



# Лекция 10: деревья решений и ансамбли моделей

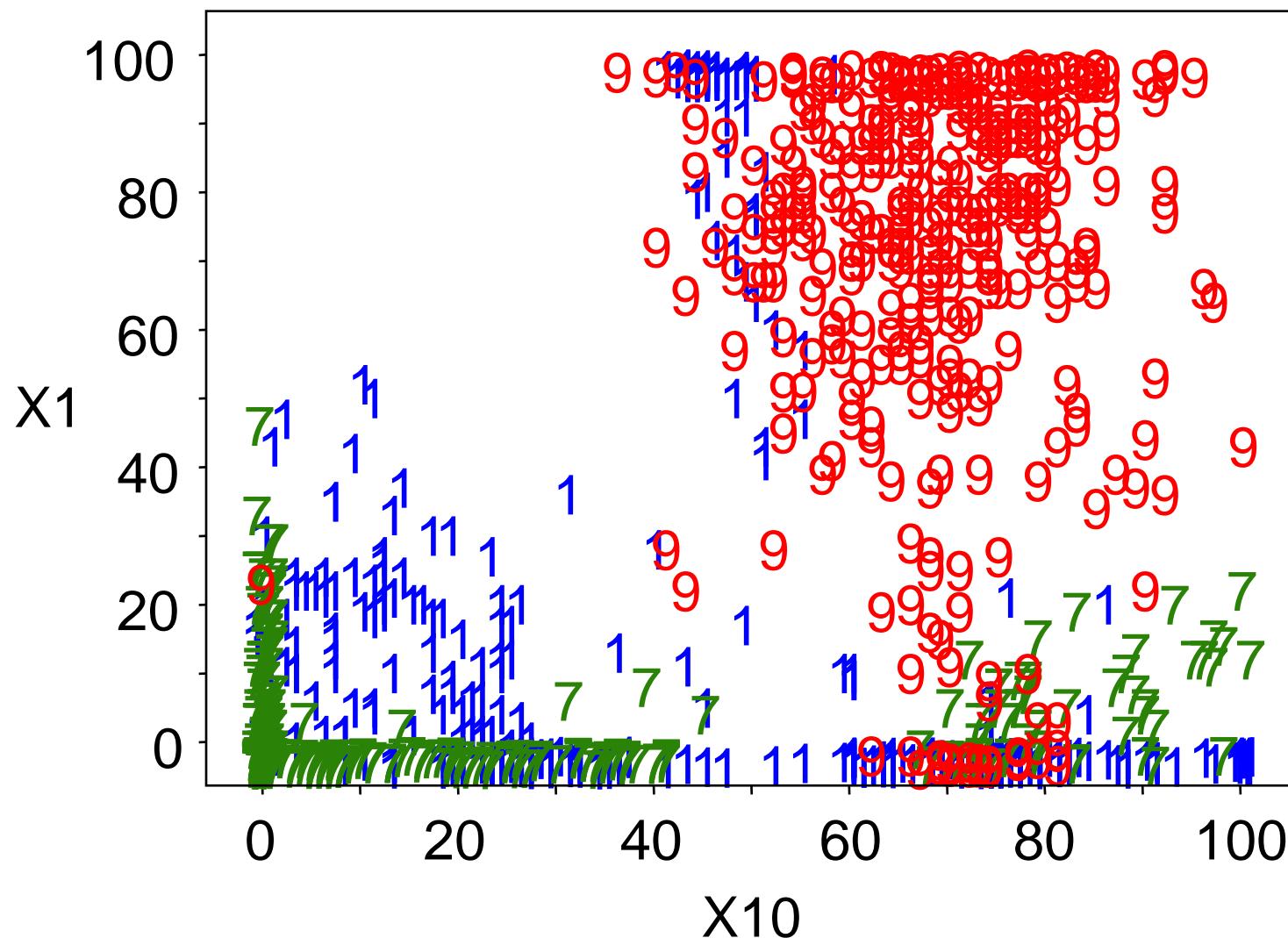
# Методы, основанные на деревьях решений

- Эти методы используют *стратификацию или сегментирование* пространства признаков на области.
- Для сегментации пространства признаков может использоваться набор правил, который можно представить в виде дерева
- Деревья решений могут применяться как к задачам регрессии, так и к классификации.
- Методы, основанные на деревьях, просты в интерпретации, при этом показывают достаточно хорошие результаты по точности прогнозирования.
- Нестабильные модели – это плюс для ансамблей бэггинг, *методы случайного леса и бустинг*. Эти методы строят множество деревьев, результаты прогнозирования которых потом объединяются для получения итогового прогноза.

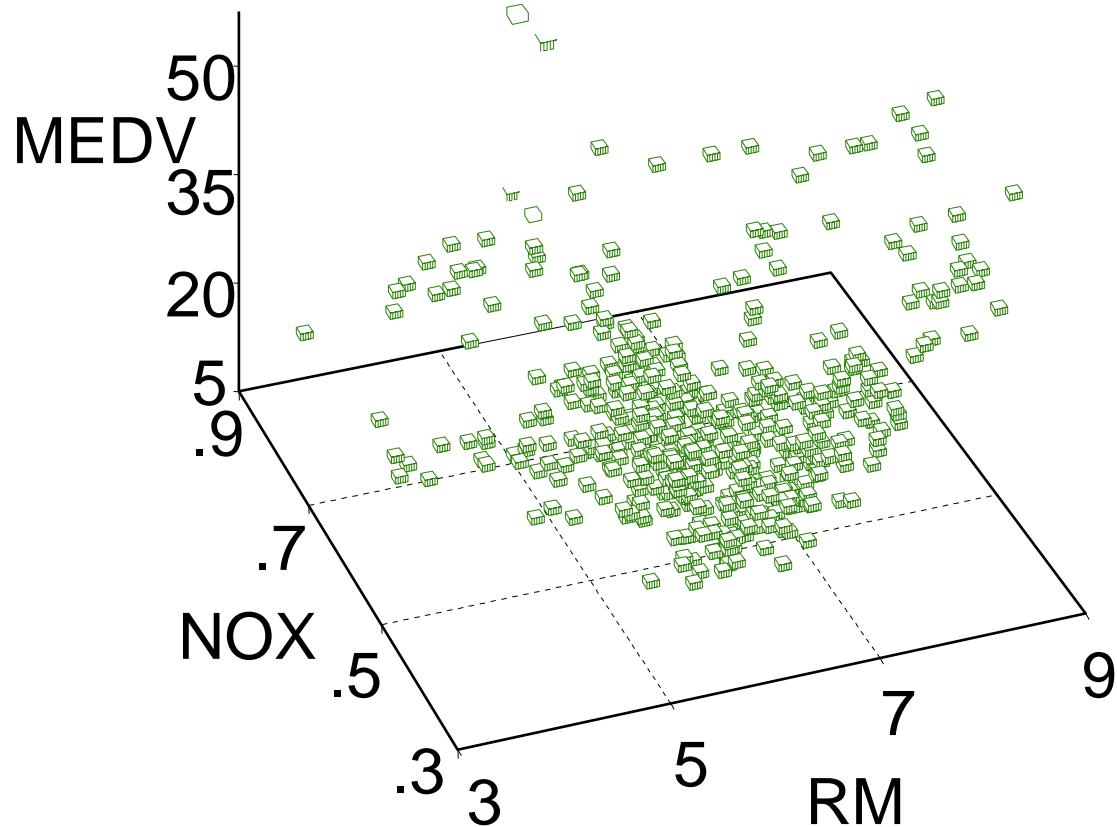
# Деревья решений в задачах классификации и регрессии

- Дерево решений - граф (древовидная структура), в котором:
  - Внутренние узлы – условия на атрибуты
  - Каждая исходящая ветка соответствует выходному значению условия, ветка целиком – альтернативное решение
  - В листьях метки классов (или распределение меток классов) или значения целевой переменной для регрессии
  - Каждому узлу соответствует область в пространстве признаков R
  - Области для листьев – финальные, не содержат внутри других областей
- Построение дерева обычно – 2 фазы
  - Построение: в начале в корне все примеры, далее рекурсивное разбиение множества примеров по выбранному атрибуту
  - «отсечение» ветвей pruning - выявление и удаление ветвей (решений), приводящих к шуму или к выбросам
- Применение дерева решений для нового объекта
  - Проверка атрибутов – путь по ветви до листа. В листе отклик.

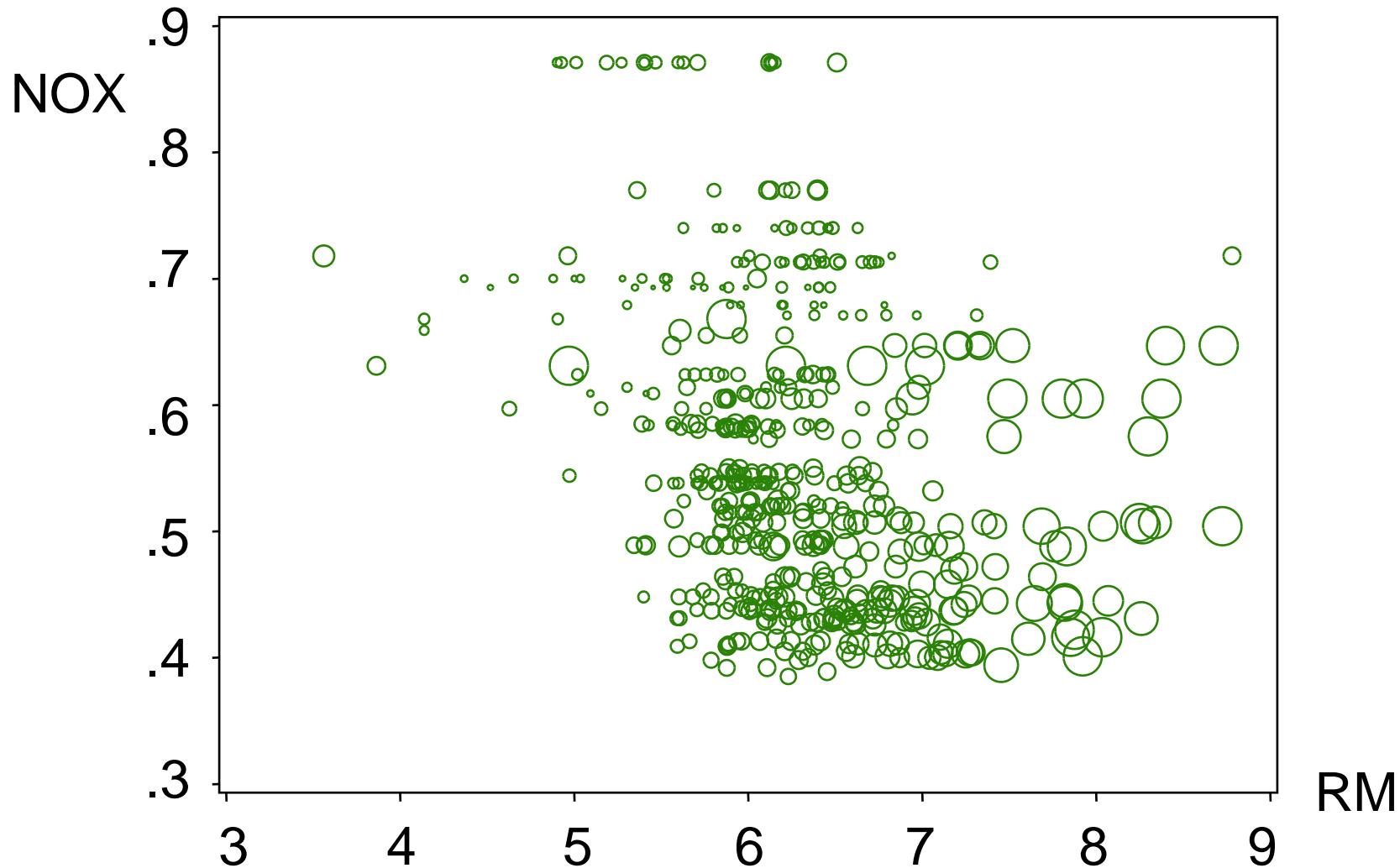
# Пример: категориальный отклик



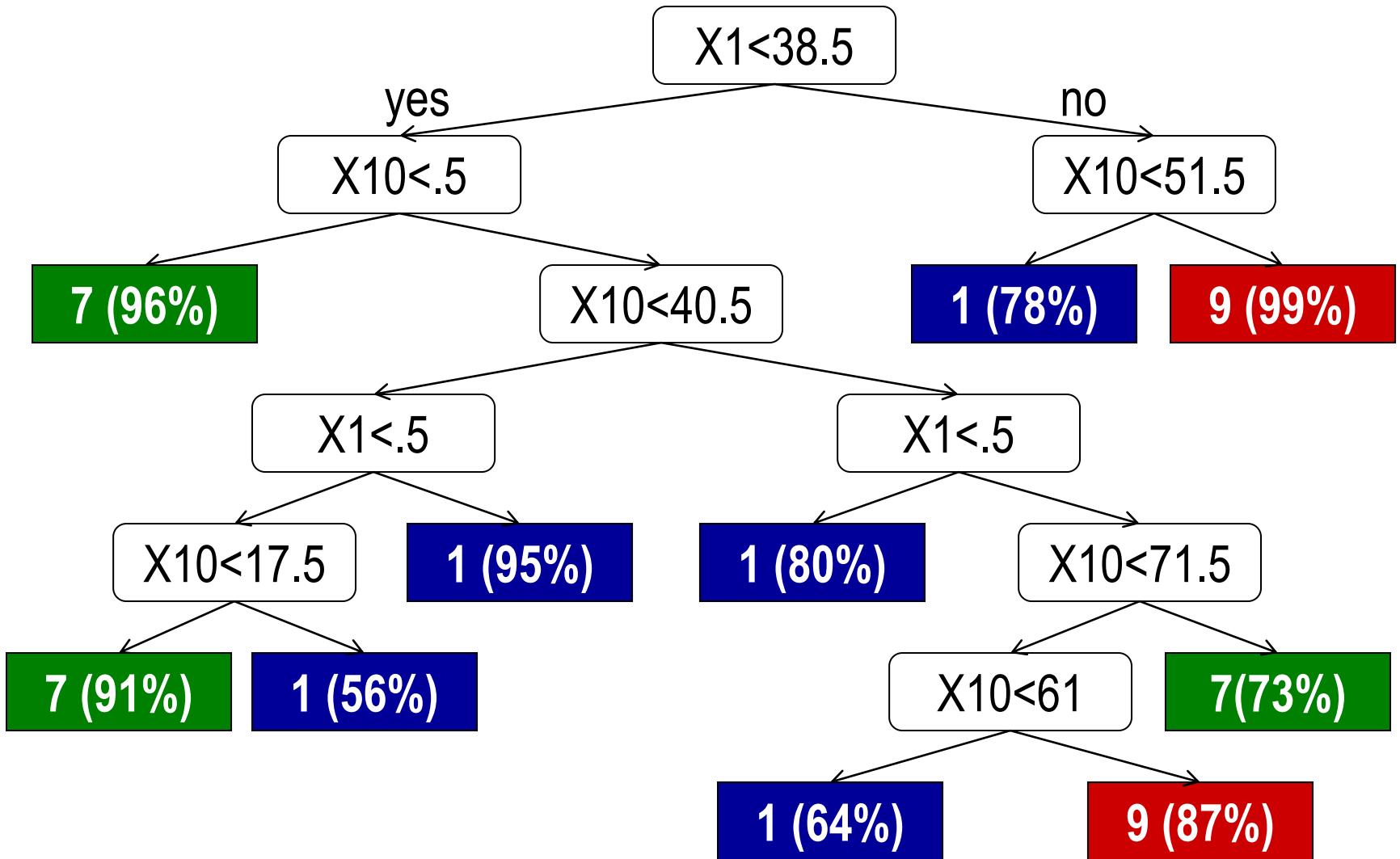
# Пример: непрерывный отклик



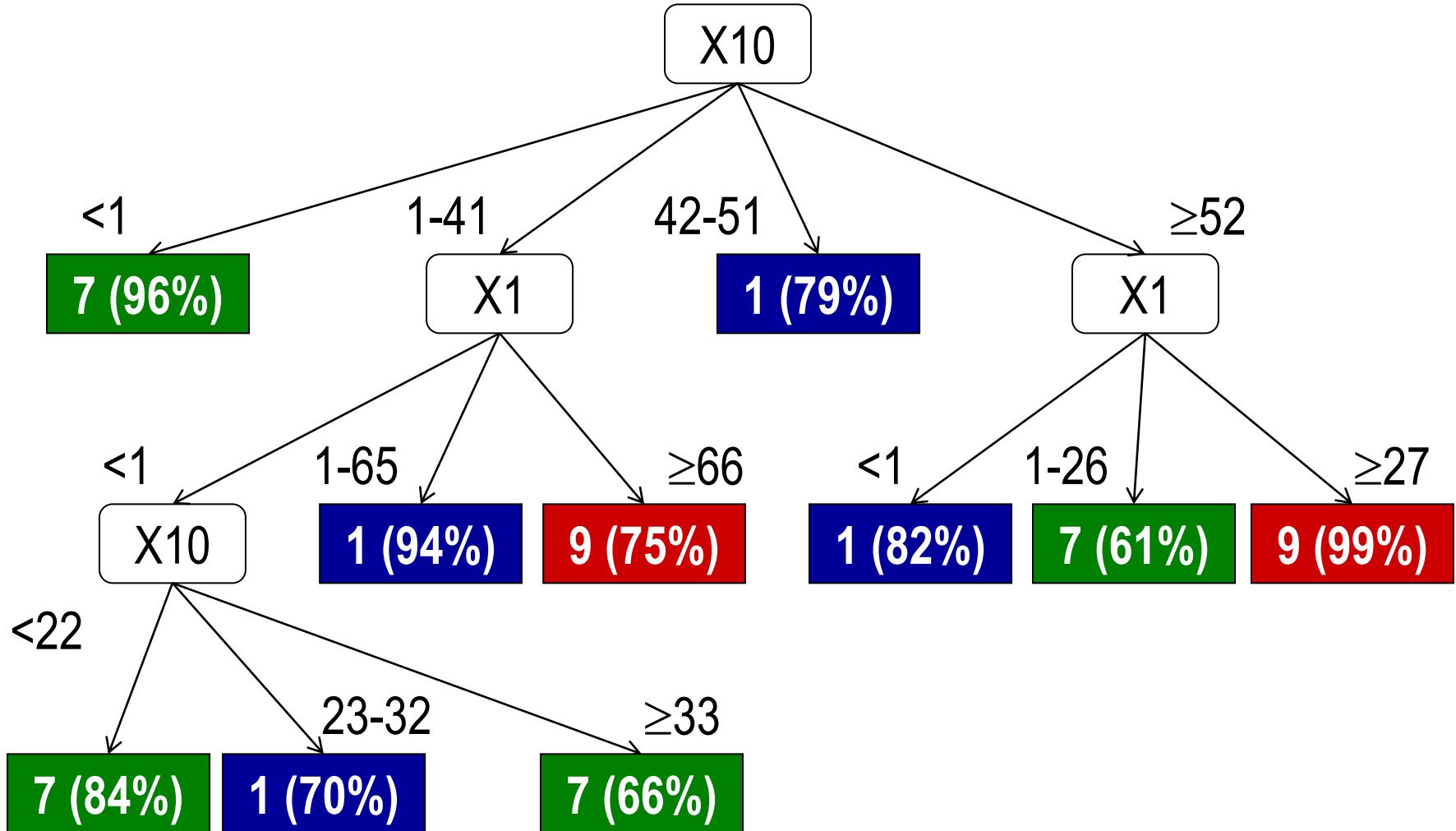
# Проекция непрерывного отклика на пространство признаков



