Лекция 9

- Ансамбли (стекинг, бэггинг, бустинг)
- Кривая обучения
- Калибровка классификаторов
- Генерация случайных наборов данных

Ансамбли

Используют для:

- ✓ Всего, где подходят классические алгоритмы (но работают точнее)
- ✓ Поисковые системы
- ✓ Компьютерное зрение

Ансамблем (Ensemble, Multiple Classifier System) называется алгоритм, который состоит из нескольких алгоритмов машинного обучения, а процесс построения ансамбля называется ансамблированием (ensemble learning).

Ансамбль алгоритмов используется с целью получения лучшей эффективности прогнозирования, чем можно было бы получить от каждого обучающего алгоритма по отдельности.

Простейший пример ансамбля в регрессии – усреднение нескольких алгоритмов:

$$a(x) = \frac{1}{n}\left(b_1(x) + \ldots + b_n(x)\right) \qquad (1)$$

Алгоритмы из которых состоит ансамбль (в (1) – bt) называются базовыми алгоритмами (base learners). Если рассматривать значения базовых алгоритмов на объекте, как независимые случайные величины с одинаковым матожиданием и одинаковой конечной дисперсией, то понятно, что случайная величина (1) имеет такое же матожидание, но меньшую дисперсию:

$$\xi = \frac{1}{n}\left(\xi_1 + \ldots + \xi_n\right)$$

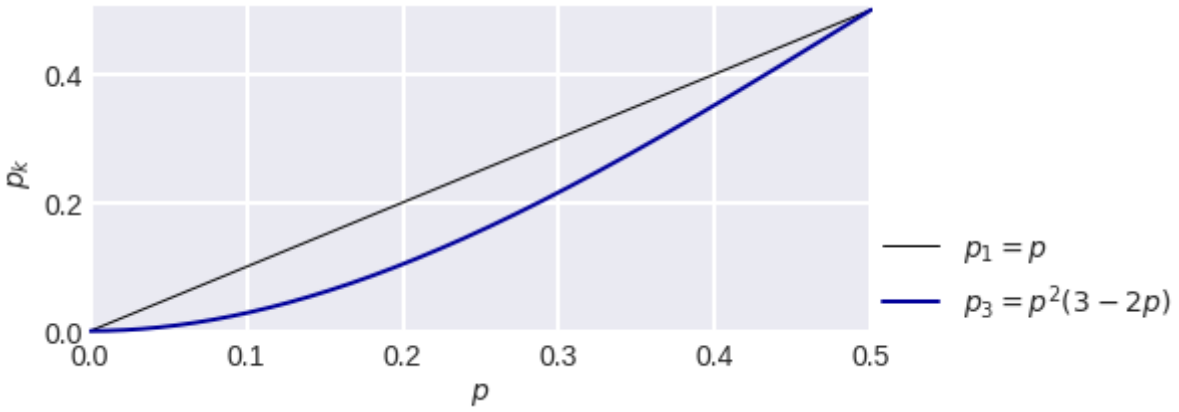
$$\mathbf{E}\xi = \frac{1}{n}\left(\mathbf{E}\xi_1 + \ldots + \mathbf{E}\xi_n\right) = \mathbf{E}\xi_i$$

$$\mathbf{D}\xi = \frac{1}{n^2}\left(\mathbf{D}\xi_1 + \ldots + \mathbf{D}\xi_n\right) = \frac{\mathbf{D}\xi_i}{n}$$

В задачах классификации простейший пример ансамбля – комитет большинства:

$$a(x) = \mathrm{mode}\left(b_1(x), \ldots, b_n(x)\right), \qquad (2)$$

где **mode** – мода (значение, которое встречается чаще других среди аргументов функции). Если рассмотреть задачу классификации с двумя классами {0, 1} и три алгоритма, каждый из которых ошибается с вероятностью p, то в предположении, что их ответы – независимые случайные величины, получаем, что комитет большинства этих трёх алгоритмов ошибается с вероятностью ***pp(3-2p)***. Как видно на рис., это выражение может быть существенно меньше p (при p=0.1 почти в два раза), т.е. использование такого ансамбля уменьшает ошибку базовых алгоритмов.



Если рассмотреть **большее число алгоритмов**, то по неравенству Хёфдинга **ошибка комитета большинства экспоненциально убывает с ростом числа базовых алгоритмов:**

$$\sum_{t=0}^{\lfloor n/2 \rfloor} C_n^t (1-p)^t p^{n-t} \leq e^{-\frac{1}{2}n(2p-1)^2}.$$

Для того чтобы ошибки отдельных алгоритмов на отдельных объектах компенсировались корректной работой других алгоритмов, **классификаторы должны быть независимыми**. НО в реальной практической ситуации, это не так, потому что классификаторы:

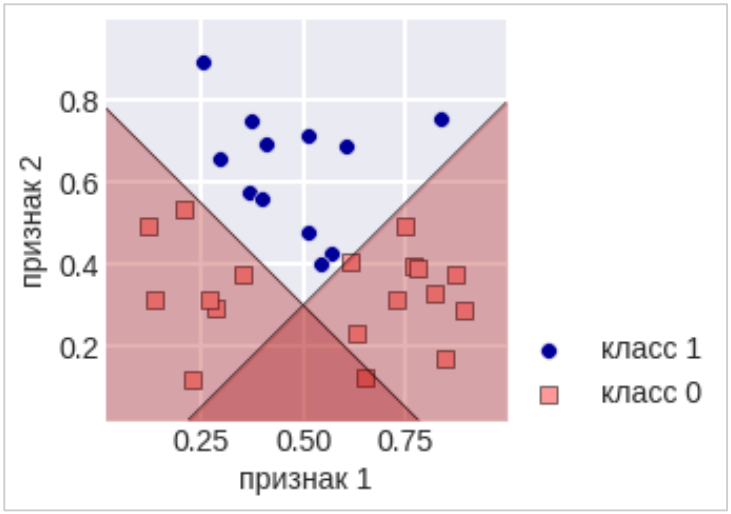
- решают одну задачу,
- настраиваются на один целевой вектор,
- могут быть из одной модели (или из нескольких, но всё равно небольшого числа).

Поэтому большинство приёмов в прикладном ансамблировании направлено на то, чтобы сделать ансамбль «достаточно разнообразным», По сути, при построении ансамбля:

- повышают качество базовых алгоритмов,
- повышают разнообразие (diversity) базовых алгоритмов.

