ДИСЦИПЛИНА **Интеллектуальные системы и технологии**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **информационных технологий**

КАФЕДРА **корпоративных информационных систем**

полное наименование кафедры

ВИД УЧЕБНОГО МАТЕРИАЛА **Лекция**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **Демидова Лилия Анатольевна**

(фамилия, имя, отчество)

СЕМЕСТР **1 семестр (осенний), 2024 – 2025 учебный год**

(семестр обучения, учебный год)

ЛЕКЦИЯ 2

ПРЕДОБРАБОТКА ДАННЫХ

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

Пакет sklearn.preprocessing предоставляет общие служебные функции и классы для преобразования необработанных векторов объектов в представление, более подходящее для последующего использования.

Алгоритмы обучения выигрывают от стандартизации набора данных.

Если в наборе присутствуют некоторые выбросы, целесообразно использовать надежные скейлеры (scalers) или трансформаторы (transformers).

Стандартизация.
Удаление (обнуление) среднего и дисперсионное масштабирование

Стандартизация наборов данных является общим требованием для многих алгоритмов машинного обучения, реализованных в scikit-learn.

Алгоритмы могут вести себя плохо, если отдельные объекты не выглядят более или менее как стандартные нормально распределенные данные: гауссовское с нулевым средним и единичной дисперсией.

На практике мы часто игнорируем форму распределения и просто преобразуем данные для их центрирования, удаляя среднее значение каждого объекта, а затем масштабируем его, деля значения признаков объектов на их стандартное отклонение.

Например, многие элементы, используемые в целевой функции алгоритма обучения (например, ядро RBF в SVM-алгоритме или регуляризаторы l1 и l2 линейных моделей), предполагают, что все объекты сосредоточены вокруг нуля и имеют дисперсию того же порядка.

Если у признака есть отклонение, которое на несколько порядков больше, чем у других, оно может доминировать в целевой функции и сделать алгоритм неспособным обучаться на других признаках правильно, как ожидалось.

Функциональное масштабирование:

```
from sklearn import preprocessing
import numpy as np
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])
X_scaled = preprocessing.scale(X_train)
X_scaled
```

```
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Преобразованные данные имеют нулевое математическое ожидание и дисперсию, равную единице.

```
X_scaled.mean(axis=0)
array([0., 0., 0.])
```

```
X_scaled.std(axis=0)
array([1., 1., 1.])
```

Модуль предварительной обработки дополнительно предоставляет служебный класс StandardScaler, который реализует Transformer API интерфейс для вычисления среднего значения и стандартного отклонения в обучающем наборе, чтобы впоследствии иметь возможность повторно применить то же преобразование к тестовому набору.

```
scaler = preprocessing.StandardScaler().fit(X_train)
scaler
StandardScaler()
```

```
scaler.mean_
array([1. ..., 0. ..., 0.33...])
```

```
scaler.scale_
array([0.81..., 0.81..., 1.24...])
```

```
scaler.transform(X_train)
array([[ 0.  ..., -1.22...,  1.33...],
```

```
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Затем экземпляр скейлера можно использовать для новых данных, чтобы преобразовать их так же, как в обучающем наборе:

```
X_test = [[-1., 1., 0.]]
scaler.transform(X_test)
array([[-2.44...,  1.22..., -0.26...]])
```

Можно отключить центрирование или масштабирование, передав конструктору StandardScaler: `with_mean=False` или `with_std=False`.

Масштабирование признаков в заданный диапазон

Альтернативная стандартизация – масштабирование объектов таким образом, чтобы они находились между заданным минимальным и максимальным значением, часто между нулем и единицей, или таким образом, чтобы максимальное абсолютное значение каждого объекта масштабировалось до размера единицы.

Это может быть достигнуто с помощью MinMaxScaler или MaxAbsScaler, соответственно.

Мотивация для использования этого масштабирования включает устойчивость к очень маленьким стандартным отклонениям признаков и сохранение нулевых записей в разреженных данных.

Масштабирование в [0, 1]

```
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])

min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_train_minmax
array([[0.5       , 0.        , 1.        ],
       [1.        , 0.5       , 0.33333333],
       [0.        , 1.        , 0.        ]])
```

Такой экземпляр преобразователя может быть применен к новым тестовым данным: будут применены те же операции масштабирования и сдвига, чтобы соответствовать преобразованию, выполненному для данных обучающей выборки:

```
X_test = np.array([[-3., -1.,  4.]])
X_test_minmax = min_max_scaler.transform(X_test)
X_test_minmax

array([[-1.5        ,  0.         ,  1.66666667]])
```

Можно проанализировать атрибуты скейлера, чтобы узнать о точном характере трансформации на обучающих данных:

```
min_max_scaler.scale_
array([0.5        , 0.5        , 0.33...])

min_max_scaler.min_
array([0.         , 0.5        , 0.33...])
```

Если в MinMaxScaler задан явный feature_range = (min, max), то формула имеет вид:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) -
X.min(axis=0))

X_scaled = X_std * (max - min) + min
```

MaxAbsScaler работает похожим образом, но масштабирует так, чтобы данные обучения находились в диапазоне [– 1, 1] путем деления на наибольшее максимальное значение в каждом признаке. Он предназначен для данных, которые уже центрированы на нулевых или разреженных данных.

```
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])
```

```
max_abs_scaler = preprocessing.MaxAbsScaler()
X_train_maxabs = max_abs_scaler.fit_transform(X_train)
X_train_maxabs
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -0.5]])

X_test = np.array([[ -3., -1.,  4.]])
X_test_maxabs = max_abs_scaler.transform(X_test)
X_test_maxabs
array([[-1.5, -1. ,  2. ]])

max_abs_scaler.scale_
array([2.,  1.,  2.])
```

Нормализация

Нормализация – это процесс масштабирования отдельных объектов для получения единичной нормы.

Этот процесс может быть полезен, если мы планируете использовать квадратичную форму, такую как точка-произведение или любое другое ядро, для количественного определения сходства любой пары объектов.

Это предположение является основой модели векторного пространства, часто используемой в контексте классификации текста и кластеризации.

Функция normalize обеспечивает быстрый и простой способ выполнения этой операции, используя нормы l1 или l2:

```
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
X_normalized = preprocessing.normalize(X, norm='l2')

X_normalized
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])
```

Модуль предварительной обработки дополнительно предоставляет служебный класс Normalizer, который реализует ту же операцию с использованием Transformer API интерфейса.

