ЛЕКЦІЯ 2

МАШИННЕ НАВЧАННЯ В ЗАДАЧАХ КІБЕРБЕЗПЕКИ

ПЛАН

1. Складові машинного навчання та штучний інтеллект
2. Класифікація методів машинного навчання
3. Методи класифікації
4. Оцінка якості класифікації

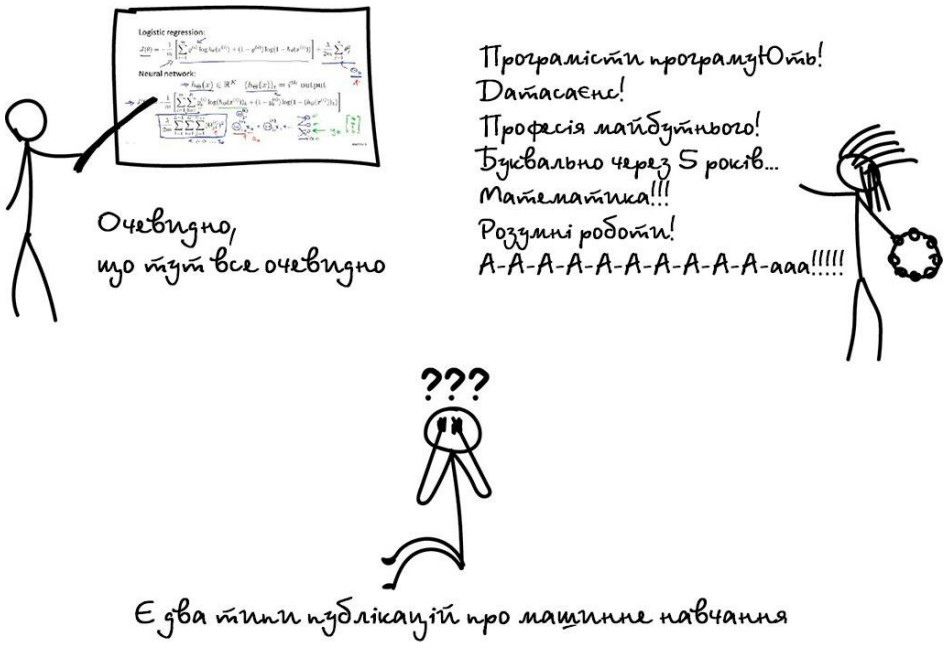
ЛІТЕРАТУРА

Сайт http://www.mmf.lnu.edu.ua/ar/1739

ВСТУП

Загалом статті про машинне навчання діляться на два типи: це або 2-3-4..-томники з формулами і теоремами, які мало хто зміг дочитати навіть до середини, або казки про штучний інтелект, професії майбутнього, чарівних дата-саєнтистів.

Нижче розглянемо пост-вступ для тих, хто хоче нарешті розібратися в машинному навчанні - простою мовою, без формул-теорем, зате з прикладами реальних задач та їх розв'язань.



Навіщо навчати машини

Приклад

Припустимо, що Олег хоче купити автомобіль і обдумує скільки грошей йому потрібно для цього накопити. Він переглянув десяток оголошень в інтернеті і побачив, що нові автомобілі коштують близько $ 20 000, однорічні - приблизно $ 19 000, дворічні - $ 18 000 і так далі.

Зрозуміло, що Олег-аналітик виводить формулу: адекватна ціна автомобіля починається від $ 20 000 і падає на $ 1000 щороку, поки не упреться в $ 10 000.

Олег зробив те, що в машинному навчанні називають регресією - передбачив ціну за відомими даними. Люди роблять це постійно, коли вираховують за скільки продати старий айфон або скільки шашлику взяти на заміську забаву (моя формула - півкіло на людину в добу).

Очевидно, що було б зручно мати формулу під кожну проблему на світі. Але взяти ті ж ціни на автомобілі: крім пробігу є десятки комплектацій, різний технічний стан, сезонність попиту і ще стільки неочевидних факторів, які Олег, навіть при всьому бажанні, не врахував би в голові.

Люди тупі (вибачайте!) і ліниві, а тому треба змусити працювати роботів. Нехай машина подивиться на наші дані, знайде в них закономірності і навчиться передбачати для нас відповідь. Найцікавіше, що в підсумку машина може знаходити навіть такі закономірності, про які люди не здогадуються.

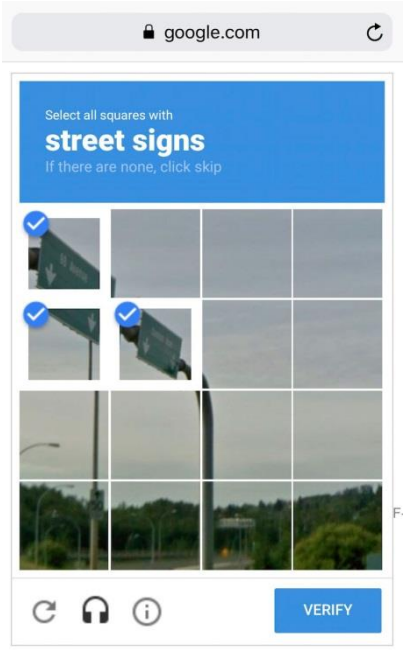
Так народилося машинне навчання.

1. СКЛАДОВІ МАШИННОГО НАВЧАННЯ ТА ШТУЧНИЙ ІНТЕЛЛЕКТ

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.**

Дані Хочемо виявляти спам - потрібні приклади спам-листів, передбачати курси акцій - потрібна історія цін, дізнатися інтереси користувача - потрібні його лайки або пости. Даних потрібно якомога більше. Десятки тисяч прикладів - це отой злий мінімум для відчайдухів.



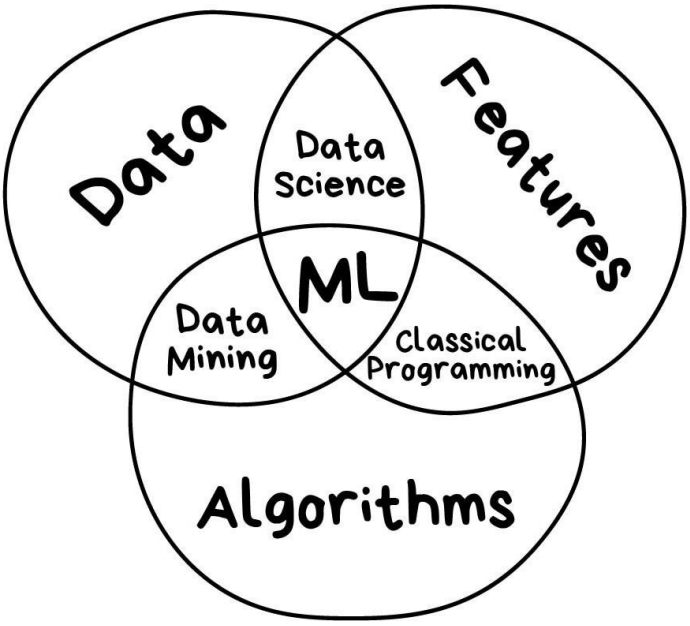
Дані збирають як тільки можуть. Хтось вручну - процес триваліший, даних менше, зате без помилок. Хтось автоматично - просто зливає машині все, що знайшлося, і вірить у щось краще. Найхитріші, типу гугла, використовують своїх же користувачів для безкоштовної розмітки. Згадайте ReCaptcha, яка іноді вимагає «знайти на фотографії всі дорожні знаки» - це воно і є.

За хорошими наборами даних (датасетами) йде велике полювання. Великі компанії інколи (й таке буває) розкривають свої алгоритми, але датасети - вкрай рідко.

Ознаки. Ми називаємо їх фічами (features), так що фаріонщикам доведеться страждати. Фічі, властивості, характеристики, ознаки - ними можуть бути пробіг автомобіля, стать користувача, ціна акцій, навіть лічильник частоти появи слова в тексті.

Машина повинна знати, на що їй конкретно дивитися. Добре, коли дані просто лежать в табличках - назви їх колонок і є фічі. А якщо у нас сто гігабайтів картинок з котами? Коли ознак багато, модель працює повільно і неефективно. Найчастіше відбір правильних фічів займає більше часу, ніж все інше навчання. Але бувають і зворотні ситуації, коли юзер сам вирішує відібрати тільки «правильні» на його погляд ознаки і вносить в модель суб'єктивність - вона починає дико брехати.

Алгоритм Зазвичай, одну й ту ж задачу майже завжди можна розв'язати різними методами-способами. Від вибору методу залежить точність, швидкість роботи і розмір готової моделі. Але є один нюанс: якщо дані - «сміття», то навіть найкращий алгоритм не допоможе. Не зациклюйтеся на відсотках, краще зберіть побільше даних.



Навчання та Інтелект

Одного разу в одному хіпстерскому виданні була стаття під назвою «Чи замінять нейромережі машинне навчання». Піарники в своїх прес-релізах обзивають «штучним інтелектом» будь-яку лінійну регресію, з якою вже дітки у дворі бавляться. Раз і назавжди пояснимо різницю на зображенні.

Штучний інтелект - назва всієї області, як біологія або хімія.

Машинне навчання - це розділ штучного інтелекту. Важливий, але не єдиний.

Нейромережі – можна розглядати як один з типів машинного навчання. Популярний, але є й інші, не гірші.

Глибоке навчання - архітектура нейромереж, один з підходів до їх побудови та навчання. На практиці мало хто відрізняє, де глибокі нейромережі, а де не дуже. Кажуть назву конкретної мережі і все.

