## 4. Алгоритми – основни методи – Резюме

**Принцип на простотата- simplicity first**- Простите алгоритми често работят много добре. Има различни видове прости структури:

- Единичен attribute извършва цялата работа
- Всички **attributes** допринасят еднакво и независимо
- Логическа структура с малко **attributes** (**decision tree**)
- A set от прости логически правила
- Взаимоотношения между групи **attributes**
- Претеглена линейна комбинация на **attributes – linear regression**
- Силни връзки между съседни примери (на база разстояние)
- Групиране на данни (**clusters**)
- Агрегиране на **bags of instances**

Успехът на алгоритъма зависи от конкр. **domain** (област на прилож.)

**1R(One Rule) алгоритъм изгражда едно-степенно решаващо дърво- прост алгоритъм, който извлича основно правило за класификация:** Избира само **един атрибут**, който най-добре разделя данните. Създава **едно просто правило** въз основа на този атрибут. **Стъпки:** За всеки атрибут: Прави клон за всяка негова стойност. За всеки клон избира **най-често срещания клас**: Изчислява **процента на грешки** (колко примера са класифицирани грешно). Избира атрибута с **най-нисък процент на грешки**.

**Псевдокод:** Преглеждаме атрибутите един по един и изчисляваме грешките. Избираме този с **най-малка грешка**. **Липсващи стойности:** Те се приемат като **отделна стойност** на атрибута.

**Числови атрибути:**

**Discretization** – разделяне на числата на групи или интервали:

1. Определяне на **breakpoints** (граници) там, където **class** се променя. Границите между интервалите се избират там, където класовете се сменят. Ако повечето хора под 18 г. не карат кола, а тези над 18 карат, логично е да сложим граница на 18 г.
2. Намаляване на **грешките при класификация -** Добре избраните интервали намаляват шанса от грешки. Ако разделим числата твърде детайлно, може да създадем грешни правила.

**Проблем с overfitting-** моделът „учи“ твърде много детайли и случайности от тренировъчните данни, вместо само важните закономерности. Много добър е за тези данни, но слаб при нови.

- Чувствителност към шум в данните- Ако има грешки или случайни вариации в данните, моделът може да ги възприеме като важни и да вземе неправилни решения.
- Единична грешна стойност може да създаде отделен **interval -** при групиране на възрасти една грешно въведена стойност (200 г.) ще създаде нова категория, моделът прави грешни прогнози.
- Решение: задаване **минимален брой примери в даден клас** на интервал- Вместо да създаваме интервал за всяка уникална стойност, изискваме поне няколко примера (5) в един клас

**Прост вероятностен/Simple probabilistic/** Naïve Bayes/ Наивен Байесов Класификатор /- използва всички attributes (за разлика от 1R, който използва само един). Две основни предположения:1)всеки атрибут се смята за еднакво важен.2) Условна независимост на атрибутите – ако класът е известен, стойностите на атрибутите не зависят една от друга. Второто предположение рядко е напълно вярно, но алгоритъмът работи добре на практика, особено при текстова класификация и спам филтри.

**Bayes' Rule – Байесова теорема** - изчислява вероятността даден обект да принадлежи към даден клас, като използва наблюдаваните

атрибути. Тя изчислява вероятността на събитие H (клас), като използва наблюдавано доказателство E (стойностите на атрибутите):

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

P(H|E) – вероятността обектът да е от клас H, дадени E атрибути.

P(E|H) – вероятността тези атрибути да се появят при клас H - колко често тези атрибути се срещат при този клас.

P(H) – колко често се среща самият клас в данните (априорна вероятност).

P(E) – обща вероятност на атрибутите (игнорира се при сравнение между класове, защото е еднаква).

**Наивно предположение:** В модела Naïve Bayes се прави предположението, че атрибутите са независими един от друг, което улеснява изчисленията-стойностите на атрибутите не влияят помежду си, ако знаем класа. Изчисление:

$$P(Class|Attributes) = P(Attr_1|Class) \times P(Attr_2|Class) \times \cdots \times P(Class)$$

**Zero-frequency проблем** - дадена стойност на атрибут не се среща с определен клас в обучаващите данни. Това води до вероятност нула, което прави невъзможно класифицирането на нови примери, съдържащи тази комбинация. **Решение:** Използваме **Laplace estimator (Корекция на Лаплас)**, който добавя 1 към всички възможни комбинации между стойност на атрибут и клас и така не получаваме нула, а малка вероятност. Това прави **Наивния Байес по-устойчив и надежден**, особено при малки и непълни данни.

