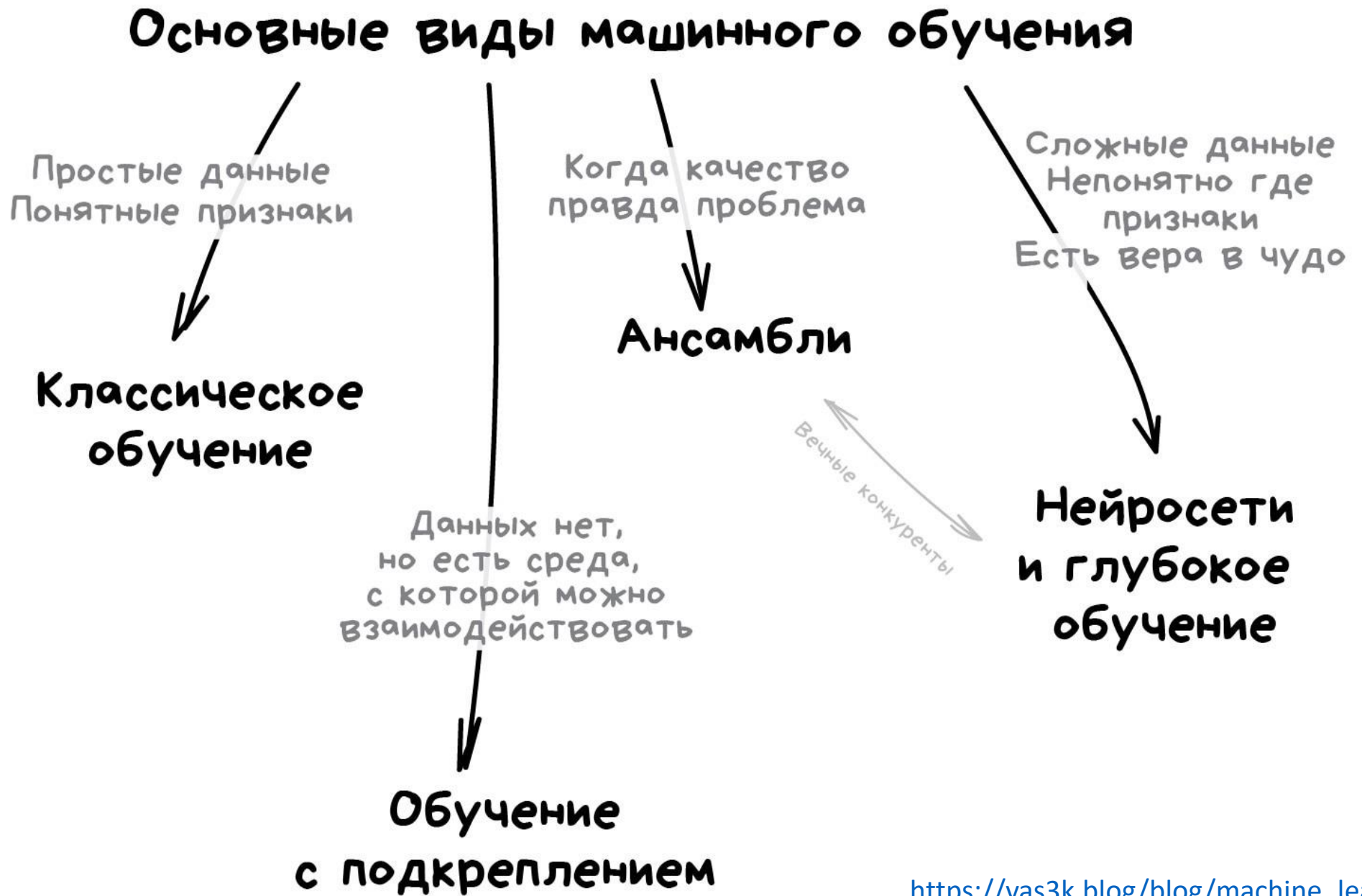
Замечания по итогам второй лабы

- **Важно понимать тему датасета, хотя бы что означают названия столбцов (параметры).** Во-первых, не понимая **о чем речь**, вам сложно будет построить логические связи и объяснить зависимости. Во-вторых, работать с обезличенными цифрами – не интересно, а находить закономерности и получать логические объяснения процессам может быть интересно пытливому уму.
- **Удаляя данные с большим количеством пропусков – рассуждайте представляет ли ценность удаляемый столбец.** Понятно, что мы учимся удалять, но удаляя все подряд, можно лишиться ценной информации. Чаще лучше заменить на подходящие значения.
- **Заменяя пропуски на какие либо значения – рассуждайте. Иногда можно найти закономерности в данных, которые помогут нам заполнить пропуски.** Например, в датасете по книгам на Амазоне много пропусков в столбце вес, но есть данные о размере книг, значит можно вычислить примерный вес книги, или пропуски в зарплате можно заполнить средней зарплатой не по всей таблице, а средним по данной должности, что будет больше соответствовать действительности.

Лекция 2

Тема: Машинное обучение



https://vas3k.blog/blog/machine_learning/

Классическое Обучение

Данные заранее категоризированы или численные

Данные никак не размечены

С учителем

Предсказать категорию

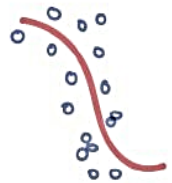
Предсказать значение

Классификация

«Разложи носки по цвету»

Регрессия

«Разложи галстуки по длине»



Без учителя

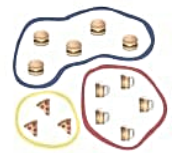
Разделить по схожести

Найти зависимости

Выявить последовательности

Кластеризация

«Разложи похожие вещи по кучкам»



Ассоциация

«Найди какие вещи я часто ношу вместе»

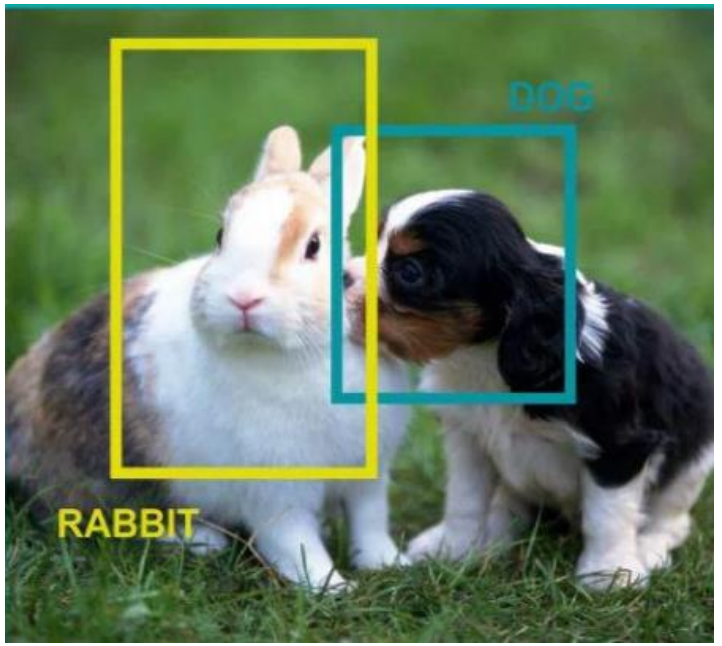
Уменьшение Размерности (обобщение)

«Собери из вещей лучшие наряды»

Классическое обучение делят на две категории — с учителем и без. Часто можно встретить их английские наименования — Supervised и Unsupervised Learning.

Обучение с учителем подразумевает, что данные в обучающей выборке должны быть размечены (в случае распознавания изображений, символов) или каждому набору признаков должны соответствовать метки (классы). **В этом случае принято говорить о задачах классификации и регрессии**.

PassengerId	Survived	Pclass	Name	Sex	Age
1	0	3	Braund, Mr. Owen Harris	male	22.0
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
3	1	3	Heikkinen, Miss. Laina	female	26.0
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
5	0	3	Allen, Mr. William Henry	male	35.0



В **обучении без учителя** разделение объектов обучающей выборки на классы не задаётся, и требуется классифицировать объекты только на основе их сходства друг с другом. **В этом случае принято говорить о задачах кластеризации**.

Обучение с учителем

В зависимости от того, какие значения может принимать ответ y, различают разные классы задач обучения с учителем.

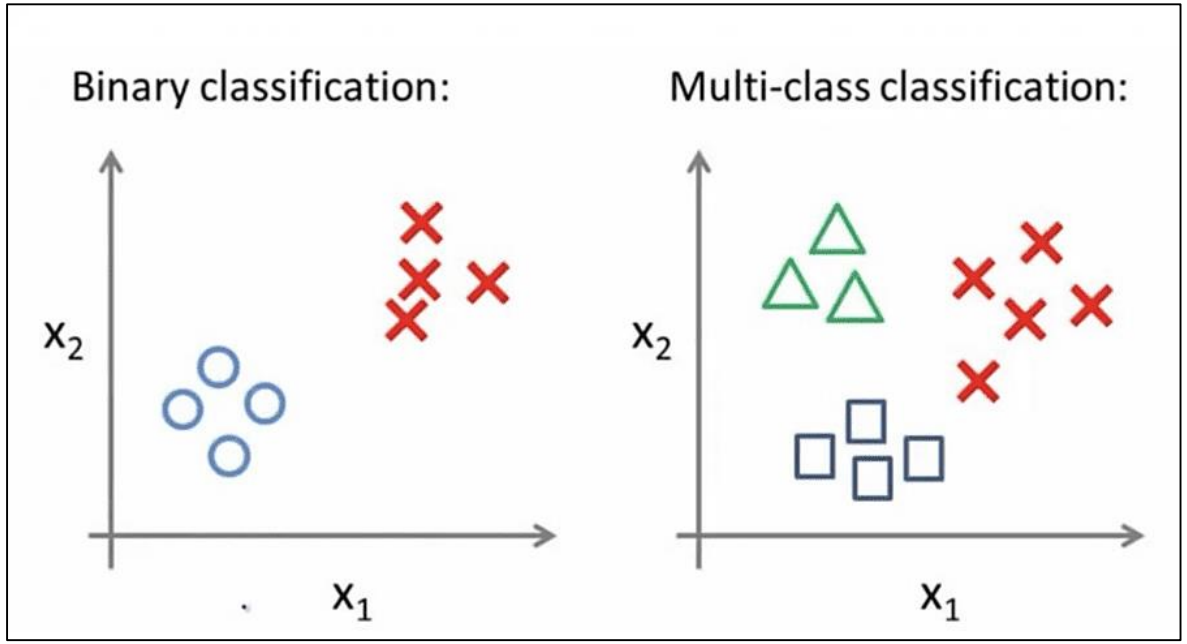
Если y - непрерывная величина , то говорят о задаче **регрессии**.