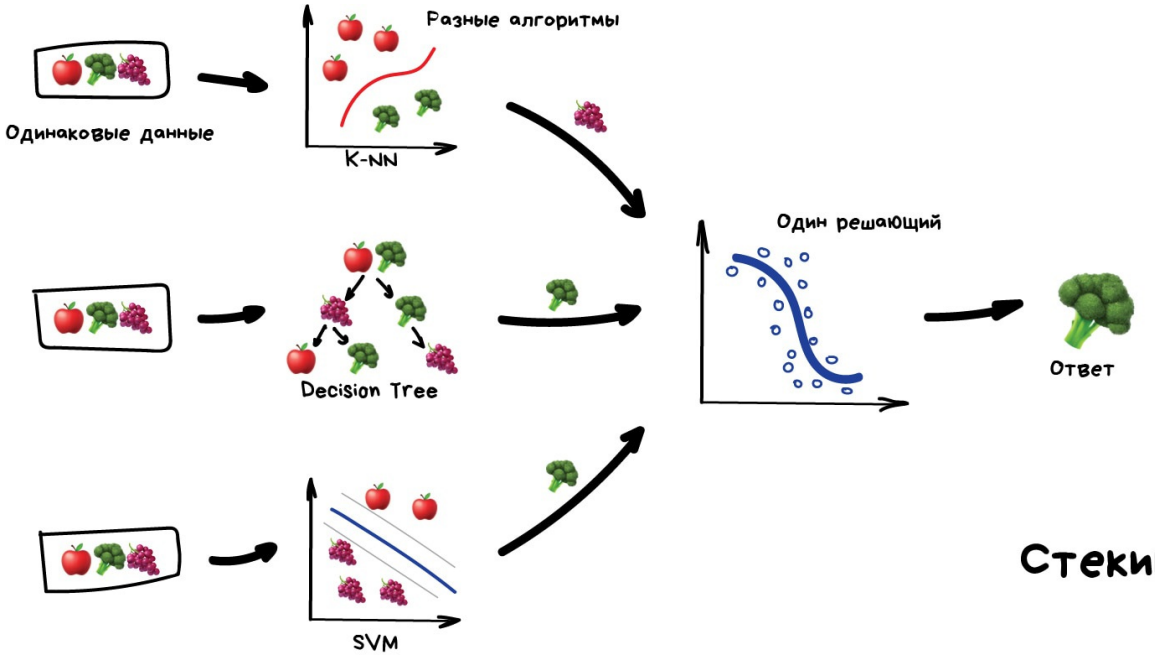
Разнообразие повышают за счёт «варьирования» обучающей выборки (бэгинг), «варьирования» моделей (стэкинг) и т.п.

Обоснования использования ансамблей в машинном обучении обычно бывают **статистическими (statistical)** – приведены выше, когда мы оценивали ошибку ансамбля исходя из вероятностной природы ответов алгоритмов, **вычислительными (computational)** – поскольку обучение это решение задачи оптимизации, то ансамбль распараллеливает процесс: мы параллельно обучаем несколько базовых алгоритмов, а метаалгоритм «комбинирует» полученные ответы, **функциональными (representational)** – часто ансамблем можно представить существенно более сложную функцию, чем любым базовым алгоритмом. *Последнее можно проиллюстрировать следующим рисунком: показана задача бинарной классификации, в которой нельзя настроить на 100%-е качество линейный алгоритм, но можно настроить простейший ансамбль линейных алгоритмов. Таким образом, ансамбли позволяют решать сложные задачи простыми моделями, усложнение решения для возможности хорошей настройки на данные происходит на уровне мета-алгоритма.*



Существуют различные **модели ансамблирования**. Рассмотрим три из них:
стекинг, бэггинг, бустинг

Стекинг (Stacking). Обучаем несколько **разных алгоритмов** и передаём их результаты на вход последнему, который принимает итоговое решение. Ключевое слово — разных алгоритмов, ведь один и тот же алгоритм, обученный на одних и тех же данных не имеет смысла. В качестве решающего алгоритма чаще берут регрессию.



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

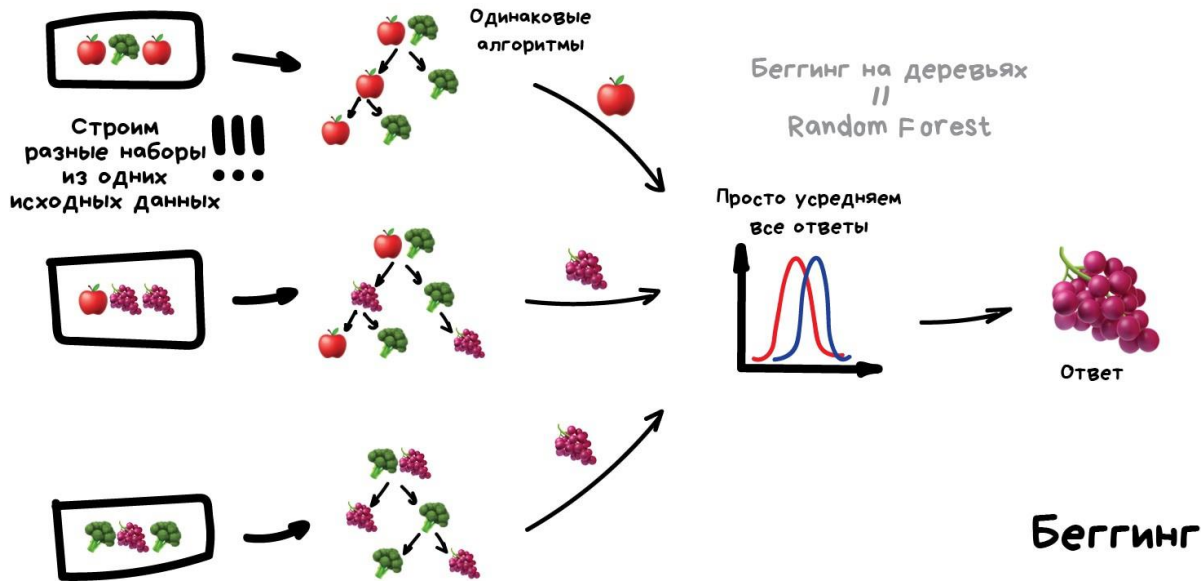
Бэггинг (Bagging). Обучаем один алгоритм много раз на случайных выборках из исходных данных. В самом конце усредняем ответы.

Данные в случайных выборках могут повторяться. То есть из набора 1-2-3 мы можем делать выборки 2-2-3, 1-2-2, 3-1-2 и так пока не надоест. На них мы обучаем один и тот же алгоритм несколько раз, а в конце вычисляем ответ простым голосованием.

Самый популярный пример беггинга — алгоритм **Random Forest**, беггинг на деревьях.

Преимущество: скорость. Например, в реальном времени нейросеть будет слишком медлительна, а беггинг идеален, ведь он может считать свои деревья параллельно.

Способность параллелиться даёт беггингу преимущество даже над бустингом, который работает точнее, но только в один поток.



