ЛЕКЦІЯ № 11

НАВЧАННЯ З ПІДКРІПЛЕННЯМ ТА АНСАМБЛІ

ПЛАН

1. Навчання з підкріпленням
2. Ансамблі

ЛІТЕРАТУРА

Сайти http://www.mmf.lnu.edu.ua/ar/1739
https://datascience.org.ua/vvedenie-v-reinforcement-learning-ili-obuchenie-s-podkrepleniem/
https://uk.education-wiki.com/machine-learning/

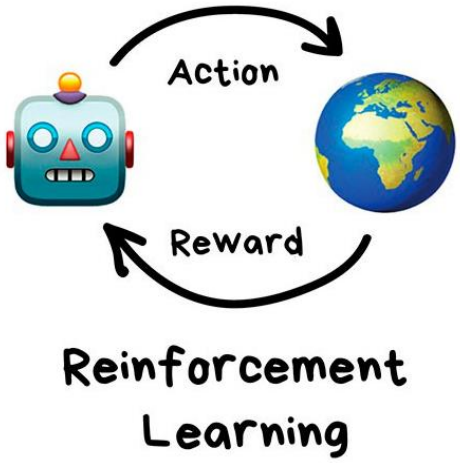
ВСТУП

На попередній лекції ми вивчили що таке машинне навчання, його складові, провели класифікацію методів машинного навчання та розглянули методи класичного навчання.

Мета машинного навчання - передбачити результат за вхідними даними. Чим різноманітніші вхідні дані, тим простіше машині знайти закономірності і тим точніший результат.

Отже, якщо ми хочемо навчити машину, нам потрібні три речі: **дані, ознаки, алгоритми.**

1. НАВЧАННЯ З ПІДКРІПЛЕННЯМ



«Кинь робота в лабіринт і нехай шукає вихід»

Сьогодні використовують для:

Автопілотів автомобілів; Роботів; Ігор; Автоматизованої торгівлі; Управління ресурсами підприємств.

Популярні алгоритми: Q-Learning, SARSA, DQN, A3C, Генетичний Алгоритм

Нарешті ми дійшли до речей, які, начебто, виглядають як справжній штучний інтелект. Багато авторів чомусь розташовують навчання з підкріпленням десь між навчанням з вчителем та без, але я не розумію чим вони схожі. Назвою?

Навчання з підкріпленням використовують там, де задача полягає не в аналізі даних, а у виживанні в реальному середовищі.

Середовищем може бути навіть відеогра. Роботи, які грають в Маріо, були популярні ще років п'ять тому.

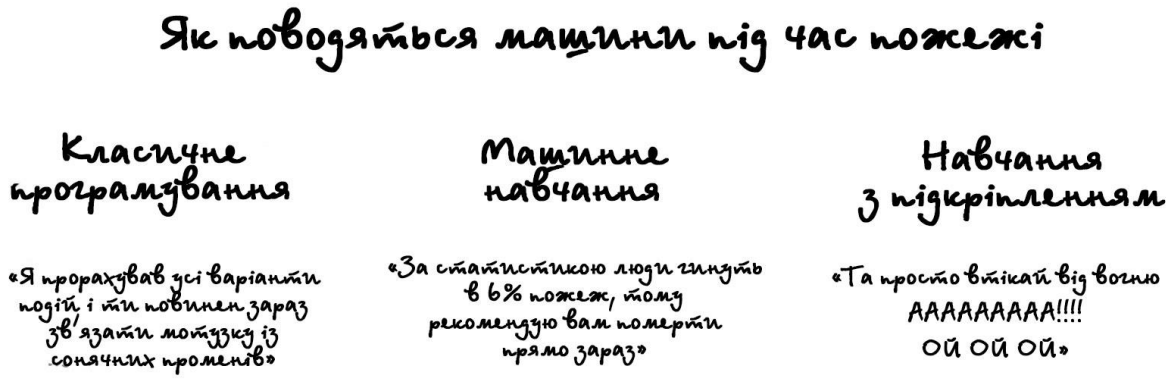
Середовищем може бути навіть реальний світ. Як приклад - автопілот Тесли, який вчиться не збивати пішоходів, або роботи-порохотяги, головне завдання яких - налякати вашого кота з максимальною ефективністю.

Знання про навколишній світ такому роботу можуть бути корисні, але лише для довідки. Не важливо скільки даних він збере, у нього все одно не вийде передбачити всі ситуації. Тому його мета - **мінімізувати помилки**, а не передбачати всі ходи. Робот вчиться виживати в просторі з максимальним зиском: зібраними монетками в Маріо, тривалістю поїздки в Теслі.

Саме виживання в середовищі і є ідеєю навчання з підкріпленням. Давайте кинемо бідного робота в реальне життя, будемо штрафувати його за помилки і нагороджувати за правильні вчинки. На людях норм працює, то ж чому б на і роботах не спробувати.

Розумні моделі роботів-порохотягів і самокеровані автомобілі навчаються саме так: їм створюють віртуальне місто (часто на основі карт справжніх міст), населяють випадковими пішоходами і відправляють вчитися нікого там не вбивати. Коли робот починає добре себе почувати в штучному GTA, його випускають тестувати на реальні вулиці.

Запам'ятовувати саме місто машині не потрібно - такий підхід називається Model-Free. Звичайно, тут є і класичний Model-Based, але в ньому нашій машині довелося б запам'ятовувати модель всієї планети, всіх можливих ситуацій на всіх перехрестях світу. Таке просто не працює. *У навчанні з підкріпленням машина не запам'ятовує кожен рух, а намагається узагальнити ситуації, щоб виходити з них з максимальною вигодою.*



Процес Reinforcement Learning

Уявімо агента, який вчиться грати в Super Mario Bros як робочий приклад. Процес навчання з підкріпленням може бути змодельований як цикл, який працює наступним чином:

Наш агент отримує стан S0 із середовища (у нашому випадку ми отримуємо перший кадр нашої гри (стан) з Super Mario Bros (середа))

На підставі цього стану агент S0 виконує дію A0 (наш агент буде рухатися вправо)

Середовище перетворюється на новий стан S1 (новий кадр)

Середовище дає деяку винагороду агенту R1 (не мертвий: +1)

Цей цикл Reinforcement learning виводить послідовність із станів, дій та нагород.

Мета агента - максимізувати очікувану сукупну винагороду.

Центральна ідея гіпотези про винагороду

Чому мета агента — максимізувати очікувану сукупну винагороду?

Навчання з підкріпленням ґрунтується на ідеї гіпотези про винагороду. Всі цілі можуть бути описані шляхом максимізації очікуваної сукупної винагороди.

Ось чому в Reinforcement Learning, щоб мати кращу поведінку, ми повинні максимізувати очікувану сукупну винагороду.

