# Gradivo za državno tekmovanje iz umetne inteligence (DTUI)

Programski odbor izbirnega tekmovanja iz umetne inteligence

## 1 Uvod

To gradivo je namenjeno pripravi na Državno tekmovanje iz umetne inteligence (DTUI). Vsebinsko je prilagojeno obsegu znanja, ki se preverja na tekmovanju, zato ne predstavlja splošnega pregleda umetne inteligence (UI), temveč se osredotoča na izbrane teme, ki jih morajo tekmovalci razumeti za uspešno reševanje nalog.

Gradivo poleg uvoda v UI pokriva štiri osrednja področja: pripravo podatkov, evalvacijo modelov strojnega učenja, spodbujevano učenje (reinforcement learning) ter nadzorovano učenje (supervised learning). Vsa področja predstavljajo predvsem osnovo za teoretični del tekmovanja, medtem ko je znanje nadzorovanega učenja namenjeno tudi pripravi na praktično nalogo. Poudarek je na razumevanju konceptov, razlogov za uporabo posameznih metod, njihovih predpostavk in omejitev ter na sposobnosti uporabe naučenega na novih problemih. Takšen način priprave sledi tudi pristopu mednarodne olimpijade IAIO, kjer se od tekmovalcev pričakuje predvsem razumevanje delovanja metod, prepoznavanje primerov, v katerih odpovejo, ter sklepanje v novih problemih, ne zgolj pomnjenje formul ali postopkov.

Glavni vsebinski vir pri pripravi tega gradiva je priročnik Eljakim Schrijvers: IAIO Training (2026), pripravljen za mednarodno olimpijado iz umetne inteligence (International Artificial Intelligence Olympiad - IAIO). Priročnik obravnava bistveno širši nabor tem, med drugim nadzorovano in nenadzorovano učenje, pripravo podatkov, evalvacijo modelov, iskalne algoritme, teorijo učenja, generativne modele in druga področja umetne inteligence. V tem gradivu so iz njega izbrane in za potrebe DTUI podrobneje predstavljene le teme, ki sodijo v letošnji tekmovalni program. Tekmovalcem zato za pripravo ni treba predelati celotnega priročnika IAIO Training.

Gradivo je vsebinsko prilagojeno po priročniku Eljakim Schrijvers, *IAIO Training* (2026), ki je objavljen pod licenco CC BY-NC 4.0. Posamezne razlage, primeri in poudarki so bili prevedeni oziroma prilagojeni programu DTUI z nekaj dodatnimi razlagami in vsebinami.

**Pomembni elementi**

Pomembni elementi, opozorila in nekateri primeri so v gradivu označeni z okvirji, kot je ta.
Branje zgolj vsebine v okvirjih praviloma ne zadostuje za razumevanje celotne snovi, lahko pa služijo kot pomoč pri hitrem pregledu in ponavljanju ključnih poudarkov.

**Komentarji in povratne informacije**

Če imate komentarje, predloge ali opazite možnosti za izboljšavo gradiva, bomo zelo veseli vaših povratnih informacij. Te se lahko nanašajo tako na vsebino kot tudi na terminologijo, razlage in predstavitev snovi.
Pri slovenskih strokovnih izrazih se, kadar je to mogoče, držimo terminologije iz Terminološkega slovarja umetne inteligence.

Če opazite odstopanje od uveljavljene terminologije ali imate predlog za izraz, ki ga v slovarju še ni, nam to lahko sporočite na elektronski naslov blaz.rolih@fri.uni-lj.si.

# 2 Uvod v umetno inteligenco

Umetna inteligenca (UI, angl. *artificial intelligence*, AI) je področje računalništva, ki se ukvarja z razvojem sistemov, sposobnih izvajati naloge, ki jih običajno povezujemo s človeško inteligenco. Med take naloge sodijo prepoznavanje slik in predmetov, razumevanje in ustvarjanje jezika, učenje iz izkušenj, reševanje problemov ter sprejemanje odločitev in napovedi.

**Priporočilo za branje**

Naslednji del podaja osnovne pojme umetne inteligence in je namenjen predvsem širšemu razumevanju področja. Čeprav je koristen za orientacijo, ni strogo del preverjane snovi za DTUI.
Pri pripravi na tekmovanje namenite večino časa poglavjem, ki podrobneje obravnavajo konkretne vsebine DTUI.

Z umetno inteligenco se danes srečujemo na številnih področjih, na primer pri prepoznavanju slik in govora, priporočanju vsebin, filtriranju neželene elektronske pošte, medicinski diagnostiki in avtonomnih sistemih. Velik del sodobne umetne inteligence temelji na strojnem učenju, pri katerem sistemov ne programiramo z natančnimi pravili za vsak mogoč primer, temveč se ti iz podatkov naučijo uporabnih vzorcev.

Umetna inteligenca pri tem ni nekaj skrivnostnega ali ločenega od običajnega računalništva. V njenem ozadju so matematika, algoritmi in podatki. Matematika daje okvir za učenje vzorcev in sprejemanje odločitev, podatki pa predstavljajo informacije, iz katerih se sistem uči. Zato je kakovost naučenega modela močno odvisna od kakovosti podatkov, ki jih uporabimo pri učenju.

**Definicija: umetna inteligenca**

Umetna inteligenca (UI) je področje računalništva, namenjeno razvoju sistemov, ki lahko izvajajo naloge, običajno povezane s človeško inteligenco. Med take naloge sodijo prepoznavanje slik in predmetov, razumevanje in ustvarjanje govora, sprejemanje odločitev in napovedovanje, reševanje kompleksnih problemov ter učenje iz izkušenj. Velik del sodobnih sistemov umetne inteligence uporablja strojno učenje in se zato uči iz podatkov, vendar učenje iz podatkov ni nujna lastnost vseh pristopov umetne inteligence.

Zgoraj navedena definicija izhaja iz gradiv za IAIO, v strokovni literaturi pa lahko najdemo tudi številne druge opredelitve umetne inteligence. Razlike izhajajo predvsem iz širine področja ter velikega števila pristopov in podpodročij, ki jih umetna inteligenca zajema. Zato naj navedena definicija ne služi kot stroga in dokončna meja področja, temveč predvsem kot izhodišče za njegovo razumevanje.

## 2.1 Nekatere ključne tehnologije umetne intelligence

Umetna inteligenca zajema veliko število različnih tehnologij in podpodročij. V nadaljevanju so predstavljena nekatera izmed njih, ki imajo pomembno vlogo predvsem v sodobnih sistemih umetne inteligence.

### 2.1.1 Strojno učenje

Strojno učenje (angl. *machine learning*, ML) je eno izmed osrednjih področij sodobne umetne inteligence. Njegova osnovna ideja je, da računalniku namesto natančnih pravil za vsak mogoč primer podamo podatke oziroma primere, iz katerih se algoritem nauči uporabnih vzorcev.

Pri klasičnem programiranju programer analizira problem, sestavi algoritem in nato zapiše pravila, ki jih računalnik izvaja. Pri strojnem učenju je postopek drugačen: algoritmu podamo podatke, pri nadzorovanem učenju pa tudi pravilne odgovore, iz katerih algoritem zgradi model, ki opisuje povezavo med vhodi in izhodi.

Ta pristop je posebej uporaben pri nalogah, pri katerih je ustrezna pravila težko zapisati neposredno. Človek na primer brez težav prepozna obraz na fotografiji, zelo težko pa bi sestavil popoln seznam pravil, ki bi zanesljivo opisal, kaj je obraz. Namesto tega lahko algoritmu pokažemo veliko označenih slik, iz katerih se nauči vzorcev, uporabnih za prepoznavanje obrazov.

**Ključna ideja**

Pri klasičnem programiranju pravila določi človek, računalnik pa jih uporabi na podatkih. Pri strojnem učenju algoritem iz podatkov in primerov sam odkrije model oziroma pravila, ki jih nato uporablja na novih podatkih.

### 2.1.2 Globoko učenje

Globoko učenje (angl. *deep learning*) je podpodročje strojnega učenja, ki uporablja nevronske mreže z več plastmi. Izraz *globoko* se nanaša na globino mreže, torej na več zaporednih računskih plasti med vhodom in izhodom.

Nevronske mreže sestavljajo med seboj povezane računske enote oziroma nevroni. Povezavam med njimi pripadajo uteži, ki se med učenjem prilagajajo tako, da mreža na danih vhodih proizvaja čim ustreznejše izhode.

Pomembna lastnost globokih nevronskih mrež je, da lahko različne plasti gradijo vse bolj kompleksne predstavitve podatkov. Pri obdelavi slik lahko zgodnejše plasti na primer zaznavajo preproste lastnosti, kot so robovi in barve, kasnejše plasti pa te informacije združujejo v oblike, teksture in na koncu celotne objekte.

Odnos med pojmi lahko poenostavljeno zapišemo kot:

$$\text{globoko učenje} \subset \text{strojno učenje} \subset \text{umetna inteligenca}. \tag{1}$$

Globoko učenje je osnova številnih sodobnih sistemov za prepoznavanje slik, obdelavo govora, strojno prevajanje ter ustvarjanje besedil in slik.

### 2.1.3 Računalniški vid

Računalniški vid (angl. *computer vision*) je področje umetne inteligence, ki se ukvarja z obdelavo in razumevanjem vizualnih informacij. Sodobni pristopi temeljijo predvsem na rabi prej opisanega globokega učenja.

Digitalna slika je za računalnik v osnovi mreža številskih vrednosti, ki predstavljajo intenzitete oziroma barve posameznih slikovnih elementov. Glavni izziv računalniškega vida je iz teh številskih podatkov pridobiti smiselno informacijo o predmetih in drugih vsebinah na sliki.

Med pogoste naloge računalniškega vida sodijo klasifikacija slik, kjer določimo, kateremu razredu pripada celotna slika, detekcija objektov, kjer na sliki poiščemo in prepoznamo več predmetov, ter segmentacija, kjer določamo pripadnost posameznih slikovnih elementov različnim objektom ali razredom.

Primeri uporabe računalniškega vida so prepoznavanje obrazov na fotografijah, zaznavanje pešcev pri avtonomnih vozilih in samodejno odkrivanje napak na izdelkih v proizvodnji.

### 2.1.4 Obdelava naravnega jezika

Obdelava naravnega jezika (angl. *natural language processing*, NLP) je področje umetne inteligence, ki se ukvarja z razumevanjem in ustvarjanjem človeškega jezika, tako v pisni kot tudi v govorjeni obliki.

Obdelava jezika je zahtevna, saj je pomen besed in stavkov pogosto odvisen od konteksta. Ista beseda ima lahko več pomenov, stavki so lahko dvoumni, razumevanje pa lahko zahteva tudi širše znanje o svetu.

Sodobne metode globokega učenja so močno izboljšale sposobnosti sistemov na tem področju. Med tipične naloge sodijo strojno prevajanje, povzemanje besedil, klasifikacija dokumentov, odgovarjanje na vprašanja in ustvarjanje novega besedila.

Pomemben sodoben primer so veliki jezikovni modeli (angl. *large language models*, LLM), ki se učijo iz velikih količin besedila in tako pridobijo vzorce o tem, kako se jezik uporablja. Verjetno vsakdo danes pozna na primer ChatGPT, sistem, ki temelji na velikih jezikovnih modelih.

### 2.1.5 Druga področja umetne inteligence

Zgoraj navedena področja ne predstavljajo popolnega pregleda umetne inteligence. Umetna inteligenca je zelo široko področje in poleg strojnega ter globokega učenja vključuje tudi številne druge pristope.

Mednje sodijo na primer preiskovanje (angl. *search*) prostora možnih rešitev, problemi zadovoljevanja omejitev, logično sklepanje, planiranje (angl. *planning*), igranje iger, priporočilni sistemi itd. Posamezna področja se med seboj pogosto prepletajo; robot lahko na primer uporablja računalniški vid za zaznavanje okolice, preiskovanje in planiranje za izbiro poti ter spodbujevano učenje za izboljševanje svojega vedenja.

Tudi izvorno gradivo IAIO obravnava nekatera od teh področij ločeno v kasnejših poglavjih. Zgoraj predstavljene tehnologije zato razumemo predvsem kot izbor področij sodobne umetne inteligence, ne kot njeno popolno razdelitev.

## 2.2 Razvoj sistema umetne inteligence

Razvoj sistema umetne inteligence ni en sam korak, temveč proces, sestavljen iz več povezanih faz. Poenostavljeno ga lahko predstavimo kot

$$\boxed{\text{zbiranje podatkov}} \rightarrow \boxed{\text{učenje modela}} \rightarrow \boxed{\text{validacija}} \rightarrow \boxed{\text{testiranje}} \rightarrow \boxed{\text{uporaba}} \tag{2}$$

Proces se z uporabo sistema običajno ne konča. Na podlagi novih podatkov in opaženih pomanjkljivosti se lahko vrnemo na prejšnje korake ter model izboljšamo.

**Priporočilo za branje**

Ta razdelek je namenjen predvsem razumevanju širšega procesa razvoja sistema umetne inteligence. Posamezni koraki, ki bodo na DTUI, predvsem priprava podatkov, učenje in evalvacija modelov, bodo podrobneje obravnavani v nadaljnjih poglavjih.

### 2.2.1 Zbiranje podatkov

Razvoj sistema se začne s podatki. Če želimo na primer izdelati model za prepoznavanje mačk, potrebujemo slike mačk in drugih objektov ter ustrezne oznake. Če želimo napovedovati neko številsko vrednost, potrebujemo zgodovinske podatke, iz katerih se lahko model nauči povezav med vhodnimi podatki in želenim izhodom.

Kakovost sistema je močno odvisna od kakovosti podatkov, na katerih se uči. Ni dovolj, da je podatkov veliko; biti morajo tudi pravilni, reprezentativni in relevantni za problem, ki ga želimo reševati.

**Glavna ideja: podatki so ključni**

Kakovost sistema umetne inteligence je močno odvisna od kakovosti učnih podatkov. Podatki morajo biti pravilno označeni, dovolj reprezentativni za primere, s katerimi se bo sistem srečeval, ter relevantni za dejanski problem.
Slabi podatki lahko vodijo do slabega modela ne glede na to, kako napreden algoritem uporabimo.

Zbiranje kakovostnih podatkov je zato pogosto eden izmed zahtevnejših delov razvoja. Podatke je treba zbrati, preveriti, po potrebi označiti in pripraviti za nadaljnjo uporabo.

### 2.2.2 Učenje modela

Učenje modela (angl. *training*) je proces, v katerem algoritem iz učnih podatkov pridobiva vzorce.

Pri nadzorovanem učenju model na učnih primerih proizvede napoved, jo primerja s pravilnim odgovorom ter izračuna svojo napako oziroma izgubo (angl. *loss*). Nato prilagodi svoje notranje parametre tako, da poskuša to napako zmanjšati. Ta postopek se večkrat ponovi in model postopoma izboljšuje svoje napovedi.

Pomembno je, da cilj učenja ni zgolj pravilno obravnavati učne primere, temveč se naučiti vzorcev, ki bodo uporabni tudi pri novih podatkih.

### 2.2.3 Validacija

Med razvojem moramo preverjati, ali se model dejansko uči uporabnih vzorcev ali pa se zgolj prekomerno prilagaja učnim podatkom.

Za to uporabimo **validacijsko množico** (angl. *validation set*). Gre za podatke, na katerih model neposredno ne prilagaja svojih parametrov. Z njimi lahko ocenjujemo njegovo uspešnost na primerih, ki jih pri učenju ni videl.

Če je model zelo uspešen na učni množici, na validacijski pa bistveno slabši, je to lahko znak **pretiranega prileganja** (angl. *overfitting*).

Validacijsko množico uporabljamo tudi pri izbiri nastavitev oziroma **hiperparametrov** modela, na primer njegove kompleksnosti, velikosti stopnje učenja (angl. *learning rate*) ali trajanja učenja.

### 2.2.4 Testiranje

Ko je učenje končano in smo s pomočjo validacije izbrali končno različico modela, njegovo uspešnost ocenimo še na **testni množici** (angl. *test set*).

Testna množica mora biti ločena od procesa učenja in izbire modela. Namenjena je končni oceni, kako dobro bo model deloval na novih, prej ne videnih podatkih.

**Pozor**

Testnih podatkov ne smemo uporabljati za učenje modela ali izbiro njegovih nastavitev.
Če po ogledu rezultatov na testni množici model spreminjamo tako, da bi na njej dosegel boljši rezultat, testna množica ni več neodvisna ocena uspešnosti in začne opravljati vlogo validacijske množice.

Razlika med učno, validacijsko in testno množico bo podrobneje obravnavana v poglavjih o pripravi podatkov in evalvaciji modelov.

### 2.2.5 Uporaba modela

Ko je model naučen in ustrezno preizkušen, ga lahko uporabimo v dejanskem okolju. Temu pravimo uvedba oziroma uporaba modela (angl. *deployment*).

Model lahko na primer deluje na strežniku, v mobilni aplikaciji, medicinski napravi ali drugem sistemu, kjer obdeluje nove podatke in ustvarja napovedi.

Z uvedbo se razvoj praviloma ne konča. Podatki v resničnem okolju se lahko razlikujejo od učnih podatkov ali se skozi čas spremenijo, zato je treba delovanje sistema spremljati. Novi podatki in opažene težave lahko nato uporabimo za razvoj izboljšane različice modela.

**Ključna ideja**

Razvoj sistema umetne inteligence je ponavljajoč se proces. Po uvedbi model spremljamo, zbiramo nove podatke in ga po potrebi ponovno izboljšamo.

## 2.3 Družbeni vplivi umetne inteligence

Sistemi umetne inteligence ne delujejo ločeno od družbe, v kateri nastajajo in se uporabljajo. Učijo se iz podatkov, ki pogosto odražajo obstoječe vzorce, odločitve in pristranskosti. Če so določene skupine v podatkih slabo zastopane ali če podatki vsebujejo posledice preteklih neenakosti, se lahko to odrazi tudi v vedenju naučenega modela.

**Ključna ideja**

Sistem umetne inteligence se uči iz podatkov, ki mu jih podamo. Če podatki vsebujejo pristranskosti ali ne predstavljajo dobro vseh primerov, lahko model te vzorce reproducira ali celo okrepi.

Pomembno vprašanje je zato **pravičnost** (angl. *fairness*) sistemov umetne inteligence. Za pravičnost ne obstaja ena sama splošno sprejeta matematična definicija. Lahko na primer zahtevamo, da različne skupine prejemajo pozitivne odločitve v podobnem deležu, da imajo podobne stopnje napak ali da imajo napovedane verjetnosti enak pomen za vse skupine. Te zahteve niso vedno združljive, zato izbira primernega kriterija ni le tehnično, temveč tudi etično vprašanje.

Uporaba velikih količin podatkov odpira tudi vprašanja **zasebnosti**. Podatki so lahko zbrani za drug namen, posamezniki morda niso privolili v njihovo uporabo za učenje modelov, navidezno anonimizirane podatke pa je včasih mogoče ponovno povezati s posamezniki. Pri razvoju sistema je zato pomembno razmisliti, katere podatke res potrebujemo in kako jih bomo zaščitili.

Ko sistemi umetne inteligence sprejemajo pomembne odločitve, se pojavi vprašanje **odgovornosti**. Če sistem naredi napako pri medicinski diagnozi, zaposlovanju ali odločanju o kreditu, ni vedno očitno, kdo je zanjo odgovoren: razvijalec modela, organizacija, ki ga uporablja, ali oseba, ki je sledila njegovi odločitvi.

S tem je povezana tudi **transparentnost in razložljivost**. Nekateri modeli, na primer preprosta odločitvena drevesa, omogočajo razmeroma neposreden vpogled v način odločanja (marsikdo se sicer tudi tu ne bi strinjal). Pri kompleksnejših modelih je razlog za posamezno napoved lahko veliko težje razložiti. V primerih z velikimi posledicami je zato lahko pomembno ne le, kako uspešen je model, temveč tudi, ali lahko njegove odločitve ustrezno razumemo in preverimo.

Sistemi umetne inteligence lahko vplivajo tudi na podatke, iz katerih se bodo učili prihodnji sistemi. Tako lahko nastanejo **povratne zanke** (angl. *feedback loops*). Če na primer model na podlagi zgodovinskih podatkov sistematično daje prednost določeni skupini, njegove odločitve vplivajo na prihodnji svet, ti novi podatki pa lahko isti vzorec še dodatno okrepijo.

Razvoj in uporaba velikih modelov imata tudi **okoljske in infrastrukturne stroške**. Učenje zahtevnih modelov lahko zahteva veliko računske moči in energije, dostop do takšnih virov pa je pogosto omejen na manjše število velikih organizacij.

Pri generativni umetni inteligenci se številna od teh vprašanj pojavijo v še večjem obsegu. Veliki modeli se učijo na ogromnih količinah podatkov in lahko ustvarijo zelo veliko vsebin brez neposrednega človeškega nadzora. Poleg že omenjenih vprašanj se zato pojavljajo tudi problemi, povezani z ustvarjanjem zavajajočih vsebin, avtorskimi pravicami in lastništvom podatkov.

**Razmislek o družbenem vplivu**

Pri presoji sistema umetne inteligence si je koristno zastaviti nekaj osnovnih vprašanj:

- Iz katerih podatkov se sistem uči in kdo je v njih zastopan?
- Kaj sistem optimizira in kako merimo njegovo uspešnost?
- Kdo ima od sistema koristi in kdo je lahko zaradi njega oškodovan?
- Kako lahko morebitne probleme zmanjšamo ali preprečimo?
- Kdo je odgovoren za posledice njegove uporabe?

