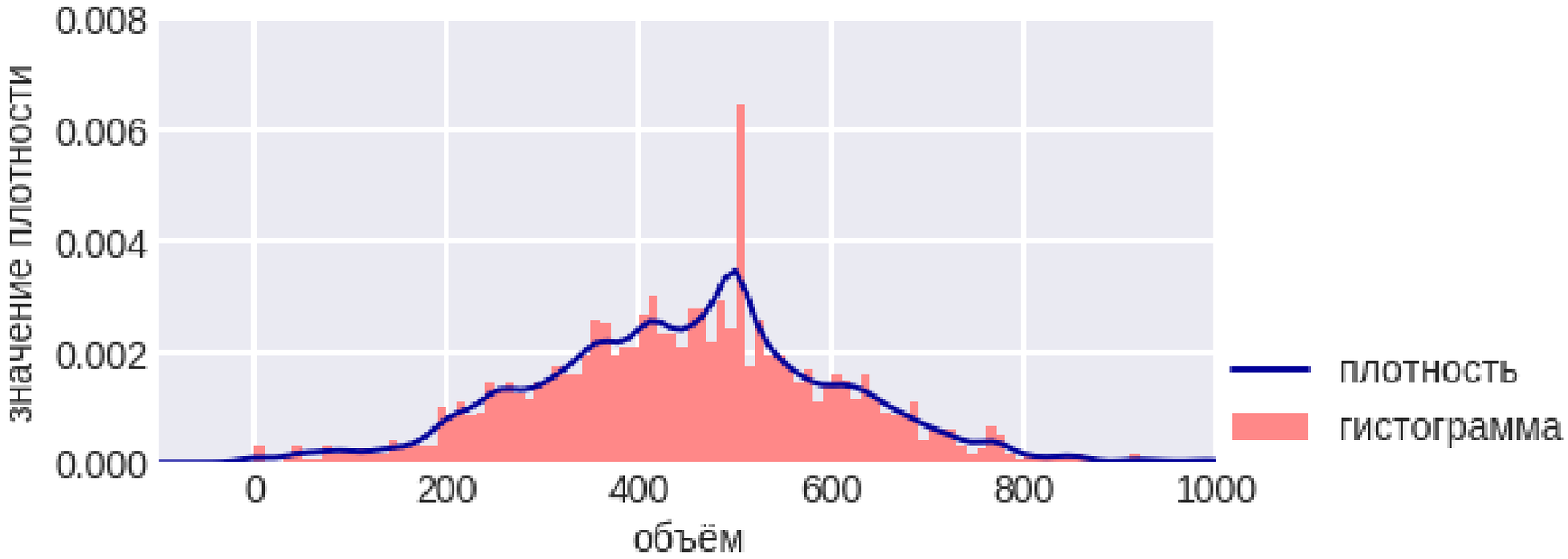
Пропуски – как выглядят в данных

- **пустые значения**
- **специальные значения** (NA, NaN, null, ...)
- **специальный код** (–999, mean, число за пределами значения признака)




```
df[name].isnull().sum() # число "нанов"
  df[name].count() # число не "нанов"
```

Пропуски – как выглядят в данных




```
500    53
540     8
523     8
469     7
451     7
419     7
435     6
   ...
```

```
df['volume'].value_counts()
```

Пропуски – что делать

- **оставляем**

(но не все модели могут работать с пропусками)

- **удаляем описания объектов с пропусками / признаки**

(радикальная мера, которая редко используется)

```
df.dropna(how='any', axis=1)
```

- **заменяем на фиксированное значение**

(например, если признак бинарный, то на 0.5)

Значение –999, как правило, плохое – является выбросом

```
df.fillna(-1)
```

- **заменяем на легковычислимое значение**

(среднее, медиана, мода)

```
df.fillna(df.mean()) # , inplace=True
```

- **восстановление значения**

(построение специальной модели для восстановления)

```
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values='NaN', strategy='mean', axis=0)
vals = imputer.fit_transform(df[['сумма']])
```

- **экспертная замена** (см. ниже)

Пропуски – итеративная процедура

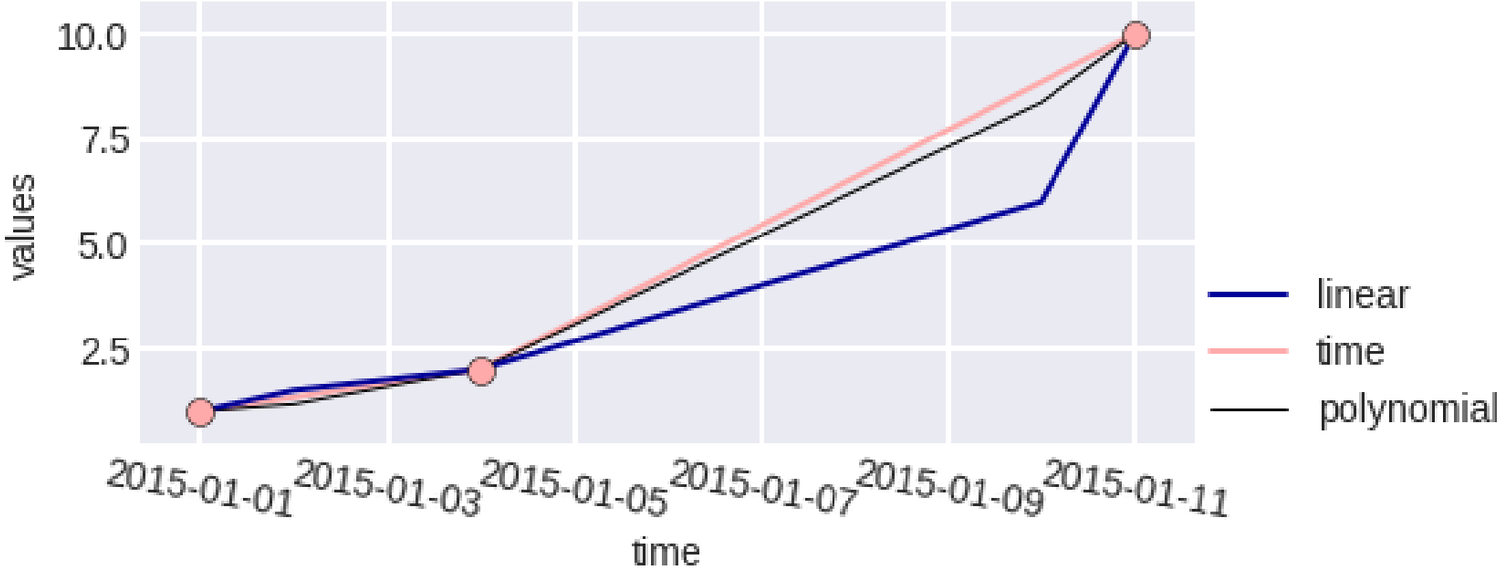
```
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

imp = IterativeImputer(max_iter=10, random_state=0)
imp.fit([[1, 2], [3, 6], [4, 8], [np.nan, 3], [7, np.nan]])
IterativeImputer(add_indicator=False, estimator=None,
                 imputation_order='ascending', initial_strategy='mean',
                 max_iter=10, max_value=None, min_value=None,
                 missing_values=nan, n_nearest_features=None,
                 random_state=0, sample_posterior=False, tol=0.001,
                 verbose=0)

X_test = [[np.nan, 2], [6, np.nan], [np.nan, 6]]

# the model learns that the second feature is double the first
print(np.round(imp.transform(X_test)))
[[ 1.  2.]
 [ 6. 12.]
 [ 3.  6.]]
```

https://scikit-learn.org/stable/modules/impute.html

Пропуски – что делать

```
ts = pd.Series(vals, index=index)
ts2 = ts.interpolate()
ts3 = ts.interpolate(method='time')
ts4 = ts.interpolate(method='polynomial', order=2)
```

	ts	ts2	ts3	ts4
2015-01-01	1.0	1.0	1.00	1.00
2015-01-02	NaN	1.5	1.33	1.17
2015-01-04	2.0	2.0	2.00	2.00
2015-01-10	NaN	6.0	8.86	8.37
2015-01-11	10.0	10.0	10.00	10.00



Пропуски – тонкости

- **добавлять характеристический признак пропусков «is_nan»**

тогда модель сама определит оптимальное значение для заполнения

- **заполнять пропуски лучше после генерации признаков**

иначе возникают дополнительные неопределённости

	площадь	цена	цена/кв.м.	площадь	цена	цена/кв.м.	площадь	цена	цена/кв.м.
0	82.0	5200000.0	63414.634146	82.0	5200000.0	63414.634146	82.0	5200000.0	63414.6
1	70.0	4400000.0	62857.142857	70.0	4400000.0	62857.142857	70.0	4400000.0	62857.1
2	74.0	4200000.0	56756.756757	74.0	4200000.0	56756.756757	74.0	4200000.0	56756.8
3	60.0	NaN	NaN	60.0	4350000.0	72500.000000	60.0	4350000.0	61009.5
4	NaN	3600000.0	NaN	71.5	3600000.0	50349.650350	71.5	3600000.0	61009.5
5	NaN	NaN	NaN	71.5	4350000.0	60839.160839	71.5	4350000.0	61009.5

```
df['цена/кв.м.'] = df['цена'] / df['площадь']
df.fillna(df.mean())
```

