Машинное обучение и анализ данных

Методы, основанные на деревьях: случайный лес, бустинг

Дьяконов А.Г.

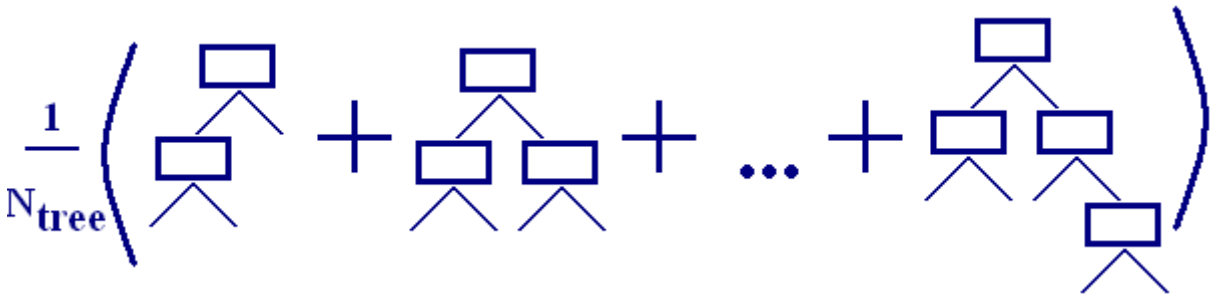
Московский государственный университет
имени М.В. Ломоносова (Москва, Россия)

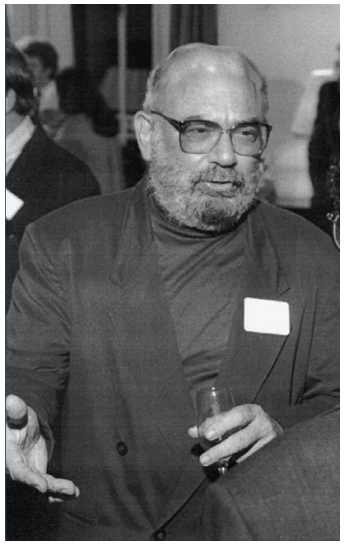
Случайный лес

– специальный метод ансамблирования

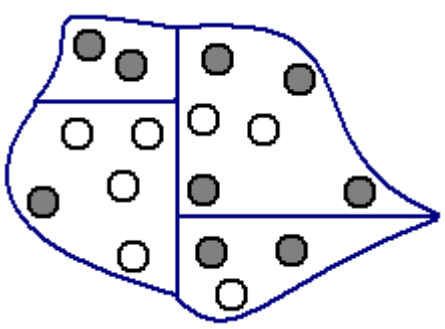
= бэггинг + специальное построение деревьев (подмножество признаков при расщеплении)

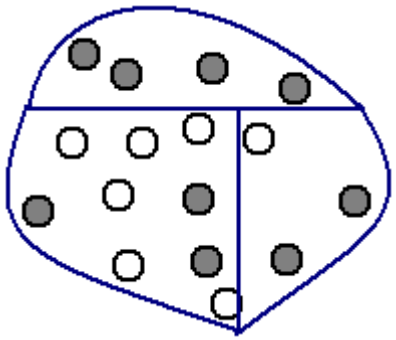
Качество одного дерева очень низкое!

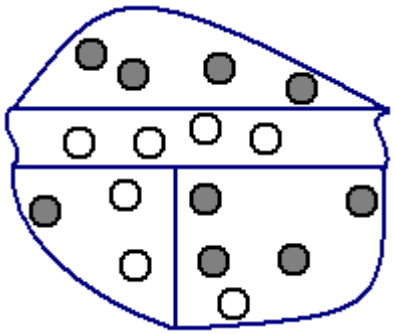




Лео Брейман, 1928 – 2005







Построение случайного леса

1. Выбирается подвыборка `samplesize` (м.б. с повторением) – на ней строится дерево

2. Строим дерево

2.1. Для построения каждого расщепления просматриваем `mtry` / `max_features` случайных признаков

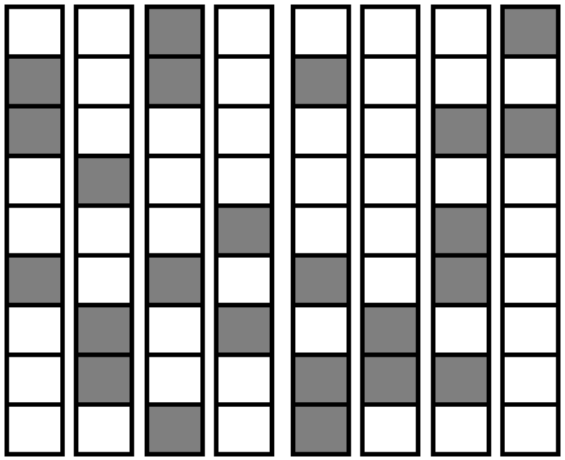
2.2. Как правило, дерево строится до исчерпания выборки (без прунинга)

Ответ леса: по большинству (в задачах классификации)
среднее арифметическое (в задачах регрессии)

Автоматически: рейтинг признаков –
`importance(model)/ .feature_importances_`

Бэггинг и OOB (out of bag)

Выбор объектов для обучения (с помощью бутстрепа), остальные – локальный контроль…



Ответы разных деревьев – можно усреднить и вычислить качество

«Решающее дерево»

criterion – **критерий расщепления «gini» / «entropy»**

splitter – **разбиение «best» / «random»**

max_depth – **допустимая глубина**

min_samples_split – **минимальная выборка для разбиения**

min_samples_leaf – **минимальная мощность листа**

min_weight_fraction_leaf – **аналогично с весом**

max_features – **число признаков, которые смотрим для нахождения разбиения**

random_state – **инициализация генератора случайных чисел**

max_leaf_nodes – **допустимое число листьев**

min_impurity_decrease – **порог «зашумлённости» для разбиения**

min_impurity_split – **порог «зашумлённости» для останова**

class_weight – **веса классов («balanced» или словарь, список словарей)**

«Случайный лес»

n_estimators – **число деревьев**

criterion

max_depth

min_samples_split

min_samples_leaf

max_features

max_leaf_nodes

min_impurity_decrease

min_impurity_split

bootstrap – **делать ли бутстреп**

oob_score – **вычислять ли OOB-ошибку**

n_jobs

random_state

verbose – **контроль процесса**

warm_start – **использовать ли существующий лес, чтобы его дополнить или учить заново**

class_weight

Параметры случайного леса

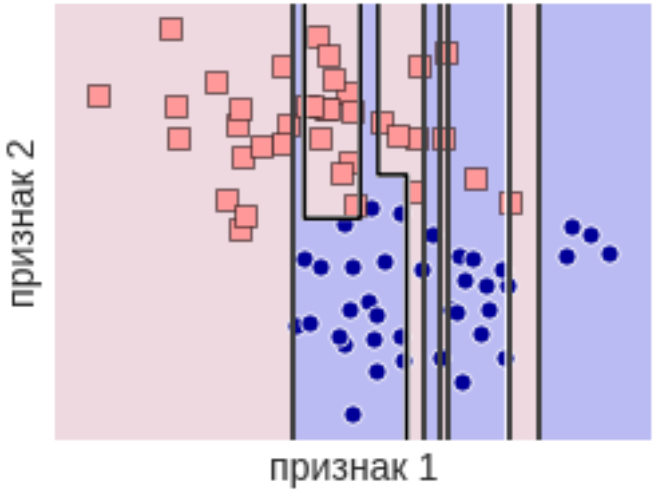
```
class
sklearn.ensemble.RandomForestClassifier
(n_estimators=10,
criterion='gini',
max_depth=None,
min_samples_split=2,
min_samples_leaf=1,
min_weight_fraction_leaf=0.0,
max_features='auto',
max_leaf_nodes=None,
bootstrap=True,
oob_score=False,
n_jobs=1,
random_state=None,
verbose=0,
warm_start=False,
class_weight=None)
```

```
{randomForest} randomForest(
x, y, xtest, ytest,
ntree=500,
mtry=if (!is.null(y) &&
!is.factor(y))
max(floor(ncol(x)/3), 1) else
floor(sqrt(ncol(x))),
replace=TRUE,
classwt=NULL,
cutoff,
strata,
sampsize = if (replace) nrow(x)
else ceiling(.632*nrow(x)),
nodesize = if (!is.null(y) &&
!is.factor(y)) 5 else 1,
maxnodes = NULL,
importance=FALSE,
localImp=FALSE,
nPerm=1,
proximity, oob.prox=proximity)
```

«Случайный лес»