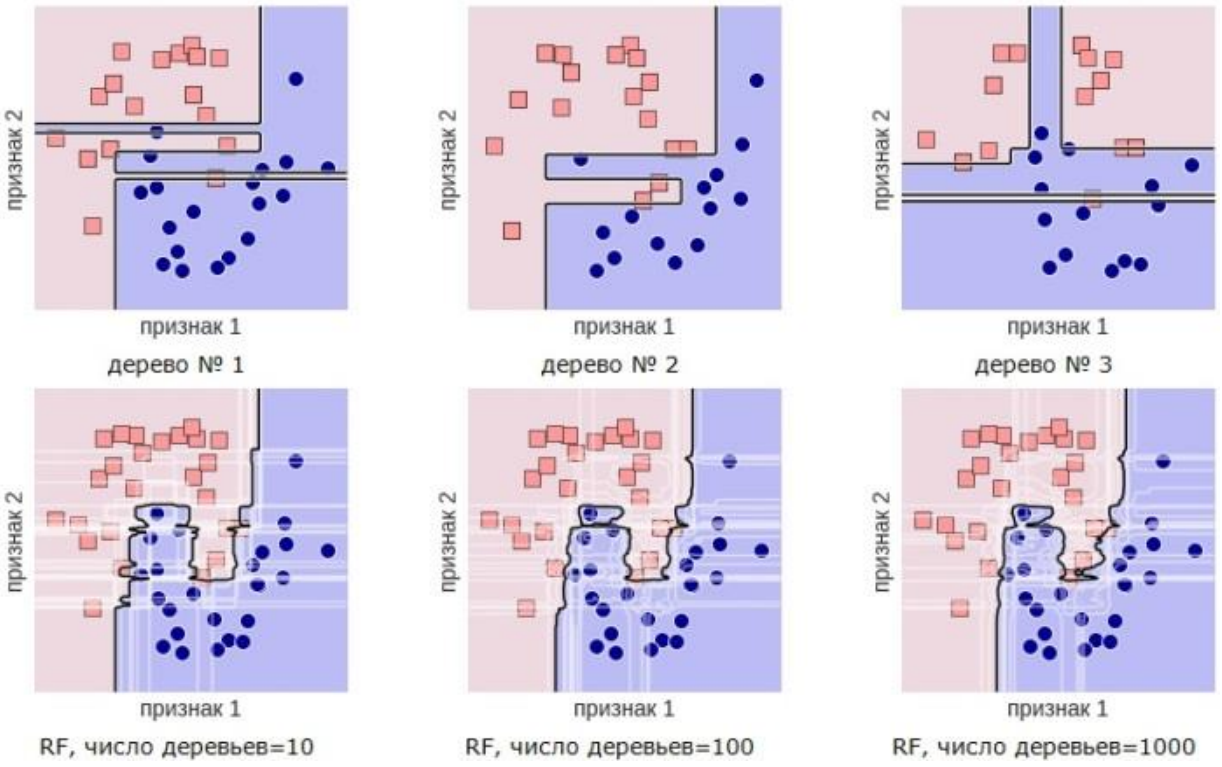
Случайный лес (Random Forest) – обобщение бэгинга

Построение случайного леса

1. Выбирается бутстреп-подвыборка* – на ней строится дерево
2. Строим дерево
   - 2.1. Для построения каждого расщепления просматриваем **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)

* Суть метода **бутстрэп** (англ. bootstrap) состоит в том, чтобы по имеющейся выборке построить эмпирическое распределение. Используя это распределение как теоретическое распределение вероятностей, можно с помощью датчика псевдослучайных чисел сгенерировать практически неограниченное количество псевдовыборок произвольного размера, например, того же, как у исходной. На множестве псевдовыборок можно оценить не только анализируемые статистические характеристики, но и изучить их вероятностные распределения. Таким образом, например, оказывается возможным оценить дисперсию или квантили любой статистики независимо от её сложности.



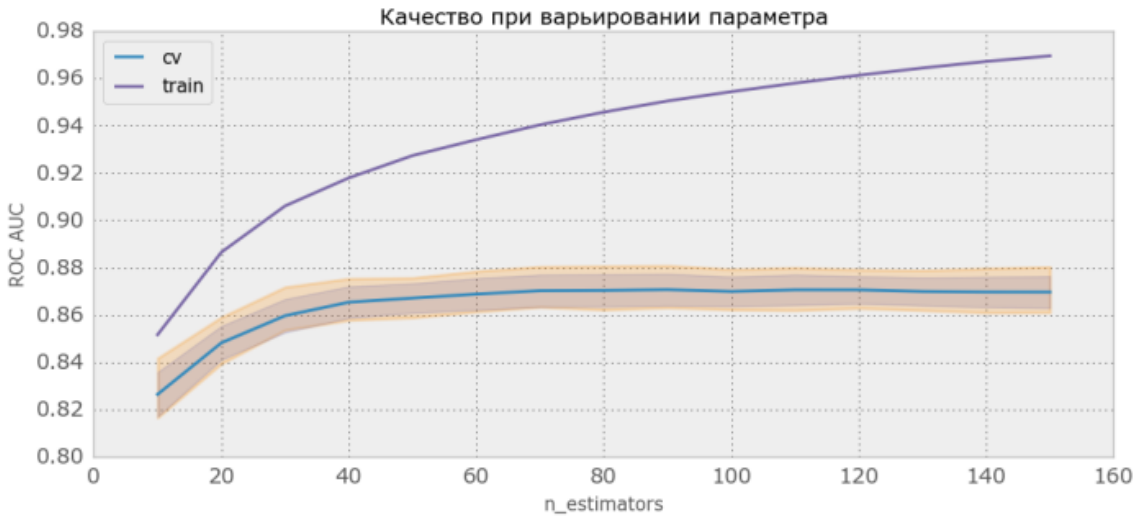
https://dyakonov.org/2019/04/19/%d0%b0%d0%bd%d1%81%d0%b0%d0%bc%d0%b1%d0%bb%d0%b8-%d0%b2-%d0%bc%d0%b0%d1%88%d0%b8%d0%bd%d0%bd%d0%be%d0%bc-%d0%be%d0%b1%d1%83%d1%87%d0%b5%d0%bd%d0%b8%d0%b8/

В библиотеке scikit-learn есть такая реализация Random Forest:

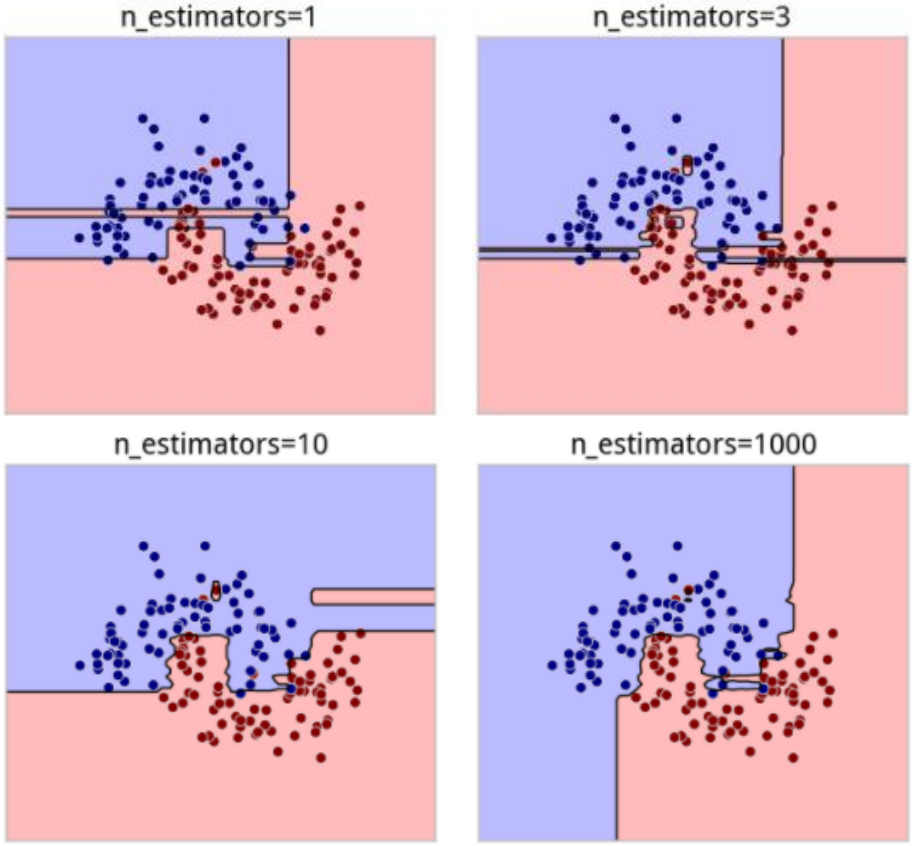
class **sklearn.ensemble.RandomForestClassifier**(**n_estimators**=10, criterion='gini', **max_depth**=None, **min_samples_split**=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, bootstrap=True, oob_score=False, **n_jobs=1**, random_state=None, verbose=0, warm_start=False, class_weight=None)

Параметр 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 достаточно).

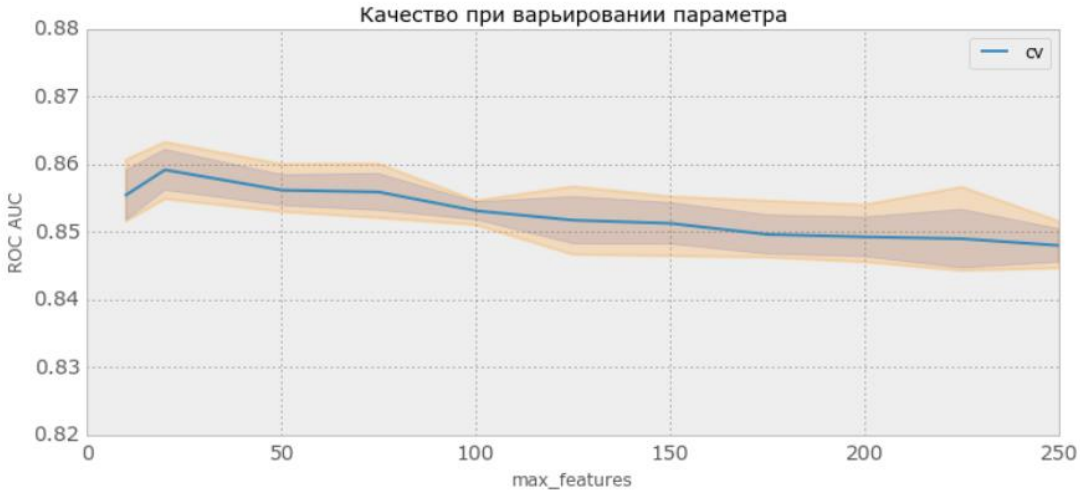


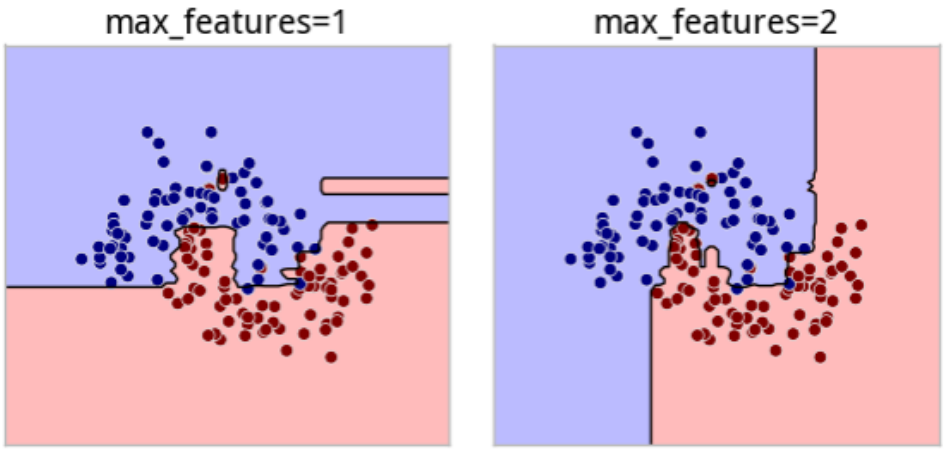
По рисунку видно, что в данной модели нецелесообразно использовать более 60 деревьев.



Число признаков для выбора расщепления — max_features

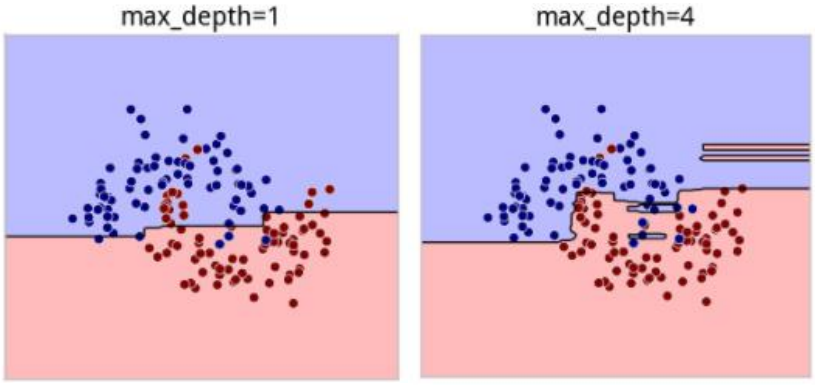
При увеличении max_features увеличивается время построения леса, а деревья становятся «более однообразными». По умолчанию он равен sqrt(n) в задачах классификации и n/3 в задачах регрессии. Это самый важный параметр! Его настраивают в первую очередь (при достаточном числе деревьев в лесе).





