Санкт-Петербургский государственный университет
Прикладная математика и информатика

Отчет по научно-исследовательской работе

Ансамбли в классическом машинном обучении

Выполнил:
студент 221 группы
Дувакин Кирилл Борисович

Научный руководитель:
(старший научный сотрудник, д.ф.-м.н.) Р. Н. Мокаев

Работа выполнена полностью и в срок.

Отметка о зачете: A

_____ Мокаев Р.Н.

Санкт-Петербург 2022

Содержание

1 Введение

Ансамблем (Ensemble, Multiple Classifier System) называется алгоритм, который состоит из нескольких алгоритмов машинного обучения, а процесс построения ансамбля называется ансамблированием (ensemble learning). Простейший пример ансамбля в регрессии – усреднение нескольких алгоритмов, а в задаче классификации - взятие моды предсказаний нескольких алгоритмов. Алгоритмы из которых состоит ансамбль называются базовыми алгоритмами (base learners).

Большинство приёмов в прикладном ансамблировании направлено на то, чтобы ансамбль был «достаточно разнообразным», тогда ошибки отдельных алгоритмов на отдельных объектах будут компенсироваться корректной работой других алгоритмов. По сути, при построении ансамбля:

1. повышают качество базовых алгоритмов,
2. повышают разнообразие (diversity) базовых алгоритмов.

Разнообразие повышают за счёт:

1. «варьирования» обучающей выборки (бэггинг),
2. «варьирования» признаков (Random Subspaces),
3. «варьирования» целевого вектора (ECOC, деформации целевого признака),
4. «варьирования» моделей (использование разных моделей, стэкинг)

5. «варьирование» в модели (использование разных алгоритмов в рамках одной, рандомизация в алгоритме – случайный лес).

В этой работе мы рассмотрим такие приемы, как бэггинг в общем, а также его применения над деревьями решений с методом Random Subspaces, а также бустинг в общем и также применительно к деревьям решений. Но для начала нужно познакомится с некоторым инструментом, который поможет нам в обосновании ансамблирования вообще.

2 Bias-variance decomposition

В машинном обучении и статистике существует такая проблема, как компромисс между смещением и дисперсией - свойство набора моделей предсказания, когда модели с меньшим отклонением от имеющихся данных имеют более высокую дисперсию на новых данных, и наоборот. Тогда и возникает конфликт при попытке одновременно минимизировать эти два источника ошибки, которые снижают обобщающую способность алгоритмов. Нестрого, но уточним понятия смещения и дисперсии в нашем контексте:

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

Дисперсия - это ошибка чувствительности к малым отклонениям в тренировочном наборе. При высокой дисперсии алгоритм может как-то трактовать случайный шум в тренировочном наборе, а не желаемый результат (переобучение). Разложение смещения-дисперсии — это способ анализа ожидаемой ошибки обобщения алгоритма обучения для частной задачи сведением к сумме трёх членов — смещения, дисперсии и величины, называемой неустранимой погрешностью, которая является результатом шума в самой задаче.

Дилемма возникает во всех формах обучения с учителем — в классификации, регрессии (аппроксимация функции). Выведем теперь разложение на примере следующей задачи:
Пусть $X = (x_i, y_i)_{i=1}^{l}$ - обучающая выборка. Рассмотрим задачу регрессии с квадратичной функцией потерь:

$$\mathscr{L}(a, x) = (a(x) - y(x)), \quad y : x \mapsto y(x) \in \mathbb{R}$$

Тут a - модель, которую мы используем для предсказания. Будем считать, что $y = f(x) + \varepsilon$, где f - некоторая детерминированная функция, а ε -случайный шум, обладающий следующими свойствами:

1. $\mathbb{E}\varepsilon = 0$
2. $\mathbb{V}ar\varepsilon = \mathbb{E}\varepsilon^2 = \sigma^2$

Заметим, что знание значения функции потерь на одном объекте не может дать нам общего понимания того, насколько хороша наша модель. Например, при оценке качества модели мы могли бы, например, учитывать, что выход модели на объекте x зависит не только от самого объекта, но и от выборки X, на которой модель обучалась:

$$a(x) = a(x, X)$$

Кроме того, значение y на объекте x зависит не только от x, но и от реализации шума в этой точке:

$$y(x) = y(x, \varepsilon)$$

Наконец, измерять качество мы бы хотели на тестовых объектах x - тех, которые не встречались в обучающей выборке, а тестовых объектов у нас в большинстве случаев более одного. При включении всех вышеперечисленных источников случайности в рассмотрение логичной оценкой качества алгоритма a кажется следующая величина:

$$Q(a) = \mathbb{E}_x \mathbb{E}_{X,\varepsilon}[y(x,\varepsilon) - a(x,X)]^2$$

Внутреннее матожидание позволяет оценить качество на одной тестовой точке в зависимости от реализаций X, ε, а внешнее усредняет это качество по всем тестовым точкам. Кроме того $\mathbb{E}_{X,\varepsilon} = \mathbb{E}_X \mathbb{E}_\varepsilon$, поскольку X и ε независимы. Преобразуем данное выражение:

$$\begin{gathered} Q(a) = \mathbb{E}_{X,\varepsilon}[y(x,\varepsilon) - a(x,X)]^2 = \mathbb{E}_{X,\varepsilon}[f(x) + \varepsilon - a(x,X)]^2 = \\ = \mathbb{E}_{X,\varepsilon} \underbrace{[f(x) - a(x,X)]^2}_{\text{не зависит от } \varepsilon} - \mathbb{E}_{X,\varepsilon} \underbrace{[2\varepsilon(f(x) - a(x,X))]}_{\text{множители независимы}} + \mathbb{E}_{X,\varepsilon}[\varepsilon^2] = \\ = \mathbb{E}_X[f(x) - a(x,X)]^2 - 2\mathbb{E}_\varepsilon[\varepsilon]\mathbb{E}_X[f(x) - a(x,X)] + \mathbb{E}[\varepsilon]^2 = \\ = \mathbb{E}_X[f(x) - a(x,X)]^2 + \sigma^2 \end{gathered}$$

На этом этапе выделилась дисперсия шума ε. Продолжим преобразование:

$$\begin{gathered} \mathbb{E}_X[f(x) - a(x,X)]^2 = \mathbb{E}_X[f(x) - a(x,X) + \mathbb{E}_X[a(x,X)] - \mathbb{E}_X[a(x,X)]]^2 = \\ = \mathbb{E}_X \underbrace{[f(x) - \mathbb{E}_X[a(x,X)]]^2}_{\text{не зависит от X}} + 2\mathbb{E}_X[(f(x) - \mathbb{E}_X[a(x,X)])(\mathbb{E}_X[a(x,X) - a(x,X)])] + \\ + \mathbb{E}_X \underbrace{[a(x,X) - \mathbb{E}_X[a(x,X)]^2]}_{\mathbb{V}ar[a(x,X)]} = \underbrace{(f(x) - \mathbb{E}_X[a(x,X)])^2}_{bias_X a(x,X)} + \mathbb{V}ar[a(x,X)] = \\ = bias_X^2 a(x,X) + \mathbb{V}ar[a(x,X)] \end{gathered}$$

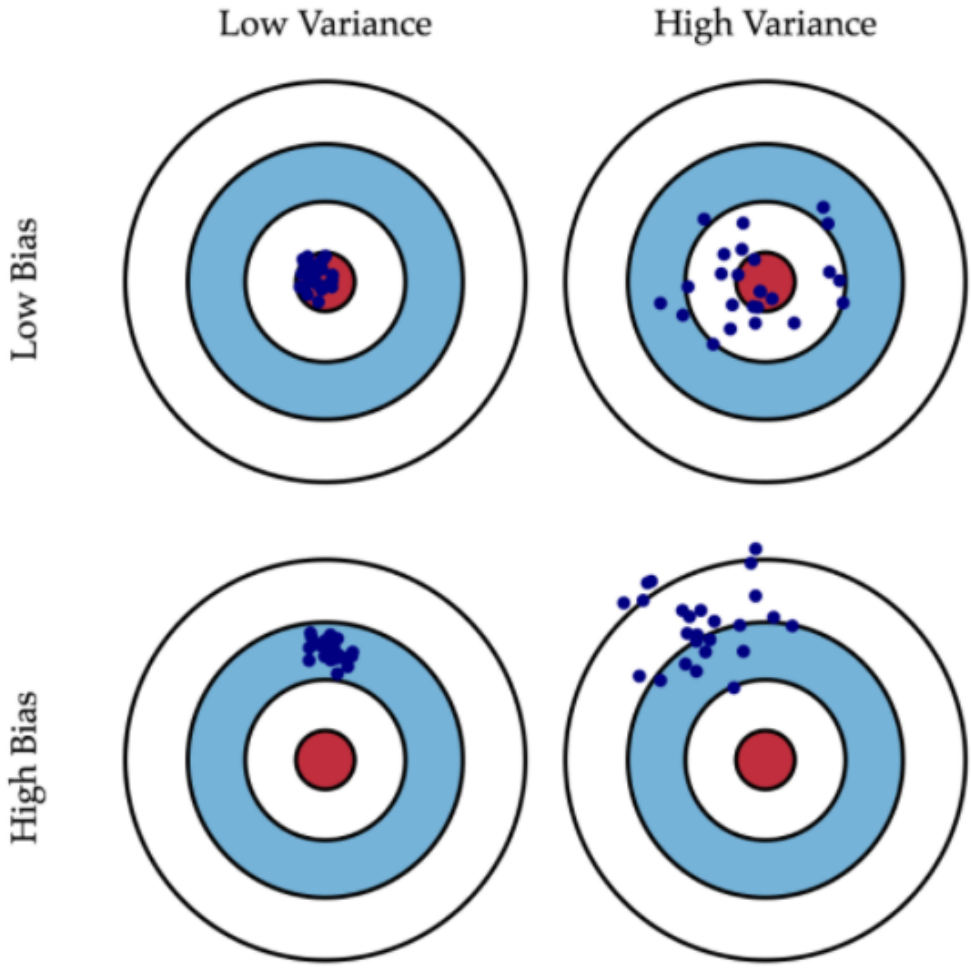
Таким образом получим:

$$Q(a) = \mathbb{E}_x bias_X^2 a(x,X) + \mathbb{E}_x \mathbb{V}ar[a(x,X)] + \sigma^2$$