## 2.4 Povzetek

**Ključne ugotovitve**

- Umetna inteligenca je področje računalništva, ki se ukvarja z razvojem sistemov za izvajanje nalog, ki jih običajno povezujemo s človeško inteligenco.
- Strojno učenje je eno izmed osrednjih področij sodobne umetne inteligence. Namesto da bi vnaprej zapisali vsa pravila, se model uporabnih vzorcev nauči iz podatkov.
- Globoko učenje je podpodročje strojnega učenja, ki uporablja nevronske mreže z več plastmi.
- Razvoj sistema umetne inteligence običajno vključuje zbiranje podatkov, učenje modela, validacijo, testiranje in uporabo. Proces se pogosto ponavlja, saj sistem na podlagi novih podatkov in rezultatov izboljšujemo.
- Kakovost sistema je močno odvisna od kakovosti podatkov. Podatki morajo biti pravilni, relevantni in reprezentativni za primere, na katerih bo model uporabljen.
- Pri razvoju in uporabi umetne inteligence je treba poleg tehnične uspešnosti upoštevati tudi pristranskost, pravičnost, zasebnost, odgovornost, razložljivost ter širše družbene in okoljske posledice.

# 3 Priprava podatkov

Priprava podatkov (angl. *data preprocessing*) je faza strojnega učenja, v kateri surove podatke spremenimo v obliko, primerno za učenje modelov. Tudi najbolj sofisticiran algoritem je nemočen, če so podatki nepopolni, napačno strukturirani ali vsebujejo zavajajoče vzorce. V praksi priprava podatkov pogosto zavzame velik del projekta: od razumevanja **tipov podatkov** (numerični, kategorični, ordinalni, časovne vrste, slike, besedilo), preko obravnave **manjkajočih vrednosti** in **šuma**, do pravilne **delitve podatkov**, skaliranja in drugih pretvorb.

Algoritmi strojnega učenja iščejo vzorce v podatkih, ki jim jih podamo. Če podatki vsebujejo napake, manjkajoče vrednosti ali nereprezentativne vzorce, se lahko to neposredno odrazi v naučenem modelu. Zato je kakovost modela v veliki meri omejena s kakovostjo podatkov, na katerih se uči.

Intuitivno je priprava podatkov podobna pripravi sestavin pred kuhanjem zahtevne jedi. Tudi če imamo vrhunski recept (algoritem), iz pokvarjenih ali napačno pripravljenih sestavin ne bo nastala dobra jed.

Najprej odstranimo očitne napake in šum, jih pravilno razdelimo na učno, validacijsko in testno množico in nato podatke ustrezno pretvorimo in skaliramo.

**Ključno pravilo strojnega učenja**

Kakovost modela je omejena s kakovostjo podatkov, na katerih se uči.
Pogosto uporabljamo izraz smeti noter, smeti ven (malo na silo preveden angl. *"garbage in, garbage out"*): tudi zelo napreden algoritem ne more zanesljivo odkriti uporabnih vzorcev, če so vhodni podatki slabe kakovosti.

## 3.1 Struktura podatkov

Pri strojnem učenju so podatki pogosto predstavljeni v obliki tabele. Posamezne vrstice predstavljajo **primere** oziroma **vzorce** (angl. *instances, samples*), stolpci pa njihove **značilke** oziroma **atribute** (angl. *features, attributes*).

Na primer:

| Starost | Višina | Teža | Barva | Ocena | Oznaka |
|---|---|---|---|---|---|
| 25 | 175 | 70 | modra | 2 | da |
| 30 | 160 | 55 | rdeča | 4 | ne |

Vsaka vrstica opisuje en primer, posamezni stolpci pa njegove lastnosti.

Značilke lahko glede na vrsto vrednosti razdelimo na več osnovnih tipov:

- **numerične značilke** (angl. *numerical features*), katerih vrednosti so števila, na primer starost, višina ali teža;
- **kategorične značilke** (angl. *categorical features*), katerih vrednosti predstavljajo kategorije, na primer barva, država ali vrsta izdelka;
- **ordinalne značilke** (angl. *ordinal features*), pri katerih med kategorijami obstaja naraven vrstni red, na primer ocena od 1 do 5.

Podatki se lahko pojavljajo tudi v drugih oblikah, na primer kot časovne vrste, slike ali besedilo. Vrsta podatkov pomembno vpliva na to, kako jih moramo pripraviti pred učenjem.

## 3.2 Težave s kakovostjo podatkov

Podatki iz resničnega sveta pogosto vsebujejo napake ali nepopolnosti. Pred učenjem modela moramo zato preveriti njihovo kakovost in po potrebi ustrezno ukrepati.

### 3.2.1 Šum

**Šum** (angl. *noise*) predstavljajo naključne napake ali odstopanja, ki kvarijo dejanske vzorce v podatkih. Povzročijo ga lahko nenatančne meritve, napake pri vnosu ali naravna variabilnost opazovanega pojava.

Prisotnost šuma otežuje učenje, saj mora model razlikovati med dejanskimi vzorci in naključnimi odstopanji.

### 3.2.2 Osamelci

**Osamelci** (angl. *outliers*) so primeri, ki močno odstopajo od večine ostalih podatkov.

Če na primer napovedujemo cene običajnih stanovanj in se v podatkih pojavi nepremičnina z večdesetkrat višjo ceno od drugih, lahko tak primer pri nekaterih modelih močno vpliva na naučeno povezavo med značilkami in ciljno vrednostjo.

Osamelca ne smemo samodejno obravnavati kot napako. Lahko gre za napačno meritev, lahko pa predstavlja tudi redek, vendar povsem veljaven primer. Pred njegovo odstranitvijo moramo zato razumeti njegov izvor.

### 3.2.3 Manjkajoče vrednosti

Pri nekaterih primerih vrednosti posameznih značilk niso zabeležene. Takšne **manjkajoče vrednosti** (angl. *missing values*) moramo pred učenjem ustrezno obravnavati.

Preprost pristop je odstranitev vrstic ali stolpcev z veliko manjkajočimi vrednostmi. S tem pa lahko izgubimo pomembne informacije.

Pogosta alternativa je **imputacija** (angl. *imputation*), pri kateri manjkajoče vrednosti nadomestimo z ocenjenimi. Numerične vrednosti lahko na primer nadomestimo s povprečjem ali mediano značilke, kategorične pa z najpogostejšo kategorijo ali posebno oznako. Uporabimo lahko tudi naprednejše metode, ki manjkajočo vrednost ocenijo na podlagi podobnih primerov s kopiranjem ali preproste naučene regresije.

**Pozor**

Vrednosti, potrebne za imputacijo, na primer povprečje ali mediano, določimo samo iz **učne množice**. Enako pravilo nato uporabimo na validacijski in testni množici.
Če pri izračunu uporabimo tudi validacijske ali testne podatke, informacije iz teh množic uhajajo v proces učenja.

### 3.2.4 Podvojeni primeri

**Podvojeni primeri** (angl. *duplicates*) lahko povzročijo, da imajo določeni podatki pri učenju večjo težo od drugih. Če podvojitve niso namerne, jih je zato običajno smiselno prepoznati in odstraniti.

## 3.3 Skaliranje značilk

Različne numerične značilke imajo lahko zelo različne velikostne rede. Na primer, starost je lahko med 18 in 80, letni dohodek pa med 20 000 in 200 000.

Pri algoritmih, ki uporabljajo razdalje ali so občutljivi na velikost vhodnih vrednosti, lahko značilka z večjimi številskimi vrednostmi neupravičeno prevlada nad drugimi. Zato značilke pogosto **skaliramo** (angl. *feature scaling*).

### 3.3.1 Standardizacija

Pri **standardizaciji** (angl. *standard scaling*) od vsake vrednosti odštejemo povprečje značilke in rezultat delimo s standardnim odklonom:

$$\tilde{x}_i = \frac{x_i - \mu}{\sigma}, \tag{3}$$

kjer je $\mu$ povprečje in $\sigma$ standardni odklon značilke.

Po standardizaciji ima značilka povprečje približno 0 in standardni odklon 1.

**Primer: standardizacija**

Naj bodo podatki

$$[10, 20, 30, 40, 50].$$

Njihovo povprečje je

$$\mu = 30,$$

standardni odklon pa približno

$$\sigma \simeq 14{,}14.$$

Po standardizaciji dobimo približno

$$[-1{,}41, -0{,}71, 0, 0{,}71, 1{,}41].$$

### 3.3.2 Min-max skaliranje

Druga pogosta možnost je **min-max skaliranje** (angl. *min-max scaling*), pri katerem vrednosti preslikamo na interval $[0, 1]$:

$$\tilde{x}_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}.$$

V praksi se za takšno pretvorbo pogosto uporablja tudi izraz normalizacija, zgodi pa se tudi da se izraz normalizacija (sicer striktno gledano napačno) kdaj uporabi za standardizacijo.

**Pozor**

Tudi parametre skaliranja, kot so $\mu$, $\sigma$, $x_{\min}$ in $x_{\max}$, določimo samo na učni množici. Na validacijski in testni množici nato uporabimo iste vrednosti.

## 3.4 Kodiranje spremenljivk

Večina algoritmov strojnega učenja potrebuje numerične vhode, zato moramo kategorične vrednosti pretvoriti v številsko obliko.

Preprosto kodiranje, na primer

$$\text{rdeča} = 1, \qquad \text{modra} = 2, \qquad \text{zelena} = 3, \tag{4}$$

je lahko problematično, saj številke nakazujejo vrstni red in razdalje med kategorijami, ki v resnici ne obstajajo.

Pogosta rešitev je **enično kodiranje** (angl. *one-hot encoding*). Za vsako možno kategorijo ustvarimo svojo binarno značilko.

Na primer:

| Barva | je_rdeča | je_modra | je_zelena |
|---|---|---|---|
| rdeča | 1 | 0 | 0 |
| modra | 0 | 1 | 0 |
| zelena | 0 | 0 | 1 |

Če ima kategorična značilka $k$ možnih vrednosti, pri takem pristopu ustvarimo $k$ binarnih značilk.

Pri **ordinalnih značilkah**, kjer obstaja naraven vrstni red, je položaj drugačen. Ocene

$$1 < 2 < 3 < 4 < 5 \tag{5}$$

na primer res vsebujejo informacijo o vrstnem redu. Vendar moramo biti previdni pri interpretaciji, saj razlika med ocenama 1 in 2 ni nujno enako pomembna kot razlika med ocenama 4 in 5.

Pri nekaterih spremenljivkah je pomembna njihova **cikličnost**. Tipičen primer so meseci v letu. Če jih naivno kodiramo s števili od 1 do 12, sta januar in december številčno najbolj oddaljena, čeprav sta si v ciklu neposredno sosednja.

Za ciklično spremenljivko $x$ s periodo $T$ lahko uporabimo

$$x_{\sin} = \sin\left(2\pi \frac{x}{T}\right), \tag{6}$$

$$x_{\cos} = \cos\left(2\pi \frac{x}{T}\right). \tag{7}$$

Za mesece velja $T = 12$.

Potrebujemo obe koordinati. Sam sinus ne zadostuje, saj lahko različnim položajem v ciklu pripiše enako vrednost.

## 3.5 Gradnja značilk

Surovi podatki niso vedno zapisani v obliki, ki najbolje izraža vzorce, pomembne za napovedovanje. **Gradnja značilk** (angl. *feature construction/engineering*) je proces ustvarjanja ali pretvarjanja značilk tako, da bolje predstavljajo pomembne lastnosti problema.

Predstavljajmo si, da želimo napovedovati ceno zemljišča in imamo podano njegovo dolžino ter širino. Namesto da uporabimo obe surovi vrednosti, lahko ustvarimo novo značilko:

$$\text{površina} = \text{dolžina} \cdot \text{širina}. \tag{8}$$

Površina v tem primeru predstavlja lastnost, ki je neposredno vezana na ceno. Za nekatere modele je tak multiplikativen odnos surovih spremenljivk dolžine in širine tudi nemogoč za modeliranje brez dodatnih "trikov".

Med pogoste pristope sodijo:

- ustvarjanje potenc ali produktov obstoječih značilk;
- razvrščanje numeričnih vrednosti v intervale oziroma skupine;
- logaritemske transformacije močno asimetričnih vrednosti;
- pridobivanje dodatnih informacij iz datumov in časa, na primer dneva v tednu, meseca ali podatka, ali je dan konec tedna.

Pri oblikovanju značilk je pogosto zelo pomembno **domensko znanje** (angl. *domain knowledge*). Poznavanje področja nam lahko pomaga prepoznati pretvorbe, ki predstavljajo problem bolje od prvotnih surovih podatkov.

**Primer**

Pri zaznavanju sumljivih finančnih transakcij so poleg zneska transakcije lahko uporabne tudi značilke, kot so:

- znesek glede na običajno porabo uporabnika,
- čas od prejšnje transakcije,
- število transakcij v zadnji uri,
- oddaljenost od običajnih lokacij uporabnika.

Takšne značilke lahko bolje opišejo nenavadno vedenje kot same surove vrednosti.

**Pozor**

Pri gradnji značilk moramo paziti, da ne pride do uhajanja podatkov.

## 3.6 Uhajanje podatkov

**Uhajanje podatkov** (angl. *data leakage*) nastane, ko pri učenju modela uporabimo informacijo, ki v trenutku dejanske napovedi ne bi bila na voljo, oziroma ko informacije iz validacijskih ali testnih podatkov neupravičeno vplivajo na proces učenja.

Uhajanje lahko povzroči zelo dobre rezultate na učnih ali validacijskih podatkih, ki se pri uporabi modela na novih podatkih nenadoma močno poslabšajo. Model namreč pri učenju izkoristi informacijo, do katere v resničnem okolju ne bo imel dostopa.

V grobem ločimo dve pomembni obliki uhajanja: **ciljno uhajanje** in **uhajanje med učenjem in testiranjem**.

### 3.6.1 Ciljno uhajanje

**Ciljno uhajanje** (angl. *target leakage*) nastane, ko je katera izmed značilk neposredno izpeljana iz ciljne vrednosti ali pa vsebuje informacijo, ki je posledica izida, ki ga želimo napovedati.

Predstavljajmo si, da želimo napovedati, ali bo pacient v naslednjih 30 dneh ponovno sprejet v bolnišnico. Med značilkami imamo tudi datum ponovnega sprejema. Ta podatek obstaja samo za paciente, ki so bili dejansko ponovno sprejeti, zato praktično neposredno razkriva pravilni odgovor.

Podobni primeri so:

- napovedovanje neplačila posojila z uporabo števila zamujenih plačil, ki so že posledica neplačevanja;
- napovedovanje bolezni z uporabo podatka o zdravilu, ki je bilo predpisano prav zaradi te bolezni;
- napovedovanje odpovedi naročnine z uporabo razloga za odpoved.

Skupna značilnost takšnih primerov je, da informacija obstaja šele zaradi izida oziroma je na voljo šele po njem.

**Praktično vprašanje**

Pri vsaki sumljivi značilki se vprašajmo:

*Ali bi imel ta podatek na voljo v trenutku, ko moram narediti napoved?*

Če je odgovor ne, lahko značilka predstavlja ciljno uhajanje.

Pomembno je, da močna povezanost značilke s ciljno vrednostjo sama po sebi **ni** uhajanje podatkov. Dobra značilka je lahko s ciljem zelo močno povezana. Težava nastane, če vsebuje informacijo iz prihodnosti ali neposredno iz ciljne vrednosti.

### 3.6.2 Uhajanje med učenjem in testiranjem

Druga oblika je **uhajanje med učenjem in testiranjem** (angl. *feature leakage* oziroma *train-test contamination*). Do njega pride, ko informacije iz validacijske ali testne množice vplivajo na učenje modela oziroma na pripravo učnih podatkov.

Pogosti primeri so:

- standardizacijo ali drugo skaliranje prilagodimo na celotni podatkovni množici še pred delitvijo;
- povprečje ali mediano za imputacijo manjkajočih vrednosti izračunamo iz vseh podatkov;
- enaki ali skoraj enaki primeri se pojavijo hkrati v učni in testni množici;
- časovno odvisne podatke naključno premešamo, tako da informacije iz prihodnosti vplivajo na učenje modela.

V vseh teh primerih testni oziroma validacijski podatki niso več popolnoma neodvisni od procesa učenja.

**Ključno pravilo**

Vse postopke priprave podatkov, ki se iz podatkov nekaj naučijo, prilagodimo **samo na učni množici**. To vključuje na primer povprečje in standardni odklon pri standardizaciji, parametre imputacije ter druge parametre pretvorb.
Na validacijski in testni množici nato uporabimo iste, že določene pretvorbe.

Na primer, pravilen vrstni red pri standardizaciji je:

1. podatke razdelimo na učno, validacijsko in testno množico;
2. povprečje in standardni odklon izračunamo samo na učni množici;
3. z njima standardiziramo učno množico;
4. z istima vrednostma standardiziramo tudi validacijsko in testno množico.

Če bi povprečje in standardni odklon izračunali iz vseh podatkov pred delitvijo, bi informacije o validacijski in testni množici že vplivale na predstavitev učnih podatkov.

### 3.6.3 Časovno uhajanje

Pri časovno odvisnih problemih moramo posebej paziti na **časovno uhajanje** (angl. *temporal leakage*).

Do njega pride, kadar za napoved preteklosti neposredno ali posredno uporabimo informacije iz prihodnosti.

Če na primer model učimo na podatkih iz leta 2023 in nato z njim ocenjujemo napovedi za leto 2022, uporabljamo prihodnost za napovedovanje preteklosti.

Pri časovnih podatkih zato praviloma ohranimo časovni vrstni red:

$$\text{preteklost} \longrightarrow \text{učenje} \qquad \text{prihodnost} \longrightarrow \text{validacija/test}. \tag{9}$$

Časovno uhajanje se lahko pojavi tudi znotraj posamezne značilke. Če želimo na primer napovedati, ali bo uporabnik jutri opravil nakup, značilka, ki vsebuje število njegovih nakupov v naslednjem tednu, uporablja informacijo, ki ob času napovedi še ni znana.

**Ciljno uhajanje ali uhajanje med učenjem in testiranjem?**

| | **Ciljno uhajanje** | **Uhajanje med učenjem in testiranjem** |
|---|---|---|
| Kaj uhaja? | Informacija iz ciljne vrednosti v značilko. | Informacija iz validacijskih ali testnih podatkov v proces učenja. |
| Tipičen vzrok | Značilka je posledica izida ali je na voljo šele po njem. | Pretvorbe ali statistike so izračunane tudi iz podatkov zunaj učne množice. |
| Vprašanje za preverjanje | *Ali bi imel to informacijo pred izidom?* | *Ali so validacijski ali testni podatki kakorkoli vplivali na učenje?* |
| Rešitev | Odstranimo oziroma pravilno časovno omejimo značilko. | Popravimo vrstni red delitve in priprave podatkov. |

### 3.6.4 Zakaj je uhajanje nevarno?

Obe obliki uhajanja lahko povzročita **pretirano optimistično oceno uspešnosti**.

Pri ciljnem uhajanju ima model posredno dostop do pravilnega odgovora. Pri uhajanju med učenjem in testiranjem pa validacijska oziroma testna množica ni več resnično neodvisna od procesa učenja.

Zato lahko model na validacijskih podatkih dosega izjemno dobre rezultate, pri uporabi na resnično novih podatkih pa njegova uspešnost močno pade.

**Pozor**

Nenavadno dober rezultat je lahko razlog za dodatno preverjanje.
Če preprost model pri zahtevnem problemu nenadoma dosega skoraj popolno uspešnost, je smiselno preveriti, ali katera značilka neposredno razkriva cilj oziroma ali je pri pripravi podatkov prišlo do uhajanja informacij.

## 3.7 Delitev in reprezentativnost podatkov

Za preverjanje, kako dobro model deluje na novih podatkih, podatkovne množice ne smemo uporabiti v celoti za učenje. Poleg pravilne delitve podatkov je pomembno tudi, da posamezne množice dovolj dobro predstavljajo podatke, na katerih bo model kasneje dejansko uporabljen.

### 3.7.1 Učna, validacijska in testna množica

Podatkovno množico običajno razdelimo na tri dele:

- **učno množico** (angl. *training set*), na kateri se model uči;
- **validacijsko množico** (angl. *validation set*), ki jo uporabljamo za izbiro hiperparametrov in drugih odločitev med razvojem modela;
- **testno množico** (angl. *test set*), ki jo uporabimo samo na koncu za oceno končnega modela na prej nevidenih podatkih.

Razmerja med množicami niso fiksna. Pogoste so na primer delitve, pri katerih je približno 60–80 % podatkov namenjenih učenju, po 10–20 % pa validaciji in testiranju.

**Analogija z izpitom**

Učno množico si lahko predstavljamo kot učbenik, validacijsko kot poskusne izpite, testno pa kot končni izpit.
Na učnih podatkih se učimo, na validacijskih preverjamo in prilagajamo svoj pristop, test pa uporabimo šele na koncu.

**Ključno pravilo**

Testnih podatkov ne uporabljamo za učenje modela ali sprejemanje odločitev o njem.
Če rezultate na testni množici pogledamo in nato na njihovi podlagi spremenimo model, postanejo tudi naše razvojne odločitve posredno odvisne od testnih podatkov. Testna množica zato ne predstavlja več povsem neodvisne ocene uspešnosti modela.

### 3.7.2 Stratificirana delitev in neuravnoteženi razredi

Pri klasifikaciji so lahko posamezni razredi zelo različno zastopani. Temu pravimo **neuravnoteženost razredov** (angl. *class imbalance*).

Če imamo na primer 90 % primerov razreda A in 10 % primerov razreda B, želimo podobno razmerje ohraniti tudi v učni, validacijski in testni množici. V ta namen pogosto uporabimo **stratificirano vzorčenje** (angl. *stratified sampling*).

Stratifikacija je posebej pomembna pri redkih razredih, saj lahko pri povsem naključni delitvi posamezna množica vsebuje zelo malo primerov redkejšega razreda.

Neuravnoteženost razredov vpliva tudi na izbiro metrike za ocenjevanje modela. Sama točnost je lahko v takih primerih zavajajoča, zato bomo primernejše metrike obravnavali v poglavju o evalvaciji modelov.

### 3.7.3 Časovna delitev podatkov

Pri časovno odvisnih podatkih moramo posebej paziti na vrstni red primerov. Naključna delitev pogosto ni primerna.

