Методы машинного обучения

Методы машинного обучения

Машинное обучение — это раздел информатики, посвященный созданию алгоритмов, опирающихся на набор данных о каком-либо явлении. Эти данные могут быть получены из естественной среды, созданы вручную или сгенерированы другим алгоритмом.

Машинное обучение = формирование набора данных + алгоритмическое построение статистической модели

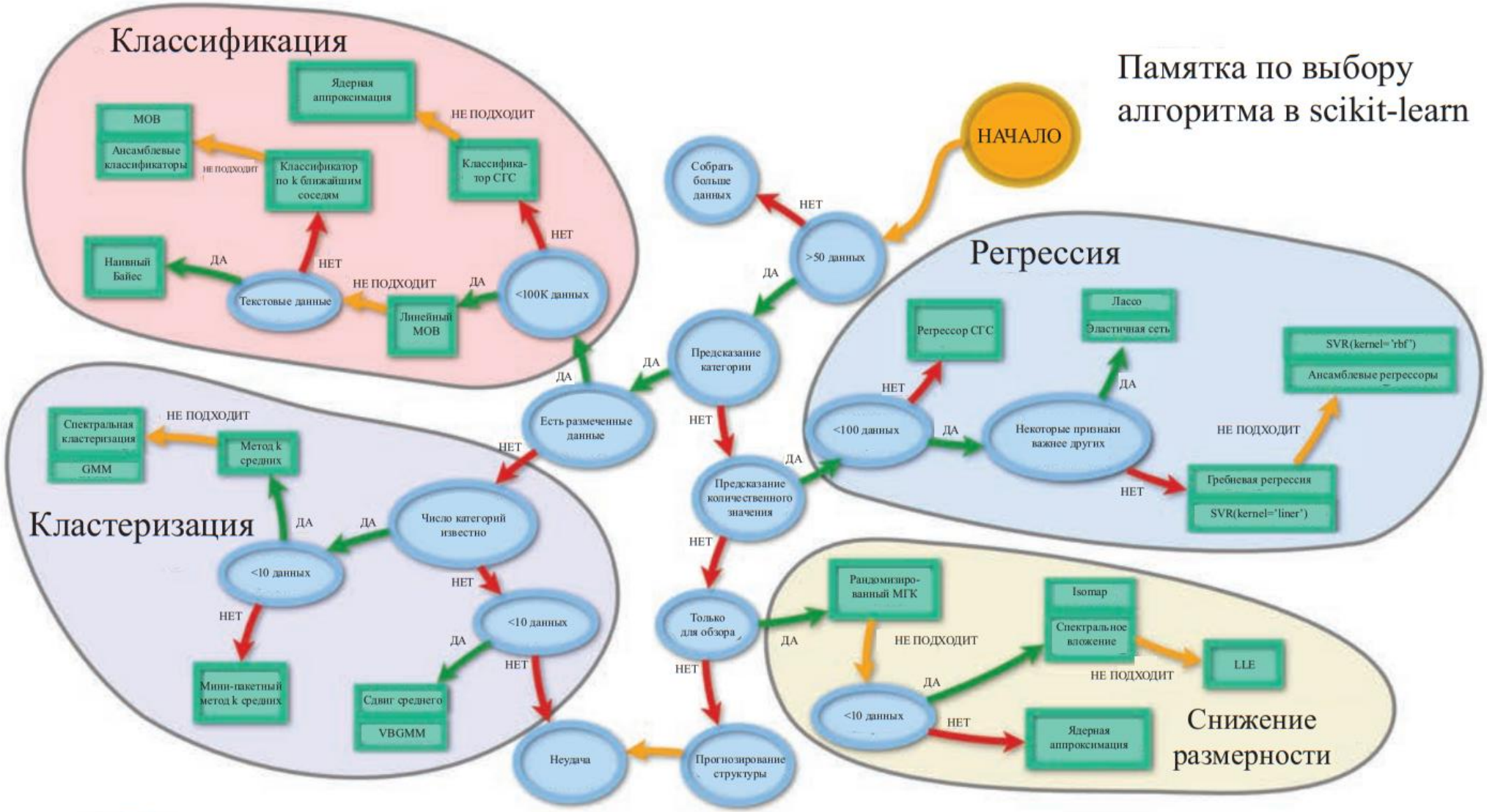
Алгоритмы машинного обучения

- Деревья решений
- Случайные леса
- Ассоциации и обнаружение последовательности
- Градиент повышения и расфасовки
- Опорные векторные машины
- Отображение ближайшего соседа
- K-средства кластеризации
- Самоорганизующиеся карты
- Методы локальной оптимизации поиска
- Максимальное ожидание

Выбор алгоритма машинного обучения

- Интерпретируемость (объяснимость)
  - Черный ящик
- Возможность сохранения набора данных в оперативной памяти.
- Число признаков и данных
- Качественные и количественные признаки
- Нелинейность данных
- Скорость обучения
- Скорость прогнозирования

Памятка по выбору алгоритма в scikit-learn



scikit learn

Регрессия

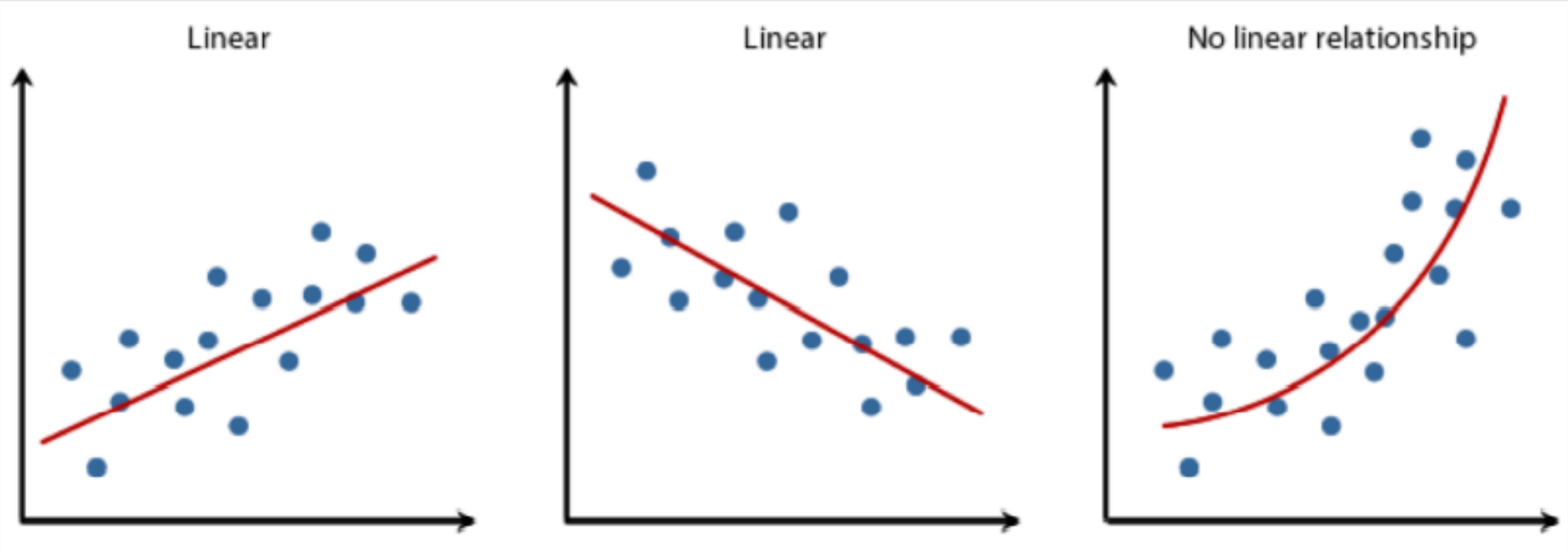
Предсказание стоимости жилья для риэлторской компании

Предсказание времени доставки

Предсказание спроса на такси в конкретном районе в конкретный час завтрашнего дня.

Предсказание ключевых точек лица — т.н. дескрипторов

Линейные и нелинейные зависимости

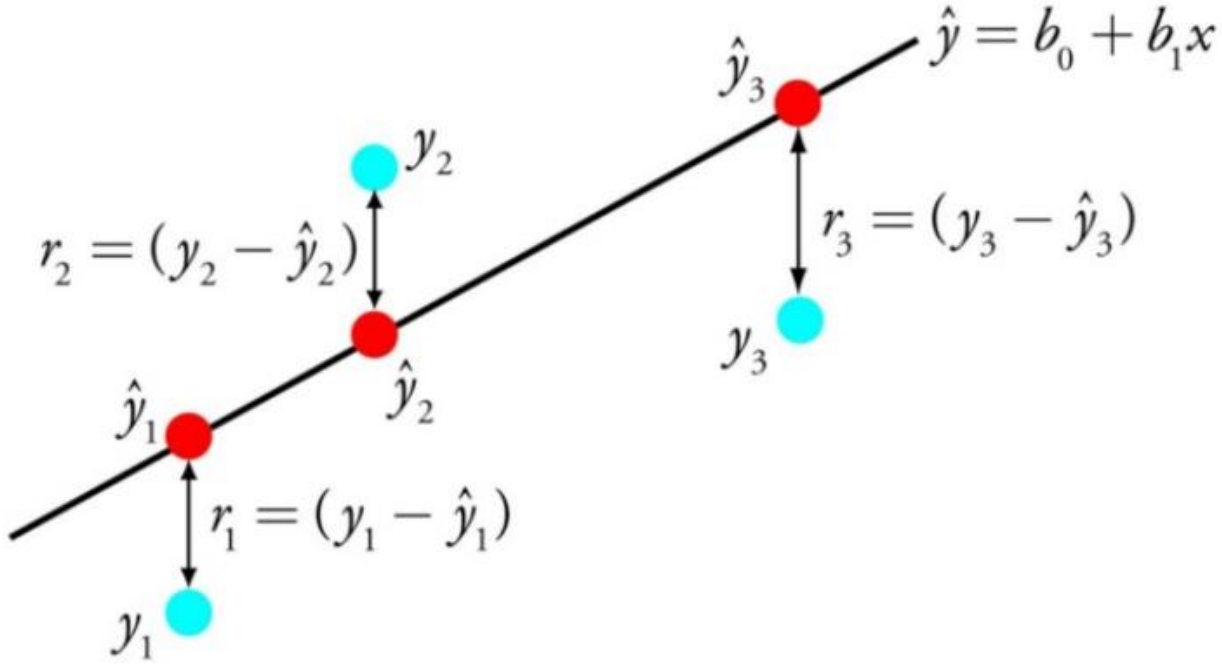


Алгоритмы МО

No.	Machine Learning Model	Category
1	Linear Regression (LR)	Linear
2	Linear Discriminant Analysis (LDA)	Linear
3	Support Vector Machine (SVM)	Linear
4	Quadratic Discriminant Analysis (QDA)	Non-linear
5	Random Forest (RF)	Non-linear
6	K-Nearest Neighbors (KNN)	Non-linear
7	Nearest Centroid	Linear
8	Naive Bayes	Linear
9	Perceptron	Linear
10	Decision Tree (DT)	Non-linear
11	Dummy	Non-linear
12	Neural Networks	Non-linear

Линейная регрессия

В датасете убрать похожие (коррелирующие) переменные и избавиться от шума



Например: `Y = B0 + B1 * X`

Множественная линейная регрессия

$$a(x) = w_0 + w_1 x_1 + .. + w_d x_d + e$$

Матричная запись системы

$$a(x) = XW + e,$$

где X= $\begin{pmatrix} x_{11} & .. & x_{1d} \\ . & . & . \\ x_{l1} & .. & x_{ld} \end{pmatrix}$ регрессионная матрица, а W= $\begin{pmatrix} w_1 \\ .. \\ w_d \end{pmatrix}$-вектор параметров

Функция потерь

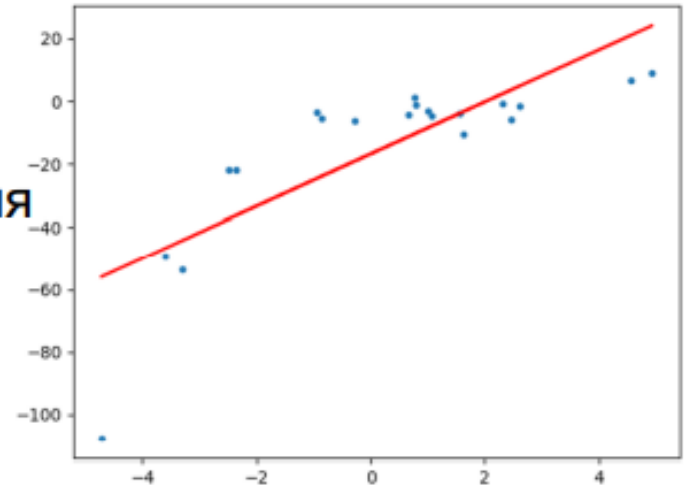
$$Q(w_1, .., w_d) = \sum (< w, x > + e - y)^2 \rightarrow min$$

Полиномиальная регрессия

Полином k-ой степени

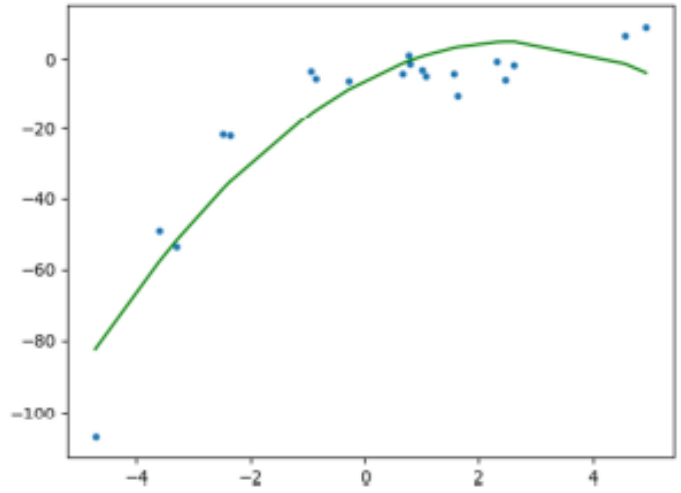
Линейная регрессия

$$Y = \theta_0 + \theta_1 x$$



Полиномиальная регрессия

$$Y = \theta_0 + \theta_1 x + \theta_2 x^2$$

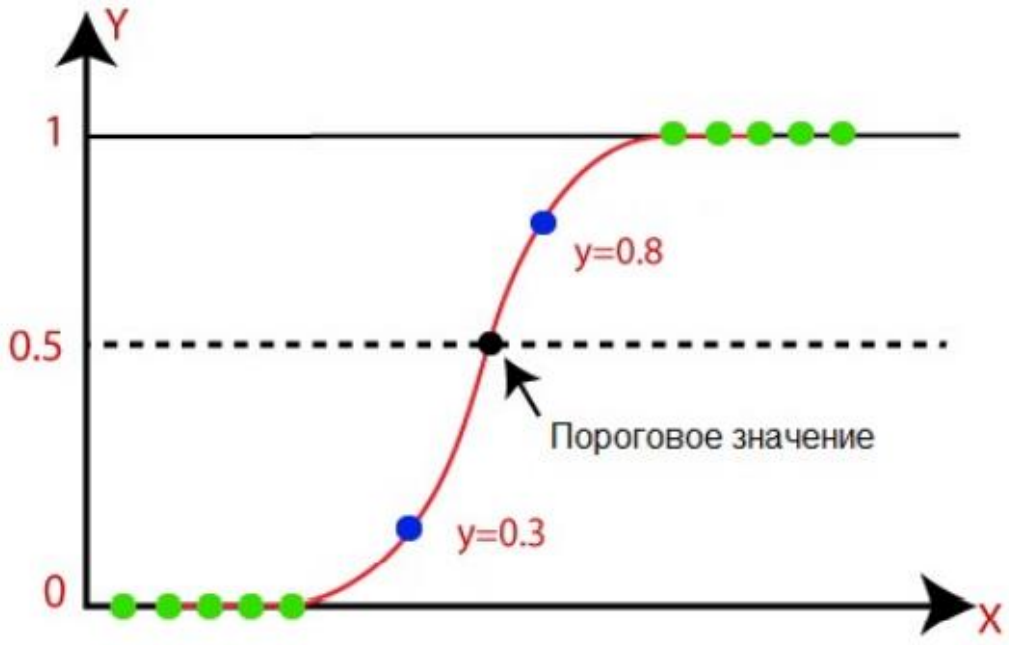


Логистическая регрессия

Бинарная классификация.

Используется, когда зависимая переменная (цель) y - является категориальной (не непрерывной как в случае с линейной регрессией). Например, решение задачи кредитного скоринга (0 - одобрить кредит, 1 - отказать).

