Машинное обучение в гидрометеорологии: Лекция №9.

Деревья решений и их оптимизация. Ансамбли деревьев решений. Оптимизация гиперпараметров.

Михаил Иванович Варенцов (mikhail.varentsov@srcc.msu.ru)

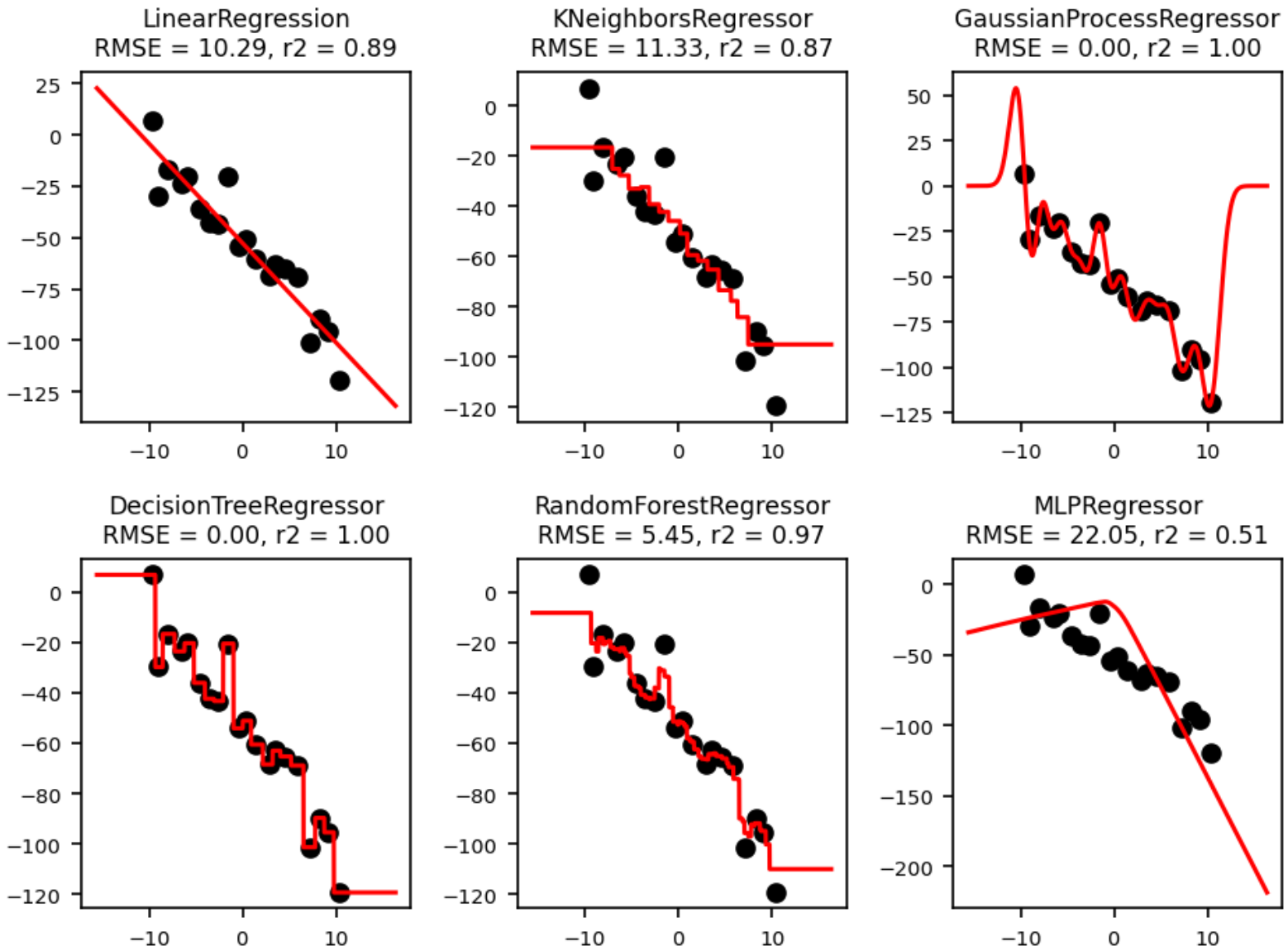
Михаил Алексеевич Криницкий (krinitsky@sail.msk.ru)

