Лекция 3: признаковое пространство, постановка задачи прогнозирования

Исходные данные

- Объект анализа (или прецедент, или кейс, или наблюдение, …) x из некоторого возможно бесконечного множества объектов X задается набором признаков f_j (или атрибутов, или свойств, …)

$$f_j: X \to D_j,\ x \in X$$

- Домен (область определения, множество значений, тип) признака:
  - Категориальный: D_j как правило конечно, нет расстояния, не задан порядок
  - Ординальный (порядковый): как категориальные, но задан порядок в виде транзитивного, антисимметричного отношения, но нет расстояний

$$R_j: D_j \times D_j\ , \left(x_a R_j x_b\ \wedge\ x_b R_j x_c\right) \Rightarrow\ x_a R_j x_c, \left(x_a R_j x_b\ \wedge\ x_b R_j x_a\right) \Rightarrow\ \ x_a = x_b$$

  - Числовой – есть расстояние, $D_j = \mathbb{R}$
  - Бинарный $D_j = \{0,1\}$
- Вектор признаков, описывающий объект $(f_1(x), f_2(x), \ldots, f_n(x))$

Пространство признаков

- Набор данных - множестно $Z = \{x_1, \dots, x_l\}$, включает l объектов (наблюдений) из X и может быть представлено:
  - Матрицей признаков

$$\|F\|_{l \times p} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \dots & f_p(x_1) \\ & \dots & & \\ f_1(x_l) & f_2(x_l) & \dots & f_p(x_l) \end{pmatrix}$$

  - Симметричной матрицей сходства (или различия)

$$\|K\|_{l \times l} = \begin{pmatrix} d(x_1, x_1) & d(x_1, x_2) & \dots d(x_1, x_l) \\ & \dots & \\ d(x_l, x_1) & d(x_l, x_2) & \dots d(x_l, x_l) \end{pmatrix}$$

  где $d: X \times X \to \mathbb{R}$ некая симметричная, мера сходства или различия, не обязательно расстояние (правило треугольника может не выполняться)
- Вопрос построения признакового пространства – не простой, а иногда и критический, причины:
  - Мусор на вход – мусор на выход
  - Сложные структурированные объекты (например, графы, тексты и т.д.)
  - Противоречия, пропуски, ошибки, артефакты
  - Взаимозависимые и незначимые признаки

Расстояния для числовых признаков

- Обычно строится на основе метрики:

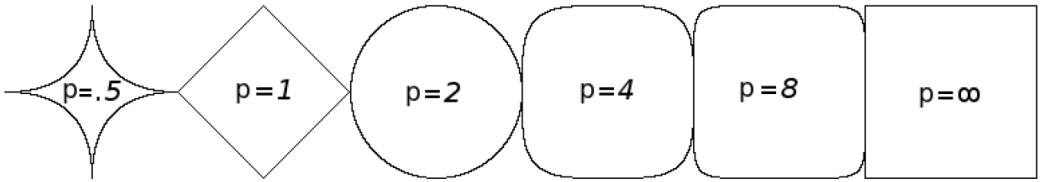
$$d(i,j) \geq 0, d(i,i) = 0, d(i,j) = d(j,i), d(i,j) \leq d(i,k) + d(k,j)$$

- Наиболее популярно расстояние Минковского:

$$d(i,j) = \sqrt[p]{|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \ldots + |x_{in} - x_{jn}|^p)}$$

где $i = (x_{i1}, x_{i2}, \ldots, x_{in})$ и $j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ - два объекта с n числовыми атрибутами, p - положительное целое число

- «Изолинии» (области точек равноудаленных от заданной):



- $p = 2$ - Евклидово расстояние: $d(i,j) = \sqrt{(|x_{i1} - x_{j1}|^2 + \ldots + |x_{in} - x_{jn}|^2)}$
- $p = 1$ – расстояние «Манхэттен»: $d(i,j) = |x_{i1} - x_{j1}| + \ldots + |x_{in} - x_{jn}|$

Приведение к близким шкалам

- Цель – исключить влияние разброса значений признака и уменьшить влияние выбросов (*l* – размер выборки)
- Нормализация на абсолютное отклонение более робастно (устойчиво к выбросам), чем через стандартное отклонение:

$$x'_{if} = \frac{x_{if} - m_f}{s_f},\ s_f = \frac{1}{l}(|x_{1f} - m_f| + |x_{2f} - m_f| + \ldots + |x_{lf} - m_f|),$$

$$m_f = \frac{1}{l}\left(x_{1f} + x_{2f} + \ldots + x_{lf}\right)$$ //можно использовать медиану

- Стандартизация:

$$x'_{if} = \frac{x_{if} - m_f}{std_f},\ std_f = \sqrt{\frac{1}{l-1}\left[(x_{1f} - m_f)^2 + (x_{2f} - m_f)^2 + \ldots + (x_{nf} - m_f)^2\right]}$$

Бинарные атрибуты

- Расстояние Хэмминга = сумма единиц после XOR: $(M_{10} + M_{01})$
- Таблица «сопряженных признаков»
  - В ячейках – число совпадающих и несовпадающих значений из p бинарных атрибутов для объектов *j* и *i*

Объект *i* \ Объект *j*	1	0	sum
1	M_{11}	M_{10}	$M_{11} + M_{10}$
0	M_{01}	M_{00}	$M_{01} + M_{00}$
sum	$M_{11} + M_{01}$	$M_{10} + M_{10}$	$M_{00} + M_{01} + M_{10} + M_{11}$

- На основе коэффициента совпадения $d(i,j) = \frac{M_{10} + M_{01}}{M_{11} + M_{01} + M_{10} + M_{00}}$
  - для симметричных атрибутов (значения равнозначны)
- На основе коэффициента Jaccard $d(i,j) = \frac{M_{10} + M_{01}}{M_{11} + M_{01} + M_{10}}$
  - для асимметричных атрибутов (единица важнее)

Пример

Имя	Пол	Жар	Кашель	Test-1	Test-2	Test-3	Test-4
Jack	M	Y	N	P	N	N	N
Mary	F	Y	N	P	N	P	N
Jim	M	Y	P	N	N	N	N

- пол - симметричный атрибут
- остальные ассиметричные
- пусть Y и P соответствует 1, а N соответствует 0

$$d(jack, mary) = \frac{0+1}{2+0+1} = 0.33$$

$$d(jack, jim) = \frac{1+1}{1+1+1} = 0.67$$

$$d(jim, mary) = \frac{1+2}{1+1+2} = 0.75$$

Категориальные атрибуты и шкалы

- Категориальные атрибуты:
  - может быть много значений
- Простое совпадение
  - m - число совпадений, p - число категориальных переменных (аналог нормированного расстояния Хэмминга):

$$d(i,j) = \frac{p - m}{p}$$

- То же самое - кодирование бинарными векторами
  - Для каждого значения категориального атрибута создается отдельная бинарная переменная: один категориальный атрибут с k возможными значениями => бинарный вектор длины k
- Отображение на числовую шкалу:
  - Могут быть и дискретными и непрерывными
  - сводятся к числовым: заменить значение переменной на его ранг или отобразить на [0, 1] с нормировкой
  - затем использовать стандартные расстояния