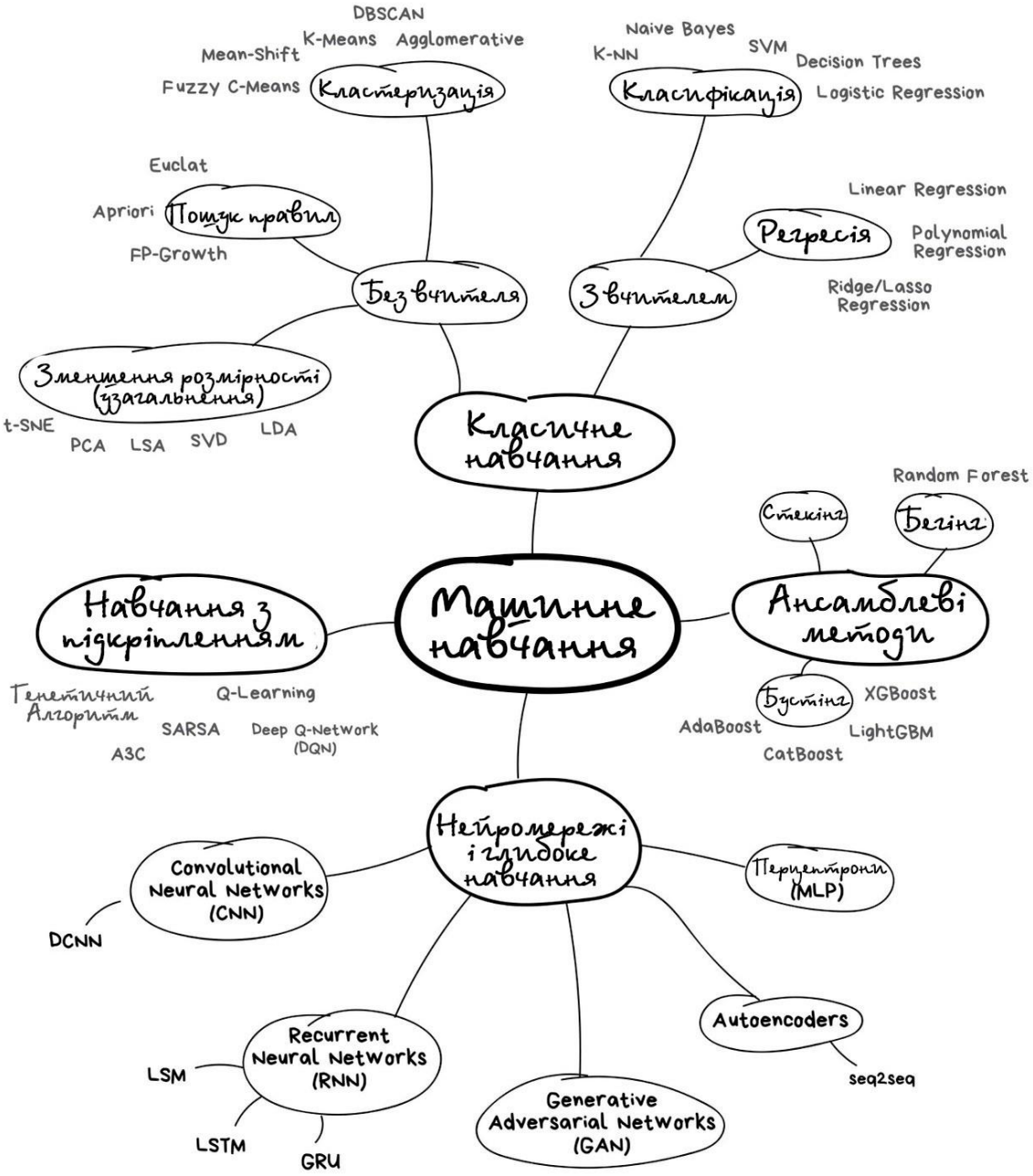
Порівнювати можна тільки речі одного рівня, інакше виходить повний буллщит типу «що краще: машина чи колесо?» Не ототожнюйте терміни без причини, щоб не виглядати дурниками.

Ось що машини сьогодні вміють, а що не під силу навіть найбільш навченим.

МАШИНА МОЖЕ	МАШИНА НЕ МОЖЕ
Передбачати	Створювати нове
Запам'ятовувати	Різко порозумнішати
Відтворювати	Вийти за рамки завдання
Вибирати найкраще	Вбити всіх людей

2. КЛАСИФІКАЦІЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ

Якщо ліньки читати - хоча б гляньте на картинку, буде корисно.



Класифікувати алгоритми можна десятком способів. Оберемо цей, тому що він здається найзручнішим для поясювання. Треба розуміти, що не буває так, щоб задачу розв'язував лише один метод. Я буду згадувати відомі приклади застосувань, але пам'ятайте, що «син маминої подруги» все це може розв'язати нейромережами.

Розпочнемо з базового огляду. Сьогодні в машинному навчанні є лише чотири основні напрями.

Перші алгоритми прийшли до нас ще в 1950-х роках з чистої статистики. Вони розв'язували формальні задачі - шукали закономірності в числах, оцінювали близькість точок в просторі і вираховували напрямки.

Сьогодні на класичних алгоритмах тримається добра половина інтернету. Коли ви зустрічаєте блок «Рекомендовані статті» на сайті, або банк блокує всі ваші гроші на картці після першої ж покупки кави за кордоном - це майже завжди справа рук одного з цих алгоритмів.

Зрозуміло, що великі корпорації люблять розв'язувати всі проблеми нейромережами. Це спричинено тим, що зайві 2% точності для них легко конвертуються в додаткові 2 мільярди прибутку. Решті ж варто включати голову. Коли задача розв'язується класичними методами, то дешевше реалізувати скільки-небудь корисну для бізнесу систему саме за їх допомогою, а вже потім думати про якесь покращення. А якщо ви не розв'язали задачу, то не розв'язати її на 2% краще вам не особливо допоможе.

Знаю кілька смішних історій, коли команда три місяці переписувала систему рекомендацій інтернет-магазину на більш точний алгоритм, і тільки потім зрозуміла, що покупці взагалі нею не користуються. Велика частина покупців просто приходить з пошукових систем.

При всій своїй популярності, класичні алгоритми настільки прості, що їх легко пояснити навіть дитині. Сьогодні вони як основи арифметики - застосовуються постійно, але дехто все одно став їх забувати.

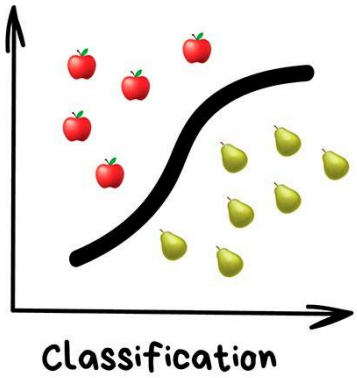
Класичне навчання люблять ділити на дві категорії — з вчителем і без. Часто можна зустріти їх англійські назви — Supervised і Unsupervised Learning.

У першому випадку у машини є якийсь учитель, який говорить їй як правильно. Розповідає, що на цій картинці кішка, а на цій собака. Тобто вчитель вже заздалегідь розділив всі дані на кішок і собак, а машина навчається на конкретних прикладах.

У навчанні без учителя, машині просто вивалюють купу фотографій тварин на стіл і кажуть «розберися, хто тут на кого схожий». Дані не розмічені, у машини немає вчителя, і вона намагається сама знайти будь-які закономірності. Про ці методи поговоримо нижче.

Очевидно, що з учителем машина навчиться швидше і точніше, тому в бойових задачах його використовують набагато частіше. **Ці задачі діляться на два типи: класифікація - передбачення категорії об'єкта, і регресія - передбачення місця на числовій прямій.**

3. МЕТОДИ КЛАСИФІКАЦІЇ



«Поділяє об'єкти за заздалегідь відомою ознакою. Шкарпетки за кольорами, документи за мовами, музику за жанрами»

Сьогодні використовують для:

Спам-фільтри

Визначення мови

Пошук схожих документів

Ааналіз тональності

Розпізнавання рукописних букв і цифр

Визначення підозрілих транзакцій

Популярні алгоритми: Наївний Баєс, Дерева Рішень, Логістична Регресія, K-найближчих сусідів, Метод Опорних Векторів.

Класифікація речей - найпопулярніша задача у всьому машинному навчанні. Машина в ній як дитина, яка навчається розкладати іграшки: роботів в один ящик, танки в інший. Опа, а якщо це робот-танк? Що ж, час розплакатися і випасти в помилку.

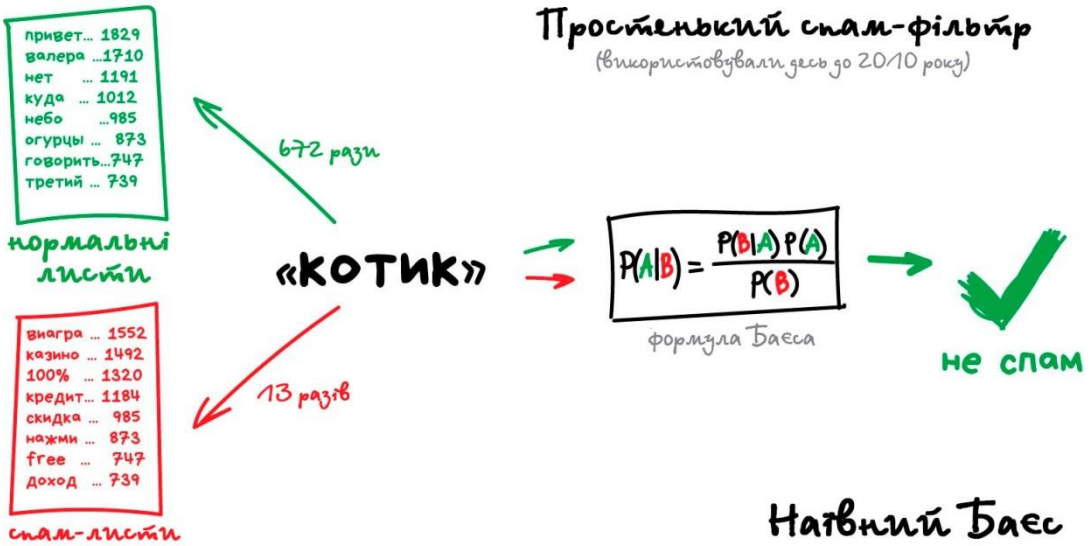
Для класифікації завжди потрібен учитель - розмічені дані з ознаками і категоріями, які машина буде вчитися визначати за цими ознаками. Далі класифікувати можна що завгодно: користувачів за інтересами - так роблять алгоритмічні стрічки, статті за мовами і тематиками - важливо для пошукових систем, музику за жанрами - згадайте плейлисти, навіть листи у вашій поштовій скриньці.

