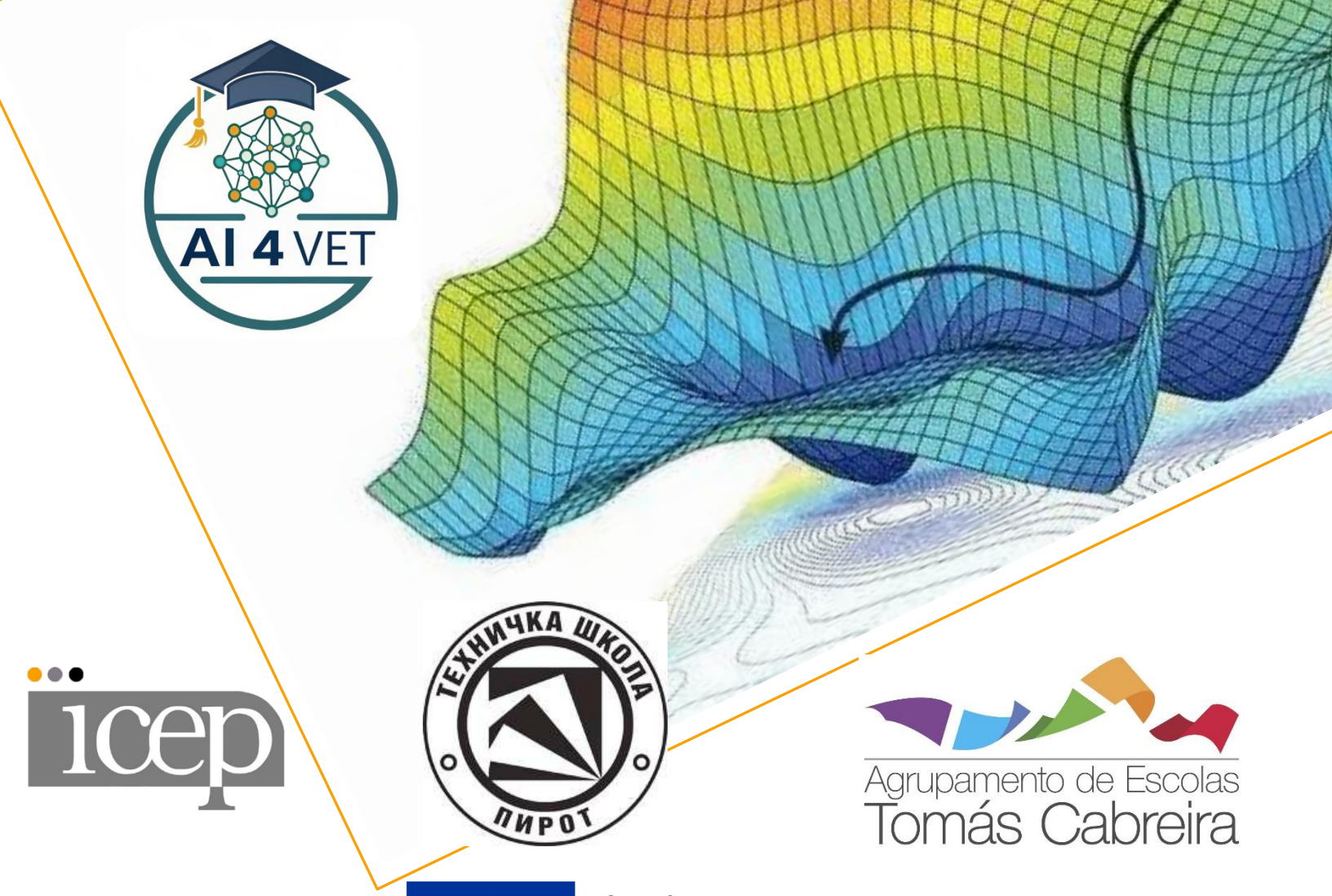


Co-funded by the European Union

OSNOVE VEŠTAČKE INTELIGENCIJE I MAŠINSKOG UČENJA

Technical school Pirot

KA220-VET - Cooperation partnerships in vocational education and training

Project Title: AI tools for VET schools

Document Date: Jun 2025

Authors: Boban Blagojevic, Bojan Ćirić, Aleksandar Madić, Bojana Stojanović

Address Information

Pirot, Takovska 22

Phone: 000.123.45678 | **Fax:** 123.456.7890 | **Web:** https://book.tsp.edu.rs

1. VEŠTAČKA INTELIGENCIJA

Dobrodošli na temu **veštačke inteligencije (AI)**! U ovom odeljku ćemo istražiti fascinantan svet AI, koji uključuje stvaranje sistema koji mogu obavljati zadatke koji obično zahtevaju ljudsku inteligenciju. Naučićete o istoriji i razvoju AI, osnovnim konceptima i etičkim razmatranjima koja okružuju njegovu upotrebu. Takođe ćemo razgovarati o različitim vrstama AI, uključujući uske, opšte i superinteligencije, i kako AI transformiše različite aspekte našeg svakodnevnog života.

1.1 Koncept veštačke inteligencije

Veštačka inteligencija (AI) je disciplina koja se bavi razvojem programa, koji svojim mogućnostima ostavljaju utisak inteligentnog ponašanja. Ove programe karakteriše sposobnost uočavanja složenih odnosa i izvlačenja zaključaka na osnovu njih. Videćemo da ove aktivnosti imaju osnovu u disciplinama kao što su matematika, informatika, informatika i robotika. Zbog svoje šire distribucije, polje je takođe povezano sa svim drugim disciplinama koje se bave razumevanjem inteligencije, kao što su neuronauka, filozofija i umetnost i njeni efekti na društvo, kao što su sociologija, pravo i etika.

Važno je naglasiti da svaki program koji ima neki oblik inteligentnog ponašanja mora biti zasnovan na veštačkoj inteligenciji. Pogledajmo program koji otvara vrata prilikom ulaska u zgradu. Ova funkcionalnost može biti omogućena senzorima blizine, koji detektuju prisustvo ili zahtevaju da se unese kod koji treba da odgovara očekivanom kodu. Mogli bismo da pokrijemo oba ova scenarija klasičnim tehnikama programiranja upoređivanjem rastojanja merenog senzorom sa nekim graničnim rastojanjem, tj. Ja ću da unesem kod sa ispravnim kodom. S druge strane, ako je potrebno da prateća kamera prepozna naše lice da uđe u zgradu, mi ćemo, kao što ćemo uskoro videti, trebati pomoć veštačke inteligencije.

Istorija veštačke inteligencije

Istorija veštačke inteligencije (AI) obeležena je značajnim prekretnicama koje odražavaju evoluciju tehnologije i ljudsko razumevanje inteligencije. Od svojih konceptualnih početaka do trenutnih aplikacija, AI je prošao kroz različite transformacije pod uticajem istraživačkih otkrića, društvenih potreba i tehnološkog napretka.

Rani temelji (1950-ih)

Putovanje AI počelo je 1950-ih, decenije koja je postavila temelje za ovu oblast. Godine 1950, Alan Turing je objavio svoj temeljni rad "Računarske mašine i inteligencija", uvodeći Turingov test, koji je imao za cilj da proceni sposobnost mašine da pokaže inteligentno ponašanje koje se ne razlikuje od ljudskog. Sledeće godine, Marvin Minski i Dean Edmonds stvorili su SNARC, prvu veštačku neuronsku mrežu (ANN), simulirajući mrežu neurona pomoću vakuumskih cevi. Godine 1956., na radionici koju su organizovali John

McCarthy, Marvin Minski, Nathaniel Rochester i Claude Shannon, skovan je termin "veštačka inteligencija". Ovaj događaj se široko smatra osnivačkim trenutkom AI kao posebnog polja.

Uspon mašinskog učenja (1960-1970-ih)

Šezdesetih godina prošlog veka došlo je do razvoja ranih AI programa kao što su Eliza, chatbot sposoban da se uključi u jednostavne razgovore, i Shakei, prvi mobilni robot sa AI mogućnostima. Međutim, ovaj period se takođe suočio sa izazovima. Ograničenja ranih neuronskih mreža istaknuta su 1969. godine kada su Marvin Minski i Seimour Papert objavili Perceptrons, što je dovelo do pada istraživanja neuronskih mreža u korist simboličkih AI pristupa. Sedamdesete su obeležile "AI zimu", koju karakteriše smanjeno finansiranje i interesovanje zbog neispunjenih očekivanja. Ključni izveštaj Jamesa Lighthilla 1973. godine kritikovao je istraživanje AI u Velikoj Britaniji, što je dovelo do značajnih smanjenja vladine podrške

Oživljavanje i ekspanzija (1980-1990-ih)

AI je doživeo renesansu 1980-ih sa komercijalizacijom Lisp mašina i obnovljenim interesovanjem za ekspertne sisteme. U ovom periodu došlo je do napretka u predstavljanju znanja i tehnikama rasuđivanja, što je omogućilo sofisticiranije AI aplikacije. Uvođenje algoritama za obuku višeslojnih ANN-ova dodatno je revitalizovalo istraživanje neuronske mreže. Do 1990-ih, AI je počeo da se integriše u praktične aplikacije kao što su prepoznavanje govora i obrada video zapisa. IBM-ov Deep Blue je dospio na naslovne strane pobedom nad svetskim šahovskim šampionom Garijem Kasparovom 1997. godine, pokazujući potencijal AI u strateškom razmišljanju

Moderno doba (2000-danas)

21. vek je bio svedok eksplozije AI sposobnosti vođenih napretkom u mašinskom učenju, posebno dubokom učenju. Tehnologije kao što su IBM Vatson, lični asistenti kao što su Siri i Aleka, sistemi za prepoznavanje lica i generativni modeli poput GPT-a postali su sastavni deo svakodnevnog života. Porast velikih podataka i povećana računarska snaga omogućili su ovim sistemima da uče iz ogromnih količina informacija, što je dovelo do značajnih poboljšanja performansi u različitim domenima. Danas diskusije o AI takođe obuhvataju etička razmatranja i društvene uticaje. Kako AI sistemi postaju sve zastupljeniji, pitanja vezana za privatnost, pristrasnost i odgovornost se sve više ispituju

Ključne prekretnice

1950: Alan Turing predlaže Turingov test.

1956: Dartmouth konferencija uspostavlja AI kao polje.

1966: ELIZA, rani NLP program, je stvoren.

1997: IBM-ov Deep Blue porazio Garija Kasparova.

2012: Proboj dubokog učenja sa AlekNet-om koji je pobedio na takmičenju ImageNet.

2020-e: AI postaje sastavni deo različitih industrija, podižući etičke i regulatorne probleme.

1.2 Turingov test

Turingov test, koji je predložio Alan Turing (1950), dizajniran je kao misaoni eksperiment koji bi zaobišao filozofsku neodređenost pitanja "Može li mašina misliti?" Računar prolazi test ako ljudski ispitivač, nakon postavljanja nekih pisanih pitanja, ne može reći da li pisani odgovori dolaze od osobe ili od računara. Poglavlje 28 govori o detaljima testa i da li bi računar zaista bio inteligentan ako je prošao. Za sada, napominjemo da programiranje računara da prođe rigorozno primenjen test pruža mnogo toga za rad. Računaru će biti potrebne sledeće mogućnosti:

- obrada prirodnog jezika za uspešnu komunikaciju na ljudskom jeziku;
- predstavljanje znanja za čuvanje onoga što zna ili čuje;
- automatizovano rasuđivanje za odgovaranje na pitanja i izvlačenje novih zaključaka;
- Mašinsko učenje da se prilagode novim okolnostima i da otkriju i ekstrapoliraju obrasce.

Turing je posmatrao fizičku simulaciju osobe kao nepotrebnu da pokaže inteligenciju. Međutim, drugi istraživači su predložili ukupan Turingov test, koji zahteva interakciju sa objektima i ljudima u stvarnom svetu. Da prođe ukupan Turingov test, robot će morati

- kompjuterski vid i prepoznavanje govora za sagledavanje sveta;
- robotika za manipulaciju objektima i kretanje.

Ovih šest disciplina čine većinu AI. Ipak, AI istraživači su posvetili malo napora da prođu Turingov test, verujući da je važnije da se prouče osnovne principe inteligencije. Potraga za "veštačkim letom" uspela je kada su inženjeri i pronalazači prestali da imitiraju ptice i počeli da koriste vazdušne tunele i uče o aerodinamici. Vazduhoplovni inženjerski tekstovi ne definišu cilj svoje oblasti kao pravljenje "mašina koje lete tako tačno kao golubovi da mogu da prevare čak i druge golubove."

Može li mašina misliti?

Zanimljivo je da je razvoj veštačke inteligencije bio prepun prepreka.

Pitanje je: "Može li mašina misliti?" Godine 1950. engleski matematičar Alan Turing signalizirao je početak razvoja polja koje je danas poznato kao veštačka inteligencija. Nekoliko godina nakon ove sjajne ideje, 1956. godine, eminentni naučnici John McCarthy, Marvin Minski, Nathaniel Rochester i Claude Shannon okupili su se na konferenciji u Dartmouthu, koja je trajala čak mjesec dana, sa željom da se definišu istraživački ciljevi i protokoli u ovoj oblasti. Tada je dobio svoje zvanično ime i po prvi put zapravo predstavljen kao veštačka inteligencija.

U vreme pojave veštačke inteligencije, računari su bili veoma različiti nego što su danas. Bili su mnogo manji u kapacitetu i brzini, i mnogo viši u ceni. Zbog toga su istraživanja u ovoj oblasti zahtevala različita dizajnerska rešenja i zavisila od zvanične podrške i stabilnih izvora finansiranja.

U prvom talasu razvoja veštačke inteligencije, koji je trajao do početka sedamdesetih godina 20. veka, pojavili su se mnogi zanimljivi programi. Prvi među njima bio je *program Logika teoretičar,* napisan 1956. godine, koji je ispitivao kapacitete matematičke logike i izvođenje zaključaka. Ovaj program je uspeo da dokaže 38 od prvih 52 teoreme navedene u čuvenoj knjizi *Principi matematike.* Tu je i ELIZA program, koji bi mogao da simulira razgovor sa korisnicima prateći jednostavna pravila i konstrukcije na engleskom jeziku. Motivacija za istraživanje u oblasti komunikacije došla je iz predloga Alana Turinga da se proglase

inteligentne mašine koje mogu da komuniciraju na takav način da ne ostavljaju utisak da osoba razgovara sa mašinom. Ovaj test je sada poznat kao Turingov test.

U periodu do početka 1970-ih, pojavile su se mnoge ideje koje će se kasnije koristiti za proboj u modernoj veštačkoj inteligenciji. Jedna takva je ideja perceptrona, osnova današnjih neuronskih mreža, koju je 1957. godine uveo Frank Rosenblatt. U optimističnim izjavama ovog istraživača, perceptron je imao moć da uči, donosi odluke i prevodi jezike, ali je trebalo mnogo vremena da se to potvrdi.

Zbog nedostatka sredstava, prvi talas razvoja veštačke inteligencije pratila je takozvana prva zima veštačke inteligencije. Ovaj status je delimično posledica ambicioznih projekata čiji su rezultati nedostajali zbog ograničenih računarskih kapaciteta i nedostatka dostupnih podataka.

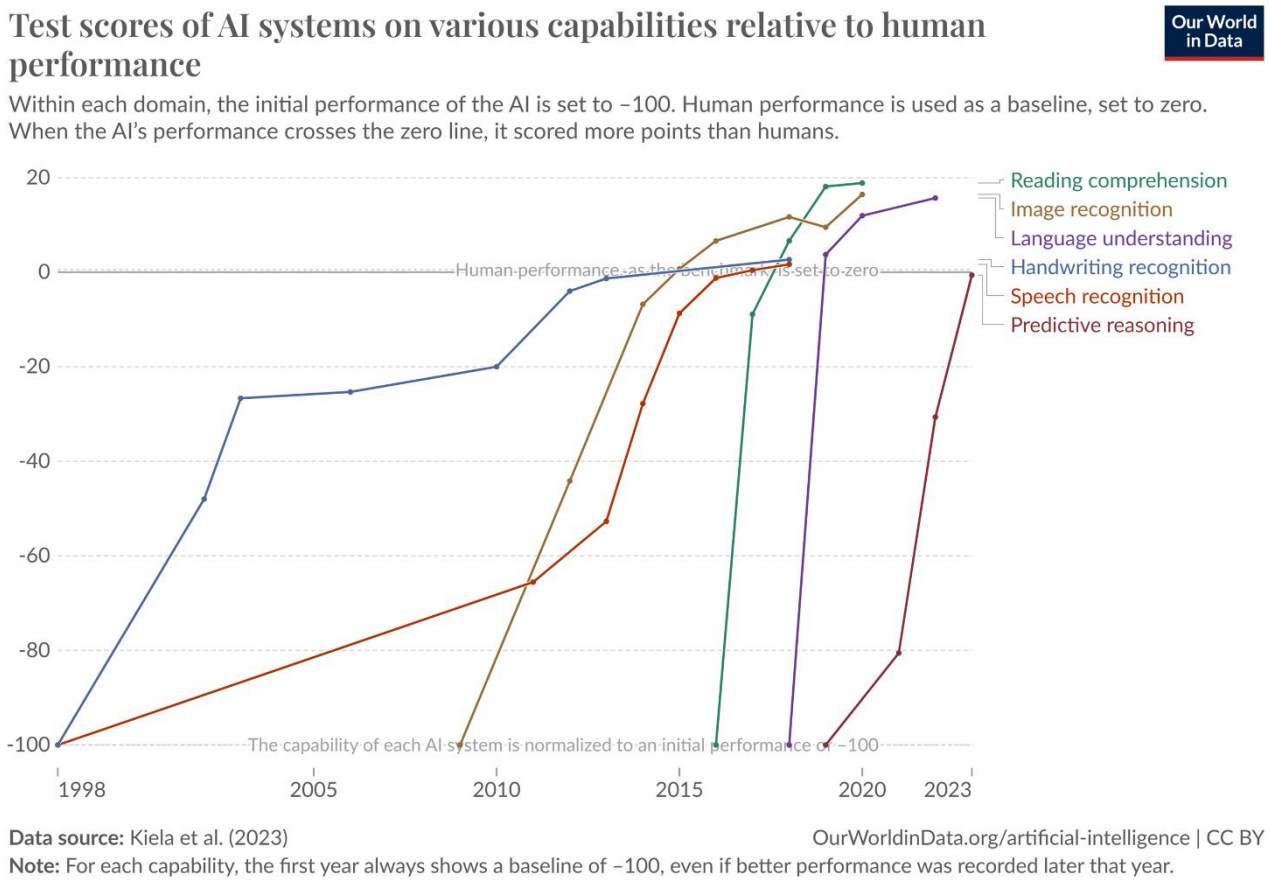
Jedan od zanimljivih i stimulativnih događaja u istoriji veštačke inteligencije dogodio se 1997. godine kada *je IBM-ov* sistem DeepBlue uspeo da pobedi šahovsku igru svetskog velemajstora Garija Kasparova. DeepBlue *sistem* je predstavnik klase takozvanih ekspertnih sistema (eng. *Ekspertni sistemi)*, sistemi koji, na osnovu baze podataka koja sadrži mnoštvo pravila u obliku ako-onda, primenom logičkih pravila, mogu imitirati obrazloženje stručnjaka domena i dati tačan rezultat. Google-ov DeepMind-ov *AlphaGo sistem imao je sličan efekat na razvoj veštačke inteligencije* skoro 20 godina kasnije, 2016. godine, kada je pobedio svetskog šampiona Lee Sedola u igri Go.

U 2011. godini, sposobnost mašina da pronađu odgovor na pitanje postavljeno na engleskom jeziku pokazala je *IBM-ov Votson* sistem u *kvizu* Opasnost!.*Votson* je pobedio svoja dva protivnika, pobednike prethodnih izdanja kviza, dajući najbrže tačne odgovore na postavljena pitanja. Izvori koji su pisali o ovom događaju izjavili su da je *Votson* bio u stanju da obradi 500 GB sadržaja u sekundi, tj. Oko milion knjiga.