Преобразование категориальных переменных

Значение	D_A	D_B	D_C	D_D	D_E	D_F	D_G	D_H	D_I
A	1	0	0	0	0	0	0	0	0
B	0	1	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	0	0
D	0	0	0	1	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	0	0	1	0	0	0
G	0	0	0	0	0	0	1	0	0
H	0	0	0	0	0	0	0	1	0
I	0	0	0	0	0	0	0	0	1

- Бинарное кодирование
- Выделение категории редких признаков
- Группировка признаков по поведению отклика (будет подробнее позже)

Значение	N_i	ΣY_i	p_i
A	1562	430	0.28
B	970	432	0.45
C	223	45	0.20
D	111	36	0.32
E	85	23	0.27
F	50	20	0.40
G	23	8	0.35
H	17	5	0.29
I	12	6	0.50
J	5	5	1.00

Категориальные признаки

Пример переменной	Мощность	***Подход***
Physical characteristics Region Partnership status Education level	10	Бинарное кодирование
Urbanicity codes State Ethnicity	100	Преобразования или отображение на числовую шкалу
Employment classification Postal Code	1000	
Address	1000000	Связывание
Social security number	100000000	
text	*Бесконечность*	Текстовые модели

Смешанные типы атрибутов

- Нелинейные шкалы:
  - логарифмические, экспоненциальные и другие
  - «обратное» преобразование к линейной шкале
  - «экспертное» преобразование к ранговой шкале
- «Взвешенное» нормированное расстояние:

$$d(i,j) = \frac{\Sigma_{f=1}^{p} w_{ij}^{(f)} d_{ij}^{(f)}}{\Sigma_{f=1}^{p} w_{ij}^{(f)}}$$

  - *f* - бинарный или номинальный: $d_{ij}^{(f)} = 0$ если $x_{if} = x_{jf}$, иначе $d_{ij}^{(f)} = 1$
  - *f* - числовой: использовать нормализованное расстояние
  - *f* - шкала: рассчитать ранги r_{if} , нормализовать и считать числовым
- Другие меры сходства:
  - Ядерные функции (будут позже)
  - Для однородны признаков

На основе корреляции

$$s(\vec{x}_i, \vec{x}_j) = \frac{cov_{\vec{x}_i \vec{x}_j}}{\sigma_{\vec{x}_i \vec{x}_i} \sigma_{\vec{x}_j \vec{x}_j}}$$

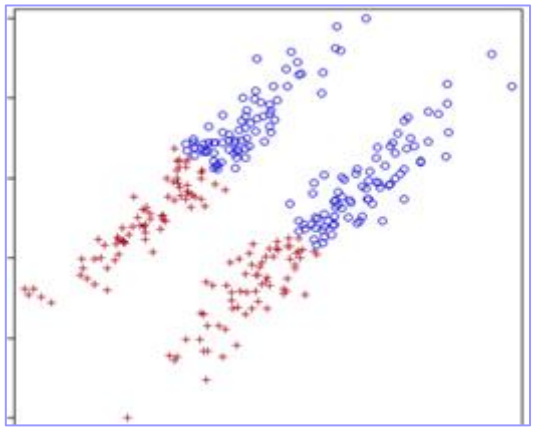
«Тригонометрические»:

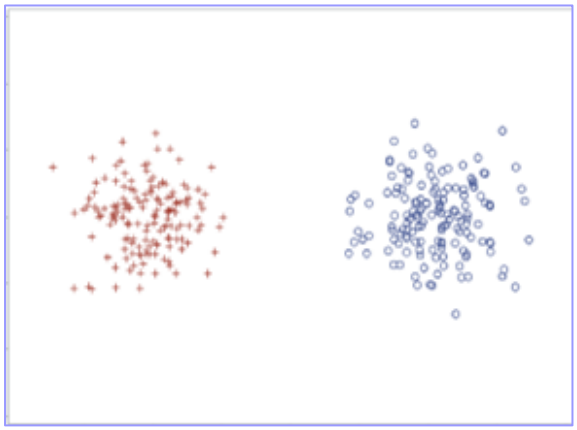
$$s(\vec{x}_i, \vec{x}_j) = \frac{\langle \vec{x}_i, \vec{x}_j \rangle}{\sqrt{\langle \vec{x}_i, \vec{x}_i \rangle \langle \vec{x}_j, \vec{x}_j \rangle}}$$

Расстояние Махалонобиса

- Учет линейной корреляции в пространстве признаков, где $\sum$ - ковариационная матрица по всей выборке

$$d(x^*, x) = (x^* - x)^T \sum^{-1} (x^* - x)$$





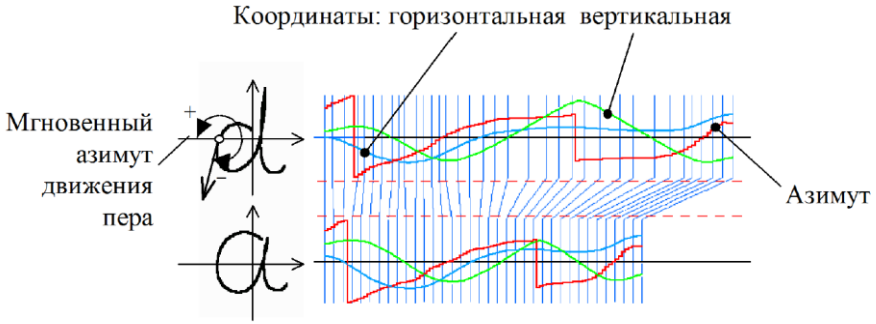
$$x \to z = \varphi(x) = \sum^{-1/2} (x - \mu)$$

Расстояние для сложных структур (примеры)

- Расстояние Левенштейна для последовательностей:
  - оценивает *минимальное* число операций (вставок и удалений) для сведения двух последовательностей к общей
  - Эффективно считается алгоритмами динамического программирования

```
CTGGGCTAAAAGGTCCCTTAGCC..TTTAGAAAAA.GGGCCATTAGGAAATTGC
CTGGGACTAAA....CCTTAGCCTATTTACAAAAATGGGCCATTAGG...TTGC
```

- DTW (и другие) – аналогично Левенштейну:
  - Оценивают «затраты» на сведения двух структур к общей
  - Эффективно считаются алгоритмами динамического программирования



Преобразованное пространство признаков

- Преобразование пространства признаков с помощью отображения Φ: $\mathbb{R}^p \mapsto H, x \mapsto \Phi(x)$, где скалярное произведение:

$$\langle \Phi(x_i), \Phi(x_j) \rangle$$

- НО: в явном виде не нужно вычислять новые признаки и образы, достаточно заменить скалярное произведение на ядерную функцию:

$$\mathrm{K}(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$$

- Ядро – симметричная неотрицательно определенная функция (конечное множество значений порождает симметричную, неотрицательно определенную матрицу сходства)
- Простые примеры:
  - Линейная: $K(x_i, x_j) = \langle x_i, x_j \rangle$
  - Полиномиальная степени *p*: $K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^p$
  - Гауссовская: $K(x_i, x_j) = e^{-\|\mathbf{x_i} - \mathbf{x_j}\|^2 / 2\sigma^2}$

Ядерные функции

- Методы построения ядерных функций для сложно структурированных объектов:
  - на основе заданной вероятностной модели
  - на основе экспертных попарных или групповых сравнений
  - на основе R-свертки по отношениям:

$$K(x,y) = \sum_{\vec{x}\in R^{-1}(x),\vec{y}\in R^{-1}(y)} \prod_{d=1}^{D} K_d(x_d, y_d), \vec{x} =< x_1, \ldots, x_D >$$