Максимальная глубина деревьев — max_depth

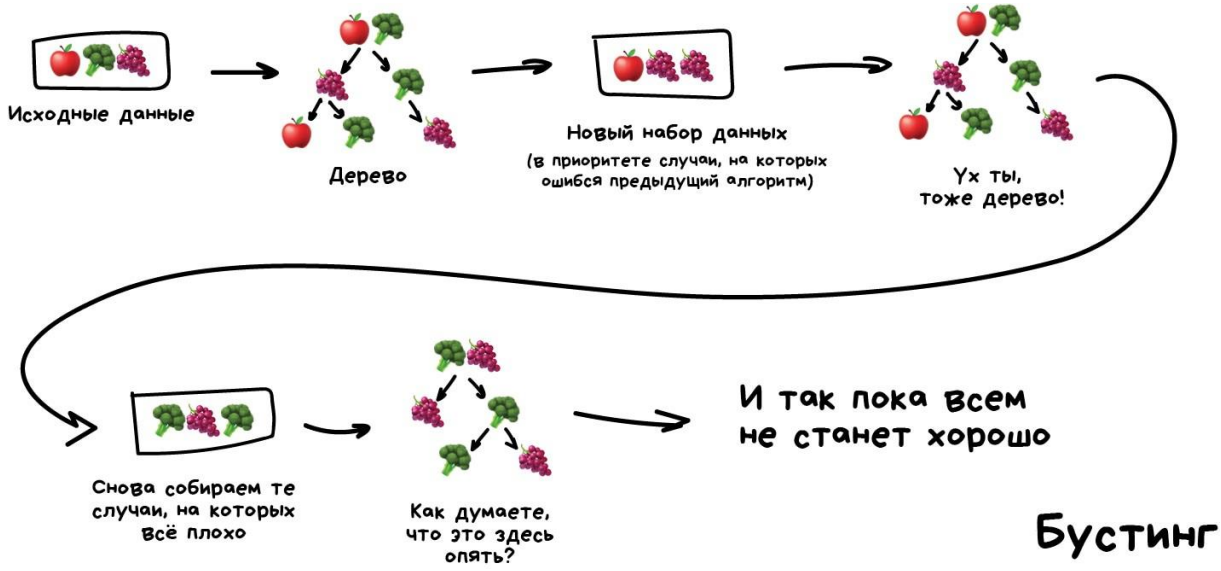
Ясно, что чем меньше глубина, тем быстрее строится и работает RF. При увеличении глубины резко возрастает качество на обучении, но и на контроле оно, как правило, увеличивается. Рекомендуется использовать максимальную глубину.



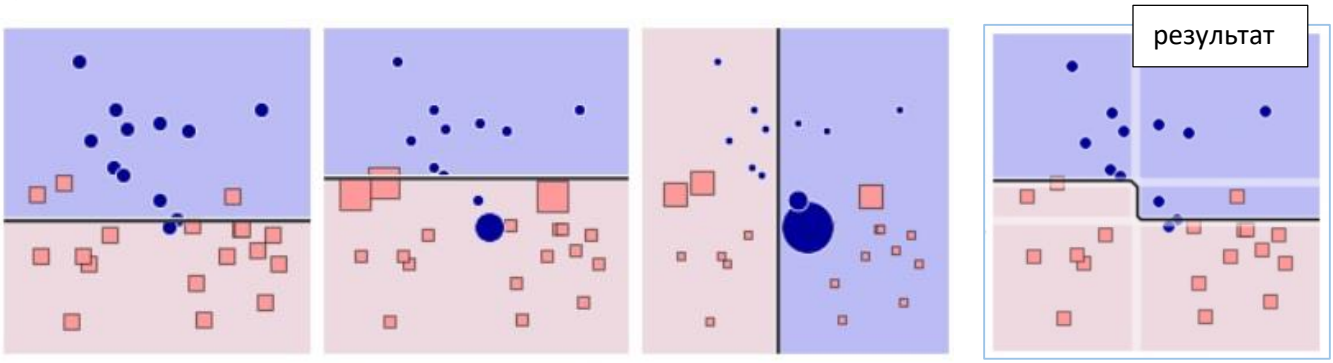
По умолчанию **n_jobs=1**, т.е. случайный лес строится на одном процессоре. Если Вы хотите существенно ускорить построение, используйте **n_jobs=-1** (строить на максимально возможном числе процессоров).

Бустинг (Boosting). Обучаем алгоритмы последовательно, каждый следующий уделяет особое внимание тем случаям, на которых ошибся предыдущий.

Как в беггинге, мы делаем выборки из исходных данных, но теперь не совсем случайно. В каждую новую выборку мы берём часть тех данных, на которых предыдущий алгоритм отработал неправильно. То есть как бы доучиваем новый алгоритм на ошибках предыдущего и повышал качество всего ансамбля.

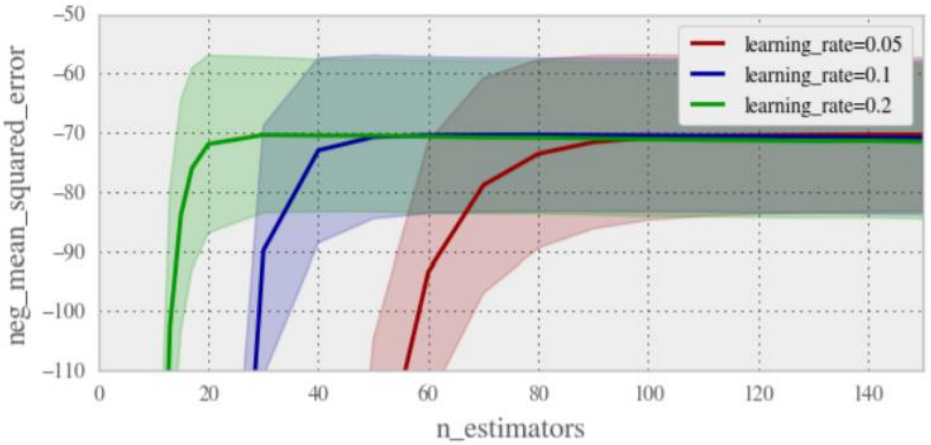


Первый успешный вариант бустинга – **AdaBoost** (Adaptive Boosting), сейчас практически не используется, поскольку его вытеснил градиентный бустинг. На рис. ниже показано (3 первые картики), как меняются веса объектов при построении очередного базового алгоритма в AdaBoost: объекты, которые классифицировались неправильно первыми двумя алгоритмами при построении третьего имеют большой вес.



В отличие от случайных деревьев, в бустинге увеличение числа деревьев не всегда приводит к улучшению качества решения на тесте. Зависимость, как правило унимодальная, см. рис. Число деревьев, при котором качество максимально, зависит от темпа обучения: чем меньше темп, тем больше деревьев нужно.

Рис. Зависимость качества решения на тесте от числа деревьев в бустинге



Как правило, бустинг показывает высокое качество над неглубокими деревьями (глубина от 3 до 6).

Плюсы — высокая точность классификации.