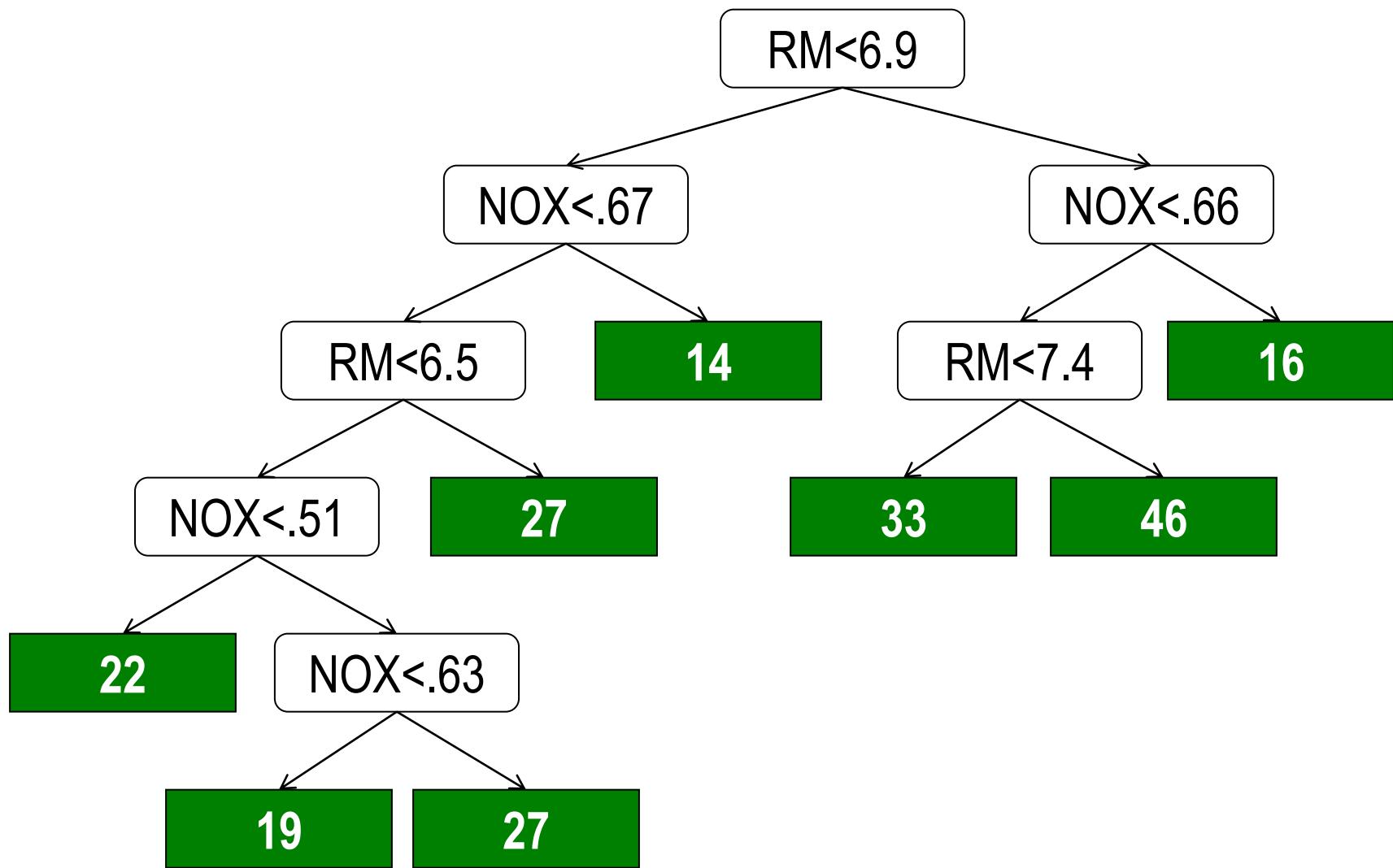
# Дерево решений для классификации



# Множественные разбияния (не бинарное дерево)



# Дерево решений для регрессии

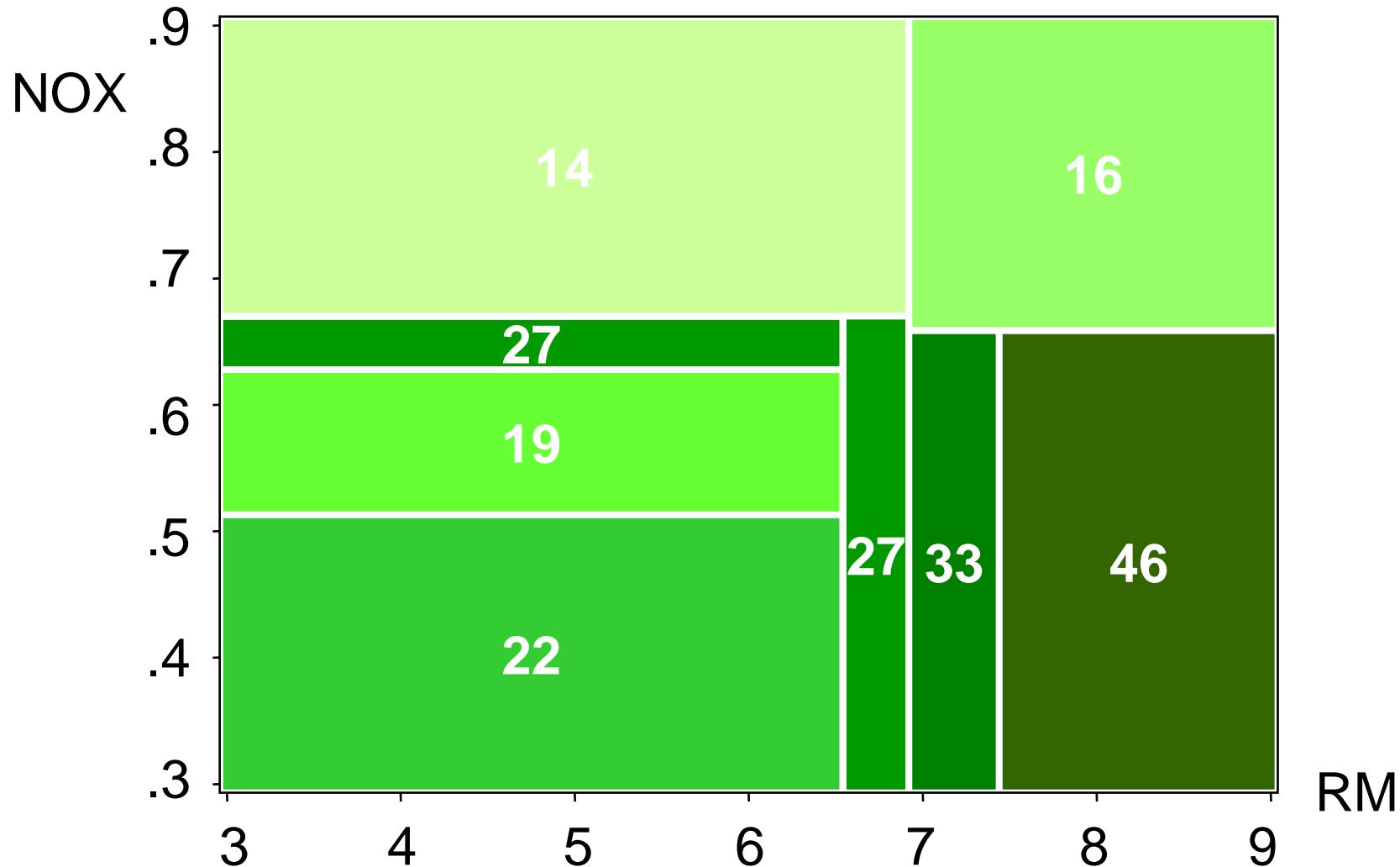


# Листья = Логические правила

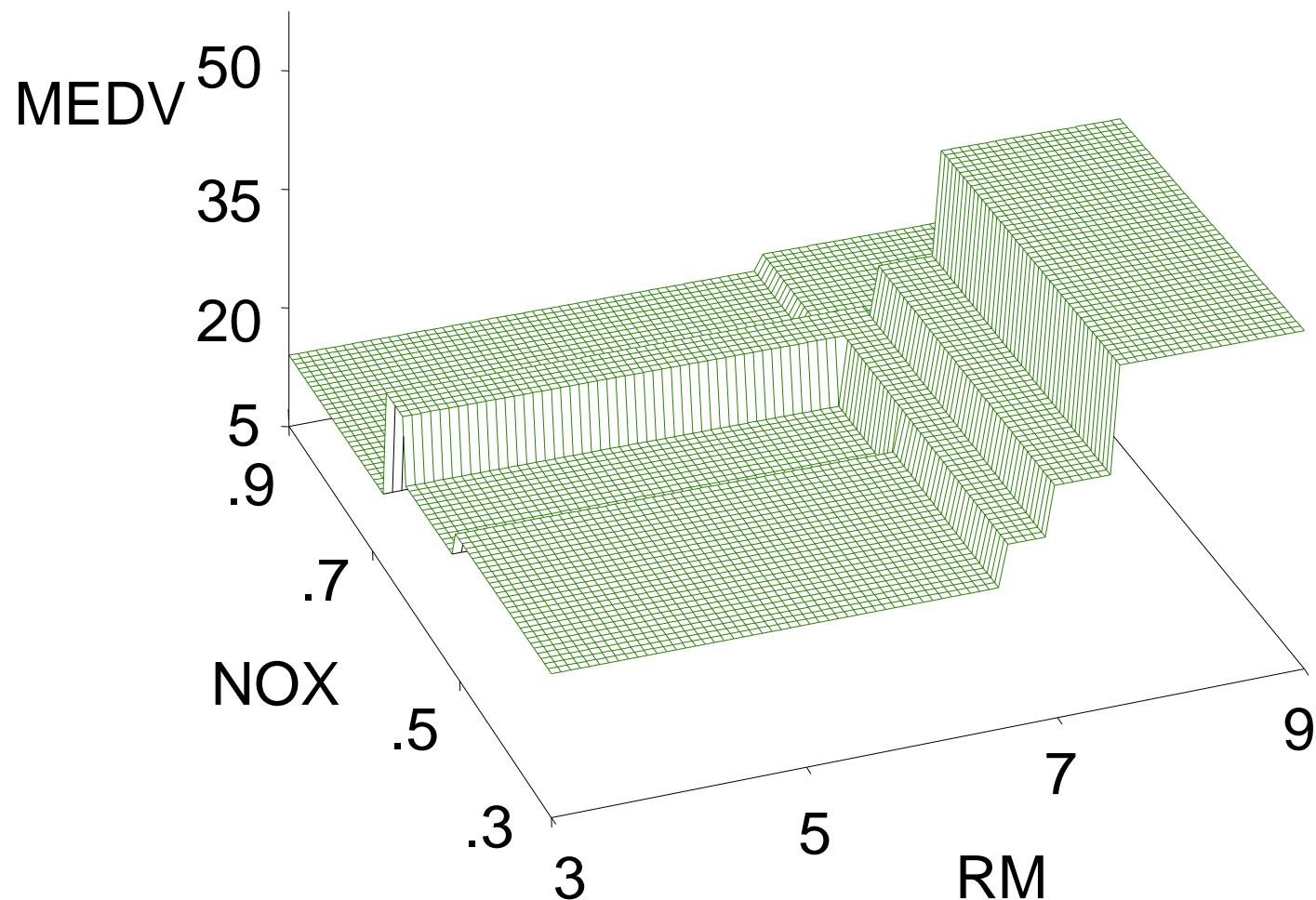
If  $RM \in \{values\}$  and  $NOX \in \{values\}$ , then MEDV=value.

<u>Leaf</u>	<u>RM</u>	<u>NOX</u>	<u>Predicted MEDV</u>
1	<6.5	<.51	22
2	<6.5	[.51, .63)	19
3	<6.5	[.63, .67)	27
4	[6.5, 6.9)	<.67	27
5	<6.9	$\geq$ .67	14
6	[6.9, 7.4)	<.66	33
7	$\geq$ 7.4	<.66	46
8	$\geq$ 6.9	$\geq$ .66	16

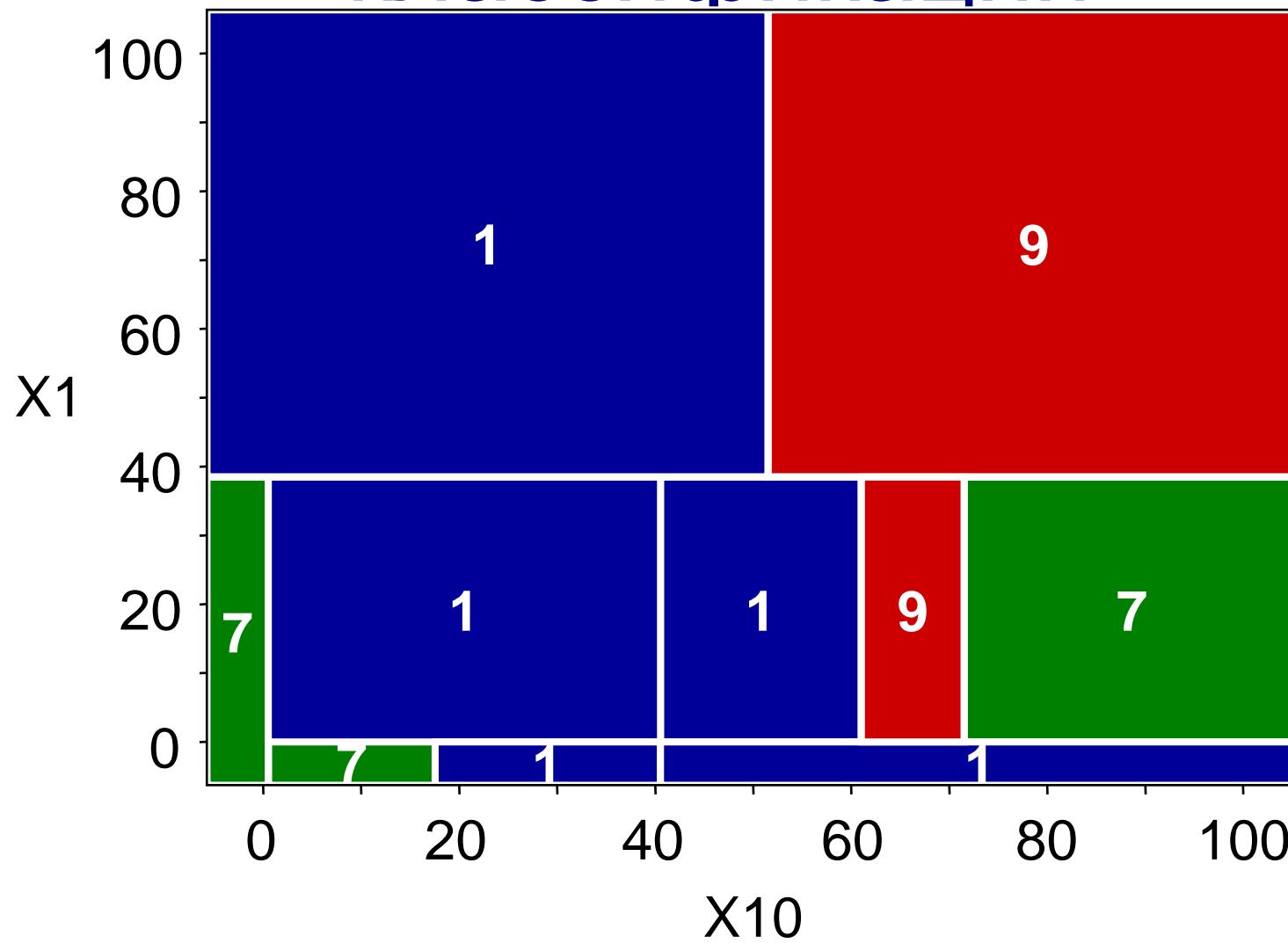
# Разбиение пространства признаков на области



# Многомерная ступенчатая функция



# Регионы решений для классификации



# Листья дерева решений для классификации

<u>Leaf</u>	<u>Pr(1 x)</u>	<u>Pr(7 x)</u>	<u>Pr(9 x)</u>	<u>Decision</u>
1	.03	.96	.01	7
2	.09	.91	.00	7
3	.56	.44	.00	1
4	.95	.05	.00	1
5	.80	.10	.10	1
6	.64	.09	.27	1
7	.00	.13	.87	9
8	.10	.73	.17	7
9	.78	.01	.21	1
10	.01	.00	.99	9

# Процесс построения деревьев решений

- Разделяем пространство признаков (то есть, набор возможных значений для  $X_1, X_2, \dots, X_p$ ) в  $J$  отдельных и непересекающихся областей  $R_1, R_2, \dots, R_J$ . Таким образом, чтобы поведение отклика в каждой области было «однородным».
- Теоретически, области могут иметь любую форму. Тем не менее, чаще всего используются многомерные *прямоугольники* для простоты и удобства интерпретации полученной модели.
- Цель состоит в том, чтобы найти прямоугольные области  $R_1, \dots, R_J$ , так чтобы минимизировать некий целевой критерий – *критерий разбиения*.

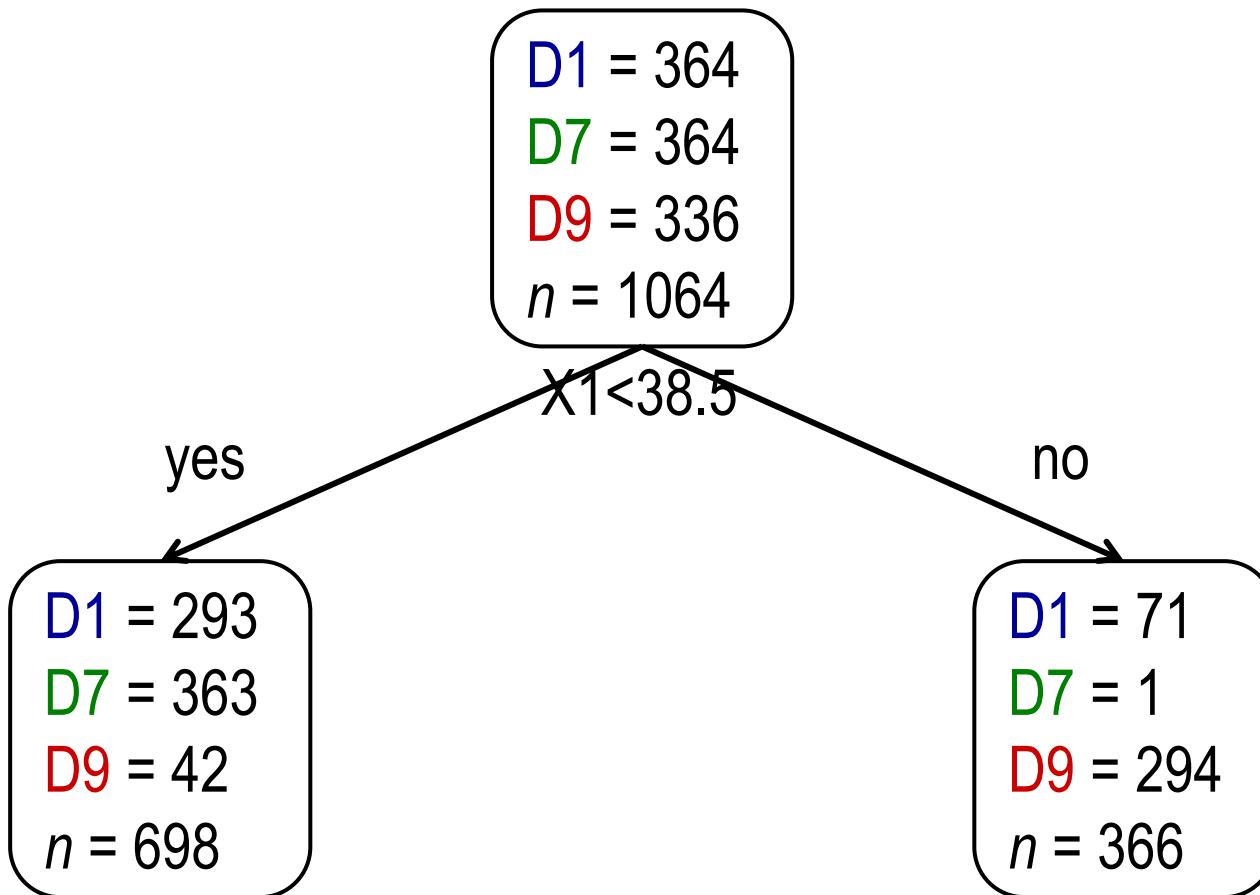
# Процесс построения деревьев решений

- Вычислительно нецелесообразно рассматривать все возможные разбиения пространства признаков в  $J$  областей.
- По этой причине используется *находящий, жадный* подход, известный как рекурсивное разделение.
- Подход называется *находящий*, потому что он начинается в верхней части дерева, а затем последовательно разбивает пространство признаков; каждое разбиение приводит к образованию новых ветвей, расположенных ниже по дереву.
- Подход называется *жадным*, потому что на каждом этапе процесса построения деревьев лучшее разбиение осуществляется на конкретном шаге, вместо того, чтобы просматривать дальше и выбирать разбиение, которое приведет к лучшему дереву в дальнейшем.

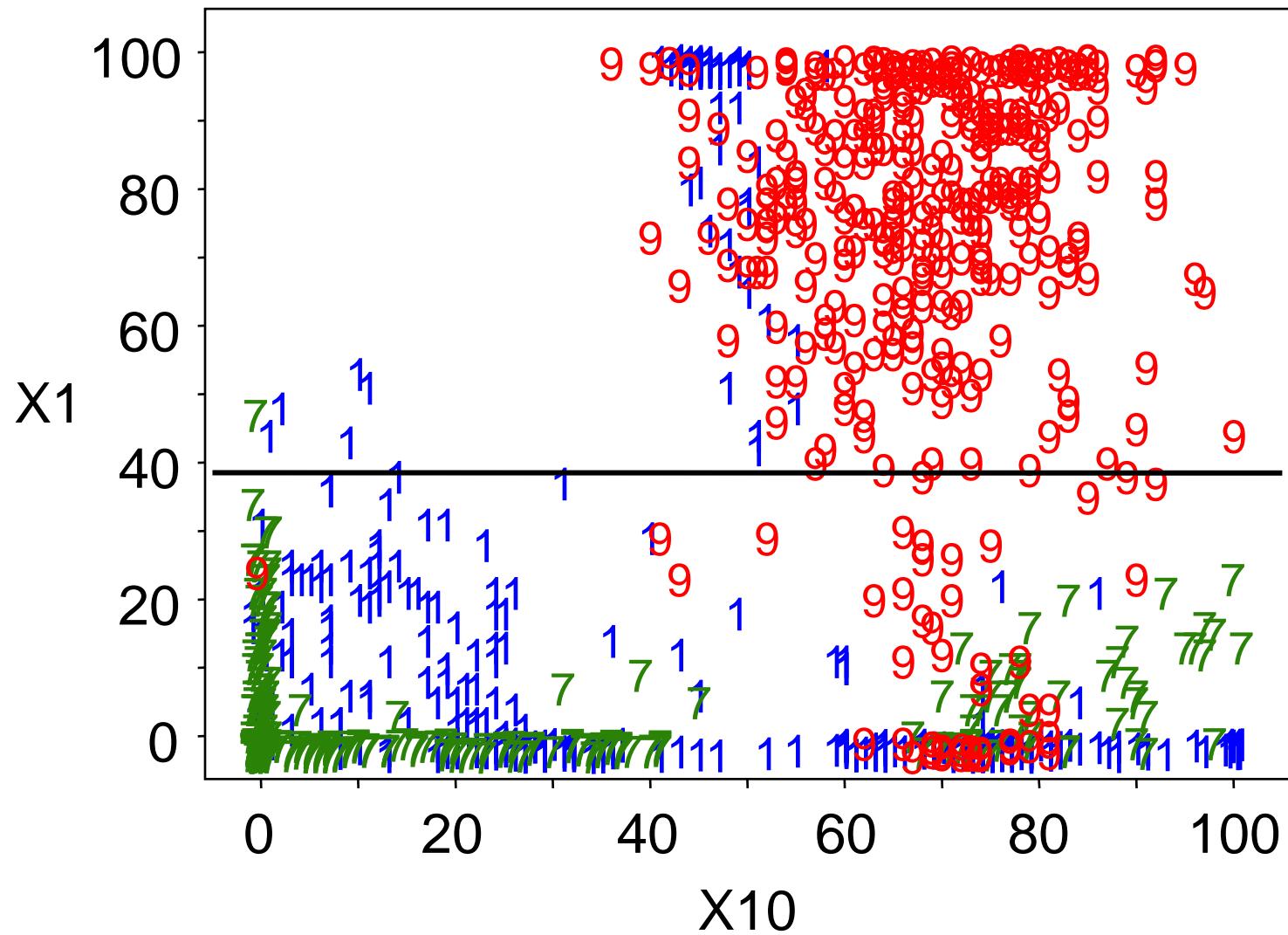
# Процесс построения деревьев решений

- Сначала выбирается переменная  $X_j$  и точка разбиения  $s$ , так что разбиение пространства признаков на области  $\{X|X_j < s\}$  и  $\{X|X_j \geq s\}$  приводит к максимально возможному улучшению критерия (для бинарного дерева).
- Затем повторяется этот процесс и ищется лучшая переменная и точка разбиения в каждой из результирующих областей.
- Однако на этот раз, вместо разделения всего пространства признаков, мы разделяем одну из ранее выделенных областей.
- Процесс продолжается до тех пор, пока не будет достигнут критерий останова; например, мы можем продолжать процесс до тех пор, пока не останется областей, содержащих более пяти наблюдений.