ml4hydromet@ml4es.ru

ИНТЕЛЛЕКТ

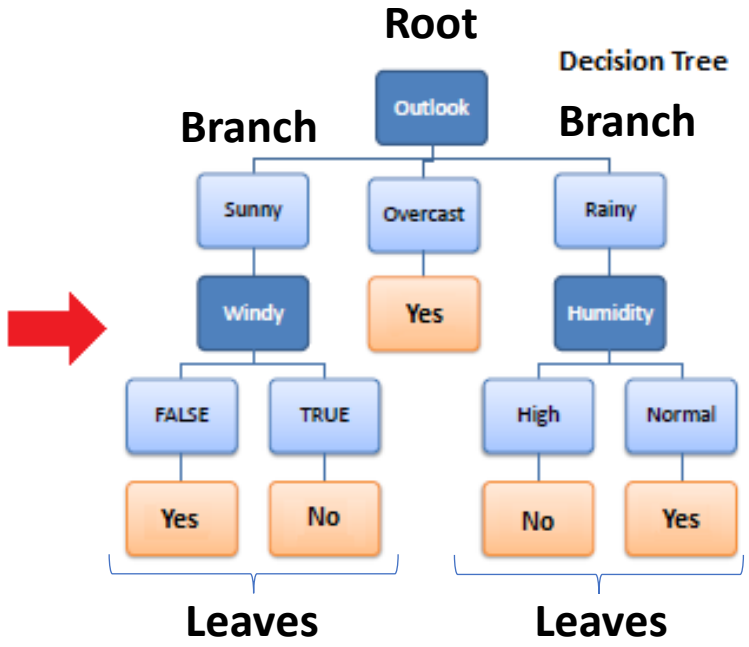
ФОНД РАЗВИТИЯ НАУКИ И ОБРАЗОВАНИЯ

Ранее в ML4hydromet

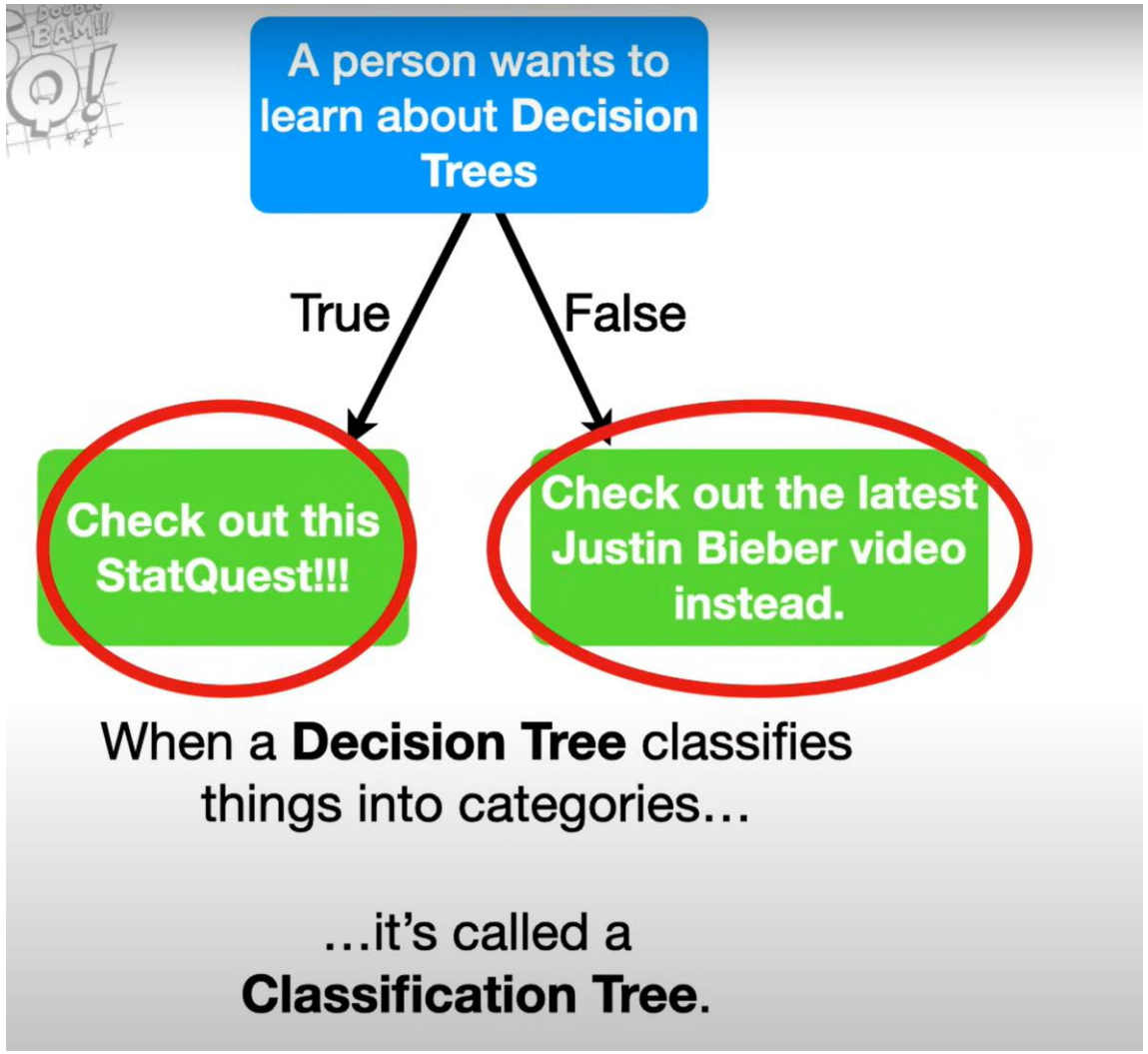


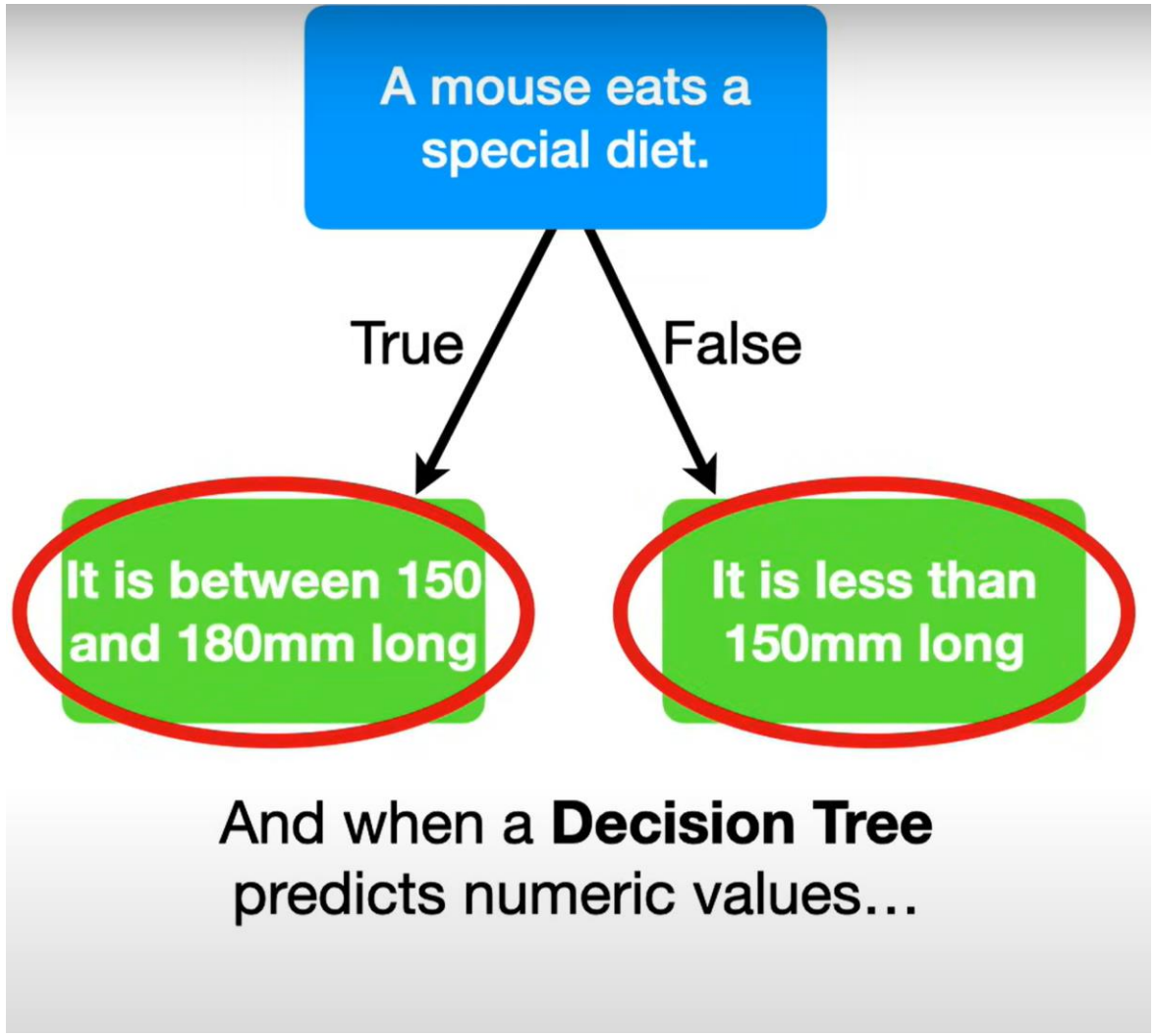
Деревья решений

Predictors				Target
Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



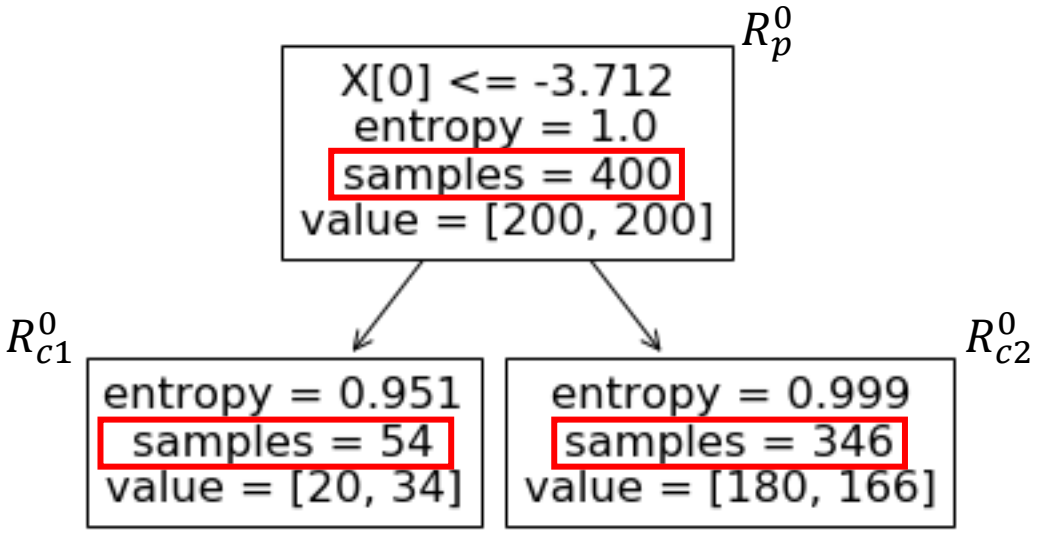
Деревья решений

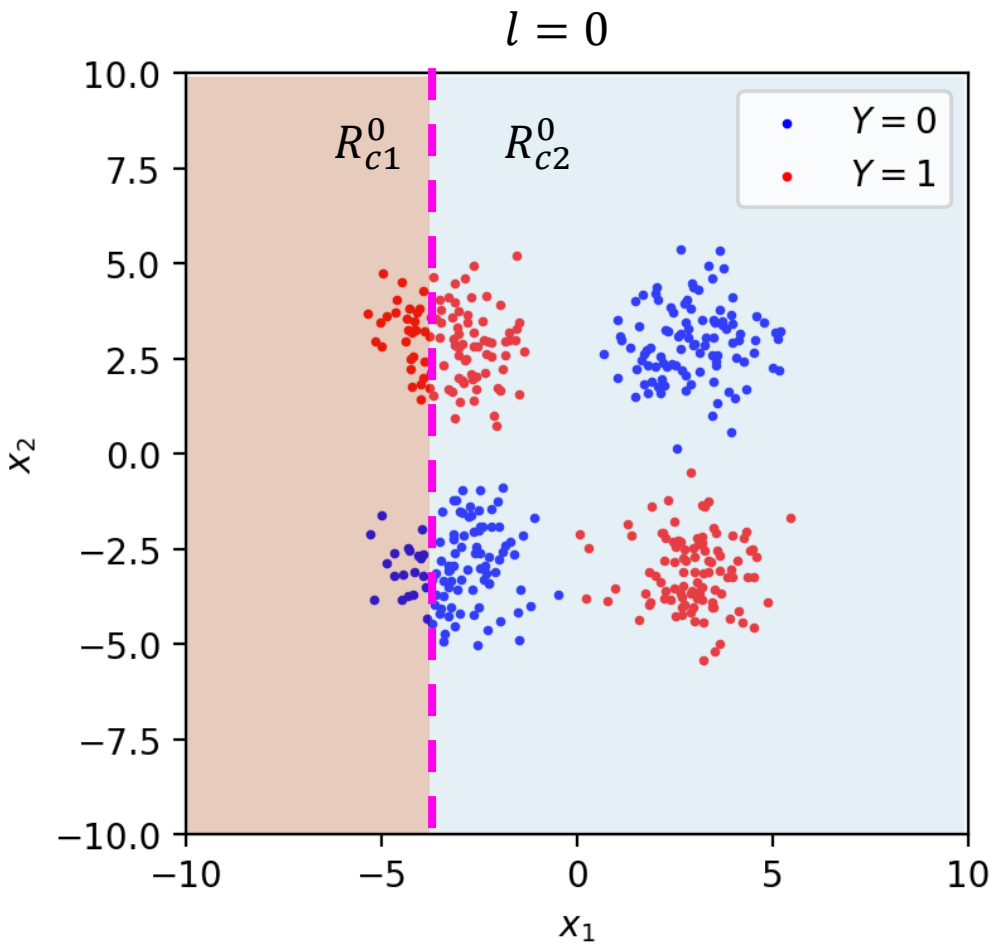




DT в режиме исполнения

Схема ветвления на первом уровне ($l = 0$)

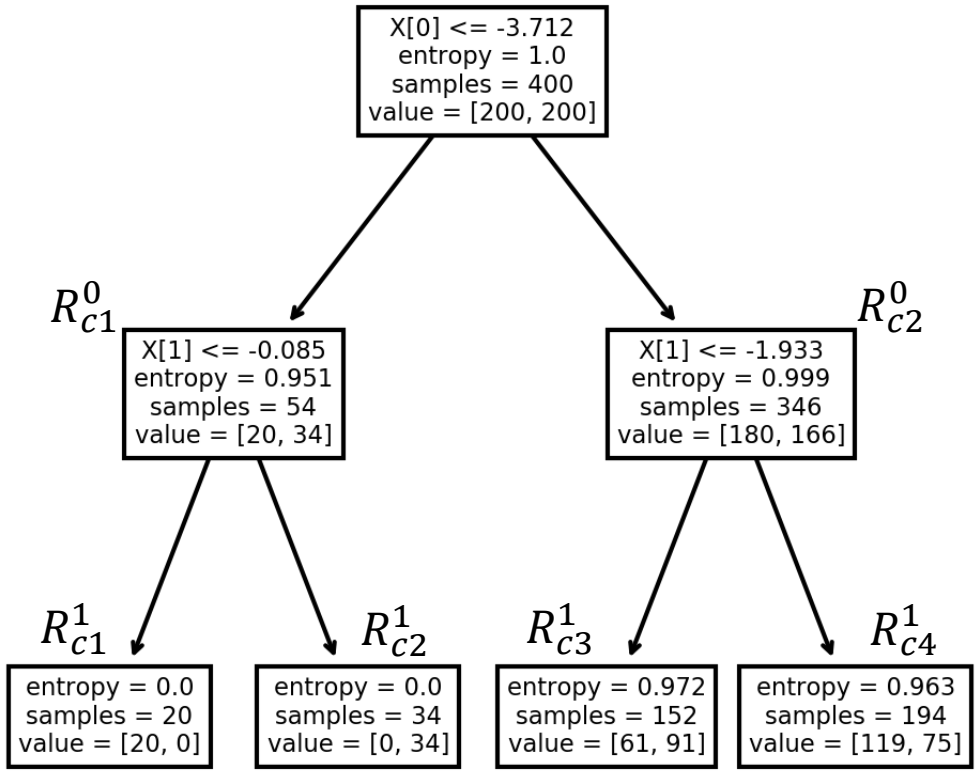


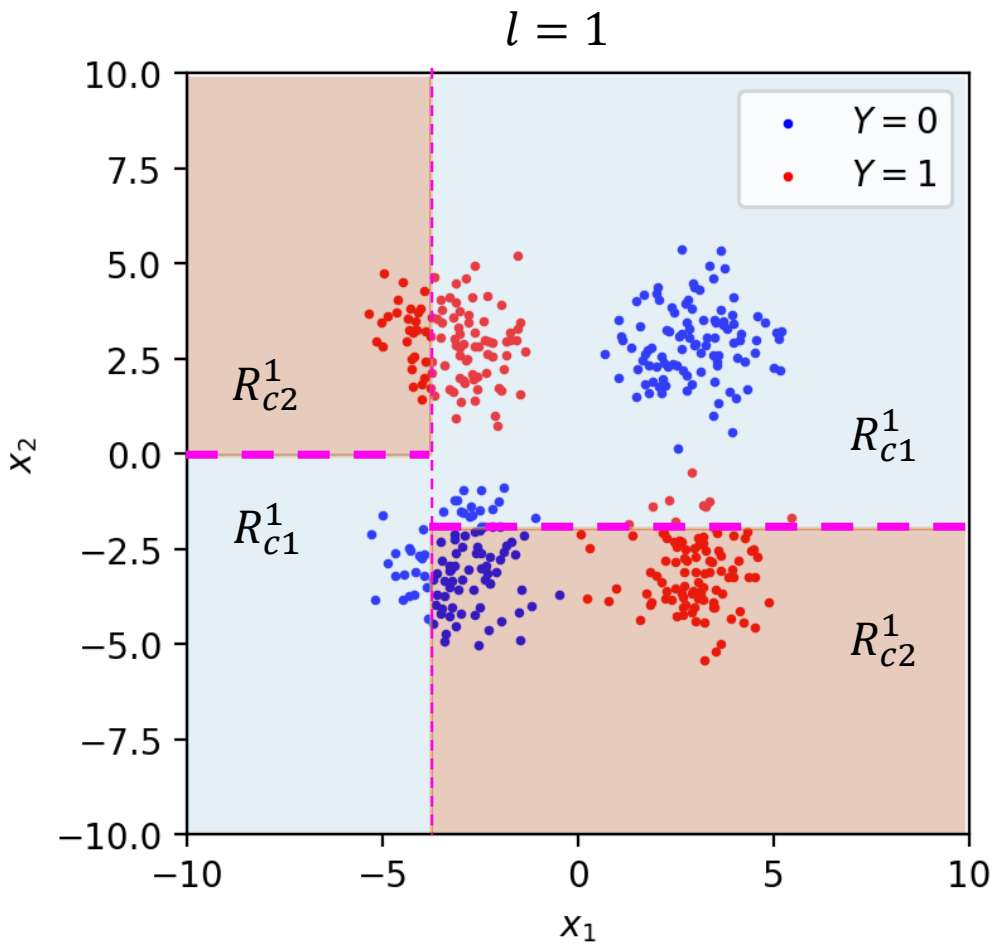


На рисунке: деление выборки всех примеров R_p^0 на R_{c1}^0, R_{c2}^0. Внимание: на рисунке – тестовая выборка, правила деления (номер признака j_l и пороговое значение $t^{(l)}$) были определены во время обучения.

DT в режиме исполнения

Схема ветвления на втором уровне ($l = 1$)

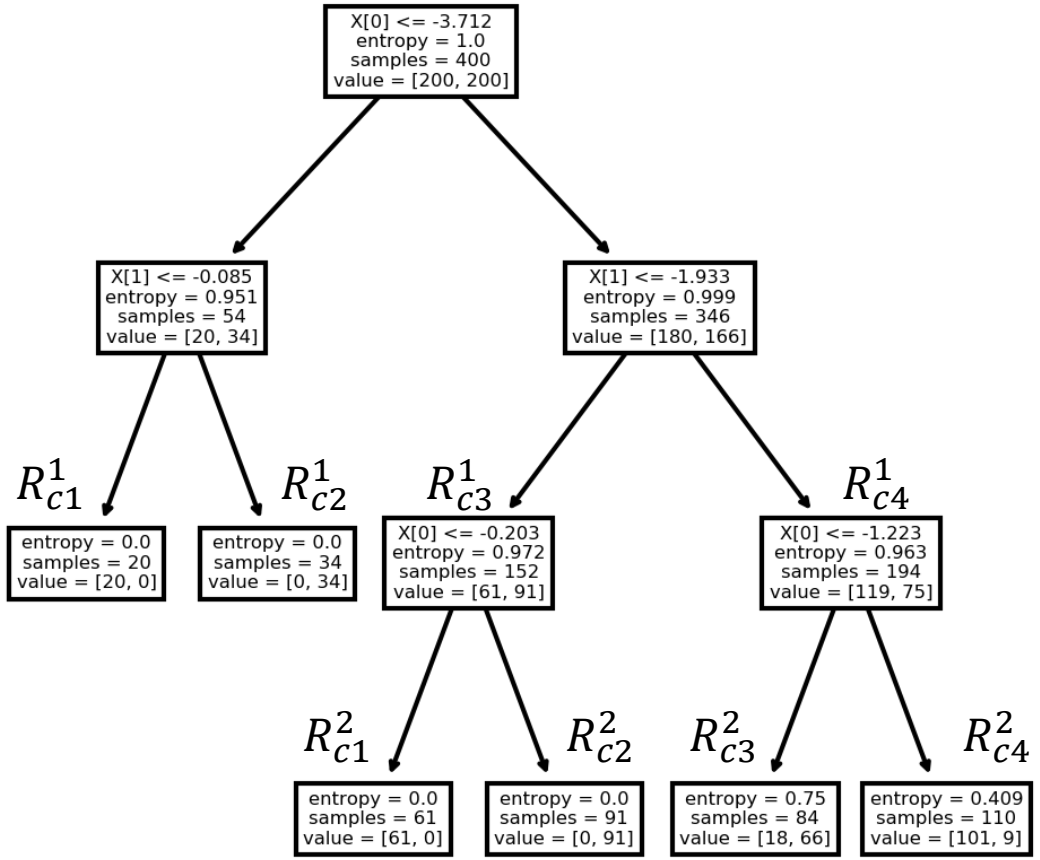


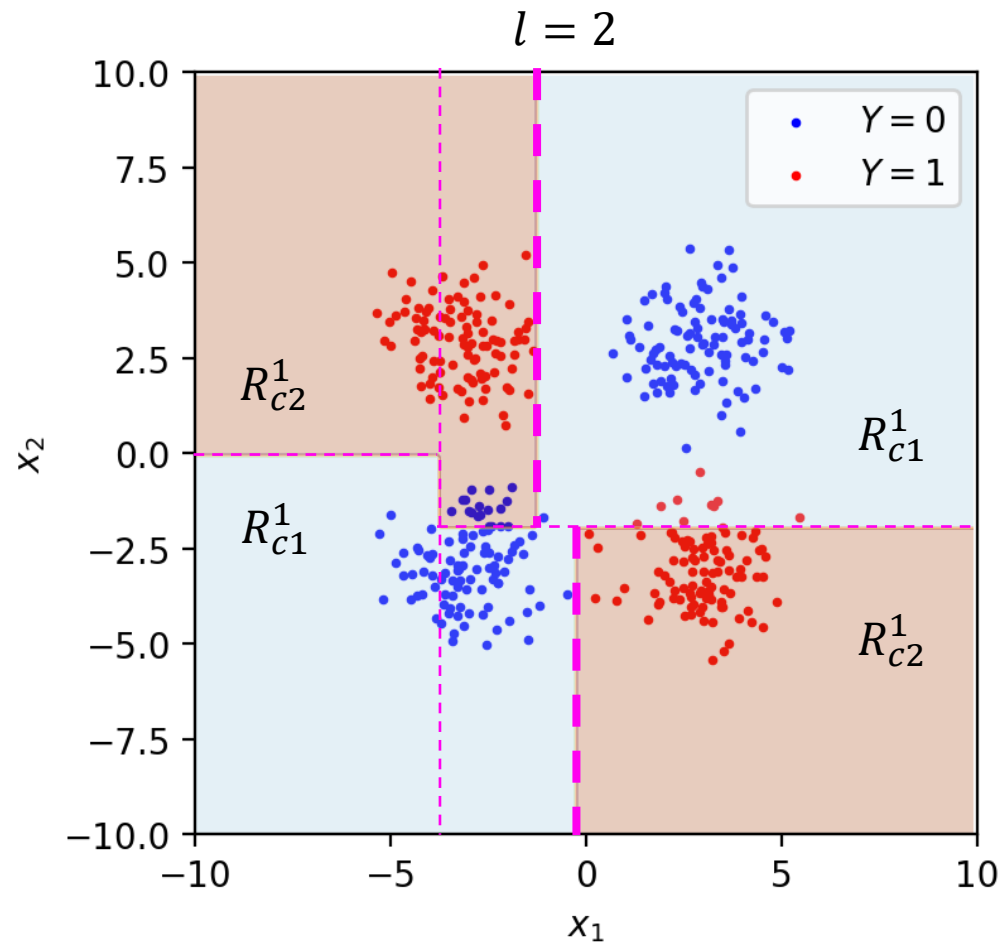


На рисунке: деление выборки всех примеров R^0_p на второй итерации ветвления. Внимание: на рисунке – тестовая выборка, правила деления (номер признака j_l и пороговое значение $t^{(l)}$) были определены во время обучения.