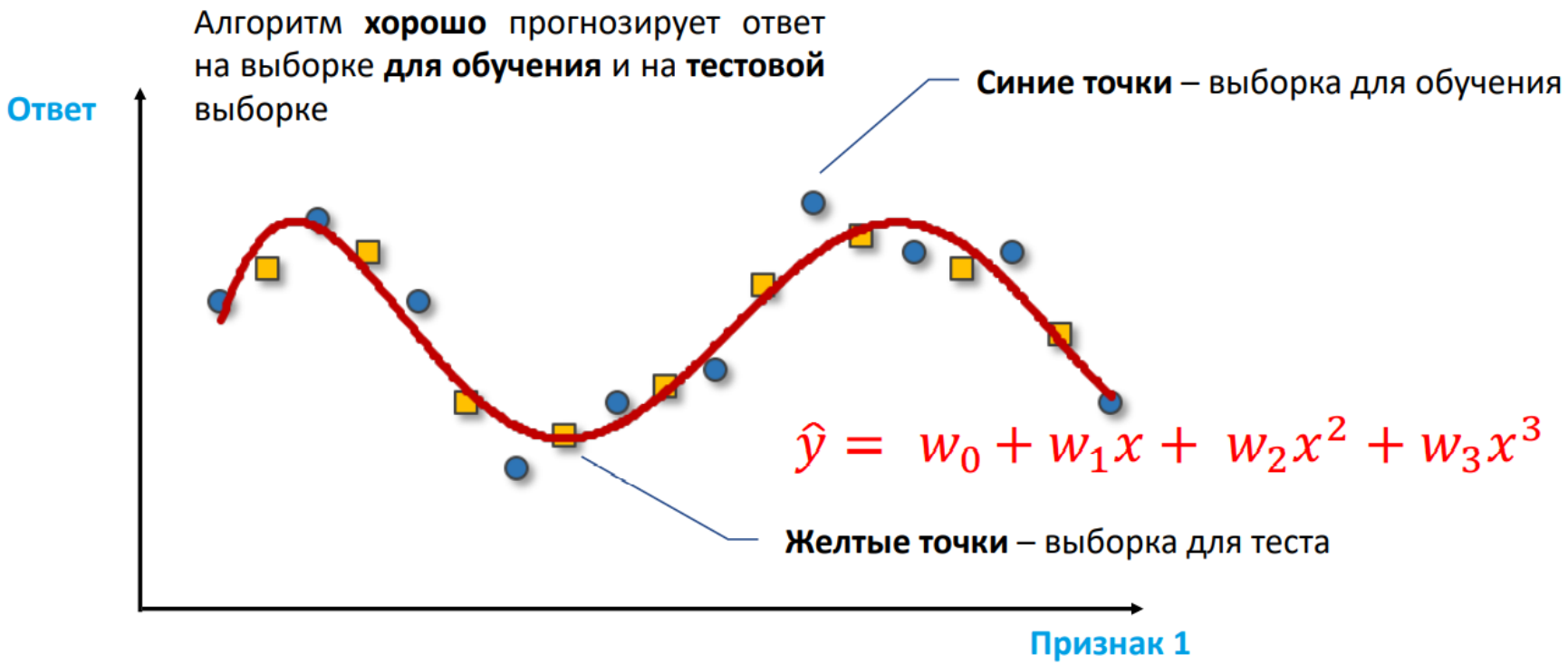
Нелинейная или логистическая функция: вероятность принадлежности образца к классу 0 или 1.



Переобучение

Регрессия хорошего качества

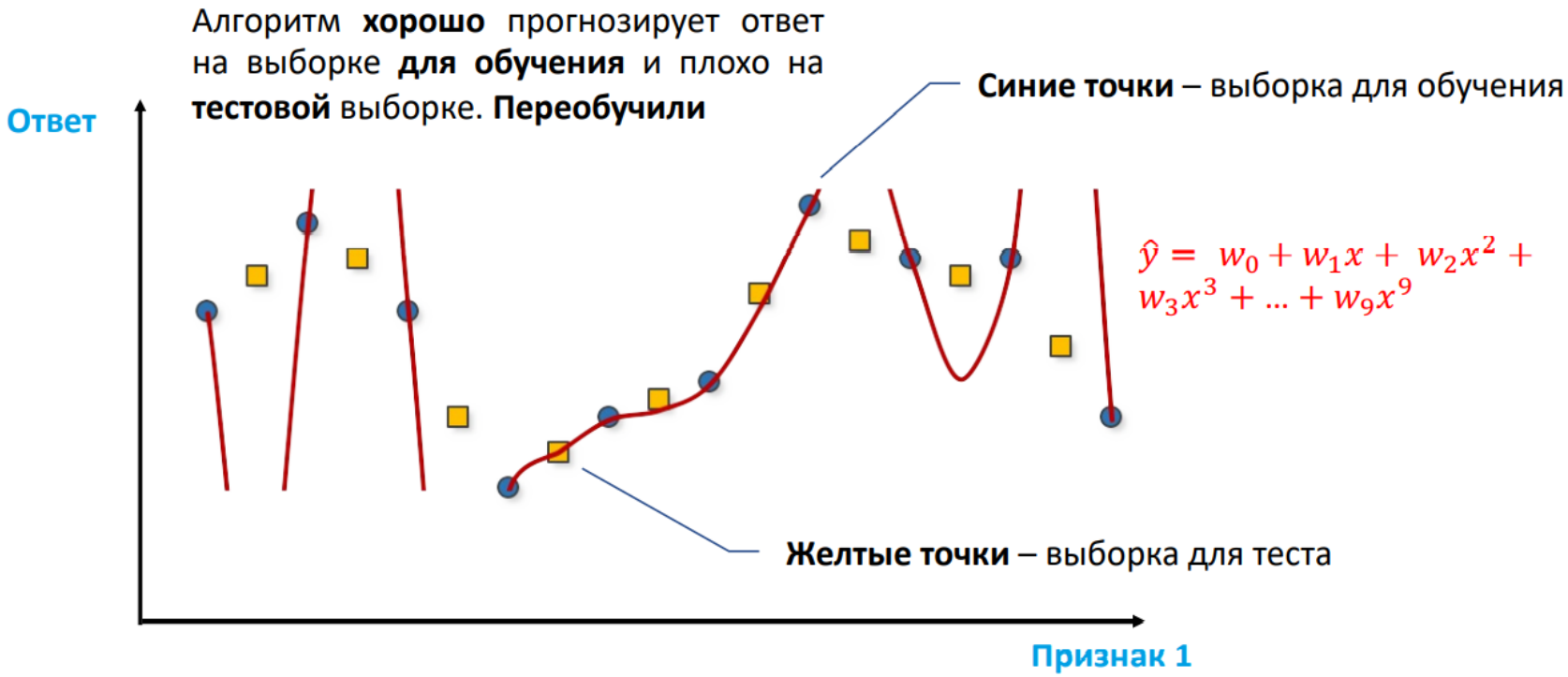
полиномиальная модель 3-й степени



Переобучение

полиномиальная модель 9-й степени

Перефразируя слова Альберта Энштейна можно придерживаться такой рекомендации: Модель должна быть **настолько простой, насколько это возможно, но не проще**.



Регуляризация – противодействие переобучению

Синтаксис для Sklearn

```
import numpy as np
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

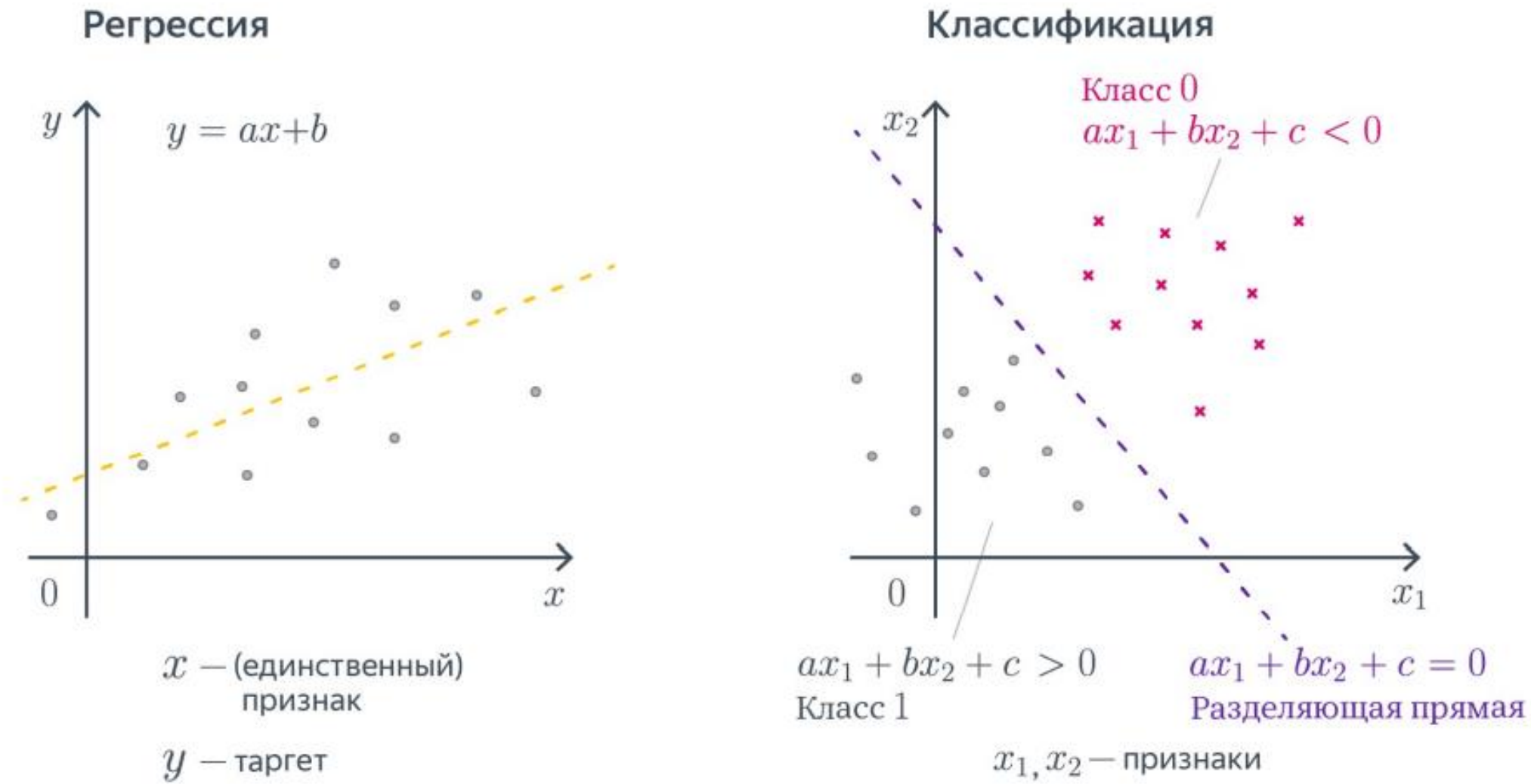
```
x, y = load_boston(return_X_y=True)
x_train, x_test, y_train, y_test =\
    train_test_split(x, y, test_size=0.33, random_state=0)
```

```
>>> regressor = RandomForestRegressor(n_estimators=10, random_stat
e=0)
>>> regressor.fit(x_train, y_train)
```

```
>>> regressor.score(x_train, y_train)
0.9680930547240916
>>> regressor.score(x_test, y_test)
0.8219576562705848
```

```
regressor.predict(x_new)
```

Классификация



Классификация

Решить задачу:

подобрать такой алгоритм, который может ***по признакам*** *каждый объект* ***отнести к определенному классу***

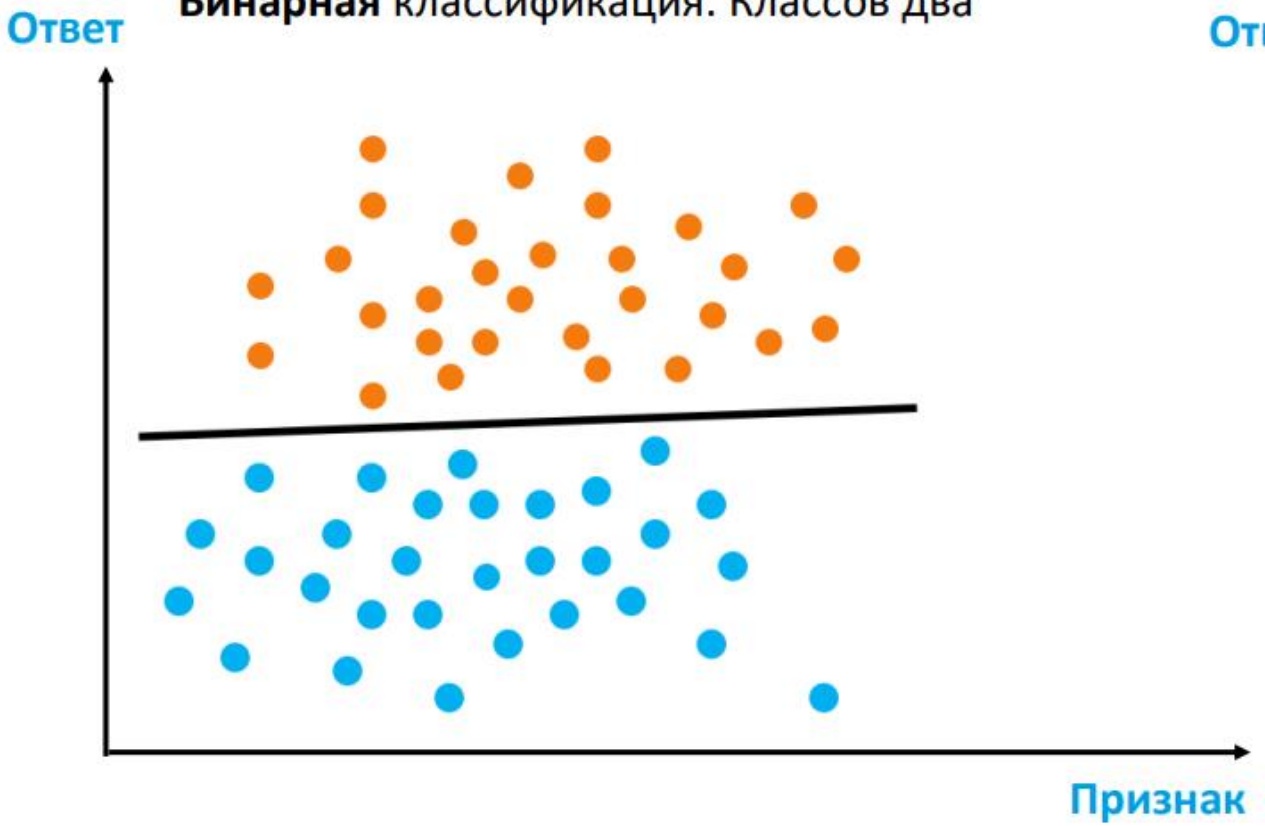
Несколько примеров:

медицина*: предсказать наличие у пациента, обратившегося к врачу, наличие ССЗ*

геологоразведка*: по данным зондирования почв определить наличие полезных ископаемых*

кредитный скоринг*: давать кредит или нет*

Бинарная классификация. Классов два



Множественная классификация. Классов больше чем два

Линейный дискриминантный анализ (LDA)

Состоит из статистических свойств данных, рассчитанных для каждого класса.

Для каждой входной переменной это включает:

- Среднее значение для каждого класса;
- Дисперсию, рассчитанную по всем классам.

Предсказания производятся путём вычисления дискриминантного значения для каждого класса и выбора класса с наибольшим значением.

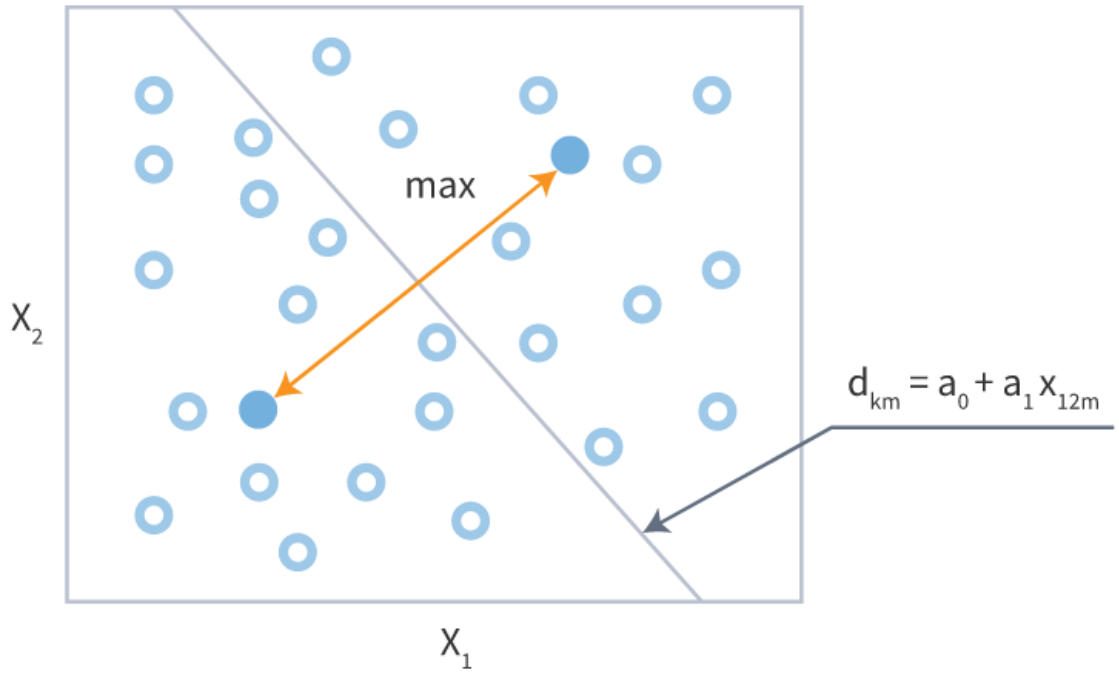
Позволяет изучать различия между двумя и более группами объектов по нескольким признакам одновременно.

Перед началом работы рекомендуется удалить из данных аномальные значения.

Это простой и эффективный алгоритм для задач классификации.

Виды задач дискриминантного анализа

- определение дискриминирующих признаков (т.е. признаков, которые позволяют отнести наблюдение к той или иной группе);
- построение дискриминирующей функции;
- прогнозирование будущих событий, связанных с попаданием объекта в ту или иную группу на основе значений его признака.



Метод опорных векторов (SVM)

Гиперплоскость — это линия, разделяющая пространство входных переменных. В методе опорных векторов гиперплоскость выбирается так, чтобы наилучшим образом разделять точки в плоскости входных переменных по их классу: 0 или 1.

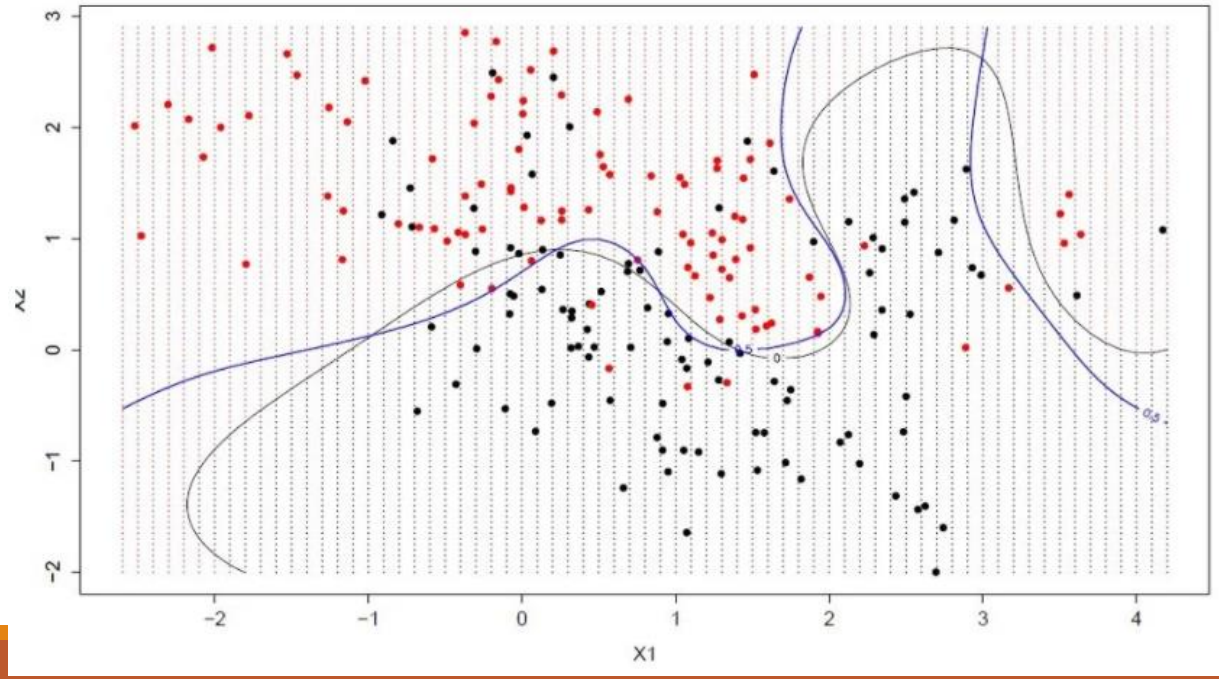
Во время обучения алгоритм ищет коэффициенты, которые помогают лучше разделять классы гиперплоскостью.

Лучшая или оптимальная гиперплоскость,

разделяющая два класса, — это

линия с наибольшей разницей.

Эти точки называются **опорными векторами**.



Метод опорных векторов

Гиперплоскость является n -1 подпространством в n-мерном пространстве.

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x_i, x_{i'}),$$

где β_0 — смещение; S — набор всех опорно-векторных наблюдений; α_i — модельные параметры; (x_i, $x_{i'}$) — пары двух опорно-векторных наблюдений.

К — это ядерная функция, которая сравнивает сходство между x_i и $x_{i'}$.

Линейная гиперплоскость – линейное ядро

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j},$$

где p — количество признаков.