**Обработка на липсващи стойности:** При обучение: липсващите стойности не се броят; При класификация: пропускат се при изчисл

**Числови атрибути-** Числовите атрибути са стойности, които могат да бъдат измерени и изразени с числа (възраст, височина, температура). Обикновено се приема, че тези атрибути следват **нормално разпределение (Гаусово разпределение)** - повечето стойности са близо до средната стойност, а по-малко наблюдения са разположени в крайните стойности (далече от средното). **Нормалното разпределение** се описва с **средно аритметично (mean, μ)** и **стандартно отклонение (σ):**

**Средно аритметично (μ):** средната стойност на всички данни.

**Стандартно отклонение(σ):** Измерва разсейването на стойностите около средното. Колкото по-голямо е стандартното отклонение, толкова по-широко са разпръснати стойностите.

Функцията на плътността на вероятността (**probability density function, PDF**) за нормално разпределение е:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

x - стойността на атрибута; μ - средното аритметично; σ- стандартното отклонение; e- основата на естествения логаритъм-константа (прибл. 2.718); π е математическата константа (3.1416)

**Средно аритметично** (Mean, μ) - показател за централната тенденция на даден набор от данни. То представлява **сумата на всички стойности, разделена на броя на тези стойности**.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- μ - **средното аритметично;** n - броят на елементите в набора от данни; $x_i$ - стойностите в този набор

**Стандартно отклонение (σ)- колко са разпръснати (различни) стойностите на даден набор от данни спрямо средното аритметично**. Малко стандартно отклонение означава, че повечето стойности са **близки** до средното, докато голямо стандартно отклонение означава, че стойностите са **разпръснати**.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

- σ - стандартното отклонение; n - броят на стойностите; $x_i$ - отделните стойности; μ - средното аритметично

**Плътност на вероятността** (PDF) показва как е "разпределена" вероятността за различни стойности на непрекъсната случайна величина. Примери за непрекъснати величини/температура; време за изчакване; разстояние/. Тези стойности не са цели числа, а могат да бъдат всякакви числа в интервал ( 21.3°C, 5.8 минути и т.н.).Важно: PDF не казва точната вероятност за една стойност (напр. точно 22°C), защото тя е почти винаги нула. Вместо това, PDF показва в кои области от числата има повече или по-малко вероятност да се падне стойността. Пр:Ако имаме PDF за температура: Ще видим, че има по-голяма вероятност температурата да е между 20°C и 25°C, отколкото между 30°C и 35°C. Това ни помага да разберем в кой интервал най-често попадат стойностите.

**Multinomial Naïve Bayes I -** Това е версия на Naïve Bayes, използвана за класификация на документи чрез модела bag of words.

$\mathbf{n_1,n_2,...,n_k}$ – брой пъти, в които дадена дума i се среща в документа

$\mathbf{P_1,P_2,...,P_k}$ – вероятност да се получи думата i, ако случайно избираме от документи в класа H

Формула за вероятността на даден документ E, принадлежащ към клас H (базирана на Multinomial distribution):

$$P(E|H) = \frac{(n_1 + n_2 + ... + n_k)!}{n_1! \cdot n_2! \cdots n_k!} P_1^{n_1} P_2^{n_2} \cdots P_k^{n_k}$$

!Тази формула **игнорира дължината на документа**, тъй като се предполага, че тя е **константна** за всички класове.

**Multinomial Naïve Bayes II** –Пример: Имаме речник с две думи: yellow и blue. Вероятности за клас H:

- $P(yellow|H) = 75\%$
- $P(blue|H) = 25\%$

Ако **документът E** съдържа думите **“blue yellow blue”**, вероятността му е:

$$P(E|H) = \frac{3!}{2!1!}(0.75)^1(0.25)^2 = 3 \times 0.75 \times 0.0625 = 0.1406$$

Ако друг клас **H′** има:

- $P(yellow|H') = 10\%$
- $P(blue|H') = 90\%,$

тогава можем да изчислим **P(E|H′)** и да приложим **Bayes’ rule** за окончателна класификация. **Факториалите в горната формула не се изчисляват реално, защото отпадат в крайните изчисления.**

Naïve Bayes: **Работи добре**, дори ако предположението за независимост на атрибутите е **грубо нарушено**. **Защо? →** Класификацията не изисква **абсолютно точни вероятностни оценки**, а само коректно **ранжиране на класовете**. **Проблем:** Ако добавим **твърде много дублиращи се атрибути**, алгоритъмът може да се **объркa**. **За числови атрибути:** Те често не следват **normal distribution → Kernel Density Estimators** може да са по-добър избор.