DT в режиме исполнения

Схема ветвления на третьем уровне ($l = 2$)





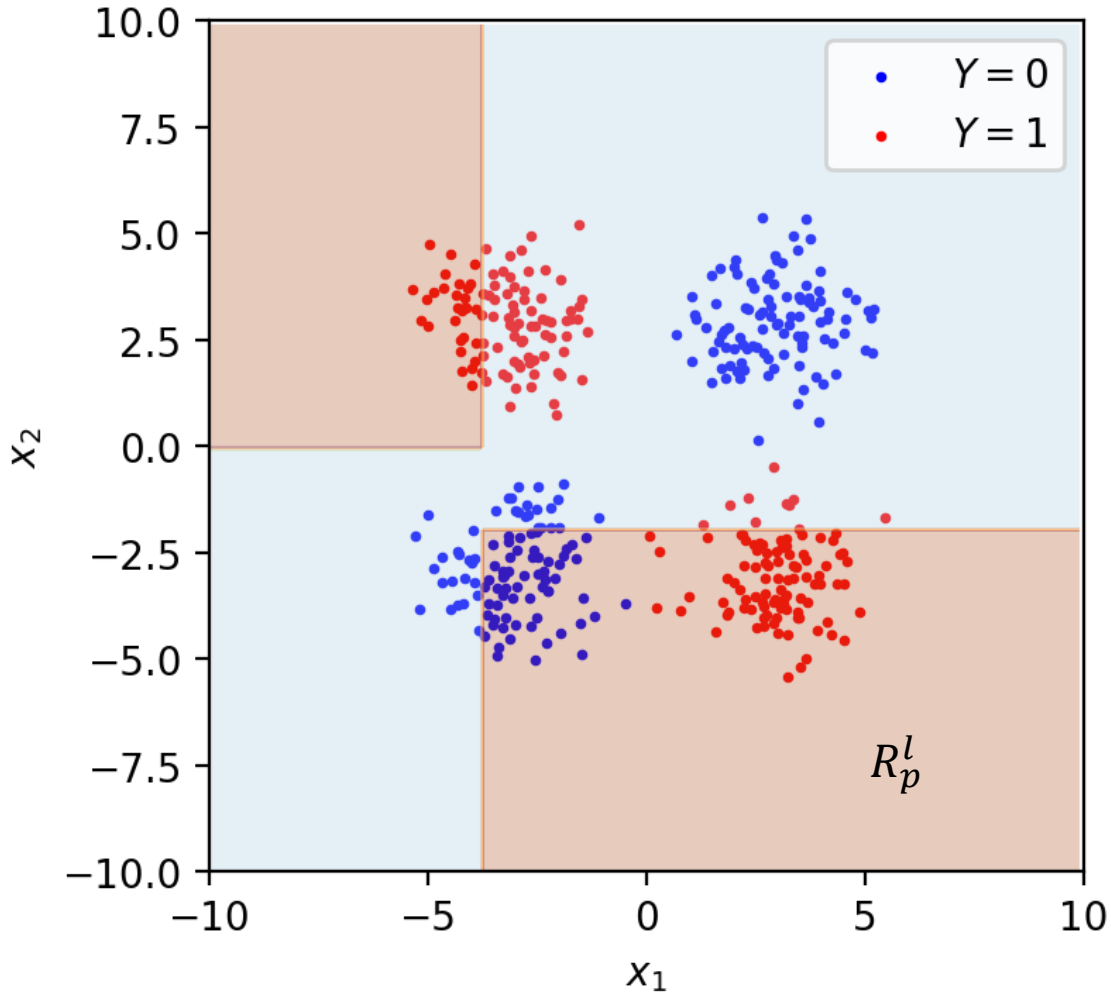
На рисунке: деление выборки всех примеров R_p^0 на третьей итерации ветвления. Внимание: на рисунке – тестовая выборка, правила деления (номер признака j_l и пороговое значение $t^{(l)}$) были определены во время обучения.

DT в режиме обучения

На рисунке – результат после l-го ветвления **тренировочной** выборки.

Цель: осуществить деление подвыборки тренировочных примеров R_p^l.

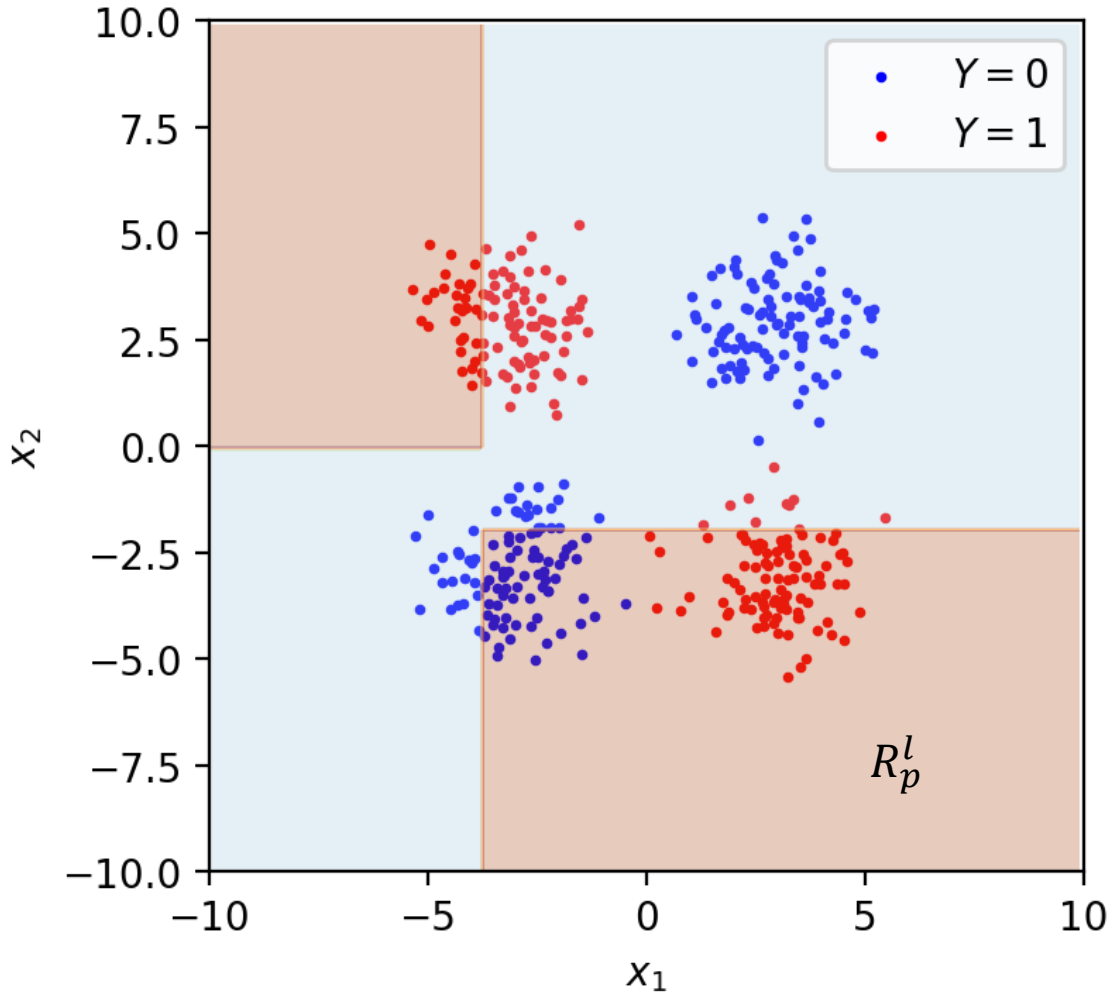
- Если в R_p^l - примеры только одного класса, - нет смысла их делить: в обеих областях после разделения будут присваиваться те же самые метки, что и в R_p^l; в этом случае разделение не производится, в текущей ветке останавливается ветвление.
- Для ветвления $s(j_l, t^{(l)})$ следует выбрать номер признака j_l и пороговое значение $t^{(l)}$, исходя из каких-то соображений. **КАКИХ?**



DT в режиме обучения

На рисунке – результат после l-го ветвления **тренировочной** выборки.
Цель: осуществить деление подвыборки тренировочных примеров R_p^l.

- Для ветвления $\boldsymbol{s(j_l, t^{(l)})}$ следует выбрать **номер признака** $\boldsymbol{j_l}$ и **пороговое значение** $\boldsymbol{t^{(l)}}$, исходя из оптимизации приращения функции потерь. В результате ветвления суммарная функция должна уменьшиться как можно сильнее.



ИДЕЯ функции потерь: это ф-я, которая должна характеризовать качество классификации в листе R. Напомним, что всем примерам, оказавшимся в листе, присваивается одинаковый класс, определяемый голосованием по классам тренировочных примеров в этом листе.

Обозначим: $\boldsymbol{p_c^{(R)}}$ - доля обучающих примеров класса $\boldsymbol{c}$ в листе $\boldsymbol{R}$
Тогда класс, присваиваемый примерам в этом листе на этапе исполнения:

$$\widehat{c_R} = \underset{\mathbb{Y}}{\operatorname{argmax}}\, p_c^{(R)}$$

DT в режиме обучения

ИДЕЯ функции потерь: это ф-я, которая должна характеризовать качество классификации в листе R. Напомним, что всем примерам, оказавшимся в листе, присваивается одинаковый класс, определяемый голосованием по классам тренировочных примеров в этом листе.

Обозначим: $\boldsymbol{p_c^{(R)}}$ - доля обучающих примеров класса $\boldsymbol{c}$ в листе R (может вычисляться с учетом весов примеров $\{w_i\}$)

Тогда класс, присваиваемый примерам в этом листе на этапе исполнения:

$$\widehat{c_R} = \underset{c \in \mathbb{Y}}{\operatorname{argmax}}\, p_c^{(R)}$$

Варианты функции потерь:

- Доля неверно классифицированных обучающих примеров:

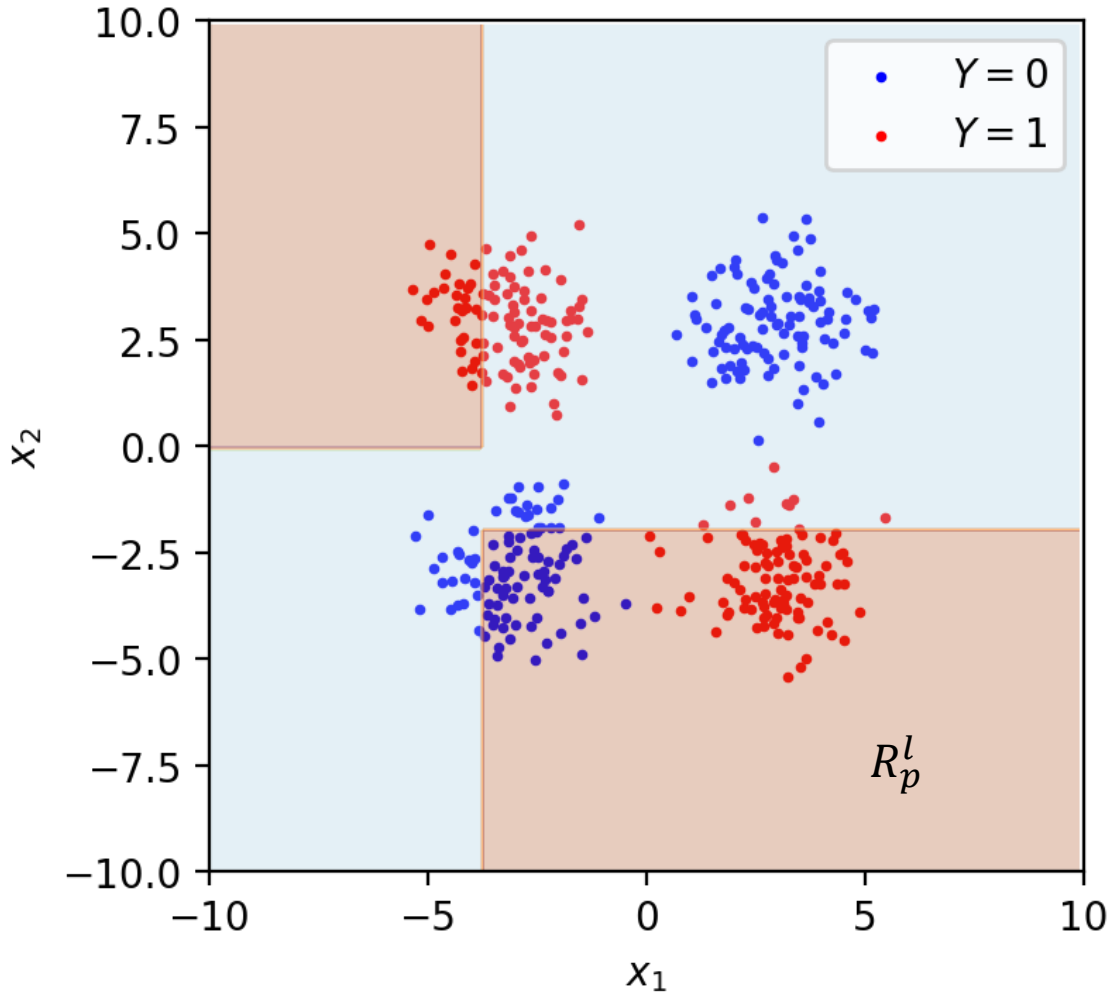
$$\mathcal{L}_{mc} = 1 - \max_{c \in \mathbb{Y}} p_c^{(R)}$$

- Коэффициент Джини (отражает степень непохожести классов в R):

$$\mathcal{L}_{gini} = \sum_{c \in \mathbb{Y}} p_c^{(R)}(1 - p_c^{(R)}) = 1 - \sum_{c \in \mathbb{Y}} {p_c^{(R)}}^2$$

- Перекрестная энтропия:

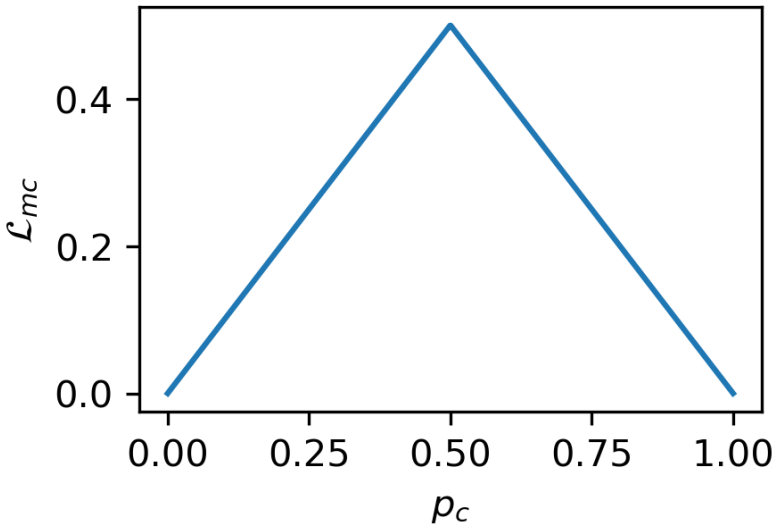
$$\mathcal{L}_{ce} = -\sum_{c \in \mathbb{Y}} p_c^{(R)} \log p_c^{(R)}$$



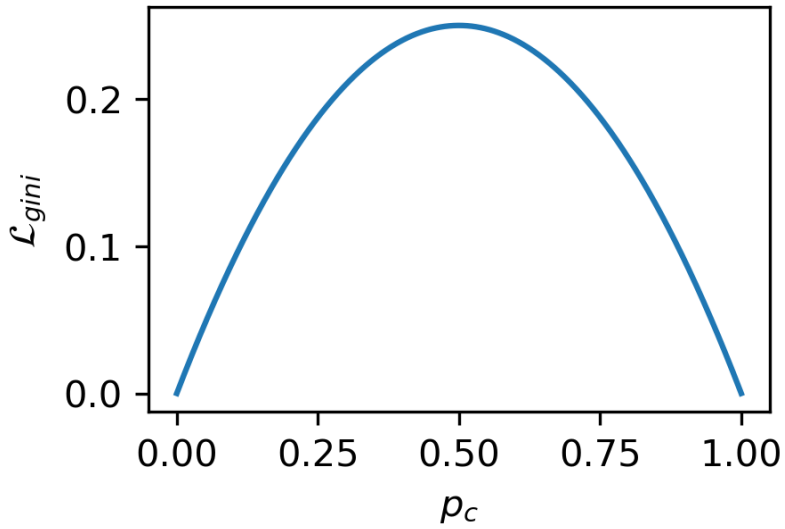
Общее в этих функциях: чем более однородна подвыборка R в смысле классов обучающих примеров, тем меньше значение функции.
Альтернативно: чем больше доля класса, по которому определяется метка для всех примеров из R, тем меньше значение функции.