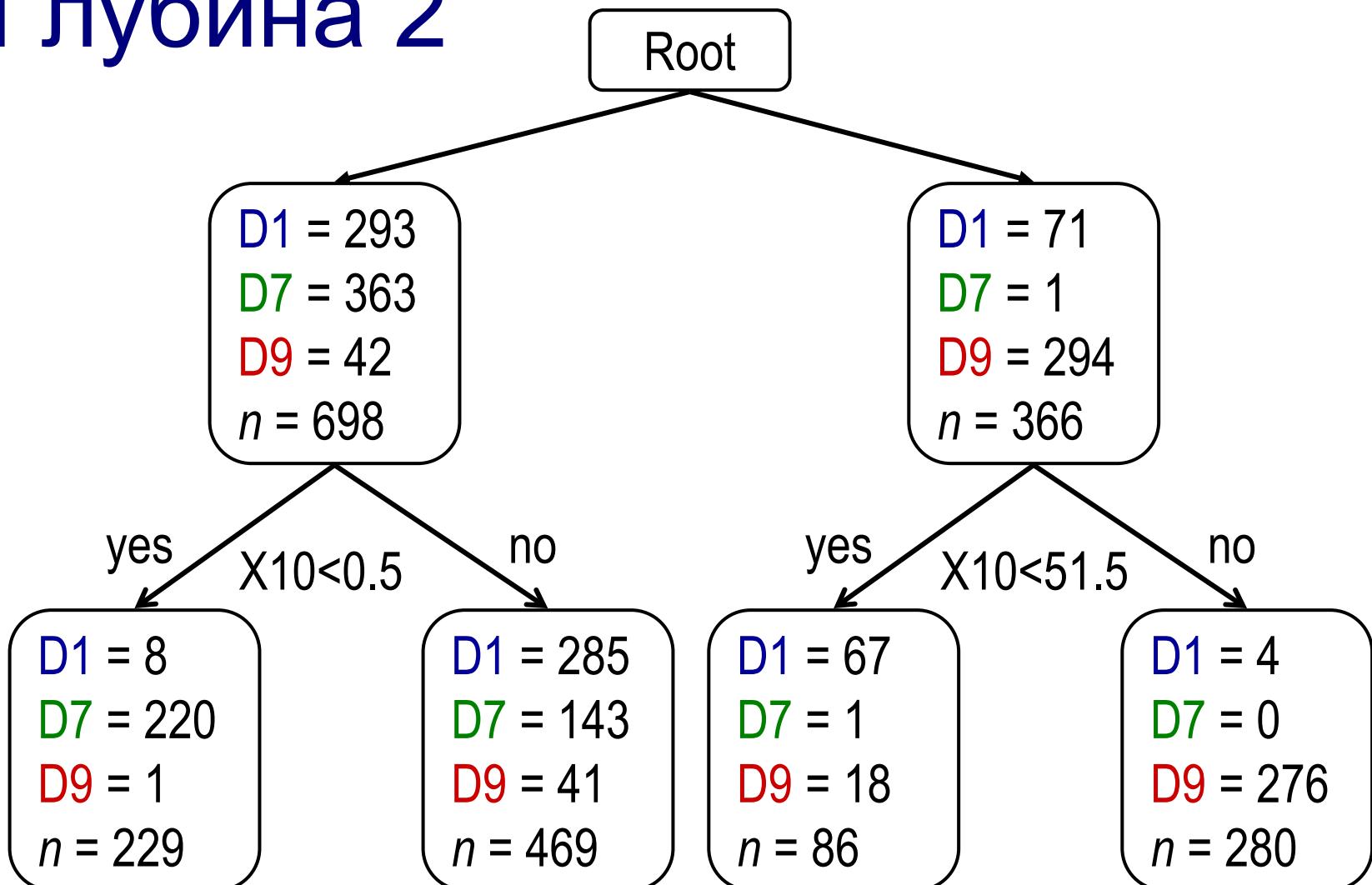
# Шаг разбиения узла



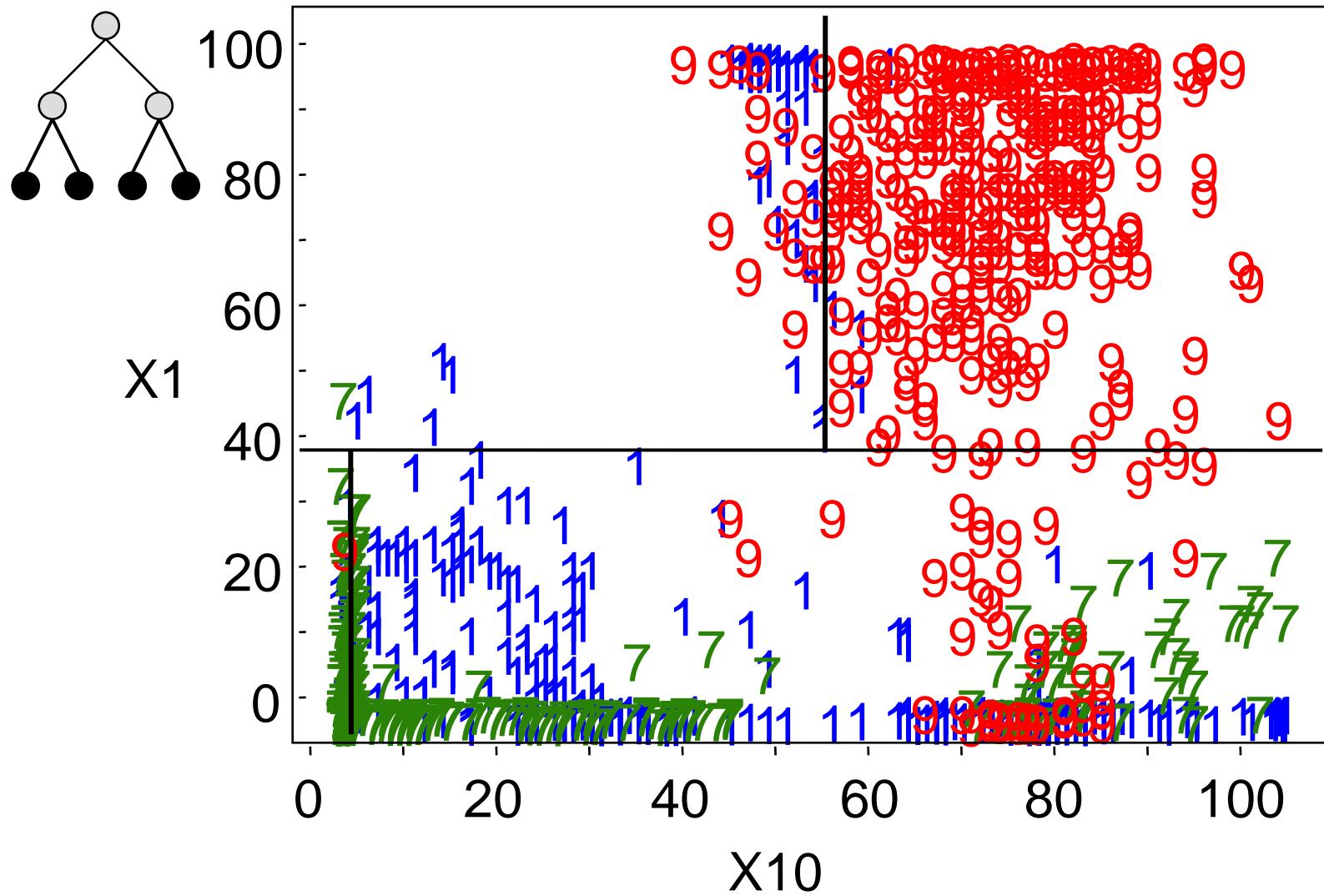
# Разбиение деревом глубины 1



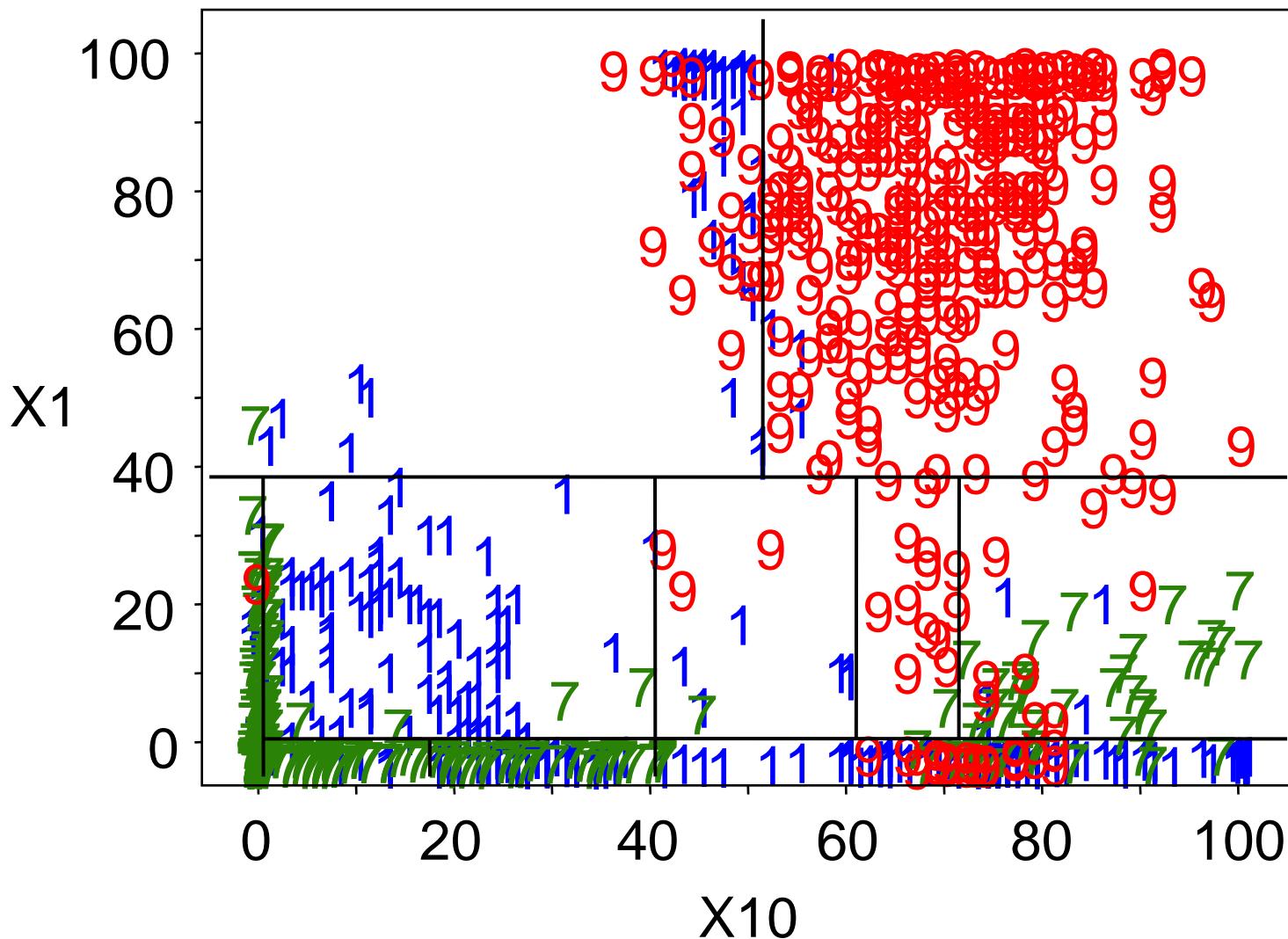
# Глубина 2



# Разбиение деревом глубины 2



# Финальное разбиение



# Рассмотрим

- Поиск разбиения по переменным

- Ординальным
  - Категориальным

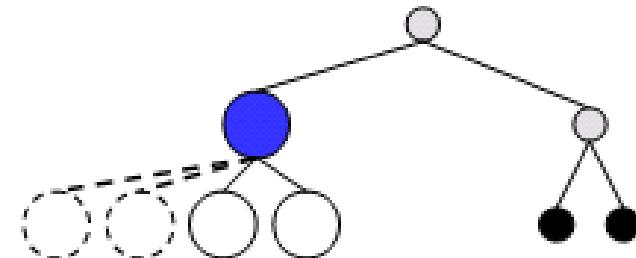
- Множественные разбиения

- Критерии разбиения

- Уменьшение разнородности
  - Хи<sup>2</sup> тест

- Регрессионные деревья

- Пропущенные значения



<u>Variable</u>	<u>Values</u>
X10	0.5
X10	1.8
X10	11, 46
X1	2.4
X1	1, 4, 61
:	:
:	:

# Разбиение по ординальной переменной

## Варианты разбиения

$$\begin{array}{l} 1-234 \\ 12-34 \\ 123-4 \end{array} \quad \binom{3}{1} = 3 \quad \binom{L-1}{B-1} = \frac{(L-1)!}{(B-1)! (L-B)!}$$

$$\begin{array}{l} 1-2-34 \\ 1-23-4 \\ 12-3-4 \end{array} \quad \binom{3}{2} = 3 \quad \sum_{l=2}^L \binom{L-1}{l-1} = 2^{L-1} - 1$$
  
$$1-2-3-4 \quad \binom{3}{3} = 1$$

# Ординальные переменные

X	.20	1.7	3.3	3.5	14	2515
$\ln(X)$	-1.6	.53	1.2	1.3	2.6	7.8
rank(X)	1	2	3	4	5	6

Потенциальные точки разбиения

# Разбиение категориальной переменной

1—234

2—134

3—124

4—123

12—34

13—24

14—23

1—2—34

1—3—24

1—4—23

2—3—14

2—4—13

3—4—12

1—2—3—4

$$S(L, B) = B \cdot S(L - 1, B) + S(L - 1, B - 1)$$

$B:$	2	3	4	total
2	1			1
3	3	1		4
4	7	6	1	14
5	15	25	10	51
6	31	90	65	202
7	63	301	350	876
8	127	966	1701	4139
9	255	3025	7770	21146

# Основные моменты поиска разбиения

## ■ Только бинарное разбиение

- ординальные =  $L - 1$
- категориальные =  $2^{L - 1} - 1$

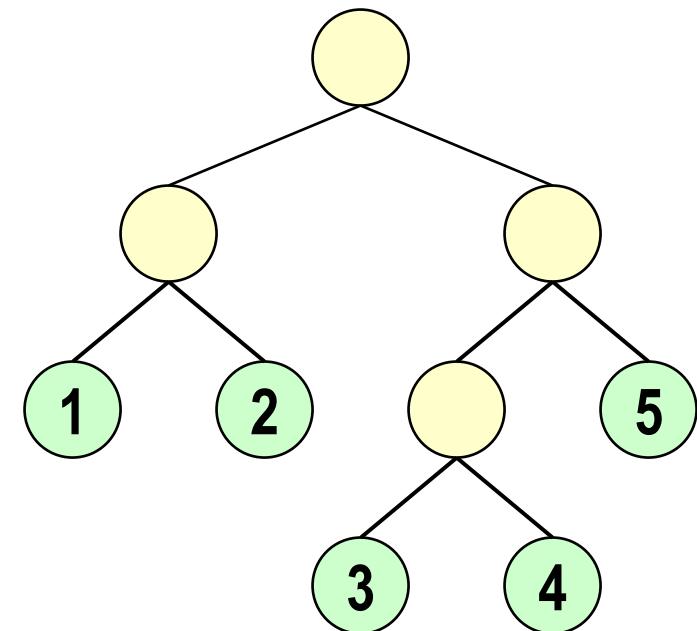
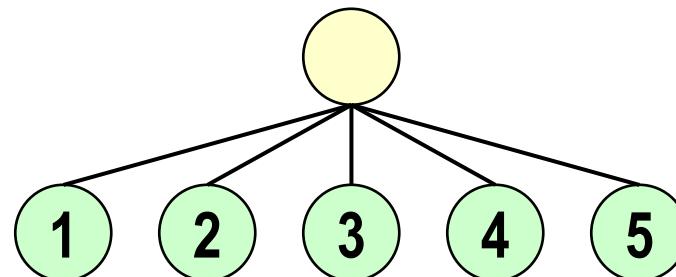
## ■ Агломеративная кластеризация

### Вариантов разбиения

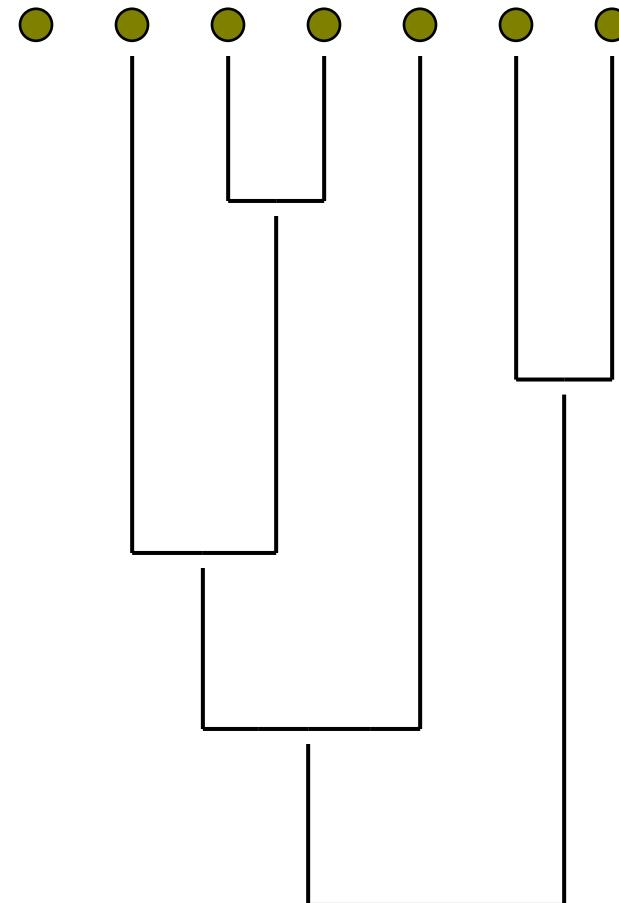
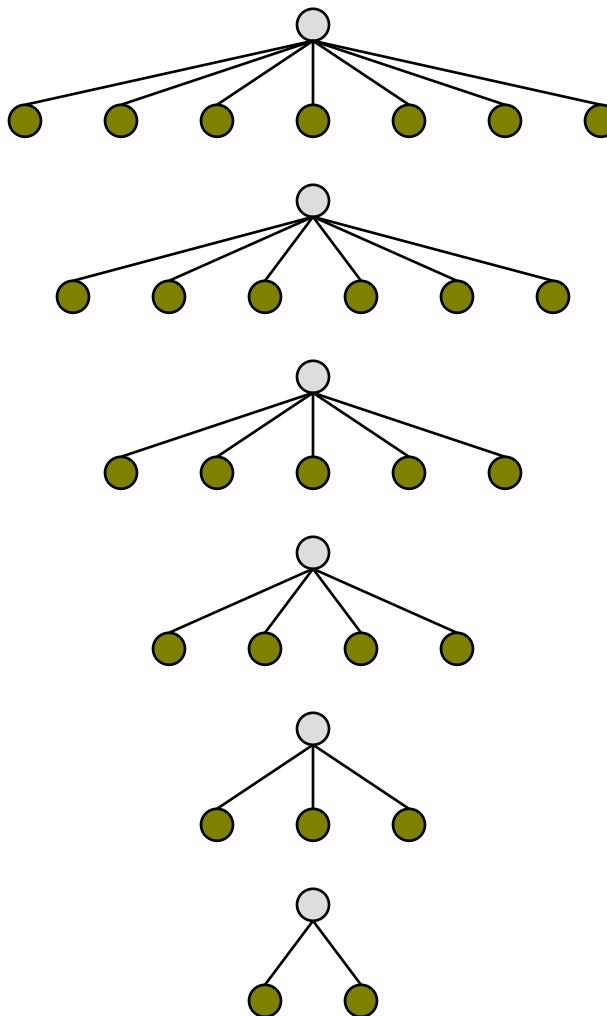
- Kass (1980)

## ■ Минимальный размер листа

## ■ Множественное или бинарное



# Кластеризация гипотез



# Критерии разбиения для категориального отклика

X1:<38.5     $\geq 38.5$

$\Delta Gini$     $\Delta \text{entropy}$    logworth

1	293	71
7	363	1
9	42	294

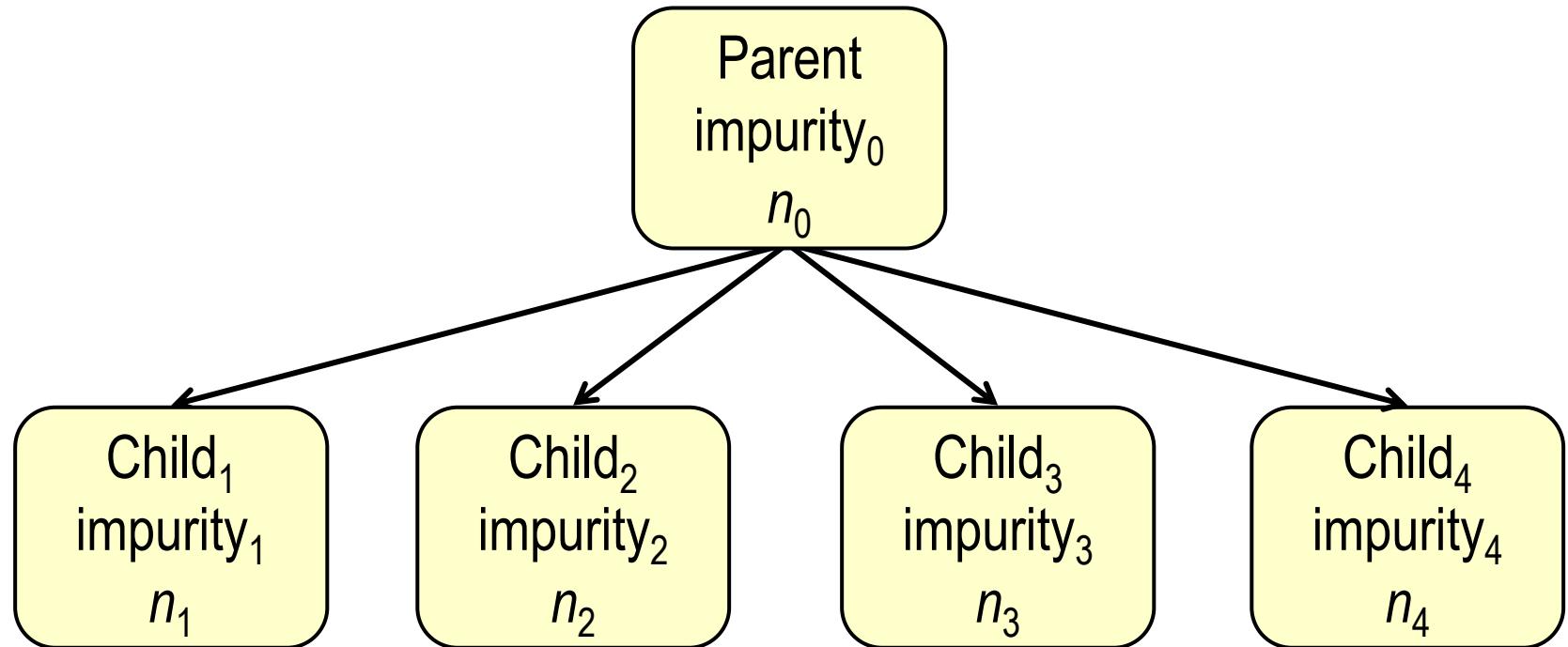
.197    .504    140

X10: <0.5    1-41    42-51     $\geq 51.5$

1	9	143	65	147
7	221	88	1	54
9	1	4	16	315

.255    .600    172

# Уменьшение разнородности

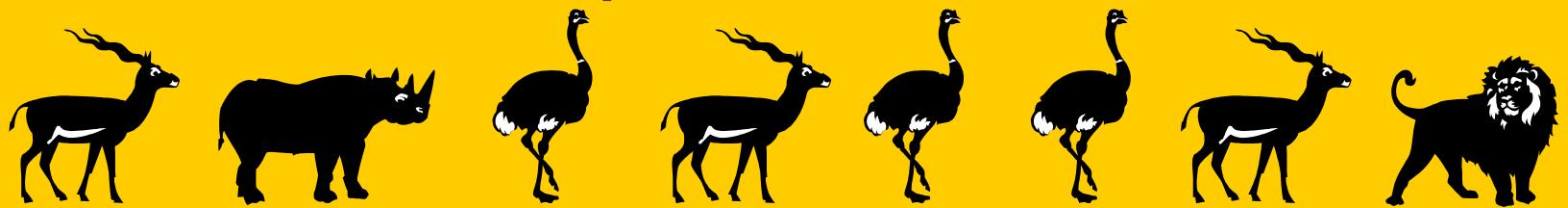


$$\Delta i = i(0) - \left( \frac{n_1}{n_0} i(1) + \frac{n_2}{n_0} i(2) + \frac{n_3}{n_0} i(3) + \frac{n_4}{n_0} i(4) \right)$$

# Индекс Джини

$$1 - \sum_{j=1}^r p_j^2 = 2 \sum_{j < k} p_j p_k$$

Высокая вариация, низкая чистота



$$\Pr(\text{interspecific encounter}) = 1 - 2(3/8)^2 - 2(1/8)^2 = .69$$

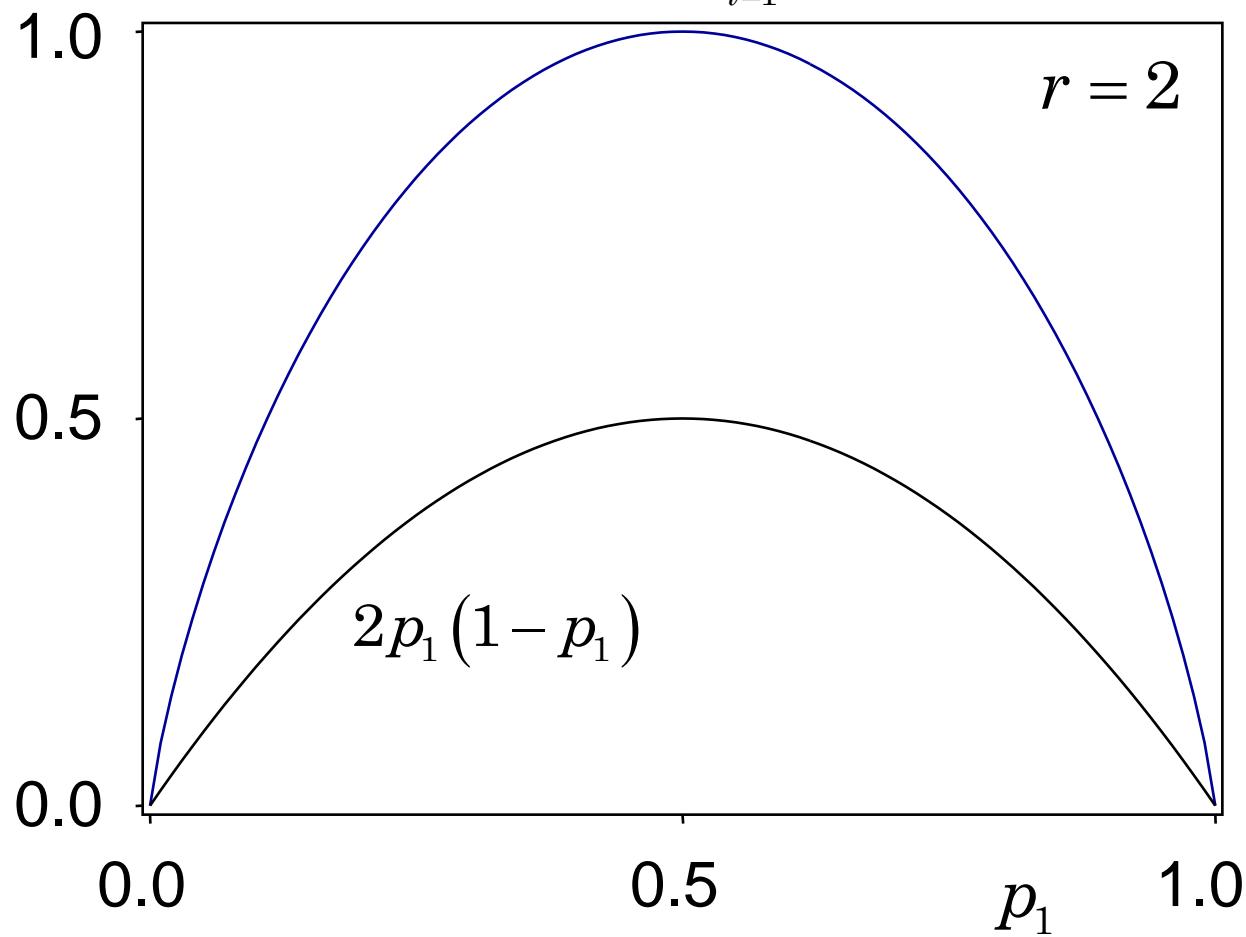
Низкая вариация, высокая чистота



$$\Pr(\text{interspecific encounter}) = 1 - (6/7)^2 - (1/7)^2 = .24$$

# Энтропия

$$H(p_1, p_2, \dots, p_r) = -\sum_{i=1}^r p_i \log_2(p_i)$$



# Хи<sup>2</sup> тест

Наблюдаемые  
частоты

X1: <38.5 ≥38.5

1	293	71	.342
7	363	1	.342
9	42	294	.316
	.656	.344	$n=1064$

Ожидаемые  
(по гипотезе)

