1. Моделі засновані на деревах

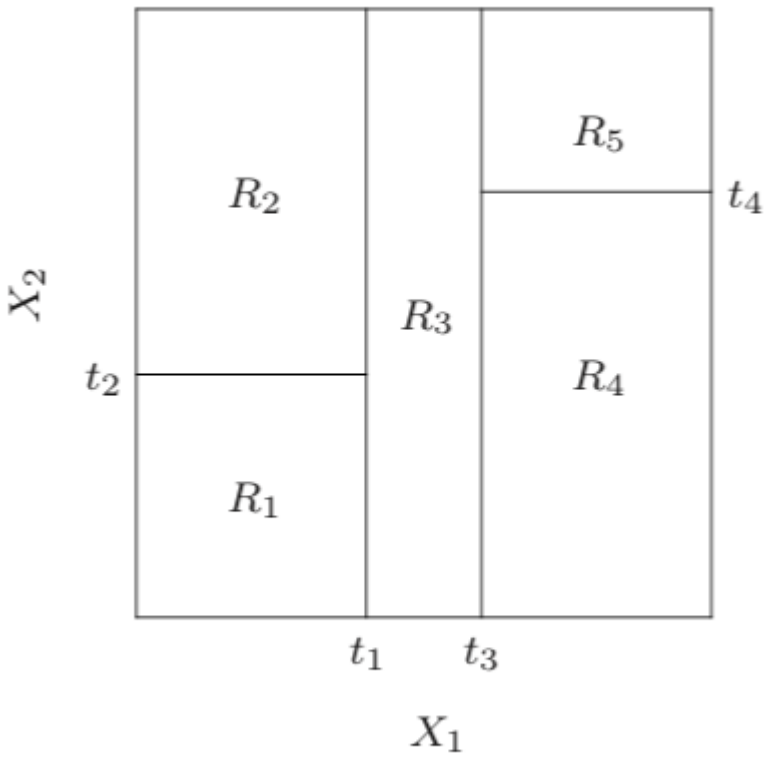
Як завжди розглядає набір регресорів $X = (X^1, \ldots, X^p)$ зі значеннями у певному просторі $\mathcal{X}$. Нехай Y - це відгук, причому відгук може бути як числовим (тоді ми говоримо про регресійні дерева) так і факторним (тоді ми говоримо про дерева прйняття рішень, або класифікаційні дерева).

Почнемо з регресійних дерев, тобто нехай Y - є числовою змінною, і для іллюстрації методу припустимо, що $p = 2$, тобто ми маємо два (числових) регресори: X^1 та X^2.

Для початку розділимо простір на два регіони, та змоделюємо відгук середнім Y у кожному з регіонів. Ми вибираємо змінну та точку поділу щоб досягти найкращої підгонки.

Тоді один з цих регіонів розділяється на два інших регіони та цей процес продовжується поки не виконається якесь зупиняюче правило.

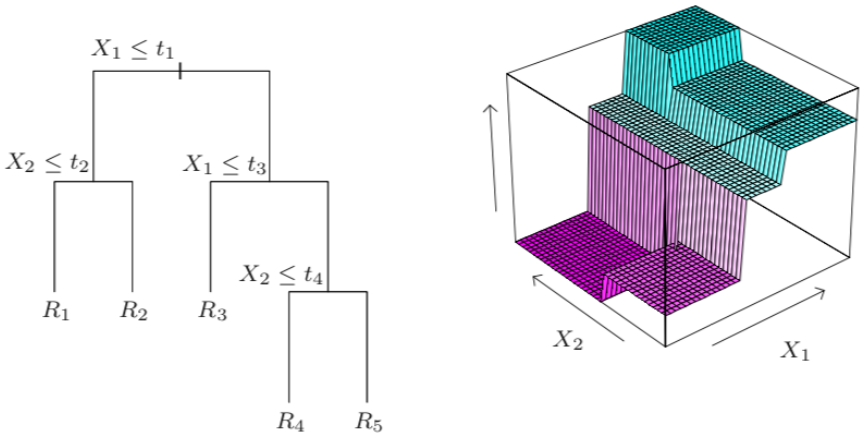
Наприклад, як показано на малюнку, ми спочатку розділимо регіон при $X_1 = t_1$, потім регіон $X_1 \leq t_1$ буде розділено прямою $X_2 = t_2$ та регіон $X_1 > t_1$ прямою $X_1 = t_3$. Нарешті регіон $X_1 > t_3$ буде роздялено $X_2 = t_4$. У результаті отримаємо розбиття простору на регіони $R_1, \ldots, R_5$ як показано на малюнку.