S druge strane, kapacitet mašina za prepoznavanje i razlikovanje objekata na slikama pokazao je 2012. godine Google *Ks* tim, koji je stvorio program koji može prepoznati mačke na slikama. Ovaj program je video preko 10 miliona slika u 3 dana i naučio da prepozna mačke. Do danas, kapacitet sistema za prepoznavanje je znatno poboljšan i u mnogim aplikacijama takvi sistemi daju tačnije odgovore od većine ljudi. Na slici ispod možete videti sisteme trendova razvoja za prepoznavanje rukom pisanog teksta, prepoznavanje govora, prepoznavanje slika, a dva novija rezultata sa izuzetnim rastom sposobnosti odnose se na zadatke razumevanja jezika.

Ova dostignuća su takođe bila uvod u daleko svetliji nastavak razvoja veštačke inteligencije, kako zbog dostupnosti Interneta, veba i više podataka, tako i zbog računara čija je procesorska snaga neuporedivo veća od računara 1950-ih. To je takođe dovelo do promene paradigme koja je bila dominantna na terenu i prelaska sa sistema zasnovanih na logici na sisteme zasnovane na statistici.

Priča o razvoju veštačke inteligencije takođe je vezana za robote. Ne samo u naučnofantastičnim romanima i filmovima, već i kada je u pitanju pojava pravih robota. Godine 1950. američki naučnik Claude Shannon dizajnirao je miša koji bi mogao pronaći svoj put i izaći iz lavirinta. U duhu grčke mitologije, miš je nazvan Tezej. Godine 1966. tim naučnika sa Istraživačkog instituta Stanford započeo je rad na razvoju robota Shakei, prvog robota sposobnog da se kreće i zaključuje o životnoj sredini. Prvo autonomno vozilo ALVINN (akronim za Autonomous *Land Vehicle In a Neural Network)*, na kojem je radio tim istraživača sa Univerziteta Carnegie Mellon, izgrađeno je 1989. godine i uspešno je prešlo 145 kilometara putujući brzinom od 110 kilometara na sat između ostalih automobila.



Slika je preuzeta sa https://ourworldindata.org/brief-history-of-ai

1.3 Oblasti veštačke inteligencije

U ovoj lekciji ćemo istražiti neke oblasti veštačke inteligencije. Granice između njih nisu stroge, a često tehnike koje se koriste za rešavanje problema jedne oblasti mogu biti od pomoći u rešavanju problema druge oblasti. Stvarna moć veštačke inteligencije će zapravo biti u povezivanju svih oblasti.

Računarski vid

Kompjuterski vid je polje veštačke inteligencije koje se bavi razvojem algoritama i alata koji računarima daju sposobnost da razumeju vizuelni svet poput ljudi. Takvi su, na primer, zadaci prepoznavanja objekata na slikama, razumevanja njihovih odnosa, prepoznavanja boja i tekstura, zatim prepoznavanja pokreta, akcija i njihovih karakteristika. Kako se ova oblast prvenstveno bavi analizom slika i video zapisa, upoznaćemo i neke od najčešćih zadataka u ovoj oblasti.

Zadatak **klasifikacije slike** se koristi da bi se utvrdilo koji tip objekta je prisutan na slici. Na primer, utvrđivanje da li postoji pas na slici je zadatak klasifikacije slika. **Detekcija objekata** je zadatak lociranja objekata i odgovaranja na pitanje gde se tačno objekti nalaze na slici. Takav je, na primer, zadatak kadriranja psa i mačke koji su na slici ispod. Zadatak **segmentacije slike** se koristi za određivanje tačnog

oblika objekata koji se pojavljuju na slici. Dakle, sada, finije razdvajanje kontura psa i mačke na trećoj slici je primer segmentacije.

Tri osnovna zadatka kompjuterskog vida u radu sa slikama

Svi ovi zadaci su veoma primenljivi u mnogim disciplinama kao što su autonomna vožnja, analiza medicinskih slika ili analiza satelitskih snimaka, i omogućavaju nam da lakše pretražujemo i organizujemo slike.

Za koje zadatke padaju sledeći problemi:
Da biste saznali da li postoji pešak na slici.
razdvajanje kontura semafora, trotoara i pešaka na slici,
Da li želite da znate gde je znak na slici?
Kada je u pitanju obrada video zapisa, najčešći zadaci su praćenje objekata, prepoznavanje akcija i pozicioniranje.

Zadatak *praćenja objekata*, kao što ime sugeriše, omogućava vam da pratite objekte u video zapisu u realnom vremenu. Na primer, praćenje susednog automobila tokom autonomne vožnje i praćenje kretanja igrača tokom meča su primeri praćenja objekata.

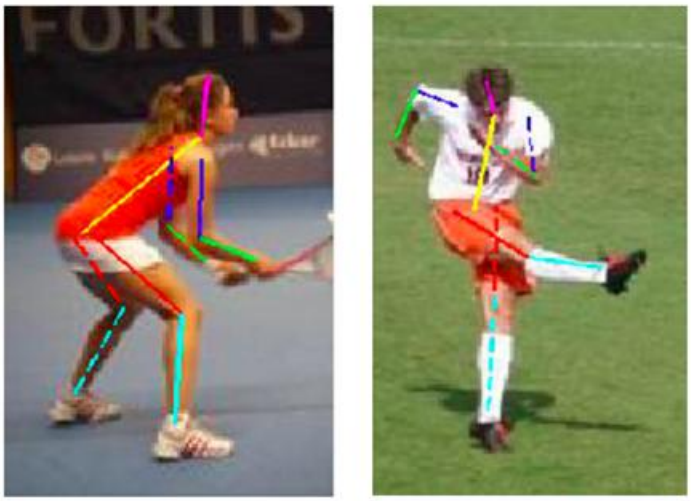
Praćenje objekata

(slika preuzeta iz https://docs.ultralytics.com/modes/track/)

Zadatak **prepoznavanja akcija** (*eng. Action recognition)* je sposobnost prepoznavanja i imenovanja akcije koja je prisutna u videu, na primer, skakanje u vodu ili zatvaranje prozora. Ovi zadaci nam pomažu da bolje razumemo video sadržaj i efikasnije ga tražimo.

Primeri prepoznavanja akcija u video zapisima

Procena poze je zadatak koji se bavi prepoznavanjem figura ljudi u video zapisima u realnom vremenu i izvlačenjem svih ključnih tačaka njihovog skeleta. To su najčešći horoidi očiju, nosa, usta, ramena, laktova, struka, ruku, kolena i stopala. Ovi zadaci nam pomažu u interaktivnim animacijama, modeliranju proširene stvarnosti i raznim drugim aplikacijama.



Zadatak za prepoznavanje položaja objekta: Slike iz skupa podataka Leeds Sports Pose

Koji zadatak treba da rešimo da bismo:

- analizirati da li sedimo ispravno,
- prepoznati izlaz u dvorište kućnog ljubimca,
- Da li želite da pratite kretanje kupca u prodavnici?

Kasnije ćemo doći do skupova podataka koji se koriste u računarskom vidu i konvolucijskim mrežnim zadacima, posebnom tipu neuronske mreže koja se koristi u zadacima obrade slika i video zapisa.

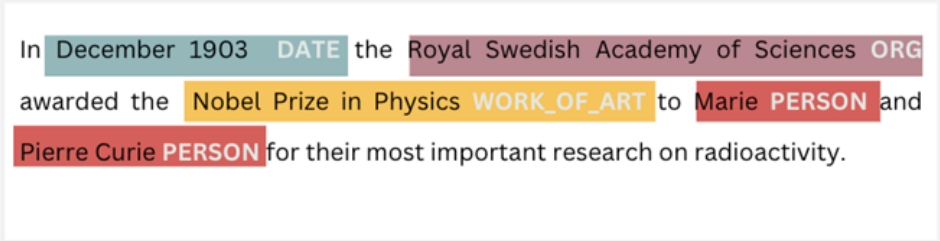
Obrada prirodnog jezika

Obrada prirodnog jezika *(NLP)* je oblast veštačke inteligencije koja se bavi zadacima vezanim za razumevanje i generisanje prirodnog jezika. Kao što znamo, postoji preko 7.000 jezika, a svaki od njih ima svoje posebnosti u pogledu vokabulara, gramatičkih pravila i značenja. Neki uobičajeni zadaci sa kojima se susreću u obradi prirodnog jezika biće opisani u nastavku.

Baš kao u zadacima klasifikacije slika, u zadacima klasifikacije teksta pokušavamo da zaključimo da li tekst pripada kategoriji ili ne. Na primer, da li je to novinski članak na temu sporta, da li je napisan na španskom, da li je pozitivan, tj. sadrži neki pohvalni komentar, da li je istinit ili lažan i slično.

Klasifikacija teksta

Imenovano prepoznavanje entiteta je zadatak koji se odnosi na prepoznavanje nekih delova teksta koji su relevantni za njegovu dalju analizu. To su obično imena ljudi koji se pojavljuju u njemu, datumi, imena geolokacije ili u nekim stručnim tekstovima, na primer u oblasti medicine, simptomi ili imena bolesti. Pod jednim imenom, ovi delovi teksta se nazivaju entitetima.



Primer označavanja imenovanih entiteta

Zadatak **mašine za prevođenje** je da razvije alate koji nam omogućavaju da prevedemo sadržaj sa jednog jezika na sadržaj drugog jezika. Složit ćemo se da je ovaj zadatak osnova za uspješnu komunikaciju i dostupnost informacija, ali i da je komplikovan jer svaki jezik i svaka kultura koju jezik predstavlja ima svoje posebnosti kao što su fraze, idiomi, sleng ili sarkazam koji su vrlo izazovni za prevođenje (kako prevesti *neplodni mozak na pašu)?)*.

Sistemi za odgovaranje na pitanja bave se pitanjem kako pronaći konkretan odgovor na određeno pitanje. Oni su generalizacije klasičnih sistema za pronalaženje informacija i omogućavaju nam da lakše dobijemo informacije koje su nam potrebne.

Sumiranje svih važnih informacija iz više različitih izvora poznato je kao zadatak **sumiranja**. Baš kao i u prethodnom zadatku, rezimei koji dolaze sa zadacima sumiranja trebalo bi da nam olakšaju da prođemo kroz veću količinu sadržaja ili da nas podsete na važne informacije i detalje sadržaja koji smo pročitali.

Sažetak

Pored zadataka vezanih za tekst i tekstualni sadržaj, obrada prirodnog jezika bavi se i analizom govora. Posebno postoje dva zadatka: *govor u tekst* i obrnuto, *tekst u govor*. Ove dve grupe zadataka su posebno važne za razvoj ličnih asistenata, programa kao što su *Siri, Cortana*, ili *Aleka,* koji mogu da razumeju glasovne poruke i u skladu sa tim obavljaju traženi zadatak, na primer, podesite alarm ili pozovite nekoga iz imenika.

Za koje zadatke padaju sledeći problemi:

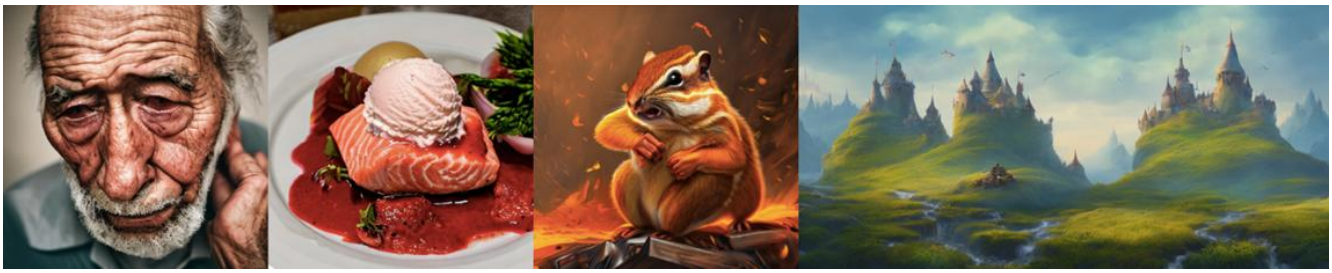
- Izdvajanje imena organizacije u tekstu,
- Pronalaženje ko je autor knjige,
- Koje je značenje reči Netizen ?

Generativna veštačka inteligencija

Generativna veštačka inteligencija (AGI) je oblast veštačke inteligencije koja se bavi generisanjem sadržaja kao što su slike, tekst, audio ili video. Tokom proteklih nekoliko godina, otkrića u ovoj oblasti su bila impresivna.

ChatGPT je program koji je napravio proboj u oblasti generisanja tekstualnog sadržaja. On, u skladu sa uputstvima korisnika, takozvanim uputstvima, može generisati odgovarajući tekstualni sadržaj. Treba imati na umu da tekstovi generisani na ovaj način ne moraju biti apsolutno tačni, mogu sadržavati netačne podatke, izmišljene reference ili uvredljiv sadržaj. Stoga, pre upotrebe, trebalo bi da proverite sve što je program generisao. Ako kreirate nalog na chat.openai.com, možete sami isprobati kako *funkcioniše program ChatGPT*. Iza programa *ChatGPT* stoji OpenAI zajednica.

Program StableDiffusion, za razliku od *ChatGPT-a,* koji generiše tekst, generiše slike na osnovu uputstava. Na primer, sve slike navedene u nastavku su generisane od strane ovog programa. To je otvorenog koda i može se preuzeti sa zvaničnog GitHub spremišta sa pratećim kodom. Sam model možete testirati na https://stablediffusionweb.com/ . Imajte na umu da ovu uslugu koristi veliki broj ljudi besplatno i ponekad nije dostupna. Sam naziv programa je popularna tehnika koja se koristi u ovoj oblasti.



Primeri slika koje generiše StableDiffusion

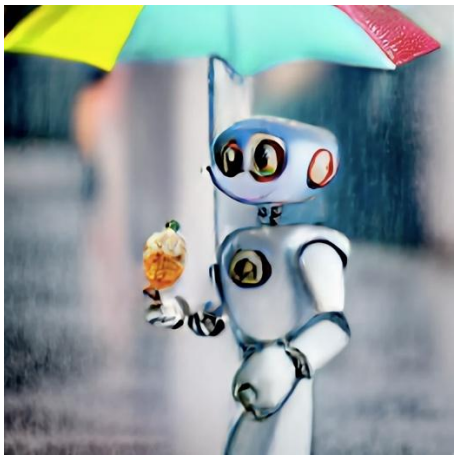
Examples of images generated by StableDiffusion

Često, prilikom generisanja slika, može se odabrati i željeni stil nove slike. Ova tehnika je poznata kao **prenos stila**. Primer možete videti na slici ispod.

Pored slika i teksta, veštačka inteligencija takođe može generisati audio sadržaj. Na ovom linku možete testirati Meta-ov *program MusicGen,* opisujući rečima kakvu muziku želite da generišete i eventualno ostavljajući primer za prenos stila. Tada možete slušati svoj sadržaj. Takođe možete probati sa programima koji obavljaju prenos stila prilikom generisanja glasa (oponašaju glas druge osobe) ili sami komponuju muziku na osnovu onoga što su već "čuli" u podacima. Jedan takav projekat je Magenta.
Link https://magenta.github.io/listen-to-transformer će vas odvesti do njega.

Zamolite ChatGPT da napravi kviz sa pitanjima o veštačkoj inteligenciji, a zatim pogledajte na koliko pitanja možete odgovoriti.

Pokušajte da date uputstva za StableDiffusion ili Dalle-mini koji će generisati sliku kao što je ova:



OpenAI-jev DALL-E program takođe generiše slike na osnovu korisničkih smernica. Dalle-mini program je javno dostupna verzija ovog programa. Dostupan je na https://huggingface.co/spaces/dalle-mini/dalle-mini.

Igranje igara

Jedan od prvih zadataka u kojima je isprobana veštačka inteligencija je igra šaha. Svojom pobedom nad velemajstorom Garrijem Kasparovim, ova oblast istraživanja dobila je mnogo simpatija i podrške sa strane veštačke inteligencije. Iako imaju precizan skup uputstava i pravila, igre se odlikuju svojstvom kombinatorne eksplozije - velikim brojem mogućih izbora akcija nakon određenog broja koraka. To dalje znači da igre ne dozvoljavaju pronalaženje rešenja primenom uobičajenih tehnika programiranja u nekim u razumnom roku. Pored igre šaha, veštačka inteligencija je takođe postala poznata u igri go sa *programom AlphaGo*, a zatim je igrala Atari video igre i strategije kao što su Dota, Starcraft i druge. Na kraju kursa saznaćete više o oblasti koja se zove učenje pojačanja, koja se aktivno primenjuje u ovoj oblasti.

Proverite da li igra koju volite koristi veštačku inteligenciju u nekim od svojih segmenata.

Robotika

Veštačka inteligencija često treba da poboljša ponašanje i sposobnosti fizičkih objekata kao što su roboti, industrijske mašine, automobili, bespilotne letelice, kućni aparati ili medicinski uređaji. Informacije o svetu dopiru do ovih objekata putem glasovnih uputstava, snimaka kamere ili merenja senzora, a njihov zadatak je da ih obrade i transformišu u odluke. Uloga veštačke inteligencije u ovoj sferi je da poboljša sposobnosti objekata i pomogne im da usvoje inteligentno ponašanje.

1.4 Regulacija veštačke inteligencije, pravni i etički izazovi

Brzi napredak tehnologija veštačke inteligencije (AI) izazvao je hitne rasprave oko njihove regulacije. Kako AI sistemi postaju sve više integrisani u različite sektore, uključujući zdravstvo, finansije i transport, potreba za efikasnim pravnim okvirima i etičkim smernicama postaje najvažnija. Ovaj tekst istražuje složenost regulisanja AI, naglašavajući pravne i etičke izazove koji se javljaju u ovom pejzažu koji se razvija.

Potreba za regulacijom

AI tehnologije poseduju jedinstvene karakteristike koje ih razlikuju od tradicionalnih tehnologija. Oni često funkcionišu kao "crne kutije", gde njihovi procesi donošenja odluka nisu transparentni čak ni njihovim programerima. Ova netransparentnost izaziva značajnu zabrinutost u vezi sa odgovornošću, pristrasnošću i potencijalom za štetu. Kako AI sistemi mogu uticati na kritične oblasti kao što su odluke o zapošljavanju, ishodi krivičnog pravosuđa i javna bezbednost, ulozi za efikasnu regulaciju su visoki.

Ključni pokretači za regulaciju:

- **Bezbednost i blagostanje**: Obezbeđivanje da AI sistemi ne ugrožavaju javnu bezbednost ili individualna prava je primarna briga. Incidenti koji uključuju autonomna vozila ili pristrasne algoritme u procesima zapošljavanja naglašavaju potrebu za regulatornim nadzorom.
- **Odgovornost:** Uspostavljanje jasnih mehanizama odgovornosti je od suštinskog značaja za rešavanje ko je odgovoran kada AI sistemi nanose štetu ili donose pogrešne odluke.
- **Etička upotreba**: Kako AI sistemi mogu produžiti predrasude prisutne u podacima o obuci, propisi moraju osigurati da se poštuju etički standardi kako bi se sprečila diskriminacija i nepravedan tretman.

Pravni izazovi u regulisanju AI

Pravni pejzaž koji okružuje regulaciju AI je složen i često zaostaje za tehnološkim napretkom. Tradicionalni pravni okviri ne mogu adekvatno adresirati nijanse AI sistema.

1. Definisanje odgovornosti

Jedan od najvažnijih izazova je utvrđivanje odgovornosti kada AI sistem nanese štetu. Postavljaju se pitanja o tome da li odgovornost treba da padne na programere, korisnike ili samu AI. Na primer, ako je autonomno vozilo uključeno u nesreću, da li bi proizvođač trebao biti odgovoran, ili odgovornost leži na vlasniku?

2. Pitanja intelektualne svojine

Kako sadržaj generisan AI postaje sve zastupljeniji, pojavljuju se pitanja o pravima intelektualne svojine. Ko je vlasnik prava na dela stvorena od strane AI? Sadašnji zakoni možda neće dovoljno pokriti ove scenarije, što dovodi do potencijalnih sukoba oko vlasništva i autorskih prava.

3. Zabrinutost za privatnost podataka

AI sistemi se u velikoj meri oslanjaju na podatke za obuku i rad. Prikupljanje i korišćenje ličnih podataka izaziva značajnu zabrinutost za privatnost. Propisi moraju uravnotežiti potrebu za podacima za poboljšanje AI mogućnosti sa pravima pojedinaca na privatnost i zaštitu podataka.

