1. **Ентропия**- мярка **за липсата на информация и несигурността в нея**, колкото е по-висока, толкова повече неща не знаем, измерва се в битове;
2. **Какво е 1 бит информация? - Информацията, ĸоято ни дава отговора на един въпрос с два равновероятни отговора.** Например, отговора на въпроса: „Каĸво ще се падне от хвърлянето на честна монета, ези или тура?“ ще ни донесе един бит информация
3. Ентропията НАМАЛЯВА, когато имаме два класа, единият от които е с по-голяма вероятност от другия. Това е така, понеже започваме да клоним към този с по-голямата вероятност- неяснотата намалява.
4. Може ли ентропията да е повече от 1?- Може при повече от два класа- 1 е при наличието на два класа и това е в най-лошия случай- когато имаме равновероятни класове
5. Как се измерва ентропията? Какво е 1 бит?- битовете са равновероятни класове, ако имаме два класа, бит е мерна единица за количество информация

Entropy(S)=-p+\*log2p+ - p-\*log2p-

1. **Задача за удовлетворяване на ограниченията—CSP**- основни техники- Backtracking, Local serach (minconflicts), Constraints Propagation (Arc Consistency)- Проблеми, които се дефинират по модела Дадено-Търси се, както следва
   * 1. **Дадено=State=състояние=** имаме множеството от **променливи** **X**, които променливи могат да приемат стойности само от **допустимо** **множество** **от** **стойности** **D**
     2. **Търси се=goal test=** да удовлетворим множеството от **ограничения** **С**, които ограничения се налагат над множеството от променливи Х и множеството от допустими стойности за тези променливи D
        1. **Ограниченията:** може да са:
           1. **Унарни-** над една (променлива
           2. **Бинарни-** над две променливи
           3. **От по-**висок ред- над 3 или повече променливи
           4. **Преференции-** т.нар. soft contraints- при тях можем да имаме 2 или повече валидни стойности , но една от тях е по-предпочитана
   1. Предимства на тези задачи са: Лесни са за представяне
   2. Пример: Map Coloring (картата на Австралия): да бъде оцветен всеки щат в такъв цвят, че да няма два съседни щата в един и същ цвят, като при тази задача:
      1. **Щатите- ПРОМЕНЛИВИТЕ- стойностите на мн-вото Х**
      2. **Цветовете- ДОПУСТИМИТЕ СТОЙНОСТИ- стойностите на мн-вото D**
      3. **2 съседни щата не са в 1 цвят- ОГРАНИЧЕНИЯТА- стойностите на мн-вото C**
   3. Най-използвани техники:
      1. **Backtracking (зад него стои DFS)- той решава задачата с оцветяването**

Пълно обхождане на възможностите. При него, ако стигнем до някоя променлива, на която не може да назначим стойност може да спрем да развиваме рекурсията и да се върнем назад и да продължим да я развиваме в друга посока.

* + 1. **Constraint propagation (Arc Consistency)**

При него елиминираме голямо количество от излишни пътища при търсене на удволетворение на всички ограничения. Това става като се опитаме да отговорим правилно на въпросите:

– От коя променлива да започнем?

– Коя да е следващата променлива, която да инициализираме?

– В какъв ред да се опитваме да инициализираме променливите?

– Може ли да засечем терминално НЕцелево състояние по-рано?

– Може ли да се възползваме по някакъв начин от структурата на проблема?

Идеята на Arc consistency е да се опитаме да орежем домейните колкото се може повече, преди да започнем да търсим решение. Променливата е съвместима с домейна (Domain consistency), ако нито една стойност на нейният домейн не е невъзможна от някакви унарни ограничения.

Дъга (половин ребро) от променлива към ограничение е консистентна (Arc consistency), ако за всяка стойност на х от домейна Х има някоя стойност y от домейна Y, за която r(x,y) e удовлетвотено, където r(x,y) e бинарното ограничение от дъгата.