239	125
239	125
225	116

$$\frac{(O - E)^2}{E}$$

12	23
64	123
149	273

$$\chi^2_{\nu} = \sum \frac{(O - E)^2}{E} = 644$$

$$\nu = (3-1)(2-1) = 2$$

# Корректировка p-Value

X1: 38.5

1	293	71
7	363	1
9	42	294

$\chi^2_\nu$	$\nu$	$-\log_{10}(P)$	$m$	$-\log_{10}(mP)$
644	2	140	96	138

X1: 17.5 36.5

1	249	42	73
7	338	25	1
9	26	16	294

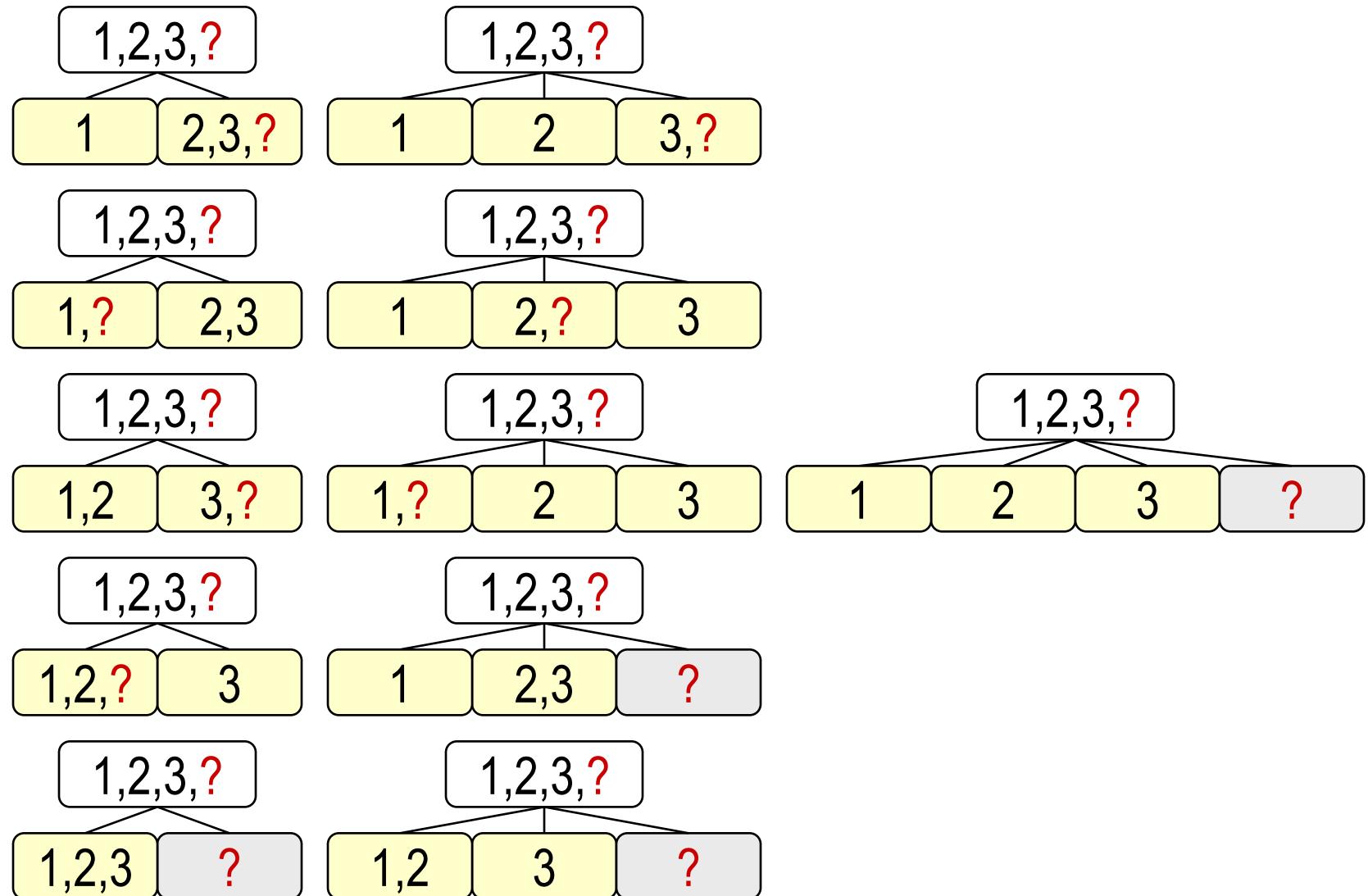
660	4	141	4560	137
-----	---	-----	------	-----

X10: 0.5 41.5 51.5

1	9	143	65	147
7	221	88	1	54
9	1	4	16	315

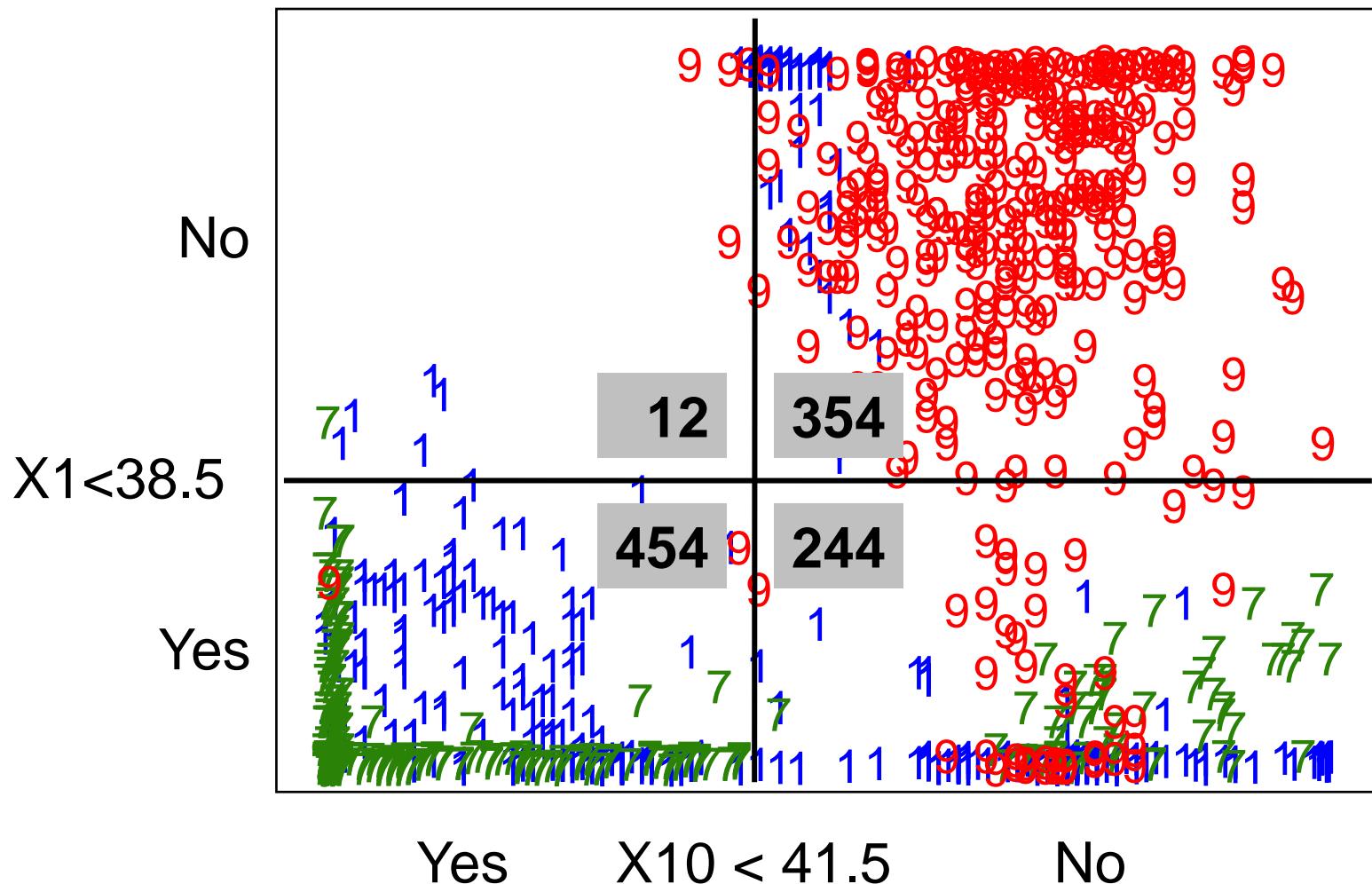
814	6	172	156849	167
-----	---	-----	--------	-----

# Пропущенные значения

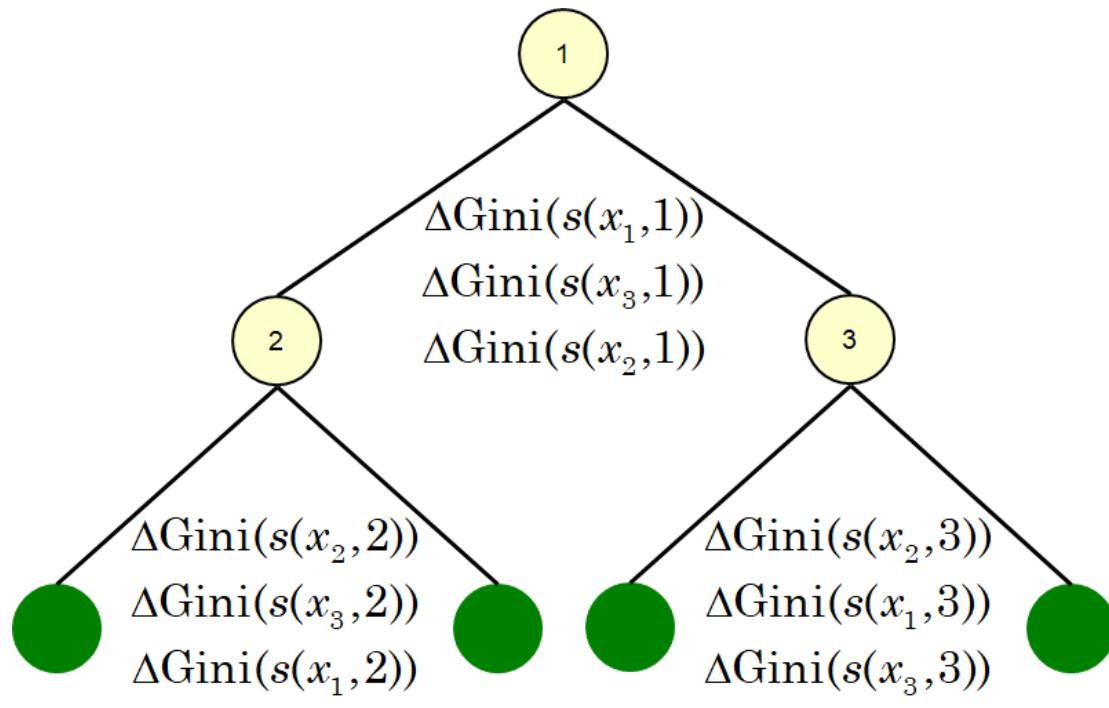


# Суррогатные разбиения

Уровень согласия=76%

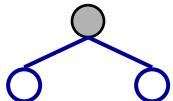
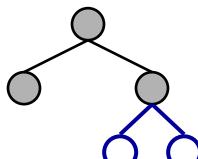
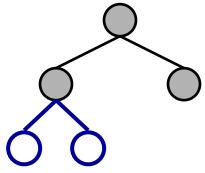
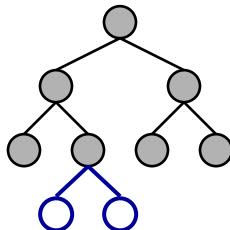


# Важность переменных



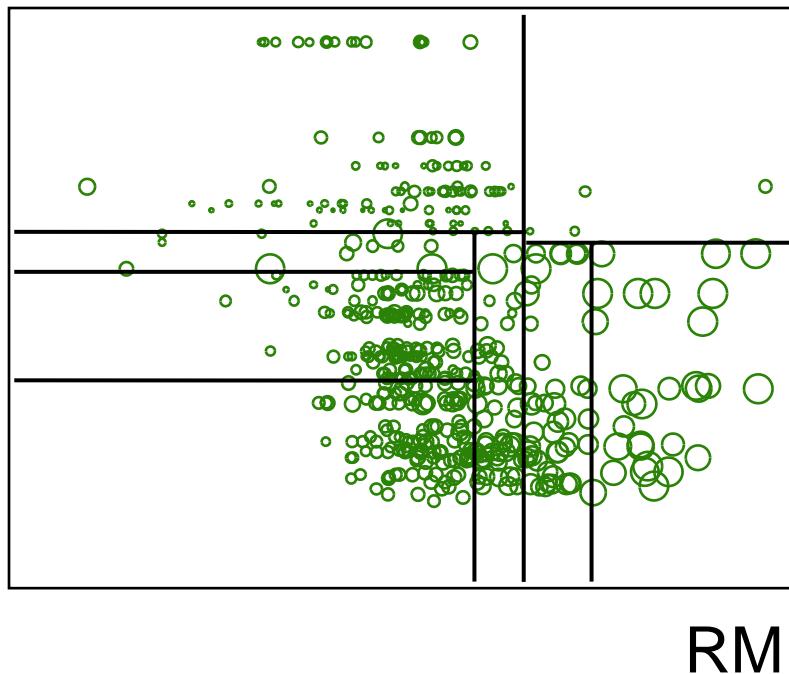
- Варианты:
  - Нормированный прирост gini или энтропии для категориального отклика или вариации (дисперсии) для числового по всем вхождениям переменной
  - Число вхождений переменной в правила дерева

# Рост дерева

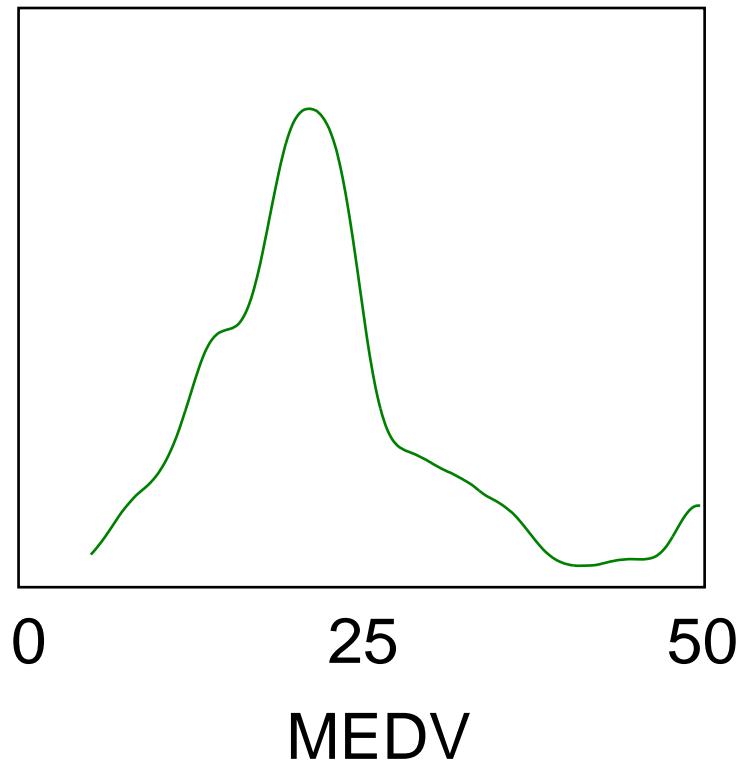
$-\log_{10}(P)$	$m$	$-\log_{10}(mP)$	$d$	$-\log_{10}(2^d mP)$	
	26.7	53	24.9	0	24.9
	3.12	14	1.97	1	1.67
	1.63	39	.039	1	-.26
	2.40	11	1.36	2	.76

# Непрерывный отклик

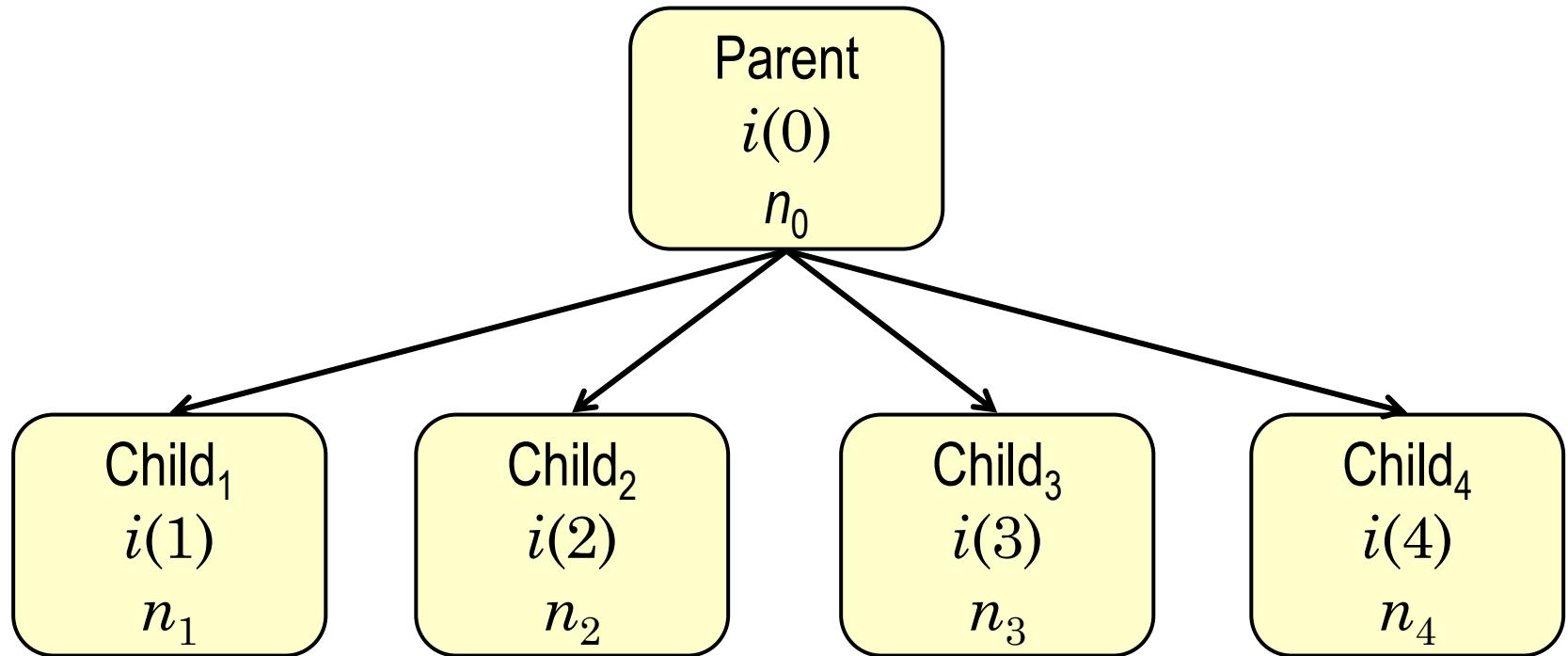
NOX



Плотность

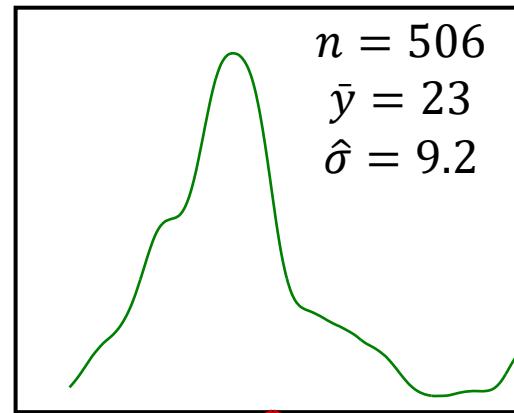
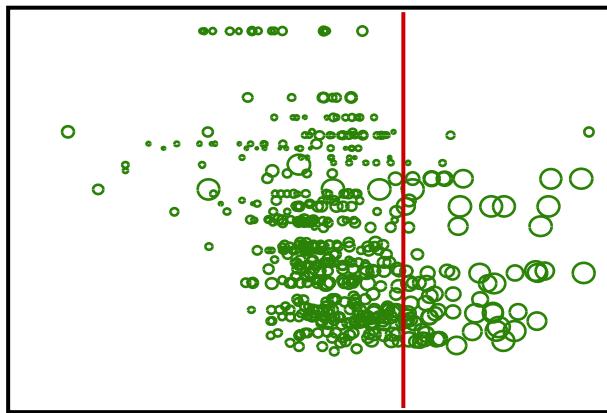


# Уменьшение вариации



$$\Delta i = i(0) - \left( \frac{n_1}{n_0} i(1) + \frac{n_2}{n_0} i(2) + \frac{n_3}{n_0} i(3) + \frac{n_4}{n_0} i(4) \right)$$

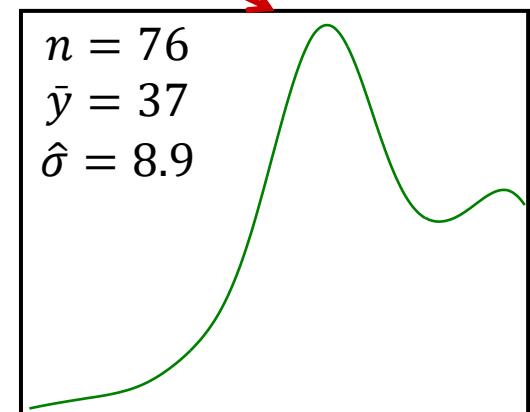
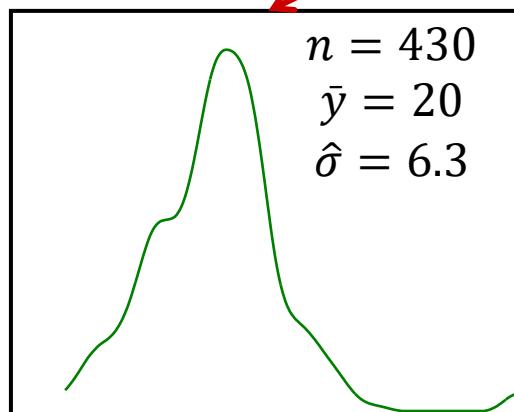
# Уменьшение вариации