Etički izazovi u regulisanju AI

Pored pravnih razmatranja, etički izazovi igraju ključnu ulogu u oblikovanju regulacije AI.

1. Pristrasnost i pravičnost

AI sistemi mogu nenamerno produžiti predrasude prisutne u podacima o obuci, što dovodi do nepravednih ishoda. Regulatori moraju uspostaviti smjernice kako bi osigurali pravednost i ublažili pristrasnost u AI algoritmima. To uključuje ne samo tehnička rešenja, već i posvećenost etičkim principima koji daju prioritet pravičnom tretmanu.

2. Transparentnost i objašnjivost

Priroda "crne kutije" mnogih AI sistema komplikuje napore za transparentnost. Korisnicima i pogođenim pojedincima često nedostaje uvid u to kako se donose odluke. Propisi treba da promovišu objašnjivost, osiguravajući da pojedinci mogu da razumeju razloge iza odluka koje donose AI sistemi.

3. Informisani pristanak

Kako AI tehnologije sve više utiču na lične odluke - kao što su odobrenja kredita ili prijave za posao - obezbeđivanje informisanog pristanka postaje kritično. Pojedinci treba da budu svesni kako se njihovi podaci koriste i kako AI utiče na procese donošenja odluka koji utiču na njih.

Pristupi regulaciji

S obzirom na ove izazove, pojavili su se različiti pristupi regulisanju AI.

1. Okviri zasnovani na riziku

Evropska unija je predložila pristup regulaciji AI zasnovan na riziku, kategorizujući aplikacije na osnovu nivoa rizika - od minimalnih do neprihvatljivih rizika. Ovaj okvir omogućava prilagođene propise koji se bave specifičnim problemima povezanim sa različitim vrstama AI aplikacija.

2. Kontinuirani nadzor

Regulisanje AI zahteva stalnu budnost zbog svoje prirode koja se brzo razvija. Regulatorna tela moraju da se prilagode novim dostignućima u tehnologiji i kontinuirano procenjuju uticaj postojećih propisa na društvo.

3. Zajedničko upravljanje

Efikasna regulacija može uključivati saradnju između vlada, zainteresovanih strana u industriji i organizacija civilnog društva. Angažovanje različitih perspektiva može dovesti do sveobuhvatnijih propisa koji uzimaju u obzir različite interese i etička razmatranja.

Zaključak

Regulacija veštačke inteligencije predstavlja višestruke pravne i etičke izazove koji zahtevaju pažljivo razmatranje i proaktivne mere. Kako AI nastavlja da prožima različite aspekte života, uspostavljanje robusnih regulatornih okvira je od suštinskog značaja za zaštitu javnog blagostanja, obezbeđivanje odgovornosti i poštovanje etičkih standarda. Rešavanjem ovih izazova kroz zajedničko upravljanje i adaptivne regulatorne pristupe, društvo može iskoristiti prednosti AI uz efikasno ublažavanje rizika.

1.5 Uska, opšta i superinteligencija

Sada kada znamo oblasti primene veštačke inteligencije i njen domet, možemo uvesti i koncepte kao što su uska i opšta veštačka inteligencija, superinteligencija i singularnost veštačke inteligencije.

Videli smo da su primeri programa veštačke inteligencije koje smo naveli fokusirani na rešavanje jednog određenog zadatka, na primer, igranje šaha, prevođenje jezika, segmentiranje slika i slično. Kažemo da takvi programi imaju **usku veštačku inteligenciju** (*uska AI)*. Za razliku od veštačke inteligencije, postoji **opšta veštačka inteligencija** (*General AI)*. Programi ove grupe bi trebalo da budu bliži ljudskoj inteligenciji, kako bi se omogućilo rešenje mnoštva različitih zadataka. Prema vodećim istraživačima u ovoj disciplini, programi ovog tipa su još uvek u povoju i neće se razvijati u doglednoj budućnosti.

Superinteligencija je termin koji se odnosi na inteligenciju veću od inteligencije ljudi. Takva inteligencija mogla bi nam pomoći da rešimo probleme kao što su globalno zagrevanje, obezbeđivanje dovoljno hrane ili pronalaženje leka za rak. To bi, hipotetički, moglo da izmakne kontroli, nastavi da se poboljšava, i na neki način ugrozi čovečanstvo. **AI singularnost** je teorijski koncept koji označava dominaciju veštačke inteligencije nad ljudskom inteligencijom i često se susreće kao tema u filmovima i knjigama naučne fantastike.

Gledanje veštačke inteligencije iz ovog ugla je od posebnog interesa za filozofe, istoričare i istraživače u oblasti društvenih nauka. Izraelski istoričar Juval Noa Harari i švedski filozof Nik Bostrom pisali su o njima.

Pokušajte da smislite još jedan primer primene superinteligencije. Koji su problemi koje ljudi, uprkos razvoju društva, nauke i tehnologije, još uvek ne znaju kako da reše?

Istražite značenje termina *egzistencijalni rizik.* Kako to vidite?

Oblast veštačke inteligencije (AI) obuhvata različite sisteme i mogućnosti, koje se mogu svrstati u tri osnovna tipa: **uska inteligencija**, **opšta inteligencija** i **superinteligencija**. Svaka kategorija predstavlja različit nivo složenosti i sposobnosti u AI sistemima, odražavajući tekuću evoluciju tehnologije i njen potencijalni uticaj na društvo. Razumevanje ovih razlika je od ključnog značaja za shvatanje trenutnog pejzaža AI i njegovih budućih implikacija.

Uska veštačka inteligencija (slaba AI)

Uska inteligencija, koja se često naziva slabom AI, dizajnirana je za obavljanje određenih zadataka ili rešavanje određenih problema. Ovi AI sistemi se ističu u svojim određenim funkcijama, ali nemaju sposobnost da rade izvan svojih unapred definisanih parametara.

Karakteristike uske inteligencije:

- Funkcionalnost specifična za zadatak: Uski AI sistemi su napravljeni za rukovanje specifičnim zadacima, kao što su prepoznavanje slika, prevođenje jezika ili igranje igara poput šaha. Oni efikasno rade u okviru svog ograničenog opsega, ali ne poseduju opšte sposobnosti rasuđivanja.
- Učenje na osnovu podataka: Ovi sistemi se u velikoj meri oslanjaju na velike skupove podataka za obuku. Algoritmi mašinskog učenja analiziraju obrasce u podacima kako bi predviđali ili odluke na osnovu informacija koje su im pružene.

- Nedostatak samosvesti: Uska AI ne poseduje svest ili samosvest. Radi na osnovu algoritama i unapred definisanih pravila bez razumevanja konteksta ili implikacija svojih akcija.

Primeri uske inteligencije:

- Glasovni asistenti: Aplikacije kao što su Siri i Aleka mogu da obavljaju zadatke kao što su postavljanje podsetnika ili odgovaranje na pitanja, ali ne mogu da se uključe u razgovore izvan svojih programiranih mogućnosti.
- Sistemi preporuka: Platforme kao što su Netflik i Amazon koriste usku AI da predloži sadržaj na osnovu preferencija korisnika, analizirajući prošlo ponašanje da predvidi buduće izbore.
- Autonomna vozila: Dok samovozeći automobili koriste složene algoritme za navigaciju putevima, oni su i dalje ograničeni na zadatke vožnje i ne mogu obavljati druge ljudske kognitivne funkcije.

Opšta veštačka inteligencija (AGI)

Opšta inteligencija, poznata i kao veštačka opšta inteligencija (AGI), odnosi se na teorijski oblik AI koji poseduje sposobnost razumevanja, učenja i primene znanja u širokom spektru zadataka na nivou koji se može uporediti sa ljudskom inteligencijom.

Karakteristike opšte inteligencije:

- Svestrano učenje: AGI može generalizovati znanje iz jednog domena u drugi, omogućavajući mu da se prilagodi novim situacijama bez potrebe za opsežnom prekvalifikacijom.
- Obrazloženje i rešavanje problema: Za razliku od uske AI, AGI može da razmišlja kroz složene probleme, razume apstraktne koncepte i donosi odluke na osnovu nepotpunih informacija.
- Ljudska interakcija: AGI sistemi bi bili sposobni da se uključe u prirodne razgovore sa ljudima, pokazujući emocionalno razumevanje i društvenu svest.

Trenutni status opšte obaveštajne službe:

Od sada, AGI ostaje uglavnom teoretski. Iako je postignut značajan napredak u mašinskom učenju i neuronskim mrežama, nijedan postojeći sistem ne poseduje čitav spektar kognitivnih sposobnosti povezanih sa ljudskom inteligencijom. Istraživači nastavljaju da istražuju puteve ka postizanju AGI, fokusirajući se na razvoj algoritama koji mogu da uče fleksibilnije i autonomnije.

Superinteligencija

Superinteligencija se odnosi na hipotetički oblik AI koji prevazilazi ljudsku inteligenciju u gotovo svakom aspektu. Ovaj koncept postavlja duboka pitanja o budućnosti čovečanstva i etičkim implikacijama stvaranja mašina koje bi potencijalno mogle nadmašiti svoje tvorce.

Karakteristike superinteligencije:

- Eksponencijalni rast: Superinteligentni sistemi bi imali kapacitet za brzo samousavršavanje, što bi dovelo do eksplozije inteligencije u kojoj bi mašine mogle da poboljšaju svoje sposobnosti izvan ljudskog shvatanja.
- Sposobnosti rešavanja problema: Takvi sistemi mogu da reše složene globalne izazove - u rasponu od klimatskih promena do iskorenjivanja bolesti - efikasnije od bilo koje ljudske ili kolektivne grupe.

- Etička razmatranja: Razvoj superinteligencije postavlja značajne etičke dileme u vezi sa kontrolom, bezbednošću i potencijalnim posledicama za društvo. Obezbeđivanje da se superinteligentni sistemi usklade sa ljudskim vrednostima je kritična briga.

Teorijske implikacije:

Diskusija oko superinteligencije često uključuje scenarije o "singularnosti", tački u kojoj tehnološki rast postaje nekontrolisan i nepovratan. Ovaj koncept je izazvao rasprave među naučnicima, etičarima i futuristima o potencijalnim rizicima povezanim sa stvaranjem entiteta koji bi mogli da rade nezavisno od ljudskog nadzora.

Izazovi u napredovanju ka opštoj i superinteligenciji

Iako potraga za opštom i superinteligentnom AI predstavlja uzbudljive mogućnosti, mora se rešiti nekoliko izazova:

1. Tehnička ograničenja: Trenutni AI sistemi se bore sa generalizacijom; oni se ističu u uskim zadacima, ali ne uspevaju kada se suoče sa nepoznatim kontekstima ili problemima izvan njihovih podataka o obuci.
2. Etička pitanja: Razvoj AGI i superinteligencije postavlja etička pitanja o autonomiji, ovlašćenju za donošenje odluka i potencijalu za zloupotrebu u vojnim ili nadzornim aplikacijama.
3. Mere bezbednosti: Obezbeđivanje da napredni AI sistemi rade bezbedno je najvažnije. Istraživači se zalažu za robusne bezbednosne protokole i strategije usklađivanja kako bi se sprečile neželjene posledice.
4. Percepcija javnosti: Nesporazumi o mogućnostima AI mogu dovesti do nerealnih očekivanja ili strahova u vezi sa njenim uticajem na društvo. Javno obrazovanje je od suštinskog značaja za podsticanje informisanih diskusija o AI tehnologijama.

Zaključak

Razlike između uske inteligencije, opšte inteligencije i superinteligencije naglašavaju raznolik pejzaž u oblasti veštačke inteligencije. Dok uska AI nastavlja da napreduje u različitim aplikacijama danas, potraga za AGI ostaje ambiciozan cilj za istraživače širom sveta. Perspektiva superinteligencije poziva i uzbuđenje i oprez dok se društvo bori sa implikacijama stvaranja mašina koje bi mogle da nadmaše ljudski intelekt. Kako se napredak nastavlja, rešavanje etičkih razmatranja i obezbeđivanje odgovornog razvoja biće od ključnog značaja za iskorištavanje prednosti AI uz efikasno ublažavanje rizika.

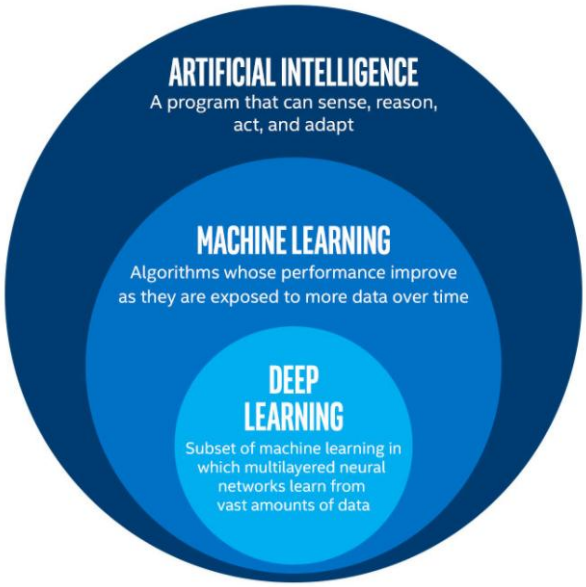
2. MAŠINSKO UČENJE

Dobrodošli na temu **Machine Learning (ML)**! Mašinsko učenje je podskup AI koji se fokusira na razvoj algoritama koji omogućavaju računarima da uče i predviđaju na osnovu podataka. U ovom odeljku ćete steći duboko razumevanje osnovnih koncepata mašinskog učenja, različitih vrsta učenja (nadgledano, bez nadzora, polu-nadzirano i učenje pojačanja) i važnost podataka u ovom procesu. Takođe ćemo pokriti korake koji su uključeni u izgradnju i validaciju modela mašinskog učenja i kako se ovi modeli mogu primeniti za rešavanje problema u stvarnom svetu.

2.1 Odnos između veštačke inteligencije i mašinskog učenja

Veštačka inteligencija (AI) i mašinsko učenje (ML) su dve međusobno povezane oblasti koje su poslednjih godina privukle značajnu pažnju zbog svog transformativnog uticaja na tehnologiju i društvo. Iako se često koriste naizmenično, oni predstavljaju različite koncepte u širem pejzažu računarske inteligencije. Razumevanje njihovog odnosa je od ključnog značaja za shvatanje kako moderni AI sistemi funkcionišu i razvijaju se.

- AI obuhvata širok spektar tehnologija i metodologija koje imaju za cilj stvaranje sistema sposobnih za obavljanje zadataka koji obično zahtevaju ljudsku inteligenciju. Ovi zadaci uključuju rezonovanje, rešavanje problema, percepciju, razumevanje prirodnog jezika i još mnogo toga. Primarni cilj AI je da razvije mašine koje mogu da oponašaju kognitivne funkcije povezane sa ljudskim bićima, omogućavajući im da efikasno obavljaju složene zadatke.



- ML je specijalizovani podskup AI fokusiran na razvoj algoritama koji omogućavaju računarima da uče iz podataka. Umesto da budu eksplicitno programirani za obavljanje zadatka, ML algoritmi identifikuju obrasce unutar skupova podataka i donose predviđanja ili odluke na osnovu tih podataka. Ova karakteristika omogućava ML sistemima da poboljšaju svoje performanse tokom vremena jer su izloženi više podataka.

2.2 Programiranje na osnovu podataka

Značaj velikih količina podataka u mašinskom učenju

U domenu mašinskog učenja (ML), podaci se često nazivaju "gorivom" koje pokreće algoritme i modele. Efikasnost i tačnost sistema mašinskog učenja u velikoj meri zavise od dostupnosti velikih količina visokokvalitetnih podataka. Ovaj odeljak naglašava značaj podataka u ML, uvodi koncept programiranja zasnovanog na podacima i objašnjava osnovne koncepte kao što su skupovi podataka, instance, atributi, ulazne varijable i modeli.

Uloga podataka u mašinskom učenju

Podaci služe kao osnova na kojoj se grade modeli mašinskog učenja. Bez dovoljno podataka, algoritmi mašinskog učenja ne mogu da nauče obrasce ili efikasno predviđaju. Značaj velikih skupova podataka može se sažeti u sledećim tačkama:

1. **Učenje iz primera:** Mašinsko učenje je u osnovi učenje iz primera. Veliki skupovi podataka pružaju raznovrstan spektar slučajeva koji pomažu algoritmima da identifikuju obrasce i odnose unutar podataka.
2. **Poboljšana tačnost**: Modeli obučeni na većim skupovima podataka imaju tendenciju da generalizuju bolje na neviđene podatke, što dovodi do poboljšane tačnosti u predviđanjima. Ovo je posebno važno u aplikacijama kao što su zdravstvena dijagnostika ili finansijsko predviđanje, gde je preciznost od vitalnog značaja.
3. **Robusnost**: Dobro zaokružen skup podataka koji obuhvata različite scenarije omogućava modelima da efikasnije rukuju rubnim slučajevima i anomalijama. Ova robusnost je od suštinskog značaja za aplikacije u stvarnom svetu gde podaci mogu biti bučni ili nepotpuni.
4. **Predstavljanje karakteristika**: Veliki skupovi podataka omogućavaju ekstrakciju značajnih karakteristika koje mogu značajno poboljšati performanse modela. Sa više podataka, algoritmi mogu naučiti složene reprezentacije koje hvataju osnovne trendove.
5. **Ublažavanje preopterećenja**: Imajući značajnu količinu podataka o obuci pomaže u sprečavanju preteranog uklapanja, gde model uči buku umesto osnovne distribucije. Veći skup podataka pruža bolju aproksimaciju prave distribucije podataka.

Programiranje na osnovu podataka

Programiranje zasnovano na podacima je pristup koji naglašava upotrebu podataka kao primarnog pokretača za donošenje odluka i razvoj algoritama. U ovoj paradigmi, programeri se fokusiraju na prikupljanje, analizu i korišćenje podataka kako bi informisali svoje prakse kodiranja, a ne da se oslanjaju isključivo na unapred definisana pravila ili logiku.

Zašto nam je potrebno programiranje zasnovano na podacima

1. **Prilagodljivost**: Programiranje zasnovano na podacima omogućava sistemima da se prilagode promenljivim uslovima kontinuiranim učenjem iz novih ulaznih podataka.
2. **Poboljšano donošenje odluka**: Koristeći velike skupove podataka, programeri mogu kreirati informisane algoritme koji dovode do boljih ishoda.
3. **Automatizacija**: Pristupi zasnovani na podacima olakšavaju automatizaciju omogućavajući mašinama da uče iz istorijskih podataka i predviđaju bez ljudske intervencije.

4. **Skalabilnost**: Kako više podataka postaje dostupno, sistemi mogu skalirati svoje mogućnosti bez potrebe za značajnim promjenama u njihovoj osnovnoj arhitekturi.

Osnovni koncepti mašinskog učenja

1. **Skup podataka**
Skup podataka je strukturirana zbirka podataka koji se koriste za obuku modela mašinskog učenja. Sastoji se od više instanci (tačaka podataka) organizovanih u redove i kolone, gde svaka kolona predstavlja atribut ili funkciju.

2. **Instanca**
Instanca se odnosi na jedan zapis ili posmatranje unutar skupa podataka. Svaka instanca sadrži vrednosti za različite atribute koji opisuju njegove karakteristike.

3. **Atribut**
Atributi su pojedinačne varijable ili karakteristike koje opisuju instancu u skupu podataka. Oni mogu biti numerički (npr. Starost, plata) ili kategorički (npr. Pol, boja). Atributi su ključni za definisanje ulaznog prostora za algoritme mašinskog učenja.

4. **Ulazne varijable**
Ulazne varijable su specifični atributi koji se koriste kao prediktori u modelu mašinskog učenja. Ove varijable pružaju neophodne informacije za model da nauče obrasce i da predviđanja o izlaznim varijablama (ciljevima).

5. **Izlazne varijable**
Izlazne varijable su ciljne vrednosti koje model ima za cilj da predvidi na osnovu ulaznih varijabli. Na primer, u modelu predviđanja cena stanova, ulazne varijable mogu uključivati kvadraturu i lokaciju, dok bi izlazna varijabla bila procenjena cena.