Пропуски – тонкости

- **не допускать ликов при заполнении пропусков**

Общий пайплайн: предобработка данных + классификация

при заполнении пропусков нельзя брать информацию из будущего

В соревнованиях м.б. выгодно знать контроль

Пропуски – тонкости

```
df['площадь'].fillna(df['площадь'].mean(), inplace=True)
df['площадь'].fillna(df[df['data'] == 'train']['площадь'].mean(), inplace=True)
df.loc[df['data'] == 'train', 'площадь'] =
df[df['data'] == 'train']['площадь'].fillna(df[df['data'] == 'train']['площадь'].mean())
df.loc[df['data'] == 'test', 'площадь'] =
df[df['data'] == 'test']['площадь'].fillna(df[df['data'] == 'test']['площадь'].mean())
# быстрее?!
df['площадь'] = df.groupby("data")['площадь'].transform(lambda x: x.fillna(x.mean()))
# ещё быстрее?!
df.loc[df['площадь'].isnull(), 'площадь'] = df.groupby('data')['площадь'].transform('mean')
# ещё ?!
gb = df.groupby('data')
mean = gb.mean()
for gn, x in gb:
    x['площадь'].fillna(mean.loc[gn])
```

	data	площадь	площадь_1	площадь_2	площадь_3
0	train	82.0	82.0	82.0	82.0
1	train	NaN	66.5	78.0	78.0
2	train	74.0	74.0	74.0	74.0
3	test	60.0	60.0	60.0	60.0
4	test	NaN	66.5	78.0	55.0
5	test	50.0	50.0	50.0	50.0

Пропуски – тонкости

- **важно понимать природу пропуска:**

значение может не быть доступно

клиент банка не указал в анкете свой возраст

отсутствие информации – тоже информация!

значение может не существовать

«Доход» для детей моложе 18 (=0)

значение не является числом

0/0 = NaN

средняя покупка в категории товаров

значение вызвана предобработкой данных

при конкатенации таблиц – несуществующие колонки

при обработке дат – исключение

Пропуски – тонкости

- **обучение и тест – одинаковые распределения**

Тоже самое для пропусков!

Пример: в задаче диагностики оборудования в прошлом слишком много пропусков, оказалось – это старые модели датчиков, т.е. все признаки снимались с «других» датчиков.

Пропуски – тонкости

- можно посмотреть, зависит ли факт пропуска от других данных

	data	площадь	target
0	train	82.0	0
1	train	NaN	1
2	train	74.0	0
3	test	60.0	0
4	test	NaN	1
5	test	50.0	0

целевой признак – характеристический признак пропуска

```
df['target'] = df['площадь'].isnull().astype(int)
```

Зашумлённые данные (Noisy Data)

Аналогия с пропусками

Что делать

- оставляем (но будет погрешность при моделировании)
- удаляем сильно зашумлённые признаки
- удаляем сильно зашумлённые объекты
- замена аномальных значений (ex: clipping)

могут нести важную информацию!

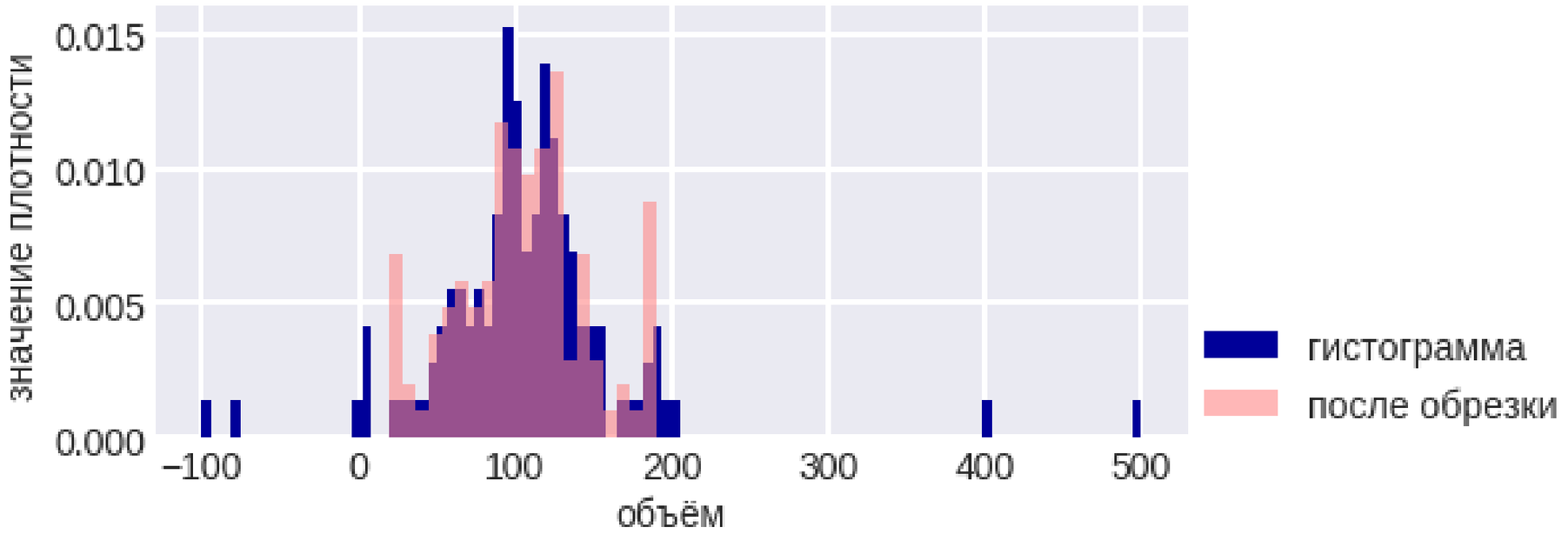
Главный вопрос: «Почему в данных есть это?»

Причины

- ошибка сбора данных (ex: погрешность прибора, ввода и т.п.)
- ошибка обработки данных
- свойство данных (ex: выброс – зарплата CEO)

Отдельная тема: обнаружение аномалий

Зашумлённые данные – Винсоризация (Winsorizing)




```
x2 = df.x.clip(lower=df.x.quantile(0.05),
               upper=df.x.quantile(0.95))

import scipy
x3 = scipy.stats.mstats.winsorize(x, limits = 0.05)
```

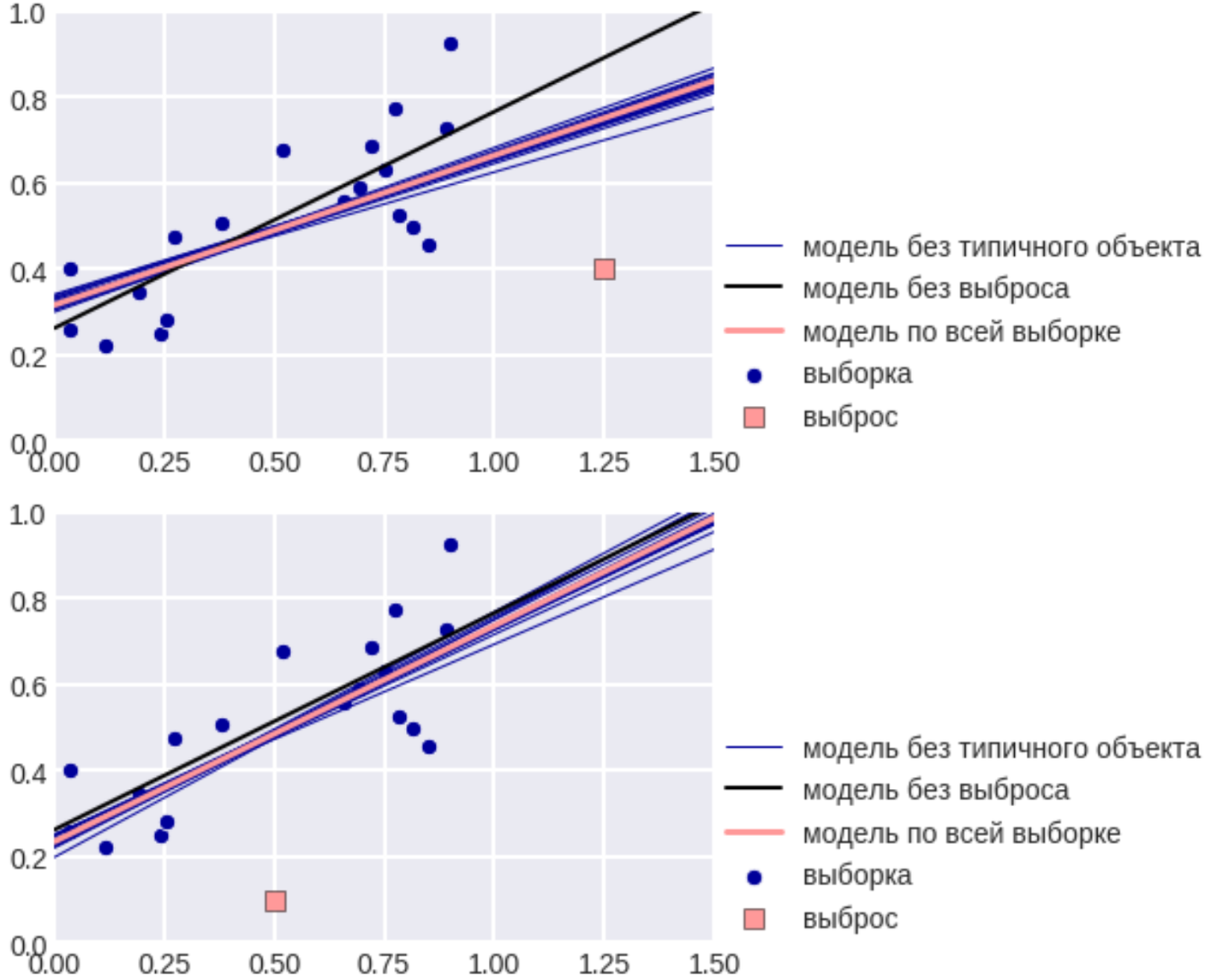
Зашумлённые данные – Тонкость

Если есть шум, можем ли доверять и другим признакам?