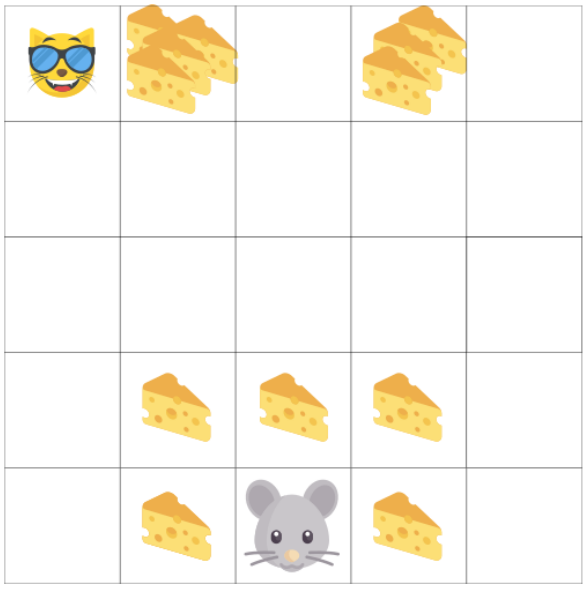
Сукупна винагорода на кожному тимчасовому кроці t може бути записана як:

$$G_t = R_{t+1} + R_{t+2} + \dots$$

що еквівалентно

$$G_t = \sum_{k=0}^{T} R_{t+k+1}$$

Насправді, ми не можемо просто додавати такі нагороди. Нагороди, які приходять раніше (на початку гри) більш імовірні, тому що вони більш передбачувані, ніж довгострокова майбутня винагорода.



Припустимо, ваш агент – це маленька мишка, а ваш супротивник – кішка. Ваша мета – з'їсти максимальну кількість сиру перед тим, як бути з'їденим кішкою.

Як видно на діаграмі, більш ймовірно, з'їсти сир поряд з вами, ніж сир поряд з кішкою (що ближче ми до кішки, то вона небезпечніша).

Як наслідок, нагорода поруч із кішкою, навіть якщо вона буде більшою (більше одного сиру), буде знецінена. Ми не впевнені, що зможемо це з'їсти.

Щоб обчислити знецінення винагороди, ми діємо так:

Ми називаємо величину знецінення гамою. Вона має бути між 0 і 1.

Чим більше гамма, тим менше знецінення. Це означає, що агент, що навчається, більше піклується про довгострокову винагороду.

З іншого боку, що менше гамма, то більше знецінення. Це означає, що наш агент більше дбає про короткострокову нагороду (найближчий сир).

Наша дисконтована (знецінена) сукупна очікувана винагорода:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} where \gamma \in [0,1)$$

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots$$

Простіше кажучи, кожну нагороду буде дисконтовано гамою до показника тимчасового кроку. Зі збільшенням часового кроку кішка стає ближчою до нас, тому майбутня нагорода стає все менш імовірною.

Епізодичні чи безперервні завдання

Завдання є прикладом проблеми навчання із підкріпленням. Може бути два типи завдань: епізодичні та безперервні.

Епізодичне завдання

У цьому випадку ми маємо початкову точку і кінцеву (кінцевий стан). Це створює епізод: список станів, дій, винагород та нових станів.

Наприклад, як у Super Mario Bros епізод починається при запуску нового Mario і закінчується коли ви вбиті або досягли кінця рівня.

Безперервні завдання

Це завдання, які продовжуються вічно (без стану закінчення). У цьому випадку агент повинен навчитися вибирати найкращі дії та одночасно взаємодіяти з середовищем.

Наприклад, агент, який займається автоматичною торгівлею акціями. Для цього завдання немає початкової точки та стану закінчення. Агент продовжує працювати, доки ми не вирішимо його зупинити.

Метод Monte Carlo чи Temporal Difference Learning?

У нас є два способи навчання:

- Збір винагород у кінці епізоду, а потім розрахунок максимальної очікуваної майбутньої нагороди: підхід Монте-Карло
- Оцінка винагороди на кожному етапі: Temporal Difference Learning

Метод Монте-Карло

Коли епізод закінчується (агент досягає «кінцевого стану»), він дивиться на загальну накопичену винагороду, щоб побачити, наскільки добре він це зробив. При методі Монте-Карло нагороди отримують лише наприкінці гри.

Потім ми розпочинаємо нову гру з додатковими знаннями. Агент приймає найкращі рішення з кожною ітерацією.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

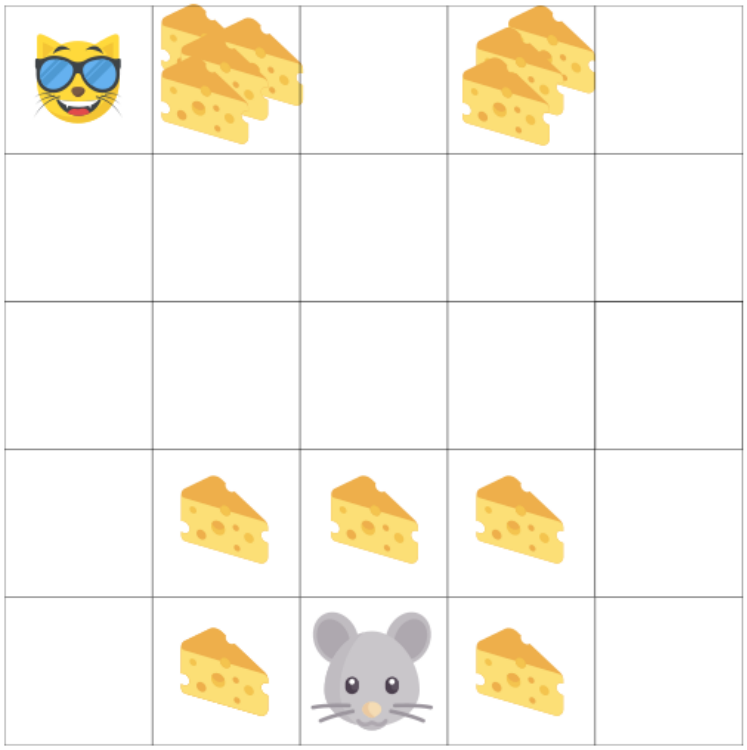
Maximum expected future reward starting at that state

Former estimation of maximum expected future reward starting at that state

learning rate

Discounted cumulative rewards

Наприклад,



якщо розглянути властивості лабіринту:

- Ми завжди починаємо з однієї і тієї ж відправної точки.
- Ми припиняємо епізод, якщо кішка з'їдає нас або ми рухаємось 20 кроків.
- Наприкінці епізоду ми маємо список станів, дій, нагород та нових станів.
- Агент підсумовує загальну суму винагород Gt (щоб побачити, наскільки добре це вийшло).
- Потім він оновить V(st) на основі наведеної вище формули.
- Тоді починається нова гра із цим новим знанням.

Виконуючи все більше і більше епізодів, агент навчиться грати все краще та краще.

Temporal Difference Learning: навчання на кожному часовому кроці

TD Learning, не чекатиме кінця епізоду, щоб оновити максимальну очікувану оцінку винагороди: він оновить свою оцінку значення V для не зупинних станів St, що виникають у цьому досвіді.

Цей метод називається TD(0) або одностадійний TD (оновлення значення функції після будь-якого окремого кроку).

Monte Carlo $$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

TD Learning $$V(S_t) \leftarrow \underbrace{V(S_t)}_{\text{Previous estimate}} + \alpha[\underbrace{R_{t+1}}_{\text{Reward t+1}} + \underbrace{\gamma V(S_{t+1})}_{\text{Discounted value on the next step}} - V(S_t)]$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\text{TD Target}}$$

Методи TD лише чекають наступного кроку часу, щоб оновити оцінки значень. У момент часу t + 1 вони відразу ж формують мету TD, використовуючи винагороду Rt + 1 і поточну оцінку V(St + 1).