Відповідна регресійна модель прогнозує Y константою c_m в регіоні R_m:

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I_{(X_1,X_2)\in R_m}.$$

Таку модель можна уявити як вирішуюче дерево, що показано на малюнку. Також на малюнку зображена поверхня для Y:



Тепер повернемось до питання - як виростити регресійне дерево. Наші дані складаються з p регресорів та відгутку, всього N спостережень, які ми позначимо (x_i, y_i), $i = 1, \ldots, N$, з $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. Алгоритм повинен автоматично вирішити по яким змінним проводити розділення та по яким точкам а також топологію (форму) дерева. Спершу припустимо, що ми маємо розбиття на M регіонів $R_1, \ldots, R_M$ та ми моделюємо відгук як константу у кожному регіоні:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

Якщо застосувати критерій мінімізації заснований на сумі квадратів $\sum(y_i - f(x_i))^2$ то легко побачити, що найкращі $\hat{c}_m$ це просто середні значення y_i в регіоні R_m:

$$\hat{c}_m = ave(y_i|x_i \in R_m).$$

Однак знаходження найркащого розбиття з точки зору мінімальної суми квадратів в цілому недосяжне з обчислювальної точки зору. Отже при будемо використовувати жадібний алгоритм. Стартуючи з усіх даних, визначимо розділяючу змінну j та розділяючу точку s, та визначимо пару пів-площин:

$$R_1(j, s) = \{X|X_j \le s\}, \text{та } R_2(j, s) = \{X|X_j > s\}.$$

Далі ми будемо шукати розділяючу змінну j та розділяючу точку s так, щоб розв'язати:

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$

Для будь-якого вибору j та s, внутрішня мінімізація розв'язується як:

$$\hat{c}_1 = ave(y_i|x_i \in R_1(j, s)), \text{ та} \hat{c}_2 = ave(y_i|x_i \in R_2(j, s)).$$

Для кожної розділяючої змінної, визначення розділяючої точки s може бути зроблено дуже швидко і тому скануючи всі вхідні дані визначення найкращої пари (j, s) є можливим з обчислювальної точки зору.

Кращою стратегією буде ростити велике дерево T_0 і зупиняти процес розділення лише коли деякий мінімальний розмір вузла (наприклад 5) буде досягнуто. Тоді таке велике дерево слід урізати використовуючи cost-complexity pruning, яке ми зараз розглянемо.

Визначимо піддерево $T \in T_0$ як будь-яке дерево, що може бути отримано шляхому урізання T_0, або згортання будь-якої кількості внутрішніх вузлів (не крайніх листків). Проіндексуємо термінальні (крайні) вузли індексом m, так що вузол m представляє регіон R_m. Нехай $|T|$ позначає кількість термінальних вузлів у T. Позначимо:

$$N_m = \#\{x_i \in R_m\},$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i,$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2,$$

визначимо ціновий критерій складності (cost complexity criterion):

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|.$$

Ідея полягає у тому, щоб для кожного α піддерево $T_\alpha \in T_0$ мінімізувало $C_\alpha(T)$. Параметр налаштування (tuning paramter) $\alpha > 0$ управляє балансом між розміром дерева та його якістю прогнозування (goodness of fit). Великі значення α приведуть до малих дерев T_α, та навпаки. Очевидно, що з $\alpha = 0$ розв'язком буде дерево T_0.

Для кожного α можна показати, що існує єдине найменше піддерево T_α таке що мінімізує $C_\alpha(T)$. Щоб знайти T_α ми використаємо урізання найслабшого зв'язку (weakest link pruning): ми згортаємо внутрішній вузол, що продукує найменше повузлове збільшення у величні: $\sum_m N_m Q_m(T)$ та продовжуємо до тих пір поки не дійдемо до дерева з одним вузлом. Це дає нам скінченну кількість піддерев, і можна показати що вона містить T_α.

Оцінку α отримують з кросс-валідації, $\hat{\alpha}$ вибирають таким щоб мінімізувати кросс-валідовану суму квадратів. Наше фінальне дерево - це $T_{\hat{\alpha}}$.

Класифікаційні дерева

Якщо відгук має факторний вигляд зі значенням в $\{1, \ldots, K\}$ єдиною зміною що потрібна в наведеному вище алгоритмі - це критерій для розділення та урізання дерева.