Резюмируем:

1. $bias_X^2 a(x,X) = f(x) - \mathbb{E}_X[a(x,X)]$ - смещение предсказания алгоритма в точке x по всем возможным обучающим выборкам X, относительно истинной зависимости f
2. $\mathbb{V}ar[a(x,X)] = \mathbb{E}_X[a(x,X) - \mathbb{E}_X[a(x,X)]]^2$ - дисперсия(разброс) предсказаний алгоритма в зависимости от обучающей выборки X
3. $\sigma^2 = \mathbb{E}_x \mathbb{E}_\varepsilon[y(x,\varepsilon) - f(x)]^2$ - неустранимый шум в данных.

Смещение показывает, насколько хорошо с помощью данного алгоритма можно приблизить истинную зависимость f, а разброс характеризует чувствительность алгоритма к изменениям в обучающей выборке.



3 Bagging (Bootstrap AGGregation)

Воспользуемся теперь соображениями из раздела про bias-variance-decomposition. Ошибка раскладывается на шум, смещение и разброс. С шумом ничего не сделать. Тогда чтобы минимизировать ошибку надо минимизировать, например, разброс, не увеличивая смещения.

Идея бэггинга:

Выберем из обучающей выборки равновероятно и с повторением n примеров. Получим X^1. С помощью некоторого алгоритма b обучим модель $b_1(x) = b(x, X^1)$. Повторим процедуру k раз и получим k моделей, обученных на k выборках:

$$b_i(x) = b(x, X^i), i = 1 \ldots, k$$

Теперь чтобы получить одно предсказание усредним предсказания моделей:

$$a(x) = \frac{1}{k}[b_1(x) + \cdots + b_k(x)]$$

Процесс генерации подвыборок с помощью семплирования с возвращением называется бутстрепом (bootstrap)

Модели $b_1(x), \ldots, b_k(x)$ - базовые алгоритмы (модели), а модель $a(x)$ называется их ансамблем.

Посмотрим, что происходит с качеством предсказания при переходе от одной модели к ансамблю. Сначала убедимся, что смещение ансамбля не изменилось по сравнению со средним смещением отдельных моделей. Будем считать, что когда мы берём матожидание по всем обучающим выборкам X, то в эти выборки включены также все подвыборки, полученные бутстрепом:

$$bias_X a(x, X) = f(x) - \mathbb{E}_X[a(x, X)] = f(x) - \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} b(x, X^i)\right] =$$

$$= f(x) - \frac{1}{k}\sum_{i=1}^{k}\mathbb{E}_X[b(x, X^i)] = f(x) - \frac{1}{k}\sum_{i=1}^{k}\mathbb{E}_X[b(x, X)] = f(x) - \mathbb{E}_X[b(x, X)] =$$

$$= bias_X b(x, X)$$

Таким образом получаем, что смещение ансамбля равно смещение одного базового алгоритма. Посмотрим теперь, что происходит с разбросом.

$$\mathbb{V}ar[a(x, X)] = \mathbb{E}_X[a(x, X) - \mathbb{E}_X[a(x, X)]]^2 =$$

$$= \mathbb{E}_X\left[\frac{1}{k}\sum_{i=1}^{k} b(x, X^i) - \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} b(x, X^i)\right]\right]^2 =$$

$$= \frac{1}{k^2}\mathbb{E}_X\left[\sum_{i=1}^{k}(b(x, X^i) - \mathbb{E}_X[b(x, X^i)])\right]^2 =$$

$$= \frac{1}{k^2}\sum_{i=1}^{k}(b(x, X^i - \mathbb{E}_X[b(x, X^i)]))^2 +$$

$$+\frac{1}{k^2}\sum_{k_1 \neq k_2}\mathbb{E}_X[(b(x, X^{k_1}) - \mathbb{E}_X[b(x, X^{k_1})])(b(x, X^{k_2}) - \mathbb{E}_X[b(x, X^{k_2})])] =$$

$$= \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{V}ar_X b(x, X^i) + \frac{1}{k^2}\sum_{k_1 \neq k_2} cov(b(x, X^{k_1}), b(x, X^{k_2}))$$

Если предположить, что базовые алгоритмы некоррелированы, то:

$$\mathbb{V}ar[a(x, X)] = \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{V}ar_X b(x, X^i) =$$

$$= \frac{1}{k^2}\sum_{i=1}^{k}\mathbb{V}ar_X b(x, X) = \frac{1}{k}\mathbb{V}ar_X b(x, X)$$

Получилось, что в этом случае дисперсия композиции в k раз меньше дисперсии одного алгоритма.