Цільовий показник TD є оцінкою: фактично ви оновлюєте попередню оцінку V(St) щодо однієї крокової мети.

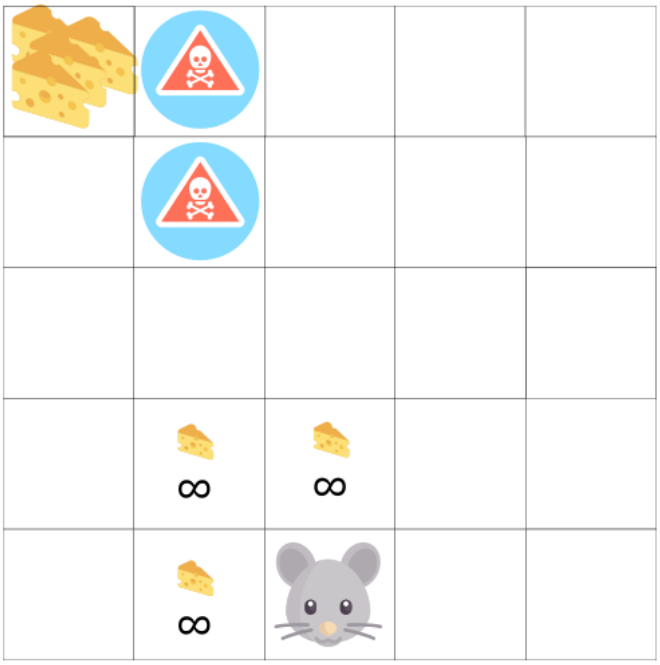
Компроміс між розвідкою та експлуатацією (explore-exploit tradeoff)

Компроміс між розвідкою та експлуатацією – це дилема, з якою ми часто стикаємося при виборі варіантів. Ви оберете те, що знаєте і отримаєте щось близьке до того, що ви очікуєте («експлуатація») або виберіть те, в чому ви не впевнені і можливо дізнаєтеся більше («дослідження»)? Таке трапляється постійно у повсякденному житті - улюблений ресторан чи новий? Поточна робота чи пошук нової? Звичайний маршрут додому чи іншого? І багато іншого. Ви жертвуєте одним заради іншого – це компроміс. Який з них ви повинні вибрати, залежить від того, наскільки дорогою буде інформація про наслідки, як довго ви зможете скористатися нею, і наскільки велика вигода для вас.

Подібно до того, як це відбувається в повсякденному житті, така ситуація часто виникає в інформатиці, звідки виник термін.

- Розвідка знаходить більше інформації про довкілля.
- Експлуатація використовує відому інформацію, щоб максимізувати винагороду.

Пам'ятайте, що мета нашого агента в Reinforcement Learning - максимізувати очікувану сукупну винагороду. Однак ми можемо потрапити до спільної пастки.



У цій грі наша миша може мати безліч маленьких сирів (+1 кожен). Але на вершині лабіринту є величезна кількість сиру (+1000).

Однак, якщо ми зосередимося лише на винагороді, наш агент ніколи не досягне величезної кількості сиру. Натомість він використовуватиме лише найближче джерело нагород, навіть якщо це джерело невелике (експлуатація).

Але якщо наш агент трохи досліджує, то він може знайти велику нагороду.

Це те, що ми називаємо компромісом між розвідкою та експлуатацією. Ми повинні визначити правило, яке допомагає впоратися із цим компромісом.

Три підходи до Reinforcement Learning

Тепер, коли ми визначили основні елементи навчання з підкріпленням, перейдемо до трьох підходів для вирішення завдань навчання з підкріпленням. Вони засновані на цінності, лінії поведінки та моделі.

Value Based підхід

У Reinforcement Learning, Value Based підхід у тому, щоб оптимізувати значення функції V(s).

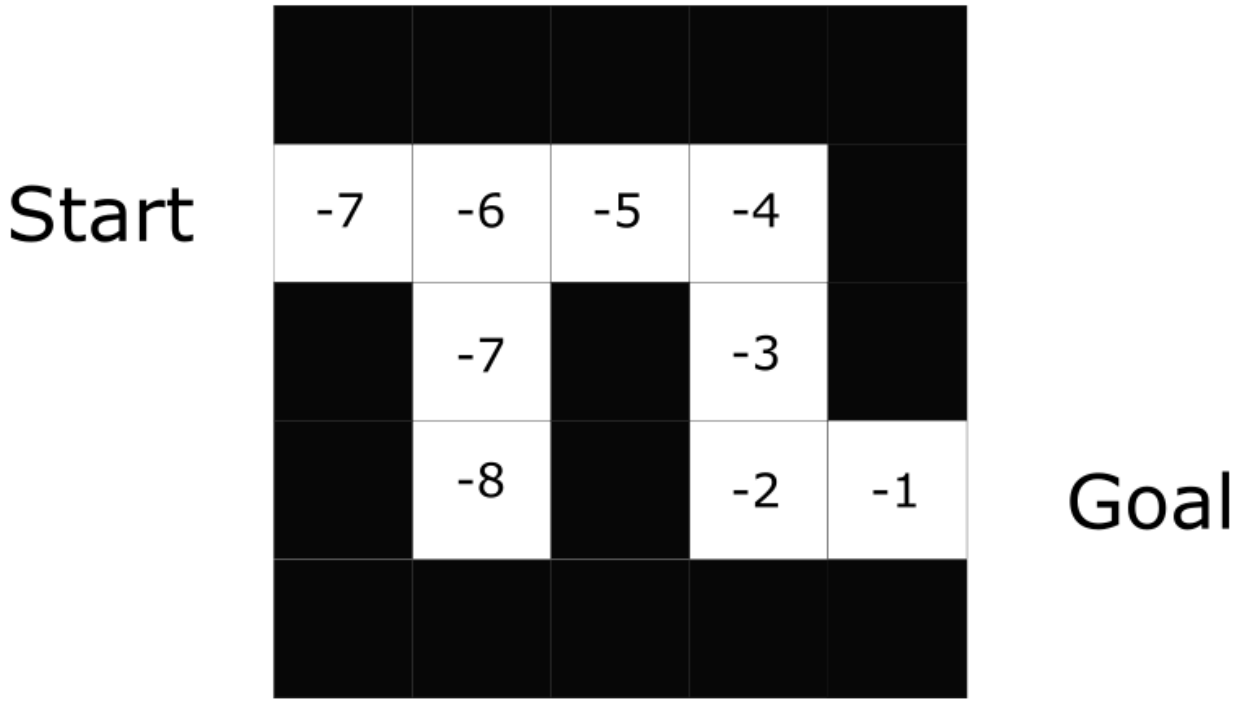
Функція value - це функція, яка повідомляє нам максимальну очікувану майбутню винагороду, яку агент отримає в кожному стані.

Значення кожного стану - це загальна сума винагороди, на яку агент може розраховувати в майбутньому, починаючи з цього стану.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s \right]$$

Expected | Reward discounted | Given that state

Агент буде використовувати цю функцію значення, щоб вибрати, який стан вибрати на кожному кроці. Агент приймає стан із найбільшим значенням цієї функції.



У прикладі з лабіринтом на кожному кроці ми приймаємо найбільше значення: -7 потім -6 потім -5 (і так далі) для досягнення мети.