```
normalizer = preprocessing.Normalizer().fit(X)  # fit
does nothing
normalizer
Normalizer()
```

Экземпляр нормализатора можно использовать для векторов объектов в качестве преобразователя:

```
normalizer.transform(X)
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])

normalizer.transform([[-1.,  1., 0.]])
array([[-0.70...,  0.70...,  0.  ...]])
```

Кодирование категориальных признаков

Часто признаки представлены в виде категориальных значений (а не непрерывных).

Например, человек может иметь признаки:

`["male", "female"], ["from Europe", "from US", "from Asia"], ["uses Firefox", "uses Chrome", "uses Safari", "uses Internet Explorer"]`.

Такие признаки могут быть эффективно закодированы как целые числа, например,

`["male", "from US", "uses Internet Explorer"]` можно записать как `[0, 1, 3]`, а `["female", "from Asia", "uses Chrome"]` можно записать как `[1, 2, 1]`.

Чтобы преобразовать категориальные признаки в такие целочисленные коды, мы можем использовать OrdinalEncoder.

Он преобразует каждый категориальный признак в новый, значениями в котором являются целые числа (от 0 до n_categories – 1):

```
enc = preprocessing.OrdinalEncoder()
```

```
X = [['male', 'from US', 'uses Safari'], ['female',
'from Europe', 'uses Firefox']]
enc.fit(X)
OrdinalEncoder()

enc.transform([['female', 'from US', 'uses Safari']])
array([[0., 1., 1.]])
```

Такое целочисленное представление, однако, не может использоваться непосредственно со всеми средствами scikit-learn, поскольку они ожидают непрерывного ввода и интерпретируют категории как упорядоченные, что часто нежелательно (например, набор браузеров был упорядочен произвольно).

Еще одна возможность преобразовать категориальные признаки в признаки, которые можно использовать со средствами scikit-learn, – это использовать «одно из K» («one of K»), также известное как «горячее» или «фиктивное» кодирование (one-hot or dummy encoding).

Этот тип кодирования может быть получен с помощью OneHotEncoder, который преобразует каждый категориальный признак с n_categories возможными значениями в n_categories двоичные признаки, с одной «1» и всеми остальными – «0».

```
enc = preprocessing.OneHotEncoder()
X = [['male', 'from US', 'uses Safari'], ['female',
'from Europe', 'uses Firefox']]
enc.fit(X)
OneHotEncoder()
enc.transform([['female', 'from US', 'uses Safari'],
               ['male', 'from Europe', 'uses Safa-
ri']]).toarray()
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

По умолчанию значения, которые может принимать признак, автоматически выводятся из набора данных и могут быть найдены в атрибуте Categories_:

```
enc.categories_
[array(['female', 'male'], dtype=object), array(['from
Europe', 'from US'], dtype=object), array(['uses Fire-
fox', 'uses Safari'], dtype=object)]
```

Это можно указать явно, используя категории параметров. В нашем наборе данных есть два пола, четыре возможных континента и четыре веб-браузера:

```
genders = ['female', 'male']
locations = ['from Africa', 'from Asia', 'from Europe',
'from US']
browsers = ['uses Chrome', 'uses Firefox', 'uses IE',
'uses Safari']
enc = preprocessing.OneHotEncoder(categories=[genders,
locations, browsers])
# Note that for there are missing categorical values
for the 2nd and 3rd
# feature
X = [['male', 'from US', 'uses Safari'], ['female',
'from Europe', 'uses Firefox']]
enc.fit(X)
OneHotEncoder(categories=[['female', 'male'],
                          ['from Africa', 'from Asia',
'from Europe',
                           'from US'],
                          ['uses Chrome', 'uses Fire-
fox', 'uses IE',
                           'uses Safari']])

enc.transform([['female', 'from Asia', 'uses
Chrome']]).toarray()
array([[1., 0., 0., 1., 0., 0., 1., 0., 0., 0.]])
```

Если существует вероятность того, что в обучающих данных могут отсутствовать значения категориальных признаков, часто лучше указывать handle_unknown = 'ignore' вместо ручной установки категорий.

Если указано handle_unknown = 'ignore' и во время преобразования обнаруживаются неизвестные категории, ошибки не будут возникать, но результирующие столбцы с горячим кодированием для этого признака будут иметь все нули (handle_unknown = 'ignore' поддерживается только для горячего кодирования)):

```
enc = preprocessing.OneHotEncoder(handle_unknown='ignore')
X = [['male', 'from US', 'uses Safari'], ['female',
'from Europe', 'uses Firefox']]
enc.fit(X)
OneHotEncoder(handle_unknown='ignore')

enc.transform([['female', 'from Asia', 'uses
Chrome']]).toarray()
array([[1., 0., 0., 0., 0., 0.]])
```

Также возможно закодировать каждый столбец в (n_categories – 1) столбцов вместо n_categories столбцов, используя параметр drop.

Этот параметр позволяет пользователю указать категорию для каждого признака, которая будет удалена. Это полезно, чтобы избежать коллинеарности входной матрицы в некоторых классификаторах. Такая функциональность полезна, например, при использовании нерегулярной регрессии (LinearRegression), поскольку совместная линейность может привести к тому, что ковариационная матрица будет необратимой.

Когда этот параметр не None, handle_unknown должен быть установлен в error:

```
X = [['male', 'from US', 'uses Safari'], ['female',
'from Europe', 'uses Firefox']]
drop_enc = prepro-
cessing.OneHotEncoder(drop='first').fit(X)
drop_enc.categories_
[array(['female', 'male'], dtype=object), array(['from
Europe', 'from US'], dtype=object), array(['uses Fire-
fox', 'uses Safari'], dtype=object)]

drop_enc.transform(X).toarray()
array([[1., 1., 1.],
       [0., 0., 0.]])
```

Сравнительный анализ влияния разных скейлеров на данные с выбросами

Набор данных: California housing dataset
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

«Мы собрали информацию о переменных, используя все группы кварталов в Калифорнии, полученные нами в 1990-х годах. В этой выборке группа кварталов в среднем включает 1425,5 человек, проживающих в географи-

чески совместной зоне. Естественно, что географический район изменяется обратно пропорционально плотности населения. Мы вычислили расстояния между центроидами каждой группы кварталов, измеренные по широте и долготе. Мы исключили все группы кварталов, дающие нулевые записи для независимых и зависимых переменных. Окончательные данные содержали 20 640 наблюдений по 9 переменным. Зависимой переменной является ln (медианное значение для домохозяйства)».