выбросы в обычном признаке / целевом

Плохо для линейных моделей,
но есть модели «устойчивые к выбросам»

В чём разница между этими выбросами?



Агрегирование (Aggregation)

pressure_1	pressure_2	pressure_3	pressure_4	pressure_5	pr_mean	pr_std	pr_max	pr_min
14	15	18	23	10	16.0	4.85	23	10
36	13	14	21	16	20.0	9.46	36	13
10	14	16	17	20	15.4	3.71	20	10
13	20	15	25	13	17.2	5.22	25	13

Составляющие суммы, замеры разными датчиками и т.п.

Часто лучше использовать различные статистики!

```
df['pr_mean'] = df[cols].mean(axis=1)
df['pr_std'] = df[cols].std(axis=1) #.round(2)
df['pr_max'] = df[cols].max(axis=1)
df['pr_min'] = df[cols].min(axis=1)
```

Совет: часто хорошо отсортировать построчно показания

Обобщение (Generalization)

	товар	group1	group2
0	стол	офис	дерево
1	тетрадь	офис	бумага
2	горшок	дом	пластик
3	стук kv-15	дом	пластик

Больше – генерация признаков

Создание описательных признаков

Интеграция данных (Data Integration)

Обычно – из разных источников

	клиент	дата	договор
0	1001	12.01.05	20050047
1	1002	14.01.05	20050054
2	1003	15.01.05	20050058
3	1004	16.01.05	20050065

	клиент	возраст	счёт
0	1001	34	12000.0
1	1002	52	0.0
2	1003	25	10000.0
3	1004	33	NaN

	договор	сумма
0	20050047	100000
1	20050065	200000

	клиент	дата	договор	возраст	счёт	сумма
0	1001	12.01.05	20050047	34	12000.0	100000.0
1	1002	14.01.05	20050054	52	0.0	NaN
2	1003	15.01.05	20050058	25	10000.0	NaN
3	1004	16.01.05	20050065	33	NaN	200000.0

```
df.merge(df2, how='left').merge(df3, how='left')
```

Интеграция данных

Анкета

id	пол	возраст	сумма	карт
12	М	34	10000	0
15	М	23	50000	1
37	Ж	37	90000	2

БКИ

id	дата	сумма	просрочек
12	10-11-12	1000	0
12	01-02-13	2000	1
15	19-10-11	1000	0
15	05-03-12	2000	0
15	03-07-13	3000	1
15	09-09-13	2000	0
37	23-11-13	5000	0

сумма + веса

среднее

максимум

минимум

медиана

Использование интеграции или нет

	user	target
0	1	0
1	3	1
2	6	0
3	7	0
4	8	1

	user	transaction
0	1	10.0
1	1	20.5
2	3	10.4
3	3	18.0
4	3	3.0

Агрегаты

	user	tr_mean	tr_std	tr_max	tr_min	target
0	1	15.25	7.42	20.5	10.0	0
1	3	10.47	7.50	18.0	3.0	1
2	6	9.83	7.52	17.0	2.0	0
3	7	7.25	3.89	10.0	4.5	0
4	8	9.00	12.73	18.0	0.0	1

На уровень транзакций

	user	transaction	target
0	1	10.0	0
1	1	20.5	0
2	3	10.4	1
3	3	18.0	1
4	3	3.0	1

Использование интеграции или нет

```
tmp = data2.groupby('user')['transaction']
tmp = tmp.agg({'tr_mean':mean, 'tr_std':std, 'tr_max':max, 'tr_min':min})
data = data.merge(tmp.reset_index(), on='user')

data2.merge(data, on='user').head()
```

Второй способ, конечно, менее эффективен,
но помогает при ансамблировании.

Нормировки (Data Normalization)

Для большинства алгоритмов машинного обучения необходимо, чтобы все признаки были вещественными и «в одной шкале».

- **Стандартизация**
  (Z-score Normalization / Variance Scaling)

$$\{u_i\}_{i\in I} \to \left\{ \frac{u_i - \text{mean}\{u_t\}_{t\in I}}{\text{std}\{u_t\}_{t\in I}} \right\}_{i\in I}$$

- **Нормировка на отрезок**
  (Min-Max Normalization)

$$\{u_i\}_{i\in I} \to \left\{ \frac{u_i - \min\{u_t\}_{t\in I}}{\max\{u_t\}_{t\in I} - \min\{u_t\}_{t\in I}} \right\}_{i\in I}$$

- **Нормировка по максимуму**

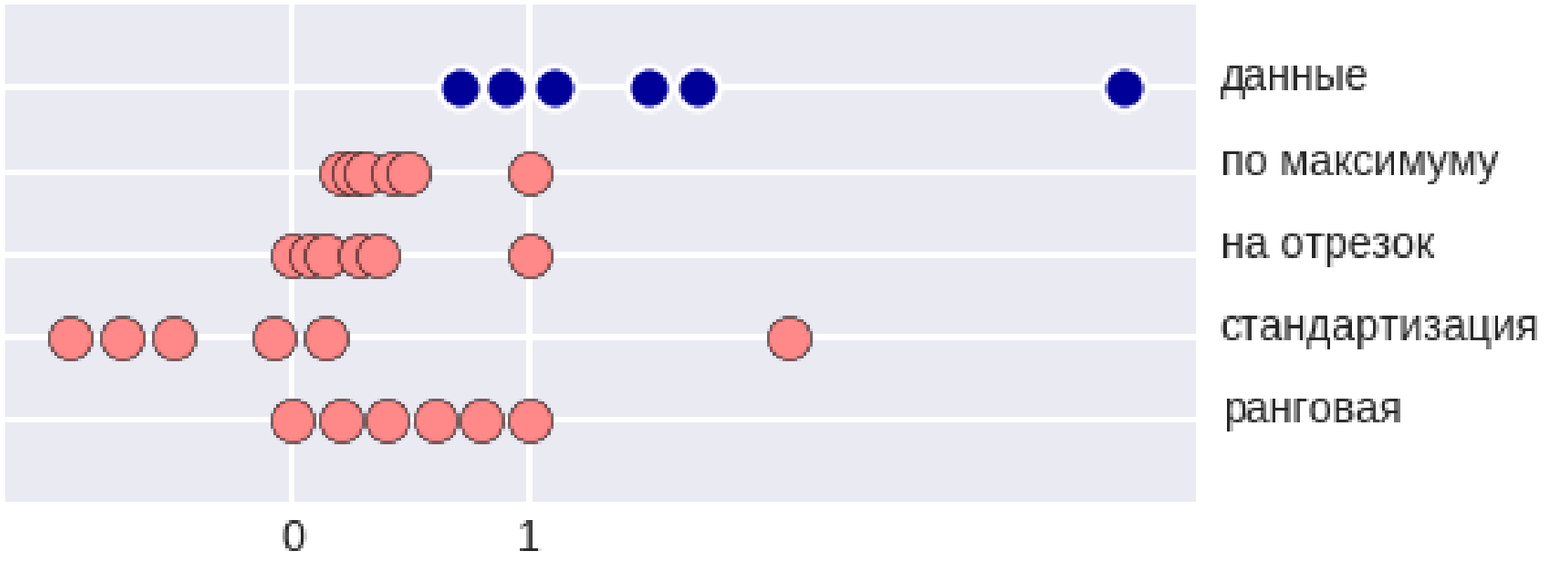
$$\{u_i\}_{i\in I} \to \left\{ \frac{u_i}{\max\{u_t\}_{t\in I}} \right\}_{i\in I}$$

- **Decimal Scaling Normalization**

$$\text{N}_{\text{ds}}(x) = \frac{x}{10^{\min\{i\,:\,10^i > x\}}}$$

- **Ранговая нормировка**
  (tiedrank, rankdata)

Нормировки – реализация




```
X = X / np.max(X)

X = X - np.min(X)
X = X / np.max(X)

X = X - np.mean(X)
X = X / np.std(X)

from scipy.stats import rankdata
X = rankdata(X, method='average')
```

```
X = X - np.min(X)
X = X / np.max(X)

import sklearn.preprocessing as prp

X['minmax'] = prp.minmax_scale(X['name'])
X['standart'] =
preproc.StandardScaler().fit_transform(X[['name']])
# требует df
```

