**Случайный лес (Random Forest)**

[2016/11/14](https://alexanderdyakonov.wordpress.com/2016/11/14/%d1%81%d0%bb%d1%83%d1%87%d0%b0%d0%b9%d0%bd%d1%8b%d0%b9-%d0%bb%d0%b5%d1%81-random-forest/) [alexanderdyakonov](https://alexanderdyakonov.wordpress.com/author/alexanderdyakonov/) [код](https://alexanderdyakonov.wordpress.com/category/%d0%ba%d0%be%d0%b4/)[оптимизация](https://alexanderdyakonov.wordpress.com/tag/%d0%be%d0%bf%d1%82%d0%b8%d0%bc%d0%b8%d0%b7%d0%b0%d1%86%d0%b8%d1%8f/), [параметры](https://alexanderdyakonov.wordpress.com/tag/%d0%bf%d0%b0%d1%80%d0%b0%d0%bc%d0%b5%d1%82%d1%80%d1%8b/), [случайный лес](https://alexanderdyakonov.wordpress.com/tag/%d1%81%d0%bb%d1%83%d1%87%d0%b0%d0%b9%d0%bd%d1%8b%d0%b9-%d0%bb%d0%b5%d1%81/),[python](https://alexanderdyakonov.wordpress.com/tag/python/), [random forest](https://alexanderdyakonov.wordpress.com/tag/random-forest/), [scikit-learn](https://alexanderdyakonov.wordpress.com/tag/scikit-learn/)

Случайный лес — один из самых потрясающих алгоритмов машинного обучения, придуманные Лео Брейманом и Адель Катлер ещё в прошлом веке. Он дошёл до нас в «первозданном виде» (никакие эвристики не смогли его существенно улучшить) и является одним из немногих универсальных алгоритмов. Универсальность заключается, во-первых, в том, что он хорош во многих задачах (по моим оценкам, 70% из встречающихся на практике, если не учитывать задачи с изображениями), во-вторых, в том, что есть случайные леса для решения задач классификации, регрессии, кластеризации, поиска аномалий, селекции признаков и т.д.



Этот пост — краткое практическое руководство для новичков — путеводитель по  основным параметрам алгоритма с картинками (которые, кстати, построены на данных последнего [конкурса Сбербанка](https://alexanderdyakonov.wordpress.com/2016/10/14/data-science-contest-%d1%81%d0%b1%d0%b5%d1%80%d0%b1%d0%b0%d0%bd%d0%ba%d0%b0/) и одной модельной задачи). Под тестом здесь понимается результат на скользящем контроле (для построения графиков использовано 5 фолдов), хотя для отложенного контроля (hold out) выводы будут такими же. Графики лежат в коридорах: дисперсионном и (если есть второй коридор) макс-минном. Все выводы и рекомендации — общие — не для конкретной задачи.

**RF (random forest)** — это множество решающих деревьев. В задаче регрессии их ответы усредняются, в задаче классификации принимается решение голосованием по большинству. Все деревья строятся независимо по следующей схеме:

* Выбирается подвыборка обучающей выборки размера samplesize (м.б. с возвращением) – по ней строится дерево (для каждого дерева — своя подвыборка).
* Для построения каждого расщепления в дереве просматриваем max\_features случайных признаков (для каждого нового расщепления — свои случайные признаки).
* Выбираем наилучшие признак и расщепление по нему (по заранее заданному критерию). Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но в современных реализациях есть параметры, которые ограничивают высоту дерева, число объектов в листьях и число объектов в подвыборке, при котором проводится расщепление.

Понятно, что такая схема построения соответствует главному принципу [ансамблирования](https://en.wikipedia.org/wiki/Ensemble_learning" \t "_blank) (построению алгоритма машинного обучения на базе нескольких, в данном случае решающих деревьев): базовые алгоритмы должны быть хорошими и разнообразными (поэтому каждое дерево строится на своей обучающей выборке и при выборе расщеплений есть элемент случайности).