Če bi podatke o cenah delnic iz let 2020–2024 naključno razdelili med učno in testno množico, bi se lahko zgodilo, da bi model pri učenju uporabljal podatke iz prihodnosti in nato z njimi napovedoval preteklost.

Primernejši pristop je **časovna delitev** (angl. *temporal split*), pri kateri starejše podatke uporabimo za učenje, novejše pa za validacijo in testiranje:

$$\text{preteklost} \longrightarrow \text{učenje}, \qquad \text{prihodnost} \longrightarrow \text{validacija/test}. \tag{10}$$

S tem bolje posnemamo dejansko uporabo modela, kjer prihodnosti v trenutku napovedi še ne poznamo.

### 3.7.4 Veljavnost testne množice

Testna množica naj bi ocenila, kako dobro bo model deloval na novih podatkih. Takšna ocena pa je smiselna samo, če testni podatki dovolj dobro predstavljajo okolje, v katerem bomo model dejansko uporabljali.

Testna množica je ustrezna, če:

- predstavlja podatke, ki jih bo model srečeval pri uporabi;
- ni bila uporabljena pri učenju ali izbiri modela;

- vsebuje dovolj primerov za smiselno oceno njegove uspešnosti.

**Primer**

Model se nauči in testira na podatkih ene bolnišnice ter na testni množici doseže 95 % uspešnost.
Ko ga uporabimo v drugi bolnišnici, uspešnost pade na 70 %.
Testni rezultat 95 % ni bil nujno napačen. Pravilno je opisoval delovanje modela na podatkih prve bolnišnice, ni pa dobro napovedoval njegovega delovanja v drugačnem okolju.

Rezultat na testni množici zato ni absolutno zagotovilo uspešnosti. Njegova veljavnost temelji na predpostavki, da so podatki pri dejanski uporabi dovolj podobni podatkom v testni množici.

### 3.7.5 Pristranskost vzorčenja

Težava lahko nastane že pri zbiranju podatkov.

Če podatkovna množica ne predstavlja dobro populacije, na kateri želimo model uporabljati, govorimo o **pristranskosti vzorčenja** (angl. *sampling bias*). Do nje lahko pride, ker način zbiranja sistematično izključi določene skupine (**pristranskost izbire**, angl. *selection bias*), ker opazujemo le primere, ki so prestali nek predhodni izbor (**pristranskost preživelih**, angl. *survivorship bias*), ker zbiramo predvsem najlažje dostopne podatke (**priložnostno vzorčenje**, angl. *convenience sampling*) ali ker se sodelujoči sistematično razlikujejo od tistih, ki se ne odzovejo (**pristranskost zaradi neodziva**, angl. *non-response bias*).

Pri presoji podatkov se je zato smiselno vprašati, kdo oziroma kaj v njih manjka, kako so bili podatki zbrani in ali bi se naši zaključki spremenili, če bi vključili manjkajoče skupine.

### 3.7.6 Premik porazdelitve

Tudi če so bili podatki ob zbiranju reprezentativni, se lahko okolje po učenju modela spremeni. Temu pravimo **premik porazdelitve** (angl. *distribution shift*). Pri **premiku vhodne porazdelitve** (angl. *covariate shift*) se spremeni porazdelitev vhodov, pri **premiku oznak** (angl. *label shift*) pogostost ciljnih vrednosti, pri **spremembi koncepta** (angl. *concept drift*) pa sama povezava med vhodom in ciljem.

$$P_{\text{train}}(X) \neq P_{\text{deploy}}(X), \;\; P(Y|X) \text{ ostane isti} \tag{11}$$

opisuje premik vhodne porazdelitve (npr. uporaba druge naprave za rentgensko slikanje).

$$P_{\text{train}}(Y) \neq P_{\text{deploy}}(Y), \;\; P(X|Y) \text{ ostane isti} \tag{12}$$

opisuje premik oznak (npr. filter neželene pošte, naučen na podatkih, kjer je bilo neželenih 5% sporočil, uporabimo v nabiralniku, kjer je neželenih 40%).

Pri spremembi koncepta velja:

$$P_{\text{train}}(Y \mid X) \neq P_{\text{deploy}}(Y \mid X), \tag{13}$$

(npr. model za napovedovanje pretoka reke iz količine padavin. V deževnem obdobju nasičena tla povzročijo večji porast pretoka, med sušo pa suha tla več vode vpijejo, zato enaka količina padavin povzroči manjši pretok).

**Povezava med pojmi**

Pristranskost vzorčenja, premik porazdelitve in veljavnost testne množice so različni vidiki istega osnovnega vprašanja:

**Ali naši podatki predstavljajo okolje, v katerem bo model uporabljen?**

**Pristranskost vzorčenja** pomeni, da podatki niso bili reprezentativni že ob zbiranju.
**Premik porazdelitve** pomeni, da se je okolje po zbiranju podatkov spremenilo.
**Neustrezna testna množica** pa pomeni, da uspešnosti modela v dejanskem okolju ne moremo zanesljivo oceniti.

## 3.8 Povzetek

**Ključne ugotovitve**

- **Kakovost podatkov:** pred učenjem moramo prepoznati in ustrezno obravnavati šum, osamelce, manjkajoče vrednosti ter podvojene primere.
- **Skaliranje značilk:** s standardizacijo ali min-max skaliranjem lahko preprečimo, da bi značilke z večjimi številskimi vrednostmi pri nekaterih algoritmih neupravičeno prevladale nad drugimi.
- **Kodiranje značilk:** kategorične značilke moramo pretvoriti v obliko, primerno za učenje, na primer z eničnim kodiranjem (angl. *one-hot encoding*). Pri ordinalnih in cikličnih značilkah moramo upoštevati tudi njihovo notranjo strukturo.
- **Gradnja značilk:** iz surovih podatkov lahko ustvarimo nove značilke, ki bolje predstavljajo pomembne vzorce. Pri tem je pogosto ključno domensko znanje.
- **Delitev in reprezentativnost podatkov:** učna množica je namenjena učenju, validacijska izbiri nastavitev in modela, testna pa končni oceni. Način delitve mora čim bolje posnemati dejansko uporabo modela.
- **Posebne delitve:** pri neuravnoteženih razredih je pogosto smiselna stratificirana delitev, pri časovno odvisnih podatkih pa moramo upoštevati časovni vrstni red in model učiti na preteklosti ter preverjati na prihodnosti.
- **Veljavnost testne množice:** rezultat na testni množici je smiseln le, če ta dovolj dobro predstavlja podatke, na katerih bo model dejansko uporabljen.
- **Pristranskost vzorčenja:** podatki so lahko nereprezentativni že zaradi načina zbiranja. Med pogoste primere sodijo pristranskost izbire, pristranskost preživelih, priložnostno vzorčenje in pristranskost zaradi neodziva.
- **Premik porazdelitve:** tudi dobro zbrani podatki lahko sčasoma prenehajo predstavljati dejansko okolje uporabe. Spremenijo se lahko vhodni podatki, pogostost ciljnih vrednosti ali sama povezava med vhodi in izhodi.
- **Uhajanje podatkov:** ciljno uhajanje ter uhajanje informacij iz validacijskih ali testnih podatkov v proces učenja lahko povzročita nerealno optimistične rezultate. Vse postopke priprave, ki se iz podatkov nekaj naučijo, prilagodimo samo na učni množici.

## 3.9 Naloge

Naslednje naloge so namenjene preverjanju razumevanja priprave podatkov. Pri nekaterih je dovolj kratek izračun, pri drugih pa je pomembna predvsem utemeljitev odgovora.

1. **Standardizacija**

   Dana je značilka z vrednostmi

$$[2, 4, 6, 8, 10]. \tag{14}$$

Izračunajte njeno povprečje in standardni odklon ter nato vse vrednosti standardizirajte. Pri izračunu uporabite populacijski standardni odklon.

2. **Enično kodiranje**

   Kategorična značilka *velikost* ima tri možne vrednosti:

   $$\{\text{majhna}, \text{srednja}, \text{velika}\}. \tag{15}$$

   Koliko stolpcev nastane pri eničnem kodiranju (angl. *one-hot encoding*)? Zapišite kodiranje vrednosti *srednja*.

3. **Delitev podatkov**

   Podatkovna množica vsebuje 5000 primerov. Razdeliti jo želimo v razmerju

   $$70\% : 15\% : 15\% \tag{16}$$

   na učno, validacijsko in testno množico.

   Koliko primerov vsebuje posamezna množica? Kakšna je vloga vsake izmed njih?

4. **Manjkajoče vrednosti**

   V podatkovni množici manjka 10 % vrednosti v stolpcu *dohodek*.

   Predlagajte tri različne načine obravnave manjkajočih vrednosti. Za vsak način navedite:

   (a) primer, ko bi bil primeren;

   (b) primer, ko bi lahko povzročil težave.

5. **Logaritemska transformacija**

   Zakaj bi pretvorba cene hiš v logaritemsko skalo lahko pomagala pri uspešnosti napovedi.

6. **Gradnja značilk**

   Na voljo imamo podatke o nakupih uporabnikov s stolpci:

   `timestamp`, `user_id`, `purchase_amount`, `product_category`.

   Želimo napovedati, ali bo uporabnik v naslednjem tednu opravil nakup.

   Predlagajte vsaj tri nove značilke, ki bi jih lahko ustvarili iz podanih podatkov. Pri vsaki na kratko pojasnite, zakaj bi lahko bila uporabna.

7. **Skaliranje značilk**

   Podatkovna množica vsebuje značilke:

   - starost: 0–100,
   - letni dohodek: 0–1 000 000,
   - število otrok: 0–10.

   Odgovorite:

   (a) Kaj se lahko zgodi, če na takih podatkih brez skaliranja uporabimo algoritem, ki temelji na razdaljah?

   (b) Zakaj bi lahko značilka *dohodek* prevladala nad ostalima?

(c) Ali pričakujemo enako težavo pri odločitvenem drevesu? Utemeljite.

8. **Iskanje ciljnega uhajanja**

   Želimo napovedati, ali bo uporabnik odpovedal naročnino. Na voljo imamo naslednje značilke:

   (a) število dni od zadnje prijave;

   (b) skupno število nakupov;

   (c) razlog za odpoved naročnine;

   (d) povprečno trajanje seje;

   (e) število zahtevkov za podporo.

   Katera značilka lahko predstavlja **ciljno uhajanje** (angl. *target leakage*)?

   Pojasnite, zakaj. Kako bi v praksi preverili, ali je značilka ob času napovedovanja sploh na voljo?

9. **Močna povezanost še ni uhajanje**

   Pri napovedovanju cene hiše imamo značilke:

   - površina hiše,
   - število spalnic,
   - leto izgradnje,
   - zemljepisna širina in dolžina.

   Nekdo predlaga, da značilke odstranimo, ker so močno povezane s ceno in zato predstavljajo uhajanje podatkov.

   Razložite zakaj ta sklep ni pravilen?

   Pojasnite razliko med legitimno napovedno značilko in dejanskim uhajanjem podatkov.

10. **Vrste uhajanja podatkov**

    Za vsak primer določite, ali gre za:

    - ciljno uhajanje (angl. *target leakage*),
    - uhajanje med učenjem in testiranjem (angl. *train-test leakage*),
    - ali pa sploh ne gre za uhajanje.

    Odgovor utemeljite.

    (a) Napovedujemo rezultat izpita in uporabimo število ur učenja, izmerjeno pred izpitom.

    (b) Napovedujemo rezultat izpita in uporabimo zadovoljstvo učenca z oceno, izmerjeno po objavi rezultatov.

    (c) Manjkajoče vrednosti nadomestimo s povprečjem, izračunanim na celotni podatkovni množici, nato pa podatke razdelimo na učno in testno množico.

    (d) Napovedujemo ponovni sprejem pacienta v bolnišnico in uporabimo dolžino njegovega začetnega bivanja, ki je znana ob odpustu.

11. **Pristranskost vzorčenja**

    Za vsak primer ugotovite, zakaj podatkovna množica morda ni reprezentativna. Kjer je mogoče, poimenujte vrsto pristranskosti vzorčenja.

(a) Sistem za samodejno vožnjo je naučen skoraj izključno na posnetkih vožnje iz Kalifornije.

(b) Model za analizo mnenj uporabnikov se uči samo iz javno objavljenih ocen izdelkov.

(c) V raziskavi o zdravilu 30 % udeležencev pred zaključkom preneha sodelovati.

(d) Model za napoved uspešnosti zagonskih podjetij je naučen samo na podatkih podjetij, ki so uspešno preživela prvih pet let.

Pri vsakem primeru navedite tudi, katera skupina ali vrsta podatkov je premalo zastopana.

12. **Premik porazdelitve**

Model za napovedovanje povpraševanja po taksijih v New Yorku je bil naučen na podatkih iz leta 2019 in uporabljen marca 2020.

(a) Zakaj lahko pričakujemo, da porazdelitev podatkov ob uporabi ni več enaka učni porazdelitvi?

(b) Ali se je lahko spremenila porazdelitev vhodov $P(X)$, ciljnih vrednosti $P(Y)$ ali povezava $P(Y \mid X)$?

(c) Kako bi lahko takšno spremembo zaznavali po uvedbi modela?

13. **Veljavnost testne množice**

Model za prepoznavanje kožnega raka je naučen na slikah iz dermatoloških klinik v severni Evropi. Testno množico sestavlja naključnih 15 % slik iz istih klinik in model na njej doseže visoko uspešnost.

Model nato uporabimo v klinikah v podsaharski Afriki.

(a) Zakaj se lahko uspešnost modela spremeni?

(b) Ali je prvotni rezultat na testni množici zato napačen?

(c) Kaj rezultat dejansko pove in česa ne?

(d) Kako bi oblikovali testno množico, če vnaprej vemo, kje bo model uporabljen?

14. **Časovna delitev podatkov**

Imamo podatke o prodaji izdelka od leta 2020 do leta 2025 in želimo napovedovati prodajo v prihodnosti.

Nekdo predlaga, da vse primere naključno premešamo in 20 % podatkov uporabimo za testiranje.

Zakaj je lahko takšna delitev problematična? Predlagajte ustreznejši način delitve podatkov.

15. **Ciklično kodiranje**

Mesec v letu je predstavljen s številom

$$x \in \{1, 2, \ldots, 12\}. \tag{17}$$

Zakaj je neposredna numerična predstavitev lahko problematična?

Izračunajte ciklično predstavitev za januar ($x = 1$) in december ($x = 12$):

$$x_{\sin} = \sin\left(2\pi \frac{x}{12}\right), \qquad x_{\cos} = \cos\left(2\pi \frac{x}{12}\right). \tag{18}$$

Na podlagi dobljenih vrednosti pojasnite, kako ciklično kodiranje bolje izrazi bližino januarja in decembra.

16. **Celoten postopek priprave podatkov**

    Prejeli ste podatkovno množico, v kateri so:

    - manjkajoče numerične vrednosti,
    - kategorične značilke,
    - numerične značilke na zelo različnih skalah,
    - časovni žig,
    - podvojene vrstice.

    Model boste na koncu ocenjevali na ločeni testni množici.

    Opišite smiseln vrstni red priprave podatkov. Pri vsakem postopku, ki iz podatkov izračuna kakšno statistiko ali drugo nastavitev, posebej navedite, na kateri množici ga smemo prilagoditi.

**Predlogi za nadaljnje branje:**

1. Jason Brownlee, *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python.*

# 4 Nadzorovano učenje

**Nadzorovano učenje** (angl. *supervised learning*) je veja strojnega učenja, pri kateri se model uči iz označenih primerov. Vsak učni primer vsebuje vhodne podatke in pripadajoči pravilni odgovor. Cilj modela je iz teh primerov odkriti povezavo med vhodi in izhodi ter jo uporabiti za napovedovanje na novih, še nevidenih podatkih.

Intuitivno je nadzorovano učenje podobno učenju s primeri in rešitvami. Učencu pokažemo nalogo in pravilni odgovor; po dovolj primerih se nauči vzorcev, ki mu omogočajo reševanje tudi novih nalog. Podobno model ne poskuša zgolj pomniti posameznih učnih primerov, temveč se mora naučiti pravil, ki veljajo tudi zunaj učne množice.

**Ključna ideja**

Pri nadzorovanem učenju poznamo pravilne odgovore za učne primere. Model se iz njih nauči povezave med vhodnimi značilkami in ciljno vrednostjo ter jo nato uporablja za napovedovanje na novih primerih.

## 4.1 Struktura nadzorovanega učenja

Učni podatki so sestavljeni iz parov

$$(x_i, y_i), \tag{19}$$

kjer $x_i$ predstavlja vhod oziroma množico **značilk** (angl. *features*), $y_i$ pa pravilni izhod oziroma **oznako** ali **ciljno vrednost** (angl. *label, target*).

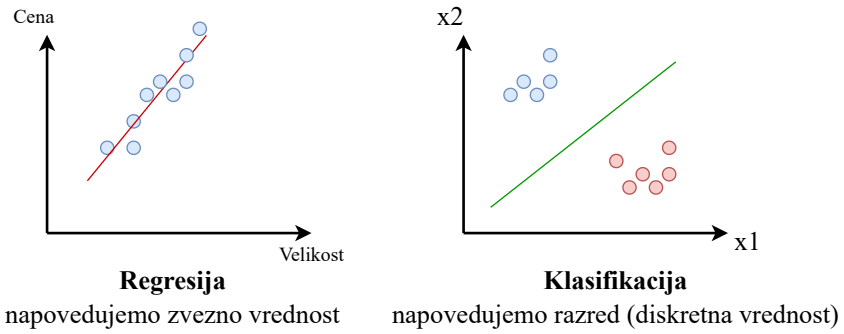
Če imamo $m$ učnih primerov, lahko učno množico zapišemo kot

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}. \tag{20}$$

Naloga učnega algoritma je poiskati funkcijo oziroma model $h$, ki vhod $x$ preslika v napoved

$$\hat{y} = h(x). \tag{21}$$

Želimo, da je napoved $\hat{y}$ čim bližje pravilni vrednosti $y$, predvsem pa, da model dobro napoveduje tudi za primere, ki jih med učenjem ni videl.




Slika 1: Primerjava regresije in klasifikacije.

**Osnovni pojmi**

**Značilke** (angl. *features*) so vhodne spremenljivke, ki jih model uporablja za napovedovanje.
**Oznaka oziroma ciljna vrednost** (angl. *label, target*) je vrednost, ki jo želimo napovedati.
**Napoved** (angl. *prediction*) je izhod, ki ga model izračuna za podani vhod.

Klasičen primer nadzorovanega učenja je podatkovna množica *Iris*. Vsebuje 150 primerov cvetov s štirimi numeričnimi značilkami, ki opisujejo dolžino in širino čašnih ter venčnih listov. Vsakemu primeru pripada ena izmed treh vrst rastline.

Naloga je klasifikacijska: iz označenih učnih primerov želimo izdelati model, ki bo na podlagi štirih značilk pravilno določil vrsto novega cveta.

Podatkovna množica Iris je zaradi majhnosti in pregledne strukture pogosto uporabljena za prikaz in primerjavo različnih klasifikacijskih algoritmov.

## 4.2 Klasifikacija in regresija

Probleme nadzorovanega učenja običajno delimo na dve glavni skupini glede na vrsto vrednosti, ki jo želimo napovedati.

### 4.2.1 Regresija

Pri **regresiji** (angl. *regression*) je ciljna vrednost numerična in lahko zavzame zvezne vrednosti.

Primeri regresijskih problemov so:

- napovedovanje cene nepremičnine,
- napovedovanje temperature,
- napovedovanje prodaje izdelka,
- ocenjevanje preostale življenjske dobe naprave.

Če model na primer napoveduje ceno stanovanja, je lahko rezultat 247 500 EUR ali katera koli druga vrednost v ustreznem območju.

### 4.2.2 Klasifikacija

Pri **klasifikaciji** (angl. *classification*) je ciljna vrednost eden izmed končnega števila razredov.

Primeri so:

- neželena ali običajna elektronska pošta,
- bolan ali zdrav pacient,
- prepoznava števke od 0 do 9,
- določanje vrste rastline.

Če imamo dva možna razreda, govorimo o **binarni klasifikaciji** (angl. *binary classification*). Če imamo več kot dva razreda, govorimo o **večrazredni klasifikaciji** (angl. *multiclass classification*). Obstajajo še druge vrste (npr. večoznačna klasifikacija (angl. *multilabel*)), ki jih tu ne bomo podrobneje obravnavali.

**Ključna razlika**

Pri **regresiji** napovedujemo numerično vrednost.
Pri **klasifikaciji** napovedujemo pripadnost enemu izmed razredov.

## 4.3 Funkcija izgube

Model med učenjem potrebuje način, kako oceniti, kako dobre so njegove napovedi. To določa **funkcija izgube** (angl. *loss function*), ki meri razliko med napovedjo modela in pravilnim odgovorom.

Manjša vrednost funckije izgube pomeni boljšo napoved. Učni algoritem zato prilagaja parametre modela tako, da poskuša funkcijo izgubo minimizirati. Pri nekaterih formulacijah namesto izgube maksimiziramo drugo ciljno funkcijo, na primer verjetje; z ustrezno spremembo predznaka lahko tak problem zapišemo tudi kot minimizacijo.

Pri regresiji je pogosta **kvadratna izguba** (angl. *squared loss*):

$$L(y, \hat{y}) = (y - \hat{y})^2. \tag{22}$$

Velike napake so zaradi kvadriranja kaznovane bistveno močneje kot majhne. Napaka velikosti 10 ima na primer stokrat večjo izgubo kot napaka velikosti 1.

Pri klasifikaciji se uporabljajo druge funkcije izgube, na primer logaritemska izguba (angl. *log loss*) oziroma navzkrižna entropija (angl. *cross-entropy*), ki jih bomo srečali pri posameznih modelih kasneje.

**Ključna ideja**

Funkcija izgube določa, kaj za model pomeni “narediti napako”. Učenje lahko razumemo kot iskanje parametrov modela, pri katerih je izguba na učnih podatkih čim manjša.