Koncept modela
U mašinskom učenju, model je definisan kao matematička reprezentacija koja mapira date ulazne varijable na izlazne varijable na osnovu naučenih obrazaca iz podataka o obuci. Proces uključuje:

1. **Obuka**: Tokom obuke, model uči iz označenih instanci u skupu podataka podešavanjem svojih parametara kako bi se smanjile greške predviđanja.
2. **Mapiranje**: Kada je obučen, model može uzeti nove ulazne varijable i generisati odgovarajuća izlazna predviđanja na osnovu svojih naučenih mapiranja.
3. **Evaluacija**: Performanse modela se procenjuju korišćenjem metrika kao što su tačnost, preciznost, opoziv i F1 rezultat na validaciji ili testnim skupovima podataka.

Značaj velikih količina odgovarajućih podataka u mašinskom učenju ne može se preceniti; Od suštinskog je značaja za obuku efikasnih modela sposobnih da naprave tačne prognoze u stvarnim aplikacijama. Programiranje zasnovano na podacima pojavljuje se kao vitalna metodologija koja koristi ovo obilje podataka za informisanje o razvoju algoritama i procesima donošenja odluka. Razumevanje osnovnih koncepata kao što su skupovi podataka, instance, atributi, ulazne varijable, izlazne varijable i modeli postavlja temelje za dalje istraživanje tehnika mašinskog učenja i njihove primene u različitim domenima. Kako nastavljamo da koristimo moć podataka u mašinskom učenju, imperativ je dati prioritet kvalitetnom

prikupljanju podataka i praksama upravljanja kako bi se osigurali robusni i pouzdani rezultati u AI sistemima.

Ovu priču ćemo započeti mozgalicom. Pokušajte da zaključite iz brojeva koje imate u levoj koloni (mi ćemo ih nazvati **ulazima**) kako se oni odnose na brojeve u desnoj koloni (mi ćemo ih zvati **izlazi)**.

Entrance	Way
1, 9, 5, 4	19
3, 8, 11	22
6, 7, 9, 2, 2	26
2, 3, 6	11

Pretpostavljamo da vam ovaj zadatak nije mnogo smetao i da ste već u drugom ili trećem redu tabele došli na ideju da brojevi u drugoj koloni, tj. izlazu, predstavljaju zbir brojeva koji se nalaze u levoj koloni, tj. na ulazu. Za ovaj zadatak, takođe znate kako da napišete algoritam koji vas vodi do rešenja. Na primer, u programskom jeziku Python *možemo* izračunati zbir elemenata niza [1, 9, 5, 4] tako što ćemo u početku proglasiti sumu nula, a zatim dodati jedan element u isto vreme dok ne dođemo do kraja niza:

```
numbers = [1, 9, 5, 4]
sum = 0
for element in numbers:
  sum = sum + element
```

(Ugrađena funkcija sume izračunava ovo brzo za nas i mi ga zapravo češće koristimo.)

Sada se okušajte u sledećem mozgalici u kojoj morate da shvatite kako su ulazi i izlazi povezani. Ulazi su sada fotografije životinja, a izlazi su brojevi 0 ili 1.

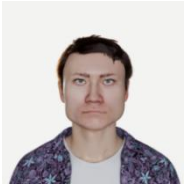
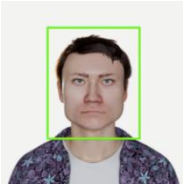
Input	Output
	1
	0

Input	Output
	0
	1
	0

Siguran sam da imate mnogo ideja ovde. I vrlo je moguće da su svi validni! Međutim, pošto su one pored fotografija mačaka, a nule su pored ostalih fotografija životinja, želeli smo da zaključite da je to zadatak u kojem morate prepoznati da li na fotografiji postoji mačka ili ne. Ako postoji fotografija mačke na ulazu, postoji 1 na izlazu, i obrnuto, ako nema fotografije mačke na ulazu, postoji 0 na izlazu. Malo kasnije ćete videti da se ovaj zadatak naziva zadatak binarne klasifikacije i veoma je čest u oblasti mašinskog učenja.

Da li je moguće kreirati algoritam za rešavanje ovog problema? Složit ćete se da nam je važno da budemo u stanju da razlikujemo šta je na fotografijama, jer na taj način možemo olakšati njihovo pretraživanje i analizu. Možete pokušati da napravite listu od desetina pravila kako biste utvrdili da li postoji mačka na slici ili ne. Ne zaboravite uzeti u obzir pozadinu, osvetljenje, ugao gledanja i činjenicu da postoji preko 50 različitih vrsta mačaka.

Zapravo postoji mnogo ovakvih problema, u kojima čak i ako se potrudimo (imenujemo 1000 pravila!) nismo u stanju da napišemo algoritme koji ih precizno rešavaju. Neki drugi primeri su prevođenje sa jednog jezika na drugi i označavanje lica na fotografijama. Složit ćete se da su nam ovi zadaci važni i zato što nam omogućavaju da razumemo sadržaj napisan na jeziku koji ne govorimo ili da poboljšamo bezbednost. Zato opisujemo takve probleme **sa skupovima podataka,** parovima ulaza i očekivanim izlazima, a rešavaju se tehnikama programiranja zasnovanim na podacima. U slučaju zadatka prepoznavanja mačaka (i bilo kojih drugih objekata), odgovarajući skup podataka može se organizovati slično kao i naš, sa slikama na ulazima i vrednostima 0 ili 1 na izlazu. U slučaju zadatka prevođenja, to mogu biti parovi rečenica na oba jezika, dok u slučaju zadatka obeležavanja lica, to mogu biti parovi slika, jedan bez označenih lica kao ulaz i jedan sa označenim licima kao izlaz. Evo primera.

Input	Output
	

Skupovi podataka koje možemo koristiti za opisivanje problema mogu se kreirati kroz svakodnevne aktivnosti. Na primer, nakon pregleda pacijenta u elektronskom zdravstvenom kartonu, lekar beleži informacije o pacijentu kao što su starost, pol, simptomi, alergije na lekove (sve ove vrednosti su ulazi) i kod koji odgovara njegovoj dijagnozi (izlaz). Slično tome, na aerodromima, za svaki let, poznate su informacije kao što su vreme polaska, aviokompanija, tip aviona itd. (sve ove vrednosti su ulazi) i informacije o tome da li je taj let kasnio ili ne (izlaz).

Skupovi podataka se takođe mogu kreirati posebno za rešavanje određenog zadatka. Na primer, skup podataka koji smo koristili za identifikaciju mačaka mogao je kreirati tim volontera koji su pogledali slike koje smo imali i pratili naše smernice, na primer, ako postoji mačka na slici, napišite 1, u suprotnom napišite 0, unesite 1 ili 0 u izlaznu kolonu. Za skupove podataka domena, na primer, prepoznajući promene u rendgenskim zracima, trebalo bi angažovati medicinske stručnjake koji imaju odgovarajuće veštine i znanja za donošenje odluka. Malo kasnije, saznaćete više o tome kako se kreiraju skupovi podataka.

Verovatno se pitate: ali kako da naučimo vezu između ulaza i izlaza u skupu podataka? Baš kao što postoji oblast koja se bavi razvojem klasičnih algoritama programiranja i analizom njihovih svojstava, postoji i oblast koja se bavi razvojem algoritama zasnovanih na podacima i ispitivanjem njihovih svojstava. To se zove **mašinsko učenje**. *Mašinsko učenje* je u srži svih modernih oblasti veštačke inteligencije jer je usko povezano sa podacima i načinima izvođenja znanja iz podataka. U sledećoj lekciji, mašinsko učenje će odgovoriti na pitanje koje vas zanima.

Važno je naglasiti da postoje i druge oblasti koje se bave podacima. Među njima, najstarija je svakako **statistika**, grana matematike koja se bavi prikupljanjem podataka, opisivanjem i analizom, kao i izvlačenjem zaključaka iz podataka. Statističke tehnike su u srcu mnogih algoritama mašinskog učenja. **Data Science** je disciplina koja je nastala kao rezultat nemogućnosti pojedinih disciplina da odgovore na mnoga zanimljiva pitanja. Na primer, svaka kompanija se suočava sa pitanjem kako da poboljša svoje usluge. Da bi to uradila, kompanija može analizirati komentare korisnika na društvenim medijima ili prodajnim sajtovima. Da bi se obradili komentari, potrebno ih je prikupiti na jednom mjestu i pohraniti u bazu podataka, zatim ih organizovati, na primjer, odvojiti pozitivne i negativne komentare, a zatim analizirati svaki od ovih skupova na finiji način kako bi se utvrdilo šta korisnici ocjenjuju kao negativne ili pozitivne, na primjer, određeni model proizvoda ili neku funkcionalnost. Ove informacije treba podeliti sa menadžmentom kompanije kako bi mogli da odluče šta dalje. Odgovor na početno pitanje, primećujete, je dugačak i zahteva znanje i rad sa alatima za preuzimanje sadržaja sa veba (tzv. strugači), rad sa bazama podataka, obradu prirodnog jezika, kao i tehnike prikaza podataka koje bi bile najinformativnije za stručnjake domena. U ovom i drugim primerima primene nauke o podacima, mašinsko učenje je neophodan deo puta znanja.

Pre nego što krenemo dalje, hajde da rezimiramo kako izgleda rešavanje problema sa klasičnim programiranjem i programiranjem zasnovanim na podacima.

Kada rešimo problem koristeći klasične tehnike programiranja, na primer, pronalaženje najvećeg elementa u nizu brojeva, prvo razmišljamo o problemu i prostornim i vremenskim ograničenjima koja imamo, zatim dizajniramo algoritam za njegovo rešavanje (na primer, sortiranje spajanja), a zatim ga programiramo u programskom jeziku i sačuvamo kod. Proveravamo tačnost implementacije na nekoliko slučajnih unosa dok se ne uverimo da sve funkcioniše tačno onako kako očekujemo. Kada treba da sortiramo novi niz, možemo koristiti program koji smo napisali, pokrenuti ga i dobiti odgovarajuće rešenje.

Kada se oslanjamo na programiranje zasnovano na podacima, u početku imamo samo podatke, na primer, hiljade parova ulaza i izlaza. Opet, mudro je prvo razmisliti o problemu. To sada radimo upoznavanjem sa skupom podataka. To se postiže tehnikama istraživačke analize o kojima ćemo kasnije raspravljati, što nam može dati ideju o tome koji oblik treba tražiti rešenje. Zatim, Umesto osmišljavanja algoritma za rešavanje problema, dizajniramo **algoritam da naučimo kako da rešimo problem**. To bi značilo da ako treba da naučimo vezu između ulaza i izlaza i promenimo skup podataka, ovaj algoritam može ponovo da pronađe najbolju vezu između njih. Veza koja postoji između ulaza i izlaza zavisi od podataka (nije statična!) i zato moramo da je naučimo, a mi to ne kažemo direktno. Kada dizajniramo i programiramo ovakav algoritam, moramo da koristimo podatke da vidimo koliko dobro funkcioniše. Ako nismo zadovoljni rezultatima, moramo da napravimo korak unazad i popravimo algoritam ili se vratimo na sam početak i proverimo da li u podacima postoji još nešto što može biti važno za rešavanje problema. Za razliku od klasičnog programiranja, ova iterativnost je veoma prisutna u programiranju zasnovanom na podacima.

2.3 Osnovni koncepti mašinskog učenja

Koncepti koje ćemo predstaviti u ovoj lekciji su osnovni koncepti mašinskog učenja. Oni će vam pomoći da pratite teme o kojima se govori u nastavku, a vi ćete saznati više o svakoj od njih.

Što se tiče mašinskog učenja, prikupljanje podataka koje imamo naziva se **skup podataka**. To mogu biti neki fini tabelarni zapisi, slični onima na koje nailazimo u bazama podataka ili *Excel* datotekama, ali i neka grupa satelitskih snimaka ili audio snimaka. Jedan specifičan element skupa podataka naziva **se instanca.** Dakle, jedan određeni red u dnevniku tabeli ili jedan određeni satelitski snimak su primeri instanci. Broj instanci u skupu podataka može odrediti izbor algoritma učenja, jer neki algoritmi zahtevaju više podataka od drugih.

U slučajevima postoje **atributi**, osobine koje koristimo za opisivanje podataka. Ako zamislimo da je to tabelarni zapis o pojavama zemljotresa, datum i vreme nastanka, geografska širina, dužina, jačina zemljotresa, nivo razaranja i drugi važni podaci mogu se pojaviti kao atributi. Atributi se podjednako nazivaju **osobinama**. Malo kasnije ćete naučiti šta sve vrste atributa postoje i šta treba da se brine o. Atributi na osnovu kojih treba da naučimo kako da rešimo zadatak nazivaju se **ulazne varijable** (ulazi), a one koje treba naučiti **izlazne varijable.** Dakle, datum i vreme zemljotresa, njegove geo-koordinate i njegova magnituda mogu biti ulazne varijable u zadatku određivanja razaranja zemljotresa. Razaranje zemljotresa je takođe prisutan kao atribut u skupu podataka, tako da bi to bila izlazna varijabla. Ponekad ćemo koristiti manje formalne termine kao što su **ulaz** i **izlaz.** Važno je napomenuti da je zadatak koji diktira šta će biti ulazne varijable i šta će biti izlazne varijable.

Koji bi mogli biti atributi satelitske slike?

Rekli smo da je cilj algoritama mašinskog učenja da odredi mapiranje datih ulaza na date izlaze. Sada možemo biti precizniji i reći da je cilj mašinskog učenja da odredi mapiranja datih ulaznih varijabli na date izlazne varijable. Ova mapiranja se nazivaju modeli.

Koncept koji povezujemo sa mapiranjem je funkcija. Na času matematike čuli ste mnogo o funkcijama kao što su mapiranja ulaza-izlaz. Na primer, funkcija jedne varijable $y = 2x + 4$ mapira ulaz $x = 5$ na vrednost $y = 14$, dok funkcija višestrukih varijabli $y = 2x_1 - 3x_2 + x_3 + 5$ mapira ulaz $(x_1, x_2, x_3) = (1, -1, 3)$ na vrednost $y = 13$. Varijable koje se pojavljuju u funkcijama su povezane sa vrednostima atributa. Dakle, x u prvoj funkciji može predstavljati kvadraturu imovine, dok x_1, x_2, x_3 u drugoj funkciji može predstavljati vrijednosti atributa kao što su geografska širina, dužina i jačina zemljotresa. Na času matematike čuli ste da postoje različite klase funkcija (linearne, polinomske, trigonometrijske, eksponencijalne, logaritamske), i da svaku od njih karakterišu neke posebne osobine kao što su kontinuitet, monotonija ili konveksnost. Sve ovo znanje je dobrodošlo kada tražite pravi model.

Složenost funkcije je nešto što nećemo formalno uvesti. Shvatićete da su neke funkcije jednostavnije od drugih "kobasica". Jednostavne funkcije su korisnije za rad i lakše razumljive, ali nam ne daju mnogo slobode da opiše neke od neobičnijih odnosa između samih atributa i izlaza. S druge strane, složene funkcije su složene sa razlogom, tako da nam može biti teško da pratimo neka od njihovih matematičkih ponašanja koja mogu uticati na učenje. Pokušavamo da uspostavimo ravnotežu između složenosti i onoga što znamo o podacima i onoga što želimo da naučimo.

U modelima, kao što smo videli u uvodnom primeru cena nekretnina, **mogu** se pojaviti parametri kao što su k i n. Takvi modeli se nazivaju **parametarski modeli** i zadatak određivanja pravog modela svodi se na zadatak određivanja najboljih vrednosti parametara. U linearnom modelu, samo dva parametra pojavila u zadatku cena nekretnina, dok moderni modeli, Oni koji se zasnivaju na neuronskim mrežama imaju milione ili milijarde parametara. Videćemo da postoje i malo drugačiji **neparametarski modeli,** čiji su oblici različito izraženi.

Proces pronalaženja modela naziva **se obuka modela**. Ako u modelu postoje nepoznati parametri, moramo odrediti njihove vrednosti tokom treninga. To je naš cilj.

U skupu podataka koji se koristi za obuku modela, mogu se naći i netačne ili kontradiktorne vrednosti. Zato modeli nikada nisu apsolutno tačni. Ovo nas dovodi do još jednog važnog koncepta u teoriji mašinskog učenja: **funkcija gubitka**. Funkcija greške nam govori koliko je model pogrešan. Aktivno koristimo njegove vrednosti tokom obuke modela i težimo onim konfiguracijama modela koje nas vode do najniže vrednosti funkcije greške. U slučaju parametarskih modela, kao što je to bio slučaj u uvodnom primeru sa nekretninama, cilj je da se utvrde one vrednosti parametara za koje je vrednost funkcije greške najniža.

Kada treniramo model mašinskog učenja, moramo da procenimo koliko je on zapravo dobar za primenu u praksi. To je ono što nam služe takozvane **mere kvaliteta** - svaka od njih je prilagođena određenom zadatku učenja i domenu u kojem će se model primenjivati. Važno je naglasiti da se, generalno, funkcija greške i mjere kvaliteta razlikuju. Oba imaju za cilj da nam daju informacije o tome koliko je dobar model, funkcija greške to radi tokom obuke modela, dok mere kvaliteta to rade nakon obuke modela. Funkcija greške je usko povezana sa modelom, dok su mere kvaliteta dizajnirane tako da ih mogu razumeti i korisnici i stručnjaci domena. Ako se ne dobiju tačne vrednosti kvaliteta, model se mora popraviti. U nastavku ćemo

govoriti o tome šta to znači i kako se to može postići. Čitav proces testiranja kvaliteta modela i izračunavanja njegovih mera kvaliteta naziva se **testiranje modela**.

Tipično, vrednosti izračunate i generisane od strane obučenog modela nazivaju **se predviđanja**. Dakle, cena nove imovine ili procena razaranja zemljotresa su primeri predviđanja modela. Zbog toga govorimo o predviđanjima u svetu veštačke inteligencije. Jasno vam je da ova predviđanja nikako nisu slučajna, već veoma dobro utemeljena i zasnovana na podacima. Primena samog modela se takođe naziva **zaključivanje**.

Svi termini koji su naglašeni su važni koncepti mašinskog učenja i uvek su prisutni u literaturi o mašinskom učenju i njegovim primenama. Zato je važno da su vam jasni i da razumete kakvu ulogu igraju u razvoju modela.

2.4 Proces mašinskog učenja

Proces mašinskog učenja je sistematski pristup razvoju algoritama koji omogućavaju računarima da uče iz podataka i donose predviđanja ili odluke bez eksplicitnog programiranja. Ovaj proces obuhvata nekoliko faza, od kojih je svaka kritična za izgradnju efikasnih modela mašinskog učenja. Razumevanje ovih faza pomaže praktičarima da se kreću kroz složenost mašinskog učenja i povećava verovatnoću uspešnih ishoda.

1. Prikupljanje podataka

Prvi korak u procesu mašinskog učenja je prikupljanje podataka, što uključuje prikupljanje relevantnih podataka koji će se koristiti za obuku modela. Kvalitet i kvantitet prikupljenih podataka direktno utiču na performanse modela. Podaci se mogu dobiti iz različitih izvora, uključujući:

- **Javni skupovi podataka**: Repozitoriji kao što su Kaggle, UCI Machine Learning Repository i vladine baze podataka nude unapred prikupljene skupove podataka za različite aplikacije.
- **Veb Scraping**: Automatizovani alati mogu izvući podatke sa sajtova za prikupljanje informacija koje nisu lako dostupne u strukturiranim formatima.
- **Ankete i eksperimenti**: Prilagođeno prikupljanje podataka putem istraživanja ili kontrolisanih eksperimenata može dati specifične skupove podataka prilagođene određenom problemu.

Ishod ovog koraka je koherentan skup podataka koji predstavlja domen problema, koji će poslužiti kao osnova za naredne korake u procesu mašinskog učenja.

2. Priprema podataka

Kada se podaci prikupe, moraju biti pripremljeni za analizu. Ova pripremna faza uključuje nekoliko kritičnih zadataka:

- **Čišćenje podataka**: Sirovi podaci često sadrže greške, duplikate ili nedostajuće vrednosti. Čišćenje podataka je od suštinskog značaja kako bi se osigurala tačnost i pouzdanost. Tehnike uključuju uklanjanje duplikata, ispravljanje nedoslednosti i popunjavanje ili uklanjanje nedostajućih vrednosti.