Здесь R – набор всех возможных вариантов декомпозиции сложных объектов x и y на составные части

K-spectrum ядро (для цепочек символов: текстов, ДНК и т.д.) – совпадение всех подстрок с весовым параметром $0 < \lambda < 1$

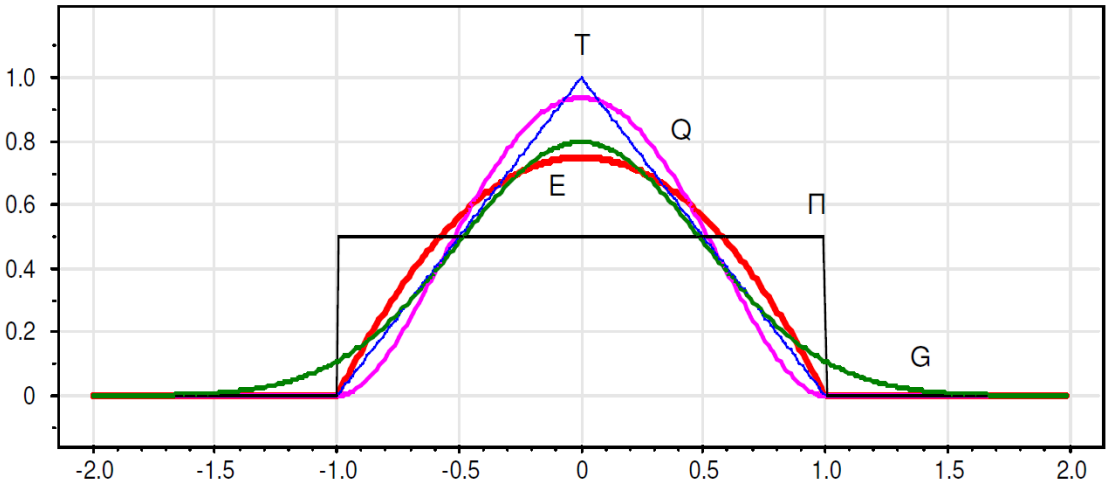
$$k_n(s,t) = \sum_{u\in\Sigma^n} (\varphi_u(s)\cdot\varphi_u(t)) = \sum_{u\in\Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}$$

Радиально-базисные ядра

- Одномерные ядра от расстояния (h – нормирующий параметр, ширина ядра):

$$K(x_j, x_i) = K\left(\frac{d(x_j, x_i)}{\mathrm{h}}\right)$$

$П(r) = [|r| \leqslant 1]$ — прямоугольное
$T(r) = (1 - |r|)[|r| \leqslant 1]$ — треугольное
$E(r) = (1 - r^2)[|r| \leqslant 1]$ — квадратичное (Епанечникова)
$Q(r) = (1 - r^2)^2[|r| \leqslant 1]$ — квартическое
$G(r) = \exp(-2r^2)$ — гауссовское



sklearn.preprocessing (предобработка данных)

- Обучение: **fit, fit_transform**
- Преобразование: **transform, inverse_transform**
- Стандартизация:
  - StandardScaler: (x - mean(x))/std(x)
  - RobustScaler: (x - median(x))/IQR(x)
- Нормализация:
  - MinMaxScaler: (x - min(x))/(max(x) - min(x))
  - MaxAbsScaler: x/max(|x|)
  - Normalizer(norm) # l1, l2, max
- Преобразование:
  - QuantileTransformer(n_quantiles)
  - PowerTransformer # yeo-johnson, box-cox
  - KBinsDiscretizer: x -> 0..k-1
  - Binarizer: x -> 0,1

```
X = np.array([[1, 10, 100, 200, 1000]]).T

StandardScaler().fit_transform(X)
RobustScaler().fit_transform(X)
MinMaxScaler().fit_transform(X)
MaxAbsScaler().fit_transform(X)
Normalizer().fit_transform(X)
QuantileTransformer().fit_transform(X)
PowerTransformer().fit_transform(X)
KBinsDiscretizer(encode="ordinal").fit_transform(X)
Binarizer(threshold=20).fit_transform(X)
```

```
StandardScaler:
 [[-0.69493808]
 [-0.67099305]
 [-0.43154271]
 [-0.16548679]
 [ 1.96296062]]
RobustScaler:
 [[-0.52105263]
 [-0.47368421]
 [ 0.        ]
 [ 0.52631579]
 [ 4.73684211]]
MinMaxScaler:
 [[0.        ]
 [0.00900901]
 [0.0990991 ]
 [0.1991992 ]
 [1.        ]]
MaxAbsScaler:
 [[0.001]
 [0.01 ]
 [0.1  ]
 [0.2  ]
 [1.   ]]
Normalizer:
 [[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
QuantileTransformer:
 [[0.  ]
 [0.25]
 [0.5 ]
 [0.75]
 [1.  ]]
PowerTransformer:
 [[-1.42849354]
 [-0.76770092]
 [ 0.21971925]
 [ 0.55897206]
 [ 1.41750316]]
KBinsDiscretizer:
 [[0.]
 [1.]
 [2.]
 [3.]
 [4.]]
Binarizer:
 [[0]
 [0]
 [1]
 [1]
 [1]]
```

Предобработка данных

- Кодирование категориальных признаков:
  - Модуль sklearn.preprocessing
  - LabelEncoder()
  - OrdinalEncoder(categories)
  - OneHotEncoder(categories, min_frequency)
  - Параметр handle_unknown (error, ignore, …)
- Обработка пропусков:
  - Модуль sklearn.impute
  - SimpleImputer(missing_values, strategy)
  - KNNImputer(n_neighbors, weights)
  - MissingIndicator(missing_values)

```
X = [['male', 'sin', 'fit'],
     ['female', 'cos', 'predict'],
     ['female', 'sin', 'transform']]

OneHotEncoder().fit_transform(X).toarray()
```

```
[[0., 1., 0., 1., 1., 0., 0.],
 [1., 0., 1., 0., 0., 1., 0.],
 [1., 0., 0., 1., 0., 0., 1.]]
```

```
X = np.array([[1, np.nan], [np.nan, 3], [7, 6]])
imp = SimpleImputer(missing_values=np.nan,
                    strategy='mean')
imp.fit_transform(X)
```

```
[[1. , 4.5],
 [4. , 3. ],
 [7. , 6. ]]
```

```
knn_imp = KNNImputer(n_neighbors=1,
                     weights="uniform")
knn_imp.fit_transform(X)
```

```
[[1., 6.],
 [7., 3.],
 [7., 6.]]
```

Предобработка признакового пространства

- Генерация признаков:
  - Модуль sklearn.preprocessing
  - PolynomialFeatures(degree, interaction_only)
  - SplineTransformer(degree, n_knots)
  - FunctionTransformer(func, inverse_func)