```
model = SVC(kernel='linear', C=C)
```

poly

rbf

Нелинейная гиперплоскость – полиномиальное ядро

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^2$$

Радиально-базисное ядро

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} x_{i'j})^2\right)$$

Наивный Байесовский классификатор

Модель состоит из двух типов вероятностей, которые рассчитываются с помощью тренировочных данных:

1. Вероятность каждого класса.
2. Условная вероятность для каждого класса при каждом значении x.

Предсказание с новыми данными на основе теоремы Байеса

Хорошо работает с категориальными признаками

Каждая входная переменная независимая

Пример

•«Путевки по низкой цене»
•«Акция! Купи шоколадку и получи телефон в подарок»

«В магазине гора яблок. Купи семь килограмм и шоколадку»

•«Завтра состоится собрание»
•«Купи килограмм яблок и шоколадку»

<u>Вычисляем для каждого слова письма:</u>

$P(x_i|\text{Спам})$

$P(x_i|\text{Не спам})$

Теорема Байеса

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

- P(A | B) – апостериорная вероятность (что A из B истинно)
- P(A) – априорная вероятность (независимая вероятность A)
- P(B | A) – вероятность данного значения признака при данном классе. (что B из A истинно)
- P(B) – априорная вероятность при значении нашего признака. (независимая вероятность B)

K-ближайших соседей (KNN)

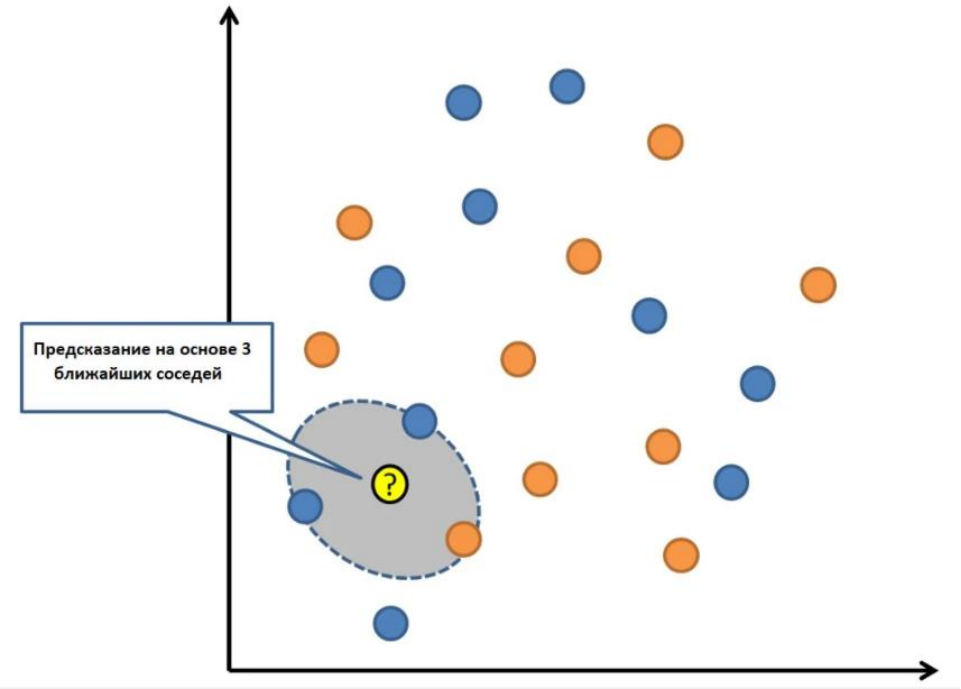
Предсказание для новой точки делается путём поиска K ближайших соседей в наборе данных и суммирования выходной переменной для этих K экземпляров.

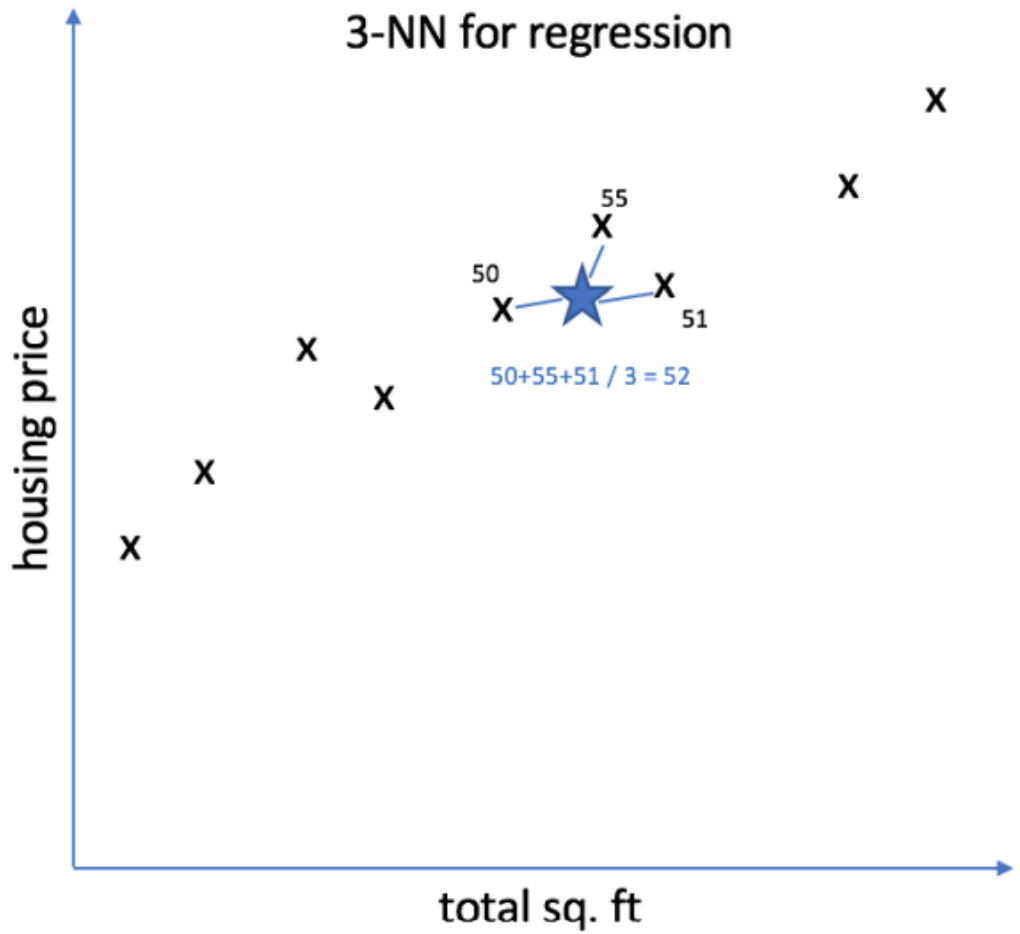
Сходство между экземплярами: Евклидово расстояние.

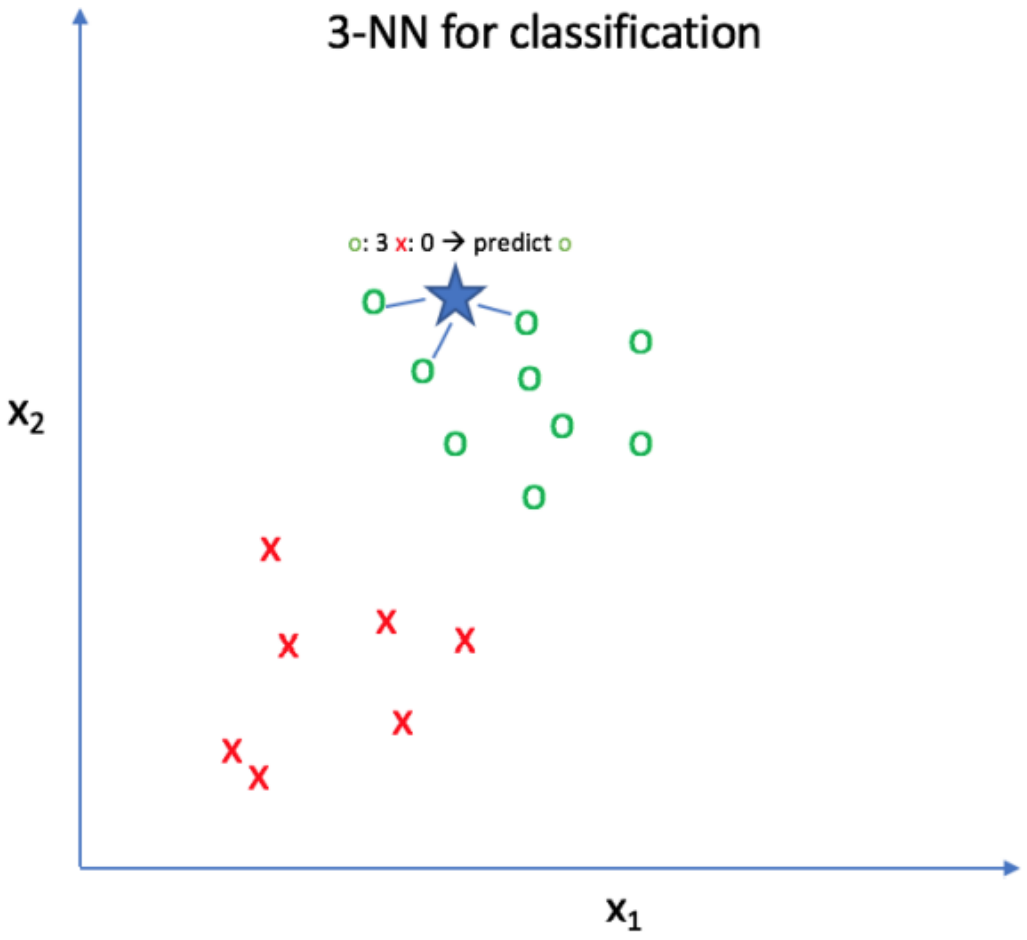
$$d(p,q) = \sqrt[2]{(q_1 - p_1)^2 + \ldots + (q_n - p_n)^2}$$

KNN может потребовать много памяти для хранения всех данных, но зато быстро сделает предсказание.

Идея ближайших соседей может плохо работать с многомерными данными.




3-NN for regression
housing price
total sq. ft
55
50
51
50+55+51 / 3 = 52

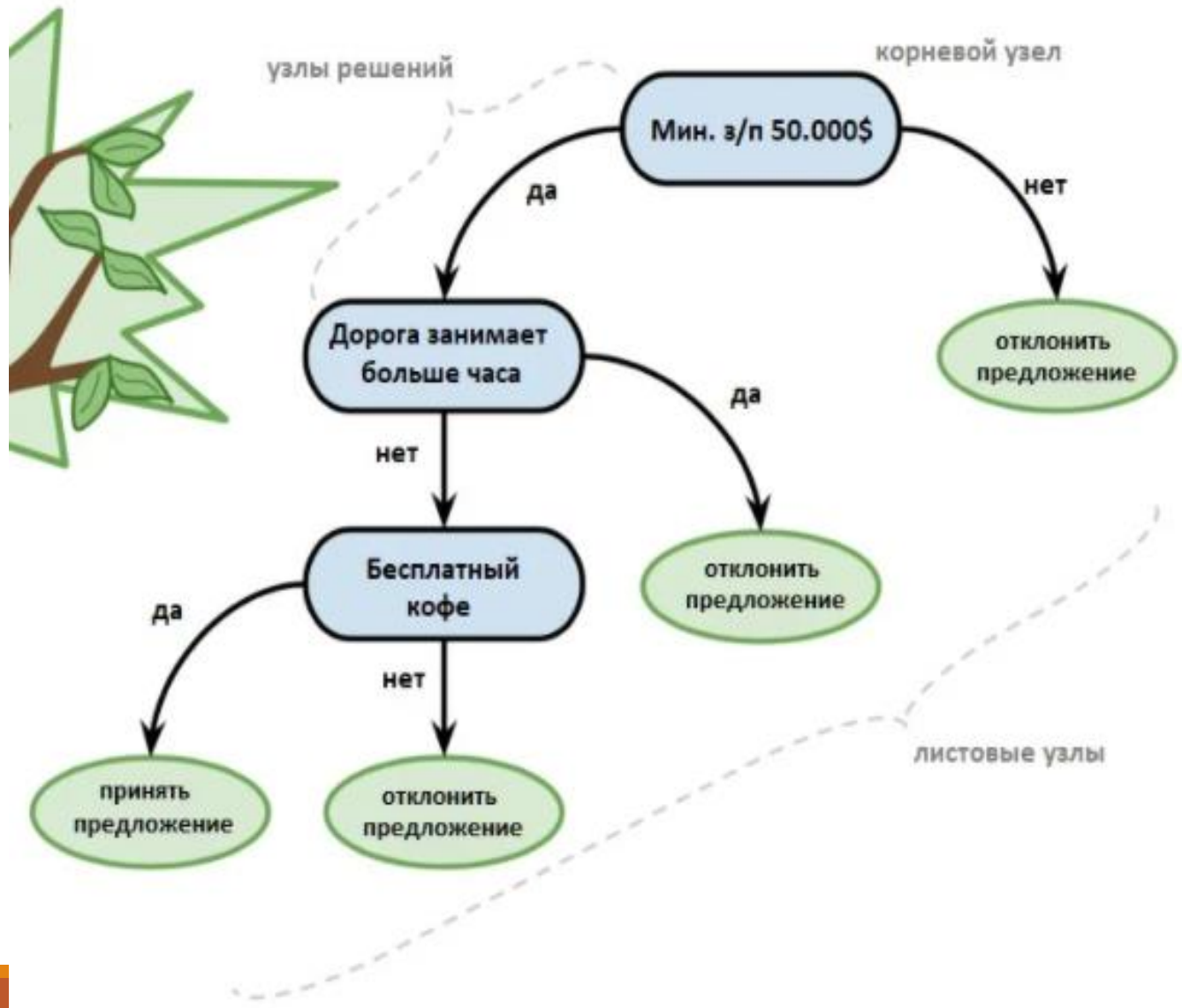

3-NN for classification
o: 3 x: 0 → predict o
x_2
x_1

Деревья принятия решений

Дерево решений можно представить в виде двоичного дерева.

Каждый узел представляет собой входную переменную и точку разделения для этой переменной.

Листовые узлы — это выходная переменная, которая используется для предсказания.



Легко интерпретируются

Проверка простых условий

Ансамблевые методы

Метод машинного обучения, где **несколько моделей обучаются** для решения одной и той же проблемы и объединяются для получения лучших результатов называется **ансамблевым методом**.

Множество **слабых учеников** являются строительными блоками для более сложных моделей.

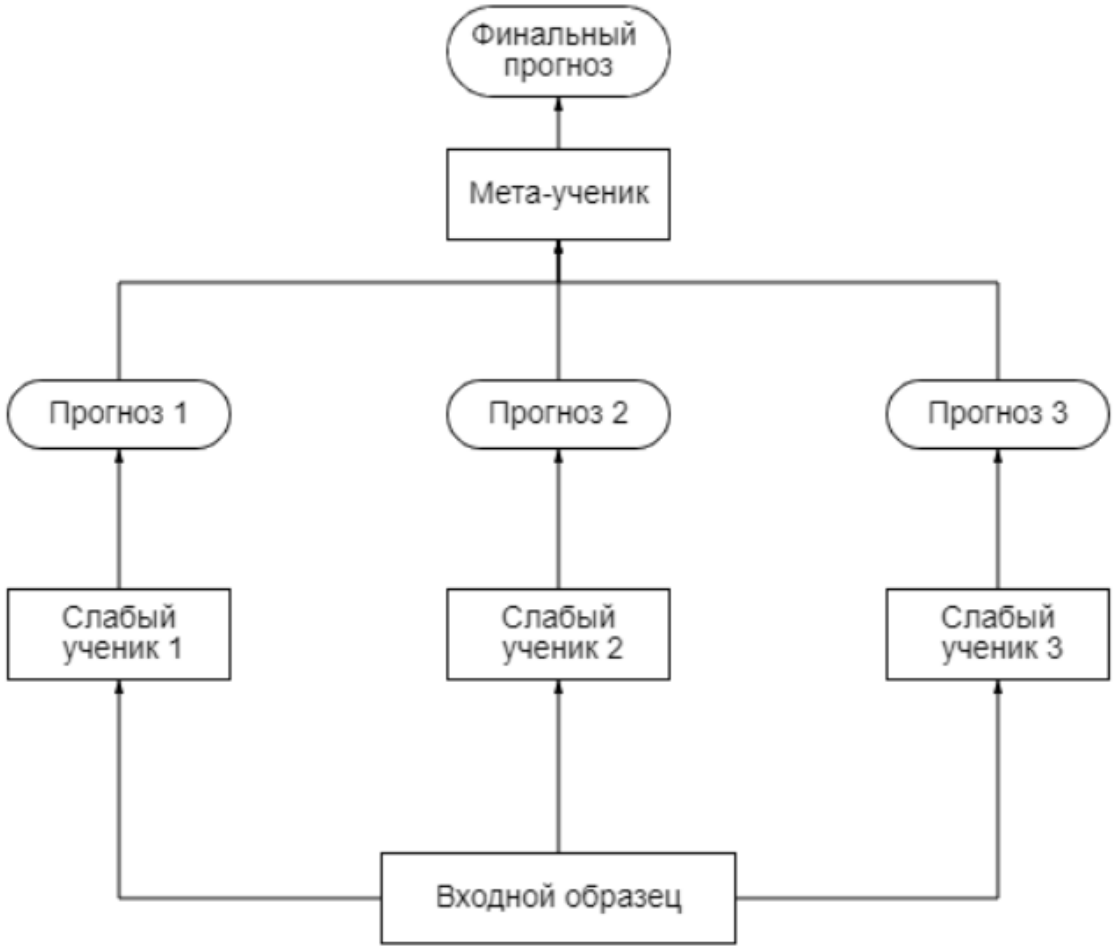
Объединение слабых учеников для улучшения качества модели, уменьшения смещения или разброса, называется **сильным учеником**.

Виды: стекинг, бэггинг, бустинг

Библиотека XGBoost

https://habr.com/ru/post/571296/

Стекинг



Обучение мета-модели

На вход всех слабых прогнозаторов подаётся обучающий набор, каждый прогноз идёт к финальной модели, которая называется смеситель, мета-ученик или мета-модель, после чего та вырабатывает финальный прогноз

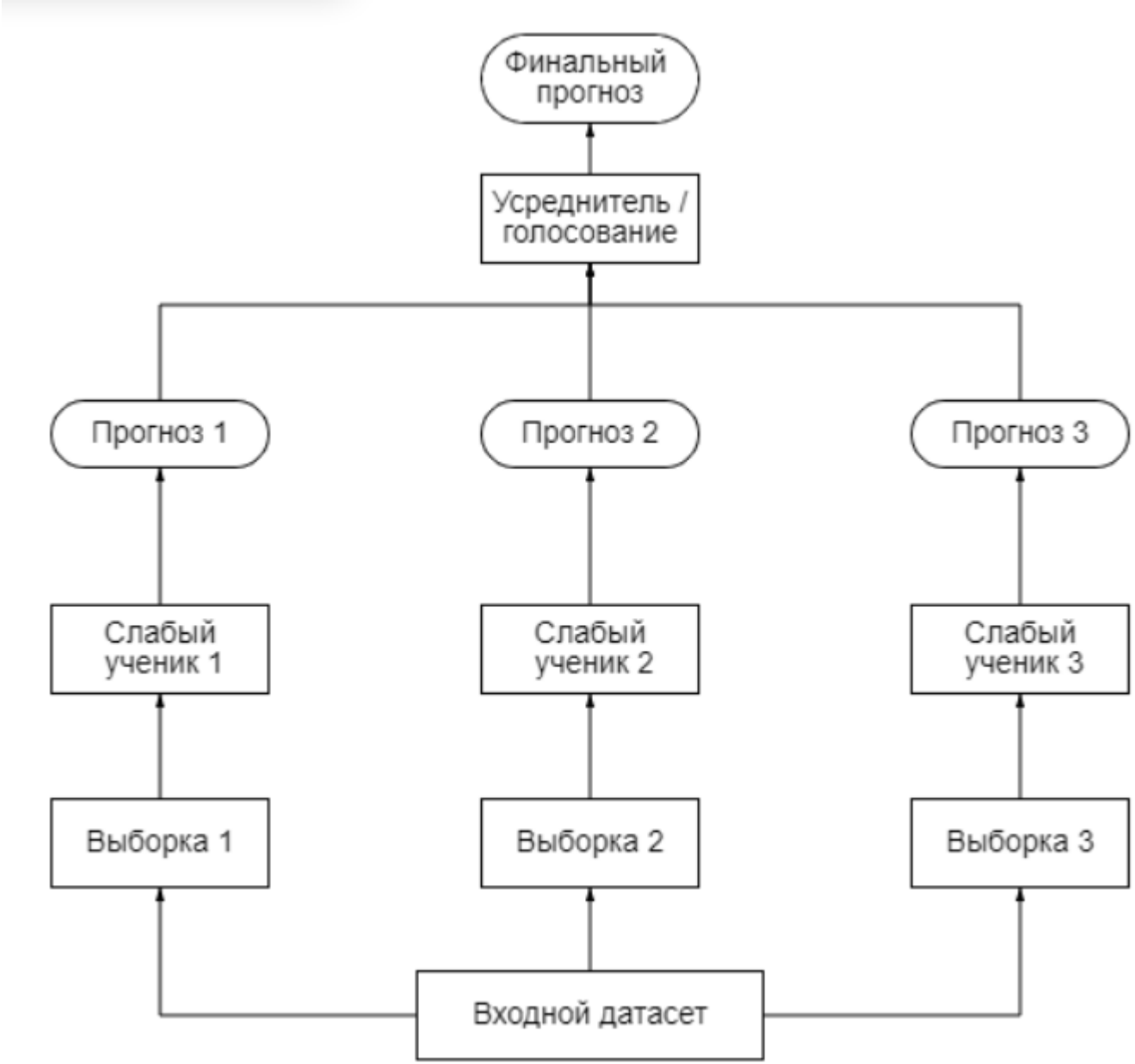
```
from sklearn.ensemble import StackingClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