4 Random Forest

Мы сделали предположение, что базовые алгоритмы некоррелированы, и за счёт этого получили очень сильное уменьшение дисперсии у ансамбля относительно входящих в него базовых алгоритмов. Однако в реальной жизни добиться этого сложно: ведь базовые алгоритмы учили одну и ту же зависимость на пересекающихся выборках. Поэтому будет странно, если корреляция

на самом деле нулевая. Но на практике оказывается, что строгое выполнение этого предположения не обязательно. Достаточно, чтобы алгоритмы были в некоторой степени не похожи друг на друга. На этом строится развитие идеи бэггинга для решающих деревьев — случайный лес.

Построим ансамбль алгоритмов, где базовый алгоритм - это решающее дерево. Будем строить по следующей схеме:

1. Для построения i-го дерева:
   (a) Из обучающей выборки X выбирается с возвращением случайная подвыборка X^i того же размера.
   (b) Случайно выбираются $n < N$ признаков, где N - общее количество признаков. Для каждого дерева своя подвыборка и свой набор признаков.
   (c) На этих n выбранных признаках и выборке X^i строится i-ое дерево.
2. Чтобы получить предсказание ансамбля на тестовом объекте, усредняем отдельные ответы деревьев (для регрессии) или берём самый популярный класс (для классификации).
3. Profit. Мы построили Random Forest (случайный лес) – комбинацию бэггинга и метода случайных подпространств (пункт. b) над решающими деревьями.

Возникают естественный вопрос: "какой должна быть глубина деревьев в случайном лесе?". И ответ вытекает из следующих рассуждений. Ошибка модели (на которую мы можем повлиять) состоит из смещения и разброса. Разброс мы уменьшаем с помощью процедуры бэггинга. На смещение бэггинг не влияет, а хочется, чтобы у леса оно было небольшим. Поэтому смещение должно быть небольшим у самих деревьев, из которых строится ансамбль. У неглубоких деревьев малое число параметров, то есть дерево способно запомнить только верхнеуровневые статистики обучающей подвыборки. Они во всех подвыборках будут похожи, но будут не очень подробно описывать целевую зависимость. Поэтому при изменении обучающей подвыборки предсказание на тестовом объекте будет стабильным, но не точным (низкая дисперсия, высокое смещение). Наоборот, у глубоких деревьев нет проблем запомнить подвыборку подробно. Поэтому предсказание на тестовом объекте будет сильнее меняться в зависимости от обучающей подвыборки, зато в среднем будет близко к истине (высокая дисперсия, низкое смещение). Вывод: используем глубокие деревья.

Также в построенном случайном лесе возникает такой гиперпараметр, как количество признаков для обучения дерева. Чем больше признаков, тем больше корреляция между деревьями и тем меньше чувствуется эффект от ансамблирования. Чем меньше признаков, тем слабее сами деревья. Практическая рекомендация – брать корень из числа всех признаков для классификации и треть признаков для регрессии.

Еще один гиперпараметр случайного леса это количество базовых решающих деревьев. Поэтому имеет смысл построить график ошибки от числа деревьев и ограничить размер леса в тот момент, когда ошибка перестанет значимо уменьшаться. Также у леса есть одно крайне положительное свойство: случайный лес можно строить и применять параллельно, что сокращает время работы, если у нас есть несколько процессоров.

5 Boosting

Вспомним, как строится ансамбль с помощью бэггинга: обучаем k базовых алгоритмов независимо друга от друга, и берем среднее предсказаний (для регрессии) или моду предсказаний (для классификации) базовых алгоритмов. Этот подход имеет следующий недостаток: не исключено, что каждый базовый алгоритм будем ошибаться на каком-то подмножестве объектов (одном и том же для всех базовых алгоритмов) и поскольку все они обучаются независимо, то и итоговый ансамбль будет иметь большую ошибку на этих объектах. Или например какие-то базовые алгоритмы могли получится не очень хорошими, поскольку присутствуют элементы случайности, а другие получатся очень точными, но в бэггинге мы это не учитываем, поскольку по итогу берем предсказание объекта с одинаковым весовым коэффициентов равным $\frac{1}{k}$. Но нам хотелось бы учесть их с разными весовыми коэффициентами. Из этих рассуждений возникает идея бустинга. Его главно отличие от бэггинга в том, что во-первых, базовые алгоритмы учитываются с разными весовыми коэффициентами, а во вторых каждый следующий алгоритм пытается уменьшить ошибку всех предыдущих, поэтому алгоритмы обучаются последовательно, а не параллельно, как это происходит в бэггинге.

