Разбор методов построения Uplift-моделей

# Условные обозначения

Вспомним условные обозначения и термины используемые в Uplift-моделировании.

X - признаки пользователя: его описание и контекст

T∈{0,1} - флаг воздействия на пользователя (было ли оно)

Y(1) - целевая переменная во вселенной, где на пользователя было воздействие

Y(0) - целевая переменная во вселенной, где воздействия не было

Y=Y(1)T+Y(0)(1−T) - целевая переменная в нашей вселенной

Один из важных показателей - Individual Treatment Effect (ITE) (также известен как Causal Effect (CE)):

ITE=Y(1)−Y(0)

Необходимо понимать, что ITE принципиально невозможно наблюдать, потому что невозможно одновременно воздействовать и не воздействовать на клиента.

Напомним также и о среднем эффекте от воздействия ATE (Average Treatment Effect) - на сколько во вселенной, где всем делают предложение, целевой показатель больше, чем во вселенных, где предложение не делают:



и при условии независимости T и X получим (T丄X)[[1]](#footnote-0):

-разность условных мат ожиданий. Мат ожидания уже можно посчитать исходя из наблюдений в нашем эксперименте. Для этого нужно посчитать среднее значение целевого показателя в ЦГ и вычесть среднее значение в контрольной[[2]](#footnote-1).

Для удобства введем обозначение



Был введен также условный средний эффект от воздействия CATE (Conditional Average Treatment Effect) - это и есть то, что прогнозирует uplift - на сколько целевой показатель при воздействии на клиента с параметрами x больше, чем во вселенной, где на клиента не воздействуют:



Или учитывая условия CIA ( {Y(1), Y(0)} 丄 T|X )[[3]](#footnote-2):

- мат ожидание Y при условии что X=x, а T=1 минус второе мат ожидание, где T=0. В такой записи мы можем получать оценки мат ожиданий, из тех данных, которые получили в эксперименте.

А также напомним и про Propensity Score:

 - вероятность отнесения пользователя к целевой группе при условии что пользователь выглядит как x.

Его смысл следующий: если в эксперименте флаг воздействия выбирался случайно (как обычно и происходит) с вероятностью p, то e(x)=p. Эта функция имеет большое значение при работе с observational data - когда воздействие производилось не в рамках контролируемого эксперимента.

Как uplift зависит от того или иного фактора[[4]](#footnote-3)?

Берем фактор, рассматриваем каждое из его возможных значений. В разрезе этих значений смотрим какой uplift в соответствующей группе пользователей получится.

# T-Learner

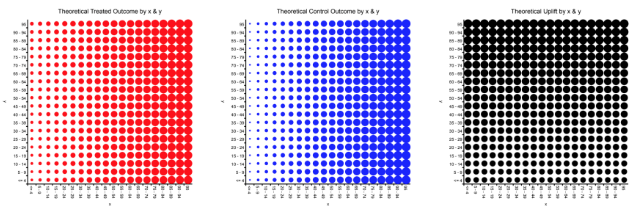
Первым метод для построения Uplift-моделей, который мы разберем называется T-Learner (T от слова two - два). Это довольно простой метод. Он заключается в построении своей модели для каждой из групп (целевой и контрольной), которая прогнозирует целевой показатель Y. И в качестве прогноза CATE (то есть по сути Uplift) мы выдаем разность прогнозов этих 2х построенных моделей.

Алгоритм следующий:

1. Построим 2 модели (x) и (x) предсказывающие целевой показатель Y: одну для целевой группы (X1, Y1) , то есть на которую осуществляется воздействие, и вторую для контрольной группы (X0, Y0)
2. Спрогнозируем Uplift (CATE) (x) как разность (x) = (x) - (x)

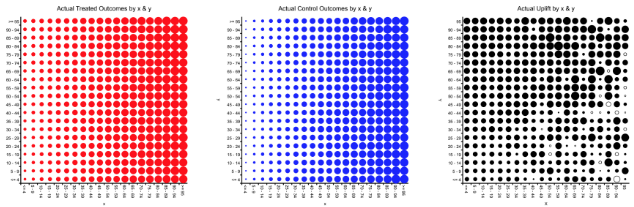
На практике метод T-learner работает хуже других подходов, особенно при небольшом количестве данных. Одна из главных причин - мы не предсказываем Uplift напрямую, а через целевую переменную, поэтому модель обращает внимание на факторы влияющие больше на саму Y, а не на его прирост.

Это иллюстрировано на 2х примерах ниже. Мы будем предсказывать Uplift от 2х факторов: x и y. Видно, что для прогноза этого показателя важен фактор x. В итоге построенные модели будут больше обращать внимание на показатель х. В то время как, если посмотреть на распределение Uplift, то для него решающим будет играть y, а не х.