data, target = load_breast_cancer(return_X_y=True)
estimators = [('lr', LogisticRegression()), ('dt', DecisionTreeClassifier())]
modelClf = StackingClassifier(estimators=estimators, final_estimator=SVC())
X_train, X_valid, y_train, y_valid = train_test_split(data, target, test_size=0.3,
random_state=12)
modelClf.fit(X_train, y_train)
print(modelClf.score(X_valid, y_valid))
```

Бэггинг

Обучение нескольких одинаковых моделей **на разных образцах**.

Распределение выборки неизвестно, поэтому модели получатся разными.

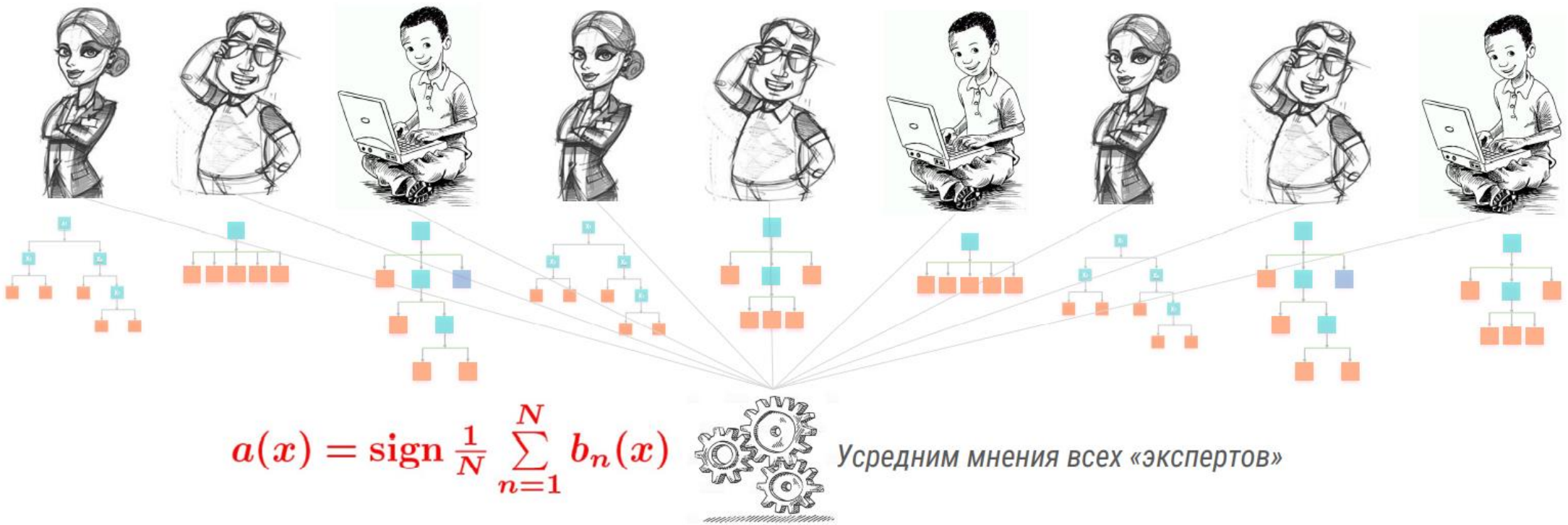
Бутстрэп - это случайный выбор данных из датасета и представление их в модель, затем данные возвращаются в датасет и процесс повторяется.



Случайный лес

При построении деревьев для создания каждого узла выбираются случайные признаки.

Параллельность обучения



$$a(x) = \text{sign}\, \frac{1}{N} \sum_{n=1}^{N} b_n(x)$$

Усредним мнения всех «экспертов»

```
>>> import numpy as np
>>> from sklearn.datasets import load_wine
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.model_selection import train_test_split
>>> x, y = load_wine(return_X_y=True)
>>> x_train, x_test, y_train, y_test =\
...     train_test_split(x, y, test_size=0.33, random_state=0)
>>> classifier = RandomForestClassifier(n_estimators=10, random_st
ate=0)
>>> classifier.fit(x_train, y_train)
```

```
>>> classifier.score(x_train, y_train)
1.0
>>> classifier.score(x_test, y_test)
1.0
```

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier


data, target = load_breast_cancer(return_X_y=True)
modelClf = BaggingClassifier(base_estimator=LogisticRegression(),
n_estimators=50, random_state=12)
X_train, X_valid, y_train, y_valid = train_test_split(data, target, test_size=0.3,
random_state=12)
modelClf.fit(X_train, y_train)
print(modelClf.score(X_valid, y_valid))
```

Бустинг

Создание сильного классификатора на основе нескольких слабых.

В данных отсутствуют аномалии

Последовательность обучения

Сначала создаётся одна модель, затем другая модель, которая пытается исправить ошибки в первой.

Модели добавляются до тех пор, пока тренировочные данные не будут идеально предсказываться или пока не будет превышено максимальное количество моделей.

Адаптивный бустинг и Градиентный бустинг

Адаптивный бустинг (AdaBoost)

Алгоритм обучает первую базовую модель(допустим деревья решений) на тренировочном наборе.

Относительный вес некорректно предсказанных значений увеличивается.
Веса хорошо классифицированных объектов уменьшаются относительно весов неправильно классифицированных объектов.

На вход второй базовой модели подаются обновлённые веса и модель обучается, после чего вырабатываются прогнозы и цикл повторяется.

Итеративный метод оптимизации

Модели, которые работают лучше, имеют больший вес в окончательной модели ансамбля.

```
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier


data, target = load_breast_cancer(return_X_y=True)
modelClf =
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2),
n_estimators=100, random_state=12)
X_train, X_valid, y_train, y_valid = train_test_split(data, target, test_size=0.3,
random_state=12)
modelClf.fit(X_train, y_train)
print(modelClf.score(X_valid, y_valid))
```

Градиентный бустинг

Обучает слабые модели последовательно, исправляя ошибки предыдущих.

Результатом градиентного бустинга является средневзвешенная сумма результатов моделей.

Отличие от Adaboost - это способ изменения весов (итеративный метод – градиентный метод)

Обобщение адаптивного бустинга для дифференцируемых функций.

```
from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.ensemble import GradientBoostingClassifier


data, target = load_breast_cancer(return_X_y=True)

modelClf = GradientBoostingClassifier(max_depth=2, n_estimators=150,
random_state=12, learning_rate=1)

X_train, X_valid, y_train, y_valid = train_test_split(data, target, test_size=0.3,
random_state=12)

modelClf.fit(X_train, y_train)

print(modelClf.score(X_valid, y_valid))
```

Метрики качества модели

Матрица ошибок отражает количество ошибочно/верно определенных наблюдений.

По матрице ошибок можно вычислить такие метрики, как **Accuracy, Precision, Recall, F1**

МАТРИЦА ОШИБОК

	$y = 1$ (Фактический класс)	$y = 0$ (Фактический класс)
$\hat{y} = 1$ (Прогнозный класс)	True Positive (TP)	Type 1 Error (FP) *Ложная тревога*
$\hat{y} = 0$ (Прогнозный класс)	Type 2 error (FN) *Пропуск цели*	True Negative (TN)

Метрики качества модели

Accuracy показывает долю верно определенных наблюдений.

$$Accuracy = \frac{TP + TN}{TN + FP + FN + TP}$$

$$Accuracy = \frac{\text{Количество верно определенных наблюдений}}{\text{Количество всех наблюдений}}$$

Precision отражает точность классификатора в определении единиц.

