**Введение**

Наверное, хочется сразу рвануть в бой, но сначала поговорим про то, какую именно задачу будем решать и каково ее место в области машинного обучения.  
Классическое, общее (и не больно-то строгое) определение машинного обучения звучит так (T. Mitchell "Machine learning", 1997):

говорят, что компьютерная программа *обучается* при решении какой-то задачи из класса *T*, если ее производительность, согласно метрике *P*, улучшается при накоплении опыта *E*.

Далее в разных сценариях под *T, P*, и *E* подразумеваются совершенно разные вещи. Среди самых популярных **задач *T* в машинном обучении**:

* классификация – отнесение объекта к одной из категорий на основании его признаков
* регрессия – прогнозирование количественного признака объекта на основании прочих его признаков
* кластеризация – разбиение множества объектов на группы на основании признаков этих объектов так, чтобы внутри групп объекты были похожи между собой, а вне одной группы – менее похожи
* детекция аномалий – поиск объектов, "сильно непохожих" на все остальные в выборке либо на какую-то группу объектов
* и много других, более специфичных. Хороший обзор дан в главе "Machine Learning basics" книги ["Deep Learning"](http://www.deeplearningbook.org/) (Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016)

Под **опытом *E***понимаются данные (без них никуда), и в зависимости от этого алгоритмы машинного обучения могут быть поделены на те, что обучаются *с учителем* и *без учителя* (supervised & unsupervised learning). В задачах обучения без учителя имеется*выборка*, состоящая из *объектов*, описываемых набором *признаков*. В задачах обучения с учителем вдобавок к этому для каждого объекта некоторой выборки, называемой *обучающей*, известен *целевой признак* – по сути это то, что хотелось бы прогнозировать для прочих объектов, не из обучающей выборки.

Наконец, третья абстракция в определении машинного обучения – это **метрика оценки производительности алгоритма P.** Такие метрики различаются для разных задач и алгоритмов, и про них мы будим говорить по мере изучения алгоритмов. Пока скажем, что самая простая метрика качества алгоритма, решающего задачу классификации – это доля правильных ответов (accuracy, не называйте ее точностью, этот перевод зарезервирован под другую метрику, precision) – то есть попросту доля верных прогнозов алгоритма на тестовой выборке.

**Дерево решений**

Энтропия Шеннона определяется для системы с N возможными состояниями следующим образом:

где – вероятности нахождения системы в i-ом состоянии.  Опуская предпосылки введения (комбинаторные и теоретико-информационные) этого понятия, отметим, что, интуитивно, энтропия соответствует степени хаоса в системе.   Чем выше энтропия, тем менее упорядочена система и наоборот.

Поскольку энтропия – по сути степень хаоса (или неопределенности) в системе, уменьшение энтропии называют приростом информации. Формально прирост информации (information gain, IG) при разбиении выборки по признаку Q определяется как

где  – число групп после разбиения,  – число элементов выборки, у которых признак Q имеет i-ое значение.

В основе популярных алгоритмов построения дерева решений, таких как ID3 и C4.5, лежит принцип жадной максимизации прироста информации – на каждом шаге выбирается тот признак, при разделении по которому прирост информации оказывается наибольшим. Дальше процедура повторяется рекурсивно, пока энтропия не окажется равной нулю или какой-то малой величине (если дерево не подгоняется идеально под обучающую выборку во избежание переобучения).  
В разных алгоритмах применяются разные эвристики для "ранней остановки" или "отсечения", чтобы избежать построения переобученного дерева

 самая простая эвристика для обработки количественных признаков в дереве решений: количественный признак сортируется по возрастанию, и в дереве проверяются только те пороги, при которых целевой признак меняет значение. Звучит не очень строго, но надеюсь, я донес смысл с помощью игрушечных примеров. Дополнительно, когда в данных много количественных признаков, и у каждого много уникальных значений, могут отбираться не все пороги, описанные выше, а только топ-N, дающих максимальный прирост все того же критерия. То есть, по сути, для каждого порога строится дерево глубины 1, считается насколько снизилась энтропия (или неопределенность Джини) и выбираются только лучшие пороги, с которыми стоит сравнивать количественный признак/

