**Практическая работа №11**

**Задание:**

Изучить приведенный ниже материал и подготовить реферат с описанием примеров использования методов: Бэггинг и случайный лес.

**Источники:**[Классификация, регрессия и другие алгоритмы Data Mining с использованием R (ranalytics.github.io)](https://ranalytics.github.io/data-mining/044-Ensembles.html)  
[Ансамбли в машинном обучении | Анализ малых данных (dyakonov.org)](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/)  
[Урок по Random Forest на Python: разбор алгоритма с примером на SkLearn (pythonru.com)](https://pythonru.com/uroki/sklearn-random-forest?)  
[Интегрированное обучение и метод случайного леса - Русские Блоги (russianblogs.com)](https://russianblogs.com/article/92681643486/)  
https://web.stanford.edu/~hastie/CASI\_files/PDF/casi.pdf

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ:

# Случайный лес (Random Forest)

Случайный лес – метод [Машинного обучения (ML)](https://www.helenkapatsa.ru/mashinnoie-obuchieniie/), использующий Ансамбль (Ensemble) [Деревьев решений (Decision Tree)](https://www.helenkapatsa.ru/dierievo-rieshienii/) для задач [Классификации (Classification)](https://www.helenkapatsa.ru/klassifikatsiia/). Каждое отдельное дерево в таком лесу дает предсказание класса, и набравший наибольшее количество голосов Класс (Class), становится предсказанием [Модели (Model)](https://www.helenkapatsa.ru/modiel/). Он использует [Бэггинг (Bagging)](https://www.helenkapatsa.ru/beghghingh/) и случайность признаков при построении каждого отдельного дерева, чтобы создать некоррелированный лес из деревьев, прогноз которого "комитетом" более точен, чем прогноз любого отдельного дерева.

Бóльшая часть Машинного обучения – это классификация: мы хотим знать, к какому типу (или группе) принадлежит [Наблюдение (Observation)](https://www.helenkapatsa.ru/nabliudieniie/). Возможность точно классифицировать наблюдения чрезвычайно важна для различных бизнес-приложений, таких как прогнозирование покупки, или вероятность неуплаты кредита.

[Наука о данных (Data Science)](https://www.helenkapatsa.ru/nauka-o-dannykh/) предоставляет множество [Алгоритмов (Algorithm)](https://www.helenkapatsa.ru/alghoritm/) классификации, таких как [Логистическая регрессия (Logistic Regression)](https://www.helenkapatsa.ru/loghistichieskaia-rieghriessiia/), [Метод опорных векторов (SVM)](https://www.helenkapatsa.ru/mietod-opornykh-viektorov/), Наивный байесовский классификатор (Naive Bayes) и деревья решений. Но в верхней части иерархии классификаторов находится классификатор случайного леса.

Мы рассмотрим, как работают базовые деревья решений, как отдельные деревья решений объединяются для создания случайного леса, и в конечном итоге выясним, почему случайные леса так хороши в том, что делают.

### Деревья решений

Давайте быстро освежим тему деревьев решений, поскольку они являются строительными блоками модели случайного леса. К счастью, они довольно интуитивны. Я готов поспорить, что большинство людей использовали дерево решений, сознательно или нет, в какой-то момент своей жизни.



Наверное, гораздо легче понять, как работает дерево решений на примере.  
Представьте, что наш набор данных состоит из списка чисел в верхней части рисунка [1, 1, 0, 0, 0, 0, 0]. У нас есть две единицы и пять нулей (1 и 0 – наши классы), и мы хотим разделить их, используя характеристики цвета и "подчеркнутости". Итак, как мы можем это сделать?

Цвет кажется довольно очевидной особенностью для разделения, поскольку все нули, кроме одного, белые. Таким образом, мы можем использовать вопрос: «Это розовый?» чтобы разделить числа в первом узле. Вы можете представить себе узел как разветвление дерева, где ветки разделяются на две части "да" и "нет".

Ветвь «нет» (белая) теперь имеет нулевые значения, и на ней дальнейшее деление невозможно. Но нашу ветвь «да» все еще можно разделить. Теперь мы можем использовать вторую характеристику и спросить: «Ряд подчеркнутый?» чтобы сделать второй раскол – "сплит".

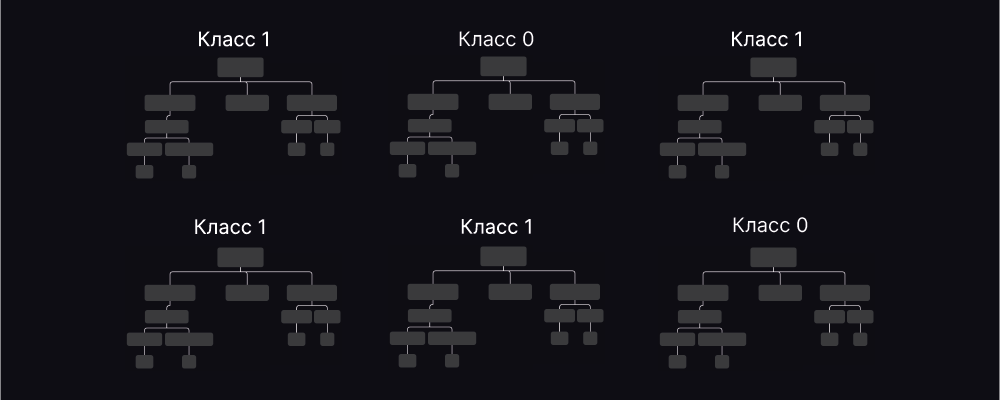
Две подчеркнутые единицы отправляются вниз по ответвлению "да", а ноль, который не подчеркнут, идет по правой ветви. Наше дерево решений использовало эти две функции-характеристики для идеального разделения данных. Победа!

Очевидно, что в реальной жизни наши данные не будут такими чистыми, но логика, которую использует дерево решений, остается прежней. На каждом узле он спросит:

Какая функция позволит мне разделить имеющиеся наблюдения таким образом, чтобы результирующие группы максимально отличались друг от друга (и члены каждой результирующей подгруппы были максимально похожи друг на друга)?

### Классификатор случайного леса

Случайный лес, как следует из названия, состоит из большого количества отдельных деревьев решений.

5 деревьев решили, что класс 1, одно – 0. "Демократический" результат – 1.

Фундаментальная концепция случайного леса проста, но действенна – мудрость толпы. Говоря языком науки о данных, причина того, что модель случайного леса так хорошо работает, заключается в следующем:  
Большое количество относительно некоррелированных моделей (деревьев), работающих как комитет, превзойдут любую из отдельных составляющих моделей.

Ключевым моментом является низкая корреляция между моделями. Подобно тому, как инвестиции с низкой корреляцией (например, акции и облигации) объединяются, чтобы сформировать портфель, превышающий сумму его частей, некоррелированные модели могут давать ансамблевые прогнозы, которые более точны, чем любые индивидуальные. Причина этого замечательного эффекта в том, что деревья "защищают"друг друга от своих индивидуальных ошибок (до тех пор, пока все они не ошибаются постоянно в одном и том же направлении). В то время как некоторые деревья могут быть неправильными, многие другие будут верны, поэтому деревья могут двигаются в правильном направлении как группа. Итак, предпосылки для хорошей работы случайного леса:

* В наших функциях должен быть какой-то реальный сигнал, чтобы модели, построенные с использованием этих [Признаков (Feature)](https://www.helenkapatsa.ru/priznak/), работали лучше, чем случайное предположение.
* Прогнозы (и, следовательно, ошибки), сделанные отдельными деревьями, должны иметь низкую корреляцию друг с другом.

### Пример того, почему некоррелированные результаты так хороши

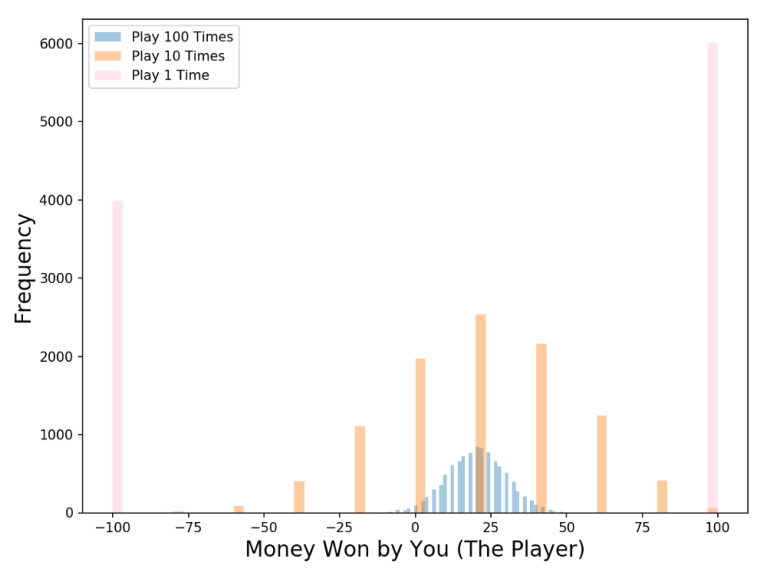
Замечательные эффекты некоррелированных моделей – это настолько важная концепция, что я хочу показать вам пример, который поможет ей по-настоящему проникнуться. Представьте, что мы играем в следующую игру:  
я использую генератор случайных чисел от 0 до 100 для получения числа. Если полученное мной число больше или равно 40, вы выигрываете (так что у вас есть шанс на победу 60%), и я плачу вам немного денег. Если оно ниже 40, я выигрываю, и вы платите мне ту же сумму.

Теперь я предлагаю вам следующие варианты. Мы можем сыграть:

* Игру 1: 100 генераций числа, ставка – $1
* Игра 2: 10 – 10
* Игра 3: 1 – 100

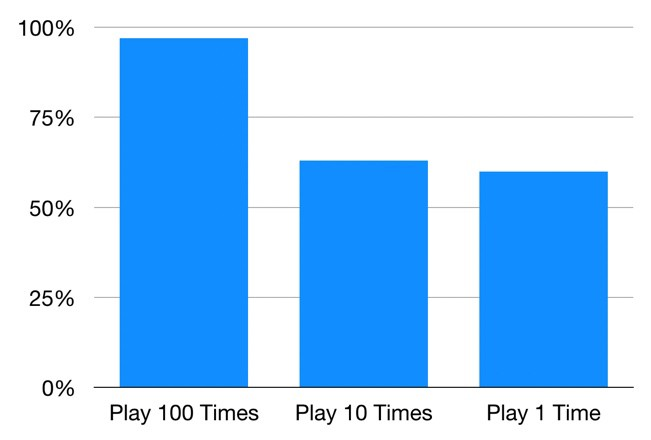
Какой вариант Вы предпочитаете? Ожидаемая ценность каждой игры одинакова:

* 1: (0,60 \* 1 + 0,40 \* -1) \* 100 = 20
* 2: (0,60 \* 10 + 0,40 \* -10) \* 10 = 20
* 3: 0,60 \* 100 + 0,40 \* -100 = 20

Распределение результатов для 10 000 симуляций для каждой игры

А что насчет распределений? Давайте визуализируем результаты с помощью моделирования Монте-Карло (мы запустим 10 000 имитаций каждого типа игры; например, мы будем моделировать 10 тысяч раз 100 партий по схеме № 1. Взгляните на таблицу: какую игру Вы бы выбрали? Несмотря на то, что ожидаемые значения одинаковы, распределения результатов сильно различаются: от положительных и узких (синего цвета) до двоичных (розового).

Игра 1 (в которой мы играем 100 раз) дает лучший шанс заработать немного денег – из 10 000 симуляций, которые я провел, вы зарабатываете деньги в 97% из них! Для игры 2 (в которой мы играем 10 раз) вы зарабатываете деньги в 63% симуляций, резкое снижение вознаграждения за риск (и резкое увеличение вероятности потерять деньги). И в игре 3, в которую мы играем только один раз, вы, как и ожидалось, зарабатываете деньги на 60% симуляций.



### Вероятность заработка на каждой игре

Таким образом, даже несмотря на то, что игры имеют одинаковую ожидаемую ценность, их распределение результатов совершенно разное. Чем больше мы разделим нашу ставку в 100 долларов на разные игры, тем больше у нас будет уверенности в том, что мы заработаем деньги. Как упоминалось ранее, это работает, потому что каждая игра не зависит от других.

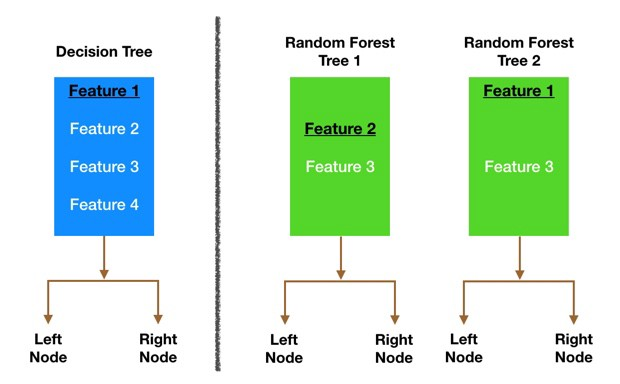
Случайный лес такой же: каждое дерево похоже на одну игру в нашей предыдущей игре. Мы просто видели, как наши шансы заработать деньги увеличивались, чем больше раз мы играли. Точно так же с моделью случайного леса наши шансы сделать правильные прогнозы увеличиваются с увеличением количества некоррелированных деревьев.