Толщина кружка пропорциональна среднему значению целевой переменной (или CATE для 3 графика) при таких значениях факторов (x, y)

Если теперь довериться полученным моделям и посмотреть спрогнозированный Uplift, то получим странную с виду случайную картину (снизу справа). В целом видно, что алгоритм плохо улавливает зависимость от y.



# X-Learner

Следующей рассматриваемой моделью будет X-Learner. Названа она так из-за своей перекрестной схемы прогноза.

1. Сначала мы как и в предыдущей модели построим 2 модели (x) и (x) на контрольной и целевой выборках. В качестве целевых переменных будем использовать Y0 и Y1 соответственно.
2. Потом сформируем новые целевые переменные: [[5]](#footnote-4) и  (здесь и появляется перекрестность). В данном случае мы получим грубую прикидку прироста целевого показателя для конкретного пользователя по сравнению со средним ожидаемым значением этого целевого показателя, если бы мы не воздействовали на него.
3. Далее уже на новых полученных парах (X0, D0) и (X1, D1) мы построим модели (x) и (x). По факту эти модели уже близки к тому, что нужно прогнозировать, а именно к Uplift'у. В целом, каждую из них по отдельности можно использовать как прогноз CATE.
4. Итоговый прогноз CATE (Uplift) делается по формуле: (x) = g(x)(x) + (1 - g(x))(x) , где коэффициент g(x) зависит от x. Авторы метода советуют выбирать g(x)=1[[6]](#footnote-5) в случае, когда целевая группа много больше контрольной, и g(x)=0 наоборот. В более сбалансированных случаях они рекомендуют в качестве g(x) брать модель прогноза propensity score e(x). Ее можно отдельно строить на данных эксперимента, если не вы проводили эксперимент.

# R-Learner

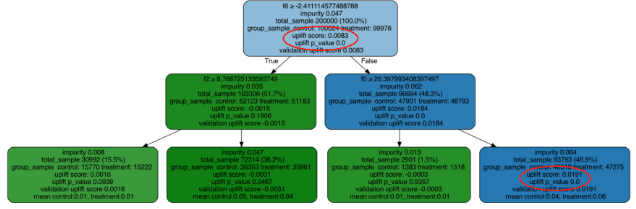
Следующий подход называется R-Learner. Он тоже относится к группе так называемых мета-подходов. Это группа подходов, под капотом которых можно использовать любые типы моделей машинного обучения, которые будут прогнозировать необходимую величину. Он устроен следующим образом:

1. Разбиваем нашу обучающую выборку на Q фолдов (обычно на 5-10)
2. Для каждого фолда q мы строим 2 модели:(x) и (x). Первая модель прогнозирует Y по X, вторая - T. Каждая модель обучается на всех фолдах, кроме выбранного фолда q (отсюда и минус в индексе) (X(-q), Y(-q)) и (X(-q), T(-q)). Как результат будем иметь 2Q моделей. Причем, разбиение на фолды не зависит от разбиения на контрольную и целевую группы. Если данные из случайного эксперимента - в качестве e можно не использовать прогноз, а брать константу (долю пользователей целевой группы в данных).
3. Строим модель (x) для минимизации следующей функции потерь:



# Uplift-деревья

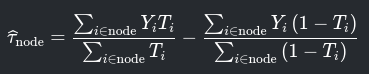
Uplift можно также моделировать использую решающие деревья. Пример простого решающего дерева для Uplift-задачи можно увидеть ниже.



Пример решающего дерева для Uplift-моделирования

Видно, что как в обычных решающих деревьях происходит разбиение всех данных по значению какого-то признака. Это разбиение происходит согласно некоторому критерию. То есть мы хотим разбить данные таким образом, чтобы прогноз Uplift'а различался максимально в листьях.

Как же оценить uplift в листьях? Будем просто сравнивать среднее значение Y в целевой группе и в контрольной:

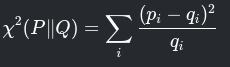


Разбиение в вершине выбирается по принципу максимизации (оптимизации) некоторого критерия. Самый простой из критериев - максимизация прироста:

 - сделать так, чтобы прогноз среднего uplift в левом и правом листе отличались как можно больше.

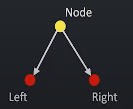
Другими критериями могут служить критерии, основанные на сравнении распределений. Сравнивать распределения можно с помощью так называемых дивергенций (меры различий, которые не являются метриками в полном смысле этого слова, а лишь обладают некоторыми их свойствами).

Пусть на множестве значений {y1, …, ym} заданы 2 распределения: P = {p1, …, pm} и Q = {q1, …, qm}. Тогда можно определить дивергенцию между этими распределениями. Вот некоторые из них:

* Дивергенция Кульбака-Лейблера: 
* Энергетическое расстояние (Energy distance): 
* Расстояние χ2 (Пирсона):

Как определить критерий разбиения по заданной дивергенции?

