Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:

«Машинное обучение»

По лабораторной работе №10

«Градиентный бустинг»

Выполнил: Сенькович Дмитрий Сергеевич

магистрант кафедры информатики

группы №858642

Проверил: Стержанов Максим Валерьевич

доцент, кандидат технических наук

Минск 2019

**Оглавление**

[1 Введение 2](#_Toc25886089)

[2 Теория 3](#_Toc25886090)

[2.1 Деревья 3](#_Toc25886091)

[2.2 Построение дерева 3](#_Toc25886092)

[2.3 Методы борьбы с переобучением одного дерева 7](#_Toc25886093)

[2.4 Понятие композиции деревьев 7](#_Toc25886094)

[2.5 Методы борьбы с переобучением в композициях деревьев 7](#_Toc25886095)

[2.6 Случайные леса 9](#_Toc25886096)

[2.7 Проблемы случайных лесов 11](#_Toc25886097)

[2.8 Бустинг 11](#_Toc25886098)

[2.9 Идея градиентного бустинга 12](#_Toc25886099)

[2.10 Методы борьбы с переобучением в градиентном бустинге 14](#_Toc25886100)

[3 Исследование различных алгоритмов с деревьями 16](#_Toc25886101)

[3.1 Чтение данных 16](#_Toc25886102)

[3.2 Разделение выборки на обучающую и валидационную 16](#_Toc25886103)

[3.3 Реализация бустинга с константным шагом обучения 16](#_Toc25886104)

[3.4 Реализация бустинга с изменяющимся шагом обучения 17](#_Toc25886105)

[3.5 Исследование влияния числа итераций и глубины дерева на переобучение градиентного спуска 17](#_Toc25886106)

[3.6 Сравнение с линейной регрессии 18](#_Toc25886107)

# 1 Введение

Линейная регрессия становится довольно сложной в вычислительном аспекте применения алгоритма, когда дело касается сложным нелинейных зависимостей. В целом, эта модель не очень хорошо отражает процесс принятия решения человеком. Альтернатива этой модели - решающие деревья. Но деревья как самостоятельная модель обучения машинного обучения очень легко переобучается, поэтому не может быть использована в сама по себе. В лабораторной работе рассмотрим градиентный бустинг - композицию решающих деревьев, которая дает хорошие результаты.

# 2 Теория

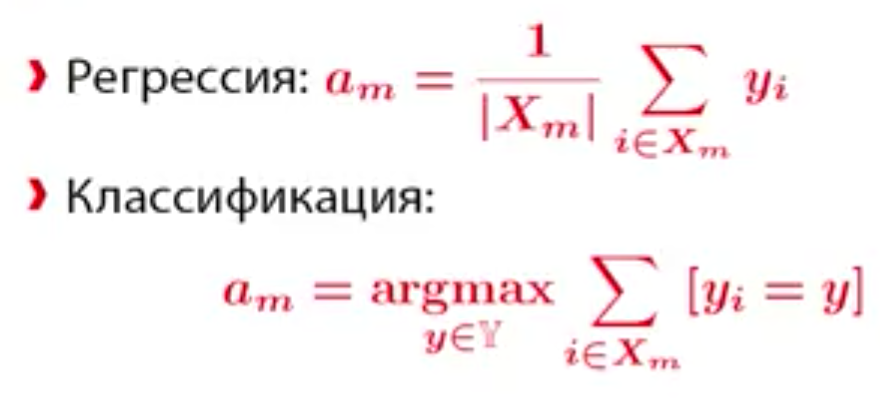
### 2.1 Деревья

Обычно под решающим деревом понимают бинарное деревом, в каждом вершине которого (кроме листов) записано условие, а в каждом листе - прогноз (будь то классификация или решение задачи регрессии). Таким образом, каждый экземпляр выборки начинает свое движение в корне и движется по дереву вниз к листам для получения прогноза. Условия обычно простые, например, сравнение значения некоторой фичи. Если значение меньше - идем влево, иначе - вправо. Прогнозом является либо число (в случае регрессии), либо класс/вероятность классов (в случае классификации). Как отмечалось, дерево очень легко переобучить: довести его ветвление до состояния, когда в каждом листе будет один объект выборки.

### 2.2 Построение дерева

Лучшее дерево - дерево с нулевой ошибкой минимальной длины. Но задача построения такого дерева NP-полная. Поэтому дерево строится жадно - путем последовательного разбиения вершины, пока не достигнет некоторого критерия остановы. Поиск наилучшего разбиения можно делать следующим образом: перебирать все фич и все значения этой фичи и смотреть, как хорошо получившееся разбиение будет разбивать нашу выборку - посредством вычисления некоторой функции стоимости (пытаемся ее минимизировать). Критерии остановы могут быть самые разнообразные: от более вычислительно сложных, как “в каждой вершине один объект”, так и до более простых - ограничение на глубину дерево. Чаще всего используется последний критерий, так как он простой.

Далее: как выбирать ответ в листе:



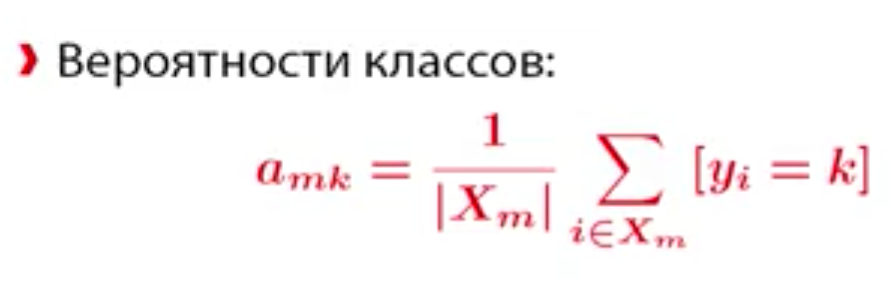


Рисунок 2.2.1. Принятие решение в листе дерева

Как уже упоминалось, выборки разбиваются в соответствии с некоторым критерием. Формула критерия представлена ниже:

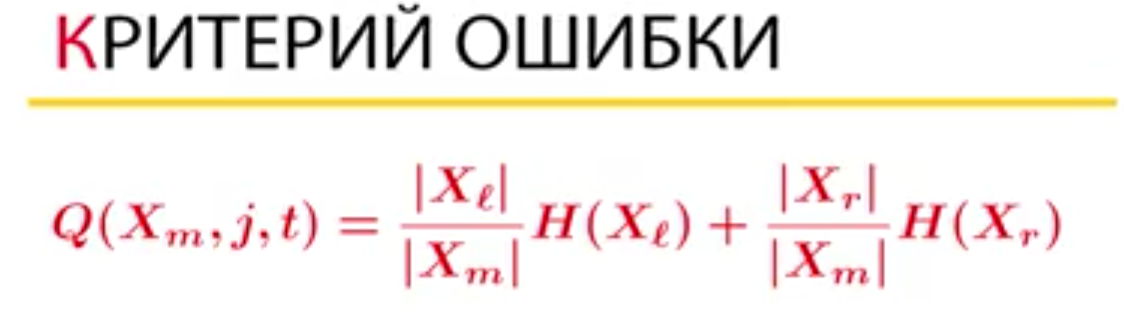


Рисунок 2.2.2. Критерий ошибки при разбиении дерева

Функция H измеряет качество подмножества, т.е. насколько сильный разброс ответов имеет место в получившейся подвыборке разбиения. H - критерий информативности. Чем меньше разброс, тем меньше значение. H зависит от задачи. Например, так выглядит критерий информативности в случае регрессии (дисперсия выборки):

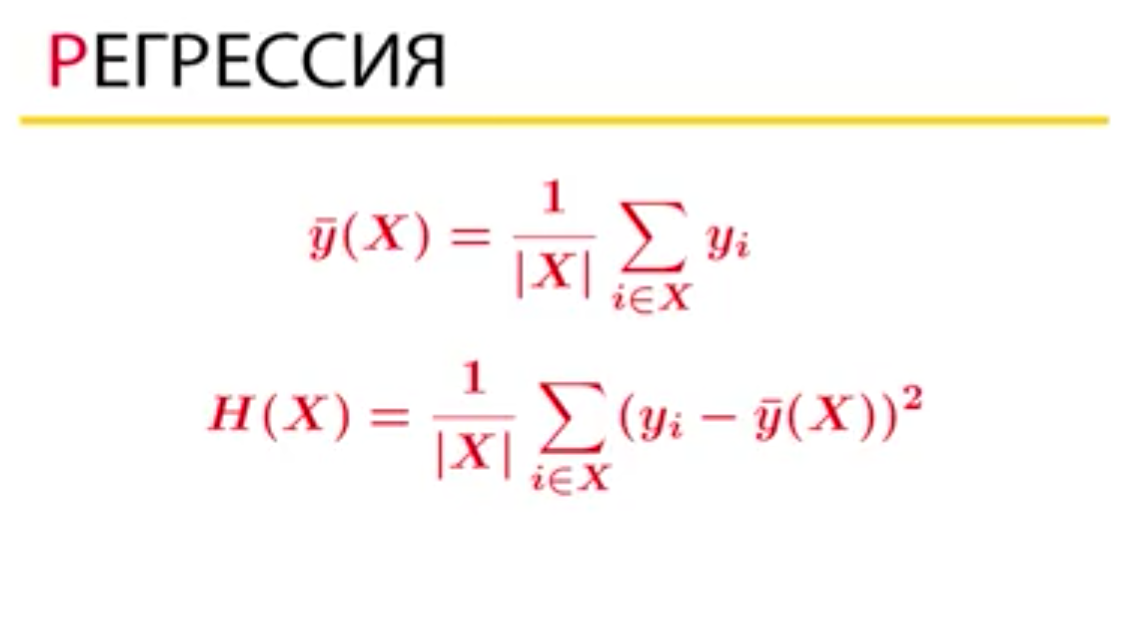


Рисунок 2.2.3. Критерий информативности в случае задачи регрессии

В случае классификации, для начала, введем вспомогательную функцию - долю объектов класса k:

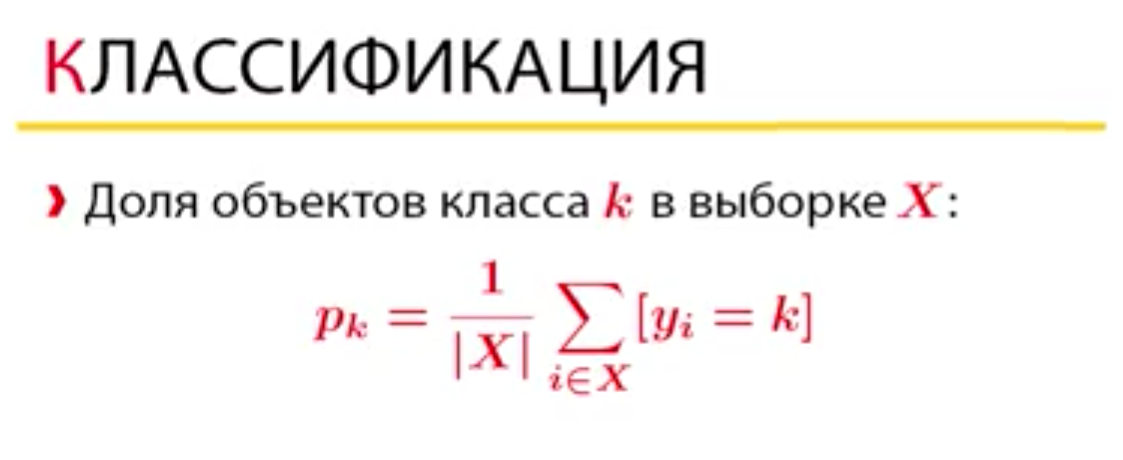


Рисунок 2.2.4. Функция доли объектов класса k

С помощью этой функции вычисляется сам критерий информативности, например, критерий Джини. Очевидно, что минимум (0) достигается только при идеальной разбиении, то есть когда в выборке объекты только одного класса:

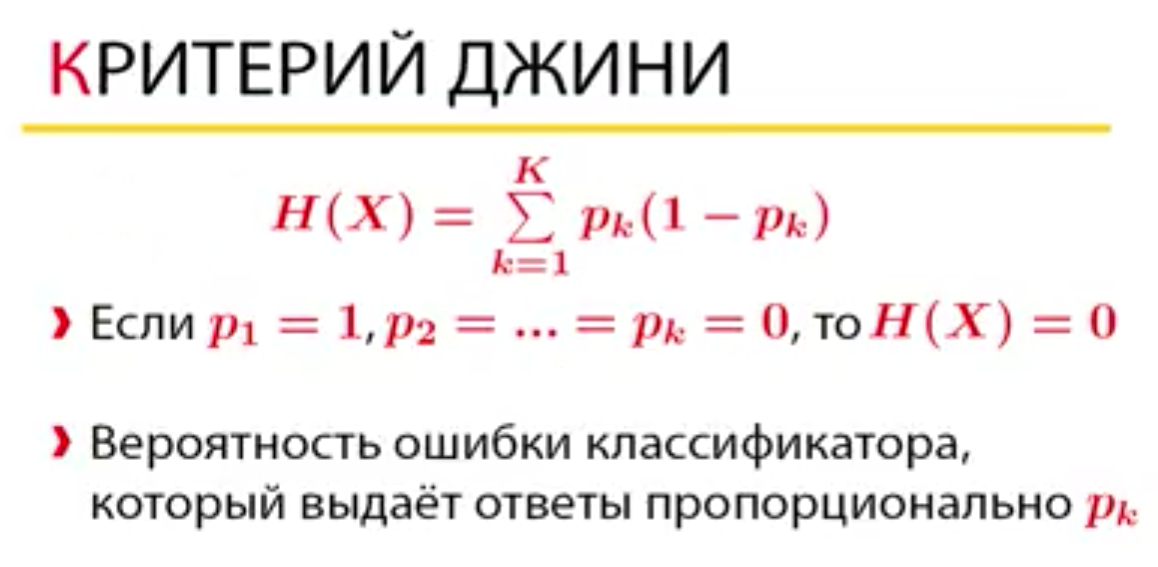


Рисунок 2.2.5. Один из видов функции критерия информативности - критерий Джини

Еще один вид критерия информативности: энтропийный:

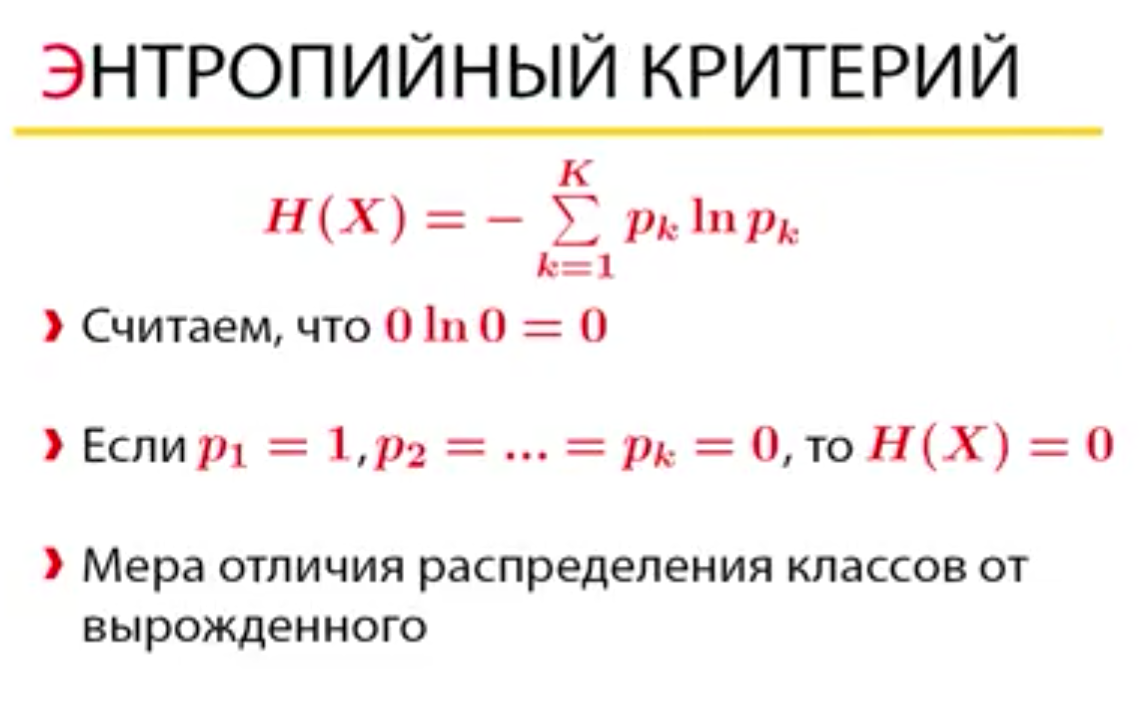


Рисунок 2.2.6. Один из видов функции критерия информативности - энтропийный критерий

### 2.3 Методы борьбы с переобучением одного дерева

Критерий остановы - когда нужно остановиться при разбиении вершины. Является способом борьбы с переобучением. Один из критериев: в каждом листе объекты одного класса. Плох тем, что при сложных зависимостях, что и имеет место быть в реальности, дерево строится максимально плохое: по одному элементу выборки в листе. Другой критерий получше: количество элементов в листе не больше некоторого количество, обычно берут 5. Еще один очень грубый, но, отлично работающий в композициях, метод - ограничение на глубину.

Еще один подход к борьбе с переобучением: стрижка деревьев. Строится максимально переобученное дерево, а листья удаляются по некоторому критерию, например, пока улучшается ошибка на валидации. Данный метод работает лучше критерия остановы, когда речь идет о переобучении. Однако данный критерий имеет смысл только при использовании одного дерева (что никто не делает), так как является трудоемкой операцией. В композициях используются простые критерии остановы.

### 2.4 Понятие композиции деревьев

Деревья сами по себе чересчур легко переобучаются и сильно меняются даже при небольшой изменении выборки. Поэтому деревья используются в композициях. Обучаются много деревьев и берется средний результат по всем ответам:



Рисунок 2.4.1. Усреднение ответов деревьев при использовании композиции

Сами деревья в композиции называются базовыми алгоритмами.

### 2.5 Методы борьбы с переобучением в композициях деревьев

Базовые алгоритмы должны быть рандомизированы, т.е. не обучаться на одной выборке, иначе они все будут одинаковыми и смысла в их усреднении не будет. Ответ: обучать на подвыборках. Один из методов - бутстрап. Выбираем случайным образом из нашей выборки размера m выборку также размера m, в которой элементы могут повторяться. В такой выборке будет в среднем около 0.632\*m различных элементов. Можно и генерировать случайное подмножество, но в таком случае придется настраивать новый гиперпараметр - размер данного подмножества.

Объясним, почему усреднение работает. Для начала, покажем, из чего состоит ошибка:

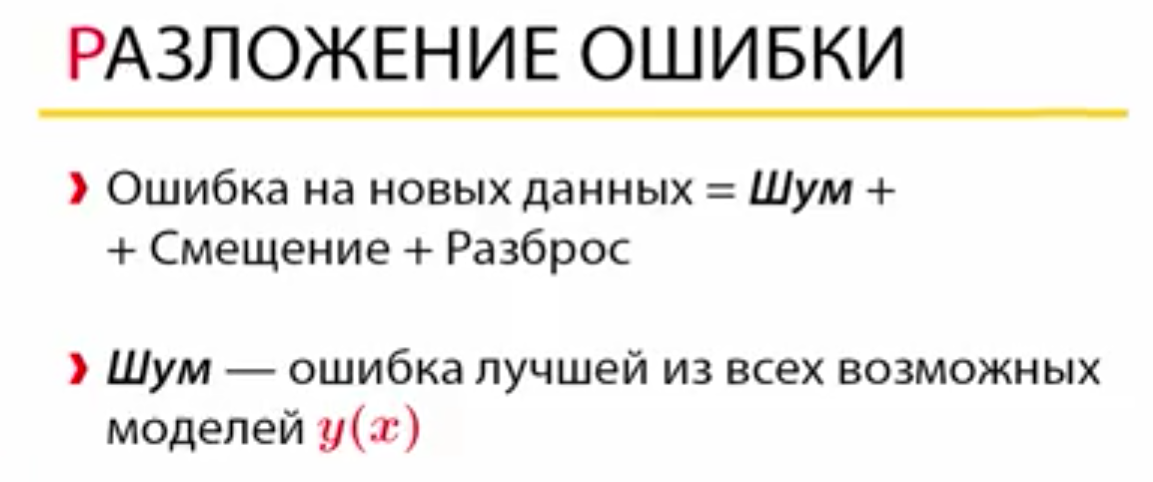


Рисунок 2.5.1. Разложение ошибки модели на составляющие

Шум невозможно улучшить (обучением модели как-либо), потому что он возникает из-за шума в исходных данных. Смещение - отклонение средних ответов от ответов лучшей модели. Разброс - дисперсия ответов наших моделей. Чем выше разброс, тем больше алгоритм зависит от обучающей выборки, что наблюдается в деревьях. Различие линейной модели и дерева: у линейной модели смещение может быть большим, но, как правило, низкий разброс. У дерева - наоборот. Усреднение не меняет смещение, но меняет разброс:

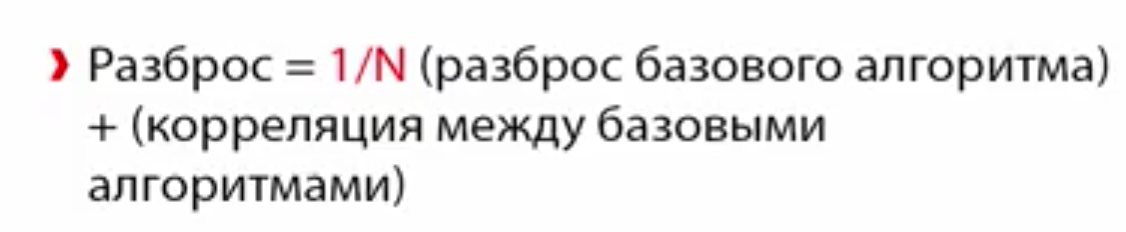


Рисунок 2.5.2. Составляющие смещения при усреднении

Поэтому, если базовые алгоритмы независимы, т.е. прогнозы не коррелируют между собой, то разброс уменьшается в N раз (N - кол-во базовых алгоритмов). Как следствие, объединяя много независимых деревьев можно получить модель как с низким смещением, так и с низким разбросом.

Но очень сложно сделать алгоритмы независимыми, так как они обучаются на одной выборке или подвыборках одной выборки данных. Но можно попытаться уменьшить корреляцию.

Методы снижения корреляции:

* Бэггинг - обучение на случайной подвыборке. Чем меньше подвыборка - тем менее коррелированы решения. Но при слишком маленькой выборке переобучения не избежать
* Метод случайных подпространств - случайное подмножество столбцов

Данные методы можно объединять, выбирая случайное подмножество строк и столбцов.

### 2.6 Случайные леса

Один из лучших способов объединения деревьев в композиции - случайные леса. Методов снижения корреляции выше недостаточно. Нужно рандомизировать сам процесс построения дерева. Это можно сделать следующим образом: при разбиении дерева выбирать лучший параметр разбиения не из всех параметров, а из некоторой случайной подвыборки всех параметров выборки. Рекомендация для выбора этого параметра: для регрессии d/3, а для классификации sqrt(d), где d - размер пространства фич.

Суммируя, алгоритм построения случайного леса выглядит так:

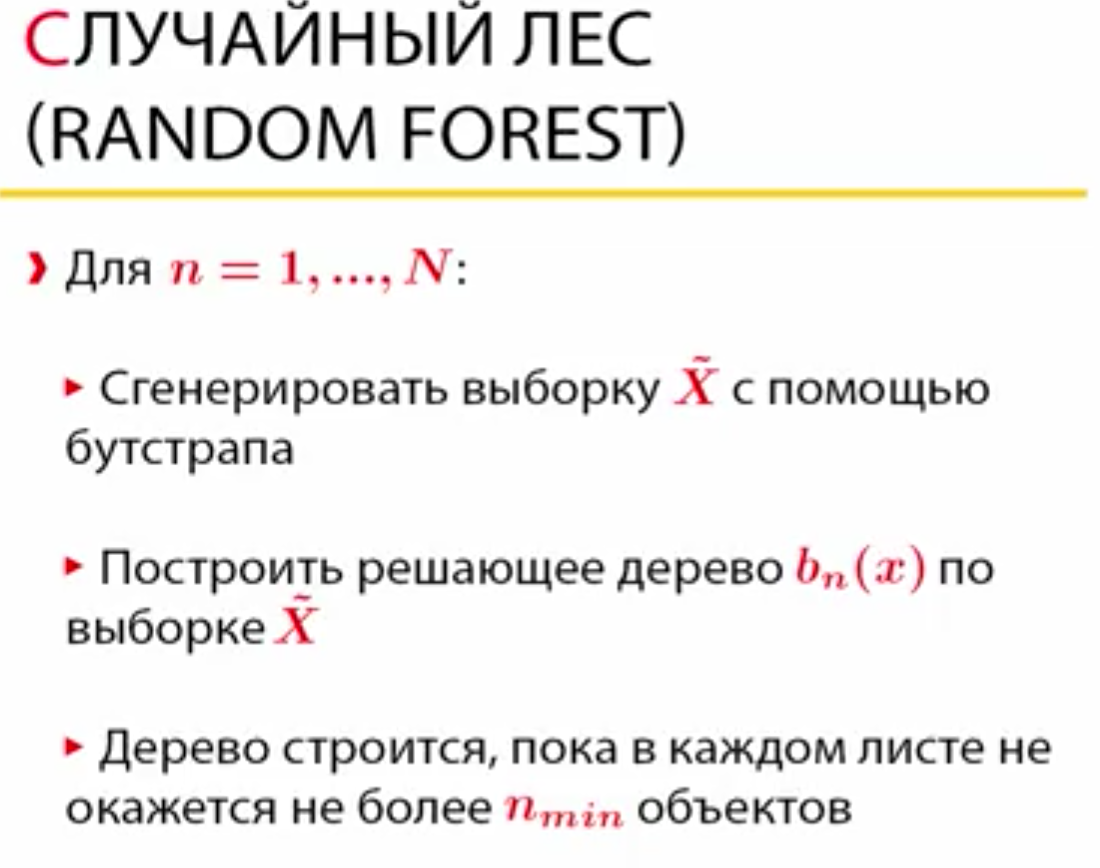


Рисунок 2.6.1. Алгоритм построения случайного леса

Примечание: часто деревья строят до конца. Не забываем использовать только подвыборку фич для выбора лучшей фичи для разбиения.