Минусы — не параллелится, НО, пока еще работает быстрее нейросетей.

Реальный пример работы бустинга — поисковые системы Яндекса.

В scikit-learn баггинг реализуется метаоценками **BaggingClassifier** и **BaggingRegressor.** Параметры max_samples и max_features позволяют контролировать размер подмножеств (с точки зрения образцов и функций).

class **sklearn.ensemble.BaggingClassifier**(*base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0*)

В качестве примера приведенный ниже фрагмент иллюстрирует, как создать экземпляр ансамбля KNeighborsClassifier базовых оценок, каждая из которых построена на случайных подмножествах из 50% выборок и 50% функций.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
bagging = BaggingClassifier(KNeighborsClassifier(),
                            max_samples=0.5, max_features=0.5)
```

Кривые обучения

Кривые обучения - это широко используемый диагностический инструмент в машинном обучении для алгоритмов, которые постепенно обучаются на основе обучающего набора данных. Модель может быть оценена на наборе обучающих данных и на тестовом наборе данных после каждого обновления во время обучения, измеренная производительность отображается на графиках для отображения кривых обучения.

Анализ кривых обучения моделей может использоваться для диагностики проблем с обучением, таких как недостаточная или избыточная модель, а также для определения того, являются ли наборы данных для обучения и проверки надлежащим образом репрезентативными.

Кривая обучения - это график эффективности обучения модели в зависимости от опыта или времени. По оси **x** (как правило) откладывают время или опыт, а по оси **y** обучение или улучшения.

Эффективность обучения может оцениваться точностью классификации, тогда большим значениям точности соответствует лучшее обучение или величиной ошибки, тогда лучшее обучение соответствует меньшим значениям ошибки.

Оценка текущего состояния модели **на обучающем наборе** данных, показывает насколько хорошо модель «**обучается**». Оценка **на тестовом наборе** данных проверки дает представление о том, насколько хорошо модель «**обобщает**».

Диагностика поведения модели

Форма и динамика кривой обучения могут использоваться для диагностики поведения модели машинного обучения и, в свою очередь, предлагать тип

изменений конфигурации, которые могут быть внесены для улучшения обучения и / или производительности.

Есть три общих динамики, которые можно заметить в кривых обучения:

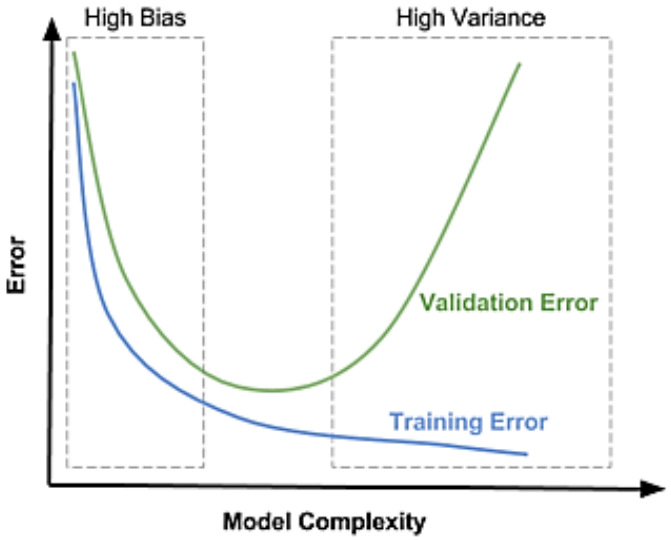
- недообучение
- переобучение
- хорошо подходит.

Обычно результаты обучения и тестирования/валидации строятся вместе, чтобы мы могли диагностировать компромисс между смещением и дисперсией (т. е. определить, выиграем ли мы от добавления большего количества обучающих данных, и оценить сложность модели, контролируя регуляризацию или количество функций).

Смещение (bias) — это погрешность оценки, возникающая в результате ошибочного предположения в алгоритме обучения. В результате большого смещения алгоритм может пропустить связь между признаками и выводом (недообучение).

Дисперсия (variance) — это ошибка чувствительности к малым отклонениям в тренировочном наборе. При высокой дисперсии алгоритм может как-то трактовать случайный шум в тренировочном наборе, а не желаемый результат (переобучение).

Компромисс отклонение-дисперсия — конфликт при попытке одновременно минимизировать эти два источника ошибки, которые мешают алгоритмам обучения с учителем делать обобщение за пределами тренировочного набора. При уменьшении одного из негативных эффектов, приводит к увеличению другого. Данная дилемма проиллюстрирована на Рис. При небольшой сложности модели мы наблюдаем большое смещение (high bias). При усложнении модели смещение уменьшается, но дисперсия увеличится, что приводит к проблеме отклонение-дисперсия.