3.1.Наївний Баєс

Наївний метод Баєса – це набір методів класифікації, заснованих на теоремі Баєса. Це не єдиний метод, а сімейство методів, які поділяють загальний принцип, згідно з яким кожна ознака, що класифікується, не залежить від значення будь-якої іншої ознаки. Наприклад, фрукт можна вважати яблуком, якщо він червоний, круглий і не перевищує 15 см у діаметрі. Наївний баєсівський класифікатор вважає, що кожна з цих «ознак» (червоний, круглий, діаметр 15 см) вносить незалежний внесок у ймовірність того, що фрукт – це яблуко, незалежно від будь-яких кореляцій між ознаками. Але насправді функції не завжди незалежні, а їхні параметри не

завжди пов'язані один з одним. Саме це є недоліком методу Баєса, і тому його називають «наївним».

Раніше всі спам-фільтри працювали на алгоритмі наївного Баєса. Машина вважала скільки разів слово «кредит» зустрічається в спамі, а скільки разів на нормальних листах. Множила ці дві ймовірності за формулою Баєса, складала результати всіх слів і бац, всім лежати, у нас машинне навчання!



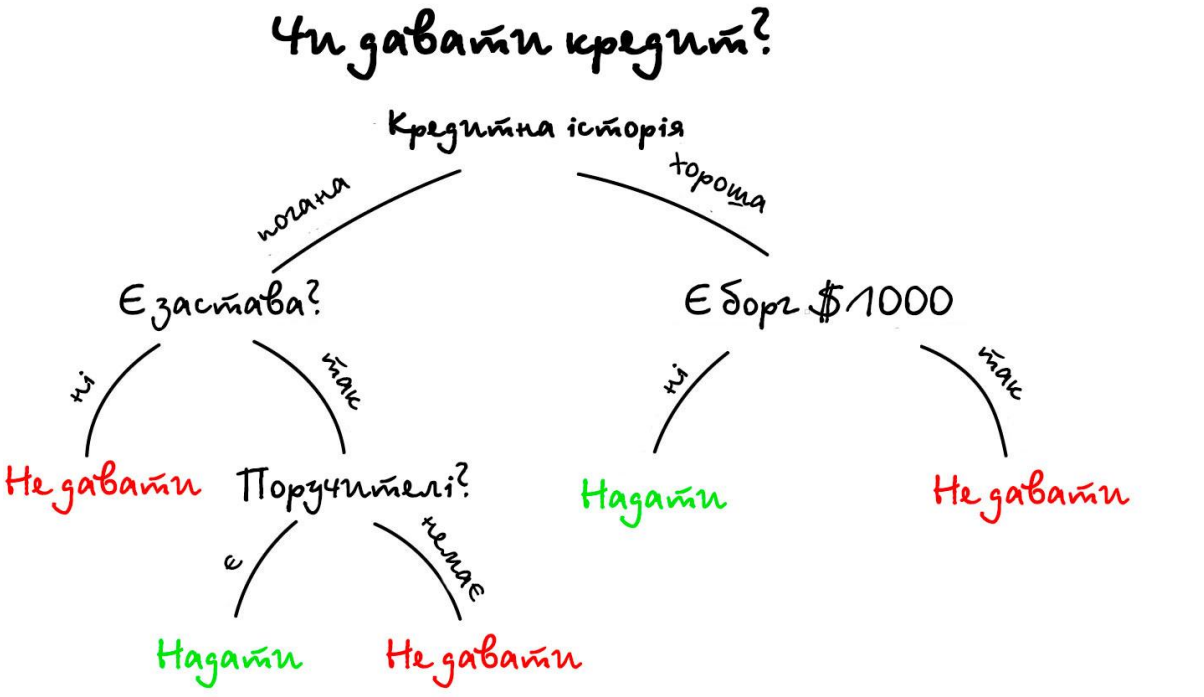
Пізніше спамери навчилися обходити фільтр Байеса, просто вставляючи в кінець листа багато слів з «хорошими» рейтингами. Метод отримав іронічну назву Отруєння Баєса, а фільтрувати спам стали іншими алгоритмами. Але метод назавжди залишився в підручниках як найпростіший, красивий і один з перших практично корисних.

3.2. Дерево рішень

Візьмемо інший приклад корисної класифікації. Ось берете ви кредит в банку. Як банку упевнитися повернете ви його чи ні? Дуже точно ніяк, але банк має тисячі профілів інших людей, які вже брали кредит до вас. Там вказано їхній вік, освіту, посаду, рівень зарплати та головне - хто з них повернув кредит, а з ким виникли проблеми.

Отож, всі здогадалися, де тут дані і який треба передбачити результат. Навчимо машину, знайдемо закономірності, отримаємо відповідь - питання не в цьому. Проблема в тому, що банк не може сліпо довіряти відповіді машини, без пояснень. Раптом збій, злі хакери або бухий адмін вирішив скриптик виправити.

Для цієї задачі придумали Дерево Рішень. Машина автоматично розділяє всі дані за запитаннями, відповідь на які «так» або «ні». Питання можуть бути не зовсім адекватними з точки зору людини, наприклад «зарплата позичальника більше, ніж 25934 гр.?», але машина придумує їх так, щоб на кожному кроці розбиття було найточнішим.



Дерево Рішень

Так отримується дерево питань. Чим вищий рівень, тим загальніше запитання. Потім навіть можна загнати їх аналітикам і вони понавидумують чому саме так.

Дерева знайшли свою нішу в областях з високою відповідальністю: діагностиці, медицині, фінансах.

Два найпопулярніших алгоритми побудови дерев - CART і C4.5 .

У чистому вигляді дерева сьогодні використовують рідко, але ось їх ансамблі (про які буде пізніше) лежать в основі великих систем і часто

обскакують навіть нейромережі. Наприклад, коли ви задаєте запитання Ґуґлу, то саме натовп дурних дерев біжить ранжувати вам результати.

Моделі дерева рішень будуються в два етапи: індукція і відсікання. Індукція - це те, де ми будуємо дерево, тобто встановлюємо всі межі ієрархічного рішення, ґрунтуючись на наших даних. За своїм характером дерева рішень можуть бути схильні до значного перенавчання. Відсікання - це процес видалення непотрібної структури з дерева рішень, ефективно спрощуючи його для розуміння та уникнення перенавчання.

Індукція. Індукція дерева рішень проходить 4 основні етапи побудови:

Почніть з навчального набору даних, в якому повинні міститися ознаки змінних і результати класифікації або регресії.

Визначте поняття *«найкраща реклама»* в наборі даних для їх розбиття. Про те, як визначити це поняття *«найкраща реклама»* поговоримо пізніше.

Розбийте дані на підмножини, які будуть містити можливі значення для кращої ознаки. Таке розбиття в основному визначає вузол на дереві, тобто кожен вузол - це розділена точка, заснована на певній ознаці з наших даних.

Рекурсивно згенеруйте нові вузли дерева за допомогою підмножини даних, створених на 3 етапі. Продовжуйте розбиття до тих пір, поки не досягнете точки, на якій буде знаходиться оптимізована якимось способом максимальна точність. Намагайтеся мінімізувати кількість розбиття і вузлів.

Перший етап простий. Просто зберіть свій набір даних!

На другому етапі вибір ознаки і певного розбиття зазвичай здійснюється за допомогою алгоритму для зменшення функції вартості. Якщо подумати, розбиття при побудові дерева рішень еквівалентно розбиттю простору ознак. Ми будемо кілька разів пробувати різні точки розбиття, а в кінці виберемо ту, яка має найменшу вартість. Звичайно, можна виконати пару розумних речей, таких як розбиття тільки в діапазоні значень в нашому наборі даних. Це дозволить скоротити кількість обчислень для тестування точок розбиття, які є свідомо марними.

Для дерева регресії можна використовувати простий квадрат помилки в якості функції вартості:

$$E = \sum (Y - \widehat{Y})^2$$

Де Y - це достовірні дані, а Y з шапкою - це прогнозоване значення. Ми підсумовуємо по всіх вибірках в нашому наборі даних, щоб отримати загальну помилку. Для класифікації використовуємо функцію коефіцієнта Джині:

$$E = \sum (p_k * (1 - p_k))$$

Де p_k - це частка навчальних прикладів класу k в певному вузлі прогнозування. В ідеалі вузол повинен мати значення помилки, що дорівнює нулю, що означає, що кожне розбиття виводить один клас 100% часу. Це саме те, що нам потрібно, так як діставшись до конкретно цього вузла прийняття рішення, ми будемо знати, яким буде висновок в залежності від того, на якій стороні кордону ми знаходимося.

Якби нам довелося вибрати розбиття, на якому кожен вихід має різні класи в залежності від вхідних даних, тоді б ми не добули ніякої інформації. Ми не дізналися б більше про те, чи впливає конкретний вузол, тобто функція, на класифікацію даних! З іншого боку, якщо наше розбиття має високий відсоток кожного класу для кожного виходу, то ми добули інформацію про те, що розбиття таким конкретним чином на цю конкретну змінну дає нам конкретний вихід!