**Constructing Decision Trees** (Конструиране на Дървета за Решения)

**Top-down learning** чрез рекурсивен метод **Divide-and-Conquer**: **Избор на атрибут за корен; Създаване на клонове** за всяка стойност на атрибута; **Разделяне на примери в подмножества; Рекурсия:** Прилагане на същата стратегия върху всяко подмножество; **Спиране на процеса**, когато всички примери принадлежат към един и същи клас

Критерий за избор на атрибут: Искаме най-малкото дърво → Избираме атрибута, който създава най- "чистите" възли.

**Използван критерий: Information Gain (IG)- помага да изберем най-полезния атрибут, който най-добре разделя данните за класификация.**

$$IG(A) = Entropy(parent) - \sum \left( \frac{|subset|}{|parent|} \times Entropy(subset) \right)$$

- **Entropy(parent)** – ентропията на целия набор от данни (преди да разделим по атрибут A).
- **subset / parent** – каква част от данните попада във всяко подмножество (след разделяне по A).
- **Entropy(subset)** – ентропията на всяко подмножество.

- **Σ (сума)** – събиране на всички резултати от подмножествата.

"Колко несигурност намаляваме, ако разделим данните по атрибут A?" Колкото по-голяма е стойността на **IG(A)**, толкова по-добре **атрибутът разделя данните**.

IG(A) = Ентропия(преди разделяне) - средна ентропия(след разделяне)

**Information Gain** се увеличава, когато подмножествата са **по-чисти**. **Избираме атрибута с най-голям Information Gain**.

Изчисляване на Information Gain - Използва се **Entropy** (ентропия-1 bit) като **мярка за неопределеност – мярка за несигурност или хаос в набор от данни S.**

$$Entropy(S) = - \sum P(c) \times \log_2 P(c)$$

**P(c)** – вероятността за всеки клас **c** в набора от данни. **$\log_2$ P(c)** – логаритъм по база 2 на тази вероятност. ∑ – сумира се за всички възможни класове. Минусът отпред гарантира, че резултатът е положителен. $\log_2$ **колко пъти трябва да умножим 2 със себе си**, за да получим това число

Ентропията измерва **колко са разбъркани данните** –Ако всички примери са от един клас → ентропия = 0 (няма несигурност).Ако класовете са равномерно разпределени → висока ентропия (много несигурност).

**Максимална ентропия:** всички класове са **равновероятни**.

**Минимална ентропия: един клас доминира** (вероятност=1).

**Gain Ratio (Коригиран Information Gain)**

**Проблем:** Атрибути с **много различни стойности** (като **ID**) могат да доведат до **изкуствено висок Information Gain**, което води до **overfitting**. **Решение: Gain Ratio (GR)- Коефициент на печалба**

$$GR(A) = \frac{IG(A)}{IntrinsicInfo(A)}$$

**Information Gain – колко несигурност премахваме след разделяне. IntrisicInfo – колко е "раздробена" информацията след разделяне (мери колко стойности има атрибутът). Защо се използва: Предотвратява избора на атрибути с много уникални стойности (напр. ID номера). Дава по-обективен избор на атрибут за разделяне.**

**Intrinsic Info (A) – вътрешна информация на атрибута- Тази формула измерва колко “раздробено” е разделението на данните при даден атрибут A. IntrinsicInfo(A) = мярка за това колко части (подмножества) прави атрибутът - колко “раздробено” е разделението на данните при даден атрибут A.**

$$IntrinsicInfo(A) = -\sum \left( \frac{|subset|}{|parent|} \log_2 \frac{|subset|}{|parent|} \right)$$

Ако атрибутът разделя данните на много малки групи → Intrinsic Info ще е висока. Ако ги разделя на малко групи → Intrinsic Info ще е по-ниска. Използва се в Gain Ratio, за да не се избират “измамни” атрибути, които просто създават много групи без да носят полезна информация.**GR коригира IG**, като отчита **размера и броя на разклоненията**.**C4.5 алгоритъм** използва този подход за по-добър избор на атрибути.

**Covering Algorithms (Правила срещу Дървета за Решения)**

**Разлика между Rules и Decision Trees: Дървета:** Пресъздават **цялата структура на решението**. **Rules:** По-компактни и по-лесни за интерпретация.

**Simple Covering Algorithm** (Алгоритъм за Генериране на Правила)

**Генерираме правило**, като добавяме тестове, които **максимизират точността**. **Подобно на Decision Trees**, но **се фокусираме върху един клас наведнъж**. **Ново правило се добавя, когато старите не обхващат всички примери**. **Краен резултат:** Компактен набор от **логически правила**.