- Уменьшение размерности:
  - Модуль sklearn.decomposition
  - PCA(n_components) # число или mle
  - TruncatedSVD(n_components, algorithm)
  - Атрибуты: **explained_variance_, singular_values_**
  - NMF(n_components, init, solver)
  - LatentDirichletAllocation(n_components)
  - Общий атрибут: **components_**

```
X = np.arange(6).reshape(3, 2)
PolynomialFeatures(2).fit_transform(X)
```

```
[[ 1.,  0.,  1.,  0.,  0.,  1.],
 [ 1.,  2.,  3.,  4.,  6.,  9.],
 [ 1.,  4.,  5., 16., 20., 25.]]
```

```
SplineTransformer(2, 1).fit_transform(X)
```

```
[[1. , 0. , 1. , 0. ],
 [0.5, 0.5, 0.5, 0.5],
 [0. , 1. , 0. , 1. ]]
```

```
X = np.arange(100).reshape(10, 10)

PCA(n_components=2).fit_transform(X)
```

```
[[ 1.42302495e+02,  1.67908651e-14],
 [ 1.10679718e+02, -2.41419792e-15],
 [ 7.90569415e+01, -3.65037002e-15],
 [ 4.74341649e+01, -1.51614199e-15],
 [ 1.58113883e+01, -2.24513984e-16],
 [-1.58113883e+01,  2.24513984e-16],
 [-4.74341649e+01,  1.51614199e-15],
 [-7.90569415e+01,  3.65037002e-15],
 [-1.10679718e+02,  2.41419792e-15],
 [-1.42302495e+02,  7.91882609e-15]]
```

Пропущенные значения

- Не все значения признаков известны или достоверны
  - важная задача, так как многие к ней сводятся (удаление шума, не консистентностей и т.д.)
- Причины появления пропущенных значений
  - Ошибки «оборудования» и/или ПО при получении данных от датчиков и из экспериментов
  - Удаление несогласованных значений атрибутов
  - Просто не введены в систему из-за халатности или ошибки
  - Часть данных может быть опциональна с точки зрения бизнес процессов организации, но важна для анализа
  - Не хранится правильная история изменений – невозможно правильно определить значение на момент анализа
- Пропущенные данные:
  - Ведут к неточным результатам анализа
  - Допускаются не всеми алгоритмами анализа

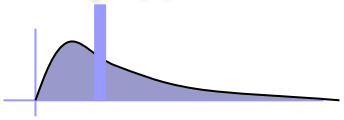
Методы обработки пропущенных значений

- Игнорировать объект или запись:
  - Можем потерять важные объекты (например, опорные вектора)
  - Можем «испортить» выборочное распределение
  - В некоторых задачах процент пропущенных значений велик (>50%)
- Заполнение пропущенных значений «вручную»:
  - Нужен очень грамотный эксперт
  - Полностью «вручную» невозможно для больших объемов
  - Правила заполнения (импутации) трудно формулировать – проблема полноты, противоречивости, достоверности
- Использование глобальной спец. константы типа “unknown”
  - Не всеми алгоритмами анализа реализуемо
- Импутация «среднего» или «наиболее ожидаемого» значения
  - По всей выборке, по страте (срезу), по классу, по кластеру и т.д.
  - Наиболее популярный метод, но можем «испортить» выборочное распределение
  - Методы импутации на основе прогнозирования

Основные подходы к подстановке пропусков

- Импутация константным значением - все пропуски для переменной заменяются на:
  - Моду (для категориальных) или мат. ожидание,
    или пользовательскую константу
    или робастные оценки

Распределения



- Импутация псевдослучайным значением:
  - В соответсвии с распределением
- Импутация прогнозом (оценкой)
- Для неслучайных пропусков – индикаторные переменные
  - Одна на все наблюдение
  - Своя для каждой переменной

Оценки

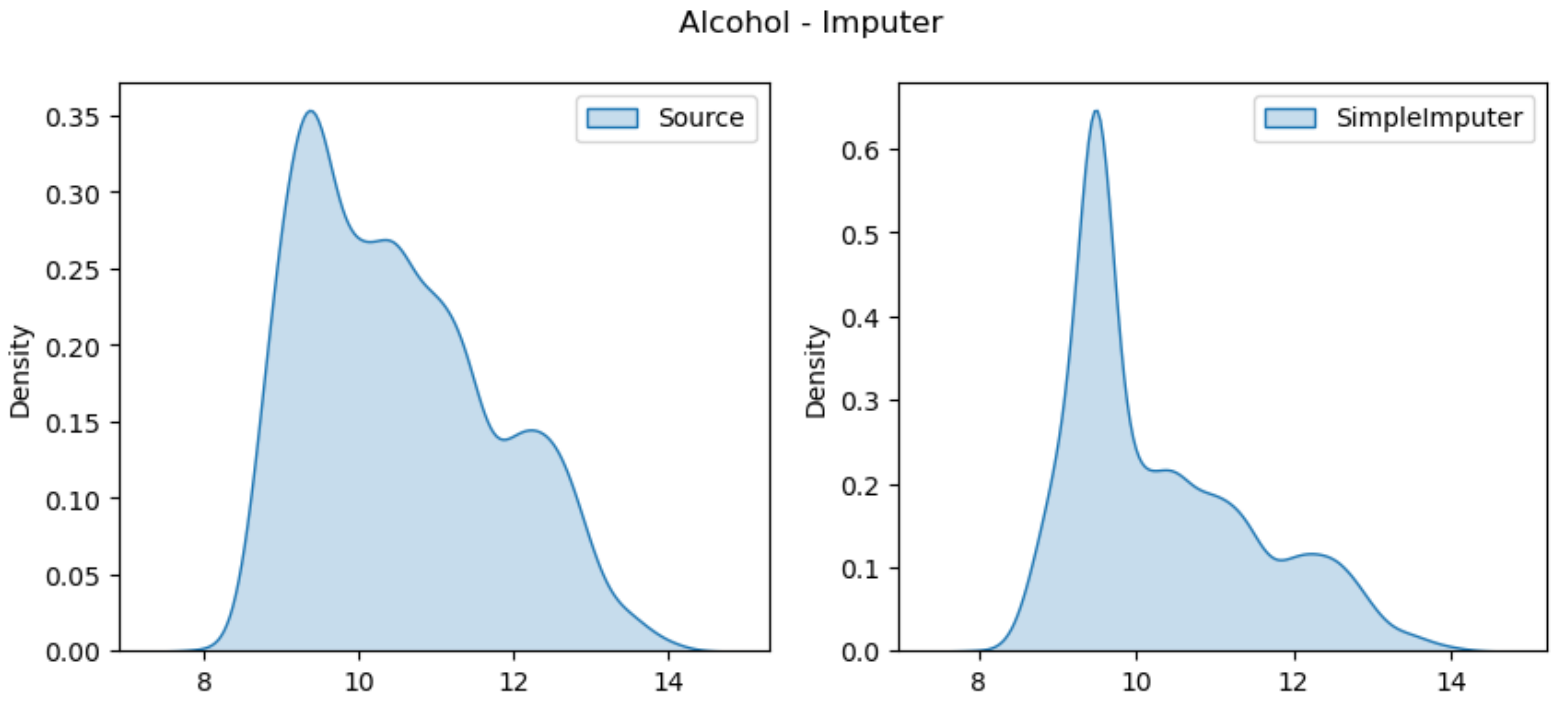
$$x_i = f(x_1, \ldots, x_p)$$

Пример Simple Imputer

```
X_ = X[["alcohol", "pH"]].values

X_nan = X_.copy()
# Генерация пропусков
ind = np.random.rand(X_nan.shape[0]) < 0.2
X_nan[ind, 0] = np.nan
np.isnan(X_nan).sum()
```