## 4.4 Linearna regresija

**Linearna regresija** (angl. *linear regression*) je eden najpreprostejših modelov nadzorovanega učenja. Uporabljamo jo pri regresijskih problemih, kjer želimo povezavo med značilkami in ciljno vrednostjo opisati z linearno funkcijo.

Če imamo eno značilko $x$, model zapišemo kot

$$h_\theta(x) = \theta_0 + \theta_1 x, \tag{23}$$

kjer sta $\theta_0$ in $\theta_1$ parametra modela. Parameter $\theta_0$ določa presečišče z osjo y, $\theta_1$ pa naklon premice. Učenje modela pomeni iskanje takšnih vrednosti parametrov, ki se čim bolje prilegajo učnim podatkom.

Za merjenje napake pogosto uporabimo **povprečno kvadratno napako** (angl. *mean squared error*, MSE):

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2, \tag{24}$$

kjer je $m$ število učnih primerov. Cilj je poiskati parametre, pri katerih je vrednost te funkcije čim manjša.

### 4.4.1 Več značilk

V praksi je ciljna vrednost običajno odvisna od več značilk. Cena hiše je lahko na primer odvisna od njene površine, števila sob, starosti in lokacije.

Linearno regresijo zato razširimo na več vhodnih značilk:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \;\; = \;\; \theta_0 + \sum_{j=1}^{n} \theta_j x_j. \tag{25}$$

Vsaki značilki $x_j$ pripada parameter $\theta_j$, ki določa njen prispevek k napovedi. Če uvedemo dodatno konstantno značilko $x_0 = 1$, lahko model krajše zapišemo vektorsko kot

$$h_\theta(x) = \theta^T x. \tag{26}$$

### 4.4.2 Gradientni spust

Kako torej poiščemo vrednosti parametrov, pri katerih je funkcija izgube čim manjša?

Zelo naiven pristop bi bil, da vsak parameter nekoliko povečamo in zmanjšamo ter pri obeh spremembah izračunamo vrednost funkcije izgube. Nato parameter premaknemo v smer, pri kateri se je izguba zmanjšala.

Za lažjo predstavo si lahko zamislimo, da želimo v popolni temi priti na dno hriba. Ker ne vidimo okolice, lahko naredimo majhne poskusne korake v različne smeri in preverimo, v kateri smeri se teren najbolj spušča. Nato naredimo korak v to smer in postopek ponavljamo.

Če se spomnimo definicije odvoda, opazimo, da je takšno preverjanje smeri tesno povezano z njim. Pri funkciji več spremenljivk podobno vlogo prevzame **gradient**: pove nam smer najhitrejšega naraščanja funkcije. Če se zato premaknemo v nasprotni smeri gradienta, se vrednost funkcije najhitreje zmanjšuje.

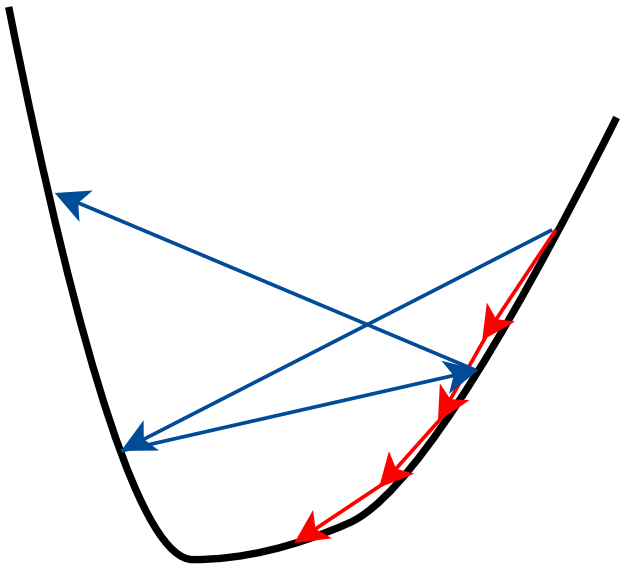
Na tej ideji temelji **gradientni spust** (angl. *gradient descent*). Parametre postopoma spreminjamo v smeri, v kateri se vrednost funkcije izgube zmanjšuje:

$$\theta_j := \theta_j - \eta \frac{\partial J}{\partial \theta_j}, \tag{27}$$

kjer $\eta$ predstavlja **stopnja učenja** (angl. *learning rate*). Postopek večkrat ponovimo: izračunamo gradient pri trenutnih parametrih, naredimo korak v nasprotni smeri in tako postopoma iščemo minimum funkcije izgube.

Če je stopnja učenja premajhna, učenje poteka počasi. Če je prevelika, lahko algoritem minimum preskoči ali se od njega celo oddaljuje.

Ilustracija gradientnega spusta je prikazana na Sliki 2.



Slika 2: Ilustracija gradientnega spusta s primerno stopnjo učenja (rdeča) in preveliko stopnjo (modra).

Pri številnih funkcijah izgube lahko gradient določimo analitično. Sodobna orodja za strojno in globoko učenje, kot je PyTorch, pa podpirajo tudi **avtomatsko odvajanje** (angl. *automatic differentiation*, *autograd*), ki gradient izračuna samodejno iz zaporedja izvedenih operacij. Pri preprostih modelih, kot je linearna regresija, PyTorch običajno ni potreben, saj lahko parametre poiščemo tudi z enostavnejšimi metodami oziroma orodji.

**Čemu toliko podrobnosti o gradientnem spustu?**

Gradientni spust je eden izmed temeljev sodobnega globokega učenja. Ključno vlogo pri učenju nevronskih mrež ima **vzvratno širjenje napake** (angl. *backpropagation*), ki z uporabo verižnega pravila učinkovito izračuna gradiente skozi več zaporednih plasti.
Tako gradientov ni treba ročno izpeljevati za vsak parameter posebej ali jih približevati z numeričnim odvajanjem. Ko so gradienti izračunani, jih lahko gradientni spust uporabi za posodabljanje velikega števila parametrov modela.

### 4.4.3 Polinomska regresija

Povezava med vhodom in ciljno vrednostjo ni vedno dobro opisana s premico. Takrat lahko iz prvotnih značilk ustvarimo nove, na primer $x^2$, $x^3$ in višje potence.

Dobimo **polinomsko regresijo** (angl. *polynomial regression*):

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \cdots . \tag{28}$$

Čeprav je dobljena funkcija glede na $x$ lahko ukrivljena, model še vedno obravnavamo kot linearni model, saj je linearen v parametrih $\theta$. Potence $x^2$, $x^3$ in podobno lahko razumemo kot dodatne značilke.

**Pozor**

Polinomi višjih stopenj omogočajo zelo prilagodljive modele, vendar se lahko preveč natančno prilagodijo učnim podatkom in šumu. S povečevanjem kompleksnosti zato narašča nevarnost pretiranega prileganja (angl. *overfitting*).

## 4.5 Logistična regresija

Kljub imenu je **logistična regresija** (angl. *logistic regression*) klasifikacijski algoritem. Najpogosteje jo uporabljamo za binarno klasifikacijo.

Osnova logistične regresije je **sigmoidna funkcija** (angl. *sigmoid function*), ki jo poznamo tudi kot logistično funkcijo, od koder izvira tudi ime logistične regresije:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{29}$$

Njena vrednost je vedno med 0 in 1, zato jo lahko uporabimo za izražanje verjetnosti. Za velike pozitivne vrednosti $z$ se vrednost funkcije približuje 1, za velike negativne vrednosti pa 0. Pri $z = 0$ velja $\sigma(z) = 0{,}5$.

Logistična regresija linearno kombinacijo značilk $\theta^T x$ preslika skozi sigmoidno funkcijo:

$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}. \tag{30}$$

Rezultat lahko interpretiramo kot ocenjeno verjetnost, da primer pripada pozitivnemu razredu:

$$h_\theta(x) = P(y = 1 \mid x; \theta). \tag{31}$$

Če model na primer vrne $h_\theta(x) = 0{,}8$, to interpretiramo kot ocenjeno verjetnost 80 %, da primer pripada pozitivnemu razredu.

Za končno odločitev izberemo **prag** (angl. *threshold*). Pri pragu 0,5 na primer velja

$$\hat{y} = \begin{cases} 1, & h_\theta(x) \geq 0{,}5, \\ 0, & h_\theta(x) < 0{,}5. \end{cases} \tag{32}$$

Praga ni nujno vedno postaviti na 0,5. Izbira je odvisna od problema in posledic različnih vrst napak. Če prag znižamo, bo model več primerov razvrstil v pozitivni razred; če ga zvišamo, bo pri pozitivni napovedi bolj zahteven.

### 4.5.1 Meja odločanja

**Meja odločanja** (angl. *decision boundary*) določa, kje se napoved modela spremeni iz enega razreda v drugega.

Pri pragu 0,5 sigmoidna funkcija spremeni razred natanko pri

$$h_\theta(x) = 0{,}5. \tag{33}$$

Ker velja $\sigma(0) = 0{,}5$, je meja odločanja določena z

$$\theta^T x = 0. \tag{34}$$

Pri dveh značilkah lahko to zapišemo kot

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0, \tag{35}$$

kar predstavlja premico v prostoru značilk. Primeri na eni strani premice so razvrščeni v en razred, primeri na drugi strani pa v drugega.

**Ključna ideja**

Logistična regresija je sestavljena iz dveh preprostih korakov: najprej izračuna linearno kombinacijo značilk, nato pa jo s sigmoidno funkcijo preslika na interval med 0 in 1. Dobljeno vrednost interpretiramo kot verjetnost, prag pa jo pretvori v končno napoved razreda.

### 4.5.2 Funkcija izgube

Za učenje logistične regresije potrebujemo funkcijo izgube, ki meri kakovost napovedanih verjetnosti. Običajno uporabimo **binarno navzkrižno entropijo** (angl. *binary cross-entropy*), oziroma **logaritemsko izgubo** (angl. *log loss*):

$$L(y, \hat{p}) = -\left[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\right], \tag{36}$$

kjer je $\hat{p} = h_\theta(x)$ napovedana verjetnost pozitivnega razreda.

Če je pravi razred $y = 1$, se izraz poenostavi v

$$L(1, \hat{p}) = -\log(\hat{p}), \tag{37}$$

zato je izguba majhna, kadar model pravilno napove verjetnost blizu 1, oziroma visoka če napačno napove 0. Če je $y = 0$, dobimo

$$L(0, \hat{p}) = -\log(1 - \hat{p}), \tag{38}$$

zato je izguba majhna, kadar model napove verjetnost blizu 0 in visoka če je napoved napačno blizu 1.

Za celotno učno množico izračunamo povprečno izgubo:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)})\log\left(1-h_\theta(x^{(i)})\right)\right]. \tag{39}$$

Parametre $\theta$ nato prilagajamo tako, da to funkcijo zmanjšujemo, na primer z gradientnim spustom.

## 4.6 Posploševanje in kompleksnost modela

Osrednji cilj nadzorovanega učenja je **posploševanje** (angl. *generalization*): model mora dobro napovedovati ne le za učne primere, temveč tudi na novih podatkih iz podobne porazdelitve.

Če je model preprost in ne more zajeti pomembnih vzorcev v podatkih, govorimo o **premajhnem prileganju** (angl. *underfitting*). Tak model običajno dosega slabe rezultate že na učnih podatkih.

Če pa je model preveč prilagodljiv, se lahko nauči tudi naključnega šuma in posebnosti učne množice. Temu pravimo **pretirano prileganje** (angl. *overfitting*). Tak model je lahko zelo uspešen na učnih podatkih, vendar bistveno slabši na novih primerih.

**Ključna ideja**

Cilj ni model, ki se učnim podatkom prilagodi čim bolj popolno, temveč model, ki iz njih odkrije vzorce, ki se prenesejo tudi na nove podatke.

### 4.6.1 Regularizacija

Eden izmed načinov zmanjševanja pretiranega prileganja je **regularizacija** (angl. *regularization*), pri kateri funkciji izgube dodamo kazen za preveliko kompleksnost modela.

Pri **L2-regularizaciji** (angl. *L2 regularization*, tudi *Ridge*) uporabimo

$$J_{L2}(\theta) = J(\theta) + \lambda\sum_{j=1}^{n}\theta_j^2. \tag{40}$$

Pri **L1-regularizaciji** (angl. *L1 regularization*, tudi *Lasso*) uporabimo

$$J_{L1}(\theta) = J(\theta) + \lambda\sum_{j=1}^{n}|\theta_j|. \tag{41}$$

Hiperparameter $\lambda$ določa moč regularizacije. Večja vrednost pomeni močnejšo kazen za velike parametre. Če je regularizacija premočna, lahko model postane preveč preprost in pride do premajhnega prileganja.

L2-regularizacija običajno zmanjšuje velikost vseh parametrov, L1-regularizacija pa lahko nekatere parametre zmanjša natančno na nič in s tem opravi tudi izbiro značilk.

## 4.7 Metoda k najbližjih sosedov

**Metoda $k$ najbližjih sosedov** (angl. *$k$-nearest neighbors*, $k$-NN) napoved za nov primer določi na podlagi učnih primerov, ki so mu najbolj podobni.

Pri klasifikaciji poiščemo $k$ najbližjih učnih primerov in napovemo razred, ki se med njimi pojavi najpogosteje (lahko tudi uteženo oziroma bolj napredno). Pri regresiji lahko na primer napovemo povprečje njihovih ciljnih vrednosti.

Pomembna izbira je vrednost $k$. Majhen $k$ omogoča zelo prilagodljivo mejo odločanja, vendar je lahko občutljiv na šum. Večji $k$ daje stabilnejše, vendar bolj zglajene napovedi.

**Pozor**

Metoda $k$-NN temelji na razdaljah med primeri, zato je občutljiva na različne mere razdalj. Numerične značilke je zato praviloma treba ustrezno skalirati.

## 4.8 Metoda podpornih vektorjev

**Metoda podpornih vektorjev** (angl. *support vector machine*, SVM) je klasifikacijski algoritem, ki k problemu pristopa nekoliko drugače kot logistična regresija. Namesto neposrednega modeliranja verjetnosti razredov išče mejo odločanja, ki razrede loči z največjim možnim robom.

### 4.8.1 Največji rob

Predpostavimo najprej, da imamo binarni klasifikacijski problem, pri katerem lahko razreda popolnoma ločimo s premico oziroma, v več dimenzijah, s **hiperravnino** (angl. *hyperplane*).

Takšnih ločilnih meja je lahko veliko. SVM med njimi izbere tisto, ki je čim dlje od najbližjih učnih primerov obeh razredov.

To razdaljo imenujemo **rob** (angl. *margin*).

**Ključna ideja**

SVM ne išče zgolj meje, ki pravilno loči učne primere, temveč med možnimi mejami išče tisto z **največjim robom**.
Intuicija je, da je takšna meja bolj robustna na majhne spremembe oziroma šum v podatkih in zato lahko bolje posplošuje.

### 4.8.2 Podporni vektorji

Učne primere, ki ležijo najbližje meji odločanja, imenujemo **podporni vektorji** (angl. *support vectors*).

Prav ti primeri določajo položaj optimalne meje. Če odstranimo učni primer, ki leži daleč od roba, se meja praviloma ne spremeni. Če pa odstranimo podporni vektor, se lahko optimalna meja spremeni.

To je pomembna lastnost SVM: končna rešitev je določena predvsem z relativno majhnim podmnožico učnih primerov ob meji med razredi.

### 4.8.3 Mehki rob in parameter C

V resničnih podatkih razredi pogosto niso popolnoma linearno ločljivi. Posamezni primeri so lahko na "napačni" strani meje ali pa se razreda deloma prekrivata.

Če bi zahtevali popolno ločitev vseh učnih primerov, bi bil model lahko zelo občutljiv na šum oziroma takšna meja sploh ne bi obstajala.

Zato uporabljamo **mehki rob** (angl. *soft margin*), pri katerem dovolimo nekatere kršitve roba in celo napačne razvrstitve.

SVM mora tako uravnotežiti dva cilja:

- čim večji rob;

- čim manj kršitev oziroma napačnih razvrstitev.

Linearno mejo odločanja lahko zapišemo kot

$$w^T x + b = 0, \tag{42}$$

kjer je $w$ vektor uteži, ki določa orientacijo meje, $b$ pa njen premik. Velikost roba je obratno sorazmerna z normo $\|w\|$, zato večji rob dosežemo z zmanjševanjem $\|w\|$.

Pri mehkem robu optimizacijski problem zapišemo kot

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \\ \text{pri pogojih} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \end{aligned} \tag{43}$$

kjer je $y_i \in \{-1, +1\}$ oznaka $i$-tega učnega primera, $\xi_i$ meri kršitev roba, parameter $C$ pa določa, kako močno takšne kršitve kaznujemo.

Pri običajnem zapisu SVM je celotna širina roba

$$\frac{2}{\|w\|}. \tag{44}$$

Zato zmanjševanje $\|w\|$ ustreza povečevanju roba.

- **Velik $C$:** kršitve močno kaznujemo. Model poskuša pravilno razvrstiti skoraj vse učne primere, pogosto za ceno manjšega roba. Pri šumnih podatkih lahko to vodi v pretirano prileganje.
- **Majhen $C$:** dovolimo več kršitev in damo večji poudarek širokemu robu. Model je lahko bolj robusten, vendar lahko pri prenizkem $C$ pride do premajhnega prileganja.

**Vloga parametra $C$**

Parameter $C$ določa kompromis med **širokim robom** in **napakami na učnih podatkih**.
Velik $C$ daje večji poudarek pravilni klasifikaciji učnih primerov, majhen $C$ pa večjemu robu.

### 4.8.4 Jedrne metode in nelinearne meje

Kaj pa, če razredov ni mogoče ločiti z ravno črto oziroma linearno hiperravnino?

SVM lahko uporablja **jedrni trik** (angl. *kernel trick*). Osnovna ideja je, da podatke obravnavamo, kot da bi jih preslikali v prostor z več dimenzijami, kjer lahko prvotno nelinearen problem postane linearno ločljiv.

Predstavljajmo si na primer podatke v ravnini, kjer en razred tvori majhen krog okoli izhodišča, drugi pa večji krog okoli njega. V dveh dimenzijah teh dveh razredov ne moremo ločiti s premico.

Če pa dodamo novo značilko

$$z = x_1^2 + x_2^2, \tag{45}$$

ki predstavlja kvadrat oddaljenosti od izhodišča, lahko postaneta razreda v novem prostoru linearno ločljiva.

Pri jedrnem triku take preslikave običajno ne izračunamo eksplicitno. Namesto tega uporabimo **jedrno funkcijo** (angl. *kernel function*), ki neposredno izračuna ustrezno podobnost med dvema primeroma.

Nekatera pogosta jedra so:

**Linearno jedro.**

$$K(x, z) = x^T z. \tag{46}$$

To ustreza običajnemu linearnemu SVM.

**Polinomsko jedro.**

$$K(x, z) = (x^T z + c)^d. \tag{47}$$

Takšno jedro omogoča modeliranje polinomskih povezav stopnje $d$.

**Jedro RBF.** Pogosta izbira je tudi jedro radialne bazne funkcije (angl. *radial basis function*, RBF), imenovano tudi Gaussovo jedro:

$$K(x, z) = \exp\left(-\gamma \|x - z\|^2\right). \tag{48}$$

Parameter $\gamma$ določa, kako hitro vpliv podobnosti med primeroma pada z njuno razdaljo. Jedro RBF lahko ustvari zelo kompleksne nelinearne meje odločanja.

**Jedrni trik**

Namesto da bi podatke eksplicitno preslikali v potencialno zelo visokodimenzionalen prostor, jedrna funkcija neposredno izračuna skalarni produkt, ki bi ga dobili v tem prostoru.
Tako lahko linearni algoritem v transformiranem prostoru ustreza nelinearni meji v prvotnem prostoru.

## 4.9 Odločitvena drevesa

**Odločitveno drevo** (angl. *decision tree*) modelira napoved kot zaporedje preprostih odločitev o značilkah.

Pri klasifikaciji drevo prostor značilk postopoma razdeli na območja, ki pripadajo posameznim razredom. Ena glavnih prednosti odločitvenih dreves je njihova razložljivost: za posamezno napoved lahko sledimo poti skozi drevo in vidimo, katere odločitve so pripeljale do rezultata. Tu gre omeniti, da kot celota drevo hitro postane nerazložljivo z večjo globino.

### 4.9.1 Zgradba drevesa

Drevo sestavljajo:

- **koren** (angl. *root node*), kjer izvedemo prvo odločitev;
- **notranja vozlišča** (angl. *internal nodes*), ki vsebujejo nadaljnje pogoje;
- **listi** (angl. *leaf nodes*), ki vsebujejo končno napoved.

Pri numerični značilki lahko notranje vozlišče na primer preverja pogoj

$$x_j > t, \tag{49}$$

kjer je $x_j$ izbrana značilka, $t$ pa prag.

Glede na rezultat pogoja sledimo eni od vej in postopek ponavljamo, dokler ne pridemo do lista.

### 4.9.2 Izbira najboljše delitve

Ključno vprašanje pri gradnji drevesa je:

*Po kateri značilki in pri katerem pragu naj podatke razdelimo?*

Dobra delitev naj ustvari skupine, v katerih so primeri posameznih razredov čim manj pomešani.

Za merjenje **nečistosti** (angl. *impurity*) množice lahko uporabimo **entropijo** (angl. *entropy*):

$$H(S) = -\sum_{c} p_c \log_2(p_c), \tag{50}$$

kjer je $p_c$ delež primerov razreda $c$ v množici $S$.

Če vsi primeri pripadajo istemu razredu, velja

$$H(S) = 0. \tag{51}$$

Pri binarni klasifikaciji je entropija največja, kadar je polovica primerov pozitivnih in polovica negativnih:

$$H(S) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1. \tag{52}$$

Razpon [0, 1] velja zgolj v binarnem primeru.