Формальное знакомство с бустингом начнем с очень известного алгоритма AdaBoost. Для простоты будем строить модель для задачи бинарной классификации: $Y = \{\pm 1\}, X = (x_i, y_i)_{i=1}^{l}, b_t : X \to Y \cup \{0\}$

Рассмотрим взвешенное голосование T базовых алгоритмов:

$$a(x) = sign\left(\sum_{t=1}^{T} \alpha_t b_t(x)\right), \quad x \in X$$

Функциалом качества ансамьля будет выступать число ошибок на X:

$$Q_T = \sum_{i=1}^{l} \left[y_i \sum_{t=1}^{T} \alpha_t b_t(x_i) < 0 \right]$$

Две основные эвристики:

1. Фиксируем $\sum_{k=1}^{t-1} \alpha_k b_k(x)$ при добавлении $\alpha_t b_t(x)$
2. Гладкая аппроксимация пороговой функции потерь $[M < 0]$

Некоторые гладкие аппроксимации пороговой функции потерь:

1. $E(M) = e^{-M}$ - экспоненциальная
2. $L(M) = log(1 + e^{-M})$ - логарифмическая
3. $Q(M) = (1 - M)^2$ - квадратичная

В алгоритме AdaBoost используется экспоненциальная аппроксимация:

$$Q_T \leq \widetilde{Q}_T = \sum_{i=1}^{l} \underbrace{exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t b_t(x_i)\right)}_{w_i} exp(y_i \alpha_T b_T(x_i))$$

Когда мы пришли к шагу T и хотим построить $b_T(x)$, то мы уже построили первые $T - 1$ базовых алгоритмов, а потому они нам известны, а значит

величину w_i можно просто вычислить. По сути эти величины есть ни что иное как веса объектов. Вес w_i большой, если под экспонентой большой положительный вес, т.е. отступ $M < 0$, значит композиция первых $T-1$ базовых алгоритмов на данном объекте x_i отработала плохо. Таким образом получаем по-сути принцип такого бустинга: обучайся на тех объектах на которых предыдущая композиция отработала плохо. Кроме того еще нужно нормировать эти весовые коэффициенты:

$$\widetilde{W}^l = (\widetilde{w}_1, \ldots, \widetilde{w}_l), \widetilde{w}_i = \frac{w_i}{\sum w_j}$$

Введем пару определений, которые пригодятся в теоретическом исследовании AdaBoost.

Взвешенное число ошибочных (negative) и правильных (positive) классификаций на векторе весов $U^l = (u_1, \ldots, u_l)$:

$$N(b, U^l) = \sum_{i=1}^{l} u_i[b(x_i) = -y_i]$$

$$P(b, U^l) = \sum_{i=1}^{l} u_i[b(x_i) = y_i]$$

Основная теорема бустинга (для AdaBoost), 1996: Пусть $\mathscr{B}$ - достаточно богатое семейство базовых алгоритмов. Пусть для любого нормированного вектора весов U^l существует алгоритм $b \in \mathscr{B}$, классифицирующий выборку хотя бы немного лучше, чем наугад: $P(b, U^l) > N(b, U^l)$. Тогда минимум функционала $\widetilde{Q}_T$ достигается при

$$b_T = \arg\max_{b \in \mathscr{B}} \sqrt{P(b, \widetilde{W}^l)} - \sqrt{N(b, \widetilde{W}^l)}$$

$$\alpha_T = \frac{1}{2} \ln \frac{P(b_T, \widetilde{W}^l)}{N(b_T, \widetilde{W}^l)}$$

Доказательство:

Воспользуемся тождеством $\forall \alpha \in, \forall b \in \{\pm 1, 0\}$:

$$e^{-\alpha b} = e^{-\alpha}[b = 1] + e^{\alpha}[b = -1] + [b = 0]$$

Полодим для краткости $\alpha =_T$, $b_i = b_T(x_i)$. Тогда

$$\widetilde{Q}_T = \left(e^{-\alpha} \underbrace{\sum_{i=1}^{l} \widetilde{w}_i[b_i = y_i]}_{P} + e^{\alpha} \underbrace{\sum_{i=1}^{l} \widetilde{w}_i[b_i = -y_i]}_{N} + \underbrace{\sum_{i=1}^{l} \widetilde{w}_i[b_i = 0]}_{1-P-N} \right) \underbrace{\sum_{i=1}^{l} w_i}_{\widetilde{Q}_{T-1}}$$

Тогда

$$\widetilde{Q}_T = (e^{-\alpha} P + e^{\alpha} N + 1 - P - N) \widetilde{Q}_{T-1} \to \min_{\alpha, b}$$

$$\frac{\partial}{\partial \alpha} \widetilde{Q}_T = (-e^{-\alpha} P + e^{\alpha} N) \widetilde{Q}_{T-1} = 0 \Rightarrow e^{-\alpha} P = e^{\alpha} N \Rightarrow e^{2\alpha} = \frac{P}{N}$$

Получили требуемое:

$$\alpha_T = \frac{1}{2} \ln \frac{P(b_T, \widetilde{W}^l)}{N(b_T, \widetilde{W}^l)}$$

Подставим теперь полученное α обратно в $\widetilde{Q}_T$. Тогда останется оптимизационная задача по $b \in \mathscr{B}$:

$$\widetilde{Q}_T = (e^{-\alpha}P + e^{\alpha}N + 1 - P - N)\widetilde{Q}_{T-1} =$$
$$= (1 + \sqrt{\frac{N}{P}}P + \sqrt{\frac{P}{N}}N - P - N)\widetilde{Q}_{T-1} = (1 - (\sqrt{P} - \sqrt{(N)})^2)\widetilde{Q}_{T-1} \to \min_{b \in \mathscr{B}}$$