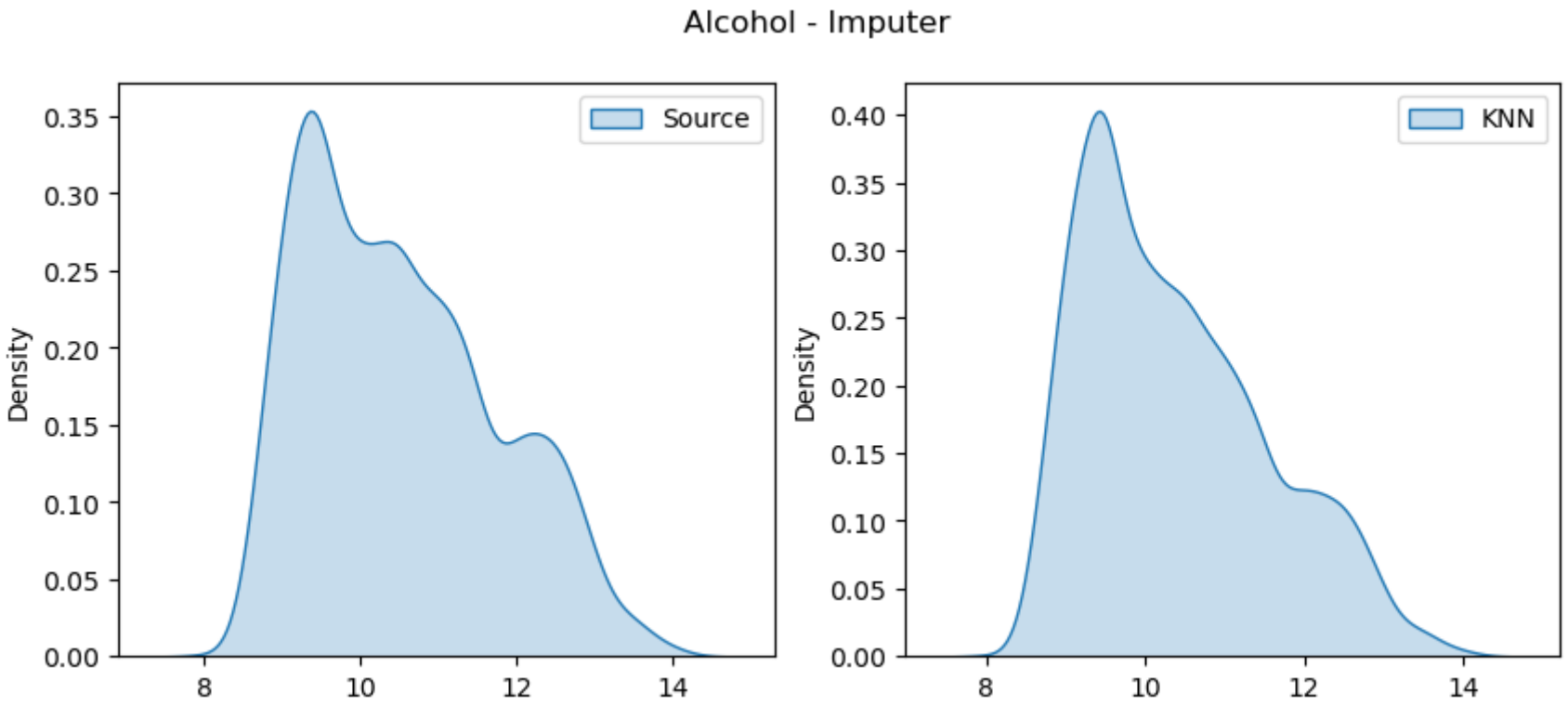
```
970
```

```
from sklearn.impute import SimpleImputer

X_only_nan = X_nan[:, 0].reshape(-1, 1)
a = pd.DataFrame(X_only_nan, columns=["Source"])
imp = SimpleImputer(strategy='most_frequent')
imp.fit_transform(X_only_nan)
```



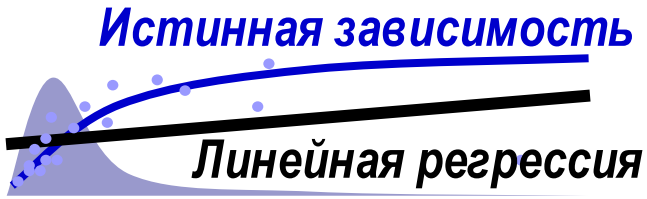
Пример KNN Imputer

```
from sklearn.impute import KNNImputer
a = pd.DataFrame(X_only_nan, columns=["Source"])
knn_imp = KNNImputer(n_neighbors=2)
knn_imp.fit_transform(X_nan)[:, 0]
```



Преобразование непрерывных переменных

- Простые преобразования:
  - Функции от исходной (log, exp, …)

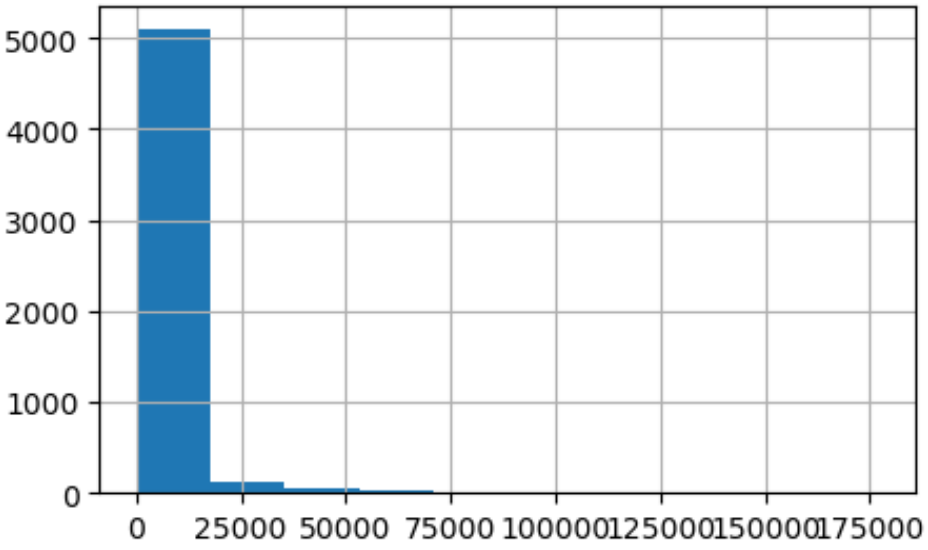


  - Нормализация (z-score, центрирование, сведение на [0,1])
  - Дискретизация (равные интервалы, равные группы, адаптивные интервалы с учетом отклика и т.д.)
- Адаптивные преобразования – перебор простых и выбор лучшего по некоторому критерию:
  - Нормальность распределения результата
  - Корреляция с откликом
  - Оптимальная дискретизация