Лінія поведінки (Policy Based)

У Reinforcement Learning на основі лінії поведінки ми хочемо безпосередньо оптимізувати функцію поведінки π(s) без використання функції значення.

Лінія поведінки це те, що визначає поведінку агента в даний момент часу.

$$a = \pi(s)$$

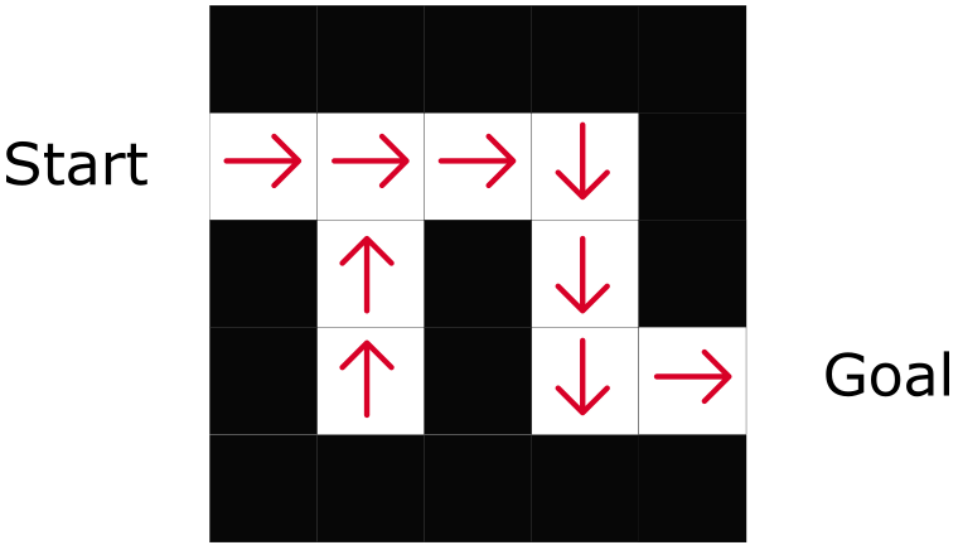
Ми вивчаємо функцію лінії поведінки. Це дозволяє нам порівняти кожний стан із найкращою відповідною дією.

У нас є два типи лінії поведінки:

- Детермінований: поведінка в даному стані завжди повертатиме одну і ту ж дію.
- Випадковий: повернення ймовірності розподілу на дії.

$$\text{Stochastic policy: } \pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

Proba of Take a particular action conditionned to a state



Як бачимо тут, лінія поведінки безпосередньо показує найкращу дію кожному за кроку.

Model Based

У разі ми моделюємо середовище, т. е. створюємо модель поведінки довкілля.

Проблема в тому, що кожне середовище потребуватиме різного представлення моделі.

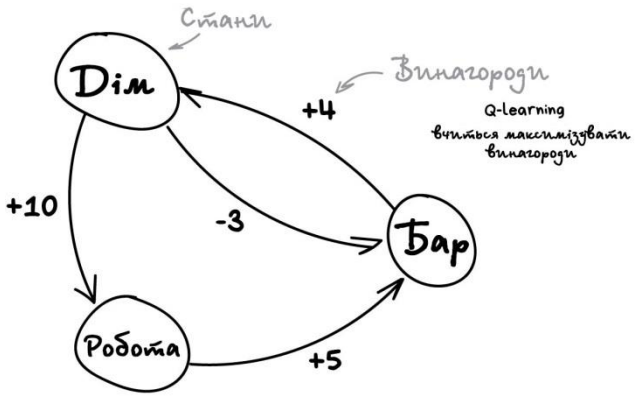
Reinforcement Learning спирається на оточення, щоб посилати йому скалярне число у відповідь на кожну нову дію. Нагороди, що повертаються навколишнім середовищем, можуть змінюватися, затримуватись або залежати від невідомих змінних, створюючи шум у контурі зворотного зв'язку.

Це призводить до більш повного вираження функції Q, яка враховує не лише безпосередні винагороди, викликані дією, але також і затримані винагороди, які можуть бути повернені на кілька кроків глибше в послідовності.

Як і люди, функція Q є рекурсивною. Подібно до того, як виклик методу програмного забезпечення human() містить у собі інший метод human(), плодом якого ми всі є, виклик функції Q для даної пари стан-дія вимагає від нас виклику вкладеної функції Q, щоб передбачити значення наступного стану, яке , своєю чергою, залежить від функції Q стану після цього тощо. Звідси і назва deep у терміні.

Пам'ятаєте новину кількарічної давнини, коли машина обіграла людину в Ґо? Хоча незадовго до цього було доведено, що кількість комбінацій фізично неможливо прорахувати, адже вона перевищує кількість атомів у всесвіті. Тобто, якщо в шахах машина реально може прорахувати усі майбутні комбінації і перемагати, з Ґо така ідея нездійснена. Тому вона просто вибирала найкращий вихід з кожної ситуації і робила це досить точно, щоб обіграти якогось вусатого дядька.

Ця ідея лежить в основі алгоритму Q-learning і його похідних (SARSA і DQN). Буква Q в назві означає слово Quality, тобто робот вчиться робити найбільш якісні кроки в будь-якій ситуації, а всі ситуації він запам'ятовує як простий марковский процес.



Рутинний Марковський Процес

Машина проганяє мільйони симуляцій в середовищі, запам'ятовуючи всі сформовані ситуації і виходи з них, які принесли максимальну винагороду. Але як зрозуміти, коли у нас склалася знайома ситуація, а коли абсолютно нова? Ось самокеруючий автомобіль стоїть біля перехрестя і загоряється зелений - значить можна їхати? А якщо справа мчить швидка допомога з мигалками?

Відповідь - ну хто його знає, ніяк, магії не буває, дослідники постійно цим займаються і винаходять свої велосипеди. Одні прописують всі ситуації

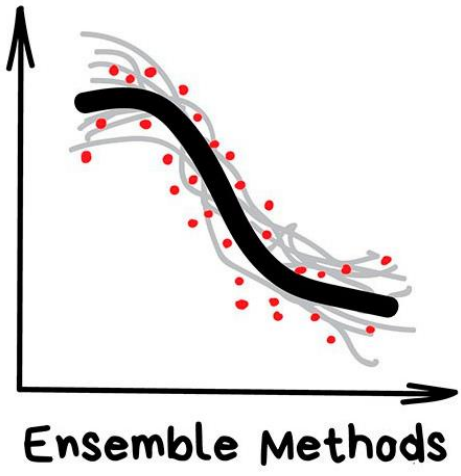
руками, що дозволяє їм обробляти виняткові випадки типу проблеми вагонетки. Інші йдуть глибше і віддають цю роботу нейромережам, нехай самі все знайдуть. Так замість Q-learning'a у нас з'являється Deep Q-Network (DQN).

Reinforcement Learning для простого юзера виглядає як справжній інтелект. Тому що "wow, машина сама приймає рішення в реальних ситуаціях!" Він зараз на хайпі, швидко пре вперед і активно прагне в нейромережі, щоб стати ще точнішим (а не стукатися в ніжку стільця по двадцять разів).

Колись були дуже популярні генетичні алгоритми (за посиланням прикольна візуалізація). Це коли ми кидаємо купу роботів в середовище і змушуємо їх йти до мети, поки не здохнуть. Потім вибираємо кращих, схрещуємо, додаємо мутації і кидаємо ще раз. Через пару мільярдів років має вийти розумна істота. Теорія еволюції у дії.