Еще примеры дискретизации количественных признаков можно посмотреть в постах, подобных [этому](http://kevinmeurer.com/a-simple-guide-to-entropy-based-discretization/)или [этому](http://clear-lines.com/blog/post/Discretizing-a-continuous-variable-using-Entropy.aspx). Одна из самых известных научных статей на эту тему – "On the handling of continuous-valued attributes in decision tree generation" (U.M. Fayyad. K.B. Irani, "Machine Learning", 1992).

Основные параметры дерева

В принципе дерево решений можно построить до такой глубины, чтоб в каждом листе был ровно один объект. Но на практике это не делается (если строится только одно дерево) из-за того, что такое дерево будет *переобученным* – оно слишком настроится на обучающую выборку и будет плохо работать на прогноз на новых данных. Где-то внизу дерева, на большой глубине будут появляться разбиения по менее важным признакам (например, приехал ли клиент из Саратова или Костромы). Если утрировать, может оказаться так, что из всех 4 клиентов, пришедших в банк за кредитом в зеленых штанах, никто не вернул кредит. Но мы не хотим, чтобы наша модель классификации порождала такие специфичные правила.

Есть два исключения, ситуации, когда деревья строятся до максимальной глубины:

* Случайный лес (композиция многих деревьев) усредняет ответы деревьев, построенных до максимальной глубины (почему стоит делать именно так, разберемся позже)
* Стрижка дерева (*pruning*). При таком подходе дерево сначала строится до максимальной глубины, потом постепенно, снизу вверх, некоторые вершины дерева убираются за счет сравнения по качеству дерева с данным разбиением и без него (сравнение проводится с помощью *кросс-валидации*, о которой чуть ниже). Подробнее можно почитать в материалах [репозитория](https://github.com/esokolov/ml-course-hse) Евгения Соколова.

Дерево решений в задаче регрессии

При прогнозировании количественного признака идея построения дерева остается та же, но меняется критерий качества:

* Дисперсия вокруг среднего: ,  
  где  – число объектов в листе, – значения целевого признака. Попросту говоря, минимизируя дисперсию вокруг среднего, мы ищем признаки, разбивающие выборку таким образом, что значения целевого признака в каждом листе примерно равны.

#### **Плюсы и минусы деревьев решений**

Плюсы:

* Порождение четких правил классификации, понятных человеку, например, "если возраст < 25 и интерес к мотоциклам, то отказать в кредите". Это свойство называют интерпретируемостью модели;
* Деревья решений могут легко визуализироваться, то есть может "интерпретироваться" (строгого определения я не видел) как сама модель (дерево), так и прогноз для отдельного взятого тестового объекта (путь в дереве);
* Быстрые процессы обучения и прогнозирования;
* Малое число параметров модели;
* Поддержка и числовых, и категориальных признаков.

Минусы:

* У порождения четких правил классификации есть и другая сторона: деревья очень чувствительны к шумам во входных данных, вся модель может кардинально измениться, если немного изменится обучающая выборка (например, если убрать один из признаков или добавить несколько объектов), поэтому и правила классификации могут сильно изменяться, что ухудшает интерпретируемость модели;
* Разделяющая граница, построенная деревом решений, имеет свои ограничения (состоит из гиперплоскостей, перпендикулярных какой-то из координатной оси), и на практике дерево решений по качеству классификации уступает некоторым другим методам;
* Необходимость отсекать ветви дерева (pruning) или устанавливать минимальное число элементов в листьях дерева или максимальную глубину дерева для борьбы с переобучением. Впрочем, переобучение — проблема всех методов машинного обучения;
* Нестабильность. Небольшие изменения в данных могут существенно изменять построенное дерево решений. С этой проблемой борются с помощью ансамблей деревьев решений (рассмотрим далее);
* Проблема поиска оптимального дерева решений (минимального по размеру и способного без ошибок классифицировать выборку) NP-полна, поэтому на практике используются эвристики типа жадного поиска признака с максимальным приростом информации, которые не гарантируют нахождения глобально оптимального дерева;
* Сложно поддерживаются пропуски в данных. Friedman оценил, что на поддержку пропусков в данных ушло около 50% кода CART (классический алгоритм построения деревьев классификации и регрессии – Classification And Regression Trees, в sklearn реализована улучшенная версия именно этого алгоритма);
* Модель умеет только интерполировать, но не экстраполировать (это же верно и для леса и бустинга на деревьях). То есть дерево решений делает константный прогноз для объектов, находящихся в признаковом пространстве вне параллелепипеда, охватывающего все объекты обучающей выборки. В нашем примере с желтыми и синими шариками это значит, что модель дает одинаковый прогноз для всех шариков с координатой > 19 или < 0.