	Bols	tols
INTERCEPT	11.4939	275.7518
MEDIAN INCOME	0.4790	45.7768
MEDIAN INCOME2	-0.0166	-9.4841
MEDIAN INCOME3	-0.0002	-1.9157
ln(MEDIAN AGE)	0.1570	33.6123
ln(TOTAL ROOMS/ POPULATION)	-0.8582	-56.1280
ln(BEDROOMS/ POPULATION)	0.8043	38.0685
ln(POPULATION/ HOUSEHOLDS)	-0.4077	-20.8762
ln(HOUSEHOLDS)	0.0477	13.0792

Признак 0 (медианный доход в блоке, median income in a block) и признак 5 (число домохозяйств, number of households) набора данных о жилье в Калифорнии имеют очень разные масштабы и содержат некоторые очень большие выбросы.

Эти две характеристики приводят к затруднениям при визуализации данных и, что более важно, они могут ухудшать прогнозирующую производительность многих алгоритмов машинного обучения.

Немасштабированные данные также могут замедлять или даже исключать сходимость многих градиентных оценок.

Многие алгоритмы разработаны с допущением, что каждый признак принимает значения, близкие к нулю, или, что более важно, все признаки варьируются в сопоставимых масштабах.

В частности, оценки на основе метрик и градиентов часто предполагают приблизительно стандартизированные данные (центрированные объекты с единичными отклонениями).

Исключением являются основанные на дереве решений алгоритмы, которые устойчивы к произвольному масштабированию данных.

В этом примере используются разные скейлеры (Scalers), преобразователи (Transformers) и нормализаторы (Normalizers), чтобы привести данные в заранее заданный диапазон.

Скейлеры (масштаберы) являются линейными (или точнее аффинными) преобразователями и отличаются друг от друга способом оценки параметров, используемых для сдвига и масштабирования каждого признака.

QuantileTransformer обеспечивает нелинейные преобразования, в которых расстояния между крайними выбросами (outliers) и типичными значениями (inliers) сокращаются.

PowerTransformer обеспечивает нелинейные преобразования, в которых данные отображаются в нормальное распределение, чтобы стабилизировать дисперсию и минимизировать асимметрию.

В отличие от предыдущих преобразований, нормализация относится к преобразованию для каждого объекта вместо преобразования для каждого признака.

```
# Author:  Raghav RV <rvraghav93@gmail.com>
#          Guillaume Lemaitre <g.lemaitre58@gmail.com>
#          Thomas Unterthiner
# License: BSD 3 clause

import numpy as np

import matplotlib as mpl
from matplotlib import pyplot as plt
from matplotlib import cm

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import minmax_scale
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import PowerTransformer

from sklearn.datasets import fetch_california_housing
```

```
print(__doc__)

dataset = fetch_california_housing()
X_full, y_full = dataset.data, dataset.target

# Take only 2 features to make visualization easier
# Feature of 0 has a long tail distribution.
# Feature 5 has a few but very large outliers.

X = X_full[:, [0, 5]]

distributions = [
    ('Unscaled data', X),
    ('Data after standard scaling',
        StandardScaler().fit_transform(X)),
    ('Data after min-max scaling',
        MinMaxScaler().fit_transform(X)),
    ('Data after max-abs scaling',
        MaxAbsScaler().fit_transform(X)),
    ('Data after robust scaling',
        RobustScaler(quantile_range=(25,
75)).fit_transform(X)),
    ('Data after power transformation (Yeo-Johnson)',
     PowerTransformer(method='yeo-
johnson').fit_transform(X)),
    ('Data after power transformation (Box-Cox)',
     PowerTransformer(method='box-
cox').fit_transform(X)),
    ('Data after quantile transformation (gaussian
pdf)',
        QuantileTransform-
er(output_distribution='normal')
        .fit_transform(X)),
    ('Data after quantile transformation (uniform
pdf)',
        QuantileTransform-
er(output_distribution='uniform')
        .fit_transform(X)),
    ('Data after sample-wise L2 normalizing',
        Normalizer().fit_transform(X)),
]

# scale the output between 0 and 1 for the colorbar
y = minmax_scale(y_full)
```

```
# plasma does not exist in matplotlib < 1.5
cmap = getattr(cm, 'plasma_r', cm.hot_r)

def create_axes(title, figsize=(16, 6)):
    fig = plt.figure(figsize=figsize)
    fig.suptitle(title)

    # define the axis for the first plot
    left, width = 0.1, 0.22
    bottom, height = 0.1, 0.7
    bottom_h = height + 0.15
    left_h = left + width + 0.02

    rect_scatter = [left, bottom, width, height]
    rect_histx = [left, bottom_h, width, 0.1]
    rect_histy = [left_h, bottom, 0.05, height]

    ax_scatter = plt.axes(rect_scatter)
    ax_histx = plt.axes(rect_histx)
    ax_histy = plt.axes(rect_histy)

    # define the axis for the zoomed-in plot
    left = width + left + 0.2
    left_h = left + width + 0.02

    rect_scatter = [left, bottom, width, height]
    rect_histx = [left, bottom_h, width, 0.1]
    rect_histy = [left_h, bottom, 0.05, height]

    ax_scatter_zoom = plt.axes(rect_scatter)
    ax_histx_zoom = plt.axes(rect_histx)
    ax_histy_zoom = plt.axes(rect_histy)

    # define the axis for the colorbar
    left, width = width + left + 0.13, 0.01

    rect_colorbar = [left, bottom, width, height]
    ax_colorbar = plt.axes(rect_colorbar)

    return ((ax_scatter, ax_histy, ax_histx),
            (ax_scatter_zoom, ax_histy_zoom,
ax_histx_zoom),
            ax_colorbar)
```

```
def plot_distribution(axes, X, y, hist_nbins=50, ti-
tle="",
                      x0_label="", x1_label=""):
    ax, hist_X1, hist_X0 = axes

    ax.set_title(title)
    ax.set_xlabel(x0_label)
    ax.set_ylabel(x1_label)

    # The scatter plot
    colors = cmap(y)
    ax.scatter(X[:, 0], X[:, 1], alpha=0.5, marker='o',
s=5, lw=0, c=colors)

    # Removing the top and the right spine for aesthet-
ics
    # make nice axis layout
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()
    ax.spines['left'].set_position(('outward', 10))
    ax.spines['bottom'].set_position(('outward', 10))

    # Histogram for axis X1 (feature 5)
    hist_X1.set_ylim(ax.get_ylim())
    hist_X1.hist(X[:, 1], bins=hist_nbins, orienta-
tion='horizontal',
                 color='grey', ec='grey')
    hist_X1.axis('off')

    # Histogram for axis X0 (feature 0)
    hist_X0.set_xlim(ax.get_xlim())
    hist_X0.hist(X[:, 0], bins=hist_nbins, orienta-
tion='vertical',
                 color='grey', ec='grey')
    hist_X0.axis('off')



def make_plot(item_idx):
    title, X = distributions[item_idx]
    ax_zoom_out, ax_zoom_in, ax_colorbar = cre-
ate_axes(title)
    axarr = (ax_zoom_out, ax_zoom_in)
```

```
    plot_distribution(axarr[0], X, y, hist_nbins=200,
                      x0_label="Median Income",
                      x1_label="Number of households",
                      title="Full data")

    # zoom-in
    zoom_in_percentile_range = (0, 99)
    cutoffs_X0 = np.percentile(X[:, 0],
zoom_in_percentile_range)
    cutoffs_X1 = np.percentile(X[:, 1],
zoom_in_percentile_range)

    non_outliers_mask = (
        np.all(X > [cutoffs_X0[0], cutoffs_X1[0]], ax-
is=1) &
        np.all(X < [cutoffs_X0[1], cutoffs_X1[1]], ax-
is=1))
    plot_distribution(axarr[1], X[non_outliers_mask],
y[non_outliers_mask],
                      hist_nbins=50,
                      x0_label="Median Income",
                      x1_label="Number of households",
                      title="Zoom-in")

    norm = mpl.colors.Normalize(y_full.min(),
y_full.max())
    mpl.colorbar.ColorbarBase(ax_colorbar, cmap=cmap,
                              norm=norm, orienta-
tion='vertical',
                              label='Color mapping for
values of y')
```