Особенность случайных лесов: они не переобучаются при росте количества базовых алгоритмов. Не подверженность случайных лесов с количеством базовых алгоритмов наглядно представлена на следующем графике:



Рисунок 2.6.2. Зависимости ошибки при увеличении количества базовых алгоритмов

Заметим, что ошибка падает и остается на некотором конечном уровне практически без изменения.

Случайные леса имеют несколько полезных особенностей. Так, случайные леса идеально распараллеливаются, т.к. построение деревьев независимы. Также, так как каждое дерево обучается бутстрапом, т.е. не на всей выборке, не нужна дополнительная валидационная выборка, так как для каждого xi можно сравнивать прогнозы по тем деревьям, которые не обучались на xi. Это позволяет оценивать ошибку по следующей формуле:

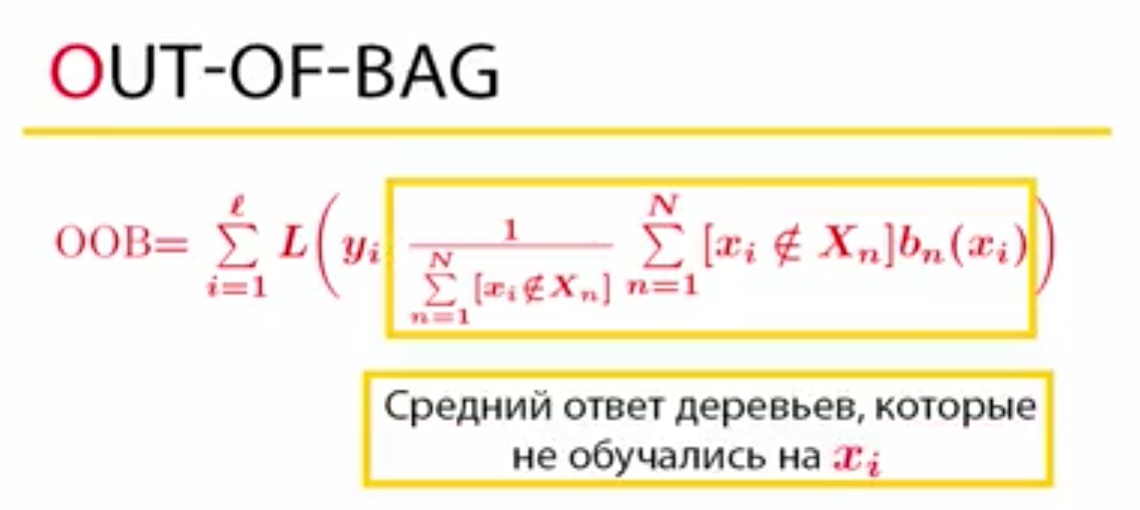


Рисунок 2.6.3. Метод оценки ошибки дерева

Здесь в качестве ошибки для каждого вектора выборки берется значение ошибки между оригинальной меткой и средним прогнозом на деревьях, которые не обучались на данном векторе. После ошибки суммируются.

### 2.7 Проблемы случайных лесов

Проблемы случайных лесов:

* Нельзя строить неглубокие леса, иначе смещение будет высоким, с этим ничего сделать уже нельзя
* Построение каждого нового дерева не зависит от предыдущего, т.е. является ненаправленным
* Нужно много деревьев для сложных задач

### 2.8 Бустинг

Бустинг позволяет решить эти проблемы. В нем:

* Базовые алгоритмы обучаются последовательно
* Каждый следующий алгоритм исправляет ошибки предыдущего

Ввиду этого, достаточно простых базовых алгоритмов, например, неглубоких деревьев.

Пример устройства бустинга: пусть каждый алгоритм уменьшает среднеквадратическую ошибку. Тогда можно представить ответ композиции как сумму ответов базовых алгоритмов, а каждый следующий алгоритм настраивать на ошибку уже построенной композиции. Бустинг является противоположностью случайному лесу: он строит направленную композицию.

### 2.9 Идея градиентного бустинга

Градиентный бустинг - один из лучших методов направленного построения композиции. Композиция - то же, что и в бустинге. Будем считать, что есть некоторая функция потерь L(y, z), которую мы пытаемся минимизировать. z - прогноз алгоритма. Функция ошибки может быть, например, mse. Сначала строим первый базовый самый простой алгоритм. Принцип построения следующего базового решения:

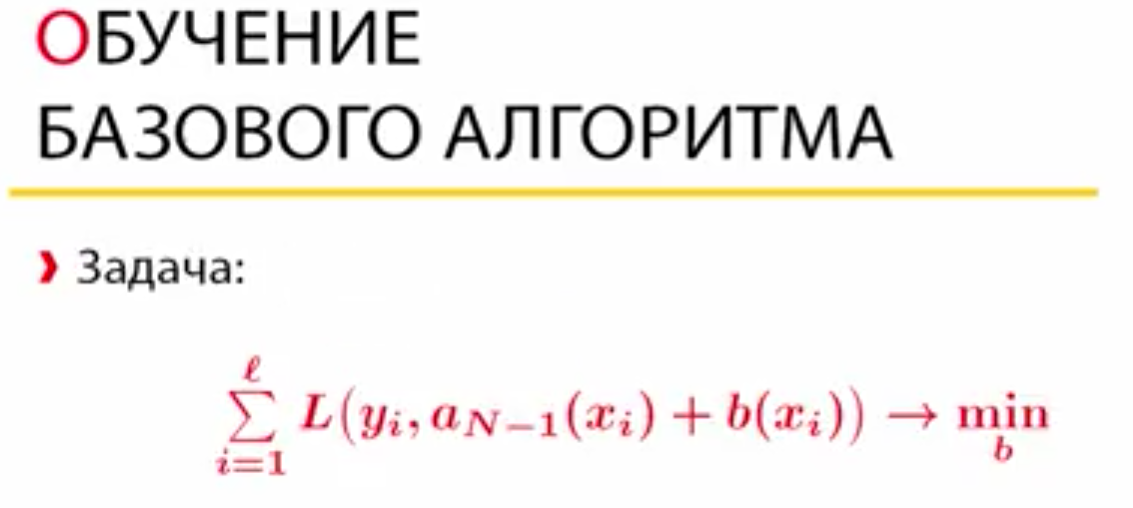


Рисунок 2.9.1. Задача построения следующего алгоритма

По сути, задача - построение такого решение, которое минимизирует уже существующую функцию потерь (в композиции с уже построенной композицией).