Если y конечно, например, y={1,2…n}, то говорят о задаче **классификации**.

Классификация — один из разделов машинного обучения, посвященный решению следующей задачи.

Имеется множество объектов (ситуаций), разделённых некоторым образом на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется обучающей выборкой. Классовая принадлежность остальных объектов не известна. Требуется построить алгоритм, способный классифицировать произвольный объект из множества.

Классификация иногда разделяется на **бинарную** классификацию (binary classification), которая является частным случаем разделения на два класса, и **мультиклассовую** классификацию (multiclass classification), когда в классификации участвует более двух классов.



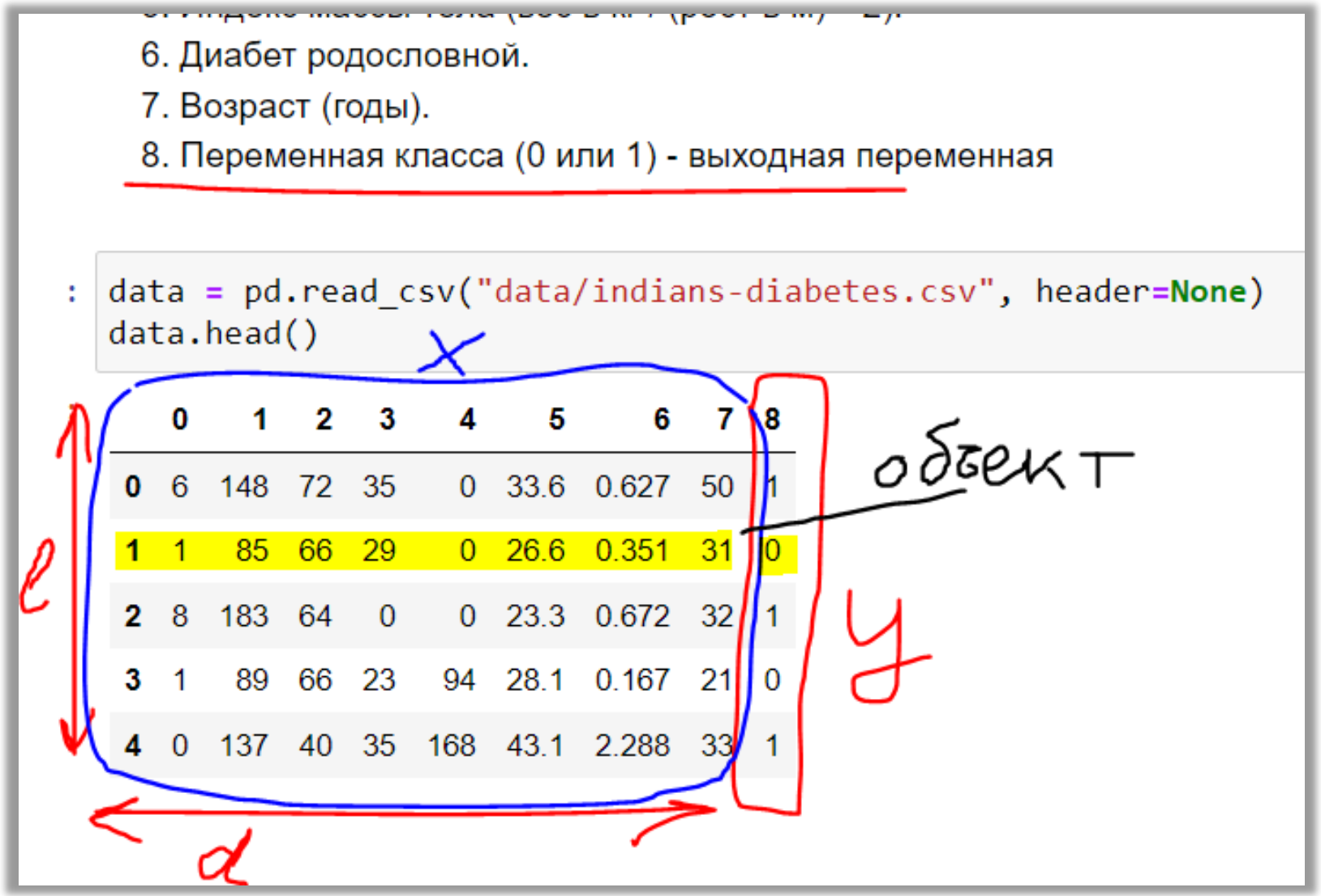
Пример задачи классификации

Дано: численные или категоризированные данные.

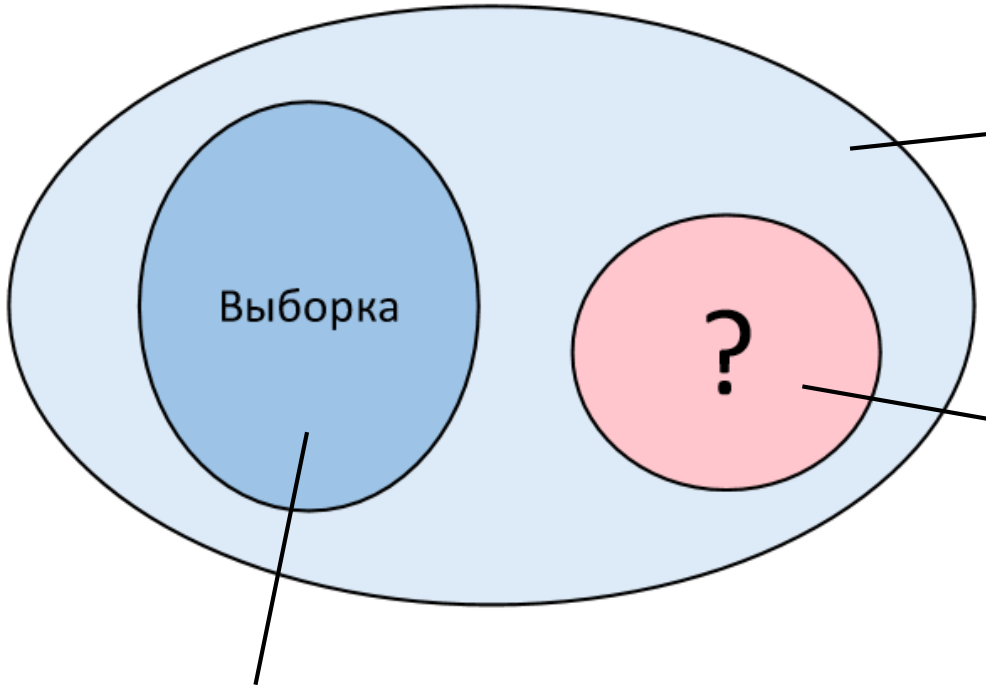
x- матрица объектов-признаков,

y- вектор меток (дискретные значения). В данном примере 1 – пациент болен, 0 – пациент здоров

Нужно: получить модель (алгоритм), ставящую в соответствие каждому **новому** объекту одну метку (класс), т.е. по данным анализов **нового пациента** определить болен или здоров



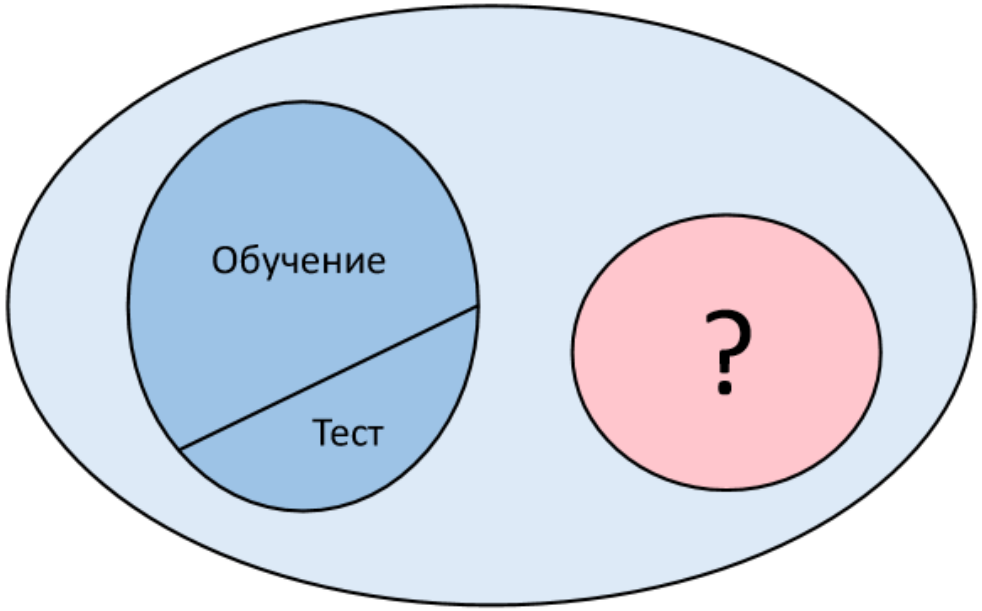
	0	1	2	3	4	5	6	7	8
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1



Генеральная совокупность

Новые данные, для которых нужно предсказать ответ

Выборка или размеченные данные



На практике все имеющиеся данные разбивают на обучающую и тестовую выборки (70% и 30%).

