Napredno strojno učenje – teorija

# Uvod

## Podatkovne množice in modeli

Spremenljivke:

* Urejena -terica napovednih spremenljivk z zalogami vrednosti je razsežnost množice
* Ciljna spremenljivka z zalogo vrednosti
* Želimo si napovedati vrednost na osnovi podanih vrednosti

Primer :

* je vrednost napovedne spremenljivke , torej
* je vrednost napovedne spremenljivke, torej

Podatkovna množica je množica primerov

Napovedni model je funkcija :

* Za podane vrednosti napovednih spremenljivk
* Izračuna napoved vrednosti ciljne spremenljivke , torej

Dva tipa napovednih modelov:

* Regresijski napoveduje vrednosti numerične ciljne spremenljivke
* Klasifikacijski napoveduje vrednosti diskretne ciljne spremenljivke

Primer: kartica zvestobe

Slika, ki vsebuje besede miza

Opis je samodejno ustvarjen

Naloga nadzorovanega učenja:

* Za podano podatkovno množico najdi čim boljši (točen in splošen) napovedni model
* Za podan nabor spremenljivk in označimo:
  + Prostor podatkov, potenčna množica
  + Prostor modelov, množica funkcij
* Algoritem nadzorovanega učenja je preslikava iz v , ki za podano množico išče najbolj točen in najbolj splošen model – kako merimo točnost in splošnost napovednega modela?

## Točnost in splošnost modelov

Funkcija izgube izračuna napako ene napovedi glede na opazovano vrednost

Izguba oz. napaka natanko takrat, ko , sicer pa je izguba oz. napaka pozitivna

Običajna funkcija izgube za regresijske modele:

* je kratica za Squared Error, kvadratna napaka oz. izguba
* Manj pogosto uporabljena alternativa je – izogibamo se jo zaradi težav z odvedljivostjo

Običajna funkcija izgube za klasifikacijske modele:

* Izguba
* Uporabljamo jo, ko je množica neurejenih diskretnih vrednosti

Napovedna napaka modela na podatkovni množici :

* Je povprečna izguba modela na primerih iz
* Funkcija izračuna napako podanega modela na podatkovni množici , kjer sta prostor podatkov in prostor modelov

Za podano podatkovno množico iz podatkovnega prostora torej poskušamo najti:

* Točen model z minimalno napako
* Splošen model , ki ima točne napovedi za

Splošnost modela:

* Merimo jo na podatkih iz testne množice
* Učna in testna množica sta tuji (disjunktni), torej
* Učno množico, v izogibanju zmedi, pogosto označimo s

Algoritem strojnega učenja je funkcija , ki na osnovi podane učne množice izračuna model za napovedovanje vrednosti ciljne spremenljivke iz podanih vrednosti napovednih spremenljivk , t.j.

## Analiza podatkov

Ocenjevanje napovedne zmogljivosti algoritma je funkcija , ki vrne oceno napovedne zmogljivosti algoritma na množici , kjer so nastavitve hiper-parametrov algoritma

Funkcija ni dobra, ker vodi k preprileganju, saj optimistično oceni napako modela, zato uporabimo prečno preverjanje ali zankanje za oceno na testnih primerih, torej npr.

Algoritmi za strojno učenje:

* Linearni modeli: linearna in logistična regresija
* Metoda najbližjih sosedov
* Odločitvena drevesa in pravila: algoritem TDIDT
* Metode podpornih vektorjev in jedrne funkcije
* Umetne nevronske mreže, vzvratno razširjanje napake
* Ansambli: vrečenje, pospeševanje in naključni gozdovi

## Neobstoj brezplačnega kosila

Boolove ciljne funkcije, ,

* Vseh možnih ciljnih funkcij je

Izberemo algoritma in , brez škode za splošnost, ker je izbira arbitrarna

* vedno napoveduje 1, razen, če ga naučimo drugače
* vedno napoveduje 0, razen, če ga naučimo drugače

Primer: Boolova ciljna funkcija in komplementarna funkcija

Slika, ki vsebuje besede miza

Opis je samodejno ustvarjenSlika, ki vsebuje besede besedilo, miza

Opis je samodejno ustvarjen

Za poljubni model torej velja , zato tudi   
 za poljubno (Boolovo) ciljno funkcijo in njen komplement in za poljubna modela oz. algoritma

Ker je izbor in poljuben, smo dokazali NBK

Zakon o ohranitvi posploševalne zmogljivosti algoritma: vsota zmogljivosti vsakega algoritma strojnega učenja je čez vse možne ciljne funkcije nespremenljiva

Izjave oblike je bolj zmogljiv od oz. ima manjšo pričakovano napako od morajo sloneti na predpostavki o ciljni funkciji ali predpostavki o apriorni in posteriorni porazdelitvi, in

Za teorijo (razvoj algoritmov) strojnega učenja ne obstaja superiorni algoritem strojnega učenja. Večina teoretičnih rezultatov o možnosti učenja je negativnih

Ne glede na popularnost ali teoretično podprtost algoritma za strojno učenje, lahko najdemo ciljno funkcijo, za katero bo njegova napaka velika (in zmogljivost majhna)

Ekspertiza omejena na en razred algoritmov, čeprav zelo močnih, ne zadostuje za uspešno napovedno modeliranje

Izkušnje z uporabo širokega nabora algoritmov so zelo pomembne pri reševanju novega problema

# Meta učenje

Meta učenje je učenje iz izkušenj strojnega učenja na prejšnjih podatkovnih množicah

Izkušnje posplošimo v model za:

* Napovedovanje: kateri algoritem naj uporabimo na podani množici?
* Napovedovanje: katere nastavitve algoritma naj uporabimo?
* Pojasnjevanje: kateri algoritem deluje kje oz. kdaj?
  + Vprašanje 4W: What Works Where and When

Vprašanje 4W ni smiselno, če bi obstajal univerzalno superioren algoritem, ki pa po izreku o neobstoju brezplačnega kosila (NBK) ne obstaja

Notacija:

* je funkcija izgube 0-1
* je učna množica primerov je ciljna funkcija
* je množica možnih modelov učnega algoritma
* je posteriorna porazdelitev modelov , rezultat učnega algoritma na podatkovni množici
* je množica možnih ciljnih funkcij
* je posteriorna porazdelitev ciljnih funkcij

Napaka algoritma :

Za poljuben algoritem velja , kjer je superioren algoritem strojnega učenja

Ocena napake algoritma : pričakovana testna napaka

* Vsota čez vse možne pare (model , ciljna funkcija ) in vse možne testne primere
* K vsoti prispevajo le napačno razvrščeni primeri , ker je za pravilno razvrščene primere
* je apriorna porazdelitev vhodnega prostora
* Pozor: poznati moramo posteriorno porazdelitev ciljnih funkcij!

Predpostavka: deterministični učni algoritem

* Algoritem vrne le en model
* Velja in , zato uporabimo poenostavitev formule, kjer namesto uporabljamo
* je posteriorna porazdelitev vrednosti ciljne spremenljivke

Predpostavka: znana ciljna funkcija

Več možnih formulacij izreka NBK:

* Osnovna: za poljuben par algoritmov in velja
* Alternative: vsoto pogosto zamenjamo s povprečjem ali pa s povprečjem čez vse apriorne porazdelitve

## Posledice NBK

Zakon o ohranitvi posploševalne zmogljivosti algoritma: vsota zmogljivosti vsakega algoritma strojnega učenja je čez vse možne ciljne funkcije nespremenljiva

Izjave oblike je bolj zmogljiv od oz. ima manjšo pričakovano napako od morajo sloneti na predpostavki o ciljni funkciji ali predpostavki o apriorni in posteriorni porazdelitvi, in

Za teorijo (razvoj algoritmov) strojnega učenja ne obstaja superiorni algoritem strojnega učenja. Večina teoretičnih rezultatov o možnosti učenja je negativnih

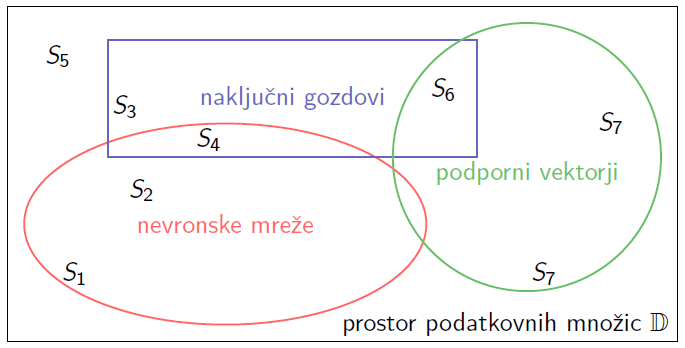
Ne glede na popularnost ali teoretično podprtost algoritma za strojno učenje, lahko najdemo ciljno funkcijo, za katero bo njegova napaka velika (in zmogljivost majhna)

Ekspertiza omejena na en razred algoritmov, čeprav zelo močnih, ne zadostuje za uspešno napovedno modeliranje

Izkušnje z uporabo širokega nabora algoritmov so zelo pomembne pri reševanju novega problema

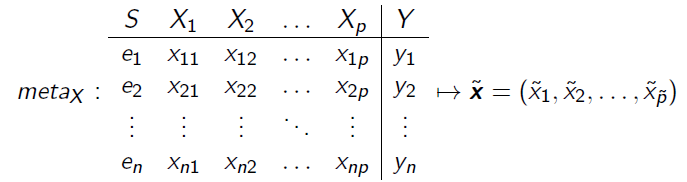
Šibka in močna predpostavka strojnega učenja:

* Proces nastajanja problemov strojnega učenja ustvarja neenakomerno porazdelitev ciljnih funkcij čez
* Porazdelitev čez je znana vsaj v obliki uporabnega približka

Predpostavka meta učenja: algoritmi imajo ''področja ekspertize'' v prostoru ciljnih funkcij   