**Введение и история появления бустинга**

В конце 80ых начали появляться работы с исследованием проблемы связи слабой и сильной обучаемости алгоритмов. Слабая обучаемость алгоритма означает, что за полиномиальное время возможно построить алгоритм распознавания, точность которого будет хотя бы немного больше 50%. Под сильной обучаемостью подразумевается, что возможно за полиномиальное время построить алгоритм, который бы мог давать сколь угодно точные результаты. Исследования показали, что сильная обучаемость эквивалентна слабой, поскольку любую слабую модель можно усилить, построив правильную композицию. . В 1996 эти идеи сформировались и приобрели законченную форму в виде алгоритма AdaBoost [7]. . Несколькими годами позже появилось важное обобщение этого алгоритма – градиентный бустинг [1].

[1] Friedman J. Greedy Function Approximation: A Gradient Boosting Machine. — IMS 1999 Reitz Lecture.

[7] Freund Y., Shapire R. Experiments with a New Boosting Algorithm. —1996.

Все началось с вопроса(статья) о тои, можно ли из большого количества относительно слабых и простых моделей получить одну сильную. Слабые модели – это произвольные алгоритмы машинного обучения точность которых может быть лишь немногим выше случайного угадывания.

Утвердительный математический ответ на этот вопрос нашёлся довольно быстро, но потребовалось несколько лет до появления работоспособных алгоритмов. Их общий подход заключался в жадном построении линейной комбинации простых моделей (базовых алгоритмов) путем перевзвешивания входных данных. Каждая последующая модель (как правило, дерево решений) строился таким образов, чтобы придать больший вес и предпочтения ранее некорректно предсказанным наблюдениям.

Добавляем (бустим) слабые алгоритмы, наращивая ансамбль постепенными улучшениями тех участков данных, где предыдущие модели не справлялись. Но при построении следующей простой модели, она строиться не просто на перевзвешенных наблюдениям, а так, чтобы лучшим образом приближать общий градиент целевой функции.

**Постоновка задачи**

Мы будем решать задачу восстановления функции в общем контексте обучения с учителем. У нас будет набор пар признаков  и целевых переменных ,

,

на котором мы будем восстанавливать зависимость вида . Восстанавливать будем приближением  , а для понимания, какое приближение лучше, у нас также будет функция потерь , которую мы будем минимизировать:

 Функция  должна быть дифференцируемой.

Будем искать наши приближения   так, чтобы в среднем минимизировать функцию потерь на тех данных, что есть:

Ограничим пространство поиска каким-нибудь конкретным параметризованным семейством функций  Это сильно упрощает задачу, так как она сводится к уже вполне решаемой оптимизации значений параметров:

Будем искать приближение параметра итеративно. Также выпишем наше приближение  за  итераций в виде суммы

выписать эмпирическую функцию потерь , показывающую, насколько хорошо мы их оценили по имеющимся у нас данных.

Осталось взять подходящий итеративный алгоритм, который будет минимизировать . Самый простой и часто используемый – градиентный спуск. Для него нужно выписать градиент и добавлять итеративный оценки вдоль него со знаком минус (для уменьшения ошибки).  Hадо только как-то инициализировать наше первое приближение θ0^ и выбрать, сколько итераций M мы эту процедуру будем продолжать.