Обучение производится с использованием обучающей выборки, а оценка качества предсказания на основе данных тестовой выборки.

Какие могут быть ошибки при таком разбиении (например, выборка была отсортирована по какому либо признаку)

В библиотеке **scikit-learn** есть функция ***train_test_split***, которая перемешивает набор данных и разбивает его на две части.

Параметр **random_state** используется в качестве начального значения для генератора чисел random. Это гарантирует, что наборы данных train и test не будут изменяться при каждом новом выполнении кода.

```
X_train, X_valid, y_train, y_valid = train_test_split(    # по умолчанию 75% и 25%
    X, y, test_size=0.3, random_state=17)
```

```
X_train.shape, X_valid.shape
```

```
((7000, 5), (3000, 5))
```

```
y_train.shape, y_valid.shape
```

```
((7000,), (3000,))
```

Деревья решений (Decision Tree, DT)

Деревья решений являются эффективным инструментом интеллектуального анализа данных и предсказательной аналитики, который позволяет решать задачи *классификации* и *регрессии*.

Они представляют собой иерархические древовидные структуры, состоящие из решающих правил вида «Если ..., то ...». Правила автоматически генерируются в процессе обучения на обучающем множестве.

В обучающем множестве для примеров должно быть задано целевое значение, т.к. деревья решений являются моделями, строящимися на основе обучения с учителем.

Метод деревьев решений для задачи классификации состоит в том, чтобы осуществлять процесс деления исходных данные на группы, пока не будут получены однородные (или почти однородные) их множества.

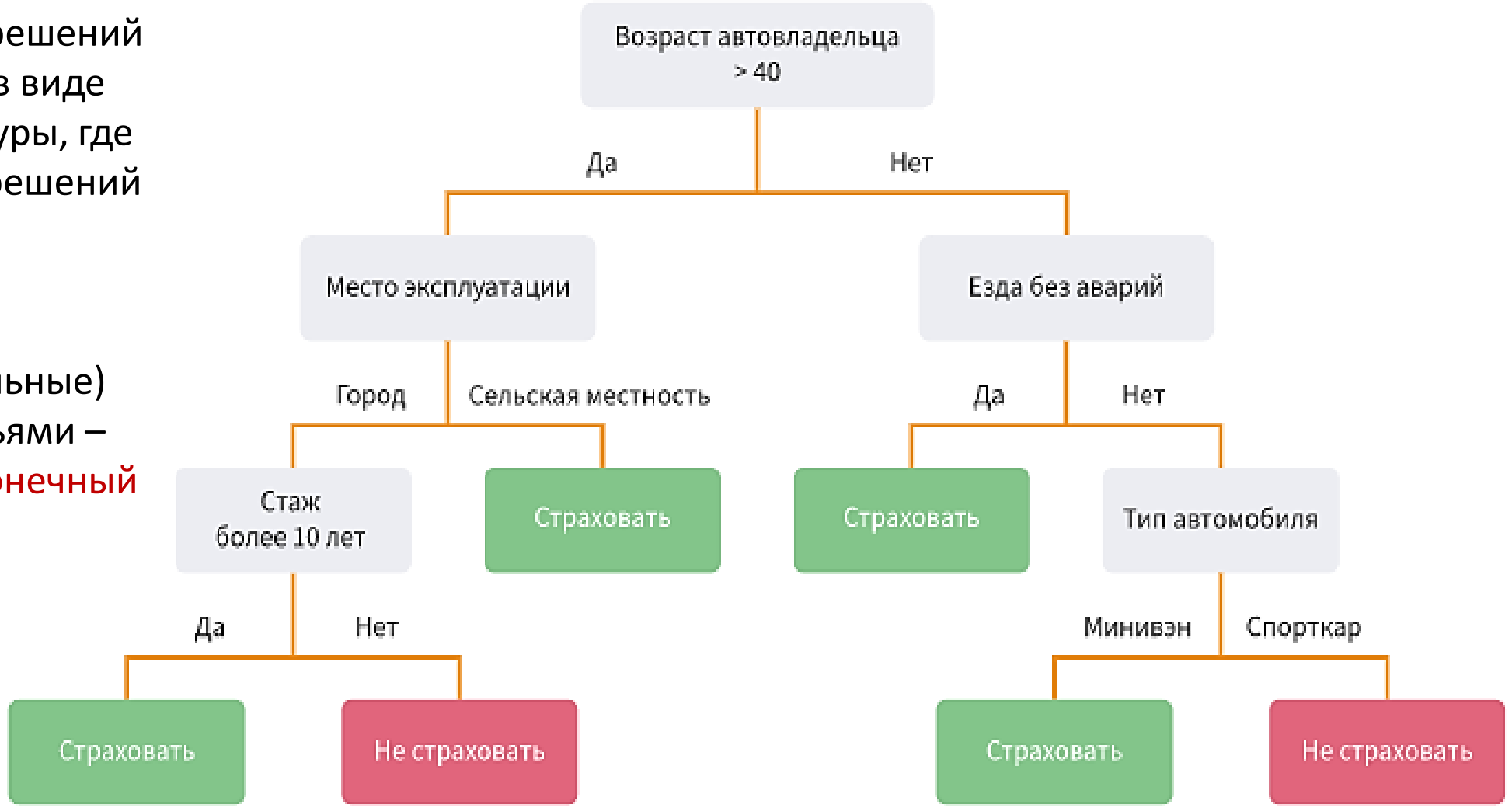
Совокупность правил, которые дают такое разбиение, позволят затем делать прогноз (т.е. определять наиболее вероятный номер класса) для новых данных.

Среди задач, успешно решаемых с помощью этого метода, можно назвать, например,
- скоринговые модели кредитования (англ.: credit scoring models);
- диагностика (медицинская или техническая), где по набору значений факторов (симптомов, результатов анализов) нужно поставить диагноз.

Структура дерева решений

Графически дерево решений можно представить в виде древовидной структуры, где моменты принятия решений соответствуют так называемым узла.

Конечные (терминальные) узлы называют листьями – каждый лист – это конечный результат последовательного принятия решений.



https://loginom.ru/blog/decision-tree-p1

Задача: разложить сценарии фильмов по трём ящикам:

- Популярные;
- Не популярные у зрителей, но получившие высокую оценку критиков;
- Не имеющие успеха.