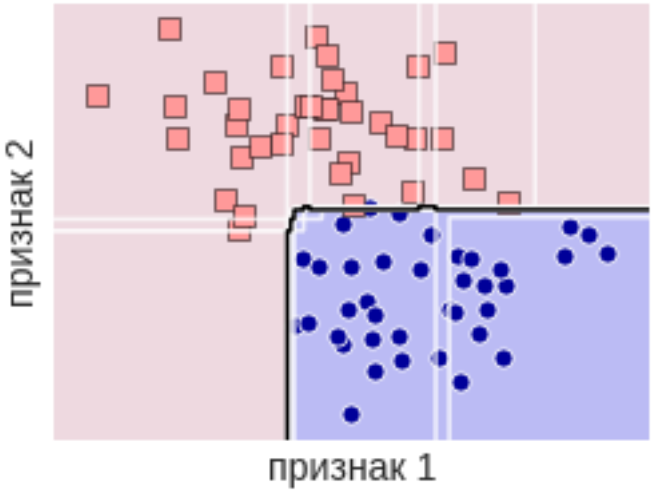
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=1)
rf.fit(X_train, y_train)
```

n_estimators=1



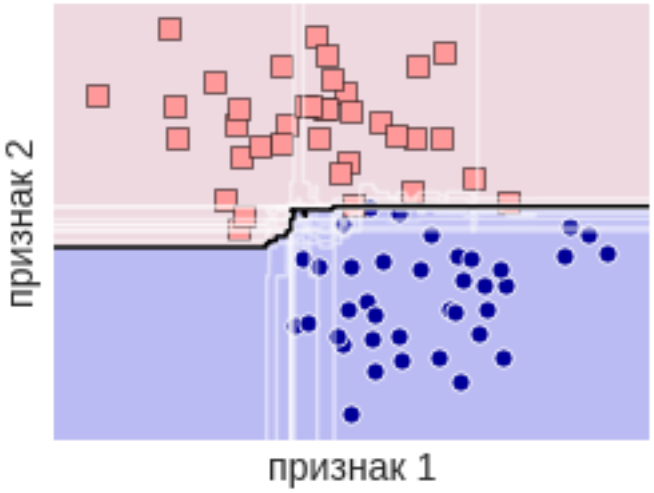
86%

n_estimators=5



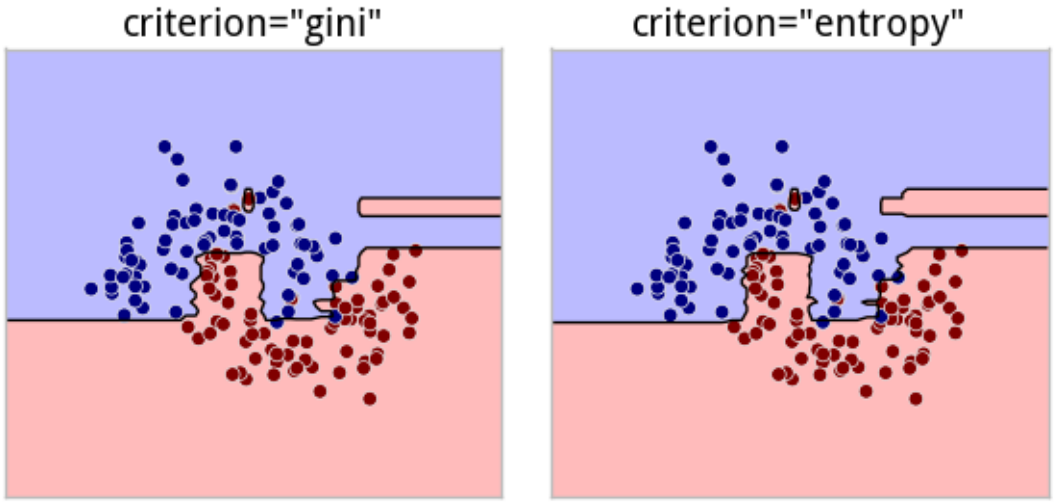
84%

n_estimators=100



88%

Различные критерии расщеления



в авторском коде был реализован Джини...

Настройка параметров: размер подвыборки `sampsize`

1. **Определиться с типом выбора**

   с возвратом / без возврата

2. **Настройка по объёму**

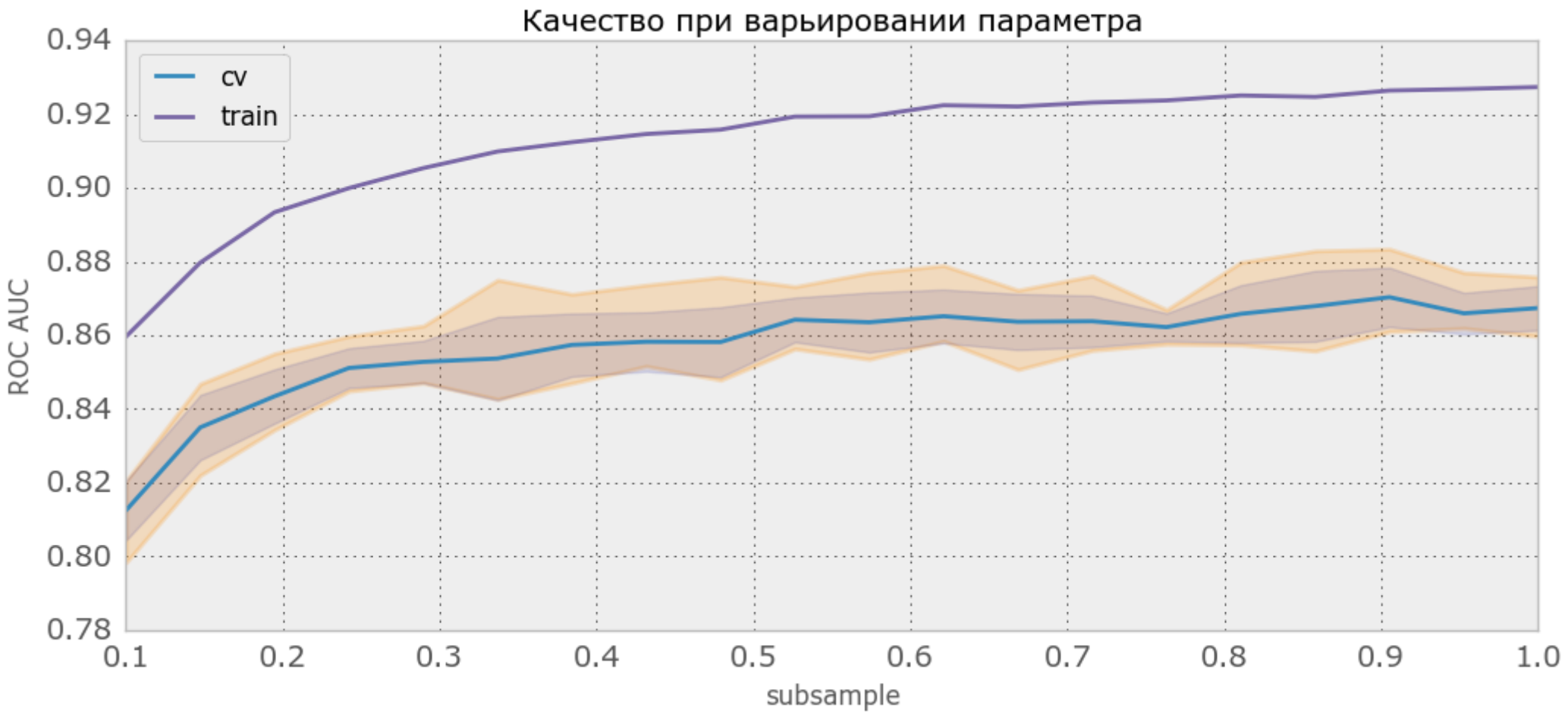
   – не в первую очередь

Часто «нужны все объекты»

Чем больше – тем однотипнее деревья

Что из этого следует?

Настройка параметров: размер подвыборки `sampsize` (СберБанк)



Всю выборку надо использовать по максимуму!

Настройка параметров: число признаков `mtry / max_features`

Самый серьёзный параметр

По умолчанию: $\sqrt{n}$ **– классификация**

$n/3$ **– регрессия**

Зависимость унимодальная

Настраивается в первую очередь

Зависит от числа шумовых признаков

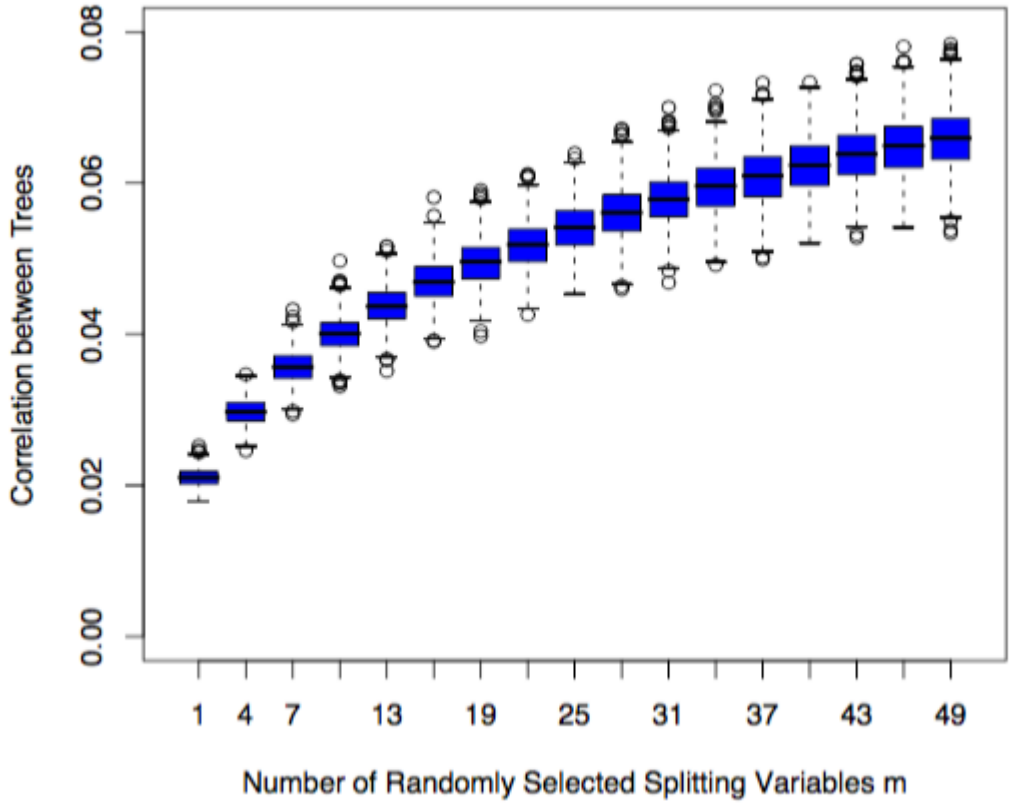
Надо перенастраивать при добавлении новых признаков

Чем больше – тем однотипнее деревья.

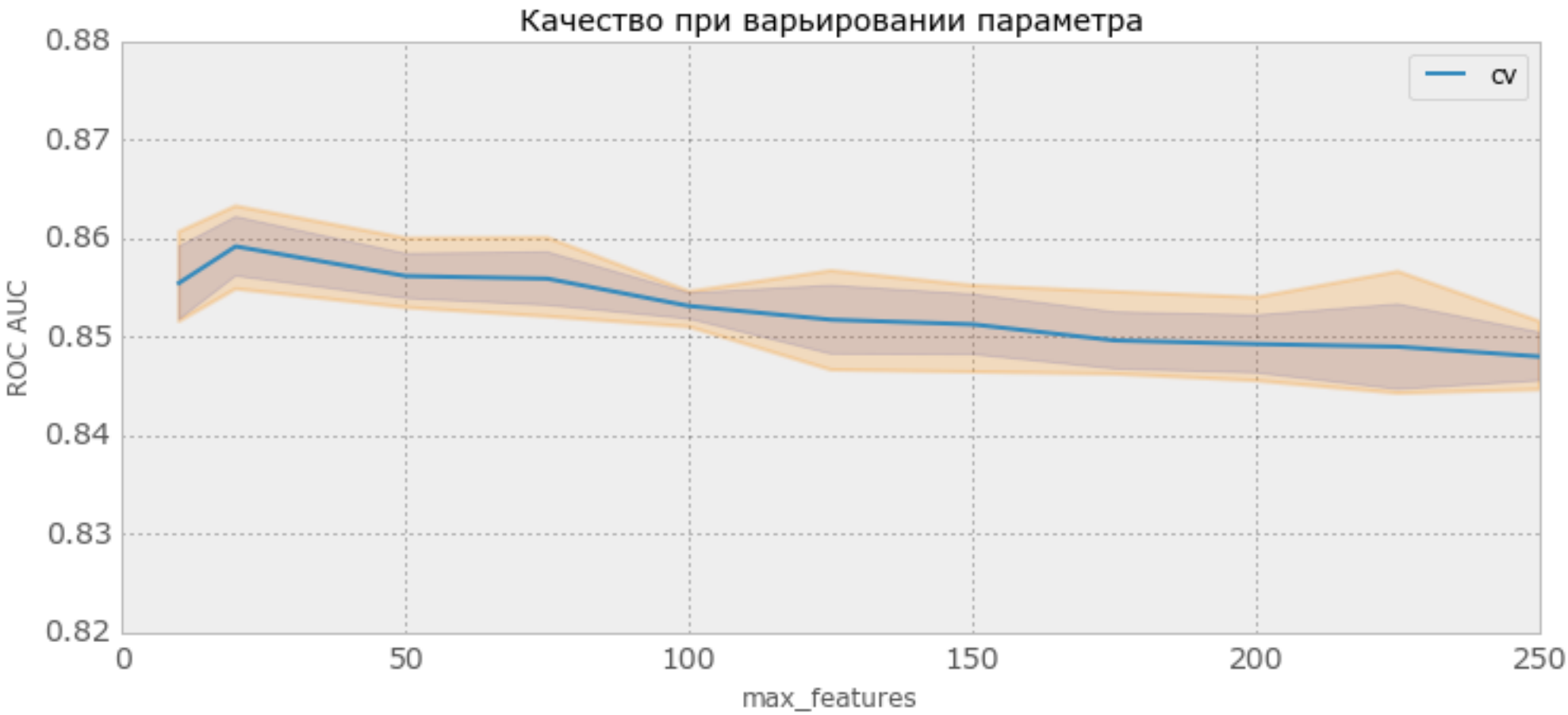
Чем больше – тем медленнее настройка!

Kaggle: часто суммируют алгоритмы с разными mtry.

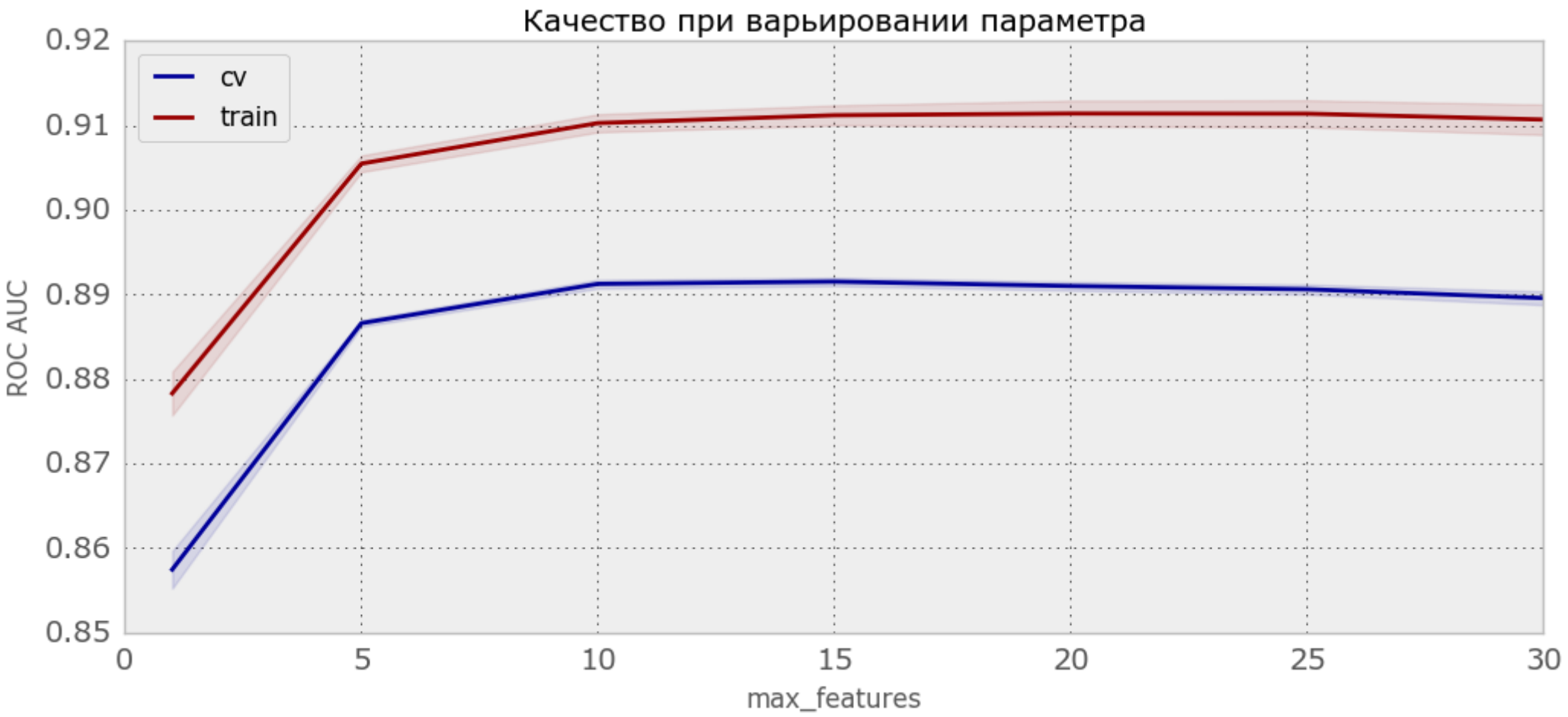
Настройка параметров: число признаков mtry / max_features



Настройка mtry / max_features (СберБанк)