### Обеспечение диверсификации моделей друг другом

Итак, случайный лес гарантирует, что поведение каждого отдельного дерева не слишком коррелирует с поведением любого из других деревьев в модели. Он использует следующие два метода:

* **Бэггинг (Bagging)**. Деревья решений очень чувствительны к данным, на которых обучаются: небольшие изменения в обучающем наборе могут привести к существенно разным древовидным структурам. Случайный лес использует это преимущество, позволяя каждому отдельному дереву случайным образом выбирать из набора данных, в результате чего получаются разные деревья. Этот процесс известен как бэггинг.

Обратите внимание, что при бэггинге мы не разделяем обучающие данные на более мелкие фрагменты и не обучаем каждое дерево на разных фрагментах. Скорее, если у нас есть выборка размера N, мы по-прежнему скармливаем каждому дереву обучающий набор размера N (если не указано иное). Но вместо исходных обучающих данных мы берем случайную выборку размера N с заменой. Например, если наши обучающие данные были [1, 2, 3, 4, 5, 6], мы могли бы дать одному из наших деревьев следующий список [1, 2, 2, 3, 6, 6]. Обратите внимание, что оба списка имеют длину шесть и что «2» и «6» повторяются в случайно выбранных обучающих данных, которые мы передаем нашему дереву (потому что мы производим выборку с заменой).

Разделение узлов основано на случайности

* **Случайность признаков.**В обычном дереве решений, когда приходит время разделения, мы рассматриваем все возможные признаки и выбираем тот, который дает наилучший сплит между наблюдениями в левом узле и наблюдениями в правом узле. Напротив, каждое дерево в случайном лесу может выбирать только из случайного подмножества функций-признаков. Это вызывает еще большее разнообразие деревьев в модели и в конечном итоге приводит к более низкой корреляции между деревьями и большей диверсификации.

Давайте рассмотрим наглядный пример: на картинке выше традиционное дерево решений (выделено синим цветом) может выбирать из всех четырех функций при принятии решения о том, как разделить узел. Он решает использовать функцию 1 (черная подчеркнутая), поскольку она разделяет данные на группы наиболее четко.

Теперь давайте посмотрим на наш случайный лес. В этом примере мы рассмотрим только два дерева в нашем лесу. Когда мы проверяем случайное дерево леса 1, мы обнаруживаем, что оно может учитывать только функции 2 и 3 (выбранные случайным образом) для принятия решения о разделении узлов. Из нашего традиционного дерева решений (выделено синим цветом) мы знаем, что признак №1 – лучший способ разделения, но Дерево 1 не может видеть его, поэтому оно вынуждено использовать признак № 2 (черный и подчеркнутый). Дерево 2, с другой стороны, может видеть только характеристики 1 и 3, поэтому выбирает первый признак.

Таким образом, в нашем случайном лесу мы получаем деревья, которые не только обучаются на разных наборах данных (благодаря бэггингу – "пакетированию"), но также используют разные функции для принятия решений.

И это создает некоррелированные деревья, которые буферизируют и защищают друг друга от их ошибок.

Случайные леса – мой личный фаворит. Приходя из мира финансов и инвестиций, святым Граалем всегда было построение набора несогласованных моделей, каждая из которых имеет положительный ожидаемую доходность. Затем их объединяют в портфель, чтобы получить огромную рыночная доходность. Случайный лес – это эквивалент такого альфа-портфеля в науке о данных.

### Случайный лес и scikit-learn

Давайте посмотрим, как метод реализована в SkLearn. Для начала импортируем необходимые библиотеки:

import sklearn

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make\_classification

Сгенерируем тысячу случайных наблюдений, состоящие из четырех признаков. Параметр n\_redundant, например, отвечает за корреляцию между признаками: если его значение равно нулю, то ни один из признаков не будет в зависимости от другого (такие пары признаков обычно "разбивают" удалением одного).

Сразу же запустим RandomForestClassifier, причем зададим глубину дерева, равную двум. В ином случае, ветвей у каждого узла может быть очень много, вплоть до числа наблюдений.

X, y = make\_classification(n\_samples = 1000, n\_features = 4,

n\_informative = 2, n\_redundant = 0,

random\_state = 0, shuffle = False)

clf = RandomForestClassifier(max\_depth = 2, random\_state = 0)

clf.fit(X, y)

Система указывает стандартные настройки модели: мы не назначаем классам веса (class\_weight=None), создаем 100 деревьев (n\_estimators=100):

RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None, criterion='gini', max\_depth=2, max\_features = 'auto', max\_leaf\_nodes = None, max\_samples=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=None, oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

Модель готова к использованию, так что отправим ей запись, где значением каждого из четырех признаков равно нулю:

print(clf.predict([[0, 0, 0, 0]]))

100 деревьев – это сила; модель относит это наблюдение к классу 1:

[1]

Ноутбук, не требующий дополнительной настройки на момент написания статьи, можно скачать [здесь](https://colab.research.google.com/drive/1b01hfNpnnHDTdh6yrOQ4A1O6AuX6zLm_?usp=sharing).

Автор оригинальной статьи: [Tony Yiu](https://towardsdatascience.com/understanding-random-forest-58381e0602d2)

Фото: [@lucabravo](https://unsplash.com/@lucabravo?utm_source=ghost&utm_medium=referral&utm_campaign=api-credit)

Интегрированное обучение и метод случайного леса

**Интегрированное обучение и метод случайного леса**

**Вступление**

Этот эксперимент знакомит с концепцией и основными методами ансамблевого обучения, включая Bootstraping, Bagging и random forest, а затем вычисляет важность каждой функции в случайном лесу и определяет функции, которые имеют больший вклад в модель.

**Очки знаний**

* интегрированный
* Bootstraping
* Bagging
* Случайный лес
* Важность функции

**интегрированный**

В предыдущих экспериментах были представлены различные алгоритмы классификации, а также методы проверки и оценки моделей. Теперь, предполагая, что для конкретной задачи была выбрана лучшая модель, если вы хотите еще больше повысить ее точность, вам необходимо применить более продвинутые методы машинного обучения: ансамбль. Интеграция - это метод машинного обучения, который использует ряд учащихся для обучения и использует определенные правила для интеграции различных результатов обучения для получения лучших эффектов обучения, чем у одного учащегося. При интеграции конечный общий результат более важен, чем производительность какой-либо отдельной детали.

В некотором смысле теорема присяжных Кондорсе ярко описывает упомянутую выше концепцию интеграции. Содержание теоремы таково: если каждый член жюри выносит независимые суждения и вероятность того, что каждый присяжный вынесет правильное решение, выше 0,5, то вероятность того, что все жюри вынесет правильное общее решение, увеличивается с увеличением количества присяжных. Прирост увеличивается и стремится к единице. С другой стороны, если вероятность того, что каждый из присяжных будет правильным, меньше 0,5, то вероятность того, что все присяжные вынесут правильное общее решение, уменьшается по мере увеличения числа присяжных и стремится к нулю.

Формула теоремы:

μ=∑i=mN(Ni)pi(1−p)N−i \mu = \sum\_{i=m}^{N}{N\choose i}p^i(1-p)^{N-i} μ=i=m∑N​(iN​)pi(1−p)N−i

среди них,

* NNN Общее количество присяжных.
* mmm Это минимум, составляющий большинство, а именноm=floor⁡(N/2)+1m=\operatorname{floor}(N / 2)+1m=floor(N/2)+1。
* (Ni) {N \choose i}(iN​) Количество комбинаций.
* p pp Вероятность того, что рецензент примет правильное решение.
* μ \muμ Это вероятность того, что все жюри примет правильное решение.

Из приведенной выше формулы видно, что еслиp>0.5 p > 0.5p>0.5, Потомμ>p \mu > pμ>p. Кроме того, еслиN→∞ N \rightarrow \infty N→∞, Потомμ→1 \mu \rightarrow 1μ→1。

Давайте посмотрим на другой пример интеграции: мудрость групп. В 1906 году Фрэнсис Гальтон посетил сельскую ярмарку в Плимуте, где увидел игру. 800 участников пытались оценить вес забитой коровы. Средний прогноз всех участников составляет 1197 фунтов, что очень близко к истинному весу коровы в 1198 фунтов.

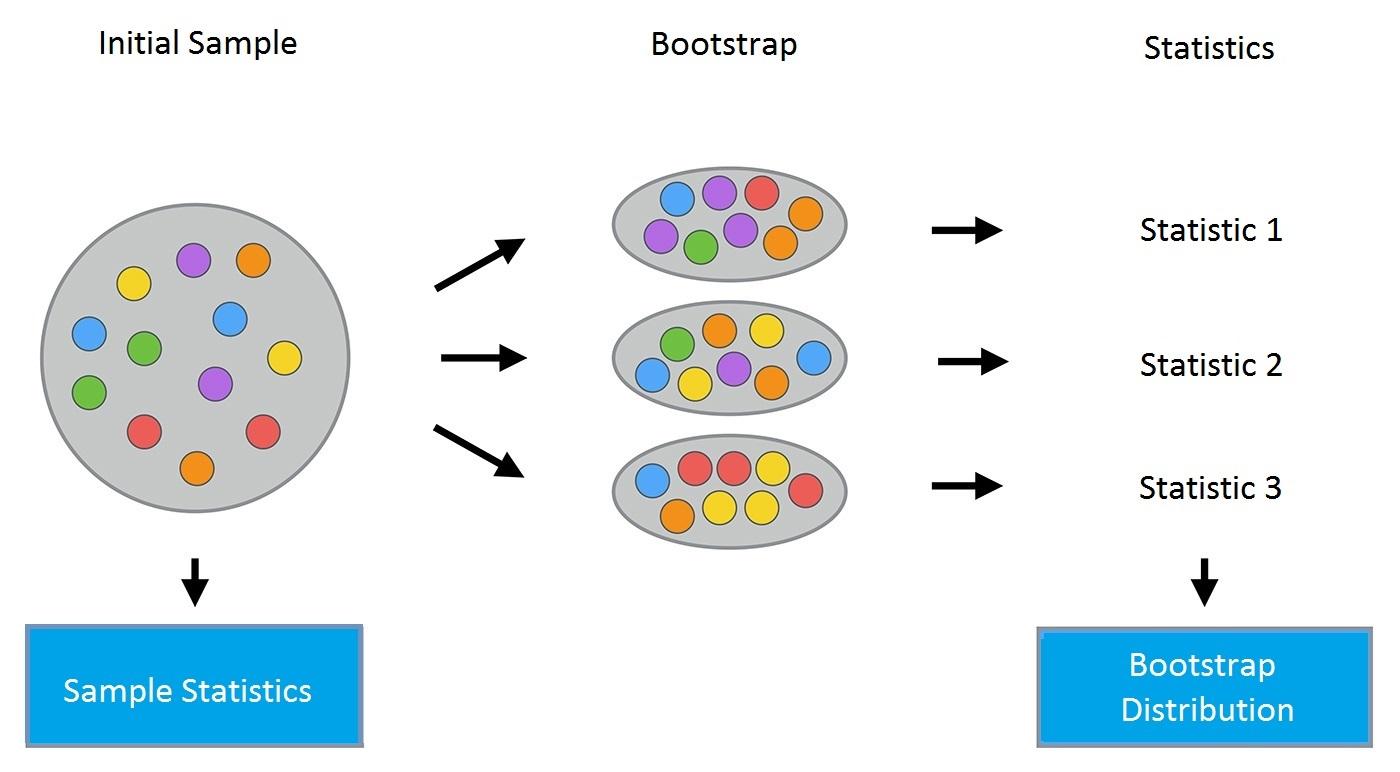
В области машинного обучения используются аналогичные идеи для уменьшения количества ошибок.

**Bootstrapping**

Bagging (также известный как Bootstrap Aggregation), предложенный Лео Брейманом в 1994 году, является одной из самых основных технологий интеграции. Пакетирование основано на загрузке статистики, что делает статистическую оценку сложных моделей более осуществимой.

Процесс метода Bootstrap выглядит следующим образом: предположим, что существует выборка X размера N, и N выборок случайным образом выбираются из выборки с заменой для создания новой выборки. Другими словами, случайным образом выберите элемент из исходной выборки размера N и повторите этот процесс N раз. Вероятность выбора всех элементов одинакова, поэтому вероятность выбора каждого элемента равна1N \frac{1}{N}N1​。

Предположим, что метод Bootstrap используется для извлечения мячей из мешка по одному. На каждом шаге кладите выбранный шар обратно в мешок, чтобы следующая розыгрыш был равновероятным, то есть вытащить из того же количества N мячей. Обратите внимание: поскольку мы возвращаем мяч обратно, в новом образце могут быть дубликаты мячей. Назовите этот новый образецX1 X\_1X1​. Повторите этот процесс M раз, чтобы создать M образцов Bootstrap.X1,…,XM X\_1, \dots, X\_MX1​,…,XM​. Наконец, размер нашей выборки был увеличен с исходной 1 до M, и теперь имеется достаточно выборок для расчета различных статистических данных исходного распределения.



Посмотрите на пример, в этом примере используется предыдущий набор данных telecom\_churn. Мы обсудили важность характеристик этого набора данных, одной из наиболее важных характеристик является количество обращений в службу поддержки клиентов (звонки в службу поддержки клиентов). Визуализируйте функцию «количество звонков в службу поддержки клиентов» и посмотрите, как она распределяется.