Граф (мрежа) е arc consistent, ако всичките и дъги са arc consistent

* + 1. **Local Search (Min Conflicts)- Hill Climbing**- хващаме конфликтите и се опитваме да ги минимизираме
    2. Forward Checking

1. kMeans **Как работи KMeans**

**Идеята е-** множество примери описани с **n** атрибута да ги групираме в **k** клъстера според техните сходства. **k** може да е дадено или да трябва да го намерим автоматично с помощта на elbow метод. Иницализират се k на брой произволни точки в пространството- центроиди, за които може да се ползва техниката- да се вземат точки от примерите, но може и да са съвсем произволни. Всеки пример се причислява към най-близкия центроид, когато всички примери са причислени се образува т.нар. **клъстер.** В самия клъстер обаче центроидът не попада съвсем в центъра и е нужно отново да се преизчисли новия център. Ъпдейтваме центроида и така е възможно някои примери да сменят своя центроид- тоест да си сменят клъстера. Продължаваме, докато центроидите спрат да се обновяват или примерите спрат да си сменят клъстера. Метриката, която се използва,е евклидово разстояние.

Този алгоритъм е **локална техника** , зависеща от първоначалните центроиди.

Алгоритъмът може да стигне до локален екстремум и като стане това правим рандъм рестарт като пазим най-доброто решение- можем да го оценим с помощта на вътрешно клъстерно разстояние и междуклъстерно разстояние.

Алгоритъмът може да бъде подобрен с използването на по-добра структура- например k-d дърво.

Предизвикателства- каква е формата на клъстерите и колко са на брой

Видове к мийнс:

* софт к мийнс
* йерархично-
  + **агломеративен**-bottom-up-всеки един пример който идва е отделен клъстер и постепенно ги обединяваме- всеки клъстер в надклъстер
  + **Divisive** клъстеризация- топ-даун- раздробяваме ти поетапно- всеки клъстер правим на подклъстери

При хард инициализация един пример е от точно един клъстер

При софт инициализация един пример може да е от повече клъстери

**Йерархично клъстеризиране:**

Изходът е дендограма- клъстери от клъстери- йерархия от клъстери

1. Локално търсещи алгоритми-
   1. Hill climbing
   2. Local beam search- Локално търсещ алгоритъм за информирано търсене. Локален вариант на Грийди. Използва приоритетна опашка с лимит l- пази l най-добри върха (най-добри се определя по евристиката). Не е пълен и не е оптимален.
   3. симулирано втвърдяване
   4. Генетичните алгоритми- локално търсене в лъч, информирано търсене. Имаме популация – мн-во от индивиди, която се сортира в низходящ ред според fitness функциятаи настъпва процес на selection- индивиди с по-добри гени се избират за репродуциране-върху тях се прилага crossover. Накрая част от децата се mutate-ват- изменение на гените с цел създаване на по-добри гени. Това изменение се повтаря епоха след епоха.АЛгоритъмът спира когато няколко поредни поколения най-добрият индивид не си изменя fitness функцията.
2. Hill Climbing Да се опише Hill Climbing

**Локално** търсещ алгоритъм за информирано търсене. Локален вариант на Greedy- Greedy BFS. Този алгоритъм използва приоритетна опашка с лимит 1- пази само най-добрия връх. Не е пълен и НЕ е оптимален.

1. Симулирано втвърдяване- подобно на Hill Climbing. Пази 1 състояние, но понякога може да позволи това състояние да не е най-доброто. Използва се рандъм рестарт- като локално търсещ алгоритъм е възможно да достигне до локален екстремум, който обаче да не е най-добрият, има параметър температура
2. Учене с учител+Учене без учител

Два основни типа учене- и в двата случая учим въз основа мн-во от примери които са описани с мн-во харакеристики- набор от данни – таблица редовете са примери колони са характеристиките

* С учител- допълнителна колона клас която се попълва от т нар учител- **казва какъв е типът на самия пример**- въз основа този тип- **категорен или числов** имаме **класификация или регресия**- ако имаме да не може би е класификация- снимки на животни, ако са числа- регресия- апартаменти
* Без учител- няма кой да ни даде таи инфо – имаме просто мн-вото от примери и техните х-ки- тук задачата която се решава е малко по – различна- един основен тип е **клъстеризация**-идеята е да намерим зависимости между дадените ни примери и да ги групираме в групи, които наричаме клъстери, така че тези , който са подобни, да попадат в един клъстер- пр. НОВИНИ- атрибутите са речник на използвани думи- новините използващи горе-олу едни и същи думи попадат в един клъстер- напр ПОЛИТИКА в един клъстер, СПОРТ в друг и тн- нещо подобно на класовете ама не ги знаем и сами трябва да ги разделим на тези категории

1. **Ансамблово учене-** техника за обучаване на алгоритми- подход за обучаване на множество слаби **класификатори** с цел получаване на един по-силен класификатор
2. Local Beam Search

**Локално** търсещ алгоритъм за информирано търсене. Локален вариант на Greedy. Този алгоритъм използва приоритетна опашка с лимит 1- пази само най-добрия връх. Не е пълен и НЕ е оптимален.