 Инициализировать начальное приближение параметров

 Для каждой итерации  повторять:

1. Посчитать градиент функции потерь  при текущем приближении
2. Задать текущее итеративное приближение  на основе посчитанного градиента
3. Обновить приближение параметров :

 Сохранить итоговое приближение 

 Пользоваться найденной функцией   по назначению

**Функциональный градиентный спуск**

Пусть мы можем проводить оптимизацию в функциональном пространстве и итеративно искать приближения  в виде самих функций. Выпишем наше приближение в виде суммы инкрементальных улучшений, каждое из которых является функцией. Для удобства сразу будем считать эту сумму, начиная с начального приближения :

Будем искать наше приближение не в виде одной большой модели с кучей параметров (как, например, нейросеть), а в виде суммы функций, делая вид, что таким образом мы двигаемся в функциональном пространстве.

ограничить свой поиск каким-то семейством функций , где – некоторый вектор параметров. Но, во-первых, сумма моделей может быть сложнее чем любая модель из этого семейства. Во-вторых, общая задача все еще происходит в функциональном пространстве. Сразу учтем, на каждом шаге для функций нам понадобится подбирать оптимальный коэффициент . Для шага  задача выглядит следующим образом:

**Классический GBM**

 На вход алгоритма нужно собрать несколько составляющих:

* набор данных ;
* число итераций ;
* выбор функции потерь  с выписанным градиентом;
* выбор семейства функций базовых алгоритмов , с процедурой их обучения;
* дополнительные гиперпараметры , например, глубина дерева у деревьев решений;

Единственный момент, который остался без внимания — начальное приближение . Для простоты, в качестве инициализации используют просто константное значение . Его, а также оптимальный коэффициент  находят бинарным поиском, или другим line search алгоритмом относительно исходной функции потерь. Итак, GBM алгоритм:

1. Инициализировать GBM константным значением
2. Для каждой итерации  повторять:
   1. Посчитать псевдо-остатки
   2. Построить новый базовый алгоритм  как регрессию на псевдо-остатках
   3. Найти оптимальный коэффициент  при  относительно исходной функции потерь
   4. Сохранить
   5. Обновить текущее приближение
3. Скомпоновать итоговую GBM модель

\_\_\_\_\_\_другое\_\_\_\_\_\_\_

Будем искать приближения функции в виде композиции

Однако подбор оптимального набора параметров очень трудоемкая задача. Будем строить такую композицию путем жадного наращивания, каждый раз добавляя в сумму слагаемое, являющееся наиболее оптимальным алгоритмом из возможных. Таким образом задача сводиться к поиску пары наиболее оптимальных параметров длины

Оптимальность здесь понимается в соответствии с принципом явной максимизации отступов [3].

Воронцов К. В. Машинное обучение (курс лекций).

Это означает, что вводится некоторая функция потерь , показывающая, насколько "сильно"предсказанный ответ отличается от правильного ответа yi. И затем минимизируется функционал ошибки

Заметим, что функционал ошибки – вещественная функция, зависящая от точек в -мерном пространстве, и нам необходимо решить задачу минимизации этого функционала. Сделаем это, реализуя один шаг метода градиентного спуска. В качестве точки, для которой мы будем искать оптимальное приращение, рассмотрим . Найдем градиент функционала ошибки

В силу метода градиентного спуска, наиболее выгодно добавить новое слагаемое следующим образом

где подбирается линейным поиском по вещественным числам :

Однако представляет из себя лишь вектор оптимальных значений для каждого объекта , а не базовый алгоритм из семейства , определенный . Поэтому нам необходимо найти наиболее похожий на . Сделаем это, опять минимизируя функционал ошибки, основанный на принципе явной максимизации отступов:

**Функция потерь**

Сначала разберемся с регрессией . Выбирая функцию потерь в этом случае, мы прежде всего решаем, какое именно свойство условного распределения  мы хотим восстановить. Наиболее частые варианты:

* , оно же L2 loss, оно же Gaussian loss. Это классическое условное среднее, самый частый и простой вариант. Если нет никакой дополнительной информации или требований к устойчивости (робастности) модели — используйте его.
* , оно же L1 loss, оно же Laplacian loss. Эта, на первый взгляд, не очень дифференцируемая вещь, на самом деле определяет условную медиану. Медиана, как мы знаем, более устойчива к выбросам, поэтому в некоторых задачах эта функция потерь предпочтительнее, так как она не так сильно штрафует большие отклонения, нежели квадратичная функция.
* , оно же Lq loss, оно же Quantile loss. Если бы мы, допустим, захотели не условную медиану с L1, а условную 75%-квантиль, мы бы воспользовались этим вариантом с . Можно видеть, что эта функция ассиметрична и больше штрафует наблюдения, оказывающиеся по нужную нам сторону квантили.

Рассмотрим частные случаи для решения задач регрессии.

**Метод наименьших квадратов**

Рассмотрим следующую функцию потерь:

.

Тогда алгоритме градиентного бустинга примет следующий вид:

Этот случай бустинга называется LS-boosting4 или GentleBoost.

**Минимизация среднего модуля отклонения**

Далее рассмотрим еще более простой вариант функции потерь – модуль отклонения:

Тогда примет следующий вид:

Тогда линейный поиск приобретает следующий вид:

где medianW – взвешенное медианное значение набора. Т.е. задача свелась к задаче поиска порядковой статистики в массиве, а для нее существуют решения за . Такая разновидность бустинга называется LAD-boosting

**Бустинг над решающими деревьями**

Сейчас же рассмотрим в качестве базового семейства алгоритмов регрессионные решаюшие деревья из вершин [9]

[9] Breiman L., Friedman J., Olshen R., Stone C. Classiﬁcation and Regression Trees. — Wadsworth, 1983.

Важен тот факт, что каждое решающее дерево имеет листовых вершин, соответствующие непересекающимся областям , на которые разбивается пространство объектов .Каждой листовой вершине соответствует некоторое значение регрессии , которое будет ответом классификатора в случае попадания анализируемого объекта в соответствующую область. Можно записать этот факт следующей формулой:

где – индикатор события . Видно, что в этой сумме ровно одно слагаемое будет ненулевым. Тогда добавление слагаемого в градиентном бустинге будет происходить следующим образом:

Таким образом мы просто прибавляем к имеющему алгоритму некоторое другое решающее дерево. Попробуем найти оптимальные значения для уже построенных :

И, поскольку области не пересекаются, можем переписать эту формулу в следующем виде:

Таким образом вместо того чтобы выполнять линейный поиск коэффициента перед новым слагаемым, как в классическом градиентном бустинге, мы полностью перенастраиваем параметры дерева с фиксированными . Это позволяет строить более качественную композицию. Такая разновидность бустинга называется Tree-boost.

**Стохастический градиентный бустинг**

Основная причина эффективности бустинга в том, что алгоритм на каждой итерации строит базовый алгоритм, который действительно эффективен лишь на части подвыборки. Этот принцип можно усилить, сделав небольшую модификацию бустинга. А именно, на каждом шаге алгоритма новое слагаемое считается, опираясь не на всю обучающую выборку, а лишь на случайную подвыборку фиксированного размера. Эта идея очень популярна и эффективна. Она является объединением техник градиентного бустинга и беггинга. Также можно брать неслучайную подвыборку объектов, а еще и случайную подвыборку признаков объектов. Это называется техникой случайных подпространств9.

9Метод случайных подпространств, RSM – от англ. Random Subspace Method [15]

[15] Ho, Tin. The Random Subspace Method for Constructing Decision Forests. — IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998.

Результаты работы таких модификаций зачастую заметно превосходят по качеству различные нестохастические варианты.