- **Transformacija podataka**: Ovaj korak može uključivati normalizaciju ili standardizaciju podataka kako bi se osiguralo da sve funkcije podjednako doprinose performansama modela. Tipovi podataka takođe mogu zahtevati konverziju (npr. Pretvaranje kategoričkih varijabli u numeričke formate).
- **Razdvajanje podataka**: Skup podataka se obično deli na skupove za obuku i testiranje. Zajednički odnos podele je 80% za obuku i 20% za testiranje, osiguravajući da se model može proceniti na nevidljivim podacima.

Priprema podataka je često jedan od najdugotrajnijih aspekata procesa mašinskog učenja, ali je ključna za razvoj efikasnog modela.

3. Izbor funkcija i inženjering

Izbor funkcija i inženjering su ključni koraci koji određuju koji atributi podataka će se koristiti u obuci modela:

- **Izbor funkcija**: Ovo podrazumeva identifikovanje najrelevantnijih karakteristika koje doprinose predviđanju ciljne varijable. Tehnike kao što su korelacijska analiza, rekurzivna eliminacija funkcija ili korišćenje algoritama kao što je LASSO mogu pomoći u odabiru važnih karakteristika dok odbacuju nebitne.
- **Feature Engineering**: U ovoj fazi, nove funkcije mogu biti kreirane iz postojećih kako bi se poboljšale performanse modela. To može uključivati polinomske karakteristike, termine interakcije ili agregirajuće karakteristike zasnovane na znanju domena.

Efikasan izbor funkcija i inženjering mogu značajno poboljšati prediktivnu moć modela osiguravajući da se fokusira na najinformativnije aspekte podataka.

4. Izbor modela

Sa pripremljenim podacima i odabranim funkcijama, sledeći korak je odabir odgovarajućeg algoritma mašinskog učenja. Izbor algoritma zavisi od nekoliko faktora:

- **Vrsta problema**: Različiti algoritmi su pogodni za različite zadatke - klasifikacija (npr. logistička regresija, stabla odlučivanja), regresija (npr. linearna regresija), klasteriranje (npr. k-sredstva) ili učenje pojačanja.
- **Karakteristike podataka**: Priroda podataka (npr, veličina, dimenzionalnost) utiče na izbor algoritma. Na primer, modeli dubokog učenja mogu biti poželjni za velike skupove podataka sa složenim obrascima.

Odabir odgovarajućeg modela postavlja temelje za efikasnu obuku i evaluaciju.

5. Model obuke

Obuka modela uključuje hranjenje pripremljenih podataka u odabrani algoritam kako bi se omogućilo učenje obrazaca unutar skupa podataka:

- **Proces obuke**: Tokom treninga, algoritam prilagođava svoje unutrašnje parametre na osnovu ulaznih karakteristika i odgovarajućih ciljnih izlaza. Ovaj iterativni proces se nastavlja sve dok se ne ispuni kriterijum zaustavljanja (npr. određeni broj epoha ili konvergencije).

- **Metrika evaluacije**: Bitno je definisati metriku uspeha pre početka treninga. Zajedničke metrike uključuju tačnost za zadatke klasifikacije, srednju kvadratnu grešku za regresijske zadatke i F1-rezultat za neuravnotežene skupove podataka.

Obuka ima za cilj da razvije model koji dobro generalizuje nove podatke, a ne samo pamćenje skupa obuke.

6. Evaluacija modela

Nakon treninga, procena performansi modela na neviđenim podacima testa je od ključnog značaja:

- **Testiranje**: Model se procenjuje korišćenjem testnog skupa podataka koji nije bio uključen u obuku. Ova evaluacija pruža uvid u to koliko se model generalizuje na nove instance.
- **Metrike performansi**: U zavisnosti od vrste problema, mogu se koristiti različite metrike:
  - Za zadatke klasifikacije: tačnost, preciznost, opoziv, F1-rezultat.
  - Za regresijske zadatke: R-kvadrat, srednja apsolutna greška (MAE), srednja kvadratna greška (MSE).

Ova evaluacija pomaže da se identifikuju oblasti u kojima se model ističe ili treba poboljšati.

KSNUMKS. Podešavanje hiperparametara

Podešavanje hiperparametara podrazumeva optimizaciju parametara koji upravljaju procesom obuke, ali se ne uče direktno iz podataka:

- **Mrežna pretraga i slučajna pretraga**: Ove tehnike sistematski istražuju kombinacije hiperparametara kako bi pronašle optimalne postavke koje poboljšavaju performanse modela.
- **Unakrsna validacija**: Implementacija unakrsne validacije tokom podešavanja hiperparametara pomaže da se osigura da su poboljšanja performansi konzistentna u različitim podskupovima podataka.

Fino podešavanje hiperparametara može dovesti do značajnih poboljšanja u tačnosti i robusnosti modela.

8. Raspoređivanje

Kada je zadovoljavajući model obučen i procenjen, može se rasporediti u proizvodnju:

- **Integracija**: Konačni model treba integrisati u postojeće sisteme gde će pružiti predviđanja ili uvide na osnovu podataka u realnom vremenu.
- **Praćenje**: Kontinuirano praćenje nakon primene je od suštinskog značaja kako bi se osiguralo da model održava svoje performanse tokom vremena kako novi podaci postaju dostupni.

Primena označava prelazak sa razvoja na praktičnu primenu, naglašavajući korisnost u stvarnom svetu.

Proces mašinskog učenja obuhvata niz strukturiranih koraka koji vode praktičare od početnog prikupljanja podataka kroz primenu prediktivnih modela. Svaka faza - prikupljanje podataka, priprema, izbor funkcija, izbor modela i obuka, evaluacija, podešavanje hiperparametara i primena - igra vitalnu ulogu u razvoju efikasnih rešenja za mašinsko učenje. Temeljnim razumevanjem ovog procesa, praktičari mogu poboljšati svoju sposobnost stvaranja robusnih modela koji pružaju dragocene uvide u različitim aplikacijama u današnjem svetu zasnovanom na podacima.

2.5 Vrste mašinskog učenja

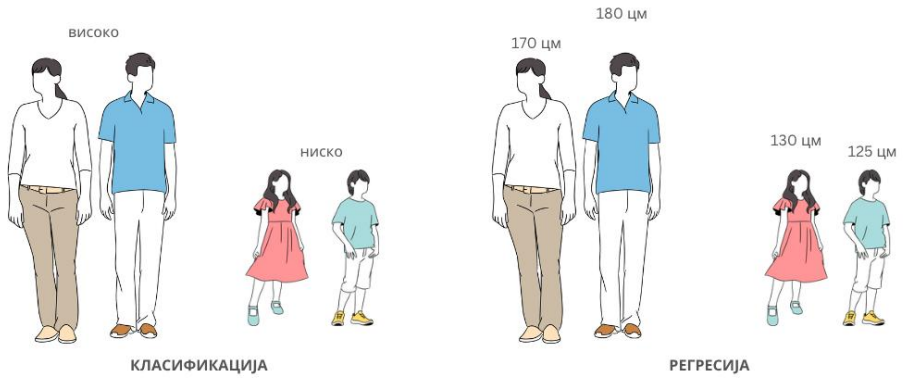
U ovoj lekciji ćemo predstaviti osnovne vrste mašinskog učenja: nadgledano mašinsko učenje, mašinsko učenje bez nadzora i učenje pojačanja. Svi oni su skriveni iza zadataka kao što su predviđanje padavina, preporučuje filmove za gledanje ili igranje igara. Svaka od ovih oblasti će biti detaljnije razmotrena kasnije u toku kursa. Takođe ćemo govoriti o nekim aktuelnim oblastima istraživanja, kao što je učenje kroz prenos znanja.

U ovoj lekciji ćemo predstaviti osnovne vrste mašinskog učenja: nadgledano mašinsko učenje, mašinsko učenje bez nadzora i učenje pojačanja. Svi oni su skriveni iza zadataka kao što su predviđanje padavina, preporučuje filmove za gledanje ili igranje igara. Svaka od ovih oblasti će biti detaljnije razmotrena kasnije u toku kursa. Takođe ćemo govoriti o nekim aktuelnim oblastima istraživanja, kao što je učenje kroz prenos znanja.

Nadgledano mašinsko učenje

Većina primera koje smo do sada videli su zapravo primeri **nadgledanog mašinskog učenja.** Nadgledano mašinsko učenje uključuje algoritme koji se savršeno uklapaju u ono o čemu smo do sada govorili i pomažu nam da naučimo kako da mapiramo jedan skup vrednosti na drugi. Zbog toga je neophodno da u skupu podataka na koji se primenjuju, pored vrednosti atributa, znamo i vrednosti ciljne varijable.

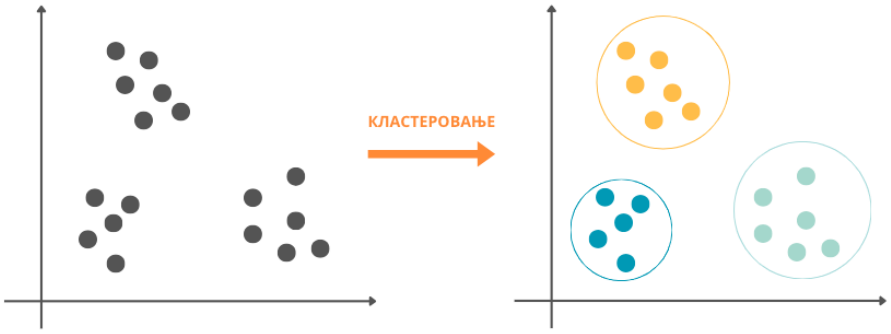
Dva glavna zadatka nadgledanog mašinskog učenja su **regresija** i klasifikacija. U oba regresijskih i klasifikacijskih zadataka, želimo da naučimo da predviđamo vrednosti, ali u regresijskim zadacima, vrednosti mogu biti proizvoljne, a u slučaju klasifikacije, one mogu biti iz unapred definisanog konačnog skupa vrednosti. Dakle, regresijski zadaci su pogodni za predviđanje vrednosti temperature, cene proizvoda (zadatak određivanja cene nekretnine sa kojom smo se susreli u uvodnom delu je primer regresijskog zadatka), razaranja zemljotresa i slično. S druge strane, utvrđivanje da li je pošta nepoželjna ili poželjna ili određivanje žanra filma su klasifikujući zadaci jer je skup vrednosti koje imamo na drugoj strani konačan - pošta može biti ili poželjna ili nepoželjna (dve vrednosti), dok žanr može biti, recimo, komedija, drama, akcija ili triler (četiri vrednosti). Nešto kasnije, predstavićemo precizniju definiciju svakog od ovih zadataka.



Klasifikacija i regresija. Utvrđivanje da li je neko visok ili nizak je zadatak klasifikacije. Određivanje tačne visine je zadatak regresije.

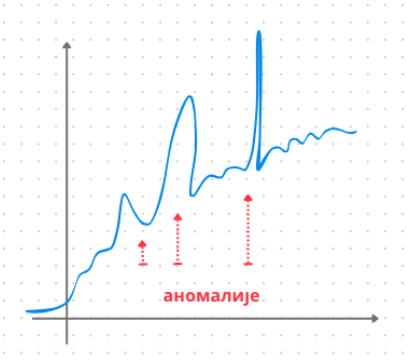
Mašinsko učenje bez nadzora

Koristimo mašinsko učenje bez nadzora u zadacima koji treba da ispitaju strukturu skupa podataka. Na primer, ako analiziramo kupovinu potrošača jedne prodavnice, može biti zanimljivo primetiti proizvode koji se često kupuju zajedno kako bi ih fino distribuirali u prodavnici, poboljšali ponudu, ali i profit. Na isti način, komentari korisnika mogu se analizirati i grupisati i usluge ili funkcije o kojima korisnici govore mogu se uočiti. Zadaci ovog tipa, u kojima želimo da vidimo grupe među podacima, nazivaju se **klasteriranje**. Kasnije u toku kursa, naučićete o algoritmu k-sredine, najpoznatijem algoritmu klasteriranja.



Clustering

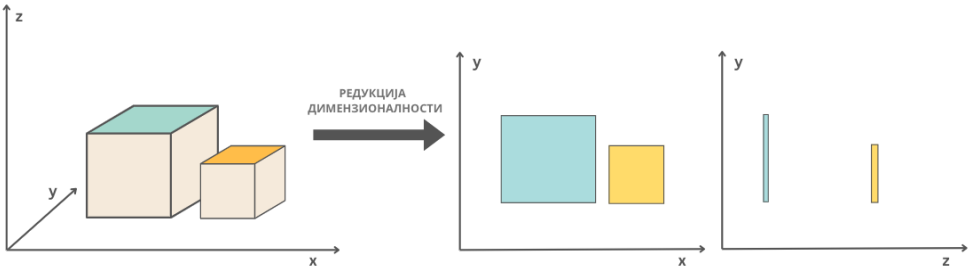
Uočavanje slučajeva podataka koji su na neki način drugačiji od drugih takođe spada u zadatke mašinskog učenja bez nadzora. Tako, uočavanje atipičnih merenja senzora fabrike može biti signal za pokretanje dodatnih bezbednosnih procedura. Slično tome, uočavanje atipičnih bankarskih transakcija, na primer, sa udaljene lokacije ili u nekom neobičnom iznosu, može biti nagoveštaji prevare. Ovaj zadatak mašinskog učenja bez nadzora naziva **se detekcija anomalija**.



Otkrivanje anomalija

Mašinsko učenje bez nadzora takođe se bavi zadacima **smanjenja dimenzionalnosti**. Često, za potrebe grafičkog prikaza podataka, moramo da pređemo sa većeg broja atributa na manji broj atributa, na primer, dva ili tri. Jasno je da se tokom ove transformacije gube neke informacije iz početnog skupa podataka, ali, s druge strane, dobija se mogućnost prikazivanja podataka i možda bolji uvid u neke zakonitosti. Manja

dimenzionalnost podataka (manje atributa) je poželjna i zbog bržeg izvršavanja algoritama i manje složenosti memorije, što može biti posebno važno ako imamo ograničene resurse za rad. Neki od najčešće korišćenih algoritama za smanjenje dimenzionalnosti su glavna analiza komponenti. *analiza glavnih komponenti (PCA)* i t-SNE.

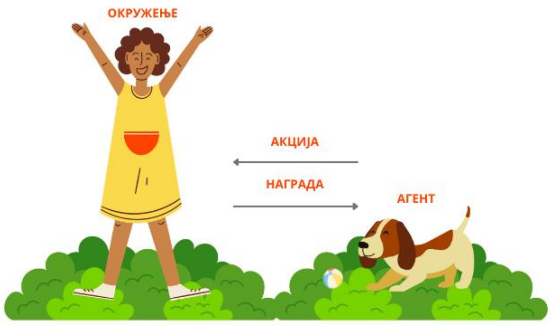


Značenje smanjenja dimenzionalnosti: dva kvadra i njihove projekcije iz trodimenzionalnog u dvodimenzionalni prostor

Zanimljivo je da u zadacima mašinskog učenja bez nadzora nije potrebno znati vrednosti ciljne varijable. Klasteriranje, otkrivanje anomalija i smanjenje dimenzionalnosti vrše se samo na osnovu vrednosti atributa.

Učenje pojačanja

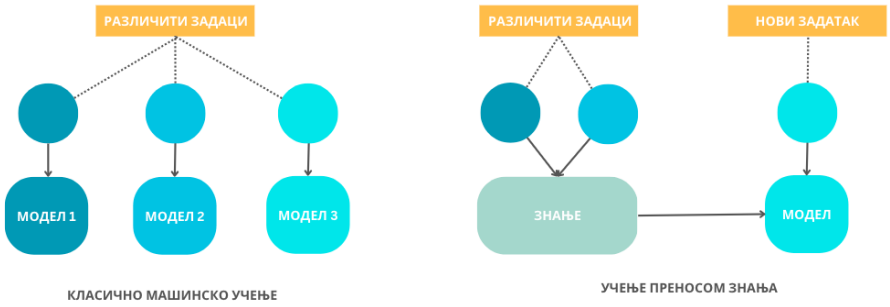
Sigurno ste mnogo puta videli kako trenirati psa. Kada mu se da zadatak, na primer, da donese loptu sa drugog kraja dvorišta, nagrada u obliku kolačića kada je donese motivisaće psa da sledeći put izvrši ovaj zadatak još uspešnije i radosnije. Ova ideja takođe leži u osnovi učenja pojačanjem. **Učenje pojačanja** je oblast mašinskog učenja koja se koristi u zadacima kao što su igranje igara ili autonomna vožnja. Karakteriše ga postojanje okruženja koje ima svoja stanja, agenta koji može da izvrši određeni skup akcija i koncept nagrade. Cilj je da agent u datom okruženju, čija se stanja menjaju, izabere (nauči) redosled akcija koji mu omogućava najveću nagradu. U kontekstu uvodnog primera, dvorište je okruženje. Njegova stanja mogu biti lopta na kraju dvorišta ili komšijska mačka na drvetu. Pas je agent, a skup akcija koje može da preduzme je da trči, da sedi, da ode na spavanje. Nagrada može biti broj kolačića ili ništa. Ako pas izabere pravi redosled akcija (trči, pronađe ga i vrati) na promenu okruženja, na primer, izgled lopte, on će moći da osvoji najveću nagradu.



Više o ovoj vrsti učenja saznaćete na kraju kursa.

Novi pravci učenja

Kada treba da savladamo novi zadatak, na primer, da naučimo da vozimo skuter, ne počinjemo od nule. Sva znanja i veštine koje smo stekli u nekim drugim zadacima, na primer, igranje košarke, vožnja biciklom, pa čak i upornost i strpljenje u zadacima koji nam nisu bili omiljeni, kao što je sređivanje podruma, pomažu nam da ga bolje savladamo. Ova ideja je osnova transfernog **učenja**. Zato često možete čuti ljude kako govore o modelima koji su korišćeni kao osnova za razvoj nekog drugog modela. Takvi modeli se prvo obučavaju na nekim opštim skupovima podataka i zadacima, a zatim se prekvalifikuju, tj. Mogu se koristiti i za rešavanje vrlo specifičnog zadatka. Na primer, GPT jezički model korišćen je kao osnova za razvoj *ChatGPT modela*, koji se ranije dobro pokazao u zadacima generisanja rezimea, skraćenih verzija teksta i odgovaranja na pitanja.



Ideja učenja kroz prenos znanja

Tehnike prenosa znanja mogu se kombinovati sa svim gore navedenim vrstama učenja. Oni su posebno važni za nas kada skupovi podataka za obuku za određeni zadatak nisu dovoljno veliki ili kada razvijamo model za određeni domen.

2.6 Podaci u mašinskom učenju

U AI zajednici često čujete dve izreke: "podaci su novo zlato" i "smeće unutra, smeće napolje". Oni nas podsećaju na vrednost podataka za razumevanje i modeliranje fenomena i važnost stvaranja visokokvalitetnih skupova podataka. Hajde da istražimo ove teme.

U ovoj lekciji ćemo predstaviti osnovne vrste mašinskog učenja: nadgledano mašinsko učenje, mašinsko učenje bez naDanas skoro svi domeni aktivnosti generišu velike količine podataka: informacije o video zapisima koje smo gledali na mreži, proizvodima koje smo kupili, prijateljima sa kojima smo se povezali na društvenim mrežama, kao i informacije o posetama lekaru, vremenskim uslovima u našem gradu ili saobraćajnim uslovima koje su zabeležile relevantne institucije. Svi ovi podaci mogu se koristiti za bolje razumevanje okruženja u kojem se generiše.

Baš kao u priči o bazama podataka sa kojima ste se susreli prošle godine, u mašinskom učenju opisujemo važne entitete i događaje čije ponašanje želimo da modeliramo pomoću atributa (koji se nazivaju i funkcije).