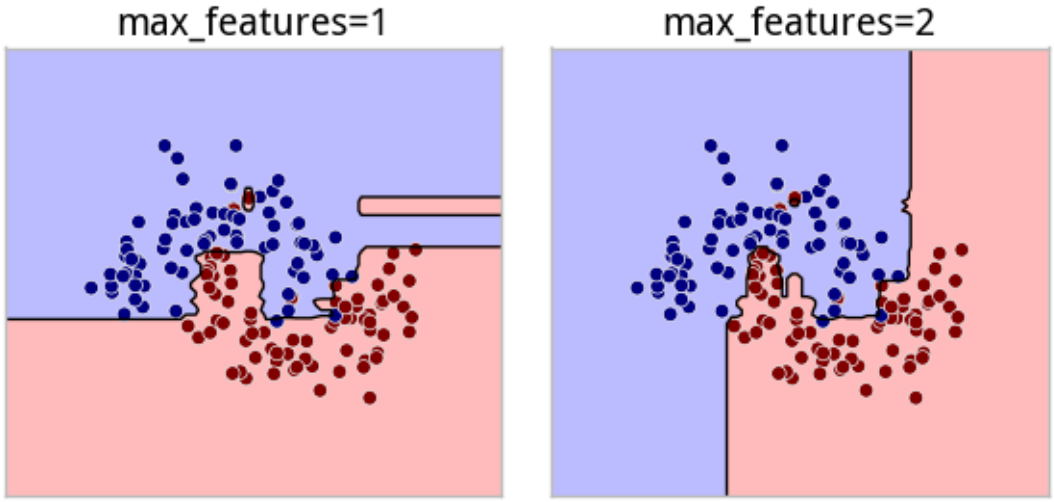


Настройка mtry / max_features (ed Бозон)

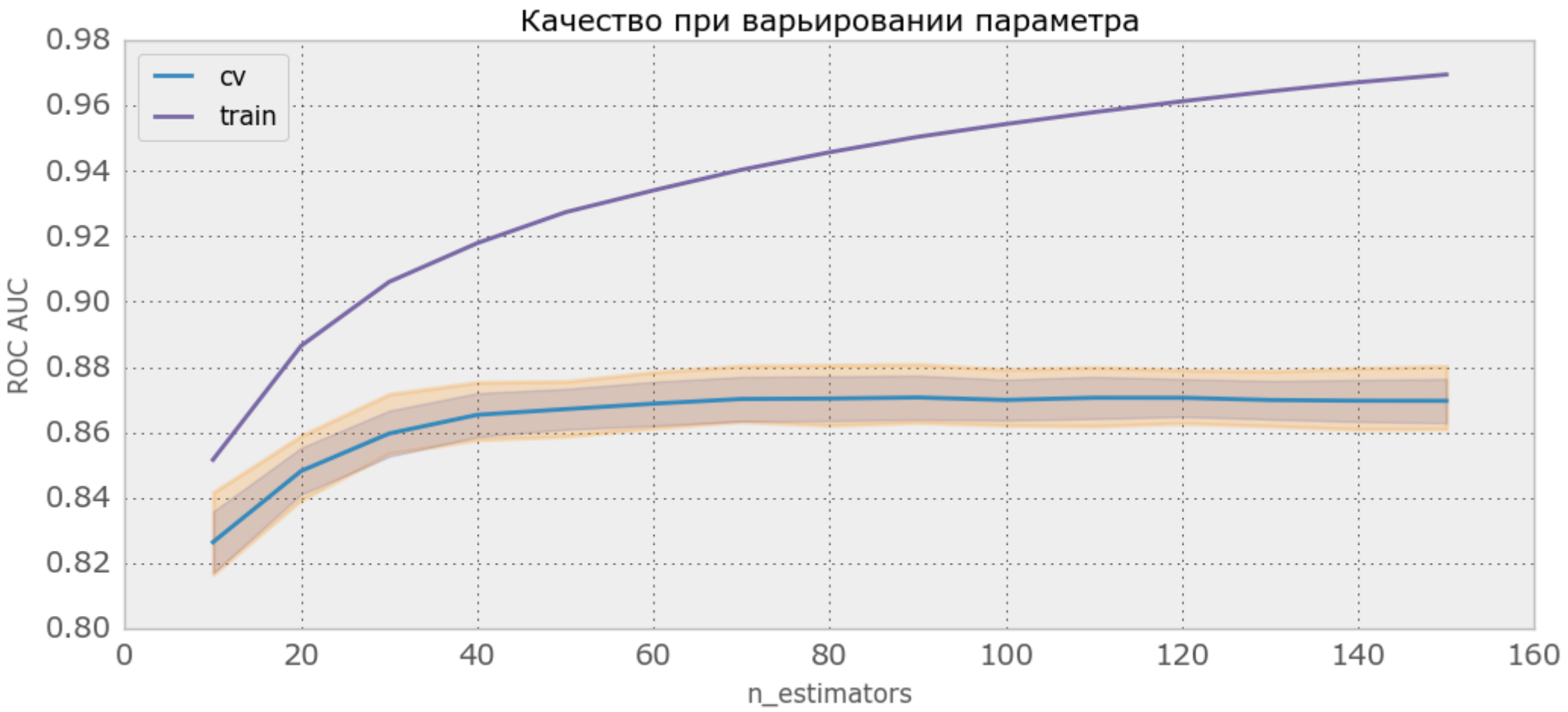


в задаче ~ 33 признака

Настройка mtry / max_features

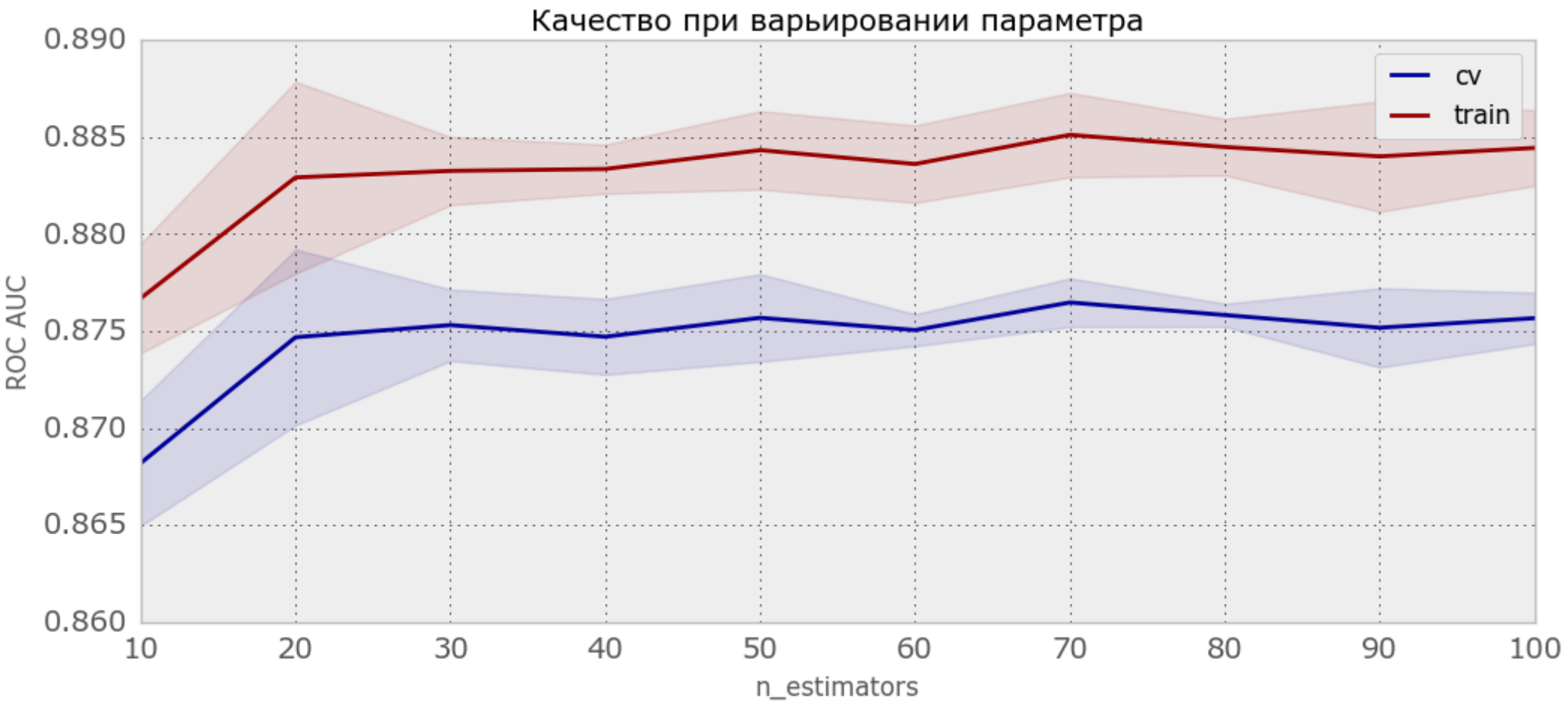


Настройка параметров ntree / n_estimators (СберБанк)

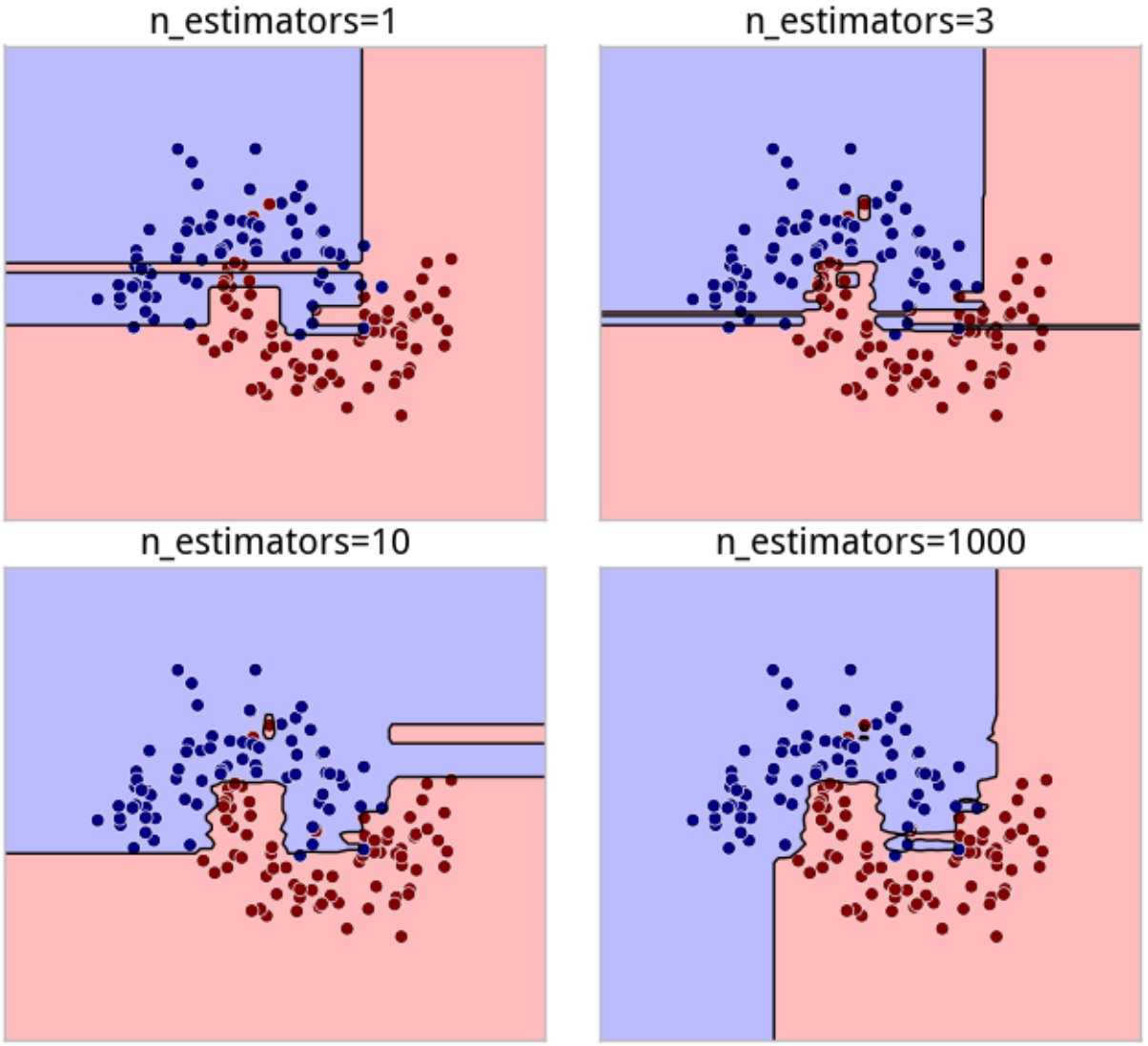


Чем больше деревьев – тем лучше!

Настройка параметров ntree / n_estimators (ed Бозон)



Настройка параметров `ntree / n_estimators`



Настройка параметров `ntree / n_estimators` (СберБанк)

Чем больше – тем лучше!

Проблемы:

- **как использовать при настройке параметров очень большое число деревьев**
- **что делать, если не помещаются в память… (в R)**

Настройка параметров: число объектов в листе, число объектов для расщепления, максимальная глубина дерева

От параметров существенно зависит скорость построения леса

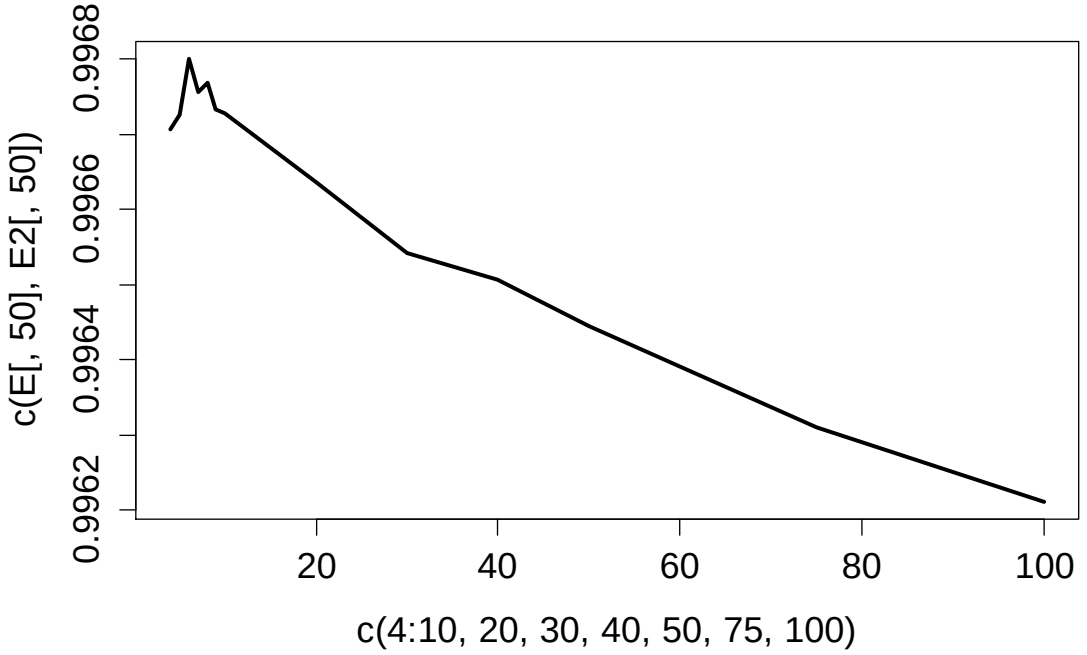
Оптимальные значения, как правило, - несколько объектов в листе.

Настраиваются не в первую очередь

В классическом случайном лесе деревья строятся до исчерпания выборки…

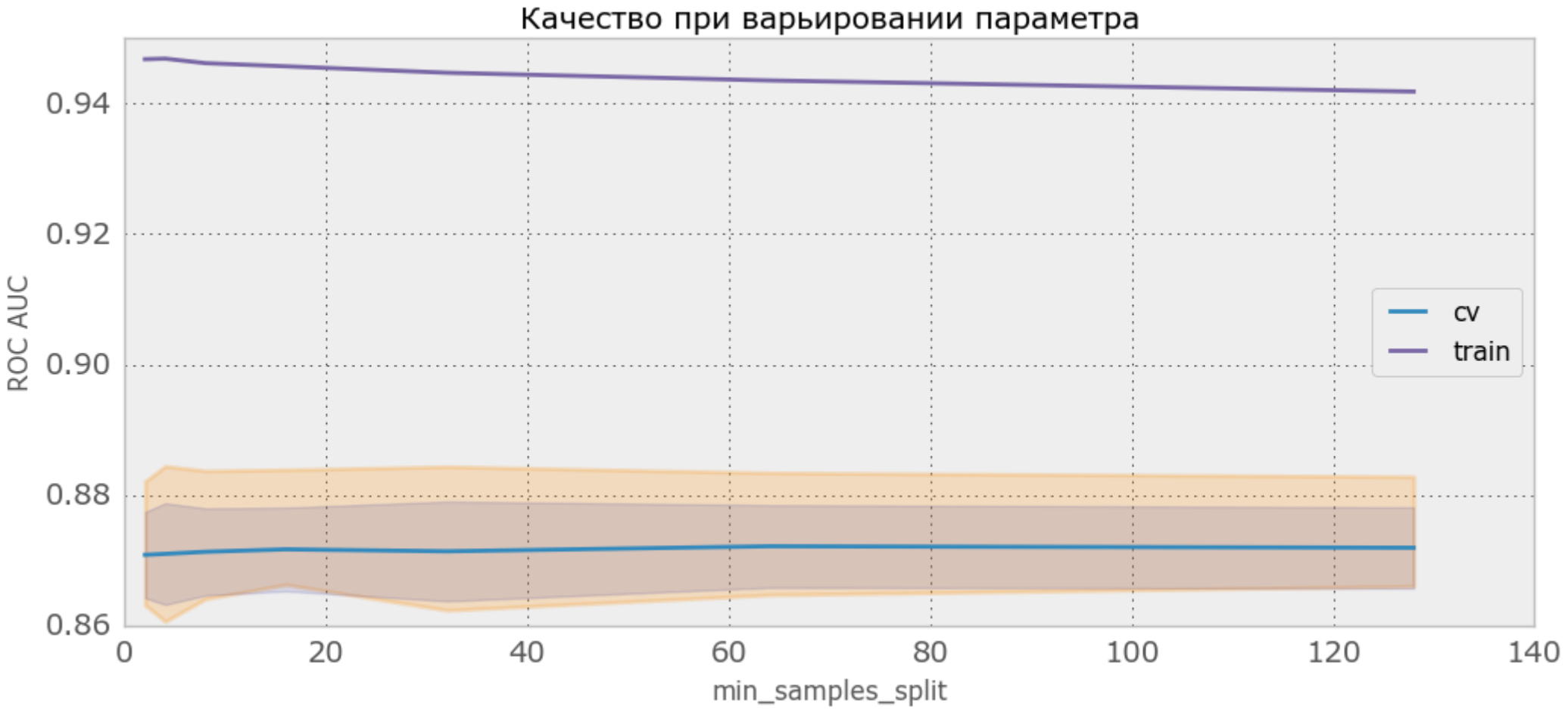
«Good results are often achieved when setting `max_depth=None` in combination with `min_samples_split=1`»

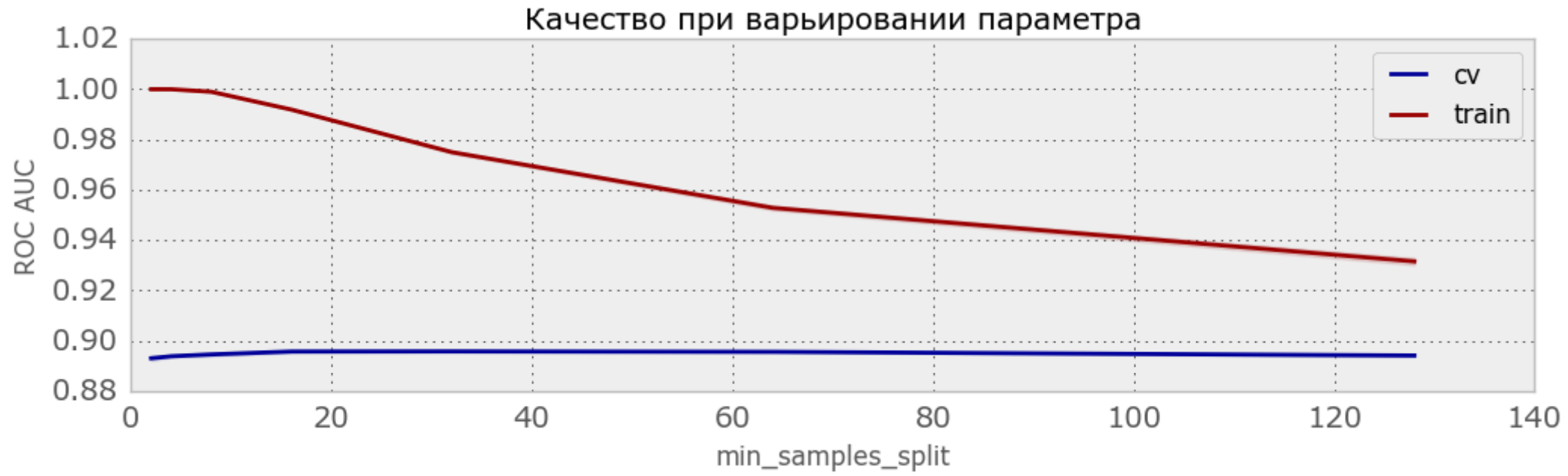
randomForest: nodesize

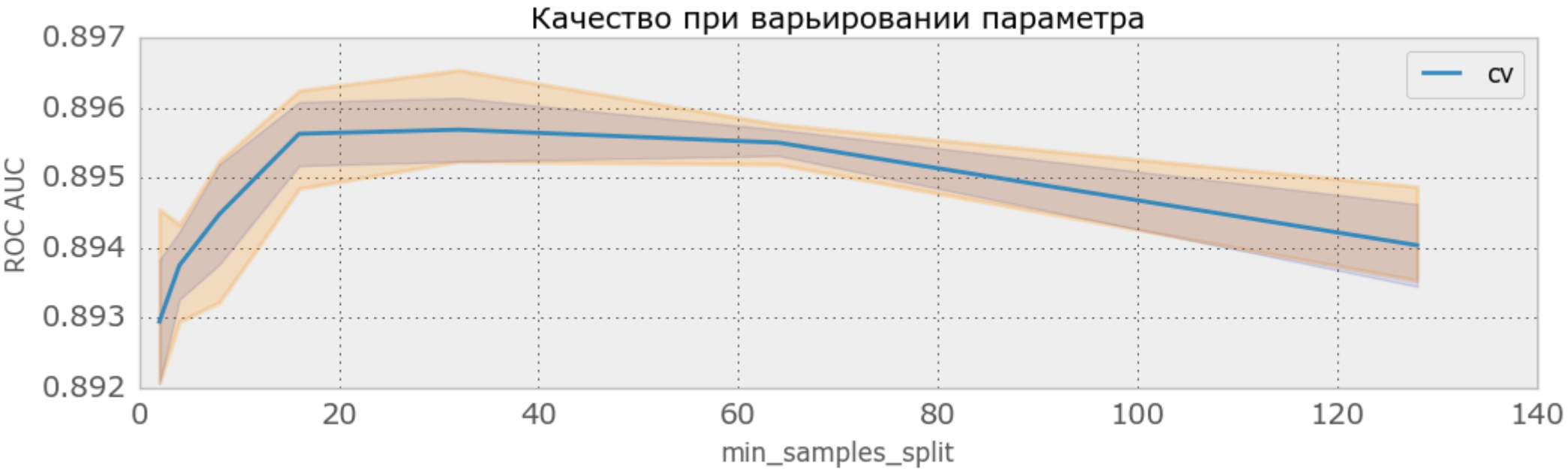


умолчание: 1 – классификация, 5 – регрессия

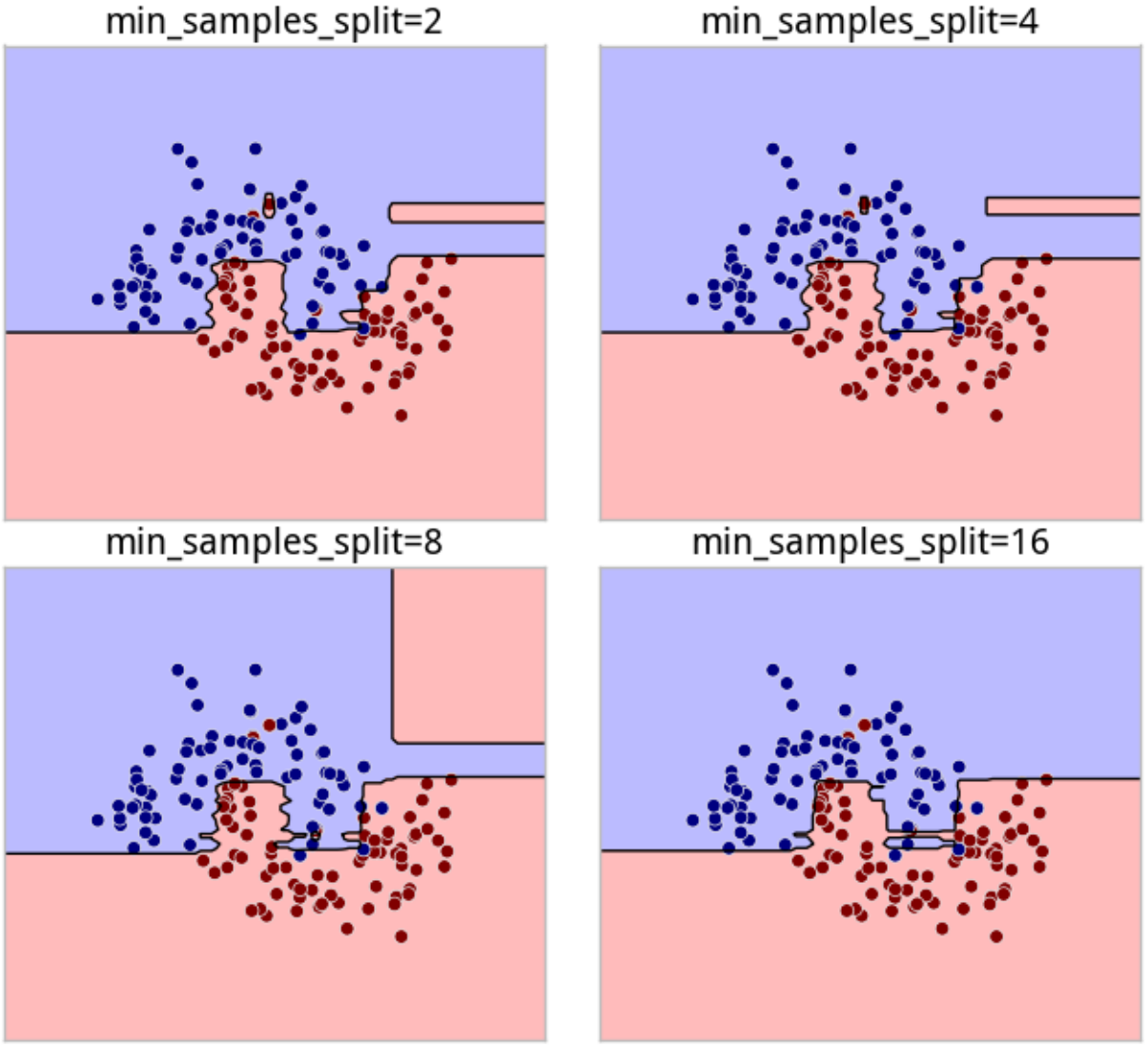
RandomForestClassifier: min_samples_split (СберБанк)