Так ось, *генетичні алгоритми теж відносяться до навчання з підкріпленням*.

2. АНСАМБЛІ



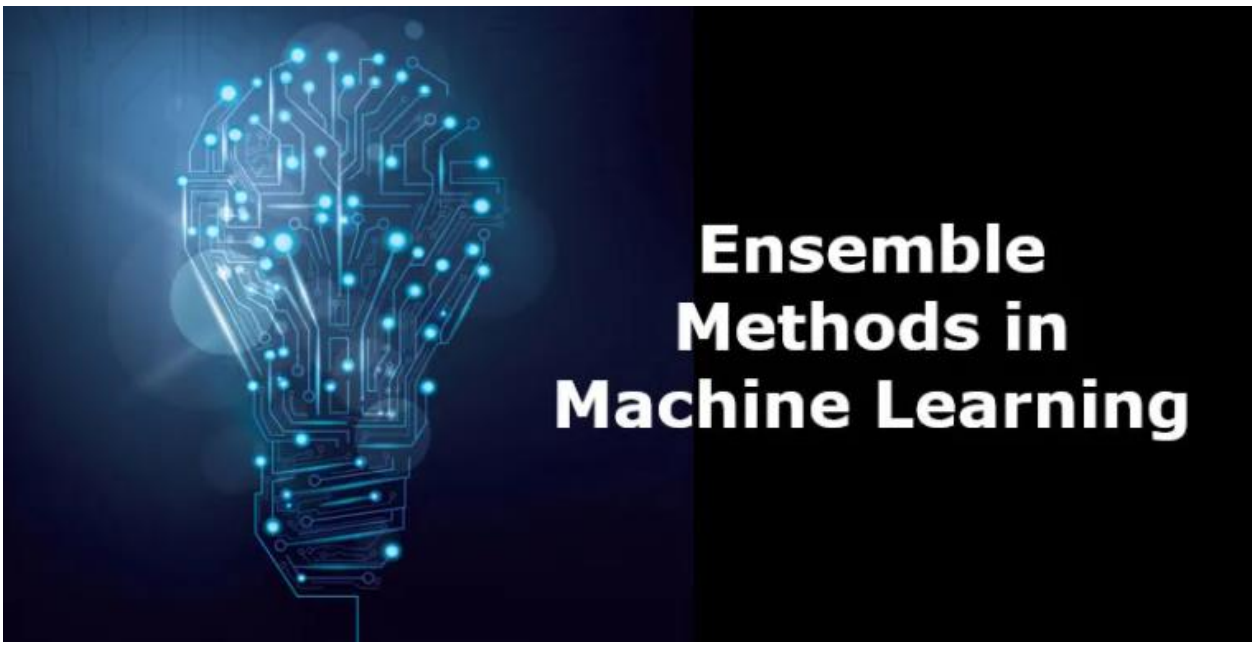
«Купа дурних дерев вчиться виправляти помилки один одного»

Сьогодні використовують для:

Всього, де можна скористатися класичними алгоритмами (але працюють точніше)

Пошукові системи (★)

Розпізнавання об'єктів



Популярні алгоритми: Random Forest , Gradient Boosting

Переходимо до справжніх дорослих методів. Ансамблі та нейромережі - наші головні бійці на шляху до неминучої сингулярності. Сьогодні вони дають найточніші результати і використовуються усіма великими компаніями в продакшені. Лише про нейромережі й тріщать у кожному кутку, а слова «бустинг» і «бегінг», напевно, лякають тільки хіпстера з теккранча.

Разом з усією їхньою ефективністю, ідея до знущання проста. Виявляється, якщо взяти декілька не дуже ефективних методів навчання і навчити виправляти помилки один одного, якість такої системи буде дуже вище, ніж кожного з методів окремо.

Причому навіть краще, коли взяті алгоритми максимально нестабільні і сильно плавають стосовно вхідних даних. Тому частіше беруть Регресію і Дерева Рішень, яким достатньо однієї сильної аномалії в даних, щоб поїхала вся модель. А ось Баєса і K-NN не беруть ніколи - вони хоч і тупі, але дуже стабільні.

Типи ансамблевих методів у машинному навчанні

Методи ансамблю допомагають створити декілька моделей, а потім об'єднати їх для отримання поліпшених результатів. Деякі ансамблеві методи класифікуються на такі групи:

1. Послідовні методи

У такому методі ансамблю існують послідовно генеровані базові учні, в яких існує залежність даних. Усі інші дані базового учня мають певну залежність від попередніх даних. Отже, попередні помилкові дані налаштовуються виходячи з його ваги для покращення продуктивності загальної системи.

Приклад : Підвищення, (Бустінг, англ. Boosting)

2. Паралельний метод

У такому методі ансамблю базовий учень генерується в паралельному порядку, в якому залежності від даних немає. Усі дані базового учня формуються незалежно.

Приклад : укладання (стекінг, stacking)

3. Однорідний (гомогенний) ансамбль

Такий метод ансамблю - це поєднання однотипних класифікаторів. Але набір даних різний для кожного класифікатора. Це змусить більш точно працювати комбіновану модель після узагальнення результатів від кожної моделі. Цей тип ансамблевого методу працює з великою кількістю наборів даних. У гомогенному методі метод вибору особливостей однаковий для різних даних тренувань. Це обчислювально дорого.

Приклад: Народні методи, такі як розфасування та підсилення, входять в однорідний ансамбль.

4. Гетерогенний ансамбль

Такий метод ансамблю - це поєднання різних типів класифікаторів або моделей машинного навчання, в яких кожен класифікатор будується на одних і тих же даних. Такий метод працює для невеликих наборів даних. У неоднорідному методі вибору особливостей для одних і тих же навчальних даних різний. Загальний результат цього ансамблевого методу здійснюється шляхом усереднення всіх результатів кожної комбінованої моделі.

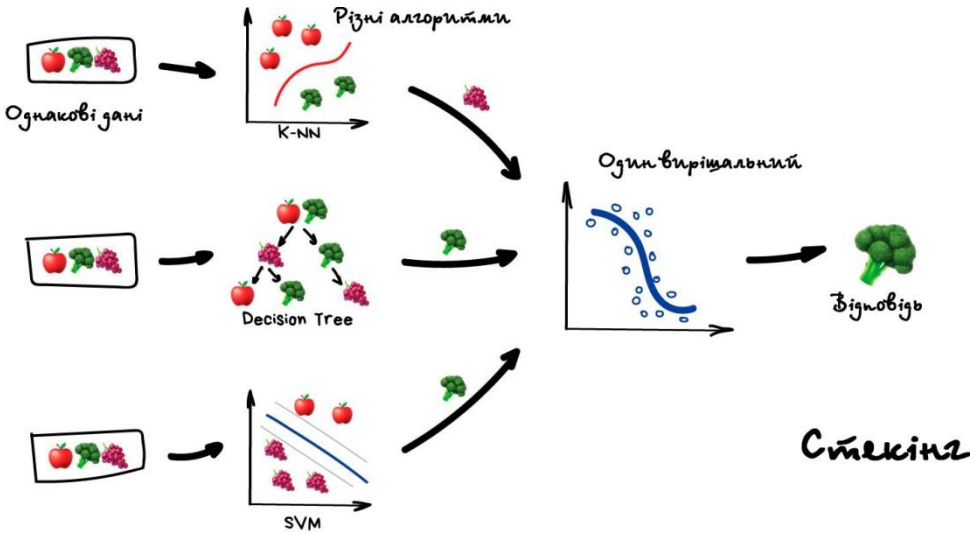
Приклад : укладання

Взагалі то ансамбль можна зібрати як завгодно, хоч випадково нарізати у каструльку класифікатори і залити регресією. За точність, правда, тоді ніхто не ручається. Тому **є чотири перевірених способи робити ансамблі.**