**Устойчивость и регуляризация**

У градиентного бустинга есть один важный внешний параметр . При увеличении повышается устойчивость и точность у результирующей композиции. Можно рассматривать это как **первый параметр регуляризации**.

Теперь немного модифицируем шаг алгоритма градиентного бустинга, на котором добавляется новый базовый алгоритм в композицию. А именно, добавим дополнительный константный коэффициент для каждого базового алгоритма:

При уменьшении ν каждый новый базовый алгоритм, будь он достаточно точен или плох, будет учитываться неполностью. Это повышает устойчивость получаемой композиции. Параметр можно рассматривать как второй параметр регуляризации. В англоязычной литературе и спецификациях различных алгоритмов он называется shrinkage.

Заметим, что при уменьшении требуется увеличение общего количества алгоритмов . Таким образом вычислительная сложность построения композиции увеличивается, однако увеличивается и финальное качество

Борьба с шумом

На каждой итерации градиентного бустинга алгоритм стремится максимально исправить все ошибки на обучении. Однако это бессмысленно при наличии шума в исходных данных и ведет к переобучению. Проблему можно решить, учитывая веса объектов на каждой итерации, ведь по ним можно судить о сложности обучения на них. Действительно, большой вес у объекта показывает, что предыдущие алгоритмы плохо работали на нем и, возможно, этот объект шумовой.

Преимущества бустинга

На сегодняшний день является одним из самых мощных алгоритмов распознавания. Это достигается благодаря вышеупомянутой адаптивной технике построения композиции. К тому же, бустинг предоставляет множество возможностей для вариаций. Во-первых, можно рассматривать различные функции потерь. Это позволяет решать как задачи классификации, так и задачи регрессии. К тому же, возможность выбора произвольной функции потерь позволяет акцентировать внимание на особенностях данных в задаче. Во-вторых, возможно рассмотрение любого семейства базовых алгоритмов. А это, опять же, дает широкиие возможности учета особенностей даннной задачи. Бустинг над решающими деревьями считается одним из наиболее эффективных вариантов бустинга. А учитывая, что решающие деревья всвою очередь тоже используют базовые алгоритмы (например, пороговые, линейные и т.п.), в результате получается огромное количество вариантов для настройки. В-третьих, благодаря достаточной простоте метода и четкому математическому обоснованию, в каждой конкретной вариации бустинга не сложно провести некоторые математические и алгоритмические оптимизации, которые заметно ускорят работу алгоритма.

5.2 Недостатки бустинга

Разумеется, бустинг не лишен недостатков. Во-первых, бустинг – трудоемкий метод, и работает он достаточно медленно. Зачастую требуется построение сотен или даже тысяч базовых алгоритмов для композиции. Во- вторых, без дополнительных модификаций он имеет свойство полностью подстраиваться под данные, в том числе под ошибки и выбросы в них. В-третьих, идея бустинга обычно плохо применима к построению композиции из достаточно сложных и мощных алгоритмов. Построение такой композиции занимает очень много времени, а качество существенно не увеличивается. В-четвертых, результаты работы бустинга сложно интерпретируемы, особенно если в композицию входят десятки алгоритмов.

**Элементы обучения с учителем**

XGBoost решает проблему обучения с учителем, где обучающий набор (с несколькими входными переменными) используется для предсказания целевую переменную .

Модель в обучения с учителем это математическая структура, с помощью которой делается прогноз из входных данных . Например, линейная модель, где предказание дается как линейная комбинация взвешенных входных переменных.

Предсказанное значение может имен различную интерпретация, зависящую от задачи т. е. регрессия или классификация. На пример, значения можно лорифмировать и получить вероятность принадлежность к классу в логистической регрессии или же оно может быть использовано в качестве ранга в задаче ранжирования.

Параметры – это неопределённая часть, зависящая от входных данных. В задачи линейной регрессии параметрами являются коэффициенты .

**Регуляция целевой функции**

Задача обучения модели сводиться к поиску параметров наилучшим образов соответствующие обучающимся данным и целевой переменной. Для обучения модели нужно определить целевую функцию, чтобы измерить, насколько хорошо модель соответствует обучающей выборке.