1. Генетични алгоритми
2. Локално търсещите алгоритми решават проблема с оптимизацията- Справят се с проблема на експоненциалното време при глобално търсещите алгоритми. Прилагат се върху оптимизационни задачи. Когато си в някакво начално състояние в даден сет от данни и искаш да стигнеш до някакво целево, финално- имаш оценъчна функция, която те насочва накъде да вървиш. Може да не се намери решение в разглежданото подмножество.
3. Дървета на решенията

**Дърво на решенията:**

Учене с учител за класификация и регресия, основано на модел, нетърпеливо и глобално учещо. То е от върхове и ребра, всички върхове които не са листа са атрибути, всички ребра изилзащи от даден връх, отговарят на стойностите които атрибутът приема, всички върхове които са листа биват възможните класове

Основен алгоритъм- ID3 Top-Down induction od Dec Tree

Как работи- взимаме най-добрия атрибут въз основа информаци ягейн филтирарме данните по негови стойности въз основа на това взимаме следващ най-собър атрибут-

**Ентропия**- мярка за липсата на информация и несугронстта в нея като формулята е сума за всеки един от въможните изходи които имаме с отрицателен знак вероятността за него по лог от вероятностат той да се случи

**Инфо Гейн-** колко е значим даден атрибут за определяне на даден клас, колко инфо ни носи той. Това нещо се измерва като намерим ентропията на класа и от нея извадим ентропията на класа и атрибута

**Основен проблем е OVERFITTING**

**Underfitting-** алгоритъмът не е способен да се обучи дори на тренировъчните данни, не може да калсифицира добре

**Overfitting-**прекомерно нагаждане – алг е назубрил тренировъчните примери, той се предсатвя мн добре на тренировъчниге, на терйнинга томността му е мн добра, на тест множеството точността е мн ниска- не е в състояние да прецени към кой клас спада примерът

**Техники за спарвяне с overfitting-** взимане на повече трен данни, за да се олучи по-добре алг, извадката от тренировъчни данни да е по-добра, намаляване броя на х-ките, като са мн х-ки аг почва да преспецифицира- feauture-selection, окастране- намаляме блоря на атрибутите които ще ползваме,

Дърво на решенията може да се представи като правила на решенията- decision tree => decision rules, които са if-then правила- дизюнкции от конюнкции

1. Защо се прави кастрене на дървото на решенията?- за да се избегне overfitting

**Кастрене- препрунинг и постпрунинг**

**ПРЕПРУНИНГ-** Строим дървото и в процес на строене казваче че повече няма да добавяме върхове- демек атрибути, за да избегнем овърфиттинг

**Постпрунинг**- построяваме цялото дърво и почваме да режем от дървото докато точността се подобрява, окастряме го спряво това да получим най-оптимален модел

1. Невронни мрежи- как работят какво е backpropagation

Back-prop – да разпространим грешката назад в предходните слоеве при многослойна- само за многослойни невронни мрежи се ползва, ако са дълбоки невронните мрежи обаче грешката запова да избледнява и бекпроп не е мн подходящо и почва да се използва енкодинг-декодинг архитектура

1. Съждителни и предикатни логики- каква е разликата

**Съждителни(and, or, not, implies, equivalent)**- най-прости логики които могат за бъдат използвани за представяне на знания. Имаме сждения и връзки- всяко едно съждение си има вярностна стойност връзките са за да рправим изрази , съставена от няколко съждения- те образуват едно голямо изречение което има стйност трю ли фолс

**Предикатни-** те имат по-голяма изразителна сила отколкото съждителните, Синтаксисът позволява освен да имаме някакви съждения, да имаме вече константи ,предикати променливи функции, куанитфкиатрои\*- позвобява ин да кажем че нещо е вярно за вс обекти- на мястото на тия променливи можем да избираме от различни стойности