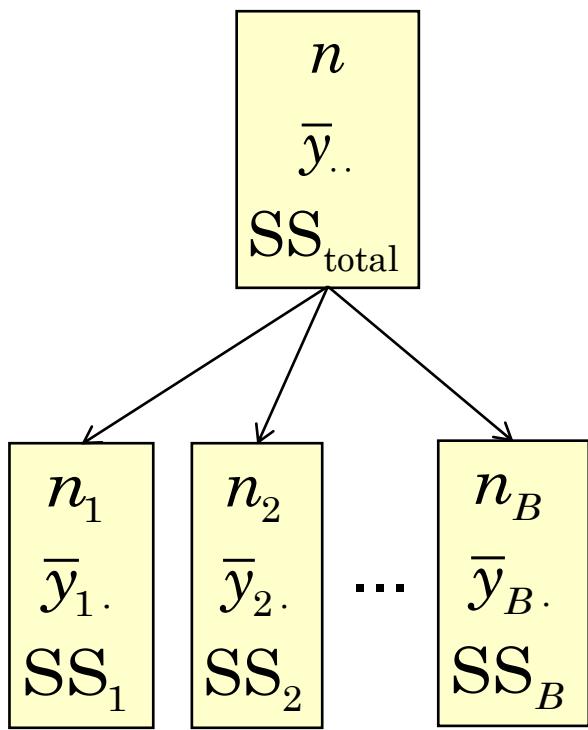
yes

$RM < 6.94$

no



# One-Way ANOVA



$$F = \left( \frac{\text{SS}_{\text{between}}}{\text{SS}_{\text{within}}} \right) \left( \frac{n - B}{B - 1} \right) \sim F_{B-1, n-B}$$

$$\text{SS}_{\text{between}} = \sum_{i=1}^B n_i (\bar{y}_{i.} - \bar{y}..)^2$$

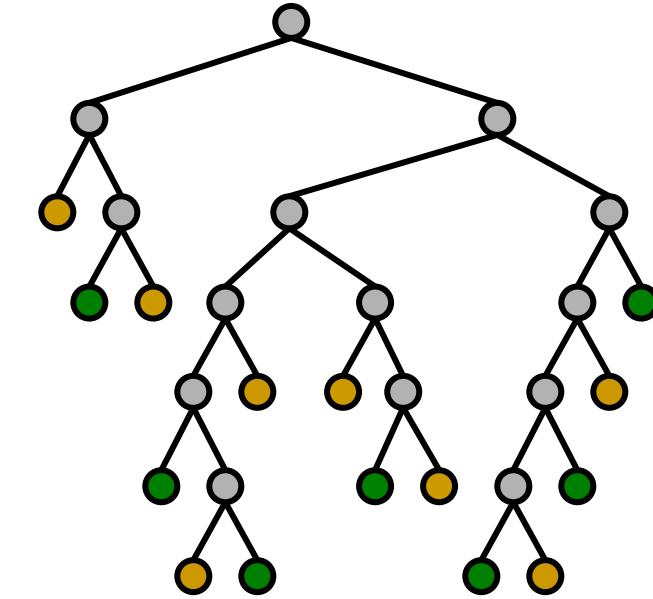
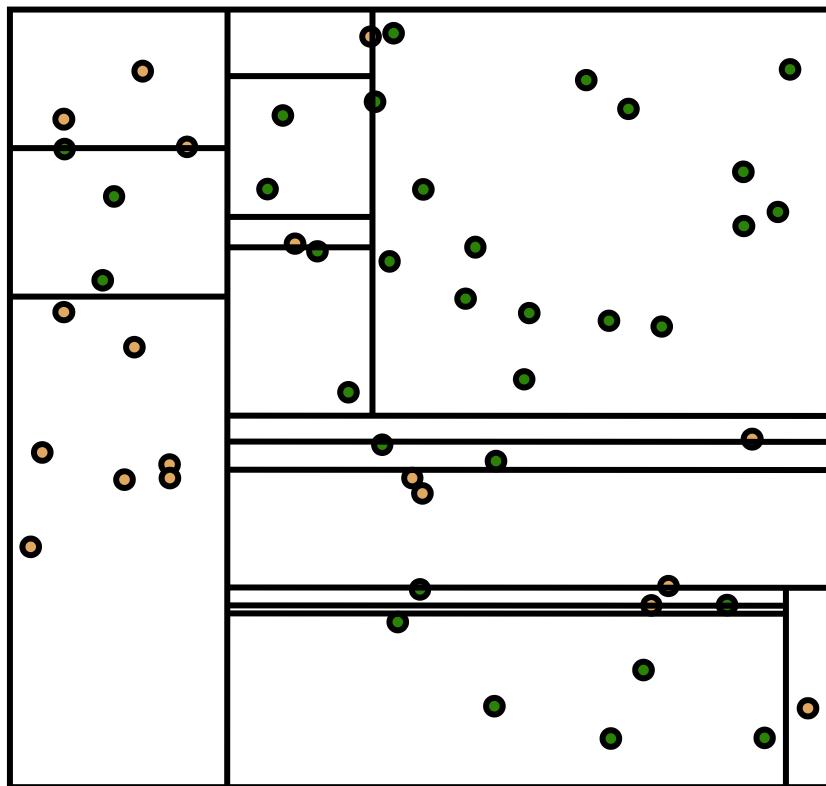
$$\text{SS}_{\text{within}} = \sum_{i=1}^B \text{SS}_i = \sum_{i=1}^B \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_{i.})^2$$

$$\text{SS}_{\text{total}} = \sum_{i=1}^B \sum_{j=1}^{n_i} (y_{ij} - \bar{y}..)^2$$

# Переобучение деревьев решений

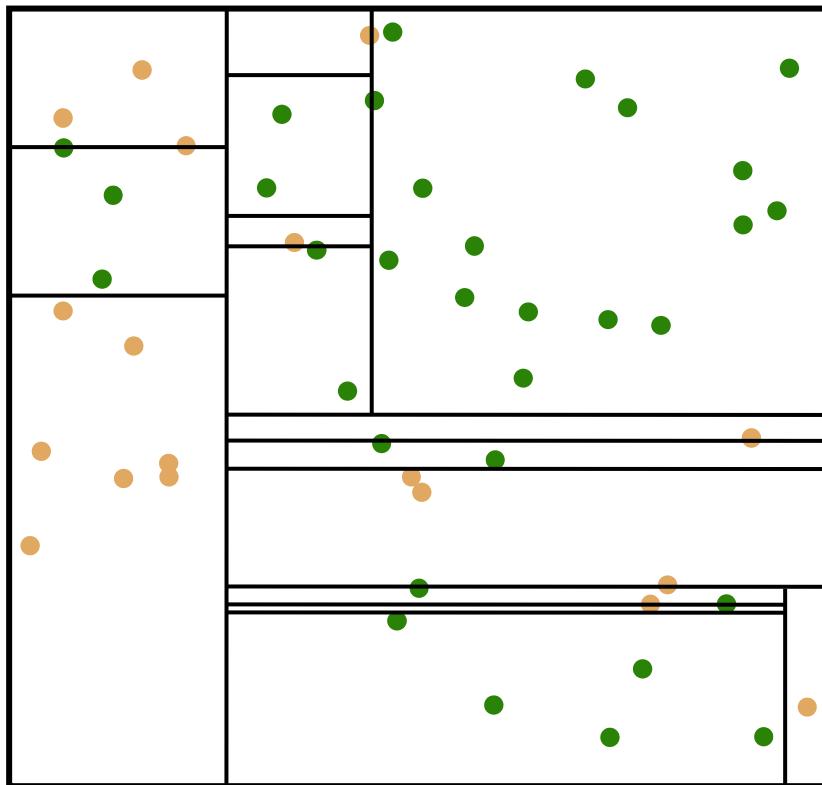
- Описанный выше процесс может давать хорошие прогнозы для обучающего набора, но может привести к *переобучению*, что приведет к ухудшению качества на тестовом наборе.
- Меньшее по размеру дерево с меньшим количеством разбиений (то есть меньшим количеством областей  $R_1, \dots, R_J$ ) может привести к снижению дисперсии и лучшей интерпретируемости.
- Одной из возможных альтернатив является ранняя остановка построения дерева, определяемая параметрами:
  - Максимальная глубина дерева
  - Минимальное число наблюдений в листе
  - Порог на p-value или другой критерий разбиения

# Максимальное дерево

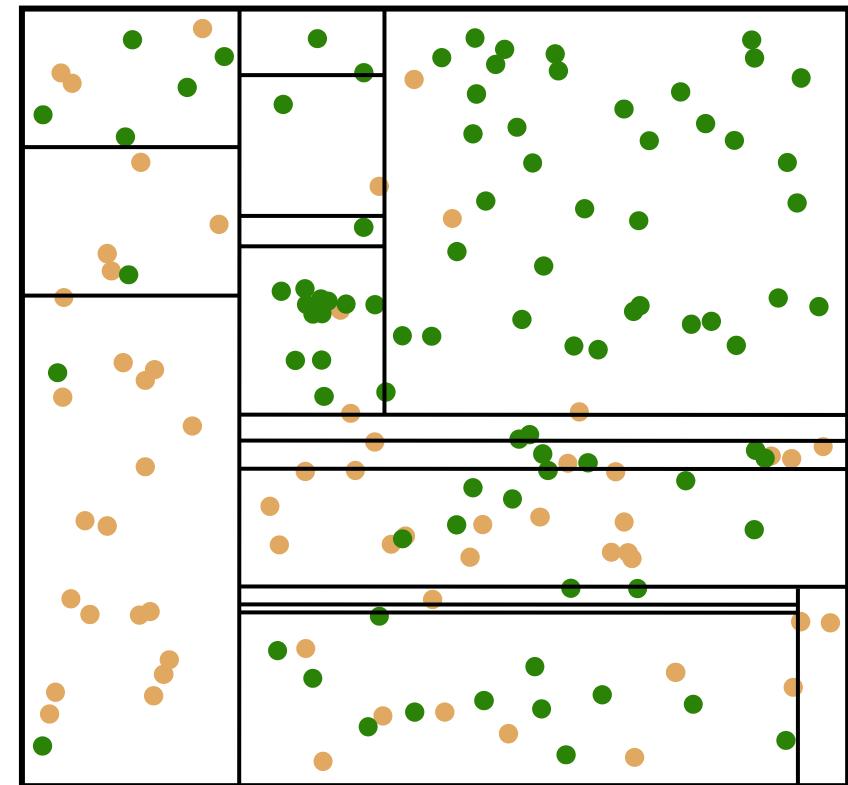


# Переобучение

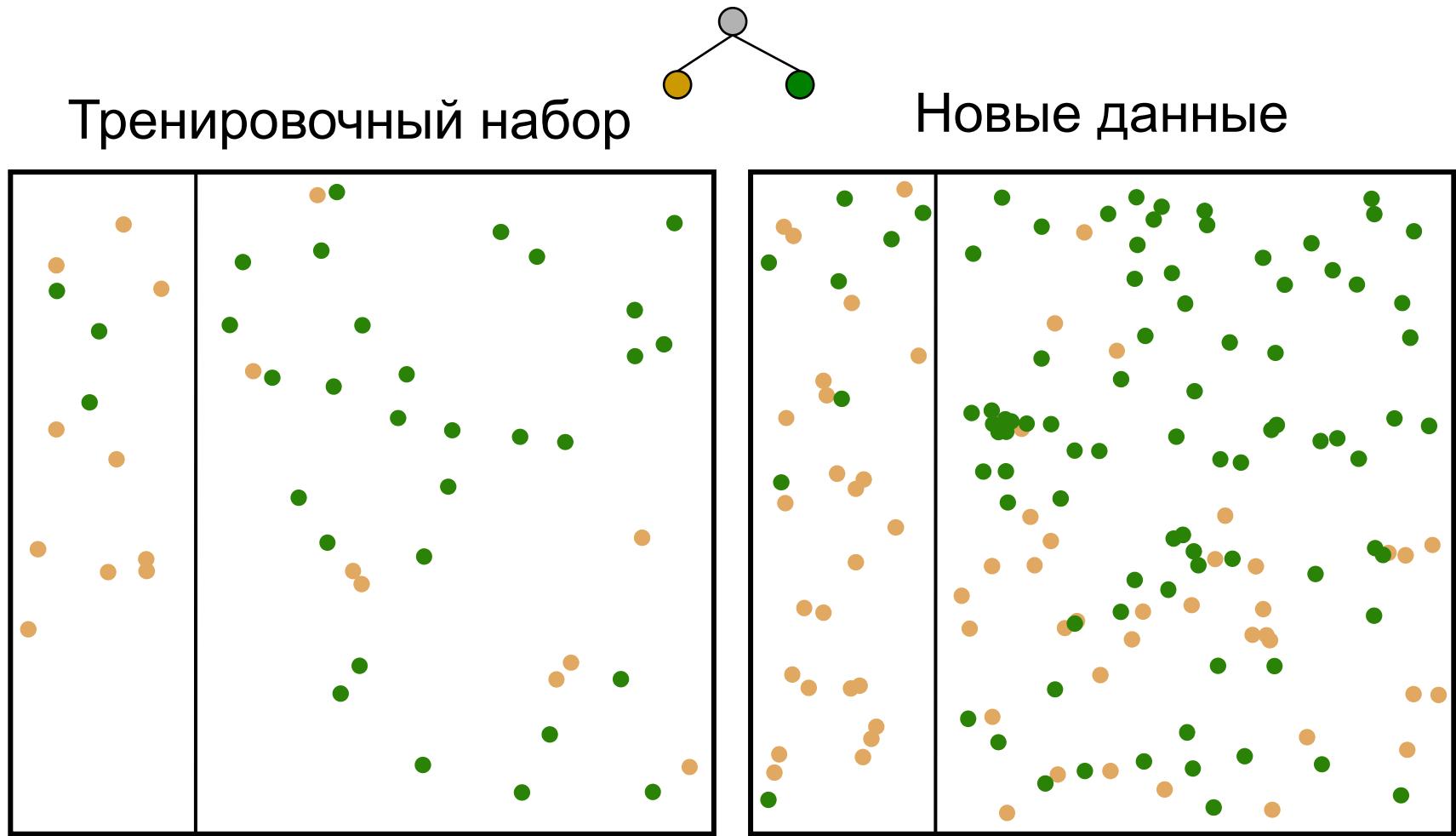
Тренировочный набор



Новые данные

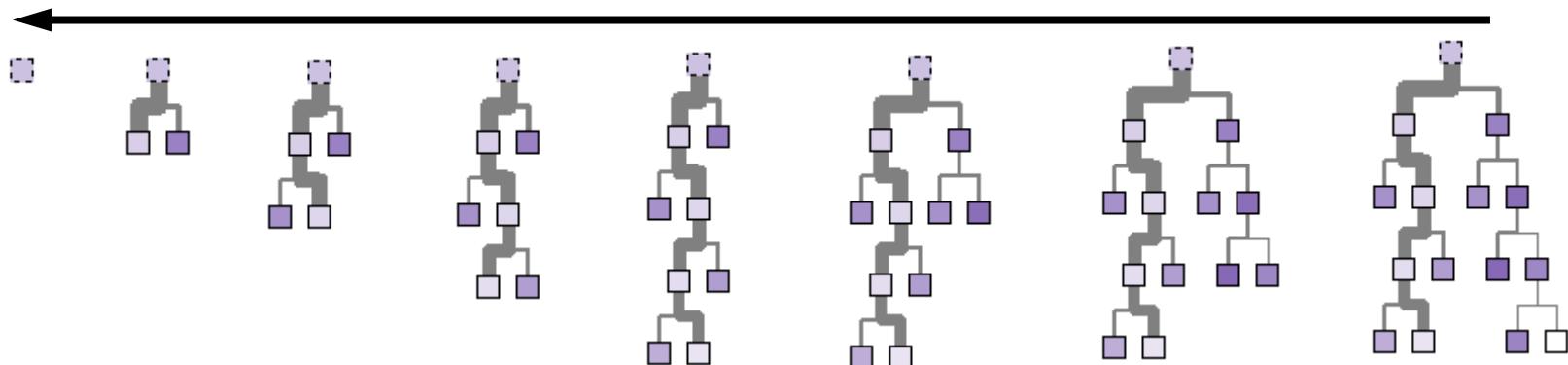


# Недообучение



# Обрубание дерева

- Лучшей стратегией является построение большого дерева  $T_0$ , а затем выполнение *отсечения* для получения поддерева
- Для этого используется подход *сокращения сложности*, также известный как *удаление самых слабых связей*
- Используем валидационный набор или кросс валидацию для выбора оптимального размера дерева.
- Алгоритм: строим максимальное дерево и последовательно обрубаем ветки с отбором



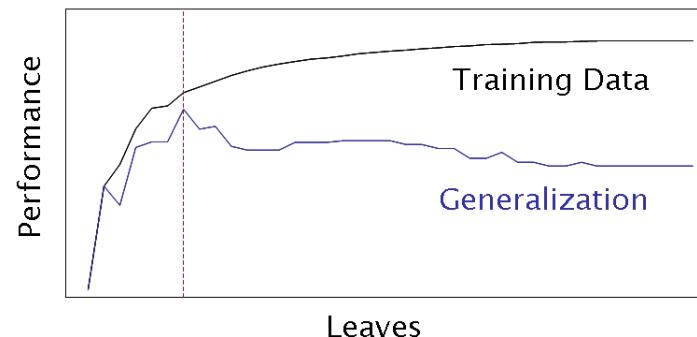
# Обрубание ветвей

Варианты:

1. Валидационный подход: выбираем лучшее по выбранной метрике на валидационном наборе
2. Cost-Complexity: оцениваем гиперпараметр  $\alpha$  компромисса между сложностью (например, число листьев) и точностью (например, среднеквадратичная ошибка)  $CC(T)=ERR(T)+\alpha|T|$
3. C4.5 – при заданном гиперпараметре  $\alpha$  оцениваем вероятность ошибки  $p_l$  для каждого узла  $l$ , решая относительно  $p_l$ , считаем

$$E_i = \sum_{l \in T_i} N_l p_l \quad \alpha = 1 - \frac{\Gamma(N_l + 1)}{\Gamma(F_l + 1) \Gamma(N_l - F_l)} \int_0^{p_l} \nu^{F_l} (1 - \nu)^{N_l - F_l + 1} d\nu$$

для обрубания выбираем пару листьев для склейки с наименьшим увеличением ошибки



# Пример дерева

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.inspection import DecisionBoundaryDisplay
```

```
iris = load_iris()
```

```
X, y = iris.data, iris.target
```

```
X = X[:, :2]
```

```
X.shape, y.shape, iris.target_names
```

```
((150, 2), (150,), array(['setosa', 'versicolor', 'virginica'], dtype='|<U10'))
```

```
tree = DecisionTreeClassifier(criterion="gini", max_depth=5, min_samples_split=5, min_samples_leaf=3,  
                               ccp_alpha=0.0) # pruning parameter
```

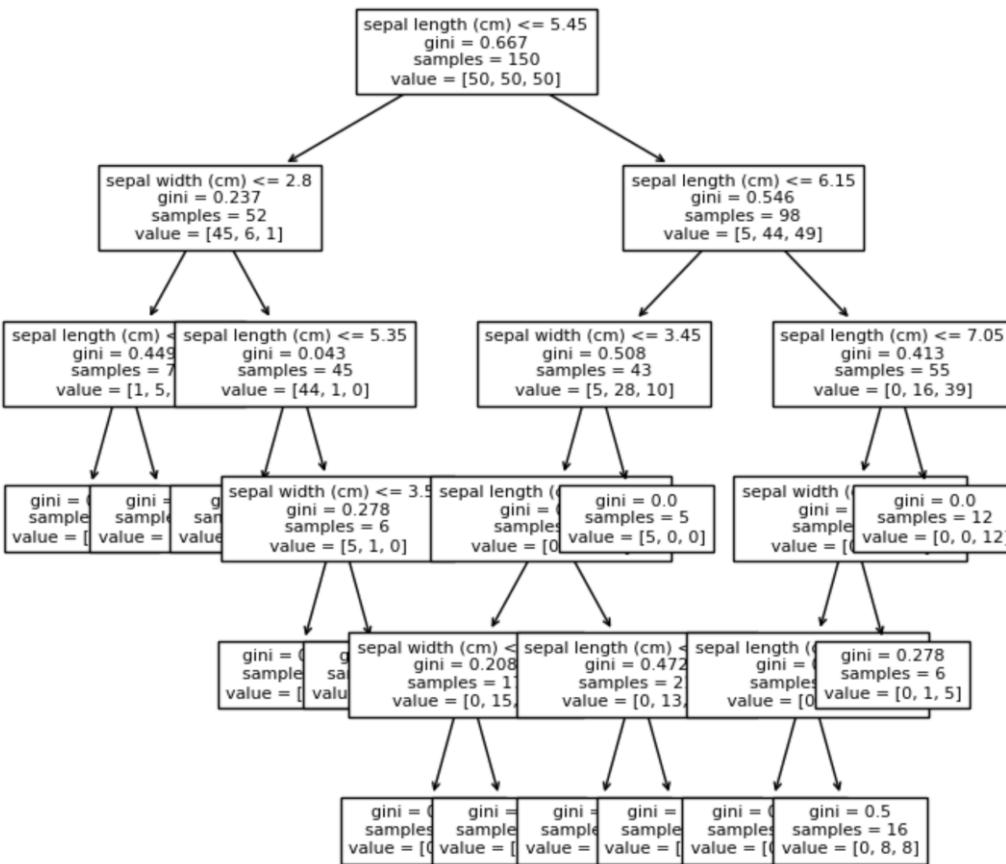
```
tree.fit(X, y)
```

```
DecisionTreeClassifier
```

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=3, min_samples_split=5)
```

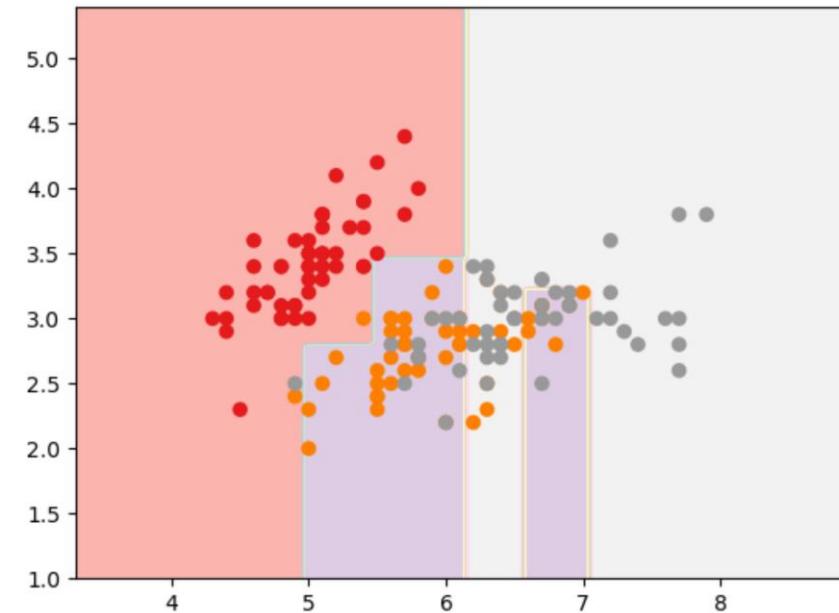
# Пример дерева

```
plot_tree(tree, fontsize=8, feature_names=iris.feature_names)  
plt.gcf().set_size_inches(8, 8)
```



```
DecisionBoundaryDisplay.from_estimator(tree, X, cmap="Pastel1")  
plt.scatter(*X.T, c=y, cmap="Set1")
```

```
<matplotlib.collections.PathCollection at 0x7fa63f56b1c0>
```



```
tree.feature_importances_
```

```
array([0.76047482, 0.23952518])
```

