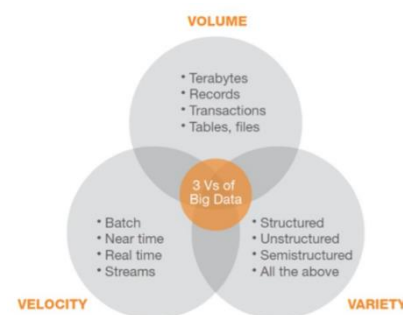


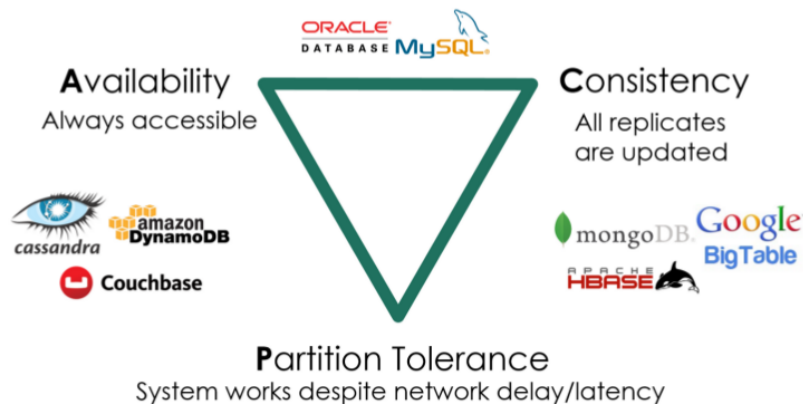
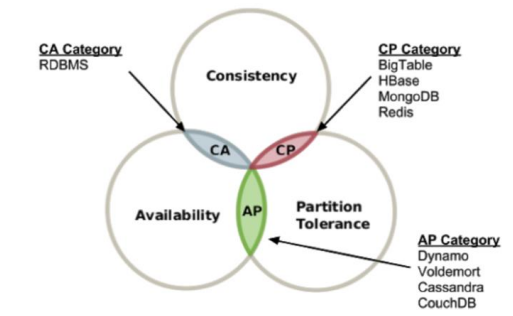
# CAP teorém a jeho vztah k NoSQL databázím

BI-BIG

- **Big Data** – manipulace s daty tak velkými, že je nemožné/obtížné s nimi pracovat za pomoci tradičních nástrojů a db (převážně relačních)
- **3V big dat:**
  - o **Volume** (obsah):
    - Data nejsou pouze text (hudba, obrázky, video, ...)
    - Exponenciální nárůst objemu
    - Data se vyhodnocují z různých úhlů
    - Terabyty, petabyty
  - o **Velocity** (rychlost):
    - Rychlost, kterou data přibývají
      - Čím větší rychlost, tím menší aktuálnost informace
      - Pohyby kurzů na burze
    - Rychlost zpracování dat
      - U výstupu pro uživatele v reálném čase
      - U periodického preprocessingu klidně déle
  - o **Variety** (různorodost):
    - data nemají homogenní strukturu – adaptabilita na různé podoby dat (video, dokumenty)
- **Distribuovaný systém**
  - o systém, jehož komponenty jsou propojeny sítí
  - o hw je levný, co se může pokazit, se nejspíše pokazí
  - o případný výsledek je brán jako očekávaný stav
  - o **Systémy s distribuovaným výpočetním výkonem:**
    - výpočet úlohy se rozdělí mezi více uzlů = paralelní zpracování – minimalizace času
    - problém s udržení škálovatelnosti (alespoň lineární)
    - pro vědecké výpočty a překládání zdrojů, moderní architektura serverových aplikací
      - proof of work kryptoměny – Bitcoin, Ethereum
  - o **Systémy s distribuovaným úložištěm:**
    - data jsou uschována na více uzlech, to zajišťuje vyšší redundanci, propustnost a dostupnost
    - disková kapacita více uzlů tvoří jeden diskový prostor (vyšší kapacita)
    - problém je zajištění konzistence dat
    - BitTorrent, proof of storage kryptoměny (Chia)
    - Datové a výpočetní operace mohou probíhat současně
- **NoSQL databáze = Not Only SQL**
  - o Nemá předem definované tabulky
  - o Dochází k **duplicity informací** (disky jsou levné, nevádí to)
  - o Tím je lepší možnost distribuce dat, zápis a čtení (lze provádět paralelně)
  - o Umožňují vkládat dokumenty za běhu bez předem definovaného formátu
  - o Horizontálně škálovatelné db (při větším vytížení stačí přidat další server a není třeba měnit kód aplikace)
  - o Automaticky rozmisťují data mezi jednotlivé servery
  - o **Nevýhody:** není deklarativní dotazovací jazyk a menší garance
  - o **Výhody:** flexibilní schéma, masivní škálovatelnost, vyšší výkon a dostupnost
- **CAP teorém = 3 základní vlastnosti distribuovaných systémů, přičemž každý může mít nejvíce 2**
  - o Konzistence, dostupnost a tolerance výpadku
  - o Hodnoty nejsou striktně binární, mohou se měnit v čase
  - o Slouží stále dobře návrhářům k ujasnění priorit



- **C = Konzistence (consistency)**
  - Pro každý požadavek je vrácen správný výsledek
  - V případě distribuovaného úložiště vrací každý uzel aktuální a stejná data
  - V případě výpočtu se výsledky od jednotlivých uzlů neliší
- **A = Dostupnost (Availability)**
  - Na každý požadavek přijde odpověď
  - V kontextu teorému čas nerozhoduje, v kontextu praxe systém musí být dostupný
- **P = Tolerance výpadku (Partition tolerance)**
  - Určuje chování podpůrného systému, na kterém služba běží
  - Určuje, jestli je systém schopen fungovat i v případě výpadku části paměti
  - výpadek dílčí části systému je brán dnes už jako předpokládaný stav (od 90. let na ústupu)
- **C + A (relační db)**
  - V případě výpadku části sítě je to problém
  - Než aby vrátil nekonzistentní výsledky, neodpoví radši vůbec
- **C + P (MongoDB)**
  - Pokud systém nevrátí aktuální data, vrátí chybu
- **A + P (Cassandra)**
  - Neexistuje záruka, že data jsou aktuální
  - tolerantní vůči výpadku části služby



#### - Segmentovaná konzistence a dostupnost

- Existují systémy, které nemají požadavky na všechny typy dotazů stejné
- Systém rozdělen na komponenty, každá bude splňovat vždy danou část CAP
- Celý systém nezaručuje ani konzistenci, ani dostupnost (to poskytují pouze jeho části)
- může probíhat na několika úrovních
- *rozdělení podle dat*: jiná data mohou vyžadovat jinou úroveň dostupnosti, file systémy
- *rozdělení podle operací*: operace pro zápis mohou mít jiné požadavky na konzistenci a dostupnost než operace pro čtení, db systémy

# MapReduce model: principy a jeho využití pro dotazování Big Data

BI-BIG

## - MapReduce model

- o Programovací model pro distribuované paralelní výpočty
- o původně od Google, Apache MapReduce = výpočetní framework nad Hadoopem
- o Master-slave architektura (jedno zařízení/proces přebírá jednosměrné řízení nad 1 a více procesy/zařízení)
- o Designovaný pro zpracování velkých dat
- o *základní myšlenka*: výpočet k datům, ne data k výpočtu (přesouvání dat je drahé na zdroje a náchylné k chybám sítě)
- o má 2 fáze: **mapování** a **redukce**
- o zpracování probíhá v místě uložení, je lepší škálovatelnost
- o distribuce dat na navzájem nezávislé části  $\Rightarrow$  lze dosáhnout snadného zpracování v uzlech
- o *principy*: paralelní zpracování, škálovatelnost, výpočet k datům, tolerantní vůči chybě

## - Mapovací funkce

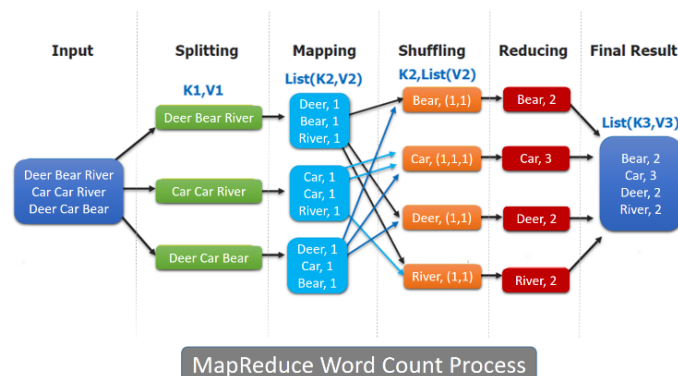
- o vstupem mapovací funkce je seznam prvků, na které je následně aplikována transformace
- o je bezstavová, převádí vstup na výstup typu  $[klíč, hodnota]$
- o např. převod stringů na UpperCase nebo počet stejných slov ve větě

## - Redukční funkce

- o agreguje seznam hodnot, který jí je poskytnut, do menšího množství
- o např. jednoduchý součet

## - Postup

- o nejprve je vstup rozdělen na části (resp. referuje se na určitou část dat, počet částí = počet mapperů), na těch proběhne mapování, poté shuffle & sort a nakonec se data zredukují
- o *mapování*: vstup je seznam hodnot, výstup je pár  $[klíč, hodnota]$  (v příkladě níže je to [jméno, počet výskytů])
- o *shuffle & sort*: obě fáze probíhají paralelně, třídění a posílání dat po síti (z map do reduce), kritické místo, data jsou seřazena podle komparátoru klíče, do redukční funkce se posílají páry se stejným klíčem,
- o *redukce*: aplikace redukční funkce, výstup není seřazený, může se provést 2x (nejprve před přenosem dat přes síť, aby se zmenšil objem dat)

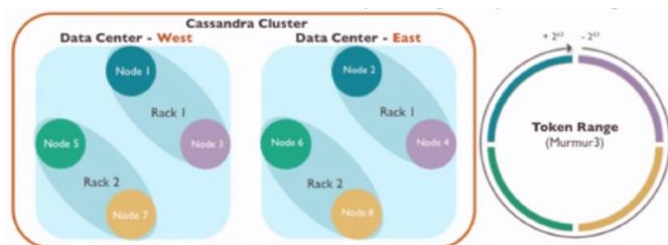


- **MapReduce InputFormat** – definuje strategii jak data přečíst a jak je rozdělit do mapperů
- **Partitioner** – vstup je výstupní klíč – hodnota z map tasku – rozhoduje o tom, do kterého reduceru půjdou která data
- **MapReduce Combiner** – stejný interface jako reducer, provádí operaci reduceru ještě předtím, než pošleme data přes síť – snížení velikosti dat k přesunu, vhodné pro komutativní funkce
- **Využití** – analytické úlohy, klasifikace, data mining, indexace a vyhledávání
  - například Google toto využívá pro generování Google indexu a Apache Hadoop pro spojení HDFS

# Typické databázové stroje pro Big Data (Cassandra nebo HBase) - architektura, databázový model, distribuce dat

BI-BIG

- **Apache Cassandra** – NoSQL db, AP systém v CAP, používá CQL
  - o open source distribuovaný databázový systém
  - o ukládá data mezi mnoha servery rozmístěnými v mnoha datacentrech
  - o **Data model** – Wide-Column Store
    - Modelování – vhodné navrhnout strukturu pro nejčastěji používané dotazy
      - de-normalizace a duplikace dat – zvyšuje rychlost čtení
  - o Vysoká dostupnost
  - o Nemá single point of failure
  - o **CAP**
    - Podporuje Consistency a Availability
    - Toleruje Network partitioning
  - o vyniká v rychlém zápisu (zapisuje se do logu a do memtable, která se periodicky flushuje do SSTables a log se vyprázdní)
  - o postavená na denormalizaci (absence JOIN operací) a datové redundanci (zapisujeme tak jak chceme číst)
  - o součást **Wide-Column family** (rozsztá se do šířky, ne do délky) a implementace ukládání stylem hodnota-klíč  $\Rightarrow$  “fancy hash table”
- **Základní funkce Cassandra**
  - o replikace: systém sám replikuje podle kritérií zadaných programátorem
  - o transparentní škálování: systém sám škáluje (programátor píše stejný kód pro lokální instanci a pro 400 uzlový cluster)
  - o nastavitelná konzistence: pro jednotlivé dotazy, i za běhu aplikace
  - o nastavitelná síťová strategie: správce nastavuje strategii, podle které se ukládají data do jednotlivých racků (množiny uzlů) a datacenter
  - o Cassandra Query Language (CQL): podobné SQL
  - o postavené na MapReduce: podpora Hadoop MapReduce
- **Architektura**
  - o **Node** – jedna Cassandra instance
    - Identifikován nejvyšším tokenem v jednom segmentu Token range
    - Odpovědný za partitions a partition keys
    - Může vlastnit jeden velký segment nebo více menších – každý menší segment je virtuální node
    - Výchozí node je v Racku 1 a Datacentru 1
    - Obsahuje v paměti:
      - *Memtables* – CQL tabulky s indexy
      - *CommitLog*
    - Obsahuje na disku:
      - *SSTables* – statické soubory periodicky ukládané z Memtables
  - o **Virtuální node**
    - chová se stejně jako normální node
    - rychlejší Bootstrap a Decommission
  - o **Bootstrap** – schopnost rovnoměrného rozdělení dat když se přidá nový node do clusteru
  - o **Decommission** – schopnost rovnoměrného rozšíření dat když se node odebere z clusteru



- **Rack** – logický set nodů
  - umístění node v racku a datacentru umožňuje geograficky vědomé směrování žádostí o zápis/čten
- **Data center** – logický set/skupina/seskupení setu racků
- **Cluster** – celý set nodes který mapuje na jeden kompletní token ring
  - node ví o svém clusteru přes conf/cassandra.yaml
- **Keyspace** – determinuje replication factor, replication strategy
- **Partition** – jedna uspořádaná a replikovatelná jednotka dat v node
  - identifikovaná tokenem = partition key – generován partitionerem
  - přidání partition do node – primární klíč tabulky nastaví hodnoty aby se zahashovali do tokenů pro užití jako Partition keys
- **Partitioner** – určuje jakým způsobem budou partitions a jejich repliky rozděleny po clusteru
  - generuje Partition keys – hashuje hodnotu primary key z žádosti o zápis na token aby byl použit jako partition key
  - Je hashovací funkce
  - **Replikace** – každý uzel si drží část segmentu clusteru (data ring), při zápisu určí partitioner (rozdělovač) hodnotu tokenu hashovací funkcí a primární replika se uloží na místo spravované daným uzlem. Podle replikačního faktoru se přidají další repliky.
    - **Typy replikace:**
      - *Murmur3Partitioner*: MurMur3 hash – rozmisťuje data rovnoměrně po clusteru
      - *RandomPartitioner*: MD5 hash
      - *ByteOrderPartitioner*: - na základě lexikálního pořadí bitů, složité vyvažování, nerovnoměrná distribuce dat, dobré sekvenční vyhledávání
    - **Replication factor** – na kolik node by měl být zápis replikován
    - **Replication strategy** – na jaký node by se měla každá replika uložit
      - Simple – jednoduchá strategie – jeden factor pro celý cluster – pro data v 1 datovém centru, určí se uzel, kde je uložena replika a pak následující po směru
        - NetworkTopology – unikátní faktor pro každý data center
          - distribuování replik přes racky a data centra
        - OldNetworkTopology - RackAware
          - distribuování dat přes racky ve stejném data centru
      - Síťová strategie – určen první uzel, kde se uloží replika a pak nejbližší po směru v jiném racku
- **Komunikace mezi uzly** – pomocí gossip protokolu – každý uzel posílá informace o sobě dalším 3 uzlům, opatřeno časovým razítkem, takže se uzly dozví rychle o problému (výpadek, přetížení...)
- **Koordinátor** – uzel vybraný klientem, který obdrží request, vybírá Cassandra driver
  - Řídí replikační proces (kolikrát se mají data na uzlech replikovat, replikační strategie)
  - Řídí úroveň konzistence – kolik nodů musí potvrdit žádost o zápis/čtení před odesláním odpovědi klientovi
    - konzistence pro zápis/čtení: ANY, ONE, ..., QUORUM, ALL
  - **Zachování konzistence**
    - **Read repair** – pokud má node nekonzistentní data tak se na pozadí pošle aktualizací požadavek a data se zaktualizují
    - **Anti entropy node repair** – opravný nástroj, který se sám dotáže na konzistenci dat a postupně je opraví
    - **Hinted handoff** – obnovovací mechanismus, když je node nedostupný
      - Coordinator uloží hinted handoff když cílový node pro zápis je offline/nepotvrdí žádost o zápis
      - Hint se uloží buď na ostatní nodes s replikami nebo na Coordinator

## - Databázový model

- **Keyspace:** udržuje pohromadě všechny CF a replikační faktor, každý keyspace může mít replikační faktor jiný.
- **Column Family (CF):** sdružuje řádky, které obsahují sloupce, mapa map (SortedMap<RowKey, SortedMap<ColumnKey, ColumnValue>)
- **Row:** řádka je identifikována jednoznačným (primárním) klíčem a obsahuje jednotlivé sloupce s daty, sloupce mohou obsahovat odlišné data ⇒ řádka nemá pevnou strukturu
- **Column:** má svůj název, hodnotu a časové razítko, které určuje čas vložení (koordinátoři se podle něj rozhodují, jestli jsou sloupce aktuální)
  - Typy sloupců
    - Standard: klasický sloupec s jednou hodnotou
    - Composite: spojený sloupec, pokud PK je složen z více sloupců
    - Expiring: sloupec s omezenou dobou platnosti
    - Counter: čítací sloupec pro zvyšování hodnoty ve sloupci
  - Sdílený sloupec – sdílený všemi řádky se stejným partition key

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

## - Apache Hbase

- CP systém (z CAP), nepodporuje SQL
- Open-source nerelační distribuovaná db
- Část Apache Hadoop
- Hosting obrovských tables
- Odolné vůči chybám při ukládání velkého množství řádkových dat
- MapReduce model
- Rychlé a konzistentní čtení (i zápis) na velkých souborech dat s vysokou propustností a nízkou latencí
- Lineární a modulární škálovatelnost

# Principy výpočtů v distribuovaných systémech – Spark framework nebo jiná podobná technologie. Principy distribuované indexace – Elastic nebo jiná podobná technologie

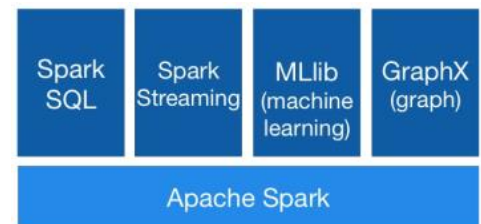
BI-BIG

## - Spark – framework pro distribuované výpočty (další např. MapReduce)

- o umí batch, stream (nekončící kolekce dat) a machine learning
- o REPL – interactive processing

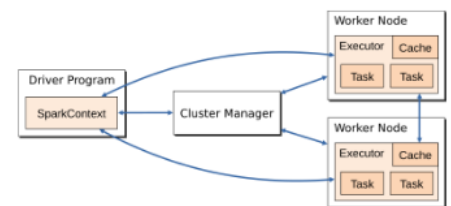
### o Architektura:

- Core – RDD (resilient distributed dataset)
- SQL – zpracování strukturovaných dat
- Streaming – datasety a DataFrame API
- MLib (machine learning) – ML algoritmy, featurization, lineární algebra, statistika
- GraphX – grafové výpočty



### o Principy:

- In-memory (narozdíl od MapReduce to nerozhází po síti)
- Fault tolerant
- Lazy evaluation (trigger je akce, vyhodnocení až v okamžiku, kdy je to opravdu potřeba)
- Immutable data, funkcionální programování
- Podporuje paralelní zpracování
- Master/slave architektura (driver program, cluster manager a worker nodes, driver komunikuje s worker nody a posílá jim tasky)

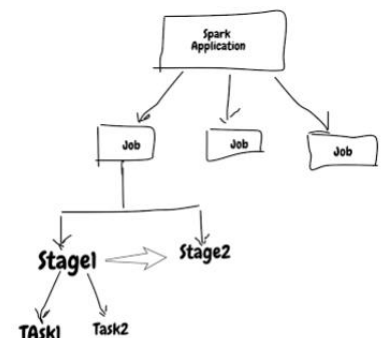


### o SparkContext – instance Spark aplikace, zodpovídá za převod aplikace na orientovaný graf (DAG) jednotlivých úloh

### o Job – počítá výsledek akce, největší akce výpočtu

### o Stage – fyzická jednotka výpočtu, od shuffle do shuffle (např. joiny, reduce, agg. fce...)

### o Task – výpočet jedné stage nad jednou částí dat, v jednom vlákně na jednom exekutoru (např. 800 tasků na 800 načtení souborů z HDFS)



### o Resilient Distributed Dataset (RDD)

- Kolekce objektů rozdělených na části
- Immutable (změna není možná), přežijí ztrátu worker nodu (ztracená partition se dopočítá) in-memory (rychlé a náročné na paměť)
- další např DataSet a DataFrame

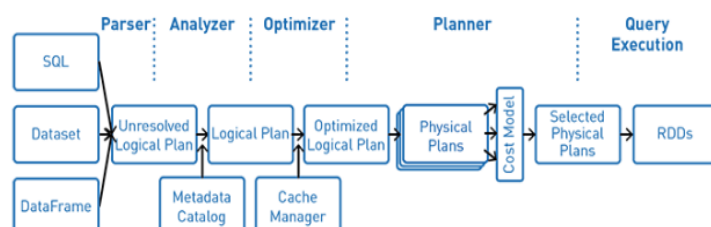
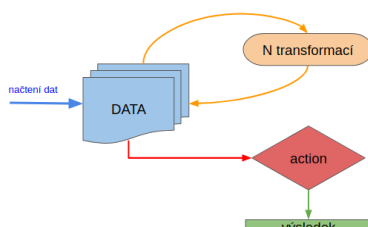
### o Transformace

- Narrow – potřebují data jen z jedné partition (map, union, filter...)
- Wide – potřebují data z více partition (groupByKey, join...), vyžadují shuffle

### o Akce – collect, take, top, count, reduce, fold, aggregate, foreach, saveAsTextFile

### o Processing pipeline – sestavuje se z transformací a akcí

- lazy evaluated – evaluace (překlad do DAG – directed acyclic graph) je spuštěna až s akcí
  - DAG se před rozdělením počítá vícekrát



- **ElasticSearch** – distribuovaný open source fulltext vyhledávač
  - Dostupný, rychlý a dobře škálovatelný
  - Komunikace probíhá přes HTTP
  - Hlavní komponenta **Elastic Stacku** (nástrojů pro ukládání, analýzu a vizualizaci dat)
  - Do ElasticSearch tečou nezpracovaná data z různých zdrojů
  - Nejprve jsou tato data analyzována, normalizována a poté jsou indexována
  - **Indexace:**
    - Po indexaci mohou uživatelé spouštět komplexní dotazy a získávat souhrny svých dat
    - Používá invertovaný index pro rychlé fulltextové vyhledávání (invertovaný index je tvořen jedinečnými slovy, které se objevují v celém setu dokumentů, a identifikuje všechny dokumenty, kde se toto slovo nachází, jako ve VWM)
    - Během indexovacího procesu Elasticsearch ukládá dokumenty a vytváří **invertovaný index**, aby bylo možné prohledat data dokumentů v reálném čase
    - Indexování je inicializováno index API, kde se dají přidávat/aktualizovat JSON dokumenty v konkrétním indexu
  - **Elasticsearch Index**
    - Kolekce navzájem souvisejících JSON dokumentů  
každý dokument koreluje množinu klíčů (názvy polí) s odpovídajícími hodnotami (stringy, čísla, geolokace a jiné typy dat)



# Dědičnost atributů a metod v jazyku Java, souvislost s kovariantními typy, statičností a nestatičností, seznamem vyhazovaných výjimek a viditelností (private, protected, public, default).

BI-PJV

- **OOP** – nahlíží na entity a vztahy jako modely věcí z reálného světa
  - o Abstrakce – zobecnění pohledů na částečná řešení
  - o Dědičnost – získávání vlastností a schopností děděním z vhodné třídy a případně i dat nebo částečných požadavků z interface
  - o Polymorfismus – jednotné zacházení s různými objekty mající některé společné zděděné schopnosti
  - o Zapouzdření – skrývání částí tříd a interface
- **Třída** – „hotový plán“ – charakterizována vlastnostmi které uchovává (data,, členské proměnné) a funkčními možnostmi (metody)
  - o **Objekt** – instance třídy (vytvářet pouze dynamicky pomocí new)
  - o Statické třídní proměnné
  - o Nestatické instanční proměnné objektu
  - o Statická metoda x instanční metoda (operace nad objektem)
  - o Po vytvoření objektu se jeho atributy inicializují nulovými hodnotami
  - o „extends“ pro dědění
- **Rozhraní – interface** – norma, požadavek, vyžaduje implementaci
  - o Lze považovat za značně omezenou abstraktní třídu, která má všechny metody abstraktní
  - o Od Java 8 ještě default a static metody
  - o Umožňuje částečný pohled na třídu a slouží jako norma či požadavek
  - o Není žádný základní interface – nemusí mít žádného předka
  - o Může být přímým potomkem více interfaců uvedených v seznamu za klíčovým slovem extends
  - o Nemůže být potomkem žádné třídy
  - o Nemůže být finální
- **Abstraktní třída** – neúplný plán
  - o Může být i přímý potomek konkrétní třídy
  - o Může implementovat více rozhraní
  - o Může, ale nemusí mít abstraktní metody
  - o Modifikátor abstract
  - o Alespoň jeden konstruktory, ale nelze podle ní objekty vytvářet
  - o Nemůže být finální
  - o Statický kontext je funkční = static atributy, metody, inicializátory a výčty
  - o Abstraktní metoda v definici místo těla jen středník
- Třída extends třída, rozhraní extends rozhraní, třída implements rozhraní
- **Object** – prátřída všech tříd, není nutné uvádět za extends – každá třída je přímým potomkem
  - o Nemá předka – nic nedědí
  - o Nemá atributy
  - o Definuje 11 generálních metod, všechny ostatní třídy je dědí
    - toString, equals, hashCode, clone, getClass, notify, wait, ...
- **Dědičnost** (inheritance)
  - o Vztah třída a nadtržída – superclass, přímou nadtržidu vyznačuje třída ve své hlavičce za klíčovým slovem extends, jinak je přímým předkem Object
  - o Vztah třída a podtržída – subclass
  - o Dědičnost usnadňuje:
    - Přístup k existujícím třídám a interfacům a jejich pochopení

- Vytváření nových tříd a interfaců s využitím vlastností a schopností již existujících, ověřených a dokumentovaných předků
- **Dědičnost atributů**
  - Rozhraní (interface) může mít jen konstanty = **public final static** atributy
  - Při vícečetném dědění stejnojmenných atributů nastane kolize
  - U atributů je vazba známa již při překladu = nepočítá se za běhu
  - Atributy v potomkovi můžeme **překrýt** – variable hiding
  - Jedná se o jiný atribut téhož jména, překrývající atribut:
    - Nemusí zachovat statičnost, či nestatičnost
    - Nemusí zachovat viditelnost
    - Může mít jiný typ
- **Dědičnost metod**
  - Dědí se pouze viditelné členy – atributy, metody, vnitřní třídy, vnitřní interfaci, výčty
  - **Viditelnost** – nesmí mít příliš restriktivní přístupový modifikátor
    - **Public** = dědí a vidí všichni potomci, vidí všichni
    - **Protected** = dědí a vidí všichni potomci
    - **Default = friend** = dědí potomci v témže balíčku, vidí všichni ve stejném balíčku
    - **Private** = nevidí a nedědí nikdo
  - Potomek nemůže dědictví odmítnout, a tedy není nikdy chudší než předek
- **Statické metody a dědičnost**
  - Statické metody mohou být definovány v: Třídách, Abstraktních třídách, Rozhraních
  - Vlastnosti statických metod:
    - Nemohou být abstraktní – vždy mají tělo
    - Lze volat pomocí jména třídy
    - Lze volat pomocí referenční proměnné (nedoporučuje se)
    - Nelze „přepsat“ (Override)
    - Lze překrýt v potomkovi (podobně jako u atributů)
    - Překladač určí vazbu v době překladu
- **Přepsání metody – overriding**
  - Nevyhovuje-li potomkovi zděděná nefinální metoda, lze v potomkovi deklarovat metodu, která:
    - Má shodnou signaturu
    - Má kovariantní návratový typ či subtype – avšak ne primitivní
      - **Kovariantní návratový typ** – návratový typ přepisované metody – povoluje „zúžit“ návratový typ přepsané metody bez potřeby castu nebo kontroly náv. typu
        - Metoda podtřídy může vrátit podtyp návratového typu té metody z nadtřídy
        - Např. když je v nadtřídě metoda co vrací hodnotu typu List, tak podtřída může vrátit List, ale taky kterýkoliv z podtypů Listu – ArrayList, Stack, Vector, ...
    - Nemění statičnost – nestatičnost
    - Nezužuje modifikátor přístupu
    - Nerozšiřuje množinu kontrolovaných výjimek udaných za throws
- **Zapouzdření (encapsulation)**
  - Ve třídách a objektech lze ukrýt atributy, metody, konstruktory, vnitřní třídy a vnitřní interfaci a tím podpořit spořádanost a bezpečnost
  - Inicializátory jsou skryté již tím, že nemají jméno
  - Míru zapouzdření určuje modifikátor viditelnosti členu či konstruktoru (public, ...)
  - K nedostup. členům nějaké třídy lze přistupovat z jiné třídy jen pomocí neprivatních metod oné třídy
  - V interfezech lze zapouzdřovat jen atributy a jen úrovní „default“
- **Hierarchie tříd**
  - Vztah nadtřída-podtřída je tranzitivní – jestliže je x nadtřídou y a y nadtřídou z, je x nadtřídou z