Предикатната е по-силна защото с едно изречение можем да разберем за верността на няколко твърдения- заради променливите, а при съжденията само за едно можем

1. Обучение на невронни мрежи- има обучаващо множество и изчисляваме грешката спрямо полученото от мрежата и се опитваме да я минимизираме- backpropagation е за мрежите

първо определяш топологията (броя на слоевете и броя на ел. във всеки слой)

после се задават случайни стойности на търсените тегла на връзките между ел, след което те итеративно се променят (уточняват) с помощта на Обучаващо правило (обучаващ алгоритъм)

1. Невронните мрежи локално търсещи алгоритми ли използват?- ДА , Gradient descend
2. **Алгоритми Учене с учител:**

**Наивен Бейсов Класификатор:**

Учене с учител- класификация, основано на модел,нетърпеливо, глобално учещ

Строи вероятностен модел чиято идея е когато дойде нов тестови пример с някакви х-ки той да намери каква е вероятността за всеки един от възможните класове при условие този тестови пример, тоест ако класовете са да и не да намерим вероятността да имаме клас да при условие този пример , вер не при условие този пример и така за вс един клас и така класът с най-голяма веротяност да се случи ози пример- то това е класът

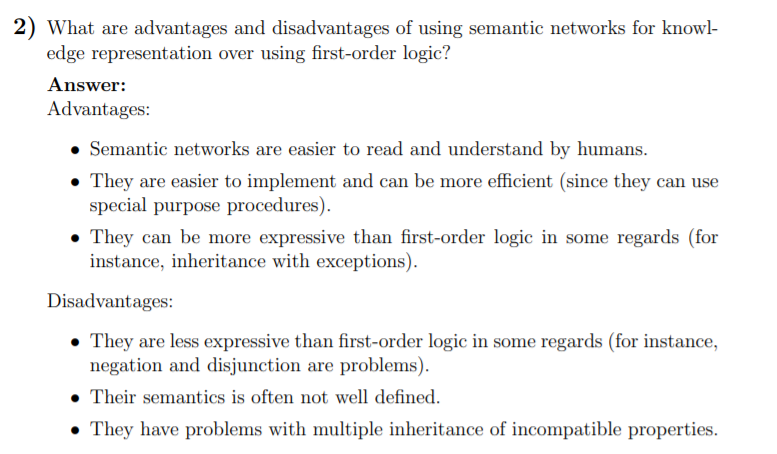
Нулева вероятност- липсваща стойност в таблицата- оправя се с Лаплас Смуутинг, или имаме малки числа

**kNN**- локално учещ, примерите се свеждат до н-мерното пространсто, близостта се измерва с метрика в пространството- евклидово разстояние например, целевата функция може да е както дискретна- най-често срещания клас, така и реална- взимаме осреднена стойност- хиперпараметъри- самото к, може да се добави и отегляване така че по-отдалечените примери да влизат с по-малка тежест,

Всички алг основани на модел са нетърпеливи!! А не мързеливи

1. **как се построява decision tree -** стремим се да сложим атрибутите които ни носят най голяма информация най-отгоре, с индукционен подход- то го има и в името му ID3 = induction of dec tree; имаш корен, който се избира от това кой атрибут ти носи най-много информация, и останалите върхове, които не са листа са ти останалите атрибути, ребрата са стойностите на тези атрибути и минаваш по тях, докато стигнеш до листо, което е някакъв клас
2. Как избираме най-добрия атрибут за дърво на решенията- на база информ гейна- колкото по-голям
3. Предимства и недостатъци на семантичните мрежи

**Семантични мрежи-**Представяне на знания, които сами по себе си представляват връзки между обекти под формата на мрежа. Най-общо се представя като граф, където обектите са възлите/върховете, а пък релациите/връзките са ребрата.



**Учене с учител:**

kNN

ID3- Decision Tree

Наивен Бейсов

**УЧене без учител:**

kMeans

**Учене по примери: мързеливите алгоритми, учене по аналогия- kNN**

**Учене по модел: Наивен Бейсов, Дърво на решения**

**Глобално ТЪРСЕНЕ vs Локално ТЪРСЕНЕ**

Глобално- търси в цялото пространство от състояия, ако се наложи, може да мине през всяко едно състояние.