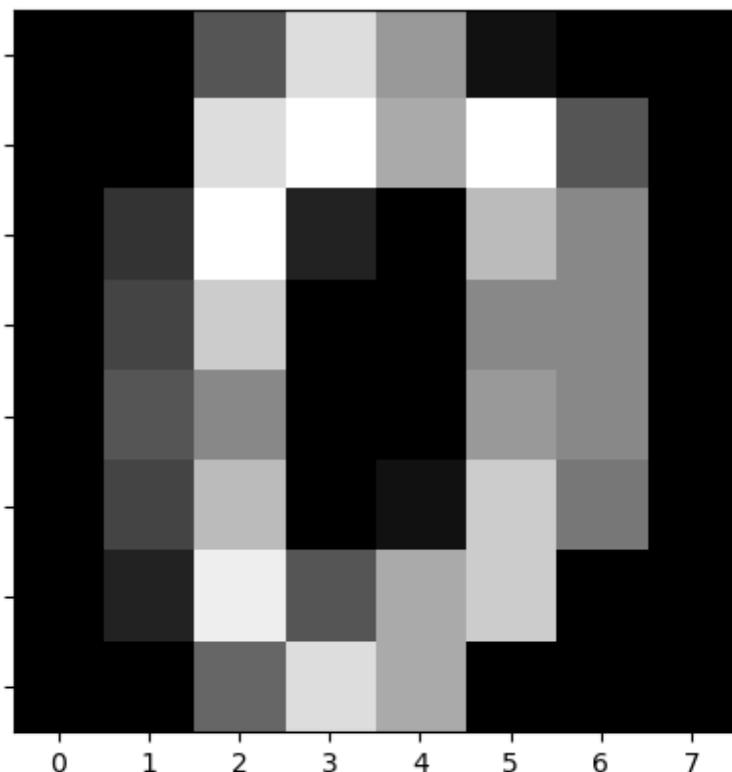
# Сложность дерева

```
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_digits
```

```
digits = load_digits()
X, y = digits.data, digits.target
X.shape, np.unique(y)

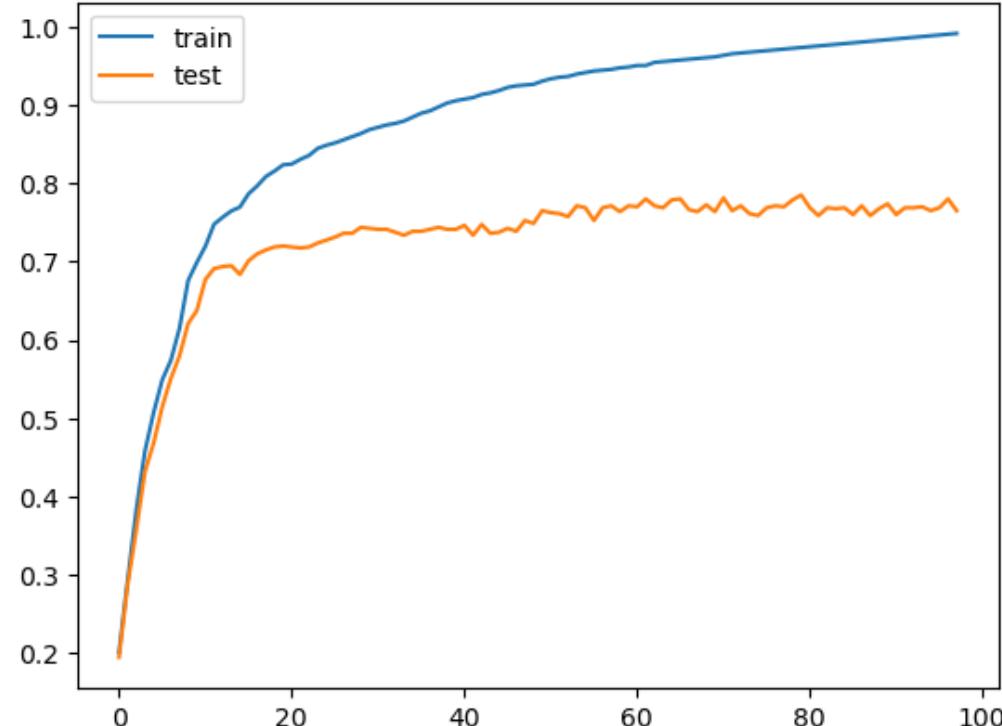
((1797, 64), array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

```
plt.imshow(X[0].reshape(8, 8), cmap="gray");
```



```
N = 1000 # Labels are scattered evenly enough
train_X, train_y = X[:N], y[:N]
test_X, test_y = X[N:], y[N:]
```

```
result = []
for max_leaf_nodes in range(2, 100):
    tree = DecisionTreeClassifier(max_leaf_nodes=max_leaf_nodes)
    tree.fit(train_X, train_y)
    score = {"train":accuracy_score(train_y, tree.predict(train_X)),
             "test":accuracy_score(test_y, tree.predict(test_X))}
    result.append(score)
pd.DataFrame(data=result).plot();
```



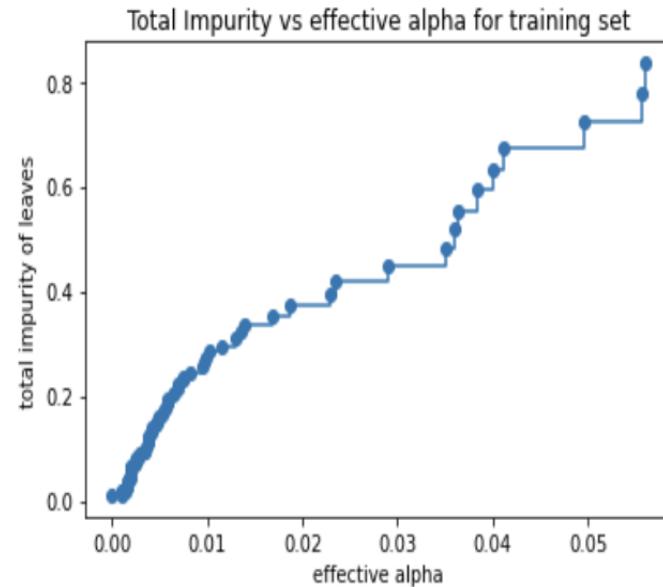
# Обрубание дерева через cost-complexity

- Компромисс между сложностью и точностью:  $CC(T) = R(T) + \alpha|T|$
- Выбор узла для обрубания по минимуму «эффективного»  $\alpha$ :

$$\alpha_{eff}(t) = \frac{\Delta R(t)}{|T_t| - 1}$$

- Пока все  $\alpha_{eff}$  меньше заданного порога
- Можно построить «трассу» зависимости качества от  $\alpha_{eff}$  в процессе обрубания ветвей

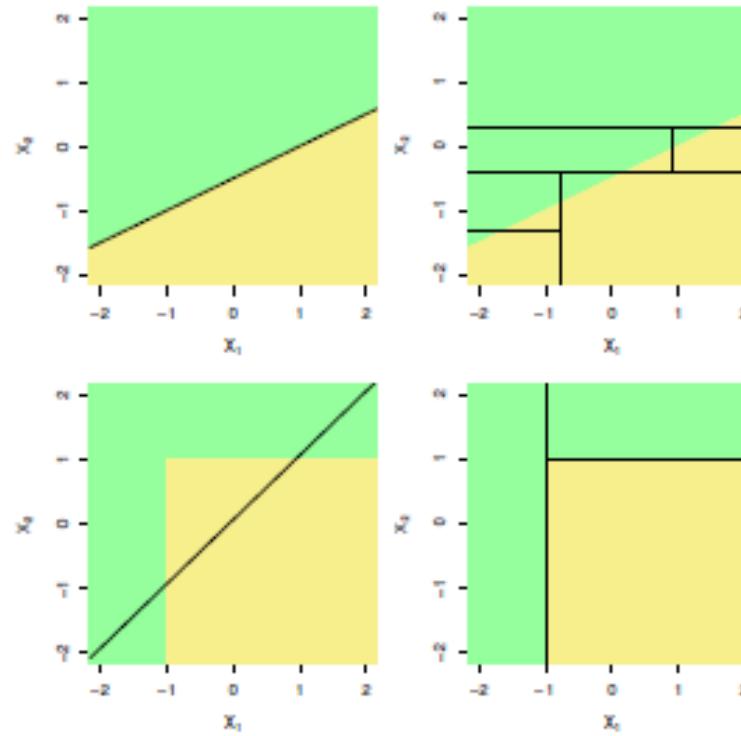
```
path = tree.cost_complexity_pruning_path(train_X, train_y)
ccp_alphas, impurities = path ccp_alphas, path impurities
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o", drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")  
Text(0.5, 1.0, 'Total Impurity vs effective alpha for training set')
```



# Особенности популярных алгоритмов построения деревьев решений

Свойства	CHAID (Kass)	CART (Breiman)	C5 (Quinlan)
Критерий для числ. отклика	Фишер	Вариация	нет
Критерий для симв. отклика	Хи-квадрат	Джини	Энтропия
Работа с пропусками	Отдельная ветвь	Подстановка	Пропорция по ветвям (игнор)
Особенности	Корректировка Бонферрони и глубина	Линейные комбинации при разбиении	Алгоритм «сокращения» правил
Обрубание вервей	НЕТ	По точности	По точности

# Деревья решений vs линейные модели



Верхний ряд: истинная линейная граница; Нижний ряд: истинная нелинейная граница.

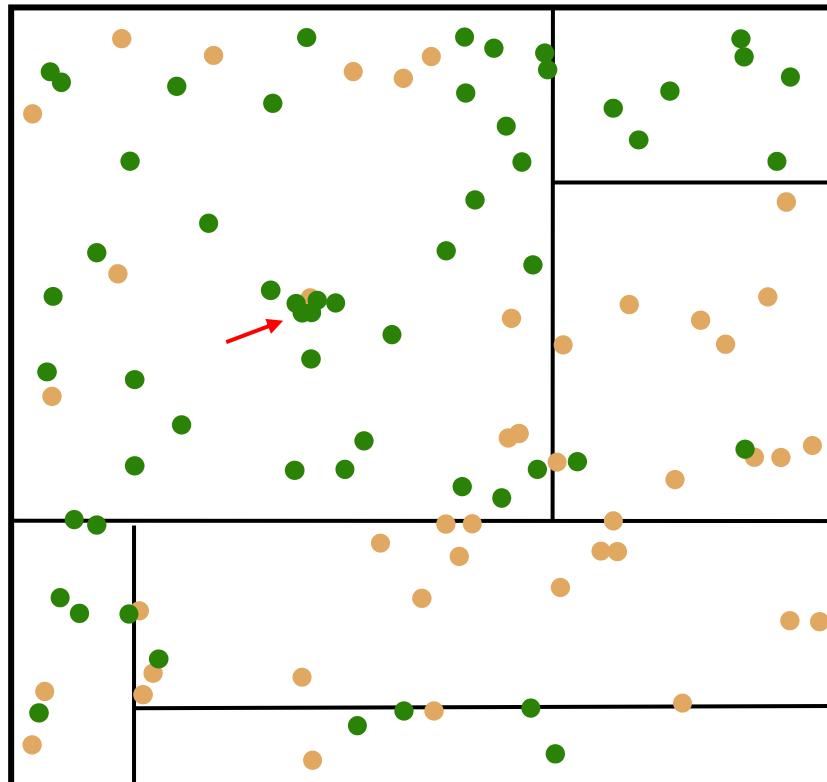
Левый столбец: линейная модель; Правый столбец: модель на основе деревьев решений

# Преимущества и недостатки деревьев решений

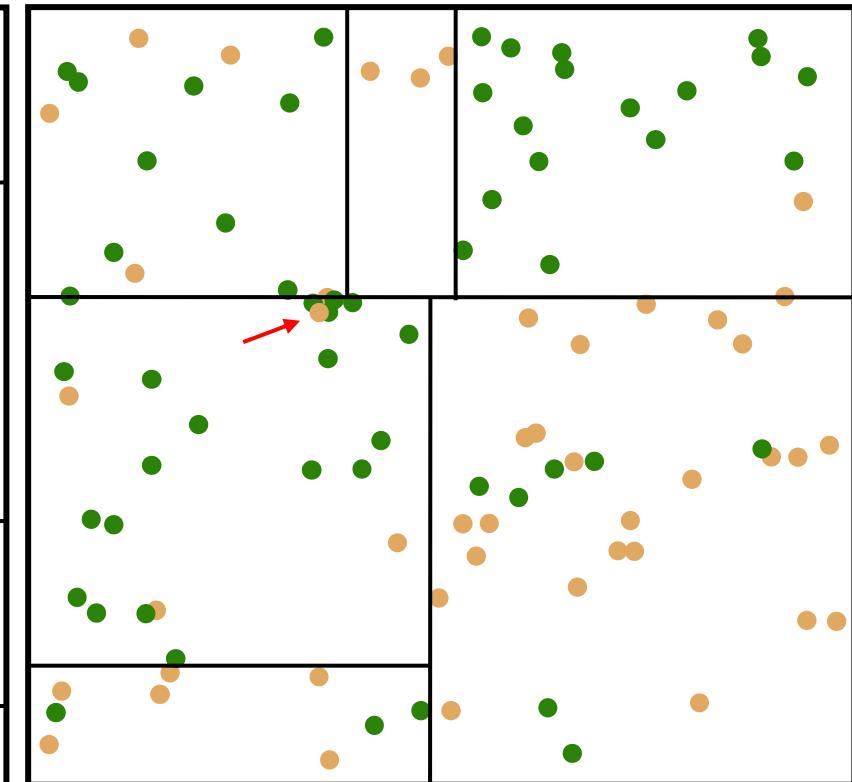
- Деревья имеют очень понятную интерпретацию. Даже проще, чем линейная регрессия!
- Существует мнение, что деревья решений отражают процесс принятия решений людьми лучше, чем подходы для решения задач регрессии и классификации, рассмотренные в предыдущих главах.
- Деревья можно наглядно отобразить графически и легко интерпретировать даже не специалистам (особенно, для небольших деревьев).
- Деревья могут легко обрабатывать качественные переменные и пропуски без необходимости создания фиктивных переменных.
- К сожалению, деревья обычно не дают такую же точность прогнозирования, как некоторые другие подходы для решения задач регрессии и классификации.

# Нестабильность модели

Один новый пример



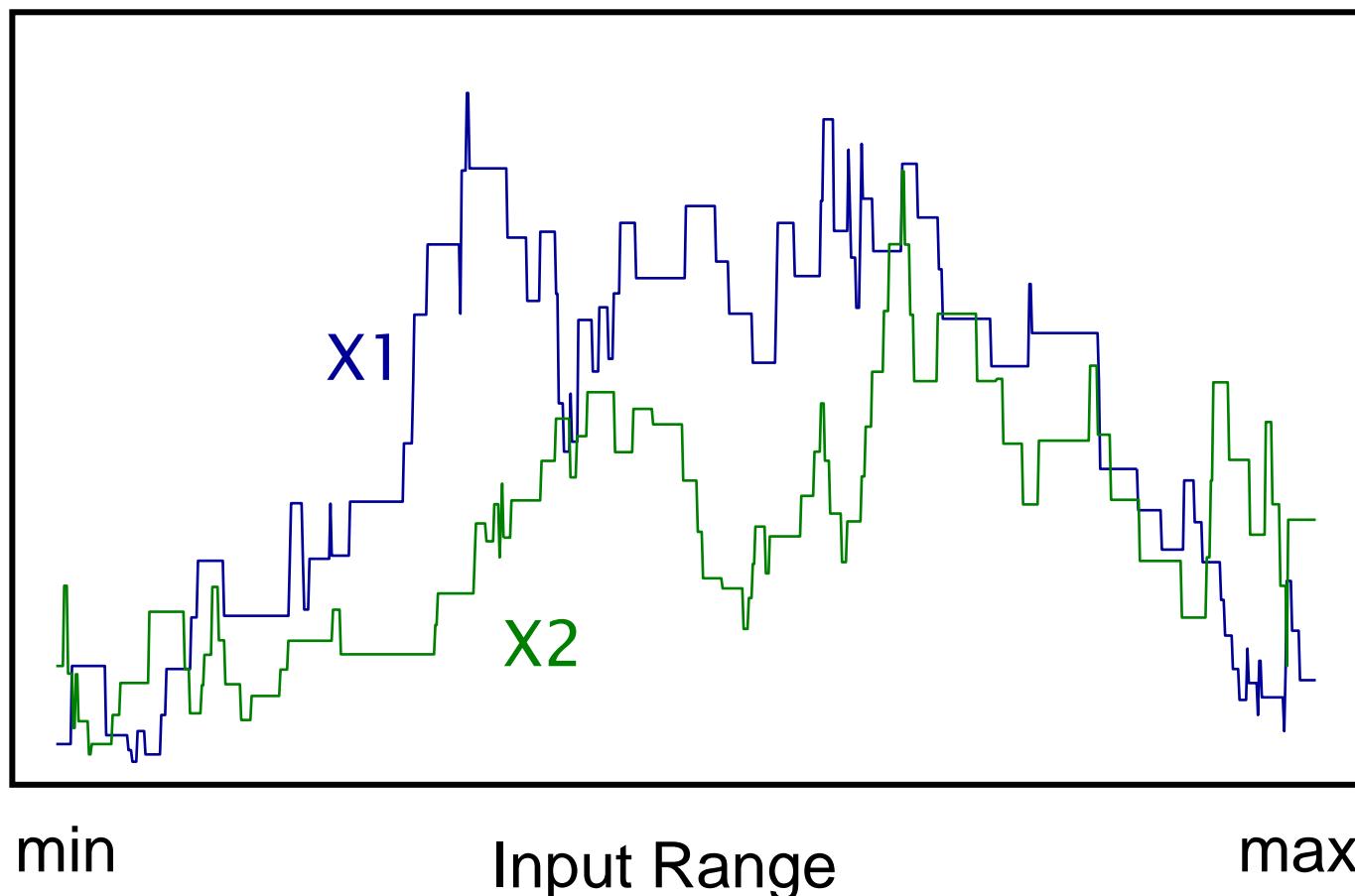
Точность = 81%



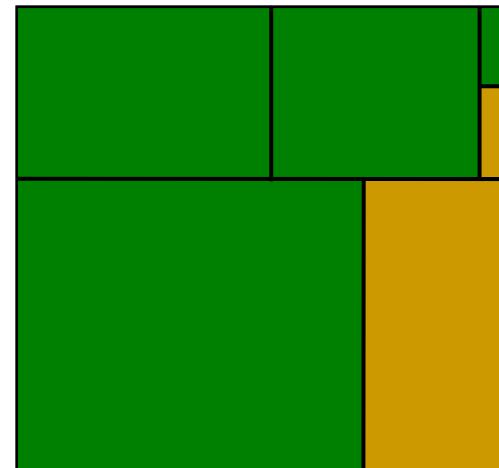
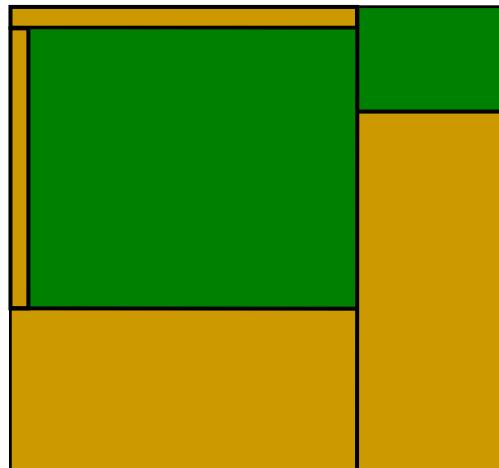
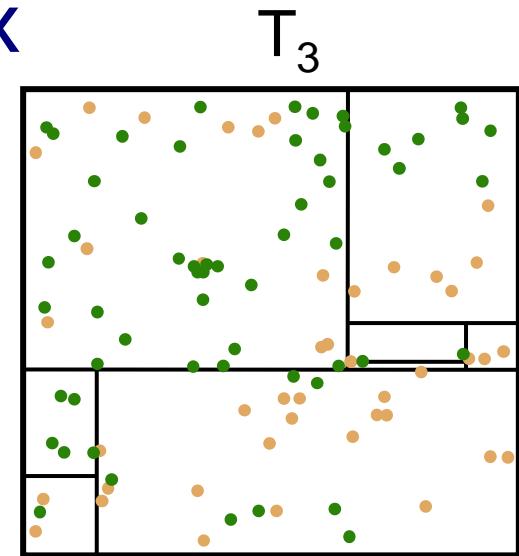
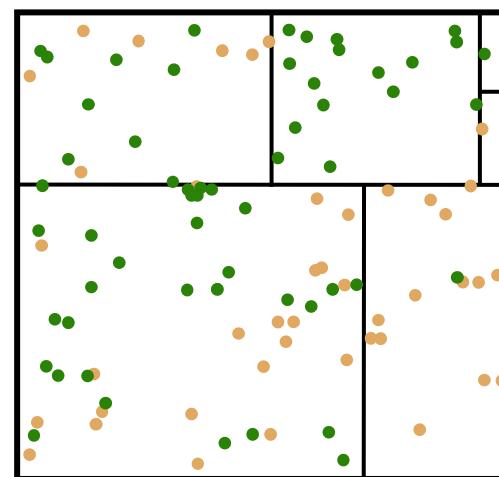
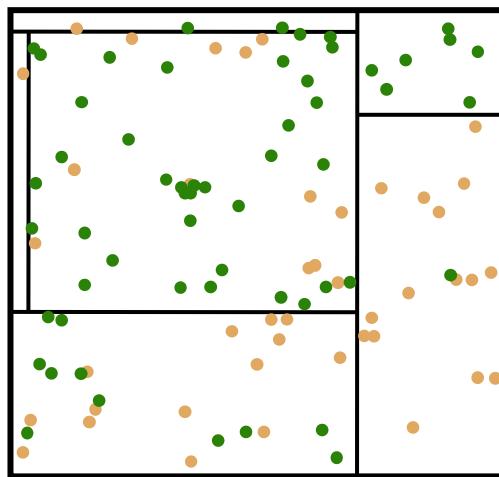
Точность = 80%

# Альтернативные точки разбиения

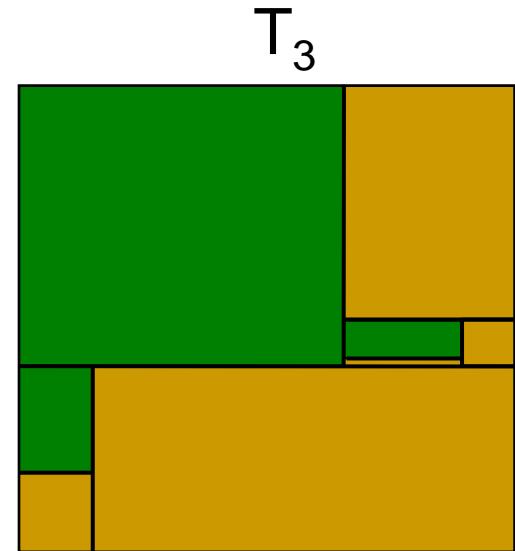
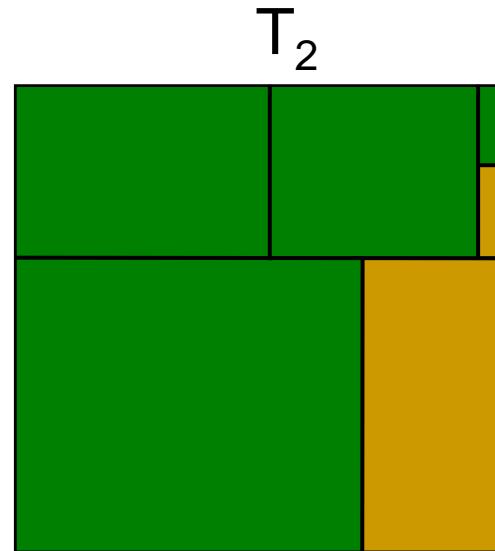
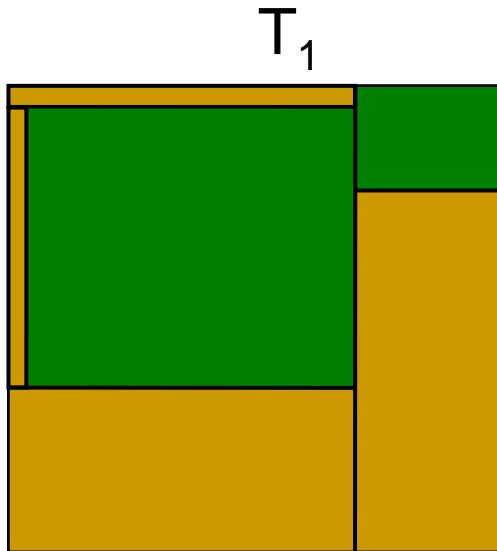
Logworth



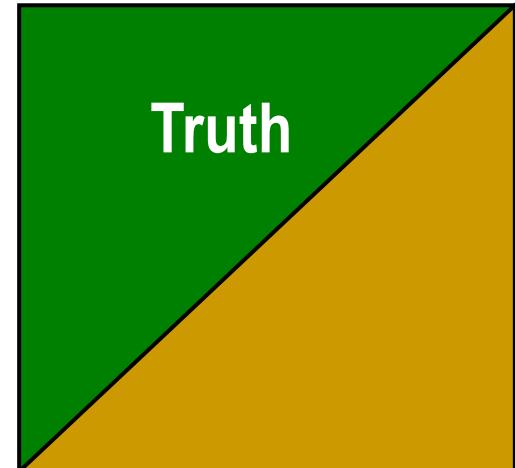
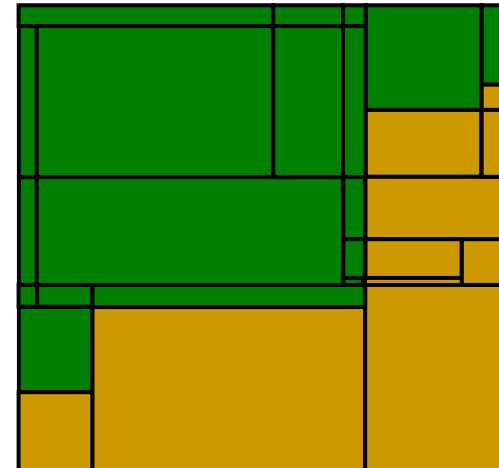
# НЕСКОЛЬКО моделей построенных в разных условиях



# Ансамбль = Комбинация моделей



$$\text{ave}(T_1, T_2, T_3) =$$



# Bagging

	k=1	k=2	k=3	k=4	...
<u>case</u>	<u>freq</u>	<u>freq</u>	<u>freq</u>	<u>freq</u>	
1	1	0	3	1	
2	0	1	1	1	
3	2	0	0	2	
4	0	2	2	0	
5	2	2	0	1	
6	1	1	0	1	

Below each frequency column is a decision tree diagram. The nodes are represented by circles: grey for internal nodes and green for leaf nodes. The trees correspond to the frequency distributions shown above them.

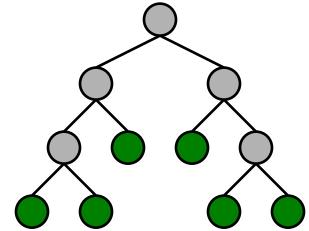
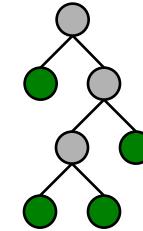
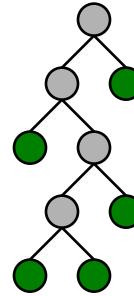
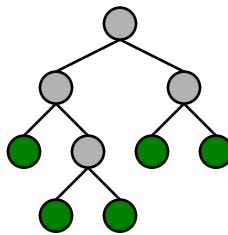
- k=1:** A tree with one internal node (grey) and three leaf nodes (green). The root node has two children, which are both leaf nodes (green).
- k=2:** A tree with one internal node (grey) and five leaf nodes (green). The root node has two children, each of which has two children (green leaf nodes).
- k=3:** A tree with one internal node (grey) and six leaf nodes (green). The root node has three children, each of which has two children (green leaf nodes).
- k=4:** A tree with one internal node (grey) and four leaf nodes (green). The root node has two children, each of which has two children (green leaf nodes).

...

# Arc-x4

	k=1	k=2	k=3	k=4	...
--	-----	-----	-----	-----	-----

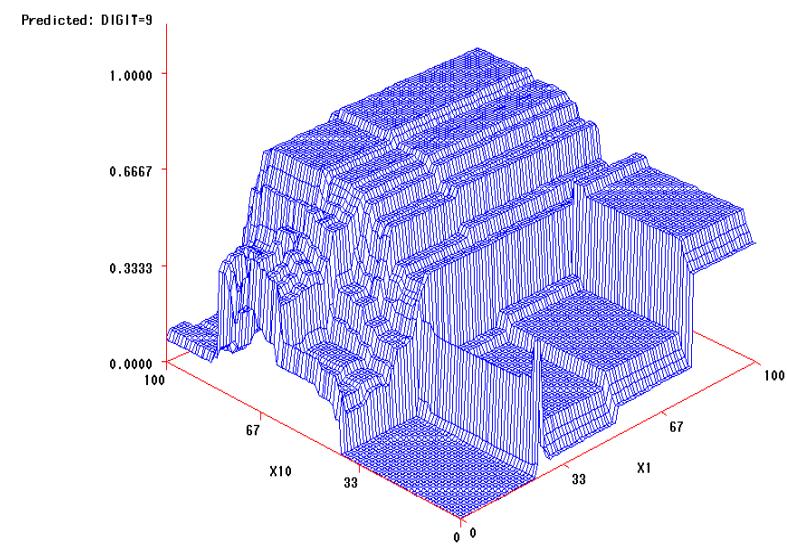
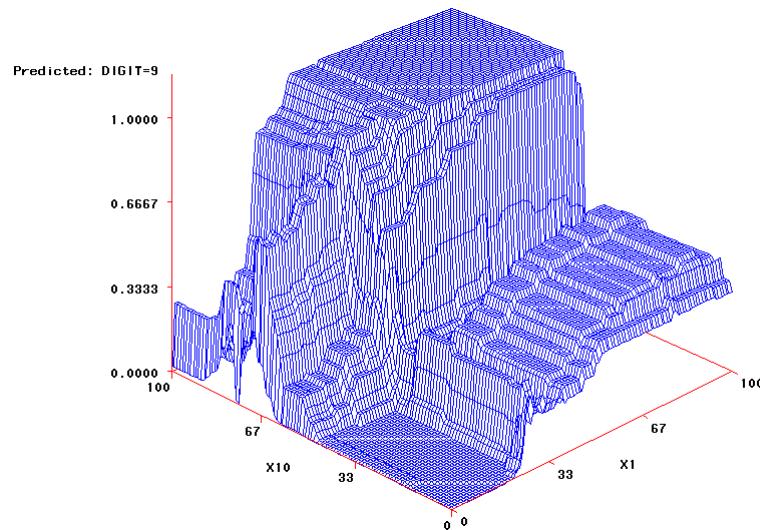
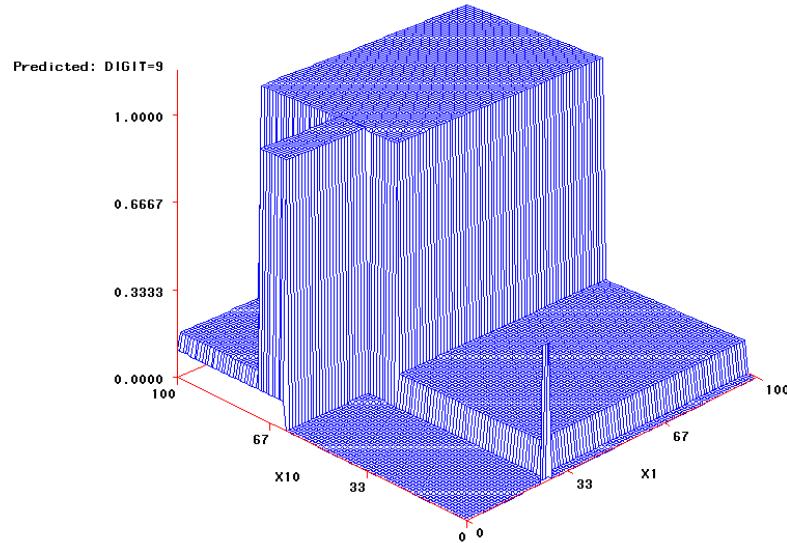
<u>case</u>	<u>freq</u>	<u>m</u>	<u>freq</u>	<u>m</u>	<u>freq</u>	<u>m</u>	<u>freq</u>
1	1	1	1.5	1	.5	2	.97
2	1	0	.75	0	.25	0	.06
3	1	1	1.5	2	4.25	3	4.69
4	1	0	.75	1	.5	1	.11
5	1	0	.75	0	.25	0	.06
6	1	0	.75	0	.25	1	.11



$$p(i) = \frac{1 + m(i)^4}{\sum (1 + m(i)^4)}$$

...

# Обычное деревья и их ансамбли



# Случайный лес

## ■ Основные особенности:

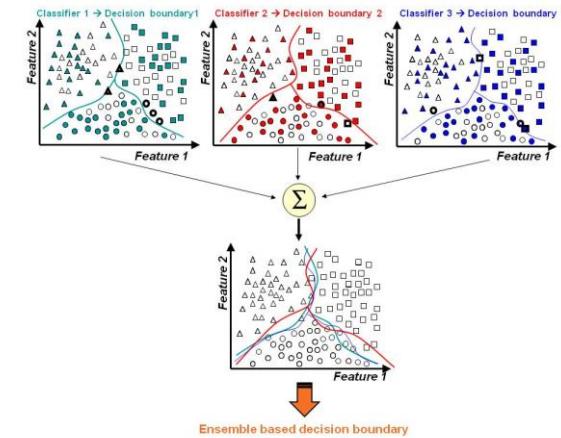
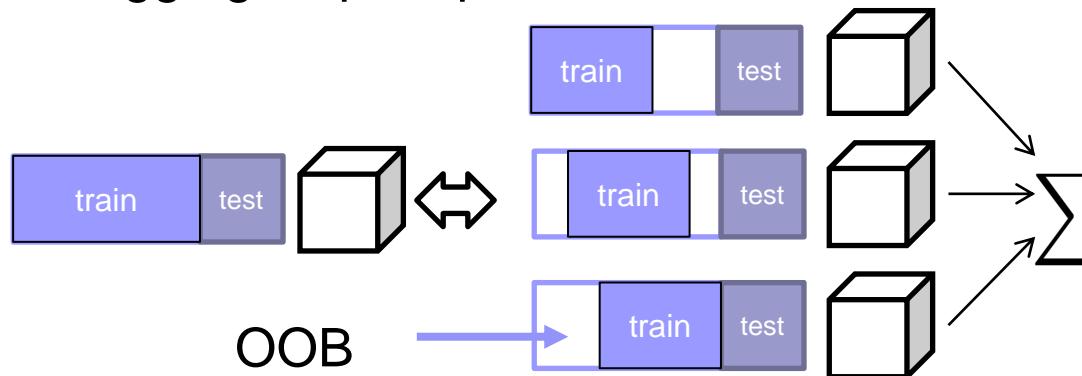
- Bagging с пропорцией – усреднение отклика по bootstrap выборке. Дополнительный sampling – выборка с возвращением набора меньшего размера, пропорция – параметр.
- Случайные подпространства признаков на каждом шаге (sampling признаков).  $\sqrt{\# \text{ of } inputs}$  -
- out-of-bag выборка – то, что не вошло в bootstrap для каждого дерева, ОOB используется для оценки качества каждого дерева в ансамбле, потом усредняется.

## ■ Почему работает:

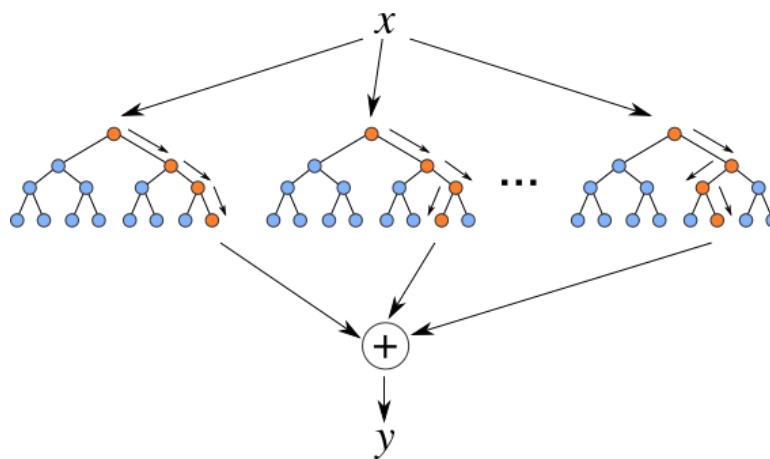
- Большее изменение тренировочных наборов чем в обычном bagging, а значит большая вариация отклика и меньшая корреляция моделей.
- Ансамбль непохожих моделей приводит к лучшему прогнозу.
- Сложность – число деревьев в ансамбле. Можно отбирать по ОOB.
- Случайный лес не склонен к переобучению даже на сложных деревьях.
- «Модель из коробки» – мало параметров, но хорошее качество!!!

# Иллюстрация работы случайного леса

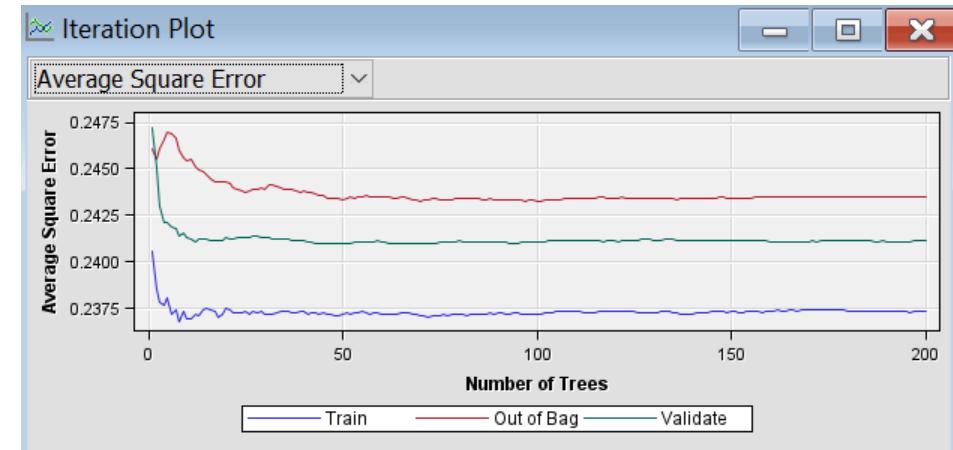
Bagging с пропорцией:



Применение ансамбля:



Оценка качества по ОOB:



# Random Forest (Python)