Поскольку $\widetilde{Q}_{T-1}$ не зависит от α_T, b_T, то мимимизация $\widetilde{Q}_{T-1}$ эквивалетна максимизации $\sqrt{P} - \sqrt{N}$. Таким образом получаем:

$$b_T = \arg\max_{b \in \mathscr{B}} \sqrt{P(b, \widetilde{W^l})} - \sqrt{N(b, \widetilde{W^l})}$$

Algorithm 1 AdaBoost

Input : training set X^l; T - number of basic algorithms
Output : basic algorithms b_t and their weights α_t, $t = 1, \ldots, T$

1: initialize weights of objects: $w_i = \frac{1}{l}$, $i = 1, \ldots, l$
2: for $t = 1, \ldots, T$ do
3: $\quad b_t = \arg\min_b N(b, \widetilde{W^l})$
4: $\quad \alpha_t = \frac{1}{2} \ln \frac{1 - N(b_t, \widetilde{W^l})}{N(b_t, \widetilde{W^l})}$
5: $\quad w_i = w_i exp(-\alpha_t y_i b_t(x_i)), i = 1, \ldots, l$
6: $\quad w_0 = \sum_{j=1}^{l} w_i$
7: $\quad w_i = \frac{w_i}{w_0}, i = 1, \ldots, l$
8: end for

Return $\{\alpha_t\}_{t=1}^T, \{b_t\}_{t=1}^T$

---

6 GB

Исторически алгоритм AdaBoost был одним из первых основанных на идее бустинга. Однако позже в результате исследований были получены модификации бустинга. Самой известной модификацией стал градиентный бустинг.

Пусть хотим построить линейный ансамбль базовых алгоритмов b_t из семейства $\mathscr{B}$:

$$a_T(x) = \sum_{t=1}^{T} \alpha_t b_t(x), \quad x \in X, \quad b_t : X \to, \quad \alpha_t \in \mathbb{R}_+$$

Также как и AdaBoost обучаем α_T, b_T при фиксированных предыдущих. Критерий качества с заданной гладкой функцией потерь $\mathscr{L}(b, y)$:

$$Q(\alpha, b, X^l) = \sum_{i=1}^{l} \mathscr{L}\left(\underbrace{\underbrace{\sum_{t=1}^{T-1} \alpha_t b_t(x_i)}_{f_{T-1,i}} + \alpha_T b_T(x_i)}_{f_{T,i}}, y_i\right) \to \min_{\alpha, b}$$

Введем обозначения: $f_{T-1} = (f_{T-1,i})_{i=1}^{l}$ - по-сути вектор значений ансамбля первых $T-1$ базовых алгоритмов на всей обучающей выборке, т.е. $f_{T-1} \in \mathbb{R}^l$. Будем называть f_{T-1} вектором текущего приближения. $f_T = (f_{T,i})_{i=1}^{l}$ - вектор следующего приближения. $f_T \in \mathbb{R}^l$. Таким образом добавляя еще один базовый алгоритм, мы осуществляем переход от f_{T-1} к вектору f_T. Такой переход похож на градиентный метод минимизации.

На данном этапе Q зависит от α и от b. В введенных обозначениях Q это функция то вектора f_T. Отвлечемся от того, как этот вектор считается и рассмотрим задачу:

$$Q(f) \to \min, \quad f \in \mathbb{R}^l$$

Будем оптимизировать градиентными методами:

1. $f_0 :=$ начальное приближение
2. $f_{T,i} = f_{T-1,i} - \alpha g_i, \qquad i = 1, \ldots, l$

где $g_i = \dfrac{\partial \mathscr{L}(f_{T-1}, y_i)}{\partial f_i}$ - компонента вектора градиента, α - градиентный шаг.

Наблюдение: это очень похоже на одну итерацию бустинга:

$$f_{T,i} = f_{T-1,i} + \alpha b(x_i), \quad i = 1, \ldots, l$$

Возникает следующая идея: будем искать такой базовый алгоритм b_T, чтобы вектор $(b_T(x_i))_{i=1}^{l}$ приближал вектор антиградиента $-(g_i)_{i=1}^{l}$:

$b_T = \arg\min\limits_{b} \sum\limits_{i=1}^{l} (b_T(x_i) + g_i)^2$

Algorithm 2 GB

Input : training set X^l; T - number of basic algorithms
Output : basic algorithms b_t and their weights α_t, $t = 1, \ldots, T$

1: initialize $f_i = 0$, $i = 1, \ldots, l$
2: for $t = 1, \ldots, T$ do
3: $\quad b_t = \arg\min\limits_{b \in \mathscr{B}} \sum\limits_{i=1}^{l} (b(x_i) + \mathscr{L}'(f_i, y_i))^2$
4: $\quad \alpha_t = \arg\min\limits_{\alpha > 0} \sum\limits_{i=1}^{l} \mathscr{L}(f_i + \alpha b_t(x_i), y_i)$
5: $\quad f_i = f_i + \alpha_t b_t(x_i)$, $i = 1, , l$
6: end for

Return $\{\alpha_t\}_{t=1}^{T}, \{b_t\}_{t=1}^{T}$

---

7 XGBoost

Рассмотрим теперь частный случай градиентного бустинга - градиентный бустинг над решающими деревьями. Деревья классификации и регрессии (CART):

$$b(x, w) = \sum_{k \in K} w_k B_k(x)$$

где $B_k(x) = [x \in R_k]$ - x попадает в область покрываемую листом R_k, w_k - значение в листе k, K - множество листов дерева. Для каждого x одно и только одно слагаемое не равно нулю.