- Pro referenční proměnné platí: do proměnné typu Tnad může být přiřazena ref. na objekt typu Tpod
- Na objekt referencovaný proměnnou typu Tnad lze vyvolat pouze metodu deklarovanou ve třídě Tnad
- Jde-li však o objekt typu Tpod, metoda se provede tak, jak je dáno třídou Tpod
- Hodnotu referenční proměnné typu Tnad lze přiřadit referenční proměnné typu Tpod pouze s použitím přetypování, které zkontroluje, zda referencovaný objekt je typu Tpod
- **Gettery a settery** – dobrovolná jmenná konvence pro přístupové metody k zejména privátním atributům, které jejich hodnotu vydávají (getter, accessor) anebo mění (setter, mutator) usnadňuje tvorbu nadstavbového softwaru
- **Použití dědičnosti – substituční princip** – dědičnost, pokud je podtřída skutečně specializací nadtříd
  - Všude, kde lze použít nějaká třída, musí jít použít i její podtřída, aniž by uživatel poznal rozdíl
  - Netýká se jen syntaxe, ale i sémantiky (tu je nutné vhodně popsat)
  - Podtřída musí dodržovat kontrakt nadtříd
  - Vztah (ISA) musí být trvalý
  - Odvozené třídy nesmí nikdy vyžadovat více (precondition)
  - Odvozené třídy nesmí poskytovat méně než bazová třída (postcondition)
- **Přetypování – casting**
  - Objekty, atributy, lokální proměnné, parametry a návratové hodnoty metod mají definovaný, neměnný typ – dva druhy typů:
    - **Primitivní** (boolean, char, byte, short, int, long, float, double)
    - **Referenční** – třídy, interfejsy a pole (primitivní i referenční)
  - **Přetypování** = změna pohledu na objekt prostřednictvím reference jiného typu, typ objektu je neměnný, daný třídou, dle níž byl zkonstruován
  - Přetypovat potomka na předka lze přiřazením
  - Přetypovat (casting) předka na potomka lze jen tehdy, je-li referovaný objekt typu potomka takto:
 

```
Potomek y = (Potomek) x;
```
  - Chybné přetypování → výjimka `ClassCastException`
- **Výjimky** – objekty, slouží k indikaci závad programu
  - Jejich třídy tvoří podstrom třídy `Throwable`, od které dědí vesměs všechny své metody
  - Kompilátor v čase kompilace (compile-time) rozeznává výjimky:
    - **Kontrolované (checked)** – vyžaduje ošetření
      - Klauzulí try-catch
      - Vyznačení v hlavičkách metod
    - **Nekontrolované (unchecked)** – kompilátor nevyžaduje vyznačení ani ošetření
      - Považovány za programátorské chyby, které budou odstraněny
  - Pokud výjimka patří do stromu `java.lang.RuntimeException`, je nekontrolovaná, jinak je kontrolovaná
  - **Odchyťování výjimek** – klauzule try může být vnořena do kteréhokoli bloku, a tedy i do bloku jiné kl. try
    - Typy bloků:
      - Hlídač — **try** (právě jeden) - dozírá, byla-li v bloku vyhozena
        - Výjimka nebyla vyhozena – dokončí se blok try a předá řízení uklížeči = finally
        - Výjimka byla vyhozena – další příkazy v try bloku se již neprovedou – hlídač zjistí typ výjimky
          - Procházením řešitelů shora dolů se snaží najít prvního kompetentního řešitele = jeho parametr je typem nebo nadtypem výjimky – nalezne-li ho, předá mu řízení
          - Nenalezne-li řešitele, předá řízení uklížeči a pak nevyřešenou výjimku vyhodí do nadbloku
      - Řešitel — **catch** (libovolný počet) – má prostřednictvím parametru referenci k aktuální výjimce i dalším proměnným

- Může situaci trochu napravit anebo alespoň řádně ohlásit, nelze však nijak zařídít pokračování v nedokončeném hlídaném bloku
- I když řešitel vůbec nic neudělá, je odchycená výjimka „vyřešena“
  - Uklízeč — **finally** (nanejvýš jeden)
- Klausule try – catch – finally může mít tři tvary:
  - try – catch – finally
  - try – catch
  - try – finally
- I řešitelé a uklízeč mohou vyhodit nějakou výjimku – tu však zpracuje (dynamicky) obalová try klauzule. Též lze sestavit novou výjimku a jako její příčinu vložit tu původní.
- Šíření neodchycené výjimky po dokončení bloku finally
  - Je-li volajícím blokem obalová try klauzule postupuje se obdobně, jako v případě nalezení odpovídající catch části ve volaném bloku
  - Je-li tím blokem metoda, pak tato metoda vyhodí výjimku do příkazu metody odkud byla zavolána a tam se postupuje obdobně
  - Nenalezne-li se žádný kompetentní řešitel, vyhodí se výjimka do obalové kl. JVM – odtud byla zavolána metoda main(String[ ] args)- pak JVM vypíše hlášení a ukončí běh
- **Vlastní výjimky** – lze vytvářet vlastní výjimky jako potomky existujících většina nepřidává žádné vlastní atributy ani metody, jen definuje dva konstruktory a vše ostatní dědí z třídy Throwable
- Try with resources – zdroje typu AutoCloseable – existuje-li blok try, budou na jeho konci všechny zdroje automaticky uzavřeny – úhlednější
- **Řetězení výjimek** – pokud při zpracování výjimky vznikne další, je původní potlačena (supressed), ale je stále dosažitelná

# Vstup a výstup v Javě, rozdělení proudů, jejich navazování, serializace a externalizace dat, výjimky, jejich rozdělení a zpracování

BI-PJV

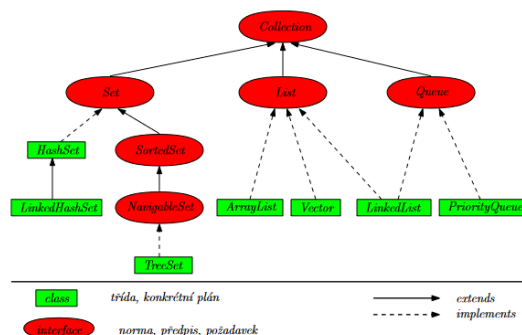
- **IO vs. NIO**
  - o IO – proudově orientované, Stream
    - IO operace jsou blokuující, vlákno stojí, dokud nejsou data načtena/zapsána
  - o NIO – používá Buffer, lze využít pro vstup i výstup
    - Operace IO jsou neblokuující, programátor kontroluje, zda jsou data připravena
    - Přináší Selector – umožňuje jednomu vláknu řídit více kanálů (Channel)
    - Path, Files, Stream API, funkcionální programování
- **Soubory**
  - o Typické **operace**
    - Otevření souboru
    - Čtení údaje
    - Zápis údaje
    - Uzavření souboru
  - o Přístup k údajům (čtení nebo zápis) může být:
    - Sekvenční nebo libovolný (adresovatelný)
  - o Sekvenční přístup umožňuje pouze postupné (sekvenční) čtení nebo zápis údajů – použitelný pro soubory, roury mezi procesy, sockety přes internet...
  - o Soubory s libovolným přístupem umožňují adresovatelné čtení nebo zápis údaje (podobně jako pole)
  - o Způsob přístupu k údajům v souboru je dán programem
  - o Zde se budeme zabývat hlavně sekvenčními soubory
- **Soubory a proudy** – Java rozlišuje soubory (file) a proudy (stream)
  - o **Soubor** = množina údajů uložená ve vnější paměti počítače
  - o **Proud** = nástroje k přenosu informací např. z/do souboru, ale také do/ze sítě, paměti, jiného programu atd.
  - o Informace může mít tvar znaků, bajtů, skupin bajtů (obrázky, ...), objektů, ...
  - o Přenos informace se děje dvoj/třívrstvě v proudech (streams):
    - Otevření přenosového proudu pro bajty či znaky
    - Otevření přenosového proudu pro datové typy Javy
    - Filtrace dat podle požadavků – bufferování, řádkování, ...
- **Rozdělení proudů:**
  - o **Textové proudy** – abstraktní třídy Reader a Writer musí zajistit konverzi znaků mezi OS a Javou
    - Public metody, vyhazují výjimku **IOException**
  - o Abstraktní třídy InputStream, OutputStream
    - Public metody, vyhazují výjimku **IOException**
  - o **Překlad bytů na znaky** – InputStreamReader a OutputStreamWriter
  - o Interface DataInput a DataOutput – definují (abstraktní) **metody pro čtení/zápis** všech primitiv z/do proudu
  - o FileOutputStream – **kopírování souboru** po bytech
- **Serializace objektů** do výstupu
  - o Serializable objekty lze přenést objektovým proudem tzv. serializací
  - o Rozloží hodnoty jeho atributů na byty:
    - Modifikátory přístupu nemají vliv, všechna pole jsou serializovatelná
    - Neserializují se atributy označené static či transient ani třídy ani metody
    - Rekurz. probírkou referencí se serializují i všechny refer. objekty tedy i velmi rozsáhlý graf
- **Obnovení objektů** – **deserializace** vstupu

- K dispozici musí být odpovídající třídy – kompatibilní dle serialVersionUID
- Nevolá konstruktory Serializable tříd – nechceme iniciální hodnoty
- Volá konstruktory nadtříd, které nejsou Serializable
- Nepřenesené atributy nastaví na jejich defaultní hodnoty
- **Identifikace serializovatelné třídy** – serialVersionUID
  - Stanovené výpočtem (hešová hodnota) či přiřazením
  - U přeložené třídy lze zjistit programem serialver
  - Při serializaci se přikládá do objektového proudu – viz třída java.io.ObjectStreamConstants
  - Při deserializaci se porovná s hodnotou třídy na vstupní straně
  - Při **neshodě se vyhodí výjimka**
  - Programátor může explicitně stanovit, že třídy jsou kompat. tím, že třídě vnutí určitou hod. Atributu
- **(De)serializace a dědičnost** – pokud serializujeme instanci potomka (musí být Serializable) pak
  - I rodič implementuje Serializable a pak je serializován v rámci potomka
  - Nebo předek musí mít bezparam. konstr. (implicitní či explicitní). Tento je volán během deserializace
  - Naše vlastní ovládání serializace a deserializace - (De)serializace funguje automaticky, ovládáme výběr a pořadí
- **Externalizace** – vlastní „serializace“ – java.io.Externalizable – implements Externalizable + explicitní public konstrukt bez parametrů a dvě metody plně zodpovědné za ser-deser objektu i dat nadtříd
  - Externalizace je rychlejší než serializace, neboť JVM neanalyzuje strukturu
- **Komprese** – třída Deflater – můžeme vytvořit standardní soubor .zip, .gzip nebo .jar, který je čitelný standardními programy, např. gzip, unzip, WinZip
- **Dekomprese** – třída Inflator – Takto lze dekomprimovat standardní .zip, .gzip a .jar.
- Třída **RandomAccessFile** – adresovatelný přístup, čtení, zápis i přepisování jakoby bytového pole realizovaného na periferním zařízení
- Collator – české řazení

# Kolekce a JCF, jejich rozdělení dle implementovaných rozhraní, kolekce a metody hashCode, equals a implementace Comparable, synchronizace kolekcí

BI-PJV

- **Kolekce** (= kontejnery) – objekty, které slouží k ukládání, načítání, zpracování a přenášení prvků stejného typu
  - o **JCF = Java Collections Framework** – sada tříd a rozhraní, která implementují běžně opakovaně použitelné datové struktury kolekce
  - o Algoritmy kolekcí – polymorfní realizace metod tříd
  - o Pomalejší než pole
  - o **Výhody** použití kolekcí
    - Ulehčení programování
    - Kratší a přehlednější kód
    - Rychlejší algoritmy
    - „neomezený rozsah“ počtu ukládaných objektů
    - Zvýšení čitelnosti programů
    - Lepší přenositelnost
- Genericita – do kolekcí je možné vkládat instance třídy Object a při výběru přetypovávat zpět
- Nadrozhraní **Collection** – potomci rozhraní Set, List, Queue
  - o **Set** – ADT množina, prvky se neopakují, může být seřazená
    - Oproti Collection nepřidává žádné další metody
    - Implementace – HashSet, LinkedHashSet, TreeSet, ...
  - o **List** – ADT množina, prvky mají indexy, prvky se mohou opakovat
    - Základní implementace – ArrayList, LinkedList – JCF, Vector
    - Indexovaný přístup prvkům
  - o **Queue**, Dequeue – ADT fronta, obousměrně zřetěžená fronta, LIFO, FIFO, prvky se mohou opakovat
    - Potomci BlockingQueue, Deque, LinkedList, PriorityQueue
- Další boís
  - o Rozhraní Iterator, ListIterator
  - o Třída **ArrayList** (extends ArrayList) – implementace List pomocí natahovacího pole, podobné Vector
  - o Třída **LinkedList** (extends AbstractSequentialList) – patří do JCF, obousměrně zřetěžený seznam jako implementace List
    - Implements List, Deque, Cloneable, Serializable
  - o Vector, Stack extends Vector – nejsou součástí JCF, synchronizované
  - o Třída **HashSet** – ADT množina, patří do JCF
    - Extends AbstractSet, implements Set, Cloneable, Serializable
    - Negarantuje pořadí procházení prvků
    - **Hešování** – technika, která v ideálním případě zaručí vložení, odebrání, zjištění přítomnosti prvku v konstantním čase
      - **Hešovací funkce** – zajišťuje mapování prvků kolekce na int, který slouží k výpočtu indexu do kolekce
        - o Ideálně pro dva různé prvky vytvoří **hashCode** dvě různé hodnoty
        - o Mnohdy to nejde, např String, počet různých řetězců výrazně převyšuje int
  - o Hashtable – extends Dictionary, implements Map
    - Index počítán přes modulo velikosti – rychlý výpočet a při zvětšování rychlé rozdělení zřetěžených kyblíků
    - Významné nižší bity
  - o HashMap – extends AbstractMap implements Map
    - Index počítán bitovým násobením



- Vylepšení výpočtu indexu kyblíku (bucket) – významné nižší bity – ztráta efektivity
    1. Výpočet hashCode
    2. Přepočet pomocnou metodou
    3. Bitové maskování podle velikosti kolekce
- Metody **hashCode** a **equals** – úzce spolu souvisí
  - Pokud zastíníme equals, musíme zastínit hashCode
  - Správně vytvořený objekt musí splňovat:
    - Pokud jsou dva objekty stejné (podle equals) musí metody hashCode vracet stejnou hodnotu
  - **Equals** – implementace relace ekvivalence
    - **Reflexivní** – `x.equals(x)` musí vrátit true, pro každé x (**mimo null**)
    - **Symetrická** – pro jakékoliv x a y musí `x.equals(y) == true` právě tehdy, když `y.equals(x) == true`
    - **Tranzitivní** – pro jakékoliv x, y, z musí platit, že když `x.equals(y) == true` a `y.equals(z) == true`, pak `x.equals(z) == true`
    - **Konzistentní** – pro jakékoliv odkazové hodnoty x a y musí platit, že buď `x.equals(y)` vrací true nebo stále vrací false za předpokladu, že nedojde ke změně žádných informací použitých v porovnáních equals daného objektu
    - Pro všechny odkazové hodnoty x, které nejsou null, musí `x.equals(null)` vrátit false
  - **hashCode** – s ohledem na equals zamixuje důležité vlastnosti objektu
    1. uložte magickou prvočíselnou konstantní nenulovou hodnotu (např. 17) v proměnné typu int nazvané result
    2. pro každý významný atribut f ve svém objektu = každý atribut zvažovaný metodou equals, vypočítejte kód tmp typu int (podle datového typu)
    3. `result = result * něco + tmp`
    4. pokračujeme s dalším atributem a bodem 2
    5. vraťte výsledek, pokud již nejsou žádné další důležité atributy
      - když konečně skončíte s vytvářením metody hashCode(), zeptejte se sami sebe, zda mají rovné instance rovné hešovací kódy.
      - Pokud ne, zjistěte proč a problém opravte.
      - Vždy nejprve vytvořte equals (případně spolu s využitím IDE)
- **Kolekce a řazení** – rozdělení
  - Způsoby řazení
    - Přirozené řazení, třída implementuje **Comparable**, pokud ne, nastane chyba za běhu
      - Metoda sort používá vylepšený quickSort
      - Metoda sort volá compareTo
    - Řazení s komparátorem
      - **Komparátor** = třída implementující rozhraní **Comparator** a tedy metodu compare (o1, o2), pokud není typována
      - Každá třída může mít 1 způsob přirozeného řazení a kolik chceme komparátorů
  - Aplikace řazení
    - Využití seřazené kolekce
    - Dodatečné seřazení neseřazené kolekce pomocí Collections
  - Vztah mezi equals, compareTo, compare a hashCode
    - List a Queue požadují equals
    - compareTo, compare musí být pro binarySearch, seřazené kolekce, řazení
    - hashCode je pro všechny kolekce s názvem HashNěco
    - `o1.equals(o2) == true ⇔ o1.compareTo(o2) == 0`
    - musí platit i pro compare a to pro všechny komparátory
- Další supr rozhraní
  - Interface SortedSet – extends Set



- Prvky se nesmí duplikovat – nesmí obsahovat prvky e1, e2, pro které platí `e1.compareTo(e2)==0`, srovnej s `e1.equals(e2)==true`
- SortedSet uchovává prvky ve vzestupném přirozeném pořadí nebo v pořadí daném komparátorem
- Interface Map – asociativní pole
  - Mapa – map = 'zobrazení keys -> values, tedy klíčů na hodnoty
- **Synchronizace kolekcí**
  - U jednovláknových aplikací není třeba synchronizovat
  - U vícevláknových je nutné zajistit, aby jedno vlákno nepřistupovalo ke kolekci, kterou upravuje jiné vlákno = nutno synchronizovat
  - Kolekce jsou standardně nesynchronizované, a tak rychlejší
  - Synchronizace obalením JCF kolekce
  - Synchronizace použitím staré, ale synchronizované kolekce
    - Možno použít Vector, Stack, Hashtable

# Java 8 - Stream API a lambda výrazy, rozdělení operací nad streamy, funkční rozhraní Predicate, Function, Supplier, atd

BI-PJV

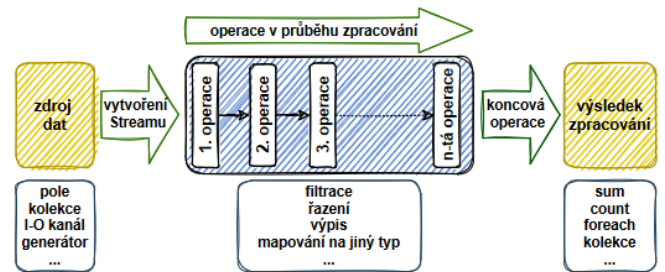
- **Lambda výrazy** – výrazy, které lze předávat v programu podobně jako proměnné bez nutnosti je obalovat do rozhraní
  - o Typ dynamicky určen podle kontextu
  - o V podstatě funkce bez formální deklarace
  - o Pomocí lambda výrazů lze procházet kolekce
- **Java 8 – funkční rozhraní**

```
MojeFunkce naDruhou = (int i) -> {  
    return i * i;  
};  
→  
MojeFunkce naDruhou = (i) -> i * i;  
MojeFunkce naDruhou = i -> i * i;
```

- o Informaci o typech můžeme vynechat (typová kontrola stále funguje), a také složené závorky a return
- o Pokud má metoda funkčního rozhraní jen jeden parametr, můžeme vynechat i kulaté závorky
- o Závorky nemůžeme vynechat, když není žádný parametr, nebo je jich víc
- o Funkce často nebudeme deklarovat jako proměnné
  - Funkci prostě vytvoříme a hned předáme jako argument metodě nebo konstruktoru, které přijímají dané funkční rozhraní
- o Odkazy na metody – v Javě 8+ se pomocí nich můžeme odkazovat na metody i konstruktory a konvertovat je na funkce, resp. funkční rozhraní
- o Odkazy na konstruktory – metoda nebo konstruktor musí mít vhodný počet a typ parametrů, které odpovídají požadovanému funkčnímu rozhraní
- o V praxi část Supplier<Long> supplier často vynecháte, nebudete deklarovat proměnnou a jen předáte "odkaz na metodu" někam, kde je požadováno funkční rozhraní
- o **Vlastní funkční rozhraní** – java.util.function
  - Funkce obvykle nebudeme používat hned (to bychom mohli rovnou zavolat ten kód), ale pravděpodobně je předáme do jiné části programu, kde se budou volat
- o **Skládání funkcí** - u compose() se nejdříve zavolá vnitřní funkce (ta předaná jako parametr)
  - Místo compose() můžeme pro skládání použít andThen(), kde to funguje přesně naopak
  - Výsledkem skládání funkcí je opět funkce s patřičnými generickými
  - Typy – nemusíme hned apply(), ale můžeme si ji uložit do proměnné a používat opakovaně nebo předat někam dál
- o **Predikáty** – Predicate – obor hodnot je boolean
  - Definiční obor je daný generickým typem
  - Metoda Predicate.test(T t) vyhodnocuje pravdivost predikátu nadzadanou hodnotu
  - Skládání predikátů – stejně jako javovské výrazy pospojované AND a OR operátory se i predikáty vyhodnocují zleva doprava a zkráceně
    - Stejně jako u klasických výrazů s && a || je dobré i predikáty řadit podle výpočetní/paměťové složitosti a podle pravděpodobnosti získání výsledku
- **Stream API** – definice **roury**
  1. Vytvoření zdroje, proudu (kolekce, pole, generující funkce, I-O kanál, ...)
  2. Operace zpracovávající Stream – 0 až N
  3. Koncová operace
    - o Data se nemění,
    - o „mezivýsledky“ jsou Stream
    - o Operace se řetězí
    - o Použitelné jen jednou
    - o Výpočet začne až po koncové operaci

- **Proud – stream** – vytvoření proudu:

- Z kolekcí prostřednictvím metod `stream()` a `parallelStream()`
- Z polí prostřednictvím metody `Arrays.stream(Object[])`
- Z továrních metod třídy `Stream`, např.: `Stream.of(Object[])`
- `IntStream range(int, int)`
- `Stream iterate(Object, UnaryOperator)`
- Řádky souboru lze získat metodou `BufferedReader.lines()`
- Proud náhodných čísel lze získat metodou `Random.ints()`
- Proud položek v ZIP-souboru lze získat metodou `JarFile.stream()`
- **Práce s proudy**
  - Obsah proudu můžeme během práce filtrovat
  - Proudů můžeme také řadit
  - Prvky proudu můžeme konvertovat na jiný typ



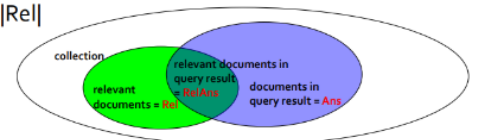
- **Funkční rozhraní** – funkční rozhraní je nový termín používaný pro rozhraní (interface), která mají právě jednu abstraktní metodu

- Známe `Comparable`, `Runnable`
- Nově `java.util.function`
- **Consumer** = čistí konzumenti – nic nevrací
  - `Consumer<T> void accept(T value)`
- **Supplier** = producenti – nemají parametry a vracejí hodnotu zadaného typu
  - `Supplier<T> T get()`
- **Function** = funkce – zpracují svůj parametr a vrátí funkční hodnotu
  - `Function<T, R> R apply(T value)`
- **Operátory** – unární, binární, ... návratová hodnota stejného typu jako parametr
  - `UnaryOperator<T> T apply(T operand)`
- **StringJoiner** – operace merge – nekládá prefix a sufix, různé oddělovače – druhý seznam vložen jako jedna položka
  - `Public StringJoiner(CharSequence delimiter, CharSequence prefix, CharSequence suffix)`

# Techniky pro vyhledávání textových, webových a multimediálních dokumentů: modely, algoritmy, aplikace. Optimalizace webových stránek pro vyhledávače

BI-VWM

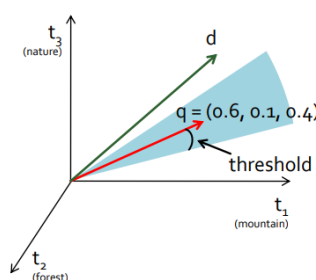
- **Dokument** – full-text objekt nebo anotace jiného objektu na webu
  - o Kolekce = množina dokumentů
  - o Term = slovo (fráze), které se objevuje v dokumentu
  - o Slovník = množina všech různých termů v souboru dokumentů
- Prvky vyhledávání na webu
  - o crawling – stahování obsahu
  - o indexing – zpracování obsahu do formátu vhodného pro vyhledávání
  - o searching – získávání relevantního obsahu pomocí query
- **Modely získávání informací z webu:** (často se kombinují)
  - o *Query* – jednorázové hledání na základě přesného dotazu
    - algoritmus vrátí množinu vyhovujících dokumentů (seřazenou podle ranku nebo neseřazenou)
  - o *Browsing* – iterativní navigace v databázi, neznáme přesný dotaz
    - manuálně prohledáváme explicitní (linky mezi entitami) nebo virtuální (série queries) graf
    - algoritmus vrátí podmnožinu db objektů, set clusterů nebo hierarchii
  - o *Filtering* – formulace pevného pořadavku (explicitní – statický dotaz x implicitní – doporučování)
    - algoritmus vrací dynamicky měnící se odpověď pro explicitní dotaz (např. subscription kanálu na YouTube) nebo implicitní (např. doporučení na YT)
- Pro vyhledávání v souboru dokumentů je možné použít klasické string-matching algoritmy (např. Knuth-Morris-Pratt), ale ty jsou pomalé. Dokumenty se musí předzpracovat a indexovat.
- **Předzpracování kolekce**
  - o Odstranění stop-slov, které nic neřeknou o obsahu (výplňková slova)
  - o Lemmatizace – zjednodušení termů (např. uvedení slov do 1. pádu)
- **Měření kvality odpovědi**
  - o Relevantní dokument – výsledkem dotazování na kolekci
  - o Efektivita = míra uživatelské spokojenosti s výsledkem
  - o **Precision** – pravděpodobnost, že dokument ve výsledku je relevantní  $P = |\text{RelAns}| / |\text{Ans}|$
  - o **Recall** – pravděpodobnost, že relevantní dokument je ve výsledku  $R = |\text{RelAns}| / |\text{Rel}|$
  - o **False alarm** – dokument není relevantní, ale byl tak vyhodnocen
  - o **False dismissal** – dokument je relevantní, ale nebyl tak vyhodnocen
  - o **P-R křivka** (precision-recall křivka) – ukazuje závislost precision a recallu – typicky kompromis
- **Booleovský model** – vytváří binární term-by-document matici
  - o Dokument = množina termů x term = množina dokumentů, ve kterých se vyskytuje (lepší pro dotazování)
  - o Řádek = vektor termů, sloupec = vektor dokumentů
  - o 0 = term v daném dokumentu není, 1 = je
  - o Hodně řídká matice, proto se implementuje pomocí invertovaných seznamů
  - o **Invertovaný seznam** – každý term má vlastní list, ve kterém jsou uvedeny seřazené (podle ID) dokumenty, ve kterém se term nachází, výsledek je potom daný "merge sortem"
  - o Operace AND, OR, NOT
  - o Výsledky dokonale reflektují dotaz, člověk musí vědět co hledá
  - o Pozitiva – jednoduchý na implementaci, efektivní pomocí invertovaného indexu
  - o Negativa – všechny dokumenty mají stejnou váhu



q = Brutus AND Caesar  
result = (2, 8)