**import warnings**

**import seaborn as sns**

**import pandas as pd**

**import numpy as np**

**from matplotlib import pyplot as plt**

**plt.style.use('ggplot')**

**plt.rcParams['figure.figsize'] = 10, 6**

**%matplotlib inline**

**warnings.filterwarnings('ignore')**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 1
* 2

**telecom\_data = pd.read\_csv(**

**'https://labfile.oss.aliyuncs.com/courses/1283/telecom\_churn.csv')**

* 1
* 2
* 3

fig = sns.kdeplot(telecom\_data[telecom\_data[‘Churn’]  
== False][‘Customer service calls’], label=‘Loyal’)  
fig = sns.kdeplot(telecom\_data[telecom\_data[‘Churn’]  
== True][‘Customer service calls’], label=‘Churn’)  
fig.set(xlabel=‘Number of calls’, ylabel=‘Density’)  
plt.show()

* 1
* 2

На приведенном выше рисунке показано, что лояльные клиенты реже обращаются в службу поддержки, чем клиенты, которые постепенно покидают сеть. Может быть полезно оценить среднее количество звонков клиентов для каждой группы клиентов, но поскольку набор данных невелик, если вы непосредственно вычисляете среднее значение исходной выборки, вы можете получить плохую оценку. Следовательно, метод Bootstrap можно использовать для создания 1000 новых выборок Bootstrap на основе исходной выборки, а затем вычислить интервальную оценку среднего значения.

Определение**get\_bootstrap\_samples()** Функция генерирует образцы Bootstrap и определяет**stat\_intervals()** Функция производит интервальную оценку.

**def get\_bootstrap\_samples(data, n\_samples):**

**indices = np.random.randint(0, len(data), (n\_samples, len(data)))**

**samples = data[indices]**

**return samples**

* 1
* 2
* 3
* 4
* 5

def stat\_intervals(stat, alpha):  
boundaries = np.percentile(  
stat, [100 \* alpha / 2., 100 \* (1 - alpha / 2.)])  
return boundaries

* 1
* 2

Разделите набор данных на постоянных клиентов и клиентов вне сети.

**loyal\_calls = telecom\_data.loc[telecom\_data['Churn'] == False,**

**'Customer service calls'].values**

**churn\_calls = telecom\_data.loc[telecom\_data['Churn'] == True,**

**'Customer service calls'].values**

* 1
* 2
* 3
* 4
* 5
* 1
* 2

Исправьте начальное число случайного числа, чтобы получить воспроизводимые результаты.

**np.random.seed(0)**

* 1
* 2
* 1
* 2

Используйте Bootstrap для создания выборок и вычисления их средних значений.

**loyal\_mean\_scores = [np.mean(sample)**

**for sample in get\_bootstrap\_samples(loyal\_calls, 1000)]**

**churn\_mean\_scores = [np.mean(sample)**

**for sample in get\_bootstrap\_samples(churn\_calls, 1000)]**

* 1
* 2
* 3
* 4
* 5
* 1
* 2

Чтобы напечатать оценочное значение интервала, используйте**stat\_intervals()** Функция определяет интервал как 95%.

**print("Service calls from loyal: mean interval",**

**stat\_intervals(loyal\_mean\_scores, 0.05))**

**print("Service calls from churn: mean interval",**

**stat\_intervals(churn\_mean\_scores, 0.05))**

* 1
* 2
* 3
* 4
* 5
* 1
* 2

Приведенные выше результаты показывают, что существует 95% -ная вероятность того, что среднее количество звонков в службу поддержки со стороны постоянных клиентов составляет от 1,4 до 1,49, в то время как среднее количество звонков в службу поддержки клиентов вне сети составляет от 2,06 до 2,40. Кроме того, можно заметить, что круг лояльных клиентов уже, что разумно, потому что лояльные клиенты звонят в службу поддержки реже (0, 1, 2 раза), чем автономные клиенты, которые звонят в службу поддержки несколько раз.

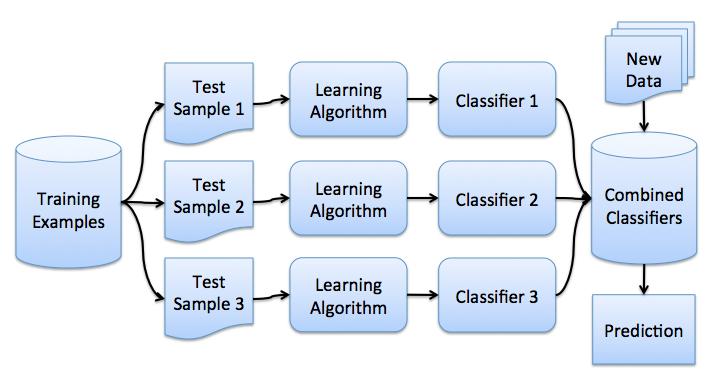
**Bagging**

Разобравшись с концепцией Bootstrap, давайте познакомимся с Bagging.

Предположим, у нас есть обучающий набор X. Мы используем Bootstrap для генерации образцовX1,…,XM X\_1, \dots, X\_MX1​,…,XM​. Теперь мы обучаем классификатор на каждом примере Bootstrap отдельно.ai(x) a\_i(x)ai​(x), Последний классификатор будет усреднять выходные данные всех этих индивидуальных классификаторов. В случае классификации это метод голосования:

a(x)=1M∑i=1Mai(x). a(x) = \frac{1}{M}\sum\_{i = 1}^M a\_i(x).a(x)=M1​i=1∑M​ai​(x).

Рисунок ниже наглядно поясняет приведенную выше формулу:



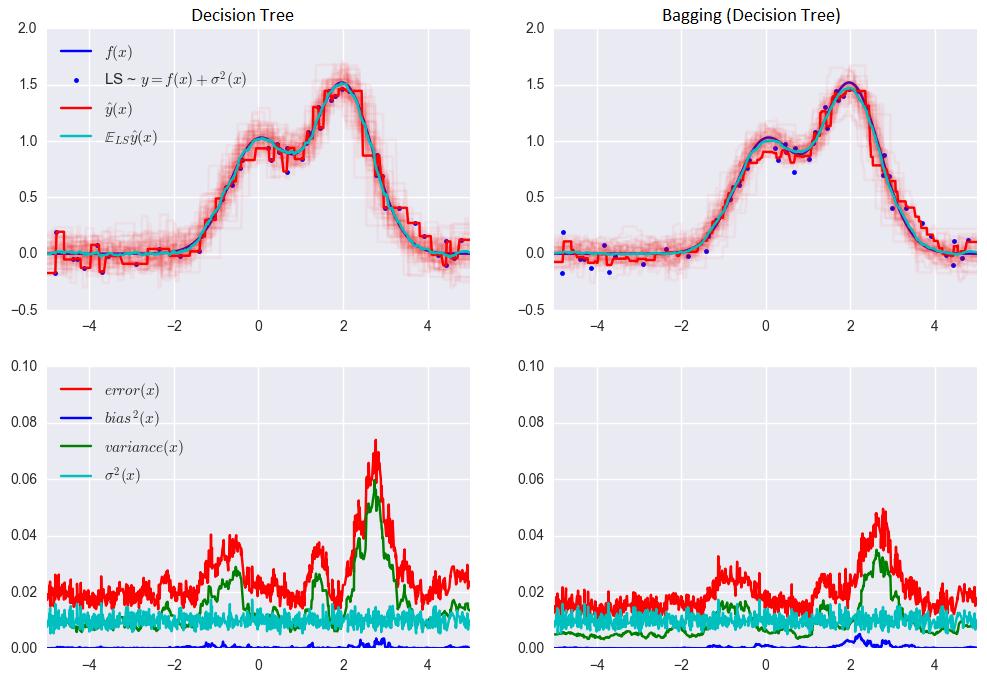
В задаче регрессии, взяв среднее значение результатов регрессии, Баггинг уменьшает среднеквадратичную ошибку до1M\frac{1}{M}M1​(M - количество регрессоров).

Вспоминая содержание предыдущего урока, ошибка предсказания модели состоит из трех частей:

Bias(f^)2+Var(f^)+σ2 \mathrm{Bias}\left(\hat{f}\right)^2 + \mathrm{Var}\left(\hat{f}\right) + \sigma^2Bias(f^​)2+Var(f^​)+σ2

Пакетирование уменьшает дисперсию классификатора за счет обучения моделей на разных наборах данных. Другими словами, упаковка в мешки может предотвратить переобучение. Поскольку модели из разных наборов обучающих данных различаются, их ошибки будут нейтрализовать друг друга в процессе голосования, поэтому использование пакетов эффективно. Кроме того, некоторые обучающие образцы Bootstrap могут игнорировать выбросы.

Давайте посмотрим на фактический эффект упаковки в мешки и сравним его с деревом решений, как показано на следующем рисунке:



Ошибка дерева решений:

0.0255(Err)=0.0003(Bias2)+0.0152(Var)+0.0098(σ2)0.0255 (\mathrm{Err}) = 0.0003 (\mathrm{Bias}^2) + 0.0152 (\mathrm{Var}) + 0.0098 (\sigma^2) 0.0255(Err)=0.0003(Bias2)+0.0152(Var)+0.0098(σ2)

Ошибка упаковки:

0.0196,(Err)=0.0004,(Bias2)+0.0092,(Var)+0.0098,(σ2) 0.0196 , (\mathrm{Err}) = 0.0004 , (\mathrm{Bias}^2) + 0.0092 , (\mathrm{Var}) + 0.0098 , (\sigma^2) 0.0196,(Err)=0.0004,(Bias2)+0.0092,(Var)+0.0098,(σ2)

Как видно из приведенного выше рисунка, по сравнению с деревом решений значение дисперсии ошибки упаковки значительно снижается.

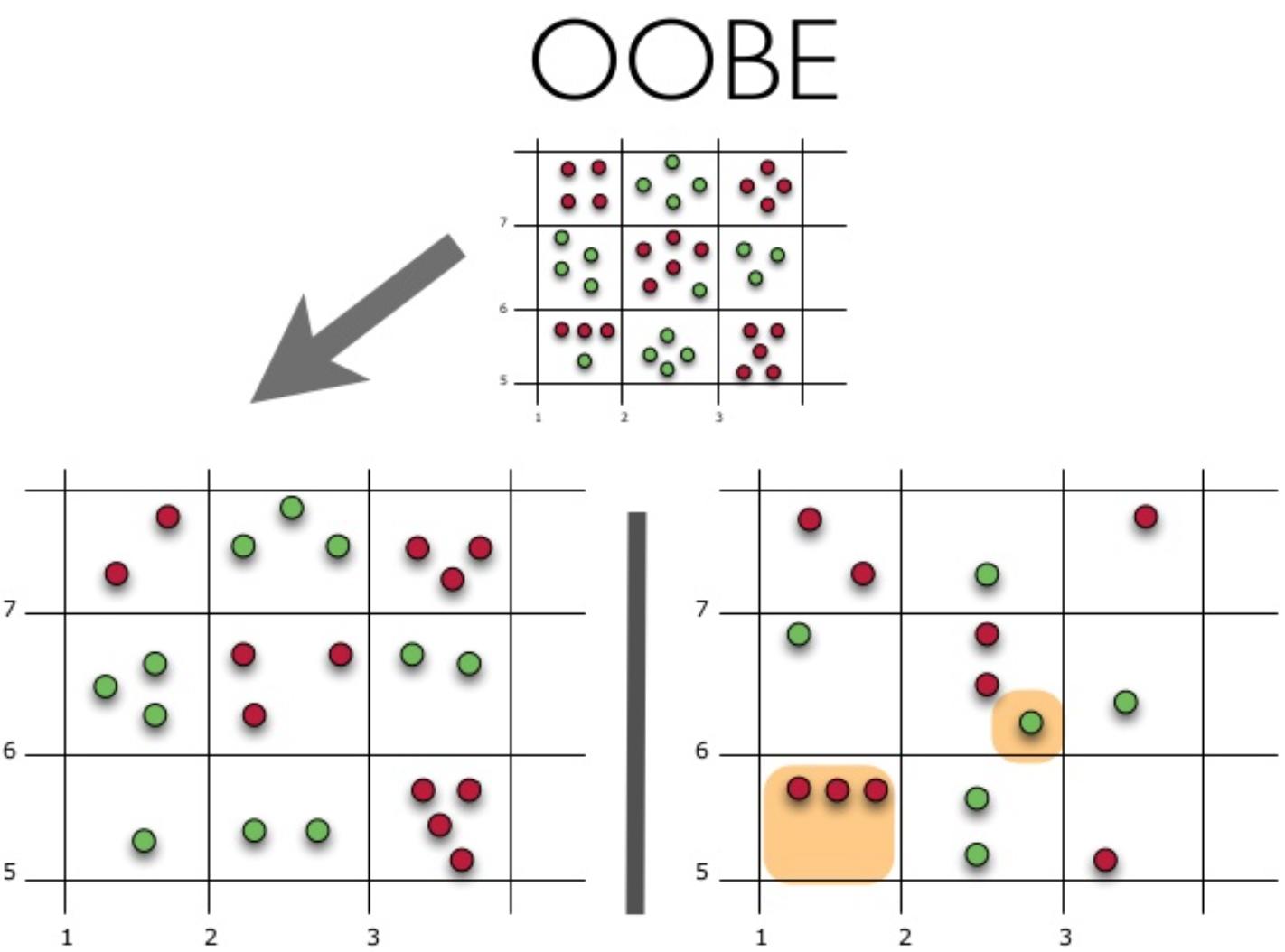
Пример на рисунке выше вряд ли появится в реальной работе, потому что он делает сильное предположение: отдельные ошибки не имеют значения. Для реальных приложений это сильное предположение слишком оптимистично. Когда это предположение не соответствует действительности, падение ошибки не будет столь значительным. В последующих экспериментах мы обсудим некоторые более сложные методы интеграции, которые могут сделать более точные прогнозы о реальных проблемах.

**Ошибка вне сумки (ошибка OOB)**

Случайный лес не требует перекрестной проверки или сохраненных выборок, поскольку он имеет встроенные оценки ошибок. Дерево решений в случайном лесу строится на основе различных выборок Bootstrap в исходном наборе данных. Для Kth-дерева его конкретный образец Bootstrap сохраняет примерно 37% входных данных.

