4. Алгоритми – основни методи – Резюме

- **Inferring rudimentary rules** – Извличане на основни правила
- **Simple probabilistic modeling** – Прост вероятностен модел
- **Constructing decision trees** – Създаване на **decision trees**
- **Constructing rules** – Конструиране на правила
- **Association rule learning** – Извличане на **association rules**
- **Linear models** – **Линейни модели**
- **Instance-based learning** – Обучение, базирано на примери
- **Clustering** – **Клъстериране**
- **Multi-instance learning** – Обучение с множество примери

Принцип на простотата - 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**)
- Комплект от прости логически правила
- Взаимоотношения между групи **attributes**
- Претеглена линейна комбинация на **attributes**
- Силни връзки между съседни примери (на база разстояние)
- Групиране на данни (**clusters**)
- Агрегиране на **bags of instances**
- Успехът зависи от конкретната **domain** (област на приложение)

Извличане на основни правила – 1R алгоритъм

1R (**One Rule**) е алгоритъм, който изгражда **едно-степенно решаващо дърво**. Той създава правила, като избира един **attribute**, който минимизира грешката при класификация.

Стъпки за 1R алгоритъм:

1. За всеки **attribute**:
    - Създаване на клон за всяка стойност на **attribute**

    - Всеки клон получава най-често срещания клас сред примерите
2. Изчисляване на **error rate** – процент на грешни класификации
3. Избира се **attribute** с най-нисък **error rate**

Примерен псевдокод:

For each attribute: For each value of the attribute, create a rule:
- Преброй честотата на класовете
- Намери най-често срещания клас
- Назначи този клас на стойността на атрибута
- Изчисли грешката на всяко правило
- Избери правилото с най-малка грешка

Обработка на липсващи стойности: Липсващата стойност се третира като отделна attribute value.

Числови атрибути:

1. **Discretization** – разделяне на числата на групи или интервали
2. Определяне на **breakpoints** (граници) там, където **class** се променя. Границите между интервалите се избират там, където класовете се сменят. Например, ако в даден момент повечето хора под 18 години не карат кола, а тези над 18 карат, логично е да сложим граница на 18 години.
3. 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- когато моделът „учи“ твърде много детайли и случайности от тренировъчните данни, вместо само важните закономерности. Това го прави много добър за тези данни, но слаб при нови данни.

- 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 -** Например, ако при групиране на възрасти една грешно въведена стойност (например 200 години) създаде нова категория, моделът ще направи грешни прогнози.
- Решение: задаване на **минимален брой примери в даден клас** на интервал- Вместо да създаваме интервал за всяка уникална стойност, изискваме поне няколко примера (например 5) в един клас, за да е стабилен моделът.

Прост вероятностен модел – Naïve Bayes - използва всички attributes (за разлика от 1R, който използва само един). Две основни предположения:

1. **Всички attributes са равнозначни-** всеки атрибут се смята за еднакво важен.
2. **Attributes** са **статистически независими** (ако класът е известен)- стойността на един атрибут не влияе на другите. Това предположение обикновено не е вярно, но алгоритъмът често работи добре на практика.

Bayes' Rule – Байесова теорема - Основава се на математическа формула, която ни помага да изчислим вероятността даден обект да принадлежи към определен клас, като използваме информация от данните. Тя изчислява вероятността на събитие H (клас), като използва наблюдавано доказателство E (стойностите на атрибутите):

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

A priori probability (P(H)) – Вероятността на събитието преди да имаме доказателства (преди да видим данни).

А **posteriori probability (P(H|E))** – Вероятността на събитието, след като имаме доказателство (след като видим данни).

Naïve Bayes за класификация

В задачата за класификация, целта е да изчислим вероятността за клас H (напр.спам или не спам), даден instance (инстанция). При класификация на имейли като "спам" или "не спам", вероятността за даден имейл да бъде "спам" може да се изчисли като продукт от вероятностите на различни думи в текста.

E – стойностите на атрибутите на инстанцията.

H – класът на инстанцията (например спам или не спам).

Наивно предположение: В модела Naïve Bayes се прави предположението, че атрибутите са независими един от друг, което улеснява изчисленията. Тоест, 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 - Понякога може да се случи, че дадена стойност на атрибут не се среща с определен клас в данните. Това води до вероятност нула за тази комбинация, което е проблем. **Решение:** Използваме **Laplace estimator**, който добавя 1 към всички възможни комбинации между стойност на атрибут и клас и така не получаваме нула, а малка вероятност. Това помага да се избегне нула и прави моделът по-устойчив.