Переформулируя, задача выглядит так (какие значения новый алгоритм должен принимать на обучающей выборке для уменьшения ошибки на обучающей выборке):

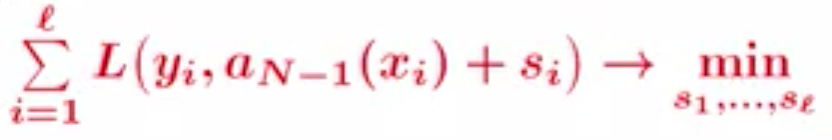


Рисунок 2.9.2. Переформулированная задача

Нужно найти сдвиги si. Вектор, сильнее всего изменяющий функцию - антиградиент, потому что он направлен в сторону наискорейшего убывания функции. Ниже представлена формула такого оптимального сдвига:

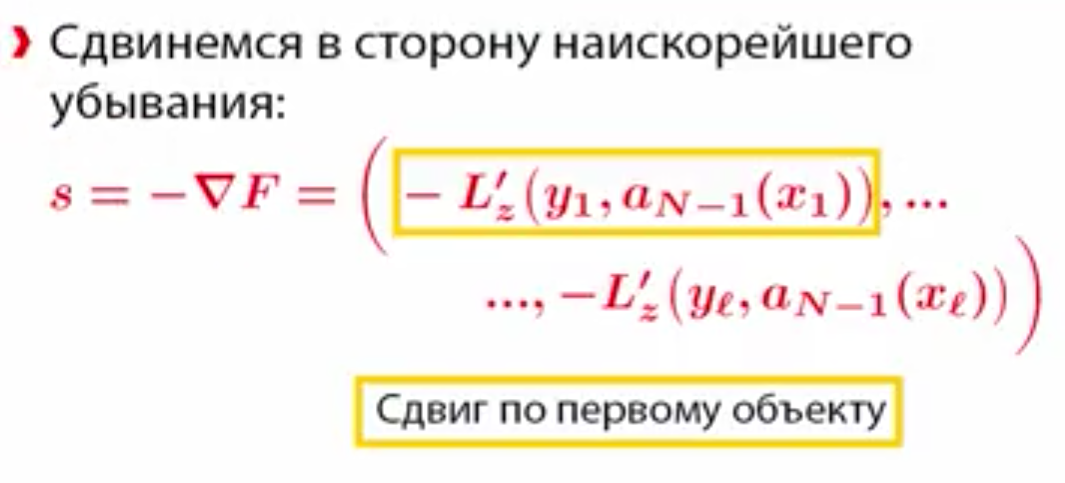


Рисунок 2.9.3. Формула оптимального сдвига для задачи построение очередного базового алгоритма в градиентном бустинге - антиградиент.

Здесь берутся частные производные функции ошибки по второму аргументу - прогнозу.

Резюмируя, теперь мы знаем, какие значения новый алгоритм должен принимать на обучающей выборке - задача машинного обучения. Таким образом, новый алгоритм композиции в градиентном спуске будет строится по следующему правилу, как обычная задача машинного обучения:

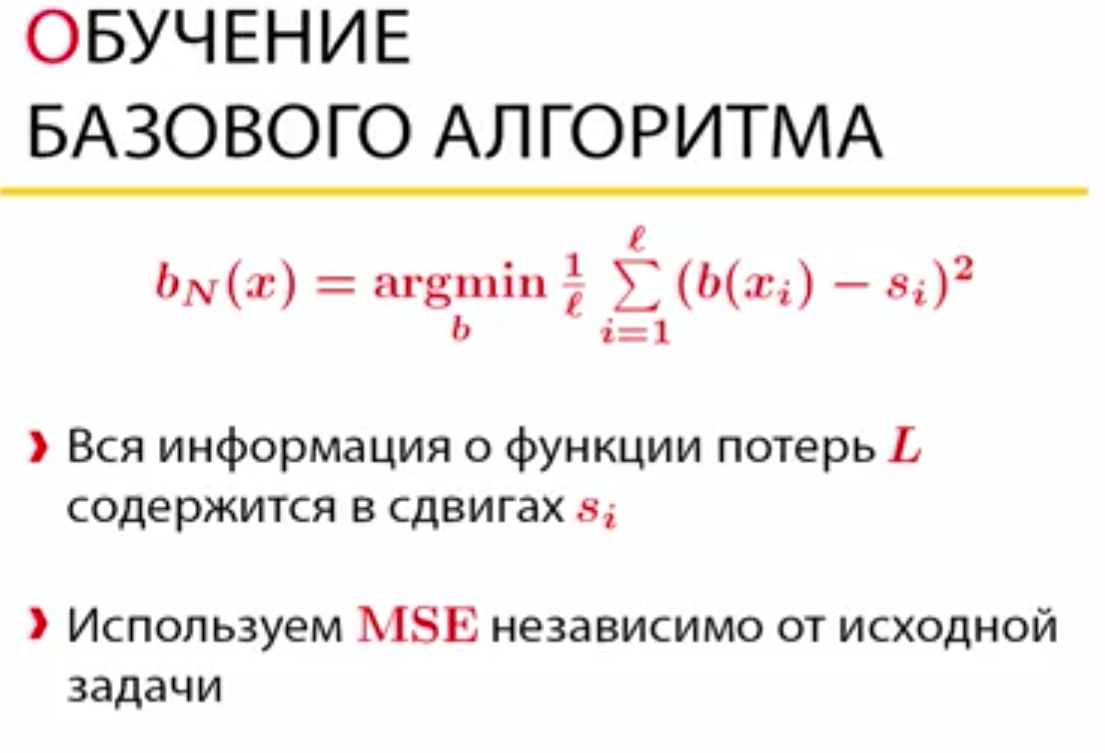


Рисунок 2.9.4. Правило построения нового решения

### 2.10 Методы борьбы с переобучением в градиентном бустинге

Градиентный бустинг, в отличие от случайного леса, может переобучаться. Это случается из-за того, что базовые алгоритмы очень слабые, приближение плохое и вместо градиентного спуска получаем случайное блуждание (вектор антиградиента будет показывать не в сторону уменьшения ошибки, а в случайную сторону). Решение - не доверять базовому алгоритму:

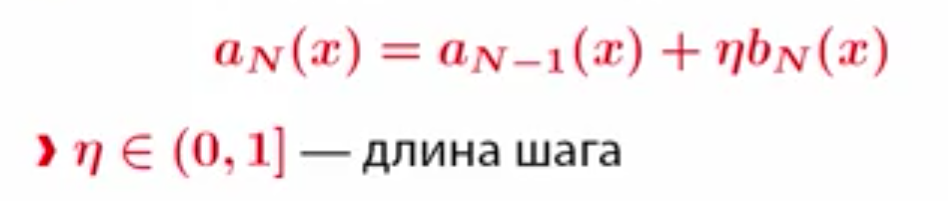


Рисунок 2.10.1. Корректировка учета ответа нового базового алгоритма при построении композиции