## Splošen okvir za meta učenje

Meta atributi :

* Od podatkovne množice do meta atributov ,
* 
* Vrednost vsakega meta atributa je izračunan z eno izmed funkcij predstavljenih v zadnjem delu poglavja

Meta ciljna spremenljivka :

* Od podatkovne množice do meta ciljne spremenljivke
* Primer ciljne spremenljivke: , kjer je način vrednotenja zmogljivosti (npr. 10-kratno prečno preverjanje) in izbrani algoritem strojnega učenja
* V tem primeru je numerična,

Podatki za izračun funkcije :

* Opazovanja zmogljivosti algoritmov na podatkovnih množicah
* Slika, ki vsebuje besede miza

  Opis je samodejno ustvarjen
* Algoritmi strojnega učenja
* Podatkovne množice
* Metode vrednotenja zmogljivosti

Običajna vira podatkov za :

* Sistematični poskusi za izračun tabele zgoraj
  + Za vsak par izračunamo vrednost
  + Računsko zahtevna naloga: ogromno procesorskega časa
* Izpolnimo tabelo s podatki iz spletnega repozitorija
  + Npr. iz repozitorija OpenML
  + Prilagodimo množici in razpoložljivim podatkom

Meta model :

* Učenje meta modela: izberemo (meta) učni algoritem ustrezen cilju
* Napovedovanje z meta modelom, : za podano novo/testno podatkovno množico izračunamo najprej , nato pa uporabimo meta model

Okvir za meta učenje: peterica

* : nabor funkcij za izračun meta atributov
* : funkcija za izračun meta ciljne/ciljnih spremenljivke/spremenljivk
* : meta podatkovna množica primerov
* : (meta) algoritem strojnega učenja za gradnjo meta modela
* : meta model,

Z vsako novo podatkovno množico ažiriramo in

Shematski prikaz meta učenja:

* Učenje meta modela:
* Napovedovanje z meta modelom:

Nekaj zaželenih lastnosti :

1. Univerzalnost, agnosticizem: uporabna za poljubno podatkovno množico
2. Fleksibilnost: število meta atributov prilagodljivo velikosti
3. Skalabilnost, razširljivost: ne sme biti (preveč) računsko zahtevna
4. Visoka korelacija z meta ciljno/ciljnimi spremenljivko/spremenljivkami

### Meta ciljne spremenljivke in naloge meta učenja

Meta primeri: podatkovne množice , kjer je (neskončni) prostor možnih podatkovnih množic

Meta spremenljivke:

* Meta ciljne spremenljivke, : različne naloge meta učenja
* Meta vhodne spremenljivke, : vektorski opis podatkovnih množic

Napovedovanje zmogljivosti algoritma :

* Meta ciljna spremenljivka
  + Algoritem z znanimi (privzetimi) nastavitvami nad-parametrov
  + je način vrednotenja zmogljivosti, npr. 10-kratno prečno preverjanje
  + , regresijska naloga nadzorovanega učenja
* Meta model je regresijski,
  + Za podano podatkovno množico model napove
  + Običajno uporabimo več modelov, po enega za vsak algoritem

Izbira najbolj zmogljivega algoritma iz množice :

* Meta ciljna spremenljivka
  + Najbolj zmogljiv algoritem za
  + je način vrednotenja zmogljivosti, npr. 10-kratno prečno preverjanje
  + , klasifikacijska naloga nadzorovanega učenja
* Meta model je klasifikacijski,
  + Za podano napove najbolj zmogljiv algoritem
  + Lahko bi uporabili tudi meta modele za napovedovanje zmogljivosti

Priporočanje/rangiranje algoritmov iz množice :

* Meta ciljna spremenljivka
  + Rangiranje algoritmov iz glede na zmogljivost na je permutacija naravnih števil , kjer je pred , če velja
  + je način vrednotenja zmogljivosti, npr. 10-kratno prečno preverjanje
* Meta model je več-ciljni regresijski
  + Za podano napove range algoritmov iz
  + Lahko bi uporabili tudi meta modele za napovedovanje zmogljivosti

Učenje iz prejšnjih modelov naučenih z algoritmom :

* Učenje prenosa, transfer learning:
  + Prejšnje modele uporabimo kot osnovo za učenje modela na novi (podobni) množici
  + lahko nastavimo tako, da bo nov model podoben prejšnjim
  + Umetne nevronske mreže: strukturo in uteži mreže prej naučene na množici/ množicah uporabimo za učenje iz
* Večopravilno učenje, multi-task learning: učenje iz množice podobnih podatkovnih množic, kjer prej naučene modele uporabimo kot pristranskost/predsodek pri učenju novih modelov

Optimalne nastavitve nad-parametrov algoritma :

* Meta ciljna spremenljivka :
  + Optimalna nastavitev nad-parametrov za algoritem na
  + je način vrednotenja zmogljivosti, npr. 10-kratno prečno preverjanje

### Meta vhodne spremenljivke

Meta vhodna spremenljivka (meta atribut) računa neko lastnost podatkovne množice ,

1. Ročno načrtovan/izbran: statistična lastnost podatkovne množice
2. Naučen (vpetje podatkovnih množic): dimenzija prostora vpetja

Omejitev:

* Funkcija mora imeti nizko računsko zahtevnost
* Ne sme presegati zahtevnosti učnega algoritma

#### Ročno načrtovani meta atributi (MA)

Kategorije ročno načrtovanih meta atributov:

1. Osnovne: število primerov, spremenljivk in podobno
2. Statistične: statistike izmerjene na numeričnih spremenljivkah
3. Informacijske: količina informacije v diskretnih spremenljivkah
4. Mere kompleksnosti: izmerjene na podatkovni množici
5. Modelske: opis modela naučenega na podatkovni množici
6. Algoritmične: zmogljivost preprostih algoritmov

Osnovni MA za primere in vhodne spremenljivke:

* Primer podatkovne množice : število primerov v podatkovni množici,
* Vhodne spremenljivke podatkovne množice :
  + Število vhodnih spremenljivk,
  + Število numeričnih spremenljivk,
  + Število diskretnih spremenljivk,
  + Število dvojiških spremenljivk,
  + Število primerov na vhodno spremenljivko,
  + Število vhodnih spremenljivk na primer,

Osnovni MA za ciljno spremenljivko:

* Za regresijske probleme:
  + Značilke porazdelitve vrednosti iz
  + Primeri: povprečje, mediana, standardna deviacija
* Za klasifikacijske probleme:
  + Značilke porazdelitve vrednosti iz
  + : število razredov za razvrščanje
  + : število, delež primerov v večinskem razredu

MA za posamezne (numerične) spremenljivke:

* Lokacijski parametri porazdelitev: povprečje, mediana
* Parametri razpona: minimum, maksimum, kvartali
* Statistike razpona:
* Statistike oblike porazdelitve: kurtosis, skewness
* Število, delež neznanih vrednosti:

MA za dve ali več (numeričnih) spremenljivk:

* Korelacija in kovarianca:
* Število, delež parov visoko-koreliranih atributov:
* Število/delež normalno porazdeljenih atributov, enovzorčni test Kolmogorov-Smirnov:
* Lastne vrednosti kovariančne matrike za numerične atribute:
* Število glavnih komponent, ki pojasni vsaj 95% variance:

MA za numerične in ciljno spremenljivko (klasifikacija):

* Centri gravitacije za razrede
* Centroid primerov iz razreda
* Opazujemo lahko Evklidske razdalje med centri in za

MA za numerične in ciljno spremenljivko (regresija):

* : korelacija/kovarianca s ciljno spremenljivko
* Učinkovitost atributa : število/delež primerov, ki jih moramo izbrisati, da bi bila
* Kolektivna učinkovitost atributov: število preostalih primerov po iterativnem brisanju primerov, ki imajo ostanke večje od 0,1

MA za posamezne (diskretne) spremenljivke:

* : entropija (nečistost)
* : količina informacije

MA za diskretne in ciljno spremenljivko (klasifikacija):

* : entropija ciljne spremenljivke
* : vzajemna informacija
* : skupna entropija
* : lastna dimenzionalnost

MA: dimenzija podatkovne množice:

* : število glavnih komponent, ki pojasni vsaj 95% variance
* Fraktalna dimenzija podatkovne množice
* Notranja dimenzija podatkovne množice

MA iz lastnosti modelov (osnovna ideja):

* Opazujemo lastnosti napovednega modela za izbran algoritem , ne pa neposrednih lastnosti množice
* Pogosto uporabljeni algoritmi: odločitvena drevesa, linearni modeli

MA iz lastnosti odločitvenih dreves:

* Zgradimo odločitveno drevo brez predhodnega ali naknadnega rezanja
* Opazovane lastnosti:
  + Število vseh, notranjih ali končnih vozlišč v drevesu
  + Globina drevesa
  + Povprečna globina končnih vozlišč
  + Zmanjševanje nečistosti v korenskem vozlišču
  + Število učnih primerov v končnih vozliščih
  + Število končnih vozlišč za posamezno vrednost iz (klasifikacija)
  + Število vozlišč v posameznem nivoju drevesa

MA iz lastnosti drugih modelov:

* Metoda podpornih vektorjev: izbrano jedro običajno polinomsko, število podpornih vektorjev
* Linearni modeli: število koeficientov značilno različnih od 0

MA iz orientirjev, landmarks (osnovna ideja):

* Meta spremenljivke
  + Izbor : hitri, preprosti algoritmi (orientirji)
  + Izbor : ocena na učni množici zaradi računske učinkovitosti
* Običajni orientirji:
  + Najbližji sosed: za : metoda najbližjega soseda
  + Linearni model: linearna oz. logistična regresija
  + Odločitvena drevesa: štor (odločitveno drevo z enim notranjim vozliščem), naključni štor (z enim, naključno izbranim notranjim vozliščem)
* Relativni in vzorčni orientirji:
  + Relativni orientir za dva izbrana orientirja oz. algoritma in
  + Vzorčni orientirji:
    - Izbor algoritmov je lahko širši (ne le preprosti)
    - Hitrost zagotovimo tako, da jemljemo majhne vzorce :
    - Zaporedje vzorcev naraščajoče velikosti