Brutus	2	4	8	16	32	64	127	
Caesar	1	2	3	5	8	13	21	34

- **Rozšířený booleovský model** – implementuje booleovský model + zavádí **váhy termů** podle toho, jak jsou v dokumentu důležité
  - o kombinace booleovské algebry s vektorovým modelem
  - o vyhazuje seřazené výsledky
  - o umožňuje hledat i částečně relevantní dokumenty, ne jen plně relevantní
  - o *pozitiva*: stále jednoduché pro uživatele, možnost specifikace velikosti odpovědi, efektivnější než standardní model
  - o *negativa*: výpočetně náročné
- **Vektorový model** = bag of words model
  - o Dokument = bag of terms (bag = multi-množina, query = bag of terms, jako dokument)
  - o hledá podobné dvojice dokumentů
  - o každý dokument je vektor dimenze  $m$  ( $m = \#$  termů ve slovníku), který obsahuje váhu každého termu ze slovníku
  - o každá dimenze prostoru patří nějakému termu ze slovníku
  - o dotaz  $q$  se převede také na vektor a hledají se vektory jemu nejbližší na základě jeho směru (podle kosinové vzdálenosti), takže podobné vektory si budou blízké, odlišné naopak vzdálené – tím se dá odlišit, jaké dokumenty do odpovědi ještě zahrnout (nastavuje se nějaký threshold)
  - o např. při hledání plagiátorství nebo vyhledávání v Googlu
  - o opět velmi řídký, takže se znovu používá invertovaný index, kde každý term si v seznamu drží dokument a svou váhu v tomto dokumentu
  - o *pozitiva*: lze vyhledávat i podle příkladu, efektivní a jednoduché, výsledky jsou seřazené
  - o *negativa*: neporadí si s homonymy a synonymy, geometrizací se v modelu ztrácí sémantika informace, syntax není zahrnutá
- **LSI vektorový model** (latent semantic indexing) – úprava vektorového modelu
  - o cíl je adresovat problém homonym a antonym a snížit dimenzi vektoru
  - o využití metody SVD na předzpracování term-by-document matice (výsledná matice je **concept-by-document**, dokument tedy není modelován termy, ale koncepty)
  - o tato matice se vynásobí s vektory v  $U$  reprezentující koncepty, které jsou seřazeny podle důležitosti (důležitost rychle klesá, takže se bere jen  $k$  konceptů)
  - o dotaz se zobrazí do prostoru konceptů jako  $q_k = U_k^T q$  a opět proběhne porovnání podle kosinové vzdálenosti
  - o *pozitiva*: zahrnutá sémantika v dokumentu, částečně řeší homonym a synonyma, redukce dimenzionality (pomocí SVD)
  - o *negativa*: drahé předzpracování, není vhodný invertovaný index, koncept nemá nic společného s lingvistikou, jen je to lineární kombinace termů
- **Word2Vec**: rychlejší, lepší
  - o Machine learning přístup – nesupervizované učení na korpusu
  - o Dva modely – continuous bag of words, skip-gramy
  - o NLP
- **Optimalizace webových stránek pro vyhledávače (SEO)**
  - o způsob úpravy webových stránek, aby byly dobře hodnocené vyhledávači
  - o obsah stránky musí být přehledný pro člověka i stroj
  - o hodnotí se každá stránka, nikoliv doména (nutnost optimalizace pro každou stránku)
  - o hodnotí se především tyto prvky:
    - *název souboru* – krátké a relevantní (co nejkratší URL)
    - *<title> tag* – nejdůležitější informace o stránce, stručný a výstižný
    - *header tagy* - důležité informace pro vyhledávače i v tagách <h1> - <h6> (sestupně)



$$A = U \Sigma V^T \approx A_k = (U_k U_0) \begin{pmatrix} \Sigma_k & 0 \\ 0 & \Sigma_0 \end{pmatrix} \begin{pmatrix} V_k^T \\ V_0^T \end{pmatrix}$$

$$\left[ \begin{matrix} A \\ m \times n \end{matrix} \right] \approx \left[ \begin{matrix} U_k \\ m \times k \end{matrix} \right] \left[ \begin{matrix} \Sigma_k \\ k \times k \end{matrix} \right] \left[ \begin{matrix} V_k^T \\ k \times n \end{matrix} \right]$$

- *meta tagy* – dnes už nejsou důležité, ale stále je to způsob, jak předat informace uživateli
  - *textové modifikátory* - <strong>, <b> ...
  - *optimalizace obrázků* – nevyužívat obrázek jako link, počítače jim zatím moc nerozumí, třeba vyplnit *alt* tag
  - *interní odkazy* – vytvořit interní strukturu linků, aby spideri mohli rychle navigovat
- **Generování klíčových slov** je nejdůležitější pro SEO proces, nejlepší klíčová slova dávat na hlavní stránky, méně důležitá na podstránky, důležité je vložit na správná místa
  - Tři kroky:
    - Analýza potenciálního dopadu klíčových slov
    - Výběr vhodných klíčových slov
    - Umístění klíčových slov na správná místa
  - Nutný mix generických klíčových slov a specifických, dohromady tak 4
- obsah stránky má být originální a kvalitní (duplikáty jsou penalizované), aktualizovaný a mít optimalizovaný ne-html dokumenty
- **Struktura webových stránek**
  - design: krátký název, tematická struktura, hierarchie linků
  - sitemap: pomáhá navigovat roboty na stránce
  - robots.txt: informace o tom, které stránky mohou být indexovány a které ne
  - kontaktní údaje a zásada ochrany osobních údajů: dělá to stránky důvěryhodnější
  - .htaccess: soubor s konfigurací pro Apache server
- **Analýza odkazů**
  - **Web graf** – uzly jsou webové stránky a orientované hrany URL adresy nalezené na stránce, které odkazují na jiné stránky (**inlink** – odchozí hrana, **outlink** – příchozí hrana)
  - **Hub** – stránka s hodně outlinky x **Authority** – stránka s hodně inlinky (lze být oboje)
  - **Podpora stránky** – web odkazuje na jiný web
  - **Relevantní stránky** – stránky odkazují jedna na druhou
  - **Ko-citace** – stránka odkazuje na několik stránek
  - **Sociální volba** – na stránku je odkazováno z několika stránek
  - **Tranzitivní podpora** – p1 odkazuje na p3 přes p2
  - **Hodnocení webových stránek** – hodnocení popularity na základě statistiky inlinků
    - **Page rank** = nezáporné reálné číslo vypočítané pouze ze strukturálních informací (query independent, fulltext independent)
    - Hodnotící algoritmy: PageRank, HITS
    - **PageRank** – stránka je důležitá, pokud je na ni odkazováno jinými důležitými stránkami
    - **HITS** – stránka je dobrý hub, pokud odkazuje na dobré autority, a stránka je dobrá autorita, pokud na ni odkazují dobré huby
- **Backlinky** – dobré hodnocení u zajetých stránek činí stránky důvěryhodnější, dávat si pozor na nedůvěryhodné stránky (např pomocí PageRanku)
- **SMO (Social Media Optimization)** – dostat se do podvědomí přes sociální síť
- **Google Analytics** – používané pro sledování celkového pohledu aktivit na stránce
- **Řazení stránek vyhledávačem** – hodnocení relevance stránky na základě uživatelských dat, strukturálního kontextu a behaviorálního kontextu
  - Funkce pro částečné hodnocení – založeny na relevanci obsahu, popularitě/reputaci stránky atd. (layout, click-data statistiky)
  - Koncové řazení
    - One-step ranking – globální hodnotící funkce agregující částečné hodnocení, top-k operátor
    - Multi-step ranking – postupný re-ranking používající částečná hodnocení
- **Neetické SEO** – způsob jak rychle a na krátkou dobu dosáhnout dobrého hodnocení od vyhledávače
  - Používá se pro spammování, často se odhalí a zabane stránku

# Vyhledávání v multimediálních databázích, podobnostní vyhledávání podle obsahu, podobnostní dotazování, agregační operátory, indexování metrické podobnosti, aproximativní vyhledávání

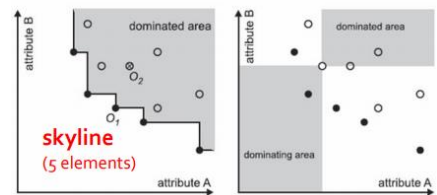
BI-VWM

- **Multimédia** – více než jeden typ digitálního média (obrazový, audio, video obsah)
  - o Jakýkoliv typ nestrukturovaného média (fotky, data ze senzorů, hudba...) – hlavně digitalizované povahy, také lidmi zpracovaná data (text, XML, ilustrace, aplikace, ...)
  - o Kontext – popis/anotace obsahu – sousední prvky, klíčová slova, GPS...
  - o Obsah – pixely, rádiové vlny...
- **Textové vyhledávání v multimédiích**
  - o Např. obrázků na FB – vyhledávání na základě popisu, tagů, komentářů, geolokaci uživateli (a jeho kontextu), albu (sousední prvky), převažující barvy
  - o Výhody – stejná implementace jako u textového vyhledávání
  - o Nevýhody – je subjektivní a nekompletní, může být nahrazeno machine learningem
- **Struktura databází**
  - o Relační – pevná struktura i sémantika
  - o Full-textové – pevná sémantika, volná struktura
  - o Multimediální – volná struktura a sémantika (problém s dotazováním)
- **Podobnostní vyhledávání podle obsahu**
  - o Pokud data nemají strukturu, ani sémantiku – musí se nějak interpretovat (raw obsah se musí dostat do vyšší úrovně informace)
  - o Řeší extrakci příznaků a stanovení podobnostní funkce
  - o **Extrakce příznaků** – transformuje multimediální objekt do strukturovaných deskriptorů
    - Získání deskriptorů:
      - Low-level features (computer vision, kognitivní psychologie)
      - Kombinace low-level features (kompetence prohledávání dat)
    - MPEG7 definuje vizuální, audio a pohybové deskriptory
    - Forma deskriptorů: vektor (histogramy), (un)ordered set (pro množiny a sekvence)
  - o **Podobnostní funkce** – pro porovnávání deskriptorů
    - Má mimikovat sémantiku podobnosti originálních multimediálních objektů
  - o **Modelování podobnostní funkce** – porovnává deskriptory, čím blíže, tím lepší vzdálenost
    - Typy vzdáleností:
      - Vektorové vzdálenosti (Minkowského, kosinové, kvadratické vzdálenosti)
      - Adaptivní vzdálenosti (Hausdorffova vzdálenost)
      - Sekvenční vzdálenosti (editační vzdálenost)
  - o Aplikace – image retrieval pomocí globálních i lokálních příznaků, shape retrieval pomocí časových řad (použití dynamic time warping vzdálenosti nebo longest common subsequence), audio retrieval pomocí waveform spectral information/melodie/score
- **Podobnostní dotazování** – na rozdíl od SQL se nelze dotazovat deskriptory
  - o **Formalizovaný model** – procedura na extrakci příznaků + funkce vzdálenosti (nepodobnosti)
  - o Dotazování příkladem – získá se multimediální objekt, extrahuje se deskriptor  $q$ , použije se podobnostní funkce a výsledky jsou buď z rozsahu, nebo kNN
  - o Typy podobnostních dotazování
    - **Podobnostní řazení** – máme-li objekt  $q$  dotazování z universa deskriptorů  $U$ , získáváme databázi seřazených deskriptorů řazených podle vzdáleností jejich prvků od  $q$
    - **Rozsahové dotazování** – máme poloměr vzdálenosti  $r$  (hranice nepodobnosti), rozsahové dotazování vrací všechny databázové deskriptory, jejichž vzdálenost od  $q$  je menší rovno  $r$

- **K nejblížešších sousedů dotazování** – máme počet požadovaných deskriptorů  $k$ , kNN dotaz vrací databázi deskriptorů nejblížešších ke  $q$

## - Agregáčnı́ operátory

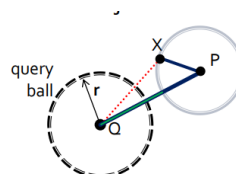
- Databázové operátory = operace nad databázı́ s výsledkem
- Dotaz – příklad jako parametr, očekávaný výsledek je malá podmnožina databáze
- Operátor – neparametrická operace – nad dynamickou databázı́, opakované zpracování neparametrického operátoru nad statickou databázı́ vede ke stejnému výsledku, odpověď dlouhá
- **Similarity join** – spojení deskriptorů databáze A s deskriptory db B
  - Spojení na základě vzdálenosti nebo kNN
  - Self-join:  $A=B$  (na detekci téměř duplicitních prvků)
  - Join založený na vzdálenostním dotazování (poloměr)
  - Join založený na kNN
- **k nejblížešších párů** – vzdálenostní funkce vybere  $k$  párů  $(x,y)$  z  $A \times B$ , které mají nejmenší vzdálenost
  - Vhodné pro dynamické nebo streamované db
- **Skyline operátor** – dotazování nad jedním příkladem nemusı́ být vypovídající, chce to více
  - K nalezení nejlepších prvků (ty, které dominují)
  - **Skyline** – podmnožina prvků, které nejsou dominovány jinými prvky
    - Prvek není **dominován**, jestliže neexistuje jiný prvek, který je lepší ve všech atributech



- Problém – není limitován velikostí, neexistuje řazenı́
- **Top-k operátor** – agregační procedura tvořící finální řazenı́ na základě částečných
  - Máme vybrané atributy a agregační funkci a podle nich vybereme  $k$  nejlepších objektů
  - Top-k operátor vyhodnocuje agregační funkci na částečných „hodnostech“ (ranks) a vracı́  $k$  objektů s nejvyšším agregovaným rankem
  - Dlouhé předzpracování, důležitá normalizace
  - **Faginův algoritmus:**
    - Seřaď data pro každý sloupec sestupně
    - Paralelně procházej jednotlivé sloupce a nalezené hodnoty zapiš do tabulky
    - Jakmile najdeš všechna data pro  $k$  objektů (napříč všemi sloupci), stop a najdi chybějící atributy pro ostatní objekty a doplň do tabulky
    - Použij agregační funkci a vyber  $k$  nejlepších
  - **Threshold algoritmus:**
    - U Fagina zbytečné procházení mnoha objektů
    - Zvol threshold
    - Procházej znovu sloupce a k tomu rovnou pro každý objekt doplň do tabulky i ostatní hodnoty a vyházej ty sloupce, které mají nejmenší číslo. Pokud je nejmenší číslo  $\leq$  thresholdu, konči

## - Indexování metrické podobnosti

- Počıtání podobnostní funkce je drahé a pro velké db nepoužitelné
- Používá se indexování pomocí metrických vzdáleností mezi některými prvky, tím se prostor rozbı́ na několik částı́ a my víme, ve které leží hledaný prvek. Tato část se projde lineárně.
- Musı́ splňovat klasické metrické vlastnosti (symetrie, reflexivita, pozitivní definitnost a trojúhelnı́kovou nerovnost)
- Rozdělenım prostoru vznikne metický index
- **Lower-bound distance** – pro filtrování prvků v db,
  - Využívá trojúhelnı́kové nerovnosti
  - $P$  je pivot – statický objekt z databáze, známe jeho vzdálenosti od všech prvků



The task: check if  $X$  is inside query ball

- we know  $\delta(Q,P)$
- we know  $\delta(P,X)$
- we do not know  $\delta(Q,X)$
- we do not have to compute  $\delta(Q,X)$ , because its lower bound  $\delta(Q,P) - \delta(P,X)$  is larger than  $r$ , so  $X$  surely cannot be in the query ball, so  $X$  is ignored



- **Metric access methods (MAMs)** – pro efektivní vyhledávání v metrických prostorech
  - Používá jen vzdálenosti mezi prvky
  - Využívá lower-bounding filtrování při hledání
  - Architektury:
    - Flat pivot tables (LAESA – vektory vybraných pivotů, AESA – každý prvek je pivot)
      - Mapování dat na prostor pivotů, získáme matici vzdáleností
    - Hierarchické metody (ball-partitioning – M-Tree, hyperplane-partitioning – GNAT)
      - Dělení metrického prostoru a tvoření hierarchie regionů
    - Hashové indexy (D-index, prozradí, jestli je prvek vně/uvnitř kružnice)
      - Podobnostní hashovací funkce – každý objekt zahashován do bucketu, objekty sobě blízké by měly být v tom samém
    - Kombinace předchozích (PM-Tree, M-index) – pivot tabulky s hierarchickými indexy,...
    - Index-free metody

- **Aproximativní vyhledávání**

- Přesné vyhledávání může být časově náročné, vystačíme si s aproximativním, které je mnohem rychlejší
- Lidé taky neumí formulovat dotazy, takže je to leckdy v pohodě
- **Indikátor indexovatelnosti** – na základě průměru a rozptylu řekne, jak dobře strukturovaná data jsou, pokud je vysoký MAMs jsou neúčinná
- Aproximativita
  - a) garance, že je výsledek dotazování do nějaké míry relevantní
  - b) každý objekt ve výsledku je relevantní s nějakou pravděpodobností
  - c) bez garance – experimentální pozorování
- **FastMap**
  - Používá kosovou vzdálenost a pythagorovu větu – projekce objektů na „pivot osu“
  - Algoritmus:
    - Od objektu vybere nejvzdálenější bod a od něj jeho nejvzdálenější
    - Udělá vektor p1, p2 a ořízne body za vektorem na straně, kde není původní objekt
    - Repeat
- **M-strom s aproximativním kNN vyhledáváním**
  - Klasicky má rekurzivně zanořené kruhové regiony (menší a menší) a využívá prioritní frontu a pole kandidátů na nejbližší sousedy
  - 4 heuristiky:
    - Zpomalení vylepšení – pokud se pole mění málo, zastavíme
    - Přibližně správné vyhledávání – vrácení sousedé nejsou o tolik dál, než ti praví, nejbližší
    - Dobrá aproximace zlomků
    - PAC dotazy (přibližně aproximativní dotazy)
- **Permutační indexy** – podobné pivot-table indexům
  - Místo pivot-table (matice vzdáleností) je pro každý databázový objekt permutace pivotů
  - V indexu nejsou uloženy vzdálenosti, ale pořadí pivotů

# Lineární regrese, regularizace pomocí hřebenové regrese

BI-VZD

- **Lineární regrese** – na základě  $p$  příznaků  $X_1 \dots X_p$  predikujeme hodnotu **vysvětlované proměnné**  $Y$ 
  - o Předpokládáme **lineární závislost** vysvětlované proměnné na hodnotách příznaků
    - $Y = w_1 x_1 + \dots + w_p x_p + \varepsilon$
    - $w_1 \dots w_p$  – neznámé koeficienty
    - $\varepsilon$  – náhodná veličina – část  $Y$ , která je nevysvětlitelná pomocí hodnot příznaků – vlivy, které neznáme/nezahrnujeme
- **Model** lineární regrese – nepředpokládáme, že je závislost perfektní, že pro stejné hodnoty  $x_1 \dots x_p$  příznaků  $X_1 \dots X_p$  dostaneme vždy stejnou hodnotu  $Y$

- o  $Y = w_0 + w_1 x_1 + \dots + w_p x_p + \varepsilon,$

- o  $E\varepsilon = 0$

- o **Intercept** – koeficient  $w_0$  – očekávaná výchozí hodnota  $Y$  při nulových příznacích

- o Pokud zavedeme konstantní příznak  $X_0 = x_0 = 1$  + vektorové značení  $x = (x_0 \dots x_p)^T$ ,  $w = (w_0 \dots w_p)^T$ :

$$Y = w^T x + \varepsilon.$$

- $w$  = vektor vah

- **Predikce** v modelu lineární regrese

- o Odhad vektoru vah  $\hat{w}$

- o Predikce  $Y$  v  $x$ :  $\hat{Y} = \hat{w}^T x = \hat{w}_0 + \hat{w}_1 x_1 + \dots + \hat{w}_p x_p.$

- o  $E\varepsilon = 0 \rightarrow EY = w^T x$  – za předpokladu  $E\varepsilon = 0$  plyne, že odhad je bodovým odhadem střední hodnoty  $EY$  v bodě  $x$ , skutečná hodnota  $Y$  v bodě  $x$  má totiž navíc koeficient  $\varepsilon$  a je náhodná veličina

- Měření chybovosti pomocí **ztrátové funkce** – odhad vektoru parametrů

- o Chceme najít vektor vah  $w$  tak, aby byla chyba modelu co nejmenší – odhad  $\hat{w}$

- o Měření chyby – **ztrátová funkce** – aplikace na skutečnou hodnotu  $Y$  a odpovídající predikci  $\hat{Y}$

- Nezáporná funkce  $L: \mathbb{R}^2 \rightarrow \mathbb{R}$

- Spojitá vysvětlovaná veličina  $\rightarrow$  kvadratická ztrátová funkce:  $L(Y, \hat{Y}) = (Y - \hat{Y})^2.$

- **Metoda nejmenších čtverců**

- o Velikost chyby modelu v  $x$  je  $L(Y, \hat{Y})$  – v jakém  $x$  bychom měli  $L(Y, \hat{Y})$  vyhodnocovat a minimalizovat vzhledem k  $w$ ?

- o **Minimalizujeme součet chyb** přes všechny body trénovací množiny  $(x_i, Y_i)$  pro  $i = 1 \dots N$

- Součet chyb pro body pro kvadratickou ztrátovou funkci

$$RSS(w) = \sum_{i=1}^N L(Y_i, w^T x_i) = \sum_{i=1}^N (Y_i - w^T x_i)^2$$

= reziduální součet čtverců

- o Metoda nejmenších čtverců = minimalizace  $RSS(w) \rightarrow$  odhad  $\hat{w}$

- **Extrémní funkce více proměnných**

- o  $w$  je vektor – minimalizace funkce více proměnných  $RSS(w)$

- o **Parciální derivace** funkce  $f(x_1 \dots x_d)$  podle  $x_i$  v bodě  $a = (a_1 \dots a_d)$

= derivace funkce  $g(x_i) = f(a_1 \dots a_{i-1}, x_i, a_{i+1}, \dots, a_d)$ :  $\partial_{x_i} f(a)$  nebo  $\frac{\partial f}{\partial x_i}(a)$

= funkce, která každému bodu, kde je konečná, přiřadí hod. parciální derivace podle  $x_i$  v tomto bodě

- o **Gradient funkce**  $f$  v bodě  $a$  – vektor parciálních derivací pro všechny příznaky  $x_1 \dots x_d$  v bodě  $a$

$$\nabla f(a) = \left( \frac{\partial f}{\partial x_1}(a), \dots, \frac{\partial f}{\partial x_d}(a) \right).$$

- Ukazuje směr maximálního růstu funkce v daném bodě, je kolmý na vrstevnici

- Pokud má funkce lokální extrém a gradient existuje, musí být nulový

- Je to zobrazení, které každému bodu, kde to lze, přiřadí gradient v tom bodě – vektor z  $\mathbb{R}^d$ , jehož složky jsou jednotlivé parciální derivace

- $f$  je funkce d proměnných, která má v bodě  $a \in \mathbb{R}^d$  konečné parciální derivace

- **Hessova matice** funkce  $f$  v bodě  $a$  – matice druhých partiálních derivací podle všech proměnných

$$\mathbf{H}_f(\mathbf{a}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{a}) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_d}(\mathbf{a}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1}(\mathbf{a}) & \dots & \frac{\partial^2 f}{\partial x_d^2}(\mathbf{a}) \end{pmatrix}$$

- Postačující podmínka pro existenci lokálního extrému:

*Bud'  $f: \mathbb{R}^d \rightarrow \mathbb{R}$  funkce  $d$  proměnných a bod  $\mathbf{x}^* \in \mathbb{R}^d$  takový, že  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ .*

- Jestliže  $\mathbf{s}^T \mathbf{H}_f(\mathbf{x}^*) \mathbf{s} > 0$ , pro každé  $\mathbf{s} \in \mathbb{R}^d, \mathbf{s} \neq \mathbf{0}$ ,

*nabývá funkce  $f$  v bodě  $\mathbf{x}^*$  ostrého lokálního minima.*

- Jestliže pro každé  $\mathbf{x}$  z nějakého okolí bodu  $\mathbf{x}^*$

$$\mathbf{s}^T \mathbf{H}_f(\mathbf{x}) \mathbf{s} \geq 0, \text{ pro každé } \mathbf{s} \in \mathbb{R}^d,$$

*nabývá funkce  $f$  v bodě  $\mathbf{x}^*$  neostrého lokálního minima.*

- **Minimalizace RSS( $\mathbf{w}$ )** – metoda nejmenších čtverců = trénování

- Problematika minimalizace funkce více proměnných

1. Vyjádření RSS:

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^N (Y_i - \mathbf{w}^T \mathbf{x}_i)^2 = \|\mathbf{Y} - \mathbf{X}\mathbf{w}\|^2.$$

2. Partiální derivace podle  $w_0 \dots w_p$ :  $\frac{\partial \text{RSS}}{\partial w_j} = \sum_{i=1}^N 2(Y_i - \mathbf{w}^T \mathbf{x}_i)(-x_{i,j})$  – dostaneme gradient

3. Gradient:

$$\nabla \text{RSS} = - \sum_{i=1}^N 2(Y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = -2\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\mathbf{w}).$$

4. Položíme  $\nabla \text{RSS} = \mathbf{0}$  a získáme normální rovnici:

$$\mathbf{X}^T \mathbf{Y} - \mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{0}.$$

5. Hessova matice

$$\mathbf{H}_{\text{RSS}}(\mathbf{w}) = 2\mathbf{X}^T \mathbf{X}$$

- Je vždy pozitivně semi-definitní

→ neostré lokální minimum v jakémkoliv bodě  $\mathbf{w}$ , který řeší normální rovnici  $\mathbf{X}^T \mathbf{Y} - \mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{0}$ .