Тепер ми могли б продовжити розбиття до тих пір, поки у нашого дерева не з'являться тисячі гілок ... Але це погана ідея! Наше дерево рішень було б величезним, повільним і перенавчання для нашого навчального набору даних. Тому ми встановимо деякі заздалегідь певні критерії зупинки, щоб припинити побудову дерева.

Найбільш поширеним методом зупинки є використання мінімального розрахунку за кількістю навчальних прикладів, призначених для кожної вершини дерева. Якщо число менше деякого мінімального значення, то розбивка не рахується і вузол призначається кінцевою вершиною дерева. Якщо всі вершини дерева стають кінцевими, то навчання припиняється. Чим менше мінімальне число, тим точніше буде змінити та, відповідно, ви отримаєте більше інформації. Але в такому випадку мінімальне число схильне до перенавчання навчальними даними. Занадто велика кількість мінімальних чисел призведе до зупинки навчання занадто рано. Таким чином, мінімальне значення зазвичай встановлюється на основі даних в залежності від того, скільки прикладів очікується в кожному класі.

Відсікання. Через свій характер навчання дерева рішень можуть бути схильні до значного перенавчання. Вибір правильного значення для мінімальної кількості прикладів на вузол може виявитися складним завданням. У багатьох випадках можна було б просто піти з безпечного шляху і зробити цей мінімум дуже маленьким. Але в такому разі, у нас була б величезна кількість розбиття і, відповідно, складне дерево. Справа в тому, що багато з гілок розбиття виявляться зайвими і не допоможуть збільшити точність моделі.

Відсікання гілок дерева - це метод, який скорочує кількість розбиття за допомогою видалення, тобто відсікання, непотрібних гілок розбиття дерева. Відсікання узагальнює кордон рішень, ефективно зменшуючи складність дерева. Складність дерева рішень визначається кількістю розбиття.

Простий, але дуже ефективний метод відсікання відбувається знизу вгору через вузли, оцінюючи необхідність видалення певного вузла. Якщо вузол не впливає на результат, то він відсікається.

Переваги методу. Їх легко зрозуміти. У кожному вузлі ми можемо точно побачити, яке рішення приймає наша модель. На практиці ми зможемо точно дізнатися, звідки виходять точності і помилки, з якими видами даних модель буде справлятися і як значення ознак впливають на вихід. Опція візуалізація в Scikit-learn є зручним інструментом, що сприяє хорошому розуміння дерев рішень.

Не вимагає об'ємної підготовки даних. Багато моделей машинного навчання вимагають попередньої обробки даних (наприклад, нормалізації) і потребують складних схемах регуляризації. З іншого боку, дерева рішень ефективні після настройки деяких параметрів.

Вартість використання дерева для виведення є логарифмічною від числа точок даних, що використовуються для навчання дерева. Це є великою перевагою, так як велика кількість даних не сильно вплине на швидкість виведення.

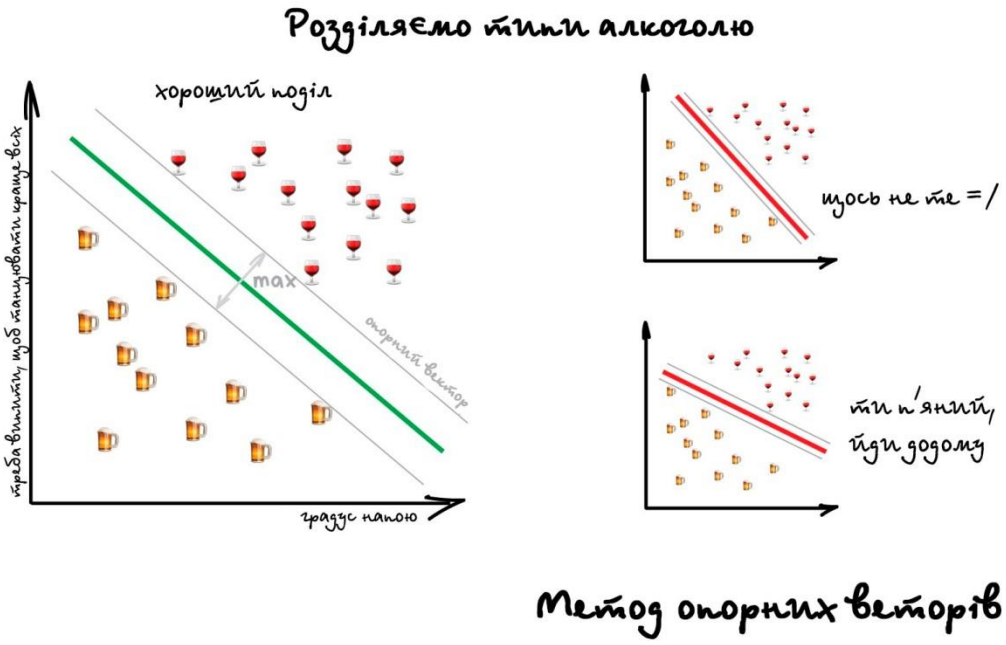
Недоліки методу. Через свій характер навчання дерева рішень схильні до перенавчання. Рекомендується часто застосовувати деякі види зниження розмірності, наприклад, PCA, щоб дерево не створювало розбиття по великій кількості ознак.

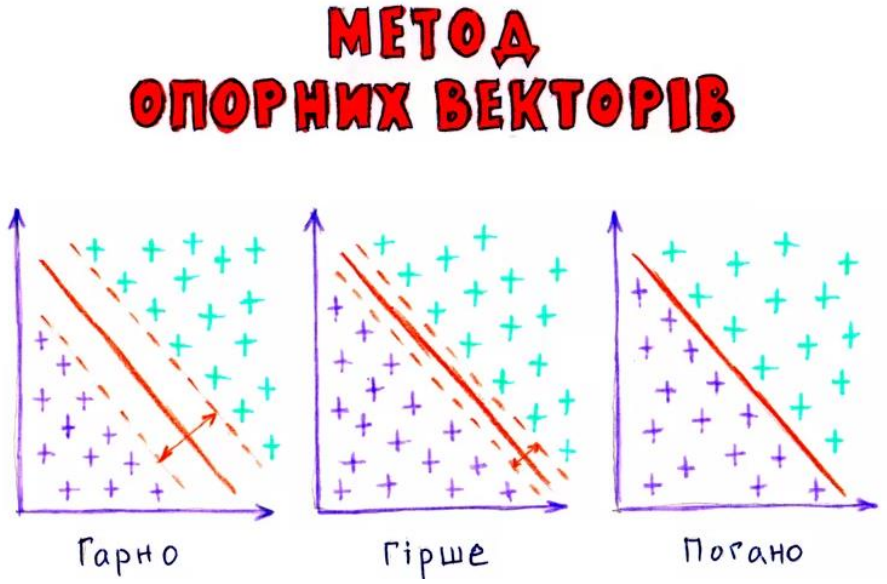
З тих же причин, що і перенавчання, дерева рішень також уразливі до зміщення класів, які є в більшості наборів даних. Хорошим рішенням в даному випадку є періодична балансування класів (ваги класу, вибірка, певна функція втрат).

3.3. Метод Опорних Векторів

Але найпопулярнішим методом класичної класифікації заслужено є Метод Опорних Векторів (SVM). Ним вже класифікували все, що тільки можна класифікувати: види рослин, обличчя на фотографіях, документи за тематиками... Багато років він був головною відповіддю на питання «який би мені взяти класифікатор».

Ідея SVM за своєю ідеєю проста - він шукає, як так провести дві прямі між категоріями, щоб між ними утворився найбільший зазор. На зображенні видно наочніше:

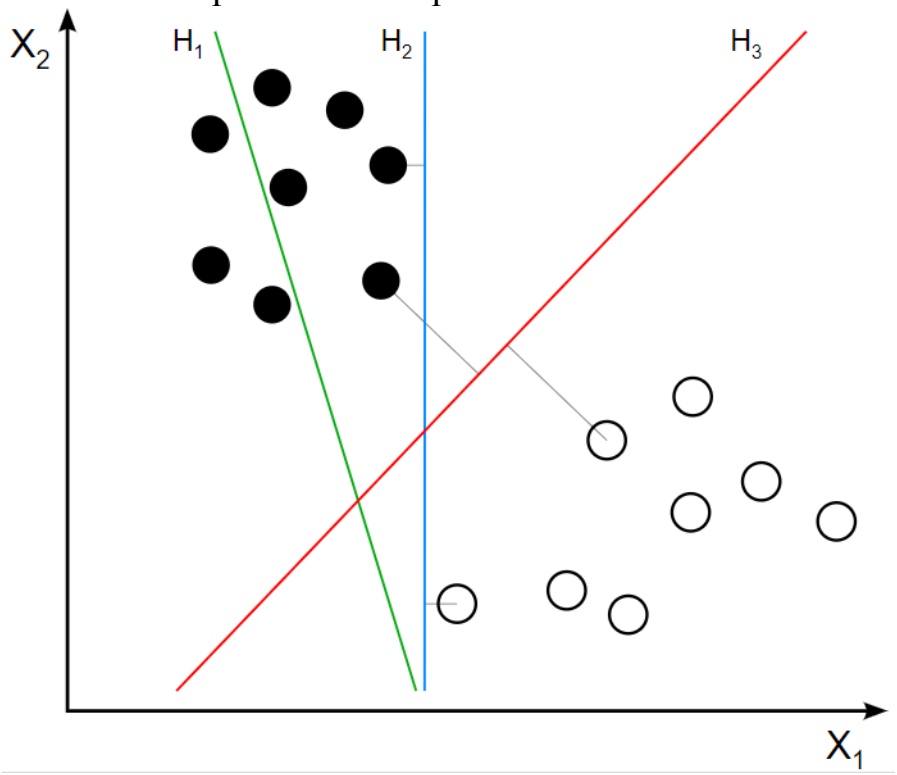




Згідно з доповіддю Всесвітньої організації інтелектуальної власності за 2019, кількість патентів у світі зростає саме в галузі застосування методу опорних векторів. Метод опорних векторів використовується для задач класифікації тексту, як-от призначення категорії і виявлення спаму. Також його використовують для розпізнавання зображень, він дуже популярний у багатьох сферах розпізнавання рукописних цифр, наприклад, у послугах поштової автоматизації, тобто розпізнавання поштового індексу відправника та отримувача.