Bagging (пакування в мішок чи сумку)

Цей метод ансамблю поєднує дві моделі машинного навчання, тобто завантаження та агрегацію, в єдину модель ансамблю. Мета методу розфасовки - зменшити велику дисперсію моделі. Дерева рішень мають дисперсію та низький ухил. Великий набір даних є (скажімо, 1000 зразків) підвибіркою (скажімо, 10 підзразків кожен містить 100 зразків даних). Кілька дерев рішень будуються на кожному навчальному даному під-зразку. Незважаючи на удари даних про підбірку даних про різні дерева рішень, стурбованість переповненням навчальних даних щодо кожного дерева рішень зменшується. Для ефективності моделі кожне окреме дерево рішень вирощується в глибині, що містить дані про підбірку навчальних даних. Результати кожного дерева рішень узагальнюються, щоб зрозуміти остаточний прогноз. Дисперсія узагальнених даних зменшується. Точність прогнозування моделі в методі упаковки залежить від кількості використовуваного дерева рішень. Різні підвідбірки вибіркових даних вибираються випадковим чином із заміною. Вихід кожного дерева має високу кореляцію.

Стекінг Навчаємо кілька різних алгоритмів і передаємо їх результати на вхід останньому, який приймає остаточне рішення. Типу як дівчата спочатку опитують всіх своїх подруг, щоб прийняти рішення зустрічатися з хлопцем чи ні.



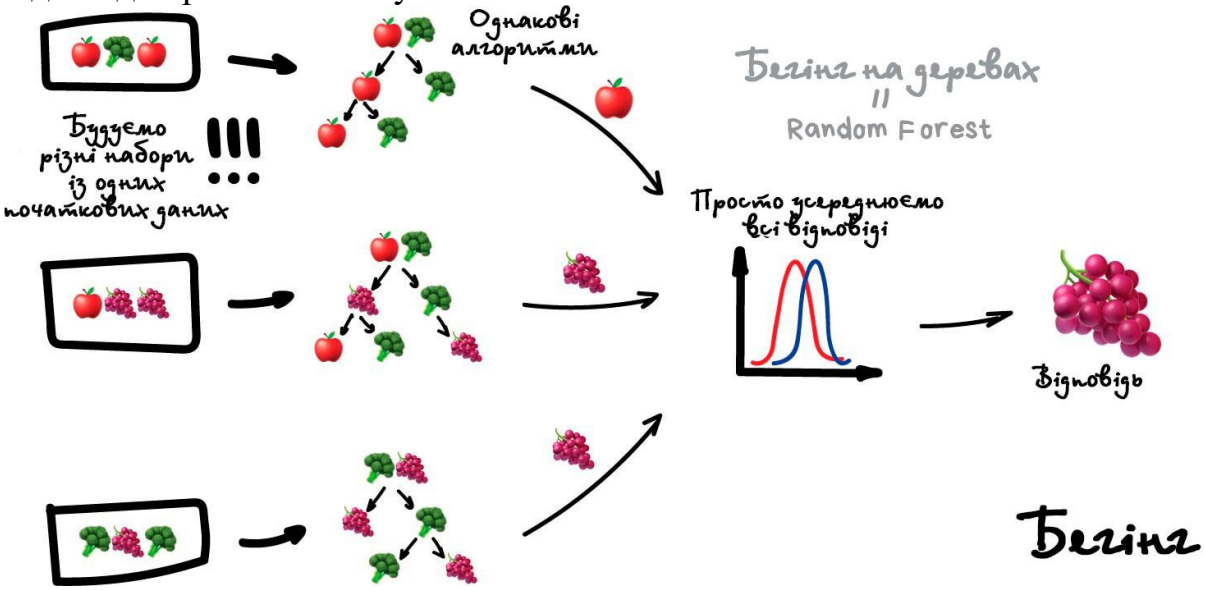
Ключові слова - *різних алгоритмів*, адже один і той же алгоритм, навчений на одних і тих же даних не має сенсу. Яких - ваша справа, хіба що в ролі вирішального алгоритму частіше беруть регресію.

Цей метод також поєднує в собі декілька методів класифікації або регресії, використовуючи метакласифікатор або мета-модель. Моделі нижчих рівнів навчаються з повним набором навчальних даних, а потім комбінована модель навчається з результатами моделей нижчого рівня. На відміну від прискорення, кожна модель нижчого рівня проходить паралельну підготовку. Прогноз із моделей нижчого рівня використовується як вхід для наступної моделі як навчальний набір даних і утворює стек, у якому верхній шар моделі більш навчений, ніж нижній шар моделі. Модель верхнього шару має хорошу точність прогнозування, і вони побудовані на основі моделей нижчого рівня. Стек збільшується до тих пір, поки найкраще прогнозування не буде виконано з мінімальною помилкою. Прогнозування комбінованої моделі чи мета-моделі ґрунтується на прогнозуванні різних слабких моделей або моделей нижнього шару. Він фокусується на тому, щоб створити меншу модель зміщення.

Чисто з досвіду - стекінг на практиці застосовується рідко, тому що два інших методи зазвичай точніші.

Бегінг Він же Bootstrap AGGregatING. Навчаємо один алгоритм багато разів на випадкових вибірках з вихідних даних. В кінці усереднюємо відповіді.

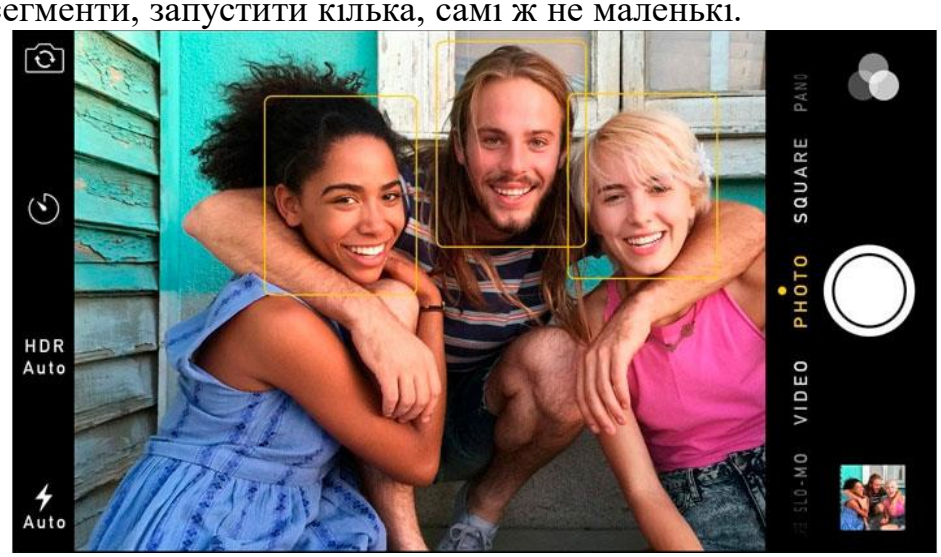
Дані в випадкових вибірках можуть повторюватися. Тобто з набору 1-2-3 ми можемо робити вибірки 2-2-3, 1-2-2, 3-1-2 і так поки не набридне. На них ми навчаємо один і той же алгоритм кілька разів, а в кінці знаходимо відповідь простим голосуванням.