В библиотеке [scikit-learn](http://scikit-learn.org/stable/" \t "_blank) есть такая реализация RF (привожу только для задачи классификации):

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, min\_impurity\_split=1e-07,

bootstrap=True, oob\_score=False, n\_jobs=1,

random\_state=None, verbose=0, warm\_start=False,

class\_weight=None)

С алгоритмом работают по стандартной схеме, принятой в scikit-learn:

from **sklearn.ensemble** import **RandomForestRegressor**

from **sklearn.metrics** import **roc\_auc\_score**

# далее - (X, y) - обучение, (X2, y2) - контроль

# модель - здесь (для контраста) рассмотрим регрессор

model = **RandomForestRegressor**(n\_estimators=10 ,

oob\_score=True,

random\_state=1)

model.**fit**(X, y) # обучение

a = model.**predict**(X2) # предсказание

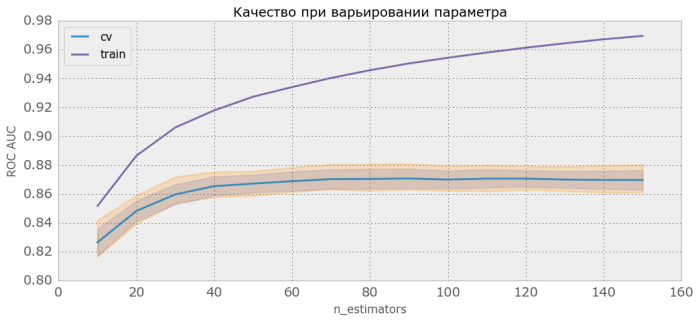
print ("AUC-ROC (oob) = ", roc\_auc\_score(y, model.oob\_prediction\_))

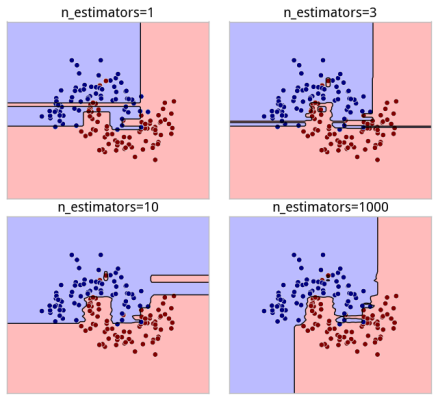
print ("AUC-ROC (test) = ", roc\_auc\_score(y2, a))

Опишем, что означают основные параметры:

**Число деревьев — n\_estimators**

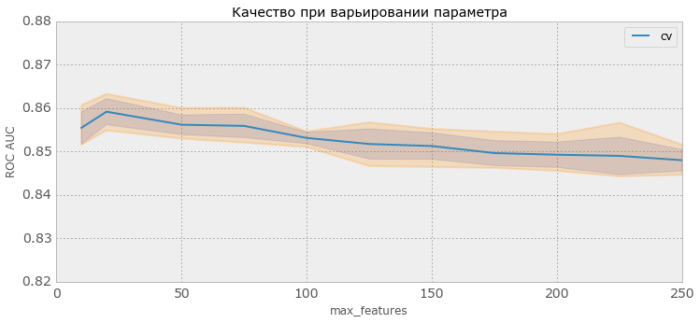
Чем больше деревьев, тем лучше качество, но время настройки и работы RF также пропорционально увеличиваются. Обратите внимание, что часто при увеличении n\_estimators качество на обучающей выборке повышается (может даже доходить до 100%), а качество на тесте выходит на асимптоту (можно прикинуть, скольких деревьев Вам достаточно).

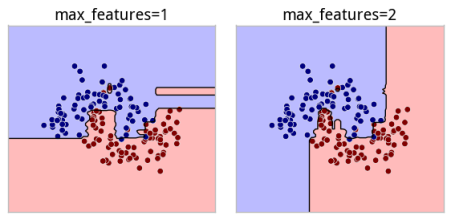




**Число признаков для выбора расщепления — max\_features**

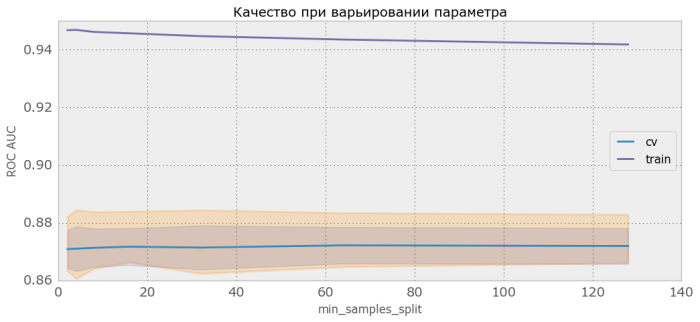
График качества на тесте от значения этого праметра унимодальный, на обучении он строго возрастает. При увеличении max\_features увеличивается время построения леса, а деревья становятся «более однообразными». По умолчанию он равен sqrt(n) в задачах классификации и n/3 в задачах регрессии. Это самый важный параметр! Его настраивают в первую очередь (при достаточном числе деревьев в лесе).