```
from sklearn.datasets import fetch_covtype
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_recall_fscore_support, accuracy_score
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split
```

```
covtype = fetch_covtype()
X, y = covtype.data, covtype.target
labels = np.unique(y)
X.shape, y.shape, labels
```

```
((581012, 54), (581012,), array([1, 2, 3, 4, 5, 6, 7], dtype=int32))
```

```
print(covtype.DESCR)
```

```
.. _covtype_dataset:
```

```
Forest covtypes
```

```
-----
```

The samples in this dataset correspond to 30×30m patches of forest in the US, collected for the task of predicting each patch's cover type, i.e. the dominant species of tree.

There are seven covtypes, making this a multiclass classification problem.

Each sample has 54 features, described on the

`dataset's homepage <<https://archive.ics.uci.edu/ml/datasets/Covtype>>`\_\_.

Some of the features are boolean indicators, while others are discrete or continuous measurements.

# Random Forest (Python)

```
covtype_split = train_test_split(X, y, train_size=10000, test_size=10000, stratify=y, random_state=0)
X_train, X_test, y_train, y_test = covtype_split
```

```
class_weight = compute_class_weight("balanced", y=y_train, classes=labels)
covtype_class_weight = dict(zip(labels, class_weight))
```

```
n_estimators = 150
forest = RandomForestClassifier(n_estimators=n_estimators,
                                criterion="entropy",
                                min_samples_split=10,
                                min_samples_leaf=10,
                                max_features=10, # features to consider for each split
                                max_depth=10,
                                max_leaf_nodes=10,
                                class_weight=covtype_class_weight,
                                bootstrap=True, max_samples=0.15, # Samples % for each tree
                                ccp_alpha=0.0, # pruning
                                oob_score=True, # compute out-of-bag
                                warm_start=True, # add trees to the existing forest
                                random_state=0)
```

# Random Forest (Python)

```
def sklearn_fit_history(model, n_estimators, X_train, y_train, valid=None):
    result = []
    for i in range(10, n_estimators, 5):
        model.set_params(n_estimators=i) # increase estimators count
        model.fit(X_train, y_train)
        d = {}
        if valid is not None:
            d["valid_score"] = model.score(*valid)
        d["train_score"] = model.score(X_train, y_train)
        if hasattr(model, "oob_score_"):
            d["oob_score"] = model.oob_score_
        result.append(d)
    return pd.DataFrame(data=result)
```

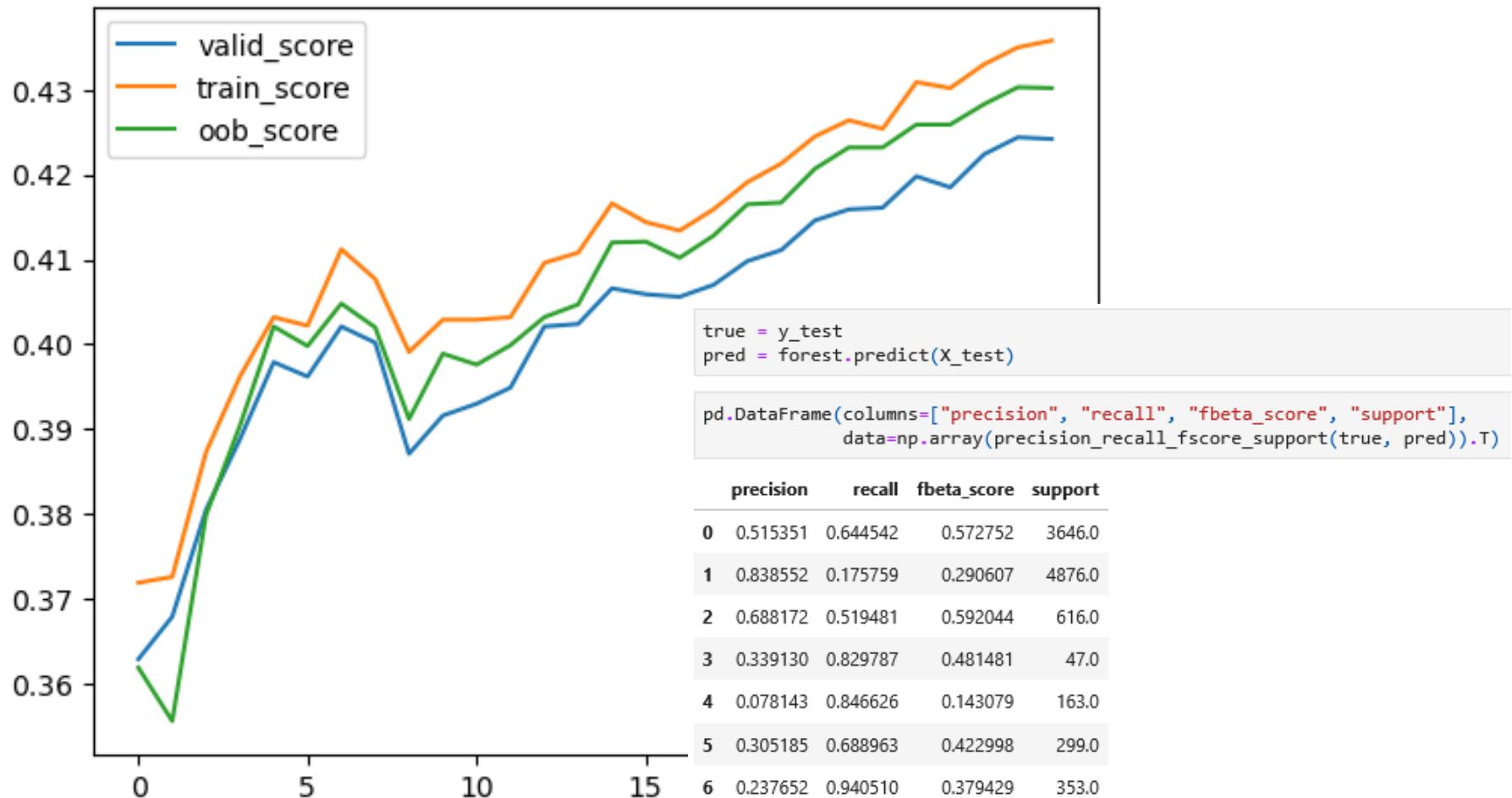
Реализуем сохранение истории out-of-bag score

```
# model.fit(X_train, y_train)
history = sklearn_fit_history(forest, n_estimators, X_train, y_train, (X_test, y_test))
```

# Random Forest (Python)

```
history.plot()
```

```
<AxesSubplot: >
```



# Градиентный бустинг

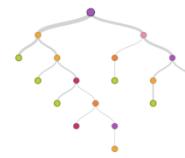
Финальная модель ( $M$  – число итераций)

$$F_M(x) = F_0 + \nu\beta_1T_1(x) + \nu\beta_2T_2(x) + \dots + \nu\beta_MT_M(x)$$

**For  $m = 1$  to  $M$ , do...**

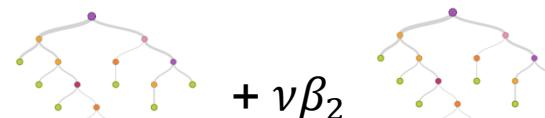
$$F_m(x) = F_{m-1}(x) + \nu\beta_mT_m(x)$$

Step 1



Step 2

$$\nu\beta_1$$



Исправляет  
ошибки

Каждая следующая  
модель обучается на  
«псевдоостатках» от  
предыдущих

Step 3

$$\nu\beta_1$$



Исправляет  
ошибки

# Почему «градиентный»?

Приближенное разложение функции потерь в ряд Тейлора до первого порядка:

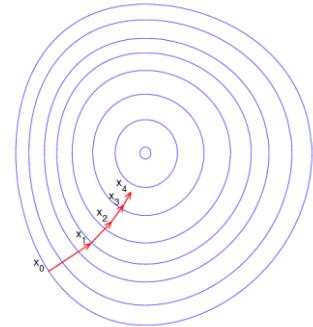
$$\sum_i L(y_i, F_{m-1}(x_i) + h(x_i; a_m)) \approx \sum_i L(y_i, F_{m-1}(x_i)) + h(x_i; a_m) \frac{\partial L(y_i, F_{m-1})}{\partial F_{m-1}}(x_i) + \dots$$

Пусть  $F = [F(x_i)]_{i=1}^N$  итерационно находим  $\operatorname{argmin} Q$  такой что:

т-я итерация  
градиентного спуска

$$F_m = F_{m-1} - b_m \nabla Q$$

$$\nabla Q = \left[ \frac{\partial Q}{\partial F_{m-1}}(x_i) \right]_{i=1}^N = \left[ \frac{\partial L(y_i, F_{m-1})}{\partial F_{m-1}}(x_i) \right]_{i=1}^N$$



На каждом шаге строим приближенную модель  $h(x; a_m)$  с  $\nabla Q$  в качестве отклика, такую что:

$$a_m = \operatorname{train}([x_i]_{i=1}^N, [\nabla Q_i]_{i=1}^N),$$

Находим размер шага с помощью «линейного поиска»

$$b_m = \operatorname{argmin}_{b \in R} \sum_{i=1}^N L(y_i, F_{m-1}(x_i) - bh(x_i; a_m))$$

# Градиентный бустинг (Python)

```
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_percentage_error

housing = fetch_california_housing()
X, y = housing.data, housing.target
X.shape, y.shape, housing.target_names

((20640, 8), (20640,), ['MedHouseVal'])
```

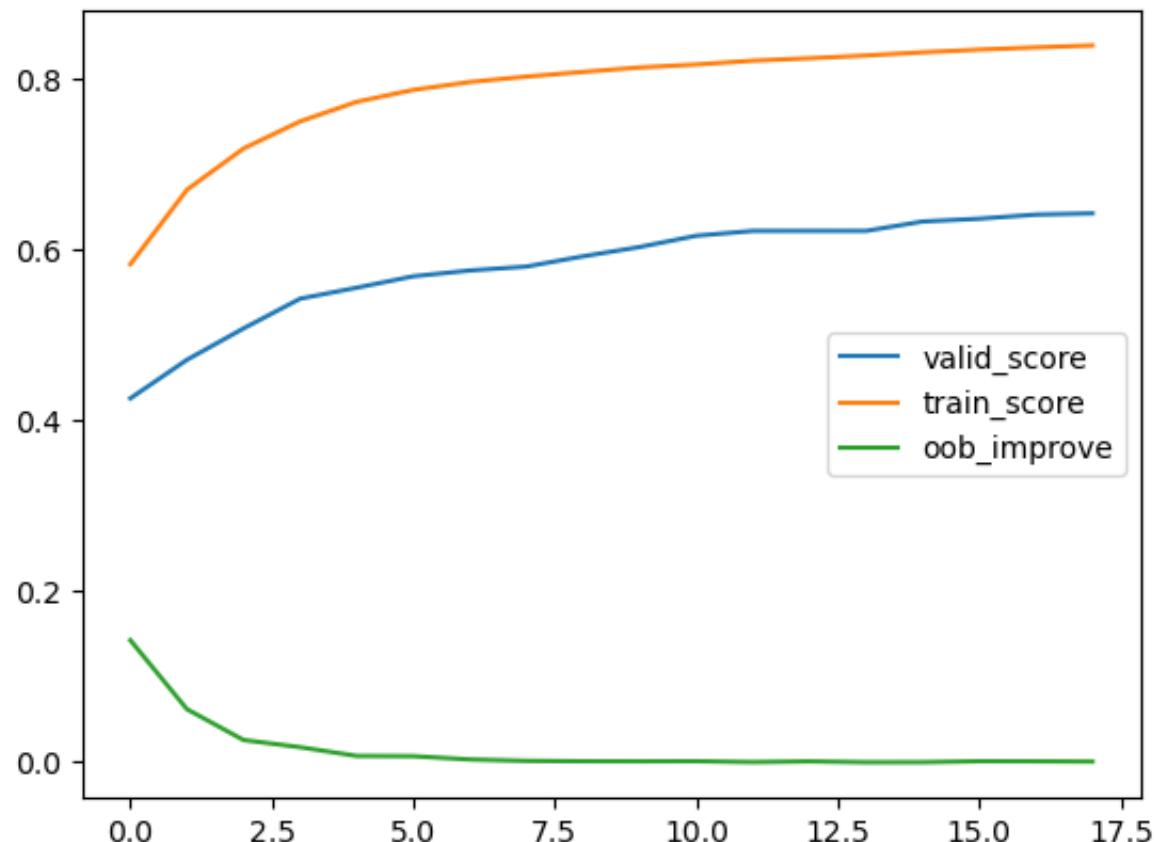
```
N = 15000
X_train, y_train = X[:N], y[:N]
X_test, y_test = X[N:], y[N:]
```