Это легко доказать. Пусть в наборе данных естьℓ \ellℓ Образцы. На каждом шаге вероятность того, что каждая точка данных в конечном итоге появится в образце Bootstrap с заменой, равна1ℓ\frac{1}{\ell}ℓ1​. Вероятность того, что образец Bootstrap в конечном итоге не содержит определенного элемента набора данных (то есть элемент находится вℓ\ellℓ Нет ничьей в этом розыгрыше) равно(1−1ℓ)ℓ (1 - \frac{1}{\ell})^\ell(1−ℓ1​)ℓ. когда ℓ→+∞ \ell \rightarrow +\inftyℓ→+∞ , Эта вероятность равна1e \frac{1}{e}e1​. Следовательно, вероятность выбора конкретного образца≈1−1e≈63 \approx 1 - \frac{1}{e} \approx 63%≈1−e1​≈63。

На следующем рисунке показано, как работает оценка ошибки вне упаковки (OOBE):



Исходный набор данных находится вверху диаграммы. Разделите его на обучающий набор (левый рисунок) и тестовый набор (правый рисунок). На тестовом наборе нарисуйте пару сеток, которая отлично реализует классификацию. Теперь мы применяем ту же сетку к набору тестов, чтобы оценить правильную скорость классификации. Как вы можете видеть на правом рисунке, классификатор дал неправильные ответы на 4 точки данных, которые не вошли в обучающую выборку. В наборе тестов 15 точек данных, и эти 15 точек данных не появляются в обучающем наборе. Следовательно, точность этого классификатора равна1115∗100 \frac{11}{15}\*100% = 73.33%1511​∗100。

Подводя итог, можно сказать, что каждый базовый алгоритм обучен примерно на 63% исходных образцов. Алгоритм можно проверить на оставшихся 37% образцов. Оценка вне сумки - это просто средняя оценка базового алгоритма на 37% выборки.

**Случайный лес**

В Bagging основным используемым базовым классификатором является дерево решений, поскольку дерево решений довольно сложное и может обеспечить нулевую ошибку классификации для любой выборки. Метод случайного подпространства может уменьшить корреляцию дерева, тем самым избегая переобучения. Основанный на Bagging, основной алгоритм обучается на случайных подмножествах различных исходных наборов функций.

Следующий алгоритм использует метод случайного подпространства для построения модели ансамбля:

* Предположим, что количество образцов равно n, а количество размеров элемента равно d.
* Выберите количество моделей M.
* Для каждой модели m выберите количество функций dm <d. Все модели используют одно и то же значение dm.
* Для каждой модели m создайте обучающий набор, случайным образом выбрав функции dm из всего набора функций d.
* Обучите каждую модель.
* Комбинируя результаты моделей M, полученная общая модель применяется к новым данным, которые могут быть основаны на голосовании большинства или на агрегировании апостериорных вероятностей.

**Алгоритм случайного леса**

Алгоритм случайного леса для построения N деревьев выглядит следующим образом:

* Для каждогоk=1,…,N k = 1, \dots, Nk=1,…,NДля создания образцов BootstrapXk X\_kXk​。
* В образцеXk X\_kXk​ Создайте дерево решений наbk b\_kbk​：
  + Выберите лучший размер объекта в соответствии с заданными критериями. В соответствии с этой функцией образцы сегментируются для создания нового слоя дерева. Повторяйте этот процесс, пока образец не будет исчерпан.
  + Создавайте дерево до тех пор, пока любой листовой узел не будет содержать не болееnmin n\_\text{min}nmin​ Или достичь определенной глубины.
  + Каждый раз, когда вы разделяетесь, сначала начинайте сddd Случайно выбранный из оригинальных элементовmmm Функции, а затем ищите только лучшую сегментацию на этом подмножестве.

Окончательный классификатор определяется как:

a(x)=1N∑k=1Nbk(x) a(x) = \frac{1}{N}\sum\_{k = 1}^N b\_k(x)a(x)=N1​k=1∑N​bk​(x)

В задаче классификации используется алгоритм большинства, а в задаче регрессии - среднее. В задаче классификации рекомендуется установитьm=d m = \sqrt{d}m=d ​, Братьnmin=1 n\_\text{min} = 1nmin​=1. В задачах регрессии обычно принимаютm=d3 m = \frac{d}{3}m=3d​，nmin=5 n\_\text{min} = 5nmin​=5。

Случайный лес можно рассматривать как модифицированную версию дерева решений по упаковке, потому что случайный лес выбирает подпространство случайных признаков при делении каждого дерева решений по упаковке.

**Сравнение случайного леса с бэггингом и деревом решений**

Импортируйте необходимые пакеты и настройте среду:

**from \_\_future\_\_ import division, print\_function**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.datasets import make\_circles**

**from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier**

**from sklearn.ensemble import BaggingClassifier, BaggingRegressor**

**from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier**

**import seaborn as sns**

**from matplotlib import pyplot as plt**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

**Disable warnings in Anaconda**

import warnings  
import numpy as np  
warnings.filterwarnings(‘ignore’)

%matplotlib inline  
plt.style.use(‘ggplot’)  
plt.rcParams[‘figure.figsize’] = 10, 6

n\_train = 150  
n\_test = 1000  
noise = 0.1

* 1
* 2

Сгенерируйте данные.

**def f(x):**

**x = x.ravel()**

**return np.exp(-x \*\* 2) + 1.5 \* np.exp(-(x - 2) \*\* 2)**

**def generate(n\_samples, noise):**

**X = np.random.rand(n\_samples) \* 10 - 5**

**X = np.sort(X).ravel()**

**y = np.exp(-X \*\* 2) + 1.5 \* np.exp(-(X - 2) \*\* 2)\**

**+ np.random.normal(0.0, noise, n\_samples)**

**X = X.reshape((n\_samples, 1))**

**return X, y**

**X\_train, y\_train = generate(n\_samples=n\_train, noise=noise)**

**X\_test, y\_test = generate(n\_samples=n\_test, noise=noise)**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 1
* 2

Используйте регрессию с одним деревом решений.

**dtree = DecisionTreeRegressor().fit(X\_train, y\_train)**

**d\_predict = dtree.predict(X\_test)**

**plt.figure(figsize=(10, 6))**

**plt.plot(X\_test, f(X\_test), "b")**

**plt.scatter(X\_train, y\_train, c="b", s=20)**

**plt.plot(X\_test, d\_predict, "g", lw=2)**

**plt.xlim([-5, 5])**

**plt.title("Decision tree, MSE = %.2f"**

**% np.sum((y\_test - d\_predict) \*\* 2))**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 1
* 2

Используйте регрессию дерева решений по упаковке.

**bdt = BaggingRegressor(DecisionTreeRegressor()).fit(X\_train, y\_train)**

**bdt\_predict = bdt.predict(X\_test)**

**plt.figure(figsize=(10, 6))**

**plt.plot(X\_test, f(X\_test), "b")**

**plt.scatter(X\_train, y\_train, c="b", s=20)**

**plt.plot(X\_test, bdt\_predict, "y", lw=2)**

**plt.xlim([-5, 5])**

**plt.title("Bagging for decision trees, MSE = %.2f" %**

**np.sum((y\_test - bdt\_predict) \*\* 2))**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 1
* 2

Использовать случайный лес.

**rf = RandomForestRegressor(n\_estimators=10).fit(X\_train, y\_train)**

**rf\_predict = rf.predict(X\_test)**

**plt.figure(figsize=(10, 6))**

**plt.plot(X\_test, f(X\_test), "b")**

**plt.scatter(X\_train, y\_train, c="b", s=20)**

**plt.plot(X\_test, rf\_predict, "r", lw=2)**

**plt.xlim([-5, 5])**

**plt.title("Random forest, MSE = %.2f" % np.sum((y\_test - rf\_predict) \*\* 2))**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 1
* 2

Из трех приведенных выше изображений и значений MSE видно, что случайный лес из 10 деревьев работает лучше, чем одно дерево решений и бэггинг из 10 деревьев. Конечно, случайный лес в этом примере нестабилен. Результаты нескольких запусков Покажите, что в случайном лесу и в дереве решений по сумке есть победители и проигравшие. Основное различие между Random Forest и Bagging заключается в том, что лучшая функция первой сегментации выбирается из подпространства случайных функций, тогда как последняя будет учитывать все функции во время сегментации.

Далее мы посмотрим на эффективность случайного леса и бэггинга в задаче классификации.

**np.random.seed(42)**

**X, y = make\_circles(n\_samples=500, factor=0.1, noise=0.35, random\_state=42)**

**X\_train\_circles, X\_test\_circles, y\_train\_circles, y\_test\_circles = train\_test\_split(**

**X, y, test\_size=0.2)**

* 1
* 2
* 3
* 4
* 5

dtree = DecisionTreeClassifier(random\_state=42)  
dtree.fit(X\_train\_circles, y\_train\_circles)

x\_range = np.linspace(X.min(), X.max(), 100)  
xx1, xx2 = np.meshgrid(x\_range, x\_range)  
y\_hat = dtree.predict(np.c\_[xx1.ravel(), xx2.ravel()])  
y\_hat = y\_hat.reshape(xx1.shape)  
plt.contourf(xx1, xx2, y\_hat, alpha=0.2)  
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=‘autumn’)  
plt.title(“Decision tree”)  
plt.show()

b\_dtree = BaggingClassifier(  
DecisionTreeClassifier(), n\_estimators=300, random\_state=42)  
b\_dtree.fit(X\_train\_circles, y\_train\_circles)

x\_range = np.linspace(X.min(), X.max(), 100)  
xx1, xx2 = np.meshgrid(x\_range, x\_range)  
y\_hat = b\_dtree.predict(np.c\_[xx1.ravel(), xx2.ravel()])  
y\_hat = y\_hat.reshape(xx1.shape)  
plt.contourf(xx1, xx2, y\_hat, alpha=0.2)  
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=‘autumn’)  
plt.title(“Bagging (decision trees)”)  
plt.show()

rf = RandomForestClassifier(n\_estimators=300, random\_state=42)  
rf.fit(X\_train\_circles, y\_train\_circles)

x\_range = np.linspace(X.min(), X.max(), 100)  
xx1, xx2 = np.meshgrid(x\_range, x\_range)  
y\_hat = rf.predict(np.c\_[xx1.ravel(), xx2.ravel()])  
y\_hat = y\_hat.reshape(xx1.shape)  
plt.contourf(xx1, xx2, y\_hat, alpha=0.2)  
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=‘autumn’)  
plt.title(“Random forest”)  
plt.show()

* 1
* 2

На приведенном выше рисунке показано, что граница, образованная деревом решений, является неровной и имеет большое количество острых углов, что означает явление переобучения. Напротив, граница между Random Forest и Bagging довольно гладкая, и нет явных признаков переобучения.

Теперь давайте посмотрим, какие параметры помогают повысить точность модели случайного леса.

**Параметры случайного леса**

Библиотека scikit-learn предоставляет**BaggingRegressor**с **BaggingClassifier** Class, некоторые параметры, на которые необходимо обратить внимание при создании новой модели, следующие:

* n\_estimators - количество деревьев в случайном лесу;
* Criterion - функция для измерения качества сегментации;
* max\_features - количество функций, учитываемых при поиске лучшей сегментации;
* min\_samples\_leaf - минимальное количество выборок листовых узлов;
* max\_depth - максимальная глубина дерева.

**Применение случайного леса в реальных задачах**

В качестве примера воспользуйтесь набором автономных данных о клиентах предыдущего оператора. Это проблема классификации, и показатель точности используется для оценки модели.

Сначала создайте простой классификатор в качестве основы. Для простоты для построения классификатора используются только числовые признаки.

**import pandas as pd**

**from sklearn.model\_selection import cross\_val\_score, StratifiedKFold, GridSearchCV**

**from sklearn.metrics import accuracy\_score**

* 1
* 2
* 3
* 4
* 1
* 2

Загрузите набор данных.

**df = pd.read\_csv(**

**"https://labfile.oss.aliyuncs.com/courses/1283/telecom\_churn.csv")**

* 1
* 2
* 3
* 1
* 2

Выберите числовые характеристики.

**cols = []**

**for i in df.columns:**

**if (df[i].dtype == "float64") or (df[i].dtype == 'int64'):**

**cols.append(i)**

* 1
* 2
* 3
* 4
* 5
* 1
* 2

Разделите набор данных на входные и целевые переменные.

**X, y = df[cols].copy(), np.asarray(df["Churn"], dtype='int8')**

* 1
* 2
* 1
* 2

Выполните иерархическую сегментацию процесса проверки.

**skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)**

* 1
* 2
* 1
* 2

Классификатор инициализируется на основе параметров по умолчанию.

**rfc = RandomForestClassifier(random\_state=42, n\_jobs=-1, oob\_score=True)**

* 1
* 2
* 1
* 2

Тренируйтесь на тренировочном наборе.

**results = cross\_val\_score(rfc, X, y, cv=skf)**

* 1
* 2
* 1
* 2

Оцените точность на тестовом наборе.

**print("CV accuracy score: {:.2f}%".format(results.mean()\*100))**

* 1
* 2
* 1
* 2

В качестве базового показателя получена точность 91,48%. Теперь попробуйте улучшить результаты, глядя на кривую обучения при изменении основных параметров. Сначала, начиная с количества деревьев улучшений, инициализируется K-кратная перекрестная проверка.

**skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)**

* 1
* 2
* 1
* 2

Создайте список для хранения показателей точности на обучающем наборе и наборе тестов.