DT в режиме обучения

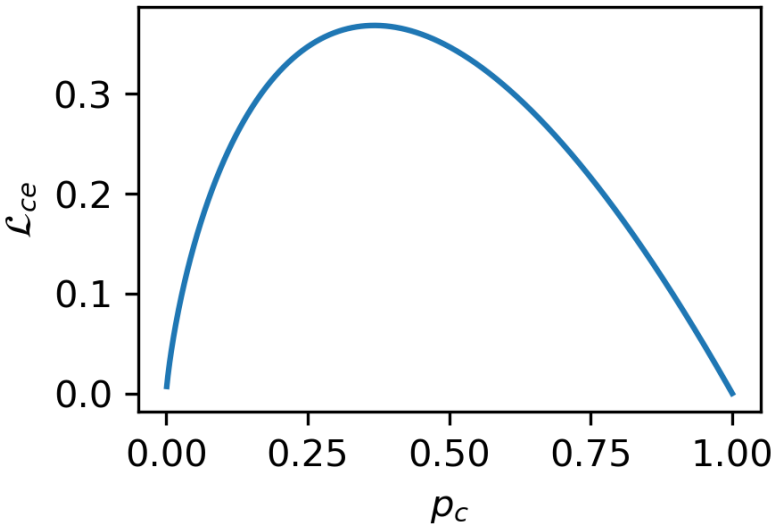
$$\mathcal{L}_{mc} = 1 - \max_{c \in \mathbb{Y}} p_c^{(R)}$$



$$\mathcal{L}_{gini} = \sum_{c \in \mathbb{Y}} p_c^{(R)}(1 - p_c^{(R)})$$



$$\mathcal{L}_{ce} = -\sum_{c \in \mathbb{Y}} p_c^{(R)} \log p_c^{(R)}$$

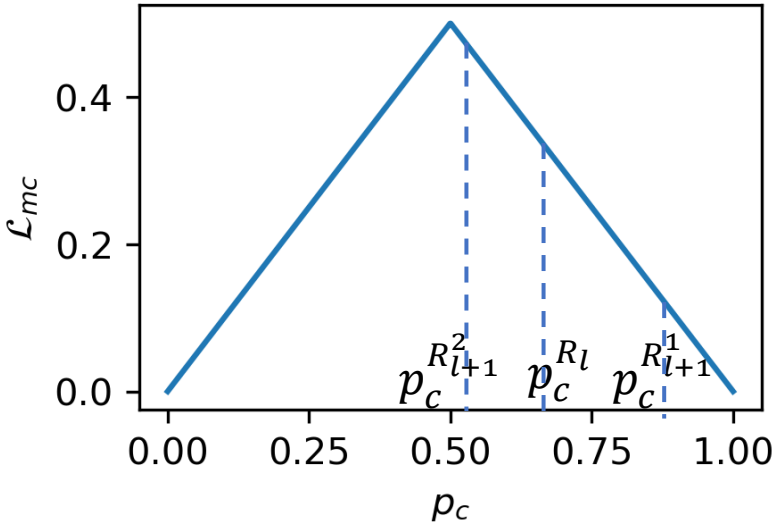


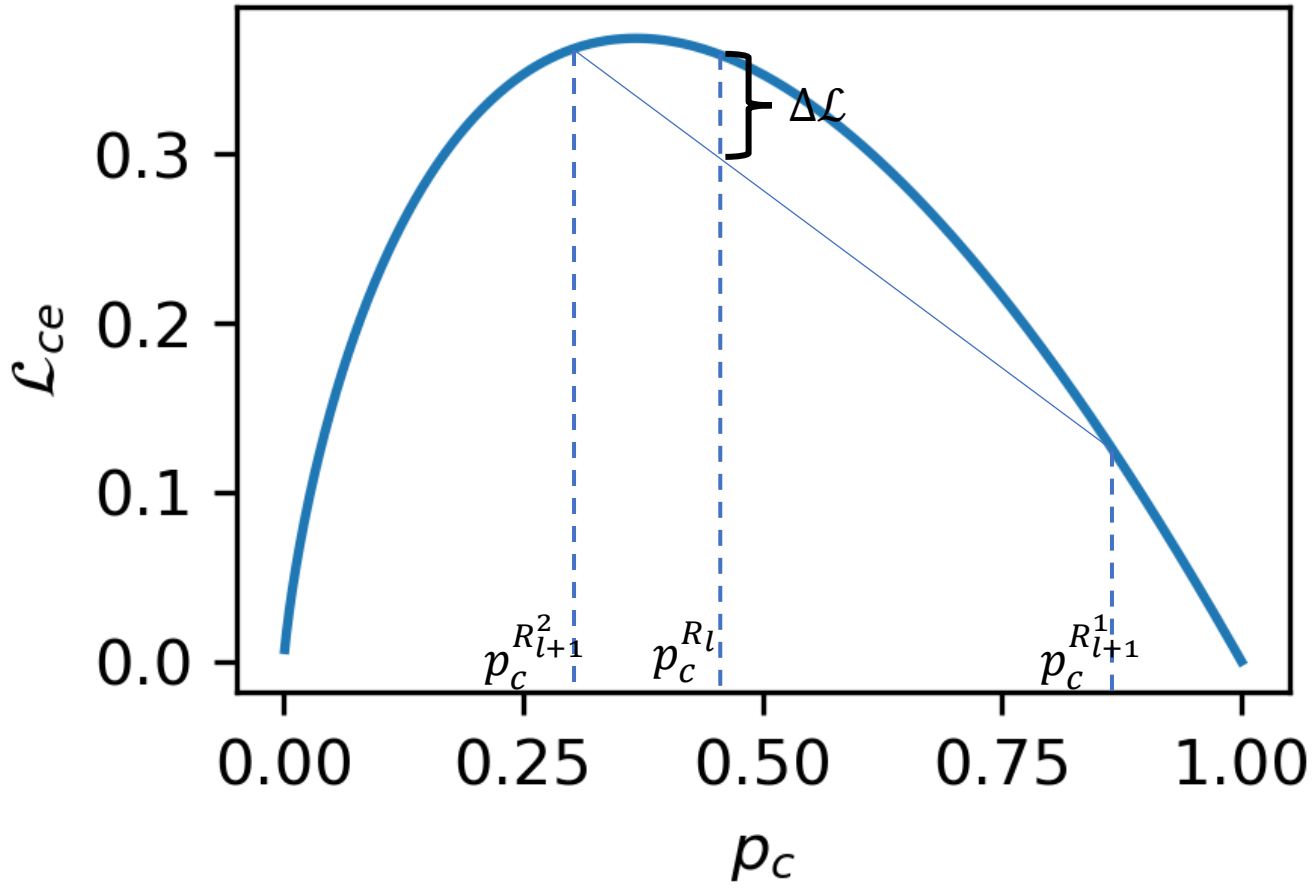
DT в режиме обучения

Разделение обучающей подвыборки R_p^l приводит к тому, что суммарная функция потерь снижается на величину $\Delta\mathcal{L}$.

$$\mathcal{L}_{ce} = -\sum_{c\in\mathbb{Y}} p_c^{(R)} \log p_c^{(R)}$$

Заметим, что в случае функции потерь, характеризующей долю неверно классифицированных примеров, снижение суммарной функции потерь может быть нулевым => $\mathcal{L}_{mc}$ - не лучший вариант функции потерь для настройки деревьев решений.





DT в режиме обучения

На рисунке – результат после l-го ветвления **тренировочной** выборки.
Цель: осуществить деление подвыборки тренировочных примеров R_p^l.

- Для ветвления $\boldsymbol{s(j_{l+1}, t^{(l+1)})}$ следует выбрать **номер признака** $\boldsymbol{j_{l+1}}$ и **пороговое значение** $\boldsymbol{t^{(l+1)}}$, исходя из оптимизации приращения функции потерь. В результате ветвления суммарная функция должна уменьшиться как можно сильнее.

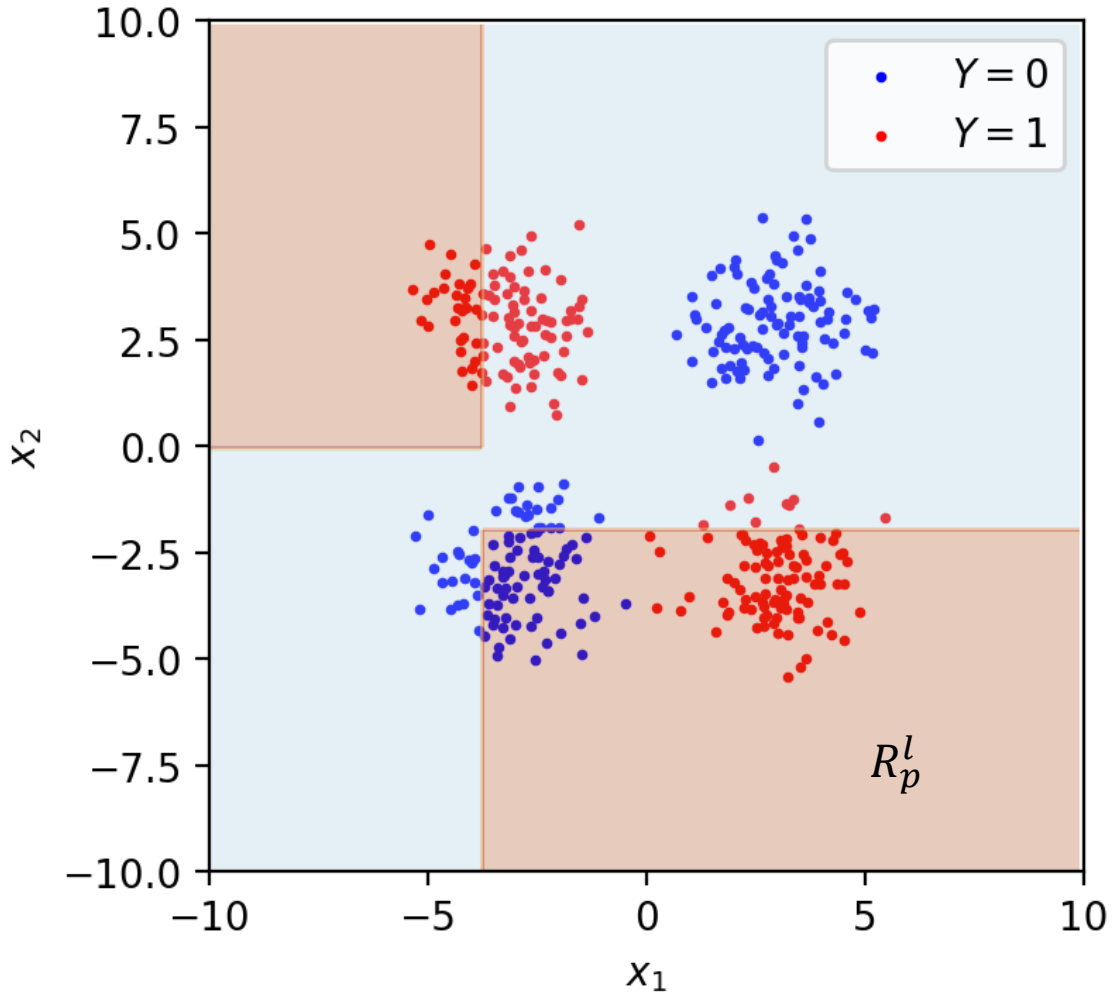
Разделение обучающей подвыборки R_p^l приводит к тому, что суммарная функция потерь снижается на величину $\Delta\mathcal{L}$.

Это означает, что можно искать разделение $l+1$ как решение задачи оптимизации:

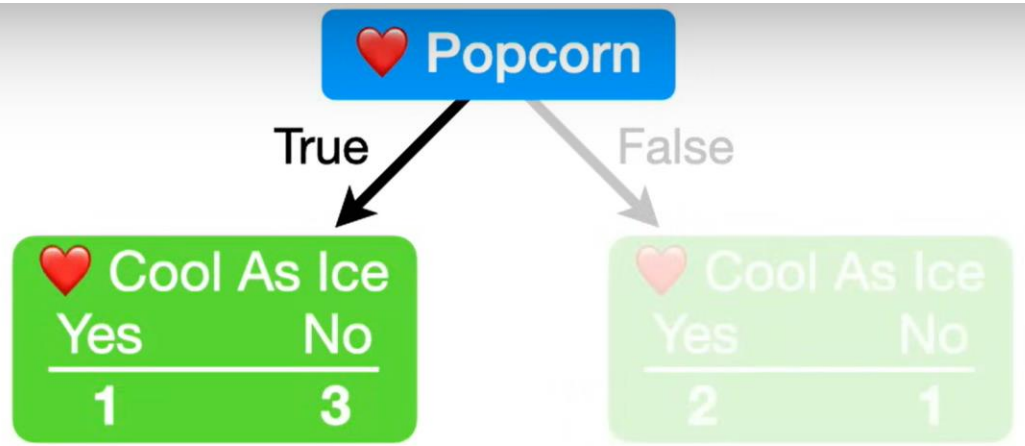
$$j_{l+1}, t^{(l+1)} = \underset{\mathrm{j}\in[1\ldots f], t_j\in\mathbb{X}_\mathrm{j}}{\operatorname{argmax}} \Delta\mathcal{L}(R_p^l, R_{c1}^l, R_{c2}^l)$$

- f – количество признаков признакового описания объектов
- пороговое значение t_j ищется среди всех возможных значений j-го признака

это – т.н. «жадный» (greedy) подход: получение локально оптимального решения на каждой итерации.



Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

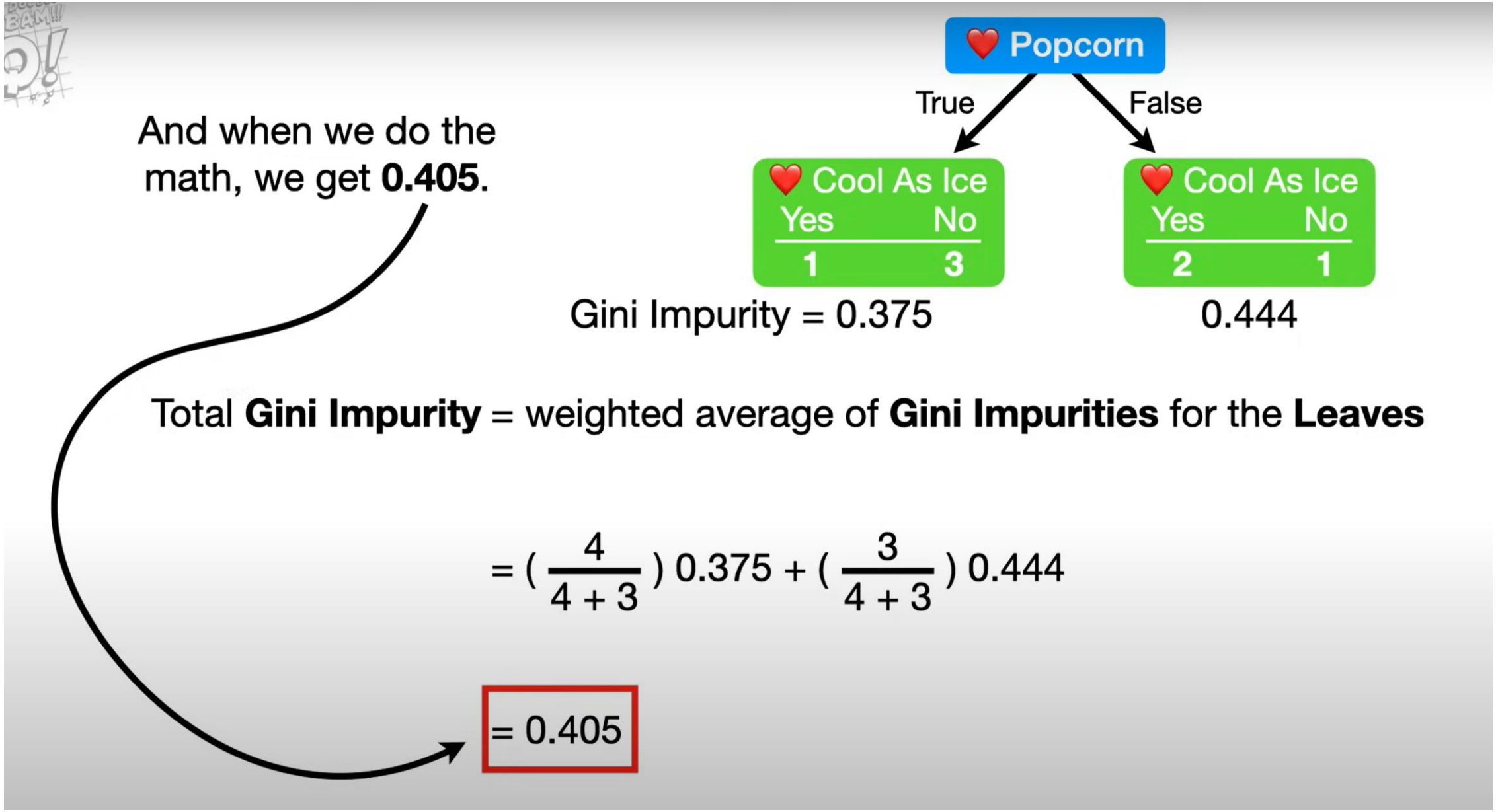


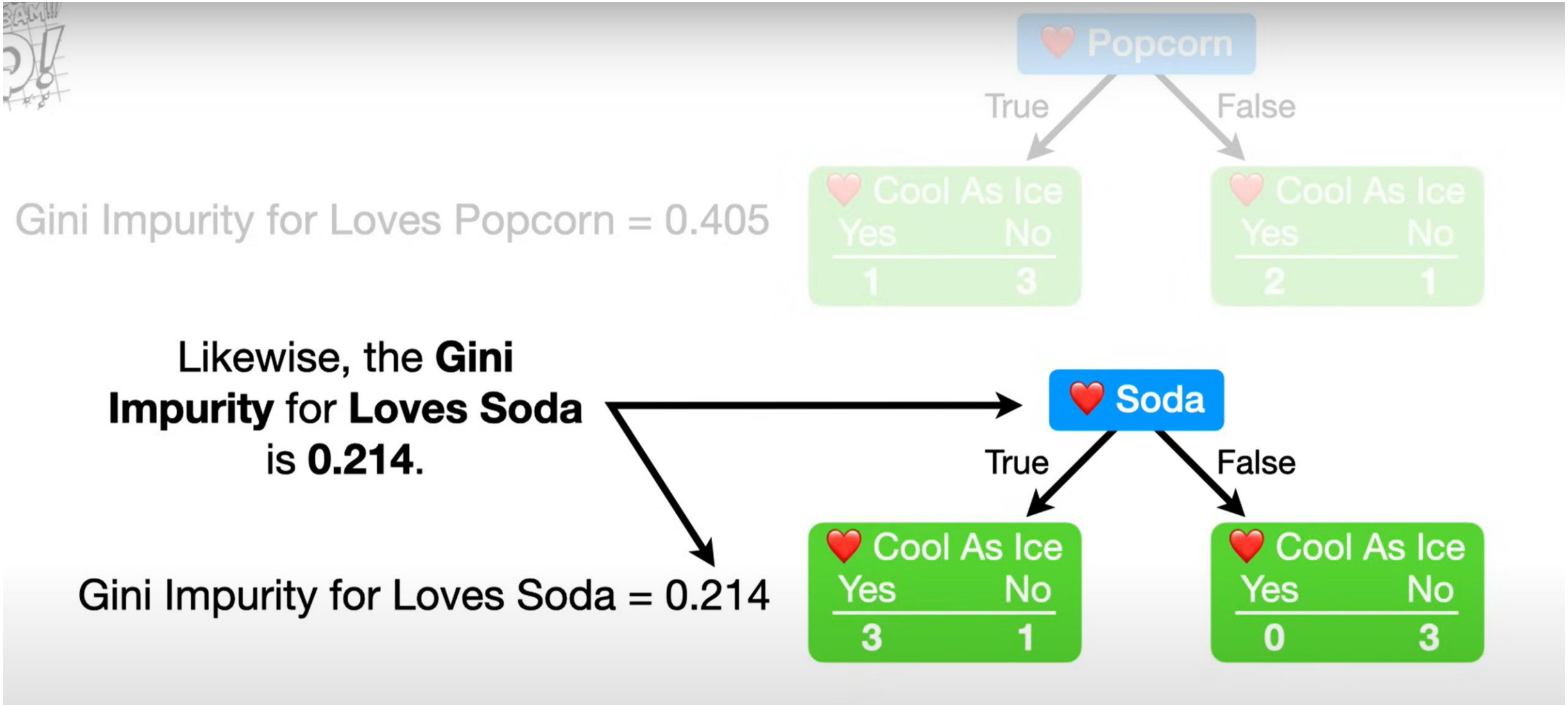
Gini Impurity for a Leaf = 1 - (the probability of "Yes")^2 - (the probability of "No")^2

$$= 1 - \left(\frac{1}{1+3}\right)^2 - \left(\frac{3}{1+3}\right)^2$$

And when we do the math, we get **0.375**.

https://www.youtube.com/@statquest


And when we do the math, we get **0.405**.
Popcorn
True
False
Cool As Ice
Yes No
1 3
Cool As Ice
Yes No
2 1
Gini Impurity = 0.375
0.444
Total **Gini Impurity** = weighted average of **Gini Impurities** for the **Leaves**
$= \left(\frac{4}{4+3}\right) 0.375 + \left(\frac{3}{4+3}\right) 0.444$
$= 0.405$


Popcorn
True
False
Cool As Ice
Yes No
1 3
Cool As Ice
Yes No
2 1
Gini Impurity for Loves Popcorn = 0.405
Likewise, the Gini Impurity for Loves Soda is 0.214.
Soda
True
False
Cool As Ice
Yes No
3 1
Cool As Ice
Yes No
0 3
Gini Impurity for Loves Soda = 0.214

Lastly, we calculate the **Gini Impurity** values for each average age.