Dobra delitev mora entropijo zmanjšati. To zmanjšanje merimo z **informacijskim dobičkom** (angl. *information gain*):

$$\text{Gain}(S, A) = H(S) - \sum_{veje} \frac{|S_v|}{|S|} H(S_v), \tag{53}$$

kjer $S_v$ predstavlja podmnožico primerov, ki po delitvi končajo v veji $v$.

Drugi člen predstavlja uteženo povprečje entropij po delitvi. Veje z več primeri imajo zato večji vpliv na skupno oceno.

Večji informacijski dobiček pomeni, da je delitev bolj zmanjšala negotovost glede razreda.

**Primer: izračun informacijskega dobička**

Imamo 10 učnih primerov:

$$\text{5 pozitivnih (+)}, \qquad \text{5 negativnih (-)}. \tag{54}$$

Ker sta razreda enako zastopana, je začetna entropija

$$H(S) = -\frac{5}{10}\log_2\frac{5}{10} - \frac{5}{10}\log_2\frac{5}{10} = 1. \tag{55}$$

Predpostavimo, da delitev po značilki $A$ ustvari dve veji:

$$S_1 = (4+, 1-), \qquad S_2 = (1+, 4-). \tag{56}$$

Entropija prve veje je

$$H(S_1) = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} \approx 0{,}72. \tag{57}$$

Zaradi simetrije velja tudi

$$H(S_2) \approx 0{,}72. \tag{58}$$

Utežena entropija po delitvi je

$$\frac{5}{10} \cdot 0{,}72 + \frac{5}{10} \cdot 0{,}72 = 0{,}72. \tag{59}$$

Informacijski dobiček je zato

$$\mathrm{Gain}(S, A) = 1 - 0{,}72 = 0{,}28. \tag{60}$$

Delitev je torej zmanjšala negotovost za približno 0,28 bita.

### 4.9.3 Gradnja drevesa

Eden klasičnih algoritmov za gradnjo odločitvenih dreves je **ID3**. Ta je bil prvotno zasnovan predvsem za kategorične značilke in kot kriterij uporablja informacijski dobiček.

Pri sodobnih implementacijah odločitvenih dreves pogosto srečamo tudi algoritem CART (angl. *Classification and Regression Trees*), ki uporablja binarne delitve. Pri numerični značilki so te lahko oblike

$$x_j \leq t. \tag{61}$$

Namesto entropije lahko kot mero nečistosti uporabimo tudi **Gini indeks**:

$$G(S) = 1 - \sum_c p_c^2. \tag{62}$$

Osnovna ideja je pri obeh pristopih podobna: na vsakem koraku iščemo delitev, ki nastali podmnožici čim bolj očisti.

ID3 dsrevo gradi rekurzivno od korena navzdol:

1. Če vsi primeri v trenutni množici pripadajo istemu razredu, ustvarimo list s tem razredom.
2. Če ni več značilk, po katerih bi lahko delili, ustvarimo list z večinskim razredom.
3. Sicer izberemo značilko z največjim informacijskim dobičkom.
4. Podatke po tej značilki razdelimo na veje in na vsaki veji postopek rekurzivno ponovimo.

ID3 je **požrešen algoritem** (angl. *greedy algorithm*): na vsakem koraku izbere trenutno najboljšo delitev. To ne zagotavlja, da bo končno drevo globalno optimalno, vendar pristop v praksi pogosto dobro deluje.

**Ključna ideja**

Odločitveno drevo gradimo od zgoraj navzdol. Na vsakem koraku izberemo delitev, ki trenutno najbolj zmanjša nečistost podatkov, nato pa isti postopek ponovimo na nastalih podmnožicah.

### 4.9.4 Omejevanje kompleksnosti in obrezovanje

Če drevesu dovolimo, da raste brez omejitev, lahko ustvari zelo specifična pravila, ki skoraj popolnoma opišejo učne podatke. Takšno drevo ima visoko varianco in se lahko močno pretirano prilega.

Kompleksnost lahko omejujemo na dva glavna načina.

Pri **predhodnem obrezovanju** (angl. *pre-pruning*) rast drevesa ustavimo dovolj zgodaj. Uporabimo lahko na primer:

- največjo dovoljeno globino;
- najmanjše število primerov v listu;
- najmanjši informacijski dobiček, potreben za novo delitev.

Pri **naknadnem obrezovanju** (angl. *post-pruning*) najprej zgradimo večje drevo, nato pa odstranimo veje, ki ne izboljšujejo uspešnosti na validacijskih podatkih.

**Pozor**

Zelo globoko drevo lahko doseže skoraj popoln rezultat na učnih podatkih, vendar slabo posplošuje. Obrezovanje omeji kompleksnost drevesa in s tem zmanjšuje nevarnost pretiranega prileganja.

### 4.9.5 Ansambli dreves

Posamezno odločitveno drevo ima pogosto visoko varianco: že majhna sprememba učne množice lahko povzroči precej drugačno drevo.

To slabost lahko zmanjšamo tako, da združimo napovedi več dreves. Takšni metodi pravimo **ansambelska metoda** (angl. *ensemble method*).

**Bagging.** Pri metodi **bagging** oziroma **bootstrap agregiranju** (angl. *bootstrap aggregating*) naučimo več dreves na različnih naključnih vzorcih učne množice, vzorčenih z vračanjem.

Ker posamezna drevesa vidijo nekoliko različne podatke, delajo različne napake. Z združevanjem njihovih napovedi zmanjšamo varianco.

**Naključni gozd. Naključni gozd** (angl. *Random Forest*) bagging dodatno razširi tako, da pri vsaki delitvi posamezno drevo ne izbira med vsemi značilkami, ampak samo med naključno izbrano podmnožico značilk.

S tem drevesa postanejo med seboj bolj različna, kar lahko dodatno izboljša korist združevanja njihovih napovedi.

**Spodbujanje.** Pri **spodbujanju** (angl. *boosting*) dreves ne učimo neodvisno, temveč zaporedno. Vsak naslednji model se osredotoči na napake predhodnih modelov.

Pri algoritmu AdaBoost se poveča utež napačno razvrščenih primerov, tako da jim naslednje drevo posveti več pozornosti.

Pri **gradientnem spodbujanju** (angl. *gradient boosting*) novo drevo modelira napake oziroma ostanke (angl. *residuals*) trenutnega ansambla.

Spodbujanje pogosto dosega zelo dobre rezultate, vendar je lahko občutljivejše na šum in zahteva skrbnejšo izbiro hiperparametrov kot bagging.

**Zakaj ansambli delujejo?**

Posamezno drevo lahko zaradi posebnosti učne množice naredi precej drugačno napoved. Če združimo več dovolj različnih dreves, se lahko njihove posamezne napake deloma izničijo.
Zato je ansambel pogosto stabilnejši od posameznega drevesa.

## 4.10 Naivni Bayesov klasifikator

**Naivni Bayesov klasifikator** (angl. *Naïve Bayes classifier*) je verjetnostni klasifikator, ki temelji na Bayesovem izreku:

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)}. \tag{63}$$

Pri klasifikaciji želimo določiti verjetnost posameznega razreda $y$ ob znanih značilkah $x$.

Naivni Bayes uvede poenostavitveno predpostavko, da so značilke ob znanem razredu med seboj pogojno neodvisne (od tod tudi ime naivni, saj je to naivna predpostavka):

$$P(x \mid y) = \prod_{i=1}^{n} P(x_i \mid y). \tag{64}$$

Napovemo razred z največjo vrednostjo

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y). \tag{65}$$

Predpostavka neodvisnosti je v resničnih podatkih pogosto napačna, vendar metoda kljub temu lahko dobro deluje, zlasti pri problemih z velikim številom značilk, na primer pri klasifikaciji besedila.

## 4.11 Primerjava algoritmov

Izbira primernega algoritma zahteva razumevanje njegovih predpostavk, prednosti in tipičnih načinov, na katere lahko odpove. Noben algoritem ni najboljši za vse probleme.

| Algoritem | Predpostavke | Tipične težave | Prednosti | Razložljiv? |
|---|---|---|---|---|
| Linearna regresija | Linearna povezava | Nelinearni vzorci; osamelci lahko močno vplivajo na prileganje | Hitra; obstaja analitična rešitev; koeficienti imajo jasen pomen | Da |
| Logistična regresija | Približno linearna meja odločanja; značilke so približno neodvisne | Nelinearne meje; multikolinearnost | Verjetnostni izhod; enostavna regularizacija | Da |
| Odločitveno drevo | Brez posebnih parametričnih predpostavk | Globoka drevesa se lahko pretirano prilegajo; meje so poravnane z osmi | Obravnava različne tipe značilk; skaliranje ni potrebno | Da, predvsem pri plitvem drevesu |
| Linearni SVM | Podatki so približno linearno ločljivi; mehki rob (angl. *soft margin*) dopušča izjeme | Zelo nelinearni podatki; počasnejši pri velikih podatkovnih množicah, občutljivost na skaliranje | Meja z največjim robom lahko dobro posplošuje | Delno |
| SVM z jedrom RBF | Podatki imajo lokalno strukturo | Veliko značilk; občutljivost skaliranje in na hiperparametra $\gamma$ in $C$ | Lahko modelira kompleksne meje odločanja | Ne |
| Naivni Bayes | Značilke so pogojno neodvisne | Korelirane značilke; zvezne značilke zahtevajo dodatne predpostavke | Hiter; lahko dobro deluje z malo podatki in veliko značilkami | Da |

**Ključna ideja**

Izbira algoritma je odvisna od lastnosti podatkov in problema. Pomembni so med drugim vrsta ciljne spremenljivke, količina podatkov, linearna ali nelinearna struktura, občutljivost na skaliranje ter potreba po razložljivosti modela. Včasih je bolje dobro poznati en algoritem (recimo logisična regresija) in kako razne probleme najbolje pripraviti, da bo ta dobro deloval.

**Kdaj uporabiti kaj?**

- **Začnimo preprosto:** linearna oziroma logistična regresija sta pogosto dobra osnovna modela (angl. *baseline models*).
- **Če potrebujemo razložljivost:** uporabimo (plitka) odločitvena drevesa ali regularizirane linearne modele.
- **Pri nelinearnih vzorcih:** uporabimo SVM z jedrom RBF ali ansamble dreves, kot je naključni gozd (angl. *Random Forest*).
- **Pri različnih tipih značilk:** odločitvena drevesa lahko naravno obravnavajo različne vrste značilk in praviloma ne zahtevajo skaliranja.

### 4.11.1 Primerjava klasifikatorjev

| Metoda | Meja odločanja | Primerna predvsem za | Veliko značilk? | Razložljiv? |
|---|---|---|---|---|
| Logistična regresija | Linearna hiperravnina | Linearno ločljive podatke; verjetnostne napovedi | Da, z regularizacijo | Da, prek koeficientov |
| Linearni SVM | Linearna meja z največjim robom | Visokodimenzionalne podatke z jasno ločitvijo | Da | Delno |
| SVM z jedrom | Nelinearna | Kompleksne meje; srednje velike podatkovne množice | Odvisno | Ne |
| Odločitveno drevo | Območja, poravnana s koordinatnimi osmi | Različne tipe značilk; probleme, kjer je pomembna razložljivost | Lahko ima težave | Da, pri plitvem drevesu |
| Naivni Bayes | Linearna / Kvadratna | Klasifikacijo besedila; približno neodvisne značilke | Da | Da, prek verjetnosti |

**Pogoste napake**

- **SVM brez skaliranja značilk:** SVM je občutljiv na različna merila značilk, zato podatke pred učenjem praviloma standardiziramo.
- **Pregloboko odločitveno drevo brez omejevanja:** globoko drevo si lahko zapomni učne podatke in se pretirano prilega.
- **Naivni Bayes pri močno koreliranih značilkah:** s tem kršimo predpostavko pogojne neodvisnosti. Ocenjene verjetnosti so lahko zato napačne, čeprav je klasifikacija še vedno lahko uspešna.
- **Logistična regresija pri nelinearni meji:** osnovna logistična regresija uporablja linearno mejo odločanja in lahko tak problem premajhno prilega. Primernejši so lahko SVM z jedrom ali odločitvena drevesa.

## 4.12 Povzetek

**Ključne ugotovitve**

- Pri **nadzorovanem učenju** se model uči iz parov vhodov in pravilnih izhodov ter skuša dobro napovedovati tudi nove primere.
- Pri **regresiji** napovedujemo numerične vrednosti, pri **klasifikaciji** pa diskretne razrede.
- **Funkcija izgube** meri napako napovedi, učenje pa išče parametre, ki to izgubo zmanjšajo.
- **Linearna regresija** modelira numerični cilj kot linearno kombinacijo značilk, **logistična regresija** pa s sigmoidno funkcijo rešuje klasifikacijske probleme.
- Cilj učenja je **posploševanje**. Preprost model lahko vodi do premajhnega prileganja, preveč kompleksen pa do pretiranega prileganja.
- **Regularizacija** omejuje kompleksnost modela in pomaga preprečevati pretirano prileganje.
- **SVM** išče mejo z velikim robom, **odločitvena drevesa** gradijo zaporedje pravil, **ansambelske metode** združujejo več modelov, **Naivni Bayes** pa temelji na verjetnostnem modelu.
- Noben algoritem ni najboljši za vse probleme. Izbira metode je odvisna od podatkov, cilja in zahtev problema.

## 4.13 Naloge

1. **Linearna regresija**

   Model linearne regresije napoveduje ceno hiše po enačbi

   $$h(x) = 150\,000 + 5000x, \tag{66}$$

   kjer je $x$ površina hiše v kvadratnih metrih.

   Kakšno ceno model napove za hišo s površino 160 kvadratnih metrov? Kako interpretiramo oba koeficienta modela?

2. **Logistična regresija**

   Model logistične regresije vrne

   $$h_\theta(x) = 0{,}73. \tag{67}$$

   Kateri razred model napove pri pragu 0,5? Kaj predstavlja vrednost 0,73?

3. **Informacijski dobiček**

   Pri gradnji odločitvenega drevesa razdelitev po značilki $A$ ustvari dve veji, ki primere razdeli v dve podmnožici:

   $$(4 \text{ pozitivni}, 0 \text{ negativnih}) \qquad \text{in} \qquad (1 \text{ pozitiven}, 5 \text{ negativnih}). \tag{68}$$

   Začetna entropija je enaka 1,0.

   Izračunajte **informacijski dobiček** (angl. *information gain*) te razdelitve.

4. **SVM in rob**

   Pojasnite, zakaj SVM imenujemo **klasifikator z velikim robom** (angl. *large margin classifier*).

   Kako parameter $C$ vpliva na velikost roba in dovoljevanje napačnih razvrstitev?

5. **Regularizacija**

   Primerjajte regularizaciji L1 in L2.

   Kako vplivata na uteži modela? V katerih primerih bi dali prednost eni oziroma drugi?

6. **IAIO-slog: algoritem, ki ga še niste videli**

   Sodelavec predlaga **uteženi klasifikator sosedov** (angl. *Weighted Neighbor Classifier*, WNC).

   Za testni primer $x$ algoritem:

   - izračuna razdaljo $d_i$ do vsakega učnega primera $x_i$;
   - vsakemu učnemu primeru dodeli utež

   $$w_i = \exp\left(-\frac{d_i^2}{\sigma^2}\right); \tag{69}$$

- napove

$$\hat{y} = \text{sign}\left(\sum_i w_i y_i\right), \qquad y_i \in \{-1, +1\}. \tag{70}$$

Odgovorite:

(a) Kaj se zgodi, ko $\sigma \to 0$? Kaj se zgodi, ko $\sigma \to \infty$?

(b) Kako je ta algoritem povezan s $k$-NN in jedrnimi metodami (angl. *kernel methods*)?

(c) Kakšne oblike je njegova meja odločanja? Ali je nujno linearna?

(d) Na kakšne načine lahko algoritem odpove?

(e) Predlagajte spremembo, s katero bi bolje obravnavali neuravnotežene razrede (angl. *class imbalance*).

7. **Problem XOR**

Obravnavajte točke

$$(0,0), \quad (0,1), \quad (1,0), \quad (1,1) \tag{71}$$

z oznakami

$$0, \quad 1, \quad 1, \quad 0. \tag{72}$$

(a) Dokažite, da problema ne more rešiti noben linearni klasifikator.

(b) Pokažite, da z dodajanjem značilke

$$x_3 = x_1 x_2 \tag{73}$$

problem postane linearno ločljiv.

(c) Narišite odločitveno drevo, ki pravilno reši problem XOR.

(d) Kako je ta primer povezan z **jedrnim trikom** (angl. *kernel trick*)?

8. **Tehnično-etični primer**

Logistična regresija, ki jo uporablja banka, vsebuje značilko *število let na trenutnem naslovu*. Njen koeficient je negativen, kar pomeni, da daljše bivanje na istem naslovu model povezuje z nižjim tveganjem neplačila.

Ta značilka je hkrati povezana s starostjo.

(a) Zakaj bi lahko uporaba te značilke povzročila starostno diskriminacijo, čeprav starost sama ni vhodna značilka?

(b) Po odstranitvi značilke AUC pade z 0,82 na 0,79. Opišite nastali kompromis.

(c) Katere metrike pravičnosti (angl. *fairness metrics*) bi izračunali za dodatno analizo? Razmislite na primer o deležu pozitivnih odločitev, stopnjah posameznih vrst napak in kalibraciji napovedi za različne skupine.

9. **IAIO-slog: še en algoritem, ki ga niste videli**

   Obravnavajte **prototipni klasifikator** (angl. *Prototype Classifier*).

   Za vsak razred $c$ najprej izračunamo središče oziroma povprečje vseh učnih primerov tega razreda:

   $$\mu_c = \frac{1}{|C_c|} \sum_{x \in C_c} x. \tag{74}$$

   Nov primer $x$ nato razvrstimo v razred, katerega središče mu je najbližje:

   $$\hat{y} = \arg\min_c \|x - \mu_c\|. \tag{75}$$

   Odgovorite:

   (a) Kakšne oblike je meja odločanja med dvema razredoma? Pomagajte si s pogojem

   $$\|x - \mu_1\| = \|x - \mu_2\|. \tag{76}$$

   (b) Kako se ta metoda primerja s $k$-NN, če bi pri $k$-NN upoštevali vse učne primere?

   (c) Kakšne lastnosti podatkov bi morale veljati, da bi ta klasifikator dobro deloval?

   (d) Sestavite preprost dvodimenzionalni primer podatkov, pri katerem prototipni klasifikator zelo slabo deluje. Katera lastnost podatkov povzroči težavo?

   (e) Koliko parametrov mora klasifikator shraniti? Primerjajte ga s $k$-NN in logistično regresijo.

10. **En korak gradientnega spusta**

    Za model

    $$h_\theta(x) = \theta_0 + \theta_1 x \tag{77}$$

    naj bo trenutno

    $$\theta_0 = 1, \qquad \theta_1 = 2. \tag{78}$$

    Gradient funkcije izgube pri trenutnih parametrih je

    $$\nabla J(\theta) = \begin{pmatrix} 2 \\ -4 \end{pmatrix}, \tag{79}$$

    stopnja učenja pa je

    $$\eta = 0{,}1. \tag{80}$$

    Izvedite en korak gradientnega spusta in izračunajte novi vrednosti $\theta_0$ in $\theta_1$.

11. **Več značilk in polinomska regresija**

Model ima obliko

$$h_\theta(x) = 3 + 2x_1 - 4x_2. \tag{81}$$

(a) Izračunajte napoved za $x_1 = 5$ in $x_2 = 2$.

(b) Kako bi model razširili, če bi želeli zajeti kvadratno odvisnost od $x_1$?

(c) Zakaj model, ki vsebuje člen $x_1^2$, še vedno imenujemo linearen glede na parametre?

(d) Kakšna težava se lahko pojavi, če dodamo zelo veliko polinomskih členov visoke stopnje?

12. **Logistična regresija in logaritemska izguba**

Model za binarno klasifikacijo pravemu razredu $y = 1$ pripiše verjetnost

$$\hat{p} = 0{,}9. \tag{82}$$

Drug model istemu primeru pripiše

$$\hat{p} = 0{,}1. \tag{83}$$

Za oba izračunajte binarno navzkrižno entropijo

$$L(y, \hat{p}) = -\left[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\right]. \tag{84}$$

Kateri model je močneje kaznovan in zakaj?

Kaj bi se zgodilo z izgubo, če bi model pravemu razredu $y = 1$ pripisal verjetnost, ki se približuje 0?

13. **Logistična regresija in meja odločanja**

Podan je model

$$h_\theta(x) = \sigma(-4 + 2x_1 + x_2). \tag{85}$$

Pri pragu 0,5:

(a) zapišite enačbo meje odločanja;

(b) določite, na kateri strani meje model napove razred 1;

(c) določite napoved za točko $(x_1, x_2) = (1, 1)$;

(d) določite napoved za točko $(x_1, x_2) = (3, 1)$.

14. **Naivni Bayes**

Želimo razvrstiti elektronsko sporočilo kot neželeno ($S$) ali običajno ($N$).

Poznamo:

$$P(S) = 0{,}2, \qquad P(N) = 0{,}8. \tag{86}$$

V sporočilu se pojavita besedi *free* in *meeting*. Ocenjene pogojne verjetnosti so

$$P(\text{free} \mid S) = 0{,}7, \qquad P(\text{meeting} \mid S) = 0{,}1, \tag{87}$$

ter

$$P(\text{free} \mid N) = 0{,}05, \qquad P(\text{meeting} \mid N) = 0{,}4. \tag{88}$$

Ob predpostavki pogojne neodvisnosti izračunajte količini

$$P(S)P(\text{free} \mid S)P(\text{meeting} \mid S) \tag{89}$$

in

$$P(N)P(\text{free} \mid N)P(\text{meeting} \mid N). \tag{90}$$

V kateri razred bi Naivni Bayes razvrstil sporočilo?

Zakaj za samo izbiro razreda ni treba izračunati normalizacijskega imenovalca iz Bayesovega izreka?