Есть тестовая выборка из 30 фильмов

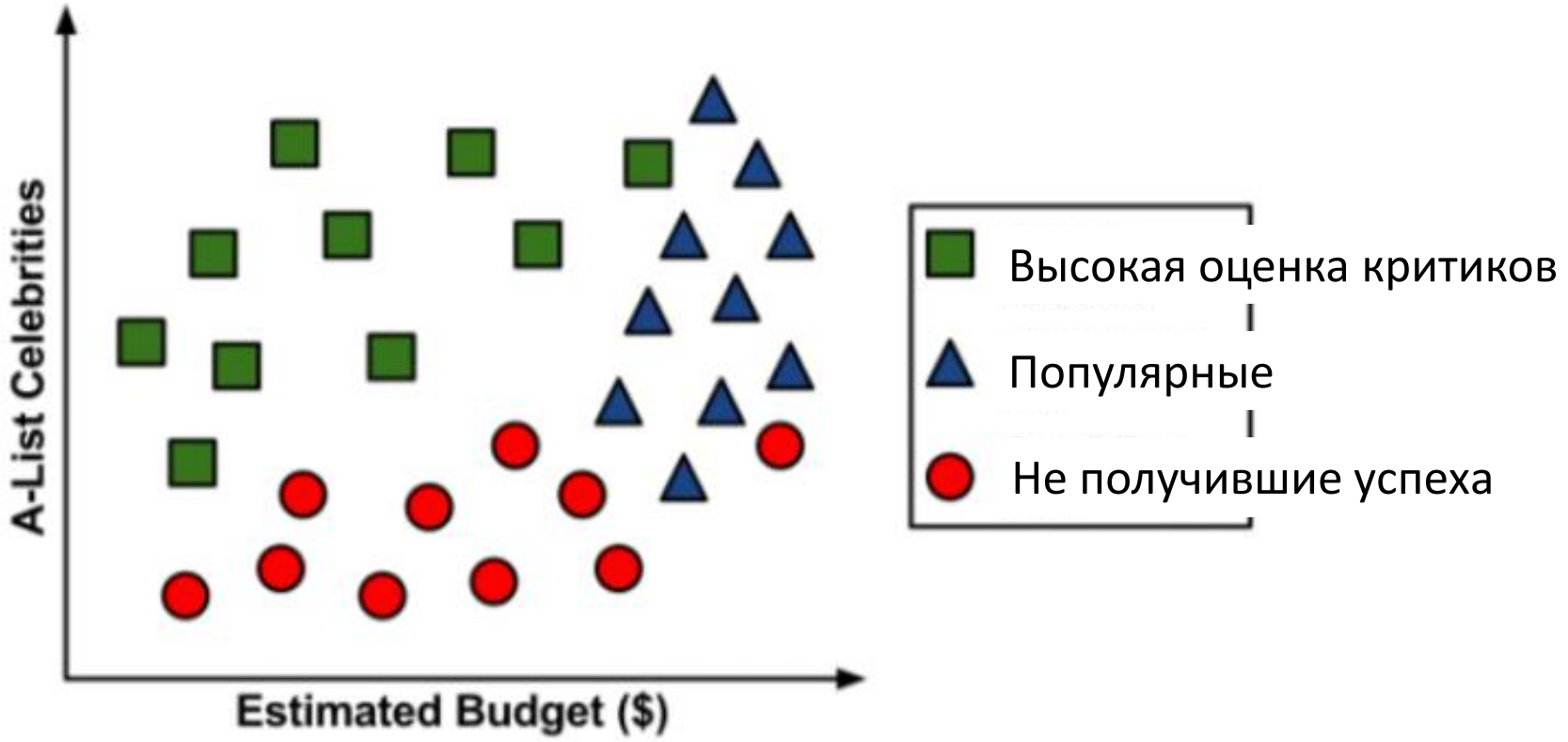


Рис.1 Зависимость количества звёзд, снимавшихся в фильме, от его бюджета

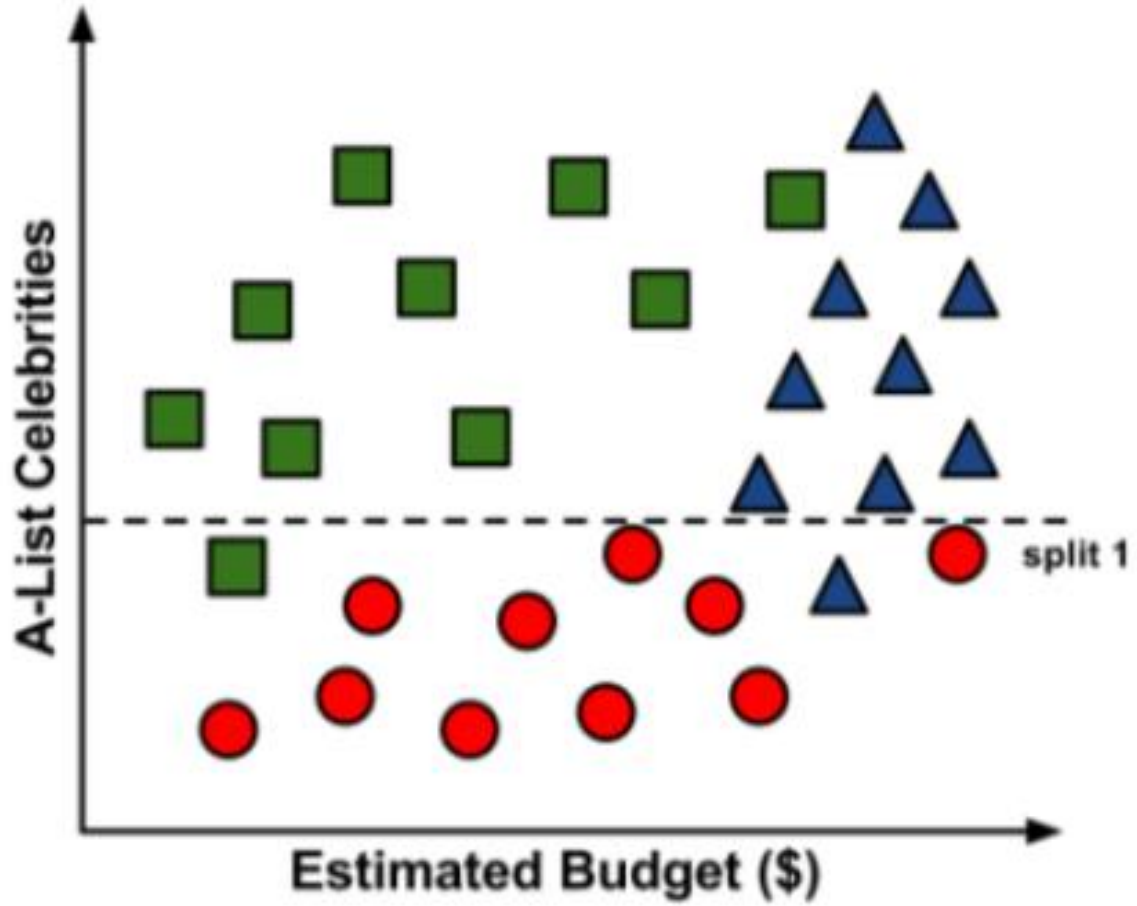


Рис.2 Разбиение множества киносценариев по признаку количества занятых звёзд

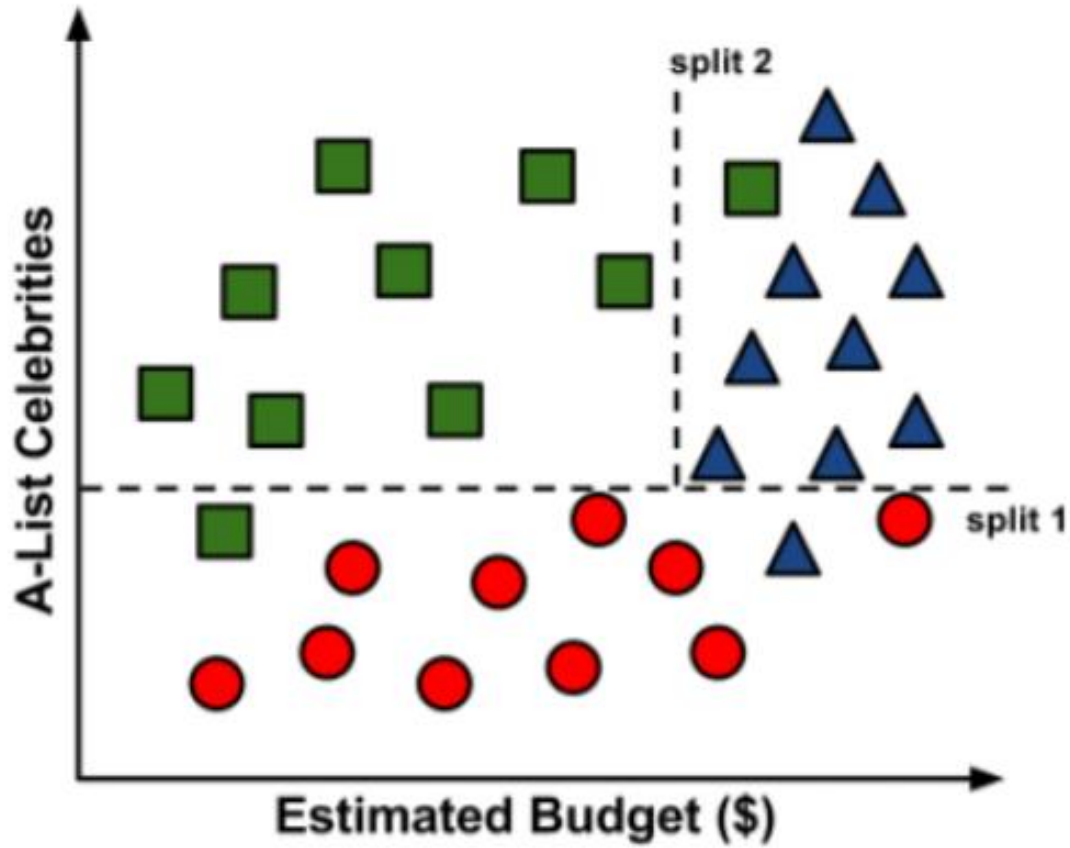
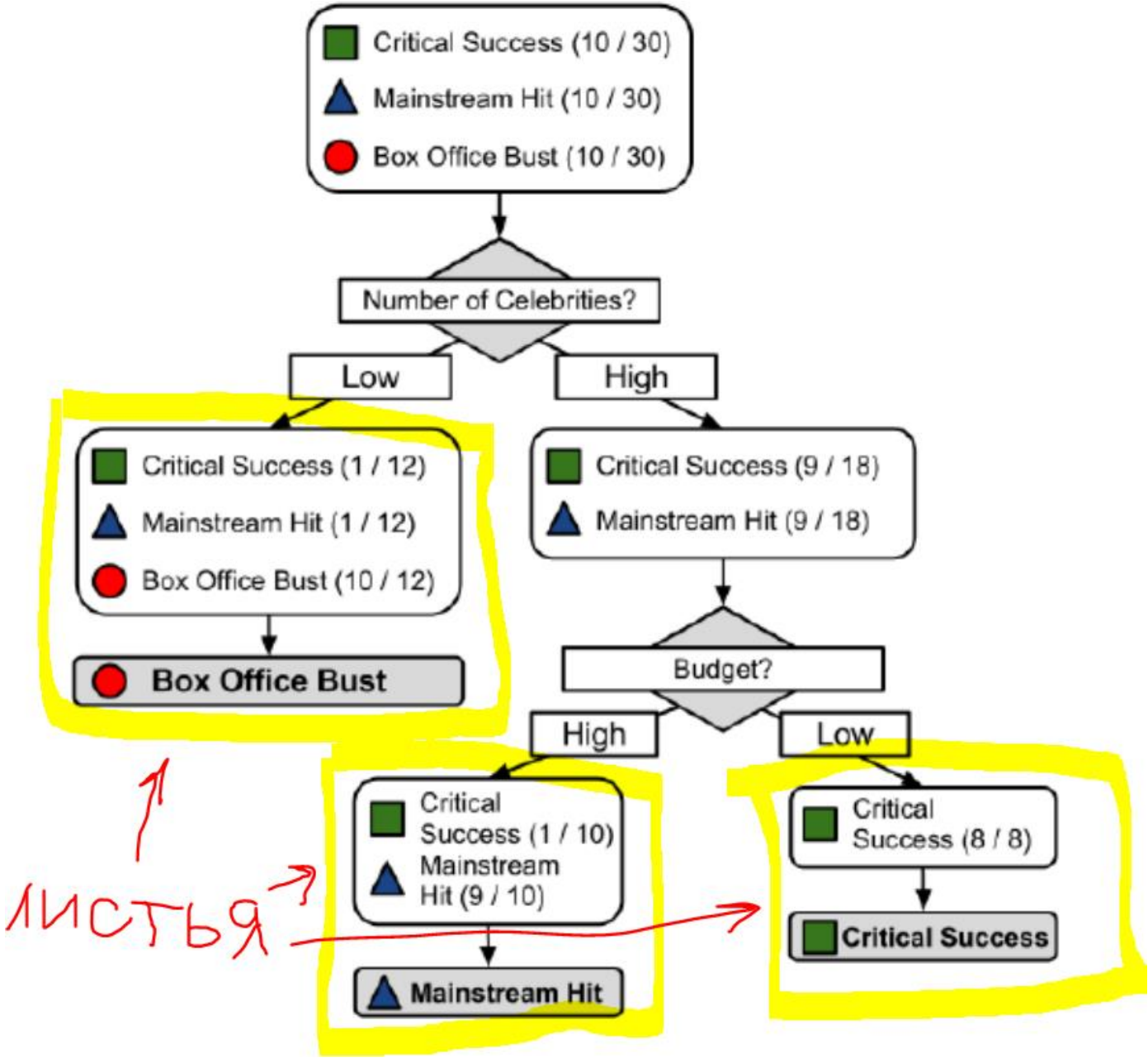


Рис.3 Разбиение группы киносценариев с большим количеством занятых звёзд по признаку размера бюджета

В нашем примере мы условно делим число задействованных в фильме звёзд по принципу «много» – «мало», и аналогично различаем малобюджетные и высокобюджетные. Соответствующее дерево решений показано на рис.