Для каждого скейлера / нормализатора / трансформатора будут показаны два рисунка.

На левом рисунке будет показан график рассеяния полного набора данных, а на правом рисунке исключены экстремальные значения. Будут учитываться только 99% набора данных (предельные выбросы исключаются).

Кроме того, предельные распределения для каждого признака будут показаны на боковой части графика рассеяния.

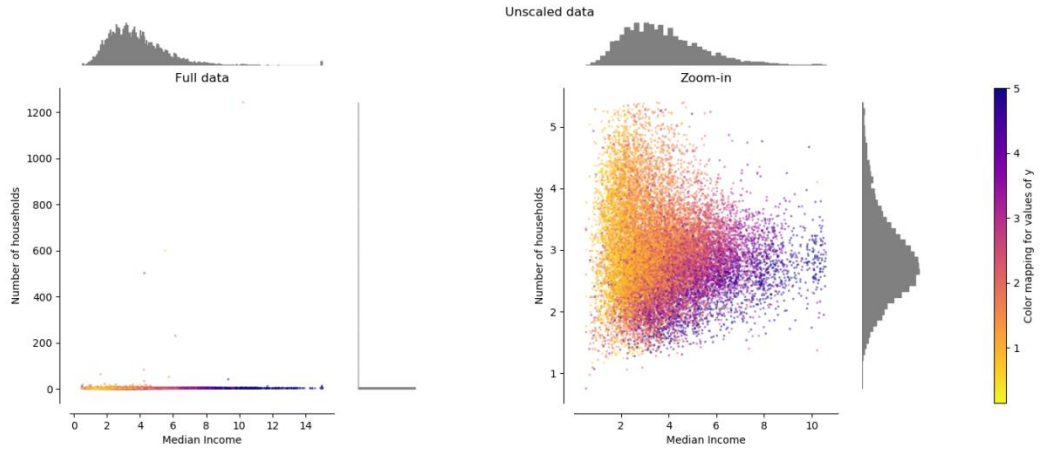
Исходный набор данных

```
make_plot(0)
```

Каждое преобразование нанесено на график с двумя преобразованными признаками, причем на левом графике показан весь набор данных, а справа увеличен масштаб, чтобы показать набор данных без предельных выбросов. Подавляющее большинство объектов уплотняется в определенном диапазоне: [0, 10] для медианного дохода (median income) и [0, 6] для числа домохозяйств (number of households).

Можно заметить, что есть некоторые предельные выбросы (в некоторых кварталах насчитывается более 1200 домохозяйств).

Следовательно, конкретная предварительная обработка может быть очень полезной в зависимости от применения.

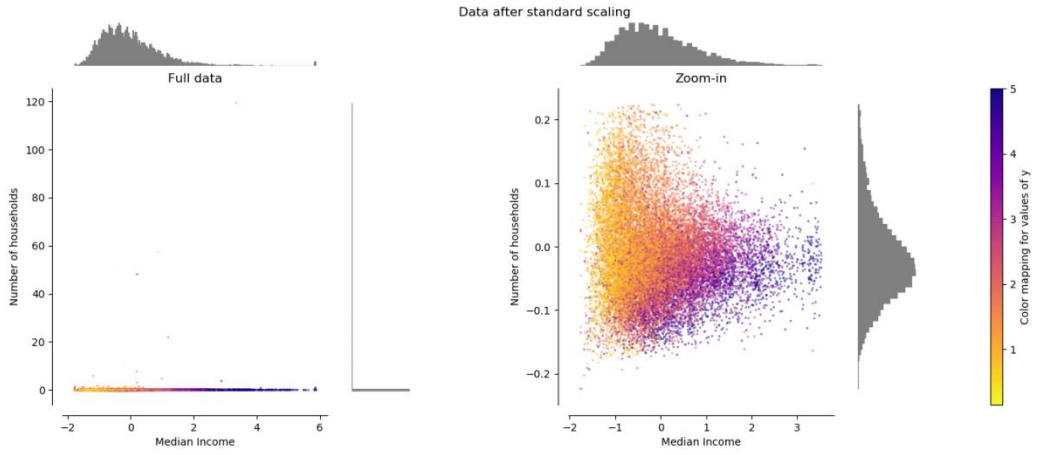


StandardScaler

```
make_plot(1)
```

StandardScaler обнуляет среднее значение и масштабирует данные до единичной дисперсии. Однако выбросы оказывают влияние при вычислении эмпирического среднего и стандартного отклонения, которые сужают диапазон значений признаков, как показано на левом рисунке. В частности, следует отметить, что поскольку выбросы по каждому признаку имеют разные ве-

личины, разброс преобразованных данных по каждому признаку сильно отличается: большинство данных лежат в диапазоне [−2, 4] для преобразованного признака среднего дохода, тогда как одинаковые данные сжимаются в меньшем диапазоне [−0,2, 0,2] для преобразованного числа домохозяйств. Поэтому StandardScaler не может гарантировать сбалансированные масштабы признаков при наличии выбросов.