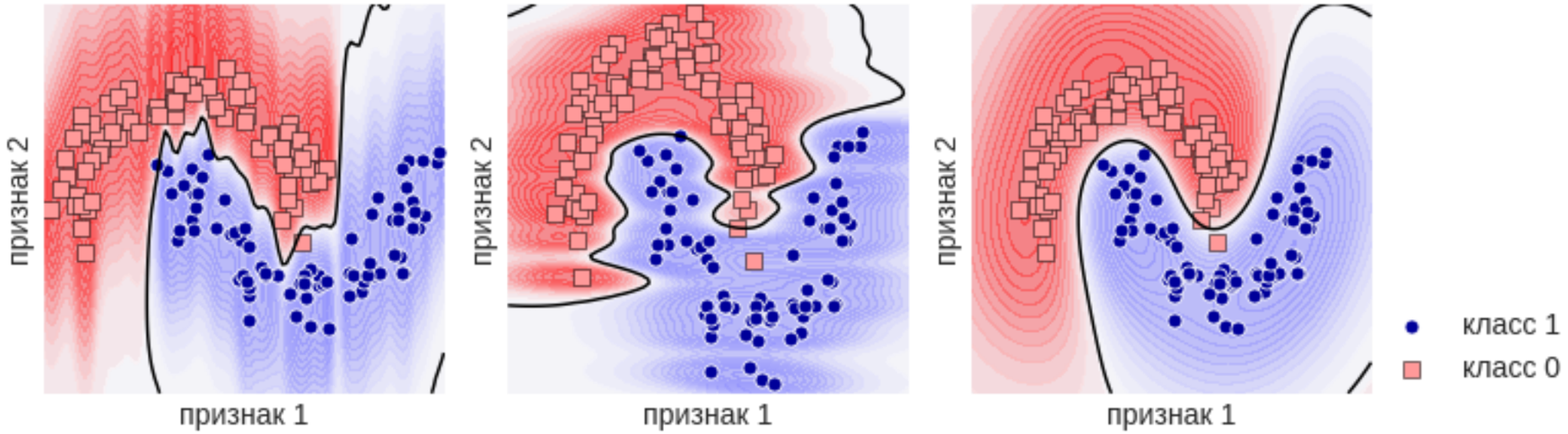
Ранговая нормировка

	feat	average	min	max	dense	ordinal
0	1	1.5	1	2	1	1
1	2	4.0	3	5	2	3
2	2	4.0	3	5	2	4
3	5	6.0	6	6	3	6
4	2	4.0	3	5	2	5
5	1	1.5	1	2	1	2
6	10	7.0	7	7	4	7

```
import scipy.stats as ss

for method in ['average', 'min', 'max', 'dense', 'ordinal']:
    data[method] = ss.rankdata(data.feat, method=method)
```

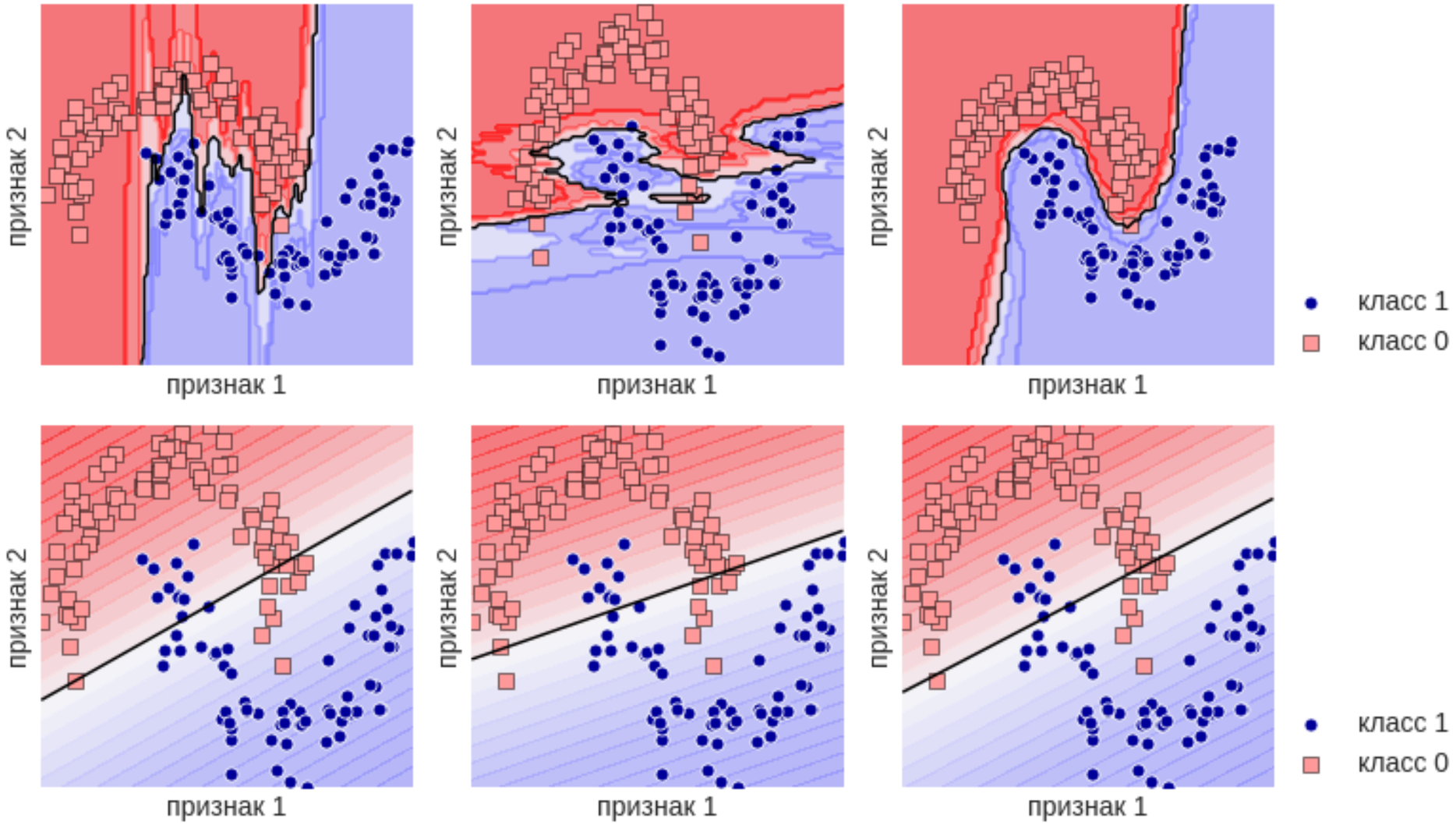
Нормировки – обоснование

Зависимость моделей от масштаба



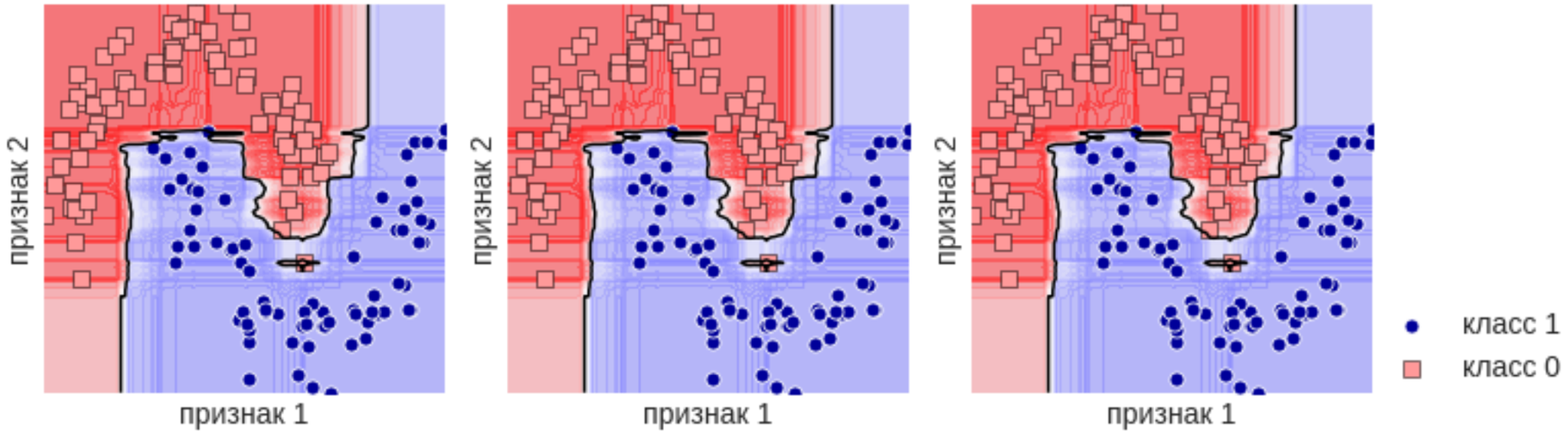
«признак 1» × 10, «признак 2» × 10, нормальный масштаб

Нормировки – обоснование



ДЗ Что за модели? Нет ли ошибки?