```
n_estimators = 100
boosting = GradientBoostingRegressor(n_estimators=n_estimators, learning_rate=0.1,
                                      max_depth=5,
                                      max_leaf_nodes=10,
                                      subsample=0.75, # stochastic if <1.0
                                      ccp_alpha=0.0, # pruning
                                      warm_start=True, # add trees to the existing forest
                                      random_state=0)

#boosting.fit(X_train, y_train)
history = sklearn_fit_history(boosting, n_estimators, X_train, y_train, (X_test, y_test))
```

# Градиентный бустинг (Python) (2)

```
history = history  
history["oob_improve"] = boosting.oob_improvement_[::-5][:-1] # stochastic boosting is required!  
history.plot()
```

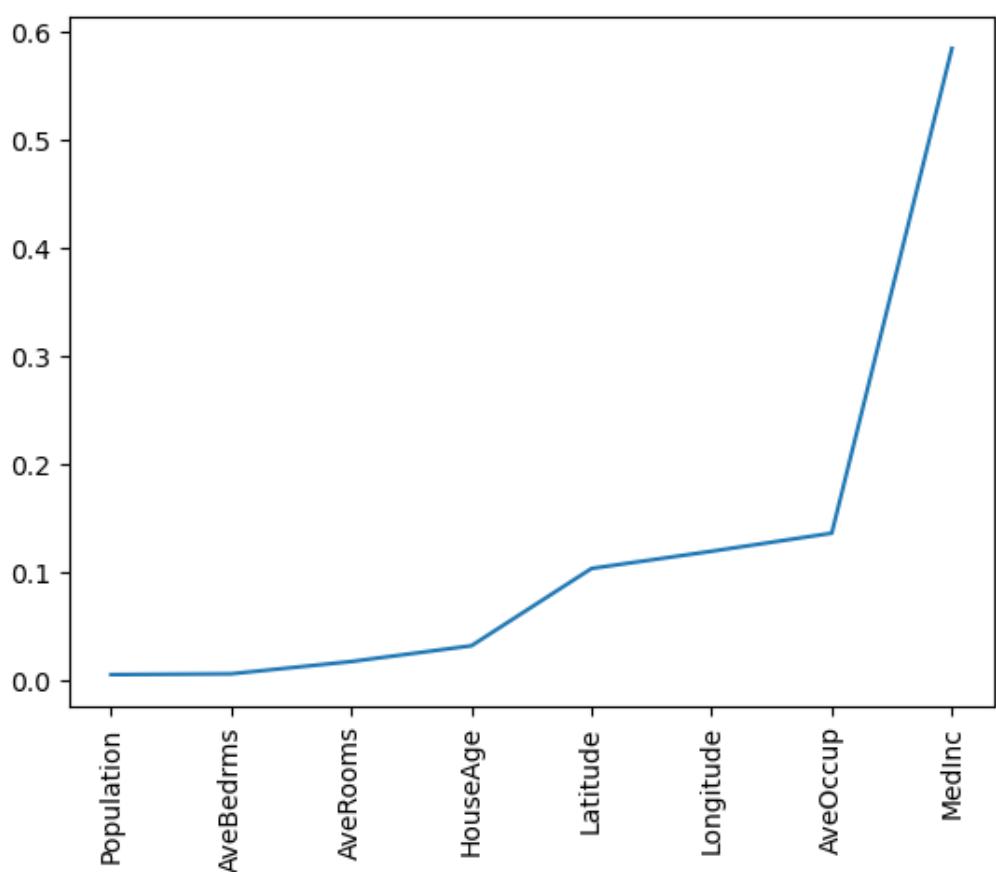


# Градиентный бустинг (Python) (3)

```
mean_absolute_percentage_error(y_test, boosting.predict(X_test))
```

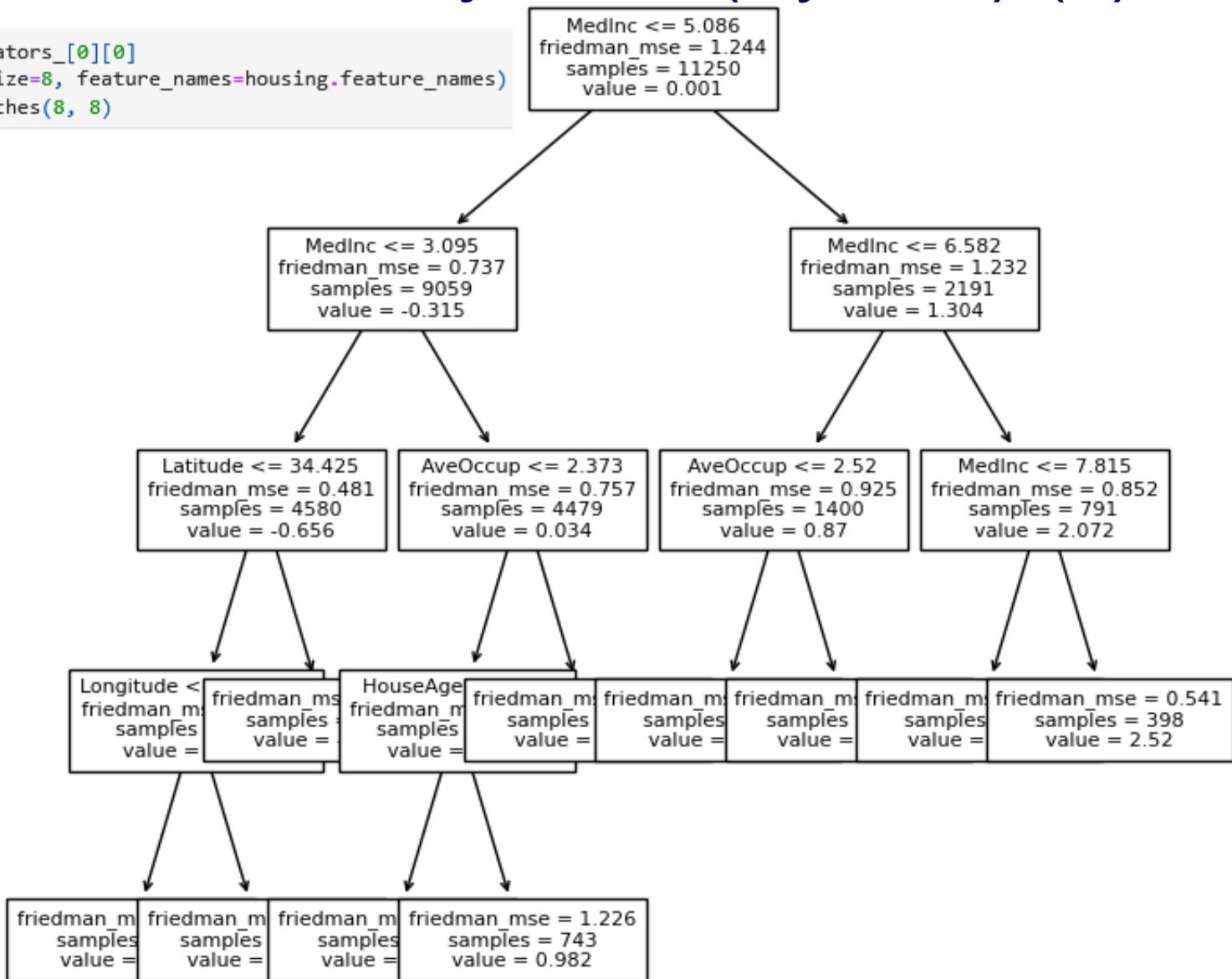
```
0.23656392616219316
```

```
importance = pd.Series(index=housing.feature_names, data=boosting.feature_importances_)  
importance.sort_values().plot()  
plt.xticks(rotation='vertical');
```



# Градиентный бустинг (Python) (4)

```
tree = boosting.estimators_[0][0]
plot_tree(tree, fontsize=8, feature_names=housing.feature_names)
plt.gcf().set_size_inches(8, 8)
```



# LightGBM

## ■ Особенности:

- Градиентный бустинг на деревьях решений с возможностью дообучения для больших объемов разнородных данных
- Функции потерь для бинарного, много-классового, числового отклика, в том числе для несимметричных распределений, а также с робастной функцией потерь
- Пользовательские функции потерь
- L1 и L2 регуляризация
- Ранняя остановка обучения, колбеки, кастомизированное журналирование
- Гистограммный метод поиска наилучшего разбиения – дискретизация числовых признаков и сортировка наблюдений по признаку для ускорения
- Gradient-based One-Side Sampling (GOSS) – адаптивный случайный сэмплинг пропорционально градиенту каждого наблюдения при расчете критериев для поиска разбиения
- Exclusive Feature Bundling – жадный алгоритм «группировки» признаков имеющих взаимоисключающие значения

# LightGBM

```
from lightgbm import LGBMClassifier, early_stopping, record_evaluation, plot_importance

X_train, X_test, y_train, y_test = covtype_split

lgbm_classifier = LGBMClassifier(boosting_type="gbdt", # dart, rf
                                  num_leaves=10,
                                  max_depth=-1,
                                  learning_rate=0.05,
                                  min_child_samples=20, # min_samples_leaf
                                  n_estimators=2000,
                                  subsample=0.15, # subsample % for stochastic
                                  subsample_freq=1, # how many times to subsample
                                  colsample_bytree=0.15, # features by trees
                                  class_weight="balanced",
                                  reg_alpha=1.0 # l1 regularization
)

history = {}
lgbm_classifier.fit(X_train, y_train, feature_name=covtype.feature_names,
                     eval_set=[(X_test, y_test), (X_train, y_train)],
                     callbacks=[early_stopping(stopping_rounds=10), record_evaluation(history)])
```

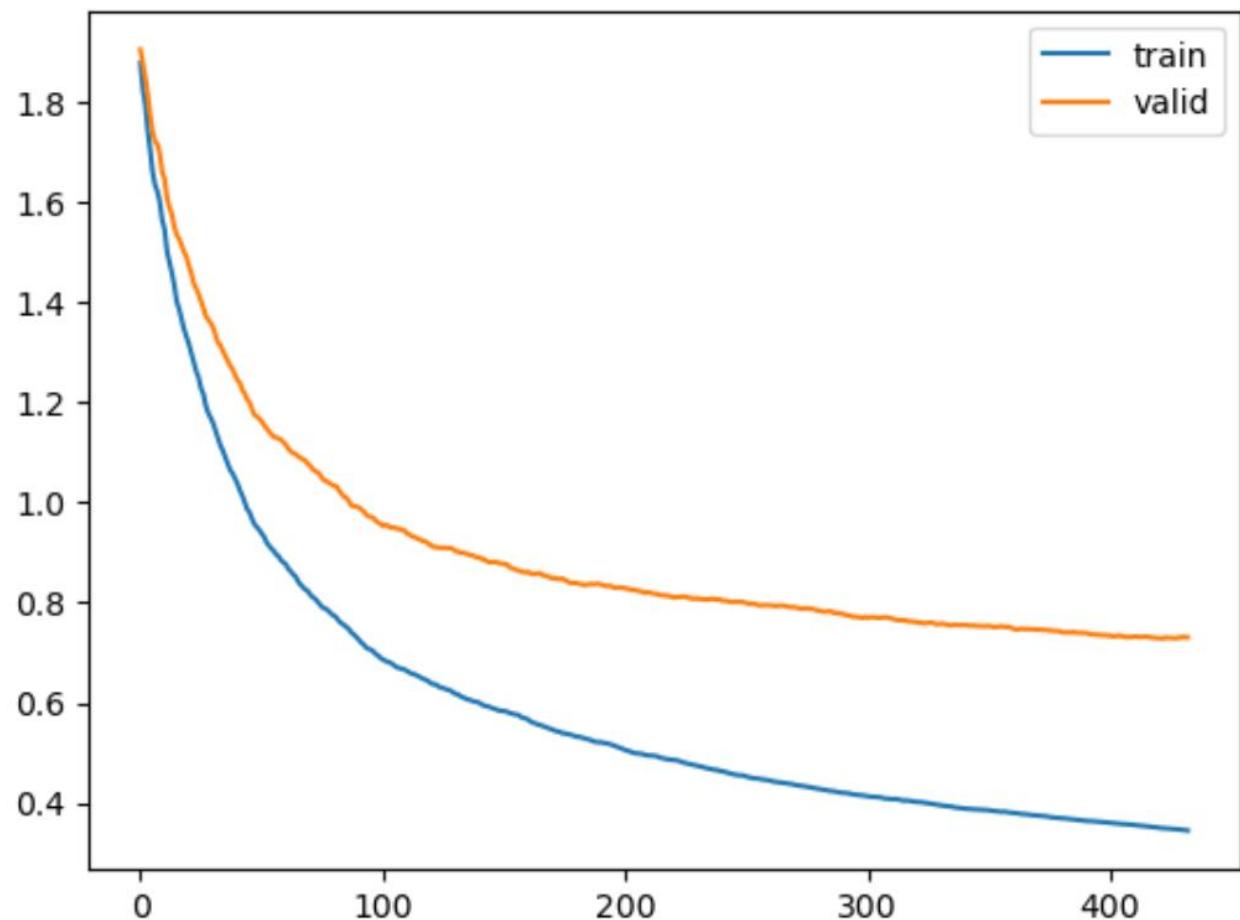
Training until validation scores don't improve for 10 rounds

Early stopping, best iteration is:

[423] training's multi\_logloss: 0.349899 valid\_0's multi\_logloss: 0.728702

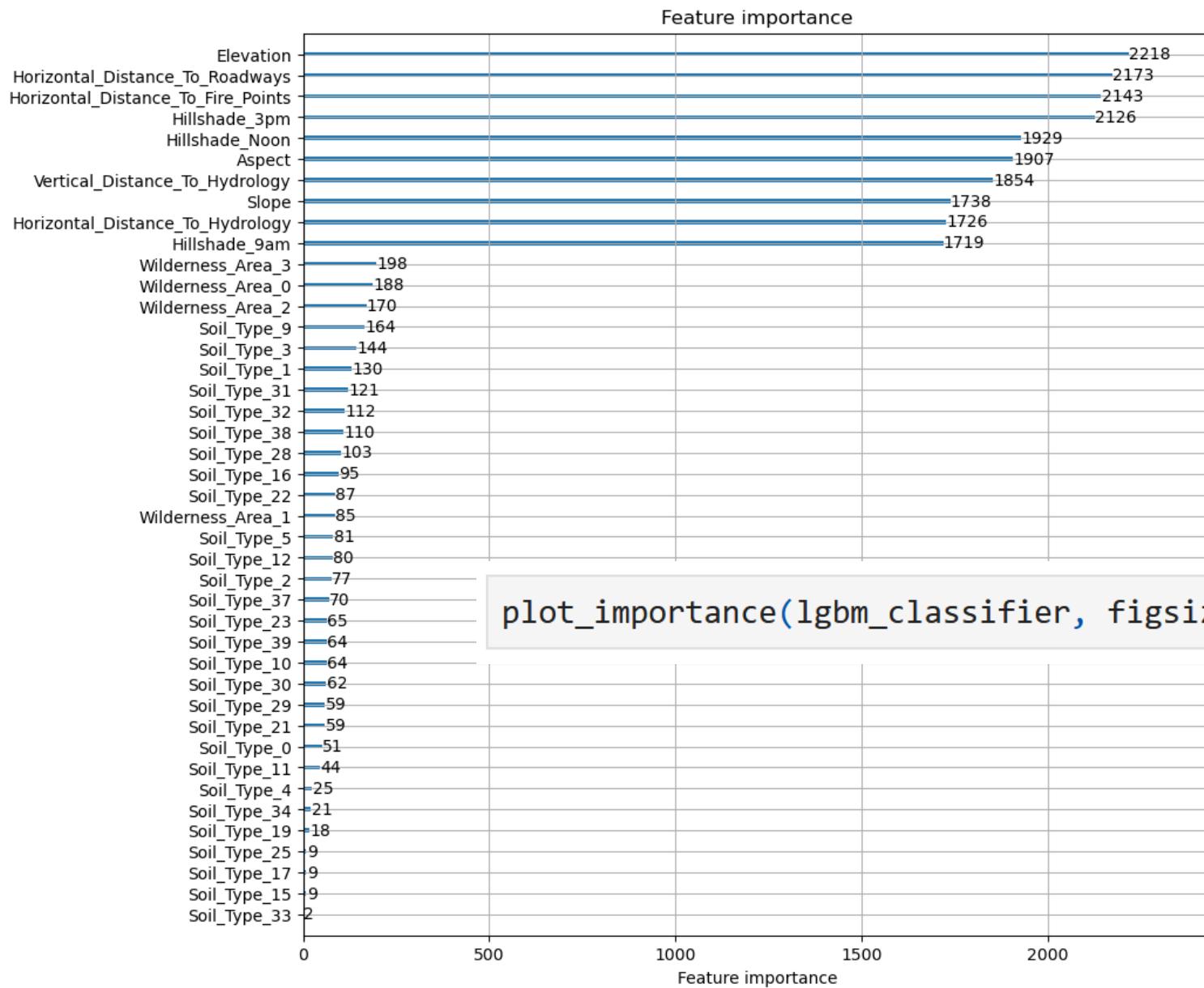
# LightGBM

```
train = history["training"]["multi_logloss"]
valid = history["valid_0"]["multi_logloss"]
pd.DataFrame(dict(train=train, valid=valid)).plot()
```



# LightGBM

Features



```
plot_importance(lgbm_classifier, figsize=(10, 10));
```

# XGBoost

## ■ Особенности:

- **Ньютоновский бустинг**  
(разложение в ряд Тейлора до второго порядка)
- Функции потерь для разных типов отклика
- L1 и L2 регуляризация
- Ранняя остановка обучения, колбеки
- Гистограммный метод поиска наилучшего разбиения

1. Initialize model with a constant value:

$$\hat{f}_{(0)}(x) = \arg \min_{\theta} \sum_{i=1}^N L(y_i, \theta).$$

2. For  $m = 1$  to  $M$ :

1. Compute the 'gradients' and 'hessians':

$$\hat{g}_m(x_i) = \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

$$\hat{h}_m(x_i) = \left[ \frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2} \right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

2. Fit a base learner (or weak learner, e.g. tree) using the training set

$$\left\{ x_i, -\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} \right\}_{i=1}^N \text{ by solving the optimization problem below:}$$

$$\hat{\phi}_m = \arg \min_{\phi \in \Phi} \sum_{i=1}^N \frac{1}{2} \hat{h}_m(x_i) \left[ -\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i) \right]^2.$$
$$\hat{f}_m(x) = \alpha \hat{\phi}_m(x).$$

3. Update the model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x).$$

3. Output  $\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^M \hat{f}_m(x).$

# XGBoost

```
import xgboost as xgb
```

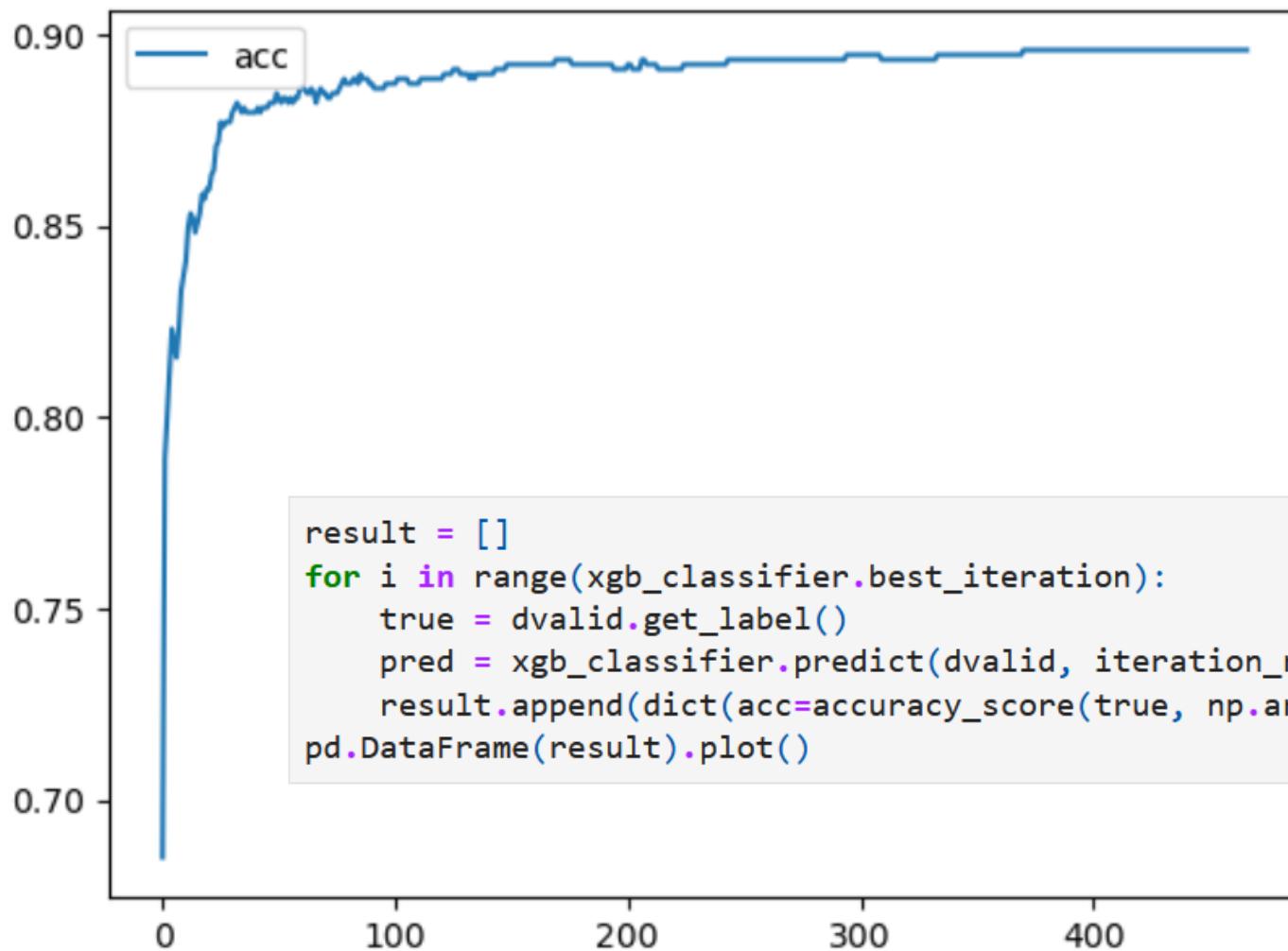
```
# https://xgboost.readthedocs.io/en/stable/parameter.html
param = dict(objective="multi:softprob", num_class=10, # softmax
              booster="gbtree", learning_rate=0.1, max_depth=5, subsample=0.5,
              alpha=5, # l1 regularization
              tree_method="auto") # approx, hist, gpu_hist
```

```
N = 1000
dtrain = xgb.DMatrix(digits.data[:N], label=digits.target[:N])
dvalid = xgb.DMatrix(digits.data[N:], label=digits.target[N:])
evallist = [(dtrain, 'train'), (dvalid, 'eval')]
```

```
xgb_classifier = xgb.train(param, dtrain, 1000, evals=evallist, early_stopping_rounds=10)
# xgb_classifier.save_model('mymodel')
```

```
[0]    train-mlogloss:2.03753  eval-mlogloss:2.08882
[1]    train-mlogloss:1.82993  eval-mlogloss:1.90860
[2]    train-mlogloss:1.66188  eval-mlogloss:1.76258
[3]    train-mlogloss:1.53163  eval-mlogloss:1.65200
...
[475]   train-mlogloss:0.16262  eval-mlogloss:0.45679
[476]   train-mlogloss:0.16262  eval-mlogloss:0.45679
```

# XGBoost



# Заключение

- В рамках курса были рассмотрены:
  - Постановка задач, терминология, основные определения и области практического применения методов Data mining
  - Основные методы DM на основе «обучения без учителя» для поиска ассоциативных правил, кластеризации, поиска зависимостей и выявления скрытых структур данных, а также методы поиска аномалий «без учителя»
  - Основные методы DM на основе «обучения с учителем» для решения задач прогнозирования, классификации и регрессии, включая регрессионные модели, нейронные сети типа MLP и RBF, деревья решений и ансамбли
  - Необходимые методы предобработки данных