**Извличане на асоциативни правила-** Наивен метод за намиране на асоциативни правила: Използва separate-and-conquer метод; Третира всяка възможна комбинация от attribute values като отделен клас.Проблеми: Изчислителна сложност – прекалено много възможни комбинации. Голям брой правила, които трябва да бъдат pruned (орязани) на базата на support и confidence. Решение: Можем директно да търсим association rules с висока support и accuracy.

**Item Sets**: Основа за намиране на правила; **Support** – броят на инстанциите, които правилото покрива правилно; **Item** – една тестова двойка attribute-value; **Item Set** – всички items, които се срещат в едно правило.

-Цел: Намиране само на тези rules, които надвишават pre-defined support. Метод: Намерете всички item sets, които отговарят на minimum support.  Генерирайте rules от тези item sets.

Генериране на Item Sets ефективно

**-One-item sets** са лесни за намиране. **(k-item set)** може да се формира чрез комбиниране на **(k-1)-item sets**. Ако **(A B)** е чест **item**

**set**, тогава **(A)** и **(B)** също трябва да са чест **item sets**. Използват се **hash tables** за бързо търсене на **(k-1)-item sets**.

**Пример:** Имаме честите **three-item sets**: (A B C), (A B D), (A C D), (A C E), (B C D)

**Кандидат four-item sets:**
✅(A B C D) → ОК, защото има (A C D) и (B C D).
❌ (A C D E) → Не е ОК, защото (C D E) не е чест.

**Алгоритъм за намиране на Item Sets**: Генерирайте (k+1)-item sets от честите k-item sets.Филтрирайте тези, които не покриват minimum support.Повторете процеса докато няма повече нови item sets.

**Да открием силни правила с висока достоверност (confidence).** Вместо да пробваме всички възможни комбинации (brute-force) – които са много (2^N-1), използваме по-умен подход: Постепенно надграждане: Създаваме (c+1)-консеквентни правила от вече проверени c-консеквентни. Използваме **hash таблица** за бързо намиране на колко често се среща лявата част на правилото (**support**). Строим по-сложни правила само от такива, които вече са показали добра достоверност.

**Ако по-сложното правило е вярно, то и по-простите части също са вярни.**

**Association Rules**: Алгоритъмът преминава през **данните по веднъж за всеки item set размер**. **Алтернативен подход:** Генериране на **(k+2)-item sets** веднага след **(k+1)-item sets**. Увеличава броя на кандидат **item sets**, но изисква **по-малко преминавания през данните**. **Практически въпроси: Support level** трябва да се настройва динамично за да генерира **достатъчно** на брой **rules**. по-добре е да се използва **sparse format**.

**Линейни модели: Linear Regression** - Най-естествено работи с **numeric attributes**. Изходът е **линейна комбинация** от атрибутите. Цел: Намаляване на squared error. Метод: Използване на matrix operations за изчисляване на weights.

Класификация с Linear Regression Проблем: Резултатите (output) не са винаги между 0 и 1, което е нужно при класификация (напр. вероятности). Често се правят клипиране и нормализация, за да се вкарат резултатите в интервала [0,1], но това не е най-доброто решение. Решение: Logistic Regression- Този метод използва logit трансформация, която преобразува стойностите от интервала [0,1] в стойности от -∞ до +∞. Това позволява на модела да предсказва вероятности правилно и да решава проблема с ограничения интервал. Пример:Linear Regression може да предскаже резултат 1.2 или -0.3, което не е валидна вероятност. Logistic Regression преобразува тези стойности, така че изходът винаги да е между 0 и 1, напр. 0.8 (80% вероятност).

**Maximum Likelihood в Logistic Regression** - Цел: **Максимизиране на вероятността** на наблюдаваните **training data**. **Логаритъмът на вероятностите** се използва вместо **произведение на вероятностите**. Използва **iteratively re-weighted least squares** или други **optimization methods**.

**Многокласова Logistic Regression-**Известна като **multinomial logistic regression** или **polytomous logistic regression**. Вместо независими **logistic regression models** за всеки клас, **максимизира вероятността** за всички класове едновременно. **Алтернативен подход: Pairwise Classification:** Построяване на **класификатор за всяка двойка класове**; **Гласуване** за определяне на крайния клас.

**Перцептрон (Perceptron) -**Изучава **линейни хиперплоскости** за **разделяне на данните**. Ако **данните са линейно разделими**, **перцептронът гарантирано** намира **разделящата хиперплоскост**.