6. Předpokládejme, že  $\mathbf{X}^T \mathbf{X}$  je **regulární matice** → normální rovnice má **jednoznačné řešení**

$$\hat{\mathbf{w}}_{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y},$$

- bod  $\hat{\mathbf{w}}_{\text{OLS}}$  = bod ostrého lokálního minima, RSS v něm nabývá globálního minima

- OLS = ordinary least squares solution

- **Predikce  $\hat{\mathbf{Y}}$  v bodě  $\mathbf{x}$ :**  $\hat{\mathbf{Y}} = \hat{\mathbf{w}}_{\text{OLS}}^T \mathbf{x} = \mathbf{x}^T \hat{\mathbf{w}}_{\text{OLS}} = \mathbf{x}^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$

- **Metoda gradientního sestupu** – k minimalizaci RSS( $\mathbf{w}$ ), pokud jsou příznaky silně korelované (sloupce matice  $\mathbf{X}$  skoro závislé)

- Pomocí rekurentního  $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \alpha \cdot \nabla \text{RSS}(\mathbf{w}^{(i)}) = \mathbf{w}^{(i)} + \alpha \cdot 2\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\mathbf{w}^{(i)})$

- Postupně konstruujeme posloupnost vektorů, o kterém doufáme, že konverguje ke skutečnému řešení  $\hat{\mathbf{w}}_{\text{OLS}}$

- $\alpha$  = učicí parametr, může záviset na  $i$

- Ukazuje směrem nejvyššího růstu → díky zápornému znaménku děláme kroky ve směru nejvyššího poklesu

- Pro vhodné  $\alpha$  metoda konverguje do globálního optima daného  $\hat{\mathbf{w}}_{\text{OLS}}$ , jinak se ale může i zaseknout v lokálním optimu

- **Vlastnosti lineární regrese**

- Diskriminativní metoda – přímo odhadujeme  $P(Y|X = \mathbf{x})$

- Rezistentní k problémům dimenzionality – je to parametrická metoda (ideálně pro  $p$  příznaků +1 intercept může k určení přesného modelu stačit  $p+1$  bodů trénovací množiny)

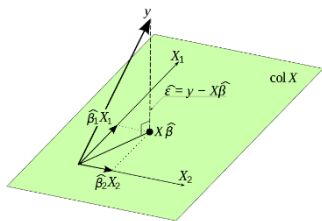
- **Geometrická interpretace**

- Minimalizace  $\text{RSS}(\mathbf{w}) = \|\mathbf{Y} - \mathbf{X}\mathbf{w}\|^2$  je ekvivalentní minimalizaci  $\|\mathbf{Y} - \mathbf{X}\mathbf{w}\| \Rightarrow$  pro optimální  $\mathbf{w}$  je Euklidovská vzdálenost  $\mathbf{Y}$  a  $\mathbf{X}\mathbf{w}$  v  $\mathbb{R}^N$  nejmenší možná

- $i$ -tý sloupec matice  $\mathbf{X} = \mathbf{X}_{\cdot i} \Rightarrow \mathbf{X}_{\cdot i} \mathbf{w} = w_0 \mathbf{x}_{\cdot 0} + w_1 \mathbf{x}_{\cdot 1} + \dots + w_p \mathbf{x}_{\cdot p}$

$\Rightarrow$  vektor  $\mathbf{X}\mathbf{w}$  je **lineární kombinací** sloupců matice  $\mathbf{X}$  s koeficienty  $w_0, \dots, w_p$

$\Rightarrow$  leží v **lineárním podprostoru** prostoru  $\mathbb{R}^N$ , který je **obalem**  $p + 1$  sloupců  $\mathbf{x}_{\cdot 0}, \dots, \mathbf{x}_{\cdot p}$



- Pro různé hodnoty  $w$  pak  $Xw$  celý prostor pokrývá:  $\langle x_0, \dots, x_p \rangle = \{Xw \mid w \in \mathbb{R}^{p+1}\}$
- Chceme-li **minimalizovat vzdálenost**  $Y$  od podprostoru sloupců matice  $X$  (roviny), hledáme **bod  $Xw$  v podprostoru sloupců matice  $X$ , který je k  $Y$  nejbližší**

- $Xw$  je k  $Y$  nejbližší, jestliže je vektor  $Y - Xw$  na ten podprostor kolmý  
 $\Rightarrow$  Je kolmý na všechny vektory  $x_0, \dots, x_p$ , které ho generují:  $(X_i)^T(Y - Xw) = 0$   
 $\Rightarrow X^T(Y - Xw) = 0 \Rightarrow X^TY - X^TXw = 0$  – **normální rovnice**
- Pro jakékoliv řešení  $w$  normální rovnice je  $\|Y - Xw\|$  a tedy i  $RSS(w)$  nejmenší možné  $\Rightarrow$  **globální minimum**

- Normální rovnice má **jednoznačné řešení**, pokud je  $X^TX$  regulární, to je právě tehdy když jsou sloupce matice  $X$  **lineárně nezávislé**

- Problém kolinearity** – sloupce  $X$  jsou „skoro“ lineárně závislé – existuje lineární kombinace sloupců, které dávají téměř nulové vektory, zatímco jiné l.k. vrací mnohem větší vektory – inverze  $X^TX$  teoreticky existuje, ale problematický výpočet – odhad  $\hat{w}_{OLS}$  je citlivý na malé nevhodné změny  $Y \Rightarrow$  velký rozptyl

- Řešení – přigenerování dat, snížení počtu příznaků, změna minimalizované funkce, regularizace (k  $RSS$  přidáme regularizační člen – odstranění kolinearity)

#### - Hřebenová regrese = $L_2$ regularizace

- Řeší problém kolinearity zavedením **penalizačního členu** – úměrný kvadrátu normy vektoru koeficientů bez interceptu ( $w_0$  nepenalizujeme, protože se jedná pouze o posun)  
 $\rightarrow$  minimalizujeme **regularizovaný reziduální součet čtverců** s parametrem  $\lambda \geq 0$

$$RSS_\lambda(w) = \|Y - Xw\|^2 + \lambda \sum_{i=1}^p w_i^2$$

- $\lambda = 0 \rightarrow$  normální  $RSS$
- $\lambda > 0 \rightarrow$  v minimu budeme cílit na vektory  $w$ , které mají co nejmenší složky
- po úpravě získáme reziduální součet čtverců

$$RSS_\lambda(w) = \|Y - Xw\|^2 + \lambda \sum_{i=1}^p w_i^2 = \|Y - Xw\|^2 + \lambda w^T I' w.$$

- $I' =$  diagonální matice s  $x_{0,0} = 0$

- Gradient:

$$\nabla RSS_\lambda(w) = -2X^T(Y - Xw) + 2\lambda I' w$$

- Hessova matice:

$$H_{RSS_\lambda}(w) = 2X^TX + 2\lambda I'.$$

- Normální rovnice:

$$X^TY - X^TXw - \lambda I' w = 0.$$

- Jednoznačným řešením je:

$$\hat{w}_\lambda = (X^TX + \lambda I')^{-1} X^TY$$

#### - Bias-variance tradeoff

- Jelikož  $Y = Xw + \varepsilon$  z trénovací množiny je v důsledku náhodnosti  $\varepsilon$  náhodný vektor, je i  $\hat{Y}_\lambda$  jakožto funkce  $Y$  náhodný vektor

- Očekávaná chyba** – měřená kvadratickou ztrátovou funkcí –  $E L(Y, \hat{Y}) = \sigma^2 + E(\hat{Y} - EY)^2$

- Bayesovská chyba** – první člen  $\sigma^2$  – neodstranitelný, je daný náhodností modelu

- MSE** – druhý člen – střední kvadratická chyba odhadu  $\hat{Y}$  parametru  $EY$

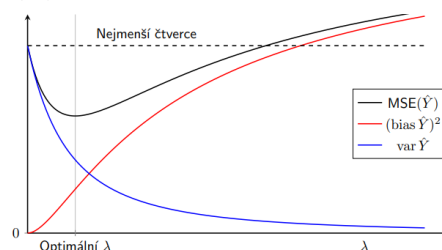
- Po úpravě MSE:  $E L(Y, \hat{Y}) = \sigma^2 + (\text{bias } \hat{Y})^2 + \text{var } \hat{Y} =$  **rozklad očekávané chyby**

- $\text{bias } \hat{Y} = E\hat{Y} - EY$  – vychýlení odhadu

- U hřebenové regrese platí, že s rostoucím  $\lambda$  vychýlení roste a rozptyl klesá = **bias-variance tradeoff**

$$(\text{bias } \hat{Y})^2 \sim \left(1 - \frac{1}{1 + \lambda}\right)^2 \quad \text{a} \quad \text{var } \hat{Y} \sim \left(\frac{1}{1 + \lambda}\right)^2$$

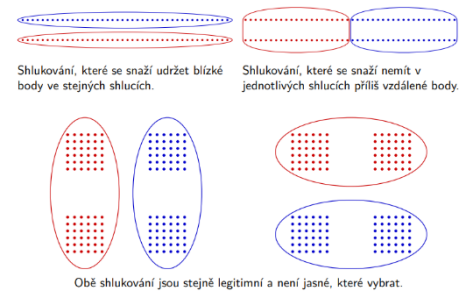
- Hledáme optimální  $\lambda$ , kde je chyba nejmenší
- Minimalizujeme MSE pomocí křížové validace
- Obvykle nejdřív standardizace příznaků



# Metoda nejbližších sousedů, metriky, metody shlukové analýzy (k-nejbližších center, hierarchické shlukování)

BI-VZD

- **Nesupervizované učení** – nemáme veličinu, kterou bychom u trénovacích dat znali a snažili se ji predikovat
  - o Cíl = porozumění struktuře dat na základě jich samotných
- **Supervizované učení** – trénovací data se známými hodnotami vysv. pr.
- **Shlukování** – clustering – třídění dat do skupin – shluků tak, že:
  - o Blízké body budou ve stejném shluku
  - o Vzdálené body budou v různých shlucích
- **Vzdálenost** – metrika na množině  $X$ : funkce  $d: X \times X \rightarrow [0, +\infty)$  taková, že  $\forall x, y, z \in X$ :
  1. **Positivní definitnost**:  $d(x, y) \geq 0 \wedge d(x, y) = 0 \Leftrightarrow x = y$
  2. **Symetrie**:  $d(x, y) = d(y, x)$
  3. **Trojúhelníková nerovnost**:  $d(x, y) \leq d(x, z) + d(z, y)$
  - o **Minkowského k-metriky**:
    - $k=2$ : Eukleidovská vzdálenost ( $L_2$ ):  $d_2(x, y) = \sqrt{\sum_{i=1}^p (x_i - y_i)^2}$
    - $k=1$ : Manhattanská vzdálenost ( $L_1$ ):  $d_1(x, y) = \sum_{i=1}^p |x_i - y_i|$
  - o **Čebyševova vzdálenost** ( $L_\infty$ ):  $d_\infty(x, y) = \max_i |x_i - y_i|$
  - o **Levenshteinova vzdálenost**: minimální počet změn z jednoho řetězce na druhý
- **Vstupy shlukování**
  - o Metrický prostor  $X$  se vzdáleností  $d$
  - o Množina dat  $D \subset X$
  - o (obvykle) požadovaný počet shluků  $k$
- **Výstupy shlukování**
  - o Rozklad množiny  $D$  na jednotlivé shluky  $C = (c_1 \dots c_k), c_i \subset D \forall i, \forall i \neq j: c_i \cap c_j = \emptyset, D = \bigcup_{i=1}^k c_i$
  - o Bod  $x \in D$  je tedy v  $i$ -tém shluku, jestliže  $x \in c_i$
  - o Grafický výstup pro hierarchické shlukování – dendrogram
- **kNN** – metoda nejbližších sousedů
  - o chceme predikovat hodnotu vysvětlované proměnné pro datový bod  $x \in \mathbb{R}^p$
  - o v trénovacích datech najdeme  $k$  bodů, které mají od  $x$  nejmenší vzdálenost
  - o predikce založena na známých hodnotách vysvětlované pro těchto  $k$  bodů – supervizované učení
  - o regrese  $\rightarrow$  průměr z hodnot  $x$  klasifikace  $\rightarrow$  nejčastější hodnota
  - o trénovací data jsou sama o sobě naučeným modelem  $\rightarrow$  učení vlastně neprobíhá
  - o výpočetně náročná predikce = hledání nejbližších sousedů (pomáhá indexace – vyhledávací strom)
  - o **Hyperparametry**:
    - $K$  – počet nejbližších sousedů – zvýšením  $k$  zabraňování přeučení
    - **Míra vzdálenosti** – nejobvyklejší Minkowského  $k$ -metriky, lze použít i vzdálenosti, které se v každé dimenzi chovají jinak
    - **Váhy nejbližších sousedů** – určují „sílu hlasu“ při predikci
      - Obvykle klesají se vzdáleností:  $w_i = \frac{1}{d(x, x_i)}$
  - o **Normalizace dat** – protože kNN je citlivá na typy jednotlivých příznaků
    - Normalizace každého příznaku do intervalu  $[0, 1]$  – zabraňování extrémním úletům způsobeným pestrými škálami příznaků a jejich významů
  - o Nominální příznaky (např. číslo městské části) – řeší modifikace metriky nebo one-hot encoding
  - o **Prokletí dimenzionality** – s přibývajícím počtem dimenzí (příznaků) exponenciálně roste vel. Prostoru
    - Řídnutí a vzájemné vzdalování se dat
    - Zmenšení rozdílů mezi vzdálenými a blízkými body



- **Hierarchické shlukování** – hladový aglomerativní algoritmus

- o N ... počet prvků množiny dat D
  1. Začátek: každý bod = 1 shluk (N shluků)
  2. Iterujeme:
    - a. Najdeme 2 shluky, které jsou si nejbliž
    - b. Spojíme je do nového shluku
 → po N-1 opakováních 1 shluk se všemi body
- o Třeba určit **zastavovací kritérium** – nejčastěji počet shluků k, příp. Lineární hodnota vzdálenosti shluků
- o Měření vzdálenosti shluků

▪ Vstup = vzdálenost dvojice bodů  $d(x,y)$

▪ **Metoda nejbližšího souseda** (single linkage) – generuje dlouhé řetězce:

$$D(A, B) = \min_{x \in A, y \in B} d(x, y)$$

▪ **Metoda nejvzdálenějšího souseda** (complete linkage) – generuje kompaktní shluky:

$$D(A, B) = \max_{x \in A, y \in B} d(x, y)$$

▪ **Párová vzdálenost** (average linkage) – kompromis předchozích dvou

$$D(A, B) = \frac{1}{|A||B|} \sum_{x \in A, y \in B} d(x, y)$$

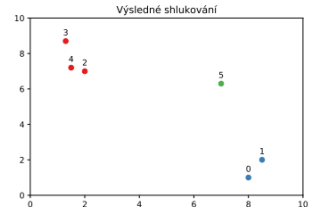
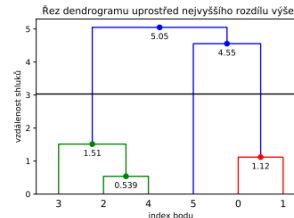
▪ **Wardova metoda** – v  $\mathbb{R}^p$  – účinná, minimalizuje nárůst vnitřního rozptylu

$$D(A, B) = \sum_{x \in A \cup B} \|x - \bar{x}_{A \cup B}\|^2 - \sum_{x \in A} \|x - \bar{x}_A\|^2 - \sum_{x \in B} \|x - \bar{x}_B\|^2$$

kde  $\bar{x}_A = \frac{1}{|A|} \sum_{x \in A} x$  je geometrický střed množiny A a  $\bar{x}_B$ ,  $\bar{x}_{A \cup B}$  analogicky.

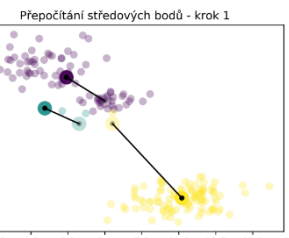
o **Dendrogram** – vizualizace hierarchického shluk. – strom, kde vrcholy = shluky, které při běhu vznikly

- Listy = počáteční jednoprvkové shluky, kořen = finální shluk všech bodů
- Na jeho základě můžeme zvolit finální počet shluků, můžeme měnit počet shluků
- Časově náročné pro velké datové soubory



- **k-Means** – k nejbližších center

- o shlukování jako **optimalizační úloha** – definována **účelová funkce** – ohodnocuje daný rozklad množiny na jednotlivé shluky, cílem je nalézt rozklad, který **účelovou funkci minimalizuje**
- o Algoritmus:
  1. počáteční rozmístění k středových bodů  $\mu_1 \dots \mu_k$
  2. Iterujeme:
    - a. Roztřídíme body do shluků: Pro každý středový bod určíme jemu odpovídající shluk jako podmnožinu bodů, ke kterým je blíže, než všechny ostatní středové body

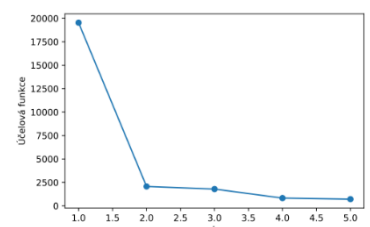


b. Spočítáme nové středové body jako geometrické středy těchto shluků

$$C_i = \{x \in D \mid i = \arg \min_j \|x - \mu_j\|\}$$

$$\mu_i \leftarrow \frac{1}{|C_i|} \sum_{x \in C_i} x$$

- o V každé iteraci má účelová funkce stejnou nebo nižší hodnotu
- o Zastavení algoritmu = dostatečně malá změna hodnoty účelové funkce mezi iteracemi
- o Výsledek významně závisí na inicializační části – obvykle náhodné generování středových bodů
- o Algoritmus následně opakovaně spouštěn – finální výsledek = výsledek běhu s nejnižší ú. f.
- o Problematika volby k – nezbytné stanovit k dopředu
  - Optimální k = hodnota, pro kterou se mění pokles ú. f. Z hodně prudkého na méně prudký – hledání **lokte** – nehodí se vždy (další metoda např. silhouette)
- o Dokáže konvergovat k lokálnímu optimu, globální je NP-těžké
- o rychlejší než hierarchické shlukování, ale nutná volba k předem



# Naivní Bayesův klasifikátor, modely podmíněných pravděpodobností

BI-VZD

## - Klasifikace pomocí podmíněné pravděpodobnosti

- Klasifikační úloha, p diskretních příznaků, chceme predikovat diskretní vysvětlovanou proměnnou
- Příznaky  $\rightarrow$  náhodný vektor  $\mathbf{X} = (X_1 \dots X_p)^T$  s hodnotami  $\mathbf{x}$
- Postup:

1. Odhad  $P(Y = y|\mathbf{X} = \mathbf{x}) \forall \mathbf{x} \in \mathcal{X} \text{ a } y \in \mathcal{Y}$  na základě trénovací množiny
2. Využijeme pravděpodobnost k predikci  $Y$ :

$$\hat{Y} = \arg \max_{y \in \mathcal{Y}} P(Y = y|\mathbf{X} = \mathbf{x})$$

= MAP odhad (maximum a posteriori)

$\hat{Y}$  = hodnota, která je při  $\mathbf{x}$  nejpravděpodobnější

## - Využití Bayesovy věty – jak odhadnout $P(Y = y|\mathbf{X} = \mathbf{x})$

1. Odhadneme  $P(\mathbf{X} = \mathbf{x}|Y = y)$
2. Podle Bayesovy věty:

$$P(Y = y|\mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{X} = \mathbf{x}|Y = y) P(Y = y)}{P(\mathbf{X} = \mathbf{x})}$$

$$P(\mathbf{X} = \mathbf{x}) = \sum_{y \in \mathcal{Y}} P(\mathbf{X} = \mathbf{x}|Y = y) \cdot P(Y = y)$$

- potřebujeme odhad  $P(Y=y)$  – triviální

- Pokud chceme argument maxima, můžeme zahodit  $P(\mathbf{X}=\mathbf{x})$  – je pro všechny  $y$  stejný:

$$P(Y = y|\mathbf{X} = \mathbf{x}) \propto P(\mathbf{X} = \mathbf{x}|Y = y) P(Y = y)$$

$\propto$  „rovnost, až na násobek konstantní vzhledem k  $y$ “

$\rightarrow$  predikce:

$$\hat{Y} = \arg \max_{y \in \mathcal{Y}} P(\mathbf{X} = \mathbf{x}|Y = y) P(Y = y)$$

- Potřebujeme odhadnout  $P(\mathbf{X} = \mathbf{x}|Y = y)$  – pomocí Naivního Bayesova klasifikátoru

## - Naivní Bayesův klasifikátor

- Předpoklad: za podmínky  $Y=y$  jsou všechny příznaky nezávislé  
 $\rightarrow \forall y \in \mathcal{Y} \text{ a } \mathbf{x} = (x_1 \dots x_p)^T \in \mathcal{X}: P(\mathbf{X} = \mathbf{x}|Y = y) = P(X_1 = x_1|Y = y) \cdot \dots \cdot P(X_n = x_n|Y = y)$
- Naivita = pro fixní hodnotu vysvětlované proměnné předpokládáme, že jsou příznaky nezávislé
- MAP odhad: 
$$\hat{Y} = \arg \max_{y \in \mathcal{Y}} \prod_{i=1}^p P(X_i = x_i|Y = y) P(Y = y).$$
- Díky rozkladu  $P(\mathbf{X} = \mathbf{x}|Y = y)$  na součin marginálních podmíněných pstí  $P(X_i = x_i|Y = y)$  jsou příznaky **separovány**  
 $\rightarrow$  odhad podmíněné pravděpodobnosti každého příznaku probíhá nezávisle na ostatních  
 $\rightarrow$  rezistence proti problémům s dimenzionalitou (k OK odhadu stačí málo dat i s nárůstkem příznaků)
- Obvykle nepřesný předpoklad nezávislosti  $\rightarrow$  špatný odhad sdružené pod. psti  $P(\mathbf{X} = \mathbf{x}|Y = y)$ 
  - Nevadí, MAP bude OK, pokud má skutečná hodnota  $y$  vyšší odhadnutou pravděpodobnost, než ostatní hodnoty – často ano

## - Modely podmíněných pravděpodobností – problematika odhadu $P(\mathbf{X} = \mathbf{x}|Y = y)$ , kde $\mathbf{x}$ je jeden z příznaků

### 1. Bernoulliho rozdělení

- Nejjednodušší situace -  $\mathbf{x} \in \{0, 1\}$   
 $\rightarrow$  vhodné Ber. Rozdělení s parametrem  $p_y = P(X = 1|Y = y)$ ,  $P(X|Y = y) \sim \text{Be}(p_y)$   
 $\rightarrow$  odhad 
$$\hat{p}_y = \frac{N_{1,y}}{N_{1,y} + N_{0,y}}$$
  - $N_{1,y}$ ...počet dat pro  $x = 1$  a  $Y = y$
- Jedná se o maximálně věrohodný odhad – MLE param. Be. Rozdělení



- Nevýhoda – v moc malé/velké  $p_y$  nemusí být v trénovací množině pro  $Y=y$  zastoupeny obě hodnoty  $x_i \rightarrow$  kolaps -  $\hat{p}_y = 0$  (nebo 1)  
 $\rightarrow$  nutné vhodné počáteční rozdělení
  - apriorní rozdělení odrážející naši expertní znalost, kterou hodláme na základě napozorovaných dat zpřesňovat
  - řešení = add-one smoothing (Laplace's rule of succession) – netrpí kolapsem

$$\hat{p}_y = \frac{N_{1,y} + 1}{N_{1,y} + N_{0,y} + 2}.$$

## 2. Kategorické rozdělení

- $X$  nabývá  $k$  různých hodnot  $c_1 \dots c_k$  a podmíněnými psmi  $p_{1,y}, \dots, p_{k,y}$  -  $P(X = c_j | Y = y) = p_{j,y}$
- Opět je to MLE a trpí kolapsem, proto je získán robustnější MAP odhad  $k$ -rozměrného parametru  $\hat{p}_y = (\hat{p}_{1,y}, \dots, \hat{p}_{k,y})^T$

$$\hat{p}_{j,y} = \frac{N_{j,y} + 1}{N_{1,y} + \dots + N_{k,y} + k}.$$

## 3. Spojité rozdělení

- $X$  je spojitá náhodná veličina  $\rightarrow$  hustota  $f_{X|y}(x) = P(X \leq x | Y = y)$ . místo podmíněné psmi
- MAP odhad:

$$\hat{Y} = \arg \max_{y \in \mathcal{Y}} \prod_{i=1}^{\ell} P(X_i = x_i | Y = y) \prod_{i=\ell+1}^p f_{X_i|y}(x_i) P(Y = y),$$

- $x_1 \dots x_\ell$  ... diskrétní příznaky a  $x_{\ell+1} \dots x_p$  ... spojitě příznaky

## 4. Gaussovo rozdělení

- Častým modelem podmíněného rozdělení je ve spojitém případě normální rozdělení  $N(\mu_Y, \sigma^2_Y)$  se střední hodnotou  $\mu_Y$  a rozptylem  $\sigma^2_Y$
- MLE odhady:

$$\hat{\mu}_y = \frac{1}{N_y} \sum_i^{N_y} x_i \quad \text{a} \quad \hat{\sigma}_y^2 = \frac{1}{N_y} \sum_i^{N_y} (x_i - \hat{\mu}_y)^2$$

- Kde  $x_1 \dots x_{N_y}$  jsou hodnoty příznaku  $X$ , pro které  $Y=y$

- **Generativní přístup k predikci** – vytváříme model sdružené pravděpodobnosti

$$P(X = x, Y = y) = P(X = x | Y = y) \cdot P(Y = y)$$

- Např. Naivní Bayes
- Model sdruž. psmi představuje úplnou informaci o rozdělení, ze kterého byla data „generována“
- Může být použit pro generování nových pozorování

- **Diskriminativní přístup k predikci** – odhadujeme na základě trénovacího datasetu podmíněnou pravděpodobnost  $P(Y = y | X = x)$  přímo, bez modelu pro sdruženou pravděpodobnost

- Např. logistická regrese nebo neuronové sítě

- **Aplikace Bayesova klasifikátoru** – klasifikace textů – bag-of-words model



# Rozhodovací stromy: algoritmus konstrukce stromů, hyperparametry. Náhodné lesy

BI-VZD

- **Supervizované učení** – zjišťujeme, jak vysvětlovanou proměnnou ovlivňují příznaky
  - o Funkční vztah „aby co nejvíce platilo“
- **Rozhodovací strom** – strom, který má ve svých vnitřních uzlech pravidla a v listech předpovězenou hodnotu
  - o klasifikační (výsledná předpovězená hodnota je ta s největším zastoupením v listu) x regresní (průměr hodnot)
  - o velmi často binární
  - o pokud listy stromu obsahují všechny kombinace hodnot příznaků, není to strom, ale index (strom by se totiž nemýlil, nereálné pro velká data)
  - o kvalita se měří rozdělením dat na trénovací a testovací množinu a pak podle nějaké metriky se měří chyba (např. RMSE)
- **Konstrukce stromu**
  - o Vstup – N-řádková tabulka s hodnotami pro binární vysvětlovanou proměnnou, p binárních příznaků
  - o Cíl – strom hloubky k, který správně přiřadí hodnotu Y co nejvíce řádkům z tabulky
  - o Konstrukce optimálního stromu je NP-úplný problém
- **ID3 algoritmus** – suboptimální hladový algoritmus
  - o 1 (nepoužitý) příznak dělí data na 2 části tak, že vzniklé rozdělení maximalizuje vybrané kritérium
  - o Množina dat rozdělena → 2 části → na každou zvlášť aplikován stejný postup → výsledkem další 2 příznaky → použité jako kritérium → rozdělení na 4 podmnožiny dat
    - Dokud nenastane zastavovací kritérium (max. hloubka stromu, nedostatek dat, ...)
- **Stromy pro klasifikaci**
  - o **Entropie** – funkce míry neuspořádanosti
    - Pro rozdělené množiny příznakem při konstrukci stromu
    - Maximální, pokud počet 1 a 0 je stejný (všechny hodnoty mají stejné zastoupení)
    - Nulová (minimální), pokud jsou v množině zastoupeny pouze stejné hodnoty
    - p ... poměr počtu hodnot v množině
$$H(\mathcal{D}) = - \sum_{i=0}^{k-1} p_i \log p_i.$$
  - o **Gini index** – použit místo entropie pro množinu D s k různými hodnotami
    - Míra toho, že nově přidáný prvek bude špatně klasifikován
$$GI(\mathcal{D}) = 1 - \sum_{i=0}^{k-1} p_i^2 = \sum_{i=0}^{k-1} p_i(1 - p_i).$$
  - o **Informační zisk** – pro výběr příznaku, který rozdělením dat nejvíce sníží neuspořádanost
    - Entropie mínus vážený součet entropií podstromů vzniklých rozdělení
    - t = počet prvků v podstromu / celkový počet prvků
    - lze použít Gini index i Entropie
    - lepší je vyšší
    - pokud je hodnota IZ pro různé příznaky vyjde stejná (a nejvyšší), můžeme si libovolně vybrat
    - v případě spojitých příznaků se používají pravidla porovnání  $X < x_i$  (např. pro každé 10. x, pokud je hodnot až moc)
$$IG(\mathcal{D}, X_i) = H(\mathcal{D}) - t_0 H(\mathcal{D}_0) - t_1 H(\mathcal{D}_1)$$
  - o **Klasifikační přesnost** – počet správně klas. dat/počet všech dat
- **Stromy pro regresi** – spojitá vysvětlovaná proměnná – nelze využít entropie/gini index
  - o **MSE** – „mean squared error“ – odchylka od střední hodnoty (skoro výběrový rozptyl)

$$MSE(Y) = \frac{1}{N} \sum_{j=0}^{N-1} (Y_i - \hat{Y})^2$$

- Chceme, aby hodnoty vysvětlované proměnné v rámci listu byly co nejblíže střední hodnoty

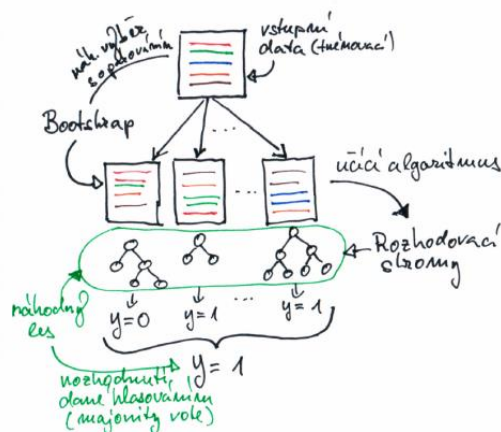
- Algoritmus **CART** – hladový algoritmus, vrcholy stromu se volí tak, aby se **minimalizovalo MSE**

$$MSE(D) - t_L MSE(D_L) - t_R MSE(D_R)$$

- Místo MSE – MAE – místo čtverce vzdálenosti absolutní hodnota

$$MAE(Y) = \frac{1}{N} \sum_{j=0}^{N-1} |Y_i - \hat{Y}|$$

- **Hyperparametry** rozhodovacího stromu – to, co určuje tvar nebo komplexnost modelu
  - Maximální hloubka stromu, minimální počet dat v množině, minimální nutná hodnota informačního zisku, gini/entropy kritérium, ...
  - Ladí se např. křížovou validací a měří se chyba (RMSE, MSE, ...), data se dělí na testovací a trénovací množiny (příp. Validační)
  - Počet náhodně vybraných příznaků, ze kterých si hladový algoritmus vybere, podle kterého se bude větvit
- Výhody rozhodovacích stromů – nenáročnost na přípravu dat, jednoduché, rychlé učení, dobře interpretovatelné, poradí si s kategoriálními i spojitými příznaky, s chybějícími hodnotami
- Nevýhody rozhodovacích stromů – nerobustní (malá změna dat – velká změna struktury stromu), většinou pouze binární stromy, optimalita NP-úplná, snadné přeučení
- **Náhodné lesy**
  - **Ensamble metoda** – místo 1 modelu více modelů a následní kombinace predikcí do fin. Rozhodnutí
    - Bagging (náhodné lesy) x boosting (AdaBoost)
      - Boosting – vážená data, strom se učí, aby správně predikoval hlavně datové body s vyšší vahou, Ada Boost = zvyšování váhy bodům, které v předchozím stromu byly klasifikovány špatně
  - Bagging = bootstrap aggregating
  - **Bootstrap** – výběr s opakováním – zajišťuje důležitou pestrost modelů
    - Chci dataset velikosti 5 – náhodně 5x vyberu řádek z tabulky – můžou se opakovat
  - **Náhodný les pro klasifikaci**
    - Ze vstupního D n datasetů pomocí bootstrapu
    - Na každém  $D_i$  naučení rozhodovacího stromu  $T_1...T_n$
    - Každý datový bod (řádek z D) proženeme stromy  $T_1...T_n$  a uložíme jejich rozhodnutí  $Y_1...Y_n$
    - $T_1...T_n$  = náhodný les, finální rozhodnutí  $Y$  = většinové rozhodnutí stromů
  - Náhodný les pro regresi – analogicky, predikce = průměr predikcí stromů
  - **Hyperparametry:**
    - Počet stromů v náhodném lese
    - Maximální hloubka stromů v lese
    - Počet příznaků, ze kterých si hladový algoritmus vybírá ten, podle kterého se bude větvit (+ pestrost modelů)
  - Výhody – robustní, odolné vůči přeučení
  - Nevýhody – ztrácí jednoduchost a snadnou interpretovatelnost