Для регресії ми використовували середньоквадратичну міру забрудненості вузла $Q_m(T)$ яка не є підходящою для класифікації. Для вузла m що представляє регіон R_m з N_m спостереженнями, покладемо:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

пропорцію значень з класу k у вузлі m. Ми класифікуємо спостереження до класу $k(m) = argmax_k \hat{p}_{mk}$ - як головний клас у вузлі m. Можуть розглядатися різні міри

$Q_m(T)$:

$$\begin{aligned} &\text{Помилка класифікації:} && \frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1-\hat{p}_{mk}(m). \\ &\text{Індекс Джині:} && \sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}). \\ &\text{Кросс-ентропія:} && -\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}. \end{aligned} \tag{1}$$

Для двох класів, якщо p це пропорція в другому класі, ці три міри - $1-\max(p,1-p), 2p(1-p), -p\log p-(1-p)\log(1-p)$, відповідно. Вони показані на малюнку нижче. Всі три схожі, але крос-ентропія та індекс Джині диференційовані, а отже більше підходять для чисельної оптимізації.

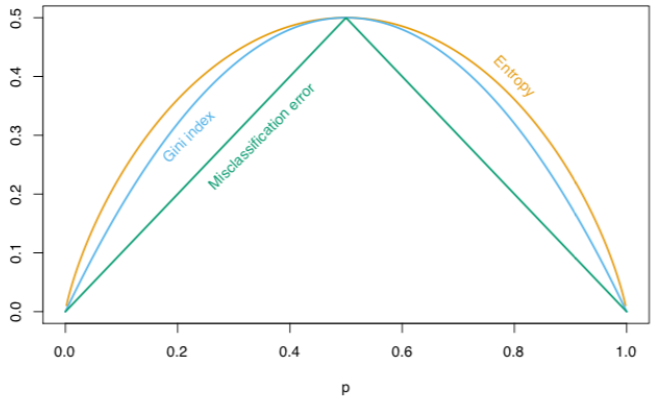


FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

Порівнюючи оптимізацію для регресії та для класифікації ми бачимо що потрібно зважити міру Q кількостями N_{m_L} та N_{m_R} - спостережень у двох дочірніх вузлах створених розділенням вузла m.

На додачу до цього, крос-ентропія та індекс Джині більш чутливі до змін у вузлових ймовірностях ніж похибка класифікації. Наприклад у задачі з двома класами з 400 спостереженнями в кожному класі (позначимо як (400,400)), припустимо відбулось розділення вузла (300,100) та (100,300), в той час як інше можливе розділення (200,400) та (200,0). Обидва розділення продукують похибку класифікації у 0.25, але друге розділення генерує чистий вузол і тому більш бажаний. І крос-ентропія і Індекс Джині краще використовувати коли вирощувати дерево. Для урізання методом cost-complexity будь-яка з трьох мір може використовуватися, але типово використовується похибка класифікації.

Індекс Джині може бути інтерпритовано двома цікавими способами. Замість того щоб класифікувати спостереження до головного класу у вузлі, ми можемо класифікувати їх до класу k із ймовірністю $\hat{p}_{mk}$. Тоді тренувальна похибка такої класифікації на даному вузлі складатиме: $\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'}$ - це індекс Джині.

Аналогічно, якщо закодувати кожне спостереження як 1 для класу k та 0 інакше, то дисперсія по такому вузлу для такого 0-1 відгука буде $\hat{p}_{mk}(1-\hat{p}_{mk})$. Просумувавши по всім класам k знову отримаємо індекс Джині.

2. Алгоритми ансамблювання.

Існує два фундаментально різних підходи до ансамблювання - **послідовний**, коли наступний алгоритм вибирається з урахуванням результатів попереднього, та **паралельний** коли декілька алагоритмів тренуються одночасно. Ми розпочнемо з послідовного підходу.

Далі ми розглянемо процедуру boosting-a. Вона полягає у тому щоб натренувати хороший алгоритм як комбінацію великої кількості простих але "дешевих" алгоритмів.