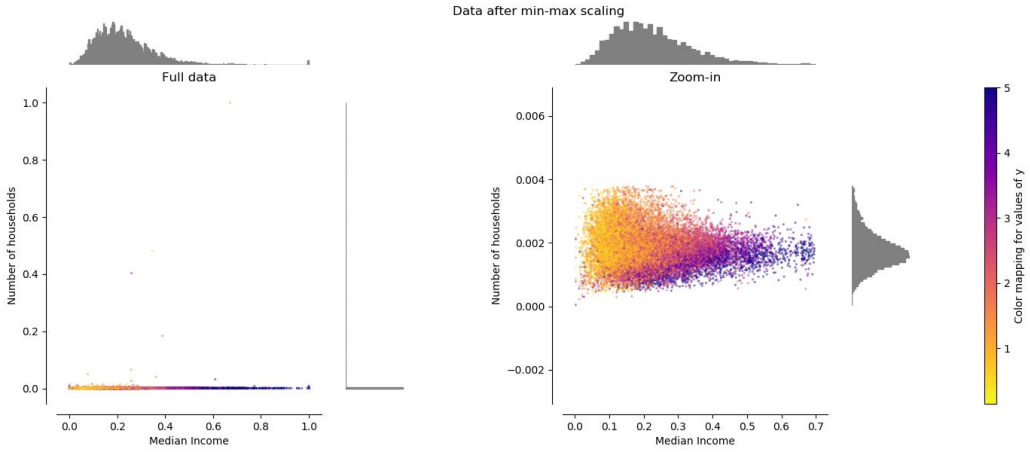
MinMaxScaler

```
make_plot(2)
```

MinMaxScaler изменяет размер набора данных так, что все значения признаков находятся в диапазоне [0, 1], как показано на правом рисунке.

Однако это масштабирование сжимает все значения в узком диапазоне [0, 0,005] для преобразованного числа домохозяйств.

Как и StandardScaler, MinMaxScaler очень чувствителен к наличию выбросов.

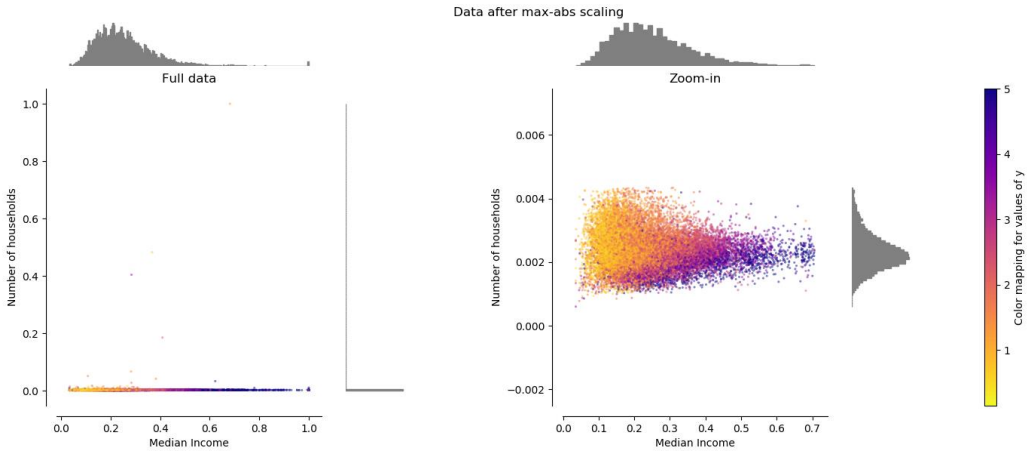


MaxAbsScaler

```
make_plot(3)
```

MaxAbsScaler отличается от предыдущего скейлера тем, что абсолютные значения отображаются в диапазоне [0, 1].

Только для положительных данных этот скейлер ведет себя подобно MinMaxScaler и, следовательно, также страдает от наличия больших выбросов.



RobustScaler

```
make_plot(4)
```

В отличие от предыдущих скейлеров, статистика центрирования и масштабирования этого скейлера основана на процентилях и поэтому не подвержена влиянию нескольких очень больших предельных выбросов.

Следовательно, результирующий диапазон преобразованных значений признаков больше, чем для предыдущих скейлеров, и, что более важно, приблизительно одинаков: для обоих признаков большинство преобразованных значений находятся в диапазоне [−2, 3], как видно в увеличенном масштабе на рисунке. Следует отметить, что сами выбросы все еще присутствуют в преобразованных данных. Если желательно отдельное ограничение выбросов, требуется нелинейное преобразование.

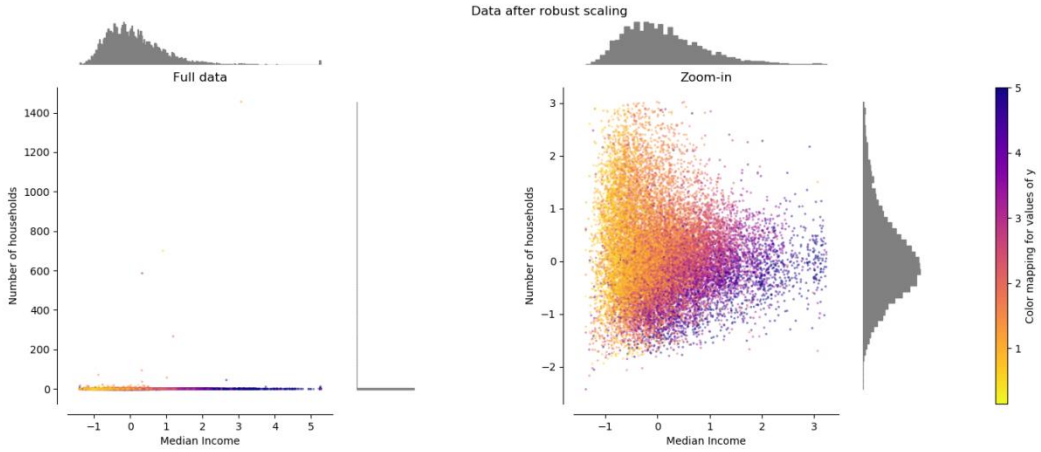
Процентиль − мера, в которой процентное значение общих значений равно этой мере или меньше ее.

Процентиль распределения (этот термин был впервые использован Галтоном в 1885 г.) – это такое число x_p, что значения p-й части совокупности меньше или равны x_p.

Например, 25-я процентиль (также называемая квантилью .25 или нижней квартилью) переменной – это такое значение (x_p), что 25% (p) значений переменной попадают ниже этого значения.

Аналогичным образом вычисляется 75-я процентиль (также называемая квантилью .75 или верхней квартилью) – такое значение, ниже которого попадают 75% значений переменной.

90 % значений переменной находятся ниже 90-го процентиля, а 10 % значений переменной находятся ниже 10-го процентиля.