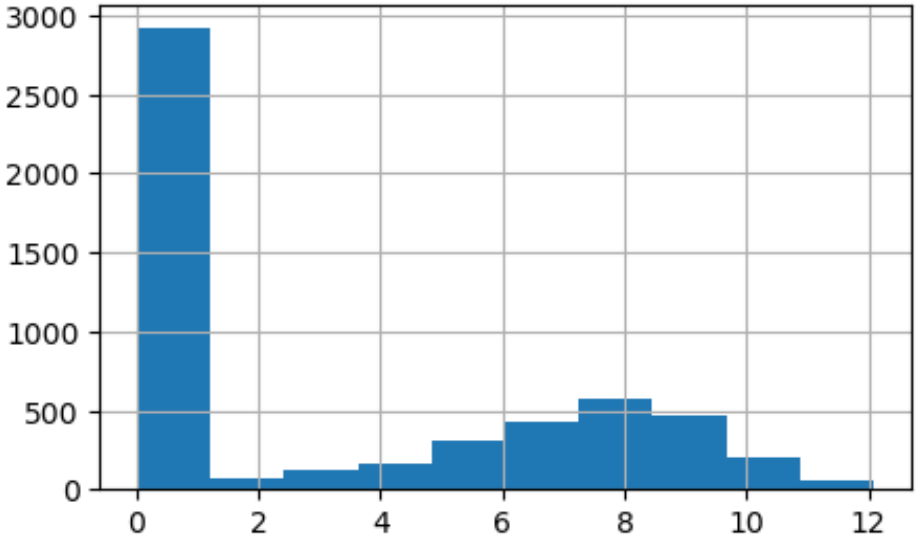
Преобразование переменных

- Простые преобразования:
  - Функции от исходной (log, exp, …), дискретизации (на бакеты и квантили), объединение редких категориальных значений и т.д.
- Адаптивные преобразования – перебор простых и выбор лучшего по некоторому криетрию:
  - Нормальность распределения результата, корреляция с откликом, Оптимальная дискретизация и т.д.

```
df["SAVBAL"].hist(figsize = (5, 3))
```



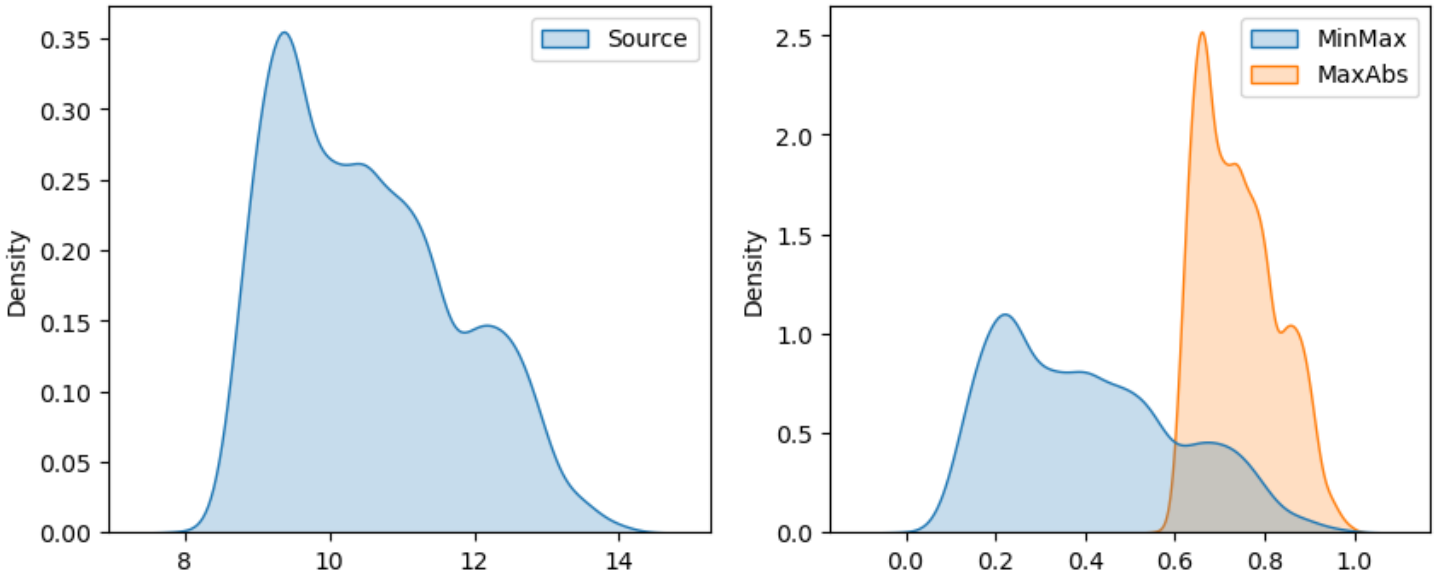

```
df["log_SAVBAL"] = np.log(df["SAVBAL"] + 1)
df["log_SAVBAL"].hist(figsize = (5, 3))
```



Преобразование непрерывных переменных (Normalization)

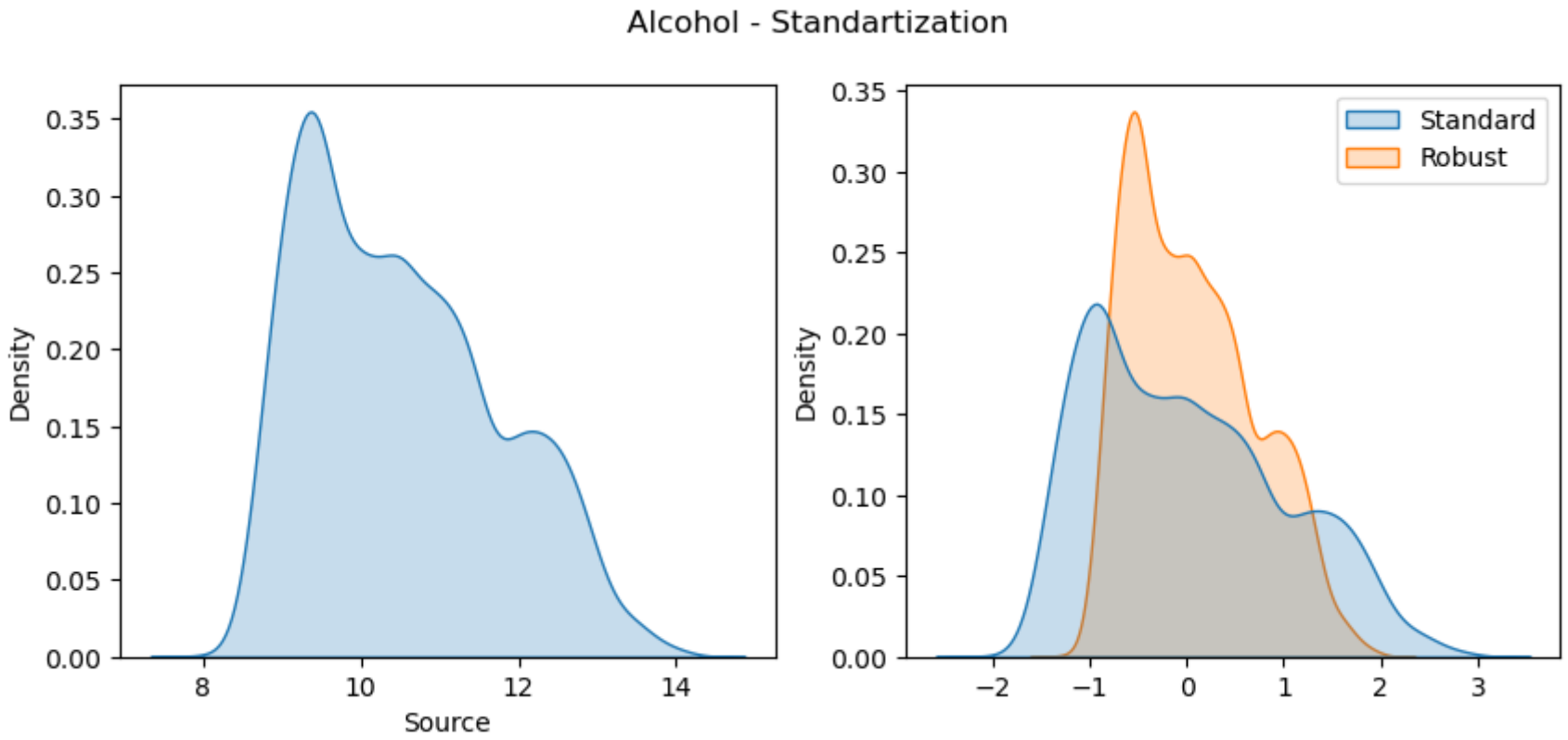
```
X_ = X["alcohol"].values.reshape(-1, 1)
```

```
from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler
a = pd.DataFrame(X_, columns=["Source"])

fig, axes = plt.subplots(ncols=2, figsize=(10, 4))
fig.suptitle("Alcohol - Normalization")
sns.kdeplot(a[["Source"]], ax=axes[0], fill=True)

mm = MinMaxScaler().fit_transform(X_)
ma = MaxAbsScaler().fit_transform(X_)
```