Получится очень аккуратное движение в пространстве. Влияние шага на ошибки на обучающей и валидационной выборках представлены ниже:

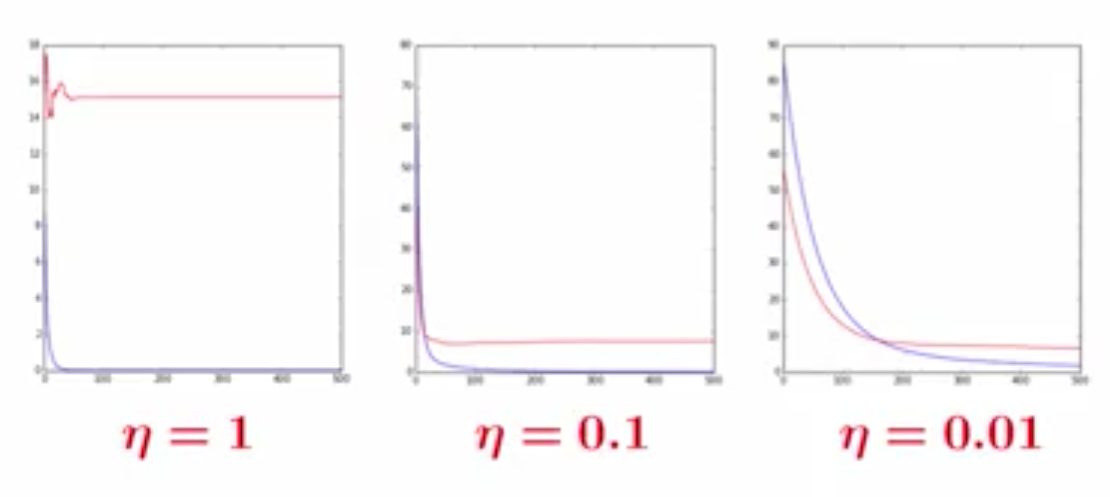


Рисунок 2.10.2. Влияние шага обучение на качество обучения

Чем меньше шаг, тем больше базовых алгоритмов надо. Шаг - гиперпараметр, как и количество базовых алгоритмов. Можно либо зафиксировать количество базовых алгоритмов и перебирать значения шага или наоборот.

Еще один подход в борьбе с переобучением - стохастический градиентный бустинг, т.е. обучение алгоритма на случайной выборке - бэггинг. Влияние и шага обучения, и бэггинга:

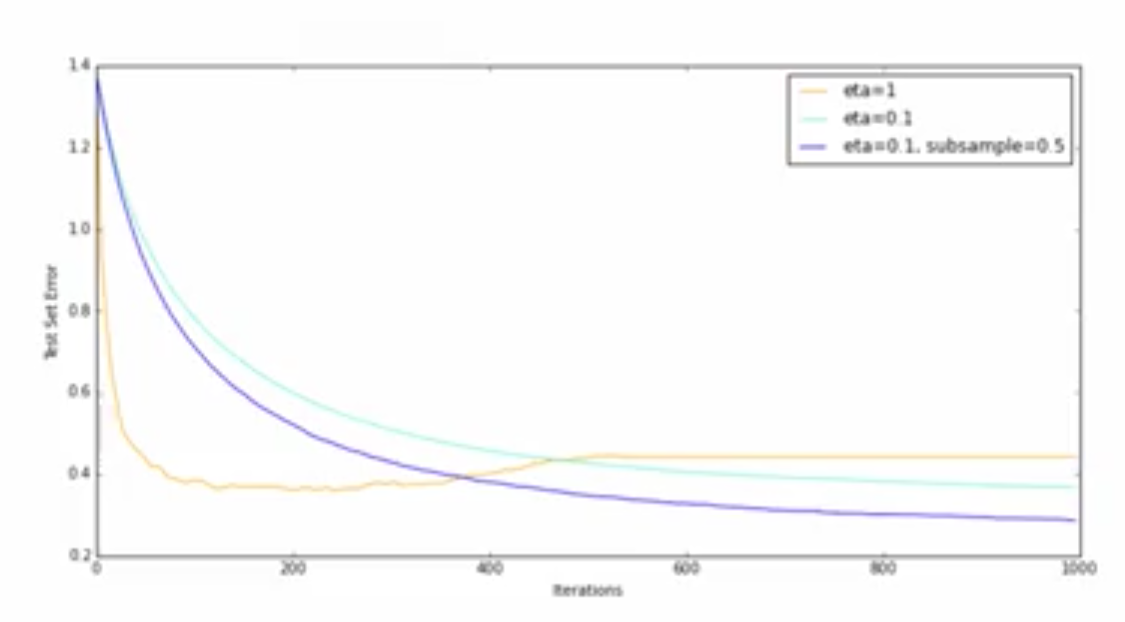


Рисунок 2.10.3. Влияние и шага обучение, и использования бэггинга на качество обучения

Видим, что при использовании обоих подходов достигается сходимость, причем более быстрая, чем в случае без бэггинга.

В градиентном бустинге в качестве базовых деревьев обычно выбираются деревья с небольшой глубиной 2-8.

# 3 Исследование различных алгоритмов с деревьями

### 3.1 Чтение данных

Считаем данные:

data = load\_boston()

X = data.data

Y = data.target

### 3.2 Разделение выборки на обучающую и валидационную

Разделим выборку в отношении 3:1:

training\_data\_set\_length = int(X.shape[0]\*0.75)

x = X[:training\_data\_set\_length]

y = Y[:training\_data\_set\_length]

x\_test = X[training\_data\_set\_length:]

y\_test = Y[training\_data\_set\_length:]

### 3.3 Реализация бустинга с константным шагом обучения

Реализуем бустинг с использованием класса DecisionTreeRegressor из библиотеки scikit learn в качестве базового алгоритма - дерева:

def predict(x, trained\_trees, weights):

return [sum([weight \* tree.predict([xi])[0] for tree, weight in zip(trained\_trees, weights)]) for xi in x]

def boosting(x, y, trees\_count, max\_depth, weights):

trained\_trees = []

curr\_labels = np.array(y)

for \_ in range(trees\_count):

trained\_tree = tree.DecisionTreeRegressor(max\_depth = max\_depth, random\_state = 42).fit(x, curr\_labels)

trained\_trees.append(trained\_tree)

curr\_labels = y - predict(x, trained\_trees, weights)

return trained\_trees

def rmse(x, y, trained\_trees, weights):

return np.sqrt(metrics.mean\_squared\_error(y, predict(x, trained\_trees, weights)))

trees\_count = 50

max\_depth = 5

weights = [0.9] \* trees\_count

trained\_trees = boosting(x, y, trees\_count, max\_depth, weights)

print('predictions:')

print(predict(x\_test, trained\_trees, weights))

print('rmse: %s' % rmse(x\_test, y\_test, trained\_trees, weights))

Здесь также реализована функция прогноза построенной композиции с учетом константного шага 0.9 и также rmse в качестве функции ошибки. Полученная ошибка: 5.455565103009402.