## Vpetja (dataset2vec)

### Splošno o vpetjih objektov v vektorski prostor

Vpetje (embedding) je funkcija , kjer je množica objektov, običajno kompleksnega,   
ne-vektorskega tipa in razsežnost vektorskega prostora za vpetje objektov iz

Ključna lastnost funkcije je

* je mera razdalje med objekti iz
* je Evklidska norma v vektorskem prostoru
* Podobni objekti so vpeti v podobne vektorje

Primer: vpetje word2vec

Osnovna opredelitev word2vec:

* Elementi so besede, je pomenska pomembnost besed
* Znani primer

Pot do vpetja word2vec: nalogi napovedovanja

* Osnovna: iz besed v kontekstu napovedati celotno besedo
* Alternativna: iz centralne besede napovedati besede iz konteksta

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjenIz podanega, vhodnega zaporedja besed zgradimo učno množico za NM (dolžina konteksta ): drseče okno dolžine za zajemanje konteksta:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Nevronski mreži za word2vec:

* Slika, ki vsebuje besede diagram

  Opis je samodejno ustvarjenCBOW: atributi , ciljna spremenljivka
* Skip-Gram: atribut , ciljne spremenljivke
* V obeh primerih vpetje besede preberemo iz stanj skritih nevronov

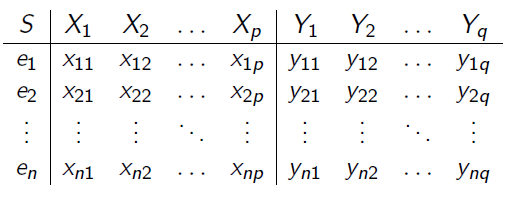
### Podatkovne množice in naloga napovedovanja

V tem primeru torej velja , torej množica je podatkovni prostor

Pomožna meta naloga :

* Napovedujemo podobnost med dvema podatkovnima množicama
* Meta naloga dvojiške klasifikacije
* , če sta množici podobni in sicer

Ideja: podobni množici sta dva naključna vzorca iste podatkovne množice



Naključni vzorci brez ponavljanja za:

* Primere
* Vhodne spremenljivke
* Ciljne spremenljivke , če bi jih bilo

Zgled: gradnja enega primera meta podatkovne množice

Slika, ki vsebuje besede besedilo, pismo

Opis je samodejno ustvarjen

Meta model :

Funkcija izgube za meta primer :

Reševanje naloge napovedovanja:  
Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

### Nevronske mreže za vpetje podatkovnih množic

Struktura nevronske mreže :  
Slika, ki vsebuje besede miza

Opis je samodejno ustvarjen

Struktura nevronskih mrež in :

* Nevronska mreža :
  + Vhodi
  + Modelira korelacijo med in eno ciljno spremenljivko
* Nevronska mreža :
  + Vhod so povprečja izhodov :
  + Modelira povprečne korelacije in interakcije med spremenljivkami
* Obe nevronski mreži sta deepSet, t.j. nevronski mreži za obravnavo množic

Izračun vpetja za podano množico na celotni množici:

Izračun vpetja za podano množico na množici naključnih vzorcev :  
, kjer izračunamo po enaki formuli kot za celotno množico

## Optimalna kofiguracija algoritma (AutoML)

Optimalna kofiguracija algoritma :

Posebna vrsta problema numerične optimizacije

* Časovno (pre)zahtevna ciljna funkcija
* Na voljo imamo omejeno število izračunov ciljne funkcije

### SMBO in osnove Bayesove optimizacije

Zaporedna, na modelih temelječa optimizacija = Sequential Model-Based Optimization (SMBO)

Vhodi:

* Algoritem strojnega učenja
* Podatkovna množica
* Prostor vrednosti nad-parametrov: konfiguracije algoritma
* Metoda za merjenje zmogljivosti
* Algoritem za učenje nadomestnega modela za funkcijo

Izhod: optimalna konfiguracija algoritma za množico

Osnovni algoritem in nadomestna funkcija:  
Slika, ki vsebuje besede besedilo, pismo

Opis je samodejno ustvarjen

#### Ključne komponente SMBO

InitialSample: začetni vzorec konfiguracij za izračun , vrstica 2

* Naključni vzorec konfiguracij (naključno vzorčenje brez omejitev ali vzorčenje po latinski hiper-kocki)

Learn: učenje nadomestnega modela z algoritmom , vrstica 6

* Verjetnostni model za regresijo: napoveduje pričakovano vrednost ciljne spremenljivke in zanesljivost napovedi (interval zaupanja ali odklon )
* Najbolj pogosto uporabljen so Gaussovi procesi

Select: naslednji vzorec konfiguracij za izračun , vrstica 7

* Izboljšava, ki jo prinese konfiguracija : , kjer je trenutna optimalna zmogljivost, ne poznamo, lahko pa jo ocenimo z verjetnostnim modelom , ki vrne oceno pričakovane vrednosti napovedi in odklon
* Pričakovana izboljšava: , kjer ima naključna spremenljivka pričakovano vrednost in varianco


  + Združimo in dobimo
* Predpostavimo, da je : , kjer sta in porazdelitvena funkcija in funkcija gostote za
  + Zamenjamo
  + , kjer je porazdelitvena funkcija za
  + , kjer je funkcija gostote za
* Optimizacijski problem:
  + Ciljna funkcija je nezahtevna za izračun, nadomestni model
  + Uporabimo lahko poljubno optimizacijsko metodo (pogosto uporabimo metodo sistematičnega iskanja z rešetko, grid search)

Terminate: ustavitveni kriterij, vsrtica 8

* Najbolj pogosto je to omejitev uporabljenega procesorskega časa, kar je ekvivalentno omejitvi števila izračunov ciljne funkcije

### Nadgradnja osnovnega SMBO

Stohastične ciljne funkcije: zaporedna optimizacija z regresijo zasnovano na Gaussovih procesih, Sequential Kriging Optimization (SKO) – uporabljene komponente:

* InitialSample: večkratni izračuni za izbrane začetne konfiguracije
* Learn: nadomestni model GP predspostavi šum v ciljni spremenljivki
* Select: optimizacija z Nelder-Mead metodo
* Select: pri izbiri se upošteva napoved plus en stranski odklon
* Select: pričakovana izboljšava je prilagojena tako, da je pristranska do konfiguracij z visoko varianco

Stohastične ciljne funkcije: zaporedna optimizacija parametrov, Sequential Parameter Optimizaction (SPO) – uporabljene komponente:

* InitialSample: večkratni izračuni za vse začetne konfiguracije
* : nadomestni model GP predpostavi odsotnost šuma
* Select: sistematični vzorec za iskanje največje pričakovane izboljšave, izbira vnaprej določenega števila konfiguracij za izračun ciljne funkcije
* Select: skrbi tudi za izbiro konfiguracij za ponovne/večkratne izračune

Več podatkovnih množic: Sequential Model-based Algorithm Configuration (SMAC):

* InitialSample: naključno vzorčenje brez latinskih kvadratov
* Learn: naključni gozd kot verjetnostni model: napovedi posameznih dreves uporabljeni za oceno in
* Select: izbira podatkovnih množic za vrednotenje ob upoštevanju števila prejšnjih izračunov

Več podatkovnih množic: Random Online Aggressive Racing (ROAR):

* Brez modela: naključna izbira konfiguracij za izračun ciljne funkcije
* InitialSample: izbira le ene nakljične konfiguracije
* Select: naključna izbira ene konfiguracije
* Presenetljivo dobri rezultati

Različni tipi parametrov: naključni gozd

* Modeli GP omejeni na numerične parametre
* Learn: uporaba naključnega gozda odpravi to omejitev

Uporaba meta modelov kot:

* InitialSample: kriterij za izbiro začetnih konfiguracij
* Select: dodaten kriterij za izbiro nadaljnih konfiguracij

## Nadomestki

### AutoML z optimizacijo TPE

TPE = Tree-Structured Parzen Estimators

Optimizacija TPE temelji na ideji SMBO z dvema spremembama:

* Drugačni, Parzenovi nadomestki
* Drevesna struktura prostora parametrov

Nadomestki SMBO:

* Za podano konfiguracijo , vektor vrednosti vseh nad-parametrov
* Napovedujejo pričakovano vrednost in standardni odklon ciljne funkcije
* En napovedni model oblike

Nadomestki TPE:

* Napovedni modeli so oblike
* En nadomestni model za vsak nad-parameter iz vektorja

Poseben nadomestek za vsak parameter se odraža v večji fleksibilnosti:

* Nadomestni modeli so bolj enostavni, Parzenovi nadomestki
* Vsak nad-parameter lahko obravnavamo ločeno od ostalih
* Določimo drevesno strukturo odvisnosti med nad-parametri

Nadomestni model TPE za nad-parameter je sestavljen iz dveh modelov:

1. Porazdelitev slabih vrednosti parametra
2. Porazdelitev dobrih vrednosti parametra

Vrednost je prag, ki loči dobre zmogljivosti od slabih. Določimo ga s parametrom , ki določa delež dobrih vrednosti. npr. določa prag tako, da je 10% najnižjih (najboljših) opazovanih vrednosti ciljne funkcije dobrih, ostale so slabe

Izbira naslednje vrednosti :

Ideja najbolj obetavne vrednosti :

* Verjetnost bi morala biti čim večja
* Verjetnost bi morala biti čim manjša

Porazdelitev:

, kjer je

Ker je , velja

Določanje apriornih porazdelitev za različne tipe nad-parametrov:

* Izbira diskretne vrednosti iz seznama dopustnih vrednosti
* Izbira celoštevilčne vrednosti iz intervala
* Izbira numerične vrednosti iz podane porazdelitve
* Porazdelitev je lahko enakomerna, normalna, logaritmična

Drevesna struktura soodvisnosti parametrov:  
Slika, ki vsebuje besede besedilo

Opis je samodejno ustvarjen

### Meta-model za optimizacijo z nadomestki (zakaj v ANG? od slajda 15 naprej)

# Podatkovni tokovi

## Učenje iz podatkovnih tokov

Podatkovni tok (data stream) je podatkovni vir, kjer vsi podatki niso na voljo od začetka, ampak postanejo na voljo sčasoma, običajno po en primer na enkrat

Primer: podatkovni tok

Slika, ki vsebuje besede miza

Opis je samodejno ustvarjen

Lastnosti podatkovnih tokov:

* Podatki prihajajo z visoko frekvenco
* Odziv na podatke je pričakovan v kratkem času (real-time)
* Podatkov je lahko poljubno veliko
* Postopek, ki ustvarja podatke, se lahko spremeni

Učenje iz podatkovnih tokov pogosto uvrščamo med “Big Data” probleme

Omejitve učenja iz podatkovnih tokov:

* Ker je podatkov veliko, vseh ne moremo shraniti – praviloma vsak podatkovni primer obravnavamo enkrat (one-pass) nato pa ga zavržemo ali arhiviramo
* Pri učenju iz podatkovnih tokov nimamo nadzora nad vrstnim redom prihoda primerov
* Zaradi visokih hitrosti prihoda primerov se želimo učiti hitro
* Na podatkovnih tokovih se pogosto učimo na napravah z malo spomina, zato želimo nizko porabo spomina
* Podatki se lahko spremenijo (concept drift), za to moramo spremembe zaznati ter se nanje prilagoditi

Primer: omejitve podatkovnih tokov

Imamo podatkovni tok, katerega primeri so prodajni artikli. Na voljo imamo le par KB spomina. Primerov je toliko, da jih ne moremo shraniti v spomin (npr. nekaj TB). Zanima nas najpogostejših artiklov

Brez dodatnih predpostavk bomo težko shajali. Predpostavimo: pogostost artiklov je neenakomerna

* Nekaj artiklov je izjemno pogostih
* Preostali artikli so zelo redki

Vzdrževali bomo () števcev

* Toliko kot jih lahko shranimo v spomin
* Vsakemu števcu lahko tudi pripišemo enega izmed artiklov

Na začetku imajo vsi števci vrednost 0 in nimajo pripisanih artiklov

Ko dobimo nov primer (artikel):

* Če temu artiklu pripada eden izmed števcev, ga povečamo za 1
* Sicer: če obstaja števec, ki ima vrednost 0, to vrednost ponastavimo na 1 in števcu pripišemo nov artikel
* Sicer: (vsi števci so zasedeni) vse števce zmanjšamo za 1

Deluje kadar in , kjer je število različnih artiklov

Definirali smo Misra-Gries povzetek [Misra& Gries, 1982]. Model lahko v vsakem trenutku uporabimo za izračun najpogostejših artiklov. Model se spreminja in ni popolnoma natančen, računa približke

### Običajno učenje

Pri običajnem učenju se metode učijo iz (končnih) podatkovnih naborov

Vsaka metoda se nauči modelov določene oblike (primer: CART/ID3 naučita odločitvena drevesa)

Metoda torej “sprejme” podatkovni nabor in “vrne” model

Naj bo prostor podatkovnih naborov

Naj bo prostor hipotez metode 𝑀

* Primer: hipoteze CART/ID3 so odločitvena drevesa
* je torej množica vseh odločitvenih dreves

Metoda 𝑀 je tedaj preslikava

### Sprotno učenje

Pri sprotnem učenju to ni dovolj, saj je (vmesnih) modelov več (ko pride nov primer, se model spremeni)

Formalno metodo za sprotno učenje podamo kot par , kjer:

* je začetna hipoteza (model)
* ali je posodobitveni operator

je prostor primerov, je prostor “vreč” primerov (množic s ponovitvami)

Začetna hipoteza je “nenaučen” model

* Misra-Gries: začetna hipoteza – števcev z vrednostmi 0 in brez pripisanih artiklov
* Pri odločitvenih drevesih je začetna hipoteza praviloma prazen list
* Pri nevronskih mrežah je začetna hipoteza naključno inicializirana mreža

Posodobitveni operator sprejme trenutni model in nov(e) primer(e) in vrne nov model

Posodobitveni operator lahko sprejme:

* En primer (model se posodobi z vsakim prihajajočim primerom, instance-incremental)
* Več primerov (model se posodobi, ko se nabere več primerov, batch-incremental)

Model po enem primeru: , po dveh: , itd.

Misra-Gries: posodobitveni operator popravlja števce

V zelo grobem posodobitveni operator deluje na dva načina:

* Inkrementalno: primer ali vrednosti njihovih atributov se shranijo
* Dekrementalno: primer ali vrednosti so izbrisane, “pozabljanje”

Pri Misra-Gries: kadar števcem prištevamo vrednosti se učimo inkrementalno, kadar odštevamo pozabljamo

Nove primere lahko metoda uporabi na več načinov

Nevronske mreže lahko vsak primer direktno uporabijo za posodobitev uteži s pomočjo vzvratnega razširjanja napake (backpropagation)

Drevesa čakajo, da se nabere podpore za delitev lista. Modelna drevesa primere uporabljajo na oba načina:

* Za delitev listov čakajo na statistične dokaze
* Modeli v listih se posodabljajo z vsakim primerom

### Približno učenje

-približno učenje: izračunane vrednosti so znotraj -okolice prave vrednosti,

-približno učenje: izračunane vrednosti so z verjetnostjo v -okolici prave vrednosti,

* Takemu učenju pravimo tudi verjetno približno pravilno učenje (probably approximately correct)

## Računanje statistik na podatkovnih tokovih

Običajno učenje: za računanje statistik so na voljo vsi primeri

* Primer: za računanje povprečja si ogledamo vrednosti vseh primerov, jih seštejemo in delimo s številom primerov

Ko se učimo iz podatkovnih tokov, vseh primerov nimamo na voljo hkrati

Koračno računanje povprečja:

* Denimo, da poznamo povprečje do -tega primera in prispe nov primer
* 1. možnost:
* 2. možnost: hranimo in ga posodobimo z vsakim novim primerom:

Koračno računanje variance:

* Hranimo in posodabljamo in

Koračno računanje korelacijskega koeficienta:

## Vrednotenje modelov na podatkovnih tokovih

Modelom se na podatkovnih tokovih metrike vrednotenja stalno spreminjajo

* Ko je model naučen le na nekaj primerih, je njegova napovedna vrednost slaba
* Ko se model nauči iz dovolj primerov, se njegova napovedna vrednost izboljša

Če napako na celotnem podatkovnem toku utežimo na enak način, dobimo pesimistično oceno napovedne vrednosti modela

Na podatkovnih tokovih zato pogosto uporabljamo bledeče napake (faded error)

Če lahko izbrano napako zapišemo v obliki funkcije izgube , jo preprosto pretvorimo v bledečo napako

Če je običajna napaka definirana kot , je ustrezna bledeča napaka definirana kot , kjer imenujemo faktor bledenja

Primer: bledeča napaka

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Zadržano (holdout) vrednotenje:

* Prihajajoče primere zbiramo v okno (window)
* Ko se okno napolni, na njem izračunamo metrike vrednotenja
* Model naučimo na zbranih primerih v oknu in okno spraznimo
* Imamo okno , napovedi , trenutni model , posodobitveni operator
* Incializacija:
* Za vsak nov primer :
  + Izračunamo napoved in jo dodamo v
  + dodamo v
  + Če je poln:
    - Za vsak : Posodobimo model

Napovedno zaporedno (predictive sequential, prequential) vrednotenje:

* Vsak primer takoj uporabimo za izračun napovedi (metrik vrednotenja)
* Primer takoj podamo modelu za učenje
* Imamo napovedi , trenutni model in posodobitveni operator
* Za vsak nov primer :
  + Izračunamo napoved in jo dodamo v
  + Posodobimo model

## Vrste sprememb

Neformalno: odvisnost ciljnih spremenljivk od vhodnih se s časom spremeni

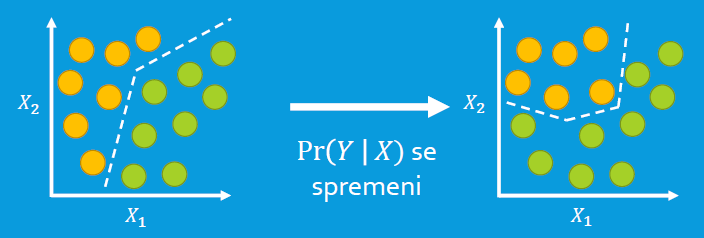
Formalno:

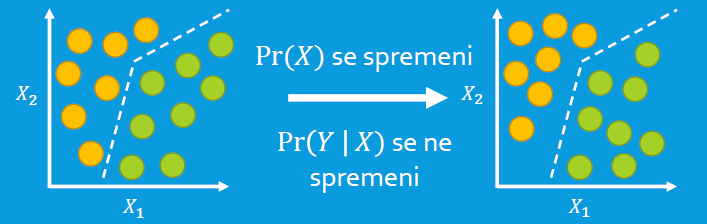
Spremembe klasificiramo glede na 2 kriterija:

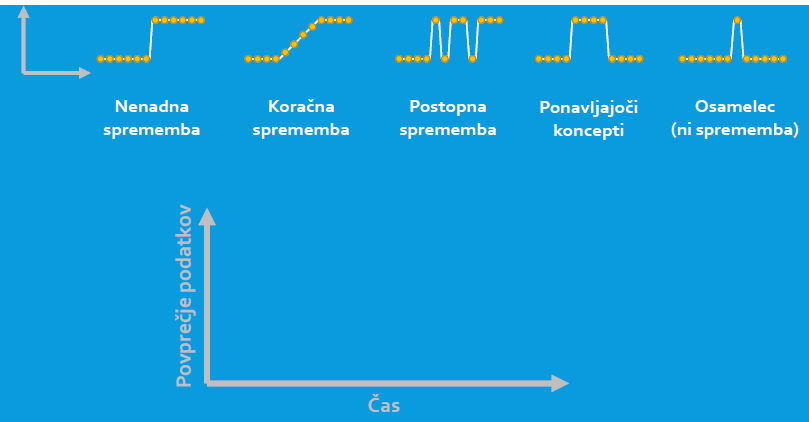
* Tip sprememb (zaradi česa se je spremenila odvisnost)
* Časovni profil sprememb

Spremembe v lahko povzroči:

* – spremeni se porazdelitev v ciljnem prostoru
* – spremeni se odvisnost ciljnih vrednosti od vhodnih
* – spremeni se porazdelitev v vhodnem prostoru

Prave spremembe se nanašajo na spremembe v , spremenjena odvisnost zahteva učenje novih modelov (premik odločitvene meje)  


Navidezne spremembe se nanašajo na spremembi v ali , naučeni modeli modelirajo odvisnost med in , zato jih ni treba posodobiti  


Časovni profili sprememb:  


### Mehanizmi za zaznavanje sprememb

Opazovanje mere ustreznosti (napovedna vrednost, hevristika)

* Page-Hinkleyev test
* Preverjanje ali pogoji za izgradnjo drevesa še držijo (Hoeffdingova neenakost)

Primerjava distribucij na dveh časovnih oknih (referenčno okno in okno nedavnih primerov)

* ADWIN

Page-Hinkleyev test:

* Opazujemo signal , najpogosteje napako
* Definiramo in kot
* Kadar je večja od predpisane , zaznamo spremembo
* Parameter uravnava najmanjšo spremembo, ki jo želimo zaznati. Parameter uravnava, koliko lažnih preplahov dovolimo

## Drevesa na podatkovnih tokovih

Pri običajnem strojnem učenju za učenje dreves najpogosteje uporabljamo TDIDT (top-down induction of decision trees)

TDIDT ne deluje na podatkovnih tokovih (nimamo dostopa do vseh podatkov hkrati za izračun hevristike)

Za gradnjo dreves na podatkovnih tokovih uporabljamo prilagojen pristop

Pri običajnem učenju dreves delitve izbiramo s pomočjo vseh podatkov – na podatkovnih tokovih to ni možno

Ideja: za izbor prave delitve potrebujemo le majhno podmnožico vseh primerov

* Delitev v korenu bo določilo prvih nekaj primerov
* Nadaljnje delitve v listih bodo podmnožice primerov, ki so v liste razvrščene

Začnemo s praznim listom (začetni model )

Ko dobimo nov primer, na dosedanjih primerih\* ovrednotimo delitve

Oglejmo si najbolje ovrednoteno delitev

* Ta delitev je lahko najboljša po naključju (še posebej, če imamo malo primerov)
* Kako vemo kdaj imamo dovolj primerov, da je najboljša delitev res najboljša?
* Pomagamo si s Hoeffdingovo neenakostjo

Hoeffdingova neenakost: naj bodo neodvisne omejene naključne spremenljivke, tj. . Naj bo . Tedaj velja

* Posledica: velja izrazimo z :

Če smo izmerili vrednost spremenljivk , lahko izračunamo njihovo povprečno vrednost , ki jo označimo z . Ker so naključne spremenljivke, verjetno odstopa od , ki jo označimo z . Hoeffdingova neenakost nam pove, da leži v -okolici z verjetnostjo

### Uporaba Hoeffdingove neenakosti

Imamo hevristiko 𝐻, ki jo želimo maksimirati (kot primer bomo uporabili redukcijo variance)

Za vsak atribut poiščemo najboljšo delitev

* Če je atribut nominalen, je to praviloma delitev po vseh vrednostih
* Če je atribut numeričen, preverimo vse smiselne točke delitve

Ali lahko izberemo atribut z maksimalno vrednostjo hevristike?

* NE.
* Izmerjeno se je lahko zgodilo po naključju. Za utemeljitev uporabimo Hoeffdingovo neenakost.

Primer: redukcija variance

Redukcija variance za regresijo:

* je množica primerov zbranih v listu
* in sta množici primerov po delitvi

Pri klasifikaciji lahko uporabimo informacijski prispevek ali Gini indeks

Poznamo redukcije varianc za vse atribute

pripada najboljši delitvi, pripada drugi najboljši delitvi – opazujemo   
 kot slučajno spremenljivko

Ker je razmerje med redukcijo variance najboljše ter druge najboljše delitve, gotovo velja , tj. . Tedaj ustreza pogojem Hoeffdingove neenakosti

z verjetnostjo leži v -okolici , kjer je oz.

Če , je z verjetnostjo tudi . Najboljša delitev je ''res'' boljša od ostalih delitev

Preverjanje delitev je potratno

Za poljuben primer je verjetnost, da bomo list razdelili ravno za njim, majhna

Delitve ovrednotimo le, ko se nabere dovolj primerov

* Ovrednotimo po 200, 400, 600, … nabranih primerih
* Interval preverjanja delitev je parameter učenja dreves na podatkovnih tokovih

### FIMT-DD

Hitra indukcija modelnih dreves (Fast Induction of Model Trees with Drift Detection, FIMT-DD)

Zaznavanje sprememb: Page-Hinkleyev test

Delitev listov: redukcija variance in Hoeffdingova neenakost

V vsakem listu hrani linearen model (perceptron)

* Za napovedovanje ciljne spremenljivke uporablja vrednosti vhodnih atributov

Zaznavanje in prilagajanje spremembam:

* V vsakem vozlišču drevesa opazujemo Page-Hinkleyev test
* Ko vozlišče sporoči spremembo, poiščemo najvišjega prednika, ki jo je še zaznal
* V najdenem vozlišču začnemo izgradnjo novega poddrevesa
* Ko je novo drevo natančnejše, staro zavržemo

Primer: sprememba



Začetni model : prazen list

Posodobitveni operator :

* Primer razvrsti v list (tudi v pomožna poddrevesa)
* Posodobitev statistik: v listu posodobi statistike za vrednotenje delitev
* Zaznavanje sprememb: na podlagi napovedi posodobi Page-Hinkleyev test v listu in njegovih prednikih
* Prilagajanje spremembam:
  + Če je sprožena sprememba, v ustreznem vozlišču ustvari pomožno poddrevo
  + Če je pomožno poddrevo že obstaja in je boljše, zamenja izvirno poddrevo
* Delitev: če je zbranih dovolj primerov, ovrednoti delitve in poizkusi razdeliti list

Parametri Page-Hinkleyevega testa

* – uravnava minimalno spremembo
* – uravnava dovoljeno pojavnost lažnih signalov

Parametri Hoeffdingove neenakosti

* – določa verjetnost za -aproksimacijo

Interval preverjanja delitev

* – posredno določi za -aproksimacijo (Hoeffdingova neenakost)

Številni drugi parametri

## Delitve na numeričnih atributih

Primer: redukcija variance

Za izračun redukcije variance potrebujemo sledeče statistike:

* Število primerov –
* Povprečje –
* Varianca –

Za vrednotenje nominalnih testov (ne glede na obliko testa) potrebujemo statistike za vsako vrednost nominalnega atributa

* Statistike hranimo v tabeli
* Opazimo: statistike za posamezne vrednosti atributov so popolnoma neodvisne

Za vrednotenje numeričnih delitev potrebujemo statistike za vsako možno delitev

* Potrebujemo statistike za in
* Statistike bi lahko hranili tabelarično
* Problem: velikost tabele je treba prilagajati, vstavljanje je počasno
* Ideja: uporabimo iskalno drevo

### EBST

Razširjeno binomsko iskalno drevo (Extended Binary Search Tree, EBST)

Vsako vozlišče pripada eni vrednosti danega atributa

V vsakem vozlišču hranimo (delne) statistike za primere, ki imajo vrednost danega atributa

Statistik za primere z vrednostmi ne hranimo eksplicitno (prihranimo spomin)

Pri vstavljanju novih vrednosti nočemo popravljati tudi “desnih” potomcev (vrednost razvrščamo le enkrat)

* Hranimo delne statistike
* Statistike za in () niso neodvisne
* Primeri so tudi

Ko vrednotimo delitve, vzdržujemo trenutne statistike

* Ko se v drevesu pomikamo v leve potomce, trenutnih statistik ne posodabljamo
* Ko se v drevesu pomaknemo v desnega potomca, trenutnim statistikam prištejemo statistike trenutnega vozlišča

Primer: EBST

Slika, ki vsebuje besede besedilo, križanka

Opis je samodejno ustvarjen

Za vsak numerični atribut hranimo EBST

Za posodobitev EBST z novim primerom z vrednostjo 𝑎moramo v drevesu najti ustrezno vozlišče oz. ga ustvariti

Povprečna časovna zahtevnost vstavljanja je , kjer je velikost drevesa (ker hranimo delne statistike)

Statistike za vrednosti izračunamo tako, da hranimo globalne statistike

Statistike za izračunamo iz globalnih statistik in statistik za :

* Na ta način zmanjšamo porabo pomnilnika

## Prilagajana okna – ADWIN

Prilagajana okna (ADaptive WINdow)

Hranimo najdaljše okno, za katerega velja, da se povprečje znotraj okna ni spremenilo

Del okna zavržemo, kadar je njegovo povprečje dovolj različno od povprečja preostanka okna

Imamo signal , kjer so omejeni

* BŠS: , dosežemo z normalizacijo

Okno primerov označimo z

Ideja: če sta dela okna in ''dovolj različna'', lahko starejši del okna zavržemo

Ko se skrči, zaznamo spremembo. skrčimo, kadar se povprečji in razlikujeta za več kot , kjer je harmonična sredina dolžin in

Omejitev napačno zaznanih sprememb (false positive rate): Če se povprečje v ni spremenilo, je verjetnost da smo okno skrčili manjša od .