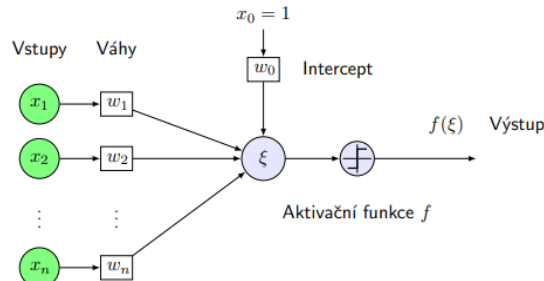


# Neuronové sítě, struktura perceptronové sítě, výpočet výstupu neuronu, učení

## perceptronové sítě

BI-VZD + BI-ZNS

- **Neuronové sítě** – inspirovány neurony živých organismů
  - o Schopnost extrakce a reprezentace závislostí na nezřejmých datech
  - o Neuron = stavební kámen, jednoduchá výpočetní jednotka
  - o Přijímá několik vstupů, produkuje jeden výstup
  - o Schopnost řešit nelineární úlohy, učit se, zevšeobecňovat
- **Jednovrstvý perceptron** – nejjednodušší model NS určený ke klasifikaci – 1 umělý neuron



- o Výstup neuronu → aplikace nelineární aktivační funkce  $f$  na hodnotu vnitřního potenciálu  $\xi$  daného součtem vstupů  $x_1 \dots x_n$  pronásobených vahami  $w_1 \dots w_n$  a interceptu  $w_0$  (bias)
- o **Vnitřní potenciál:**

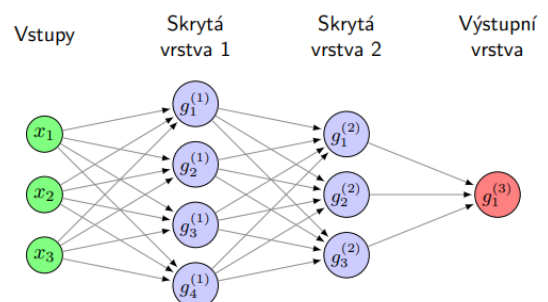
$$\xi = w_0 + \sum_{i=1}^n w_i x_i = \mathbf{w}^T \mathbf{x} + w_0$$
- o Výstup perceptronu:
 
$$\hat{Y} = f(\xi) = f(\mathbf{w}^T \mathbf{x} + w_0)$$
  - $f$  ... aktivační funkce – u perceptronu skoková funkce  $f(\xi) = \begin{cases} 1 & \text{když } \xi \geq 0, \\ 0 & \text{když } \xi < 0. \end{cases}$
  - tento výpočet = dopředný chod
  - neuron je aktivován,  $f(\xi) = 1$ , pokud  $\sum_{i=1}^n w_i x_i \geq -w_0$ 
    - $-w_0$  = prahová hodnota
- o Prostor  $\mathbb{R}^n$  vstupů  $(x_1 \dots x_n)^T$  rozdělen na 2 lineárně separované poloprostory, kdy v jednom z nich je neuron aktivní a v druhém ne. Váhy  $(w_0 \dots w_n)^T$  určí nadrovinu, která tyto 2 poloprostory separuje
- o Dopředný a zpětný chod
  - **Dopředný chod** – pro zadané váhy můžeme na základě hodnot vstupu  $x$  spočítat výstup
  - **Zpětný chod** – na základě hodnot získaných v dopředném chodu určíme chybu predikce a na jejím základě provedeme inkrementální update vah
 
$$\text{error} = Y - \hat{Y}$$

$$w_i \leftarrow w_i + \text{error} \cdot x_i, \quad i = 1, \dots, n$$

$$w_0 \leftarrow w_0 + \text{error}$$
    - $\hat{Y} = f(\mathbf{w}^T \mathbf{x} + w_0)$  ... predikce v bodě  $x$  trénovací množiny
    - $Y$  ... skutečná hodnota

### - Vícevrstvá neuronová síť

- o **Vícevrstvý perceptron** – síť se skládá z vrstev, které jsou propojené tak, že výstupy neuronů z jedné vrstvy tvoří vstupy neuronů do další vrstvy
- o **Skryté vrstvy** – všechny vrstvy kromě té výstupní, jsou příznaky pro další vrstvy
- o Preference hlubších sítí – sofistikovanější a zajímavější příznaky
- o Hluboké učení – síť s více než 3-5 skrytými vrstvami
- o Zvýšení počtu vrstev zvyšuje flexibilitu sítě, přidání neuronů snižuje výkonnost
- o Síť s jednou skrytou vrstvou dokáže na výstupu s libovolnou přesností aproximovat jakoukoliv spojitou funkci v  $\mathbb{R}^n$



## - Učení vícevrstvých sítí

- **Dopředný chod** – uvažujme  $l$  vrstvou neuronovou síť a označme  $n_1, \dots, n_l$  počty neuronů v jednotlivých vrstvách a počet vstupních proměnných jako  $n_0$ . Celá neuronová síť při dopředném chodu je reprezentována funkcí  $g: \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$ , která vznikne složením jedn. vrstev  $g = g^{(l)} \circ g^{(l-1)} \circ \dots \circ g^{(1)}$
- **Zpětný chod** – algoritmus zpětného šíření chyby – back-propagation
  - NS jako funkce parametrů musí být skoro všude diferencovatelná
  - Použití gradientního sestupu
- Docílení diferencovatelnosti – volba vhodné aktivační funkce pro skryté vrstvy

- **RELU** – oříznutá lineární funkce

$$f(\xi) = \max(0, \xi) = \begin{cases} \xi & \xi \geq 0 \\ 0 & \xi < 0 \end{cases}$$

- **Hyperbolický tangens**

$$f(\xi) = \tanh(\xi) = \frac{e^\xi - e^{-\xi}}{e^\xi + e^{-\xi}}$$

- **Aktivační funkce pro výstupní vrstvy** – převod napočítaných hodnot na hodnoty použitelné k predikci

- Regresní úloha – predikce spojité proměnné
  - Ve výstupní vrstvě 1 neuron bez aktivační funkce -  $f(\xi) = \xi$
- Binární klasifikace
  - Ve výstupní vrstvě 1 neuron
  - Aktivační funkce = **sigmoida** – logistická funkce

$$f(\xi) = \frac{1}{1 + e^{-\xi}} = \frac{e^\xi}{1 + e^\xi}$$

- Hodnota = pravděpodobnost příslušnosti ke třídě 1:  $\tilde{P}(Y = 1|X = x)$
- Klasifikace do  $c$  tříd
  - Ve výstupní vrstvě  $c$  neuronů
  - Aktivační funkce = **softmax**

$$f_i(\xi) = \frac{e^{\xi_i}}{e^{\xi_1} + \dots + e^{\xi_c}}$$

- $\xi = (\xi_1 \dots \xi_c)^T$  ... vektor vnitřních potenciálů  $c$  neuronů

- $f_i(\xi)$  ... aktivační funkce  $i$ -tého neuronu

- Výstup = pravděpodobnost příslušnosti ke třídě  $i$

→ predikce  $\hat{Y}$  v  $x$ :  $\hat{Y} = \operatorname{argmax}_{i \in 1 \dots c} \hat{P}(Y = i|X = x)$

- **Učení neuronových sítí** – minimalizujeme chybu predikce – měření pomocí průměrné hodnoty ztrátové funkce na trénovací množině

- **Ztrátová funkce** – měří, jak dobře model predikuje konkrétní hodnotu z trénovací množiny

- Regresní úloha – kvadratická ztrátová funkce

$$L(Y, \hat{Y}) = (Y - \hat{Y})^2$$

- Binární klasifikace – binární relativní entropie

$$\hat{p} = \hat{P}(Y = 1|X = x)$$

$$L(Y, \hat{P}) = -Y \log \hat{p} - (1 - Y) \log (1 - \hat{p})$$

- Klasifikace pro  $c$  tříd – kategorická relativní entropie

$$\hat{p}_i = \hat{P}(Y = i|X = x)$$

$$\hat{p} = (\hat{p}_1 \dots \hat{p}_c)^T$$

$$L(Y, \hat{p}) = - \sum_{j=1}^c \mathbb{1}_{Y=j} \log \hat{p}_j = - \log \hat{p}_Y$$

$$\mathbb{1}_{Y=j} = 1 \text{ když } Y = j \text{ a } \mathbb{1}_{Y=j} = 0 \text{ jinak.}$$

- **Učení gradientním sestupem**

- Minimalizace chyby predikce měřené pomocí průměrné hodnoty ztrátové funkce  $L$  vzhledem k parametrům sítě  $w$ , které se vyskytují ve funkci  $g$

$$J(w) = \frac{1}{N} \sum_{i=1}^N L(Y_i, g(x_i; w))$$

- Minimalizace gradientním sestupem

- Výpočet gradientu:

Pro  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $g: \mathbb{R} \rightarrow \mathbb{R}^n$ , kde  $g = (g_1, \dots, g_n)^T$  tak platí

$$\frac{\partial f \circ g}{\partial x}(x) = \sum_{i=1}^n \frac{\partial f}{\partial x_i}(g(x)) \frac{\partial g_i}{\partial x}(x).$$

→ pronásobování a sčítání parciálních derivací vrstev ve směru od výstupní vrstvy směrem ke vstupní = **zpětné šíření**

- Při standardním gradientním sestupu se počítá chyba přes celou trénovací množinu, což může představovat problém z pohledu paměti u velkých datasetů a mnoha parametrů, proto existují různá trénovací schémata

- Dávkové učení = batch training** – počítá chybu přes celou trén. množinu, pak až krok proti směru gradientu

- Neuronová síť s parametry  $(w_0 \dots w_n)^T$  a trénovací data  $((Y_1, x_1), \dots, (Y_N, x_N))$

- Postup učení:

1. Inicializace  $w$  jako náhodná malá čísla

2. Opakujeme, dokud nesplní kritéria k zastavení:

- i. Pro trénovací množinu spočteme průměrnou chybu predikce

$$J(w) = \frac{1}{N} \sum_{i=1}^N L(Y_i, g(x_i; w))$$

- ii. Spočteme gradient

$$\nabla_w J = \left( \frac{\partial J}{\partial w_1}, \dots, \frac{\partial J}{\partial w_m} \right)^T$$

- iii. Přepočítání vah

$$w \leftarrow w - \alpha \nabla_w J$$

- Trénink v **minibatches** – výběr z několika bodů (přes ně výpočet chyby) a poté update vah

- Online training** – update pro každý bod zvlášť

- Konvergence** – gradientní sestup mívá v prostoru parametrů do míst, kde je gradient nulový

- Jednovrstvá síť → globální minimum

- Vícevrstvá síť → můžeme uváznout v lokálním minimu/sedlovém bodě (řídce rozmístěny, jejich hodnota je blízko globálnímu minimu)

- Regularizace** – kvůli problému s přeučováním u hlubokých NS

- Přidání členu penalizujícího velikosti vah k účelové funkci

- Dropout – náhodné vynulování některých neuronů

- Nedostatek dat → Data augmentation – tvorba více trénovacích dat s poruchami (otočení, škálování)

- Využití předtrénované sítě

- Příklady architektur:

- Autoenkodér** – enkodér + dekodér (pro kompresi dat → redukce dimenzionality)

- Enkodér – tvorba abstraktních příznaků

- Dekodér – návrat do pův. prostoru příznaků a replikace vstupních dat

- Detekce odlehlých hodnot (velká rekonstrukční chyba)

- Konvoluční NS** – neurony spolupracují lokálně → z toho skládání celku – sady vrstev, které počítají vhodné diskrétní konvoluce vstupů, využití okolních bodů pro rozpoznávání (např. klasifikace obrazu)

- Rekurentní NS** – zpracování posloupnosti dat (analýza textu, ...)

- při výpočtu výstupu využití výstupů předchozího časového kroku – paměťové buňky

- skryté stavy → podchycení dlouhých časových závislostí

# Logistická regrese

BI-VZD

- Určena pro klasifikaci, zde omezení na **binární klasifikaci**
- Rozhodnutí se jako u lineární regrese konstruuje pomocí lineární kombinace příznaků  $w_0 + w_1x_1 + \dots + w_px_p$
- Z diskrétního problému dělá problém **spojitý** – namísto hodnoty vysvětlované proměnné predikujeme pravděpodobnost, že Y má hodnotu 1 (tzn.  $P(Y = 1)$  z intervalu  $[0, 1]$ ):

$$P(Y = 1 | \mathbf{x}, \mathbf{w})$$

- o je závislá na hodnotách příznaků  $\mathbf{x}=(x_1 \dots x_p)$  a koeficientů  $\mathbf{w}=(w_1 \dots w_p)$
- Platí součet pravděpodobností  $P(Y = 0 | \mathbf{x}, \mathbf{w}) = 1 - P(Y = 1 | \mathbf{x}, \mathbf{w})$  – Stačí určit jedno
- **Logistická funkce**
  - o Jak donutit  $w_0 + w_1x_1 + \dots + w_px_p$ , aby neutekl z intervalu  $[0, 1]$ ?
  - o Dosadíme ho do vhodně zvolené funkce, jejíž obor hodnot je podmnožina intervalu  $[0, 1]$ 
    - Vždy dostaneme číslo smysluplně vyjadřující pravděpodobnost

- Obvyklá volba log. funkce – **Sigmoida**:

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

- o  $D_f = \mathbb{R}$ , za x dosazujeme  $w_0 + w_1x_1 + \dots + w_px_p$
- o  $H_f = (0,1)$  – nemůže být  $P(Y=1) = 1$  nebo  $P(Y=0) = 1$
- o Ostře rostoucí na  $\mathbb{R}$ , prostá, lichá,  $f(0)=1/2$ , limity pro  $x \rightarrow -\infty$  a  $x \rightarrow +\infty$  jsou 0 resp. 1
- **Model logistické regrese**
  - o Binární vysvětlovaná proměnná Y s hodnotami 0 a 1, p příznaků  $X_1, \dots, X_p$  s konstantním  $X_0 = 1$
  - o hledáme model pro odhad pravděpodobnosti, který pro x a koeficienty w má tvar:

$$P(Y = 1 | \mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^T \mathbf{x}}}{1 + e^{\mathbf{w}^T \mathbf{x}}}$$

- o Predikce – pro x se spočítá odhad pravděpodobnosti, je-li  $> 1/2$ ,  $Y=1$  – hranice rozhodnutí je  $1/2$
- **MLE odhad** parametrů w – metoda maximální věrohodnosti

$$p_1(\mathbf{x}, \mathbf{w}) = P(Y = 1 | \mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^T \mathbf{x}}}{1 + e^{\mathbf{w}^T \mathbf{x}}}$$

a

$$p_0(\mathbf{x}, \mathbf{w}) = P(Y = 0 | \mathbf{x}, \mathbf{w}) = 1 - \frac{e^{\mathbf{w}^T \mathbf{x}}}{1 + e^{\mathbf{w}^T \mathbf{x}}} = \frac{1}{1 + e^{\mathbf{w}^T \mathbf{x}}}$$

- o Pravděpodobnost i-tého datového bodu s hodnotou vysvětlované proměnné  $Y_i$  a s hodnotami příznaků  $\mathbf{x}_i$  pro zadané hodnoty parametrů w:  $P_{Y_i}(\mathbf{x}_i, \mathbf{w})$
- o Předpokládáme vzájemnou nezávislost datových bodů
- o **Pravděpodobnost** konkrétních trénovacích dat lze zapsat jako součin psť  $L(\mathbf{w}) = \prod_{i=1}^N p_{Y_i}(\mathbf{x}_i, \mathbf{w})$ , jednotlivých datových bodů při daném w – tuto funkci **maximalizujeme**
- o Odhad MLE odpovídá hodnotě p, pro která jsou trénovací data nejpravděpodobnější možná
- **Postup MLE**:
  1. Hledáme **maximum fce**  $L(\mathbf{w}) = \prod_{i=1}^N p_{Y_i}(\mathbf{x}_i, \mathbf{w})$ ,
  2. **Logaritmuje**me – protože maximalizujeme pravděpodobnost, ale nezajímají nás konkrétní hodnoty – hledáme maximum funkce  $\ell(\mathbf{w}) = \sum_{i=1}^N \left( Y_i \mathbf{w}^T \mathbf{x}_i - \ln(1 + e^{\mathbf{w}^T \mathbf{x}_i}) \right)$ .
  3. Najdeme **gradient**  $\nabla \ell(\mathbf{w}) = \mathbf{X}^T (\mathbf{Y} - \mathbf{P})$ , kde  $\mathbf{P} = (p_1(\mathbf{x}_1, \mathbf{w}), p_1(\mathbf{x}_2, \mathbf{w}), \dots, p_1(\mathbf{x}_N, \mathbf{w}))^T$ .

P ... vektor plný sigmoid se všemi exponenciálami
  4. Položíme gradient nule, abychom našli **maximum**

$$\nabla \ell(\mathbf{w}) = \mathbf{X}^T (\mathbf{Y} - \mathbf{P}) = 0.$$

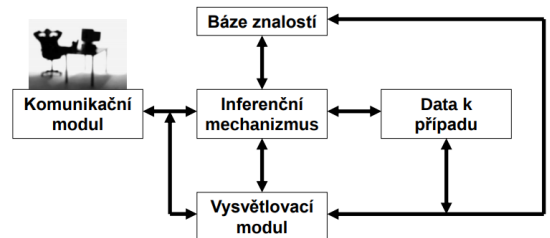
- Neumíme najít explicitní řešení jako u lineární regrese  $\rightarrow$  je nutné použít numerické aproximativní metody (více-rozměrná verze Newtonovy metody nebo gradientní vzestup, které konvergují k lokálnímu maximu)
- stojí na předpokladu, že se chování dat dá zachytit ve tvaru daném sigmoidou jakožto fci  $w_0 + w_1x_1 + \dots + w_px_p$
- výpočetně náročné a odhad není dán explicitně (počítač vrátí aproximaci nebo nic)

# Architektura znalostního systému, možnosti reprezentace znalostí

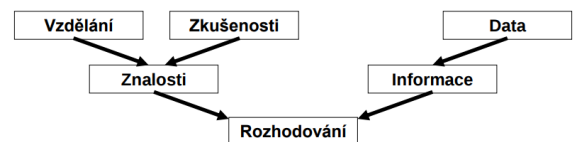
BI-ZNS

- **Znalostní systém** = inteligentní počítačový program, který využívá znalosti a inferenční procedury k řešení problémů, které jsou natolik obtížné, že pro své řešení vyžadují významnou lidskou expertizu
  - o Simuluje rozhodovací činnost experta (podtyp – expertní systém)
  - o V současnosti pohled na expertní systémy jako na podpůrný systém (asistent) → ZNS – „nadvstavba“ ES, automatizované získávání dat a znalostí
- **Charakteristické rysy ZS:**
  - o Oddělení báze znalostí a mechanismu jejich využívání – databáze pravidel + „prázdný“ engine
  - o Rozhodování za neurčitosti – podle pravděpodobnosti – důvěryhodnosti pravidel (použití heuristik)
  - o Schopnost vysvětlování – zobrazení cesty, dialogový režim (možnosti)
  - o Modularita a transparentnost báze znalostí
    - Kvalita přímo úměrná kvalitě báze znalostí
- **Typy ZS:**
  - o **Prázdný ZS** (shell) – báze znalostí je prázdná (prolog)
  - o **Problémově orientovaný ZS** – báze znalostí obsahuje znalosti z určité domény
  - o **Aplikačně orientovaný** – obsahuje i konkrétní znalosti k řešení
  - o **Diagnostický** – určuje, která z hypotéz koresponduje s daty z konkrétního systému
    - Diagnóza, interpretace, monitorování dat
  - o **Generativní** – hypotézy generovány dynamicky, splnění úlohy s omezeními na řešení
    - Plánování (nalezení posloupnosti k dosažení cíle), návrh (tvorba konfigurací), predikce
- **Architektura ZS**

- o **Báze znalostí** – zakódované znalosti experta (pravidla)
- o **Inferenční mechanismus** – odvozovací mechanismus (odvozování závěrů)
- o **Báze dat** – ke konzultovanému případu
- o **Vysvětlovací modul** – vysvětluje a zdůvodňuje své rozhodnutí uživateli
- o Rozhraní pro komunikaci s uživatelem (GUI)



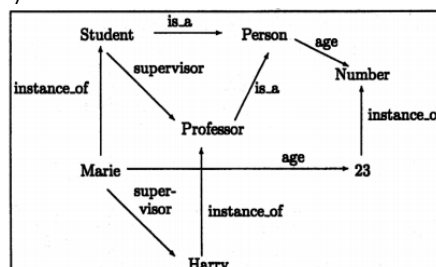
- **Báze znalostí**
  - o **Data** – získána automatickým procesem, lze objektivně verifikovat
  - o **Znalosti** – získány od experta, nelze objektivně verifikovat (zkušenosti, vzdělání, ...)



- **Metody reprezentace znalostí**

- o **Predikátová logika** – pro automatické dotazování – Robinsonův rezoluční princip
  - Jednoduchá reprezentace a aktualizace znalostí
  - Znalosti zapsány v jazyce predikátové logiky
- o **Sémantické sítě** – popis reality pomocí objektů a jejich vztahů (relací)
  - Relace → vyjadřování znalostí
  - Přirozená reprezentace pomocí grafu (uzly – objekty, hrany – relace)
  - Kompaktní, obtížná změna struktury

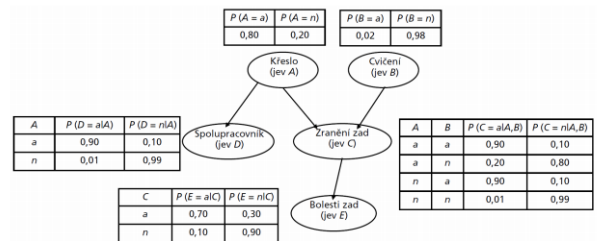
```
∀x, student(x) ⇒ person(x)
∀x, professor(x) ⇒ person(x)
∀x, person(x) ⇒ ∃y, numer(y) ∧ age(x,y)
∀x, student(x) ⇒ ∃y, professor(y) ∧ supervisor(x,y)
student(marie)
profesor(harry)
numer(23)
```





- **Bayesovské sítě** – popis pravděpodobnostní sítě vzájemných vazeb

- Odhad pravděpodobnosti nastání jevu
- Postaveny na podmíněné nezávislosti náhodných jevů

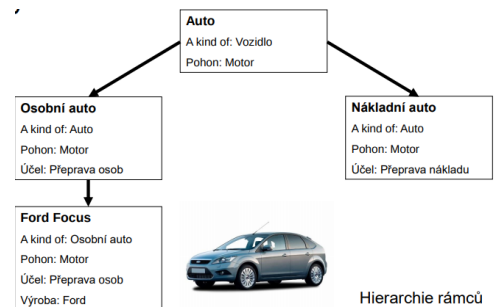


$$P(u_1, \dots, u_n) = \prod_{i=1}^n P(u_i | \text{rodice}(u_i))$$

Pravděpodobnost zranění zad ( $C = a$ ) při cvičení ( $B = a$ ), získáme po dosazení do vzorce  $p = 0,018$ , tedy z pravděpodobnost vzniku zranění zad při sportu pouze 1,8 %.

- **Rámce** – reprezentují stereotypní situace – statické znalosti

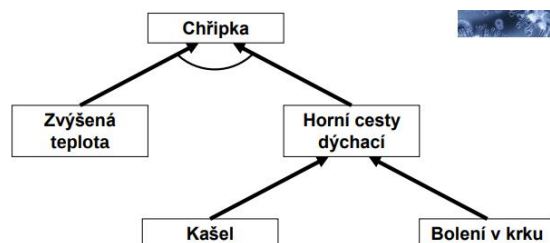
- Vyplňování stránek, do kterých se zapisují hodnoty položek
- Hierarchie pojmů (a kind of) nebo dekompozice (part of)
- Dají se zobrazit grafem
- Objekty (jako v OOP)



Hierarchie rámců

- **Pravidla** – nejběžnější způsob reprezentace

- Pomocí IF-THEN pravidel (žádné else není)
- **Tvrzení** (v předpokladu nebo závěru) může mít podobu:
  - Výrok – auto je červené
  - Dvojice (atribut, hodnota) – barva\_auto = červená
  - Trojice (objekt, atribut, hodnota) – auto\_32: barva = červená
- Atributy mohou být numerické nebo kategoriální (binární, nominální, ordinální)
- Pravidla v bázi znalostí se mohou znázornit **AND/OR grafem**
  - Uzly – výroky, orientace hran – pravidla
  - Konjunktivní vazba se znázorňuje obloučkem mezi hranami
- 3 typy tvrzení – uzlů:
  - Dotazy – pouze v předpokladech
  - Cíle – pouze v závěrech
  - Mezilehlá tvrzení – v předpokladech i závěrech

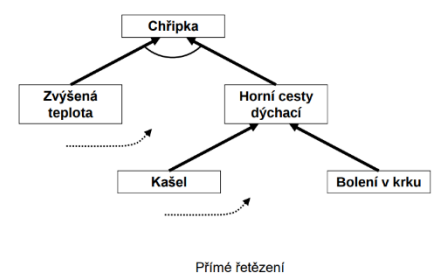
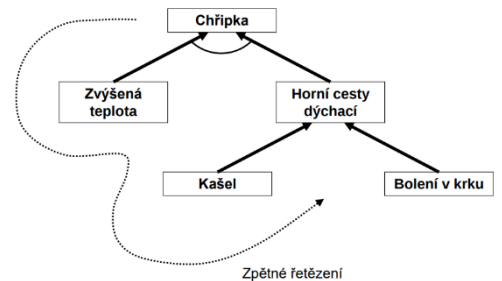




# Inferenční mechanismus, způsoby realizace inferenčního mechanismu

BI-ZNS

- **Inferenční mechanismus** – tvoří jádro ZS, „vyvozovací mechanismus“
- **Metody inferenčního mechanismu**
  - o **Logické metody** – dedukce, abdukce a indukce vychází z implikace z výrokové logiky
    - Převádíme znalosti do matematické podoby výrokové logiky (pravidla)
    - Vyvozujeme závěry ze souboru pravidel
    - **Dedukce**
      - platí pravidlo (implikace) a platí předpoklad  $\rightarrow$  odvozujeme platnost závěru (modus ponens) –  $A, (A \Rightarrow B \mid B)$
      - platí pravidlo a neplatí závěr  $\rightarrow$  odvozujeme neplatnost závěru (modus tollens) –  $(\neg B, A \Rightarrow B \mid \neg A)$
      - jestliže nemůže současně platit A a B a platí A, nemůže platit B (modus ponendo tollens) –  $B (\neg(A \wedge B) \wedge A \Rightarrow \neg B)$
    - **Abdukce** – platí pravidlo a platí závěr
      - předpoklad může a nemusí platit  $\rightarrow$  domníváme se, že může platit ( $B, A \Rightarrow B \mid A$ )
    - **Indukce** (generalizace z příkladů) – opakované pozorování, že A a B se vyskytuje současně
      - odvozujeme, že mezi A a B je implikace –  $A \Rightarrow B$  nebo  $B \Rightarrow A$
  - o **Rezoluční metoda** – vyvozování závěrů ze souboru pravidel
  - o **Zpětné řetězení** (backward chaining), též usuzování řízení cíli – diagnostické ZS
    - Vycházíme z cílů, chceme je odvodit a pokoušíme se nalézt pravidla umožňující tyto cíle potvrdit nebo vyvrátit
    - Procházíme zpětně od cílů (závěrů pravidel) k dotazům (předpokladům)
    - Využití dedukce
    - Pravidla se vyhodnocují v pořadí tak, jak jsou uložena v bázi znalostí nebo v pořadí stanovených priorit (míra neurčitosti)
    - Výroky se v předpokladech vyhodnocují tak jsou zapsány nebo podle cen
    - Výhody – účinné při malém množství hypotéz, diagnostika, hledá v logickém sledu
    - Nevýhody – postupuje se slepě od cíle dolů, nevýhodné při velkém množství hypotéz a málo vstupních datech
    - Vhodné pro hlubokou bázi znalostí s méně cíli, nemáme fakta a chceme vědět, co potřebujeme pro odvození cíle
    - *“Mám chuť na palačinky. ZS na základě stavu zásob řekne, zda je to možné.”*
  - o **Přímé (dopředné) řetězení** (forward chaining), též usuzování řízení daty – generativní ZS, hry
    - vycházíme z předpokladů a odvozujeme závěry, které slouží jako předpoklady pro další pravidla (od předpokladů s cílům – cíle nejsou předem známy)
    - inference probíhá přímočaře – pokud neznáme platnost podmínky, snažíme se ji získat z jiného pravidla a rekurzivně pokračujeme dál
    - hodně záleží na pořadí vyhodnocování pravidel
    - výhody – z malého množství mnoho nových faktů, výhodné pro sběr informací a plánování
    - nevýhody – nepozná důležitost vstupních informací, může se ptát v nelogickém sledu, nevýhodné při malém množství hypotéz a velkém množství vstupních dat
    - *“Co si dáme k večeři? ZS se zeptá na co máme chuť a podle zásob doporučí recept.”*
    - Vhodné pro plochou bázi znalostí s více cíli, kdy máme fakta a chceme naše možnosti