Age	Loves Cool As Ice
7	No
12	No
18	Yes
35	Yes
38	Yes
50	No
83	No

9.5 → Gini Impurity = 0.429

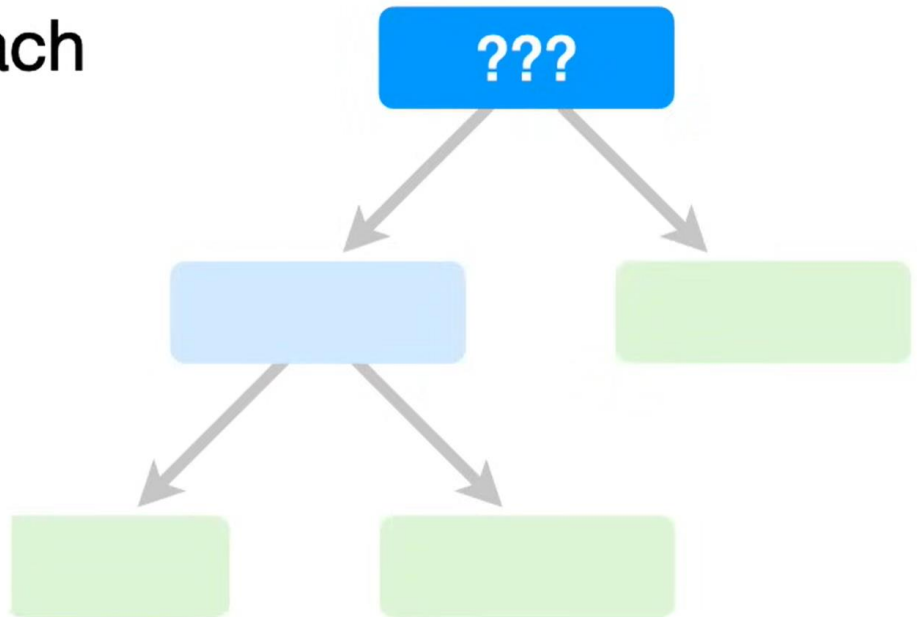
15 → Gini Impurity = 0.343

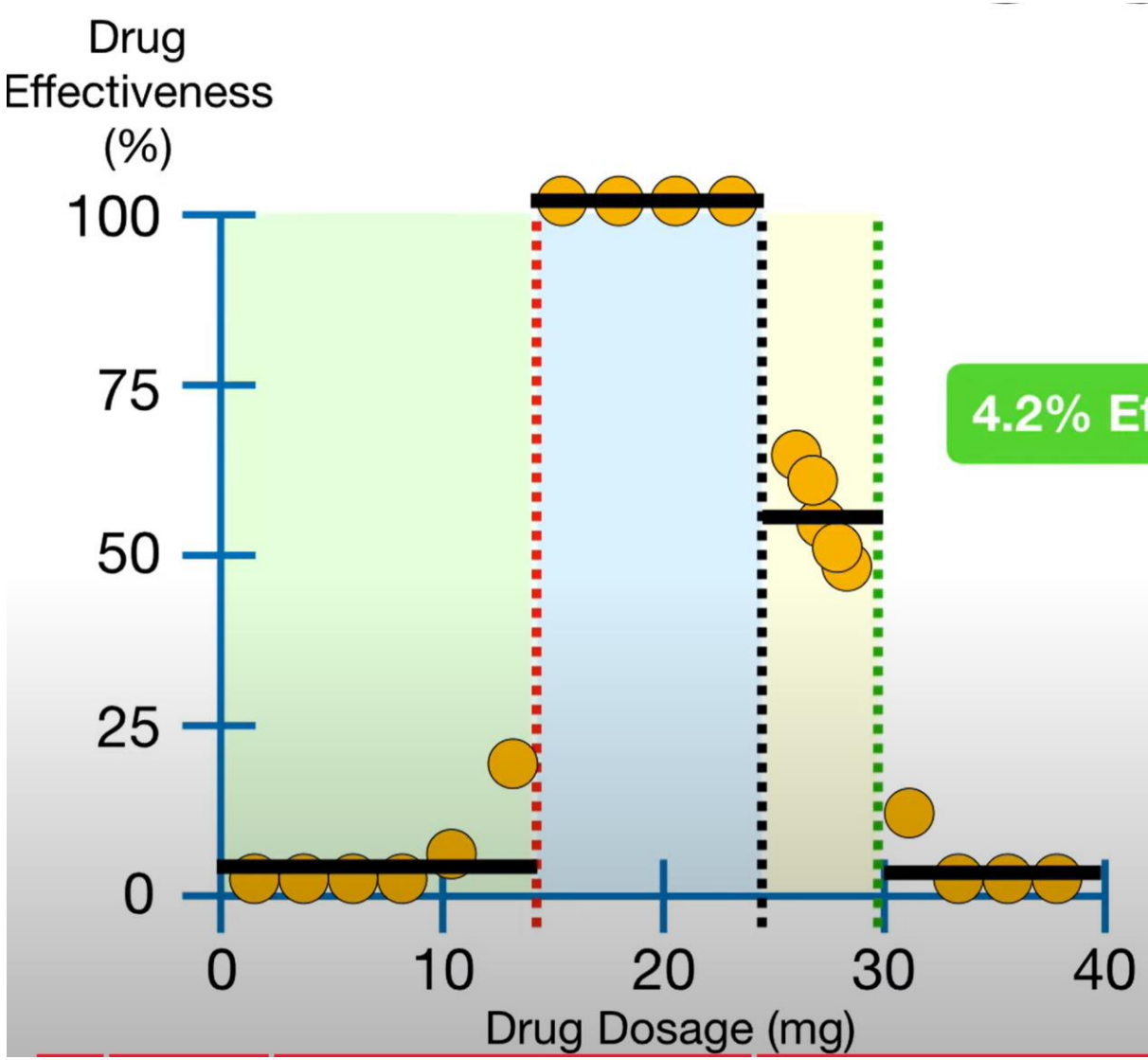
26.5 → Gini Impurity = 0.476

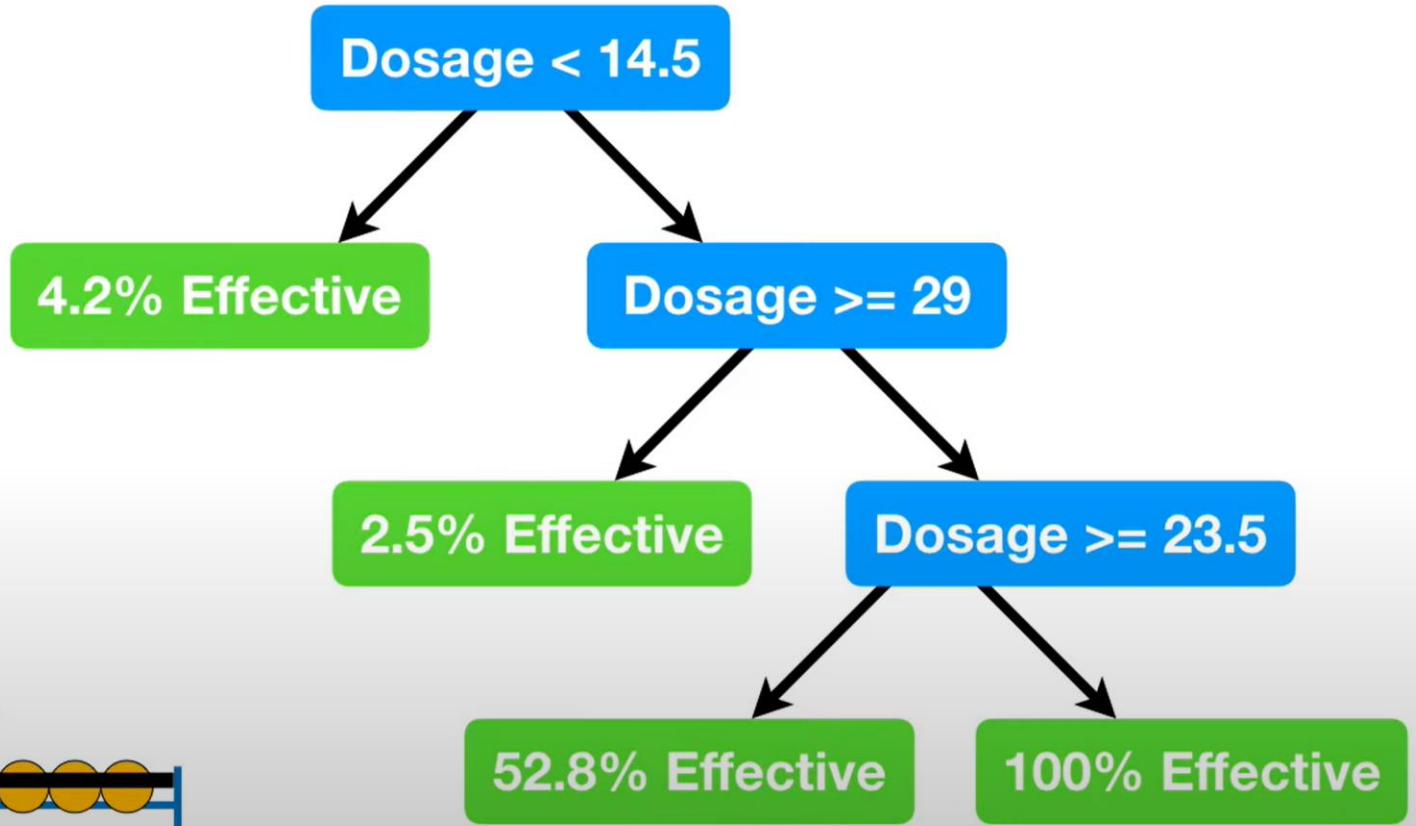
36.5 → Gini Impurity = 0.476

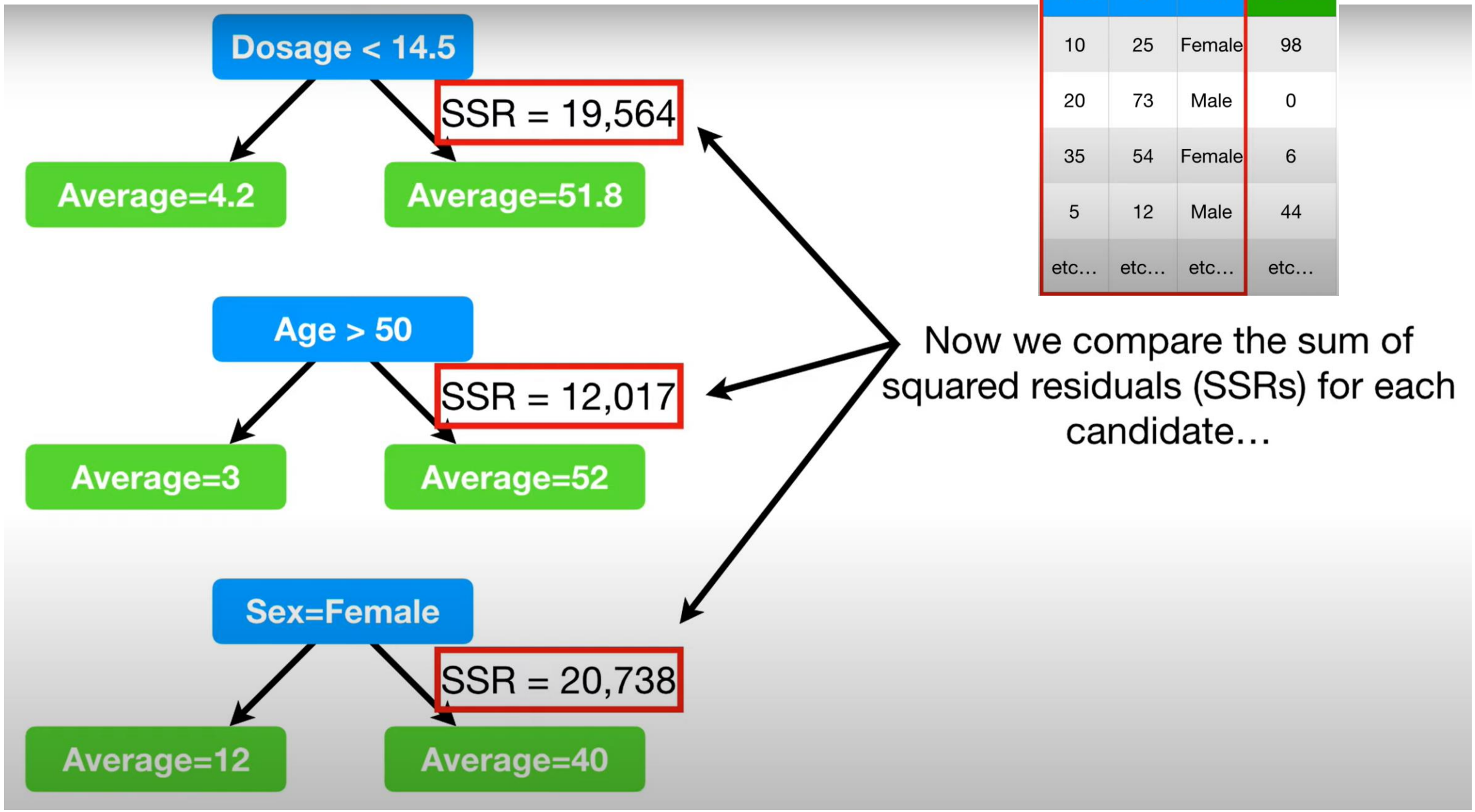
44 → Gini Impurity = 0.343

66.5 → Gini Impurity = 0.429




Drug Effectiveness (%)
100
75
50
25
0
0
10
20
30
40
Drug Dosage (mg)


Dosage < 14.5
4.2% Effective
Dosage >= 29
2.5% Effective
Dosage >= 23.5
52.8% Effective
100% Effective


Dosage < 14.5
SSR = 19,564
Average=4.2
Average=51.8
Age > 50
SSR = 12,017
Average=3
Average=52
Sex=Female
SSR = 20,738
Average=12
Average=40
Dosage
Age
Sex
Drug Effect.
10
25
Female
98
20
73
Male
0
35
54
Female
6
5
12
Male
44
etc…
etc…
etc…
etc…
Now we compare the sum of squared residuals (SSRs) for each candidate…

Ансамбли моделей

деревья решений **сильно склонны к переобучению. НИКОГДА НЕ ПРИМЕНЯЙТЕ** деревья решений как таковые!
Способ борьбы с этой особенностью – ансамблирование моделей.

Виды ансамблей:

- **Weighted averaging** (взвешенное осреднение): обучить K различных методов («базовых алгоритмов») на одних и тех же данных; результат взвешенно осреднять (в случае регрессии) или получать взвешенным голосованием (в случае классификации):

$$\widehat{y_i} = \frac{1}{\sum_i w_i} \sum_{k=1}^{K} w_k y_i^{(k)}$$

$$\hat{c}_i = \underset{c \in \mathbb{Y}}{\operatorname{argmax}} \sum_{k=1}^{K} w_k * [c_i^{(k)} == c]$$

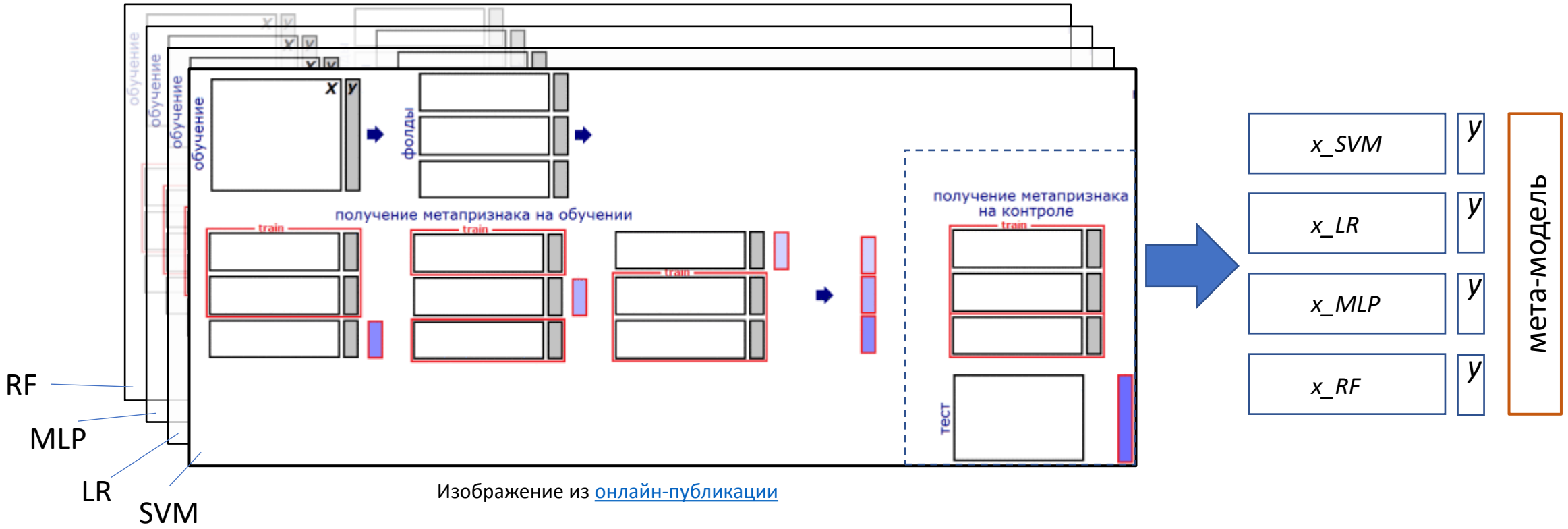
если все веса w_k равны 1, получим простое голосование/осреднение; в случае, когда веса зависят от x (а значит обучаемые) – получим т.н. «смесь экспертов» (blending)

- **Stacking** (стекинг): обучить K различных базовых алгоритмов на одних и тех же данных; вывод каждого из алгоритмов (значения параметров целевого распределения μ_i или p_i) использовать как новые признаки для новой мета-модели (обычно довольно простого, напр., любой из вариантов GLM/GAM: линейная регрессия в случае регрессии или логистическая регрессия в случае классификации);
- Обучать один и тот же базовый алгоритм на K полностью различных тренировочных выборках; результаты взвешенно осреднять (см. выше) или использовать эти K моделей в подходе стекинга. При этом рассчитывать на то, что каждой из этих выборок достаточно для обучения модели; все они порождены из одного и того же распределения. Однако набирать две или больше достаточно объемные тренировочные выборки – дорого и долго;
- **Random Subspace Method** (метод случайных подпространств): обучать K базовых алгоритмов (различных или одинаковых)
- **Bagging** (**B**ootstrap **Agg**regat**ing**, аггрегирование в подходе бутстрэп) – см. далее;
- **Boosting** («бустинг») – см. далее.

Ансамбли моделей: stacking

Идея:

1. обучить несколько базовых алгоритмов, каждый из которых где-то хорошо работает, а где-то систематически ошибается, получать этими моделями т.н. «метапризнаки»;
2. агрегировать результаты еще одной (тоже обучаемой) моделью – «метамоделью».



Изображение из [онлайн-публикации]

Ансамбли моделей: bagging

Bagging (**B**ootstrap **Agg**regat**ing**, аггрегирование в подходе бутстрэп)

Идея:

1. обучить множество базовых алгоритмов, склонных к переобучению, на подвыборках, гарантированно порожденных одним и тем же распределением (identically distributed, “i.d.”);
2. агрегировать результаты в подходе простого голосования/осреднения.

Сэмплирование из тренировочной выборки в подходе Bootstrap гарантирует* идентичность порождающего распределения**. В случае ограниченного количества выборок Bootstrap предоставляет лучшее из доступных приближений***.

Размер каждой выборки bootstrap:

- в случае сильно ограниченного размера тренировочной выборки – берут размером с тренировочную;
- в случае большого тренировочного набора данных размер bootstrap-выборки – гиперпараметр, подбирается по качеству на валидационной выборке.

Почему вообще ансамблирование одинаковых переобучающихся алгоритмов может работать, если сам алгоритм «плохой»?

* в пределе бесконечного количества выборок

** в смысле статистик, оцениваемых эмпирически

*** Efron B. Bootstrap Methods: Another Look at the Jackknife Springer Series in Statistics / под ред. S. Kotz, N.L. Johnson, New York, NY: Springer, 1992. 569–593 с.

Ансамбли моделей: bagging

Bagging (**B**ootstrap **Agg**regat**ing**, аггрегирование в подходе бутстрэп)

Почему вообще ансамблирование K одинаковых переобучающихся алгоритмов может работать, если сам алгоритм «плохой»?