Omejitev zgrešenih sprememb (false negative rate): Denimo, da za neko particijo velja . Tedaj ADWIN z verjetnostjo okno skrajša na vsaj (kjer vsebuje novejše primere).

Posledica Hoeffdingove neenakosti: velja izrazimo z :

Priprava za ADWIN:

* za neko delitev
* – pravo in izmerjeno povprečje na
* – pravo in izmerjeno povprečje na
* – pravo in izmerjeno povprečje na

### Omejitev napačno zaznanih sprememb

Če se pravo povprečje ni spremenilo, velja

Za vsak velja:

Hoeffdingova neenakost:

Izberemo pri katerem sta člena enaka (minimira vsoto):

Za ta velja

Če želimo, da , zadostuje oz.

Da dobimo končni rezultat (verjetnost napačno zaznane spremembe ), seštejemo verjetnosti vseh možnih delitev (teh pa je ravno ):

### Omejitev zgrešenih sprememb

Privzemimo

Za vsak :

Hoeffdingova neenakost:

Izberemo kot prej in dobimo

## Učenje iz samovzorcev na podatkovnih tokovih

Uporabljamo več osnovnih modelov, njihove napovedi združujemo

Ideja: če modeli ansambla delajo raznolike napake, z združevanjem njihovih napovedi dobimo boljšo napoved (znižujemo pristranskost)

Raznolikost modelov dosežemo tako, da jih naučimo na različnih (samo)vzorcih podatkovnega nabora

* Samovzorce danega podatkovnega nabora dobimo z vzorčenjem s ponovitvami
* Vsak model se uči iz svojega samovzorca zato je bolj prilagojen na ponovljene primere v vzorcu
* V povprečju je v samovzorcu delež ponovljenih primerov , preostali se pojavijo le enkrat

Samovzorcev ne moremo vzorčiti kot običajno

* Spremenimo zorni kot:
  + Samovzorca ne opazujemo kot množice (nabora)
  + Samovzorec opazujemo primer po primer
* Opazimo: vsak primer se v samo vzorcu pojavi 0-, 1-ali več-krat, do največ 𝑛-krat
* Samovzorec za vsak primer natančno določa kolikokrat se pojavi (“vektor” ponovitev)

Samovzorec dobimo z vzorčenjem s ponovitvami

Število ponovitev danega primera je med 0 in in je porazdeljeno

Verjetnost ponovitev danega primera je

Podatkovni tok lahko opazujemo kot podatkovni nabor, katerega velikost narašča proti

Število ponovitev danega primera: , kjer

* Ker je lahko poljubno visok
* Ta porazdelitev je natanko
* Za vsak primer znamo izračunati koliko ponovitev je v danem samovzorcu

Namesto eksplicitnega računanja samovzorcev obravnavamo vsak primer posamično. Tedaj:

* Za vsak primer in osnovni model vzorčimo
* Osnovni model naučimo s ponovitvami danega primera

### Sprotno učenje iz samovzorcev

Online bagging

Drugačen pogled kot pri običajnem učenju iz samovzorcev

* Samovzorcevne računamo eksplicitno
* Za vsak primer izračunamo, kolikokrat se pojavi za dani osnovni model

Začetna hipoteza : nabor začetnih hipotez osnovnega modela :

Posodobitveni operator (za primer ): za vsak osnovni model s posodobitvenim operatorjem ( šteje člene ansambla, primere):

* Vzorčimo

ADWIN Bagging:

* Kombinacija sprotnega učenja iz samovzorcev in ADWIN
* ADWIN kot mehanizem za zaznavanje sprememb (en ADWIN na osnovni model)
  + Opazujemo napako
* Ker imamo ansambel, ne izgubimo preveč, če zavržemo posamezni model
* Ko za osnovni model zaznamo spremembo, ga zavržemo in začnemo učiti novega

## Na primerih osnovano sprotno učenje

Poseben pomen dajemo primerom

“Leno”učenje – računamo šele, ko smo “vprašani”

Posplošitev metode najbližjih sosedov (kNN)

Potrebujemo razdaljo na vhodnem prostoru

Trenutne izkušnje predstavlja baza primerov (pri običajnem učenju najpogosteje učna množica)

Na primerih osnovano učenje:

* Posodabljanje je poceni (samo dodajamo ali odstranjujemo primere)
* Računanje napovedi je relativno zahtevno
  + Iskanje ustreznih primerov
  + Računanje napovedi iz najdenih primerov

Učenje z modeliranje

* Posodabljanje je potratno
* Računanje napovedi je preprosto

Na primerih osnovano učenje uporabljamo, kadar imamo veliko učnih podatkov, ne potrebujemo pa pogostih napovedi

Na primerih osnovana klasifikacija:

* Napoved praviloma za primer izračunamo z uporabo večinskega glasovanja:
* Posplošimo z uteženostjo primerov glede na njihovo razdaljo od :   
  , kjer in je neka padajoča funkcija (manjša razdalja vodi v večjo utež)

Zahteve za bazo primerov pri sprotnem učenju:

* Časovna ustreznost: novejši primeri so pomembnejši kot starejši.
* Prostorska ustreznost: baza idealno enakomerno pokriva vhodni prostor. Primeri, ki ne vplivajo na napovedane vrednosti, so odveč.
* Konsistentnost: ciljne vrednosti bližnjih primerov naj se ne razlikujejo preveč.

### IBLStreams

Sprotno vzdrževanje baze primerov

Za vsak primer obravnavamo „okolico“ , ki vsebuje najbližjih primerov

Napoved je najpogostejši razred v

Posodabljanje baze parametrov:

* Pri posodabljanju baze primerov ne odstranjujemo najnovejših primerov (časovna ustreznost)
* Pri posodabljanju baze primerov (učenju) pri novem primeru opazujemo testno množico
* Testno množico določa -u najbližjih primerov
* Med najbližjimi primeri določimo večinski razred
* Če , odstranimo primere, za katere ( najnovejših primerov ne odstranjujemo)
* Če z dodajanjem novega primera prekoračimo velikost , zavržemo še najstarejši primer

Pri statičnih tokovih se natančnost povečuje z večjo bazo primerov

Pri spremenljivih tokovih imamo več zastarelih primerov, ki jih moramo odstraniti

* Počasnejše prilagajanje spremembam

IBLStream eksplicitno zaznava nenadne spremembe s pomočjo napovedne napake, njene standardne deviacije in -testa

Neposredno prilagajanje velikosti okolice , tj. števila relevantnih sosedov

* Na vsakem koraku obdržimo ali pa povečamo ali zmanjšamo za 1
* Hranimo povprečne napake čez zadnjih 100 primerov za in
* posodobimo na vrednost z najnižjo napako

Implicitno prilagajanje velikosti okolice z ustrezno utežitveno funkcijo

* Uporaba eksponentnega ali Gaussovega jedra

# Učenje iz kompleksnih podatkovnih tipov

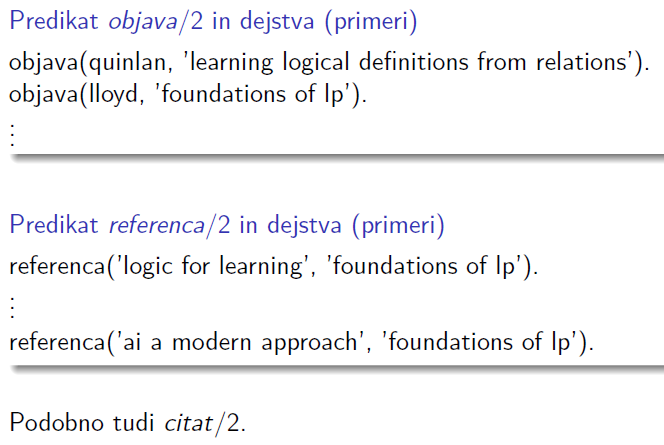
## Predstavitev kompleksnih podatkov

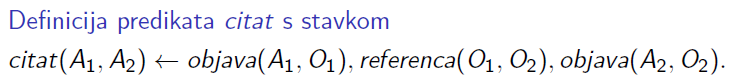
Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Slika, ki vsebuje besede besedilo

Opis je samodejno ustvarjen





Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

## Logika prvega reda, stavčna logika

Razlika od običajnega strojnega učenja: primeri so množice namesto vektorjev oz. -teric, vsebujejo strukturirane izraze (predikate)

Elementi sintakse stavčne logike (clasual logic):

* Izrazi (terms) – ne prirejamo jim resničnostnih vrednosti
* Atomi (atoms) – lahko jim priredimo resničnostno vrednost
* Stavki (clauses)

Izrazi se nanašajo na objekte v opazovani domeni:

* Konstante: nizi znakov z malo začetnico
* Spremenljivke: nizi znakov z veliko začetnico, lahko jim priredimo poljubni izraz kot vrednost
* Strukture ali sestavljeni izrazi (compund terms): funkcija , kjer označuje število argumentov

Popolnoma opredeljen izraz (ground term) se nanaša na en objekt iz opazovane domene. Lahko je konstanta ali sestavljen izraz brez spremenljivk

Predikat , kjer je število argumentov, predstavlja relacijo med izrazi ali pa lastnost izraza

Razlika med atomi (predikati) in (sestavljenimi) izrazi:

* Strukturirani izrazi se nanašajo na objekte v domeni
* Atomi (predikati) se nanašajo na relacije med objekti
* Za atome lahko ugotovimo resničnostno vrednost