15. **Izbira algoritma**

    Za vsak spodnji problem izberite enega ali več smiselnih algoritmov iz tega poglavja in izbiro utemeljite.

    (a) Želimo preprost in razložljiv model za napovedovanje zvezne vrednosti.

    (b) Imamo visokodimenzionalne redke podatke iz besedila.

    (c) Imamo nelinearno klasifikacijsko mejo in srednje veliko podatkovno množico.

    (d) Potrebujemo model, katerega odločitve lahko predstavimo kot zaporedje preprostih pravil.

    (e) Potrebujemo verjetnostni izhod za binarno klasifikacijo in pričakujemo približno linearno mejo odločanja.

    Pri odgovoru navedite tudi eno pomembno omejitev izbranega algoritma.

**Predlogi za nadaljnje branje:**


1. Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer.
2. G. James, D. Witten, T. Hastie, & R. Tibshirani, *An Introduction to Statistical Learning*, Springer.


# 5 Evalvacija modelov strojnega učenja

**Evalvacija modelov strojnega učenja** (angl. *model evaluation*) je postopek, s katerim ocenimo, kako dobro se naučeni model obnese na novih, še nevidenih podatkih.

Dober rezultat na učnih podatkih sam po sebi ni dovolj. Model si lahko učne primere zelo dobro zapomni, ne da bi se naučil splošnih vzorcev, ki veljajo tudi zunaj učne množice. Osrednje vprašanje evalvacije je zato, kako dobro model **posplošuje** (angl. *generalizes*).

Podobno je pri učenju za izpit. Če učenec vadi sto nalog in nato rešuje natanko istih sto nalog, visok rezultat še ne pove veliko o njegovem razumevanju. Boljši preizkus so nove naloge, pri katerih mora naučeno znanje uporabiti v nekoliko drugačni situaciji.

**Ključna ideja**

Namen strojnega učenja ni doseči čim boljši rezultat na učnih podatkih, temveč izdelati model, ki dobro deluje na novih podatkih.
Evalvacija mora zato meriti **posploševanje**, ne pomnjenja učne množice.

## 5.1 Parametri in hiperparametri

Pred nadaljevanjem je pomembno razlikovati med **parametri** in **hiperparametri** modela.

**Parametri** (angl. *parameters*) so vrednosti, ki se jih model sam nauči iz učnih podatkov. Pri linearni regresiji so to na primer koeficienti $\theta_j$, pri nevronskih mrežah pa uteži povezav med nevroni.

**Hiperparametri** (angl. *hyperparameters*) so nastavitve, ki jih določimo zunaj samega procesa učenja. Mednje sodijo na primer stopnja učenja, jakost regularizacije, največja globina odločitvenega drevesa ali število plasti nevronske mreže.

**Parametri ali hiperparametri?**

**Parametre** se model nauči iz učnih podatkov.
**Hiperparametre** izberemo sami in z njimi določimo, kako bo učenje oziroma model potekal.

Hiperparametre običajno izbiramo na podlagi uspešnosti na validacijski množici oziroma z prečnim preverjanjem, testno množico pa pri tem pustimo nedotaknjeno.

## 5.2 Delitev na učno, validacijsko in testno množico

Temeljno pravilo evalvacije je, da modela ne ocenjujemo na istih podatkih, na katerih se je učil. Model lahko na učnih podatkih dosega zelo visoko uspešnost zgolj zato, ker se jim je močno prilagodil, ne da bi znal dobro napovedovati nove primere.

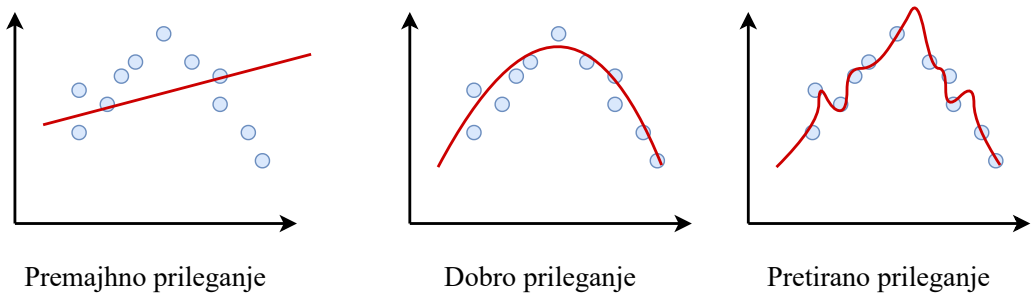
Zato podatke običajno razdelimo na tri ločene množice:

- **učno množico** (angl. *training set*), na kateri model uči svoje parametre;
- **validacijsko množico** (angl. *validation set*), ki jo uporabljamo za izbiro hiperparametrov, primerjavo modelov in druge razvojne odločitve;
- **testno množico** (angl. *test set*), ki jo uporabimo šele na koncu za končno oceno izbranega modela.

Pogosta razmerja so približno 60–80 % podatkov za učenje ter po 10–20 % za validacijo in testiranje. Natančna razmerja so odvisna predvsem od količine podatkov in problema.

**Analogija z izpitom**

Učno množico si lahko predstavljamo kot **učbenik**, iz katerega se učimo.
Validacijska množica predstavlja **poskusne izpite**, s katerimi preverjamo svoj napredek in prilagajamo način učenja.
Testna množica pa predstavlja **končni izpit**. Uporabimo jo šele takrat, ko smo z izbiro modela in njegovih nastavitev že zaključili.




Slika 3: Ilustracija različnih kakovosti prileganja podatkom.

**Ključno pravilo**

**Testnih podatkov ne uporabljamo za učenje ali razvojne odločitve.**
Če pogledamo rezultat na testni množici in nato na njegovi podlagi spremenimo model, izberemo druge značilke ali prilagodimo hiperparametre, je testna množica že vplivala na razvoj modela. V tem primeru ne predstavlja več povsem neodvisne končne ocene.

Pri razvoju modela zato običajno večkrat uporabljamo učno in validacijsko množico, testne podatke pa pustimo nedotaknjene do konca.

Pomembno je tudi, da posamezne množice ustrezno predstavljajo problem, ki ga rešujemo. Pri klasifikaciji lahko uporabimo stratificirano delitev, pri časovno odvisnih podatkih pa moramo podatke deliti po času. Te primere smo podrobneje obravnavali že v poglavju o pripravi podatkov.

## 5.3 Pristranskost in varianca

Napake modela lahko pogosto razumemo skozi kompromis med **pristranskostjo** (angl. *bias*) in **varianco** (angl. *variance*).

Model z visoko pristranskostjo je preveč preprost, da bi zajel pomembne vzorce v podatkih. Temu pravimo **premajhno prileganje** (angl. *underfitting*). Tak model običajno dosega slabe rezultate že na učnih podatkih.

Model z visoko varianco pa je preveč občutljiv na posebnosti svoje učne množice. Lahko se zelo dobro prilagodi učnim podatkom, vendar slabo deluje na novih. Temu pravimo **pretirano prileganje** (angl. *overfitting*).

Vizualno je prilagajanje prikazano na Sliki 3.

**Kako ju prepoznamo?**

Če je uspešnost slaba že na učni množici, gre pogosto za **premajhno prileganje** oziroma visoko pristranskost.
Če je uspešnost na učni množici zelo dobra, na validacijski pa bistveno slabša, gre pogosto za **pretirano prileganje** oziroma visoko varianco.

Premajhno prileganje lahko pogosto zmanjšamo z bolj zmogljivim modelom. Pri pretiranem prileganju pa lahko pomagajo preprostejši model, regularizacija ali več učnih podatkov.

## 5.4 Metrike za klasifikacijo

Pri klasifikaciji želimo primerjati napovedani razred s pravilnim razredom. Pri binarni klasifikaciji lahko rezultate razdelimo v štiri skupine.

### 5.4.1 Matrika zamenjav

**Matrika zamenjav** (angl. *confusion matrix*) prikazuje, koliko primerov je model razvrstil pravilno in katere vrste napak je naredil.

| | **Napoved pozitivna** | **Napoved negativna** |
|---|---|---|
| **Dejansko pozitivna** | TP | FN |
| **Dejansko negativna** | FP | TN |

Pri tem pomenijo:

- **TP** — pravilno pozitivni primeri (angl. *true positives*);
- **TN** — pravilno negativni primeri (angl. *true negatives*);
- **FP** — napačno pozitivni primeri (angl. *false positives*);
- **FN** — napačno negativni primeri (angl. *false negatives*).

### 5.4.2 Točnost, natančnost, priklic in mera F1

**Točnost** (angl. *accuracy*) je delež vseh pravilnih napovedi:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{91}$$

Točnost je intuitivna, vendar je lahko zelo zavajajoča pri neuravnoteženih razredih. Če je na primer samo 0,5 % primerov pozitivnih, model, ki vedno napove negativni razred, doseže 99,5 % točnost, čeprav ne prepozna niti enega pozitivnega primera.

**Natančnost** (angl. *precision*) odgovarja na vprašanje:

*Kolikšen delež vseh pozitivnih napovedi je bil res pozitiven?*

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{92}$$

Visoka natančnost je posebej pomembna, kadar so napačno pozitivne napovedi drage. Pri filtru neželene pošte na primer ne želimo, da bi legitimna sporočila označeval kot neželena.

**Priklic** (angl. *recall*) odgovarja na vprašanje:

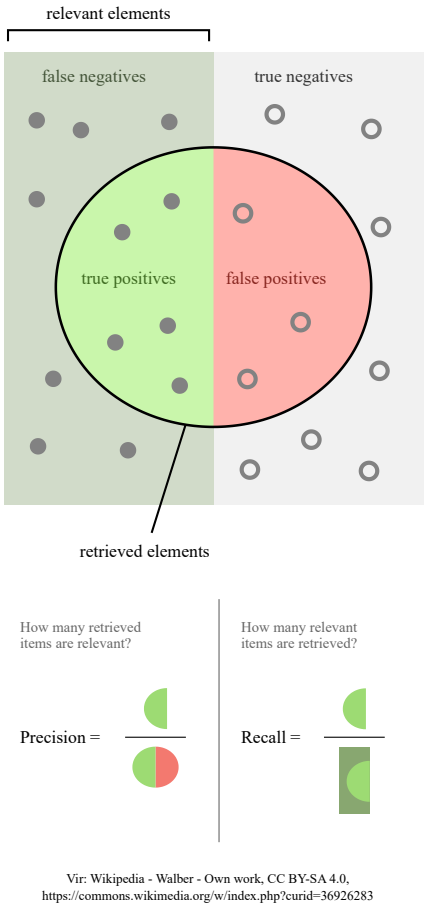
*Kolikšen delež vseh dejansko pozitivnih primerov smo odkrili?*

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{93}$$

Visok priklic je pomemben, kadar so napačno negativne napovedi posebej nevarne, na primer pri presejalnem testu za bolezen.

Za lažjo predstavo, sta obe enačbi vizualno ilustrirani na Sliki 4.

**Mera F1** (angl. *F1 score*) združi natančnost in priklic v eno vrednost:




Slika 4: Vizualizacija izračuna natančnosti in priklica.

$$F_1 = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{94}$$

**Primer: izračun klasifikacijskih metrik**

Pri modelu za zaznavanje bolezni imamo

$$TP = 80, \qquad FP = 20, \qquad FN = 10, \qquad TN = 890. \tag{95}$$

Točnost je

$$\text{Accuracy} = \frac{80 + 890}{80 + 890 + 20 + 10} = 0{,}97. \tag{96}$$

Natančnost je

$$\text{Precision} = \frac{80}{80 + 20} = 0{,}80. \tag{97}$$

Priklic je

$$\text{Recall} = \frac{80}{80+10} \approx 0{,}889. \tag{98}$$

Mera F1 je zato

$$F_1 = 2\frac{0{,}80 \cdot 0{,}889}{0{,}80 + 0{,}889} \approx 0{,}842. \tag{99}$$

### 5.4.3 Vpliv praga odločanja

Veliko klasifikatorjev ne vrne neposredno razreda, temveč verjetnost ali oceno. Končno napoved dobimo šele z izbiro praga odločanja.

Sprememba praga vpliva na kompromis med natančnostjo in priklicem.

Če prag **zvišamo**, model redkeje napove pozitivni razred. Običajno dobimo manj napačno pozitivnih primerov in višjo natančnost, vendar hkrati spregledamo več pozitivnih primerov in zmanjšamo priklic.

Če prag **znižamo**, model pogosteje napove pozitivni razred. Priklic se običajno poveča, natančnost pa lahko pade.

**Izbira praga je odvisna od problema**

Pri presejalnem testu za nevarno bolezen je lahko pomembneje doseči visok priklic, saj je spregledan primer bolezni zelo drag.
Pri filtru neželene pošte pa je lahko pomembnejša visoka natančnost, saj napačno odstranjeno legitimno sporočilo predstavlja večjo težavo kot nekaj neželene pošte v nabiralniku.
Univerzalno najboljšega praga zato ni.

### 5.4.4 Krivulja ROC in AUC

**Krivulja ROC** (angl. *Receiver Operating Characteristic curve*) prikazuje obnašanje klasifikatorja pri različnih pragih odločanja.

Na navpični osi je **stopnja pravilno pozitivnih primerov** (angl. *true positive rate*, TPR), ki je enaka priklicu:

$$TPR = \frac{TP}{TP + FN}. \tag{100}$$

Na vodoravni osi je **stopnja napačno pozitivnih primerov** (angl. *false positive rate*, FPR):

$$FPR = \frac{FP}{FP + TN}. \tag{101}$$

Vsak prag odločanja poda eno točko na krivulji ROC. Idealen klasifikator bi dosegel

$$TPR = 1, \qquad FPR = 0. \tag{102}$$

Za povzetek celotne krivulje pogosto uporabimo **ploščino pod krivuljo** (angl. *Area Under the Curve*, AUC).

Velja:

- $AUC = 1$ predstavlja popolno ločevanje razredov;
- $AUC = 0{,}5$ ustreza naključnemu razvrščanju;
- višji AUC pomeni boljšo sposobnost modela razlikovati med pozitivnimi in negativnimi primeri.

AUC lahko interpretiramo tudi kot verjetnost, da bo naključno izbranemu pozitivnemu primeru model dodelil višjo oceno kot naključno izbranemu negativnemu primeru.

**AUC ne nadomesti vseh drugih metrik**

ROC-AUC ocenjuje sposobnost razvrščanja čez različne pragove, ne pove pa neposredno, kako dobro model deluje pri določenem izbranem pragu.
Pri močno neuravnoteženih razredih so lahko natančnost, priklic in krivulja natančnost–priklic in **povprečna natančnost** (angl. *average precision*, AP), ki povzema uspešnost modela čez različne vrednosti priklica.

### 5.4.5 Kalibracija

Dober klasifikator lahko pravilno razvršča primere po tveganju, njegove napovedane verjetnosti pa kljub temu niso nujno pravilno interpretirane.

**Kalibracija** (angl. *calibration*) opisuje, ali napovedane verjetnosti ustrezajo dejanskim pogostostim.

Če je model dobro kalibriran, bi med primeri, ki jim pripiše približno 70 % verjetnost pozitivnega izida, približno 70 % primerov res moralo biti pozitivnih.

Model ima lahko dober AUC, vendar slabo kalibracijo: pozitivne primere pravilno razvrsti nad negativne, vendar so njegove številčne verjetnosti napačne.

Velja tudi obratno. Model lahko vsem primerom napove enako osnovno verjetnost in je v povprečju dobro kalibriran, vendar sploh ne zna razlikovati med bolj in manj tveganimi primeri.

**Kdaj je kalibracija pomembna?**

Kalibracija je posebej pomembna takrat, ko uporabljamo samo vrednost napovedane verjetnosti.
Če zdravnik pacientu pove, da je njegova ocenjena verjetnost bolezni 30 %, mora številka 30 % imeti dejanski pomen. Če pa primere zgolj razvrščamo od najbolj do najmanj tveganega, je pomembnejša sposobnost razvrščanja.

**Etični premislek: ko dobre metrike niso dovolj**

Predstavljajmo si bolnišnico, ki uporablja model za napovedovanje tveganja ponovnega sprejema pacientov. Model dosega

$$AUC = 0{,}85. \tag{103}$$

Na prvi pogled se rezultat zdi dober, vendar sama vrednost metrike še ne pove, ali je način uporabe modela primeren.
Razmislimo o naslednjih vprašanjih:

1. Bolnišnica želi pacientom z nizkim ocenjenim tveganjem omejiti dodatno spremljanje, da bi zmanjšala stroške. Kakšne posledice ima lahko napačna ocena tveganja za takšne paciente?
2. Bolnišnica zniža prag odločanja, da bi več pacientov označila kot visoko tvegane. S tem se poveča priklic, vendar zmanjša natančnost. Kdo ima od takšne spremembe korist in kdo lahko zaradi

nje utrpi škodo?

3. Zdravnik meni, da bi moral napoved modela preglasiti vedno, kadar se z njo ne strinja. Drug zdravnik trdi, da je dosledna uporaba istega modela pravičnejša od subjektivne človeške presoje. Katere so prednosti in slabosti obeh pristopov?

Dobra vrednost ene same metrike torej še ne zagotavlja, da je model primeren za uporabo. Pomembni so tudi posledice posameznih vrst napak, način uporabe napovedi ter skupine ljudi, na katere odločitve vplivajo.

### 5.4.6 Izbira primerne metrike

Različne metrike poudarjajo različne lastnosti modela. Zato pri evalvaciji ni dovolj vprašati, katera metrika je največja, ampak predvsem, **katere vrste napak so pri konkretnem problemu pomembne**.

| **Metrika** | **Kdaj jo uporabiti** | **Na kaj moramo paziti** |
|---|---|---|
| Točnost (angl. *accuracy*) | Ko so razredi približno uravnoteženi in so različne vrste napak podobno pomembne. | Pri neuravnoteženih razredih je lahko zelo zavajajoča. |
| Natančnost (angl. *precision*) | Ko so napačno pozitivne napovedi posebej drage ali nezaželene. | Ne upošteva neposredno napačno negativnih primerov. |
| Priklic (angl. *recall*) | Ko so napačno negativne napovedi posebej drage ali nevarne. | Ne upošteva neposredno napačno pozitivnih primerov. |
| Mera F1 (angl. *F1 score*) | Ko želimo eno samo metriko, ki uravnoteži natančnost in priklic. | natančnost in priklicu pripisuje enako pomembnost. |
| ROC-AUC (angl. *Area Under the ROC Curve*) | Ko želimo oceniti sposobnost modela razlikovati med razredi čez različne pragove odločanja. | Ne pove neposredno uspešnosti pri konkretnem izbranem pragu. |
| Stroškovno utežena metrika (angl. *cost-weighted metric*) | Ko imajo različne vrste napak različne posledice oziroma stroške. | Potrebujemo smiselno oceno relativnih stroškov različnih napak. |

**Ključna ideja**

Ne obstaja ena sama najboljša metrika za vse probleme.
Primerno metriko izberemo glede na to, katere napake so pri konkretni uporabi najpomembnejše in kakšne posledice imajo napačne odločitve.

## 5.5 Metrike za regresijo

Pri regresiji je izhod numerična vrednost, zato uspešnost modela ocenjujemo z velikostjo napak med pravo vrednostjo $y_i$ in napovedjo $\hat{y}_i$.

**Povprečna absolutna napaka** (angl. *mean absolute error*, MAE) je

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{104}$$

MAE ima enake enote kot ciljna spremenljivka in vse napake obravnava linearno.

**Povprečna kvadratna napaka** (angl. *mean squared error*, MSE) je

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{105}$$

Ker napake kvadriramo, MSE močneje kaznuje velike napake.

**Koren povprečne kvadratne napake** (angl. *root mean squared error*, RMSE) je

$$RMSE = \sqrt{MSE}. \tag{106}$$

S korenjenjem dobimo metriko nazaj v enakih enotah kot ciljna spremenljivka, kar običajno omogoča lažjo interpretacijo.

Pogosta metrika je tudi **koeficient determinacije** $R^2$ (angl. *R-squared*):

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}. \tag{107}$$

Vrednost $R^2 = 1$ pomeni popolne napovedi, $R^2 = 0$ pa, da model po tej meri ni boljši od vedno napovedane povprečne vrednosti. Na novih podatkih je lahko $R^2$ tudi negativen, kar pomeni, da je model po tej meri slabši od vedno napovedane povprečne vrednosti.

## 5.6 Prečno preverjanje

Pri majhnih podatkovnih množicah je lahko rezultat močno odvisen od ene same naključne delitve na učno in validacijsko množico. Nekatera delitev je lahko po naključju lažja ali težja od druge.

To težavo zmanjšuje **$k$-kratno prečno preverjanje** (angl. *$k$-fold cross-validation*).

Podatke razdelimo na $k$ približno enako velikih podmnožic (angl. *folds*). Nato izvedemo $k$ ponovitev:

1. eno podmnožico uporabimo za validacijo;
2. preostalih $k - 1$ podmnožico uporabimo za učenje;
3. postopek ponovimo tako, da vsaka podmnožico natanko enkrat služi kot validacijska.

Končno oceno dobimo kot povprečje rezultatov vseh $k$ ponovitev:

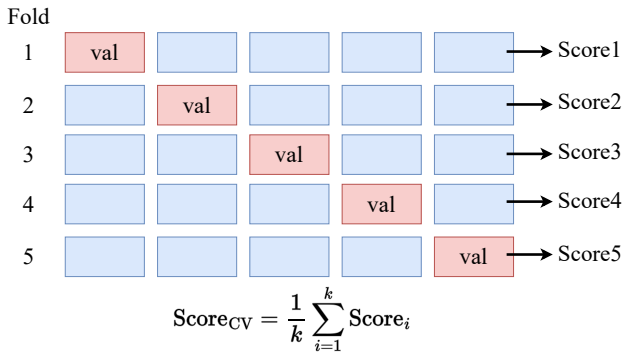
$$\text{Score}_{CV} = \frac{1}{k}\sum_{i=1}^{k}\text{Score}_i. \tag{108}$$

Pogosti izbiri sta $k = 5$ in $k = 10$.