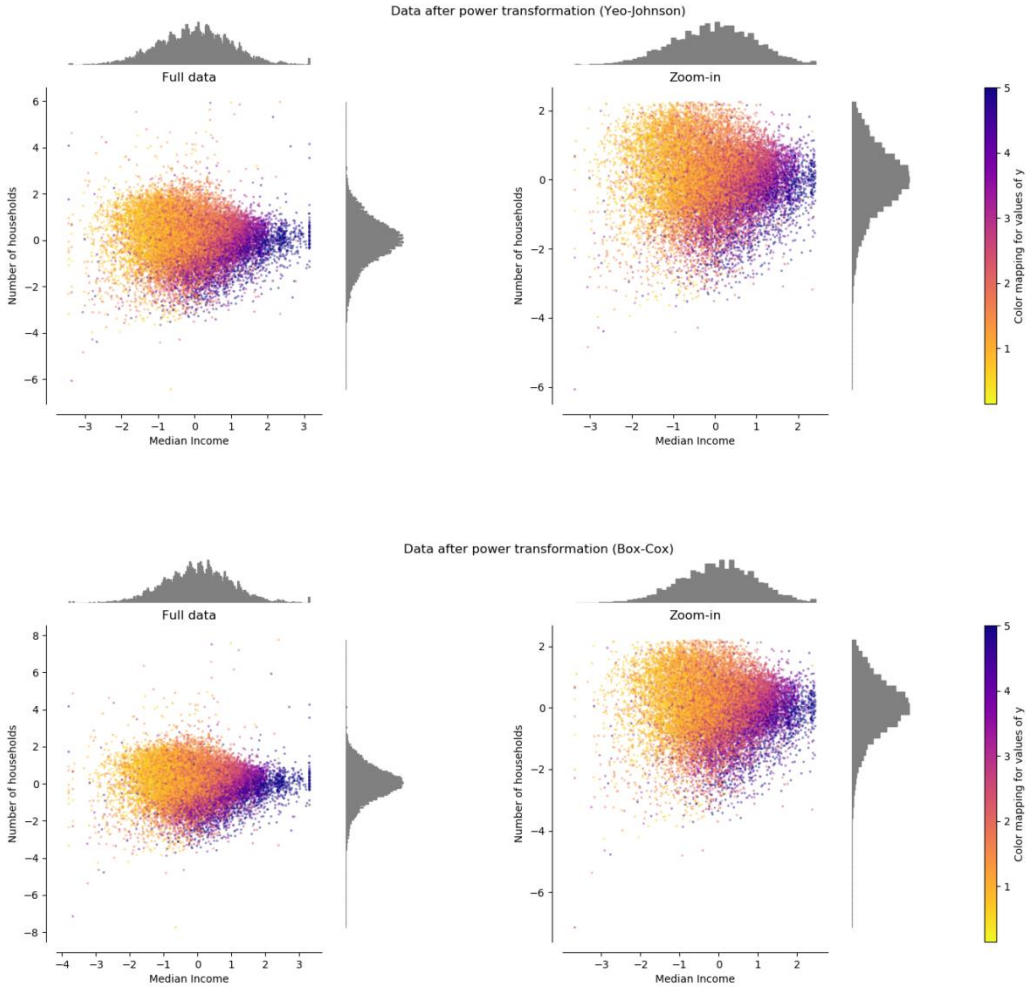


PowerTransformer

```
make_plot(5)
make_plot(6)
```

PowerTransformer применяет степенное преобразование для каждого признака, чтобы сделать данные более похожими на гауссовы. В настоящее время PowerTransformer реализует преобразования Йео-Джонсона и Бокса-Кокса. PowerTransformer находит оптимальный коэффициент масштабирования для стабилизации дисперсии и минимизации асимметрии посредством оценки максимального правдоподобия. По умолчанию PowerTransformer обеспечиваем нулевое среднее и дисперсию, равную единице.

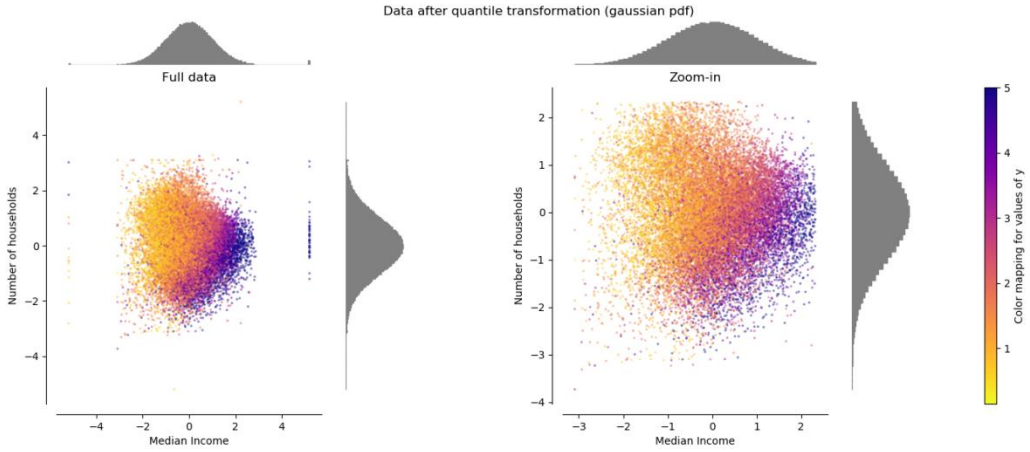
Следует отметить, что преобразование Бокса-Кокса может применяться только к строго положительным данным. Доход и число домохозяйств оказываются строго положительными, но если присутствуют отрицательные значения, преобразование Йео-Джонсона должно быть предпочтительным.



QuantileTransformer (Gaussian output)

```
make_plot(7)
```

QuantileTransformer имеет дополнительный параметр output_distribution, позволяющий применить распределение Гаусса вместо равномерного распределения. Следует отметить, что этот непараметрический преобразователь вводит артефакты насыщения для экстремальных значений.

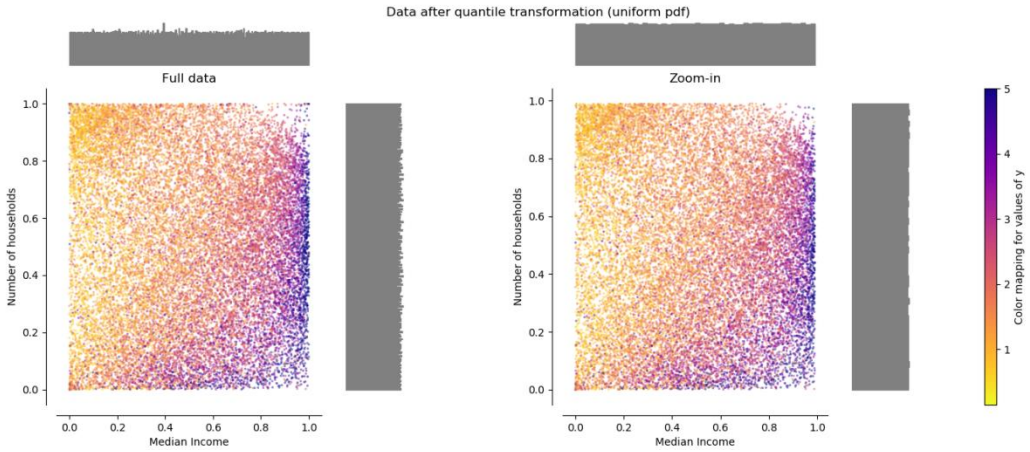


QuantileTransformer (uniform output)

```
make_plot(8)
```

QuantileTransformer применяет нелинейное преобразование, так что функция плотности вероятности каждого признака будет отображаться в равномерное распределение. В этом случае все данные будут отображаться в диапазоне [0, 1], даже выбросы, которые уже нельзя будет отличить от выбросов.