Метод опорних векторів – найпопулярніший для класичної класифікації, тому що він простий та швидкий. Виходячи з того, що об'єкт, який перебуває в N-вимірному просторі, належить до одного з двох класів, метод опорних векторів будує гіперплощину з розмірністю (N – 1), щоб всі об'єкти виявилися в одній з двох груп. Що ж таке гіперплощина? Як простий приклад для завдання класифікації, що має тільки два класи, ви можете уявити собі

гіперплощину як лінію, яка розділяє і класифікує набір даних. До того ж, що далі від гіперплощини лежать наші точки даних, то більше ми впевнені в тому, що вони були правильно класифіковані. Що менше схожих ознак між даними, то менша ймовірність належності до одного класу. Тому ідеально, якщо точки даних перебувають якомога далі від гіперплощини, і до того ж залишаються на правильній стороні.



H1 не розділяє ці класи. H2 розділяє, але лише з невеликим розділенням. H3 розділяє їх із максимальним розділенням.

Головним недоліком методу є те, що він підходить тільки до розв'язання завдань з двома класами.

SVM є корисними для категоризації текстів та гіпертекстів, оскільки їхнє застосування може значно знижувати потребу в мічених тренувальних зразках як у стандартній індуктивній, так і в трансдуктивній[en] постановках.

Із застосуванням SVM може виконуватися й класифікація зображень. Експериментальні результати показують, що SVM можуть досягати значно вищої точності пошуку, ніж традиційні схеми уточнення запиту, всього лише після трьох-чотирьох раундів зворотного зв'язку про відповідність.

За допомогою ОВМ може здійснюватися розпізнавання рукописних символів.

3.4. Логістична регресія

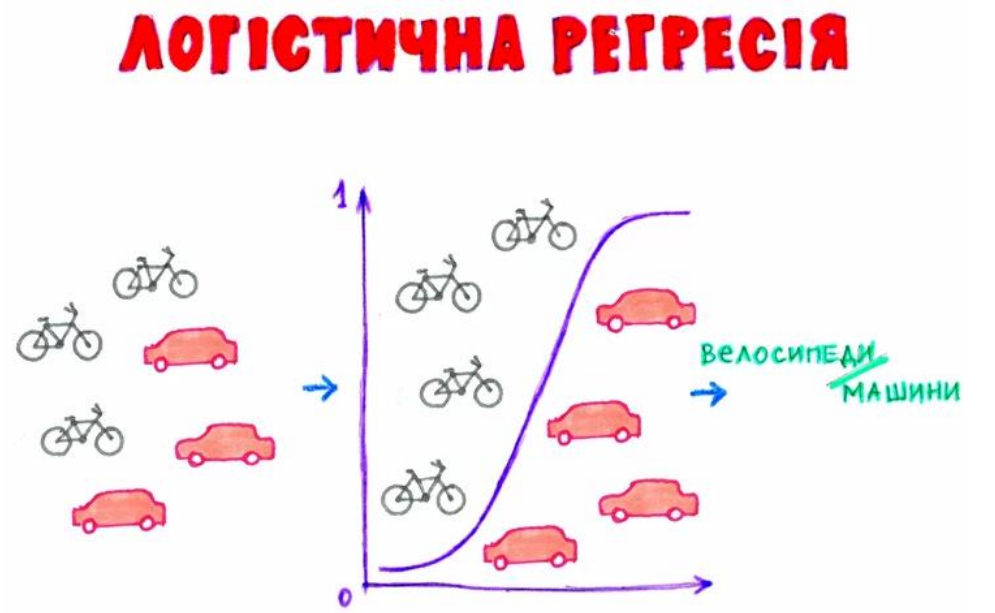
На Заході все більше розвивається сфера контролю стану здоров'я на основі постійного аналізу його активності. Пацієнт носить зчитувач фізичної активності та показників, вони відправляються його лікарю та аналізуються

разом з його аналізами та базою даних інших пацієнтів. Для цього зазвичай використовується логістична регресія.

Наприклад, маємо вибірку пацієнтів з показниками куріння, харчування, фізичної активності, вживання алкоголю. За допомогою логістичної регресії можемо побудувати модель, яка використовує ці чотири характеристики способу життя для передбачення наявності або відсутності ішемічної хвороби серця у вибірці пацієнтів. Використовувати модель можна, щоб дізнатися, наприклад, наскільки більше курці схильні до ішемічної хвороби серця, ніж некурці. Тобто, за отриманими значеннями, беремо пацієнта-курця та визначаємо, до якого класу він, ймовірно, належить: тих, хто має ішемічну хворобу серця, чи ні.

Логістичну регресію добре використовувати для завдань бінарної класифікації, тобто коли на виході потрібно отримати відповідь, до якого з двох класів належить об'єкт. Логістична регресія схожа на лінійну тим, що в ній теж потрібно знайти значення коефіцієнтів для вхідних змінних, але різниця в тому, що вихідне значення перетвориться за допомогою нелінійної або логістичної функції. Логістична функція перетворює будь-яке значення в число в межах від 0 до 1. Так і передбачається ймовірність належності до одного чи другого класу. Тобто, на відміну від логістичної регресії, тут не проводять передбачення значення числової змінної, виходячи з вибірки вихідних значень, а замість цього значенням функції є ймовірність того, що це початкове значення належить до певного класу.

До недоліків логістичної регресії можна віднести залежність від набору даних та низьку стійкість до помилок.



Логістична регресія - це статистичний інструмент, спрямований на моделювання біноміального результату з однією або кількома пояснювальними змінними.

За допомогою статистичних методів логістична регресія дозволяє генерувати результат, який насправді представляє ймовірність того, що задане вхідне значення належить даному класу.

У задачах двочленної логістичної регресії ймовірність того, що вихід належить одному класу, буде P, тоді як він належить іншому класу 1-P (де P - число між 0 і 1, оскільки воно виражає ймовірність).

Прикладами проблем, які можна вирішити логістичною регресією, є:

електронний лист є спамом чи ні;

покупка в Інтернеті є шахрайською чи ні, оцінюючи умови покупки;

у пацієнта перелом, оцінюючи його радіуси.

За допомогою логістичної регресії ми можемо робити прогнозний аналіз, вимірюючи взаємозв'язок між тим, що ми хочемо передбачити (залежною змінною), і однією або декількома незалежними змінними, тобто характеристиками. Оцінка ймовірності здійснюється за допомогою логістичної функції.

В подальшому ймовірності перетворюються на двійкові значення, і для того, щоб зробити прогноз реальним, цей результат присвоюється класу, якому він належить, виходячи з того, близький він чи ні до самого класу.

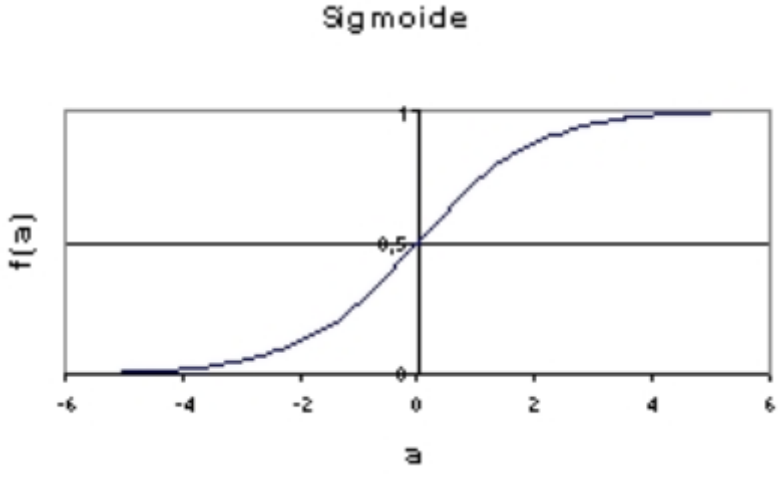
Наприклад, якщо застосування логістичної функції повертає 0,85, то це означає, що вхідні дані генерують позитивний клас, присвоюючи його класу 1. І навпаки, якщо він отримав таке значення, як 0,4 або більш загально <0,5 ..

Логістична функція, яка також називається сигмоїдною, - це крива, здатна приймати будь-яке число реальної величини і відображати її до значення між 0 і 1, виключаючи крайності. Функція:

$$f(x) = \frac{1}{1+e^{-(b0 + b1*x)}}$$

де: e: основа природних логарифмів (число Ейлера або функція excel exp ())

b0 + b1 * x: фактичне числове значення, яке ви хочете перетворити.



Представлення, що використовується для логістичної регресії

Логістична регресія використовує рівняння як подання, подібно до лінійної регресії

Нижче наведено ще один приклад рівняння логістичної регресії:

y = e ^ (b0 + b1 * x) / (1 + e ^ (b0 + b1 * x))

Де:

y - залежна змінна, тобто передбачуване значення;

b0 - термін поляризації або перехоплення;

b1 - коефіцієнт для одного вхідного значення (x).

Кожен стовпець у вхідних даних має пов'язаний b коефіцієнт (постійне реальне значення), який необхідно засвоїти з навчальних даних.