Локално- търси в някакво подпространство от състоянията, изрязва някаква част от него и гледа в тази подобласт.

**Глобално УЧЕНЕ vs Локално УЧЕНЕ:**

Глобално- Прави извод върху всички примери от набора от данни.

Локално- Прави извод върху част от примерите от набора от данни.

**Машинно самообучение-** машините да се научат да решават проблеми както ние можем да решаваме проблеми- да не им е казано изрично ако имаш това правиш еди си какво- да не е иф елсе а те сами въз основа примери да си направят изводите да си научат как да **постъпват учене по индукция- въз основа мн-во наблюдения да се изгради хипотеза която да се прилага върху нови проблеми които настъпват от същия тип- описани със същите характеристики**

Данните които иднват е възможно да са с шум да са нечисти затова е необходим Data Preprocessing:

* Data Cleaning-запълване на липсващи стойности, откриване на outliers, изчистване на неконсистентности
* Data Transformation-нормализиране на данни, да са в една скала, за да може х-ките да са с една тежест, да се агрегират, да се генерализират в някаква йерархия, да се разделят атрибути- напр поле Дата може да се раздели на ден от седмица/част от деня и тн, да се обединят атрибути
* Data Reduction- редуциране по брой атрибути, брой стойности, по брой примери- напр ако са мн примери, ако омжем да обучим класификатор с по-малко го правим

Линейна и логистична регресия

Линейна- идва от статистиката, използва се за регресионни задачи- идеята е да върне като резултат число

Логистична- класифициращ алгоритъм- прилага се там където класът е категория, стандартно рабонти за два класа, може да се промени и за повече класове-принцип ЕДИН СРЕЩУ ВСИЧКИ: например ако има да не може би може да е ДА не е ДА, НЕ не е НЕ, МОЖЕ БИ не е МОЖЕ БИ

Линейни класиф- решават линейно разделими проблеми, строят линейни ф-ии, предсатвяме ги в н-мерно пр-во където н е брой атрибути , а напр с цветове се показват класове- така като дойде нов пример спрямо линията може да решим къде да попадне

Наивен Бейсов класификатор

Персептрон- еднослойна невронна мрежа и тя може да решава само линейно-разделими проблеми

kNN- К на брой най-близки съседи

Боостинг- итеративно алг се подобрява, в началото се строи крива познава но много бърка, алг се обучава наново и апочва да познава повече примери- итеративно подобрение

Дървета на решения-

Невронни мрежи- еднослойни имат само входен и изходен слой и решават само линейно-разделими проблеми, а многослойните имат един или повече скрити слоеве

**Обучение на алгоритмите от тип Учене с учител:**

Данните са разделят на тренировъчни, валидационно и тестово множество- тренировъчното служи за обучение на алг, валидационното служи да се тества колко добре се е научил алг, там където бърка да се донастрои и да не бърка, тестовият сет служи да се тества алг как се представя- върху данни които не са нито в обучене били нито във валидиране

Метриката доколко добре се справя е Accuracy- какъв процент от примерите са познати

Крос-валидация техника когато данните са малко- к-фолд валидация- разбиваме на 10 части вътим 10 пъти и всеки път 1 тества 9 тренираме, точност на всяка врътка, накрая осредняваме и това ни носи колко добре би се справил алг като дойдат примери отвън

**Техники за учене без учител: мн-во от примери на които не знаем класовете**

Клъстеризация-сегментация- да видим какви клъстери се формират- продълговати, обли, дали са по-разредени, тестваме до каква степен алг праща в правилните клъстери примерите

Йерархична клъстеризация

Агломеративна клъстеризация- боттъм ъп- всеки един пример който идва е отделен кръстер и постепенно ги обединяваме- всеки клъстер в надклъстер

Divisive клъстеризация- топ-даун- раздробяваме ти поетапно- всеки клъстер правим на подклъстери

K Means клъстеризация-подава му се К и се инициализират к на брой произв точки в пр-вото- центроиди

**Глобално вс Локално УЧЕНЕ (търсене- как търсим дали глобално глледаме вс примери или локално – в някакво подмн-во)- при учене ДАЛИ СЕ ползват всички примери**