Почнемо з бінарної класифікації. Нехай ми маємо справу, з фазовим простором $\mathcal{X}$, на якому задані незалежні однаково розподілені випадкові величини з розподілом P та справжньою істинною функцією f. Уявімо собі, що простір $\mathcal{X}$ розділено на три частини $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3$ кожна з яких має вагу в третину розподілу, і алгоритм з випадковим вибором працює з 50% відсотковою ймовірністю. Ми б хотіли отримати хороший класифікатор (тобто з нульовою похибкою) для цієї проблеми, але маємо "поганий" класифікатор який правильно класифікує спостереження в частинах $\mathcal{X}_1$ та $\mathcal{X}_2$ але неправильно в $\mathcal{X}_3$, а отже має похибку класифікації 1/3. Назвемо цей алгоритм h_1. Очевидно, h_1 не є бажаним класифікатором.

Ідея boosting-у полягає у тому щоб виправити помилки, які допускає алгоритм h_1. Ми можемо спробувати вивести новий розподіл P' з P який зробить помилки h_1 більш очевидними, тобто буде зосереджений в основному на $\mathcal{X}_3$. Тоді ми зможемо натренувати новий класифікатор з P', назвемо його h_2. Припустимо, що h_2 теж є слабким класифікатором, і наприклад лише класифікує спостереження з $\mathcal{X}_1$ та $\mathcal{X}_3$, та неправильно класифікує спостережння з $\mathcal{X}_2$.

Тоді скомбінувавши класифікатори h_1 та h_2 певним чином (пізніше ми покажемо, як саме) ми отримаємо новий класифікатор який корректно класифікуватиме на $\mathcal{X}_1$ та можливо матиме певні помилки на $\mathcal{X}_2$ та $\mathcal{X}_3$.

Далі ми виведемо деякий новий розподіл P'' щоб зробити похибки комбінованого класифікатора більш очевидними, і натренуємо класифікатор h_3 з цим розподілом, так що h_3 коректно класифікуватиме в $\mathcal{X}_2$ та $\mathcal{X}_3$. Тоді комбінуючи h_1, h_2 та h_3 ми отримаємо бажаний класифікатор, оскільки в кожній частині простору $\mathcal{X}_1$, $\mathcal{X}_2$ та $\mathcal{X}_3$, як мінімум два класифікатори здійснять коректну класифікацію.

Наступний алгоритм показує процедуру ансаблювання у загальному випадку, однак функції AdjustDiscribution та Combine ще потрібно визначити.

Алгоритм

Вхід: Початковий розподіл P, Початковий навчальний алгоритм $\mathcal{L}$, кількість раундів навчання T.

Процес:

1. $P_1 = P$ - Ініціалізація розподілу.
2. **for** $t = 1, \ldots, T$:
3. $\quad h_t = \mathcal{L}(P_t)$; - тренуємо слабкий алгоритм з розподілом P_t
4. $\quad \varepsilon_t = P_t(h_t(X) \neq f(X))$; - обчислюємо помилку h_t
5. $\quad P_{t+1} = \text{AdjustDistribution}(P_t, \varepsilon_t)$
6. **end**;

Вихід: $H(x) = \text{Combine}(\{h_1(x), \ldots, h_t(x)\})$

2.1. **Алгоритм AdaBoost.** Розглянемо бінарну класифікацію на класах $\{-1, +1\}$. Одна з версій алгоритму AdaBoost полягає у мінімізації експотенційної функції втрат:

$$\mathcal{L}(h|P) = E\left[e^{-f(X)h(X)}\right],$$

використовуючи адитинвну вагову комбінацію слабких алгоритмів:

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

Експоненційна функція втарта використовується, оскільки вона дає елегантну і просту рекурентну формулу (update formule), є узгодженною із задачею мінімізації помилки класифікації а також, як може бути показано, пов'язана зі стандартним відношенням правдоподібності.

H отримується шляхом ітеративної геренації h_t та α_t. Перший слабкий класифікатор h_1 згенеровано з використаннями слабкого алгоритму з початковим розподілом. Коли класифікатор h_t згенеровано з розподілом P_t, його вага α_t має бути визначена так, щоб мінімізувати експотенційну втрату:

$$\mathcal{L}(\alpha_t h_t|P_t) = \mathbb{E}_t\left[e^{-f(X)\alpha_t h_t(X)}\right] = \mathbb{E}_t\left[e^{-\alpha_t}\mathbb{I}_{f(X)=h_t(X)} + e^{\alpha_t}\mathbb{I}_{f(X)\neq h_t(X)}\right] =$$

$$e^{-\alpha_t}\mathbb{P}_t(f(X) = h_t(X)) + e^{\alpha_t}\mathbb{P}_t(f(X) \neq h_t(X)) = e^{-\alpha_t}(1-\varepsilon_t) + e^{\alpha_t}\varepsilon_t,$$