**train\_acc = []**

**test\_acc = []**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

* 1
* 2
* 3
* 4
* 5
* 1
* 2

Выполните поиск по сетке. В приведенном ниже списке есть много возможных значений, которые увеличивают время выполнения онлайн-среды. Вы можете уменьшить число самостоятельно, чтобы ускорить операцию.

**trees\_grid = [5, 10, 15, 20, 30, 50, 75, 100]**

* 1
* 2
* 1
* 2

Обучите модель на обучающем наборе.

**for ntrees in trees\_grid:**

**rfc = RandomForestClassifier(**

**n\_estimators=ntrees, random\_state=42, n\_jobs=-1, oob\_score=True)**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

**for train\_index, test\_index in skf.split(X, y):**

**X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]**

**y\_train, y\_test = y[train\_index], y[test\_index]**

**rfc.fit(X\_train, y\_train)**

**temp\_train\_acc.append(rfc.score(X\_train, y\_train))**

**temp\_test\_acc.append(rfc.score(X\_test, y\_test))**

**train\_acc.append(temp\_train\_acc)**

**test\_acc.append(temp\_test\_acc)**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 1
* 2

Распечатайте результат.

**train\_acc, test\_acc = np.asarray(train\_acc), np.asarray(test\_acc)**

**print("Best accuracy on CV is {:.2f}% with {} trees".format(max(test\_acc.mean(axis=1))\*100,**

**trees\_grid[np.argmax(test\_acc.mean(axis=1))]))**

* 1
* 2
* 3
* 4
* 1
* 2

Когда количество деревьев изменяется на 50, получается точность 92,44%. Затем нарисуйте соответствующую кривую обучения.

**plt.style.use('ggplot')**

* 1
* 2

fig, ax = plt.subplots(figsize=(8, 4))  
ax.plot(trees\_grid, train\_acc.mean(axis=1),  
alpha=0.5, color=‘blue’, label=‘train’)  
ax.plot(trees\_grid, test\_acc.mean(axis=1), alpha=0.5, color=‘red’, label=‘cv’)  
ax.fill\_between(trees\_grid, test\_acc.mean(axis=1) - test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + test\_acc.std(axis=1), color=’#888888’, alpha=0.4)  
ax.fill\_between(trees\_grid, test\_acc.mean(axis=1) - 2\*test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + 2\*test\_acc.std(axis=1), color=’#888888’, alpha=0.2)  
ax.legend(loc=‘best’)  
ax.set\_ylim([0.88, 1.02])  
ax.set\_ylabel(“Accuracy”)  
ax.set\_xlabel(“N\_estimators”)

* 1
* 2

На приведенном выше рисунке показано, что когда количество деревьев увеличивается до определенного значения, точность тестового набора больше не улучшается, а точность обучающего набора достигает 100%, что означает, что происходит переобучение.

Чтобы избежать переобучения, в модель необходимо добавить параметры регуляризации. Давайте зафиксируем количество деревьев на 100, а затем посмотрим на эффект различных max\_depth, сначала создадим список, чтобы сохранить точность на обучающем и тестовом наборах.

**train\_acc = []**

**test\_acc = []**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

**max\_depth\_grid = [3, 5, 7, 9, 11, 13, 15, 17, 20, 22, 24]**

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2

Обучите модель на обучающем наборе.

**for max\_depth in max\_depth\_grid:**

**rfc = RandomForestClassifier(**

**n\_estimators=100, random\_state=42, n\_jobs=-1, oob\_score=True, max\_depth=max\_depth)**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

**for train\_index, test\_index in skf.split(X, y):**

**X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]**

**y\_train, y\_test = y[train\_index], y[test\_index]**

**rfc.fit(X\_train, y\_train)**

**temp\_train\_acc.append(rfc.score(X\_train, y\_train))**

**temp\_test\_acc.append(rfc.score(X\_test, y\_test))**

**train\_acc.append(temp\_train\_acc)**

**test\_acc.append(temp\_test\_acc)**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 1
* 2

Распечатайте результат.

**train\_acc, test\_acc = np.asarray(train\_acc), np.asarray(test\_acc)**

**print("Best accuracy on CV is {:.2f}% with {} max\_depth".format(max(test\_acc.mean(axis=1))\*100,**

**max\_depth\_grid[np.argmax(test\_acc.mean(axis=1))]))**

* 1
* 2
* 3
* 4

fig, ax = plt.subplots(figsize=(8, 4))  
ax.plot(max\_depth\_grid, train\_acc.mean(axis=1),  
alpha=0.5, color=‘blue’, label=‘train’)  
ax.plot(max\_depth\_grid, test\_acc.mean(axis=1),  
alpha=0.5, color=‘red’, label=‘cv’)  
ax.fill\_between(max\_depth\_grid, test\_acc.mean(axis=1) - test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + test\_acc.std(axis=1), color=’#888888’, alpha=0.4)  
ax.fill\_between(max\_depth\_grid, test\_acc.mean(axis=1) - 2*test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + 2*test\_acc.std(axis=1), color=’#888888’, alpha=0.2)  
ax.legend(loc=‘best’)  
ax.set\_ylim([0.88, 1.02])  
ax.set\_ylabel(“Accuracy”)  
ax.set\_xlabel(“Max\_depth”)

* 1
* 2

max\_depth играет в модели регуляризирующую роль, явление чрезмерной подгонки было уменьшено, а точность модели была улучшена.

Еще один важный параметр, который стоит отрегулировать, - min\_samples\_leaf, который также может играть регуляризирующую роль. Сначала создайте список, чтобы сохранить точность для тренировочного и тестового наборов.

**train\_acc = []**

**test\_acc = []**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

**min\_samples\_leaf\_grid = [1, 3, 5, 7, 9, 11, 13, 15, 17, 20, 22, 24]**

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2

Обучите модель на обучающем наборе.

**for min\_samples\_leaf in min\_samples\_leaf\_grid:**

**rfc = RandomForestClassifier(n\_estimators=100, random\_state=42, n\_jobs=-1,**

**oob\_score=True, min\_samples\_leaf=min\_samples\_leaf)**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

**for train\_index, test\_index in skf.split(X, y):**

**X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]**

**y\_train, y\_test = y[train\_index], y[test\_index]**

**rfc.fit(X\_train, y\_train)**

**temp\_train\_acc.append(rfc.score(X\_train, y\_train))**

**temp\_test\_acc.append(rfc.score(X\_test, y\_test))**

**train\_acc.append(temp\_train\_acc)**

**test\_acc.append(temp\_test\_acc)**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 1
* 2

Распечатайте результат.

**train\_acc, test\_acc = np.asarray(train\_acc), np.asarray(test\_acc)**

**print("Best accuracy on CV is {:.2f}% with {} min\_samples\_leaf".format(max(test\_acc.mean(axis=1))\*100,**

**min\_samples\_leaf\_grid[np.argmax(test\_acc.mean(axis=1))]))**

* 1
* 2
* 3
* 4

fig, ax = plt.subplots(figsize=(8, 4))  
ax.plot(min\_samples\_leaf\_grid, train\_acc.mean(  
axis=1), alpha=0.5, color=‘blue’, label=‘train’)  
ax.plot(min\_samples\_leaf\_grid, test\_acc.mean(  
axis=1), alpha=0.5, color=‘red’, label=‘cv’)  
ax.fill\_between(min\_samples\_leaf\_grid, test\_acc.mean(axis=1) - test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + test\_acc.std(axis=1), color=’#888888’, alpha=0.4)  
ax.fill\_between(min\_samples\_leaf\_grid, test\_acc.mean(axis=1) - 2*test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + 2*test\_acc.std(axis=1), color=’#888888’, alpha=0.2)  
ax.legend(loc=‘best’)  
ax.set\_ylim([0.88, 1.02])  
ax.set\_ylabel(“Accuracy”)  
ax.set\_xlabel(“Min\_samples\_leaf”)

* 1
* 2

В этом случае точность набора для проверки не улучшилась, но когда точность набора для проверки оставалась выше 92%, ситуация с избыточной подгонкой уменьшилась на 2%.

Рассмотрим параметр max\_features ниже. В задаче классификации характеристикой по умолчанию является «квадратный корень из числа всех функций». Сначала проверьте, является ли выбор из 4 функций в этом примере лучшим, создайте список, чтобы сохранить точность наборов для обучения и тестирования.

**train\_acc = []**

**test\_acc = []**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

**max\_features\_grid = [2, 4, 6, 8, 10, 12, 14, 16]**

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2

Обучите модель на обучающем наборе.

**for max\_features in max\_features\_grid:**

**rfc = RandomForestClassifier(n\_estimators=100, random\_state=42, n\_jobs=-1,**

**oob\_score=True, max\_features=max\_features)**

**temp\_train\_acc = []**

**temp\_test\_acc = []**

**for train\_index, test\_index in skf.split(X, y):**

**X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]**

**y\_train, y\_test = y[train\_index], y[test\_index]**

**rfc.fit(X\_train, y\_train)**

**temp\_train\_acc.append(rfc.score(X\_train, y\_train))**

**temp\_test\_acc.append(rfc.score(X\_test, y\_test))**

**train\_acc.append(temp\_train\_acc)**

**test\_acc.append(temp\_test\_acc)**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 1
* 2

Распечатайте результат.

**train\_acc, test\_acc = np.asarray(train\_acc), np.asarray(test\_acc)**

**print("Best accuracy on CV is {:.2f}% with {} max\_features".format(max(test\_acc.mean(axis=1))\*100,**

**max\_features\_grid[np.argmax(test\_acc.mean(axis=1))]))**

* 1
* 2
* 3
* 4

fig, ax = plt.subplots(figsize=(8, 4))  
ax.plot(max\_features\_grid, train\_acc.mean(axis=1),  
alpha=0.5, color=‘blue’, label=‘train’)  
ax.plot(max\_features\_grid, test\_acc.mean(axis=1),  
alpha=0.5, color=‘red’, label=‘cv’)  
ax.fill\_between(max\_features\_grid, test\_acc.mean(axis=1) - test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + test\_acc.std(axis=1), color=’#888888’, alpha=0.4)  
ax.fill\_between(max\_features\_grid, test\_acc.mean(axis=1) - 2*test\_acc.std(axis=1),  
test\_acc.mean(axis=1) + 2*test\_acc.std(axis=1), color=’#888888’, alpha=0.2)  
ax.legend(loc=‘best’)  
ax.set\_ylim([0.88, 1.02])  
ax.set\_ylabel(“Accuracy”)  
ax.set\_xlabel(“Max\_features”)

* 1
* 2

На рисунке выше показано, что в этом примере модель работает лучше всего, когда max\_features = 12.

Мы видели, как меняется кривая обучения при изменении основных параметров. Теперь просто используйте**GridSearch()** Чтобы найти лучшие параметры. Сначала инициализируйте набор параметров для исчерпывающего поиска и подгонки.

**# Исходные параметры поиска эксперимента**

**# parameters = {'max\_features': [4, 7, 10, 13], 'min\_samples\_leaf': [**

**# 1, 3, 5, 7], 'max\_depth': [5, 10, 15, 20]}**

**# Чтобы ускорить выполнение онлайн-среды, оптимизируйте параметры поиска**

**parameters = {'max\_features': [10, 13], 'min\_samples\_leaf': [1, 3], 'max\_depth': [5, 10]}**

**rfc = RandomForestClassifier(n\_estimators=100, random\_state=42,**

**n\_jobs=-1, oob\_score=True)**

**gcv = GridSearchCV(rfc, parameters, n\_jobs=-1, cv=skf, verbose=1)**

**gcv.fit(X, y)**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 1
* 2

Распечатайте результат.

**gcv.best\_estimator\_, gcv.best\_score\_**

* 1
* 2
* 1
* 2

После настройки параметров точность модели достигла 92,7%. Самым важным моментом случайного леса является то, что его точность не будет снижаться с увеличением количества деревьев. Это означает, что вы можете использовать 10 деревьев для настройки гиперпараметров, а затем увеличить количество деревьев с 10 до 500 для повышения точности.