Ограничим ветвление дерева – например, когда каждая группа хотя бы на **80% будет состоять из элементов одного и того же класса**. (В нашем примере в первой группе все зелёные квадраты; во второй группе из 10 элементов 1 квадрат и 9 треугольников (90%); в третьей группе 12 элементов, из них 10 одинаковых – красных кружков, – т.е. показатель однородности = 10/12 = 83,33… %.



Процесс построения дерева решений

Процесс построения деревьев решений заключается в последовательном, рекурсивном разбиении обучающего множества на подмножества с применением решающих правил в узлах. Эта стратегия рекурсивного разбиения также называется «Разделяй и властвуй».

Процесс разбиения продолжается до тех пор, пока все узлы в конце всех ветвей не будут объявлены листьями. Объявление узла листом может произойти естественным образом (когда он будет содержать единственный объект, или объекты только одного класса), или по достижении некоторого условия остановки, задаваемого пользователем (например, достижение заданного показателя однородности в листьях или максимальная глубина дерева).

Основные этапы построения

- ✓ Выбор признака, по которому будет производиться разбиение в данном узле.
- ✓ Выбор критерия остановки обучения.
- ✓ Оценка точности построенного дерева.

Выбор признака

Какой признак выбрать первым?

Здесь можно вспомнить игру "20 вопросов", которая часто упоминается во введении в деревья решений.

> *Один человек загадывает знаменитость, а второй пытается отгадать, задавая только вопросы, на которые можно ответить "Да" или "Нет". Какой вопрос отгадывающий задаст первым делом? Конечно, такой, который сильнее всего уменьшит количество оставшихся вариантов. К примеру, вопрос "Это женщина?" отсечет уже около половины знаменитостей. То есть, признак "пол" намного лучше разделяет выборку людей, чем признак "это Джонни Депп", "национальность-испанец" или "любит футбол".*

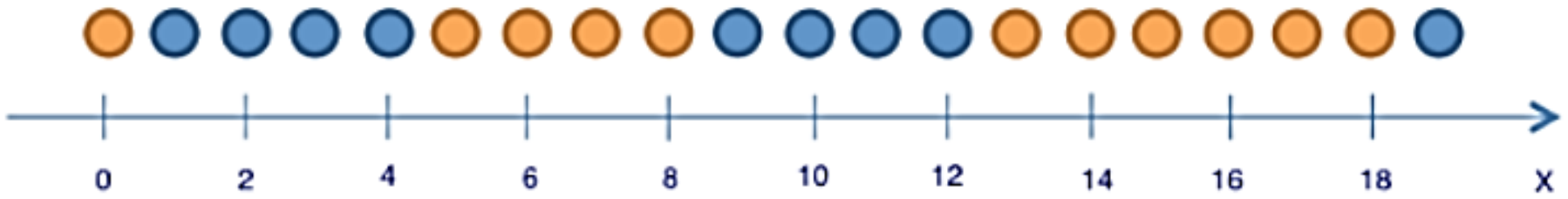
Энтропия определяется для системы с N возможными состояниями следующим образом:

$$S = -\sum_{i=1}^{N} p_i \log_2 p_i,$$

где p_i – вероятности нахождения системы в i-ом состоянии.

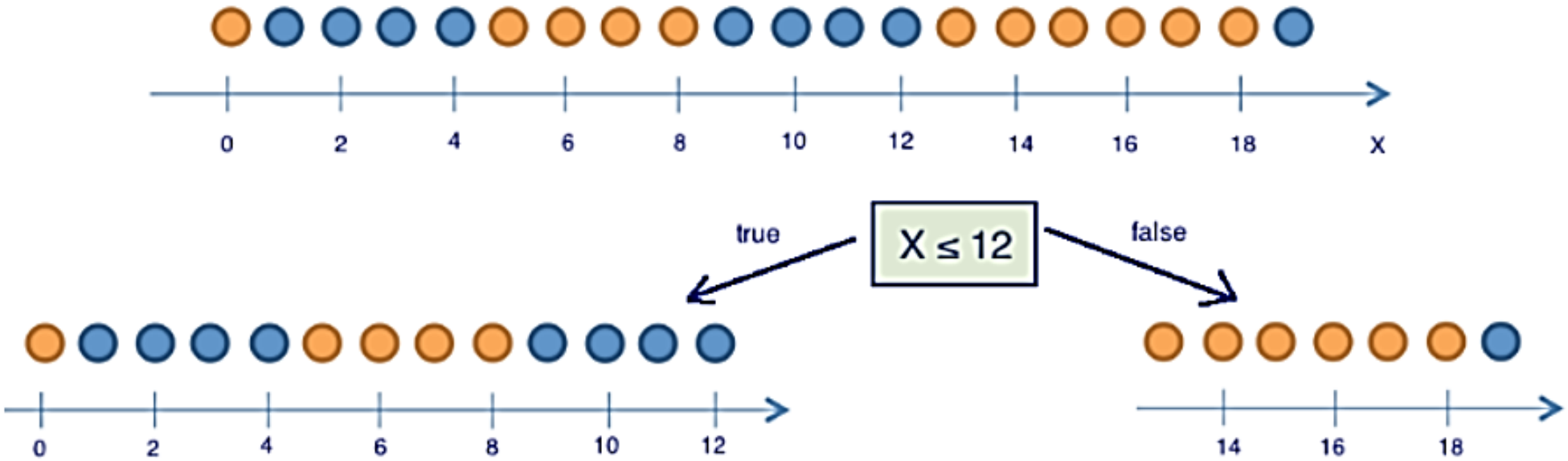
Энтропия соответствует степени хаоса в системе. Чем выше энтропия, тем менее упорядочена система и наоборот. Это понятие поможет формализовать "эффективное разделение выборки".

Для иллюстрации того, как энтропия поможет определить хорошие признаки для построения дерева, приведем игрушечный пример. Будем предсказывать цвет шарика по его координате.



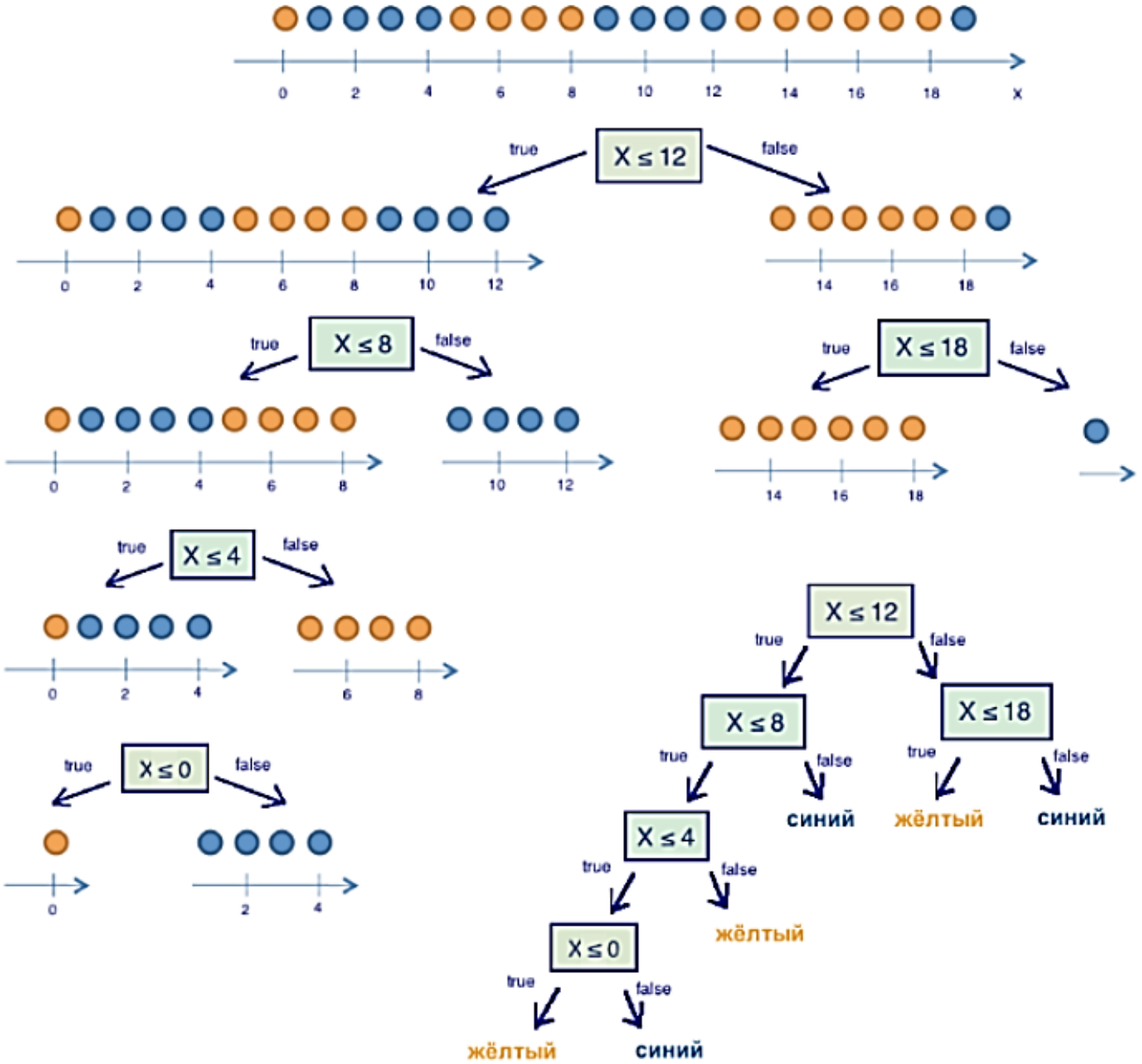
Здесь 9 синих шариков и 11 желтых. Если мы наудачу вытащили шарик, то он с вероятностью $p_1 = \frac{9}{20}$ будет синим и с вероятностью $p_2 = \frac{11}{20}$ – желтым. Значит, энтропия состояния $S_0 = -\frac{9}{20}\log_2\frac{9}{20} - \frac{11}{20}\log_2\frac{11}{20} \approx 1$. Само это значение пока ни о чем нам не говорит. Теперь посмотрим, как изменится энтропия, если разбить шарики на две группы – с координатой меньше либо равной 12 и больше 12.