Нормировки – обоснование



Модели по-разному реагируют на изменение масштаба по признакам

Нормировки – тонкости

Как всегда: предобработка + классификация – общий пайплайн

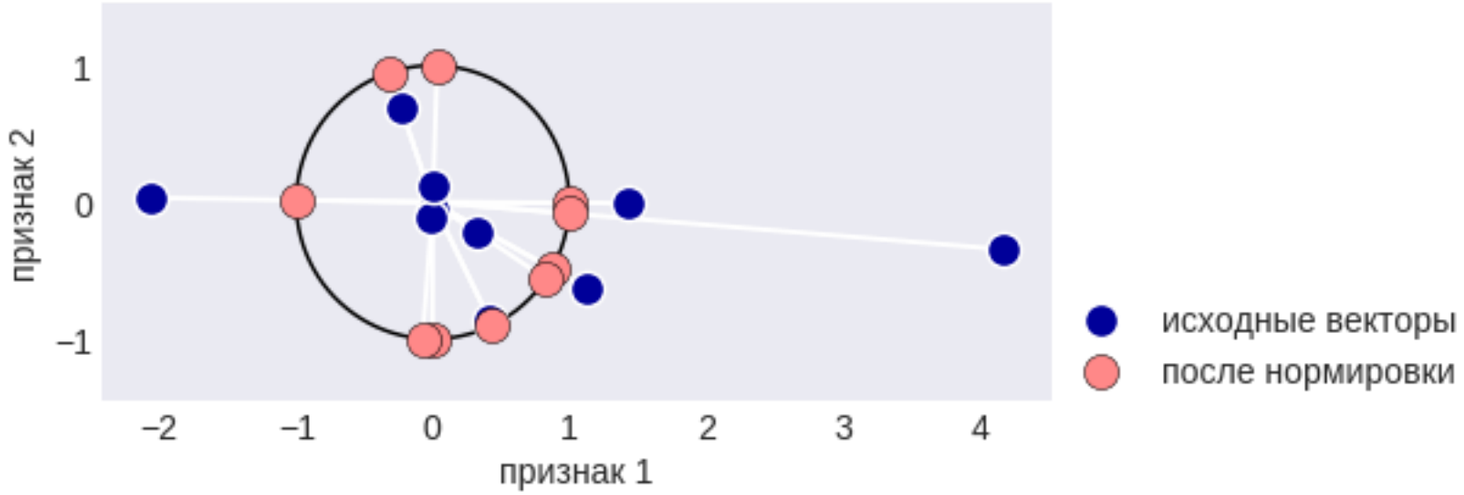
параметры нормировок вычисляются по выборке
использовать все данные не всегда корректно

если вычислить параметры на обучении,
то на контроле может не быть желаемого эффекта
может выйти за пределы [0, 1]

ex: rankdata

Нормировки (Data Normalization)

Если данные имеют смысл векторов (признаки однородны),
то нормировки векторов




```
import sklearn.preprocessing as preproc
nrm = preproc.Normalizer()
X2 = nrm.fit_transform(X)
```

тонкости:

- **не центрируйте разреженные данные**

Нормировки в пределах группы

```
z_score = lambda x: (x - x.mean()) / x.std()
df['stand'] = df.groupby('gr').transform(z_score)
```

	gr	sum	stand
0	alpha	1	0.000000
1	beta	4	1.120897
2	alpha	0	-1.000000
3	beta	0	-0.800641
4	beta	1	-0.320256
5	alpha	2	1.000000

Нормировка

Приведение всех признаков «в одну шкалу»: k-NN, k-means, SVM

Хотя формально это слабо помогает –
нужен адекватный масштаб

Пример: регуляризация временных рядов

Трансформация

Box-Cox Transformation положительного признака

$$f_\lambda(x) = \begin{cases} \dfrac{x^\lambda - 1}{\lambda}, & \lambda > 0, \\ \ln(x), & \lambda = 0. \end{cases}$$

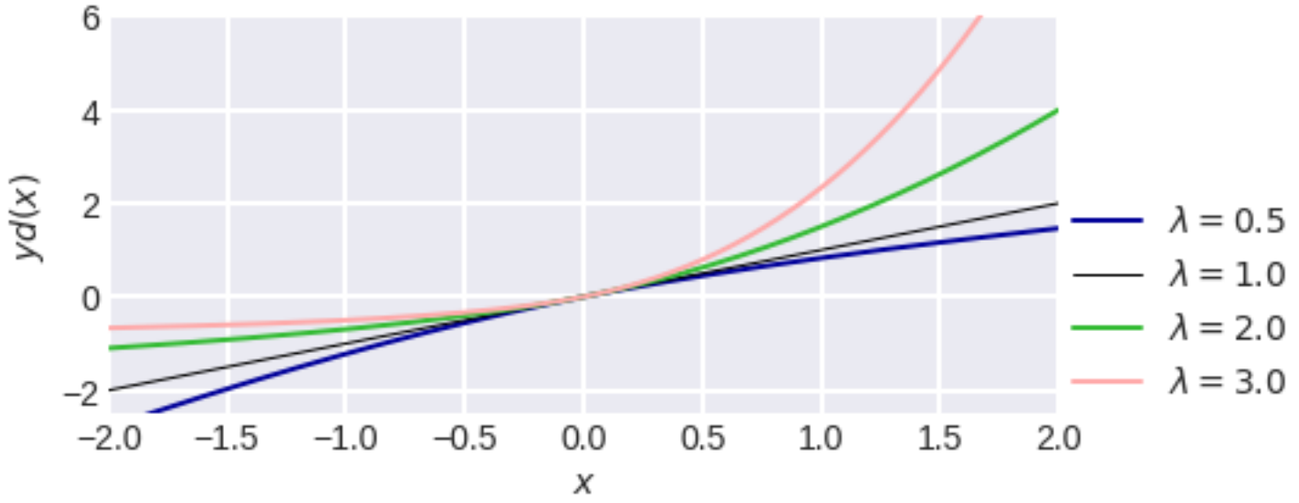
Как правило применяют, чтобы распределение признака стало похожим на нормальное

можно оценивать схожесть оценённого распределения преобразованных данных и нормального

```
x2 = np.log1p(x) # с предварительным +1
x2 = np.log(x)
```

Трансформация

Преобразование Йео-Джонсона

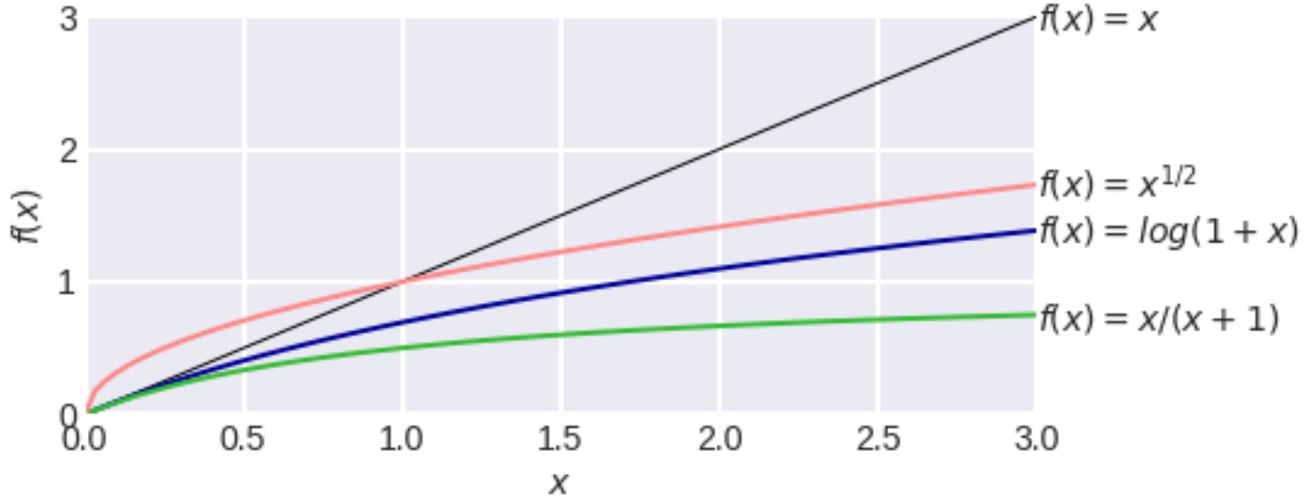


$$\mathrm{yd}_\lambda(x) = \begin{cases} \dfrac{(x+1)^\lambda - 1}{\lambda}, & \lambda \neq 0, x \geq 0, \\ \log(x+1), & \lambda = 0, x \geq 0, \\ -\log(-x+1), & \lambda = 2, x < 0, \\ \dfrac{(-x+1)^{2-\lambda} - 1}{\lambda - 2}, & \lambda \neq 2, x < 0. \end{cases}$$