Na primer, film se može opisati naslovom, žanrom, godinom izdanja, produkcijskom kućom, budžetom, profitom, sinopsisom, imenom reditelja i imenima glavnih glumaca. Odabir pravih atributa za praćenje i snimanje prilikom prikupljanja podataka nije lak zadatak jer ne znamo unapred koji će atributi biti najkorisniji za zadatak koji želimo da rešimo u budućnosti. Na primer, ako želimo da koristimo podatke za predviđanje profita filma (regresijski zadatak), informacije o glumcima i produkcijskoj kući mogu biti korisnije, dok za određivanje žanra filma (zadatak klasifikacije), sinopsis može biti korisniji. U složenijim domenima, ovi izbori dolaze sa još više dilema i izazova.

Zbog potrebe da se koriste podaci za širok spektar aplikacija, mogli bismo razmotriti prikupljanje što više vrednosti atributa moguće. Iako ova ideja važi u nekim situacijama, generalno, moramo imati na umu da velike količine podataka zahtevaju odgovarajuće skladištenje, hardver koji podržava njihovu obradu i tim stručnjaka sa potrebnim veštinama i znanjem za obavljanje ovih zadataka. Stoga, takvi izbori mogu biti skupi i zahtevaju posebno planiranje. Takođe je važno napomenuti da je analiza i razumevanje velikih količina podataka izazov i zahteva odgovarajuće tehničke kompetencije, kao što su tehnike vizuelizacije podataka. Pored toga, mnogi domeni koji uključuju privatne i osetljive podatke moraju dosledno pratiti propise i etičke smernice o prikupljanju podataka, što nameće dodatna ograničenja na izbor atributa i mogućnosti skladištenja. Dakle, zadatak prikupljanja podataka i stvaranja visokokvalitetnih skupova podataka je izazovan i zahtevan, što zahteva pažljivu organizaciju.

U narednim lekcijama videćemo da je svaki atribut definisan svojim tipom i skupom vrednosti, a ove osobine utiču na to kako pripremamo podatke. Na kraju, algoritmi mašinskog učenja mogu se primeniti samo na numeričke vrednosti. Broj atributa i njihove osobine takođe utiču na izbor algoritma mašinskog učenja.

Napredni algoritmi mašinskog učenja, kao što su neuronske mreže, mogu sami da identifikuju važne atribute za rešavanje zadatka. Ovo nas oslobađa od razmišljanja o izboru atributa i kombinacijama. Ovo je posebno korisno kada se radi sa složenim podacima kao što su slike ili tekstualni sadržaj, gde definisanje i izdvajanje atributa nije uvek intuitivno. Ovi algoritmi mogu raditi sa sirovim podacima.

- Šta smatrate izazovnim u prikupljanju podataka u domenu koji vas zanima? To može biti sport, naučna disciplina, društveni fenomen ili bilo šta drugo.
- Da li imate bilo kakvih nedoumica ili rezervi u vezi sa prikupljanjem i obradom podataka?
- Šta je vama lično najvažnije u procesu prikupljanja podataka?

Popularni skupovi podataka

Možda će vas iznenaditi, ali skupovi podataka mogu biti popularni previše! Neki od njih su poznati po tome što se koriste u prvim zadacima mašinskog učenja, dok su neki postigli svoju popularnost kroz uporne angažmane zajednice kako bi ih proširili i dopunili. Kako različiti skupovi podataka prate različite domene AI, ovde ćemo ovo koristiti kao kriterijum za grupisanje i prikazivanje. Naime, upoznaćemo setove koji sadrže slike, tekstualne podatke, audio arhive i video zapise. Veliki broj biblioteka koje se koriste u oblasti mašinskog učenja omogućavaju brzo i jednostavno učitavanje skupova o kojima ćemo razgovarati.

Računarski vid

MNIST

Svakako jedan od najpopularnijih setova u oblasti kompjuterskog vida je **MNIST**, skup slika rukom pisanih brojeva. Njegov razvoj je započeo američki Nacionalni institut za standarde i tehnologiju (eng. *Nacionalni institut za standarde i tehnologiju (NIST)* još 1998. godine. Sve slike su 28k28 piksela, crno-bele, a ima ih ukupno 70.000: 60.000 slika čine set za obuku i 10.000 slika čine test set. Na slici možete videti neke od cifara iz ovog skupa podataka.

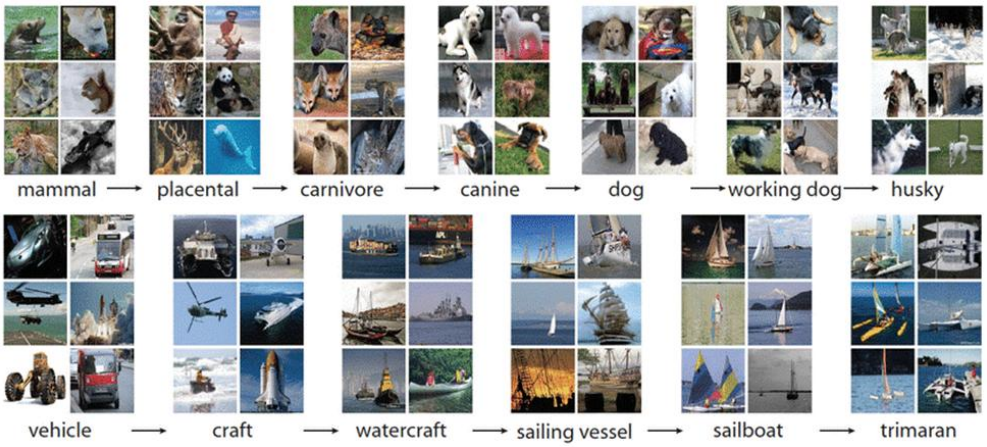
Neke od figura MNIST konferencije

MNIST set se koristi za obuku multiklasnih klasifikatora, najčešće u kombinaciji sa konvolucijskim neuronskim mrežama, o čemu ćete čuti više kasnije u toku kursa.

Za svaku cifru MNIST skupa obezbeđena je jedna klasa. Razmislite o tome koje su cifre potencijalno problematične za razlikovanje (na primer, cifre 1 i 7 mogu ličiti jedna na drugu), a zatim pokušajte da pronađete neke primere na vebu.

ImageNet

Slike u **ImageNet** setu predstavljaju slike opštih objekata: računara, prozora, aviona, sadnica, tropskih životinja i raznih drugih entiteta. Zanimljivo je da su ove slike organizovane u srodne grupe (tzv. Synsets) između kojih se primenjuje odnos roditelj-dete. Na primer, svi jedrenjaci pripadaju jednoj grupi (jedan sinset), u hijerarhiji ispod njih postoje grupe jedrilica i trimarana, dok u hijerarhiji iznad postoje grupe plovila, plovila i vozila. Na slici, u donjem redu, možete pratiti ovu hijerarhiju: na dnu su trimarani, a na vrhu vozila. U gornjem redu su sinseti koji se odnose na pse i neke od njihovih kategorizacija.



Primer ImageNet slike

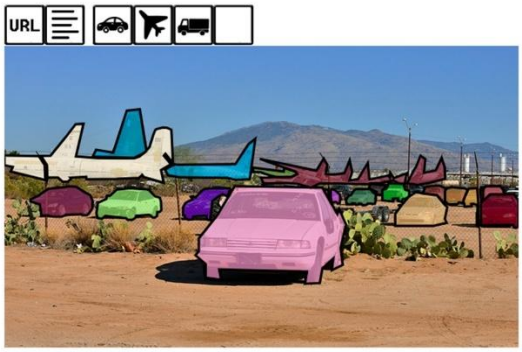
Zbirka trenutno sadrži oko 14 miliona slika i preko 21.000 sinseta. Koristi se u različitim zadacima klasifikacije slika i otkrivanja objekata na slikama.

Zvanični sajt ImageNet konferencije je https://www.image-net.org/index.php . Istraživači sa univerziteta Stanford i Princeton aktivno rade na njegovom razvoju.

Pokušajte da saznate kojoj grupi računar pripada u ImageNet skupu i koje grupe su u hijerarhiji ispod i iznad

KOKO

Skup podataka **COCO** (akronim za *Common Objects in Context)* koristi se u zadacima otkrivanja objekata, segmentacije slika i automatskog povezivanja naslova sa slikama. Kreirao ga je Microsoft i podelio sa zajednicom 2015. godine.



Slika COCO seta sa označenim prepoznatim objektima: avionima, kamionima i automobilima

Set se može interaktivno pregledati na zvaničnom sajtu: za svaku sliku postoji URL sa kojeg je slika snimljena, nekoliko naslova povezanih sa slikom, a zatim niz ikona koje odgovaraju prepoznatim objektima. Broj slika u skupu podataka je 330.000 i sadrži 80 kategorija objekata sa preko 1,5 miliona instanci. Link do odeljka za pretragu na sajtu je https://cocodataset.org/#explore.

Obrada prirodnog jezika

IMDB

Ako volite da gledate filmove i TV emisije, bićete zainteresovani **za IMDB** skup podataka, koji sadrži recenzije korisnika sa popularne IMDB platforme. Za svaki pogled u ovom skupu podataka, takođe je poznato da li je pozitivan ili negativan, tj. da li prvenstveno sadrži nešto pohvalno i dobro o filmu ili neku kritiku i prigovor. Kada je reč o skupovima podataka koji sadrže tekstualni sadržaj, Uvek je važno naglasiti na kom jeziku su napisani. IMDB skup podataka sadrži prikaze koji su na engleskom jeziku sa ukupno 50.000 pregleda, 25.000 pozitivnih i 25.000 negativnih pregleda. Ispod možete videti pozitivan i negativan unos u ovom skupu podataka.

Преглед	Сентимент (0 - негативан, 1 - позитиван)
Don't even ask me why I watched this! The only excuse I can come up with that I was sick with Bronchitis and too weak to change the channel. :) It's too terrible for words, the movie that is, not the Bronchitis.	0
this movie is the best movie ever it has a lot of live action It's just great everyone should watch it and the actor are great the location is Rome Italy thats the best place ever the actors are great	1

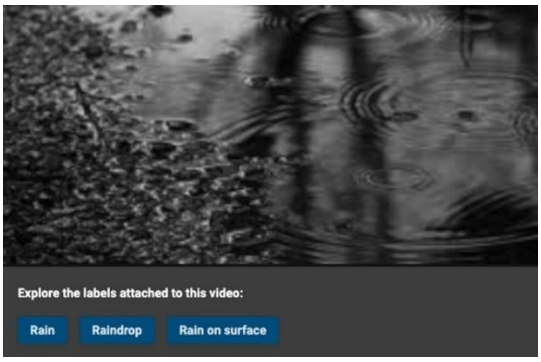
Primeri pozitivnih i negativnih kritika IMDB seta

IMDB skup podataka se koristi u zadacima analize osećanja - podsetimo se da su to zadaci u kojima je potrebno prepoznati emociju ili stav prisutan u tekstu. Pošto skup sadrži samo informacije da li je pregled pozitivan ili negativan, zadatku analize raspoloženja u IMDB skupu se pristupa kao problemu binarne klasifikacije. Generalno, skala osećanja može biti finija i uključuje ocene kao što su veoma pozitivan, pozitivan, neutralan, negativan ili veoma negativan.

Obrada zvuka

Pretraživanje

AudioSet je skup podataka koji sadrži 10-sekundne isječke video zapisa sa IouTube-a. Svaki od ovih isečaka povezan je sa karakteristikama zvukova koji se čuju u njima. Set je kreirao Google i sadrži preko 2 miliona klipova sa ukupnim trajanjem od 5,8 hiljada sati.



Primer video klipa sa pripadajućim audio napomenama koje sadrži

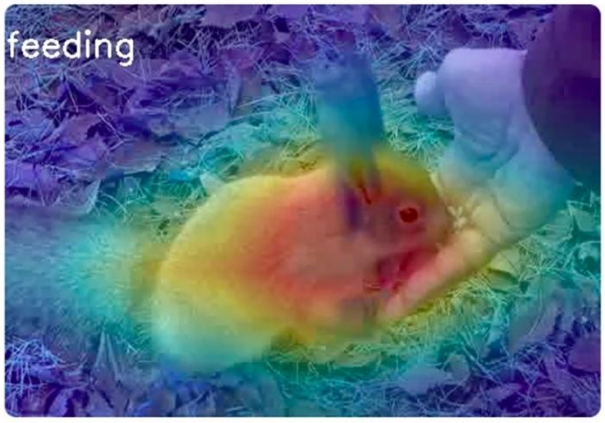
Zvanični sajt konferencije pruža pregled primera i uvid u organizaciju konferencije. Korišćene su 632 različite kategorije, kao što su zvuci muzičkih instrumenata, zvuk vetra, zvuk čoveka, buka itd. Možete posetiti adresu https://research.google.com/audioset/index.html i poslušati još neke primere. Sama konferencija je stvorena sa idejom da podrži razvoj algoritama za prepoznavanje zvuka.

Obrada video zapisa

Trenuci u vremenu

Moments in Time je skup podataka koji se razvija sa idejom da pomogne sistemima veštačke inteligencije da nauče da prepoznaju akcije i događaje. Ovaj set trenutno sadrži milion video zapisa dužine 3 sekunde u kojima su aktivnosti označene. Video snimci sadrže ljude, životinje, predmete i prirodne fenomene. Samo neki od događaja koji su pokriveni su ples, vežbanje, penjanje na drvo, skakanje u vodu i spavanje.

Okupljanje Moments in Time razvija tim sa Masačusetskog instituta za tehnologiju (MIT), a na zvaničnom sajtu projekta možete videti još nekoliko primera video zapisa i priznatih akcija. Link ka zvaničnom sajtu je http://moments.csail.mit.edu/.



Video u kojem se prepoznaje da čovek hrani zeca

2.7 Eksplorativna analiza skupa podataka (EDA)

Susret sa novim skupom podataka je kao putovanje na novo mesto. Morate ga pažljivo istražiti, saznati gde je sve i kakve veze postoje između različitih delova. U ovom odeljku ćete naučiti nekoliko tehnika koje će vam pomoći u našoj avanturi sa podacima.

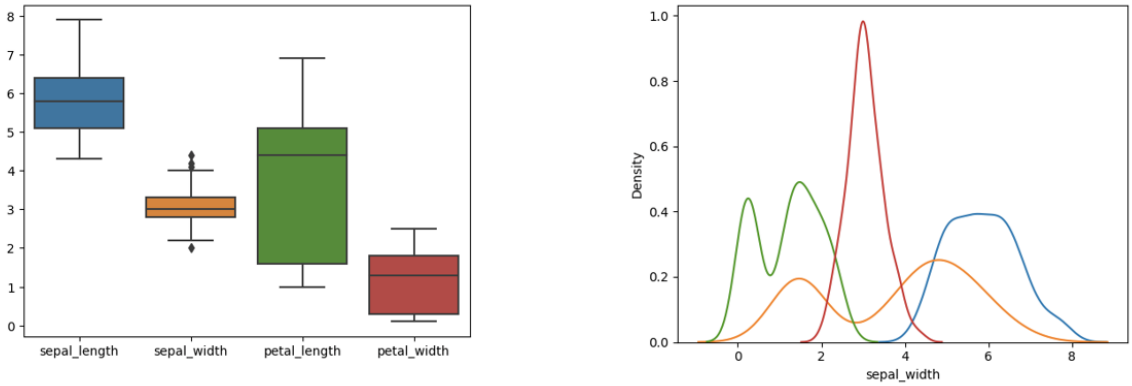
Svaki zadatak mašinskog učenja započinjemo upoznavanjem sa skupom podataka. Ako koristimo tabelarne podatke, zanima nas koji se atributi pojavljuju, koje vrednosti imaju i da li neki od njih mogu biti povezani. Kada radimo sa drugim vrstama podataka, kao što je tekst, obično nas zanima da li su svi tekstovi napisani na istom jeziku i koliko su dugi. Pošto nijedan skup podataka nije savršen, u analizi pokušavamo da pronađemo potencijalne duplikate i neke atipične unose. Svi ovi zadaci se nazivaju **istraživačka analiza**

skupa podataka. *Istraživačka analiza podataka* (EDA). Njegov cilj je da nam pomogne, kroz raznovrstan skup zadataka, da bolje upoznamo skup podataka i da donosimo dalje odluke u vezi sa pripremom podataka više informisani. S obzirom na važnost podataka u narednim koracima (zapamtite izreku "smeće na ulazu, smeće na izlazu"), pokušavamo da posvetimo dovoljno vremena istraživačkoj analizi skupa podataka i pređemo na sledeći korak samo kada smo sigurni da razumemo podatke.

Istraživački zadaci analize podataka

Analiza atributa

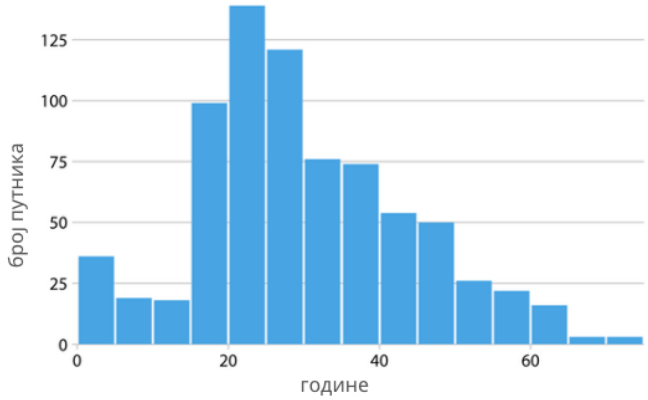
Pošto se atributi koriste za opisivanje širokog spektra svojstava, njihovi tipovi i opseg vrednosti variraju. Dve velike grupe atributa sa kojima se susrećemo su **numerički** (kvantitativni) i **kategorički** (kvalitativni) atributi. Numerički atributi imaju, kao što ime kaže, numeričke vrednosti. Takvi su, na primer, visina igrača, udaljenost od aerodroma, broj kućnih ljubimaca, spoljašnja temperatura, broj prodatih sladoleda, koncentracija glukoze u krvi i mnogi drugi. Za ove atribute, obično gledamo opsege vrednosti, najviše i najniže vrednosti, prosečnu vrednost, medijan, kao i samu distribuciju. Sve ove **analize nazivamo deskriptivnim analizama,** jer nam pomažu da opiše količinu na koju je atribut povezan.



Primeri nekih opisnih analiza atributa Iris skupa podataka

Kategorički atributi su vrsta atributa koji mogu imati konačan skup vrednosti. Takvi atributi su, na primer, boja automobila, vrsta odeće, pol pacijenta, tekuća sezona i drugi. Ovi atributi su obično predstavljeni nizovima ili ekvivalentnim numeričkim kodovima. Na primer, mesec u godini može biti naveden kao naziv "februar" ili kao broj "dva" (jer je februar drugi mesec u godini). Važno je napomenuti da čak i ako koristimo

numeričke kodove za predstavljanje ovih atributa, nema smisla izračunavati vrednosti kao što su prosek ili maksimum, jer ove vrednosti nisu inherentno numeričke. Za njih obično analiziramo koje vrednosti mogu uzeti i koliko često se pojavljuju, a te zaključke prikazujemo pomoću grafikona.



Primer analize atributa "godina" u setu Titanik

Ujedinjenje vrednosti

U toku analize podataka, možemo ustanoviti da vrednosti atributa nisu ravnomerno postavljene. Na primer, imena boja mogu biti napisana nedosledno, ponekad malim, a ponekad velikim slovima, ili datumi mogu biti dati u različitim formatima kao što su dan-mesec-godina i godina/mesec/dan. Da bi mogli pravilno izvršiti zadatak analize, poželjno je ujediniti ove vrijednosti, tj. Hajde da ih smanjimo na isti način predstavljanja. Obično postoji način koji je poželjniji ili korisniji, ali se takođe dešava da su izbori potpuno jednaki.

Nedostajuće vrednosti

Kada analiziramo skup podataka, možemo primetiti da nedostaju vrednosti nekih atributa. To može biti zbog nepažnje u unosu podataka ili jednostavno nedostupnosti informacija. Takve vrednosti u skupu podataka nazivaju nedostajuće vrednosti.