https://habr.com/ru/company/ods/blog/322534/



В левой группе оказалось 13 шаров, из которых 8 синих и 5 желтых. Энтропия этой группы равна $S_1 = -\frac{5}{13}\log_2\frac{5}{13} - \frac{8}{13}\log_2\frac{8}{13} \approx 0.96$. В правой группе оказалось 7 шаров, из которых 1 синий и 6 желтых. Энтропия правой группы равна $S_2 = -\frac{1}{7}\log_2\frac{1}{7} - \frac{6}{7}\log_2\frac{6}{7} \approx 0.6$. Как видим, энтропия уменьшилась в обеих группах по сравнению с начальным состоянием, хоть в левой и не сильно.

Получается, разделив шарики на две группы по признаку "координата меньше либо равна 12", мы уже получили более упорядоченную систему, чем в начале. Продолжим деление шариков на группы до тех пор, пока в каждой группе шарики не будут одного цвета.

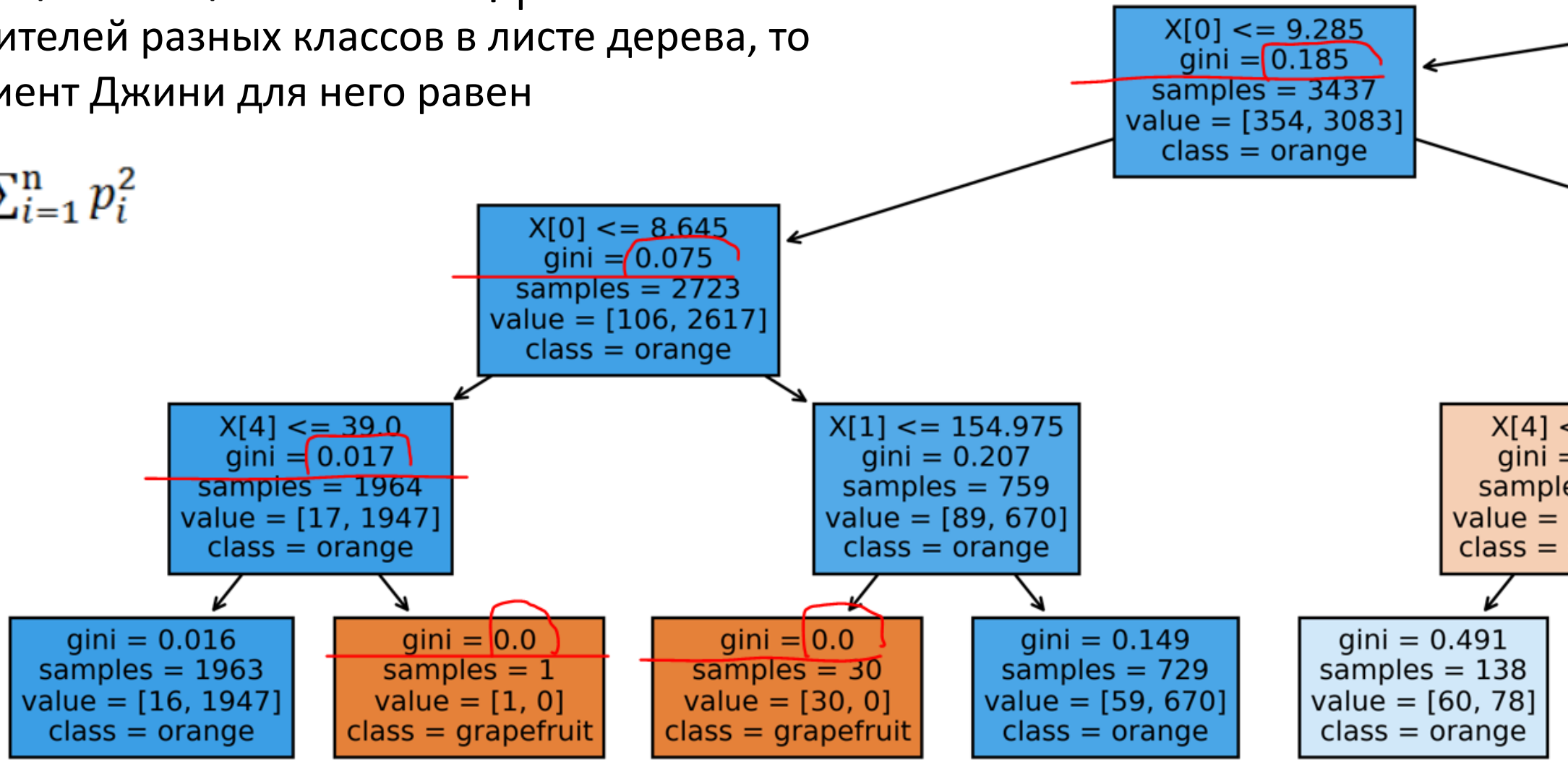


Вверху дерева – общие правила
Внизу - частные

Очевидно, энтропия группы с шариками одного цвета равна 0 ($\log_2 1 = 0$), что соответствует представлению, что группа шариков одного цвета – упорядоченная.

Индекс Джини (Gini impurity) используется в решающих деревьях при выборе расщепления. Он измеряет «равномерность», если p_i — частоты представителей разных классов в листе дерева, то коэффициент Джини для него равен

$$1 - \sum_{i=1}^{n} p_i^2$$



Переобучение и недообучение модели

Если модель может выдавать точные прогнозы на ранее не встречавшихся данных, мы говорим, что модель обладает **способностью обобщать** (generalize) результат на тестовые данные.

Необходимо построить модель, которая будет обладать максимальной обобщающей способностью.

Если обучающий и тестовый наборы имеют много общего между собой, можно ожидать, что модель будет точной и на тестовом наборе. Однако в некоторых случаях этого не происходит.

Чем сложнее модель, тем лучше она будет работать на обучающих данных. Однако, если наша модель становится слишком сложной, мы начинаем уделять слишком много внимания каждой отдельной точке данных в нашем обучающем наборе, и **эта модель не будет хорошо обобщать результат на новые данные.**

Существует оптимальная точка, которая позволяет получить наилучшую обобщающую способность.

Рис. Компромисс между сложностью модели и правильностью на обучающей и тестовой выборка

Переобученное дерево

Переобученное дерево может иметь по одному объекту в листьях.

Переобученные деревья имеют сложную структуру и их сложно интерпретировать.

Для избегания переобучения деревьев разработаны следующие подходы.

- ✓ **Ранняя остановка** — алгоритм будет остановлен, как только будет достигнуто заданное значение некоторого критерия, например процентной доли правильно распознанных примеров. Преимуществом подхода является снижение времени обучения. Недостатком снижение точности дерева.
- ✓ **Ограничение глубины дерева** — задание максимального числа разбиений в ветвях, по достижении которого обучение останавливается. Это тоже снижает точность дерева.
- ✓ **Задание минимально допустимого число примеров в узле** — запретить алгоритму создавать узлы с числом примеров меньше заданного (например, 5). Это позволит избежать создания малозначимых правил.

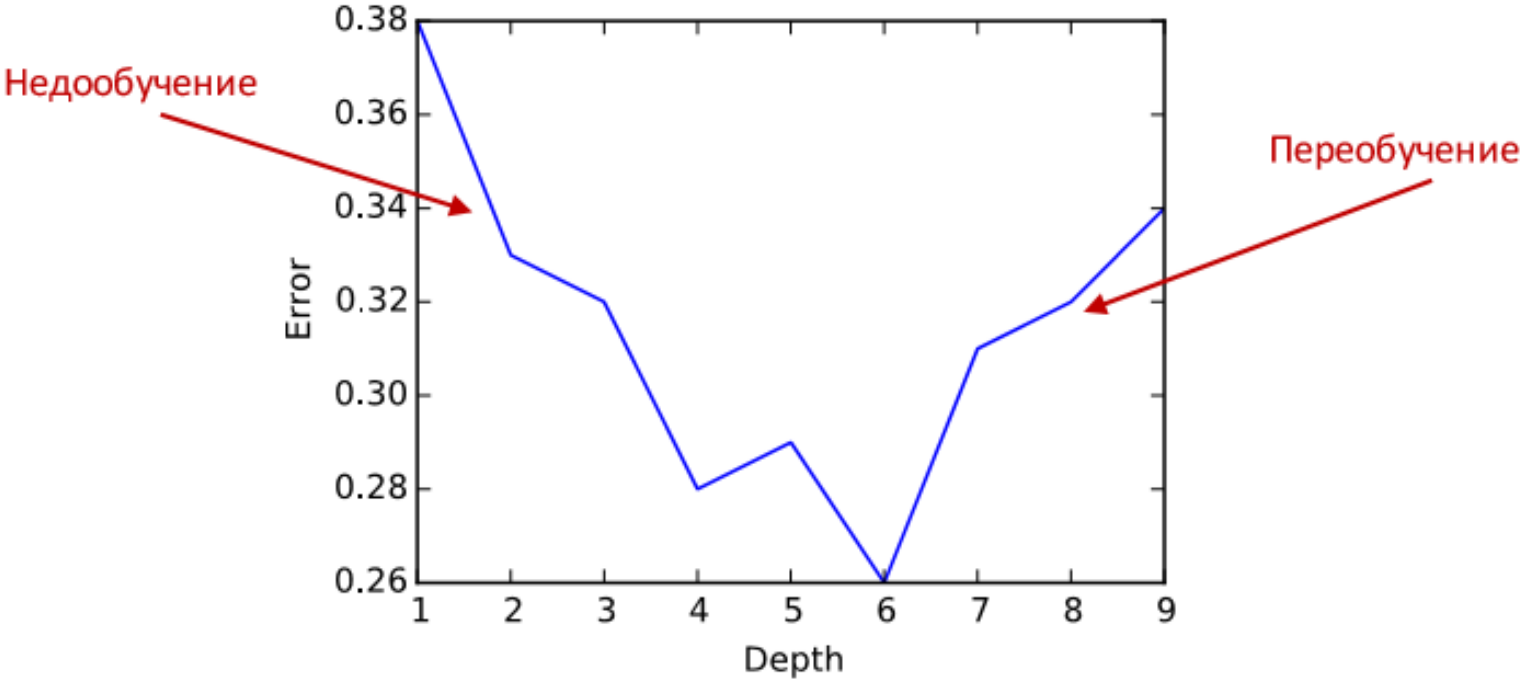
Как найти оптимальную глубину дерева?