Минимизируем ложные срабатывания (ложные тревоги)

$$Precision = \frac{TP}{TP + FP}$$

$$Precision = \frac{\text{Количество верно определенных "единиц"}}{\text{Кол} - \text{во наблюдений, помеченных моделью как "единица"}}$$

Метрики качества модели

Recall показывает процент фактических «единиц», определяемых моделью.

Минимизируем пропуски цели

$$Recall = \frac{TP}{TP + FN}$$

$$Recall = \frac{\text{Количество верно определенных "единиц"}}{\text{Количество всех "единиц" выборки}}$$

F1 описывает качество модели в целом, в равной мере учитывая Precision и Recall.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Гармоническое среднее

Кластеризация

нужно сгруппировать все похожие объекты на группы

Решить задачу:

*подобрать такой алгоритм, который для каждого объекта **проставит метку** так, чтобы **объекты с одной меткой** были в определенном смысле **похожи** друг на друга, а **объекты с разными метками отличались***

Несколько примеров:

***Сегментация** новостей (спорт, экономика…) или клиентов*

***Сжатие** изображений: Есть изображение с палитрой в 32 цвета. Тогда кластеризация найдёт все «примерно красные» пиксели изображения, высчитает из них «средний красный» и заменит все красные на него. Меньше цветов — меньше файл.*

***Геология**: найти на карте схожие по структуре горные породы*

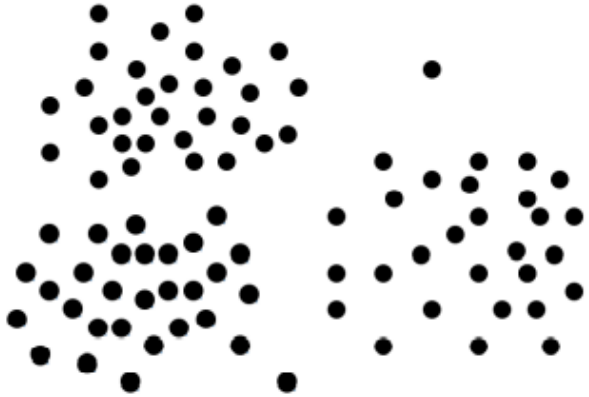
Для проведения кластеризации существуют разные методы

- В зависимости от формы кластеров
- Возможности делать кластеры вложенными друг в друга
- Основная задача кластеризации или вспомогательная
- Жесткая кластеризация (относим к одному классу) или мягкая – к нескольким с весами

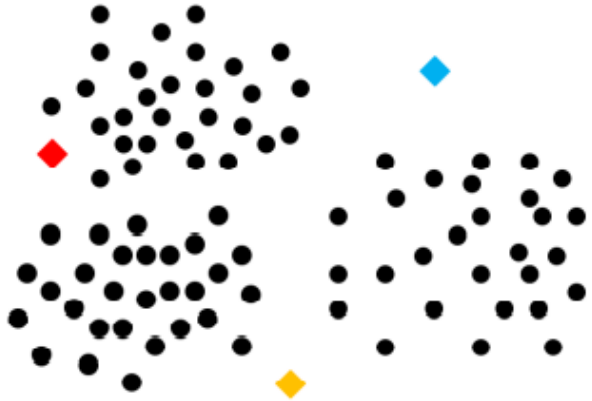
K-means (K-средних)

Силуэт выборки показывает насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров

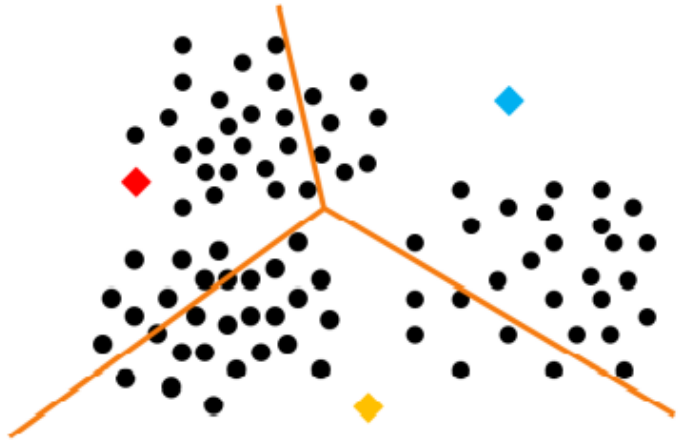
Шаг 1. Выбрать количество кластеров k, которое нам кажется оптимальным для наших данных



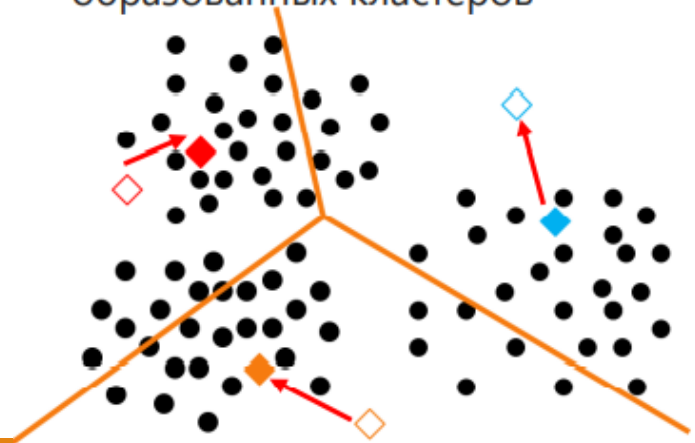
Шаг 2. Определить среди наших объектов те, которые будут являться центрами наши k кластеров (можно случайно)



Шаг 3. Для каждого объекта посчитать, к какому центру он ближе



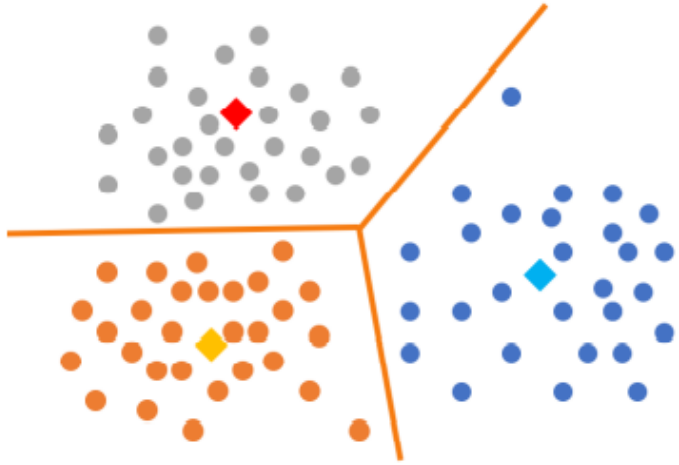
Шаг 4. Пересчитать центры образованных кластеров



Шаг 5. Повторяем шаги 3 и 4

либо фиксированное число раз, либо до тех пор, пока смещение центра относительно предыдущего положения превышает какое-то заранее заданного небольшого значения

Шаг 6. Итоговый результат кластеризации



Формальное описание алгоритма

На вход: обучающая выборка $X = (x_i, y_i)_{i=1}^{l}$

Обучение: Запоминание выборки

Применение:

1) Выбираем новый объект z;

2) Инициализируем k путем выбора оптимального числа соседей;

3) Для каждого образца в данных вычислим расстояние $\rho(z, X_i)$

4) Сортируем объекты по близости расстояний в порядке возрастания метрик

$$\rho(z, X_1) \leq \rho(z, X_2) \leq .. \leq \rho(z, X_l)$$

5) Выбираем класс, наиболее популярный среди k ближайших соседей:

$$a(x) = \arg\max_{y \epsilon \mathbb{Y}} \sum_{i=1}^{k} [x_{i,z} = y],$$

где $x_{i,z}$-объект обучающей выборки, который является $i-$ м соседом объекта z,

[ ] **- Скобка Айверсона** — функция, возвращающая 1 для истинного высказывания, и 0 для ложный:

ВЫБОР МЕТРИКИ $\rho(z, X_i)$

Зададим метрику $\rho(x, y)$

- *Аксиома тождества* $\rho(x, y) = 0 \iff x = y$.
- *Аксиома симметрии* $\rho(x, y) = \rho(y, x)$
- *Неравенство треугольника* $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$

В случае числовых признаков:

- **Метрика Минковского** $\boldsymbol{\rho(x, z) = \sqrt[p]{\sum_{i=1}^{l}(x_i - z_i)^p}}$

Категориальные признаки:

- **Расстояние Хэмминга** $\boldsymbol{\rho(x, z) = \sum_{i=1}^{l}[x_i \neq z_i]}$

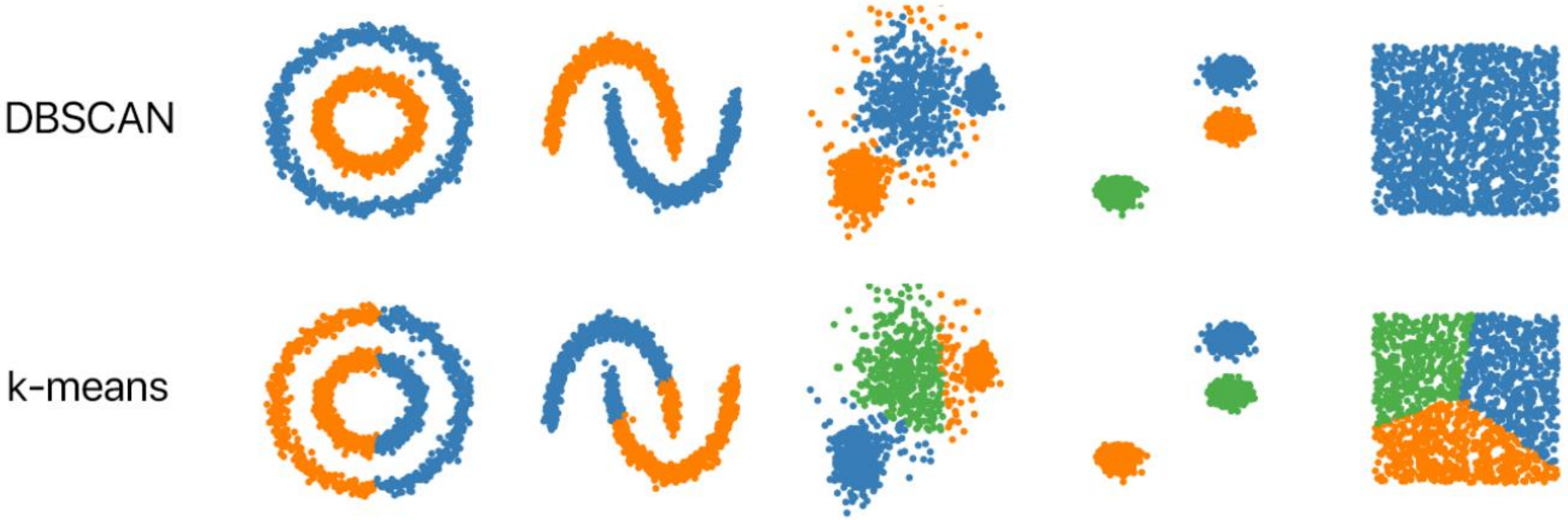
Агломеративная кластеризация

1. Каждая точка свой кластер
2. Сортируем попарные расстояния между центрами кластеров по возрастанию
3. Берём **пару ближайших кластеров**, склеиваем их в один и пересчитываем центр кластера
4. Повторяем п. 2 и 3 до тех пор, пока все данные не склеятся в один кластер

Алгоритм DBSCAN

плотностный **алгоритм** для кластеризации пространственных данных с присутствием шума

найти области высокой плотности, которые отделены друг от друга областями низкой плотности.



Алгоритм DBSCAN

Если дан набор точек в некотором пространстве, алгоритм **группирует вместе точки, которые тесно расположены** (точки со многими близкими соседями]), помечая **как выбросы точки**, которые находятся одиноко в областях с малой плотностью (ближайшие соседи которых лежат далеко).

Ключевые шаги алгоритма

1. Находим точки в epsilon окрестности каждой точки и выделяем основные точки которые должны образовывать плотную область minPts соседями.
2. Находим связные компоненты основных точек на графе соседей, игнорируя все неосновные точки.
3. Назначаем каждую неосновную точку, ближайшему кластеру, если кластер является epsilon -соседним, в противном случае считаем точку шумом.

```
db = DBSCAN(eps=0.2, min_samples=10)
```

Метрики качества кластеризации

Внешние

- используют информацию об истинном разбиении на кластеры

Внутренние

- не используют никакой внешней информации и оценивают качество кластеризации, основываясь только на наборе данных

Силуэт

a - среднее расстояние от данного объекта до объектов из того же кластера

b - среднее расстояние от данного объекта до объектов из ближайшего кластера

$$s = \frac{b - a}{\max(a, b)}$$

Силуэт показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров (лучшее значение-1, худшее – -1.)

Синтаксис

```
>>> import numpy as np
>>> from sklearn.cluster import KMeans
>>> x = np.array([(0.0, 0.0),
...               (9.9, 8.1),
...               (-1.0, 1.0),
...               (7.1, 5.6),
...               (-5.0, -5.5),
...               (8.0, 9.8),
...               (0.5, 0.5)])
```

```
>>> cluster_analyzer = KMeans(n_clusters=3, init='k-means++')
```

```
>>> cluster_analyzer.fit(x)
```

```
>>> cluster_analyzer.cluster_centers_
array([[ 8.33333333,  7.83333333],
       [-0.16666667,  0.5       ],
       [-5.        , -5.5       ]])
>>> cluster_analyzer.labels_
array([1, 0, 1, 0, 2, 0, 1], dtype=int32)
```

Сохранение модели

```
>>> from sklearn import svm
>>> from sklearn import datasets
>>> clf = svm.SVC()
>>> iris = datasets.load_iris()
>>> X, y = iris.data, iris.target
>>> clf.fit(X, y)
```

```
>>> from sklearn.externals import joblib
>>> joblib.dump(clf, 'filename.pkl')
```

```
>>> clf = joblib.load('filename.pkl')
```

Пример создания проекта по МО

- Очистка и форматирование данных.
- Разведочный анализ данных.
- Конструирование и выбор признаков.
- Сравнение метрик нескольких моделей машинного обучения.
- Гиперпараметрическая настройка лучшей модели.
- Оценка лучшей модели на тестовом наборе данных.
- Интерпретирование результатов работы модели.

https://habr.com/ru/company/nix/blog/425253/

Описание задачи

Датасет: Данные об энергоэффективности зданий в Нью-Йорке

Цель: Построение модели, которая прогнозирует количество баллов Energy Star Score для конкретного здания, и интерпретирует результаты для поиска факторов, влияющих на итоговый балл.