де $\varepsilon_t = \mathbb{P}_t(f(X) \neq h_t(X))$. Щоб отримати оптимальне α_t, прирівняємо похідну експотенційної функції втрат до нуля:

$$\frac{\partial \mathcal{L}(\alpha_t h_t|P_t)}{\partial \alpha_t} = -e^{-\alpha_t}(1-\varepsilon_t) + e^{\alpha_t}\varepsilon_t = 0,$$

тоді отримаємо розв'язок:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right).$$

Новий розподіл p_{t+1} буде мати вигляд:

$$p_{t+1}(x) = \frac{p(x)e^{-f(x)H_t(x)}}{\mathbb{E}e^{-f(X)H_t(X)}} = \frac{p(x)e^{-f(x)H_{t-1}(x)}e^{-f(x)\alpha_t h_t(x)}}{\mathbb{E}e^{-f(X)H_t(X)}} =$$

$$p_t(x)e^{-f(x)\alpha_t h_t(x)}\frac{\mathbb{E}e^{-f(X)H_{t-1}(X)}}{\mathbb{E}e^{-f(X)H_t(X)}}.$$

Таким чином алгорим AdaBoost має вигляд:
Вхід: Набір даних $D = \{(X_1, y_1), \ldots, (X_m, y_m)\}$; Базовий алгоритм навчання L, кількість раундів навчання: T. Процес:
1. $p_1(X) = 1/m$ - Ініціалізація розподілу.
2. **for** $t = 1, \ldots, T$:
3. $h_t = L(D, P_t)$; - тренуємо класифікатор h_t з розподілом P_t
4. $\varepsilon_t = P_t(h_t(X) \neq f(X))$; - обчислюємо помилку h_t
5. **if** $\varepsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$; Визначаємо ваги h_t
7. $p_{t+1}(X) = \frac{p_t(X)}{Z_t} \times \left(e^{-\alpha_t}I_{h_t(X)=f(X)} + e^{\alpha_t}I_{h_t(X)\neq f(X)}\right) = \frac{p_t(X)e^{-\alpha_t f(X)h_t(X)}}{Z_t}$, де Z_t це нормуючий множник.
6. **end**;
Вихід: $H(x) = \operatorname{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right)$.

2.2. **Приклад.** Розглянемо роботу алгоритму AdaBoost на прикладі класифікації XOR. Нехай маємо 4 точки:

$$z_1 = (1, 0), y_1 = 1$$
$$z_2 = (-1, 0), y_1 = 1$$
$$z_3 = (0, 1), y_1 = -1$$
$$z_4 = (0, -1), y_1 = -1$$

Розглянемо також 8 слабких класифікаторів:

$$h_1(x) = \mathbb{I}(x_1 > -0.5) - \mathbb{I}(x_1 \leq -0.5), h_2 = -\mathbb{I}(x_1 > -0.5) + \mathbb{I}(x_1 \leq -0.5),$$
$$h_3(x) = \mathbb{I}(x_1 > +0.5) - \mathbb{I}(x_1 \leq +0.5), h_4 = -\mathbb{I}(x_1 > +0.5) + \mathbb{I}(x_1 \leq +0.5),$$
$$h_5(x) = \mathbb{I}(x_2 > -0.5) - \mathbb{I}(x_1 \leq -0.5), h_6 = -\mathbb{I}(x_2 > -0.5) + \mathbb{I}(x_1 \leq -0.5),$$
$$h_7(x) = \mathbb{I}(x_2 > +0.5) - \mathbb{I}(x_1 \leq +0.5), h_8 = -\mathbb{I}(x_2 > +0.5) + \mathbb{I}(x_2 \leq +0.5).$$

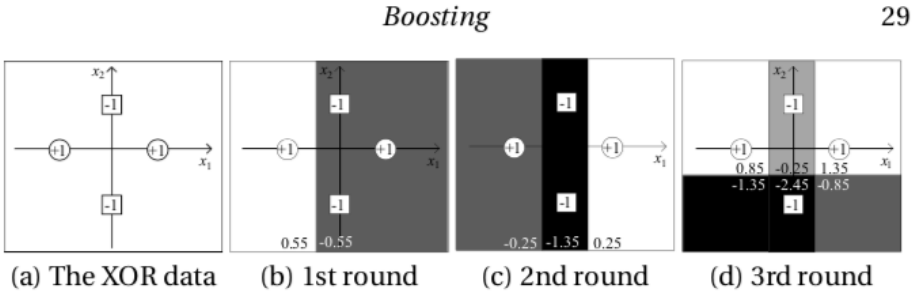


FIGURE 2.3: AdaBoost on the XOR problem.