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 (например възраст, височина, температура и др.). Обикновено се приема, че тези атрибути следват нормално разпределение (или Гаусово разпределение). Това означава, че стойностите на атрибутите са разпределени около средната стойност, като повечето стойности са близки до средното аритметично, а малък брой стойности се отклоняват на двете страни. Нормалното разпределение се описва с средно аритметично (mean, μ) и стандартно отклонение (σ):

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

Стандартно отклонение(σ): измерва колко разпръснати са ст-тите около средното аритметично. Колкото по-голямо е стандартното отклонение, толкова по-широко са разпръснати ст-тите.

Нормалното разпределение се описва с **средно аритметично (mean, μ)** и **стандартно отклонение (σ)**.

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 е стойността на атрибута
μ\mu (mean) е средното аритметично
σ\sigma (standard deviation) е стандартното отклонение
e е основата на естествения логаритъм (приблизително 2.718)

π е математическата константа (приблизително 3.1416)

Средно аритметично (Mean, μ) - Средното аритметично, означавано като μ, е 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 е **средното аритметично**
- n е броят на елементите в набора от данни
- xi са стойностите в този набор

Стандартно отклонение (σ)- Стандартното отклонение, означавано като **σ (сигма)**, показва **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 е стандартното отклонение
- n е броят на стойностите
- xi са отделните стойности
- μ е средното аритметично

Обяснение на формулата:

xi−μ: Изчислява се **разликата** между всяка стойност и средното аритметично. Разликата се **повдига на квадрат. Сумират се всички стойности**. **Деление на броя на елементите n** – получава се средната стойност на отклоненията. **Извлича се квадратен корен**, за да се върне обратно в оригиналните мерни единици.

Probability Densities (Плътност на вероятностите) - В теорията на вероятностите, функциите на плътността на вероятността (PDFs) описват как е разпределена вероятността за стойностите на непрекъснати случайни величини. **Непрекъсната случайна величина** е такава, чийто възможни стойности не са ограничени до отделни точки, а **могат да приемат всяко число в даден интервал** (температурата или време на изчакване). **Плътността на вероятността** (PDF) показва колко вероятно е дадено число да бъде избрано от възможните стойности на случайната величина. Тя не дава конкретната вероятност за стойност, а по-скоро показва "концентрацията" на вероятността в дадена област. Например, ако имате PDF за температура, тя ще покаже каква е вероятността температурата да бъде в различни диапазони.

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

n1,n2,...,nk – брой пъти, в които дадена дума I се среща в документа

P1,P2,...,Pk – 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):

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

⚠ **Забележка:** 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 **обърка**.

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**.

1. **Избор на атрибут за корен**
2. **Създаване на клонове** за всяка стойност на атрибута
3. **Разделяне на примери в подмножества**
4. **Рекурсия:** Прилагане на същата стратегия върху всяко подмножество
5. **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)**

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

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

Изчисляване на Information Gain - Използва се Entropy (ентропия) като мярка за неопределеност**:**

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

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)**

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

Intrinsic Information:

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

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)

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

Mining Association Rules -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

1. **Генерирайте (k+1)-item sets** от честите **k-item sets**.
2. **Филтрирайте** тези, които не покриват **minimum support**.
3. **Повторете процеса** докато няма повече нови **item sets**.

Генериране на Rules ефективно

- Търсим **high-confidence rules**.
- **Support** на **antecedent** (лявата част на правилото) може да бъде получен от **item set hash table**.
- Вместо **brute-force** метод (2^N-1 за N-item set), използваме **по-ефективен алгоритъм**:
    - Построяване на **(c+1)-consequent rules** от **c-consequent ones**.
    - Ако едно **(c+1)-consequent rule** е вярно, значи всички **c-consequent rules** също трябва да са верни.

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**. ARFF форматът е **неефективен** за **market basket data** – 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.

Кладификация с Linear Regression - Може да се използва за **classification**, но има проблем: **Output values** не са в диапазона [0,1]. Често се **клипират** в този интервал и се **нормализират**.

📌 **Решение: Logistic Regression**

- Преобразува **target variable** чрез **logit transformation**.
- **Преобразува [0,1] в (-∞,+∞)**, което решава проблема с интервала.

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**.

📌 **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) и др. проблеми: 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:

-disjoint vs. overlapping (разделени vs. припокриващи се)

-deterministic vs. probabilistic (детерминистични vs. вероятностни)

-flat vs. hierarchical (плоски vs. йерархични)

k-means Algorithm (Алгоритъмът k-means):

1. **Избиране на k случайни клъстер центъра**.
2. **Присвояване на всяка инстанция** към най-близкия клъстер център (**Euclidean distance**).
3. **Пресмятане на новите клъстер центрове** като средна стойност (centroid).
4. **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.