- **Fungování ZS** – běh = smyčka 3 fází
  - Porovnání – tvorba rozhodovací množiny s aplikovatelnými pravidly
  - Rozhodnutí sporu – výběr 1 pravidla (instance) z množiny (podle priority, heuristiky, ...)
  - Úkon – provedení akcí pravé strany vybrané instance (vytváření, modifikace, rušení objektů, ...)
    - Důsledek může být přidání nebo odstranění předpokladu z množiny, přidání pravidla do báze znalostí, ...
- **Strategie řešení konfliktu**
  - DFS (prohledávání do hloubky) – preferují se pravidla používající aktuálnější data
  - BFS (prohledávání do šířky) – preferují se pravidla používající starší data
  - Strategie složitosti – preferují se speciálnější/složitější pravidla (s více podmínkami)
  - Strategie jednoduchosti – preferují se jednodušší pravidla
- **Analogie** – při případovém usuzování – vyhledávání podobných případů
  - Znalosti = soubor dříve vyřešených případů
  - Snazší vývoj ZS – netřeba zodpovídat experta, ale třeba definovat vhodnou metriku
  - Odvozování založeno na srovnávání vzdáleností mezi případy
- **Generování a testování** – pro generativní ZS
  - Znalosti jsou reprezentovány pravidly (IF situace THEN akce)
  - Opakovaně generujeme možná řešení a testujeme, jestli vyhovují všem požadavkům
  - Mohou být splněny podmínky více pravidel
  - Nasycení předpokladů – existuje objekt vyhovující podmínkám pravidla
  - Instance – dvojice tvořená pravidlem a jeho nasycením
- Inference v diagnostických ZS – prohledávání báze pravidel, aplikace pravidel
  - Příp. práce s neurčitostí
- Vyhledávání – exhaustivní (užití všeho) x neexhaustivní (spokojení se s prvním)
  - To samé aplikace pravidel
- **Vyvozování závěrů ZS**
  - Vysvětlování – třeba zdůvodnit rozhodnutí
    - Vysvětlovací schopnost ZS umožňuje lépe ladit BZ
    - Možnosti vysvětlovacího modulu:
      - Why – cesta v síti pravidel
      - How – aktivovaná pravidla
      - What-if
- Implementační jazyky – Prolog, LISP

# Neurčitost ve znalostních systémech, její vyjadřování a zpracování

BI-ZNS

- **Neurčitost v ZS** – poznatky, které získáváme ze složitých systémů jsou neurčité a vágní nebo jsou nepřesně vyjádřeny
  - o Z neurčitých znalostí a odpovědí v průběhu konzultace se odvozují neurčité závěry
- Příčiny neurčitosti
  - o Problémy s daty – chybějící, nedůvěryhodné, nepřesné, nejisté
  - o Nejisté znalosti – neplatí univerzálně, můžou obsahovat vágní pojmy
- **Vyjádření neurčitosti** – většinou numerická hodnota – parametr (jedno číslo, dvojice), liší se podle ZS
  - o Váhy, pravděpodobnosti, stupně důvěry, faktory jistoty, ... – většinou intervaly (0, 1) nebo (-1,1)
  - o Numerické hodnoty neurčitosti se přiřazují jednotlivým tvrzením nebo pravidlům
- Přístupy ke zpracování neurčitosti
  - o **Ad hoc** – faktory jistoty, pseudobayesovské přístupy
  - o **Teoretické principy** – teorie pravděpodobnosti, fuzzy množiny, fuzzy míra
- Problémy při zpracování neurčitosti – jak kombinovat neurčitá data, neurčitost předpokladu s neurčitostí celkového pravidla, stanovení neurčitosti závěru
- Vyjádření a zpracování neurčitosti

A	¬A
0	1
X	X
1	0

A	B	0	X	1
0	0	0	0	0
X	0	X	X	X
1	0	X	1	1

A	∨	B	0	X	1
0	0	X	1		
X	X	X	1		
1	1	1	1		

A	⇒	B	0	X	1
0	1	1	1		
X	X	X	1		
1	0	X	1		

- o **Trojhodnotová logika** – true (1) – false (0) – unknown (X)
  - Dodefinování X do pravdivostní hodnoty logických spojek
- o **Váhy** – algebraická teorie – předpokládáme, že báze pravidel je tvořena pravidly ve tvaru  $A \rightarrow B(w)$ 
  - A – předpoklad pravidla – kombinace (konjunkce) výroků nebo jejich negací
  - B – závěr pravidla – tvořen jedním výrokem
  - W – váha pravidla – číslo z intervalu (-1, 1) - -1 = určitě ne, 0 = nevím, 1 = určitě ano
- o **Bayesovský přístup** – pravidla ve tvaru  $E \rightarrow H$ , které říká, že předpoklad E podporuje závěr H

- Neurčitost závěru H v závislosti na předpokladu E může být vyjádřena pomocí podmíněné pravděpodobnosti  $P(H|E)$
- Apriorní pravděpodobnostní šance:

$$O(H) = \frac{P(H)}{P(\neg H)} = \frac{P(H)}{1 - P(H)}$$

- Aposteriorní pravděpodobnostní šance:

$$O(H|E) = \frac{P(H|E)}{P(\neg H|E)} = \frac{P(H|E)}{1 - P(H|E)}$$

- Pravděpodobnost lze ze šance vypočítat podle vztahu  $P = \frac{O}{O+1}$
- **Míra postačitelnosti L** – z Bayesova vzorce pro aposteriorní pravděpodobnost:

$$O(H|E) = L \cdot O(H) \quad \text{kde} \quad L = \frac{P(E|H)}{P(E|\neg H)}$$

- Velká hodnota  $L \gg 1$  = předpoklad E je postačující pro dokázání závěru H
- Míru postačitelnosti L zadává expert

- **Míra nezbytnosti  $\bar{L}$ :**

$$O(H|\neg E) = \bar{L} \cdot O(H) \quad \text{kde} \quad \bar{L} = \frac{P(\neg E|H)}{P(\neg E|\neg H)}$$

- Malá hodnota  $\bar{L} \ll 1$  = předpoklad E je nezbytný pro dokázání závěru H
- Opět zadává expert

- Váhy pravidel – pravidlo  $E \rightarrow H$  se chápe jako "if E then H with weight L else H with weight  $\bar{L}$ ", tedy dvojice pravidel  $E \rightarrow H$  a  $\neg E \rightarrow H(\bar{L})$

- Místo uvedených měr mohou být expertem zadány pravděpodobnosti  $P(H|E)$  a  $P(H|\neg E)$ , z nichž se tyto míry vypočtou

$$\text{aposteriorní pr.} \rightarrow P(H|E) = \frac{\text{podmíněná pr. } P(E|H)P(H)}{\text{úplný rozkl. pr. } P(E)} \leftarrow \text{apriorní pr.}$$

- Kombinace více pravidel  $E_1 \rightarrow H \dots E_n \rightarrow H$ 
  - Aposteriorní šance při nezávislosti předpokladů:

$$O(H | E_1 \wedge \dots \wedge E_n) = L_1 \cdot \dots \cdot L_n \cdot O(H)$$

- Kombinace více předpokladů – pomocí vztahů používaných ve fuzzy logice:

Disjunkce předpokladů:

$$P(E_1 \vee E_2) = \max\{P(E_1), P(E_2)\}$$

Konjunkce předpokladů:

$$P(E_1 \wedge E_2) = \min\{P(E_1), P(E_2)\}$$

Negace předpokladu:

$$P(\neg E) = 1 - P(E)$$

- Výhody a nevýhody bayesovských přístupů
  - Výhody – dobře podložené teoretické základy, dobře definovaná sémantika rozhodování
  - Nevýhody – potřeba stanovení velkého množství různých pravděpodobností, riziko neúplnosti dat, předpoklady by měly být nezávislé

- **Přístup založený na faktorech jistoty** – pravidla ve tvaru  $E \rightarrow H$  s faktorem jistoty CF

- faktor jistoty z intervalu  $(-1, 1)$  a je určen spojením míry důvěry (MB) a nedůvěry (MD)

$$CF = \frac{MB - MD}{1 - \min\{MB, MD\}}$$

- Vyjadřuje stupeň důvěry v hypotézu H, jestliže je předpoklad E pravdivý

- $CF > 0$ : E zvyšuje důvěru H (+1 = maximální důvěra)
- $CF < 0$ : E snižuje důvěru H (-1 = maximální nedůvěra)

- **Míra důvěry** (Measure of Belief) – interval  $(0, 1)$ :

- Přírůstek pravděpodobnosti (důvěry) hypotézy H získané (podporované) evidencí E

$$MB(H, E) = \begin{cases} 1 & \text{pro } P(H) = 1 \\ \frac{\max\{P(H|E), P(H)\} - P(H)}{1 - P(H)} & \text{jinak} \end{cases}$$

- **Míra nedůvěry** (MD) – interval  $(0, 1)$ :

- Pokles pravděpodobnosti (důvěry) hypotézy H získané (podporované) evidencí E

$$MD(H, E) = \begin{cases} 1 & \text{pro } P(H) = 0 \\ \frac{P(H) - \min\{P(H|E), P(H)\}}{P(H)} & \text{jinak} \end{cases}$$

- předpoklad E nemusí být znám s absolutní jistotou (např.

je odvozením z jiného pravidla s vlastním faktorem jistoty), výsledný faktor jistoty se pak vypočte

$$CF_{new}(H, E) = CF_{old}(H, E) \cdot CF(E)$$

- Kombinace více pravidel:

Konjunkce:  $CF(E_1 \wedge E_2) = \min\{CF(E_1), CF(E_2)\}$

Disjunkce:  $CF(E_1 \vee E_2) = \max\{CF(E_1), CF(E_2)\}$

- **výhody**: jednoduchý a účinný výpočetní model, sběr potřebných dat je snazší než u ostatních metod

- **nevýhody**: chybí teoretické základy, předpoklad nezávislosti  $E_i$

$$CF_n = \begin{cases} CF_{n-1} + CF(H, E_n) \cdot \overbrace{(1 - CF_{n-1})}^{\text{přírůstek jistoty}} & \text{váha přírůstku} \\ & \text{pro } CF_{n-1} > 0 \text{ a } CF(H, E_n) > 0 \\ CF_{n-1} + CF(H, E_n) \cdot (1 + CF_{n-1}) & \text{pro } CF_{n-1} < 0 \text{ a } CF(H, E_n) < 0 \\ \frac{CF_{n-1} + CF(H, E_n)}{1 - \min\{|CF_{n-1}|, |CF(H, E_n)|\}} & \text{jinak} \end{cases}$$

- **Nemonotónní usuzování** – neopírá se vyjádření neurčitosti jako číselné hodnoty

- způsob inference, kdy je dříve učiněný závěr zpochybněn ve světle nové informace (předcházející znalost může přestat platit, když se dozvíme nové informace)
- např. „Každý pták létá“  $\rightarrow$  zpochybnění „tučňák nelétá“ - přidání dodatečné formule (znalosti)

# Základy fuzzy logiky, neurčitost, relace

BI-ZNS

- **Fuzzy logika** – podmnožina klasické logiky (stejně značení) odvození od teorie fuzzy množin
  - o Věci v reálném světě nemají pouze binární jako u normální logiky (nepatří/patří)
  - o Zavádí se **funkce příslušnosti** – každému prvku přidělí číslo z intervalu  $(0,1)$ :  $\mu_A: X \rightarrow [0, 1]$
  - o **Fuzzy množina** – psána ve tvaru:  $A = \{(x, \mu_A(x)) | x \in X\}$
  - o **spočetná fuzzy množina** – výčet prvků s přísl. hod. funkce příslušnosti (nejde o součet, ale sjednocení)
$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \dots + \mu_A(x_n)/x_n = \sum_{i=1}^n \mu_A(x_i)/x_i$$
  - o můžeme znázornit pomocí zobecněných Vennových diagramů.
  - o *příklad*: rozdělení lidí na malé a velké. Kde je ta hranice? Výčet malých můžeme zapsat jako  $A = 1/150 + 0.9/160 + 0.5/170 + 0.1/180$  (fce příslušnosti/výška)

- **Základní pojmy fuzzy množin**

Nosič fuzzy množiny (support):  $\text{supp } A = \{x \in X, \mu_A(x) > 0\}$

Výška fuzzy množiny (height):  $\text{hght } A = \sup_{x \in X} \mu_A(x)$

Normální fuzzy množina (normal):  $\exists x \in X, \mu_A(x) = 1$

Prázdná fuzzy množina (empty):  $\forall x \in X, \mu_\emptyset(x) = 0$

Jádro fuzzy množiny (kernel):  $\text{Ker } A = \{x, \mu_A(x) = 1\}$

Fuzzy jednotka (singleton):  $A = \{x/1\}$

$\alpha$ -řez ( $\alpha$ -cut):  $A_\alpha = \{x \in X, \mu_A(x) \geq \alpha\}$

$\alpha$ -hladina ( $\alpha$ -level):  $A^\alpha = \{x \in X, \mu_A(x) = \alpha\}$

Skalární kardinalita (scalar cardinality):  $|A| = \sum_{x \in X} \mu_A(x)$

**Fuzzy podmnožina** (fuzzy subset) – fuzzy množina A je

podmnožinou fuzzy množiny B ( $A \subseteq B$ ), jestliže pro

všechny prvky těchto fuzzy množin platí, že  $\mu_A(x) \leq \mu_B(x)$ .

**Rovnost fuzzy množin** -  $A \subseteq B$  a  $B \subseteq A$ , resp.  $\mu_A(x) = \mu_B(x)$

**Fuzzy komplement** množiny A:  $\mu_{A'}(x) = 1 - \mu_A(x)$ .

**Fuzzy průnik** množin A a B:  $\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$ .

**Fuzzy sjednocení** množin A a B:  $\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$ .

**Kartézský součin** množin A a B:  $\mu_{A \times B}(x) = \min\{\mu_A(x), \mu_B(x)\}$ .

**n-tá mocnina** množiny A:  $A^n = R, \mu_R(x) = [\mu_A(x)]^n$ .

Algebraický (pravděpodobnostní) součet  $C = A + B$

$$\mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$$

Omezený součet  $C = A \oplus B$

$$\mu_{A \oplus B}(x) = \min\{1, \mu_A(x) + \mu_B(x)\}$$

Omezená difference  $C = A \ominus B$

$$\mu_{A \ominus B}(x) = \max\{0, \mu_A(x) + \mu_B(x) - 1\}$$

Algebraický součin  $C = A \cdot B$

$$\mu_{A \cdot B}(x) = \mu_A(x) \cdot \mu_B(x)$$

- **Operace modifikující funkce příslušnosti**

**koncentrace** – snižuje fuzzy neurčitost (ostřejší fuzzy množina)

$$\text{CON}(A) = A^2$$

$$\mu_{\text{CON}(A)} = (\mu_A(x))^2$$

**dilatace** – zvyšuje fuzzy neurčitost (plošší fuzzy množina)

$$\text{DIL}(A) = A^{0.5}$$

$$\mu_{\text{DIL}(A)} = (\mu_A(x))^{0.5}$$

- **Multikriteriální rozhodování** používá operace, které jsou kombinacemi různých operací (např. vztahem pro funkci příslušnosti

$$\mu_k(a, b) = \gamma \min(a, b) + (1 - \gamma) \max(a, b)$$

- o parametr  $\gamma$  z intervalu  $(0, 1)$  řídí, jakou měrou se výsledná operace bude průnikem (minimum,  $\gamma = 1$ ) nebo sjednocením (maximum,  $\gamma = 0$ )

- **Příklad:**

$$A = 0,8/3 + 0,3/4 + 0,2/5 + 0,6/6$$

$$B = 0,7/3 + 1/4 + 0,5/6$$

Operace	Označení	Hodnoty
komplement	$A'$	$0.2/3 + 0.7/4 + 0.8/5 + 0.4/6$
max	$A \cup B$	$0.8/3 + 1/4 + 0.2/5 + 0.6/6$
min	$A \cap B$	$0.7/3 + 0.3/4 + 0.5/6$
mocnina	$A^2$	$0.64/3 + 0.09/4 + 0.04/5 + 0.36/6$
alg. součet	$A+B$	$0.94/3 + 1/4 + 0.2/5 + 0.8/6$
alg. součin	$A \cdot B$	$0.56/3 + 0.3/4 + 0.3/6$
omez. součet	$A \oplus B$	$1/3 + 1/4 + 0.2/5 + 1/6$
omez. difer.	$A \ominus B$	$0.5/3 + 0.3/4 + 0.1/6$
drast. součet	-	$1/3 + 1/4 + 0.2/5 + 1/6$
drast. součin	-	$0.3/4$

- **Fuzzy relace** – funkce příslušnosti pro dvojice prvků

- o relace vznikne jakou kartézský součin jednotlivých množin (tzn. minimum z hodnot funkcí příslušnosti)
- o dá se zobrazit *sagitálním grafem*: prvky (vrcholy) jsou spojeny ohodnocenými hranami (relace)
- o *Příklad*: dvojice měst, které jsou “velmi daleko”

- Binární relace “velmi daleko” můžeme zapsat

jako fuzzy množinu:

$$R = 0,7/(Praha, Plzeň) + 0,65/(Praha, Rokycany) + 0,1/(Plzeň, Rokycany) + 1/(Praha, Mnichov) + 0,9/(Plzeň, Mnichov)$$

	Praha	Plzeň
Praha	0	0.7
Rokycany	0.65	0.1
Mnichov	1	0.9

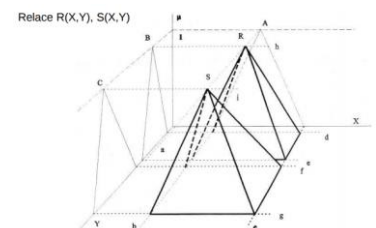
- **Výška fuzzy relace** – max. hodnota funkce příslušností dané relace  $h(R(x, y)) = \max_{y \in Y} \max_{x \in X} \mu_R(x, y)$

- **Projekce relace** – projekcí relace je fuzzy množina, značení: *Proj R na V* ( $R \downarrow V$ )

- **Cylindrické rozšíření** – množina všech relací, které mají stejnou projekci, značení:  $Ce(A)(A \uparrow V)$

- o  $Ce(B)$  je hranol o podstavě tvořené funkcí příslušnosti množiny B s povrchovými přímkami d, e, h (tzn. udělá se kvádr místo čtyřštěnu)

- Projekcí relace  $R(X, Y)$  do Y je množina B
- Projekcí relace  $S(X, Y)$  do Y je množina C
- Projekcí množin A a B do X je množina A



- **Kompozice relací**  $R(X, Z) = P(X, Y) \circ Q(Y, Z)$  je podmnožina kartézského součinu  $X \times Z$  taková, že  $(x, z) \in \mathbb{R}$ , tehdy a jen tehdy, jestliže existuje alespoň jedno  $y \in Y$  takové, že  $(x, y) \in P$  a  $(y, z) \in Q$

- o Základní typy kompozic:

- **Max-min kompozice** – vezmou se složky z matic jako u násobení a z každé dvojice se vybere to nižší číslo, poté z těchto vybraných vezmu to největší a umístím na odpovídající místo ve výsledné matici

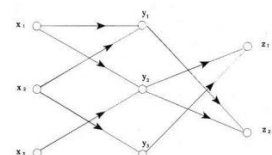
$$\mu_{P \circ Q}(x, z) = \max_{y \in Y} \min \{ \mu_P(x, y), \mu_Q(y, z) \}$$

- **Max-product kompozice** – jako násobení matic, ale místo násobení a sčítání se prvky nejdříve vynásobí a potom se vybere ta maximum ze součinů

$$\mu_{P \otimes Q}(x, z) = \max_{y \in Y} \{ \mu_P(x, y) \cdot \mu_Q(y, z) \}$$

**Příklad:**

$$P(x, y) = \begin{matrix} & y_1 & y_2 & y_3 \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0,1 & 0,4 & 0,8 \\ 0,5 & 1 & 0,9 \\ 0,2 & 0,5 & 0,4 \end{bmatrix} \end{matrix} \quad Q(y, z) = \begin{matrix} & z_1 & z_2 \\ \begin{matrix} y_1 \\ y_2 \\ y_3 \end{matrix} & \begin{bmatrix} 0 & 0,8 \\ 0,3 & 0,2 \\ 0,1 & 0 \end{bmatrix} \end{matrix}$$



$$\begin{bmatrix} 0,1 & 0,4 & 0 \\ 0,5 & 0 & 0,9 \\ 0 & 0,5 & 0,4 \end{bmatrix} \circ \begin{bmatrix} 0 & 0,8 \\ 0,3 & 0,2 \\ 0,1 & 0 \end{bmatrix} = \begin{bmatrix} 0,3 & 0,2 \\ 0,1 & 0,5 \\ 0,3 & 0,2 \end{bmatrix}$$

$$\begin{bmatrix} 0,1 & 0,4 & 0 \\ 0,5 & 0 & 0,9 \\ 0 & 0,5 & 0,4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0,8 \\ 0,3 & 0,2 \\ 0,1 & 0 \end{bmatrix} = \begin{bmatrix} 0,12 & 0,08 \\ 0,09 & 0,4 \\ 0,15 & 0,1 \end{bmatrix}$$

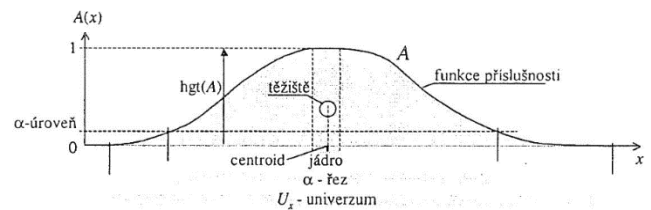


# Reprezentace znalostí a inference pomocí fuzzy systémů

BI-ZNS

- **Funkce příslušnosti**  $\mu_A: X \rightarrow [0, 1]$  – určuje, jak moc hodnota  $x$  patří do fuzzy množiny  $A$

- o může mít různé tvary (gaussova křivka, trojúhelník, lichoběžník...)



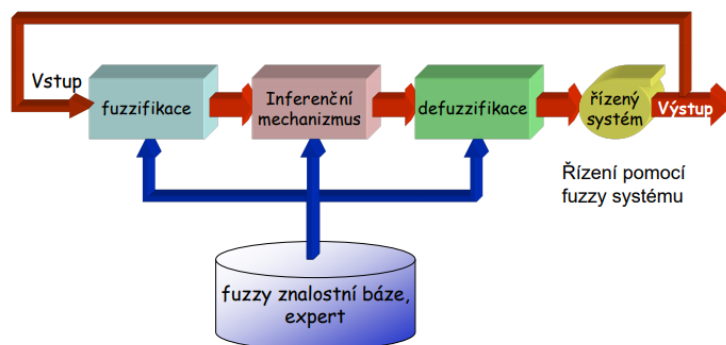
- Základní pojmy:

- o vstupní/výstupní jazyková proměnná (nemusí být pouze číselné hodnoty)
- o vstupní logické fuzzy proměnné (termy)
- o ostrý (crisp) vstup/výstup
- o fuzzy hodnota – číslo, pravdivostní hodnota proměnné

- Operace ve fuzzy logice

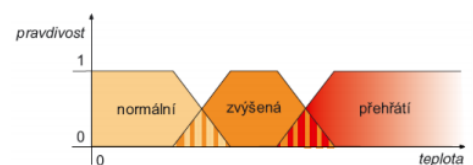
- o negace:  $\text{NOT}(A) = 1 - A$
- o fuzzy-logický součin: různé implementace, např.  $A \text{ AND } B = \min(A, B)$
- o fuzzy-logický součet: různé implementace, např.  $A \text{ OR } B = \max(A, B)$

- Struktura fuzzy systému



- **Fuzzifikace** – proces, který převádí číselné vstupní hodnoty jazykové proměnné na pravdivostní hodnoty souboru vstupních termů

- o např. teplota 37 °C  $\Rightarrow$  normální 0.0, zvýšená 0.3, přehřátí 0.7

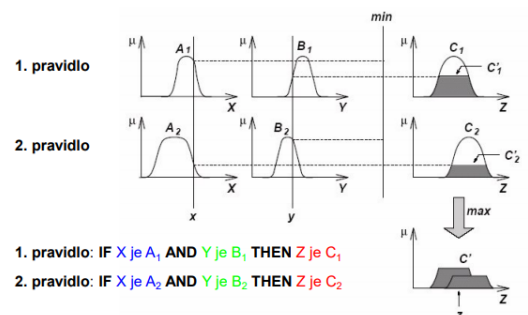


- **Inferenční mechanismus** (vyvozovací mechanismus)

- o Soubor pravidel ve tvaru „IF podmínka THEN důsledek“
- o transformuje pravdivostní hodnoty vstupních proměnných (termů) na pravdivostní hodnoty výstupních proměnných
- o složení výsledku z dílčích závěrů – několik možných způsobů:

- **Mamdaniho model** – pravidla tvaru „IF  $x$  je malé AND  $y$  je střední THEN  $z$  je velké“

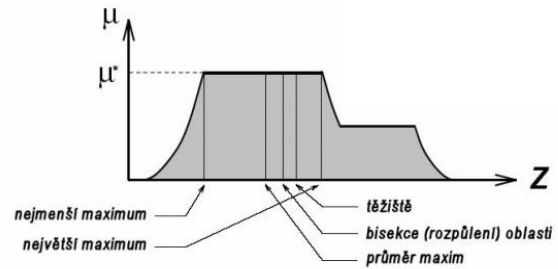
- na levé straně podmínka vyjádřená jako fuzzy logická relace mezi stupni příslušnosti proměnných k fuzzy množinám
- na pravé straně je podmínka vyjádřená jako stupeň příslušnosti proměnné k fuzzy množině



- **Takagiho-Sugenův model** – pravidla tvaru „IF  $x$  je malé AND  $y$  je střední THEN  $z=f(x,y)$ “

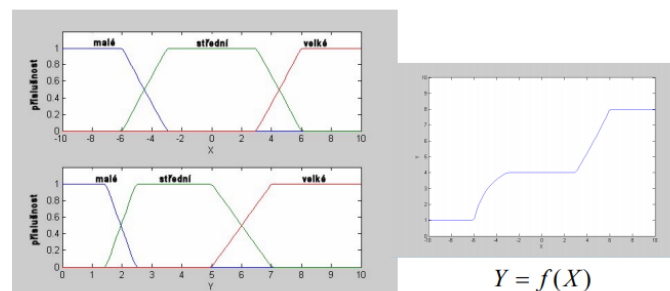
- Levá strana stejná jako u Mamdaniho
- na pravé straně je podmínka vyjádřená jako **funkce proměnných**
- Nejčastěji se volí funkce  $f$  jako polynom (lineární funkce)
  - o  $z = f(x,y) = a.x + b.y$  ( $a, b$  koeficienty, které známe předem)

- **Defuzzifikace** – proces, kterým se převádí fuzzy výstup inferenčního mechanismu (pravdivostní hodnoty výstupních proměnných) na ostrou hodnotu s pomocí funkcí příslušnosti
  - o Metody:
    - nejmenší maximum
    - největší maximum
    - těžiště
    - bisekce oblasti (stejně velké plochy nalevo i napravo)
    - průměr maxim
  - o může tam být obrovský rozdíl, třeba zkoušet co funguje (lepší je těžiště/bisekce – nejsou to samé!)
- Pro nastavení modelů fuzzy systému se často používá neuronová síť.
- **Použití fuzzy systému** – termostaty, pračky, pohyb robota...
- Nevýhoda – potřeba odborníka na seřízení systému, ladění řady parametrů
- **Příklad**



Pravidlo 1 : IF  $X$  je malé THEN  $Y$  je malé  
 Pravidlo 2 : IF  $X$  je střední THEN  $Y$  je střední  
 Pravidlo 3 : IF  $X$  je velké THEN  $Y$  je velké