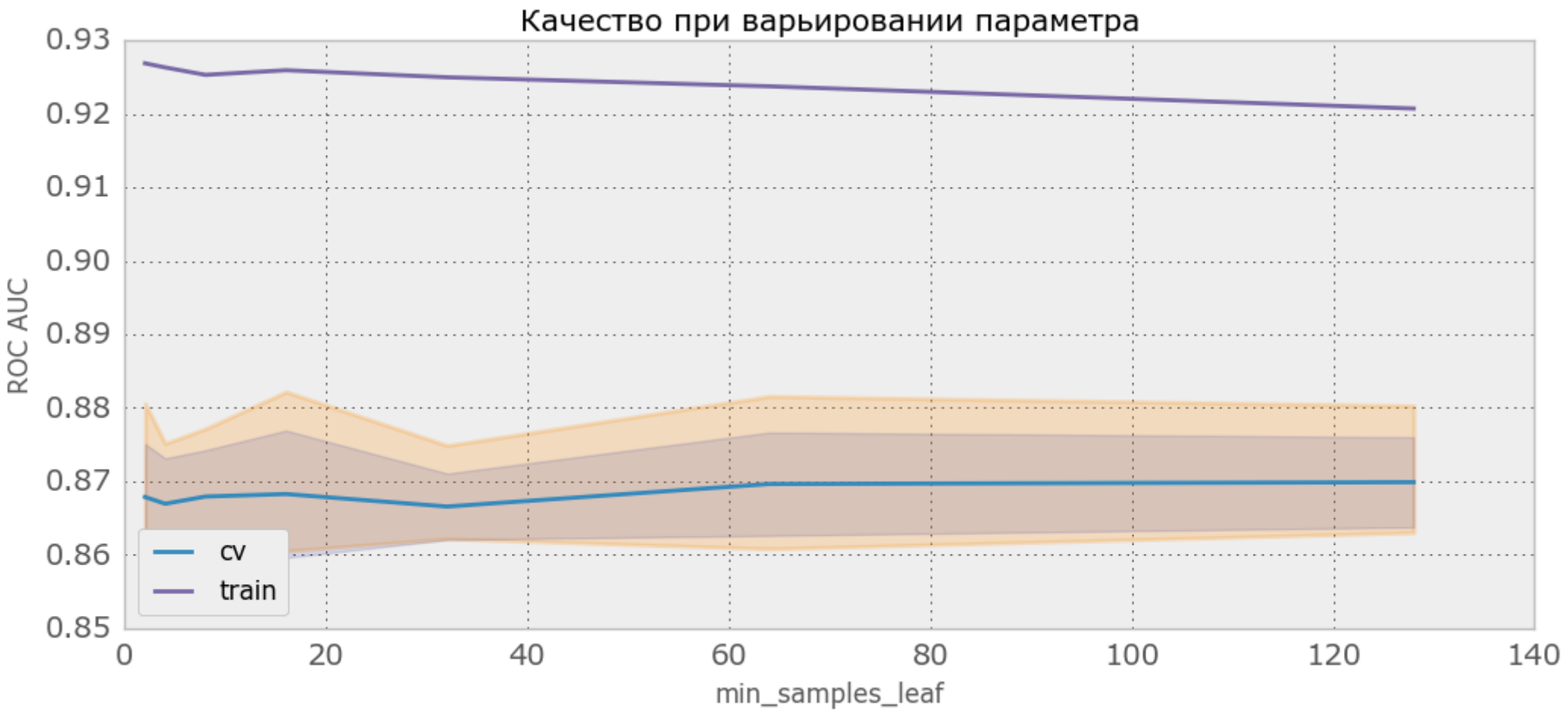

Качество при варьировании параметра
ROC AUC
1.02
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
20
40
60
80
100
120
140
min_samples_split
cv
train

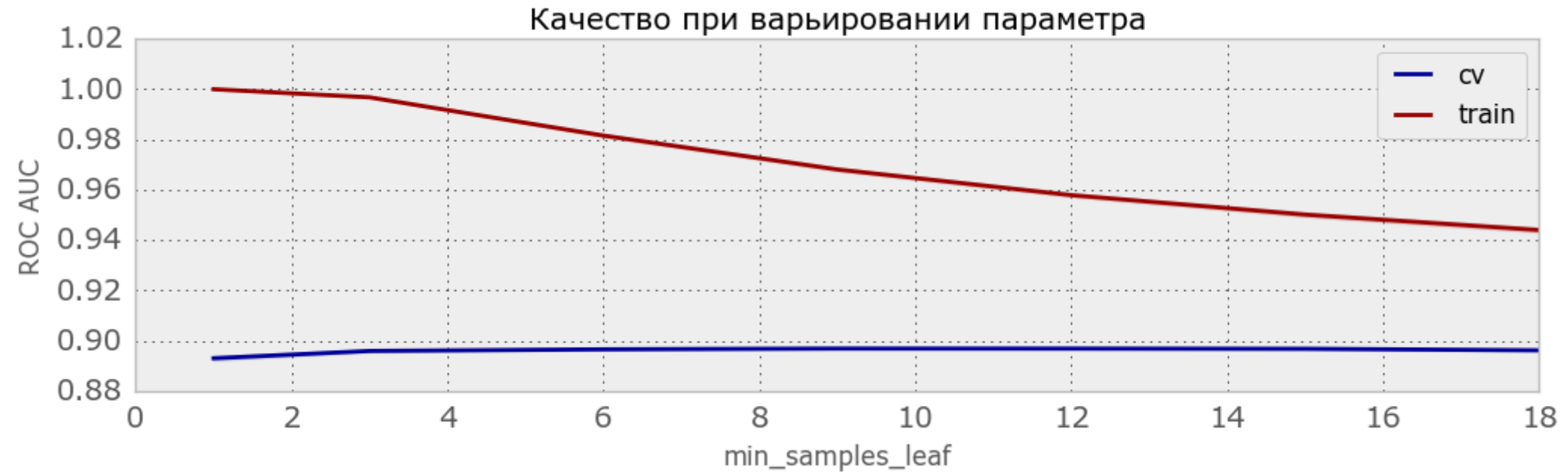

Качество при варьировании параметра
ROC AUC
0.897
0.896
0.895
0.894
0.893
0.892
0
20
40
60
80
100
120
140
min_samples_split
cv

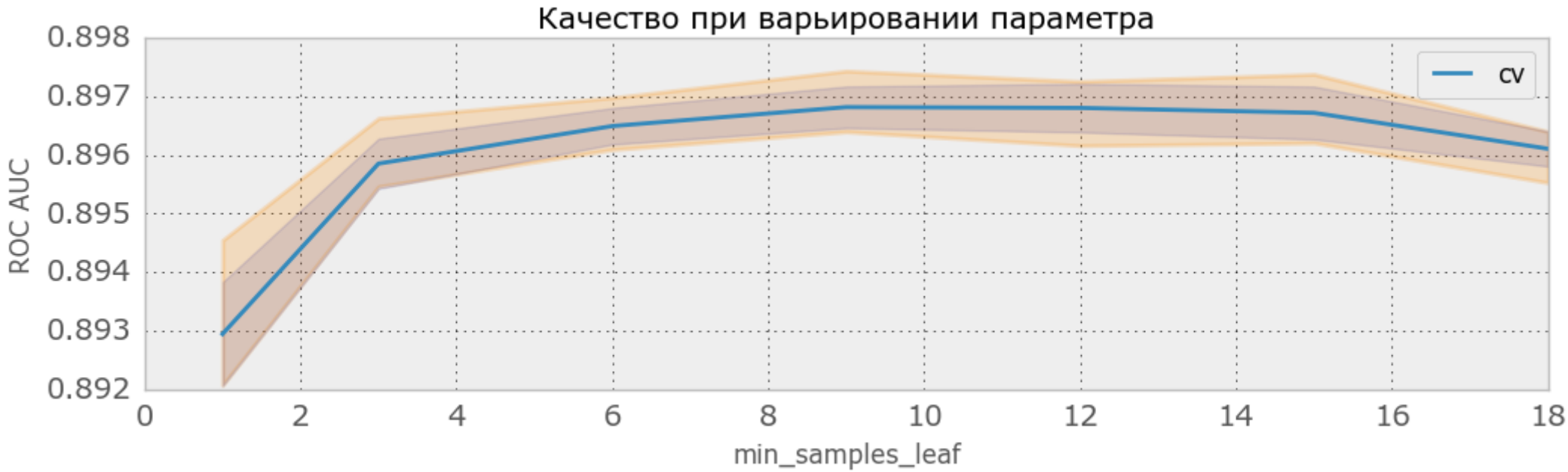
RandomForestClassifier: min_samples_split



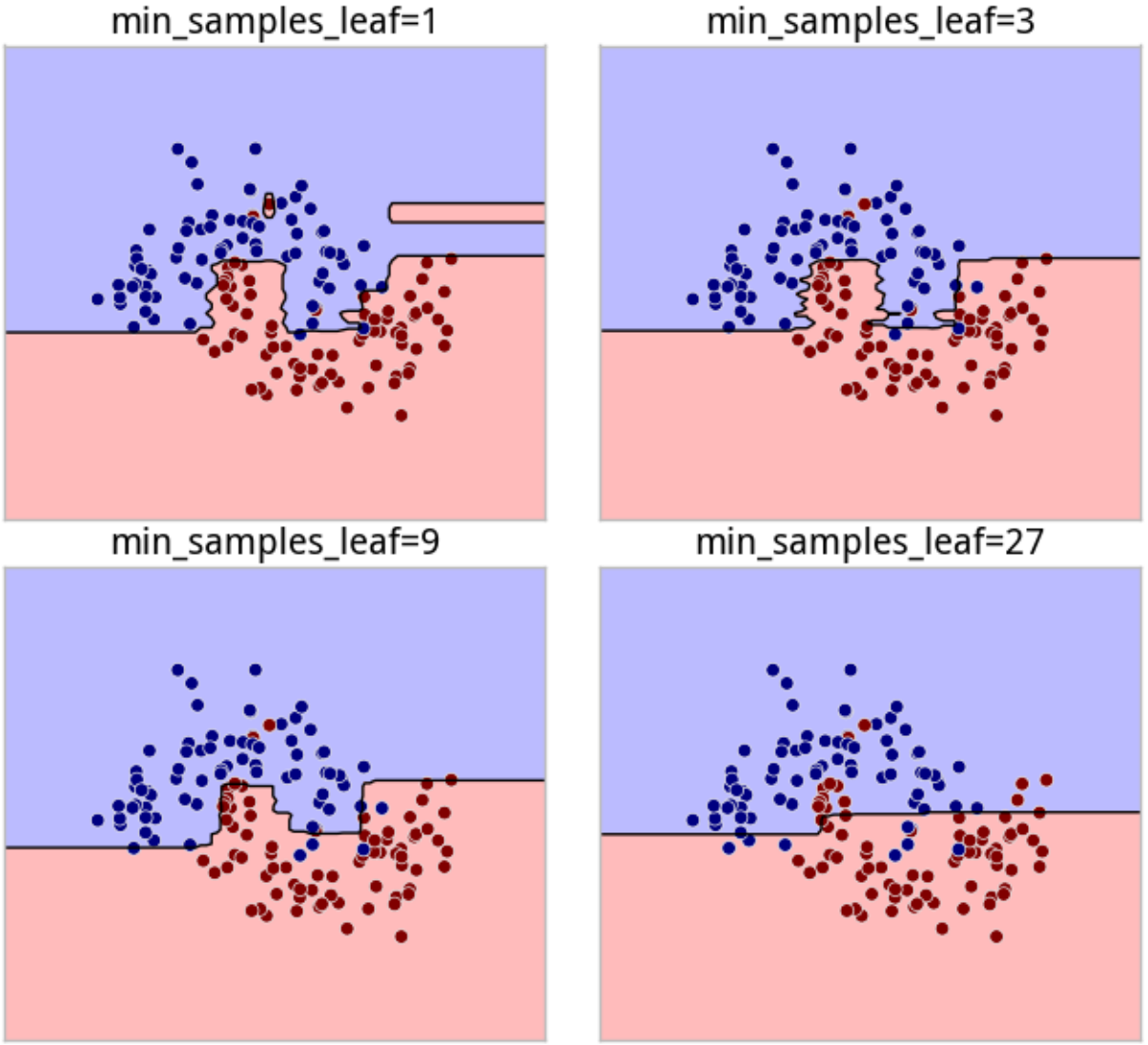
RandomForestClassifier: min_samples_leaf (СберБанк)



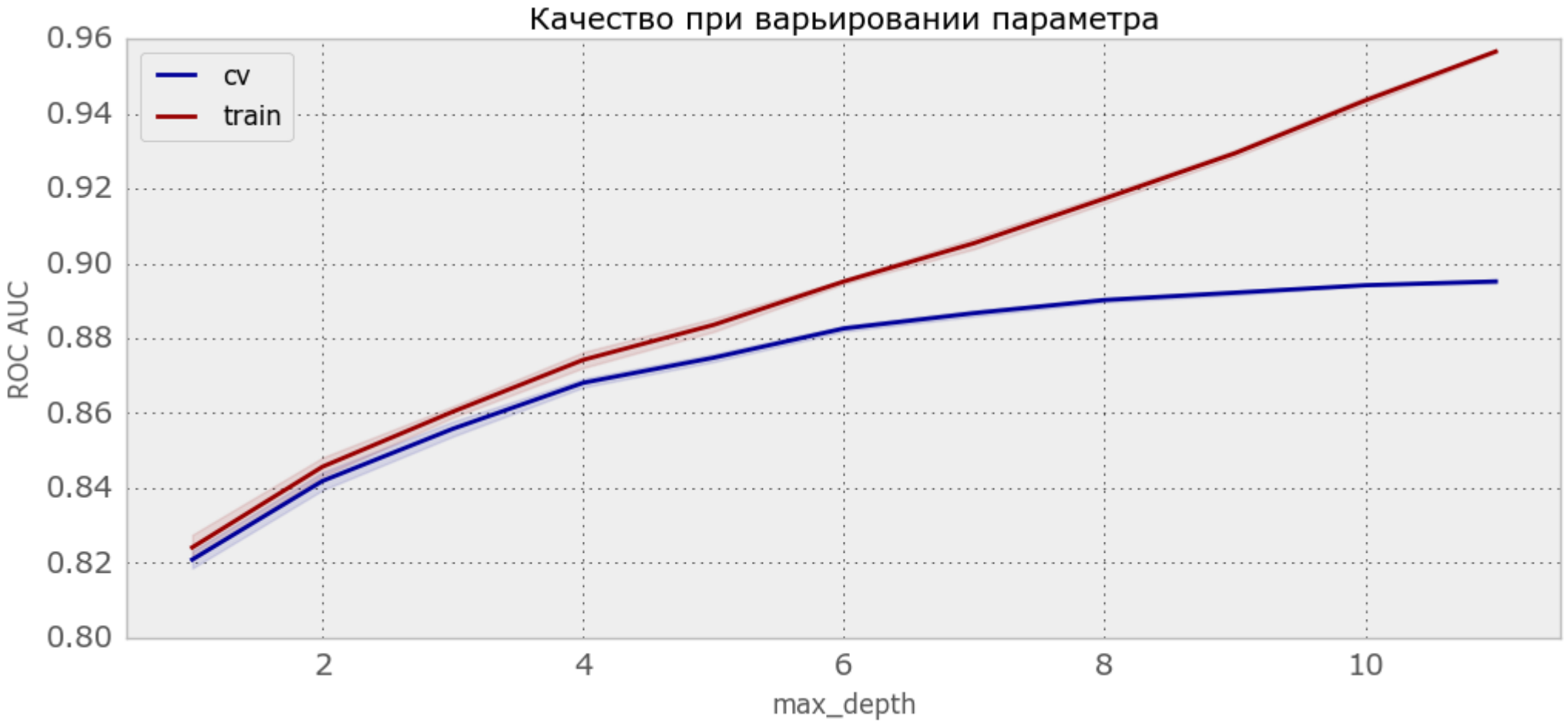

Качество при варьировании параметра
ROC AUC
1.02
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
2
4
6
8
10
12
14
16
18
min_samples_leaf
cv
train


Качество при варьировании параметра
ROC AUC
0.898
0.897
0.896
0.895
0.894
0.893
0.892
0
2
4
6
8
10
12
14
16
18
min_samples_leaf
cv

RandomForestClassifier: min_samples_leaf

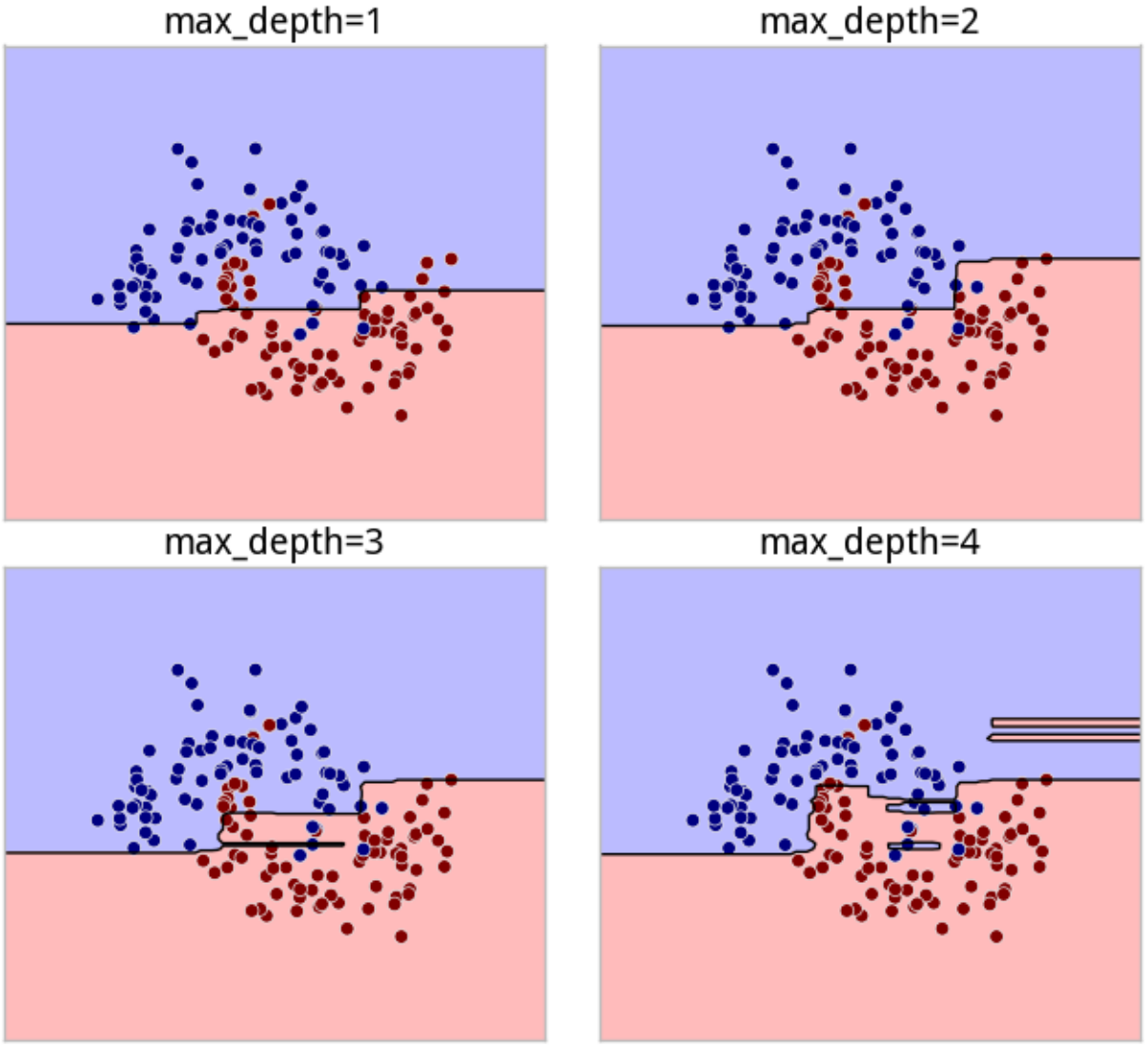


Глубина дерева: max_depth (СберБанк)



Как правило, чем больше, тем лучше!

Глубина дерева: max_depth

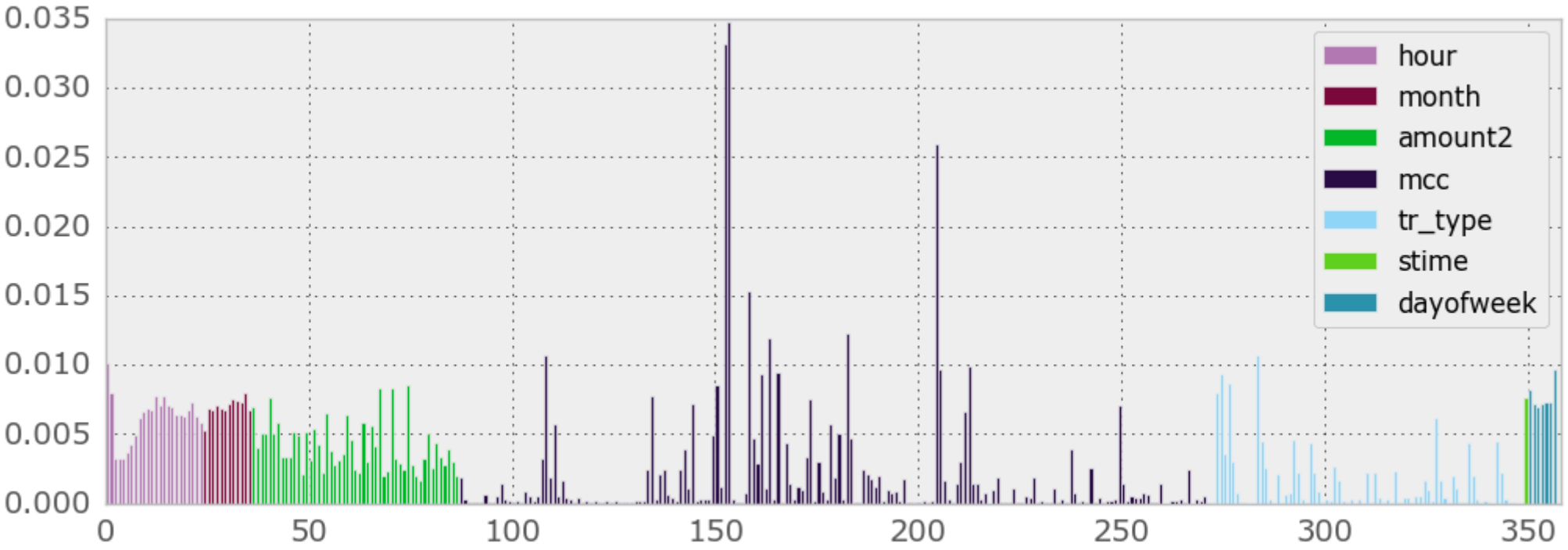


Глубина дерева: `max_depth`

Неглубокие деревья:

- **в задачах с выбросами**
- **когда много объектов**
  (деревья большие и долго строятся)
- **настройка некоторых других (каких?) параметров не имеет смысла**

Важность признаков (СберБанк)

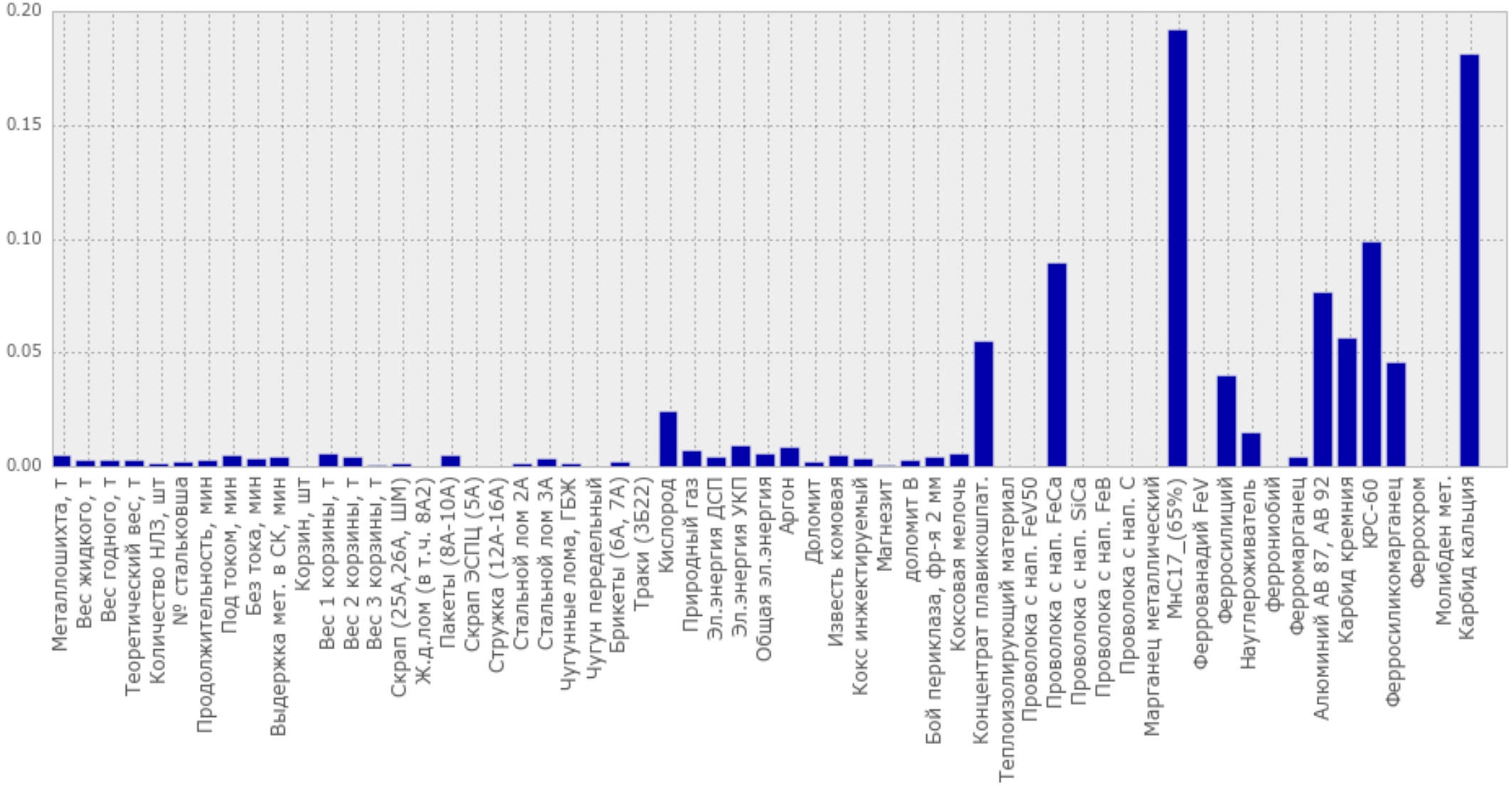


rf.feature_importances_