Задача: Регрессия

1. Очистка и форматирование данных

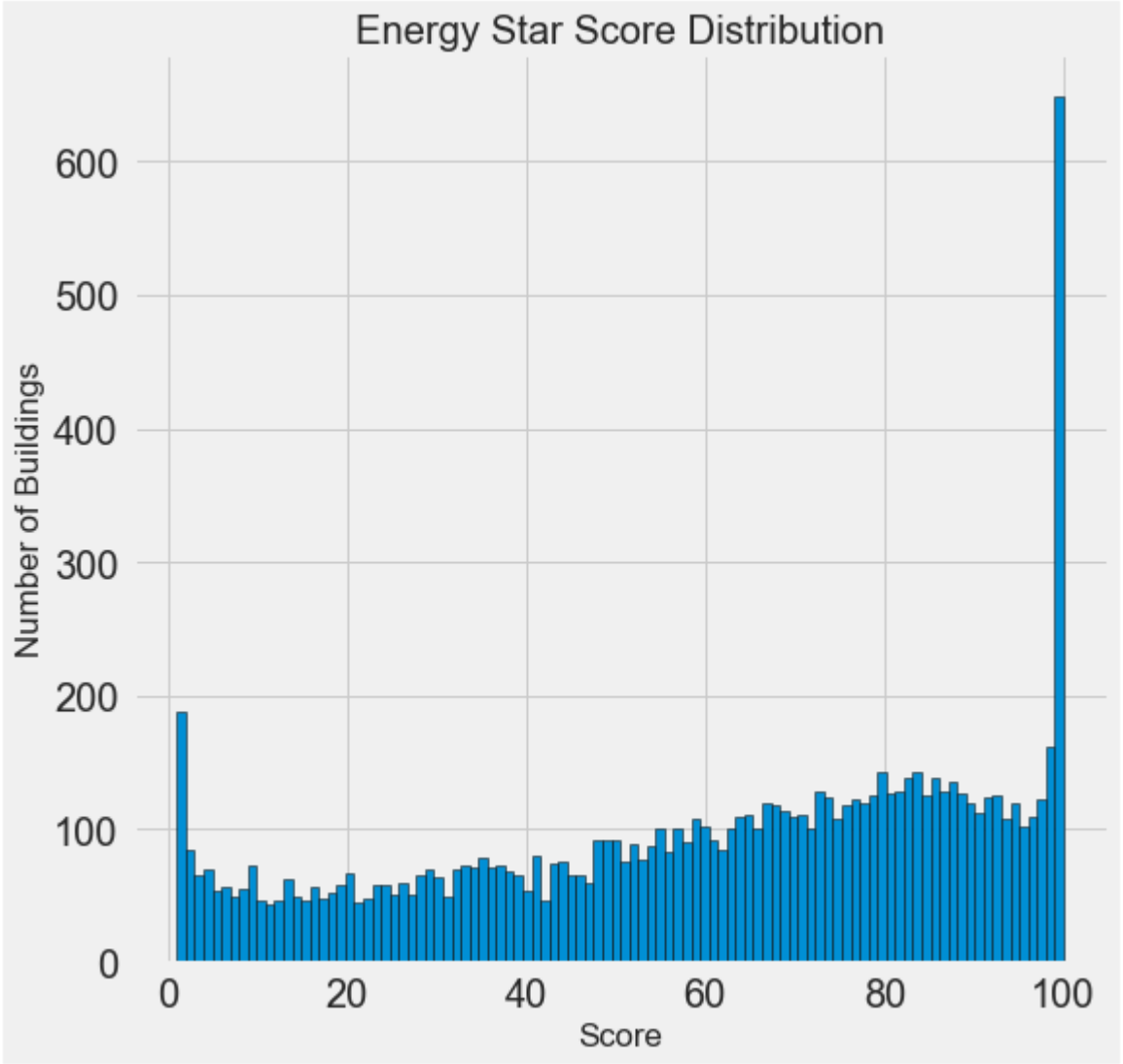
3rd Largest Property Use Type - Gross Floor Area (ft²)	Year Built	Number of Buildings - Self-reported	Occupancy	Metered Areas (Energy)	Metered Areas (Water)	ENERGY STAR Score	Site EUI (kBtu/ft²)	Weather Normalized Site EUI (kBtu/ft²)	Weather Normalized Site Electricity Intensity (kWh/ft²)	Weather Normalized Site Natural Gas Intensity (therms/ft²)
Not Available	1963	2	100	Whole Building	Not Available	Not Available	305.6	303.1	37.8	Not Available
Not Available	1969	12	100	Whole Building	Whole Building	55	229.8	228.8	24.8	2.4
Not Available	1924	1	100	Not Available	Not Available	Not Available	Not Available	Not Available	Not Available	Not Available

Отсутствующие значения Not Available => np.nan или удаление
Конвертирование типа данных Object => числовой
Удаление аномальных данных
Заполнение отсутствующих значений
Масштабирование признаков (нормализация)

2. Разведочный анализ данных

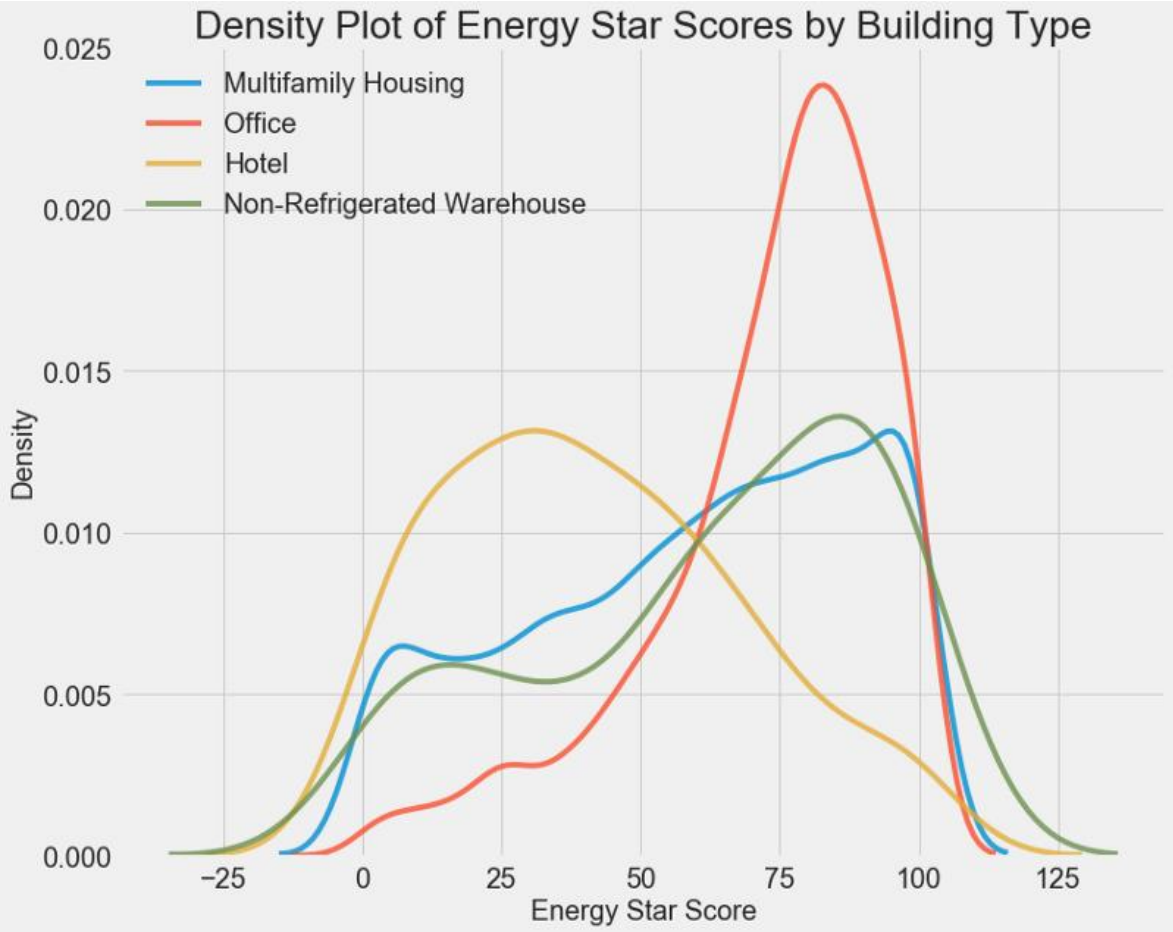
Вычисляется статистика и поиск в данных тенденции, аномалий, шаблонов или взаимосвязей.

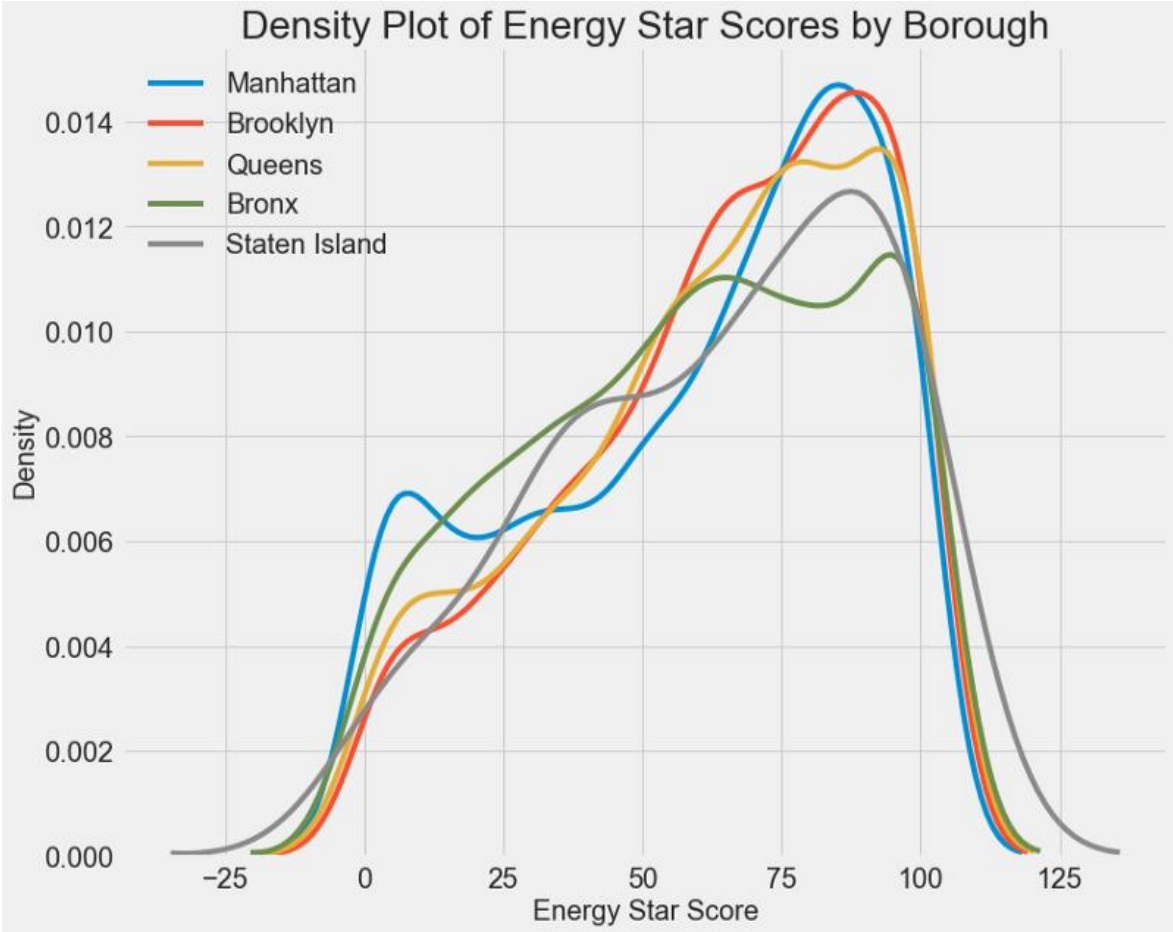
Анализ распределения прогнозированного значения **Energy Star Score**



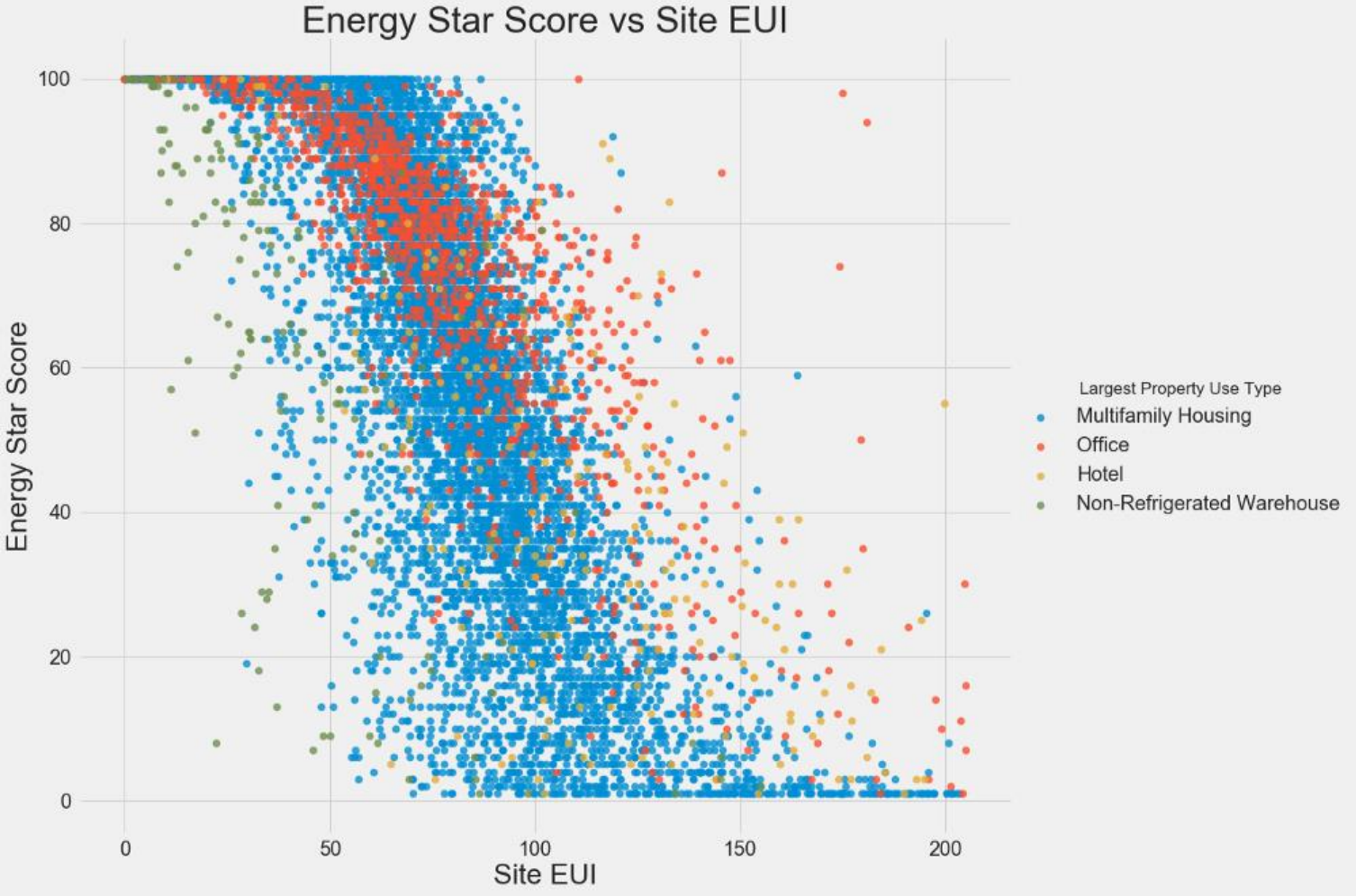
Поиск взаимосвязей

График плотности





Диаграммы рассеивания



EUI (Energy Use Intensity, интенсивность использования энергии) — это количество энергии, потреблённой зданием, делённое на квадратный фут площади. Эта удельная величина используется для оценки энергоэффективности, и чем она меньше, тем лучше.

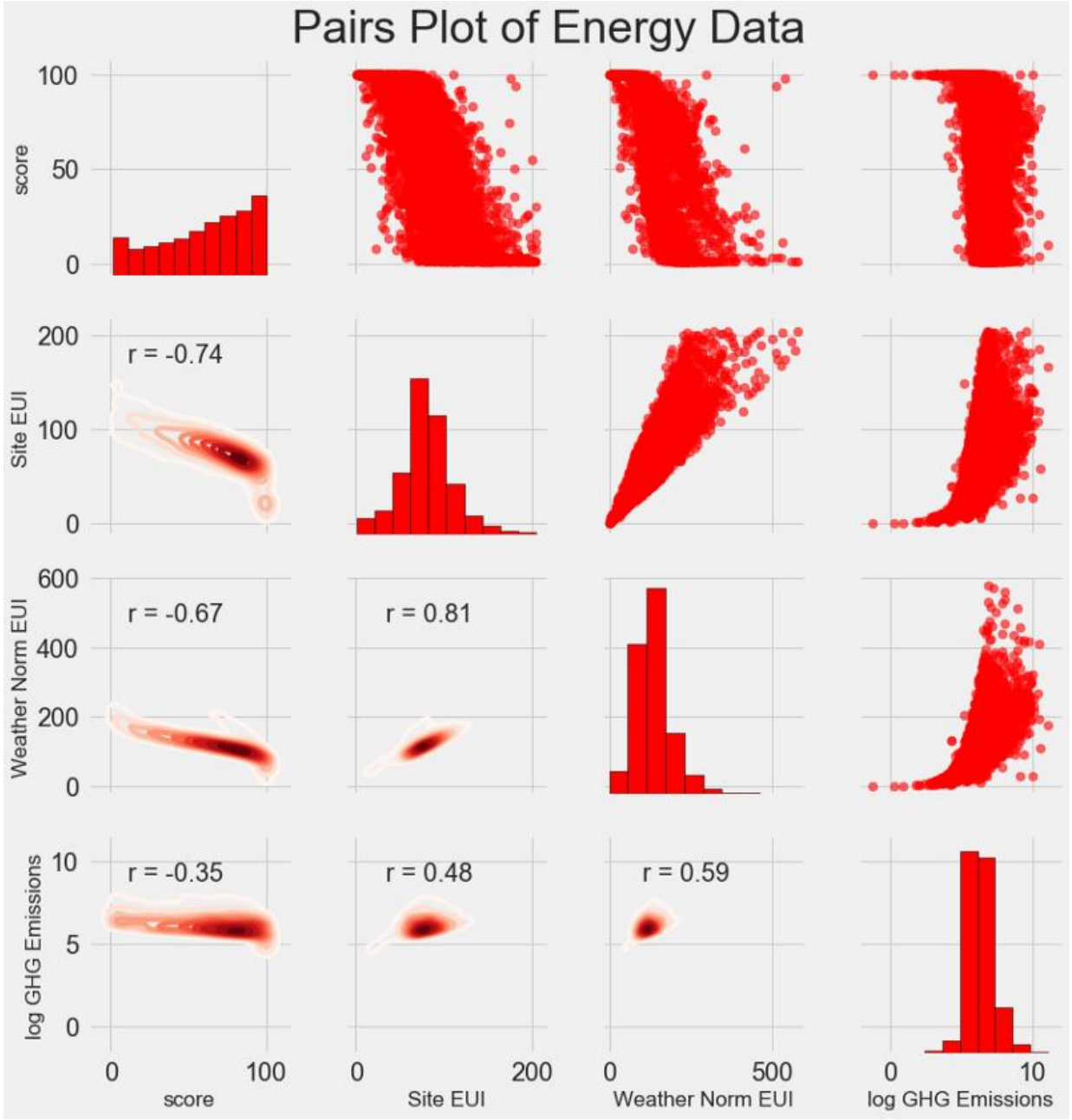
Парный график

Диаграмма рассеивания – верхний треугольник

Гистограмма – по диагонали

Диаграмма плотности – в нижнем треугольнике

Коэффициент корреляции Пирсона - мера интенсивности и направления линейной зависимости между двумя переменными. Значение +1 - идеальная линейная положительная зависимость, а -1 - идеальная линейная отрицательная зависимость.