Оценка целевой переменной (точнее, какого-то параметра распределения $P(y|x)$, например, $\mu(x)$) – тоже случайная величина (обозначим T), с определенным распределением $P(T)$.

Обычно (в предположении, что $T_1, T_2, \dots T_K$ – оценки K разными алгоритмами, i.i.d.* случайные величины):

$$Var(T_1) = Var(T_2) = \cdots = Var(T_n) = \sigma^2$$

тогда

$$Var(\bar{T}) = Var\left(\frac{1}{K}\sum_{k=1}^{K} T_k\right) = \frac{\sigma^2}{K}$$

Представим, что эти случайные величины – не независимы (в случае обучения K одинаковых алгоритмов на bootstrap-выборках уже нельзя говорить о независимости результатов). Например (простейший вариант), попарные корреляции между ними одинаковы и составляют ρ:

$$\rho = \frac{Cov(T_j, T_i)}{\sigma_{T_i}\sigma_{T_j}}$$

$$Cov(T_i, T_i) = Var(T_i) = \sigma^2$$

Тогда:

$$Var(\bar{T}) = Var\left(\frac{1}{K}\sum_{k} T_k\right) = \frac{1}{K^2}\sum_{i,j} Cov(T_i, T_j) = K\frac{\sigma^2}{K^2} + \frac{K(K-1)}{K^2}\rho\sigma^2 = \rho\sigma^2 + \frac{1-\rho}{K}\sigma^2$$

*i.i.d. – independent, idetically distributed

Ансамбли моделей: bagging

Bagging (**B**ootstrap **Agg**regat**ing**, аггрегирование в подходе бутстрэп)

$$Var(\bar{T}) = \rho\sigma^2 + \frac{1-\rho}{K}\sigma^2$$

где T – случайная переменная оценки параметра условного распределения $P(y|x)$ целевой переменной (μ для регрессии, p для классификации)
$Var(\bar{T})$ - дисперсия средней оценки этого параметра при ансамблировании K одинаковых алгоритмов при смягчении предположения о независимости их ответов (например, при обучении на пересекающихся bootstrap-выборках)

Выводы: для снижения дисперсии (неопределенности) ответов ансамбля

- следует повышать К – количество членов ансамбля
- следует снижать ρ, характеризующую степень их скоррелированности – делать результаты базовых алгоритмов как можно менее похожими

Bagging эксплуатирует подход обучения большого количества ($K \gg 1$) моделей, склонных к переобучению (σ^2 - существенна, но ρ сильно меньше единицы, алгоритмы раскоррелированы за счет склонности к переобучению и за счет обучения на различающихся подвыборках).

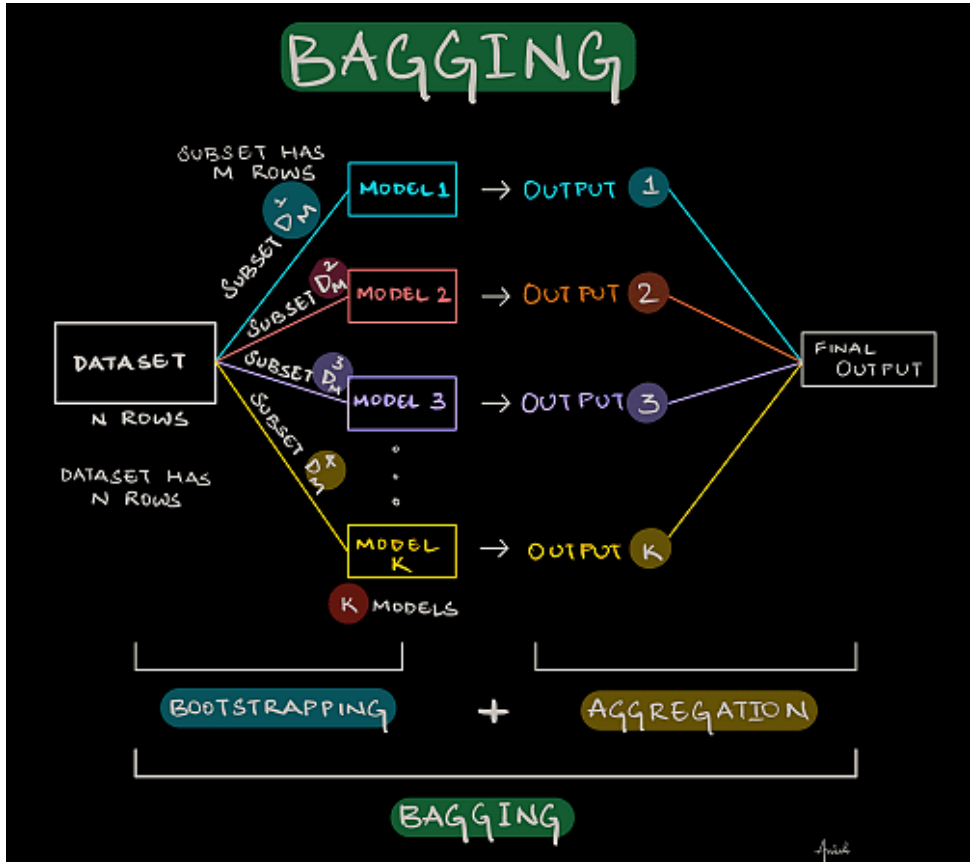
Способ применения в случае решающих деревьев: обучить очень много довольно решающих деревьев до конца (не ограничивая их глубину, без регуляризаций); обучать на bootstrap-выборках, агрегировать результаты по принципу простого голосования (в случае классификции) или простого осреднения (в случае регрессии).

Ансамбли моделей: Random Forests

Bagging + Random Subspace Method =* Random Forests**

$$Var(\bar{T}) = \rho\sigma^2 + \frac{1-\rho}{K}\sigma^2$$

Идея метода Случайных Лесов: снизить корреляцию ρ результатов базовых алгоритмов еще больше (по сравнению с подходом бэггинга над решающими деревьями) за счет выбора случайных подпространств (совокупности признаков), на которых ищется оптимальное ветвление на итерациях обучения решающих деревьев.



Псевдоалгоритм обучения случайных лесов:

Начальное состояние:
Выборка $R = \{x_i, y_i\}$; множество признаков x_i – F; кол-во деревьев в композиции B (задается исследователем); множество базовых алгоритмов H – пустое.

ФУНКЦИЯ RANDOM_FOREST(R):

Повторять B раз:

1. $R_k = bootstrap(R)$
2. $h_k = RANDOMIZED\_TREE(R_k)$
3. $H = H \cup h_k$

Возврат H

ФУНКЦИЯ RANDOMIZED_TREE(R_k):

В каждом узле ветвления:

1. f – небольшое подмножество признаков F
2. поиск оптимального разделения $s(j_l, t^{(l)})$ только среди признаков из подмножества f

Возврат обученного дерева

* Не совсем так. См. псевдоалгоритм случайных лесов

** Breiman L. Random Forests // Machine Learning. 2001. № 1 (45). С. 5–32.

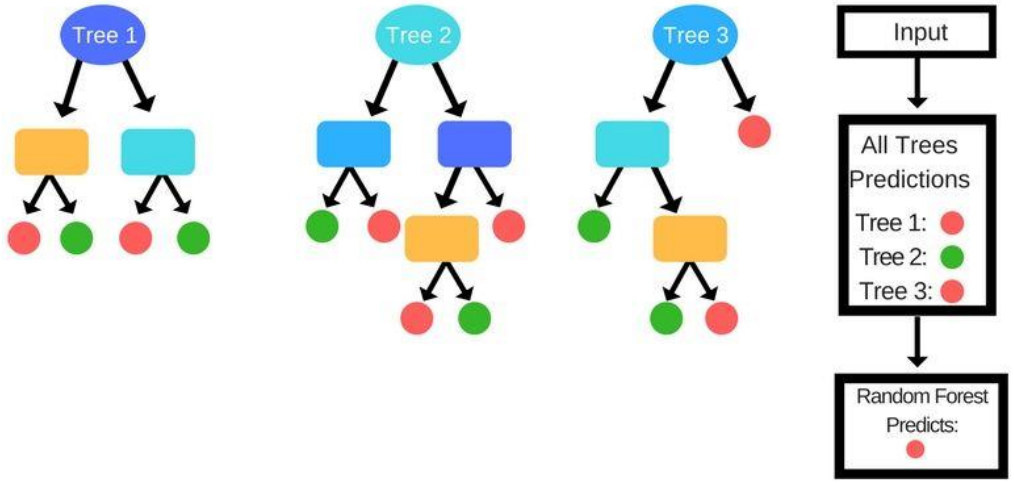
Ансамбли моделей: Bagging, Random Forests

$$Var(\bar{T}) = \rho\sigma^2 + \frac{1-\rho}{K}\sigma^2$$

Особенности бэггинга (включая RF):

+ Возможность снижения дисперсии оценки параметра распределения целевой переменной, применяя слабые базовые алгоритмы (для RF это свойство выражено еще сильнее)
+ => снижение неопределенности решения
+ => повышение точности решения
+ «бесплатная» валидация – оценка качества на OOB-выборках, получаемых при bootstrap-сэмплировании
+ сниженная чувствительность к выбросам и отсутствующим значениям (за счет применения метода случайных подпространств, за счет bootstrap-сэмплирования)
+ повышение количества членов ансамбля не приводит к переобучению, зато приводит к снижению дисперсии ответов

- слишком низкая выразительная способность членов ансамбля может приводить к низкой выразительной способности всей композиции
- решение ансамбля сложнее интерпретировать (по сравнению с GLM/GAM или DT)
- более вычислительно затратны при обучении по сравнению с GLM/GAM или DT

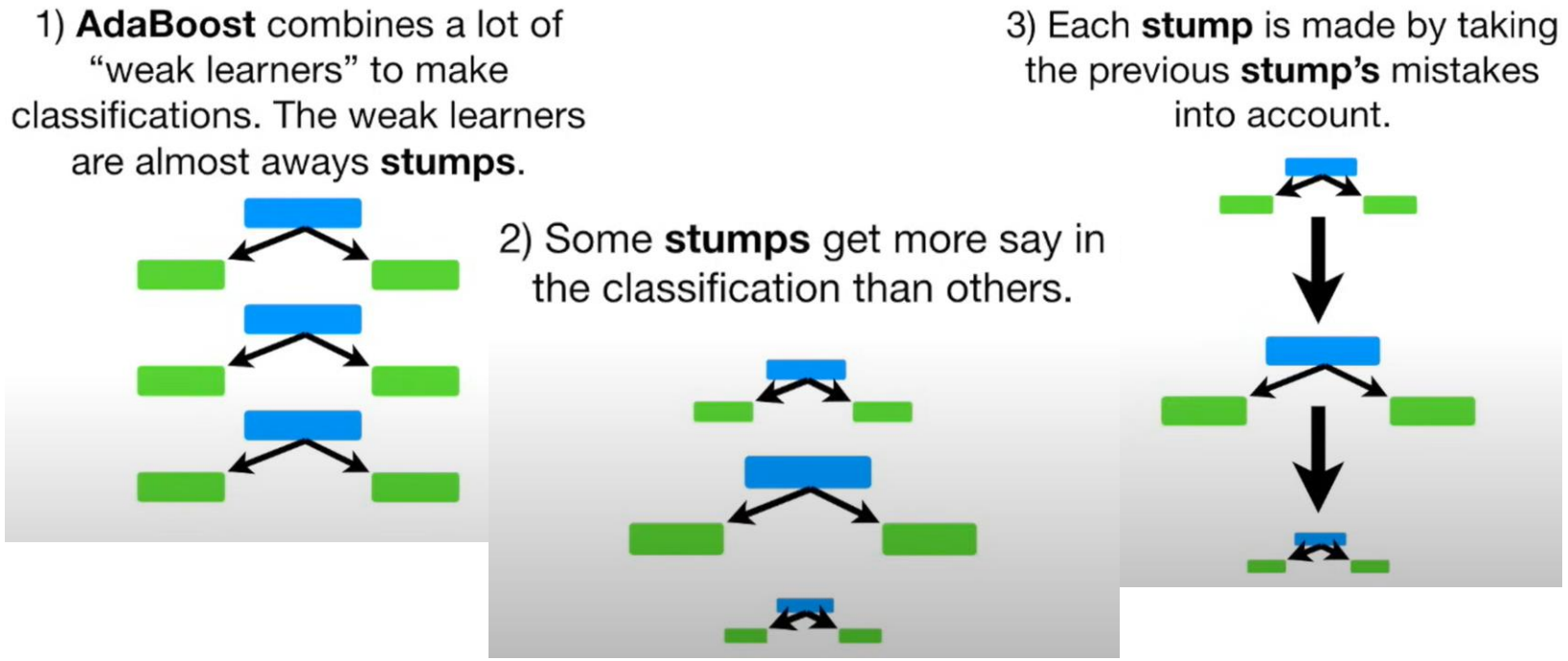


Ансамбли моделей: boosting

Идея:

Обучать слабые базовые алгоритмы (“weak classifier/regressor”) с низкой выразительной способностью – последовательно, с итерационным изменением весов w_i примеров x_i тренировочной выборки. Веса примеров модифицировать, руководствуясь ошибками композиции, построенной к этой итерации.

Применение в случае решающих деревьев: отдельные деревья обучают сильно регуляризуя, например, сильно ограничивая глубину (в случае глубины в 1-2 ветвления они называются «решающими пнями», decision stumps)



https://www.youtube.com/@statquest

Ансамбли моделей: boosting

Идея:

Обучать слабые базовые алгоритмы (“weak classifier/regressor”) с низкой выразительной способностью – последовательно, с итерационным изменением весов w_i примеров x_i тренировочной выборки. Веса примеров модифицировать, руководствуясь ошибками композиции, построенной к этой итерации.

Применение в случае решающих деревьев: отдельные деревья обучают сильно регуляризуя, например, сильно ограничивая глубину (в случае глубины в 1-2 ветвления они называются «решающими пнями», decision stumps)

AdaBoost:

Начальное состояние:

Выборка $R = \{x_i, y_i\}$, количество примеров $N = |R|$; Начальные значения весов примеров: $w_i = 1/N$ для всех $i = 1 \dots N$. Количество членов ансамбля K (задается исследователем).