Найпопулярніший приклад бегінга - алгоритм Random Forest, бегінг на деревах, який намальований на рисунку вище. Коли ви відкриваєте камеру на телефоні і бачите як вона окреслила обличчя людей в кадрі жовтими прямокутниками - швидше за все це їх робота. Нейромережа буде занадто

повільна в реальному часі, а бегінг ідеальний, адже він може обчислювати свої дерева паралельно на всіх шейдерах відеокарти.

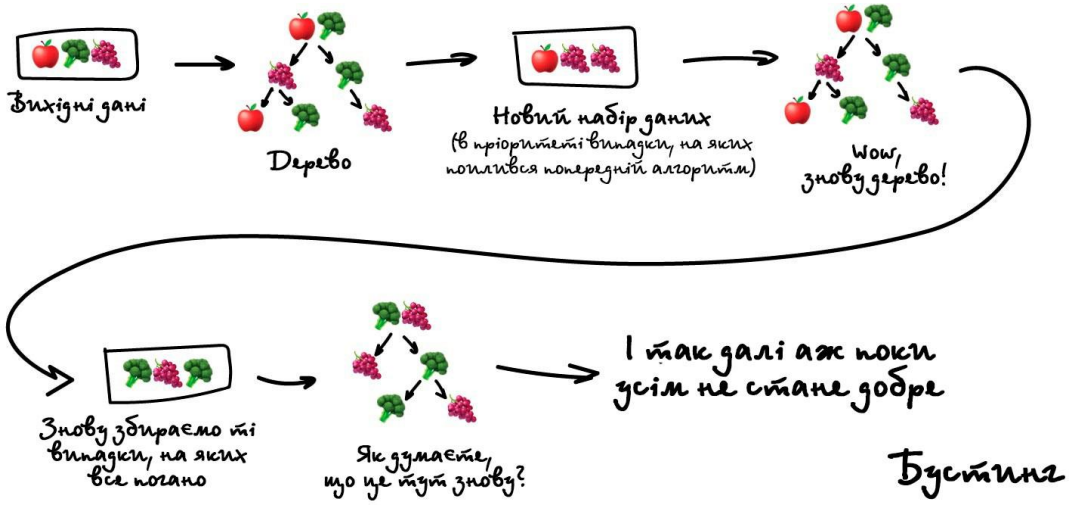
Дика *здатність паралелетися* дає бегінгу перевагу навіть над наступним методом, який працює точніше, але тільки в один потік. Хоча можна розбити на сегменти, запустити кілька, самі ж не маленькі.



Бустинг Навчаємо алгоритми послідовно, кожен наступний приділяє особливу увагу тим випадкам, на яких помилився попередній.

Підвищуючий ансамбль також поєднує різні класифікатори одного типу. Підвищення - це один із методів послідовного ансамблю, в якому кожна модель або класифікатор працює на основі функцій, які будуть використані наступною моделлю. Таким чином, метод прискорення забезпечує сильнішу модель для студентів із слабких моделей учнів шляхом усереднення їх ваги. Іншими словами, більш сильно навчена модель залежить від декількох слабко навчених моделей. Слабка модель, яка навчається, або модель, яка навчається зносу, дуже менш корелює з справжньою класифікацією. Але наступний слабкий учень трохи більше корелює з справжньою класифікацією. Поєднання таких різних слабких учнів дає сильного учня, що добре корелює з справжньою класифікацією.

Як в бегінгу, ми робимо вибірки з вихідних даних, але тепер не зовсім випадково. У кожну нову вибірку ми беремо частину тих даних, на яких попередній алгоритм відпрацював неправильно. Тобто, так би мовити, донавчаємо новий алгоритм на помилках попереднього.



Плюси - гарна точність класифікації. Мінуси вже названі - не паралелиться. Хоча все одно працює швидше нейромереж, які мов навантажені КРАЗи з піском у порівнянні зі спритним бустингом.

Потрібен реальний приклад роботи бустинга - відкрийте Яндекс і введіть запит. Чуєте, як Матрикснет гуркоче деревами і ранжує вам результати? Ось це якраз воно, Яндекс зараз весь на бустингу. Про Google не знаю.

Сьогодні є три популярних методи бустингу, відмінності між якими добре висвітлено у статті [CatBoost vs. LightGBM vs. XGBoost]