**Алгоритъм: Инициализиране на weights** на нула; Докато всички **инстанции не бъдат класифицирани правилно**. Ако **инстанцията е класифицирана грешно**: **Добавяне** на вектора към weights, ако е **първия клас**. **Изваждане** на вектора от weights, ако е **втория клас**.

**Winnow Algorithm**-Разлика с Perceptron: Използва мултипликативни обновявания вместо адитивни. Позволява работа с бинарни данни (0/1). Алгоритъм: Класифициране на всяка инстанция. Ако има грешка: Умножаване на weights с alpha, ако инстанцията е от първия клас. Деление на weights с alpha, ако инстанцията е от втория клас. Winnow е ефективен в откриване на релевантни features. Може да се използва в **онлайн-обучение** -вид машинно обучение, при което моделът се обучава последователно върху данни, които идват един по един или на малки партиди, вместо да използва цялата тренировъчна база наведнъж.

**Balanced Winnow:** Използва **две weight вектори** (един за всеки клас). **Увеличава един вектор и намалява другия** при грешна класификация.

**Обучение на базата на инстанции** - функцията за разстояние определя какво се научава. Метрики за разстояние: Euclidean distance (Евклидово разстояние):

$$d(a^{(1)}, a^{(2)}) = \sqrt{\sum_{i=1}^{k} (a_i^{(1)} - a_i^{(2)})^2}$$

**Забележка:** Няма нужда да се изчислява квадратният корен, когато се сравняват разстояния. Друга популярна метрика е **city-block metric** (манхатънско разстояние), която просто сумира разликите без да ги повдига на квадрат.

**Normalization (Нормализация)**: Различните атрибути могат да имат различни мащаби, което изисква нормализиране, например до диапазона [0,1]. Ако атрибутът е nominal (номинален), разстоянието е: 0, ако стойностите са еднакви. 1, ако стойностите са различни. Ако има missing values (липсващи стойности), те често се третират като максимално отдалечени.

**Ефективно намиране на най-близки съседи** -Най-простият начин да намерим nearest neighbor е линеен скан на всички инстанции. Проблем: Изчислителната сложност е пропорционална на произведението от броя на инстанциите в обучаващия и тестовия сет.

За по-ефективно търсене използваме структури от данни като: **kD-trees** (k-размерни дървета) и **ball trees** (балови):

**kD-trees:** Дърво, което разделя пространството на хиперправоъгълници. Разделянето се извършва по **посока с най-голяма вариация**. Разделителната стойност е **медианата** по избраната посока.

**Ball trees:** Вместо хиперправоъгълници, използват **хиперсфери**. Позволяват **по-добро адаптиране** към разпределението на данните.

**Clustering (Клъстеризация)** - нямаме предварително дефинирани класове – вместо това се опитваме да групираме естествено сходни инстанции. Клъстерите могат да бъдат: разделени vs. припокриващи се; детерминистични vs. вероятностни; плоски vs. йерархични

**k-means Algorithm (Алгоритъмът k-means):** Избиране на k случайни клъстер центъра. Присвояване на всяка инстанция към най-близкия клъстер център (Euclidean distance). Пресмятане на новите клъстер центрове като средна стойност (centroid). Ако центровете са се променили, повтаряме процеса. Проблем: Може да попаднем в локален минимум, зависейки от началните точки. Решение: Стартираме с различни начални семена и избираме най-добрия резултат.

**Hierarchical Clustering (Йерархична клъстеризация):** Bisecting k-means: Разделя данните отгоре-надолу; Agglomerative clustering: Започва с единични точки и ги слива отдолу-нагоре.

Методи за измерване на разстоянието между клъстери: **Single-linkage** (най-близки точки от всеки клъстер); **Complete-linkage** (най-отдалечени точки); **Average-linkage** (средно разстояние); **Centroid-linkage** (разстояние между центроидите); **Ward's method** (минимизира квадрата на разстоянията).

**Multi-instance Learning (Обучение с множество инстанции) -** примерите представляват групи от инстанции (bags), които споделят един и същ етикет. Два подхода:

**-Aggregating the input** (Обобщаване на входа): Преобразуваме групата (bag) в **единична инстанция**, като използваме **средна стойност, мода, минимум, максимум** и др. Обучаваме **стандартен класификатор** върху тези обобщени стойности.

**-Aggregating the output** (Обобщаване на изхода): Обучаваме класификатор върху **отделните инстанции**. Прогнозираме за всяка инстанция, а после **агрегираме резултатите** за целия bag.