$X = \text{vstup} \in [-10, 10]$   
 $Y = \text{výstup} \in [0, 10]$



- **Postup:**
  - o **Fuzzifikace** – Uděláme si pro každý objekt grafy (viz graf na straně 1) a přijdou nám nějaké číselné vstupy (37 °C). Mrknem se do těch grafů a řekneme, že s hodnotou, která přišla, to patří z 0.3 do funkce příslušnosti zvýšená a z 0.7 do přehřátí
  - o **Inferenční mechanismus** – aplikujeme všechna pravidla ve tvaru IF podmínka THEN důsledek (např. *IF teplota je přehřátí AND sucho je velké THEN šance požáru je vysoká*) a vyhodí nám to pro každou podmínku nějaké číslo. Pokud 0, výsledek zahodíme, jinak vezmeme fuzzy graf pro objekt, kterému chceme predikovat stav (tady *šance požáru*) a podle toho, jaké číslo nám vyšlo, tak s takovým množstvím zahrneme závěr pravidla (zde *vysoká*) do celkového závěru (takže např. ve finále bude šance požáru vysoká 0.7, střední 0.3)
  - o **Defuzzifikace** – vezmeme tenhle závěrečný graf a spočítáme (třeba pomocí těžiště všech funkcí příslušnosti) výslednou hodnotu (např. vyjde 70, takže šance požáru je ve výsledku 70 %)

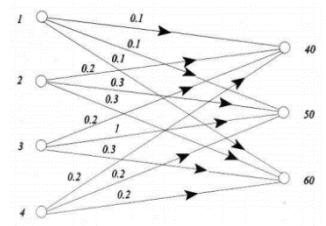


# Reprezentace znalostí a inference pomocí neuronových sítí

## BI-ZNS + BI-VZD

- **Neuronové sítě** – inspirovány neurony živých organismů
    - o Schopnost extrakce a reprezentace závislostí na nezřejmých datech
    - o Schopnost řešit nelineární úlohy, učení se a zevšeobecnování
    - o Využití – klasifikace, regrese, predikce časových řad
    - o Distribuovaný výpočetní systém sestávající se z dílčích podsystémů – neuronů – inspirovaných nervovým systémem organismů, který jejich činnosti více či méně realizuje
    - o Více v otázce 15
  - **Fuzzy-logická NS** – chování systému lze napsat jako fuzzy relaci vyjádřenou tabulkou (také pomocí grafů – sagitální graf)
    - o do vstupních proměnných je zakódován vstup i se všemi fuzzy hodnotami
    - o neuronová síť se ze své podstaty po naučení chová přesně jako znalostní systém (snaží se co nejvíc přiblížit systému, které známe z dřívějšíka)
      1. Vrstva – převod hodnoty na míry příslušející do dané fuzzy množiny
      2. Vrstva – stupeň aktivace pravidla
      3. Vrstva – výpočet míry důležitosti pravidla k podílu na vstupu
      4. Vrstva – výpočet dílčího výstupu daného pravidla
      5. Vrstva – defuzzyfikace (výpočet konkrétní hodnoty výstupu)
- |   | Vstup (PM) |     |     |
|---|------------|-----|-----|
|   | 40         | 50  | 60  |
| 1 | 0.1        | 0.1 | 0.1 |
| 2 | 0.2        | 0.3 | 0.3 |
| 3 | 0.2        | 1   | 0.3 |
| 4 | 0.2        | 0.2 | 0.2 |

Malý kladný vstup pak může být chápán jako fuzzy množina  
 $PS = 0,1/1 + 0,3/2 + 1/3 + 0,2/4$  a  
 střední kladný výstup jako fuzzy množina  
 $PM = 0,2/40 + 1/50 + 0,3/60$


- Navržený neuro-fuzzy systém obsahuje řadu parametrů (odpovídají vahám na spojích), které lze modifikovat – učit
    - o Učení – nejčastěji backpropagation – algoritmus zpětného šíření chyby
  - komplexní architektura NS umožňuje namapovat celou řadu algoritmů a následně optimalizovat váhy pomocí optimalizační metody (deep learning, fuzzy systémy...)
  - Typy NS – vícevrstvá perceptronová síť (MLP), síť RBF, Kohonenovy samoorganizující se mapy
  - Základní úlohy NS – klasifikace a refrese (aproximace) s učitelem/bez učitele
  - **Extrakce pravidel z NS**
    - o **TREPAN algoritmus** – „pedagogický algoritmus“ – back box, známe jen vstupy a odp. Výstupy
      - na bázi konstrukce rozhodovacího stromu s pomocí dotazů a již získaných odpovědí z naučené NS – jen generátor odpovědí na náhodné vstupy
      - vlastní konstrukce stromu – modifikovaný algoritmus ID2-of-3 – expanduje uzly s největším potenciálem zvýšit důvěryhodnost RS ve srovnání s původní NS
      - pro štěpení uzlu pravidla ve tvaru m-of-n, m je prahová hodnota kolik z n podmínek musí být splněno, kritéria zastavení = dostatečná čistota uzlu, počet vnitřních uzlů
      - nahrazuje NS, tedy méně přesný, s NS pracuje jen jako s black boxem
    - o **CRED algoritmus** – „dekompoziční algoritmus“ – známe strukturu sítě, kterou musíme rozložit na části
      - Generuje po vrstvách – my generujeme pravidla mezi vstupem a 1. Skrytou vrstvou a druhou sadu mezi 1. S.v. a výstupem = dvouvrstvá síť
        1. Převod spojitých výstup. prom. na diskretní třídy – využití rozhodovacích stromů
        - 2./3. Tvorba pravidel mezi skrytou vrstvou a výstupem/vstupem – tvorba dvou stromů
        4. sestavení celkových pravidel z dílčích pravidel – substituce vstupních p. do mezilehlých p.
        5. zjednodušení celkových pravidel
      - Naučená NS se umí vypořádat se šumem – extrahovaná pravidla jím nebudou zatížena
      - Výběr vhodných atributů – zpětná vazba – zlepšení modelu
  - **Báze znalostí** – trénovací data a NS na nich zkonstruované (včetně vah)
  - **Inference** – rozhodování NS na základě vstupu nových dat

# Reprezentace znalostí a inference pomocí rozhodovacích stromů

BI-ZNS + BI-VZD

- Postupné hierarchické rozdělování prostoru dat na podskupiny tak, aby v listech vytvářeného stromu byly (homogenní) skupiny dat náležící (v případě klasifikace) jedné třídě nebo reprezentující skupinu podobných hodnot (v případě regrese)
- **Klasifikační stromy = rozhodovací stromy**
  - o Založeny na postupném rozdělování prostoru příznaků
  - o Zařazení objektu do odpovídající třídy na základě diskretních příznaků
  - o Jednoduché, rychlé, snadná vizualizace
  - o Nekladou podmínky na typ pravděpodobnostního rozdělení sledovaných veličin
  - o Odolné vůči odlehlým a chybějícím hodnotám
- **Regresní stromy**
  - o Neodhaduje se diskretní veličina (kategorický výstup, jejíž rozdělení se uvnitř identifikovaných skupin – uzlů znázorňuje pomocí histogramu), ale spojitá veličina
    - Prokládá se hodnotami uvnitř uzlu normálním rozdělením – průměr a rozptyl
    - Průměr hodnot závislé veličiny
    - Výsledek je schodovitá predikce
- **C&RT stromy** – binární klasifikační a regresní stromy
  - o Diskretní veličiny → přirozeně určené dělicí body
  - o Spojité veličiny → diskretizace
  - o **CART stromy** – tvorba hladovým způsobem
    - Tvorba: v každém kroku se úplným prohledáváním najde ta proměnná a takový dělicí bod, které přinesou okamžité (aktuálně) nejvyšší zlepšení klasifikace nebo regrese
    - Tvořeny tak, aby co nejlépe odpovídaly trénovacím datům
  - o Regresní stromy – minimalizuje se reziduální součet čtverců
  - o Klasifikační stromy – minimalizuje se % chybné klasifikace, Gini index nebo vzájemná entropie
  - o Učíme strom na TROCHU jiných datech → úplně jiná struktura
  - o **GC&RT** – zobecněné klasifikační a regresní stromy – klasické C&RT obohacené o podpůrné metody jako prořezávání, křížovou validaci atd.
- **Klasifikace** – přesnost ovlivňuje:
  - o **Cenová/ztrátová matice** – sloupce skutečné hodnoty, řádky predikované třídy → tabulka s penalizačními hodnotami, když vzor zařadím do špatné třídy
  - o **Apriorní pravděpodobnosti** zastoupení jednotlivých tříd – vyjadřují pravděpodobnost s jakou vzor patří do dané třídy
    - Odhadnuté – na základě poměrného zastoupení tříd
    - Stejně – stejná pst zařazení vzoru do libovolné třídy
    - Definované uživatelem – vlastní pravděpodobnosti
    - Přímá vazba s cenovou maticí
  - o **Poměrné zastoupení vzorů** jednotlivých tříd v datech
    - Gini index „nečistoty“ uzlu
    - $p_i$  = pst zastoupení kategorie uzlu
    - 0 když jsou v uzlu vzory jedné kategorie, maximální, když jsou rovnoměrně zastoupeny všechny
- **Zastavovací pravidla růstu stromu**
  - o **Prořezávání na základě klasifikační chyby** – strom je prořezán tak, aby byla zachována stejná chyba klasifikace do všech tříd, používá nákladovou matici
  - o **Prořezávání na základě odchylky**: používá pro prořezávání stromu rozdíl (odchylku) mezi pravděpodobností správné klasifikace nejlepšího modelu a aktuálně prořezávaného modelu

Class	Class SETOSA	Class VERSICOL	Class VIRGINIC
SETOSA		2,1	1
VERSICOL	0,4		1
VIRGINIC	1	1	

$$GI = \sum_{i=1}^k p_i(1 - p_i) = 1 - \sum_{i=1}^k p_i^2$$

- **Přímé zastavení růstu stromu** – zastavení růst stromu na základě dosažení maximálního předepsaného zastoupení chybných vzorů v uzlu, nepoužívá se zpětné prořezávání.
  - **Min. počet případů** – pokud počet příp. (vzorů) v uzlu klesne pod tuto hod., nebude se uzel dále dělit
  - **Maximální počet uzlů** – dělení stromu se zastaví při dosažení této hodnoty.
  - **Maximální výška stromu** – maximální počet úrovní stromu
  - **Konstrukce stromu** – náhradní proměnné
    - umožňuje vytvořit strom, který dokáže klasifikovat vzory, kterým chybí nějaké hodnoty atributů
    - pokud chybí hodnota atributu, algoritmus využije náhradní atribut, který dělí vzory podobně jako původní (pokud ani ten nenajde, hledá dokud není vyčerpán seznam náhradníků)
  - **Random forest** – kolekce libovolného počtu jednoduchých stromů, kde každý je modelem klasifikační/regresní úlohy a jedním výstupem, na kterém se podílí hlasováním (klasifikace), nebo průměrováním (regrese)
  - **Boosted trees**
    - založené na boostingu (generování více modelů a jejich složení na základě tak, aby vznikl z vnějšího pohledu jeden výsledný model)
    - generujeme jednoduché stromy a každý následující strom je motivován, aby byl trénován na neúspěšných vzorech stromu předcházejícího
    - *inicializace*: každému vzoru (bodu) přiřadíme stejnou váhu a ta určuje šanci s jakou bude vzor vybrán (většinou rovnoměrně)
    - vybereme množinu (podle vah) a na ní se natrénuje jednoduchý strom, ten vyzkoušíme na trénovacích/testovacích datech
    - podle úspěšnosti upravíme váhy těm bodům, které tento strom klasifikoval špatně (ta určuje pravděpodobnost vybrání bodu v další iteraci)
    - v každém kroku dostáváme strom, který má velkou šanci být lepší než ten předchozí, výsledný model je složen ze stromů každé iterace (podle jejich vah)
  - Přeučení – overfitting – přehnaný důraz na přesnost → naučení chyb a šumu
  - **Báze znalostí**: trénovací data, zkonstruované stromy a hodnoty v jejich listech
  - **Inference**: rozhodování a odpověď jednotlivých stromů na základě vstupu
  - **Extrakce pravidel z rozhodovacích stromů** – už na první pohled představuje RS soubor viditelných pravidel, ale existují sofistikované postupy na lepší extrakci sady pravidel
    - *důvody*: vyhovění struktuře báze pravidel, inferenčnímu mechanismu a vysvětlovacímu modulu, zjednodušení a zlepšení přesnosti (potlačení přeučení)
    - *počáteční stav*: existující rozhodovací strom a trénovací množina
    - *koncový stav*: sada optim. pravidel ve formátu (*IF složená podmínka THEN třída – neurčitost*)
  - 1) **vytvoření primitivních pravidel** (výchozích stavů pro optimalizaci) – procházení stromu od kořene k listům
 

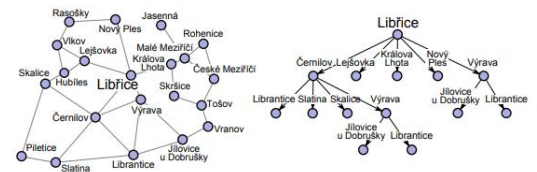
**IF  $P_1$  &  $P_2$  & ... &  $P_n$  THEN třída C,**  
 $P_i$  ... jednotlivé podmínky mezilehlých uzlů, C... třída představovaná listem
  - 2) **optimalizace primitivního pravidla** – zjednodušení podmínek, sloučení podmínek a mezilehlých uzlů, prořezání stromu (ještě před vygenerováním pravidel)
  - 3) **analýza důležitosti** – tabulka podmínky  $P_i$  a třídy C, vytvoření kontingenční tabulky a vytvoření faktoru jistoty
 

	Třída C	Jiná třída
$P_i$ splněna	TP	FP
$P_i$ nesplněna	TN	FN
  - 4) **odstranění podmínky  $P_i$**  – podmínka  $P_i$  je z pravidla vypuštěna, pokud její odstranění nesníží faktor jistoty, nebo hypotéza, že podmínka  $P_i$  je zbytečná, nemůže být zamítnuta na dané hladině významnosti (např. 1 %)
  - 5) **přepočítání kontingenčních tabulek**
- **Heuristický prosévací algoritmus** – odstraňuje pravidla, které přispívají klasifikaci
  - počítá se výhodnost pravidla  $r$ :  $\text{výhodnost} = \# \text{vzorů}(R) - \# \text{vzorů}(R - \{r\})$ , odstraní se ty se zápornou a nulovou výhodností → výstup: lokálně optimalizovaná redukována množina pravidel
- **Kombinování více souborů pravidel** – nejjednodušší způsob, jak sloučit více stromů, které řeší stejnou úlohu je převést na soubory pravidel a postupně slučovat tyto soubory

# Stavový prostor a neinformované prohledávání, stromová expanze, náhodné prohledávání, prohledávání do hloubky a do šířky

BI-ZUM

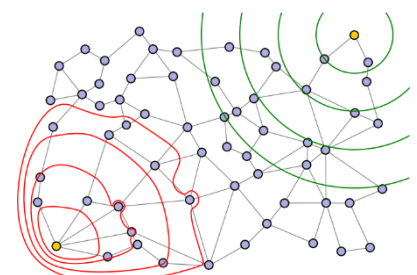
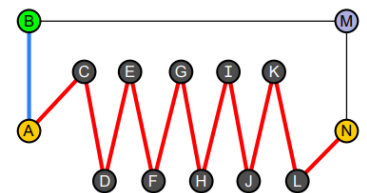
- **Stavový prostor** = orientovaný graf, který lze prohledávat
  - o Uzly = stavy  $S$  (popis stavu řešeného problému)
  - o Hrany = akce  $A$  (umožňují přechod mezi stavy)
- **Prohledávání stavového prostoru**  $(S, A)$  – úloha je zadána počátečním stavem  $I \in S$  a množinou koncových stavů  $G \subseteq S$ 
  - o Následníci stavu  $s$  – takové stavy  $s'$ , pro které platí  $(s, s') \in A$  (lze aplikovat akci ze stavu  $s$  vedoucí do stavu  $s'$ )  $\rightarrow$  množina následníků –  $\Gamma(s)$
  - o Cesta ve stavovém prostoru – orientovaná cesta z  $s_1$  do  $s_n$  je posloupnost akcí  $(s_1, a_1, \dots, a_{n-1}, s_n)$  taková, že platí  $\forall i \in \{1, 2, \dots, n-1\}: a_i = (s_i, s_{i+1}) \wedge s_{i+1} \in \Gamma(s_i)$
  - o Problém popsáný stavovým prostorem se převádí na
    - Hledání cesty z počátečního stavu do koncového stavu – chci celou cestu
    - Hledání cílového stavu – cesta je mi jedno
- **Prohledávací strom** – Uvažujme stavový prostor  $X = (S, A)$ , počáteční stav stavem  $I \in S$  a množinu koncových stavů  $G \subseteq S$ . Prohledávací strom  $X$  je orientovaný kořenový strom s kořenem  $I$  takový, že každá cesta v tomto stromě se vyskytuje i v grafu  $(S, A)$ .
- **Prohledávání stromovou expanzí**
  - o Vytvoř kořen stromu – počáteční uzel  $I$
  - o Dokud žádný list není koncový stav:
    - Vyber nějaký list a expanduj
      - Připoj k listu všechny jeho následníky z  $(S, A)$
  - o Většina algoritmů zakazuje, aby se jeden uzel vyskytl ve stromě vícekrát, proto se rozlišují **3 stavy uzlů**
    - FRESH – uzel nebyl během hledání nalezen
    - OPEN – uzel byl nalezen, ale ne expandován
    - CLOSED – uzel byl nalezen a expandován
- Základní algoritmy pro neinformované prohledávání grafu
  - o **Random search** – triviální algoritmus, který v každém kroku volí náhodný list pro expanzi stromu
    - v praxi nepoužitelný a může nalézt neoptimální řešení
  - o **BFS** – prohledávání do šířky – expanze listů systematicky po patrech, zaručuje nejkratší cestu
    - Nároky na paměť a čas exponenciálně rostou s délkou nejkr. cesty k cíli – v praxi nepoužitelný
  - o **DFS** – prohledávání do hloubky – expanduje strom systematicky, snaha o co největší zanoření
    - Typická implementace je expanze prvního nejhlubšího listu
    - Vhodný pro problémy, kde se nedá příliš bloudit, a přitom je vhodné se co nejrychleji vzdálit od počátečního stavu (např. pro N dam)
  - o **Algoritmus rekonstrukce trasy** – společný pro všechny algoritmy
    - Pracuje s vytvořenou tabulkou předchůdců, kde si pro každý stav uložíme jeho předchůdce
    - Začíná rekonstruovat od koncového stavu/uzlu
  - o Všechny tyto algoritmy jsou neinformované (zbytečně expandují irelevantní uzly) a neberou v potaz cenu prováděných akcí



# Heuristické prohledávání stavového prostoru, heuristiky pro odhad ceny cesty, hladové prohledávání, algoritmus A\*.

BI-ZUM

- **Stavový prostor s ohodnocenými akcemi** – Necht'  $(S, A)$  je stavový prostor, kde  $S$  je množina stavů a  $A$  je množina akcí. Trojici  $(S, A, c)$ , kde  $c$  je funkce  $A \rightarrow \mathbb{R}_0^+$ , nazýváme stavový prostor s ohodnocenými akcemi. Hodnota  $c(a)$  přiřazuje **cenu** každé akci  $a \in A$
- **Cena cesty** = součet cen akcí nacházejících se na této cestě
- Algoritmy pro informované prohledávání pracují s expanzí stavů – přidávají do open množiny ještě neobjevené sousedy.
- **Dijkstrův algoritmus** – hledá nejlevnější cestu z počátečního uzlu
  - o Každý uzel si pamatuje cenu cesty k němu
  - o Bez záporných cen, optimální řešení v čase  $O(|A| + |S| \cdot \log(|S|))$
  - o při expanzi se vybírá ten list, který má nejlevnější délku cesty a pokud se při ní stane, že se expanduje uzel ze stavu OPEN, pro kterou je nalezená kratší cesta, je tento uzel přemístěn do odpovídající větve a cena je upravena (**relaxace**)  $s^* \in \arg \min_{s \in OPEN} g(s)$
  - o Jakmile se v listu objeví koncový uzel, máme zajištěno, že cesta je nejkratší možná
- **Informované prohledávání SP** – kdybychom měli odhad přibližné ceny cesty z jednotlivých open uzlů do cíle, mohli bychom uzly expandovat inteligentněji
- **Heuristika** – Necht'  $(S, A, c)$  je stavový prostor s ohodnocenými akcemi a  $G \subseteq S$  je množina stavů pro úlohu prohledávání tohoto prostoru. Heuristika (heuristická funkce) je libovolná funkce  $h: S \rightarrow \mathbb{R}_0^+$  taková, že  $h(s)$  udává odhad ceny cesty stavu  $s \in S$  k nejbližšímu  $s_g \in G$  a pro všechna  $s_g \in G$  platí  $h(s_g) = 0$ .
  - o **Optimální heuristika**  $h^*$  - vrací neomylnou, přímou cestu – vrací skutečnou cenu (často nedosažitelné)
  - o **Přípustná heuristika**  $h - \forall s \in S: h(s) \leq h^*(s)$ 
    - Optimistická – nevrátí větší odhad, než je skutečná cena (jinak by např. A\* nefungoval)
  - o **Monotónní heuristika** (konzistentní) -  $\forall (x, y) \in A: h(x) - c(x, y) \leq h(y)$ 
    - Přejdem do dalšího stavu by se heuristický odhad délky zbývající cesty neměl snížit o více než o reálnou cenu této akce
  - o **Dominující heuristika** –  $h_1$  dominuje  $h_2$ , když  $\forall s \in S: h_1(s) \geq h_2(s)$ 
    - $h^*$  dominuje všechny ostatní přípustné  $h$
- **Best-first search algoritmy** – podle heuristického kritéria volí k expanzi nejslibnější uzel z OPEN
  - o **Hladové prohledávání** (greedy search) – v každém kroku vybírá k expanzi stav s minimální hodnotou heuristické funkce  $s^* \in \arg \min_{s \in OPEN} h(s)$ 
    - Nalezená cesta nemusí být nejkratší, pochybná rychlost (závisí na heuristice)
    - Minimalizace délky cesty zbývající k cíli
  - o **A\*** – optimální Dijkstra a Greedy a. – součet funkcí
    - Expanduje uzly, které mají nejmenší součet funkcí (nejkratší cesta + nejkratší teoretická cesta – přípustná heuristika)  $s^* \in \arg \min_{s \in OPEN} (g(s) + h(s))$
    - „paralelní větve“ směřující k cíli
    - pokud je heuristika monotónní, nebo přípustná, najde optimální řešení (když není monotónní, tak pouze pokud nepoužívá množinu CLOSED)
- Minimum:  $\min f(x) = x_-$  takové, že  $\forall x' \in X: f(x_-) \leq f(x')$
- Argument minima:  $\arg \min f(x) = \{x_- \in X \mid f(x_-) = \min f(x)\}$
- Porovnávání heuristik – dominance, výpočetní náročnost





# Princip metod hill-climbing a tabu search. Simulované žíhání

BI-ZUM

## - Algoritmy iterativní optimalizace

- o Známa vnitřní struktura problému
- o Postupné zlepšování kandidujícího koncového stavu
- o řeší obecný optimalizační problém, znalosti o struktuře problému zjišťují až během běhu
- o metodou pokus-omyl

## - Optimalizační problém

### Optimalizační problém: Zjednodušená definice

Nechť  $X$  je libovolná množina, kterou budeme nazývat **množina přípustných řešení**, a  $f$  je zobrazení  $f: X \rightarrow \mathbb{R}$ , které nazýváme **kriteriální funkce**.

**Optimalizační problém** je pak formulován jako hledání  $\mathbf{x}^* \in X$  maximalizujícího funkční hodnotu  $f(\mathbf{x})$ :

$$\mathbf{x}^* \in \arg \max_{\mathbf{x} \in X} f(\mathbf{x})$$

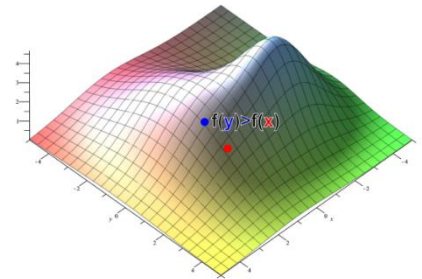
Zapisujeme jako:

$$\underset{\mathbf{x} \in X}{\text{maximize}} \quad f(\mathbf{x})$$

- o Lineární regrese (kurzy sázek), maximalizace x minimalizace (hill climbing x gradient descent)
- o Nezáleží, jestli maximalizujeme nebo minimalizujeme, výsledek je určen stejně
- o Obecný optimalizační problém obtížně řešitelný, ale existují speciální třídy, pro které jsou známy rychlé algoritmy (např. nejmenší čtverce, lineární programování, konvexní optimalizační problém)

## - Hill climbing

- o Vrací lokální maximum, postup směrem největšího růstu (*steepest ascent*)
- o Končí, pokud dosáhl vrcholu
- o Výstup na Mt. Everest v mlze s amnézií (nezajímá ho, co leží za následujícím krokem)
- o Hladové lokální prohledávání – může vést k uváznutí v lokálním maximu, v hřebenu či plošině
- o Prokletí dimenzionality – s rostoucí dim. Roste objem prostoru exponenciálně, není možné navzorkovat okolí bodu s dostatečnou hustotou
- o **Varianty HC:**
  - Stochastic HC – výběr náhodného vyššího souseda
  - First-choice HC – náhodně generuje následníky, dokud nevygeneruje lepšího
  - Random-restart HC – náhodně generované počáteční stavy, dokud nedosáhneme cíle jako počátečního stavu

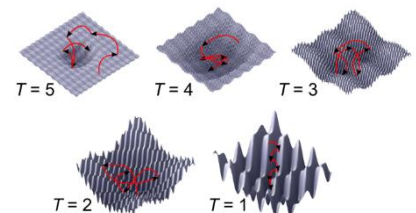


## - Simulované žíhání

- o Rozšiřuje klasický HC o parametr  $t$  (teplotu) – pst, že následník bude vybrán, i když je horší (čím horší, tím menší pst)
- o Pokud je následník lepší, je vždy vybrán, jinak vybrán na základě *míry zhoršení* a *teploty*

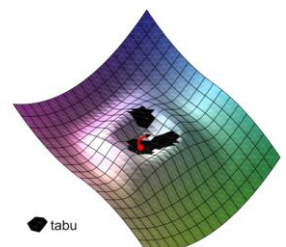
Typická pravděpodobnostní funkce:

$$P(f_{\text{curr}}, f_{\text{new}}, t) = e^{\frac{f_{\text{new}} - f_{\text{curr}}}{t}}$$



## - Tabu search

- o Snaha zabránit oscilaci a přinutit optimalizační algoritmus vymanit se z lokálního optima
- o **Tabu list** – popisuje části prostoru, kam se kandidující řešení nesmí vrátit
  - Většinou se vychází z podobnosti – Euklidovská, kosinová pro  $\mathbb{R}$ , Hammingova pro binární vektory...
- o Pokud se algoritmus vyšplhal na vrchol (lokální optimum), je nucen zahájit sestup a najít další

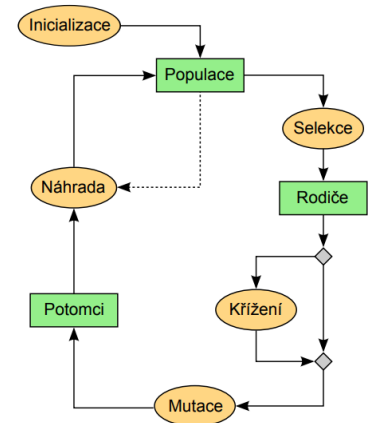


# Evoluční výpočetní techniky

BI-ZUM

## - Evoluční výpočetní techniky

- Rodina metod stochastické populační iterativní optimalizace
- Inspirováno evoluční biologií
- Fungují na principu šlechtění populace kandidujících řešení
- Kvalitní řešení jsou vybrána selekcí k reprodukci, kde může probíhat křížení a mutace
- Kompromisem mezi **exploitací** (zkoumání bezprostředního okolí) a **explorací** (náhodné procházky prostorem bránící v uváznutí v lokálním optimu)



## - Genotyp – reprezentace řešení

## - Fitness funkce – míra adaptace kandidujícího řešení na dané prostředí, je maximalizována

## - Jedinec – označení pro kandidující řešení, dvojice (genotyp, fitness)

## - Populace – množina šlechtěných jedinců

## - Generace – čítač hlavních cyklů evolučních algoritmů (0. - inicializační)

## - Genetické operátory:

- **Inicializace** – vytvoření počáteční populace
- **Selekce** – výběr z jedinců na základě populace, výsledná množina = rodiče
- **Reprodukce** – proces tvorby potomků, probíhá křížením (vzájemná výměna informací mezi jedinci) a mutací (drobná změna genotypu jedince, důležité udržovat ji dynamickou, nejdříve velkou, pak malou, jinak předčasně konverguje), výsledná množina je označována jako potomstvo
- **Náhrada** – náhrada všech nebo části jedinců v původní populaci potomky