Literal je lahko atom ali logična negacija atoma

Stavke tvorimo iz atomov in literalov z implikacijo

* in so atomi oz. literali
* Glava (head) stavka je disjunkcija
* Telo (body) stavka je konjunkcija
* Logični pomen stavka je

Če upoštevamo, da je isto kot

Teorija (tudi logični program) je množica stavkov

Posebni razredi stavkov:

* Dejstva: prazno telo in en atom v glavi,
* Definitni (določeni) stavki: en atom v glavi, poljuben
* Zanikanja ali poizvedbe: prazna glava, poljuben
* Hornovi stavki: največ en atom v glavi, poljuben

Hierarhija tipov stavčne logike:

* Propozicijska logika: izrazi so lahko le resnični ali neresnični: ni spremenljivk
* Relacijska logika: objekte lahko naslavljamo s konstantami ali spremenljivkami
* Polna logika: lahko imamo strukturirane izraze
* Definitna (določena) logika: le definitni (določeni) stavki: ni disjunkcije v glavi stavka

Zamenjava (substitution) preslika spremenljivke v izraze: od neopredeljenih proti popolnoma opredeljenim izrazom (stavki)

Instanca stavka:

* Ko zamenjavo apliciramo na stavek, dobimo njegovo instanco
* Če instanca ne vsebuje spremenljivk, dobimo popolnoma določen stavek (popolnoma določeno instanco)

Vse instance stavka so njegove logične posledice

Herbrandove interpretacije

* Herbrandov univerzum: množica vseh določenih izrazov (konstant)
* Herbrandova baza: množica vseh določenih atomov, t.j. literalov brez spremenljivk
* Herbrandova interpretacija: množica resničnih določenih atomov

Stavek je neresničen za podano interpretacijo, če so vsi literali v telesu resnični in vsi literali v glavi neresnični. Sicer je stavek resničen za podano interpretacijo

Herbrandov model stavka je interpretacija, kjer so vse določene instance stavka resnične

Dva pristopa k učenju v stavčni logiki:

* Neposredno učenje
  + Učenje relacijskih dreves iz predikatov in poizvedb
  + Naštevanje vseh možnih poizvedb
  + Izbira optimalnih poizvedb za napovedovanje
* Propozicionalizacija in agregati
  + Pretvorba kompleksnih podatkov v vektorje fiksne dolžine
  + Pretvorba poizvedb v atribute
  + Dva tipa atributov: Boolovi pogoji in numerični agregati

## Neposredno učenje iz predikatov

### Naštevanje možnih poizvedb

Pokritost :

* je poizvedba, je Herbrandova interpretacija
* Pokritost je enaka uspešnosti poizvedbe

Pokritje :

* Množica vseh Herbrandovih interpretacij, za katere je poizvedba uspešna

je bolj splošna od poizvedbe , če . Rečemo tudi, da je manj specifična od

Robni poizvedbi:

* : pokriva vse Herbrandove interpretacije
* : ne pokriva nobene interpretacije,

Hassejev diagram:

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Operator izostritve :

* Od splošnega k specifičnemu (lahko tudi od specifičnega k splošnemu)

Lastnosti operatorjev izostritve:

* Idealnost: nasledniki so najbolj splošne poizvedbe
* Popolnost: iz najbolj splošne poizvedbe z zaporedjem izostritev pridemo do katerekoli druge poizvedbe vsaj na en način
* Optimalnost: isto kot zgoraj, a na točno en način

-subsuma :

* je zamenjava
* je stavek po zamenjavi
* je množica literalov iz disjunktne oblike
* Ustrezen operator izostritve:

Lastnosti operatorja -subsumpcija:

* Enostaven za implementacijo
  + Stavku dodamo poljuben literal
  + Zamenjavo lahko izvedemo z literalom oblike
* Težave s semantiko
  + , ampak
  + Zato je možno in tudi za
  + Takrat rečemo, da sta in sintaktični varianti istega stavka
  + Posledica sintaktičnih variant so težave z optimalnostjo izostritve

### Izbira optimalne poizvedbe

Funkcija nečistoče meri varianco vrednosti ciljne spremenljivke v množici

Regresija, :

Klasifikacija, : , kjer so verjetnosti

Zaželene lastnosti funkcije nečistoče :

* Doseže maksimalno vrednost pri enakomerni porazdelitvi
* Slika, ki vsebuje besede diagram

  Opis je samodejno ustvarjenDoseže minimalno vrednost pri porazdelitvah, kjer
* Simetrična: neobčutljiva na vrstni red parametrov
* Konkavna, zvezna in zvezno odvedljiva

Dve pogosto uporabljeni funkciji:

* Entropija
* Indeks Gini

Funkcija zmanjšanja nečistoče (Impurity Reduction):

* je množica primerov, ki jih poizvedba pokriva
* je množica primerov, ki jih poizvedba ne pokriva

Test izberemo z optimizacijo ciljne funkcije :

Možne teste naštevamo z operatorjem izostritve

Učenje relacijskih dreves iz množice Herbrandovih interpretacij , pri začetnem klicu je

Slika, ki vsebuje besede besedilo, pismo

Opis je samodejno ustvarjen

## Propozicionalizacija in agregati

### Poizvedbe kot Boolovi atributi

Predikatna relacija poveže nek ciljni objekt z drugimi objekti in lastnostmi teh objektov

Primer predikatne relacije:

Zgornji stavek definira relacijo, ki poveže diagram z obliko enega izmed elementov diagrama

Stavčna definicija relacije ima na desni strani poizvedbo , ki je lahko uspešna ali ne

Relacija je lahko Boolov atribut ciljnih objektov:

Do celotnega nabora možnih atributov pridemo z uporabo operatorja izostritve :

* Tako kot smo ga uporabili za naštevanje testov v drevesih
* Od splošne, oz. , k bolj specifičnim poizvedbam
* Ustavitveni kriterij: poizvedba je enaka za vse ciljne objekte
* Dodatni kriterij: maksimalno število vključenih literalov

Vsako poizvedbo pretvorimo v relacijo in nato v atribute:

* Izračunamo vrednosti na osnovi stavka za

### Poizvedbe kot agregati

Agregate rešitev poizvedbe lahko uporabimo kot numerične atribute

Tabela za strojno učenje je enaka kot prej, le da so atributi sedaj numerični in ustrezajo agregatom

Za numerične lastnosti atributov so možni naslednji agregati:

* Mean: povprečna vrednost numerične lastnosti
* Median: mediana, srednja vrednost numerične lastnosti
* Minimumi, maksimumi, kvantili in podobno

Matej's Alternative: Tests with Aggregates

## Vpetja omrežij

### Heterogena informacijska omrežja (HIO)

HIO je urejena šesterica

* je množica vozlišč v omrežju
* je množica povezav
* je množica tipov vozlišč iz
* je množica tipov povezav iz
* je preslikava, ki določa tipe vozlišč
* je preslikava, ki določa tipe povezav

Tipi povezanih vozlišč določajo tip povezave (in obratno)

Za dve povezavi iz velja natanko takrat, ko

Shema HIO določa tipe vozlišč in povezav v HIO

Graf

* Vozlišča v shemi so tipi vozlišč v HIO
* Povezave v shemi so tipi povezav v HIO

Vpetje: vozliščem v grafu priredimo vektorje fiksne dolžine

### Vpetja vozlišč omrežij/grafov

#### Vpetje word2vec za besede

Vhod je zaporedje besed

Iz zaporedja do podatkovne množice za učenje NM (kontekst ):  
Slika, ki vsebuje besede miza

Opis je samodejno ustvarjen

* Drseče okno velikosti za zajemanje konteksta

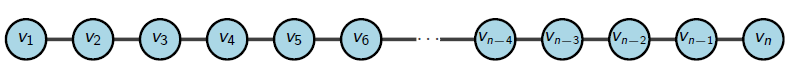
Slika, ki vsebuje besede diagram

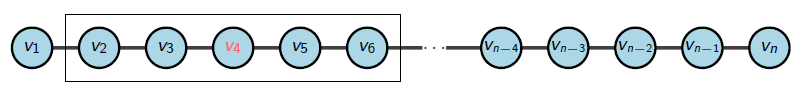
Opis je samodejno ustvarjen

Zajemanje konteksta besede: drseče okno  
Slika, ki vsebuje besede besedilo

Opis je samodejno ustvarjen

#### Od besedila do omrežja

Graf sprehodov velikosti : predstavlja pravzaprav zaporedje vozlišč  


Drseče okno grafov sprehoda:  


* Pravokotnik predstavlja kontekst vozlišča velikosti

Kontekst v grafih sprehodov:  
Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Za podan graf sprehoda velikosti zgradimo podatkovno množico za učenje NM (kontekst ):  
Slika, ki vsebuje besede miza

Opis je samodejno ustvarjen

* Drseče okno velikosti za zajemanje konteksta

Obstoječe metode za vpetje:

* Kodiranje tipa one-hot vozlišč v grafu (koda dolžine )
* CBOW: atributi , cilj
* Skip-Gram: atribut , cilji
* Vektor vpetja preberemo iz stanj nevronov v skriti plasti

Podpora za zelo omejen nabor grafov sprehodov – uporabimo tehniko ekstrakcije grafov sprehodov iz omrežja (naključni sprehod)

#### Naključni sprehod

Splošni naključni sprehod po omrežju je sprehod dolžine iz vozlišča za graf

1. Nastavimo sprehod dolžine 0 na
2. Vzorčimo (enakomerno) naključnega soseda vozlišča ,
3. Dodamo in povezavo grafu , zamenjamo z
4. Ponavljamo 2 in 3, dokler nima vozlišč

Primer: naključni sprehod dolžine 5 iz vozlišča

Slika, ki vsebuje besede besedilo, ura, sličica

Opis je samodejno ustvarjen

Predpostavka splošnega naključnega sprehoda:

* Verjetnosti izbire naslednjega vozlišča so enakomerno porazdeljene med sosedi
* Alternativa: verjetnosti sorazmerne utežem na povezavah

V pristranskem sprehodu so verjetnosti obtežene,

* in sta trenutno in naslednje vozlišče v sprehodu
* je matrika sosednosti (uteži) podanega omrežja

Dve ekstremni pristranskosti:

* Sprehod najprej v širino (konzervativen, lokalna okolica)  
  Slika, ki vsebuje besede besedilo, ura, sličica

  Opis je samodejno ustvarjen
* Sprehod najprej v globino (raziskovalen, izven lokalne okolice)  
  Slika, ki vsebuje besede besedilo, ura

  Opis je samodejno ustvarjen

#### node2vec

Parametrizacija kompromisa med širino in globino:  
Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Predsodek node2vec za naključne sprehode: naključni sprehod druge stopnje, t.j. z dvema parametroma,

* Povratek : visoke vrednosti spodbujajo raziskovanje
* Raziskovanje : nizke vrednosti spodbujajo raziskovanje

Tipične nastavitve za primerjavo:

1. : raziskovalna strategija, makro-vpetje grafa
2. : konzervativna strategija, mikro-vpetje grafa

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Algoritem node2vec:

1. Izračunamo vrednosti na osnovi podanih in
2. Nastavimo množico sprehodov na
3. Za vsako vozlišče opravimo pristranskih sprehodov dolžine
4. Uporabimo izračunane vrednosti za kontrolo predsodka
5. Dodamo naključne sprehode v
6. Uporabimo sprehode z drsečim oknom za podatkovno množico
7. Treniramo nevronsko mrežo iz podatkovne množice

Za vprejte node2vec uporabljamo Skip-Gram nevronsko mrežo

### Vpetje povezav

Na osnovi dejstva, da povezava poveže dve vozlišči, sledi

* Poljubna funkcija
* Najbolj enostaven primer je funkcija stika dveh vektorjev,
* S pomočjo vpetja povezav lahko naslovimo nalogo napovedovanja povezav

Alternativne funkcije :  
Slika, ki vsebuje besede besedilo

Opis je samodejno ustvarjen

### Iz heterogenega v homogeno omrežje

Meta sprehod je sprehod v shemi heterogenega omrežja in ustreza poizvedbi v podatkovnih bazah ali stavčni logiki

Primera meta sprehodov:

-ima--je-kvadrat-je--ima-

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

-ima--(vsebuje--je-trikotnik)-je-trikotnik-je--ima-

Slika, ki vsebuje besede diagram

Opis je samodejno ustvarjen

Do meta sprehodov lahko pridemo z naštevanjem vseh možnih sprehodov - (podobno kot poizvedbe) ali z ročnim sestavljanjem naborov meta sprehodov

# Odkrivanje enačb in simbolna regresija

Definicija naloge: za podani par podatkovne množice prostora matematičnih izrazov iz spremenljivk najdi enačbo oblike , za katero je

Primer: odkrivanje Keplerjevega zakona

Podana podatkovna množica :  
Slika, ki vsebuje besede besedilo, pisava, posnetek zaslona, številka

Opis je samodejno ustvarjen

Najdi enačbo oz.

Simbolna regresija je nelinearna regresija

* Pri linearni regresiji je ciljna enačba oblike
* Pri simbolni regresiji je ciljna oblika enačbe , kjer je poljubne oblike

Včasih je možna ročna transformacija:

* Vpeljemo nove spremenljivke in
* Linearna ciljna enačba med novimi spremenljivkami:

Odkrivanje enačb je posebna vrsta strojnega učenja

* Prostor možnih modelov so enačbe (za razliko od dreves, najbližjih sosedov ali nevronskih mrež)
* Rezultat simbolne regresije naj bi bil bolj razumljiv
* Enačbe so standarden in dobro uveljavljen formalizem v znanosti

Tukaj tudi možna pretvorba:

* Posebne vrste nevronov v umetnih nevronskih mrežah
* Opravljajo aritmetične operacije namesto običajne obtežene vsote

Hevristični pristop Bacon: tri preproste hevristike:

1. Uvajanje nove spremenljivke: če spremenljivka narašča kadarkoli pada, uvedi
2. Uvajanje nove spremenljivke: če spremenljivka narašča kadarkoli narašča, uvedi
3. Ugotavljanje invariante: če ima spremenljivka nizko varianco, zastavi enačbo

Primer: Bacon za odkrivanje tretjega Keplerjevega zakona

Slika, ki vsebuje besede besedilo, pisava, posnetek zaslona, številka

Opis je samodejno ustvarjen

Kadarkoli narašča , narašča tudi : uvedi

Slika, ki vsebuje besede besedilo, pisava, posnetek zaslona, številka

Opis je samodejno ustvarjen

Kadarkoli narašča pada: uvedi

Slika, ki vsebuje besede besedilo, pisava, posnetek zaslona, številka

Opis je samodejno ustvarjen

Kadarkoli narašča pada: uvedi

Slika, ki vsebuje besede besedilo, pisava, posnetek zaslona, številka

Opis je samodejno ustvarjen

Varianca je nizka Ž(manjša od ): vzpostavi enačbo

Dva (prepletena) koraka odkrivanja enačb:

1. Iskanje ustrezne strukture enačbe – problem kombinatorične optimizacije
2. Ocenjevanje vrednosti parametrov – problem numerične optimizacije

Splošni pristop ustvari-in-preizkusi (generate-and-test):  
Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava, algebra

Opis je samodejno ustvarjen

Preizkus strukture : EstimateParameters

* Za podano enačbo in množico z neznanimi vrednostmi -tih parametrov   
   majdi optimalne vrednosti , tako da velja
* izračunamo na primerih iz
* Običajni problem numerične optimizacije: velika izbira algoritmov (npr. Bayesova optimizacija)

Ustvarjanje strukture : Generate – več pristopov:

1. Stohastični evolucijski pristop (običajna, široko uporabljena možnost)
2. Sistematični pristop: naštevanje vseh možnosti
3. Sistematični pristop: operator izostritve

## Evolucijski pristop

Kromosomi za predstavitev objektov:

* Osnovni kromosomi: Boolovi vektorji
* Za enačbe: drevesna predstavitev matematičnih izrazov

Nastanek novih kromosomov:

* Evolucijski operatorji križanja in mutacije
* Izbira kromosomov na osnovi funkcije uspeha, fitness
* Paradigma ''preživijo najuspešnejši'', survival of the fittest
* Funkcija uspeha je pravzaprav ciljna funkcija za optimizacijo

Primer: drevesna predstavitev matematičnih izrazov

Slika, ki vsebuje besede diagram, vrstica, origami

Opis je samodejno ustvarjen

Vloga vozlišč v drevesu:

* Notranja vozlišča ustrezajo aritmetičnim operatorjem (ali funkcijam)
* Končna vozlišča ustrezajo spremenljivkam in konstantnim parametrom

Slika, ki vsebuje besede besedilo, diagram, posnetek zaslona, oblikovanje

Opis je samodejno ustvarjen

Slika, ki vsebuje besede besedilo, diagram, vrstica, oblikovanje

Opis je samodejno ustvarjen

Slika, ki vsebuje besede besedilo, posnetek zaslona, risanka

Opis je samodejno ustvarjen

Slika, ki vsebuje besede besedilo, posnetek zaslona, vrstica, diagram

Opis je samodejno ustvarjen

Primer: uspešnost kromosoma pri odkrivanju enačb

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava, vrstica

Opis je samodejno ustvarjen

Slika, ki vsebuje besede besedilo, pisava, posnetek zaslona, številka

Opis je samodejno ustvarjen

### Evolucijska optimizacija

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

Pomožne funkcije:

* Init: ustvari množico pop.size naključnih kromosomov (dreves)
* Eval: izračuna uspešnost kromosomov v podani učni množici
* ApplyOperator:
  + Naključno izbere enega izmed evolucijskih operatorjev
  + Za mutacijo iz populacije izbere en kromosom, za križanje dva
  + Nad izbranimi kromosomi izvede evolucijski operator
  + Izbira operatorja: parametri algoritma (verjetnost križanja, verjetnost(i) (različnih vrst) mutacije)

Izbiranje kromosomov:

* Popolnoma naključna izbira enega izmed kromosomov
* Ruleta: verjetnost izbire kromosoma proporcionalna uspešnosti
* Turnirska izbira: izbor para, nato boljšega od dveh
* Elitna izbira: le med najboljših kromosomov

Mere kompleksnosti enačbe:

* Število notranjih in/ali končnih vozlišč drevesa
* Število notranjih parametrov
* Dolžina enačbe v karakterjih

Dve možnosti upoštevanja kompleksnosti enačb:

1. Uspešnost izračunamo kot kombinacijo napake in kompleksnosti , kjer je stopnja vpliva kompleksnosti: večji kot je, manj je preprileganja
2. Opazujemo oboje hkrati: več-ciljna optimizacija

## Propozicionalizacija

Vpeljava novih spremenljivk z naborom transformacij podanih

* Z množenjem do določene, omejene stopnje:
* Z apliciranjem funkcij, npr. trigonometričnih ali krožnih
* S kombinacijami enih in drugih transformacij

Rezultat: razširjen nabor spremenljivk

Metoda Lagrange:

* Naštevanje kombinacij (omejenega reda) spremenljivk iz
* Linearna regresija za ocenjevanje parametrov

Metoda Sindy:

* Redka linearna regresija v razširjenem naboru spremenljivk
* Regulacijski člen poskrbi, da je malo parametrov
* je moč regularizacije: večji kot je, manj je preprileganja

## Odkrivanje enačb in predznanje

### Kontekstno-neodvisne gramatike