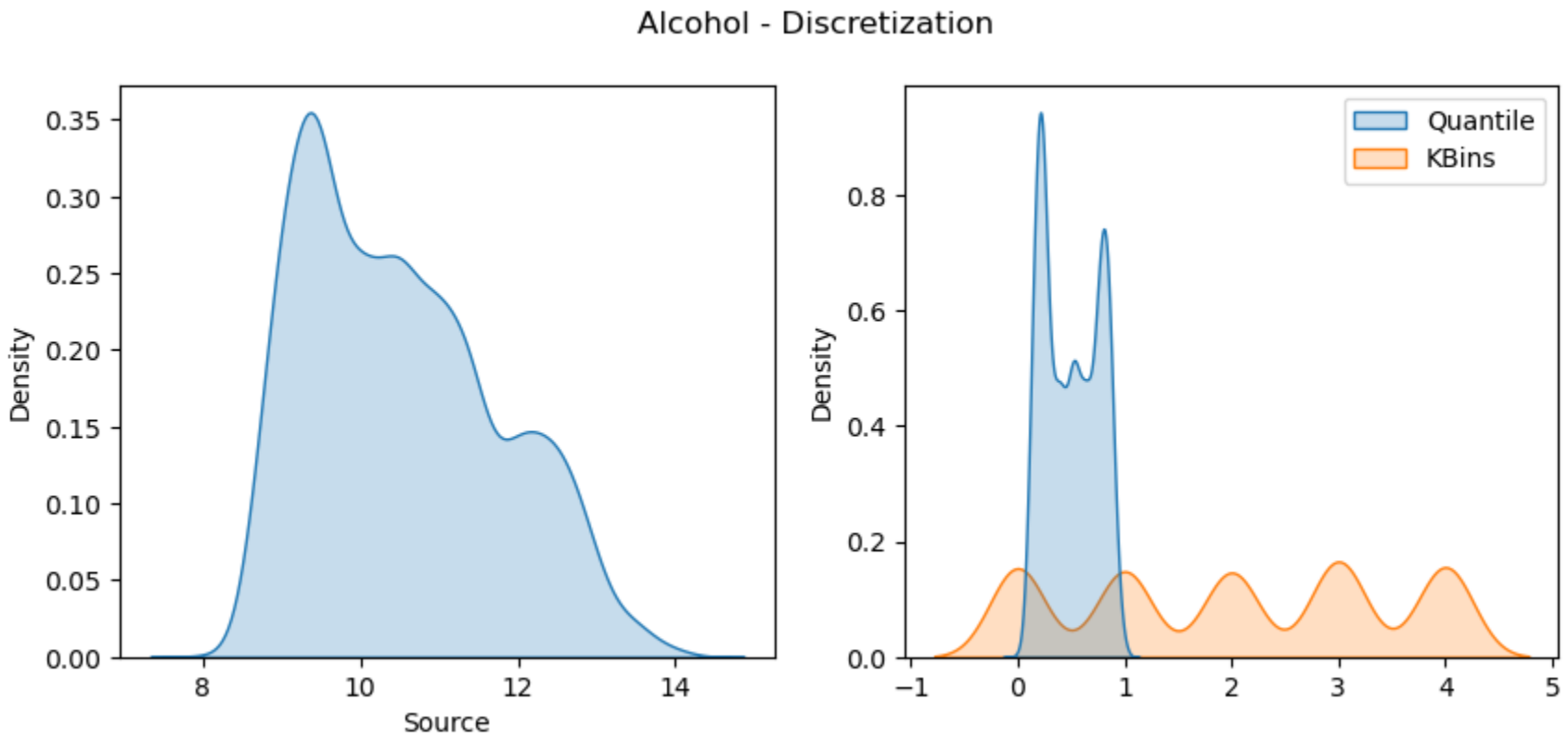
Преобразование непрерывных переменных (Standartization)

```
from sklearn.preprocessing import StandardScaler, RobustScaler

st_sc = StandardScaler().fit_transform(X_)
rb_sc = RobustScaler().fit_transform(X_)
```



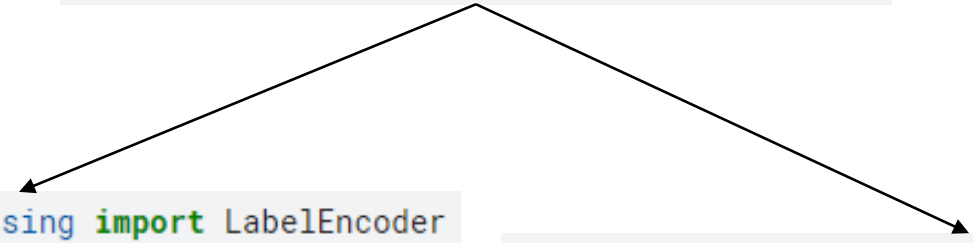
Преобразование непрерывных переменных (Discretization)

```
from sklearn.preprocessing import QuantileTransformer, KBinsDiscretizer

q_tr = QuantileTransformer(n_quantiles=5).fit_transform(X_)
kbn = KBinsDiscretizer(encode="ordinal").fit_transform(X_)
```



Преобразование категориальных переменных

```
quality = y.to_numpy().reshape(-1, 1)
```



```
from sklearn.preprocessing import LabelEncoder

print("quality", quality[-5:, 0])
le = LabelEncoder()
labels = le.fit_transform(quality)
print("Classes", le.classes_)
print("LabelEncoder")
print(labels[-5:])
```

```
quality [6 5 6 7 6]
Classes [3 4 5 6 7 8 9]
LabelEncoder
[3 2 3 4 3]
```

```
from sklearn.preprocessing import OneHotEncoder
print("quality", quality[-5:, 0])
ohe = OneHotEncoder()
code = ohe.fit_transform(quality).toarray()
print("OneHotEncoder")
print(code[-5:])
```

```
quality [6 5 6 7 6]
OneHotEncoder
[[0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0.]]
```