**Случайная дисперсия леса и декорреляция**

Дисперсия случайного леса может быть выражена следующим образом:

Varf(x)=ρ(x)σ2(x) \mathrm{Var} f(x) = \rho(x)\sigma^2(x)Varf(x)=ρ(x)σ2(x)

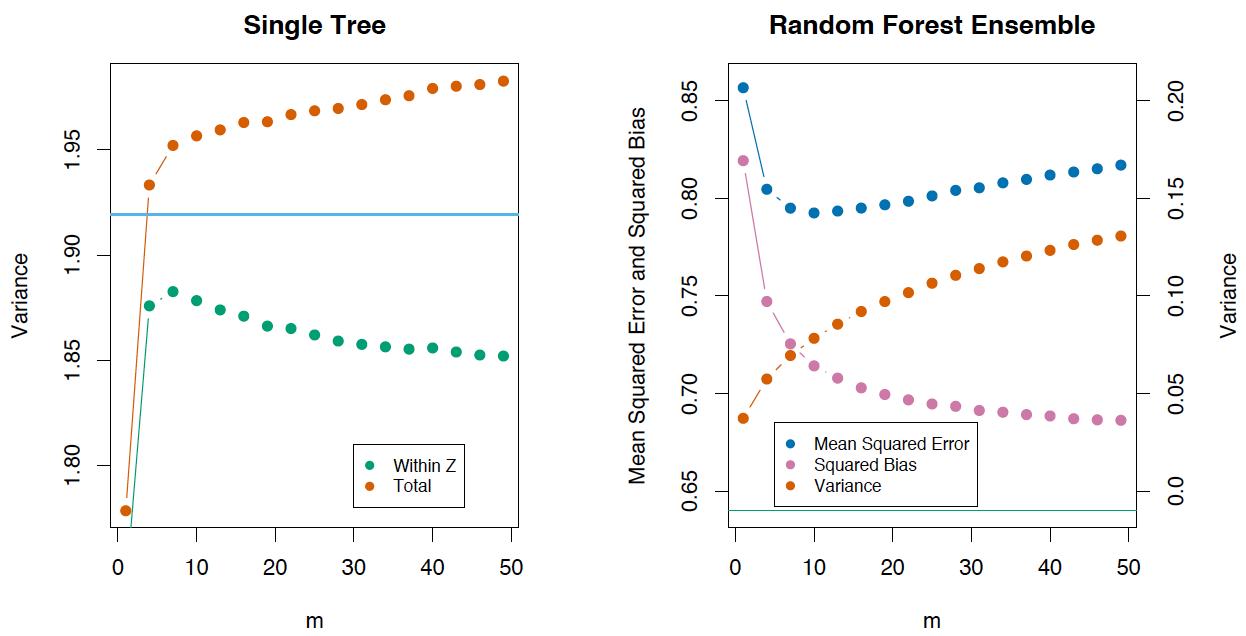
ρ(x)=Corr[T(x1,Θ1(Z)),T(x2,Θ2(Z))], \rho(x) = \mathrm{Corr}\left[T(x\_1,\Theta\_1(Z)),T(x\_2,\Theta\_2(Z))\right],ρ(x)=Corr[T(x1​,Θ1​(Z)),T(x2​,Θ2​(Z))],

среди них,

* ρ(x) \rho(x)ρ(x) Корреляция выборки между любыми двумя деревьями;
* Θ1(Z) \Theta\_1(Z)Θ1​(Z) с Θ2(Z) \Theta\_2(Z)Θ2​(Z) Для образцаZZZ На случайных элементах случайным образом выбирается пара деревьев;
* T(x,Θi(Z)) T(x,\Theta\_i(Z))T(x,Θi​(Z)) Во-первыхiii Древовидный классификатор во входном вектореxxx Выход на;
* σ2(x) \sigma^2(x)σ2(x) Для выборочной дисперсии на любом случайно выбранном дереве:σ2(x)=Var[T(x,Θ(X))]\sigma^2(x) = \mathrm{Var}\left[T(x,\Theta(X))\right]σ2(x)=Var[T(x,Θ(X))]

Легкоρ(x) \rho(x)ρ(x) Непонимание того, что средняя корреляция обученных деревьев в случайном лесу на самом деле неверна.ρ(x) \rho(x)ρ(x) Пара случайных деревьев на входеxxx Теоретическая корреляция вышеуказанной оценки, ее значение выводится из обучающего набора с повторной выборкой и пары дерева решений, случайно выбранной впоследствии. Статистически эта корреляция вызванаZ ZZ с Θ \ThetaΘ Вызвано распределением выборки.

Условная корреляция любой пары деревьев равна 0, потому что Bootstrap и выбор функций независимы и одинаково распределены. Если вы рассматриваете дисперсию одного дерева, на нее почти не влияют параметры разделения (m mm), но параметр сегментации является ключевым параметром интеграции. Кроме того, дисперсия одного дерева решений намного выше, чем дисперсия ансамбля, как показано на следующем рисунке: (Рисунок из «Элементы статистического обучения»)



**Смещение случайного леса**

Отклонение случайного леса и мешков такое же, как и для одного дерева решений:

\begin{array}{rcl} \mathrm{Bias} &=& \mu(x) - \mathrm{E}*Z , f*{rf}(x) \ &=& \mu(x) - \mathrm{E}*Z , \mathrm{E}*{\Theta | Z} , T(x,\Theta(Z))\end{array}

С точки зрения абсолютного значения, отклонение случайного леса и упаковки обычно больше, чем отклонение одного дерева решений, потому что случайный процесс и сокращение пространства выборки ограничивают модель. Следовательно, причина того, что точность прогнозирования Bagging и Random Forest выше, чем у одного дерева решений, заключается только в уменьшении дисперсии.

**Экстремальное случайное дерево**

Чрезвычайно рандомизированные деревья применяют больше случайности, когда узлы раздваиваются. Подобно случайным лесам, экстремально случайные деревья используют подпространство случайных признаков. Однако экстремальные случайные числа не ищут лучший порог. Напротив, он случайным образом генерирует порог для каждой возможной функции, затем выбирает функцию, соответствующую лучшему порогу для разделения узла, и уменьшает дисперсию, добавляя небольшое количество отклонения.

библиотека scikit-learn**ExtraTreesClassifier()** с **ExtraTreesRegressor()** Класс реализует этот метод. Если вы столкнулись с серьезным переоснащением при использовании случайного леса или повышения градиента, вы можете попробовать экстремально случайные деревья.

**Сходство между случайным лесом и k ближайшими соседями**

Метод случайного леса и метод k-ближайших соседей имеют сходство. Прогноз случайного леса основан на метках похожих выборок в обучающей выборке. Чем чаще эти выборки появляются в одном листовом узле, тем выше их сходство.

Чтобы доказать это, рассмотрим задачу регрессии квадратичной функции потерь:Tn(x) T\_n(x)Tn​(x) Для вводаxxx В случайном лесуnnn Количество листовых узлов дерева. Алгоритм ввода вектораxxx Ответ равен всем выпавшим листовым узламTn(x) T\_n(x)Tn​(x) Средний отклик обучающих выборок.

bn(x)=∑i=1lwn(x,xi)yi, b\_n(x) = \sum\_{i=1}^{l}w\_n(x,x\_i)y\_i,bn​(x)=i=1∑l​wn​(x,xi​)yi​,

среди них,

wn(x,xi)=[Tn(x)=Tn(xi)]∑j=1l[Tn(x)=Tn(xj)] w\_n(x, x\_i) = \frac{\left[T\_n(x) = T\_n(x\_i)\right]}{\sum\_{j=1}^{l}\left[T\_n(x) = T\_n(x\_j)\right]}wn​(x,xi​)=∑j=1l​[Tn​(x)=Tn​(xj​)][Tn​(x)=Tn​(xi​)]​

Итак, состав ответа следующий:

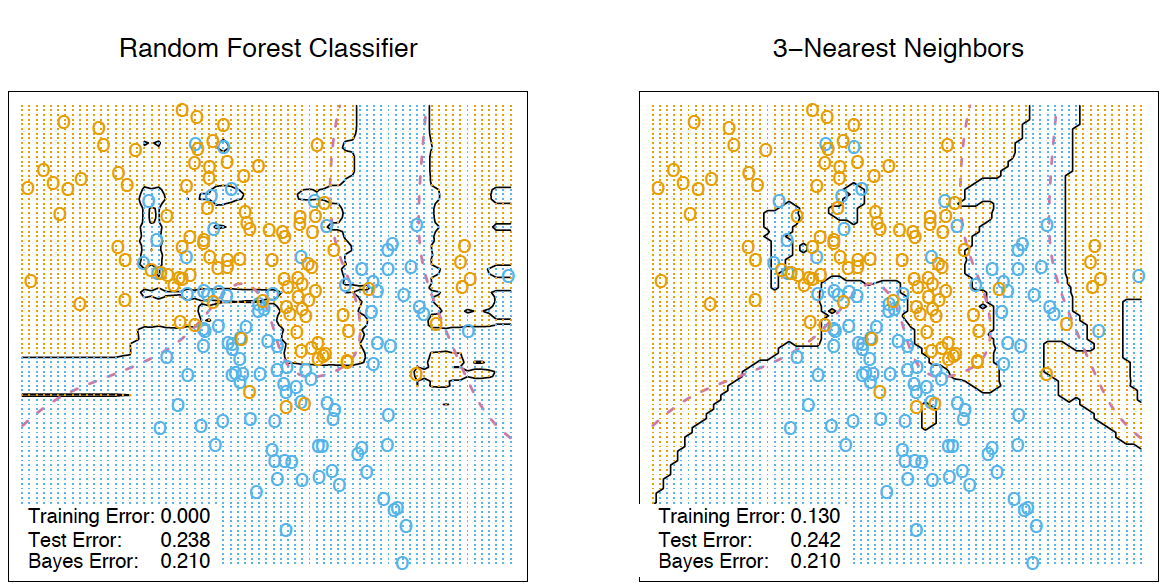
$$ \begin{array}{rcl} a\_n(x) &=& \frac{1}{N}\sum\_{n=1}^{N}\sum\_{i=1}^{l}w\_n(x,x\_i)y\_i \ &=& \sum\_{i=1}^{l}\left(\frac{1}{N}\sum\_{j=1}^{N}w\_n(x,x\_j)\right)y\_i \end{array}$$

Приведенная выше формула показывает, что ответ случайного леса представляет собой взвешенную сумму ответов всех обучающих выборок.

При этом стоит отметить, что примерыxxx Конечное количество листовых узловTn(x) T\_n(x)Tn​(x) Само по себе это ценная особенность. Например, мы можем сделать это:

* На основе технологии случайного леса или повышения градиента для обучения составной модели деревьев решений на выборках
* Категориальные характеристикиT1(x),…,Tn(x) T\_1(x), \dots, T\_n(x)T1​(x),…,Tn​(x) Добавьте образец.

T1(x),…,Tn(x) T\_1(x), \dots, T\_n(x)T1​(x),…,Tn​(x) Характеристики являются результатом нелинейной пространственной сегментации, и они предоставляют информацию о сходстве между выборками. В книге «Элементы статистического обучения» есть пример, демонстрирующий сходство между технологиями случайного леса и k-ближайшего соседа, как показано на следующем рисунке:



**Преобразуйте набор данных в многомерное представление**

Случайный лес в основном используется для обучения с учителем, но его также можно применить к обучению без учителя. Используйте scikit-learn**RandomTreesEmbedding()** Метод, набор данных можно преобразовать в разреженное представление большой размерности, а затем использовать в обучении без учителя.

Сначала создайте несколько чрезвычайно случайных деревьев, а затем используйте индекс листового узла, содержащий образец, в качестве новой функции. Например, если первый листовой узел содержит входные данные, присвойте 1 как значение функции; в противном случае присвойте 0. Это называется двоичным кодированием, и мы можем контролировать количество функций и разреженность, увеличивая или уменьшая количество и глубину деревьев. Поскольку близкие точки данных (соседи) имеют тенденцию попадать в один листовой узел, это преобразование обеспечивает неявную непараметрическую оценку плотности точек данных.

**Преимущества и недостатки случайных лесов**

Преимущество:

* Высокая точность прогнозов. Он работает лучше, чем линейные алгоритмы для большинства задач, и его точность эквивалентна Boosting;
* Случайная выборка приводит к большей устойчивости к дискретным значениям;
* Случайный выбор подпространства приводит к нечувствительности к масштабированию признаков и другим монотонным преобразованиям;
* Никакой точной настройки параметров не требуется.
* Очень эффективен для наборов данных с большим количеством функций и категорий;
* Могут обрабатываться как непрерывные, так и дискретные значения;
* Он не склонен к переобучению. На практике увеличение количества деревьев почти всегда улучшает общую производительность. Однако, когда количество деревьев увеличивается до определенного значения, кривая обучения стабилизируется;
* Существуют теоретические методы оценки важности функций;
* Умение хорошо справляться с отсутствующими данными. Даже если большая часть данных отсутствует, точность сохраняется;
* Поддержка взвешенной классификации для всего набора данных и одного образца дерева;
* Вычисление подобия, используемое в нижней части дерева решений, может использоваться для последующей кластеризации, обнаружения дискретных значений или представления данных интереса;
* Вышеупомянутые функции и свойства могут быть расширены до немаркированных данных для поддержки неконтролируемой кластеризации, визуализации данных и обнаружения дискретных значений;
* Легко распараллеливать и масштабировать.

Недостатки:

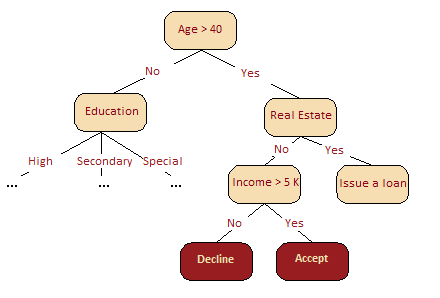
* По сравнению с одним деревом решений, выходные данные случайного леса труднее интерпретировать;
* Не существует формального p-значения для оценки важности функции;
* Когда данные немногочисленны (например, ввод текста, набор слов), производительность не так хороша, как у линейной модели;
* В отличие от линейной регрессии, случайные леса не могут быть экстраполированы. Однако это не приведет к экстремальным значениям из-за наличия дискретных значений;
* При некоторых проблемах легко переиграть, особенно при работе с данными с высоким уровнем шума;
* При работе с данными разных порядков величины Random Forest фокусируется на данных более высоких порядков, потому что это может более очевидно повысить точность;
* Полученная модель относительно велика и для ее поддержки требуется ОЗУ большой емкости.

**Важность функций**

Нам часто нужно объяснять выходные результаты алгоритма.Если мы не можем полностью понять алгоритм, мы, по крайней мере, надеемся выяснить, какая входная функция больше всего влияет на результат. На основе случайного леса мы можем довольно легко получить такую ​​информацию.

**Суть метода**

Рисунок ниже ясно показывает, что в задаче кредитного скоринга возраст важнее дохода. Исходя из концепции получения информации, мы можем объяснить это формально.



В случайных лесах, чем ближе объект к корню дерева во всех деревьях, тем важнее эта функция в данной задаче классификации или регрессии. Согласно стандарту сегментации, выигрыш (например, примесь Джини), полученный при каждой оптимальной сегментации каждого дерева, является мерой важности, непосредственно связанной с функцией сегментации.