```
rf = RandomForestClassifier(n_estimators=1000, max_features=30, n_jobs=-1)
rf.fit(X, y)
plt.bar(np.arange(len(rf.feature_importances_)), rf.feature_importances_,
color='black')
```

Можно сразу увидеть важные признаки и целые группы…

Важность признаков (Металлургия)



сразу понятно, от чего зависит целевой признак

Важность признаков: два подхода (`importance(model)` в R)

%IncMSE

OOB (out of bag)

1. **Вычисляем качество Q на OOB**
2. **Для *i*-го признака – делаем случайную перестановку значений, вычисляем качество Q_i на OOB**
3. **Информативность *i*-го признака =** $\max(Q - Q_i, 0)$

Важность признаков: два подхода (importance(model) в R)

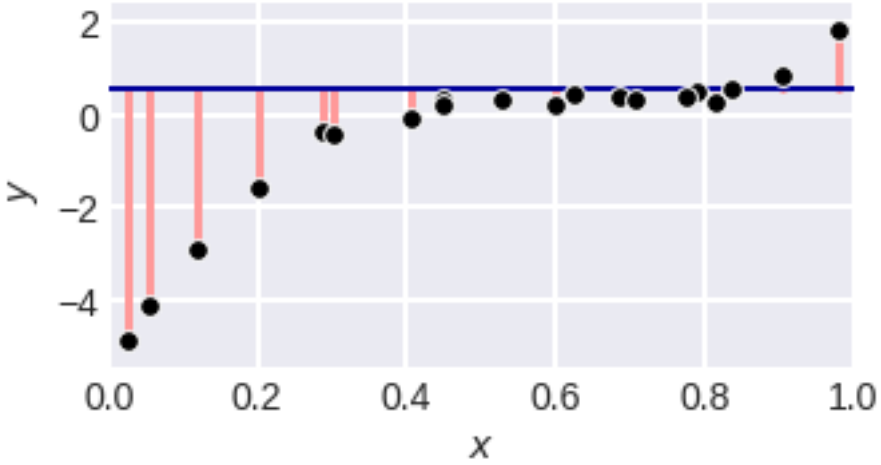
IncNodePurity

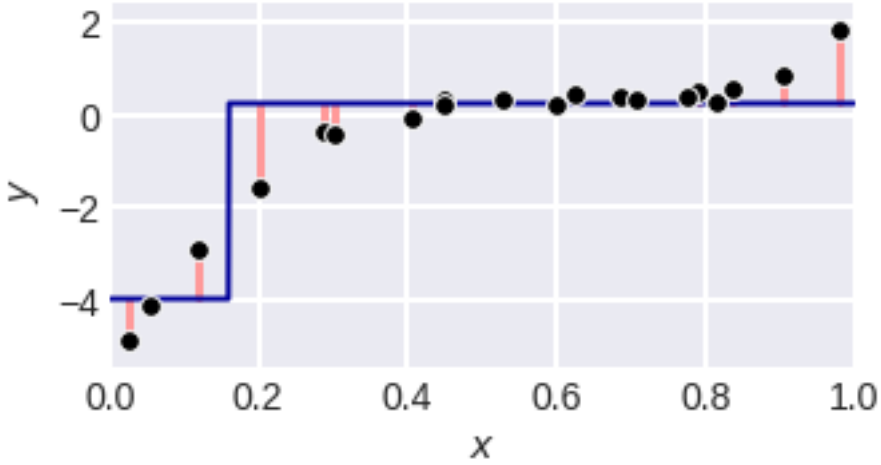
При каждом расщеплении – $RSS_{\text{old}} - RSS_{\text{new}}$

Берётся сумма по всем расщеплениям для конкретной переменной, по всем деревьям.

residual sum of squares (RSS)

$$\sum_{i \in \text{left}} (y_{\text{left}} - y_i)^2 + \sum_{i \in \text{right}} (y_{\text{right}} - y_i)^2$$





В sklearn (feature_importances_) аналогичная идея с критерием Gini

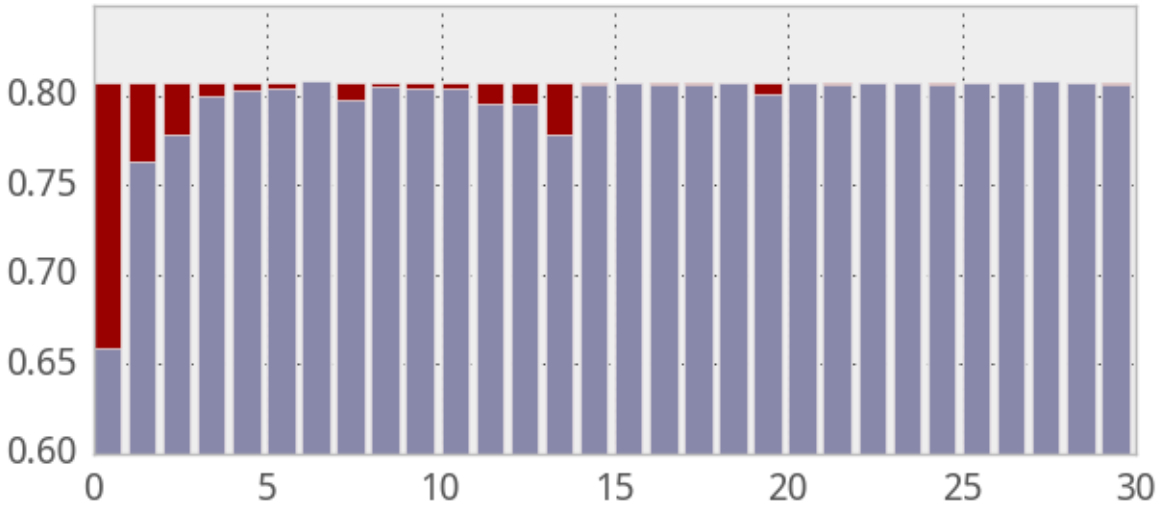
Shuffle-важность