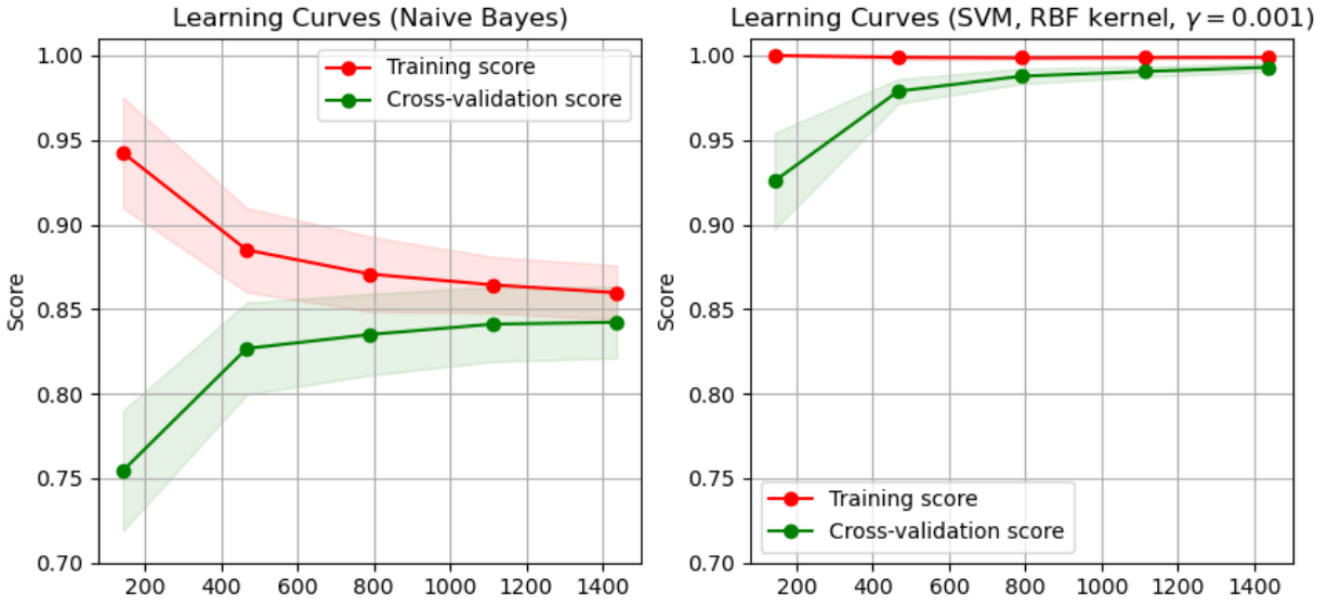
Возможные решения при переобучении

- ✓ Увеличение количества данных в наборе;
- ✓ Уменьшение количества параметров модели;
- ✓ Добавление регуляризации / увеличение коэффициента регуляризации.

Возможные решения при недообучении

- ✓ Добавление новых параметров модели;
- ✓ Использование для описания модели функций с более высокой степенью;
- ✓ Уменьшение коэффициента регуляризации.

Вопрос: какая модель лучше?



https://scikit--learn-org.translate.goog/stable/auto_examples/model_selection/plot_learning_curve.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=nui,sc

Калибровка классификаторов

https://ichi.pro/ru/kalibrovka-klassifikatora-137489987687506

Для оценки полученной модели классификатора мы используем вероятность, с которой модель определяет класс для каждого объекта. Но всегда ли это отражает реальность?

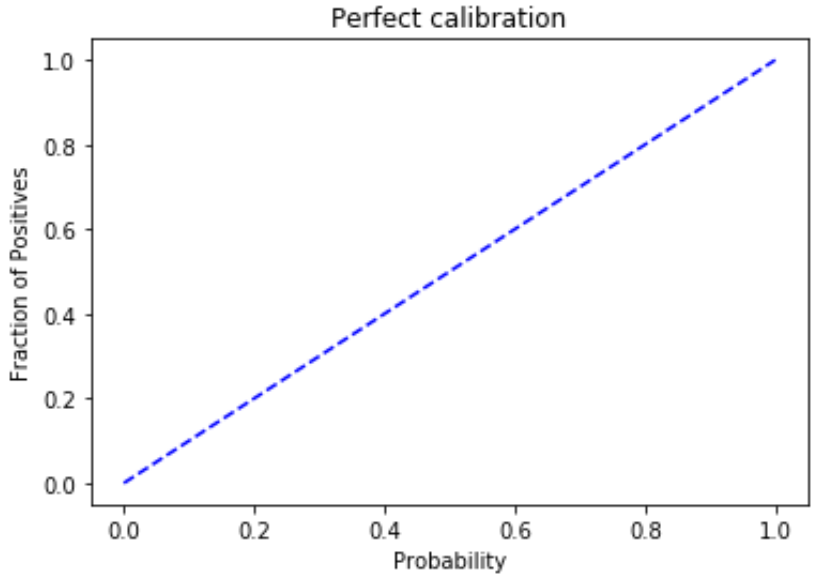
Представьте, что у нас есть два бинарных классификатора; модель A и модель B. Модель A имеет точность 85% и уверенность 0,86 для каждого сделанного прогноза. С другой стороны, модель B также на 85% точна, но достоверна на 0,99 для каждого из своих прогнозов. Как вы думаете, какая модель лучше?

Модель A лучше, потому что считает себя точной в 86% случаев, и это действительно так. Напротив, модель B слишком уверена в своих прогнозах. Этот пример демонстрирует интуицию вероятности и калибровки модели.

Калибровка модели относится к процессу, в котором мы берем модель, которая уже обучена, и применяем операцию постобработки, которая улучшает ее оценку вероятности. Таким образом, если бы мы проверили образцы, которые были оценены как положительные с вероятностью 0,85, мы бы ожидали, что 85% из них действительно положительные.

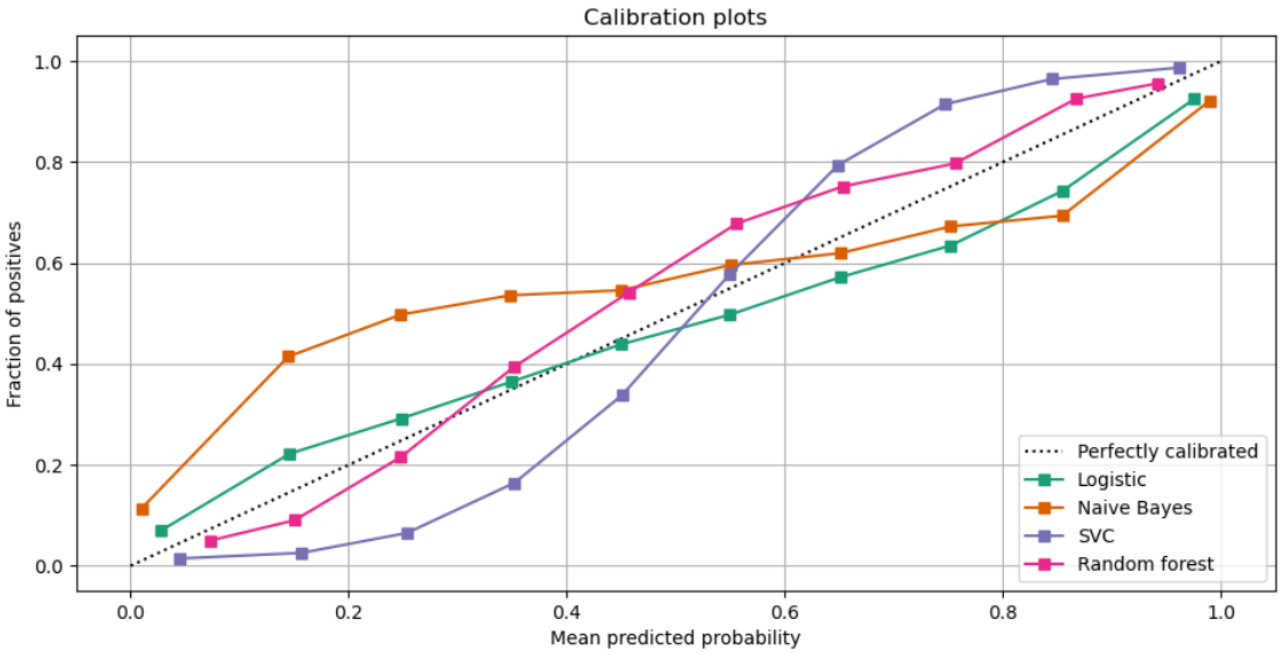
Формально модель идеально откалибрована, если для любого значения вероятности ***p*** уверенное предсказание класса в процентах случаев ***p*** является правильным ***100*p***.

Если построить каждое значение ***p*** в интервале от 0 до 1, мы ожидаем получить идеальную линейную связь между вычисленной вероятностью и долей положительных результатов.



Калибровочные кривые (также известные как диаграммы надежности) позволяют сравнить, насколько хорошо откалиброваны вероятностные прогнозы двоичного классификатора. Он отображает истинную частоту положительной метки против ее прогнозируемой вероятности для прогнозов с разбиением на

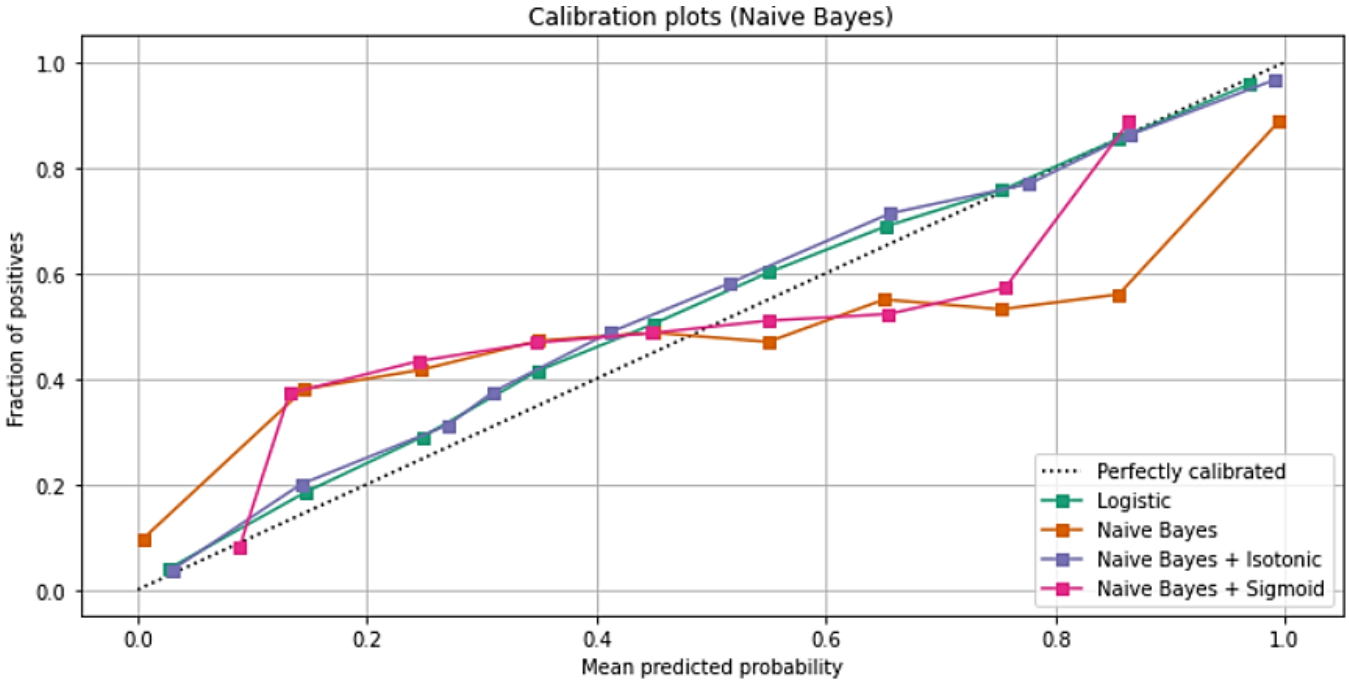
интервалы. Ось x представляет собой среднюю прогнозируемую вероятность в каждом интервале. По оси ординат отложена доля положительных результатов, т. е. доля образцов, класс которых является положительным (в каждой ячейке). График верхней калибровочной кривой создается с помощью CalibrationDisplay.from_estimators, который используется calibration_curve для расчета средней прогнозируемой вероятности для каждого бина и доли положительных результатов. CalibrationDisplay.from_estimator принимает в качестве входных данных подобранный классификатор, который используется для вычисления прогнозируемых вероятностей. Таким образом, классификатор должен иметь pred_proba метод. Для нескольких классификаторов, у которых нет метода pred_proba, его можно использовать CalibratedClassifierCV для калибровки выходных данных классификатора по вероятностям.



CalibratedClassifierCV - Класс используется для калибровки классификатора.

CalibratedClassifierCV поддерживает использование двух «калибровочных» регрессоров: «**сигмовидной**» и «**изотонической**».