Целевая функция состоит из двух частей: потери обучения и регуляризации.

Функция потерь измеряет насколько точно прогнозируем значения соответствует соответствует обучаемым данным. Например, средняя квадратичная ошибка или функция логистических потерь.

Регуляризация контролирует сложность модели, что помогает избежать переобучения. Общий принцип заключается в том, что требуется как простая, так и предсказывавшая модель. Компромисс между эти двумя требованиями так же отсылает к дилемме смещения-дисперсии.

Ансамбли деревьев решений.

Модель ансамбля дерева состоит из набора деревьев классификации и регрессии (CART).

GART немного отличается от деревьев решений, в которых лист содержит только значения решений. В CART реальная оценка связанная с каждым из листьев, что дает более богатые интерпретации, выходящие за рамки классификации.

Обычно одного дерева недостаточно, чтобы использовать его на практике. Больше распространены ансамблевые модели, которые суммируют прогноз нескольких деревья вместе.

Где – количество деревьев

– функция из функционального пространства

Целевая функция принимает вид

Tree Bosting

Аддитивное обучение

Каковы параметры деревьев? Для этого нужно изучить функции , каждая из которых содержит структуры дерева и оценку для листьев. Древовидная структура труднее оптимизировать, невозможно выучить все деревья одновременно.

Вместо этого будим использовать аддитивную страданию

На каждом шаге добавляется дерево, которое лучше оптимизирует цель.

Если рассмотреть среднеквадратичную ошибку в качестве функции потерь, целевая функция становиться

Разложим функцию потерь по формуле Тейлора до второго члена

Параметрами модели становятся и

**Сложность модели**

Определение дерева как

Где - вектор оценки для листов,

– функция, присваивавшая каждую точку данных соответствующему листву

– количество листов

Функция регуляризации

**Структура оценки**

Перепишем целевую функцию

множество точек данных, присвоенных листу

В этой записи независимы друг от друга, форма – квадратичный и лучший для данной структуры и лучшее объективное сокращение, которое можно получить

**Обучение древовидной структуры**

Будем оптимизировать один уровень дерева за раз. В частности, будет пытаться разделить лист на два, и полученный результат равен

Это формула состоит из

1. Оценки на новом левой листе
2. Оценки на новом правом листе
3. Оценки на исходной листе
4. Регуляризации на дополнительно листе

Здесь, заметен метод обреки в древовидных моделях. Если прирост мошне лучше не добавлять новой ветки.

Для поиска оптимального разбиения, данные размещаются в отсортированном порядке. Сканирование слева направо достаточно для расчета оценки всех возможных разбиений и нахождения лучшего разбиения.

**XGBoost параментры**

Parameters for Tree Booster

eta [alias: learning\_rate] – размер шага сжатия, используемый в обновлении для предотвращения переоснащения. После каждого шага в boosting, eta уменьшает весовые характеристики, чтобы сделать boosting процесс более консервативным

gamma [alias: min\_split\_loss]

Минимальное уменьшения функции потерь необходимое для дальнейшего разбиения дерева на узлы. Чем больше gamma, тем консервативнее модель.

max\_depth

Максимальная глубина дерева. Увеличение этого значения сделает модель более сложной и скорее всего переобученной.

min\_child\_weight

минимальная сумма весов экземпляра (гессиан) необходимая для ребенка. Если в результате шага разбиения дерева получится листовой узел с суммой весов экземпляра меньше, чем min\_child\_weight, то процесс построения дальнейшего разбиение прекращается. Чем больше значение, тем более консервативный алгоритм.

Subsample

Соотношения подвыборки от тренировочной выборки. Прореживание происходит каждую итерации boosting.

colsample\_bytree   
Соотношения подвыборки столбцов при построении каждого дерева. Подвыборка выполняется одни раз для каждого построенного дерева.

**n\_estimators**

Количество деревьева для обучения