Нехай наш алгоритм $\mathcal{L}$ обчислює всі 8 значень $h_i(x)$ із заданим розподілом, і обирай той, що має найменшу похибку. Якщо таких декілька то він вибирає навмання.

Запишемо тепер покроково роботу алгоритму AdaBoost:

1. На першому кроці найменшу похибку класифікації в 0.25 дають алгоритми: h_2, h_3, h_5, h_8. Припустимо $\mathcal{L}$ вибирає h_2 як класифікатор. Тоді маємо неправильну класифікацію для z_1, тобто похибка $\varepsilon_1 = P_0(h_1(X) \neq f(X)) = P_0(z_1) = 0.25$. Тоді обчислимо вагу α_1:

$$\alpha_1 = \frac{1}{2}\ln(0.75/0.25) = 0.5 * \ln(3) \approx 0.55.$$

Обчислимо тепер ймовірність p_1, з точністю до нормованого множника Z_t:

$$p_1(z_1) \propto 0.25e^{\alpha_t} = 0.25\sqrt{3} \approx 0.433,$$
$$p_1(z_2) \propto 0.25e^{-\alpha_t} = \frac{0.25}{\sqrt{3}} \approx 0.1443,$$
$$p_1(z_3) = p_1(z_4) = p_1(z2) \propto 0.1443.$$

Тоді нормуючий множник: $Z_1 \approx 0.8629$. Отже: $P_1 \approx (0.5018, 0.1673, 0.1673, 0.1673)$

2. Обчислимо тепер похибки для кожного з алгоритмів h_i, $i = 1, 8$ з новим розподілом P_1;

$$\varepsilon(h_1) = P_1(h_1(X) \neq f(X)) = p_1(\{z_2, z_3, z_4\}) \approx 0.5,$$
$$\varepsilon(h_2) = P_1(h_2(X) \neq f(X)) = p_1(z_1) \approx 0.5018,$$
$$\varepsilon(h_3) = P_1(h_3(X) \neq f(X)) = p_1(z_2) \approx 0.1673,$$

$$\varepsilon(h_4) = P_1(h_4(X) \neq f(X)) = p_1(\{z_1, z_3, z_4\}) \approx 0.832,$$
$$\varepsilon(h_5) = p_1(z_3) \approx 0.1673,$$
$$\varepsilon(h_6) = p_1(\{z_1, z_2, z_4\}) \approx 0.832,$$
$$\varepsilon(h_7) = p_1(\{z_1, z_2, z_3\}) \approx 0.832,$$
$$\varepsilon(h_8) = p_1(z_4) \approx 0.1673.$$

Таким чином найменшу похибку дають класифікатори h_3, h_5, h_8. Нехай $\mathcal{L}$ вибирає h_3. тоді:

$$\varepsilon_2 \approx 0.1673,$$
$$\alpha_2 \approx 0.5 \ln(0.83273/0.16727) = 0.5 \ln(4.97836) = 0.5 * 1.6051 = 0.80255.$$

Тоді

$$P_2 \propto (0.2249, 0.37328, 0.075, 0.075), Z_2 = 0.74818$$

3. На наступному кроці найменшу похибку дадуть алгоритми h_5 та h_8, нехай $\mathcal{L}$ вибирає h_5. Тоді:

$$\varepsilon_3 = P_2(h_5(X) \neq f(X)) = p_2(z_3) = 0.075/0.74818 = 0.1,$$
$$\alpha_3 = 0.5 \ln(9) = 1.0986.$$

Тоді класифікатор $H(x)$ має вигляд: $H(x) = \text{sign}(0.55 * h_1(x) + 0.80255 * h_3(x) + 1.0986 * h_5(x))$, він зображений на малюнку під індексом (d) і коректно класифікує всі значення.

3. Паралельні алгоритми ансамблювання.

Тепер ми перейдемо до іншого способу ансамблювання, а саме **паралельного** ансамблювання, або упаковки(?), англ. bagging. Якщо основною ідеєю послідовного ансамблювання було покращення (boost), результатів роботи попереднього алгоритму, то основна ідея паралельного підходу полягає у використанні **незалежності** базових (слабких) алгоритмів очікуючи значного зменшення похибки в результаті комбінування таких незалежних алгоритмів.