### 3.4 Реализация бустинга с изменяющимся шагом обучения

Реализуем бустинг, где вместо константного шага будет изменять шаг по формуле 0.9 / (1.0 + i), где i - номер итерации:

weights = [(0.9 / (1.0 + i)) for i in range(trees\_count)]

Полученная ошибка снизилась до 4.812550945781193.

### 3.5 Исследование влияния числа итераций и глубины дерева на переобучение градиентного спуска

В этот раз воспользуемся библиотечной реализацией градиентного спуска GradientBoostingRegressor из той же библиотеки. Построим графики:

def rmse\_sklearn(x, y, predictor):

return np.sqrt(metrics.mean\_squared\_error(y, predictor.predict(x)))

trees\_counts = np.linspace(10, 100, 10)

max\_depths = np.linspace(2, 20, 10)

fig, ax = plt.subplots(nrows = 2, ncols = 5)

ax = ax.flatten()

i = 0

for max\_depth in max\_depths:

rmse\_train = []

rmse\_test = []

for trees\_count in trees\_counts:

gradient\_boosting\_regressor = GradientBoostingRegressor(n\_estimators = int(trees\_count), max\_depth = int(max\_depth), random\_state = 42).fit(x, y)

rmse\_train.append(rmse\_sklearn(x, y, gradient\_boosting\_regressor))

rmse\_test.append(rmse\_sklearn(x\_test, y\_test, gradient\_boosting\_regressor))

ax[i].plot(trees\_counts, rmse\_train, color = "green")

ax[i].plot(trees\_counts, rmse\_test, color = "red")

i += 1

plt.show()

Ниже представлены полученные графики:

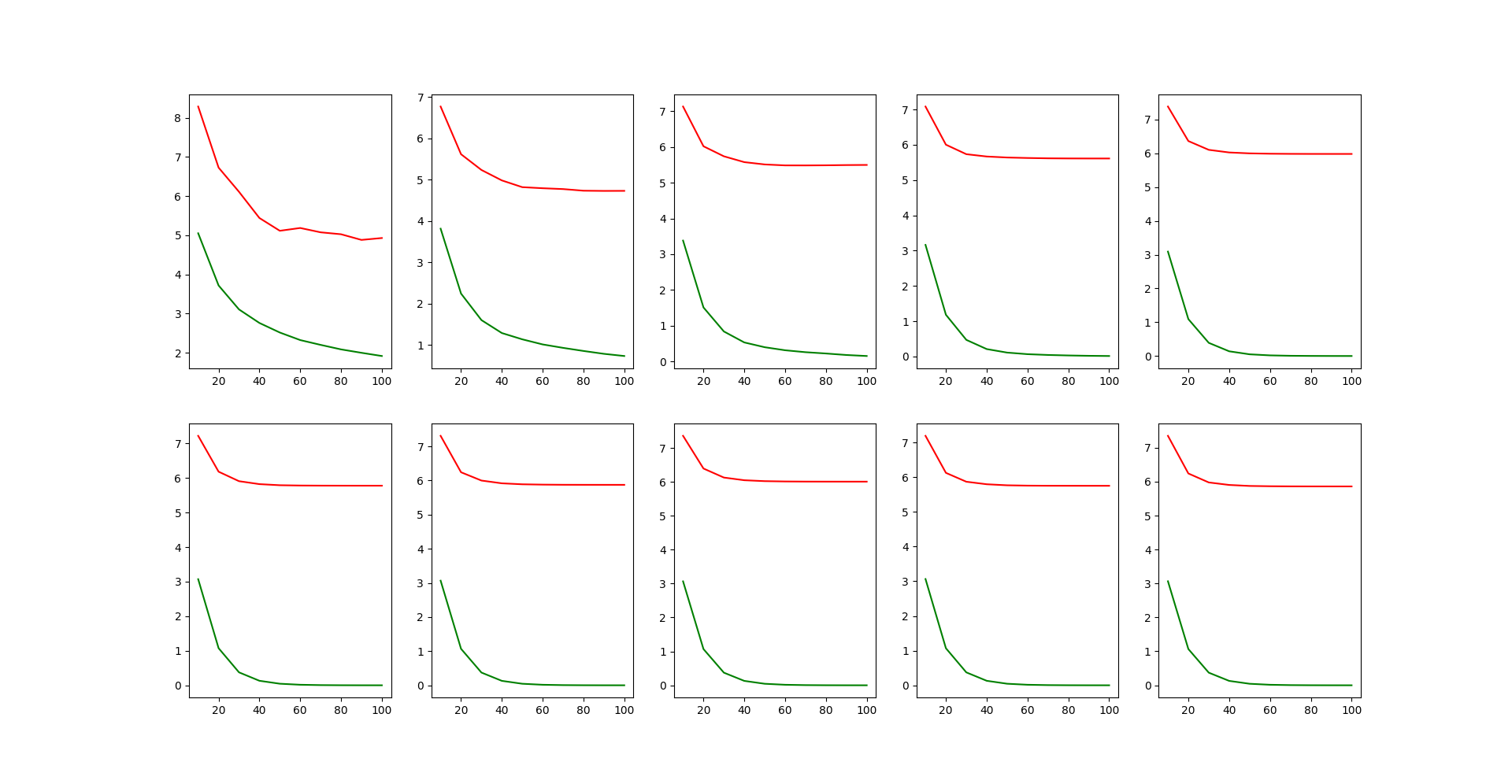


Рисунок 3.5.1. Влияние количества итераций и глубины дерева на переобученность градиентного бустинга

Здесь для каждой глубины дерева начиная от 2 до 20 с шагом 2 построен график зависимости количества деревьев от ошибок на обучающей и валидационных выборках. Видим, что при значительно больших значениях максимальной глубины деревьев алгоритм переучивается: ошибка на обучающей выборке невысока, в то время как на валидационной она очень высокая.

### 3.6 Сравнение с линейной регрессии

Используем реализацию линейной регрессии LinearRegression из библиотеки scikit learn:

linear\_regression = LinearRegression().fit(x, y)

print('linear regression rmse: %s' % rmse\_sklearn(x\_test, y\_test, linear\_regression))

Получим rmse = 8.254979753549401. Заметим, что на графиках из предыдущих пунктах удавалось получить на порядок меньшую ошибку на валидационной выборки - порядка 5.5 в случае с 100 деревьями глубины 2.