и для отрицательных x

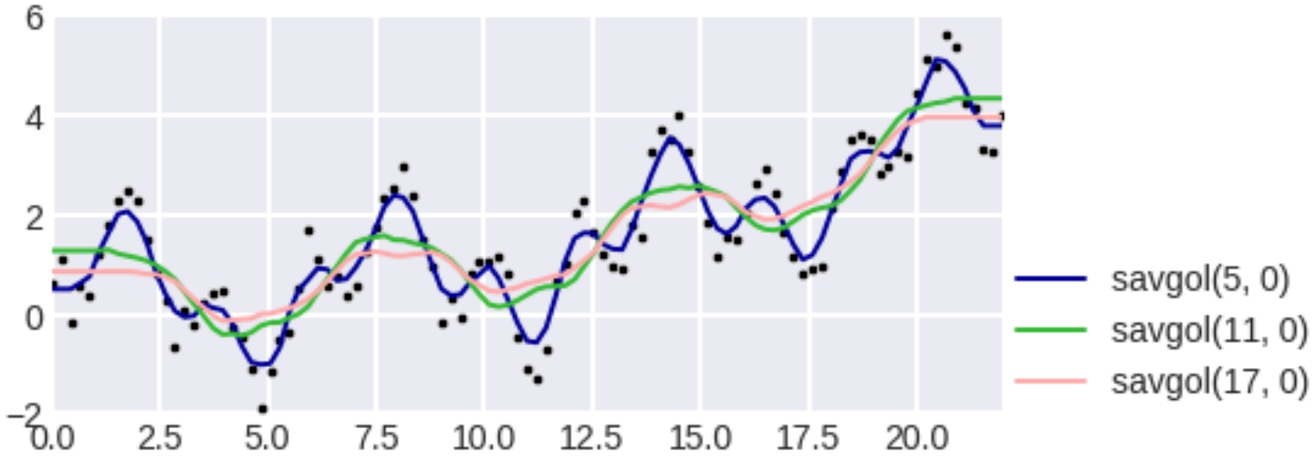
Трансформация

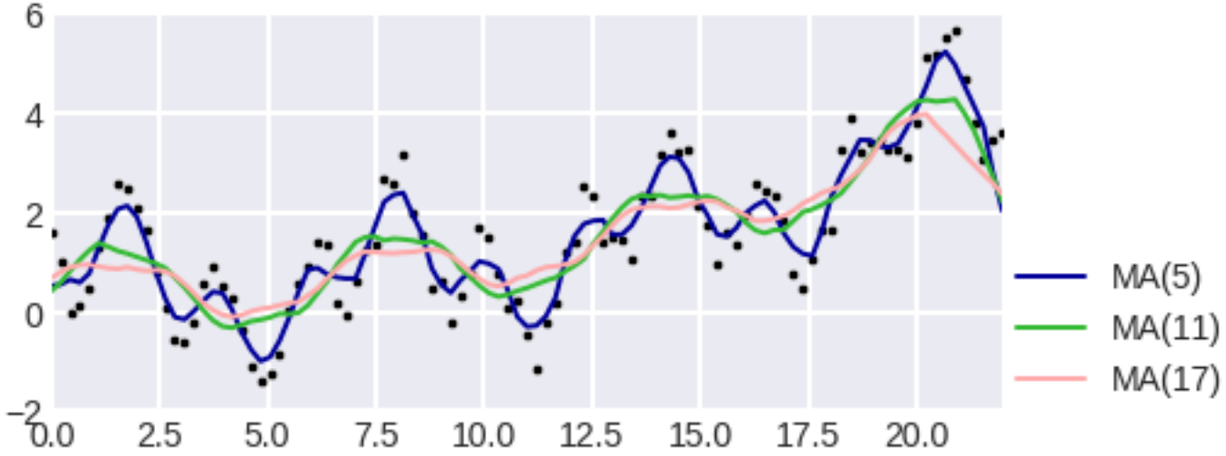
совет: попробовать функцию $x/(x+1)$



Сглаживание (Smoothing)

Moving Average



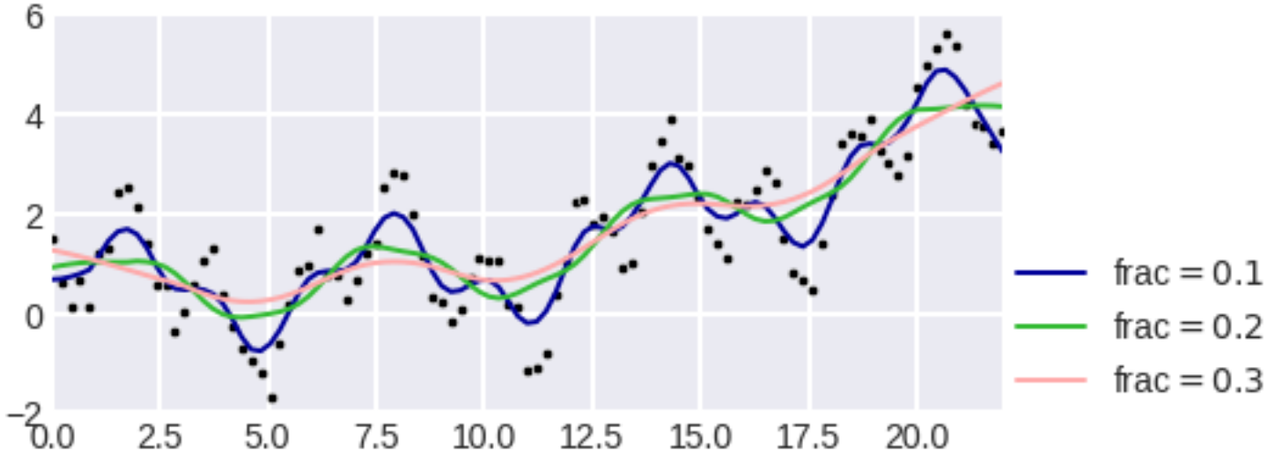



```
def smooth(y, box_pts):
    box = np.ones(box_pts)/box_pts
    y_smooth = np.convolve(y, box,
                           mode='same')
    return y_smooth

y_MA = smooth(y, 5)
```

см. на артефакты
верхний график – начало,
нижний – конец!

Сглаживание (Smoothing)

LOWESS = locally weighted scatterplot smoothing




```
import statsmodels.api as sm
lowess = sm.nonparametric.lowess(y, x,
                                 frac=0.1)[:1]
```

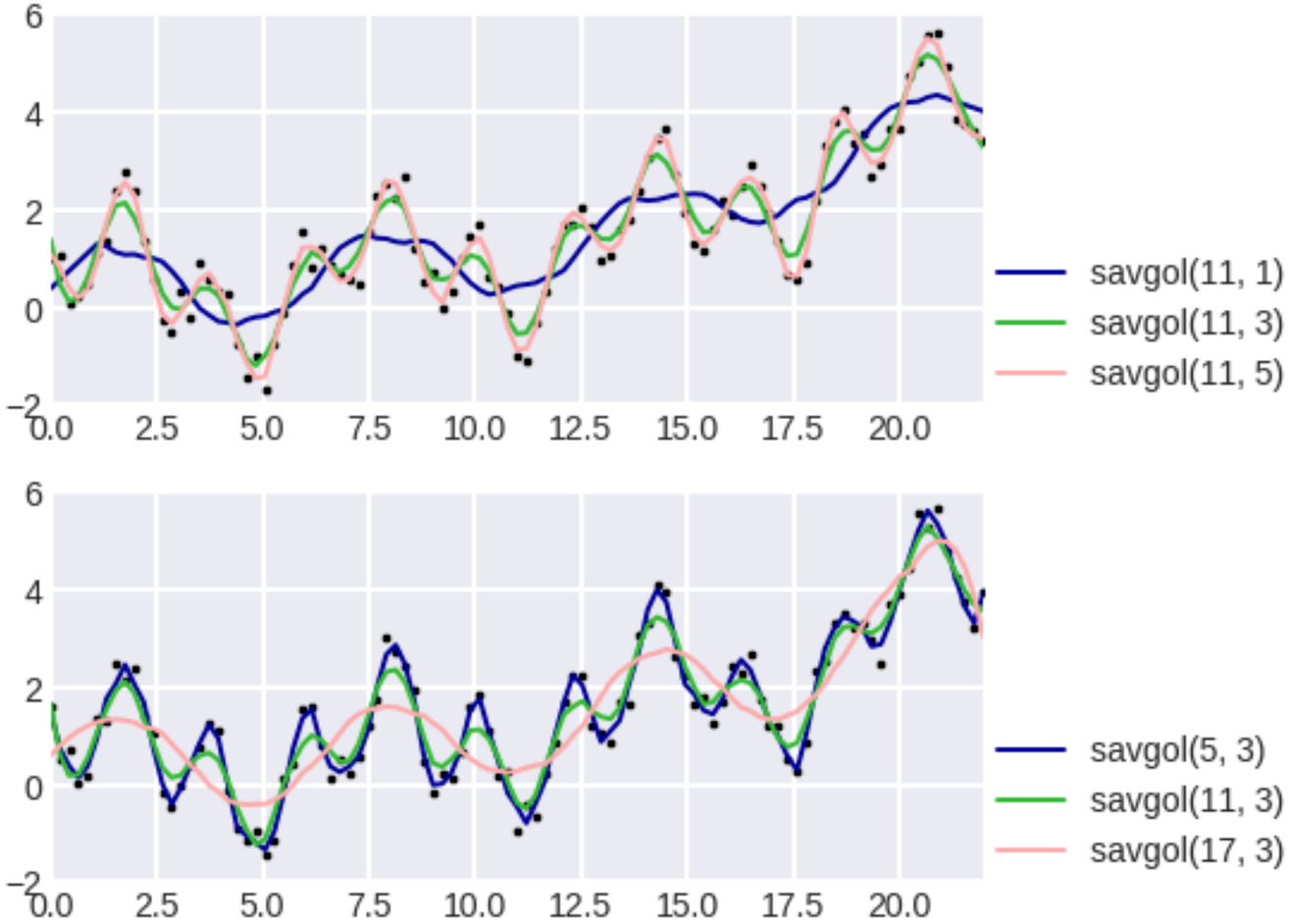
также называют Savitzky–Golay filter

– подгоняют полином определённой степени на окрестности

эквивалентно взвешенному среднему

Сглаживание (Smoothing)