име и презиме	године	радно искуство	приходи
Марко	48	22	83
Иван	67	30	110
Ана	34	10	
Сара		4	
Милан	21		85

Primer skupa sa nedostajućim vrednostima

Najjednostavniji korak koji možemo preduzeti kada primetimo nedostajuće vrednosti je da izbrišemo ili atribute (kolone skupa podataka) ili instance (vrste skupa podataka) u kojima se pojavljuju. Na primer, ako ne znamo vrednost atributa za više od 50% instanci skupa podataka, ima smisla da ga izbrišemo. Ako, s druge strane, imamo samo nekoliko slučajeva u kojima nedostaje vrednost atributa, najbolje je da obrišete instance i zadržimo atribut. Međutim, ove odluke nisu uvek lake. Na primer, može se desiti da različite vrednosti atributa nedostaju u različitim slučajevima, tako da na ovaj način brišemo i ignorišemo značajan broj instanci, što može biti problematično ako nemamo veliki skup podataka. Zato ima smisla razmotriti još neke opcije u radu sa nedostajućim vrednostima.

Ako je atribut koji nedostaje numerički, na primer, udaljenost do aerodroma ili visina igrača, možemo zameniti vrednosti koje nedostaju sa prosečnom vrednošću poznatih vrednosti. Argument koji imamo za ovaj izbor je da ćemo koristiti informacije koje već postoje u skupu podataka i da nećemo mnogo promeniti o nekim drugim svojstvima atributa. S druge strane, ako govorimo o kategoričkim atributima, kao što su boja automobila ili zemlja proizvodnje, koji mogu imati konačan skup vrijednosti, možemo zamijeniti nedostajuću vrijednost najčešćom vrijednošću. Druga opcija koja važi i za numeričke i za kategoričke atribute je upotreba slučajnih vrednosti - tako da možemo zameniti nedostajuću boju slučajnom bojom iz mogućeg skupa boja, a nedostajuću visinu igrača sa vrednošću iz opsega najmanje i najviše visine u setu. U svim slučajevima, moramo biti oprezni jer promene u podacima mogu uticati na uspeh modela i rezultate koje dobijemo. Takođe je veoma važno u kom trenutku vršimo ove popravke. O tome ćemo pričati kasnije.

Duplikate

Prisustvo duplikata u skupu podataka može uticati na moć generalizacije modela. Zato je uvek zgodno proveriti da li postoje podaci koji se ponavljaju ili su veoma slični. Kada su u pitanju tabelarni podaci, duplikati se mogu naći direktnim upoređivanjem vrednosti atributa. Kada radimo sa različitim vrstama podataka, obično su nam potrebne naprednije tehnike. Na primer, duplirane slike mogu biti simetrične, kao u ogledalu, bilo horizontalno ili vertikalno. Isto je i sa tekstualnim podacima. Dve vesti mogu sadržavati istu objavu (koju su izveštale neke novinske agencije) sa malo drugačijim naslovima, tako da su u smislu direktnog poređenja karaktera, različite, ali iste.

Uočavanje izuzetaka

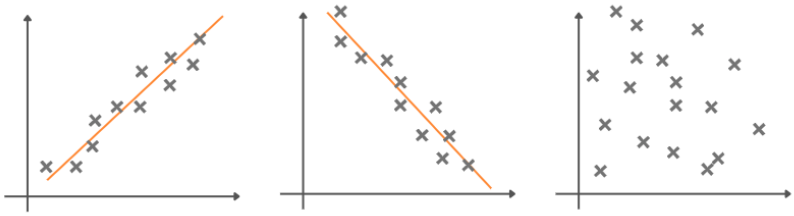
Uočavanje podataka koji se na neki način razlikuju od ostalih omogućava nam da uočimo greške u podacima ili otkrijemo nova, atipična ponašanja. Takvi podaci se nazivaju izuzeci ili *izuzetci*. Udaljenost od aerodroma, koja je -1,2 km, bila bi razlika u brojci jer očekujemo da će udaljenost biti pozitivna vrednost. Na taj način smo mogli uočiti grešku i ispraviti je. S druge strane, temperatura od 45 °C je takođe neobična vrednost, ali stvarna zbog klimatskih promena i možda veoma korisna kao informacija za preduzimanje određenih koraka i akcija.

Grafički prikaz neslaganja

Odstupanja takođe mogu uticati na ishod algoritama mašinskog učenja. Zbog toga, kada su uočeni i obrađeni, važno je odlučiti da li ih treba zadržati ili izbrisati.

Atribut korelacija

Atributi mogu biti povezani jedni sa drugima. Možemo videti vezu ako nacrtamo graf koji ima vrednost jednog atributa duž k-ose i vrednost drugog atributa duž y-ose. Na primer, možemo pratiti parove atributa spoljnu temperaturu i broj prodatih sladoleda, spoljnu temperaturu i potrošnju električne energije i spoljašnju temperaturu i broj knjiga u biblioteci. Neka svaki od ovih parova odgovara grafikonu kao na slici ispod. Možemo primetiti da je porast temperature praćen povećanjem broja prodatih sladoleda. Ako povećanje vrednosti jednog atributa prati povećanje vrednosti drugog atributa, kažemo da su oni **pozitivno povezani**. Na grafu takođe možemo primetiti da je ova zavisnost linearna, tj. da prati imaginarnu liniju koja prolazi kroz skup tačaka. S druge strane, čini se da je situacija sa spoljašnjom temperaturom i potrošnjom struje nešto drugačija, tj. da je smanjenje temperature praćeno većom potrošnjom električne energije, Verovatno zbog upotrebe grejača. Atributi u kojima je povećanje vrednosti jednog atributa praćeno smanjenjem vrednosti drugog atributa se kaže da su **negativno korelirani.** Iz grafike, možemo zaključiti, opet, da je ova vrsta korelacije linearna. Treći grafikon, koji prikazuje spoljašnju temperaturu i broj knjiga u biblioteci, ne ukazuje na nikakvu, barem ne očiglednu, pravilnost između atributa. Svakako možemo zaključiti da ovi atributi nisu linearno povezani.



Grafika asocijacije atributa

Da bi se izmerio linearni odnos atributa, možemo koristiti različite vrste koeficijenata koji su uspostavljeni u domenu matematičke statistike. Jedan takav koeficijent je Pirsonov koeficijent korelacije. Njegove vrednosti se kreću od -1 do 1 i ukazuju na pravac i snagu veze. Vrednosti koeficijenata bliže -1 ukazuju na negativnu korelaciju, vrednosti koeficijenata bliže 1 ukazuju na pozitivnu korelaciju, a vrednosti oko nule ukazuju na odsustvo linearne korelacije.

Uobičajeno je da se vrednosti koeficijenata korelacije između atributa grafički prikazuju u obliku takozvane toplotne mape. Svaki kvadrat na ovoj mapi odgovara jednom paru atributa i njegova boja je prilagođena vrednosti koeficijenta korelacije. Kolona koja se nalazi na strani ove mape povezuje vrednosti i nijanse boja. Posmatrajući ovu mapu, lako možemo videti korelacije u podacima. Na slici ispod prikazani su parovi atributa jednog skupa podataka koji kombinuje informacije o zaposlenima. Iako malo znamo o ovom setu, možemo zaključiti da iskustvo i broj godina (atribut starosti) najbolje prate *vrednosti plata*. Takođe možemo videti da postoji korelacija između broja godina (atribut starosti) i atributa iskustva.

Toplotna mapa sa vrednostima koeficijenta korelacije

Uočavanje atributa koji su povezani omogućava nam, pre svega, da bolje razumemo domen na koji se podaci odnose. Neke veze se mogu očekivati, dok druge mogu da nam donesu nova znanja. Brisanjem atributa koji su povezani, možemo smanjiti dimenzionalnost skupa podataka. Na taj način možemo ubrzati rad nekih algoritama i lakše razumeti rezultate. Postoje i algoritmi mašinskog učenja koji se ne ponašaju dobro ako postoje asocijacije u skupu podataka - brisanje atributa za koje se ovo odnosi može poboljšati uspeh algoritma.

Ova lekcija je uparena sa Jupiter notebook 03-exploratory-data-analysis.ipynb. Ako želite da vežbate zadatke koje smo opisali, kliknite na link, a zatim na dugme Open in Colab da biste otvorili sadržaj u *Google Colab okruženju*. Ako pogledate sveske na lokalnoj mašini, pronađite svesku sa istim imenom među sadržajem i pokrenite je. Za detaljnija uputstva, pogledajte odeljak *Hands-on zona* i lekcija *Jupiter vežbe sveske.*

U beležnici Jupiter, koristeći funkcije biblioteke *Pandas*, analizirani su podaci seta Titanik. Ovaj set sadrži informacije o putnicima koji su bili na čuvenom brodu Titanik kada je 1912. godine, ploveći u Atlantskom okeanu, udario u ledeni breg i doživio brodolom.

2.8 Kreiranje reprezentacije skupa podataka

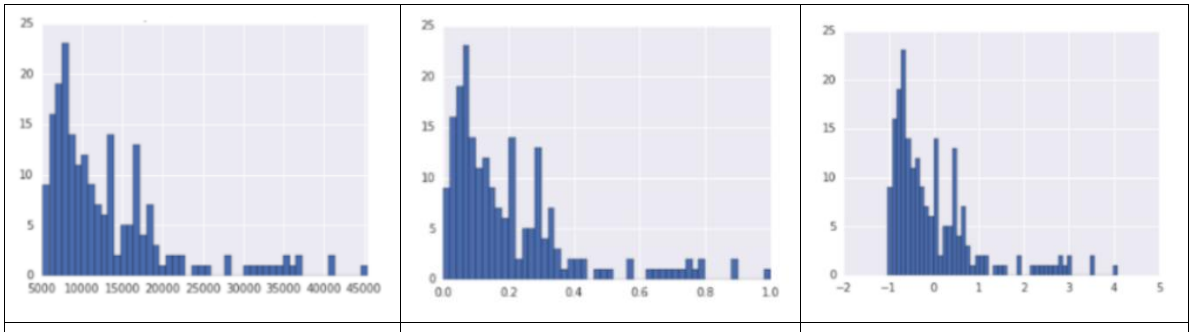
Nakon eksplorativne analize skupa podataka možemo da donesemo odluke o tome koje atribute i instance da odbacimo. Preostali skup podataka treba da pripremimo tako da nad njime možemo da primenimo neki od algoritama mašinskog učenja. U zavisnosti od tipa atributa i skupa vrednosti koje sadrži u nastavku možeš da pročitaš o nekim tehnikama pripreme. U narednoj lekciji ćeš naučiti kada je zapravo idealan trenutak da to uradiš.

Priprema numeričkih atributa

U radu sa numeričkim atributima susrećemo se sa veličinama koje se izražavaju na različitim skalama vrednosti. Recimo, u jednom skupu medicinskih podataka mogu postojati laboratorijske analize sa vrednostima u rasponu od 0 do 1 i informacija o visini pacijenta izražena u centimetrima, od 100 do 250, koja je značajno veća. Mnogi modeli mašinskog učenja su osetljivi na prisustvo ovakvih atributa pa im zbog toga treba više vremena da pronađu rešenje. Dodatno, nije lako interpretirati rezultate koji se dobijaju u ovakvoj postavci. Zato u radu sa numeričkim podacima koristimo tehnike normalizacije koje nam pomažu da skup vrednosti atributa dovedemo na iste opsege vrednosti.

Jedna takva normalizacija je *min-max* normalizacija. Da bismo pojasnili kako se sprovodi, pretpostavimo da treba da normalizujemo vrednost atributa X kojim se izražava visina pacijenata. Neka Xmax = 180 označava najveću visinu pacijenata a Xmin = 110 najmanju. Normalizacija *min-max* se sprovodi tako što se nad svakom vrednošću atributa x primeni formula x − XminXmax − Xmin. Ako je x = 165, nova normalizovana vrednost će iznositi x’= 165 − 110180 − 110 = 0,786. Na ovaj način vrednost atributa svodimo na opseg od 0 do 1.

Poseban vid normalizacije je standardizacija. Ona podrazumeva centriranje vrednosti atributa oko nule i skaliranje na jediničnu varijansu. Da bismo pojasnili kako se sprovodi, možemo opet pretpostaviti da treba da normalizujemo vrednost atributa X kojim se izražava visina pacijenata. Neka je sada Xmean = 153,2 prosečna visina u skupu podataka i σ = 40,23 standardna devijacija. Standardizacija se sprovodi tako što se nad svakom vrednošću atributa x primeni formula x − Xmeanσ. Nova standardizovana vrednost za pacijenta čija je visina x = 165 sada iznosi x’= 165 − 153,240,23 = 0,293.



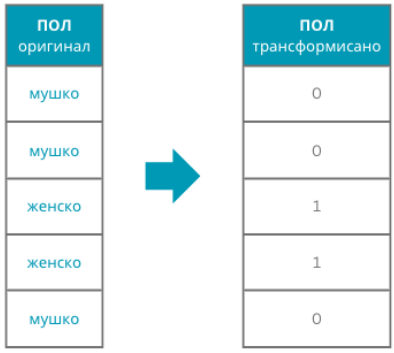
Originalni set	min -max normalizacija	Standardizacija

Efekat normalizacije i standardizacije na skup podataka

Priprema kategoričkih atributa

S obzirom na to da algoritmi mašinskog učenja mogu da se primene samo nad brojevima, kategorički atributi zahtevaju posebnu pripremu. Za njih smo rekli da predstavljaju veličine koje imaju konačan broj vrednosti i da se često pojavljuju u formi niski. Neki od primera koje smo pomenuli su ime boje, pol pacijenta i mesec u godini.

Ukoliko atribut ima samo dve vrednosti, na primer, predstavlja pol pacijenta, njegove vrednosti obično mapiramo u brojeve 0 ili 1. Recimo vrednost ”žensko”možemo mapirati u broj 1, a vrednost ”muško” u broj 0. Ovakve atribute inače zovemo binarnim atributima.

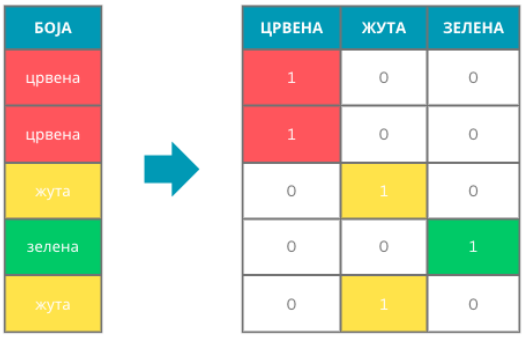


пол оригинал
мушко
мушко
женско
женско
мушко

пол трансформисано
0
0
1
1
0

Primer mapiranja vrednosti

For attributes that can have multiple values, we use *one-hot* coding. To clarify its meaning, we can look at an attribute that represents a color, which can have three values: red, yellow, and green. The idea is to represent the default color attribute using three new attributes, each of which will correspond to one of the values that the color can take: red, yellow and green (look at the picture, this was a complicated sentence). This further means that we will transform each of the values of the initial attribute into a triplet of values, namely the value *of red* into a triplet of *1, 0*, *0*, the value *of yellow* into a triplet of *0, 1, 0, and* the value *of green* into a triplet *of 0, 0, 1.* The triplets, as we can see, consist of zeros and exactly one unit in the column that corresponds to the value of the attribute.



БОЈА
црвена
црвена
жута
зелена
жута

ЦРВЕНА	ЖУТА	ЗЕЛЕНА
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Primer one-hot mapiranja

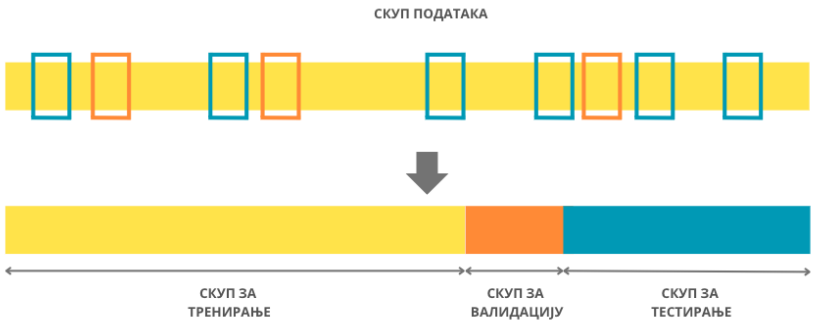
Reprezentacija skupa podataka

Nakon koraka transformacije atributa stižemo do finalnog oblika podataka koji možemo da iskoristimo za pokretanje algoritama učenja. Taj finalni oblik zovemo **reprezentacijom skupa podataka**. U priči do sada smo pokrili, pre svega, kako se stiže do reprezentacije tabelarnih podataka. I za sve druge vrste podataka kao što su slike, audio-zapisi, tekst, video-sadržaji, ali i kompleksne strukture kao što su grafovi, potrebno je da kreiramo odgovarajuće reprezentacije. U delu sa neuronskim mrežama upoznaćemo još neke načine kreiranja reprezentacija.

2.9 Skupovi za treniranje, validaciju i testiranje

U ovoj lekciji ćeš upoznati skupove za treniranje, validaciju i testiranje. O skupu za treniranje možeš da razmišljaš kao o literaturi iz koje model mašinskog učenja uči, dok skup za testiranje možeš da zamisliš kao kontrolni zadatak koji proverava koliko dobro je model naučio i razumeo potrebno gradivo. Skup za validaciju se koristi u procesu učenja modela i možeš da ga zamisliš kao skup pomoćnih testova kojima se proverava koliko je model zapravo spreman za kontrolni i na osnovu čijih rezultata model može da popravi način i uspešnost učenja.

Nakon što se analiziraju podaci i odaberu odgovarajuće instance i atributi, skup svih podataka se deli na **skup za treniranje** (eng. *training set)* i **skup za testiranje** (eng. *test set)*. Kao što se može naslutiti iz imena skupa, skup za treniranje se koristi za treniranje samog modela mašinskog učenja. Nad njim se primenjuje odabrani algoritam i kreira sam model. Skup za testiranje se koristi za testiranje modela, tj. izračunavanje podesnih mera kvaliteta modela. Zahvaljujući njemu može objektivno da se oceni koliko dobro je model naučio potrebni zadatak. Obično jedan deo podataka polaznog skupa koristimo i za kreiranje **skupa za validaciju** (eng. *validation set)*. Skup za validaciju se koristi za praćenje procesa treniranja modela i određivanje nekih konfiguracija modela koje dovode do boljih mera kvaliteta. O ovim temama će još biti reči u nastavku kursa.



Podela skupa podataka na skup za treniranje, skup za validaciju i skup za testiranje

Skupovi za treniranje, validaciju i testiranje se po pravilu kreiraju nasumičnom podelom polaznog skupa podataka. Prvo definišemo koliko veliki treba da budu ovi skupovi, a potom nasumično biramo instance koje će se naći u svakom od njih. Obično je skup za treniranje najveći dok su skupovi za testiranje i validaciju manji jer želimo da imamo dovoljno podataka da obučimo model, ali i dovoljno podataka da adekvatno ocenimo njegove performanse. Praksa je da se odnosi veličina ovih skupova izražavaju proporcijom. Na primer, često će se naći odnos skupa za treniranje i odnos skupa za testiranje izražen kao 2:1, što bi značilo da dve trećine polaznog skupa čini skup za treniranje a jedna trećina skup za testiranje. Slično, proporcija

2:1:1 bi značila da se dve četvrtine (tj. jedna polovina) polaznog skupa koriste kao skup za treniranje a po jedna četvrtina kao skup za validaciju i testiranje.

Iako je zgodno što se skupovi za treniranje, validaciju i testiranje kreiraju nasumično, ipak bi značio i neki uvid u to kako je ova podela izvršena. Na primer, kada želimo da ponovimo eksperiment ili omogućimo drugima da ga samostalno izvedu (ovo je važno svojstvo eksperimenata i zove se reproducibilnost), poželjno je da se koriste isti skupovi za treniranje, validaciju i testiranje. Slično, kada rešavamo zadatak, nismo baš odmah sigurni šta je najbolje uraditi pa isprobavamo veći broj algoritama i kreiramo veći broj modela. Zbog korektnosti upoređivanja značilo bi da sve modele kreiramo nad istim skupom za treniranje i ocenjujemo nad istim skupom za testiranje. Zato je dobro na nivou biblioteke sa kojom se radi podesiti parametar koji utiče na slučajnost podele (obično se zove *random seed* i ima istu svrhu kao podešavanje semena kod generatora slučajnih brojeva) ili prosto u startu izvršiti podelu podataka i nadalje je konzistentno koristiti. Neki često korišćeni skupovi podataka imaju ove predefinisane podele na skup za treniranje, validaciju i testiranje (na primer, možeš da pogledaš skup *MNIST)*.