## - Genetický algoritmus – každý jedinec je řetězec nad abecedou (binární, ACGT, ...)

- navržen jako univerzální black-box solver optimalizující binární řetězce

### ○ Inicializace

#### ▪ Informovaná

- Vychýlení počáteční populace směrem ke „slibným“ oblastem stavového prostoru
  - Pomocí jednoduché heuristiky
  - Recyklace jedinců z předchozích běhů algoritmu
- Hrozí nebezpečí, že nenávrtně umístí celou populaci do lokálně optimální oblasti, z níž nepůjde vyváznout mutací ani křížením

#### ▪ Neinformovaná – nejčastěji používaná, univerzální

- Vygeneruje populaci zcela náhodných binárních vektorů – pro každý bit hod mincí

### ○ Selektce

- **Ruletová** – pravděpodobnost výběru jedince je přímo úměrná fitness

$$P_i = \frac{f_i}{\sum_{j=1}^{\mu} f_j}$$

- $p_i$  = pravděpodobnost výběru
- $\mu$  = počet jedinců v populaci
- $i$  = daný chromozom
- $f_i$  = fitness hodnota chromozomu  $i$
- Pravděpodobnostní rozdělení  $P = \{p_1, \dots, p_{\mu}\}$  je rozdělení, ze kterého vybíráme rodiče pro křížení

- **Turnajová** – náhodné vylosování  $k$  jedinců a výběr toho s nejlepším fitness

- Není závislá na konkrétních hodnotách fitness

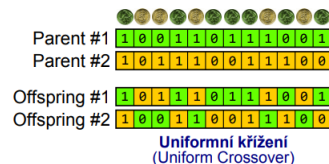
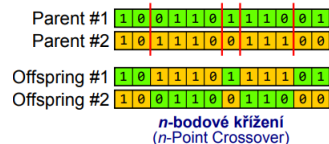
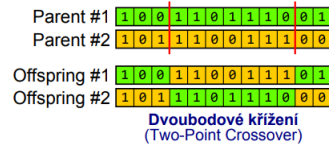
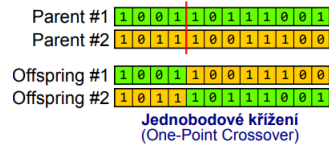
### ○ Mutace

- Nejčastěji bit-flip mutace, kde se s pravděpodobností mutace invertují jednotlivé bity

- Mutation rate – určuje, jak často budou mít potomci náhodné mutace

## ○ Křížení

- Crossover point – bod, ve kterém se rodičovské řetězce rozdělí a části se zkříží, aby zformovaly potomka
- Někdy se zcela vynechává, šlechtění pak probíhá pouze mutací
- Případají v úvahu i křížící operátory specializované pro daný typ problému
- Nejčastější operátory:

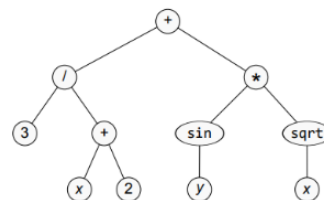
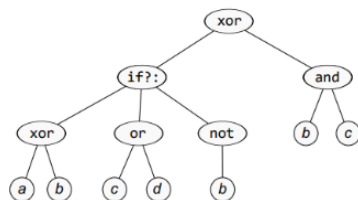


## ○ Elitismus

- V nové generaci zahrnutí některých nejlepších rodičů z předchozí generace – garance růstu fitness funkce
- Culling – všichni jedinci pod limitem jsou odhozeni

## - Genetické programování – Jedinec je program

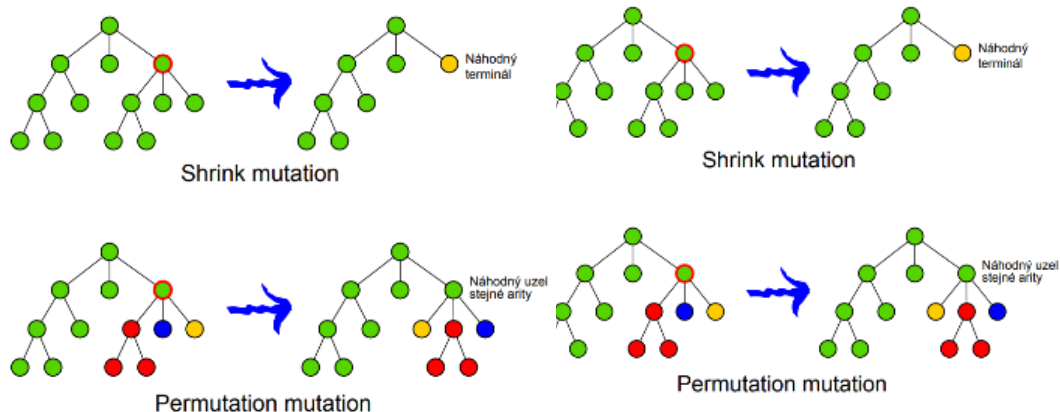
- Genotyp – orientované kořenové stromy
- Flexibilnější (struktura a velikost řešení jsou rovněž předmětem evoluce)
- Listy – terminály, nezávislé proměnné
- Vnitřní uzly – funkce (aritmetické operace, logické funkce atd.)



## ○ Inicializace

- vygenerování náhodného stromu
- GROW: větve stromu mají náhodnou hloubku, v nejnižší vrstvě jsou uzly vybrány z terminálů, v ostatních náhodně z funkcí i terminálů
- FULL: větve mají stejnou hloubku, poslední vrstva je z terminálů, ostatní z funkcí
- PCT1: náhodně generovaný strom se střední hodnotou počtu uzlů

## ○ Mutace

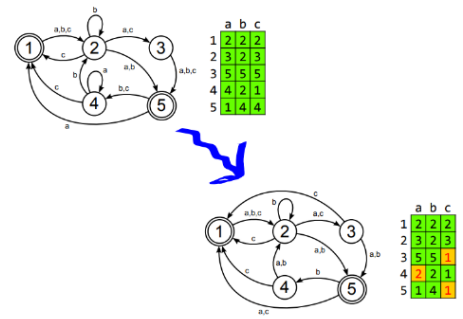


## ○ Křížení – nejčastěji se vymění 2 podstromy rodičů



## - Evoluční programování

- Chování agentů bylo vyjádřeno **stavovými automaty**
  - Stavový automat popsán počátečním a cílovým stavem, množinou stavů a tabulkou přechodů
- Používá operátory mutace, chybí operátor křížení
- Mutaci může podléhat počáteční a cílový stav, množina stavů a tabulka přechodů



## - Evoluční strategie

- Genotyp: vektory reálných čísel (např. float váhy se v ANN blbě kódují jako binární řetězce)
- **Selekce**
  - $(1 + \lambda)$ -ES: 1 rodič vyprodukuje  $\lambda$  potomků, nejlepší je rodičem celé následující generace, odpovídá hill-climbingu
  - $(\mu + \lambda)$ -ES: z  $\mu + \lambda$  potomků je vybráno  $\mu$  nejlepších
  - $(\mu, \lambda)$ -ES: do další generace vybráno  $\mu$  nejlepších potomků
- **Mutace**
  - Malá změna jednotlivých složek  $n$ -rozměrného vektoru
  - **Gaussovská** – nejčastější, přičtení náhodného čísla z normálního rozdělení
  - **Endogenní strategické parametry** – jedinec ukládá i vektor rozptylů gaussovských rozdělení pro jednotlivé dimenze, které se mění v závislosti na úspěšnosti mutace (20 %) = forma metaevoluce
  - Míra mutace
    - Příliš velká – poškozuje užitečnou informaci v genotypech
    - Příliš malá – brání náhodnému zlepšování řešení
    - Dynamická – řízená úspěšností (ES) nebo řízená časem, např. v kombinaci s principem simulovaného žití (v počátečních generacích velké mutace, v pozdějších malé)
- **Křížení**
  - Zásadně uniformní (po složkách)
  - **Diskrétní** – daná složka vektoru se překopíruje vždy od jednoho z rodičů
  - **Aritmetické** – průměr z hodnot rodičů
  - Další parametr  $\rho$ , který určuje počet rodičů, kteří se podílí na tvorbě potomka
    - ▶  $\rho = 1$  ... standardní  $(\mu + \lambda)$ -ES,
    - ▶  $\rho = 2$  ... křížení podobné GA – 2 rodiče,
    - ▶  $\rho = \mu$  ... extrémní případ, kdy se na tvorbě potomka podílí celá populace
- Problém **předčasné konvergence** – celá populace začne být stejná, ovládaná několika typy jedinců
  - Mutace není schopná to zlepšit, musela by být příliš velká
  - Křížení nefunguje, jedinci jsou příliš rozdílní
  - Odpovídá uváznutí v lokálním optimu
- **Niching** – metody, které berou v potaz podobnost genotypů – nepodobní jedinci se nekříží nebo nesoupeří
  - **Ostrovní model** – oddělené populace na vlastních ostrovech
    - Malá výměna informací mezi ostrovy
    - Každý ostrov sleduje vlastní evoluční trajektorii ve stavovém prostoru
  - **Fitness sharing** – podobní jedinci sdílejí fitness hodnotu
  - **Novelty search** – evoluční výhodu mají jedinci přicházející s novými využitími pro existující fenotyp
    - Inspirace biologickou fitness
    - Niching mezi fenotypy
  - **Deterministic crowding** – potomci soutěží o přežití s podobnějším z rodičů, výhra podle fitness
    - Mechanismus brání přemnožení příliš dobrého typu jedince
    - Potomek s vyšší fitness, než rodič jej nahradí, potomek s nižší fitness, než rodič přežije

# Návrh inteligentního agenta, typy agentů, vlastnosti a požadavky na agenty.

## Multiagentní systémy, organizace multiagentního systému (kooperace, kompetice, koordinace, vyjednávání, komunikace).

BI-ZUM

- **Agent** – cokoliv, co je schopno vnímat prostředí pomocí senzorů a jedná v něm pomocí efektorů
  - o Agentní funkce (popisuje chování agenta) → agentní program (fyzická implementace agentní funkce)
  - o Agent = architektura + program
    - Program implementuje agentní funkci, která mapuje vjemy na akce
    - Architektura – přijímá vjemy od senzorů, posílá je programu a na základě jeho odpovědi jedná pomocí efektorů
  - o Počítačový systém umístěn v prostředí, kde je schopen samostatné činnosti, aby dosáhl cíle
    - Klíčová vlastnost – autonomie, účelné činnosti
- **PEAS** – prostředí úlohy – musí být při návrhu agenta definováno
  - o P – performance measure – metrika úspěšnosti
  - o E – environment – prostředí
  - o A – actuators – efekторы
  - o S – sensors – senzory
- **Vlastnosti prostředí:**
  - o Plně pozorovatelné x částečně p. – jestli pořád zná celkový stav prostředí
  - o Deterministické x stochastické – v det. není neurčitost
  - o Epizodální x sekvenční – v ep. Je agentovo chování rozděleno na epizody, které nezávisí na akcích v předešlé
  - o Statické x dynamické – v dyn. Se prostředí může v průběhu měnit
  - o Diskrétní x spojitě – diskrétní má konečný jasně daný počet stavů
  - o Single-agent x multi-agent
- **Multiagentní systém** – prostředí, ve kterém se pohybuje několik nezávislých agentů najednou
  - o Distribuované řešení úloh
  - o Každý z agentů:
    - Vnímá okolní prostředí
    - Jedná (provádí akce) tak, aby dosáhl svých cílů
    - Interaguje s ostatními agenty (kooperace a koordinace akcí)
  - o aplikační oblasti – distribuované systémy, vysoce dyn. prostředí, nekooperativní rozvržení
- **Typy agentů v multiagentním systému**
  - o ZNS
    - Reaktivní – reaguje na podněty z okolí a má předem danou množinu akcí, které může provádět, vybere tu nejvhodnější, nepoužívá paměť, nenabírá zkušenost
    - Intencionální – zvažuje své možnosti, jak dosáhnout cíle, je motivován k co nejlepší reakci, má paměť a agenti tohoto typu se navzájem informují
    - Sociální – má k dispozici modely chování ostatních agentů a ty může využívat ke svému rozhodování
    - Hybridní – může vykazovat vlastnosti ostatních typů dle situace, univerzální
  - o ZUM
    - Izolovaní – ekvivalent single-agentního systému
    - Kooperativní – jednají společně za účelem dosažení týmového cíle
    - Self-interested – maximalizuje vlastní dobro, zvažuje ostatní agenty
    - Hybridní – kombinace výše uvedených

- **Typy agentů z hlediska agentních funkcí**
  - o Jednoduchý reflexní agent – aktuální vjemy, if-then
  - o Reflexní agent se stavem – paměť, historie vjemů
  - o Goal-based agent – akce voleny s ohledem na cíle pomocí plánování
  - o Utility-based agent – zavádí utilitní funkci mapující stav prostředí na reálné číslo
- **Vlastnosti a požadavky na agenty**
  - o **Autonomní** – pracuje bez zásahu z vnějšku, do jisté míry řídí své akce
  - o **Sociální** – interaguje s ostatními agenty (kooperace, koordinace, vyjednávání)
  - o **Reaktivní** – vnímá okolí a reaguje na změny
  - o **Proaktivní** – sám se iniciativně pouští do řešení problémů
  - o **Mobilní** – působit aspoň virtuálně na různých místech. Umístění v konkrétním místě prostředí je považováno za důležitou vlastnost.
  - o **Korektní** – neposkytovat vědomě nepravdivé informace
  - o **Benevolentní** – snažit se udělat to, o co je žádán
  - o **Racionální** – jednat tak, aby dosáhl svých cílů
  - o další důležité vlastnosti – *vysílání a přijímání informací ostatních agentů, provádění akcí, generování cílů a plánů...*
- Pro realizaci se využívá AI (machine learning, fuzzy, teorie her, genetické programování...)
- **Architektura multiagentního systému**
  - o Množina agentů + komunikační síť – fyzická struktura spolu s vlastnostmi a požadavky kladenými na jednotlivé části
  - o Požadavky:
    - **Pokrytí** – je třeba zajistit, aby každé podúlože byla přidělena skupina řešících agentů
    - **Spojení** – aby agenti byli schopni si mezi sebou předávat informace (kooperace + předávání si dílčích výsledků)
    - **Výpočetní potenciál** – dostatečné množství agentů daný problém zvládnout
- **Organizace multiagentního systému**
  - o **Kooperace** – schopnosti jednotlivých agentů vzájemně spolupracovat (sdělování dílčích výsledků, plánů, získávání informací)
  - o **Kompetice** (soutěživost mezi agenty) – ostatní agenti jsou bráni za konkurenty. Snaha maximalizovat svůj vlastní zisk.
  - o **Koordinace** (součinnost kooperujících agentů) – zvýšit pravděpodobnost a rychlosti dosažení stanovených cílů. Koordinace realizuje kooperaci. Prostředky: závazek, smlouva, kostra plánů a řešení úlohy
  - o **Vyjednávání** – cílem je odstranit rozpory mezi agenty, vyvarovat se konfliktům, maximálně využít dostupné zdroje apod. Probíhá na základě vyjednávacího protokolu. Kooperující i kompetiční agenti. Lingvistické, rozhodovací a procesní vyjednávání
    - **Protokol spolupráce** – rekurzivní rozdělování práce mezi agenty
  - o **Komunikace** – způsob předávání informací mezi agenty. Je prostředkem zejména pro kooperaci agentů

# Hry v normální formě. Analýza akčních profilů: Paretoovo optimum, Nashovo equilibrium

BI-ZUM

- **Teorie her** – matematická disciplína studující komplikované situace, kdy jsou utilitní funkce agentů ve vzájemné interakci

- o Kategorie

- **Kooperativní** – modelovací jednotkou je tým hráčů
- **Nekooperativní** – modelovací jednotkou je jeden hráč

- o Aparáty:

- **Normální forma** – hra vyjádřená maticí
- **Extenzivní forma** – hra vyjádřená stromem

- **Hry v normální formě**

- o **Konečná hra v NF pro n hráčů** je trojice  $G = (N, A, u)$ , kde:

- $N$  = konečná množina hráčů
- $A = A_1 \times \dots \times A_n$ , kde  $A_i$  je množina akcí, které má k dispozici hráč  $N_i$ , a  $z A$  je akční profil vyjadřující jednu konkrétní volbu akcí provedenou nezávisle **všemi hráči** ve hře
- $u = (u_1, \dots, u_n)$  je posloupnost utilitních funkcí pro jednotlivé hráče
  - utilitní funkce udává výhodnost/nevýhodnost dané situace pro konkrétního hráče

- o **Common-payoff hra** – hra  $G$ , pro kterou platí, že  $\forall a \in A: u_1(a) = u_2(a) = \dots = u_n(a)$

- tzn. utilitní funkce pro jeden akční profil je pro všechny hráče stejný
- např. po které straně silnice jezdit (viz obrázek)

- o **Constant-sum hra** –  $G$ , pro kterou platí  $\exists c \in \mathbb{R}: \forall a \in A: \sum(u_i(a)) = c$

- tzn. součet utilitních funkcí pro všechny akční profily jsou rovny konstantě  $c$
- pokud  $c = 0$ , pak je to **zero-sum** hra
- např. kámen-nůžky-papír (viz obrázek)

		$N_2$	
		L	R
$N_1$	L	1, 1	0, 0
	R	0, 0	1, 1

		$N_2$		
		K	N	P
$N_1$	K	0, 0	1, -1	-1, 1
	N	-1, 1	0, 0	1, -1
	P	1, -1	-1, 1	0, 0

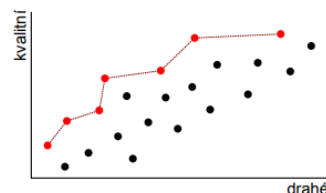
- **Akční profily**

- o Hry popsané v normální formě poskytují pohled shora:

- Hráči jsou si rovnocenní
- Nenahlížíme na ně pohledem “hráč 1 má vyhrát, hráč 2 prohrát”
- Jde o hledání akčních profilů optimálních z jistých globálních hledisek

- o **Paretoovo optimum** – hledání kompromisu mezi vzájemně si odporujícími kritérii (hledání nejvíc dominujících prvků v nějakém kritériu, jako ve skyline)

- Změnou akčního profilu již žádný hráč nemůže zvýšit svou utilitu, aniž by se snížila utilita jiných hráčů
- Např. nákup levných, kvalitních výrobků
- Graf: Červené jsou paretoovsky optimální, protože neexistuje kvalitnější výrobek o stejné nebo nižší ceně a neexistuje levnější výrobek o stejné nebo vyšší kvalitě. Černé nejsou paretoovsky optimální.



- **Paretoovská dominance** – Uvažujme hru v normální formě  $(N, A, u)$ . Řekneme, že akční profil  $a' = (a'_{N_1} \dots a'_{N_n}) \in A$  paretoovsky dominuje akční profil  $a = (a_{N_1} \dots a_{N_n}) \in A$ , jestliže jsou splněny podmínky:

- $\forall i \in \{1 \dots n\}: u_i(a') \geq u_i(a)$
- $\exists i \in \{1 \dots n\}: u_i(a') > u_i(a)$

- Akční profil  $a \in A$  je **paretoovsky optimální**, jestliže neexistuje žádný akční profil  $a' \in A$ , který jej paretoovsky dominuje

		$N_2$			
		A	B	C	D
$N_1$	E	6, 3	8, 2	8, 3	7, 1
	F	3, 2	4, 5	6, 4	6, 5
	G	4, 5	0, 8	5, 7	6, 1

○ Nashovo equilibrium

- Hry v normální formě předpokládají omezenou pozorovatelnost
- Kdyby jeden hráč věděl, jaké akce zvolili ostatní hráči, byla by volba optimální akce triviální
- **Best response** – Uvažujme hru v normální formě  $(N, A, u)$ , akční profil  $a = (a_{N_1} \dots a_{N_n})$ , jednoho konkrétního hráče  $N_i \in N$  a jeho utilitní funkci  $u_i$ . Označme

$$a_{-i} = (a_{N_1}, \dots, a_{N_{i-1}}, a_{N_{i+1}}, \dots, a_{N_n})$$

jako redukovaný akční profil definující akce všech hráčů kromě hráče  $N_i$ . Potom best response na  $a_{-i}$  je množina

$$BR(a_{-i}) = \arg \max_{\hat{a}_{N_i} \in A_i} u_i((a_{N_1}, \dots, a_{N_{i-1}}, \hat{a}_{N_i}, a_{N_{i+1}}, \dots, a_{N_n}))$$

- Uvažujme hru v normální formě  $(N, A, u)$  a akční profil  $a = (a_{N_1} \dots a_{N_n})$ . Řekneme, že  $a$  je Nashovo equilibrium, jestliže

$$\forall i \in \{1, \dots, n\}: a_{N_i} \in BR(a_{-i})$$

- Nashovo equilibrium je takový akční profil, kde akce každého hráče představuje best response na akce ostatních hráčů
- Za znalosti akcí zvolených jinými hráči jsou všichni hráči spokojeni s akcí, kterou zvolili
- **Equilibrium** = rovnováha, stabilita, žádný hráč nechce svou strategii měnit
- Oba hráči vybírají svoji akci tak, aby byla utilitní funkce pro jejich konkrétní akční profil co největší. Třetí sloupec nemůže mít NE, protože kdyby  $N_2$  vybral C,  $N_1$  by pro sebe vybral jako nejlepší E, to by se ale nelíbilo  $N_2$ , protože pro něj by byla nejvýhodnější akce  $N_1$  G, tudíž by svůj tah chtěl změnit, což nesplňuje podmínky Nashova equilibria.

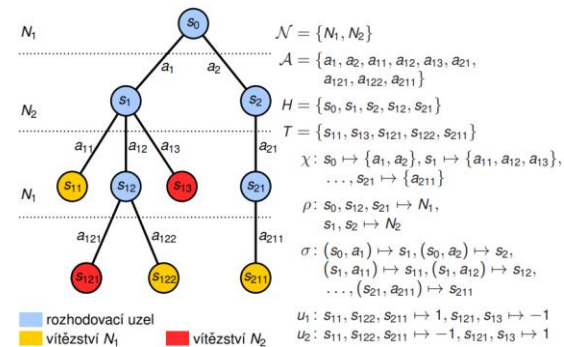
		$N_2$			
$N_1$		A	B	C	D
	E	6, 3	8, 2	8, 3	9, 8
	F	7, 9	4, 5	6, 4	6, 5
	G	4, 5	9, 9	5, 7	6, 1

# Prohledávání herního stromu: Algoritmus Minimax, alfa-beta prořezávání a heuristiky

BI-ZUM

## - Konečná hra v extenzivní formě pro $n$ hráčů – osmice $(N, A, H, T, \chi, \rho, \sigma, u)$

- $N$  = konečná množina hráčů
- $A$  = množina akcí
- $H$  = množina rozhodovacích uzlů
- $\chi \rightarrow 2^A$  popisuje, které akce jsou v daném uzlu k dispozici
- $\rho: H \rightarrow N$  říká, který hráč je v daném uzlu na tahu
- $T$  = množina terminálních uzlů (listů stromu),  $T \cap H = \emptyset$
- $\sigma$  = prostá funkce následnictví  $\sigma: H \times A \rightarrow H \cup T$ 
  - $\forall h_1, h_2 \in H \forall a_1, a_2 \in A: \sigma(h_1, a_1) = \sigma(h_2, a_2) \Rightarrow (h_1 = h_2 \wedge a_1 = a_2)$
  - je prostá  $\rightarrow$  graf má tvar stromu  $\rightarrow$  herní strom



## - Dvouhráčová zero-sum hra v extenzivní formě – hra v extenzivní formě $(N, A, H, T, \chi, \rho, \sigma, u)$ , kde:

- $|N| = 2$
- $u = (u_1, u_2)$
- $\forall t \in T: u_1(t) + u_2(t) = 0$ 
  - V terminálu  $t$  nastává:
    - $u_1(t) = 1, u_2(t) = -1 \rightarrow$  výhra  $N_1$
    - $u_1(t) = -1, u_2(t) = 1 \rightarrow$  výhra  $N_2$
    - $u_1(t) = 0, u_2(t) = 0 \rightarrow$  remíza

## - Hráči MIN a MAX – uvažujeme zero-sum hru dvou hráčů, pak lze utilitní funkce obou hráčů nahradit jednou funkcí $u: T \rightarrow \mathbb{R}$

- MAX:** snaží se maximalizovat  $u$ , na tahu v kořeni stromu
- MIN:** snaží se minimalizovat  $u$  – „soupeř“

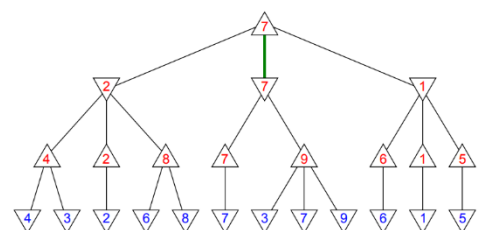
## - Perfektní hra – hráč v každém tahu volí bezchybnou akci

- Bezchybné akce** – MAX zahrál akci  $a^*$  maximalizující zaručenou minimální hodnotu  $u$  v terminálu podstromu při pesimistickém očekávání reakce hráče MIN (např. hráč může hru vyhrát a zvolil akci, která k výhře směřuje)
- Chybné akce** – hráč zahrál suboptimální akci vzhledem k pesimistickému odhadu reakce soupeře a jeho pozice se zhoršila (např. mohl zaručeně vyhrát, ale zvolil akci, kdy to dovolil soupeři ještě zvrátit)
- Nezbytné využití aproximací a heuristik
- Prohledávání stromu s omezenou hloubkou – v hloubce  $d$  ohodnotíme uzly heuristickou evaluační funkcí

## - Algoritmus Minimax – průchodem do hloubky generuje kompletní herní strom (do hloubky $d$ ) z počátečního uzlu $x_0$ (MAX)

- Umožňuje zvolit optimální akci v kořenu stromu vzhledem k ohodnoceným listům
- Předpokládá perfektní hru ze strany soupeře
- Časová složitost  $O(b^d)$ ,  $b$  = průměrný větvicí faktor,  $d$  = hloubka stromu
- Algoritmus:**

- Vygeneruj strom z kořene do hloubky  $d$
- Pokud narazíš na terminál, ohodnoť ho reálnou utilitní funkcí
- Pro neterminály v hloubce  $d$  spočítej heuristickou evaluaci uzlu
- pokud je rozhodovací uzel MAX, vyber maximální hodnotu utilitních funkcí synů
- Pokud je rozhodovací uzel MIN, vyber minimální hodnotu utilitních funkcí synů
- Po návratu do kořenu vidíš z jakého syna přišla největší hodnota a tím směrem se vydej (protože kořen je MAX uzel)



- Problém – omezení hloubkou  $d$ . Ta je většinou malá, protože průměrný větvicí faktor např. v šachách je 35, množství možných stavů roste exponenciálně. Proto je nutné Minimax optimalizovat. K tomu se nejčastěji používá *alfa-beta prořezávání*.
- **Alfa-beta prořezávání** – optimalizace Minimaxu
  - Algoritmus v každém expandovaném uzlu uchovává dvě hodnoty –  $\alpha$  a  $\beta$
  - $\alpha$  – největší utilita, kterou hráč MIN zaručeně nesníží při perfektní hře hráče MAX
  - $\beta$  – nejmenší utilita, kterou hráč MAX zaručeně nezvýší při perfektní hře hráče MIN
  - Tyto hodnoty se aktualizují v průchodu stromem, opět nejprve od listů, poté rodičům
  - Pokud dojde k  $\beta \leq \alpha$ , tak se ostatní větve stromu proříznou a neprohledávají se další potomci, rodičům se updatují  $\alpha$  a  $\beta$  hodnoty (počátečně nastavené na  $-\infty$ / $+\infty$ )
  - Časová složitost  $O(b^{\frac{3}{4} * d})$ , optimálně až  $O(b^{\frac{1}{2} * d})$  – záleží na pořadí uzlů
  - Zlepšení lze dosáhnout kvalitními heuristikami
    - **Killer heuristics** – jestliže daný tah způsobil prořez někde jinde ve stromu, prioritizuj tento tah
  - **Evaluační funkce** – minimax s a-b volí optimální akci v kořenu vzhledem k ohodnocení listů -> nutná rychlost, spolehlivost
    - Omezená hloubka – problém horizontu – co když v hloubce  $n + 1$  dojde ke zvratu v můj prospěch, ale já přestanu v hloubce  $n$ ?
    - Quiescence search – expanzi stromu končíme pouze ve stavech, po nichž bezprostředně nehrozí „divoké výměny“ figur
- Alternativy – neuronové sítě -> deep learning