Как и RobustScaler, QuantileTransformer является устойчивым к выбросам в том смысле, что добавление или удаление выбросов в обучающем наборе приведет к примерно одинаковому преобразованию на данных. Но в отличие от RobustScaler, QuantileTransformer также автоматически сворачивает любые выбросы, устанавливая для них предварительно определенные границы диапазона (0 и 1).

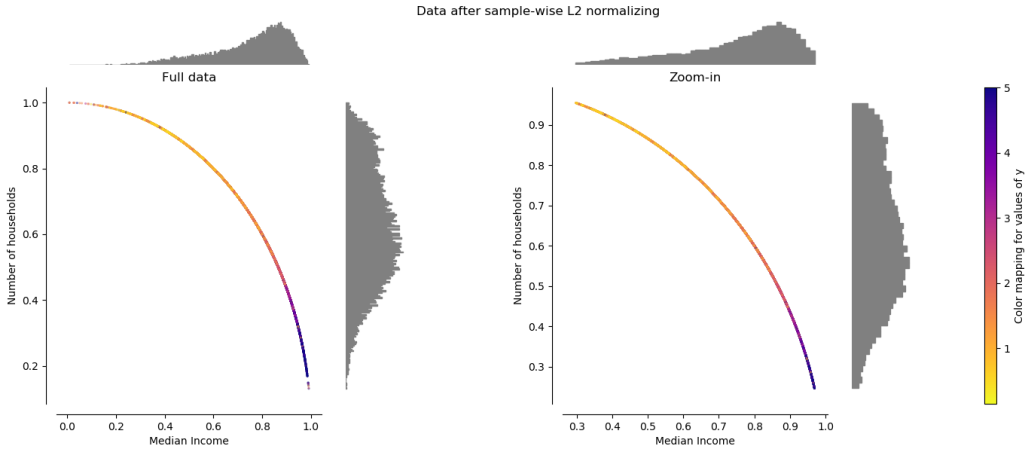


Normalizer

```
make_plot(9)

plt.show()
```

Нормализатор изменяет масштаб вектора для каждого объекта, чтобы иметь единичную норму, независимо от распределения объектов. Это видно на обоих рисунках ниже, где все объекты нанесены на единичную окружность. В нашем примере два выбранных признака имеют только положительные значения; поэтому преобразованные данные лежат только в положительном квадранте. Это было бы не так, если бы некоторые признаки имели сочетание положительных и отрицательных значений.



Проклятие размерности

Проклятие размерности – проблема, связанная с экспоненциальным возрастанием количества данных из-за увеличения размерности пространства.

Проклятие размерности (ПР) – термин, используемый в отношении ряда свойств многомерных пространств и комбинаторных задач.

В первую очередь это касается экспоненциального роста необходимых экспериментальных данных в зависимости от размерности пространства при решении задач вероятностно-статистического распознавания образов, машинного обучения, классификации и дискриминантного анализа.

Также это касается экспоненциального роста числа вариантов в комбинаторных задачах в зависимости от размера исходных данных, что приводит к соответствующему росту сложности переборных алгоритмов.

«Проклятие» действует и на непрерывные оптимизационные методы в силу усложнения многомерной целевой функции. В более широком смысле термин применяется по отношению ко всем «неудобным» или необычным свойствам многомерных пространств и к трудностям их исследования. В комбинаторике чаще имеют в виду не размерность пространства, а размер исходных данных. В частности, в задачах теории Рамсея было бы точнее говорить о «проклятии размера» исходных данных даже в случае задач минимальной размерности – числа параметров, определяющих задачу. Впервые термин «проклятие размерности» был введен Ричардом Беллманом в 1961 году применительно к общей задаче динамического программирования. Это выражение продолжает употребляться в работах по технической кибернетике, машинному обучению и анализу сложных систем, в том числе, в заголовках научных статей.

Проблема «проклятия размерности» часто возникает в машинном обучении, например, при применении метода ближайших соседей и метода парзеновского окна.

«Проклятие размерности» особенно явно проявляется при работе со сложными системами, которые описываются большим числом параметров.

«Проклятие размерности» влечет за собой следующие проблемы:

– трудоемкость вычислений;
– необходимость хранения огромного количества данных;
– увеличение доли шумов.

В линейных классификаторах увеличение числа признаков ведет к проблемам мультиколлинеарности и переобучения.

В метрических классификаторах расстояния обычно вычисляются как средний модуль разностей по всем признакам.

Согласно Закону Больших Чисел, сумма n слагаемых стремится в некоторому фиксированному пределу при $n \to \infty$.

Таким образом, расстояния во всех парах объектов стремятся к одному и тому же значению, а значит, становятся неинформативными.

Примеры.

1. Рассмотрим единичный интервал [0,1].

Возьмем 100 равномерно разбросанных точек в этом интервале. Этого числа точек будет достаточно, чтобы покрыть этот интервал с частотой не менее 0,01.

Теперь рассмотрим 10-мерный куб. Для достижения той же степени покрытия потребуется уже 1020 точек.

По сравнению с одномерным пространством требуется в 1018 раз больше точек.

Поэтому, например, использование переборных алгоритмов становится неэффективным при возрастании размерности системы.

2. Пусть надо восстановить вероятностное распределение простейшей бинарной случайной величины, и с точностью, достаточной для поставленных целей, это можно сделать по выборке из $200 = 2 \cdot 100$ измерений или наблюдений. Тогда для восстановления вероятностного распределения вектора из бинарных случайных величин с той же точностью потребуется выборка из измерений, что часто оказывается неприемлемым по материальным затратам или времени. -мерный вектор бинарных величин имеет возможных значений и попытки прямолинейного восстановления его распределения по экспериментальной выборке бесперспективны.