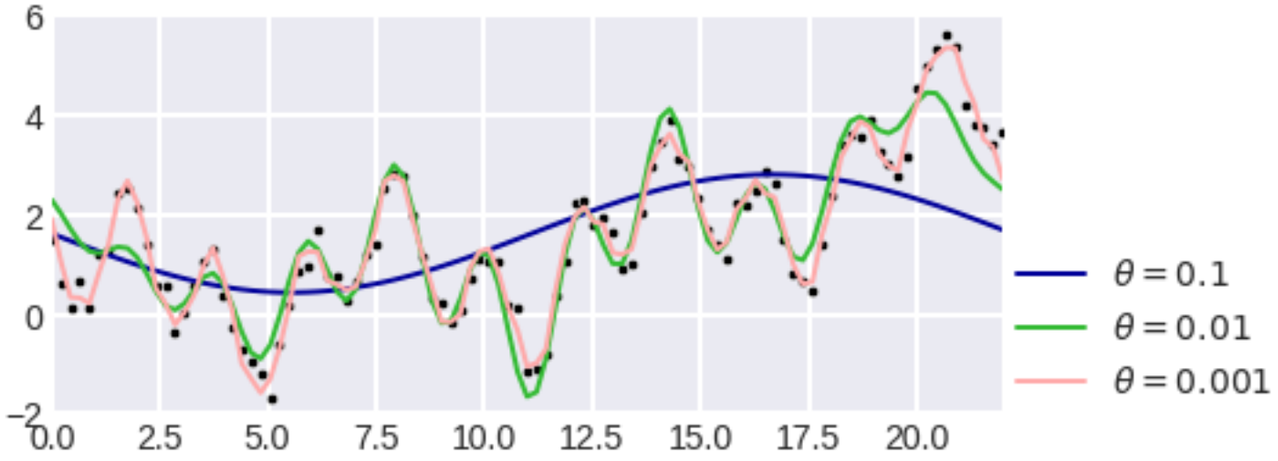
Savitzky–Golay filter




```
from scipy.signal import savgol_filter
y_sg = savgol_filter(y, 11, 2)
```

Сглаживание (Smoothing)

FFT




```
import scipy.fftpack

N = len(y)
w = scipy.fftpack.rfft(y)
spectrum = w**2

theta = 0.1
w2 = w.copy()
w2[spectrum < (theta*spectrum.max())] = 0
y2 = scipy.fftpack.irfft(w2)
```

Можно просто: заменять выбросы (далеко от сглаживания) на сглаженные значения!

Ещё способы:

регрессия обычная / локальная

например, kernel-regression

```
.rolling (window=20).median()
```

Дискретизация (биннинг / Binning, квантование / Quantization)

переход от вещественного признака к порядковому за счёт кодирования интервалов одним значением.

доход от 0 до 10000, от 10000 до 25000, от 25000 до 50000 и т.д.

+ Улучшает интерпретацию,
+ Позволяет решать задачу простыми алгоритмами
(качество ухудшается)

```
bins = pd.cut(df[name], 5)

points = [0, 12, 18, 25, 50, 100]
labels = ['ребёнок', 'юноша', 'молодой
человек', 'мужчина', 'пожилой']
factors = pd.cut(ages, points,
labels=labels)
factors.describe()
```

	counts	freqs
categories		
ребёнок	8	0.16
юноша	9	0.18
молодой человек	7	0.14
мужчина	26	0.52
пожилой	0	0.00

Способы дискретизации

- **Equal-width (distance) partitioning**

Делим область значения признаков на области-интервалы равной длины м.б. в другой шкале!

0 – 9, 10 – 99, 100 – 999

```
factors = pd.cut(ages, 4)
```

categories	counts	freqs
(5.964, 15.0]	14	0.200000
(15.0, 24.0]	21	0.300000
(24.0, 33.0]	15	0.214286
(33.0, 42.0]	20	0.285714

- **Equal-depth (frequency, Quantile-based) partitioning**

Делим область значения признаков на области-интервалы: в каждую попало одинаковое число точек.

```
factors = pd.qcut(ages, 4)
```

categories	counts	freqs
(5.999, 17.0]	19	0.271429
(17.0, 25.0]	16	0.228571
(25.0, 34.0]	19	0.271429
(34.0, 42.0]	16	0.228571

- **Кластеризация**
- **Экспертно**

Способы дискретизации: Equal-width (distance) partitioning

$$\{x : hi \le x < h(i+1)\} \to i$$

+ простая быстрая реализация

– неравномерные бины

```
x2 = np.round(x).astype(int)
```

сюда же округление! Почему нужно?

есть средства бинаризации:

```
from sklearn.preprocessing import Binarizer
bn = Binarizer(threshold=0.9)
new = bn.transform(old)
```

Способы дискретизации: Equal-depth (Frequency, Quantile-based) partitioning

+ равномерные бины

```
X['name'].quantile([.2, .4, .6, .8]) # пороги-квантили
x2 = pd.qcut(x, 10, labels=False) # квантильная дискретизация
```

Способы дискретизации: Кластеризация

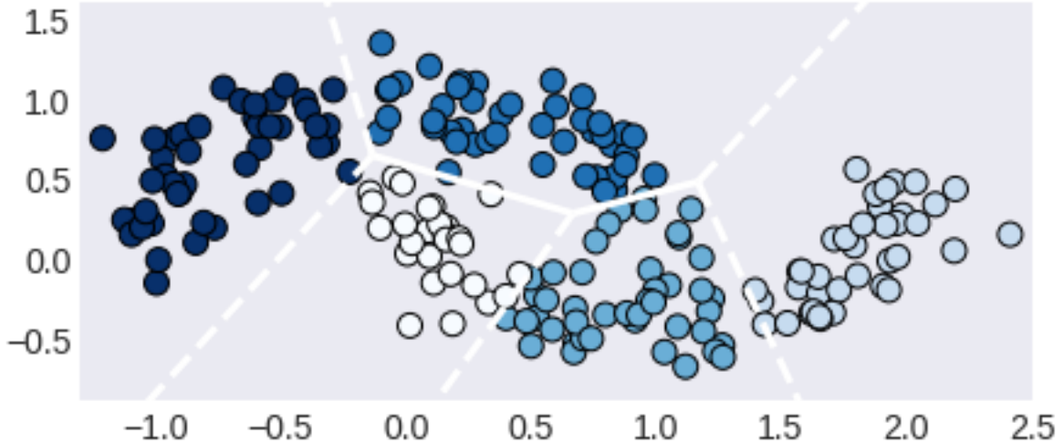

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=4, random_state=0)
a  = model.fit_predict(x)
c = np.sort(model.cluster_centers_[:, 0])
c = (c[1:] + c[:-1]) / 2
c = np.concatenate([min(x), c, max(x)])
```

Способы дискретизации

Если не угадать с кластеризацией…

Способы дискретизации:

Кластеризация



Если есть группа однородных признаков, то можно с помощью кластеризации получить новый категориальный признак «номер кластера»

Или «расстояние до центра кластера»

Дискретизация (биннинг, Binning)

Способы кодирования при дискретизации:

- первым / последним значением
- средним (арифметическим, медианой и т.п)
- номером бина

Пример

1, 1, 1, 2, 2, 4, 4, 7, 8, 9

1, 1, 1, 1, 1, + 4, 4, 4, 4, 4

2, 2, 2, 2, 2, + 9, 9, 9, 9, 9

1.4, 1.4, 1.4, 1.4, 1.4, + 6.4, 6.4, 6.4, 6.4, 6.4

1, 1, 1, 1, 1, + 7, 7, 7, 7, 7

1, 1, 1, 1, 1, + 2, 2, 2, 2, 2

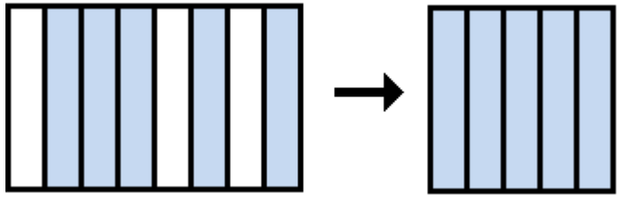
ДЗ Написать кодировщик бинов

Сокращение данных (Data Reduction)

– уменьшение объёма исходных данных, сохраняя полезную информацию

- **отбор признаков (Feature Selection)**

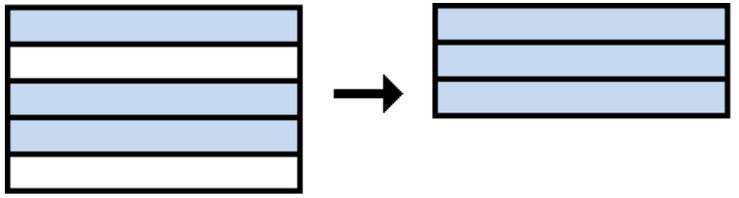
  отдельная тема **next**



- **отбор объектов (Instance Selection)**

  редко используется,

  как правило, по анализу или экспертами

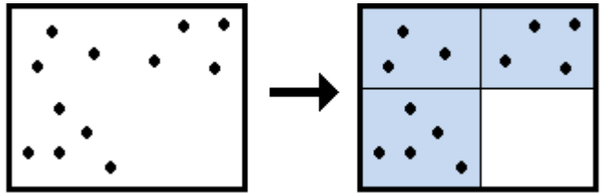


Удаление дубликатов, «пустых» данных!

- **дискретизация, огрубление информации (Discretization)**

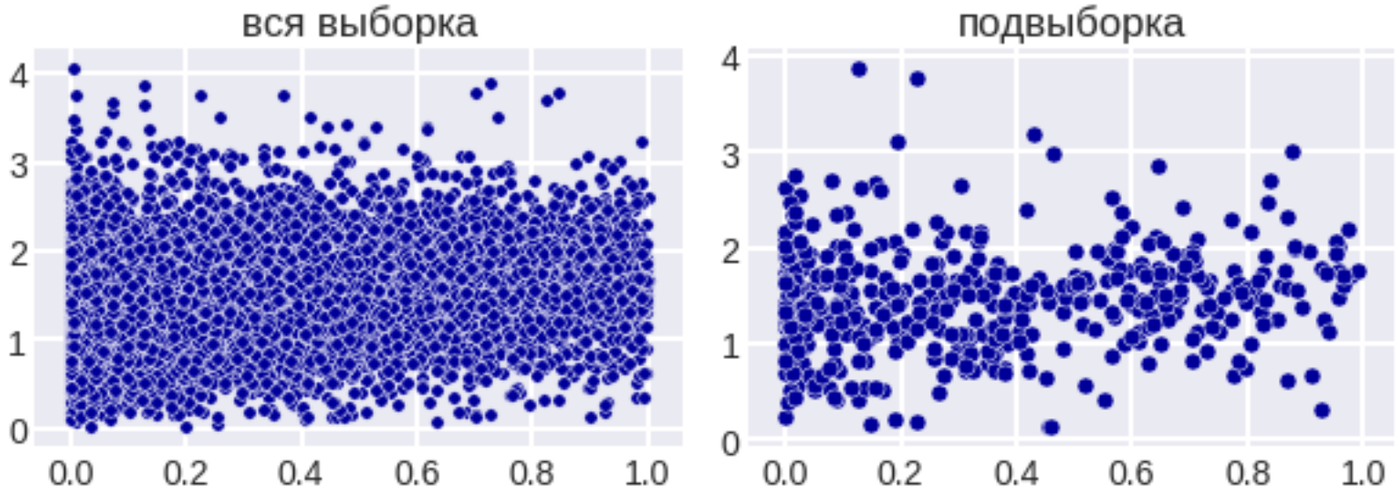
  увеличение шага дискретизации

  перевод вещественных признаков в дискретные



Сокращение данных (Data Reduction)

- **сэмплированое (Sampling)**



- **сокращение размерности (Dimensionality reduction)**
  - факторный анализ (factor analysis)
  - метод главных компонент (PCA), SVD, случайные проекции
  - нелинейные модели: LLE, ISOMAP
  - многомерное шкалирование (MDS)

Сокращение данных (Data Reduction)

цели

- **удаление лишних (нерелевантных) данных**
- **повышение качества решения задачи**
- **уменьшение стоимости данных**
- **увеличение скорости последующего анализа**

(в частности, настройки моделей)

- **повышение интерпретируемости моделей**

Сокращение данных – удаление дубликатов