Пусть мы находимся в вершине 0 и должны решить как разбить множество записей, которые в нее попали на 2 группы.



Мы должны максимизировать разницу 2х показателей: дивергенцию после разбиения - дивергенция до.



До разбиения будут интересовать 2 распределения таргета: в ЦГ и КГ.



После разбиения дивергенция считается: мы пробежимся по каждому из листов и считаем аналогичную (Dbefore\_split) дивергенцию, но только по тем записям, которые к этому потомку попали. И взвешиваем 2 получившихся числа с весом доли попавших в эту подвершину записей. Логика в том, чтобы сделать распределения после разбиения на левую и правую часть - эти части были максимально непохожими.



Или вот еще один критерий: Contextual Treatment Selection. Этот метод больше подходит, когда нужно выбирать, какой из методов воздействия применить, но тем не менее он подходит и для случая с единственным воздействием.

В этом подходе мы пытаемся максимизировать следующий показатель:



Выглядит громоздко и непонятно, но на самом деле смысл этого выражения в том, что мы разбиваем наши данные на 2 группы так, что если в каждой из этих групп выбрать тот вариант воздействия, который приносит наибольшую выгоду, то взвешенная сумма этих максимальных прогнозов будет лучше, чем если мы оставим все записи в исходной одной группе.

Резюме

В данной лекции мы разобрали 3 мета-алгоритма построения Uplift-моделей:

1. T-Learner
2. X-Learner
3. R-Learner

Посмотрели, что решающие деревья тоже можно использовать, как Uplift-модель, причем есть множество критериев построения таких решающих деревьев, основанные на так называемых дивергенциях.

В заключение хотелось бы отметить некоторые "плюсы" и "минусы" подхода с решающими деревьями.

Плюсы (+):

1. Деревья прогнозируют Uplift напрямую - в каждом листе получается несмещенная оценка
2. Легко контролировать надежность алгоритма (робастность)
3. Вполне легко и удобно интерпретировать результаты (можно смотреть на структуру дерева)

Минусы (-):

1. К сожалению, в текущей реализации они очень медленно строятся
2. CTS кажется несколько “странным” критерием в случае с единственным типом воздействия - он постоянно пытается выделить подвершину, в которой CATE будет отрицательным. Метод будет плохо работать с данными, в которых истинный CATE на всех пользователях положителен.

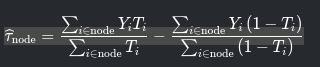
# Домашнее задание

В этом задании предлагается собственными руками написать алгоритм построения uplift-дерева.

(To do: мотивация)

Нужно будет реализовать uplift-дерево с критерием разбиения DeltaDeltaP. Напомним, как его расчитать.

Определим оценку uplift в вершине дерева. Она будет равна разности средних значений Y в целевой группе и в контрольной:



Тогда нужно выбрать такое разбиение вершины, при котором максимизируется



Подробнее про uplift-деревья можно почитать в конспекте второй лекции.

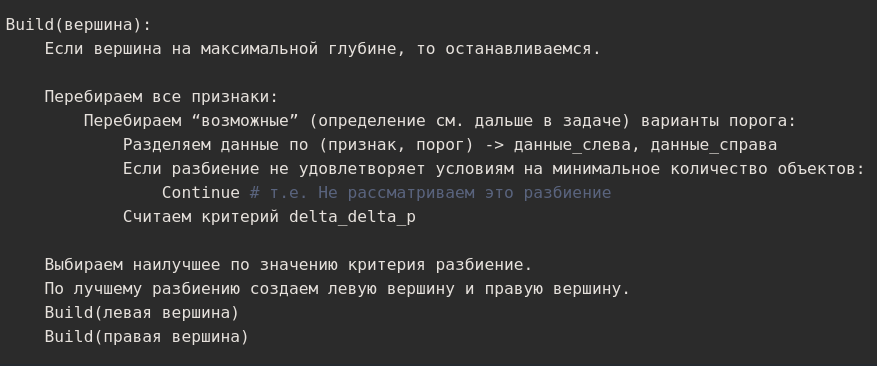
Также примеры расчета можно посмотреть в excel файле (ссылка на delta\_delta\_p.xlsx).

Краткое описание процесса построения дерева:

Создаем корневую вершину. В нее кладем все объекты выборки

Для корневой вершины выполняем (рекурсивно) процедуру Build

**Псевдокод**



Обратите внимание: и построение, и применение модели будут детерминированными. В данном задании будет проверяться точное соответствие прогнозов, сделанных вашим алгоритмом, и сделанных “правильным” (автора задания) алгоритмом.