3. Конструирование и выбор признаков

- Процесс извлечения или создания новых признаков из сырых данных.
- Выбор признаков можно рассматривать как удаление «лишнего», чтобы осталось только самое важное.

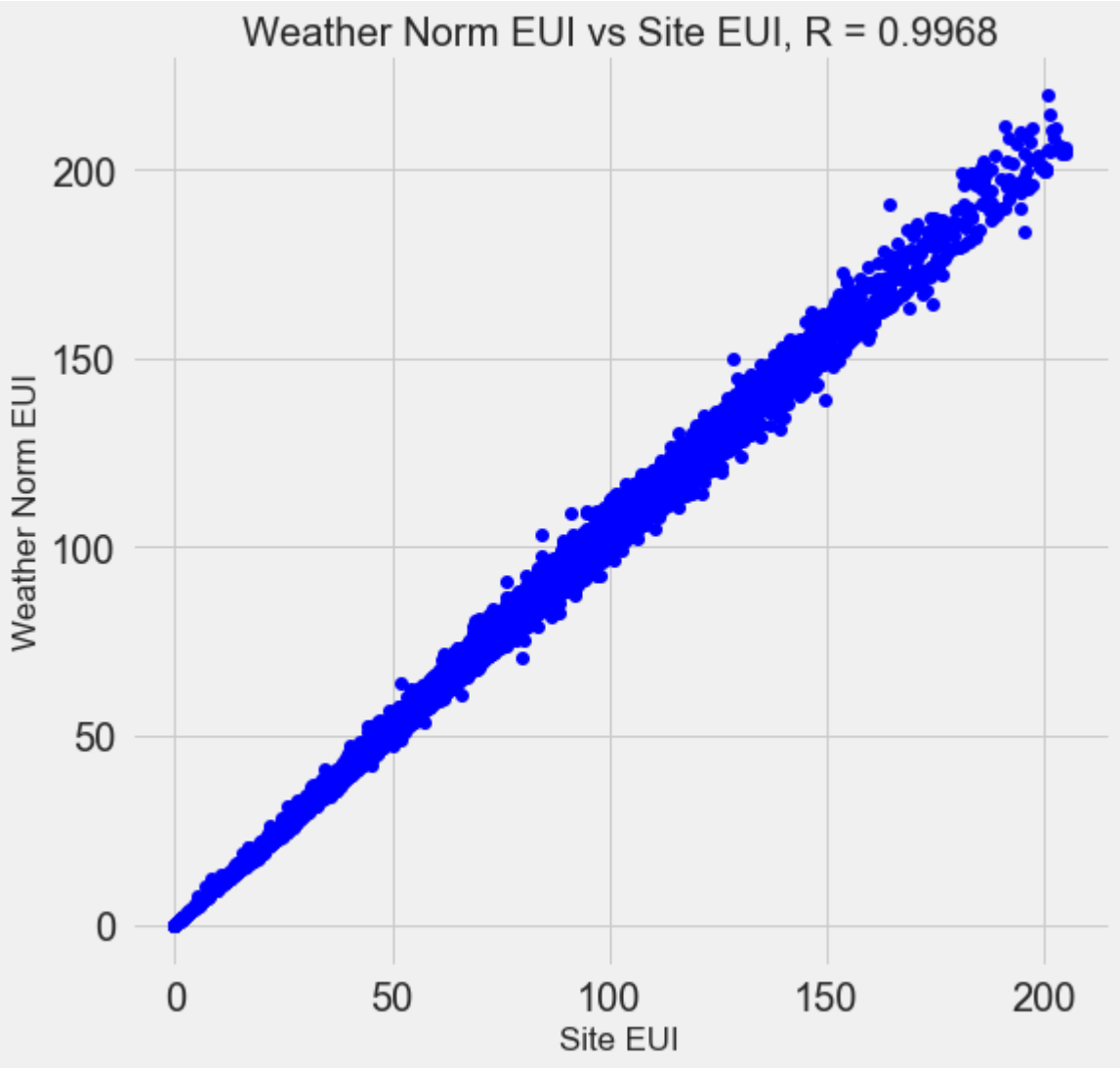
==========

- Применение к категориальным переменным (квартал и тип собственности) one-hot кодирования.
- Взятие натурального логарифма от всех числовых переменных.

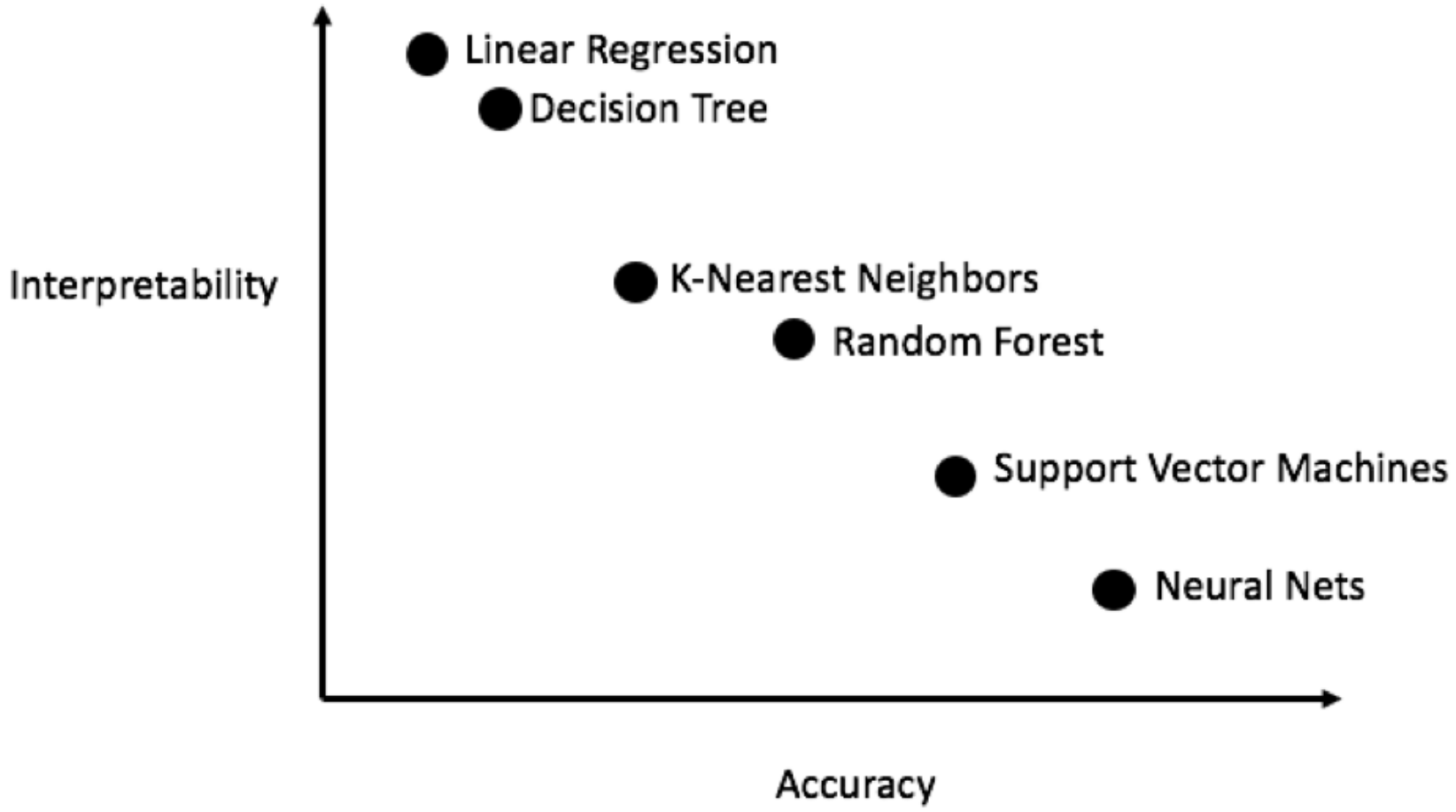
Выбор признаков

Коллинеарные признаки:
удаление одного из признаков

Коэффициент корреляции больше 0.6 – признаки коллинеарны



4. Оценка и выбор модели МО



http://blog.fastforwardlabs.com/2017/09/01/LIME-for-couples.html

```
from sklearn.ensemble import GradientBoostingRegressor

# Create the model
gradient_boosted = GradientBoostingRegressor()

# Fit the model on the training data
gradient_boosted.fit(X, y)

# Make predictions on the test data
predictions = gradient_boosted.predict(X_test)

# Evaluate the model
mae = np.mean(abs(predictions - y_test))

print('Gradient Boosted Performance on the test set: MAE = %0.4f' % mae)

Gradient Boosted Performance on the test set: MAE = 10.0132
```

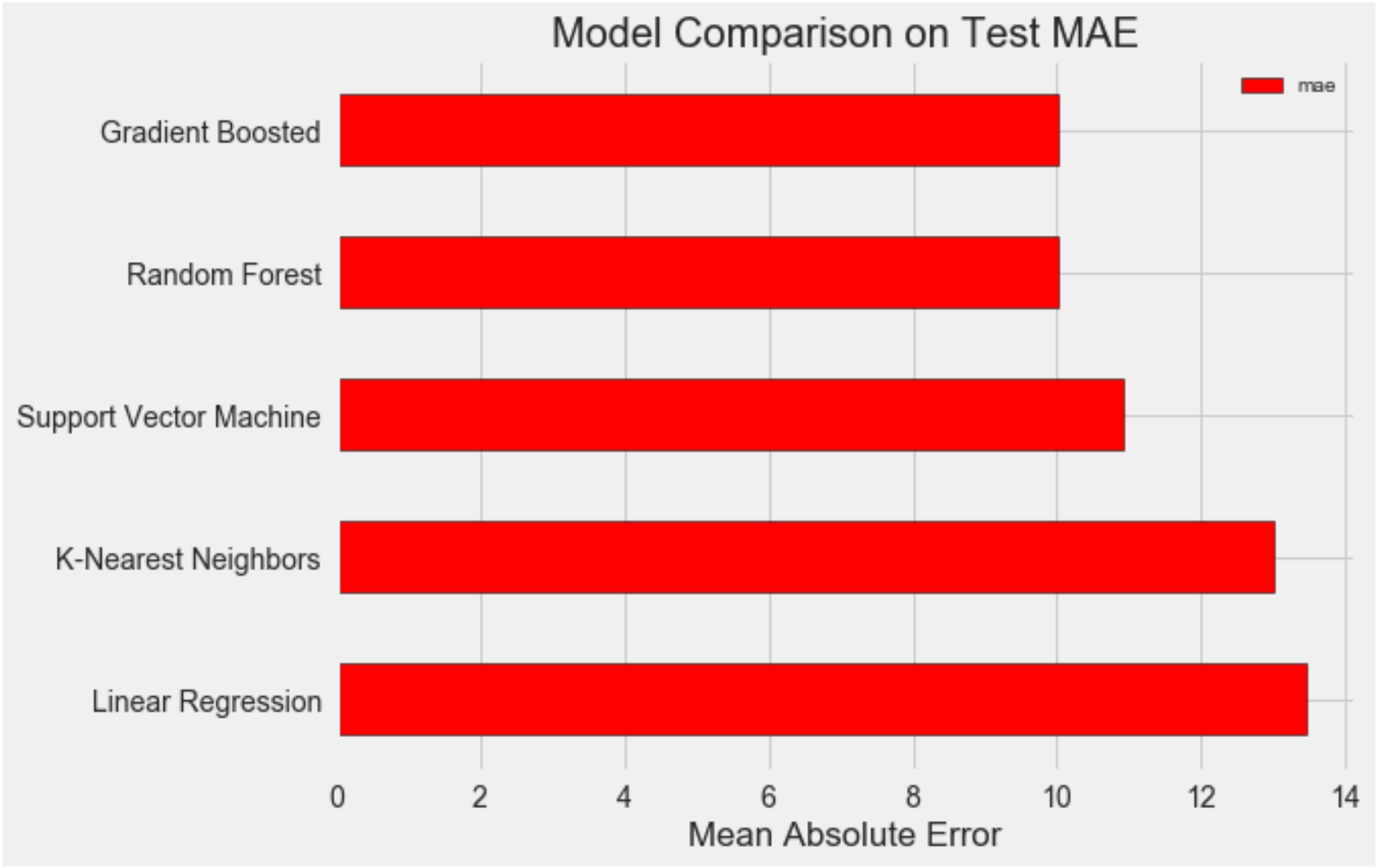
Линейная регрессия.

Метод k-ближайших соседей.

«Случайный лес».

Градиентный бустинг.

Метод опорных векторов.


Model Comparison on Test MAE
mae
Gradient Boosted
Random Forest
Support Vector Machine
K-Nearest Neighbors
Linear Regression
0
2
4
6
8
10
12
14
Mean Absolute Error

5. Гиперпараметрическая оптимизация модели

- **Гиперпараметры** модели можно считать настройками алгоритма, которые задаются до начала его обучения. Например, гиперпараметром является количество деревьев в «случайном лесе», или количество соседей в методе k-ближайших соседей.
- **Параметры** модели — то, что она узнаёт в ходе обучения, например, веса в линейной регрессии.
- Гиперпараметры влияют на недообучение и переобучение

Метод настройки гиперпараметров

Случайный поиск с перекрёстной проверкой

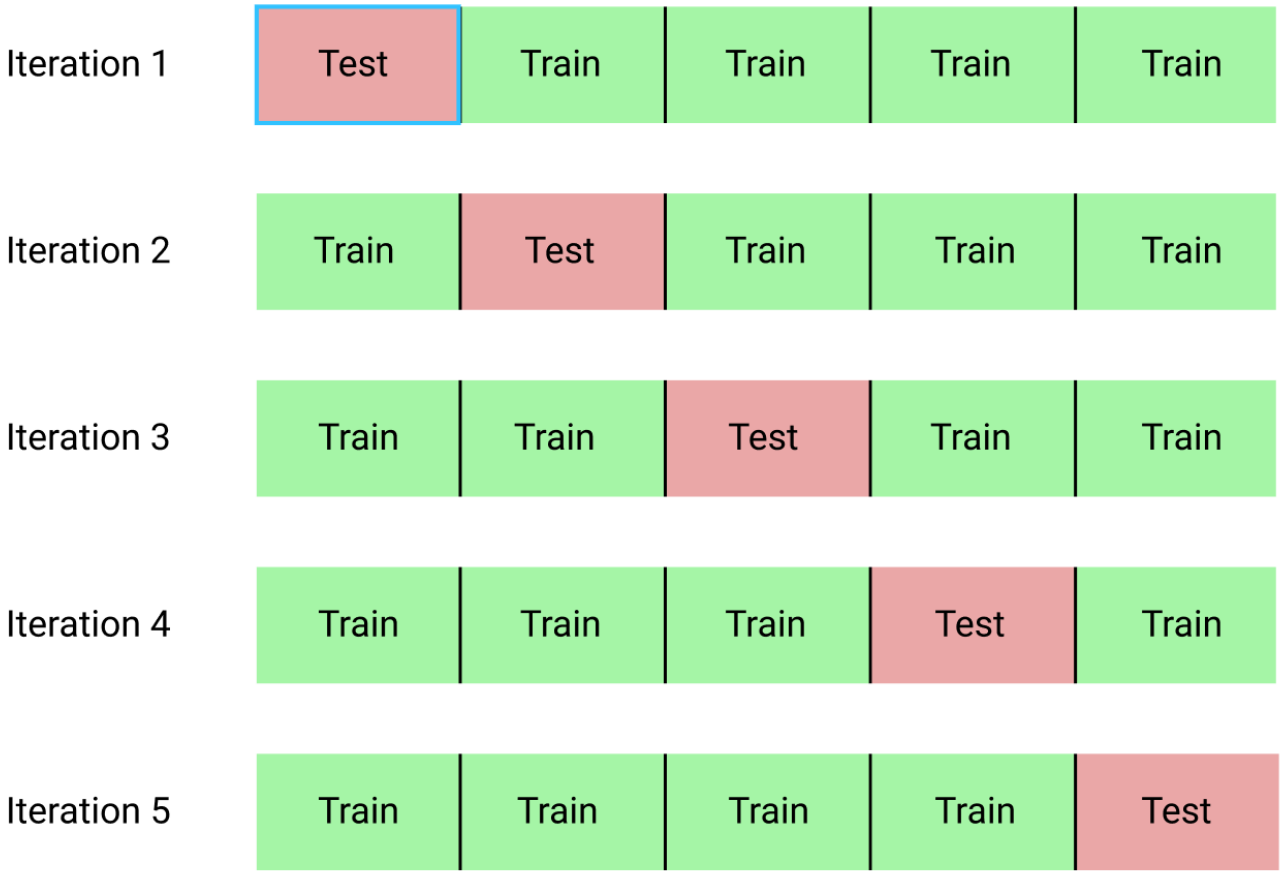
Случайный поиск

- Методика выбора гиперпараметров. Задается сетка, из неё случайно выбираются различные комбинации

Перекрестная проверка

- способ оценки выбранной комбинации гиперпараметров на основе k-блочной перекрестной проверки.