```
e = [] # качество классификации

a = rf.predict(X2)
q = roc_auc_score(y2, a) # базовое качество классификации

for t in range(X2.shape[1]):
    Xt = X2.copy()
    np.random.shuffle(Xt[:, t]) # перемешиваем
    at = rf.predict(Xt)
    e.append(roc_auc_score(y2, at))

e = np.array(e)
plt.bar(np.arange(len(e)), e*0 + q, color = '#990000')
plt.bar(np.arange(len(e)), e, color = '#8888AA')
```



Proximity

при построении деревьев можно много чего считать…

**Чем чаще 2 объекта попадают в один лист,
тем они ближе…**

Какую метрику можно придумать?

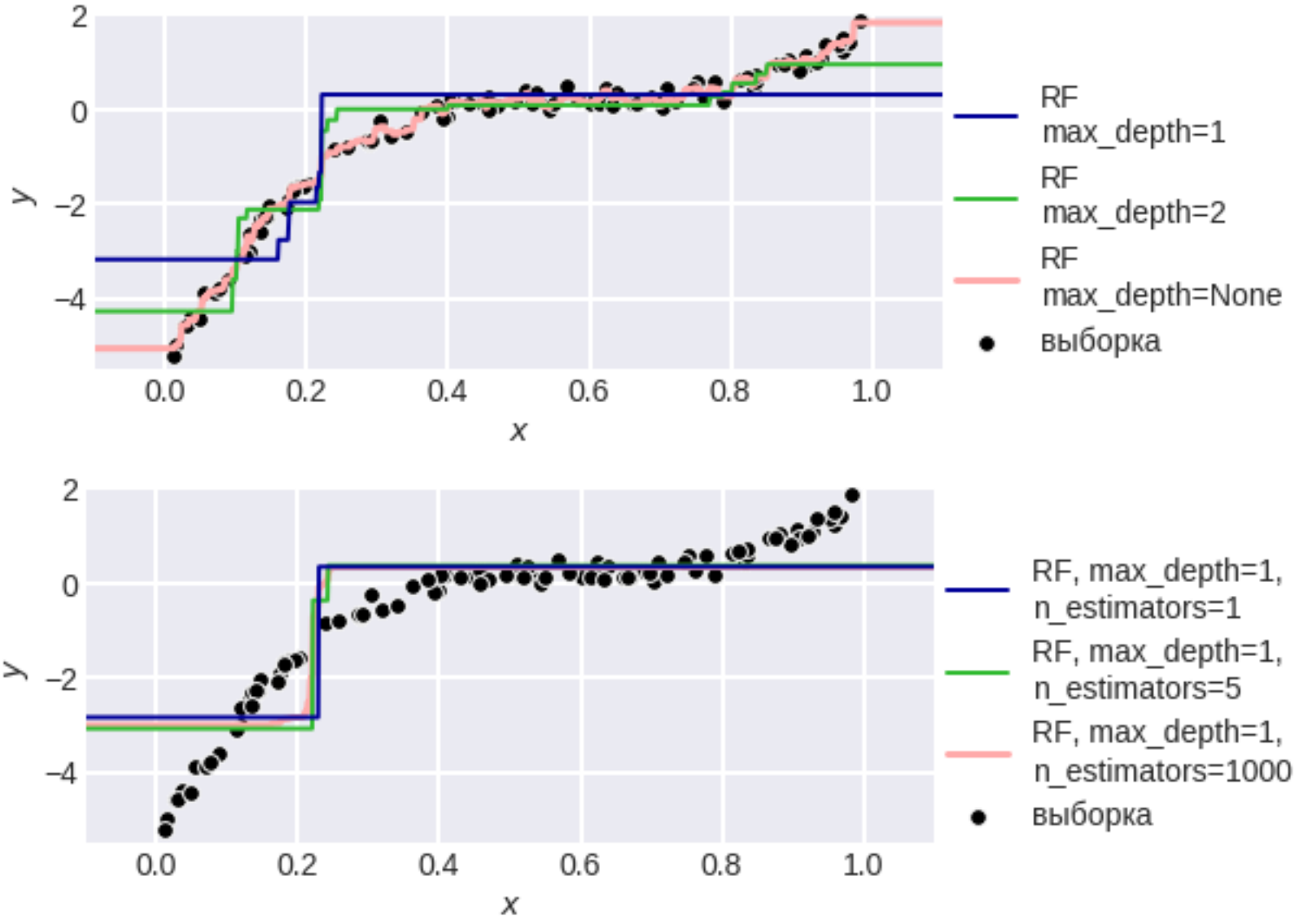
Extreme Random Trees

- нет бутстрепа (используем всю выборку)
- генерируем несколько пар (признак, порог)
  - выбираем оптимальную для разбиения пару
- также есть параметр «число признаков для просмотра»

- ET быстрее RF
- ET чуть хуже RF, когда много шумных признаков

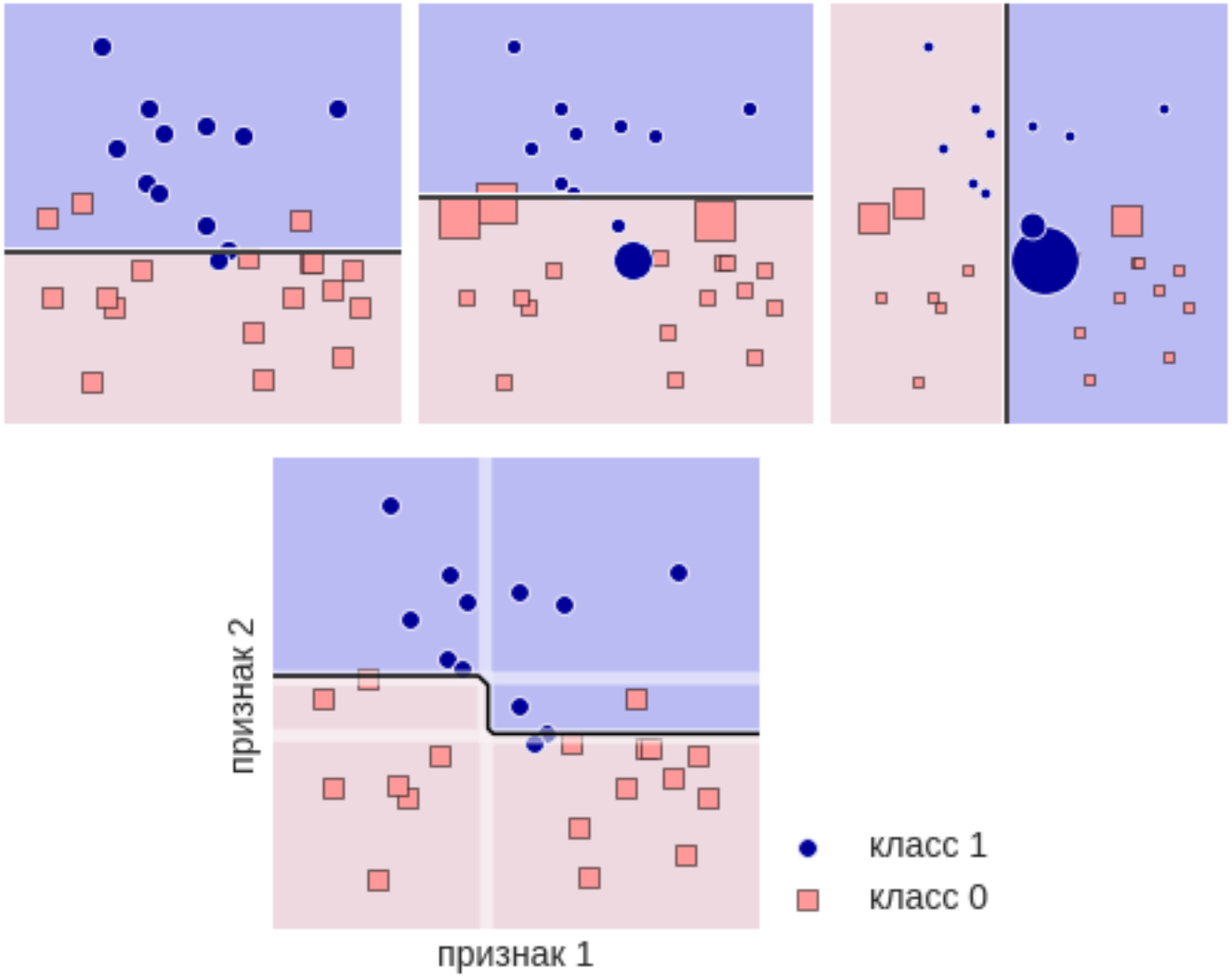
```
from sklearn.ensemble import ExtraTreesClassifier
clf = ExtraTreesClassifier(n_estimators=10, max_depth=None,
                           min_samples_split=2, random_state=0)
```

Когда плохи методы, основанные на деревьях...



Градиентный бустинг над деревьями

Вспоминаем идею...



Идея градиентного бустинга

FSAM + минимизация в случае дифференцируемой ф-ии ошибки

Задача регрессии

$$(x_i, y_i)_{i=1}^m$$

дифференцируемая функция ошибки

$$L(y, a)$$

уже есть алгоритм $a(x)$ **строим** $b(x)$**:**

$$a(x_i) + b(x_i) = y_i, i \in \{1,2,\ldots,m\}.$$

Надо:

$$\sum_{i=1}^m L(y_i, a(x_i) + b(x_i)) \to \min,$$

а не

$$\sum_{i=1}^m L(y_i - a(x_i), b(x_i)) \to \min$$

Проблема

Задача

$$\sum_{i=1}^{m} L(y_i, a(x_i) + b(x_i)) \to \min$$

может не решаться аналитически

$$F = \sum_{i=1}^{m} L(y_i, a(x_i) + b_i) \to \min_{(b_1,\ldots,b_m)}$$

Функция $F(b_1,\ldots,b_m)$ **убывает в направлении антиградиента, поэтому выгодно считать**

$$b_i = -L'(y_i, a(x_i)), \; i \in \{1,2,\ldots,m\},$$

новая задача для настройки второго алгорпитма:

$$(x_i, -L'(y_i, a(x_i)))_{i=1}^{m}.$$

Алгоритм градиентного бустинга (примитивный вариант)

- **Строим алгоритм в виде**

$$a_n(x) = \sum_{t=1}^{n} b_t(x),$$

для удобства можно даже считать, что $a_0(x) \equiv 0$.

- **Пусть построен** $a_t(x)$**, тогда обучаем алгоритм** $b_t(x)$ **на выборке**

$$(x_i, -L'(y_i, a_t(x_i)))_{i=1}^{m}$$

- $a_{t+1}(x) = a_t(x) + b_t(x)$.

Итерационно обучаем сумму алгоритмов…

Вот почему называется градиентный бустинг

Частный случай

Регрессия с СКО

$$L(y,a) = \frac{1}{2}(y-a)^2$$

$$L'(y,a) = -(y-a)$$

Задача для настройки следующего алгоритма

$$(x_i,\ y_i - a_t(x_i))_{i=1}^m$$

т.е. очень логично:

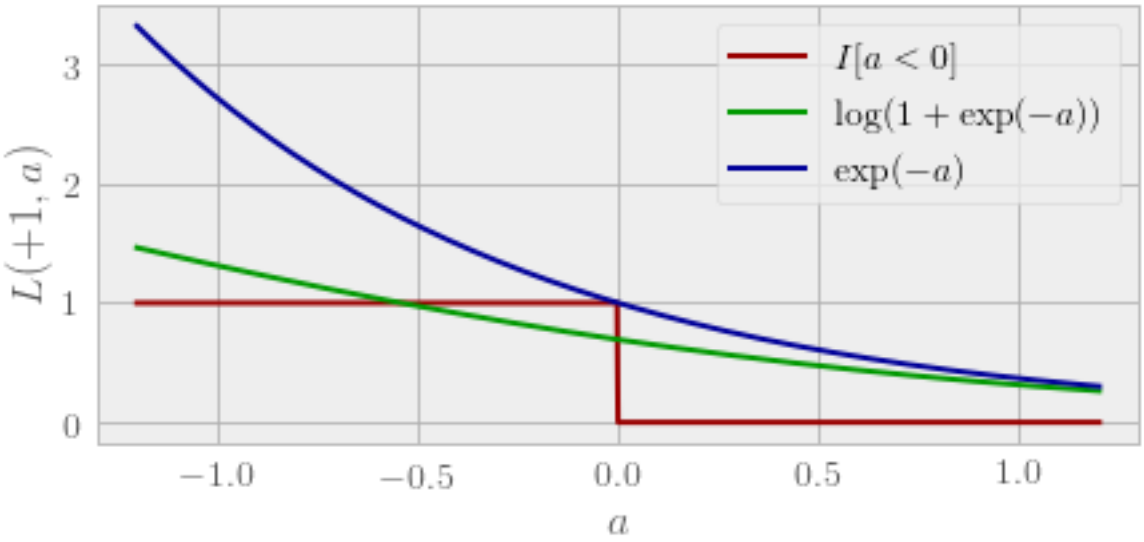
настраиваемся на невязку!

Частный случай

Классификация на два класса

надо найти дифференцируемую функцию ошибки…

- **предполагаем, что алгоритм выдаёт вещественные значения**
- **делаем функцию похожей на «совпадение»**



Частный случай

Классификация на два класса

BinomialBoost – логистическая функция ошибки:

$$L(y,a)=\log(1+e^{-y\cdot a}),\ a\in(-\infty,+\infty),\ y\in\{-1,+1\},$$

$$L'(y,a)=-\frac{y}{1+e^{-y\cdot a}}.$$

Функция ошибки типа Adaboost:

$$L(y,a)=e^{-y\cdot a},\ a\in(-\infty,+\infty),\ y\in\{-1,+1\},$$

$$L(y,a)=-ye^{-y\cdot a}.$$

здесь что-то выводится явно…

Итерация градиентного бустинга

Как решать задачу

$$(x_i, -L'(y_i, a_t(x_i)))_{i=1}^{m}?$$

Любым простым методом!

Мы уже настраиваемся на нужную функцию ошибки.

Проблема

Шаг в сторону антиградиента

- **не приводит в локальный минимум (сразу) $\Rightarrow$ итерации**
- **мы всё равно не можем сделать такой шаг, а лишь шаг по ответам какого-то алгоритма модели $\Rightarrow$ не нужно стремиться шагать именно туда**

Дальше решение проблем…