**Требования**

Будет необходимо написать класс UpliftTreeRegressor, у которого есть 3 обязательных метода

*# Конструктор*

def \_\_init\_\_(

self,

max\_depth: int = 3, # максимальная глубина дерева.

min\_samples\_leaf: int = 1000, # минимальное необходимое число обучающих объектов в листе дерева.

min\_samples\_leaf\_treated: int = 300, # минимальное необходимое число обучающих объектов с T=1 в листе дерева.

min\_samples\_leaf\_control: int = 300, # минимальное необходимое число обучающих объектов с T=0 в листе дерева.

):

# do something

*# Обучение*

def fit(

self,

X: np.ndarray, # массив (n \* k) с признаками.

treatment: np.ndarray, # массив (n) с флагом воздействия.

y: np.ndarray # массив (n) с целевой переменной.

) -> None:

# fit the model

*# Применение*

def predict(self, X: np.ndarray) -> Iterable[float]:

# compute predictions

return predictions

**Важно!**

Перебор возможных вариантов порогов для разбиения должен (!) производиться согласно следующему псевдокоду:

# column\_values - одномерный массив со значениями признака в текущей вершине.

# threshold\_options - получившиеся варианты порога. Их и нужно будет перебрать при подборе оптимального порога.

import numpy as np

unique\_values = np.unique(column\_values)

if len(unique\_values) > 10:

percentiles = np.percentile(column\_values, [3, 5, 10, 20, 30, 50, 70, 80, 90, 95, 97])

else:

percentiles = np.percentile(unique\_values, [10, 50, 90])

threshold\_options = np.unique(percentiles)

Сравнение значений c порогом производится так: если f <= threshold, то объект относится к левой вершине, иначе - к правой.

Описание проверки

В каждом тест-кейсе будет происходить:

* Создается объект model класса UpliftTreeRegressor (реализованный вами класс) с параметрами max\_depth, min\_samples\_leaf, … (см. Описание конструктора выше)
* model обучается на входных данных (X, treatment, y).
* model применяется к данными X\_test.
* Получившиеся прогнозы сверяются с “правильными” прогнозами.

Псевдокод

def \_check(model\_constructor, model\_params: dict, X, treatment, y, X\_test, pred\_right) -> bool:

model = model\_constructor(\*\*model\_params)

model.fit(X, treatment, y)

pred = np.array(model.predict(X\_test)).reshape(len(X\_test))

passed = (np.max(np.abs(pred - pred\_right)) < EPS)

return passed

**Ограничения**:

* len(X) <= 100000.
* Число признаков <= 10.
* Значения всех признаков - вещественные числа.
* Значения флага воздействия из {0, 1}.
* Значения целевой переменной - вещественные числа.

**Пример**

Для отладки вашего решения приводим пример данных и получившихся из них дерева и прогнозов.

Данные, на которых строилось дерево

* example\_X.npy - таблица с признаками X
* example\_treatment.npy - вектор флагов воздействия
* example\_y.npy - вектор целевой переменной

**Параметры дерева**

* max\_depth=3,
* min\_samples\_leaf=6000,
* min\_samples\_leaf\_treated=2500,
* min\_samples\_leaf\_control=2500

**Результат**

* example\_preds.npy - вектор прогнозов модели (дерева), если применить к обучающим данным (таблице X).
* example\_tree.txt - тестовое описание получившегося дерева. Для каждой вершины указано
  + n\_items - сколько обучающих примеров попало в вершину
  + ATE
  + split\_feat - признак, по которому было произведено разбиение
  + split\_threshold - порог, по которому было произведено разбиение

**Полезные ссылки**

Хорошая статья с разбором написания decision tree (не для uplift). Можно взять этот пример за каркас своего решения.https://towardsdatascience.com/decision-tree-from-scratch-in-python-46e99dfea775

Реализация uplift-дерева в библиотеке causalml. Обратите внимание, что там не реализован критерий DeltaDeltaP из нашего домашнего задания.https://github.com/uber/causalml/blob/master/causalml/inference/tree/models.py

1. Флаг воздействия не зависит от того, какому пользователю делаем предложение. [↑](#footnote-ref-0)
2. EDA/Как устроены treatment, visit и conversion? [↑](#footnote-ref-1)
3. Мы знаем, что пользователь описывается X, то тот факт, что мы воздействуем на него или нет, не зависит от того, какое значение принимает целевой показатель в той или иной вселенной. [↑](#footnote-ref-2)
4. EDA/Как uplift зависит от факторов? [↑](#footnote-ref-3)
5. Берем целевой показатель для ЦГ и вычитаем из него прогноз модели для КГ [↑](#footnote-ref-4)
6. Т.е. мы будем использовать τ0, прогнозирующую D0, которая использует прогнозы μ0, построенной на бОльшем кол-ве данных. [↑](#footnote-ref-5)