Можно сравнить результаты деревьев разных глубин и выбрать наибольшую точность.

Например, сравнив точность деревьев, видим, что оптимальная глубина в данном случае – 3.

1- 72% 2-84% **3-86%** 4-76%

Можно построить график зависимости точности от глубины, или как в данном примере зависимости ошибки от глубины дерева и визуально определить оптимальную глубиину.



К <u>достоинствам</u> деревьев решений следует отнести:

- Возможность производить обучение на исходных данных без их дополнительной предобработки (нормализация и т. п.);
- Нечувствительность к монотонным преобразованиям данных;
- Устойчивость к выбросам;
- Поддержка работы с входными переменными разных типов;
- Возможность интерпретации построенного дерева решений.

<u>Недостаток</u>:

- Деревья решений не устойчивы даже к небольшим изменениям.
- Проблематичность построения оптимального дерева решений. Построение и поиск такого дерева решений являются NP- полной задачей, сложно разрешимой на практике. Поэтому практическое построение деревьев решений связано с применением эвристических «жадных» алгоритмов, оптимальных только в каждом узле дерева, но не оптимальных для дерева в целом.

Scikit-learn — библиотека для машинного обучения, построена на NumPy, SciPy и matplotlib

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.DecisionTreeClassifier(*, *criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*)

```
from sklearn.tree import DecisionTreeClassifier
```

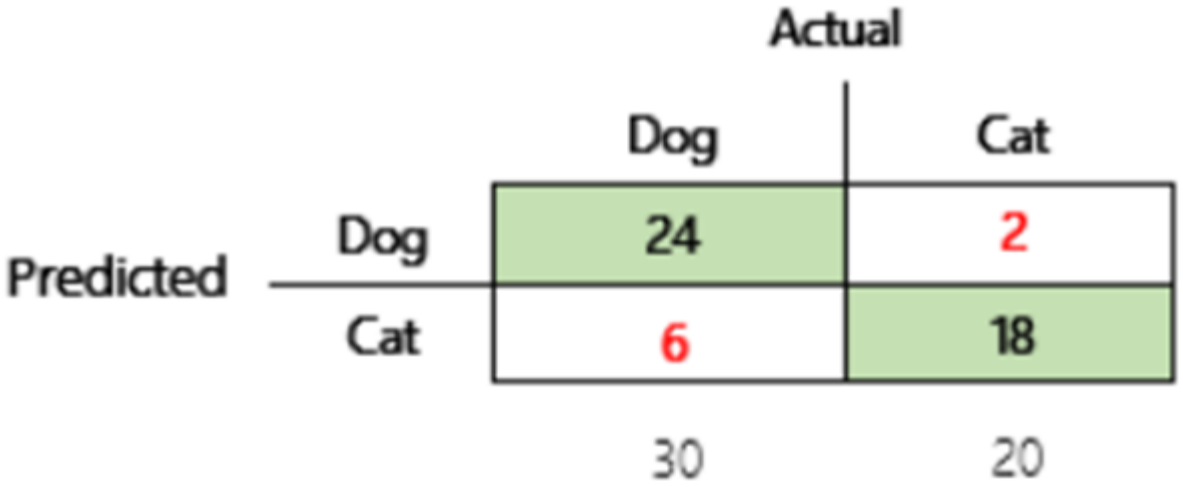
```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

x		y
0	0	0
1	1	1

Матрица ошибок (Confusion matrix)

После того, как модель ML обучена на обучающей выборке, она тестируется на тестовой выборке, которые можно исключить из процесса обучения. Таким образом можно сравнить прогнозы обученной модели с фактическими значениями. Матрица ошибок предоставляет средство оценки успешности решения задачи классификации и мест возникновения ошибок (то есть, когда она "путается").

Например, ваша цель — предсказать, является ли домашнее животное собакой или кошкой, на основе некоторых физических и поведенческих атрибутов. Если имеется тестовый набор данных, содержащий **30 собак и 20 кошек**, то матрица ошибок может быть похожа на следующую иллюстрацию.

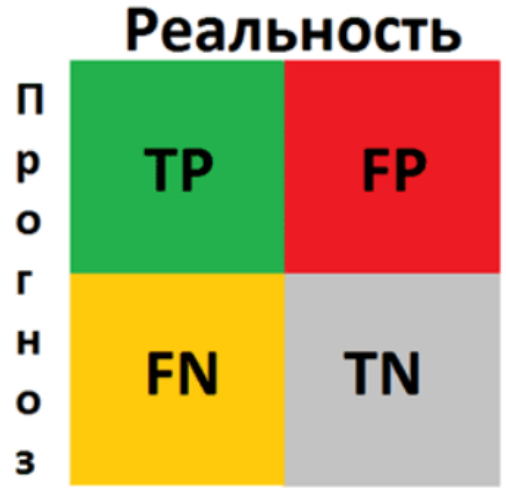


Числа в зеленых ячейках представляют собой правильные прогнозы.

Общую точность модели легко рассчитать. В данном случае это 42 ÷ 50 или 0,84.

Матрица ошибок (confusion matrix) используется с целью сопоставления предсказаний и реальности в Data Science. Это таблица с 4 различными комбинациями прогнозируемых и фактических значений. Прогнозируемые значения описываются как положительные и отрицательные, а фактические – как истинные и ложные

Прогноз	Реальность	
	+	–
+	***True Positive (истинно-положительное решение)***: прогноз совпал с реальностью, результат положительный произошел, как и было предсказано ML-моделью	***False Positive (ложноположительное решение)***: ошибка 1-го рода, ML-модель предсказала положительный результат, а на самом деле он отрицательный
–	***False Negative (ложноотрицательное решение)***: ошибка 2-го рода – ML-модель предсказала отрицательный результат, но на самом деле он положительный	***True Negative (истинно-отрицательное решение)***: результат отрицательный, ML-прогноз совпал с реальностью



Матрица ошибок (confusion matrix)

P – число истинных результатов, P = TP + FN;
N – число ложных результатов, N = TN + FP.

Точность (Accuracy) – доля правильных ответов алгоритма;

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

Accuracy не работает в случае несбалансированных классов
пример на след слайде

Precision (точность) – доля объектов, названных классификатором положительными и при этом действительно являющимися положительными

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

Recall (полнота) показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

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

Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а **precision** — способность отличать этот класс от других классов.
Precision и **recall** не зависят, в отличие от **accuracy**, от соотношения классов и потому применимы в условиях несбалансированных выборок.

$accuracy$ бесполезна в задачах с неравными классами, и это легко показать на примере.

https://habr.com/ru/companies/ods/articles/328372/

Допустим, мы хотим оценить работу спам-фильтра почты. У нас есть 100 не-спам писем, 90 из которых наш классификатор определил верно (True Negative = 90, False Positive = 10), и 10 спам-писем, 5 из которых классификатор также определил верно (True Positive = 5, False Negative = 5). Тогда accuracy:

$$accuracy = \frac{5 + 90}{5 + 90 + 10 + 5} = 86,4$$

Однако если мы просто будем предсказывать все письма как не-спам, то получим более высокую accuracy:

$$accuracy = \frac{0 + 100}{0 + 100 + 0 + 10} = 90,9$$

При этом, наша модель совершенно не обладает никакой предсказательной силой, так как изначально мы хотели определять письма со спамом. Преодолеть это нам поможет переход с общей для всех классов метрики к отдельным показателям качества классов.

In[58]:

```
from sklearn.tree import DecisionTreeClassifier

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(tree.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(tree.score(X_test, y_test)))
```

Out[58]:

```
Правильность на обучающем наборе: 1.000
Правильность на тестовом наборе: 0.937
```

In[59]:

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

print("Правильность на обучающем наборе: {:.3f}".format(tree.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(tree.score(X_test, y_test)))
```

Out[59]:

```
Правильность на обучающем наборе: 0.988
Правильность на тестовом наборе: 0.951
```

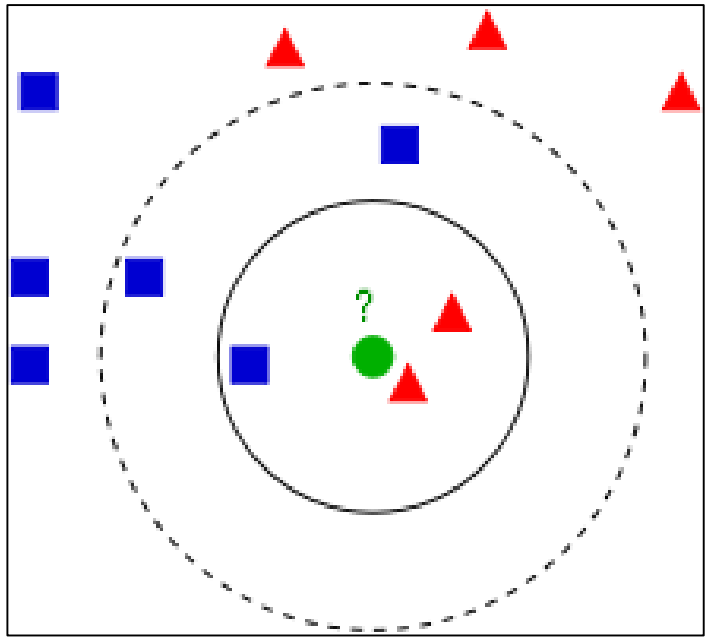
Алгоритм kNN (k Nearest Neighbor или k Ближайших Соседей)

Метрический алгоритм для классификации объектов или регрессии.