Наискорейший спуск

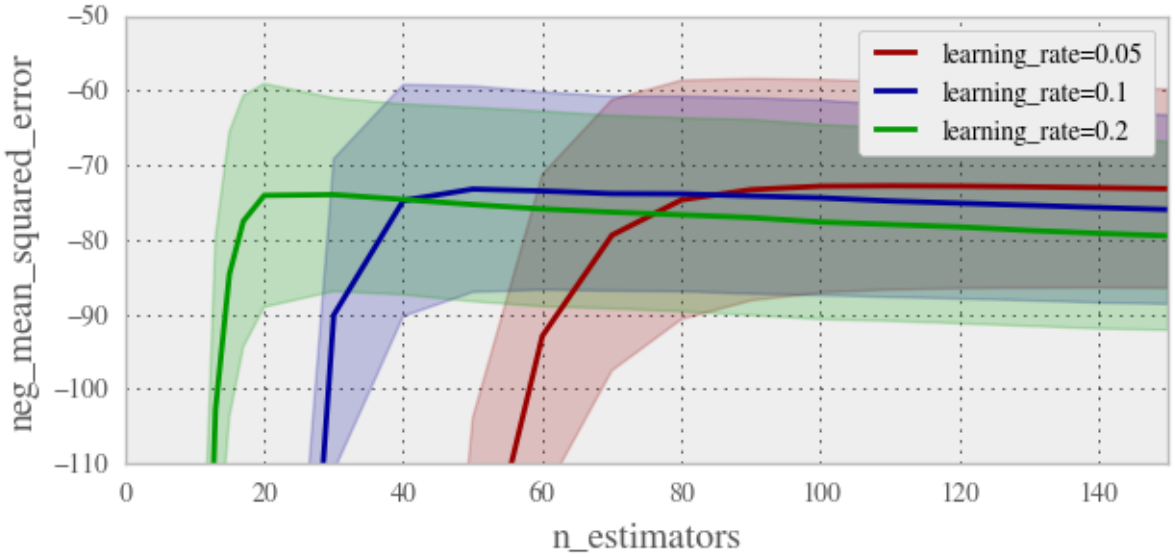
$$\sum_{i=1}^{m} L(y_i, a_t(x_i) + \eta \cdot b_t(x_i)) \to \min_{\eta},$$

$$a_{t+1}(x) = a_t(x) + \eta_t \cdot b_t(x) = \eta_1 \cdot b_1(x) + \ldots + \eta_t \cdot b_t(x)$$

Эвристика сокращения – Shrinkage

$$a_{t+1}(x) = a_t(x) + \eta \cdot b_t(x),$$

$\eta \in (0, 1]$ **– скорость (темп) обучения (learning rate)**

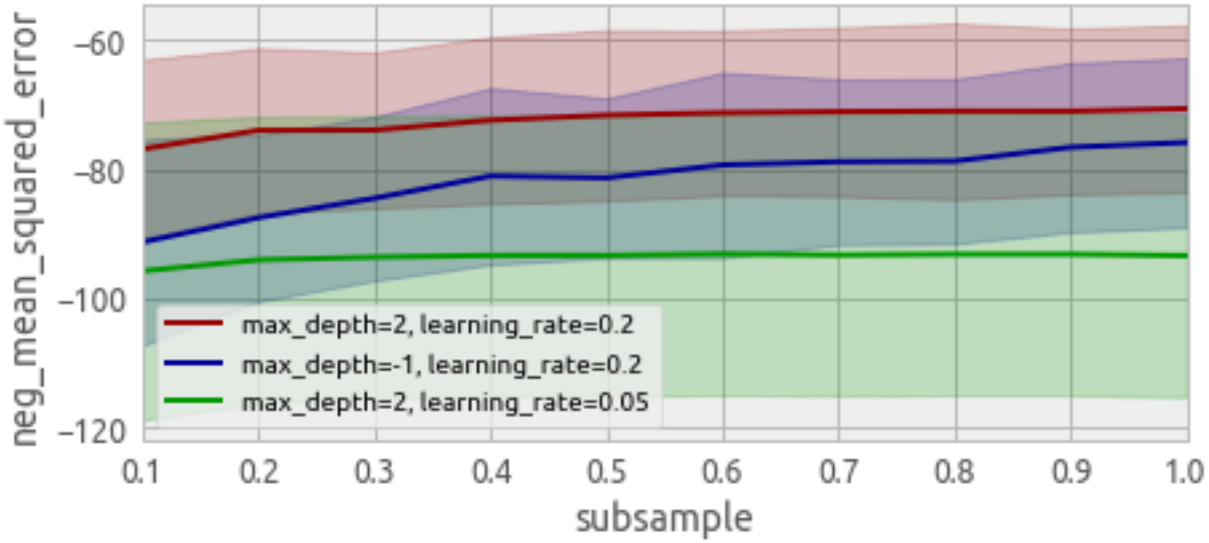


Видно, что число слагаемых (базовых алгоритмов) – шагов бустинга – надо контролировать (при увеличении можем переобучиться)

Чем меньше скорость, тем больше итераций надо

Стохастический градиентный бустинг (Stochastic gradient boosting)

Идея бэгинга Бреймана



bag fraction ~ берём часть всей выборки

- быстрее
- регуляризация
- аналог обучения по минибатчам

J. Friedman «Stochastic Gradient Boost» // 1999 http://statweb.stanford.edu/~jhf/ftp/stobst.pdf

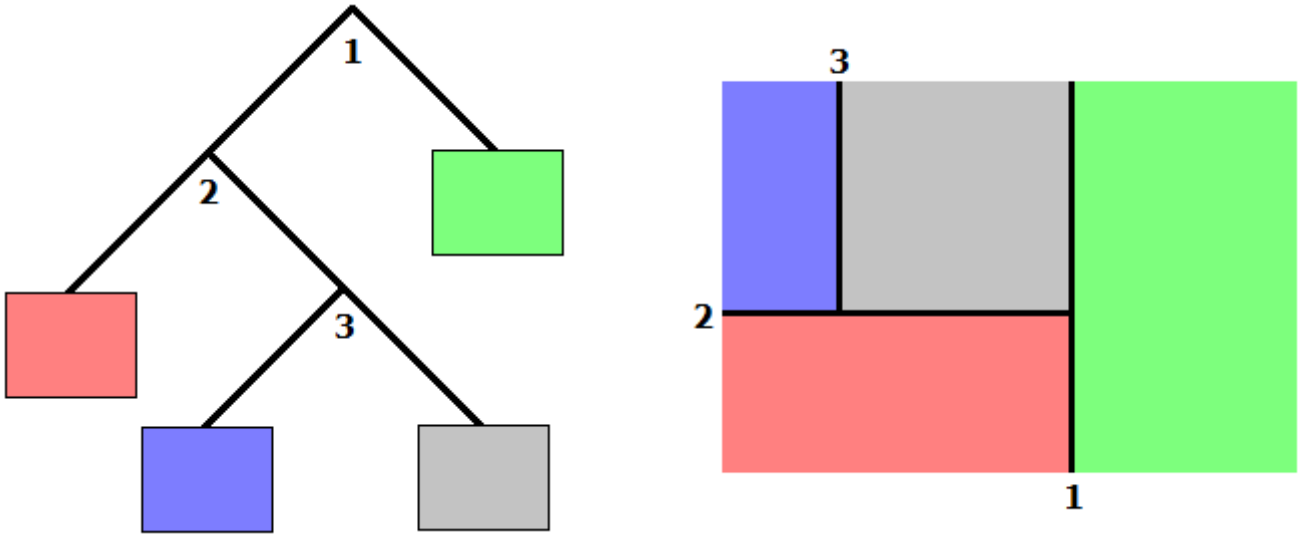
Column / Feature Subsampling for Regularization

аналогичная идея с признаками

TreeBoost – градиентный бустинг над деревьями

Решающее дерево:

$$b(x) = \sum_j \beta_j I[x \in X_j].$$



TreeBoost – градиентный бустинг над деревьями

Наша основная задача

$$\sum_{i=1}^{m} L(y_i, a(x_i) + \sum_j \beta_j I[x \in X_j]) \to \min$$

Разбиваем по областям:

$$\sum_{x_i \in X_j} L(y_i, a(x_i) + \beta_j) \to \min_{\beta_j}.$$

Продвинутые методы оптимизации

Наша основная задача

$$F = \sum_{i=1}^{m} L(y_i, a(x_i) + b_i) \to \min_{(b_1,\ldots,b_m)},$$

заметим, что

$$F = \sum_{i=1}^{m} L(y_i, a(x_i) + b_i) \approx$$

$$\sum_{i=1}^{m} \left[L(y_i, a(x_i)) + L'(y_i, a(x_i)) \cdot b_i + \frac{1}{2} L''(y_i, a(x_i)) \cdot b_i^2 \right]$$

(частные производные по второму аргументу функции ошибки)

Продвинутые методы оптимизации

$$\sum_{i=1}^{m}\left[g_i b_i + \frac{1}{2} h_i b_i^2 \right] \to \min,$$

$$g_i = L(y_i, a(x_i))',$$

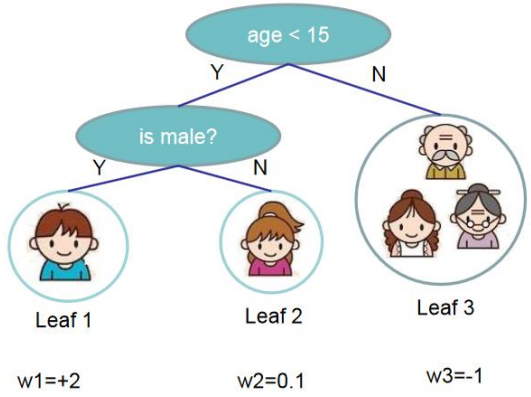
$$h_i = L''(y_i, a(x_i)).$$

Сделаем оптимизацию с регуляризацией.

Продвинутые методы оптимизации

Пусть дерево $b(x)$ делит пространство объектов на T областей $X_1,\ldots,X_T$, в каждой области X_j принимает значение β_j.

$$\Phi = \sum_{i=1}^{m}\left[g_i b_i + \frac{1}{2} h_i b_i^2 \right] + \gamma T + \lambda \frac{1}{2}\sum_{j=1}^{T} \beta_j^2 \to \min$$



$$\Omega = \gamma 3 + \tfrac{1}{2}\lambda(4 + 0.01 + 1)$$

Продвинутые методы оптимизации

$$\Phi = \sum_{j=1}^{T}\left[\sum_{x_i \in X_j}\left[g_i\beta_j + \frac{1}{2}h_i\beta_j^2\right] + \lambda\frac{1}{2}\beta_j^2\right] + \gamma T =$$

$$= \sum_{j=1}^{T}\left[\beta_j \sum_{x_i \in X_j} g_i + \frac{1}{2}\beta_j^2\left(\sum_{x_i \in X_j} h_i + \lambda\right)\right] + \gamma T$$

Приравнивая производную к нулю:

$$\beta_j = -\frac{\sum_{x_i \in X_j} g_i}{\sum_{x_i \in X_j} h_i + \lambda}.$$

Продвинутые методы оптимизации

Минимальное значение (при фиксированной структуре дерева)

$$\Phi_{\min} = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum\limits_{x_i \in X_j} g_i\right)^2}{\sum\limits_{x_i \in X_j} h_i + \lambda} + \gamma T.$$

Можно использовать при построении дерева для его оценки.

Не используем какой-то традиционный критерий расщепления

Исходим из функции ошибки!

Параметры градиентного бустинга

- `objective` – **параметры определяющие, какая задача решается и в каком формате будет ответ**
- `eval_metric` – **значения какой функции ошибки смотреть на контроле** (как правило, задание этого параметра не означает, что эту функцию будем минимизировать при настройке бустинга)

Параметры, определяющие тип бустинга

- `booster` – **какой бустинг проводить: над решающими деревьями или линейный**
- **способы построения деревьев** (`grow_policy` – порядок построения дерева: на следующем шаге расщеплять вершину, ближайшую к корню, или на которой ошибка максимальна)