Повторять K раз, $k = 1 \dots K$:

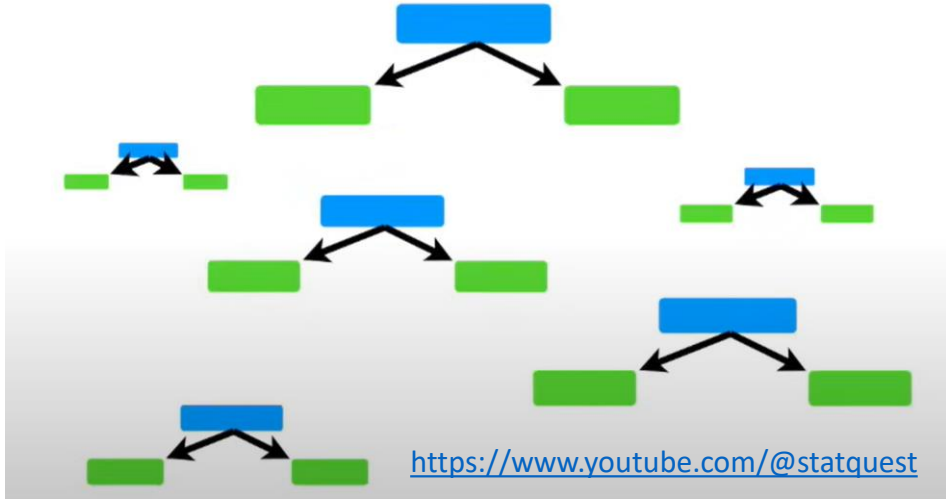
1. Создать и оптимизировать слабый базовый алгоритм g_k на обучающей выборке с учетом весов примеров $\{w_i\}$
2. Вычислить взвешенную ошибку этого классификатора: $err_k = \frac{1}{\sum w_i} \sum w_i * [g_k(x_i) \neq y_i]$
3. Вычислить фактор модификации весов: $\alpha_k = \ln \frac{1-err_k}{err_k}$
4. Адаптировать веса примеров, на которых допущена ошибка: $w_i = w_i * \exp(\alpha_k [g_k(x_i) \neq y_i])$

Итоговый алгоритм – агрегирующая композиция всех обученных слабых алгоритмов с соответствующими весами:

$$F(x) = \sum_k \alpha_k g_k(x)$$

Ансамбли моделей: boosting

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes	Yes	172	No	1/8	0.05	0.07



Ансамбли моделей: gradient boosting*

Идея:

- в подходе бустинга воспринимать построение композиции как задачу градиентной оптимизации в отношении некоторой функции потерь в пространстве функций (базовых алгоритмов):

$$d_k(x_i) = \frac{\partial \mathcal{L}(y, F_k(x_i))}{\partial F_k(x_i)}$$

- на каждом шаге градиентной оптимизации обучается новый слабый алгоритм, аппроксимирующий $g_k(x_i)$ с той точностью, с которой позволяет его выразительная способность:

$$\tilde{d}_k(x_i) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} \big(g(x_i, \gamma) - d_k(x_i)\big)^2$$

(в случае функции ошибки MSE в задаче регрессии)

- в отношении композиции производится итерация градиентной оптимизации с шагом β:

$$F_{k+1}(x) = F_k(x) + \beta \tilde{d}_k(x)$$

Подход градиентного бустинга также называют подходом пошагового аддитивного моделирования (Stepwise Additive Modeling)

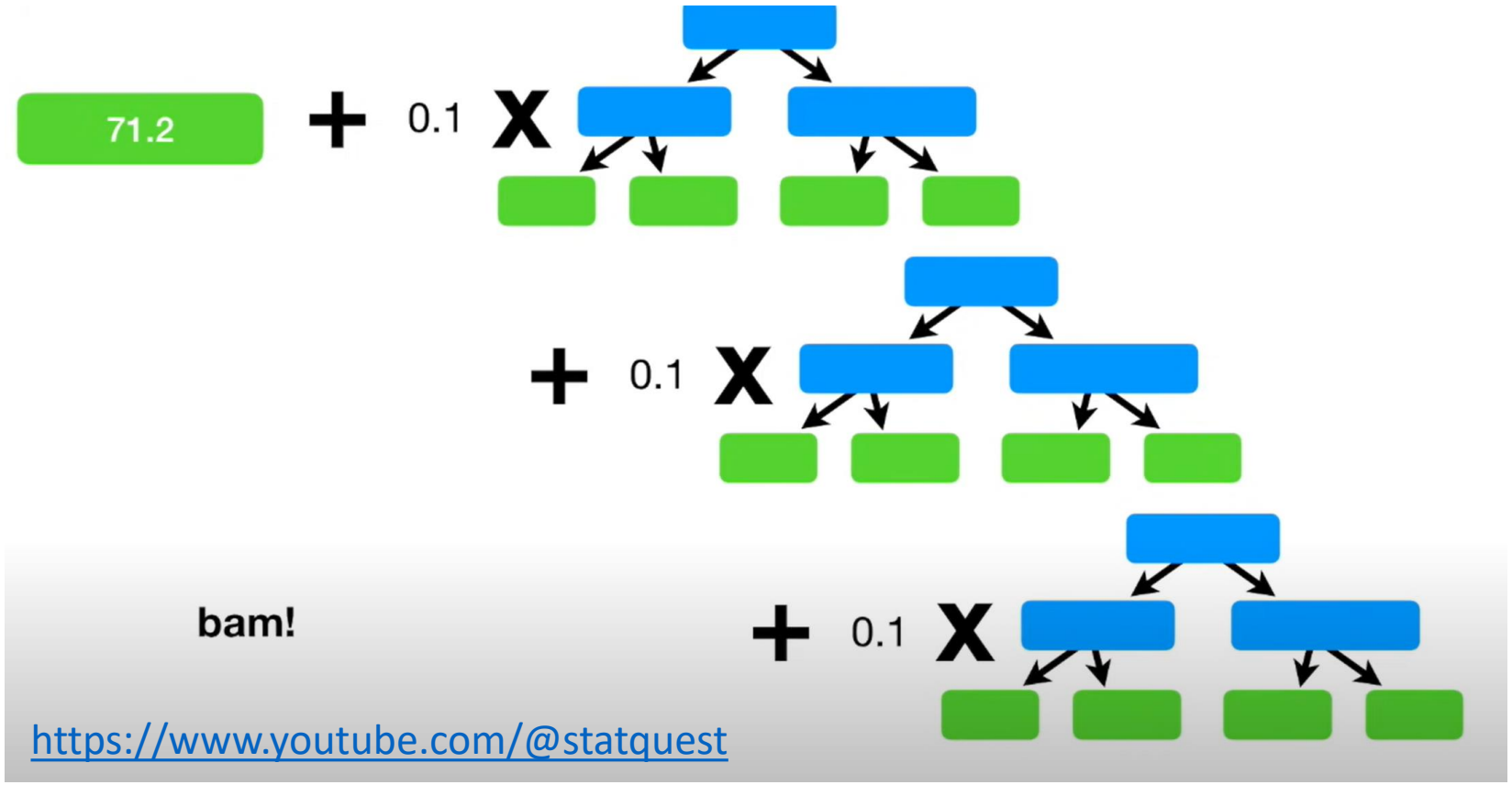
* Breiman L. Technical Report 486, Statistics Department, University of California. ”Arcing the edge”. 1997;
Friedman J.H. Greedy Function Approximation: A Gradient Boosting Machine // The Annals of Statistics. 2001. № 5 (29). С. 1189–1232.

Ансамбли моделей: gradient boosting

Average Weight

71.2

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2



https://www.youtube.com/@statquest

Ансамбли моделей: gradient boosting

Реализации градиентного бустинга над решающими деревьями:

XGBoost (в стандартном составе scikit-learn, поддерживается и развивается усилиями сообщества на принципах open source)

1. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. San Francisco, California, USA: ACM Press, 2016. 785–794 с.
2. https://xgboost.ai/

LightGBM (первоначально разработана и поддерживалась **Microsoft**, сейчас – community efforts, на принципах open source)

1. Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.
2. Qi Meng, Guolin Ke, Taifeng Wang, Wei Chen, Qiwei Ye, Zhi-Ming Ma, Tie-Yan Liu. "A Communication-Efficient Parallel Algorithm for Decision Tree". Advances in Neural Information Processing Systems 29 (NIPS 2016), pp. 1279-1287.
3. Huan Zhang, Si Si and Cho-Jui Hsieh. "GPU Acceleration for Large-scale Tree Boosting". SysML Conference, 2018.
4. https://github.com/microsoft/LightGBM

CatBoost (**Яндекс**, развитие и поддержка на принципах open source)

1. Anna Veronika Dorogush, Andrey Gulin, Gleb Gusev, Nikita Kazeev, Liudmila Ostroumova Prokhorenkova, Aleksandr Vorobev "Fighting biases with dynamic boosting". arXiv:1706.09516, 2017.
2. Anna Veronika Dorogush, Vasily Ershov, Andrey Gulin "CatBoost: gradient boosting with categorical features support". Workshop on ML Systems at NIPS 2017.
3. https://catboost.ai/

Ансамбли моделей: gradient boosting

Особенности алгоритмов бустинга:

+ Наиболее выразительные модели среди всех «классических» методов;
+ Наибольшая точность (в большинстве случаев) среди всех «классических» методов;
+ Наиболее гибкие методы, в практическом/«спортивном» применении (задачи на kaggle.com) чаще всего показывают лучшие результаты среди задач на табличных данных;
+ Можно использовать общий подход, используя другие слабые быстро обучаемые модели
+ Процесс оптимизации параллелизуется, есть ускоренные реализации на GPU, распределенные реализации;

- Склонны к переобучению: могут градиентно «настраиваться» на выбросы/шум, теряя обобщающую способность
- Много гиперпараметров (параметры базовых алгоритмов, параметры градиентной оптимизации, параметры регуляризаций…); настройка гиперпараметров на валидационной выборке может требовать большого количества итераций, что приводит к т.н. «утечке данных» из валидационной выборки в обучение
- Обучение на больших объемах данных может требовать существенных вычислительных ресурсов

Гиперпараметры ансамблевых моделей

RandomForestRegressor

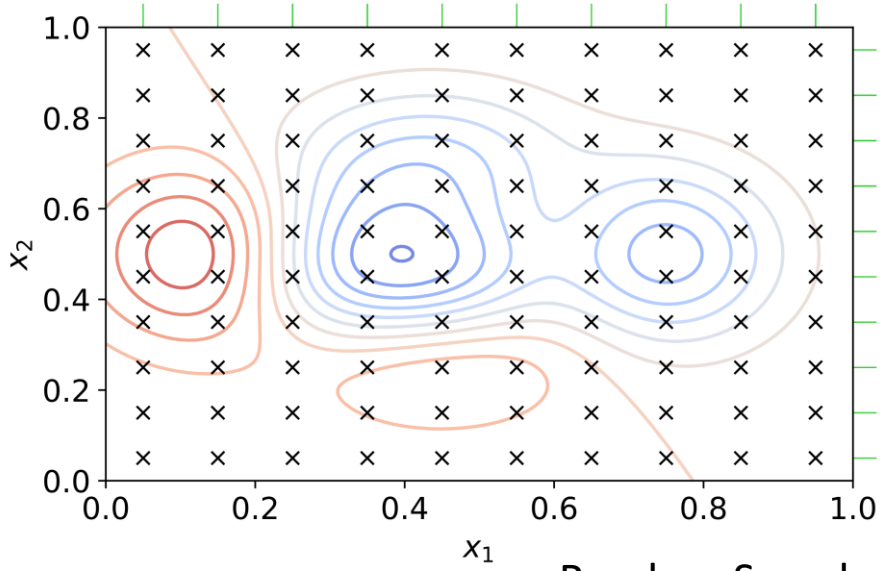
```
class sklearn.ensemble.RandomForestRegressor(n_estimators=100, *,
criterion='squared_error', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0,
max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True,
oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
ccp_alpha=0.0, max_samples=None, monotonic_cst=None) #                [source]
```

GradientBoostingRegressor

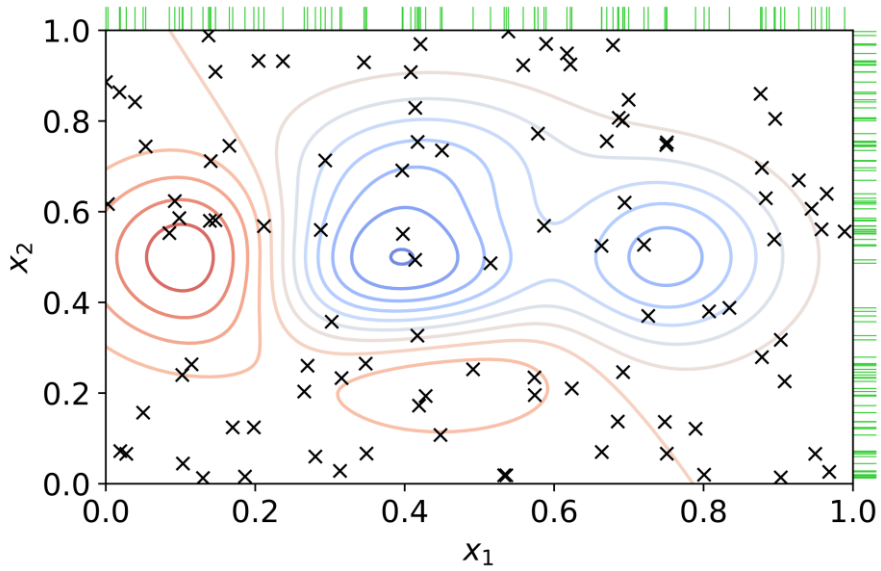
```
class sklearn.ensemble.GradientBoostingRegressor(*, loss='squared_error',
learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse',
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None,
max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False,
validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

Оптимизация гиперпараметров

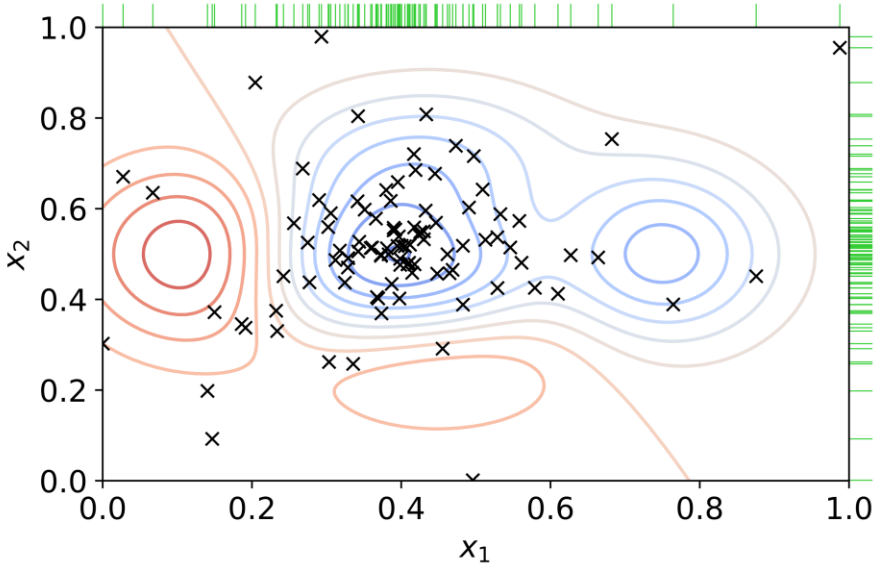
Grid Search



Random Search



Bayesian optimization



OPTUNA

Optuna: A hyperparameter optimization framework

Optuna is an automatic hyperparameter optimization software framework, particularly designed for machine learning. It features an imperative, *define-by-run* style user API. Thanks to our *define-by-run* API, the code written with Optuna enjoys high modularity, and the user of Optuna can dynamically construct the search spaces for the hyperparameters.

The end