3. В комбинаторике перебор вариантов на современных компьютерах также невозможен. Поэтому точные решения для NP-трудных и более сложных задач путём перебора можно искать только в случае, когда размер исходных данных исчисляется несколькими десятками вершин графа, требований, приборов и т.п. То, что некоторые игры с полной информацией, например шахматы, по сей день представляют интерес, есть следствие ПР.

4. Нетрудно убедиться, что, если задать любое положительное ε, то доля объёма многомерного куба или шара в ε -окрестности границы стремится к 1 при неограниченном увеличении размерности. Таким образом, в многомерных телах большая часть объёма находится «рядом» с границей. Поэтому точки даже больших экспериментальных выборок, как правило, являются граничными – лежат либо на выпуклой оболочке множества, либо близко от неё.

5. Согласно центральной предельной теореме, вероятность, что два случайных вектора будут нормальны, в пределах любой наперёд заданной положительной ошибки, стремится к 1 с ростом размерности пространства.

6. Какова роль расстояния в метрических алгоритмах? Анализ расстояний должен позволить сделать вывод о похожести/непохожести объектов. Чем больше расстояние, тем более различны объекты.

В задачах машинного обучения приходится работать с объектами, у которых очень много признаков. Каковы особенности у расстояния в пространстве высокой размерности?

К чему могут привести небольшие отличия в большом числе координат?

Пусть есть 3 вектора:

$x_1 = (a_1, a_2, ..., a_n)$,

$x_2 = (a_1 + \varepsilon, a_2 + \varepsilon, ..., a_n + \varepsilon)$,

$x_3 = (a_1, a_2 + \Delta, ..., a_n)$,

где ε – очень маленькое число (таким образом, x_2 отличается от x_1 по каждой координате, но на очень маленькую величину), Δ – очень больше число (таким образом, x_3 отличается от x_1 по одной координате, но на очень большую величину).

Какой из векторов будет ближе к x_1? С одной стороны, всё зависит от метрики. С другой стороны, когда признаков очень много, даже очень ма-

ленькие различия могут накопиться и стать более значимыми, чем одно большое.

7. Когда расстояния будут почти одинаковыми?

Рассмотрим расстояния в n-мерном пространстве до (n+1) равноудаленной точки.

На плоскости – три точки, равноудаленные друг от друга, – это правильный треугольник, в трехмерном пространстве – это правильный тетраэдр, формируемый 4-мя равноудаленными точками. Эти точки не группируются в одной области, в которой, например, лежат точки одного класса.

В n-мерном пространстве – (n+1) точка. Получается, что, когда число признаков сравнимо с числом объектов, объекты могут быть равноудаленными (или почти равноудаленными друг от друга).

8. Экспоненциальный рост данных.

Рассмотрим бинарные векторы размерности n, например, x=(0,1,0,1,…,1).

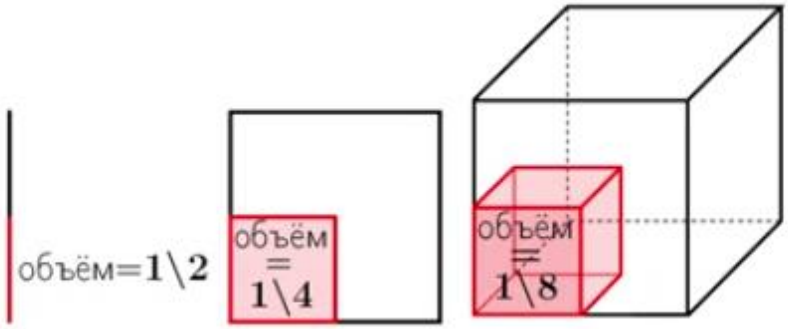
Всего имеем 2^n комбинаций значений признаков. С ростом n число данных (число объектов в обучающей выборке), которое необходимо иметь, чтобы покрыть все возможные ситуации с той же точностью, увеличивается экспоненциально.

Рассмотрим в куб с ребром, равным 1, а в нем – куб с ребром, равным 1/2.

Какую часть от объема большого куба будет составлять маленький куб?

Рассмотрим вероятность попадания в куб с ребром, равным 1.

В одномерном пространстве объем =1/2, в двумерном – 1/4, в трехмерном 1/8 и т.д. В *n*-мерном случае – 1/*n*. Таким образом, объем будет составлять всё меньшую и меньшую долю.



Куб с длиной ребра 0.99 будет составлять 0.99^n часть от объема единичного куба: $\lim_{n\to\infty} 0.99^n = 0$. При *n*, стремящемся к бесконечности, доля объема маленького куба стремиться к нулю!

Это значит, что в пространстве высокой размерности основная часть объема концентрируется вблизи границы области. Получается, что объекты оказываются очень часто примерно одинаково удалены друг от друга.

Способы устранения «проклятия размерности»

В пространстве высокой размерности необходимый объем обучающей выборки растет экспоненциально, что становится серьёзной проблемой для метрических алгоритмов. Объекты часто оказываются на примерно одинаковом расстоянии друг от друга.

Основная идея при решении проблемы – понизить размерность пространства, а именно спроецировать данные на подпространство меньшей размерности. На этой идее, например, основан метод главных компонент.

Кроме того, можно осуществлять отбор признаков и использовать алгоритм вычисления оценок.

В задачах вероятностного распознавания существуют две ситуации, в которых можно преодолеть проклятие размерности с помощью общих подходов.

1. Возможна ситуация, когда распределение вектора взаимозависимых случайных величин сосредоточено в подпространстве меньшей размерности, то есть вектор может быть хорошо приближен линейной функцией меньшего числа переменных. В этом случае метод главных компонент позволяет снизить размерность. Далее поставленные задачи распознавания (дискриминации) могут решаться уже в пространстве малой размерности.

2. Возможна ситуация, когда случайные величины исследуемых векторов независимы или, что более реально, слабо взаимозависимы. В этом случае можно восстановить одномерные распределения случайных величин и построить многомерные распределения как их произведения. Далее, используя ту же обучающую выборку в режиме скользящего окна можно восстановить одномерное распределение отношения функций правдоподобия дифференцируемых классов, что устраняет смещения, связанные с взаимозависимостью в решающем правиле.

Обычно для анализа многомерных данных предлагают использовать метрический метод ближайшего соседа. Достоинство метода состоит в том, что формально он может применяться к выборкам любого объёма независимо от размерности векторов. Но принципиально прикладная проблема ПР состоит в том, что в ограниченной выборке данных недостаточно информации об объекте, описываемом большим числом значимых параметров. И если это описание является действительно сложным и многомерным, а для решения задачи требуется анализ всего комплекса параметров, то проблема оказывается трудной и не решается общими методами.