5-kratno prečno preverjanje je ilustrirano na Sliki 5.

**Prednost in cena prečnega preverjanja**

Vsak primer je enkrat uporabljen za validacijo in večkrat za učenje, zato dobimo stabilnejšo oceno kot pri eni sami delitvi.
Cena je večja računska zahtevnost, saj moramo model naučiti $k$-krat.




Slika 5: Ilustracija k-kratnega prečnega preverjanja.

**Pazimo na strukturo podatkov**

Tudi pri prečnem preverjanju moramo spoštovati strukturo podatkov.
Če imamo na primer več meritev istega pacienta, različne meritve iste osebe ne smejo končati hkrati v učni in validacijski množici. V takem primeru uporabimo skupinsko prečno preverjanje (angl. *grouped cross-validation*), kjer vse primere iste skupine ohranimo skupaj.
Pri neuravnoteženi klasifikaciji lahko uporabimo stratificirano prečno preverjanje (angl. *stratified cross-validation*), pri katerem posamezne podmnožice približno ohranijo razmerje razredov.
Pri časovno odvisnih podatkih moramo ohraniti časovni vrstni red: učimo se na preteklosti in validiramo na prihodnosti.

## 5.7 Pogoste napake pri evalvaciji

Tudi če modela neposredno ne učimo na testnih podatkih, lahko testno množico nehote vključimo v razvoj modela.

### 5.7.1 Pretirano prilagajanje lestvici

Na tekmovanjih lahko del testnih podatkov določa javno lestvico (angl. *public leaderboard*). Če po vsaki oddaji pogledamo rezultat, spremenimo model in ponovno oddamo napovedi, rezultat lestvice postopoma vpliva na naše razvojne odločitve.

Po dovolj ponovitvah se lahko tako posredno prilagodimo tudi podatkom javne lestvice. Visok rezultat na njej zato ne zagotavlja dobrega rezultata na skritem delu testne množice.

### 5.7.2 "Samo še en pogled" na testno množico

Testna množica je uporabna prav zato, ker ni vplivala na razvoj modela.

Če preverimo testni rezultat, nato zaradi njega spremenimo značilke, model ali hiperparametre in test ponovno pogledamo, testna množica postane del procesa izbire modela.

Več modelov kot na ta način preizkusimo, večja je možnost, da bo kateri po naključju posebej dobro deloval ravno na tej testni množici.

**Ključno pravilo**

Validacijske podatke uporabljamo za razvojne odločitve.
Testno množico uporabimo šele na koncu za končno oceno.

Uhajanje podatkov pri pripravi značilk, časovno uhajanje ter veljavnost testne množice smo podrobneje obravnavali že pri pripravi podatkov. Vsi ti pojavi lahko povzročijo pretirano optimistično oceno uspešnosti modela.

## 5.8 Izbira hiperparametrov

Dobre vrednosti hiperparametrov lahko pomembno vplivajo na uspešnost modela.

Pri **mrežnem iskanju** (angl. *grid search*) vnaprej določimo možne vrednosti hiperparametrov in preizkusimo vse njihove kombinacije.

Če imamo pet hiperparametrov in za vsakega preizkusimo deset vrednosti, to pomeni

$$10^5 = 100\,000 \tag{109}$$

različnih kombinacij, zato lahko mrežno iskanje hitro postane računsko zelo drago.

Pri **naključnem iskanju** (angl. *random search*) namesto vseh kombinacij naključno vzorčimo možne nastavitve hiperparametrov. To je lahko učinkovitejše, saj vsi hiperparametri niso enako pomembni.

Pri nekaterih hiperparametrih je koristno preizkušati vrednosti na logaritemski skali, na primer

$$0{,}001, \qquad 0{,}01, \qquad 0{,}1, \tag{110}$$

namesto enakomerno razmaknjenih vrednosti.

Obstajajo tudi naprednejše metode, kjer se upoštevajo verjetnosti ali posebni algoritmi za preiskovanje prostora hiperparametrov.

**Tudi validacijski množici se lahko pretirano prilagodimo**

Če izvedemo zelo veliko poskusov in vedno izberemo nastavitev z najboljšim validacijskim rezultatom, se lahko postopoma pretirano prilagodimo tudi validacijski množici.
Testna množica mora zato ostati ločena od izbire hiperparametrov.

## 5.9 Povzetek

**Ključne ugotovitve**

- Namen evalvacije je oceniti, kako dobro model **posplošuje** na nove podatke, ne kako dobro si je zapomnil učno množico.
- **Parametre** se model nauči iz podatkov, **hiperparametre** pa izbiramo med razvojem modela.
- **Premajhno prileganje** je povezano z visoko pristranskostjo, **pretirano prileganje** pa z visoko varianco.
- Pri klasifikaciji matrika zamenjav razdeli napovedi na TP, TN, FP in FN. Iz nje izpeljemo točnost, natančnost, priklic in mero F1.

- Izbira **praga odločanja** vpliva na kompromis med natančnostjo in priklicem.
- **ROC-AUC** ocenjuje sposobnost modela razlikovati med razredi čez različne pragove, **kalibracija** pa pove, ali napovedane verjetnosti ustrezajo dejanskim pogostostim.
- Pri regresiji pogosto uporabljamo MAE, MSE, RMSE in $R^2$.
- **$k$-kratno prečno preverjanje** omogoča stabilnejšo oceno uspešnosti in učinkovitejšo uporabo omejene količine podatkov.
- Testni podatki ne smejo vplivati na razvojne odločitve. Tudi večkratno preverjanje rezultata in prilagajanje modela na njegovi podlagi pomeni posredno uporabo testne množice.
- Hiperparametre lahko iščemo z mrežnim ali naključnim iskanjem, njihovo kakovost pa ocenjujemo na validacijskih podatkih oziroma z navzkrižnim preverjanjem.

## 5.10 Naloge

Naslednjih devet nalog sledi vajam iz poglavja o evalvaciji modelov v gradivu IAIO.

1. **Delitev podatkov**

   Imamo 10 000 podatkovnih primerov.

   Opišite, kako bi podatke razdelili na učno, validacijsko in testno množico, če želimo izbirati hiperparametre. Kolikšen delež podatkov bi namenili posamezni množici?

2. **Pristranskost in varianca**

   Model doseže 99 % točnost na učni množici, vendar samo 65 % točnost na validacijski množici.

   Diagnosticirajte težavo z vidika pristranskosti in variance. Kaj se najverjetneje dogaja?

3. **Matrika zamenjav**

   Dana je naslednja matrika zamenjav:

   | | **Napoved +** | **Napoved −** |
   |---|---|---|
   | **Dejansko +** | 450 | 50 |
   | **Dejansko −** | 100 | 400 |

   Izračunajte:

   (a) natančnost (angl. *precision*);

   (b) priklic (angl. *recall*);

   (c) mero F1 (angl. *F1 score*).

4. **Natančnost ali priklic**

   V katerih primerih bi želeli optimizirati **natančnost** (angl. *precision*) namesto **priklica** (angl. *recall*)?

   Navedite primer iz resničnega sveta in pojasnite, katera vrsta napake je v njem posebej problematična.

5. **Zakaj deluje prečno preverjanje**

   Pri $k$-kratnem prečnem preverjanju (angl. *$k$-fold cross-validation*) naučimo $k$ modelov. Vsak model se uči na deležu $(k-1)/k$ podatkov, preostali del pa služi za validacijo. Končno oceno dobimo s povprečenjem validacijskih rezultatov.

   Odgovorite:

   (a) Intuitivno pojasnite, zakaj povprečenje rezultatov čez $k$ gub daje zanesljivejšo oceno kot ena sama delitev na učno in validacijsko množico.

(b) Ko se $k$ približuje številu primerov $n$, dobimo prečno preverjanje z izpuščanjem enega primera (angl. *leave-one-out cross-validation*). Kaj se pri tem zgodi z:

- pristranskostjo ocene napake;
- varianco ocene napake;
- računsko zahtevnostjo?

(c) Zakaj sta $k = 5$ in $k = 10$ pogosto smiselna kompromisa?

(d) Predpostavimo, da imajo podatki naravne skupine, na primer več meritev istega pacienta. Zakaj lahko običajno $k$-kratno prečno preverjanje poda preveč optimistično oceno? Kako bi morali podatke razdeliti?

6. **Razcep pristranskosti in variance**

Predpostavimo, da velja

$$Y = f(x) + \varepsilon, \tag{111}$$

kjer je $\mathbb{E}[\varepsilon] = 0$ in $\mathrm{Var}(\varepsilon) = \sigma^2$.

Pri regresiji lahko pričakovano kvadratno napako zapišemo kot

$$\mathbb{E}\left[(Y - \hat{f}(x))^2\right] = \mathrm{Bias}^2 + \mathrm{Variance} + \sigma^2. \tag{112}$$

Odgovorite:

(a) Definirajte pristranskost s pomočjo $\mathbb{E}[\hat{f}(x)]$ in $f(x)$.

(b) Zakaj ima konstantni model

$$\hat{f}(x) = c \tag{113}$$

običajno visoko pristranskost?

(c) Zakaj ima model, ki popolnoma interpolira vse učne primere, pogosto visoko varianco?

(d) Regularizacija običajno poveča pristranskost modela. Zakaj lahko kljub temu zmanjša napako na novih podatkih?

7. **Ko točnost zavaja**

Presejalni test za bolezen, ki je prisotna pri približno 1 % populacije, da naslednjo matriko zamenjav:

| | **Test +** | **Test −** |
|---|---|---|
| **Bolezen prisotna** | 90 | 10 |
| **Bolezen ni prisotna** | 495 | 9405 |

Odgovorite:

(a) Izračunajte točnost (angl. *accuracy*), natančnost (angl. *precision*), priklic (angl. *recall*) in mero F1.

(b) Zakaj je točnost 94,95 % v tem primeru zavajajoča?

(c) ojasnite, zakaj je pozitivna napovedna vrednost (angl. *positive predictive value*, PPV) enaka natančnosti. Zakaj je v tem primeru nizka kljub visokemu priklicu?

8. **Odkrivanje nepravilne evalvacije**

   Sodelavec predstavi model, ki doseže $AUC = 0{,}99$.

   Navedite tri načine, na katere bi lahko med razvojem modela nenamerno povzročil, da je rezultat pretirano optimističen.

9. **Kateri model bolje posplošuje?**

   Model A doseže 100 % točnost na učni množici in 60 % na testni množici.

   Model B doseže 70 % točnost na učni množici in 65 % na testni množici.

   Primerjajte modela in pojasnite, kaj njuni rezultati povedo o pristranskosti, varianci in posploševanju.

### 5.10.1 Dodatne naloge

10. **Prag odločanja**

    Binarni klasifikator za posamezne primere vrača verjetnost pozitivnega razreda.

    (a) Kaj se običajno zgodi z natančnostjo in priklicem, če prag odločanja zvišamo z 0,5 na 0,8?

    (b) Kaj se običajno zgodi, če prag znižamo z 0,5 na 0,2?

    (c) Pri presejalnem testu za nevarno bolezen bi želeli zmanjšati število napačno negativnih primerov. Bi prag praviloma zvišali ali znižali? Utemeljite.

11. **ROC in AUC**

    Model A ima $AUC = 0{,}92$, model B pa $AUC = 0{,}71$.

    (a) Kaj vrednost AUC meri?

    (b) Kako lahko interpretiramo $AUC = 0{,}92$ z vidika naključno izbranega pozitivnega in negativnega primera?

    (c) Ali iz višjega AUC neposredno sledi, da bo model pri izbranem pragu imel višjo natančnost in višji priklic? Pojasnite.

12. **Kalibracija**

    Model A med vsemi primeri, ki jim pripiše približno 80 % verjetnost pozitivnega izida, dejansko opazi pozitivni izid le v 45 % primerov.

    (a) Ali je model dobro kalibriran? Zakaj?

    (b) Ali lahko ima tak model kljub temu visok AUC?

    (c) Navedite primer uporabe, kjer bi bila slaba kalibracija posebej problematična.

13. **Regresijske metrike**

    Za štiri primere so prave vrednosti

    $$y = [3, 5, 2, 10] \tag{114}$$

    in napovedi

    $$\hat{y} = [2, 5, 4, 7]. \tag{115}$$

    Izračunajte:

(a) povprečno absolutno napako MAE;

(b) povprečno kvadratno napako MSE;

(c) koren povprečne kvadratne napake RMSE.

Katera od teh metrik močneje poudari velike posamezne napake?

14. **Testna množica ni razvojna množica**

Ekipa razvije deset različnih modelov. Vsakega oceni na isti testni množici in nato izbere model z najboljšim testnim rezultatom.

(a) Kaj je narobe s tem postopkom?

(b) Kakšno vlogo je testna množica dejansko prevzela?

(c) Kako bi postopek spremenili, da bi na koncu še vedno imeli nepristransko končno oceno?

15. **Izbira primerne metrike**

Za vsak primer izberite metriko oziroma metrike, ki bi bile posebej smiselne, in odgovor utemeljite.

(a) Odkrivanje zelo redke nevarne bolezni, pri kateri je spregledan primer posebej nevaren.

(b) Filter neželene pošte, pri katerem želimo predvsem preprečiti, da bi legitimno pošto označili kot neželeno.

(c) Uravnotežena klasifikacijska naloga, pri kateri so vse vrste napak približno enako pomembne.

(d) Sistem, pri katerem napovedana verjetnost neposredno določa višino finančnega tveganja.

(e) Primerjava sposobnosti več klasifikatorjev za razvrščanje pozitivnih primerov nad negativne, ne da bi vnaprej določili prag.

**Predlogi za nadaljnje branje:**


1. G. James, D. Witten, T. Hastie, & R. Tibshirani, *An Introduction to Statistical Learning*, Springer.
2. Géron, Aurélien, *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*, O'Reilly.


# 6 Spodbujevano učenje

**Spodbujevano učenje** oziroma **okrepitveno učenje** (angl. *reinforcement learning*, RL) je veja strojnega učenja, pri kateri se **agent** uči z neposredno interakcijo z okoljem. Namesto pravilnih odgovorov za posamezne primere agent izbira akcije, opazuje njihove posledice in prejema numerične nagrade.

S poskušanjem različnih akcij skozi čas odkriva, katera vedenja vodijo do večje skupne nagrade. Cilj spodbujevanega učenja zato ni napovedati pravilnega odgovora za posamezen primer, temveč se naučiti dobre **strategije** (angl. *policy*) za zaporedno sprejemanje odločitev.

Podobno se pogosto učimo tudi ljudje. Ko se na primer učimo igrati novo videoigro, sprva ne poznamo najboljše poteze v vsaki situaciji. Poskušamo različne akcije, opazujemo njihove posledice in sčasoma ugotovimo, katere odločitve pogosteje vodijo do uspeha. Posamezna odločitev pri tem ne vpliva le na takojšnji rezultat, temveč tudi na situacije, v katerih se bomo znašli kasneje.

Klasičen primer spodbujevanega učenja je okolje **Lunar Lander**, v katerem mora agent upravljati pristajalni modul. Iz trenutnega položaja, hitrosti in nagiba mora izbirati med prižiganjem različnih motorjev ter se naučiti varno pristati na označenem območju. Uspešen pristanek prinese visoko nagrado, trk pa kazen. Agent pri tem vnaprej ne dobi navodil, katero akcijo naj izbere v posameznem trenutku, temveč dobro strategijo odkrije skozi veliko število poskusov.

Spodbujevano učenje se uporablja tudi pri nekaterih sodobnih velikih jezikovnih modelih (angl. *large language models*, LLM). Po začetnem učenju lahko model ustvarja različne odgovore, ki se nato ocenijo glede na njihovo kakovost. Pri **spodbujevanem učenju iz človeških povratnih informacij** (angl. *reinforcement learning from human feedback*, RLHF) signal nagrade izhaja iz človeških ocen oziroma modela, naučenega iz takšnih ocen. Model se nato prilagaja tako, da pogosteje ustvarja odgovore z višjo pričakovano nagrado.

**Ključna ideja**

Agent opazuje stanje okolja, izbere akcijo ter prejme nagrado in novo stanje. Njegov cilj je naučiti se strategije, ki maksimizira skupno pričakovano nagrado.

Za razliko od nadzorovanega učenja agent nima podatkovne množice s pravilnimi akcijami. Pravilnega odgovora mu nihče ne pove neposredno. Namesto tega mora skozi poskuse in napake ugotoviti, katere akcije dolgoročno vodijo do dobrih rezultatov.

Pomembna posebnost spodbujevanega učenja je tudi, da so odločitve **zaporedne**: akcija, ki jo agent izvede sedaj, vpliva na stanje okolja in s tem na odločitve ter nagrade, ki bodo mogoče v prihodnosti.

## 6.1 Ogrodje spodbujevanega učenja

Spodbujevano učenje opisuje interakcijo med **agentom** in **okoljem** (angl. *environment*). V vsakem časovnem koraku agent opazuje trenutno stanje, izbere akcijo, okolje pa mu vrne nagrado in novo stanje.

Osnovni cikel lahko zapišemo kot

$$s_t \longrightarrow a_t \longrightarrow R_{t+1},\, s_{t+1}. \tag{116}$$

Pri tem so:

- **agent** (angl. *agent*) — del sistema, ki se uči in sprejema odločitve;
- **okolje** (angl. *environment*) — vse, s čimer agent interagira;
- **stanje** (angl. *state*) $s_t$ — opis trenutne situacije;
- **akcija** (angl. *action*) $a_t$ — odločitev, ki jo agent izvede;
- **nagrada** (angl. *reward*) $R_{t+1}$ — numerična povratna informacija o posledici akcije (nagrada oziroma kazen);
- **strategija** (angl. *policy*) — pravilo, po katerem agent izbira akcije.

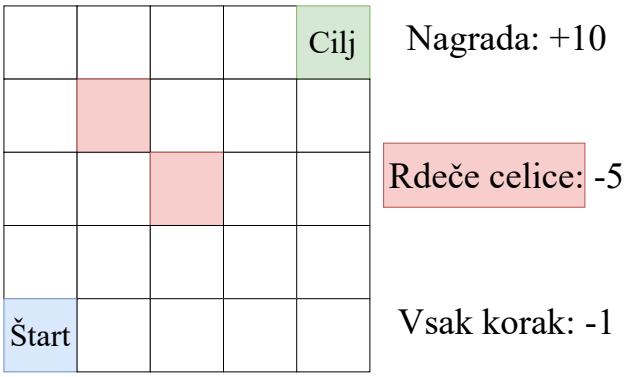
Agent je pri tem "možgani" sistema, ne nujno celoten fizični sistem. Pri učenju robota je lahko agent na primer algoritem za krmiljenje, medtem ko telo robota skupaj z zunanjim svetom predstavlja del okolja.

Stanje mora vsebovati informacije, ki jih agent potrebuje za sprejemanje dobrih odločitev. V šahu je lahko stanje trenutna postavitev figur, pri samovozečem avtomobilu pa na primer položaj in hitrost vozila ter položaji bližnjih ovir.

Nagrada je eno samo število, ki agentu pove, kako zaželena je bila posledica njegove akcije. Višja nagrada pomeni boljši rezultat.

### 6.1.1 Primer: premikanje po labirintu

Predstavljajmo si agenta, ki se premika po mreži $5 \times 5$, prikazano na Sliki 6. Njegov cilj je priti od začetnega polja do cilja.




Slika 6: Labirint kjer je naloga agenta da se od spodnjega levega kota premakne do zgornjega desnega, pri tem pa se mora izogibati rdečim poljem.

Vsako polje predstavlja stanje, agent pa lahko izbira med štirimi akcijami: premikom gor, navzdol, levo ali desno (razen na robovih).

Nagrade določimo na primer tako:

- $+10$ za prihod na cilj,
- $-5$ za prihod na nevarno (rdeče) polje,
- $-1$ za vsak izvedeni korak.

Kazen $-1$ za vsak korak spodbuja agenta, da ne poišče zgolj katerekoli poti do cilja, temveč čim krajšo primerno pot.

**Nagrada določa nalogo**

Agent optimizira natanko tisto nagrado, ki smo jo definirali, ne nujno vedenja, ki smo ga imeli v mislih. Če je funkcija nagrade slabo določena, lahko agent odkrije nezaželen način pridobivanja visoke nagrade. Temu pravimo **zloraba nagrade** (angl. *reward hacking*).
Načrtovanje ustrezne funkcije nagrade je zato pomemben del problemov spodbujevanega učenja.

## 6.2 Skupna nagrada in faktor znižanja

Agent običajno ne želi maksimizirati samo naslednje nagrade. Akcija z majhno trenutno nagrado lahko namreč v prihodnosti vodi do bistveno boljšega rezultata.

Zato uporabljamo **skupno nagrado** (angl. *return*), ki seštevek vseh prihodnjih nagrad.

Brez znižanja bi jo zapisali kot

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots . \tag{117}$$

V praksi prihodnje nagrade pogosto znižamo z **faktorjem znižanja** (angl. *discount factor*) $\gamma$, kjer običajno velja

$$0 \leq \gamma \leq 1. \tag{118}$$

Pri epizodnih problemih lahko uporabimo tudi $\gamma = 1$. Pri problemih, ki se lahko nadaljujejo neskončno dolgo, pa mora biti skupna nagrada ustrezno definirana, sicer lahko neskončna vsota divergira.

Znižana skupna nagrada je

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{119}$$

Nagrada, ki jo prejmemo $k$ korakov v prihodnosti, je tako utežena s faktorjem $\gamma^k$.

Če je $\gamma = 0$, agent upošteva samo neposredno naslednjo nagrado. Ko se $\gamma$ približuje 1, postajajo prihodnje nagrade vedno pomembnejše.

**Intuicija**

Faktor znižanja določa, kako "daljnoviden" je agent.
Majhen $\gamma$ daje večji pomen takojšnjim nagradam, večji $\gamma$ pa dolgoročnim posledicam odločitev.