Иллюстрация k-блочной перекрестной проверки



RandomizedSearchCV
из Scikit-Learn

Метод МО: Регрессионная модель на основе градиентного бустинга

Последовательное обучение слабых методов (дерево решений), учитывая ошибки, сделанные предшественниками

Гиперпараметры

loss: минимизация функции потерь;

n_estimators: количество используемых слабых деревьев решений (decision trees);

max_depth: максимальная глубина каждого дерева решений;

min_samples_leaf: минимальное количество примеров, которые должны быть в «листовом» (leaf) узле дерева решений;

min_samples_split: минимальное количество примеров, которые нужны для разделения узла дерева решений;

max_features: максимальное количество признаков, которые используются для разделения узлов.

```
# Loss function to be optimized
loss = ['ls', 'lad', 'huber']

# Number of trees used in the boosting process
n_estimators = [100, 500, 900, 1100, 1500]

# Maximum depth of each tree
max_depth = [2, 3, 5, 10, 15]

# Minimum number of samples per leaf
min_samples_leaf = [1, 2, 4, 6, 8]

# Minimum number of samples to split a node
min_samples_split = [2, 4, 6, 10]

# Maximum number of features to consider for making sp
max_features = ['auto', 'sqrt', 'log2', None]
```

```
# Find the best combination of settings
random_cv.best_estimator_
GradientBoostingRegressor(loss='lad', max_depth=5,
    max_features=None,
    min_samples_leaf=6,
    min_samples_split=6,
    n_estimators=500)
```

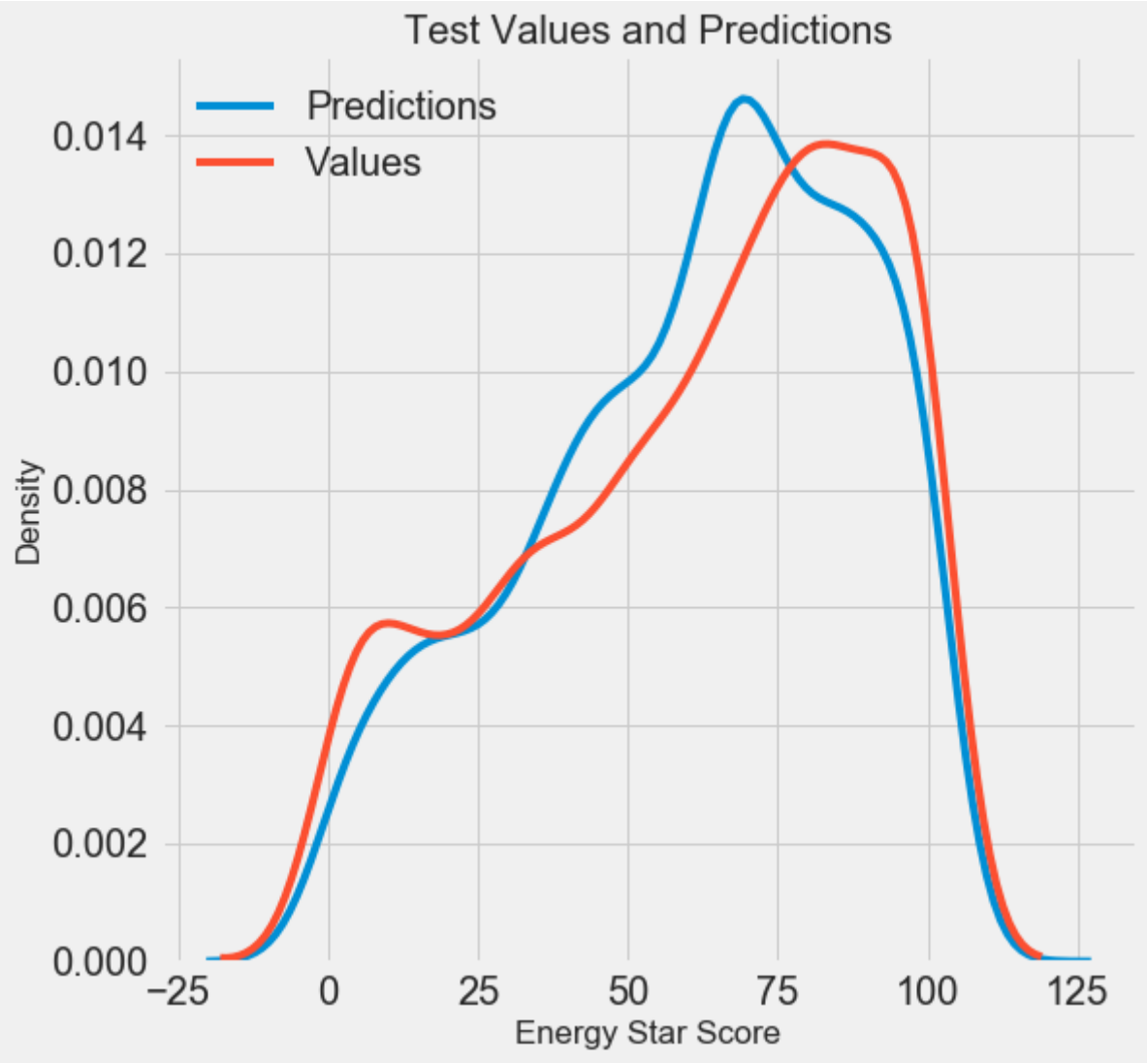
```
# Define the grid of hyperparameters to search
hyperparameter_grid = {'loss': loss,
    'n_estimators': n_estimators,
    'max_depth': max_depth,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'max_features': max_features}

# Create the model to use for hyperparameter tuning
model = GradientBoostingRegressor(random_state = 42)

# Set up the random search with 4-fold cross validation
random_cv = RandomizedSearchCV(estimator=model,
    param_distributions=hyperparameter_grid,
    cv=4, n_iter=25,
    scoring = 'neg_mean_absolute_error',
    n_jobs = -1, verbose = 1,
    return_train_score = True,
    random_state=42)
```

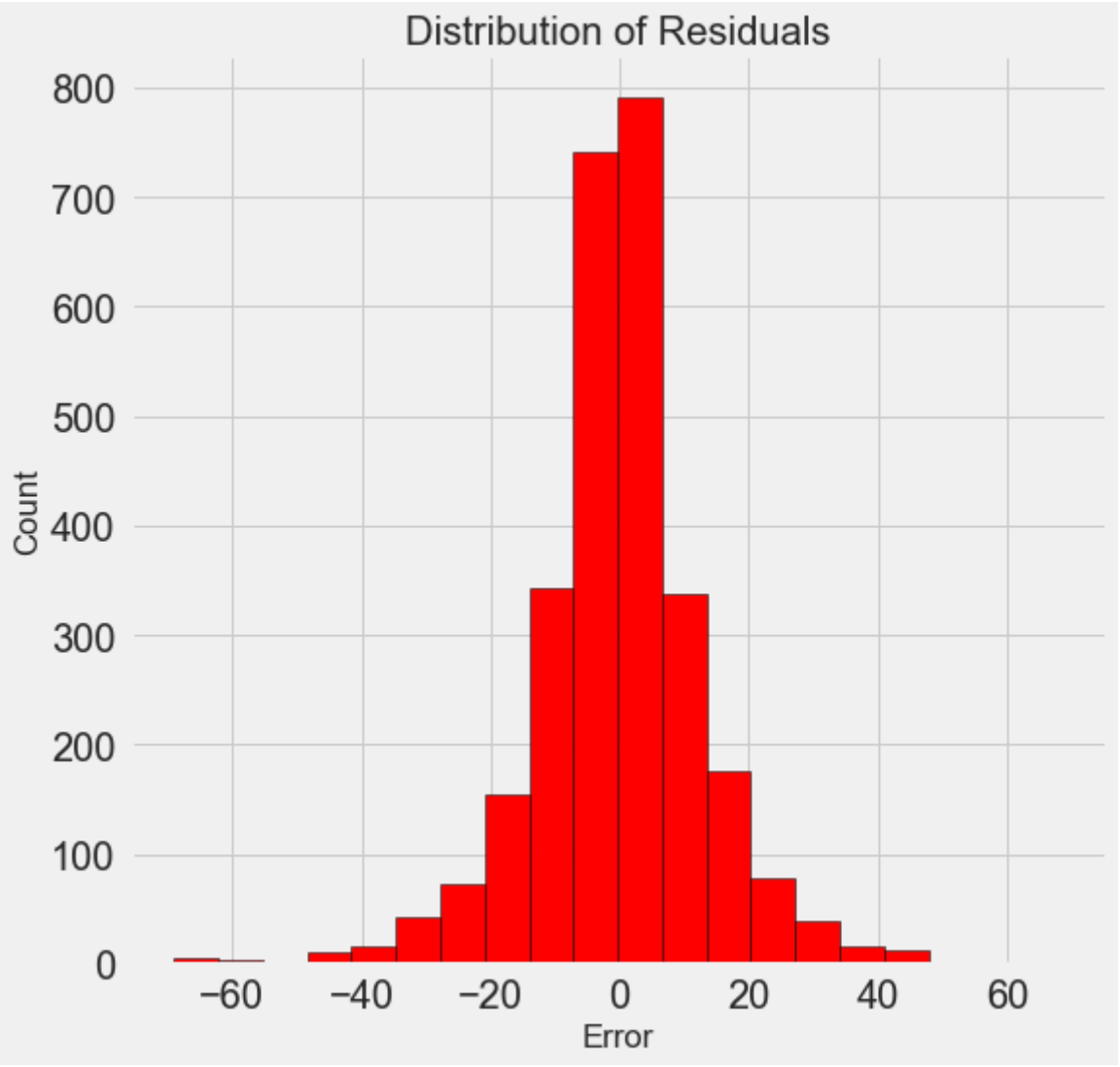
5. Оценка с помощью тестовых данных

```
# Make predictions on the test set using default and final model
default_pred = default_model.predict(X_test)
final_pred = final_model.predict(X_test)
Default model performance on the test set: MAE = 10.0118.
Final model performance on the test set: MAE = 9.0446.
```

```
%%timeit -n 1 -r 5
default_model.fit(X, y)
1.09 s ± 153 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

```
%%timeit -n 1 -r 5
final_model.fit(X, y)
12.1 s ± 1.33 s per loop (mean ± std. dev. of 5 runs, 1 loop each)
```


Test Values and Predictions
Predictions
Values
Density
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
−25
0
25
50
75
100
125
Energy Star Score


Distribution of Residuals
Count
800
700
600
500
400
300
200
100
0
−60
−40
−20
0
20
40
60
Error

6. Интерпретация модели

1. Оценить важности признаков.
2. Визуализировать одно из деревьев решений.
3. Применить метод LIME — Local Interpretable Model-Agnostic Explainations, локальные интерпретируемые моделезависимые объяснения.

Важности признаков

```
import pandas as pd

# model is the trained model
importances = model.feature_importances_

# train_features is the dataframe of training features
feature_list = list(train_features.columns)

# Extract the feature importances into a dataframe
feature_results = pd.DataFrame({'feature':
feature_list,'importance': importances})

# Show the top 10 most important
feature_results =
feature_results.sort_values('importance',ascending =
False).reset_index(drop=True)

feature_results.head(10)
```

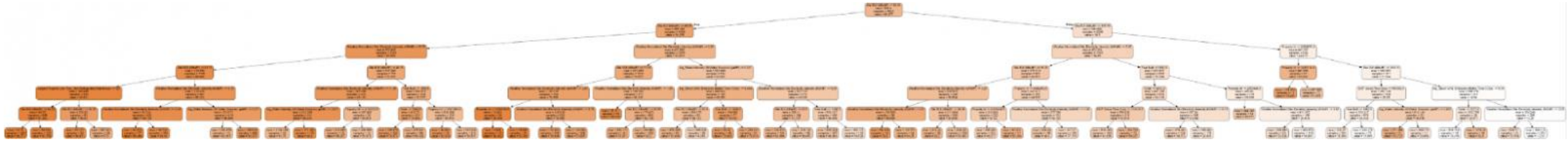
	feature	importance
0	Site EUI (kBtu/ft²)	0.403532
1	Weather Normalized Site Electricity Intensity ...	0.263059
2	Water Intensity (All Water Sources) (gal/ft²)	0.071286
3	Property Id	0.035165
4	Largest Property Use Type_Non-Refrigerated War...	0.031924
5	DOF Gross Floor Area	0.027900
6	log_Water Intensity (All Water Sources) (gal/ft²)	0.026058
7	Order	0.024592
8	log_Direct GHG Emissions (Metric Tons CO2e)	0.023655
	Year Built	0.022100

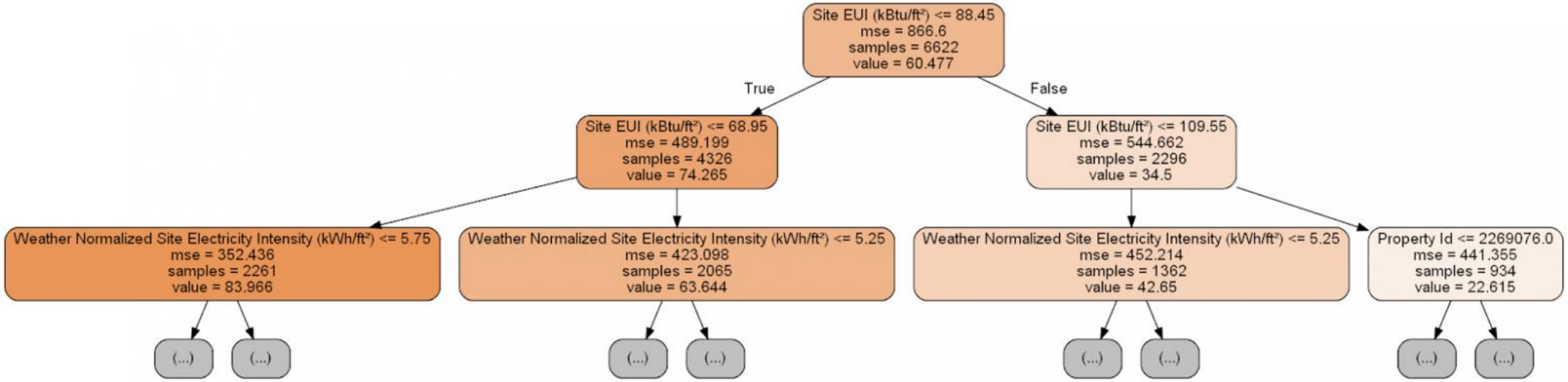
Визуализация дерева решений

```
from sklearn import tree

# Extract a single tree (number 105)
single_tree = model.estimators_[105][0]

# Save the tree to a dot file
tree.export_graphviz(single_tree, out_file =
'images/tree.dot', feature_names = feature_list)
```

С помощью визуализатора Graphviz преобразуем dot-файл в png

```
dot -Tpng images/tree.dot -o images/tree.png
```




Site EUI (kBtu/ft²) <= 88.45
mse = 866.6
samples = 6622
value = 60.477
True
False
Site EUI (kBtu/ft²) <= 68.95
mse = 489.199
samples = 4326
value = 74.265
Site EUI (kBtu/ft²) <= 109.55
mse = 544.662
samples = 2296
value = 34.5
Weather Normalized Site Electricity Intensity (kWh/ft²) <= 5.75
mse = 352.436
samples = 2261
value = 83.966
Weather Normalized Site Electricity Intensity (kWh/ft²) <= 5.25
mse = 423.098
samples = 2065
value = 63.644
Weather Normalized Site Electricity Intensity (kWh/ft²) <= 5.25
mse = 452.214
samples = 1362
value = 42.65
Property Id <= 2269076.0
mse = 441.355
samples = 934
value = 22.615
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

Задачи сложные для машинного обучения

- Перевод с одного языка на другой
- Автоматическое резюмирование
- Морфологическая сегментация
- Генерация и понимание естественного языка
- Разметка частей речи
- Анализ тональности и распознавание речи

Спасибо за внимание