Рассмотрим подробнее. Вычисляя ошибку вне пакета, можно определить, какая переменная привела к снижению средней точности. Какая переменная вызывает большее снижение точности, тем выше оценка важности этой переменной. Степень уменьшения примеси Джини или MSE в задаче регрессии представляет собой степень вклада каждой переменной в однородность полученного узла случайной модели леса. Более высокое уменьшение указывает на то, что сегментация на основе этой переменной может получить узел более высокой чистоты.

Приведенный выше анализ можно выразить следующей формулой:

VI^{T} = \frac{\sum\_{i \in \mathfrak{B}^T}I \Big(y\_i=\hat{y}*i^{T}\Big)}{\Big |\mathfrak{B}^T\Big |} - \frac{\sum*{i \in \mathfrak{B}^T}I \Big(y\_i=\hat{y}\_{i,\pi\_j}^{T}\Big)}{\Big |\mathfrak{B}^T\Big |}

среди них,

* y^i(T)=fT(xi) \hat{y}\_i^{(T)} = f^{T}(x\_i)y^​i(T)​=fT(xi​) Это прогнозирование типа перед заменой или исключением функции;
* $ \hat{y}*{i,\pi\_j}^{(T)} = f^{T}(x*(i, \ pi\_j)) $ - предсказание типа после замены или исключения признака;
* xi,πj=(xi,1,…,xi,j−1,xπj(i),j,xi,j+1,…,xi,p) x\_{i,\pi\_j} = (x\_{i,1}, \dots , x\_{i,j-1}, \quad x\_{\pi\_j(i),j}, \quad x\_{i,j+1}, \dots , x\_{i,p})xi,πj​​=(xi,1​,…,xi,j−1​,xπj​(i),j​,xi,j+1​,…,xi,p​).
* Обратите внимание, что еслиxj x\_jxj​ Не в деревеT TT В тоVIT(xj)=0 VI^{T}(x\_j) = 0VIT(xj​)=0。

Теперь формула для расчета важности функций при интеграции может быть дана:

* Формула без регуляризации:

VI(xj)=∑T=1NVIT(xj)N VI(x\_j) = \frac{\sum\_{T=1}^{N}VI^{T}(x\_j)}{N} VI(xj​)=N∑T=1N​VIT(xj​)​

* Формула после использования регуляризации стандартного отклонения:

zj=VI(xj)σ^N z\_j = \frac{VI(x\_j)}{\frac{\hat{\sigma}}{\sqrt{N}}} zj​=N ​σ^​VI(xj​)​

**Рассчитайте важность функций в реальных задачах**

Давайте рассмотрим набор данных. Содержимое этого набора данных представляет собой некоторую информацию об отелях, перечисленных на Booking.com и TripAdvisor.com. Характеристики набора данных - это средние оценки различных категорий (включая качество обслуживания, условия проживания, стоимость -эффективность и т. д.). Целевая переменная - общая оценка отеля на сайте. Сначала импортируйте соответствующую библиотеку.

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

**import numpy as np**

**import pandas as pd**

**import seaborn as sns**

**from matplotlib import pyplot as plt**

**from \_\_future\_\_ import division, print\_function**

* 1
* 2
* 3
* 4
* 5
* 6
* 7

%matplotlib inline

* 1
* 2

Импортируйте набор данных.

**hostel\_data = pd.read\_csv(**

**"https://labfile.oss.aliyuncs.com/courses/1283/hostel\_factors.csv")**

**features = {"f1": u"Staff",**

**"f2": u"Hostel booking",**

**"f3": u"Check-in and check-out",**

**"f4": u"Room condition",**

**"f5": u"Shared kitchen condition",**

**"f6": u"Shared space condition",**

**"f7": u"Extra services",**

**"f8": u"General conditions & conveniences",**

**"f9": u"Value for money",**

**"f10": u"Customer Co-creation"}**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 1
* 2

Используйте случайный лес для обучения модели.

**forest = RandomForestRegressor(n\_estimators=1000, max\_features=10,**

**random\_state=0)**

**forest.fit(hostel\_data.drop(['hostel', 'rating'], axis=1),**

**hostel\_data['rating'])**

**importances = forest.feature\_importances\_**

**indices = np.argsort(importances)[::-1]**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2

Отметьте важность функций в модели случайного леса.

**num\_to\_plot = 10**

**feature\_indices = [ind+1 for ind in indices[:num\_to\_plot]]**

* 1
* 2
* 3
* 1
* 2

Распечатайте рейтинг важности функций (ранжирование от высокой к низкой важности).

**print("Feature ranking:")**

**for f in range(num\_to\_plot):**

**print("%d. %s %f " % (f + 1,**

**features["f"+str(feature\_indices[f])],**

**importances[indices[f]]))**

**plt.figure(figsize=(15, 5))**

**plt.title(u"Feature Importance")**

**bars = plt.bar(range(num\_to\_plot),**

**importances[indices[:num\_to\_plot]],**

**color=([str(i/float(num\_to\_plot+1))**

**for i in range(num\_to\_plot)]),**

**align="center")**

**ticks = plt.xticks(range(num\_to\_plot),**

**feature\_indices)**

**plt.xlim([-1, num\_to\_plot])**

**plt.legend(bars, [u''.join(features["f"+str(i)])**

**for i in feature\_indices])**

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 1
* 2

Приведенный выше рисунок показывает, что потребители часто больше озабочены качеством обслуживающего персонала (Персонал) и соотношением цены и качества (Соотношение цена / качество), эти два фактора имеют наибольшее влияние на итоговую оценку. Однако разница между этими двумя функциями и другими функциями не очень велика, поэтому исключение какой-либо функции приведет к снижению точности модели.

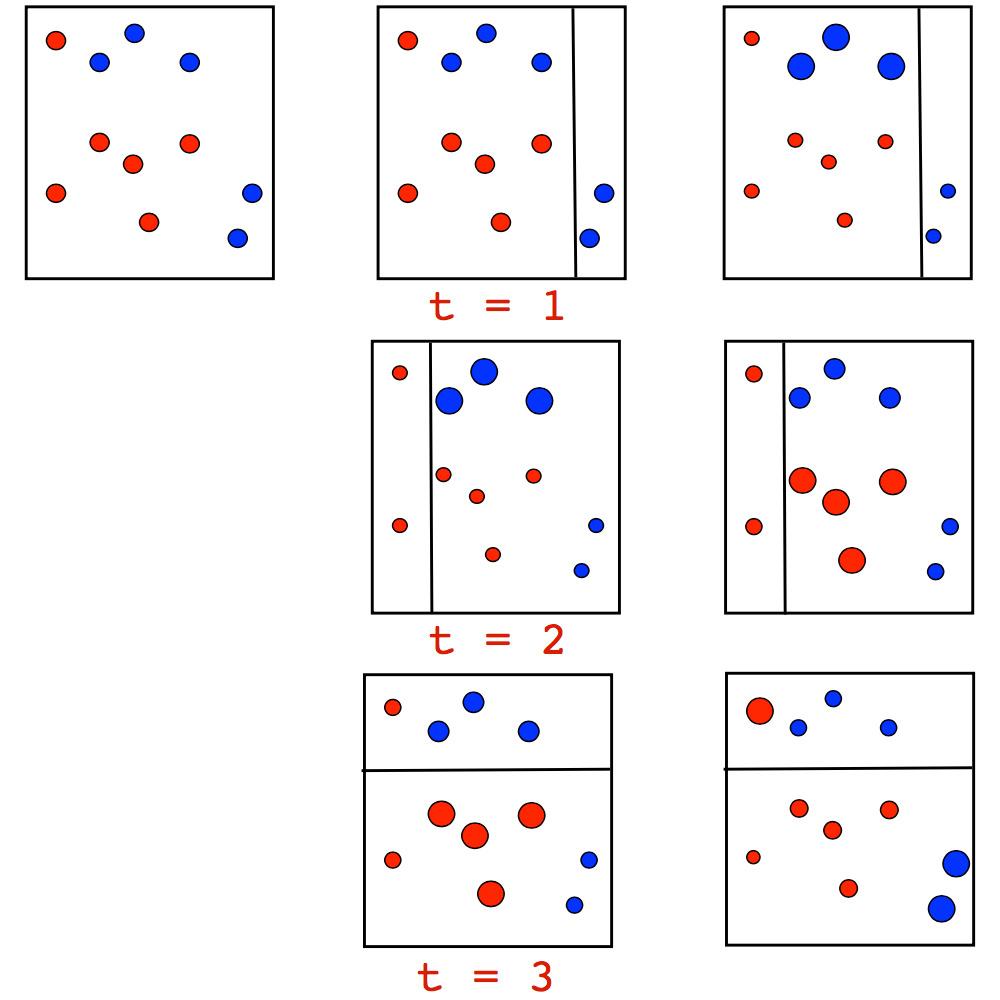
**Введение в повышение градиента**

В машинном обучении некоторые алгоритмы часто называют слабыми моделями. «Слабые модели», упомянутые здесь, относятся к простым базовым моделям, таким как деревья решений, а также относятся к моделям с относительно низкой точностью, то есть модели, которая немного лучше, чем случайная модель.

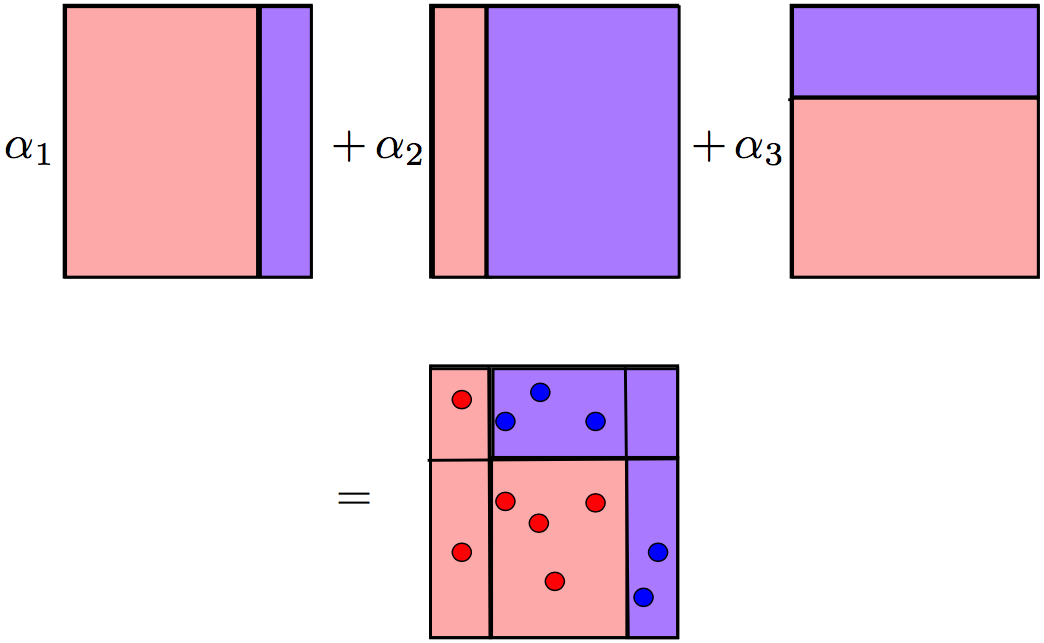
Давайте сначала подумаем об одном[*проблема*](https://www.cis.upenn.edu/~mkearns/papers/boostnote.pdf): Можно ли каким-либо образом получить сильную модель из большого количества относительно слабых и простых моделей?

Как вы думаете, проблема[*отвечать*](https://www.cs.princeton.edu/~schapire/papers/strengthofweak.pdf) Очевидно да. Один из них - метод интеграции.Алгоритмов интегрированного обучения много.Начнем с самого простого: AdaBoost.

AdaBoost использует жадный метод обучения. При обучении алгоритма к каждой выборке данных добавляется значение веса.Перед тем, как каждый базовый классификатор начинает классифицировать, значение веса выборки будет скорректировано в соответствии с ошибкой классификации предыдущего классификатора. Затем объедините все базовые классификаторы линейной комбинацией, чтобы получить сильный классификатор. Весь процесс можно описать, как показано на рисунке ниже.



Как видно из приведенного выше рисунка, после создания первого дерева классификации (t = 1) три синие точки данных будут неправильно классифицированы. Следовательно, веса этих трех точек данных будут увеличены при создании второго дерева и так далее. Наконец, мы объединяем три базовых классификатора, чтобы получить идеальный классификатор. Как показано ниже:



Если весь процесс описывается математическими формулами, то весь процесс выглядит следующим образом.

* Сначала дайте каждому образцу начальное значение веса:wi(0)=1l,i=1,…,l w\_i^{(0)} = \frac{1}{l}, i = 1, \dots, lwi(0)​=l1​,i=1,…,l
* Пройти всеt=1,…,Tt = 1, \dots, Tt=1,…,T
  + Классификатор учебной базыbt b\_tbt​, И сделатьϵt\epsilon\_tϵt​ Равно ошибке классификации.
  + αt=12ln1−ϵtϵt\alpha\_t = \frac{1}{2}ln\frac{1 - \epsilon\_t}{\epsilon\_t}αt​=21​lnϵt​1−ϵt​​
  + Обновите веса образца:wi(t)=wi(t−1)e−αtyibt(xi),i=1,…,l w\_i^{(t)} = w\_i^{(t-1)} e^{-\alpha\_t y\_i b\_t(x\_i)}, i = 1, \dots, lwi(t)​=wi(t−1)​e−αt​yi​bt​(xi​),i=1,…,l
  + Нормализованный вес образца:w0(t)=∑j=1kwj(t),wi(t)=wi(t)w0(t),i=1,…,l w\_0^{(t)} = \sum\_{j = 1}^k w\_j^{(t)}, w\_i^{(t)} = \frac{w\_i^{(t)}}{w\_0^{(t)}}, i = 1, \dots, lw0(t)​=∑j=1k​wj(t)​,wi(t)​=w0(t)​wi(t)​​,i=1,…,l
* Вернуться к∑tTαtbt\sum\_t^{T}\alpha\_tb\_t∑tT​αt​bt​

При итерации вы можете увидеть увеличение веса неправильно классифицированных точек, особенно на границах между классами.[*Здесь*](https://www.youtube.com/watch?v=k4G2VCuOMMg) Есть более подробный пример об AdaBoost.