Фактичне представлення моделі, яку ви б зберігали в пам'яті або файлі, є коефіцієнтами рівняння (значення бета або b).

Коефіцієнти (бета або b значення) алгоритму логістичної регресії оцінюються на етапі навчання. Для цього ми використовуємо максимальну оцінку ймовірності.

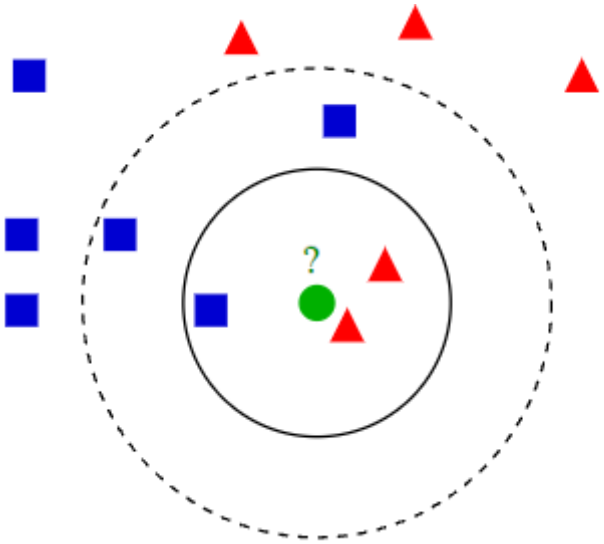
Максимальна оцінка ймовірності - це алгоритм навчання, який використовується декількома алгоритмами машинного навчання. Коефіцієнти, отримані в результаті моделі, передбачають значення, близьке до 1 (наприклад, Чоловік) для класу за замовчуванням, а значення, дуже близьке до 0 (наприклад, Жіноче) для іншого класу. Максимальна ймовірність логістичної регресії - це процедура знаходження значень коефіцієнтів (значень бета або b), що мінімізують похибку ймовірностей, передбачену моделлю відносно даних у даних (наприклад, ймовірність 1, якщо дані є первинним класом).

3.5. Метод k-найближчих сусідів

Ме́тод k-найбли́жчих сусі́дів (англ. k-nearest neighbor method) — простий непараметричний класифікаційний метод, де для класифікації об'єктів у рамках простору властивостей використовуються відстані (зазвичай евклідові), пораховані до усіх інших об'єктів. Вибираються об'єкти, до яких відстань найменша, і вони виділяються в окремий клас.

Метод k-найближчих сусідів — метричний алгоритм для автоматичної класифікації об'єктів. Основним принципом методу найближчих сусідів є те, що об'єкт приписується класу, найпоширенішому серед його сусідів. Сусіди беруться, виходячи з множини об'єктів, класи яких уже відомі, і, виходячи з ключового для даного методу значення k, вираховується найчисленніший серед них клас. Кожен об'єкт має кінцеву кількість атрибутів (розмірностей). Передбачається, що існує певний набір об'єктів з уже наявною класифікацією.

Прокляття розмірності. Цей класифікатор робить припущення, що подібні точки мають подібні мітки. На жаль, у високорозмірнісних просторах, точки, вибрані з розподілу ймовірностей, майже ніколи не опиняються близько одна до одної.



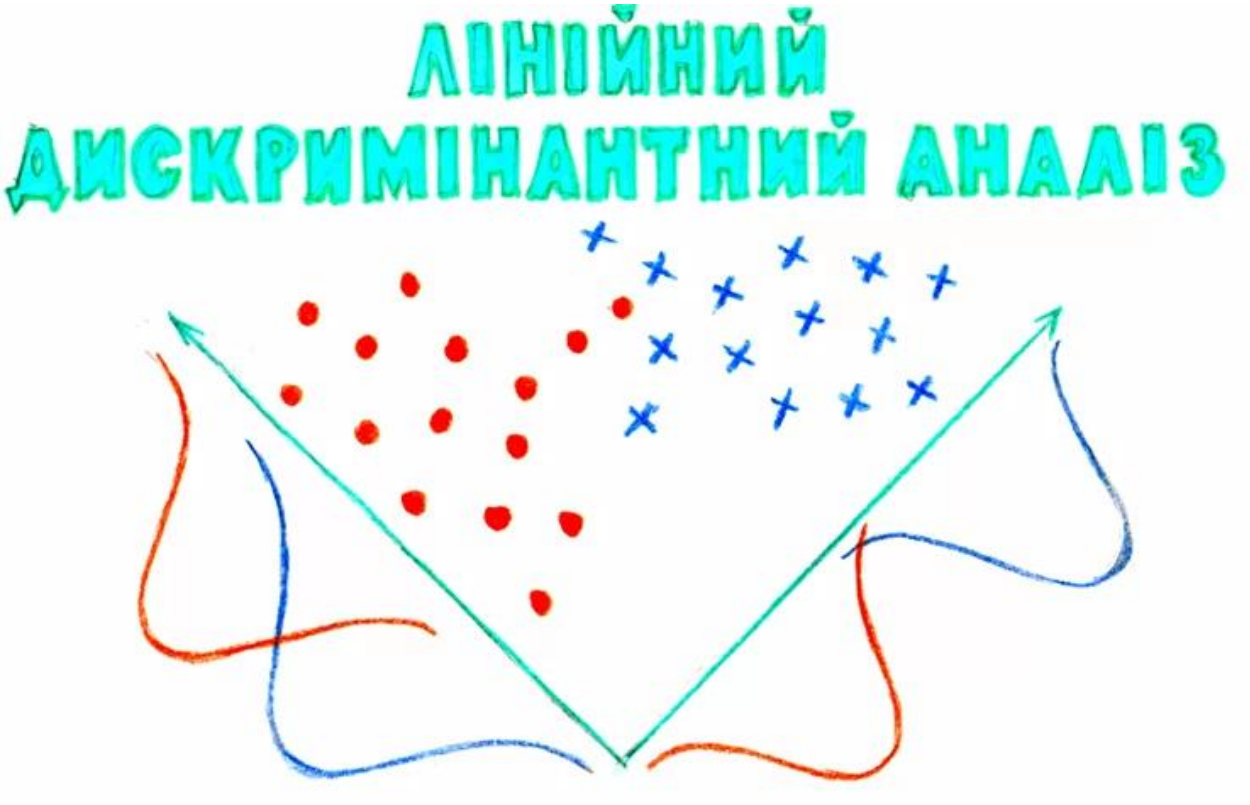
Приклад класифікації k найближчих сусідів. Тестовий зразок (зелене коло) повинен бути класифікований як синій квадрат (клас 1) або як червоний трикутник (клас 2). Якщо k = 3, то класифікується як 2-й клас, тому що всередині меншого кола 2 трикутника і тільки 1 квадрат. Якщо k = 5, то він буде класифікований як перший клас (3 квадрата проти 2-ох трикутників всередині більшого кола).

3.6. Лінійний дискримінантний аналіз

Якщо класів більше, ніж два, то краще використовувати лінійний дискримінантний аналіз. Наприклад, лікарю потрібно визначити діагноз пацієнта. У нього є множина значень аналізів для кожного можливого захворювання. Лікар бере ці аналізи у пацієнта і порівнює їхні значення зі значеннями для захворювань. Так визначають ймовірності кожного з переліку захворювань у пацієнта.

Метод містить статистичні властивості даних, розраховані для кожного класу. Під час цього передбачається, що дані мають нормальний розподіл, тому перед початком роботи необхідно видалити з даних аномальні значення. Варто зазначити, що, на відміну від цього методу, в інших нормальний розподіл не обов'язковий. Прогнозні класи знаходять шляхом обчислення дискримінантного значення для кожного класу, тобто знаходження лінійної залежності комбінації значень для вибору класу, а потім обирають клас із найбільшим значенням.

Головна перевага методу – це простота реалізації та інтерпретації результатів, недолік – чутливість до розподілу вхідних даних, коли навіть невелике їхнє змінення призводить до значних змін результатів класифікації.



У класифікації є корисна зворотна сторона - пошук аномалій. Коли якась ознака об'єкта аж занадто не вписується в наші класи, ми яскраво підсвічуємо його на екрані. Зараз так роблять в медицині: комп'ютер підсвічує лікарю всі підозрілі області МРТ або виділяє відхилення в аналізах. На біржах таким же чином визначають нестандартних гравців, які швидше за все є інсайдерами. У мережах так виявляють атаки чи неадекватну роботу Навчивши комп'ютер «як правильно», ми автоматично отримуємо і зворотний класифікатор - як неправильно.

Сьогодні для класифікації все частіше використовують нейромережі, адже по-суті їх для цього і винайшли.

Правило таке: складніші дані - складніший алгоритм. Для тексту, цифр, табличок я б починав з класики. Там моделі менші, навчаються швидше і працюють зрозуміліше. Для картинок, відео та іншої незрозумілої бігдати - одразу б дивився в сторону нейромереж.

Років п'ять тому ще можна було зустріти класифікатор осіб на SVM, але сьогодні під цю задачу сотня готових сіток по інтернету валяються, чом би їх не взяти. А ось спам-фільтри як на SVM писали, так і не бачу сенсу зупинятися.

4. ОЦІНКА ЯКОСТІ КЛАСИФІКАЦІЇ

Провели ми класифікацію і що далі? Як оцінити наскільки якісно ми провели цю класифікацію? Чи правильно вона виконана? Чи достатньо нам

цього алгоритму, чи може необхідно застсувати якийсь інший? Відповіді на ці питання дають показники (метрики) якості класифікації.