Розглянемо для ілюстрації бінарну класифікацію з класами $\{-1, +1\}$. Нехай істина функція f, та кожен базовий класифікатор видає незаелжну похибку ε:

$$P(h_i(X) \neq f(X)) = \varepsilon.$$

Після комбінування T подібних класифікаторів у фінальний класифікатор: $H(x) = \text{sign}\left(\sum_{i=1}^{T} h_i(x)\right)$, ансамбль $H(x)$ видасть похибку лишу коли як мінімум половина всіх базових класифікаторів згенерує похибку. Тоді за нерівністю Хофдінга (Hoeffding inequality), загальна похибка ансамблю рівна:

$$P(H(X) \neq f(X)) = \sum_{k=0}^{[T/2]} \binom{T}{k} (1-\varepsilon)^k \varepsilon^{T-k} \leq e^{-\frac{1}{2}T(2\varepsilon-1)^2}.$$

Звідки явно видно, що загальна похибка спадє експоненційно зі зростанням розміру ансамблю T, та прямує до нуля при $T \to \infty$.

Однак на практиці неможливо досягнути повної незалежності базових алгоритмів оскільки вони натреновані на тій же самій вибірці. Можна досягти меншої залежності базових алгоритмів якщо додати деякої випадковості у процес навчання.

Іншою перевагою паралельного підходу є можливість паралельних обчислень класифікаторів, що може суттєво спростити застосування.

3.1. **Bagging.** Типовим алгоритмом паралельного ансамблювання є bagging, що походить від: **B**ootstrap **AGG**ergat**ING**. Як випливає з назви два ключових компонента це boostrap та aggregation.

Алгоритм bagging був запропонований Breiman, в 1996 році у роботі: Bagging predictors. Machine Learning, 24(2):123–140,

Основна ідея полягає у тому, щоб із заданої вибірки зробити певну кількість неперетинуючихся під-вибірок і натренувати базові алгоритми на кожній під-вибірці. Однак, оскільки розмір вибірки скінчений, такий підхід приведе до малих і нерепрезентативних вибірок і до дуже слабких базових класифікаторів.

Bagging використовує техніку boostrap щоб отримати від-вибірки. Зокрема, для вибірки розміру m генеруються під-вибірки розміру m використовуючи вибір із заміною. Таким чином деякі початкові дані увійдуть у під-вибірку декілька разів а інші не потраплять взагалі. Повторюючи процес T разів отримаємо T-підвибірок розміру m. Після цього на кожній під-вибірці буде натренований базовий класифікатор. Метод bagging-у використовує найбільш популярні стратегії для агрегації, а саме - **голосування** для класифікації, та **усереднення** для регресії.

Алгоритм виглядає наступним чином:
Вхід: Набір даних $D = \{(X_1, y_1), \dots (X_m, y_m)\}$, навчаючий алгоритм L, кількість ітерацій T.
Процес:
1. **for** $t = 1, \dots, T$:
2. $h_t = L(D, P_{bs})$, де P_{bs} - це розподіл на підвибірці отриманій методом бутстрап.
3. **end**
Вихід: $H(x) = \text{argmax}_{y \in \{1,\dots,K\}} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y)$.

Варто зазначити що bootstraping також забезпечує ще одну перевагу для bagging-а. Як було показано Брейманом, якщо задано m тренувальних прикладів, ймовірність що i-тий приклад буде вибрано $0, 1, 2, \dots$ разів приблизно має розподіл Пуассона за $\lambda = 1$, а отже ймовірность того, що i-тий приклад з'явиться хоча б раз рівна: $1 - e^{-1} \approx 0.632$. Іншими словами, для кожного базового класифікатора h що використано в bagging-у, приблизно 36.8% оригінальних тренувальних прикладів які не будуть використані в його тренувальному процесі (ймовірність того, що об'єкт не потрапить у вибірку $(1 - 1/l)^l \to 1/e \approx 0.368$). Якість базового класифікатора може бути оцінена використовуючи **out-of-bag** приклади (ті що не попали у вибірку) і тому таким чином може бути обчислена загальна оцінка похибки ансамблю.