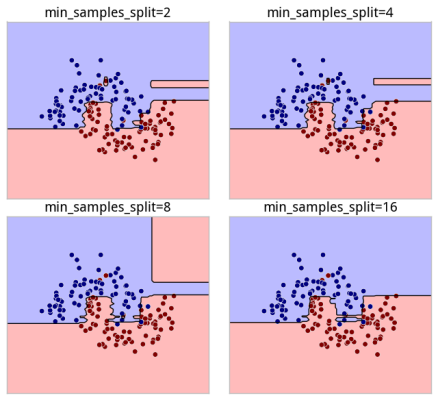




**Минимальное число объектов, при котором выполняется расщепление — min\_samples\_split**

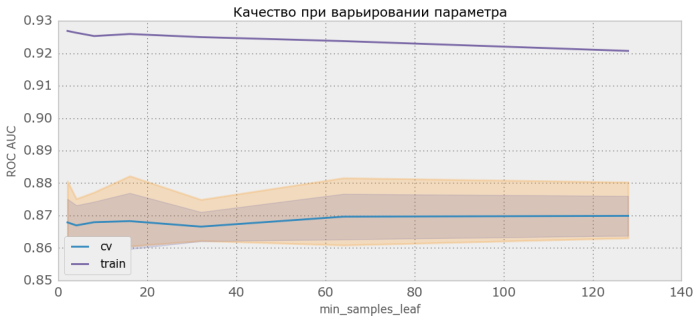
Этот параметр, как правило, не очень важный и можно оставить значение по умолчанию (2). График качества на контроле может быть похожим на «расчёску» (нет явного оптимума). При увеличении параметра качество на обучении падает, а время построения RF сокращается.

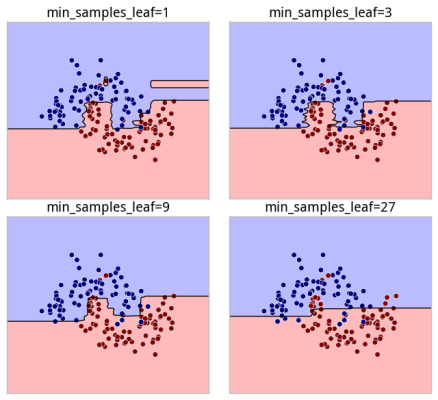




**Ограничение на число объектов в листьях — min\_samples\_leaf**

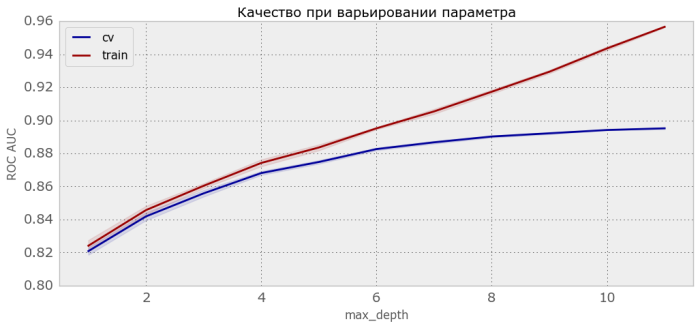
Всё, что было описано про min\_samples\_split, годится и для описания этого параметра. Часто можно оставить значение по умолчанию (1). Кстати, по классике, в задачах регрессии рекомендуется использовать значение 5 (в библиотеке [randomForest](https://cran.r-project.org/web/packages/randomForest/index.html" \t "_blank) для R так и реализовано, в sklearn — 1).