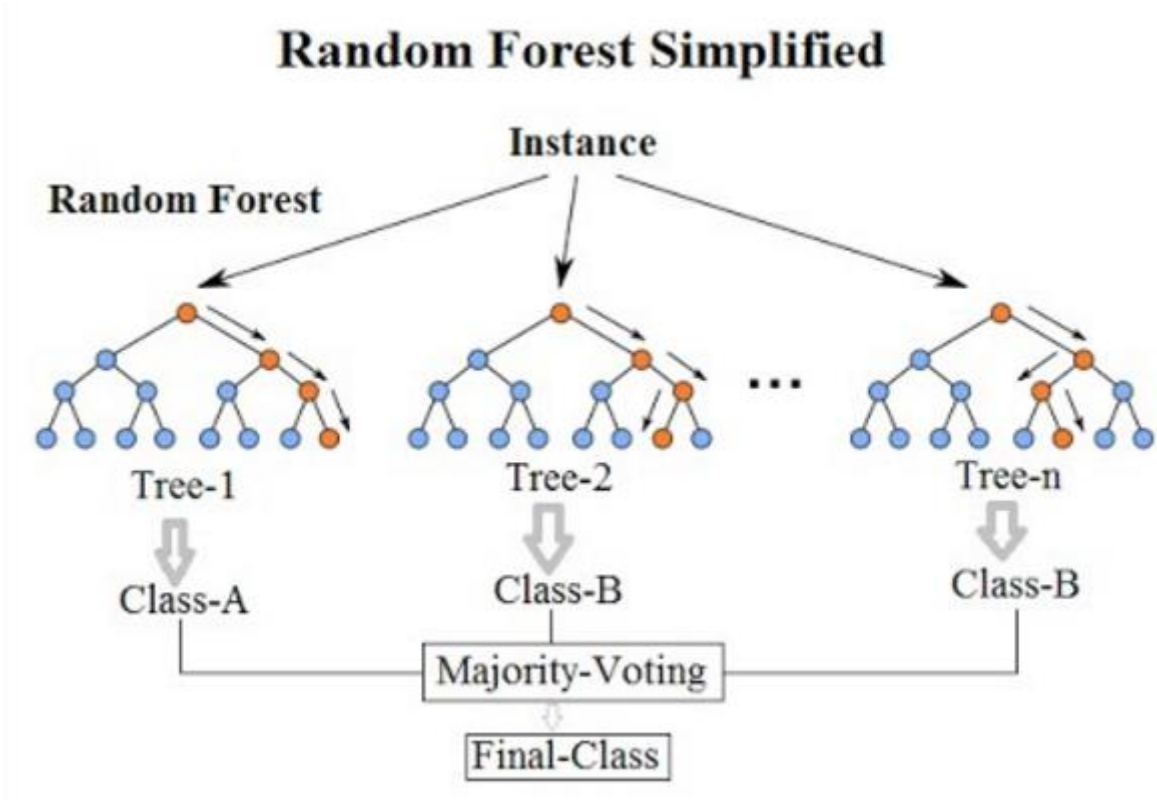
Випадковий ліс

Дерева рішень - найпопулярніший метод для вирішення різних завдань машинного навчання. Навчання на основі дерева «найближче підходить до задоволення вимог для використання в якості стандартної процедури інтелектуального аналізу даних», - кажуть фахівці. Однак вони рідко бувають точними. Зокрема, дерева, які виросли дуже глибоко, зазвичай, засвоюють дуже нерегулярні моделі: вони перевищують свої навчальні набори, тобто, мають низьку систематичну помилку, але дуже високу дисперсію. Випадкові ліси - це спосіб усереднення кількох глибоких дерев рішень, навчених на різних частинах одного й того ж навчального набору, з метою зменшення дисперсії. Це відбувається за рахунок невеликого збільшення систематичної помилки та деякої втрати інтерпретованості, але загалом значно підвищує продуктивність остаточної моделі.

Тобто, **Випадкові ліси або ліси випадкових рішень (Random forests or random decision forests)** – це метод навчання ансамблю для класифікації, регресії та інших завдань, який працює шляхом побудови множини дерев рішень під час навчання. Для завдань класифікації виходом довільного лісу є клас, обраний більшістю дерев. Для завдань регресії повертається середнє значення чи середній прогноз окремих дерев. Випадкові ліси рішень коригують схльність дерев рішень до перевчання їх на навчальному наборі. Випадкові ліси зазвичай перевершують дерева рішень, але їх точність нижче, ніж у дерев з градієнтним посиленням. Однак характеристики вхідних даних також можуть вплинути на їхню продуктивність.

Випадкові ліси часто використовуються як моделі «чорної скриньки» на підприємствах, оскільки вони генерують розумні прогнози для широкого діапазону даних, не вимагаючи невеликої настройки.

Випадковий ліс дещо відрізняється від пакування в мішки (Bagging), оскільки він використовує глибокі дерева, які розміщені на зразках завантажувальної машини. Вихід кожного дерева поєднується для зменшення дисперсії. Під час вирощування кожного дерева, замість того, щоб генерувати зразок завантаження на основі спостереження в наборі даних, ми також відбираємо набір даних на основі функцій і використовуємо лише випадкову підмножину такого зразка для побудови дерева. Іншими словами, вибірка набору даних проводиться на основі функцій, що зменшують співвідношення різних результатів. Випадковий ліс хороший для відновлення відсутніх даних. Випадковість лісу означає випадковий вибір підмножини вибірки, що зменшує шанси отримати відповідні значення прогнозування. Кожне дерево має різну будову. Випадковий ліс призводить до незначного збільшення зміщення лісу, але завдяки усередненню все менш пов'язаних прогнозів для різних дерев результуюча дисперсія зменшується та дає загальну кращу продуктивність.



Алгоритм навчання для випадкових лісів застосовує загальну техніку агрегування початкового завантаження чи бегінга до учнів дерев. Враховуючи навчальний набір $X = x_1, ..., x_n$ з відповідями $Y = y_1, ..., y_n$, повторна упаковка (B разів) вибирає випадкову вибірку із заміною навчального набору та підганяє дерева до цих зразків:

Для $b = 1, ..., B$:

1. Приклад, із заміною, n навчальних прикладів з X, Y; назвемо їх X_b, Y_b.
2. Навчіть дерево класифікації або регресії f_b на X_b, Y_b.

Після навчання прогнози для невидимих вибірок x' можуть бути зроблені шляхом усереднення прогнозів всіх окремих дерев регресії x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

або шляхом отримання більшості голосів у разі класифікації дерев.

Ця процедура початкового завантаження (bootstrapping procedure) призводить до кращої продуктивності моделі, оскільки зменшує дисперсію моделі без збільшення зміщення. Це означає, що хоча прогнози одного дерева дуже чутливі до шуму в його навчальному наборі, середнє значення для багатьох дерев незмінюється, поки дерева не корельовані. Просте навчання множини дерев на одному навчальному наборі дасть сильно корельовані дерева (або навіть те саме дерево багато разів, якщо алгоритм навчання детермінований). Самостійна вибірка - це спосіб декореляції дерев шляхом показу різних навчальних наборів.

Крім того, оцінка невизначеності прогнозу може бути зроблена як стандартне відхилення прогнозів по всіх окремих дерев регресії по x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

Кількість зразків/дерев, B є вільним параметром. Зазвичай використовується від кількох сотень до кількох тисяч дерев, залежно від розміру та характеру навчального набору. Оптимальну кількість дерев B можна знайти за допомогою перехресної перевірки або спостереження за помилкою поза пакетом: середньою помилкою прогнозування для кожної навчальної вибірки x_i, використовуючи тільки дерева, у яких не було x_i у їх вибірці початкового завантаження. Помилка навчання та тестування має тенденцію вирівнюватися після припасування деякої кількості дерев.

Вищезгадана процедура визначає вихідний алгоритм упаковки дерев. Випадкові ліси також включають інший тип схеми упаковки: вони використовують модифікований алгоритм навчання дерева, який вибирає при кожному розбиття кандидатів у процесі навчання випадкову підмножину функцій. Цей процес іноді називають «складанням функцій». Причина для цього - кореляція дерев у звичайній вибірці початкового завантаження: якщо одна або кілька функцій є дуже сильними предикторами для змінної відповіді (цільовий результат), ці функції будуть обрані в багатьох B деревах, викликаючи їх щоб стати корельованими.

Зазвичай завдання класифікації з p ознаками у кожному розбиття використовуються ознаки $\sqrt{p}$ (з округленням у менший бік). Для завдань регресії винахідники рекомендують $p/3$ (з округленням у менший бік) з

мінімальним розміром вузла 5 за замовчуванням. На практиці найкращі значення цих параметрів залежатимуть від проблеми, і їх слід розглядати як параметри налаштування.

ExtraTrees

Додавання ще одного кроку рандомізації дає дуже рандомізовані дерева або ExtraTrees. Хоча вони схожі на звичайні випадкові ліси в тому сенсі, що вони є ансамблем окремих дерев, є дві основні відмінності: по-перше, кожне дерево навчається з використанням всієї навчальної вибірки (а не вибірки початкового завантаження), а по-друге, низхідне розбиття в учень дерева рандомізовано. Замість того, щоб обчислювати локально оптимальну точку відсікання для кожної розглянутої функції (на основі, наприклад, отримання інформації або домішки Джіні), випадкова точка відсічення обрана. Це значення вибирається з рівномірного розподілу в межах емпіричного діапазону функції (навчальному наборі дерева). Потім зі всіх випадково згенерованих розбиття вибирається розбиття, що дає найвищий бал, для розбиття вузла. Подібно до звичайних випадкових лісів, можна вказати кількість випадково обраних об'єктів, які будуть враховуватися в кожному вузлі. Значення за промовчанням для цього параметра: $\sqrt{p}$ для класифікації і p для регресії, де p кількість функцій у моделі.