Матриця помилок (або матриця неточностей, англ. Confusion matrix)

Перед переходом до самих метрик необхідно ввести важливу концепцію для опису цих метрик в термінах помилок класифікації - confusion matrix (матриця помилок). Припустимо, що у нас є два класи $y = \{0,1\}$ і алгоритм, який пророкує (передбачає) приналежність кожного об'єкта одному з цих класів. *Розглянемо приклад*. Нехай система захисту мережі використовує систему класифікації для виявлення атаки: нормальна робота мережі чи аномальна (наявність атаки). При цьому у першому випадку система і далі нормально працює, а у другому – видається сигнал аномальної роботи. Таким чином, виявлення неадекватної (аномальної) роботи мережі ($y = 1$) можна розглядати як "сигнал тривоги", що повідомляє про можливі ризики.

Будьякий реальний класифікатор робить помилки. У нашому випадку таких помилок може бути дві:

Нормальна ситуація у мережі за даними трафіка розпізнається моделлю як аномальна. Даний випадок можна трактувати як "помилкову тривогу".

Аномальна ситуація розпізнається як нормальна і ніяких дій по захисту від атаки не відбувається. Даний випадок можна розглядати як "пропуск цілі".

Неважко помітити, що ці помилки нерівноцінні по зв'язаних з ними наслідками. У разі "помилкової тривоги" втрати складуть тільки марно потрачений час та ресурси на протидію неіснуючій загрозі. У разі "пропуску цілі" можна втратити набагато більше (інформацію, роботу мережі та інше, це залежить від виду атаки). Тому системі захисту важливіше не допустити "пропуск цілі", ніж "помилкову тривогу".

Оскільки з точки зору логіки завдання виявлення аномалій нам важливіше правильно розпізнати аномалію (атаку) з міткою $y = 1$, ніж помилитися в розпізнаванні нормальної роботи мережі, будемо називати відповідний результат класифікації позитивним (аномалія чи атака виявлені вірно), а протилежний - негативним (аномалії чи атаки немає $y = 0$). Тоді можливі наступні чотири результати класифікації:

True Positive (TP) – наявність атаки класифікована як наявна атака, тобто позитивний клас розпізнано як позитивний. Спостереження, для яких це має місце називаються істинно-позитивними.

True Negative (TN) – нормальна робота мережі класифікована як нормальна робота без аномалій, тобто негативний клас розпізнано як негативний. Спостереження, яких це має місце, називаються істинно негативними.

False Positive (FP) – нормальна робота мережі класифікована як аномальна, тобто мала місце помилка, в результаті якої негативний клас був розпізнаний як позитивний. Спостереження, для яких було отримано такий результат класифікації, називаються помилково-позитивними, а помилка класифікації називається помилкою I роду.

False Negative (FN) – атака чи аномальна робота мережі розпізнана як нормальна, тобто мала місце помилка, в результаті якої позитивний клас був розпізнаний як негативний. Спостереження, для яких було отримано такий результат класифікації, називаються помилково-негативними, а помилка класифікації називається помилкою II роду.

Таким чином, помилка I роду, або хибно-позитивний результат класифікації, має місце, коли негативне спостереження розпізнано моделлю як позитивне. Помилкою II роду, або хибно-негативних результатом класифікації, називають випадок, коли позитивне спостереження розпізнано як негативне. Пояснимо це за допомогою матриці помилок класифікації:

	y = 1	y = 0
a (x) = 1	Істинно-позитивний (True Positive - TP)	Помилково-позитивний (False Positive - FP)
a (x) = 0	Помилково-негативний (False Negative - FN)	Істинно-негативний (True Negative – TN)

Тут a(x) - це відповідь алгоритму при конкретній ситуації, а y - справжня мітка класу для цієї ситуації. Таким чином, помилки класифікації бувають двох видів: False Negative (FN) і False Positive (FP). P означає що класифікатор визначає клас об'єкта як позитивний (N - негативний). T означає що клас передбачений правильно (відповідно F - неправильно). Кожен рядок в матриці помилок представляє прогнозований клас, а кожен стовпець - фактичний клас.

Тобто у загальному випадку, матриця неточностей - це матриця розміром N на N, де N – кількість класів, яка представляє собою табличне представлення прогнозованих і фактичних значень для кожного можливого класу.

Матриця помилок, одна з наважливіших речей, на яку потрібно дивитися при оцінці моделі класифікації. Це матриця, яка візуалізує кількість фактичних екземплярів класу в порівнянні з прогнозованими екземплярами класу. Таке подання дозволяє нам швидко побачити кількість правильних і неправильних прогнозів для кожної категорії.

На основі цієї матриці будується ряд інших характеристик. Розглянемо кожну з них більш детально.

Акуратність (англ. Accuracy).

Акуратністю називається пропорція точних прогнозів по відношенню до загальної кількості прогнозів, тобто це ймовірність того, що клас буде передбачений правильно (3.1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Тобто, accuracy - частка правильних відповідей алгоритму:

Хоча акуратність є швидким та інформативним індикатором продуктивності моделі, ми не можемо покладатися виключно на неї. Це пов'язано з тим, що вона приховує наявність зсуву в моделі, що є звичайним явищем, якщо набір даних незбалансований, тобто негативних моментів значно більше, ніж позитивних, або навпаки. Тобто, ця метрика марна в задачах з нерівними класами, що як варіант можна виправити за допомогою алгоритмів семпліювання. Семпліювання (англ. Data sampling) - метод коригування навчальної вибірки з метою балансування розподілу класів у вихідному наборі даних. Розглянемо це *на простому прикладі*.

Припустимо, ми хочемо оцінити роботу спам-фільтра пошти. У нас є 100 НЕ-спам листів, 90 з яких наш класифікатор визначив вірно (True Negative = 90, False Positive = 10), і 10 спам-листів, 5 з яких класифікатор також визначив вірно (True Positive = 5, False Negative = 5). Тоді accuracy:

$$Accuracy = \frac{5 + 90}{5 + 90 + 10 + 5} = 86{,}4$$

Але якщо ми просто будемо передбачати, що всі листи Не-спам, то отримаємо більш високу акуратність:

$$Accuracy = \frac{0 + 100}{0 + 100 + 0 + 10} = 90{,}9$$

При цьому, наша модель абсолютно не володіє ніякою прогностичною силою, оскільки спочатку ми хотіли визначати листи зі спамом. Подолати це нам допоможе перехід із загальної для всіх класів метрики до окремимих показників якості класів.

Точність (англ. Precision).

Точністю називається частка правильних відповідей моделі в межах класу - це частка об'єктів, що дійсно належать даному класу, щодо всіх об'єктів які система віднесла до цього класу.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Саме введення precision не дозволяє нам записувати всі об'єкти в один клас, так як в цьому випадку ми отримуємо зростання рівня False Positive.

Повнота (англ. Recall).

Повнота - це частка істинно позитивних класифікацій. Повнота показує, яку частку об'єктів, що реально належать до позитивного класу, ми передбачили вірно. Або ж іншими словами: це частка варіантів, класифікованих як позитивні, які насправді виявилися позитивними.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Повнота (recall) демонструє здатність алгоритму виявляти даний клас взагалі.

Маючи матрицю помилок, дуже просто можна обчислити точність і повноту для кожного класу. Точність (precision) дорівнює відношенню відповідного діагонального елемента матриці і суми всього рядка класу. Повнота (recall) - відношенню діагонального елемента матриці і суми всього стовпчика класу. Оскільки класів може бути багато (не обов'язково два), то формально:

$$Precision_{\text{c}} = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{c,i}}$$

$$Recall_{\text{c}} = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{i,c}}$$

Тобто, результуюча точність класифікатора розраховується як середнє арифметичне його точності по всіх класах. Те ж саме з повнотою.

F-міра (англ. F-score).

Precision і recall не залежить від співвідношення класів (на відміну від accuracy) і тому можуть бути застосовні в умовах незбалансованих вибірок. Часто в реальній практиці стоїть завдання знайти оптимальний (для замовника) баланс між цими двома метриками. Зрозуміло що чим вище точність і повнота, тим краще. Але в реальному житті максимальна точність і повнота недосяжні одночасно і доводиться шукати якийсь баланс. Тому, хотілося б мати якусь метрику яка об'єднувала б у собі інформацію про точність та повноту нашого алгоритму. У цьому випадку нам буде простіше приймати рішення про те, яку реалізацію запускати у виробництво (у кого більше той і крутіше). Саме такою метрикою є F-міра.

F-міра є гармонійним середнім між точністю і повнотою. Вона прагне до нуля, якщо точність або повнота прагне до нуля.

$$F = \frac{2 \times precision \times recall}{precision + recall} \tag{4}$$

Дана формула надає однакову вагу точності і повноти, тому F-міра буде падати однаково при зменшенні і точності і повноти. Можливо розрахувати F-міру надавши різну вагу точності і повноті, якщо ви свідомо віддаєте пріоритет одній з цих метрик при розробці алгоритму:

$$F_{\beta} = \frac{(1 + \beta^2) \times precision \times recall}{(\beta^2 \times precision) + recall} \tag{5}$$

де β приймає значення в діапазоні 0 <β <1 якщо ви хочете віддати пріоритет точності, а при β> 1 пріоритет віддається повноті. При β = 1 формула зводиться до попередньої і ви отримуєте збалансовану F-міру (також її називають $\boldsymbol{F_1}$).