**Локално търсещи- невронни мрежи и к-меанс**

**невронни мрежи строят локална хипотеза стигат до локален екстермум въз основа произв тегла, но учат от всички примери- ЛОКАЛНО ТЪРСЕЩИ НО ГЛОБАЛНО УЧЕЩИ**

**Върху учене с учител**

**Глобално УЧЕНЕ-** извод върху всички примери от набора от данни- не са всички в пространстовто-невронни мрежи, наивен бейсов

**Локално УЧЕНЕ-** ползват се част от тях – кнн

**Основано на примери и основано на модел:**

Основано нап римери- кнн- прави си извода въз основа тези к примера- ползва примерите в суров вид

Основано на модел- наивен бейсов строи вероятностен модел- наборът от данни се траснформира до някакъв модел, дърво на решения също

**Мързеливо срещу нетърпеливо-**

**Мързеливо- не правиш нищо по време на ученетео- бавни по време на заявка но не се отделя време за изграждане на модел- това са яалг Учене основано на примери учат директно от новите примери**

**НЕтърпеливо- бързи по време на заявка защото отделят време за изграждане на модел**

**МЪРЗЕЛИВО:** кнн, локално измерима регресия, кейс-бейсд рийзанинг

**Ансамблово учене-** техника за обучаване на алгоритми- подход за обучаване на множество слаби **класификатори** с цел получаване на един по-силен класификатор.

Bagging- рандъм форест

Boosting-нещо си буст

Stacking

Okham’s Razor- при две или повече налични равноправни хипотези – гласят едон и също- на две различни нива на дървото- по-плитко и по-дълбоко, използваме по-простата- демек взимаме по-плитката,

**KMeans- учене без учител**

**Идеята е-** множество примери описани с н атрибута да ги групираме в к клъстера. Прилага се често върху учене с учител където клъсетризираме колоните, к може да е дадено или да трябва да го намерим автоматично

Имаме множество примери, не им знаем класовете, трябва да ги групираме по техните сходства.

ПРедизвикателства- каква е формата на клъстерите и колко са на брой

Видове: к мийнс софт к мийнс , йерархично- агломеративно и разделително

K Means клъстеризация-подава му се К и се инициализират к на брой произв точки в пр-вото- центроиди- като понякога те са част от самите примери- точка от примерите са – центроидите се ъпдейтват последвоталено- всеки пример се причислява към най-близкия центроид, това като се случи се образуват клъсетрите, като в самия клъстер центроидът НЕ попада точно в центъра и е нужно преизчислеине на новия цевтър- ъпдейтваме центроида и отново се прилага същата техника- причисляваме примерите към най-близския центроид- възможно е някои примери да сменят своя центроид. Продължава докато центроиди да се обновяват или примерите спрат да сменят своя клъстер. Разстоянието между пример и центроид измерваме с евклидово разстояние. ЪПдейтватето става като вземе средната точкамежду всички точки в клъстера

Колко добре се справя- локална техника зависеща от първоначално избраните произв центроиди- оценяваме спрямо вътрешно клъстерно разстояние и междуклъстерно разстояните

АЛгоритъмът стига до локален екстермум- тогава каот стигне там правим рандъм рестарт и пазим най-доброто решение- оценяваме го въз основа на вътрешно и външно клъстерно разстояние

Много добър алгоритъм, простичък, който е локално търсещ трябва да ползва рандъм рестарт, търсим най-добро разстояние въз основа вътрешно клъстерно разстояние, трябва да осигурим к, за определяне на брой клъстери- ELBOW метод – Можем да подобрим алгоритъма с по-добра структура- в случая k-d дърво,

При хард инициализация един пример е от точно един клъстер

При софт инициализация един пример може да е от повече клъстери

**Йерархично клъстеризиране:**

Изходът е дендограма- клъстери от клъстери- йерархия от клъстери

**Невронни мрежи:**

Симулират мозъка- имам изкуствен неврон, аксорс от който предава и дендрити от които приема инфо, aj предава от своя аксорс към дендритите на ai и ai предавва на останалите нататък.

Основни архитектури- силно-свързана мрежа – едносолйа и многослойна, само входен и изходен слой, входен един или повече скрити и един изхозен слой

Рекурентни невронни мрежи.- може да имат връзки и в слоевете