Гипотеза компактности (схожие параметры ведут к схожим меткам)

В случае использования метода для классификации объект присваивается тому классу, который является наиболее распространённым среди k соседей данного элемента, классы которых уже известны. В случае использования метода для регрессии, объекту присваивается среднее значение по k ближайшим к нему объектам, значения которых уже известны.

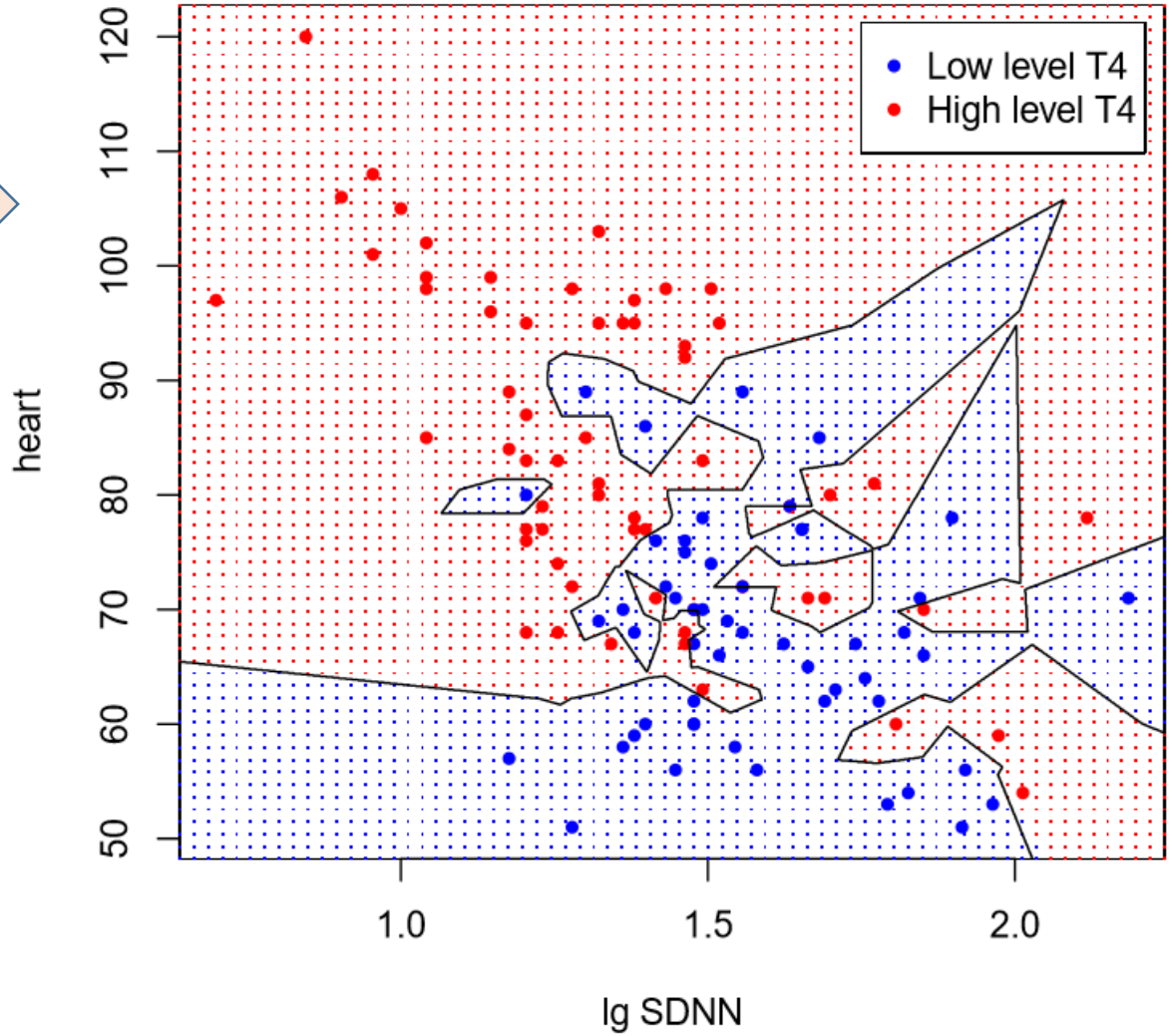
Алгоритм может быть применим к выборкам с большим количеством атрибутов (многомерным). Для этого перед применением нужно определить функцию расстояния; классический вариант такой функции — евклидова метрика



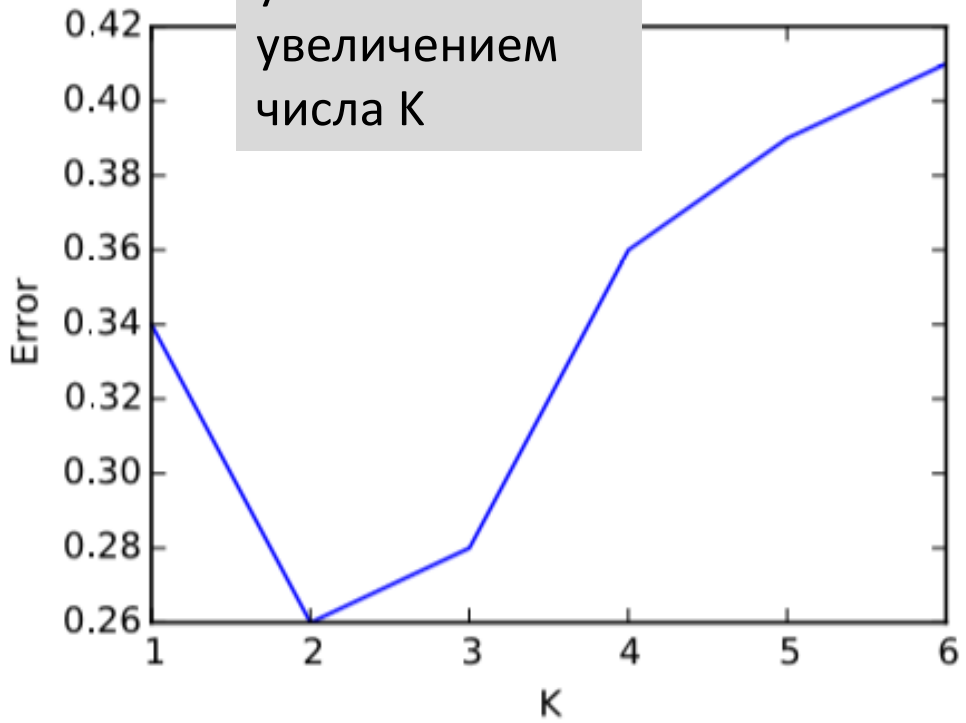
Пример классификации k-ближайших соседей. Тестовый образец (зелёный круг) должен быть классифицирован как синий квадрат (класс 1) или как красный треугольник (класс 2). Если k = 3, то он классифицируется как 2-й класс, потому что внутри меньшего круга 2 треугольника и только 1 квадрат. Если k = 5, то он будет классифицирован как 1-й класс (3 квадрата против 2 треугольников внутри большего круга)

Переобучение в методе k ближайших соседей

Здесь явное переобучение, модель очень точно описывает все объекты выборки



Ошибка увеличивается с увеличением числа K



Метод ближайшего соседа (с масштабированием)

Ошибка на обучающей выборке — 0 %.

Достоинства и недостатки алгоритма

- устойчивость к выбросам и аномальным значениям, поскольку вероятность попадания содержащих их записей в число k-ближайших соседей мала;
- программная реализация алгоритма относительно проста;
- результаты работы алгоритма легко поддаются интерпретации.

К недостаткам алгоритм KNN можно отнести:

- данный метод не создает каких-либо моделей, обобщающих предыдущий опыт, а интерес могут представлять и сами правила классификации;
- при классификации объекта используются все доступные данные, поэтому метод KNN является достаточно затратным в вычислительном плане, особенно в случае больших объёмов данных;
- высокая трудоёмкость из-за необходимости вычисления расстояний до всех примеров;
- повышенные требования к репрезентативности исходных данных.

Ещё одной проблемой алгоритма KNN, характерной, впрочем, и для большинства методов классификации, является различная значимость признаков с точки зрения определения класса объектов. Учет фактора значимости признаков в алгоритме может позволить повысить точность классификации.

sklearn.neighbors.NearestNeighbors

class **sklearn.neighbors.NearestNeighbors**(*, *n_neighbors=5*, *radius=1.0*, *algorithm='auto'*, *leaf_size=30*, *metric='minkowski'*, *p=2*, *metric_params=None*, *n_jobs=None*) [source]

```
from sklearn.neighbors import KNeighborsClassifier

first_knn= KNeighborsClassifier(n_neighbors=3)

knn_model=first_knn.fit(X_train,y_train)

knn_model.score(X_train,y_train)
```