## 6.3 Strategije in funkcije vrednosti

**Strategija** (angl. *policy*) določa, katero akcijo naj agent izbere v posameznem stanju.

Pri deterministični strategiji velja

$$\pi(s) = a, \tag{120}$$

kar pomeni, da agent v stanju $s$ vedno izbere akcijo $a$.

Pri stohastični strategiji (angl. *stochastic policy*) pa

$$\pi(a \mid s) \tag{121}$$

predstavlja verjetnost, da agent v stanju $s$ izbere akcijo $a$.

Za primerjavo različnih stanj in akcij uporabljamo **funkcije vrednosti** (angl. *value functions*).

**Funkcija vrednosti stanja** $V^\pi(s)$ pove, kolikšen pričakovano skupno nagrado lahko dosežemo, če začnemo v stanju $s$ in nato sledimo strategiji $\pi$:

$$V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right]. \tag{122}$$

**Funkcija vrednost akcije** $Q^\pi(s, a)$ pa odgovarja na nekoliko drugačno vprašanje: kako dobra je akcija $a$, če jo izvedemo v stanju $s$ in nato sledimo strategiji $\pi$?

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right]. \tag{123}$$

**Intuicija funkcij vrednosti**

$V(s)$ odgovarja na vprašanje:

*Kako dobro je biti v stanju s?*

$Q(s, a)$ pa na vprašanje:

*Kako dobro je izvesti akcijo a v stanju s?*

Če poznamo optimalne vrednosti $Q$, lahko v vsakem stanju preprosto izberemo akcijo z največjo vrednostjo.

## 6.4 Markovska lastnost in Markov proces odločanja

Probleme spodbujevanega učenja pogosto opišemo z **Markovim procesom odločanja** (angl. *Markov Decision Process*, MDP).

Osnovna predpostavka je **Markovska lastnost** (angl. *Markov property*): trenutno stanje vsebuje vse informacije, ki jih potrebujemo za napoved prihodnjega razvoja sistema. Ko poznamo trenutno stanje, poznavanje celotne pretekle zgodovine ne poda dodatnih informacij:

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, \ldots). \tag{124}$$

Dober primer je šah. Če poznamo trenutno postavitev na igralni plošči, za določitev naslednjih dovoljenih potez ni pomembno, po kakšnem zaporedju potez smo prišli do nje. Pomembna je sicer informacija o pravici do rošade in možnosti en passant.

Markov proces odločanja sestavljajo:

- množica stanj $S$,
- množica akcij $A$,
- prehodne verjetnosti $P(s' \mid s, a)$,
- funkcija nagrade $R(s, a)$,
- faktor znižanja $\gamma$.

**Ko stanje ne vsebuje vseh informacij**

V resničnih problemih stanje pogosto ni popolnoma opazljivo. Če agent nima dostopa do vseh informacij, ki vplivajo na prihodnost, Markovska predpostavka ne velja več neposredno. Takšne probleme lahko opišemo z delno opazljivimi Markovimi procesi odločanja (angl. *Partially Observable Markov Decision Process*, POMDP).

## 6.5 Bellmanove enačbe

Vrednost stanja lahko izrazimo rekurzivno: sestavljena je iz neposredne nagrade ter znižane vrednosti naslednjega stanja. To idejo formalizirajo **Bellmanove enačbe** (angl. *Bellman equations*).

Za strategijo $\pi$ velja Bellmanova enačba pričakovane vrednosti:

$$V^\pi(s) = \sum_a \pi(a \mid s) \left[ R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s') \right]. \tag{125}$$

Zunanji seštevek upošteva možne akcije in verjetnosti, s katerimi jih izbere strategija. Notranji seštevek pa upošteva možna naslednja stanja.

Če želimo poiskati najboljšo možno strategijo, namesto povprečenja po akcijah izberemo najboljšo:

$$V^*(s) = \max_a \left[ R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right]. \tag{126}$$

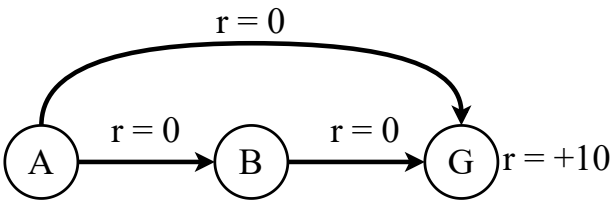
Funkcija $V^*(s)$ predstavlja največja pričakovana skupna nagrada, ki ga lahko dosežemo iz stanja $s$.

**Ključna ideja Bellmanove enačbe**

Vrednost trenutnega stanja lahko razdelimo na dva dela:
**takojšnja nagrada + znižana vrednost prihodnosti.**
Ta rekurzivna povezava je osnova številnih algoritmov spodbujevanega učenja.

### 6.5.1 Primer: iteracija vrednosti na roko

Oglejmo si preprost Markov proces odločanja s tremi stanji $A$, $B$ in $G$. Stanje $G$ je končno stanje (angl. *terminal state*) z nagrado +10, faktor znižanja pa je $\gamma = 0{,}9$. Prikaz sistema ja na Sliki 7.




Slika 7: Ilustracija okolja.

Iz stanja $A$ lahko agent izbere dve poti: $A \longrightarrow G$ ali $A \longrightarrow B \longrightarrow G$.

Vsi vmesni prehodi imajo nagrado 0. Zanima nas, katera pot je boljša.

Pri **iteraciji vrednosti** (angl. *value iteration*) začnemo z začetnimi ocenami

$$V(A) = 0, \qquad V(B) = 0, \qquad V(G) = 10. \tag{127}$$

Za stanje $B$ je možna samo pot do cilja, zato dobimo

$$V(B) = 0 + \gamma V(G) = 0 + 0{,}9 \cdot 10 = 9. \tag{128}$$

V stanju $A$ lahko izbiramo med potjo prek $B$ in neposredno potjo do $G$:

$$V(A) = \max\left\{0 + 0{,}9V(B),\ 0 + 0{,}9V(G)\right\}. \tag{129}$$

Po prvi iteraciji, kjer je $V(B) = 0$, dobimo da je $V(A) = max\{0, 9\} = 9$.

V tem primeru smo uporabili še stare vrednosti, zato ponovimo postopek. Ko uporabimo posodobljene vrednosti, dobimo

$$V(A) = \max\{0 + 0{,}9V(B),\ 0 + 0{,}9V(G)\} = \max\{0{,}9 \cdot 9,\ 0{,}9 \cdot 10\} = \max\{8{,}1, 9\} = 9. \tag{130}$$

Končne vrednosti so torej

$$V(A) = 9, \qquad V(B) = 9, \qquad V(G) = 10. \tag{131}$$

Čeprav obe poti na koncu pripeljeta do G je neposredna pot $A \to G$ boljša. Zaradi zmanjšanja je namreč nagrada, ki jo prejmemo prej, vredna več kot enaka nagrada, ki jo prejmemo korak kasneje.

## 6.6 Raziskovanje in izkoriščanje

Eden osrednjih problemov spodbujevanega učenja je kompromis med **raziskovanjem** (angl. *exploration*) in **izkoriščanjem** (angl. *exploitation*).

Pri izkoriščanju agent uporabi trenutno znanje in izbere akcijo, za katero trenutno meni, da je najboljša. Pri raziskovanju pa poskusi tudi manj poznane akcije, da bi morda odkril še boljše možnosti.

Če agent vedno izkorišča trenutno znanje, lahko obstane pri prvi dovolj dobri strategiji in nikoli ne odkrije boljše. Če ves čas samo raziskuje, pa ne izkoristi znanja, ki ga je že pridobil.

Pogosta rešitev je **$\varepsilon$-požrešna strategija** (angl. *$\varepsilon$-greedy strategy*):

- z verjetnostjo $1 - \varepsilon$ agent izbere trenutno najboljšo akcijo;
- z verjetnostjo $\varepsilon$ izbere naključno akcijo.

Če naključno akcijo izbiramo enakomerno med vsemi akcijami, lahko pri raziskovanju po naključju ponovno izberemo tudi trenutno najboljšo akcijo.

Če je na primer $\varepsilon = 0{,}1$, agent približno 90 % časa izkorišča trenutno znanje, 10 % časa pa raziskuje.

**Analogija z restavracijami**

Če v neznanem mestu vedno obiščemo prvo dovolj dobro restavracijo, morda nikoli ne odkrijemo boljše.
Če vsak večer izberemo novo restavracijo, pa pogosto tvegamo slab obrok.
Smiselna strategija zato združuje raziskovanje novih možnosti in izkoriščanje tistih, za katere že vemo, da so dobre.

## 6.7 Metode dinamičnega programiranja

Če poznamo model okolja, torej prehodne verjetnosti $P(s' \mid s, a)$ in nagrade $R(s, a)$, lahko optimalno strategijo izračunamo z metodami **dinamičnega programiranja** (angl. *dynamic programming*).

Takšne metode imenujemo **metode z modelom** (angl. *model-based methods*), saj predpostavljajo, da poznamo delovanje okolja.

Pri **iteraciji strategije** (angl. *policy iteration*) izmenjujemo dva koraka:

1. ocenimo vrednosti stanj za trenutno strategijo;
2. na podlagi teh vrednosti strategijo izboljšamo.

Postopek ponavljamo, dokler se strategija spreminja.

Pri **iteraciji vrednosti** (angl. *value iteration*) neposredno večkrat uporabimo Bellmanovo enačbo optimalnosti:

$$V_{k+1}(s) = \max_a \left[ R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V_k(s') \right]. \tag{132}$$

Takšni postopki lahko poiščejo optimalno rešitev, vendar zahtevajo poznavanje modela okolja (verjetnosti in nagrade), ki ga v praksi pogosto nimamo.

## 6.8 Učenje neposredno iz izkušenj

Če prehodov in nagrad okolja ne poznamo vnaprej, se mora agent učiti neposredno iz opazovanih izkušenj. Takšnim pristopom pravimo **metode brez modela** (angl. *model-free methods*).

### 6.8.1 Učenje s časovno razliko

Pri **učenju s časovno razliko** (angl. *temporal-difference learning*, TD) vrednost stanja posodobimo po vsakem opazovanem prehodu:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[ R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]. \tag{133}$$

Parameter $\alpha$ predstavlja **stopnjo učenja** (angl. *learning rate*).

Izraz

$$R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{134}$$

imenujemo **napaka časovne razlike** (angl. *temporal-difference error*, TD error). Pove nam razliko med trenutno oceno vrednosti in novo oceno, ki jo dobimo po opazovanju naslednjega stanja in nagrade.

### 6.8.2 SARSA in Q-učenje

TD-učenje lahko razširimo na vrednosti akcij.

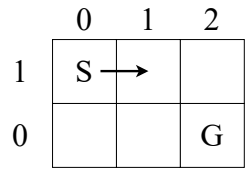
Algoritem **SARSA** uporablja vrednost akcije, ki jo agent dejansko izbere v naslednjem stanju:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]. \tag{135}$$

**Q-učenje** (angl. *Q-learning*) pa namesto dejansko izbrane naslednje akcije uporabi najboljšo trenutno znano vrednost:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right]. \tag{136}$$

Glavna razlika je, da se SARSA uči vrednosti strategije (učenje s strategijo, angl. *on-policy*), ki ji agent dejansko sledi, medtem ko se Q-učenje uči vrednosti optimalne strategije (učenje mimo strategije angl. *off-policy*), tudi če agent med učenjem zaradi raziskovanja izbira drugačne akcije.




Slika 8: Ilustracija okolja.

### 6.8.3 Primer Q-učenja

Predpostavimo, da so vse začetne vrednosti $Q$ enake 0, agent pa ob prihodu na cilj prejme nagrado +10. Naj bo $\gamma = 0{,}9, \alpha = 0{,}5$.. Vse začetne Q-vrednosti pa nastavimo na 0. Okolje je predstavljeno na Sliki 8

V prvi epizodi agent zaradi raziskovanja izbere pot (stanje so predstavljena kot (X, Y):

$$S \longrightarrow (1,1) \longrightarrow (2,1) \longrightarrow (2,0) = G. \tag{137}$$

Ko iz stanja $(2,1)$ z akcijo navzdol doseže cilj, uporabimo pravilo Q-učenja:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right]. \tag{138}$$

Ker so vse Q-vrednosti na začetku enake 0, dobimo

$$Q((2,1), \text{navzdol}) \leftarrow 0 + 0{,}5\,[10 + 0{,}9 \cdot 0 - 0] = 5. \tag{139}$$

Agent se je tako naučil, da je akcija navzdol v stanju $(2,1)$ dobra.

V naslednji epizodi agent ponovno pride do stanja $(2,1)$ iz stanja $(1,1)$. Za akcijo desno dobimo

$$Q((1,1), \text{desno}) \leftarrow 0 + 0{,}5 \left[ 0 + 0{,}9 \max_{a} Q((2,1), a) - 0 \right]. \tag{140}$$

Ker je trenutno najboljša Q-vrednost v stanju $(2,1)$ enaka 5,

$$Q((1,1), \text{desno}) = 0{,}5\,[0 + 0{,}9 \cdot 5] = 2{,}25. \tag{141}$$

Informacija o nagradi na cilju se je tako začela širiti nazaj tudi na prejšnja stanja. Po dovolj velikem številu epizod Q-vrednosti po celotni mreži postopoma odražajo, katere akcije vodijo proti cilju in kako oddaljena je nagrada.

**Omejitev Q-tabele**

Pri tabelaričnem Q-učenju shranjujemo vrednost za vsak par $(s, a)$. To je izvedljivo pri majhnem številu stanj, pri velikih problemih pa število možnih stanj hitro postane ogromno.
Namesto tabele lahko vrednosti približujemo s funkcijo, na primer z nevronsko mrežo $Q(s, a; \theta)$. To je ena od osnovnih idej **globokega spodbujevanega učenja** (angl. *deep reinforcement learning*).

## 6.9 Strategije raziskovanja

Pri $\varepsilon$-požrešni strategiji vrednost $\varepsilon$ pogosto med učenjem zmanjšujemo. Na začetku agent veliko raziskuje, kasneje pa vedno pogosteje uporablja naučeno znanje.

Na primer lahko začnemo z

$$\varepsilon = 1 \tag{142}$$

in ga postopoma zmanjšamo proti majhni vrednosti, kot je 0,01.

Druga možnost je **strategija softmax** (angl. *softmax exploration*), pri kateri boljše akcije izbiramo z večjo verjetnostjo, vendar imajo tudi slabše akcije še vedno možnost izbire:

$$P(a \mid s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}}. \tag{143}$$

Parameter $\tau$, imenovan **temperatura** (angl. *temperature*), določa, kako močno izbira daje prednost akcijam z višjimi vrednostmi $Q$.

## 6.10 Povzetek

**Ključne ugotovitve**

- Pri **spodbujevanem učenju** agent z interakcijo z okoljem odkriva strategijo, ki maksimizira skupno pričakovano nagrado.
- Osnovni elementi so **stanja, akcije, nagrade, okolje in strategija**.
- **Skupna nagrada** združuje prihodnje nagrade, faktor znižanja $\gamma$ pa določa razmerje med takojšnjimi in prihodnjimi nagradami.
- $V(s)$ opisuje vrednost stanja, $Q(s, a)$ pa vrednost izvedbe določene akcije v danem stanju.
- **Markovska lastnost** predpostavlja, da trenutno stanje vsebuje vse informacije, potrebne za napoved prihodnosti. RL probleme pogosto formaliziramo kot Markove procese odločanja.
- **Bellmanove enačbe** vrednost stanja povežejo s takojšnjo nagrado in znižano vrednostjo prihodnjih stanj.
- Agent mora uravnotežiti **raziskovanje** novih akcij in **izkoriščanje** trenutno najboljšega znanja.
- Če poznamo model okolja, lahko uporabimo metode dinamičnega programiranja. Če ga ne poznamo, se lahko učimo neposredno iz izkušenj.
- **Q-učenje** posodablja vrednosti parov stanj in akcij na podlagi prejete nagrade ter najboljše ocenjene vrednosti naslednjega stanja.

## 6.11 Naloge

1. **Bellmanove enačbe**

   Obravnavajte Markov proces odločanja z dvema stanjema $A$ in $B$ ter faktorjem znižanja $\gamma = 0{,}9$.

   Nagrade za posamezne prehode so:

   - $A \to A$: $+1$,
   - $A \to B$: $0$,
   - $B \to A$: $+2$,
   - $B \to B$: $+1$.

V vsakem stanju agent izbira med dvema akcijama, ki deterministično določita, ali bo naslednje stanje $A$ ali $B$.

Zapišite Bellmanovi enačbi za optimalni funkciji vrednosti $V^*(A)$ in $V^*(B)$, pri čemer predpostavite, da agent vedno izbere optimalno akcijo.

2. **Posodobitev pri Q-učenju**

   Pri Q-učenju naj bodo $\alpha = 0{,}1$, $\gamma = 0{,}9$ in $Q(s, a) = 5$.

   Agent izvede akcijo $a$, prejme nagrado $r = 2$ in preide v stanje $s'$, za katerega velja

$$\max_{a'} Q(s', a') = 8. \tag{144}$$

   Izračunajte novo vrednost $Q(s, a)$.

3. **Raziskovanje in izkoriščanje**

   Pojasnite kompromis med **raziskovanjem** (angl. *exploration*) in **izkoriščanjem** (angl. *exploitation*).

   Zakaj agent ne more vedno preprosto izbrati trenutno najboljše znane akcije?

4. **Vpliv faktorja znižanja**

   Kaj se zgodi z vedenjem optimalne strategije, ko se $\gamma$ približuje 0?

   Kaj pa, ko se $\gamma$ približuje 1?

   Pojasnite, kako faktor znižanja vpliva na pomen takojšnjih in prihodnjih nagrad ter s tem na skupno nagrado.

5. **Izpeljava Bellmanove enačbe**

   Začnite z enačbo

$$Q^\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]. \tag{145}$$

   Iz nje izpeljite **Bellmanovo enačbo optimalnosti** (angl. *Bellman optimality equation*) za $Q^*$.

6. **IAIO-slog: ko požrešna intuicija odpove**

   Obravnavajte Markov proces odločanja, pri katerem ima agent v začetnem stanju $s_0$ tri možne akcije:

   - **Akcija A:** takojšnja nagrada +10, nato se epizoda konča;
   - **Akcija B:** takojšnja nagrada 0, nato agent preide v stanje $X$;
   - **Akcija C:** takojšnja nagrada 0, nato agent preide v stanje $Y$.

   Iz stanja $X$ je na voljo samo ena akcija, ki prinese nagrado +5 in konča epizodo.

   Iz stanja $Y$ je na voljo samo ena akcija, ki prinese nagrado +20 in konča epizodo.

   Naj bo faktor znižanja $\gamma = 0{,}9$.

   Odgovorite:

   (a) Izračunajte $Q^*(s_0, A)$, $Q^*(s_0, B)$ in $Q^*(s_0, C)$.

   (b) Požrešen agent (angl. *greedy agent*) je do sedaj preizkusil samo akcijo A in zanjo prejel nagrado +10. Zakaj bo tak agent brez raziskovanja vedno znova izbiral akcijo A?

(c) Agent uporablja $\varepsilon$-požrešno strategijo (angl. *$\varepsilon$-greedy strategy*) z $\varepsilon = 0{,}1$.

Po 1000 epizodah približno kolikokrat bo preizkusil posamezno akcijo? Ali bo odkril optimalno strategijo?

(d) Predlagajte preprosto spremembo, zaradi katere bi požrešno vedenje hitreje odkrilo optimalno strategijo.

7. **Robot na mreži**

Robot se premika po mreži velikosti $4 \times 4$. Začne v polju $(1, 1)$. Cilj $(4, 4)$ in nevarno polje $(2, 2)$ sta končni stanji.

V vsakem nekončnem stanju lahko robot izbere eno izmed štirih akcij: gor, navzdol, levo ali desno. Če bi z akcijo zapustil mrežo, ostane v trenutnem stanju.

Nagrade so:

- $+10$ za prihod na cilj;
- $-10$ za prihod v nevarno polje;
- $-1$ za vsak drug izvedeni korak.

Odgovorite:

(a) Koliko stanj vsebuje okolje? Če so v vsakem nekončnem stanju na voljo štiri akcije, koliko različnih deterministčnih strategij je možnih?

(b) Primerjajte dve poti pri faktorju znižanja $\gamma = 0{,}9$: varno pot, ki potrebuje 8 korakov, in krajšo pot, ki potrebuje 6 korakov, vendar ima pri enem prehodu 20 % verjetnost, da robot konča v nevarnem polju. Katero pot bi agent raje izbral? Utemeljite z izračunom pričakovane skupne nagrade.

(c) Zapišite posodobitev pri Q-učenju za stanje $(1, 1)$ in akcijo desno, če agent pristane v stanju $(2, 1)$ in prejme nagrado $r = -1$.

**Predlogi za nadaljnje branje:**


1. Richard S. Sutton & Andrew G. Barto, *Reinforcement Learning: An Introduction*, Bradford Books.
2. Miguel Morales, *Grokking Deep Reinforcement Learning*, Manning.


# 7 Matematični zapis

množice in intervali; znaka za vsoto in produkt; funkcije in njihov zapis; primerjave in logični simboli; posebni zapisi v strojnem učenju

# 8 Verjetnost in statistika

kaj je verjetnost; osnovna pravila verjetnosti; pogojna verjetnost; Bayesov izrek; neodvisnost; slučajne spremenljivke; pričakovana vrednost; povprečje in mediana; varianca in standardni odklon; pogoste porazdelitve; vzorčenje

# 9 Odvodi

naklon kot osnovna ideja; odvod kot naklon v točki; pravila odvajanja; pomembne funkcije (linearna, polinomi, eksponentna, logaritemska, sigmoidna) in njihovi odvodi; parcialni odvodi; gradient; gradientni spust; minimumi in maksimumi

# 10 Linearna algebra

skalarji; vektorji; skalarni produkt; dolžina vektorja in razdalja; matrike; osnovne operacije z matrikami