параметр «распределение» см. дальше

Основные параметры:

- `eta / learning_rate` – **темп (скорость) обучения**
- `num_iterations / n_estimators` – **число итераций бустинга**
- `early_stopping_round` – **если на отложенном контроле заданная функция ошибки не уменьшается такое число итераций, обучение останавливается**

Параметры ограничивающие сложность дерева:

- `max_depth` – **максимальная глубина**
- `max_leaves / num_leaves` – **максимальное число вершин в дереве**
- `gamma / min_gain_to_split` – **порог на уменьшение функции ошибки при расщеплении в дереве**
- `min_data_in_leaf` – **минимальное число объектов в листе**
- `min_sum_hessian_in_leaf` – **минимальная сумма весов объектов в листе, минимальное число объектов, при котором делается расщепление**

Параметры формирования подвыборок

- `subsample / bagging_fraction` – **какую часть объектов обучения использовать для построения одного дерева**
- `colsample_bytree / feature_fraction` – **какую часть признаков использовать для построения одного дерева**
- `colsample_bylevel` – **какую часть признаков использовать для построения расщепления в дереве**

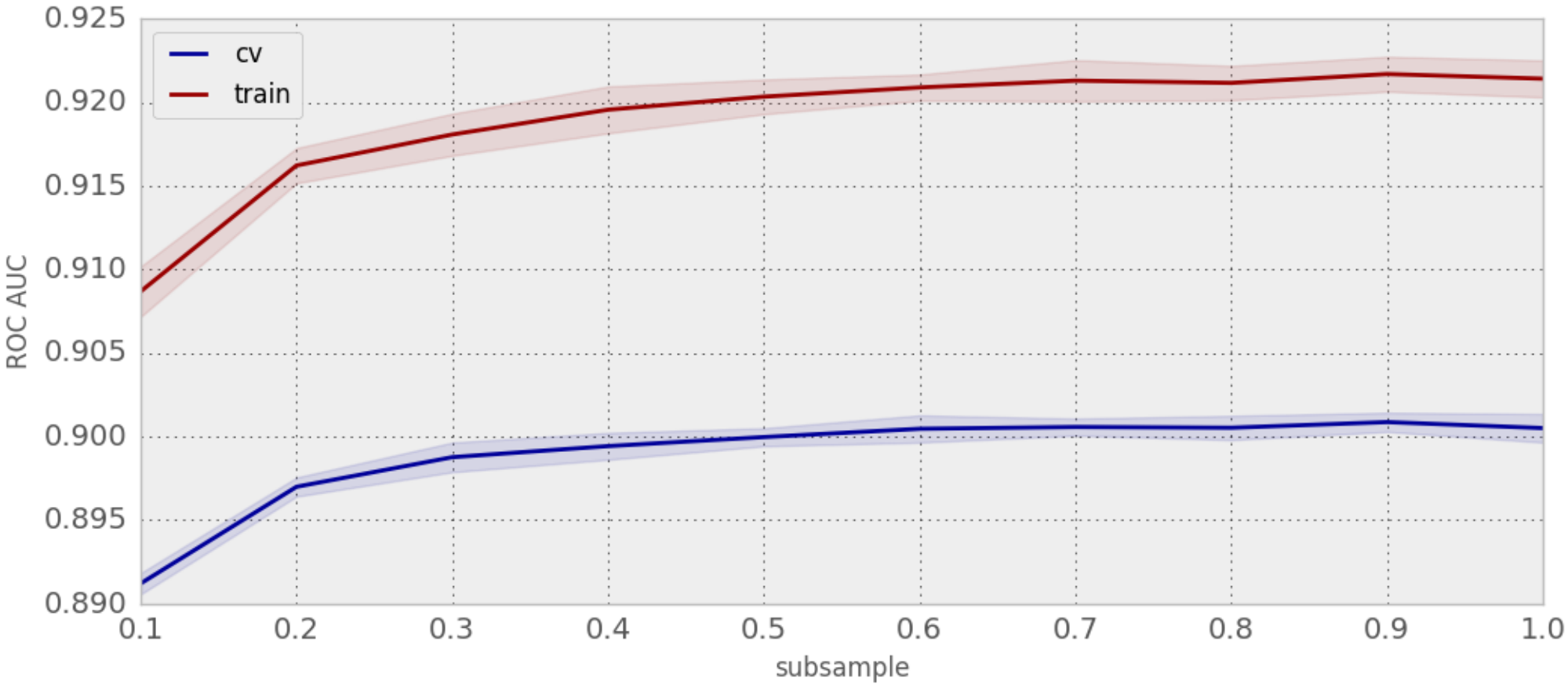
Параметры регуляризации:

- `lambda / lambda_l2` **(L2)**
- `alpha / lambda_l1` **(L1)**

Параметры которые помогают обучать бустинг быстрее:

- **число используемых потоков**
- **CPU / GPU**
- **хранить модель в ОЗУ**
- **метод поиска расщепления**

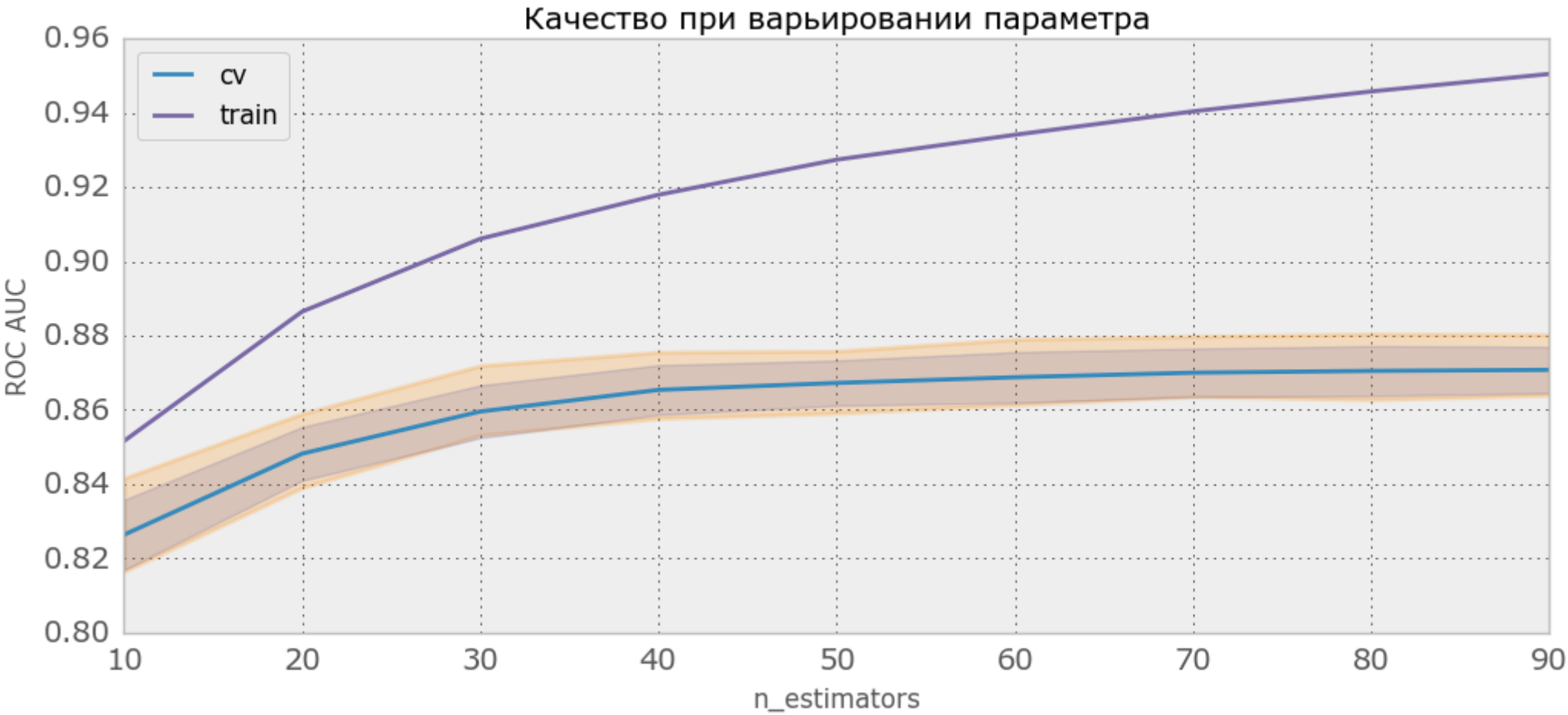
Объём выборки `subsample` (ed Бозон)



Опять, больше – лучше

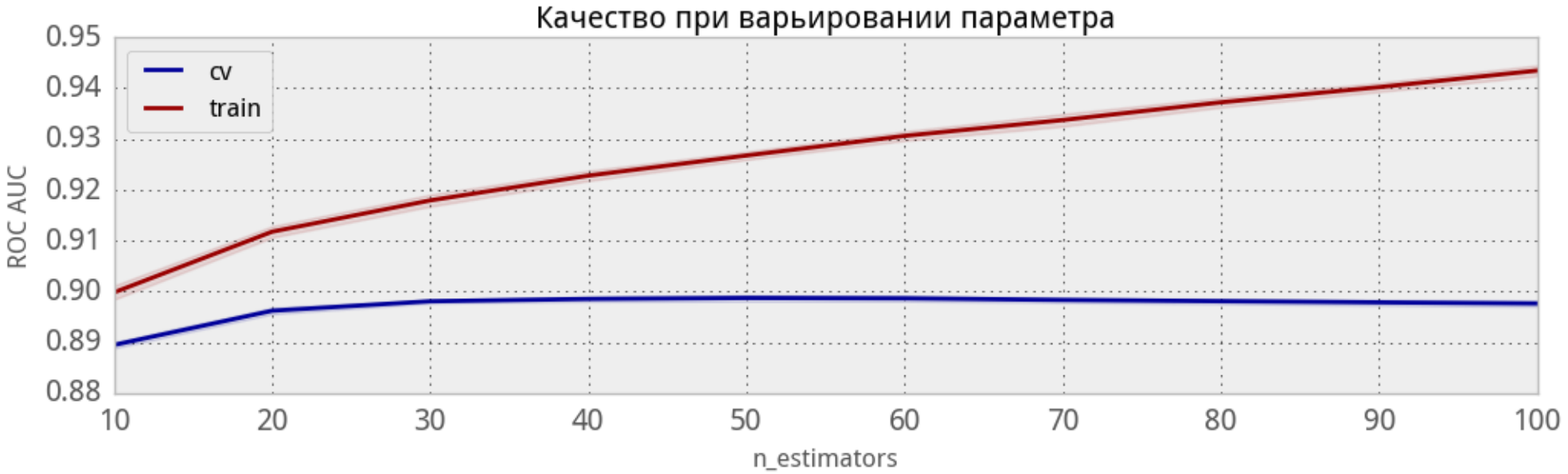
(в XGBoost это не всегда так)

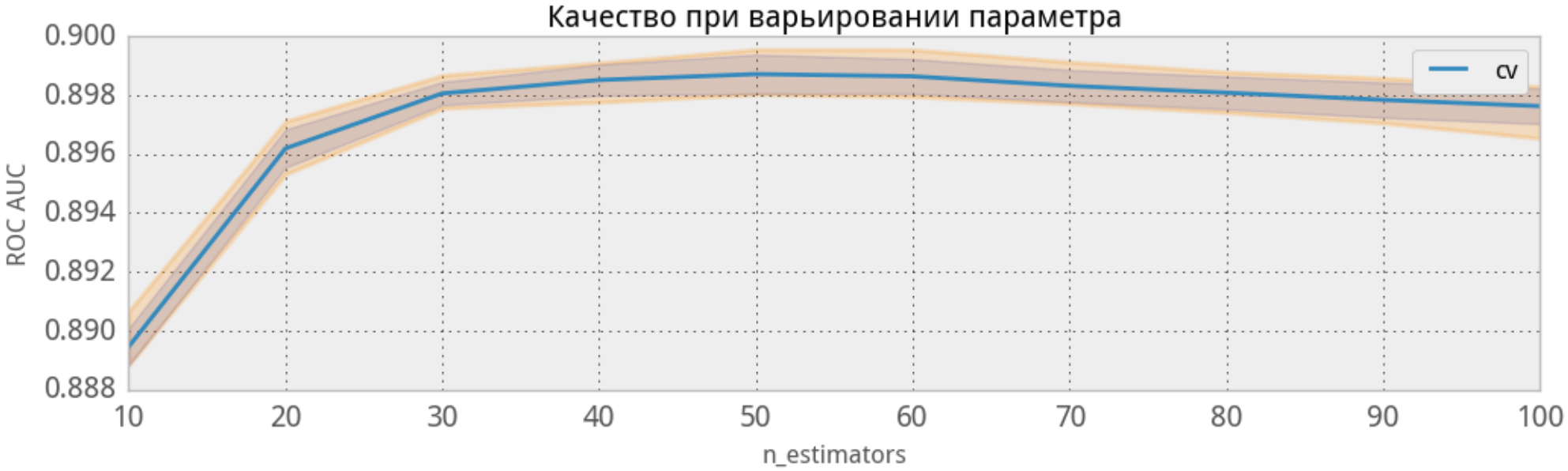
Число деревьев: n_estimators (ed Сбербанк)



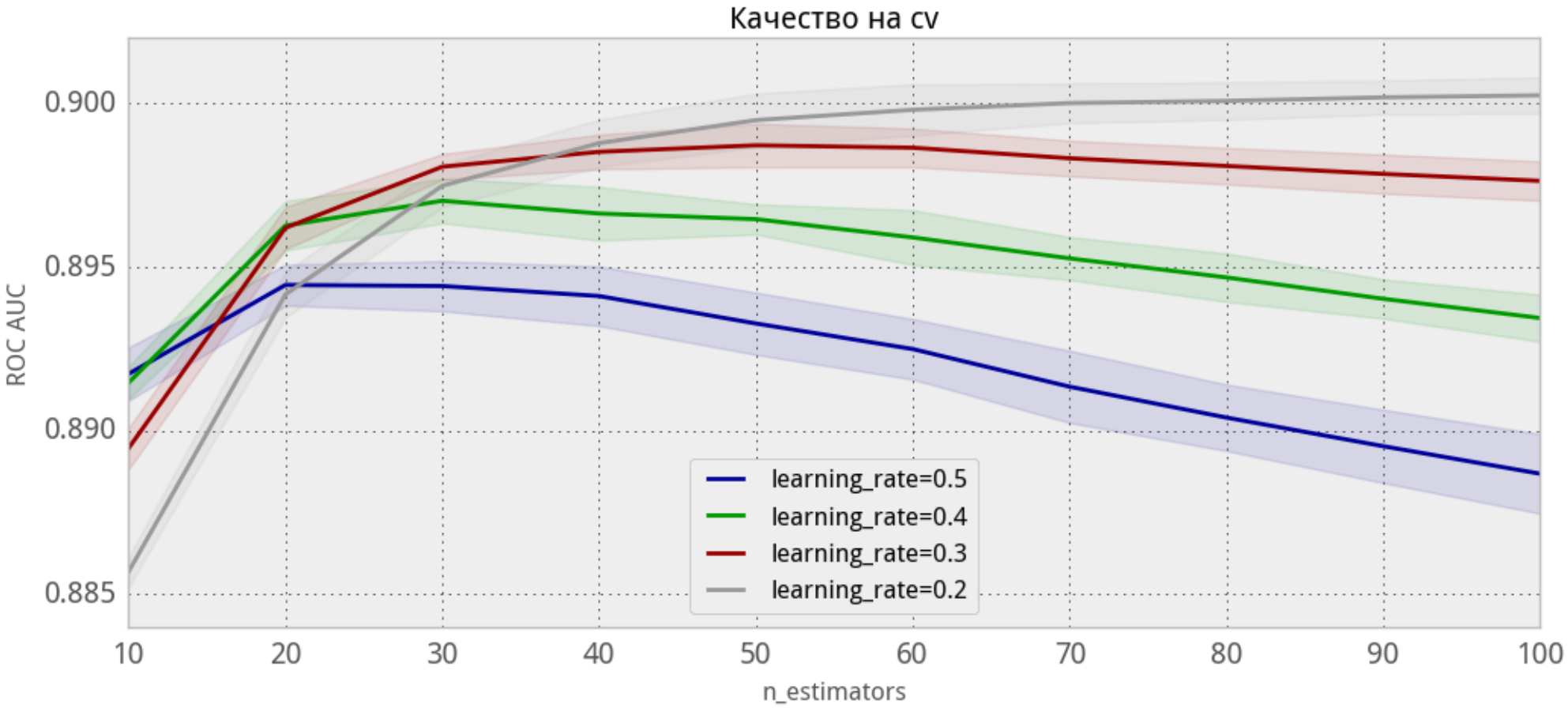
Здесь уже нет логики «чем больше, тем лучше»

Число деревьев: n_estimators (ed Бозон)





Темп обучения learning_rate (ed Бозон)



Темп обучения `learning_rate`

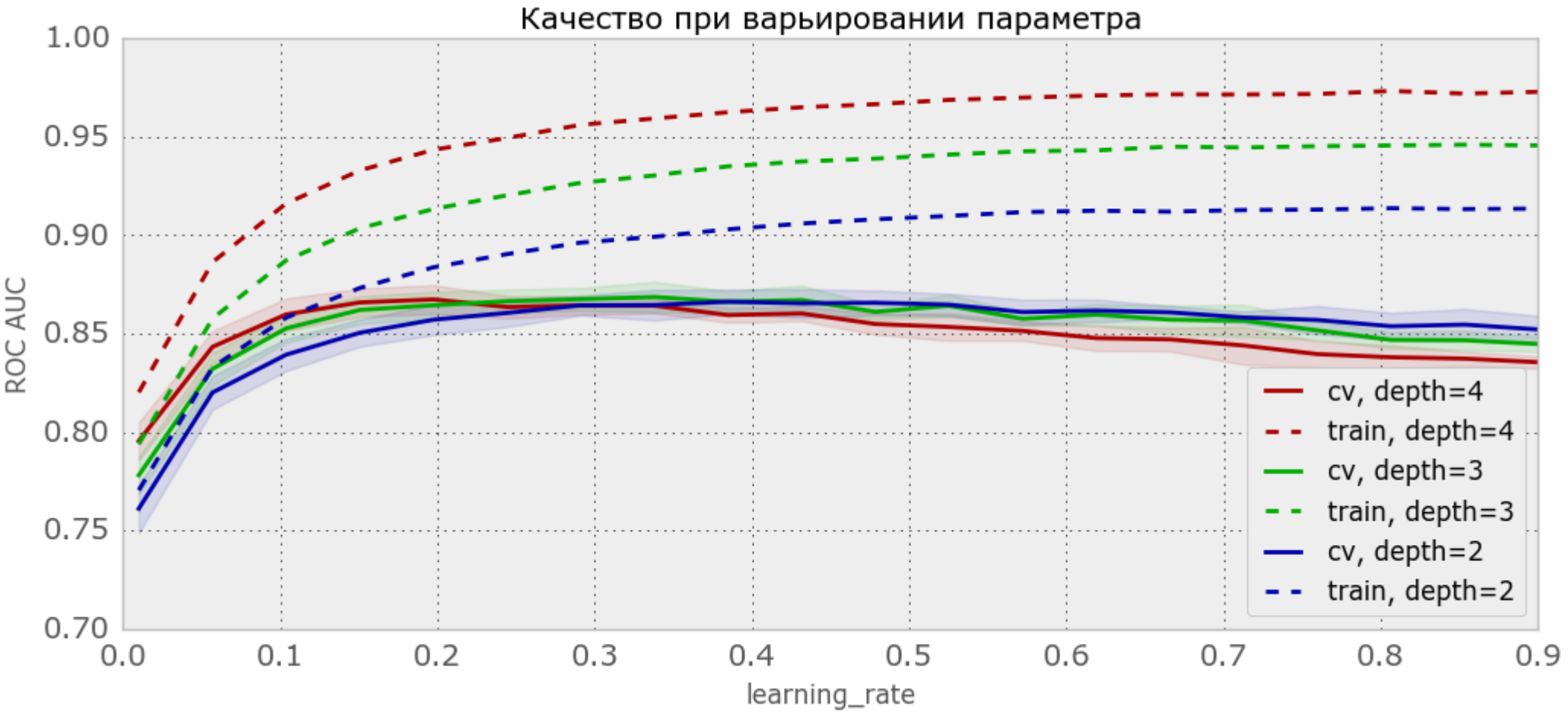
Нет логики «уменьшили темп в 2 раза – число деревьев надо увеличить в 2 раза»!

Есть стратегия – сделать очень маленький темп и очень много деревьев (но для настройки других параметров не годится)

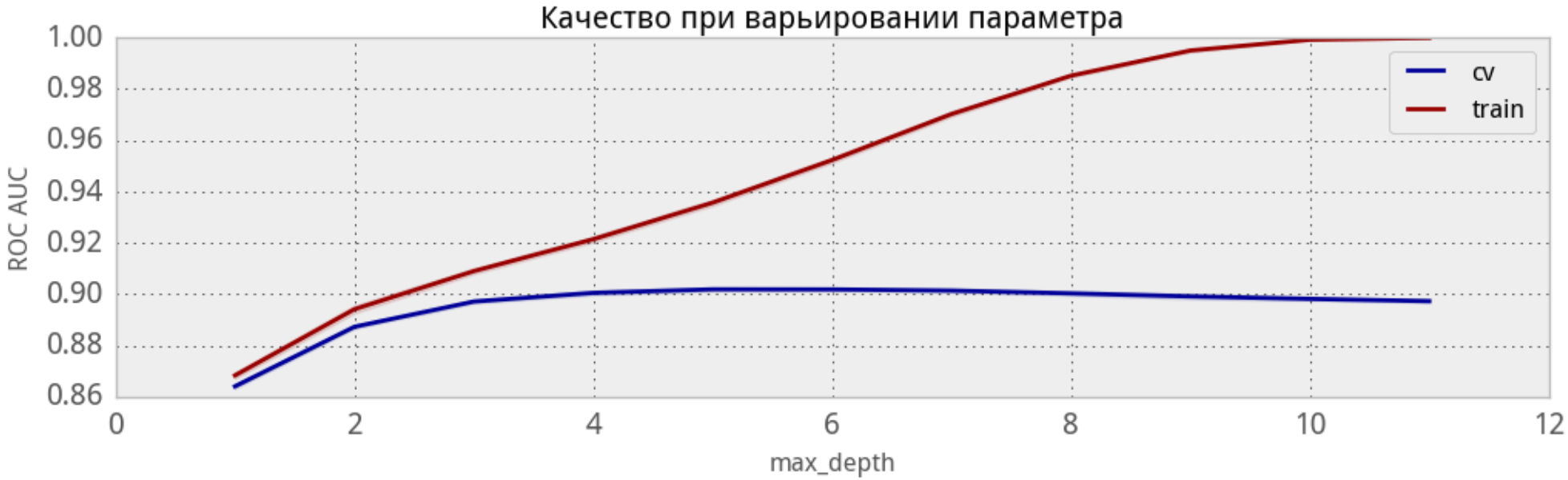
Совет:

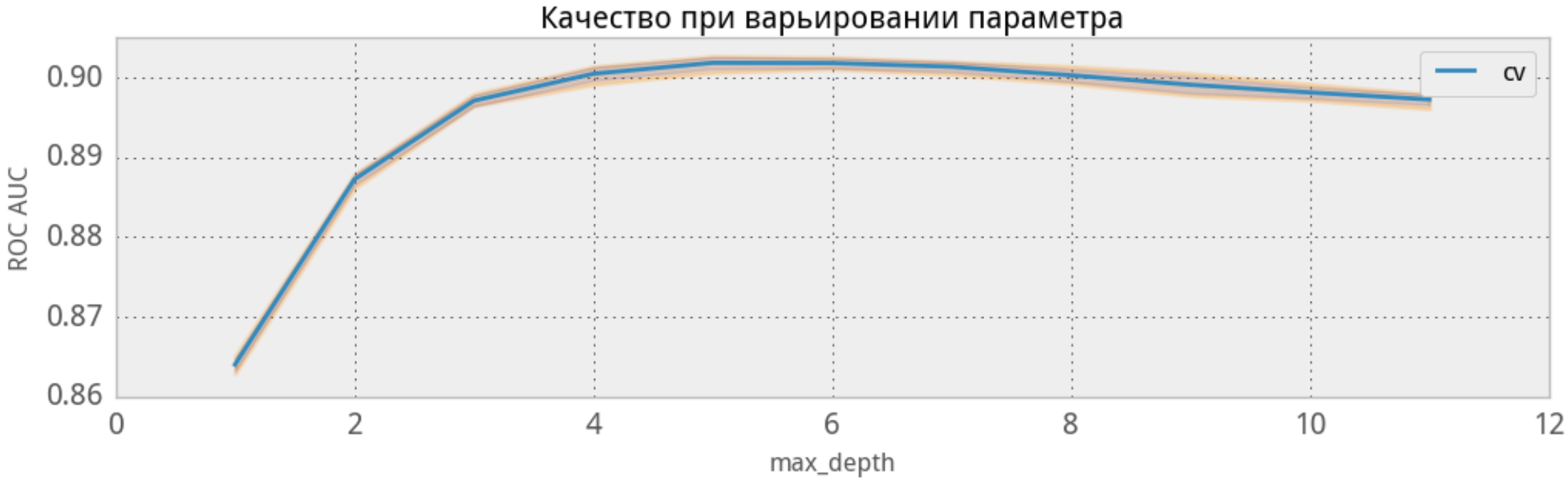
- **зафиксируйте достаточно большое число деревьев, которое ещё можно быстро построить**
- **настройте `learning_rate`**
- **настраивайте другие параметры (первым делом – глубину), но помните, что оптимальный темп может поменяться!**

Темп обучения learning_rate



Глубина деревьев



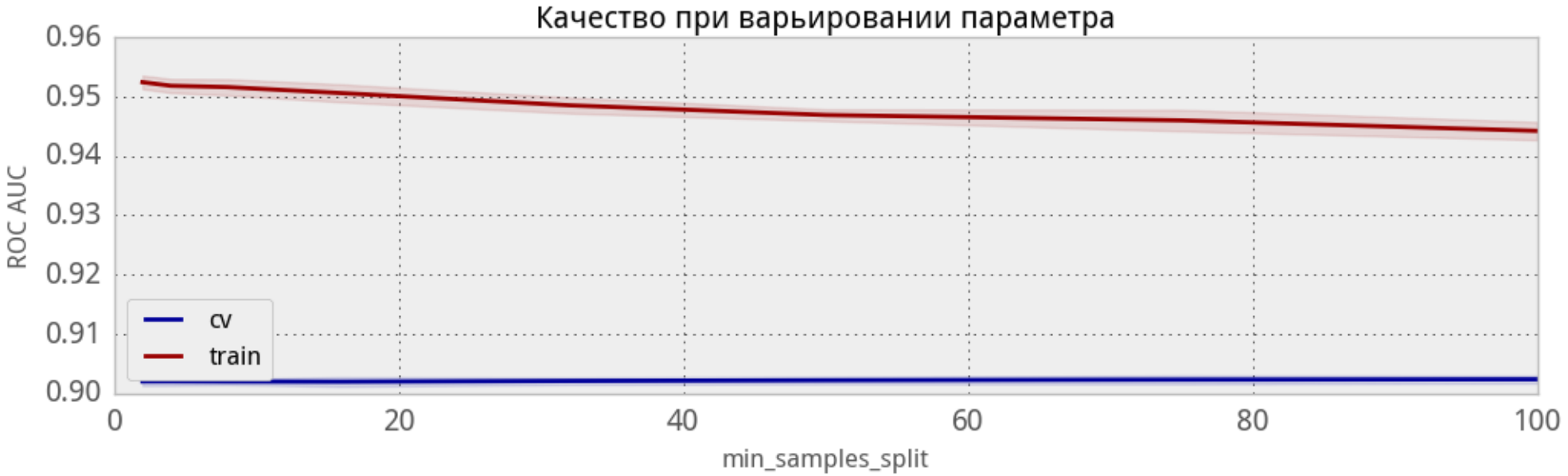


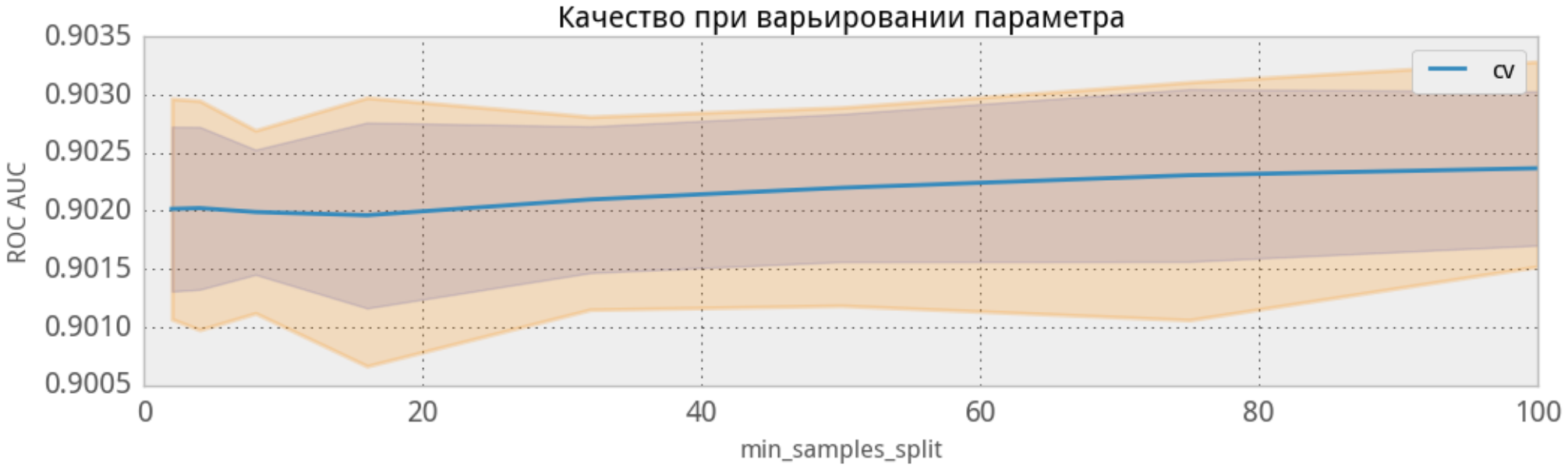
Глубина деревьев

Здесь есть понятие оптимальной глубины!

Как правило, строят неглубокие деревья (3 – 6).

Ограничение на расщепления / листья



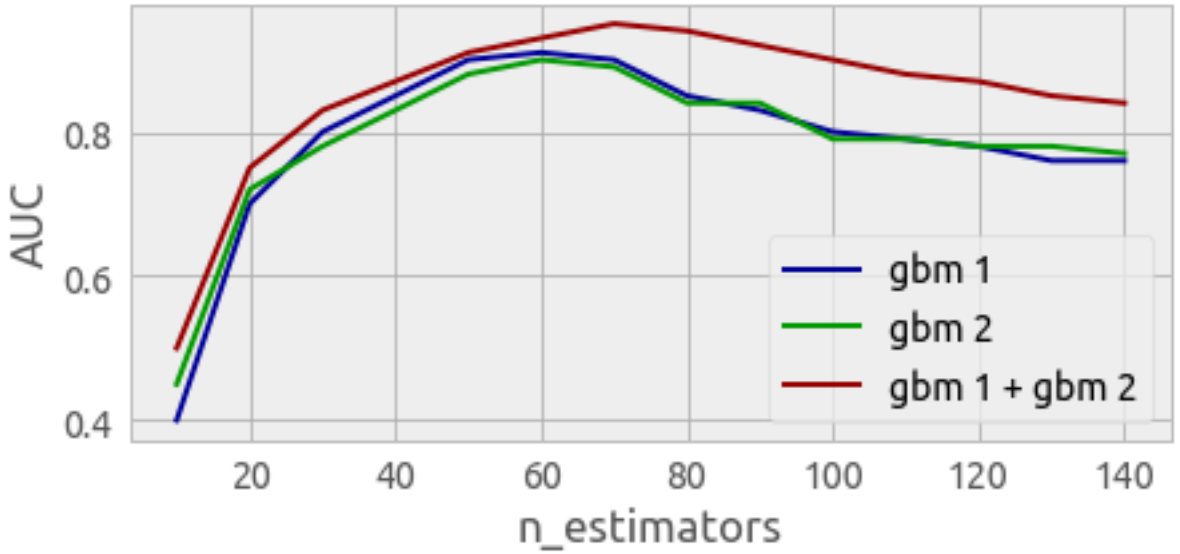


Ограничение на расщепления / листья

Здесь могут быть большие оптимальные значения (10 – 50),

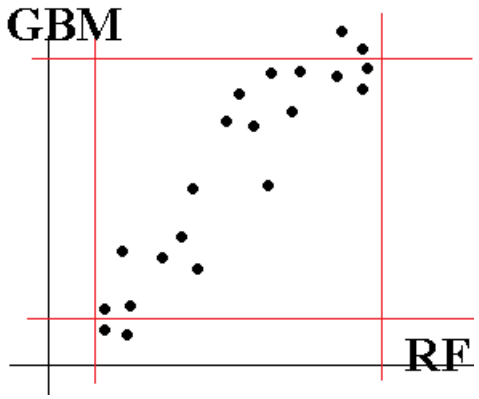
но параметры менее значимые, чем другие…

GBM можно усреднять!



Качество может улучшиться,
но оптимальные параметры меняются!

Важно



Значения gbm могут выходить за пределы отрезка!

Литература

A. Liaw, M. Wiener Classification and Regression by randomForest // R News (2002) Vol. 2/3 p. 18.
http://www.bios.unc.edu/~dzeng/BIOS740/randomforest.pdf

И. Генрихов О критериях ветвления, используемых при синтезе решающих деревьев // Машинное обучение и анализ данных, 2014, Т.1, №8, С.988-1017
http://jmlda.org/papers/doc/2014/no8/Genrikhov2014Criteria.pdf

A. Natekin, A. Knoll Gradient boosting machines, a tutorial // Front Neurorobot. 2013; 7: 21.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/