Критерий качество с суммой L_0 и L_2 регуляризаторов:

$$Q(w) = \sum_{i=1}^{l} \mathscr{L}(a(x_i) + b(x_i, w), y_i) + \gamma|K| + \frac{\lambda}{2}\sum_{k\in K} w_k^2 \to \min_w$$

где $a(x_i) = \sum\limits_{t=1}^{T-1} \alpha_t b_t(x_i)$ - ранее построенная асть ансамбля.

Приблизим $\mathscr{L}(a+b, y) \approx \mathscr{L}(a, y) + b\mathscr{L}'(a, y) + \frac{b^2}{2}\mathscr{L}''(a, y)$. Тогда получим:

$$Q(w) = \sum_{i=1}^{l} \left[\mathscr{L}(a(x_i), y_i) + g_i b_i + \frac{1}{2} h_i b_i^2\right] + \gamma|K| + \frac{\lambda}{2}\sum_{k\in K} w_k^2 \to \min_w$$

Поскольку $L(a(x_i), y_i)$ не зависит от нового базового алгоритма, то его можно выкинуть. Тогда имеем:

$$Q(w) = \sum_{i=1}^{l} \left[g_i b_i + \frac{1}{2} h_i b_i^2\right] + \gamma|K| + \frac{\lambda}{2}\sum_{k\in K} w_k^2 \to \min_w$$

где $b_i = \sum\limits_{k\in K} w_k B_k(x_i)$, $g_i = \mathscr{L}'(a(x_i), y_i)$, $h_i = \mathscr{L}''(a(x_i), y_i)$ (производные по первому аргументу).

Хотим теперь приравнять к нулю градиент $\frac{\partial Q(w)}{\partial w} = 0$. Рассмотрим компонентру градиента $\frac{\partial Q(w)}{\partial w_k}$:

$$\frac{\partial Q}{\partial w_k} = \sum_{i=1}^{l}\left[g_i \frac{\partial}{k} b_i\right] + \frac{1}{2}\sum_{i=1}^{l}\left[h_i \frac{\partial}{k} b_i^2\right] + \lambda w_k$$

$$\frac{\partial}{\partial w_k} b_i = \frac{\partial}{\partial w_k}\sum_{s\in K} w_s B_s(x_i) = B_k(x_i)$$

$$\frac{\partial}{\partial w_k} b_i^2 = \frac{\partial}{\partial w_k}\left[\sum_{s\in K} w_s B_s(x_i)\right]^2 = \frac{\partial}{\partial w_k}\sum_{s\in K} w_s^2 B_s(x_i) = 2 w_k B_k(x_i)$$

Переход от квадрата суммы к сумме квадратов произошел по причине того, что произведение $B_s(x_i)B_m(x_i) = 0$ при $s \neq m$. Если бы это было не так, то это значило бы, что один и тот же объект оказался одновременно в двух разных листах дерева, что противоречит построению дерева. Выразим теперь w_k из условия $\frac{\partial Q(w)}{\partial w_k} = 0$:

$$w_k = -\frac{\sum\limits_{i=1}^{l} g_i B_k(x_i)}{\lambda + \sum\limits_{i=1}^{l} h_i B_k(x_i)}$$

Если теперь подставить w_k в $Q(w)$, то можно вывести критерий ветвления для дерева:

$$\Phi(B_1, \ldots, B_k) = -\frac{1}{2}\sum_{k\in K} \frac{\left(\sum\limits_{i=1}^{l} g_i B_k(x_i)\right)^2}{\lambda + \sum\limits_{i=1}^{l} h_i B_k(x_i)} + \gamma|K| \to$$

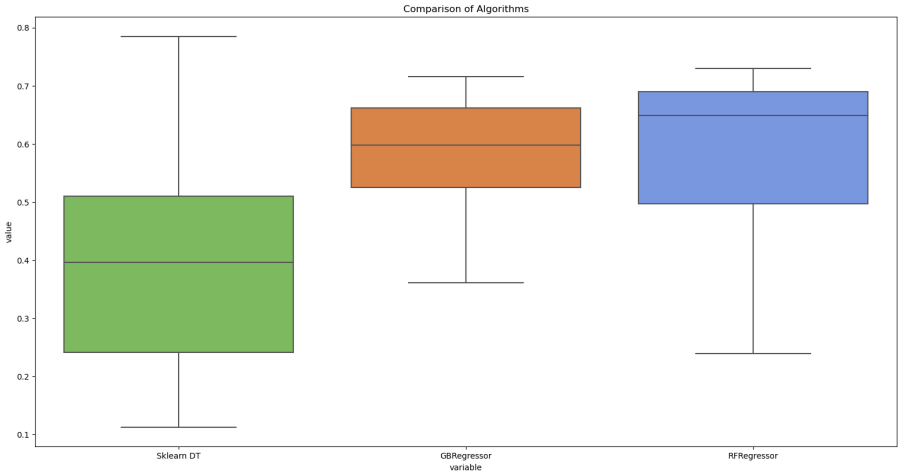
Преимущества XGBoost:

1. L_2 регуляризация сокращает переобучение
2. L_0 регуляризация упрощает деревья (pruning)
3. Также как и общий GB допускает произвольную функцию потерь
4. Быстрая реализация засчет аналитических формул

8 Сравнение, анализ и вывод

Чтобы на практике убедиться в том, что действительно ансамбли базовых алгоритмов, лучше чем один алгорит, рассмотрим по отдельности задачи регрессии и классификации.

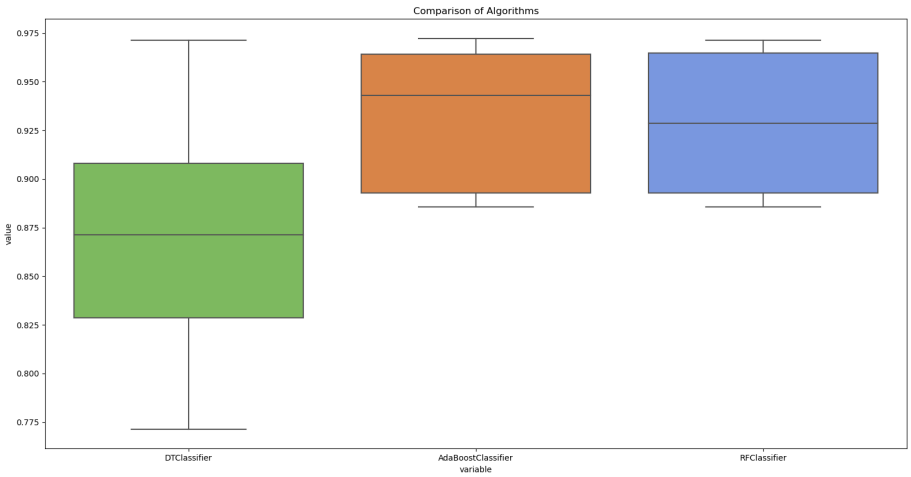
Сначала рассмотрим задачу регрессии и сравним следующие алгоритмы: DecisionTreeRegressor (встроенный в модуль sklearn.tree), Random Forest, Gradient Boosting. Сравнивать будем на наборе данных Tesla Stock Price по метрике r2-score. Тестировать алгоритмы будем с помощью KFold кросс-валидации и построим грифики boxplot, чтобы проанализировать стабильность алгоритма и то как, они отрабатывают в среднем.



Легко видеть, что медианы предсказаний бустинга и бэггинга над деревьями (в данном случае использовалось по 40 базовых деревьев), как минимум в полтора раза выше, чем медиана предсказаний одного лишь только решающего дерева. Также отчетливо видно, что разброс предсказаний в ансамблях значительно ниже, чем у одного решающего дерева, что несомненно является преимуществом ансамблирования.

Рассмотрим теперь задачу классификации на 2 класса на примере набора данных Ionosphere Data Set. Нам нужно будет классифицировать данные на хорошие («g») и плохие («b»). Хороший («g») радар показывает некоторые признаки отражения в ионосфере, тогда как плохой («b») радар — это те сигналы, которые не проходят через ионосферу.

Сравнение будем проводить по тем же графикам boxplot, построенным по KFold кросс-валидации, анализируя метрику accuracy. Рассмотрим Алгоритмы DecisionTreeClassifier (встроенный в модуль sklearn.tree), RandomForestClassifier и AdaBoostClassifier.



Наблюдаем ту же картину, что и в задаче регрессии: медианы точности предсказаний ансамблей лучше, чем у одного решающего дерева, а также "скученность"предсказаний ансамблей гораздо лучше, что говорит о лучшей стабильности и устойчивости ансамблей.

В итоге можно заключить, что теоретические выкладки, возможно, даже с некоторыми ограничительными предположениями (например, предположение и некоррелированности базовых алгоритмов) полностью оправдываются на практике, что видно из примеров. Более того, как видно из теоретических рассуждений, иногда удается вывести почти аналитическое решение задачи оптимизации (например, AdaBoost), что очень сильно ускоряет такие алгоритмы и на практике мы почти бесплатно получаем более стабильный и точный алгоритм, чем один базовый.

9 Список литературы

1. Christopher M.Bishop, Pattern Recognition and Machine Learning, Springer Science+Business Media, LLC, 2006. -758 с.

2. Воронцов К. В. Лекции по алгоритмическим композициям, МГУ, 2012

3. Воронцов К.В. Продвинутые методы ансамблирования, МФТИ, 2021

4. Yandex School of Data Analysis, ML-Handbook, 2021