```
gnb = GaussianNB()
gnb_isotonic = CalibratedClassifierCV(gnb, cv=2, method="isotonic")
gnb_sigmoid = CalibratedClassifierCV(gnb, cv=2, method="sigmoid")
```



Так ли важна калибровка модели?

Калибровка модели важна только в том случае, если вы заботитесь о вероятностях, которые вычисляет ваша модель. Например, предположим, что вы создаете механизм рекомендаций, который ранжирует продукты в соответствии с предпочтениями пользователя. Если ваша модель оценивает, что пользователь **u** купит продукт **a** с вероятностью 0,9, а элемент – **b** с вероятностью 0,7, вы можете сначала подать продукт **a**. Калибровать эту модель не нужно.

Однако, если вы создаете критически важное приложение, которое вычисляет вероятность того, что человек заболел, фактическое значение вероятности имеет большое значение.

Генерация случайных наборов данных

> *scikit-learn включает в себя различные генераторы случайных выборок для задач регрессии, классификации и кластеризации.*

Генерация случайной задачи классификации n-классов:

sklearn.datasets.make_classification

> ***sklearn.datasets. make_classification*** *(n_samples = 100 , n_features = 20 , * , n_informative = 2 , n_redundant = 2 , n_repeated = 0 , n_classes = 2 , n_clusters_per_class = 2 , weights = None , flip_y = 0.01 , class_sep = 1.0 , hypercube = True ,shift = 0.0 , scale = 1.0 , shuffle = True , random_state = None)*

Первоначально это создает кластеры точек, нормально распределенных (std = 1) вокруг вершин **n_informative**-мерного гиперкуба со сторонами длины, 2***class_sepи*** присваивает равное количество кластеров каждому классу. Он вводит взаимозависимость между этими функциями и добавляет к данным различные типы дополнительного шума.

Без перетасовки X горизонтально складывает объекты в следующем порядке: основные n_informative признаки, затем n_redundant линейные комбинации информативных признаков, за которыми следуют n_repeated дубликаты, нарисованные случайным образом с заменой информативных и повторяющихся признаков. Остальные особенности заполнены случайным шумом. Таким образом, без перетасовки все полезные функции содержатся в столбцах .X[:, :n_informative + n_redundant + n_repeated]

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html#sklearn.datasets.make_classification

Пример: Набор данных синтетической бинарной классификации с 100 000 выборок и 20 признаками. Из 20 признаков только 2 являются информативными, 10 - избыточными (случайные комбинации информативных признаков), а остальные 8 - неинформативными (случайные числа). Из 100 000 образцов 1 000 будут использованы для подгонки моделей, а остальные - для испытаний.

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

X, y = make_classification(
    n_samples=100_000, n_features=20, n_informative=2, n_redundant=10,
random_state=42
)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.99, random_state=42
)
```