```
df['is_dup'] = df.duplicated(subset=['date', 'sum'],
                             keep='first')
df.drop_duplicates(subset=['date', 'sum'], keep='first')
```

	date	sum	k	is_dup
0	2012-01-01	55	1.2	False
1	2012-02-03	117	4.3	False
2	2012-01-01	55	1.5	True
3	2012-02-03	117	0.2	True

	date	sum	k	is_dup
0	2012-01-01	55	1.2	False
1	2012-02-03	117	4.3	False

тонкости

- **дубликаты могут быть по подмножеству признаков**

(ex: есть шумовой признак)

- **как с пропусками и выбросами:**

факт дублирования м.б. важен (лучше установить причину)

Совет: смотреть данные, отсортированные по отдельным признакам

Сокращение данных – удаление дубликатов

По числу уникальных значений, можно догадаться, что категориальные признаки равны

	a	b	c	d	e
0	1	0.05	B	1.12	0.76
1	1	0.76	B	1.12	0.96
2	2	0.05	F	0.78	0.79
3	1	0.56	B	1.12	0.96
4	3	0.47	A	1.09	0.15
5	2	0.22	F	0.78	0.79
6	2	0.69	F	0.78	0.66
7	3	0.59	A	1.09	0.25
8	2	0.43	F	0.78	0.79

```
2    4
1    3
3    2
Name: a, dtype: int64
0.05    2
0.56    1
0.43    1
0.47    1
0.76    1
0.22    1
0.59    1
0.69    1
Name: b, dtype: int64
```

```
F    4
B    3
A    2
Name: c, dtype: int64
0.78    4
1.12    3
1.09    2
Name: d, dtype: int64
0.79    3
0.96    2
0.15    1
0.76    1
0.66    1
0.25    1
Name: e, dtype: int64
```

Сокращение данных – сэмплирование

- **Без возвратов** (Simple random sampling without replacement)
- **С возвратами (**Simple random sampling with replacement)
- **Балансированное** (Balanced sampling) – сэмплирование при котором подвыборка будет удовлетворять некоторому заранее заданному условию (например, 90% описаний будет соответствовать пациентам старше 60 лет)
- **Кластерное** (Cluster sampling) – предварительно данные разбиваются на кластеры и выбирается поднабор кластеров (м.б. поднабор из каждого).
- **Стратифицированное** (Stratified sampling) – предварительно данные разбиваются на кластеры, в каждом кластере отдельно осуществляется сэмплирование, таким образом в подвыборку попадают представители всех кластеров.

Для более быстрого поиска оптимальных параметров.
Составляющая часть алгоритма (RF)
Для получения выборки, обладающей специальными свойствами.

Сэмплирование

может производиться с предварительно заданными вероятностями

~ сложность классификации объектов –

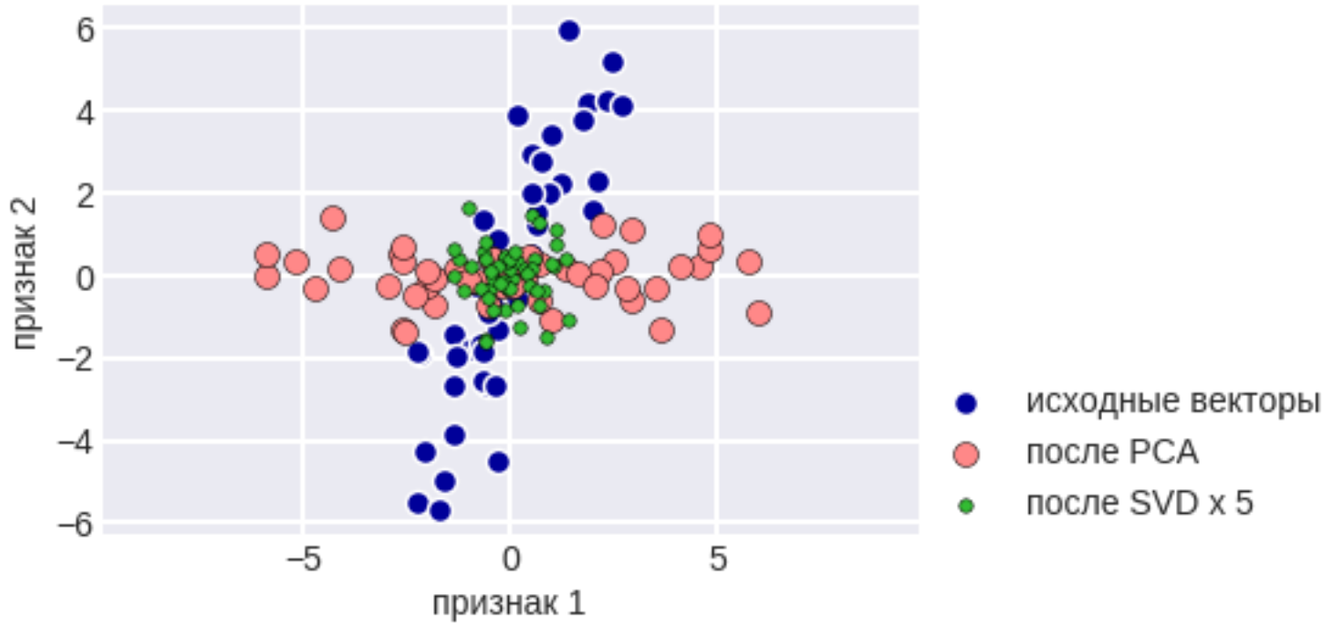
для получения «трудных» выборок

~ веса в функциях ошибок

~ свежесть данных / доверие к данным

Сокращение размерности (Dimensionality reduction)

PCA




```
from sklearn.decomposition import PCA
pca_transformer = PCA()
X2 = pca_transformer.fit_transform(X)

X = X - X.mean(axis=0)
U, L, V = svd(X)
```

Сокращение размерности (Dimensionality reduction)

Изначальный размер изображения 300×451

```
from numpy.linalg import svd
U, L, V = svd(image)
k = 5
plt.imshow(U[:,:k].dot(np.diag(L[:k]).dot(V[:k,:])), cmap=plt.cm.gray)
```

Тонкости: Truncated SVD, с точностью до знака

ДЗ как ещё использовать SVD в изображениях?

После загрузки до разбиения train / test

- удаление служебных переменных и переменных, содержащих утечки (информацию о целевом векторе или из будущего)

Для категориальных: уникальных значений ~ число объектов,
одно уникальное значение
какие-то признаки, которые в будущем не будем собирать

- преобразование типов данных
- предобработка строк (общий регистр), устранение дубликатов
- обработка редких категорий (если не использовать данные)
- заполнение пропусков (если не использовать данные, т.е. константами)
- генерация признаков (если не использовать данные)

После разбиения на обучение и контроль

- масштабирование признаков (в том числе стандартизация), деформации, OHE
- обработка редких категорий
- заполнение пропусков
- генерация признаков

Итог

- **Предобработка данных нужна,**
чтобы данные обладали желаемым свойством

- **Может производиться с учётом модели**
(особенно, если тесно связана с конструированием признаков) **дальше**

- **качественные данные $\Rightarrow$ качественная модель**
очень трудозатратно (м.б. 90% времени)

- **Главный вопрос: «Почему в данных есть это?»**

Ссылки

Fraboni Ec «Data preprocessing»

https://www.slideshare.net/FraboniEc/data-preprocessing-61426734

Статья в блоге «Python и Pandas: делаем быстрее»

https://dyakonov.org/2019/09/23/python-и-pandas-делаем-быстрее/