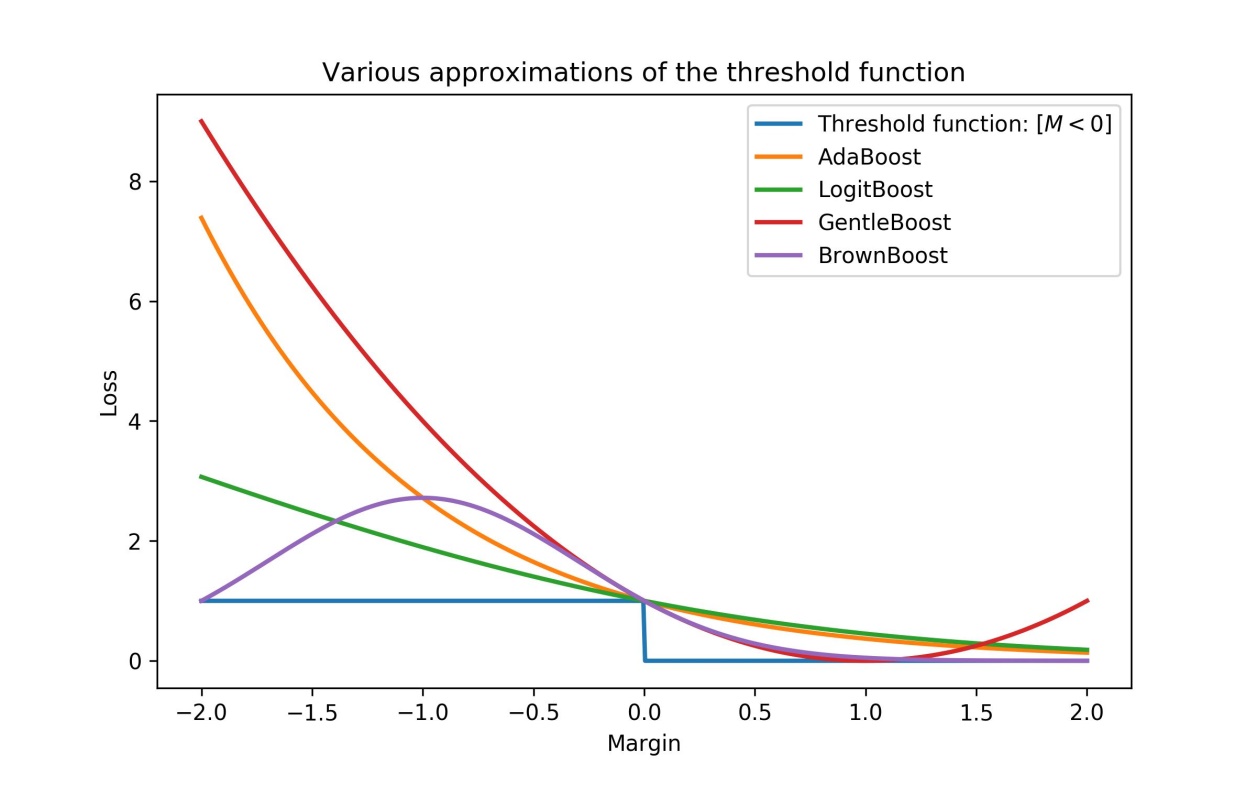
Эффект классификации AdaBoost действительно хорош, но почему этот алгоритм так успешен?[*Отсутствие объяснения*](https://www.cs.princeton.edu/courses/archive/spring07/cos424/papers/boosting-survey.pdf). Это заставит некоторых людей сомневаться в том, что AdaBoost просто переоснащен.

Проблема чрезмерной подгонки действительно существует, особенно когда данные имеют очень странные выбросы, явление чрезмерной подгонки более серьезно. Следовательно, в подобных проблемах AdaBoost работает нестабильно. Для решения этих проблем в 1999 году Джером Фридман (Jerome Friedman) предложил обобщенную версию AdaBoost: Gradient Boosting (Machine), также известную как GBM.

CART, бутстрап и многие другие алгоритмы возникли на кафедре статистики Стэнфордского университета. Эти алгоритмы очень практичны, но в последние годы есть также некоторые исследовательские работы, которые не получили широкого распространения. Например,[*glinternet*](https://arxiv.org/abs/1308.2719)。

Видео информации о Фридмане сейчас не так много. Однако есть очень интересный[*Опрос*](https://www.youtube.com/watch?v=8hupHmBVvb0), О том, как предлагать CART и как CART решает статистические задачи. У Хасти тоже есть отличный[*Лекция*](https://www.youtube.com/watch?v=zBk3PK3g-Fc)。

Для алгоритма AdaBoost, хотя мы можем улучшить алгоритм ансамбля, добавив слабые алгоритмы и постепенно улучшая веса неверно классифицированных данных. Однако этого недостаточно.Например, GBM не только основывается на повторно взвешенных точках данных, но также улучшает его приближение к градиенту общей целевой функции. Эта концепция во многом открыла исследовательские идеи интегрированных алгоритмов.



**История GBM**

С момента появления GBM прошло более 10 лет, прежде чем он стал незаменимой частью инструментария для анализа данных. Для применения к различным статистическим задачам GBM имеет множество расширенных версий, например: GLMboost и GAMboost используются для улучшения существующей модели GAM, CoxBoost используется для кривых выживаемости, а RankBoost и LambdaMART используются для задач ранжирования.

Многие реализации GBM также появляются на разных платформах под разными именами. Например: стохастический GBM, GBDT (дерево решений с градиентным усилением), GBRT (дерево регрессии с градиентным усилением), MART (дерево множественной аддитивной регрессии) и так далее. Кроме того, поскольку сообщество ML выполняет свою работу, нам трудно узнать, насколько широко используется GBM, сколько версий и так далее.

В то же время GBM также широко использовался для поиска и сортировки. С добавлением GBM поисковое ранжирование рассматривается как проблема оптимизации функции потерь. Изначально AltaVista была одной из первых компаний, внедривших улучшения ранжирования. Но вскоре эти идеи распространились на Yahoo, Яндекс, Bing и другие сайты. С тех пор GBM стал одним из основных алгоритмов, используемых не только в исследованиях, но и в промышленном производстве.



Многие соревнования по машинному обучению, особенно Kaggle, сыграли важную роль в продвижении применения популярных алгоритмов. Поскольку Kaggle предоставляет общую платформу, это позволяет специалистам по обработке данных конкурировать с большим количеством участников со всего мира по различным вопросам науки о данных. Люди могут тестировать новые алгоритмы на реальных данных на Kaggle, давая шанс многим алгоритмам «засветиться».

С 2011 года в интервью на чемпионате Kaggle многие победители упоминали, что они использовали алгоритмы повышения градиента. Это также делает популярным повышение градиента. Особенно после появления библиотеки XGBoost. Хотя XGBoost не является новым и уникальным алгоритмом, это эффективная реализация классического GBM.

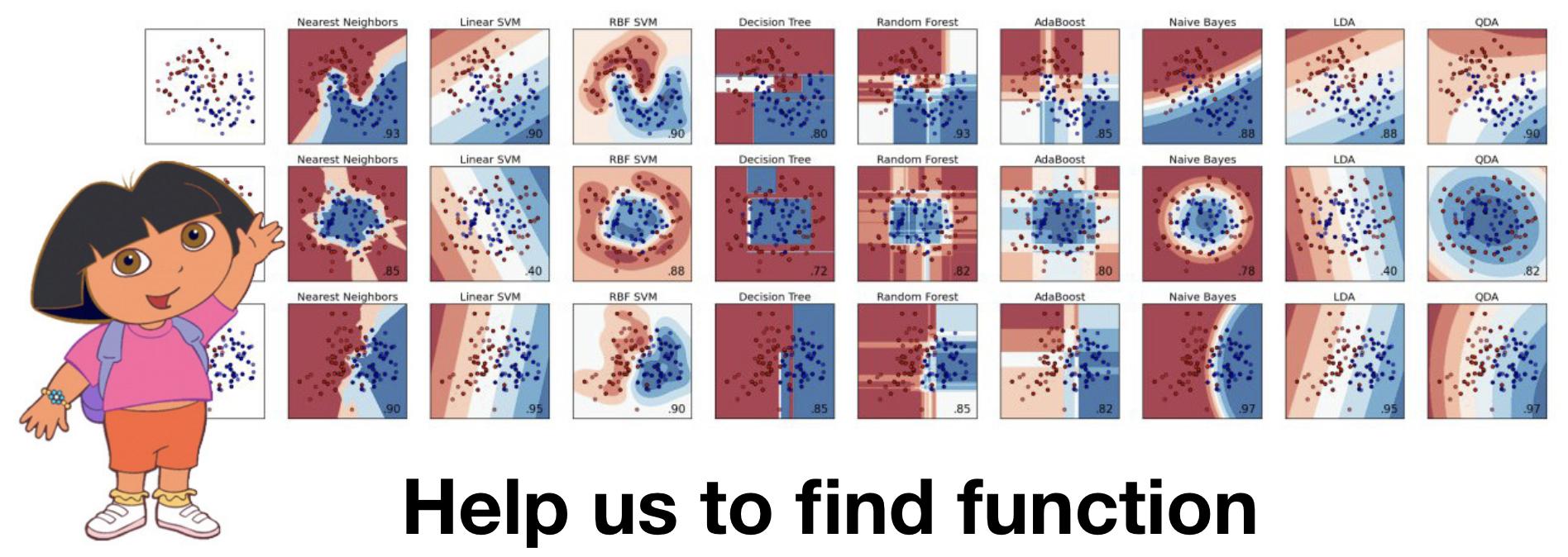
Опыт GBM такой же, как и у многих алгоритмов машинного обучения: прошло много лет от предложения до успешного практического применения и широкомасштабного использования.

**Алгоритм GBM**

Давайте сначала рассмотрим классическую задачу контролируемого обучения. Предположим, что существует набор данных $ \ left {(x\_i, y\_i) \ right} \_ {i = 1, \ ldots, n} $,x x x Представляет значение характеристики,yyy Указывает целевое значение. Построение модели эквивалентно построению функцииf(x)f(x)f(x) ,использовать f(x)f(x)f(x) Чтобы приблизиться к целевому значениюy yy. Определить функцию потерьL(y,f(x))L(y,f(x))L(y,f(x)) Для представления прогнозируемого значенияf(x)f(x)f(x) И истинная ценностьyyy расстояние между. Минимизируя функцию потерь, мы можем получить:

y≈f^(x) y \approx \hat{f}(x)y≈f^​(x)

f^(x)=arg⁡min⁡f(x) L(y,f(x)) \hat{f}(x) = \underset{f(x)}{\arg\min} \ L(y,f(x)) f^​(x)=f(x)argmin​ L(y,f(x))



Оставь это здесьf(x)f(x)f(x) Какая конкретная модель или конкретное выражение, предполагается только функция потерьL(y,f(x))L(y,f(x))L(y,f(x)) Дифференцируема. Если функция потерь принимает среднее значение для всей выборки, минимизированная функция потерь может быть выражена как:

f^(x)=arg⁡min⁡f(x) Ex,y[L(y,f(x))] \hat{f}(x) = \underset{f(x)}{\arg\min} \ \mathbb {E} \_{x,y}[L(y,f(x))] f^​(x)=f(x)argmin​ Ex,y​[L(y,f(x))]

В целом,f(x)f(x)f(x) Вариантов много, а значит искать лучшееf(x)f(x)f(x) Будет очень сложно. Так что будь правf(x)f(x)f(x) Ограничьте его функциональным пространством. Предположим, что параметры модели равныθ\theta θ, Определениеθ∈Rd\theta \in \mathbb{R}^d θ∈Rd, Модель может быть описана какf(x,θ) f(x, \theta)f(x,θ). Итак, теперь ищем лучшееf(x)f(x)f(x) Эквивалентно поиску набора параметровθ\theta θ , Делаем функцию потерьL(y,f(x))L(y,f(x))L(y,f(x)) Достигните минимума. Формула описывается следующим образом:

f^(x)=f(x,θ^) \hat{f}(x) = f(x, \hat{\theta})f^​(x)=f(x,θ^)

θ^=arg⁡min⁡θ Ex,y[L(y,f(x,θ))] \hat{\theta} = \underset{\theta}{\arg\min} \ \mathbb {E} \_{x,y}[L(y,f(x,\theta))] θ^=θargmin​ Ex,y​[L(y,f(x,θ))]

Из-заf(x)f(x)f(x) Относительно сложно. Как правило, аналитическое решение не существует или не может быть получено напрямую. Поэтому для аппроксимации параметров используются итерационные методы. Определим эмпирическую функцию потерь какLθ(θ^) L\_{\theta}(\hat{\theta}) Lθ​(θ^) , Его формула расчета выглядит следующим образом:

θ^=∑i=1Mθi^ \hat{\theta} = \sum\_{i = 1}^M \hat{\theta\_i} θ^=i=1