Обучение без учителя

- Иногда называют задачей «самоорганизации»
- Все признаки равнозначны, нет отклика, поэтому:
  - Обучение без учителя более «субъективное» (нет единых интуитивно понятных мер оценки качества типа «точности») и менее «автоматизируемо»
  - Сложнее подбирать метапараметры и сравнивать полученные модели
- Важность этого направления велико:
  - «Истинный data mining» - ищет неизвестные заранее зависимости без «подсказок» эксперта. Много важных прикладных задач по сегментации, выявление скрытых характеристик, зависимостей между атрибутами и т.д.
  - Для больших данных получить качественно размеченный набор тяжело или невозможно, часто возникают задачи *semisupervised learning*, когда размечена лишь часть набора
- Большинство задач сводится к поиску скрытых (латентных) признаков или скрытых структур (групп, отношений, зависимостей …) в данных:

$$z_j: D_1 \times \ \ldots \times D_n \to D_{zj}, z_1 = F_1(f_1, \ldots, f_n), \ldots, z_k = F_k(f_1, \ldots, f_n)$$

  - Основные случаи: z_j или отношения (например, кластеры, ассоциативные правила) или новые признаки (например, главные компоненты и степень принадлежности кластеру), иногда как в SOM (сетях Кохонена) и то, и другое сразу.

Обучение с учителем

- Дано: множество «размеченных» примеров :
  - обучающая выборка или тренировочный набор:
    $$Z = \{(x_i, y_i)\}_{i=1}^{l} \in X \times Y$$
  - $y_i \in Y$: известный «отклик» и его множество значений
  - Неизвестная зависимость, которую необходимо «восстановить» $\mathrm{y}: X \to Y$
- Постановка задачи:
  - Найти алгоритм (или гипотезу, или модель, или решающую функцию)
    $$a_Z: X \to Y$$
  - «качественно» приближающую неизвестную $\mathrm{y}: X \to Y$ на всем признаковом пространстве
- Два этапа:
  - Обучение (построение модели), метод обучения μ выбирает «лучший» алгоритм a (модель, гипотезу) среди заданного семейства A:
    $$a_Z = \mu(Z) \in A$$
  - Применение (скоринг модели), алгоритм a выдает (прогнозирует) значения отклика $\acute{y} = a_Z(\acute{x})$ для «новых» объектов (возможно для множества «новых» объектов) с неизвестным откликом

Типы задач обучения с учителем в зависимости от типа отклика

- Вообще тип задачи обучения с учителем определяются не только типом допустимых значений «отклика», но и **оценкой качества**, которая используется для выбора модели
- Типы задач:
  - Бинарная классификация $Y = \{0,1\}$ – решающая функция бинарная
  - Много-классовая классификация $Y = \{C_1, \dots, C_k\}$ – категориальный признак, значения взаимоисключающие, не задано отношение порядка, наблюдение не может принадлежать нескольким классам одновременно, решающая функция дискретная
  - Много-темная (multi-label) классификация $Y = \{0,1\}^k$ или Y – множество всех подмножеств $\{C_1, \dots, C_k\}$, решающая вектор функция выдает бинарный вектор релевантных тем
  - Регрессия $Y = \mathbb{R}$ или $Y = \mathbb{R}^{\mathrm{k}}$, решающая функция – вещественная (вектор) функция
  - Порядковая регрессия $Y = \{1, \dots, K\}$ – дискретный признак, задано отношение порядка, решающая функция дискретная
  - Ранжирование $Y = \{C_1, \dots, C_k\}$ – категориальный признак, значения взаимоисключающие, решающая вектор функция выдает вектор степеней соответствия наблюдений каждому из классов

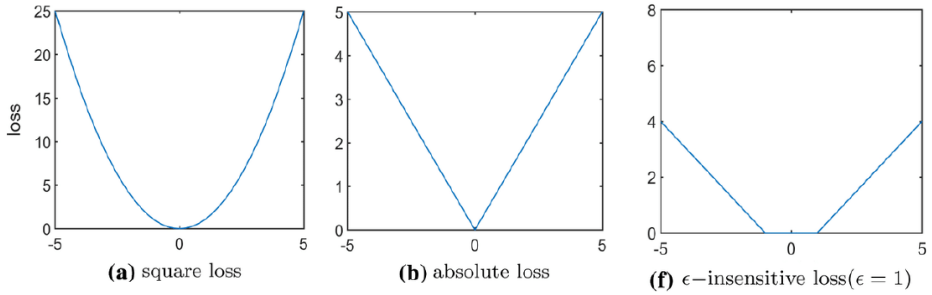
Другие виды обучения с учителем

- Обучение с подкреплением (reinforcement learning) – агент обучается итерационно в зависимости от отклика среды и своего состояния
- (До)обучение «на лету» (online leering) – корректировка модели по одному примеру или пакетом новых примеров
- Многозадачное (multi-task) – решается несколько ML задач сразу
- Активное обучение – процесс обучения влияет на формирование обучающей выборки
- Meta-learning – обучение как обучать модели машинного обучения
- Прогнозирование сложных откликов - отклик не атрибут, а сложный объект, например, граф или текст или последовательность действий как в MBR

Функция потерь

- $L: Y \times Y \to \mathbb{R}^+$ характеризует отличие истинного отклика от спрогнозированного $L(y(x), a(x))$
- Примеры:
  - Классификация и регрессия:

$$L(y, y') = [y \neq y'],$$
$$L(y, y') = |y - y'|,$$
$$L(y, y') = (y - y')^2,$$
$$L(y, y') = [|y - y'| > \varepsilon]$$



  - Много-темная классификация:

$$L(y, y') = |y \nabla y'|, a \nabla b = (a \cup b) \backslash (a \cap b), a \subseteq Y, b \subseteq Y$$

  - Ранжирование:

$$L(y, y') = \frac{|\{(r, s): y_r \leq y_s \wedge y'_r \leq y'_s\}|}{|y||y'|}$$

Функционал качества

- Теоретический риск:

$$\mathrm{E}\big[L\big(y(x),a(x)\big)\big] = \int_{X\times Y} L\big(y(x),a(x)\big)\partial P(x,y) \to min$$

- Но $P(x,y)$ мы не знаем, а если бы знали, то и решать ничего не нужно, поэтому используем эмпирический риск:

$$Q(a,Z) = \frac{1}{l}\sum_{Z} L\big(y_i,a(x_i)\big) \to min$$

- Получаем, что метод обучения μ выбирает «лучший» алгоритм a_Z^* (модель, гипотезу) среди заданного семейства A на наборе данных Z как:

$$a_Z^* = \mu(Z) = \arg\min_{a\in A} Q(a,Z)$$

- Можно ли использовать оценку качества на обучающей выборке как объективную? Ответ – НЕТ

Важные на практике свойства метода машинного обучения

- Масштабируемость на этапе обучения и применения
- Поддержка различных типов признаков и структур данных
- Возможность находить зависимости сложной формы
- Отсутствие или наличие требований к присутствию априорных знаний о выборке (например, о распределениях)
- Устойчивость к «шуму» и выбросам
- Возможность работы с высокой размерностью и с большой выборкой
- Возможность включать пользовательские ограничения
- Интерпретируемость модели и наглядность визуализации
- Интуитивность метапараметров