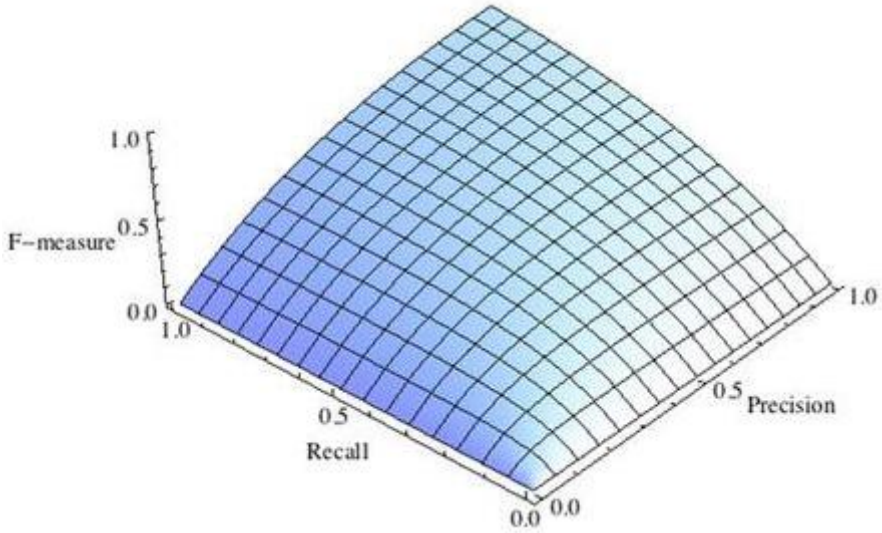


Рис.1 Збалансована F-міра, β=1

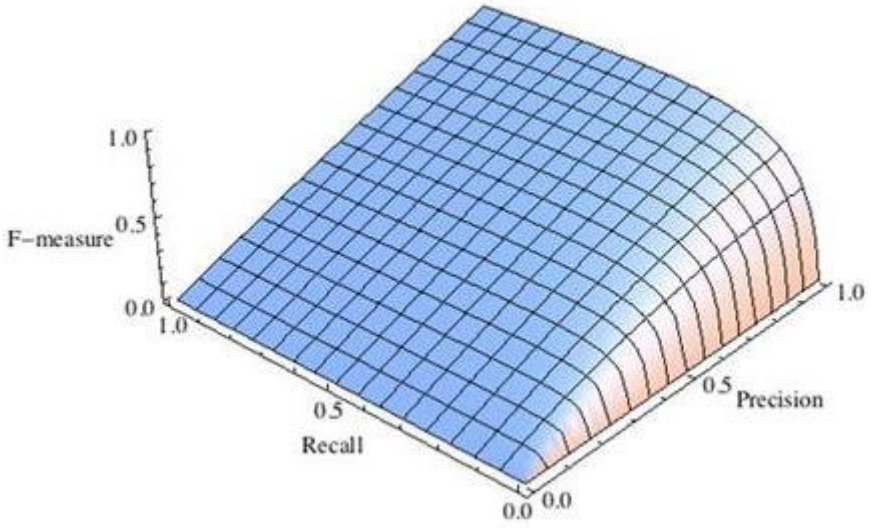


Рис.2 F-міра з приоритетом точності, $\beta^2=1/4$

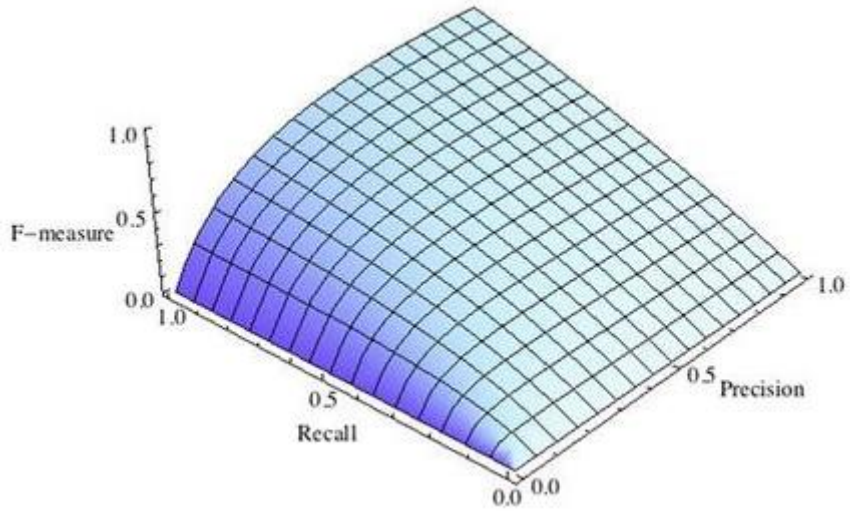


Рис.3 F-міра с пріоритетом повноти, $\beta^2=2$

F-міра досягає максимуму при максимальній повноті і точності, і близька до нуля, якщо один з аргументів близький до нуля.

F-міра є хорошим кандидатом на формальну метрику оцінки якості класифікатора. Вона зводить до одного числа дві інші основоположні метрики: точність і повноту. Маючи "F-міру" набагато простіше відповісти на питання: "змінився алгоритм в кращу сторону чи ні?"

ROC-крива

Крива робочих характеристик (англ. Receiver Operating Characteristics curve). Використовується для аналізу поведінки класифікаторів при різних порогових значеннях. Дозволяє розглянути всі порогові значення для даного класифікатора. Показує частку хибно позитивних прикладів (англ. False positive rate, FPR) в порівнянні з часткою істинно позитивних прикладів (англ. True positive rate, TPR).

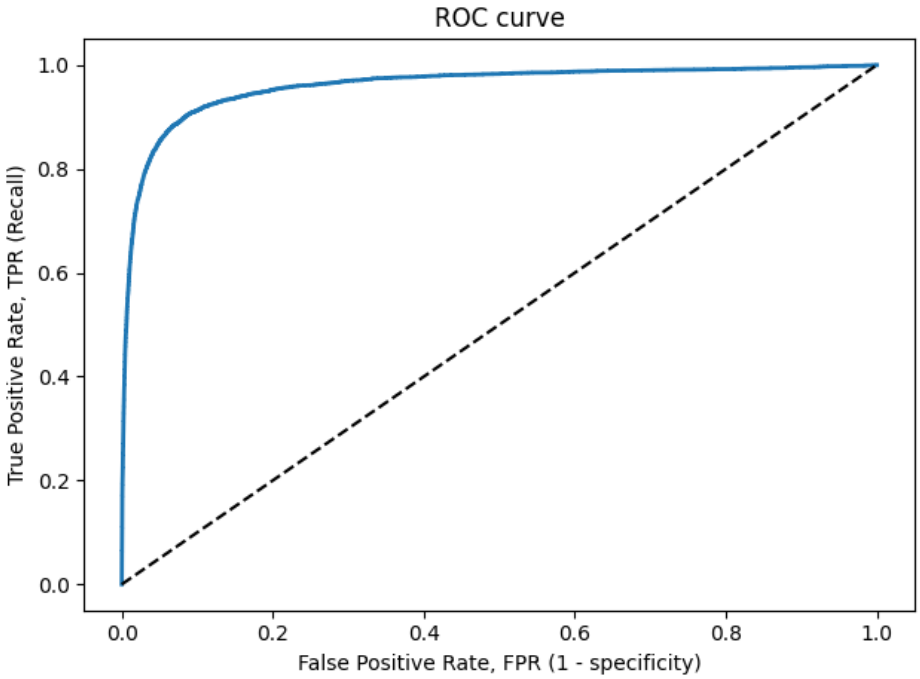


Рисунок 6 –ROC-крива

$$TPR = \frac{TP}{TP + FN} = Recall \quad (6)$$

$$FPR = \frac{FP}{FP + TN} \quad (7)$$

Частка FPR - це пропорція негативних зразків, які були некоректно класифіковані як позитивні.

$$FPR = 1 - TNR$$

де TNR - частка істинно негативних класифікацій (англ. Trie Negative Rate), що представляє собою пропорцію негативних зразків, які були коректно класифіковані як негативні.

Частка TNR також називається специфічністю (англ. Specificity). Отже, ROC-крива зображає чутливість (англ. Sensitivity), тобто повноту, в порівнянні з різницею 1 - specificity.

Пряма лінія по діагоналі представляє ROC-криву чисто випадкового класифікатора. Хороший класифікатор тримається від зазначеної лінії настільки далеко, наскільки це можливо (прагнучи до лівого верхнього кута).

Один із способів порівняння класифікаторів передбачає вимір площі під кривою (англ. Area Under the Curve - AUC). Бездоганний класифікатор матиме площу під ROC-кривою (ROC-AUC), що дорівнює 1, тоді як чисто випадковий класифікатор - площу 0.5.

Графік ROC допомагає прийняти рішення про те, де встановити поріг класифікації, щоб максимізувати істинно позитивний рівень або мінімізувати псевдопозитивний показник, що в кінцевому підсумку є бізнес-рішенням.

Precison-recall крива

Чутливість до співвідношення класів. Розглянемо задачу виділення математичних статей з безлічі наукових статей. Припустимо, що за все мається 1.000.100 статей, з яких лише 100 належать до математики. Якщо нам вдасться побудувати алгоритм *a(x),* що ідеально вирішує завдання, то його TPR буде дорівнює одиниці, а FPR - нулю. Розглянемо тепер поганий алгоритм, що дає позитивну відповідь на 95 математичних і 50.000 нематематичних статтях. Такий алгоритм абсолютно даремний, але при цьому має TPR = 0.95 і FPR = 0.05, що вкрай близько до показників ідеального алгоритму. Таким чином, якщо позитивний клас істотно менше за розміром, то AUC-ROC може давати неадекватну оцінку якості роботи алгоритму, оскільки вимірює частку невірно прийнятих об'єктів щодо загального числа негативних. Так, алгоритм *b(x)*, що поміщає 100 релевантних документів на позиції з 50.001-й по 50.101-ю, матиме AUC-ROC 0.95.

Позбутися від зазначеної проблеми з незбалансованими класами можна, перейшовши від ROC-кривої до Precison-recall (PR) PR-кривої. Вона визначається аналогічно до ROC-кривої, тільки по осях відкладаються НЕ FPR і TPR, а повнота (по осі абсцис) і точність (по осі ординат). Критерієм якості сімейства алгоритмів виступає площа під PR-кривою (англ. Area Under the Curve - AUC-PR).