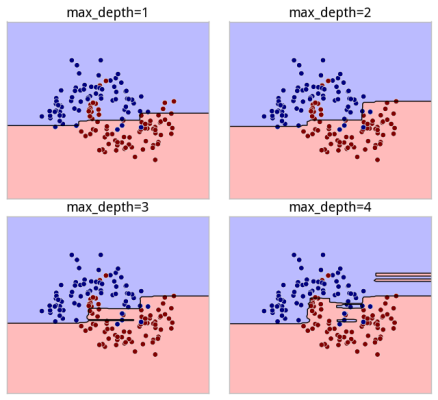




**Максимальная глубина деревьев — max\_depth**

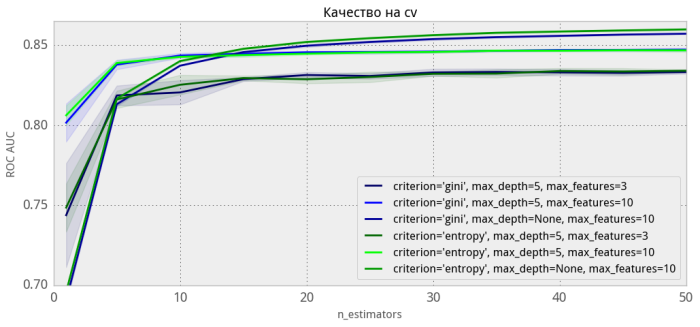
Ясно, что чем меньше глубина, тем быстрее строится и работает RF. При увеличении глубины резко возрастает качество на обучении, но и на контроле оно, как правило, увеличивается. Рекомендуется использовать максимальную глубину (кроме случаев, когда объектов слишком много и получаются очень глубокие деревья, построение которых занимает значительное время). При использовании неглубоких деревьев изменение параметров, связанных с ограничением числа объектов в листе и для деления, не приводит к значимому эффекту (листья и так получаются «большими»). Неглубокие деревья рекомендуют использовать в задачах с большим числом шумовых объектов (выбросов).

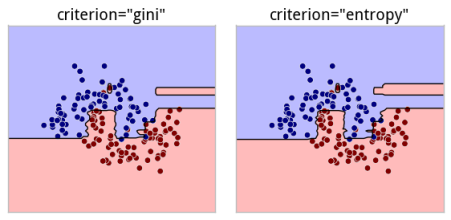




**Критерий расщепления — criterion**

По смыслу это очень важный параметр, но по факту здесь нет вариантов выбора. В библиотеке sklearn для регрессии реализованы два критерия: “mse” и “mae”, соответствуют функциям ошибки, которые они минимизируют. В большинстве задач используется mse. Сравнить их пока не берусь, т.к. mae появился совсем недавно — в версии 0.18 (и по-моему, реализован с ошибкой). Для классификации реализованы критерии “gini” и “entropy”, которые соответствуют классическим критериям расщепления: [Джини](https://alexanderdyakonov.wordpress.com/2015/12/15/%d0%b7%d0%bd%d0%b0%d0%ba%d0%be%d0%bc%d1%8c%d1%82%d0%b5%d1%81%d1%8c-%d0%b4%d0%b6%d0%b8%d0%bd%d0%b8/) и энтропийному. Простой перебор поможет Вам выбрать, что использовать в конкретной задаче (в авторской реализации алгоритма использовался Джини). Подробнее о критериях надо писать отдельный пост;)





В sklearn-реализации случайного леса нет параметра samplesize, который регламентирует, из скольких объектов делать подвыборку для построения каждого дерева. Такой параметр есть в R-реализации, но, по сути, часто оптимально выбирать из всей выборки. Также рекомендуется выбирать подвыборку с возвращением: bootstrap=True (это и есть бэггинг — bootstrap aggregating).

**Совет**

По умолчанию в sklearn-овских методах n\_jobs=1, т.е. случайный лес строится на одном процессоре. Если Вы хотите существенно ускорить построение, используйте n\_jobs=-1 (строить на максимально возможном числе процессоров). Для построения воспроизводимых экспериментов используйте предустановку генератора псевдослучайных чисел: random\_state.

П.С. Метод RF хорош ещё тем, что при построении леса параллельно может вычисляться т.н. oob-оценка качества алгоритма (которая очень точная и получается не в ущерб разделения на обучение/тест), oob-ответы алгоритмы (ответы, которые выдавал бы алгоритм на обучающей выборке, если бы «обучался не на ней»), оцениваются важности признаков (но об этом, опять же, надо писать в отдельном посте). Ну, и не стоит забывать про [полный перебор значений параметров](http://scikit-learn.org/stable/modules/grid_search.html) (если объектов в задаче не очень много).