3.2. **Випадковий ліс.** Випадковий ліс (Random Forest), є одним з найбільш поширених алгоритмів машинного навчання який є ефективним на майже всіх типах задач (окрім обробки зображень). Цей алгоритм є типовим прикладом застосування паралельного ансамблювання - bagging-у, який ми розглядали нещодавно.

Оскільки дерева типово позначаються літерою T, то в цьому розділі ми будемо використовувати символ B для позначення кількості паралельних алгоритмів.

Алгоритм має такий вигляд:
1. **for** $i = 1, \dots, B$:
2. Генеруємо бустрап-вибірку Z^* розміру N з тренувальних даних.
3. Вирощуємо випадкове дерево T_i на Z^*, шлахом рекурсивного повтору наступних кроків для кожного термінального вузла поки не буде досягнуто мініміального розміру вузла n_{min}.
4. i. Випадково вибираємо m ознак з p.
5. ii. Вибираємо найкращу змінну та точку розщеплення з m ознак вибраних раніше.
6. iii. Розщеплюємо вузол.

7. **end**

Виводимо результуючий ансамблевий класифікатор. Для регресії це: $\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{i=1}^{B} T_i(x)$.

Для класифікації вибирають клас за який проголосувало більшість дерев.

Емпіричне правило: параметр m вибирають як $\sqrt{p}$ в задачах класифікації або $p/3$ в задачах регерсії (де p- це кількість ознак). Для задач класифікації рекомендується обирати $n_{min} = 1$ а регресії $n_{min} = 5$.

Дослідимо дисперсію та зміщення такого алгоритму. Відомо, що для B н.о.р.в.в. кожна з яких має дисперсію σ^2, їхнє середнє має дисперсію $\frac{\sigma^2}{B}$.

Якщо ж змінні лише однаково розподілені але не обов'язково незалежні з попарною кореляцією ρ, дисперсія середнього має вигляд:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

При збільшенні B другий доданок зникає, але перший залишається, таким чином величина попарної кореляції дерев обмежує переваги усереднення (bagging-у).

Ідея випадкового лісу полягає в тому щоб покращити зменшення дисперсії з використанням bagging-у шляхом зменшення кореляції між деревами, не збільшуючи дисперсію занадто сильно. Цього вдається досягти шляхом випадкового вибору ознак, зокрема:

Перед кожним розщепленням, вибрати $m \leq p$ ознак випадковим чином, як кандидати для розщеплення.

Після того як B таких дерев $\{T(x;\Theta_i)\}_i^B$ вирощено, прогнозуюча функція (для регресії) має вигляд:

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{i=1}^{B} T(x;\Theta_i). \tag{2}$$

Тут Θ_i характеризує i-те випадкове дерево в термінах розділяючих змінних, значеннях у термінальних вузлах, точки розщеплення кожного вузла. Інтуїтивно, зменшення m приведе до зменшення кореляції між будь-якими парами дерев у ансамблі, а тому за формулою для дисперсії зменшить дисперсію середнього.

Взагалі кажучи, далеко не кожен класифікатор може бути покращений таким чином. Виглядає що лише глибоко нелінійні класифікатори як дерева отримують найбільше користі від bagging-а (лінійні класифікатори взагалі не покращуються bagging-ом). Для дерев натренованих на бустрап-вибірках, ρ-типово мале (0.05, або менше), тоді як σ^2 не набагато більша за дисперсію звичайного дерева.

Зауважимо, що зміщення випадкового лісу рівне зміщенню дерева, і при bagging-у майже не змінюється. Таким чином покращення результату досягається лише за рахунок зменшення дисперсії.

На різних задачах boosting та випадкові дерева демонструють схожу результативність. Часто boosting працює трохи краще. Однак перевагою випадкового лісу є те, що він часто показує результати бизькі до оптимальних "з коробки" (out-of-the-box), тобто без суттєвого налаштування.

Випадковий ліс використовується у дослідженнях даних помірної вимірності (наприклад фінансові, медичні, дані що відносяться до електронної комерції, реклами та інші), для даних високої розмірності, таких як зображення або аналіз тексту випадковий ліс часто поступається іншим методам, хоча існують модифікації випадкового лісу для багатовимірних даних також.

Зауважимо, що випадковий ліс є близьким до методу $k-nn$, оскільки дерева часто вирощують або до вичерпання вибірки, або близько до того, що веде до того, що класифікатор передбачає клас близький до того який мають сусіди тестової точки.

На практиці результати RF часто близькі до аналогічних результатів $k - nn$, хоча і можуть бути трохи кращими.