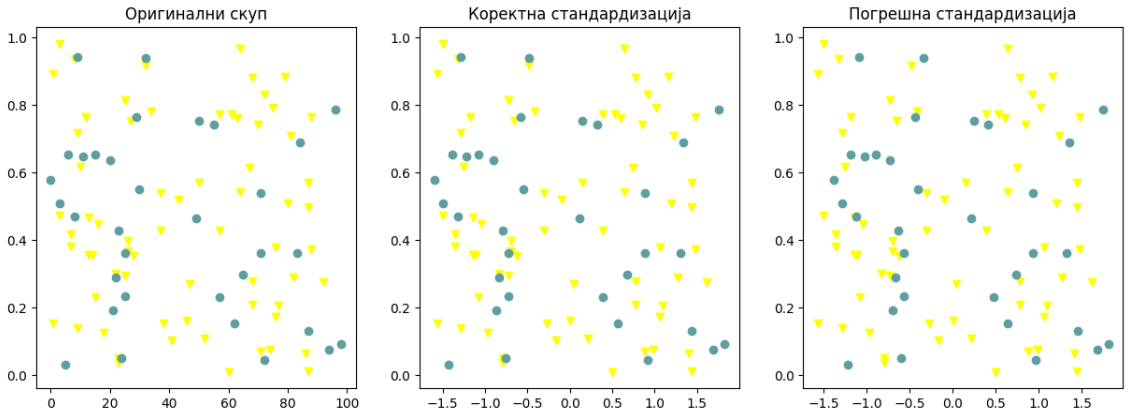
Važno svojstvo koje treba da ispune skupovi za treniranje, validaciju i testiranje je da budu disjunktni. To znači da svaka instanca polaznog skupa podataka prilikom kreiranja skupova za treniranje, validaciju i testiranje mora pripasti tačno jednom od ovih skupova, ne smeju postojati preseci i zajedničke instance. Podsetimo se da se od modela mašinskog učenja očekuje da dobro generalizuju, tj. da se dobro ponašaju za nove instance koje model nije imao prilike da susretne u skupu za treniranje. Ukoliko se skupovi za treniranje i skupovi za testiranje preklapaju, nećemo biti u mogućnosti da objektivno ocenimo da li model zaista uči ili memoriše, tj. pamti informacije iz skupa za treniranje. Slično važi i za odnos skupa za treniranje i skupa za validaciju: funkcija skupa za validaciju je da pomogne u odabiru konfiguracija koje će učenje učiniti što uspešnijim. Ukoliko se ovi skupovi preklapaju, nećemo biti u mogućnosti da objektivno i nepristrasno ocenimo ponašanje modela i odaberemo podesne konfiguracije.

Uslov da skupovi za treniranje, validaciju i testiranje treba da budu disjunktni znači i da se informacije iz jednog od skupova nikako ne smeju prelivati na druge skupove. Ovo je posebno važno prilikom primene tehnika pretprocesiranja i pripreme skupova. Razmotrimo sledeći primer. Pomenuli smo da se zbog osetljivosti modela mašinskog učenja na vrednost atributa često vrši standardizacija numeričkih atributa. Neko može pristupiti ovoj transformaciji tako što će na osnovu celog skupa izračunati srednju vrednost i standardnu devijaciju, a zatim je iskoristiti, redom, za standardizaciju skupova za treniranje i testiranje. Kako ne bi došlo do prelivanja informacija, korektan redosled ovih koraka je zapravo sledeći:

1. podela skupa podataka na skup za treniranje i testiranje,
2. izračunavanje srednje vrednosti i standardne devijacije samo na skupu za treniranje,
3. transformacija skupa za treniranje koristeći izračunate vrednosti,
4. transformacija skupa za testiranje koristeći vrednosti izračunate nad skupom za treniranje.

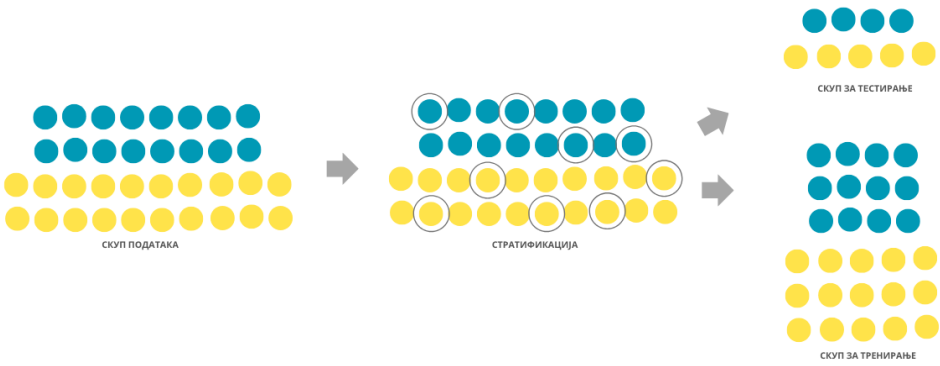
Zbog želje da ne pogreši, neko može razmišljati i na sledeći način: nakon što podelim polazni skup podataka na skup za treniranje i skup za testiranje, izvršiću posebno standardizaciju skupa za treniranje i skupa za testiranje. I ovaj pristup, iako oprezniji, nije korektan jer dovodi do modifikacije skupa za testiranje. Na donjoj levoj slici žuti trouglići predstavljaju instance skupa za treniranje, a plavi krugovi instance skupa za testiranje. Slika u sredini predstavlja ove instance nakon korektne standardizacije (možeš pažljivo da uporediš slike i raspored tačaka - skala duž x-ose se promenila usled standardizacije, sve drugo je ostalo

isto). Na desnoj slici možeš da vidiš instance nakon što se standardizacija zasebno izvrši nad skupom za treniranje i skupom za testiranje - prostorni raspored se sada prilično promenio.



Primeri korektne i pogrešne standardizacije

Prilikom podele polaznog skupa podataka bilo bi idealno da očuvamo proporcije u odnosu na vrednosti atributa i vrednost ciljne promenljive. Na primer, ukoliko je u skupu medicinskih podataka odnos muških i ženskih pacijenata 4:5, bilo bi idealno da, nakon podele, u skupu za treniranje i u skupu za testiranje odnos pacijenata bude približno 4:5. Tehnike koje omogućavaju ovakvu vrstu podele nazivamo tehnikama **stratifikacije.** Ipak, zbog broja atributa i njihovih kombinacija, u praksi ovo često nije realističan zahtev pa se najčešće insistira na proporcionalnosti u odnosu na vrednosti ciljne promenljive. O ovoj temi ćemo posebno razgovarati u kontekstu zadatka klasifikacije.



Stratifikovani skupovi za treniranje i testiranje

3. MODELI UČENJA

Dobrodošli na temu **Modeli učenja**! Ovaj odeljak se bavi različitim modelima i tehnikama koje se koriste u mašinskom učenju za rešavanje različitih vrsta problema. Naučićete o linearnoj regresiji, klasifikaciji, stablima odlučivanja i algoritmu k-najbližih suseda, između ostalog. Razgovaraćemo o karakteristikama, prednostima i nedostacima svakog modela, kao i o važnosti validacije modela kako bi se osigurala tačnost i pouzdanost. Kroz praktične vežbe, primenite ove modele na stvarne skupove podataka i interpretirati rezultate.

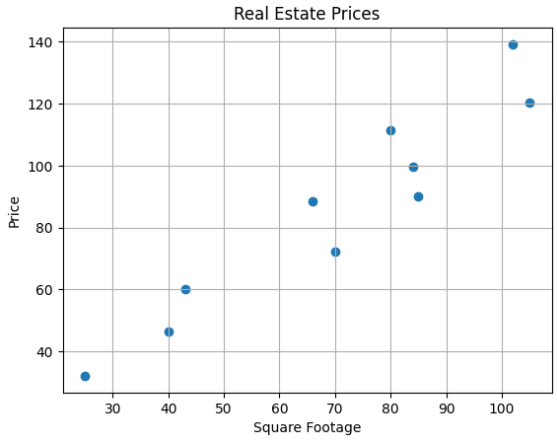
3.1 Linearna regresija

Vratimo se sada na primer koji smo koristili za upoznavanje paradigme programiranja vođenog podacima. U skupu podataka imali smo informacije o kvadraturi nekretnina i njihovim cenama, a naš zadatak je bio da naučimo vezu između ovih vrednosti tako da možemo da procenjujemo cene za nove nekretnine.

Jednostavnosti radi, neka bude da raspolažemo skupom za treniranje koji sadrži 10 instanci koje su popisane u donjoj tabeli:

Square Footage (m^2)	Real estate Prices (1000€)
43	60
25	32.1
66	88.4
80	111.4
105	120.32
70	72.1
40	46.3
85	90.1
84	99.6
102	139.2

Ove instance možemo da prikažemo i grafički. Duž x-ose ćemo postaviti vrednosti kvadratura, duž u-ose vrednosti cena nekretnina i parove vrednosti obeležiti plavim kružićima.



Odaberimo za model funkciju koja povezuje kvadrature nekretnina x i cene nekretnina y jednačinom $y = \beta_0 + \beta_1 x$, gde β_0 i β_1 predstavljaju nepoznate parametre. Ovo je takozvani **linearni model,** a pošto ga koristimo za rešavanje zadatka regresije zovemo ga i **modelom linearne regresije.** Primetimo da je ovo zapravo jednačina prave $y = kx + n$, gde je koeficijent pravca prave obeležen sa β_1 a slobodni član sa β_0. Motivacija za uvođenje ovog modela leži u tome što tačke prate zamišljenu dijagonalu kvadranta, možda malo spuštenu niže.

Ova sekcija je uparena sa Jupyter sveskom 05-1-linear_regression.ipynb. Da bi mogao da pratiš sadržaj dalje, klikni na link, a potom i na dugme Open in Colab da bi se sadržaj otvorio u okruženju *Google Colab*. Ukoliko sveske pregledaš na lokalnoj mašini, među sadržajima pronađi svesku sa istim imenom i pokreni je. Za detaljnije instrukcije pogledaj sekciju *Hands-on zona* i lekciju *Jupyter sveske za vežbu*.

Tvoj zadatak je da u pratećoj svesci učitaš skup podataka o nekretninama i da odabereš vrednosti parametara β_0 i β_1 za koje misliš da najbolje odgovaraju ovim podacima. Možeš ih fino podešavati pomeranjem slajdera levo i desno. Zapamti vrednosti koje si odabrao i koje ideje su te vodile prilikom određivanja parametara.

Verovatno si se trudio da prilikom izbora parametara dobiješ pravu koja prilazi što bliže zadatim tačkama i pravi što manja odstupanja. Pojedinim izborima parametara si bio podjednako zadovoljan dok su neki bili baš loši. I iz ugla mašinskog učenja pokušavamo da pronađemo vrednosti parametara β_0 i β_1 za koje pravimo najmanju grešku s tim što moramo precizno da definišemo šta je to zapravo greška. Evo kako ćemo to uraditi.

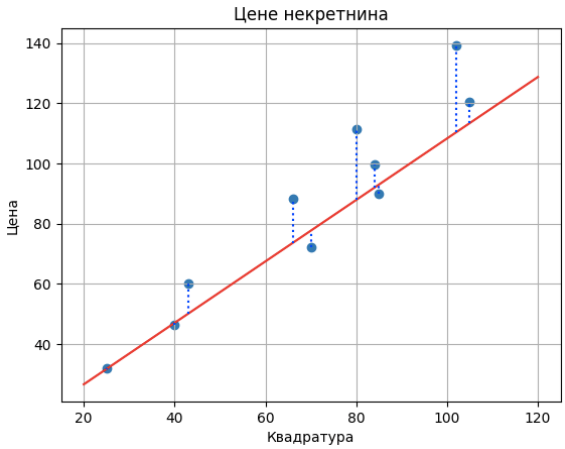
Pretpostavimo da su odabrane vrednosti $\beta_0 = 6.3$ i $\beta_1 = 1.02$. Proširimo sada tabelu sa podacima kolonom sa vrednostima koje je izračunao ovaj model linearne regresije za vrednosti kvadratura kojima raspolažemo. Njih iz ugla modela predstavljamo veličinom x.

Square Footage (m^2)	Real estate Prices (1000€)	Model price y=6.3+1.02x (1000€)
43	60	50.16
25	32.1	31.8
66	88.4	73.62
80	111.4	87.9
105	120.32	113.4
70	72.1	77.7
40	46.3	47.1
85	90.1	93
84	99.6	91.98
102	139.2	110.34

The difference between the values that are expected (known in the data set) and the values that we have calculated (remember to call them predictions) is an error. Now let's calculate all the errors and record them in the table.

Square Footage (m^2)	Real estate Prices (1000€)	Model price y=6.3+1.02x (1000€)	Model error
43	60	50.16	9.84
25	32.1	31.8	0.3
66	88.4	73.62	14.78
80	111.4	87.9	2.53
105	120.32	113.4	6.92
70	72.1	77.7	-5.6
40	46.3	47.1	-0.8
85	90.1	93	-2.9
84	99.6	91.98	7.62
102	139.2	110.34	28.86

Da bi lakše mogli da ispratimo ponašanje grešaka, na donjoj slici su njihove vrednosti prikazane plavim isprekidanim linijama.



Da bismo dobili predstavu o ukupnoj grešci modela, nije mudro sabirati pojedinačne greške pošto su neke vrednosti grešaka pozitivne a neke vrednosti negativne. Zato možemo da ih kvadriramo pa saberemo - ovo će nam preneti i jaču informaciju o veličini greške bez obzira na to da li je pozitivna ili negativna. Ukoliko ovako dobijeni zbir podelimo brojem instanci u skupu, dobićemo predstavu o prosečnoj grešci modela. U našem slučaju to je:

$$(9.84^2 + 0.32^2 + 14.782^2 + 23.52^2 + 6.92^2 + (-5.6)2 + (-0.8)2 + (-2.9)2 + 7.622 + 28.86^2)/10 = 184.687$$

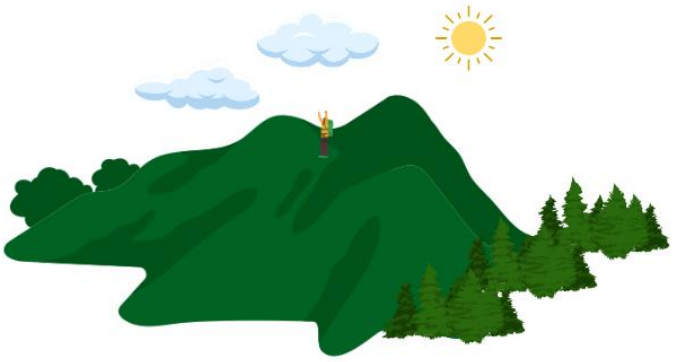
Ovako izračunata greška modela linearne regresije se zove **srednjekvadratna greška** (eng. *mean squared error, MSE*). Za fiksirane vrednosti parametara β0 i β1 postupak izračunavanja koji smo opisali možemo skraćeno prikazati formulom $\frac{1}{N}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))^2$. U njoj parovi x_i, y_i odgovaraju pojedinačnim instancama, kvadraturama nekretnina x_i i njihovim cenama y_i, a brojem n je označen ukupan broj instanci. To je 10 u našem slučaju. Izraz koji figuriše u sumi predstavlja razliku očekivanih y_i i izračunatih $\beta_0 + \beta_1 x_i$ vrednosti.

Srednjekvadratna greška je greška koju uvek uparujemo sa modelom linearne regresije i koju želimo da što više smanjimo izborom pravix parametara β_0 i β_1. Iz iskustva podešavanja parametara si video da to i nije baš lak zadatak. Srećom, postoje matematičke tehnike koje nam u tome mogu pomoći. Da bismo otkrili kako ovo da uradimo, pređimo na sledeću lekciju o gradijentnom spustu.

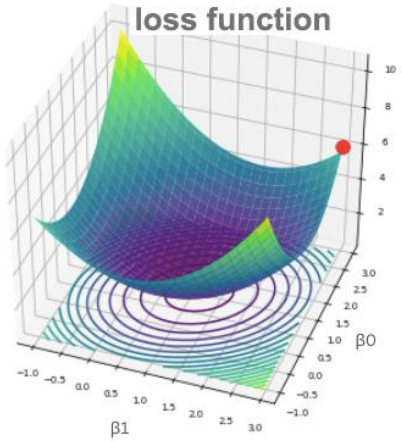
3.2 Gradijentni spust

U ovoj lekciji upoznaćemo gradijentni spust, tehniku koja nam pomaže da pronađemo parametre za koje funkcija srednjekvadratne greške ima najmanju vrednost. Ovu tehniku možemo, pod određenim uslovima, da primenimo i na druge funkcije.

Opet ćeš morati nešto da zamisliš - ovoga puta da se nalaziš na vrhu lepe planine. To i ne pada tako teško! Nevolja je što sada sledi zadatak: da se brzo spustiš do podnožja! Jedan način da to uradiš je da prvo pogledaš oko sebe i proveriš duž kog pravca u tvojoj okolini je planina najstrmija - imaj na umu da treba baš brzo da se spustiš! Onda možeš da napraviš jedan pažljivi korak u tom pravcu pa da zastaneš i opet pogledaš oko sebe. Ponovo možeš da primetiš duž kog pravca je planina najstrmija u tvojoj okolini, napraviš korak u tom pravcu i zastaneš. Jasno ti je da ovaj redosled osmatranja, izbora pravca i iskoraka možeš da nastaviš da ponavljaš sve dok ne stigneš do podnožja. Tu te čeka osveženje za uspešno obavljeni zadatak!



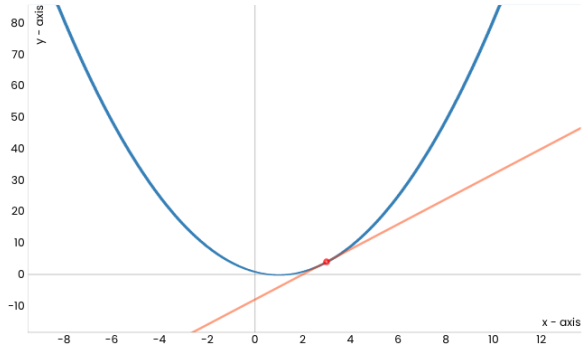
Malo fizičke aktivnosti usred priče o linearnoj regresiji nije na odmet, ali naslućuješ da postoji još nešto. Funkcija srednjekvadratne greške zavisi od izbora parametara β_0 i β_0 - za različite kombinacije vrednosti β_0 i β_1 dobijamo različite vrednosti greške. Ako nacrtamo grafik ove funkcije, na primer, duž x-ose zabeležimo vrednosti β_0, duž y-ose zabeležimo vrednosti β_1, a duž z-ose vrednosti greške, dobićemo grafik koji izgleda kao na donjoj slici. Ako crvenom tačkom obeležimo neki nasumični izbor parametara β_0 i β_1, da bismo stigli do tačke za koju je vrednost greške najmanja, zaista moramo da se spustimo u podnožje ove površi. Zato je "tehnika" koju smo razvili u prethodnom primeru vrlo relevantna. Potrebno je samo da osmislimo kako da tražimo najstrmije pravce spusta. U tome će nam pomoći izvodi funkcija.



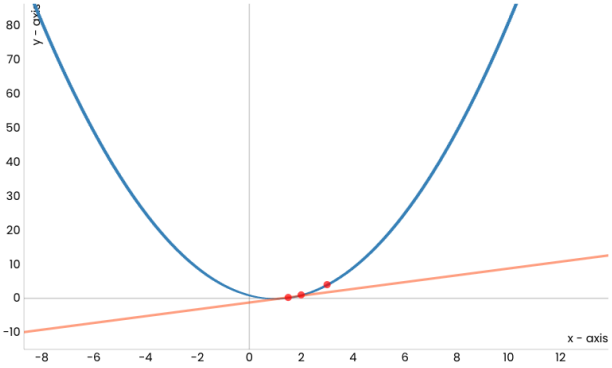
Grafik funkcije srednjekvadratne greške

Najmanja vrednost jedne funkcije se zove minimum.

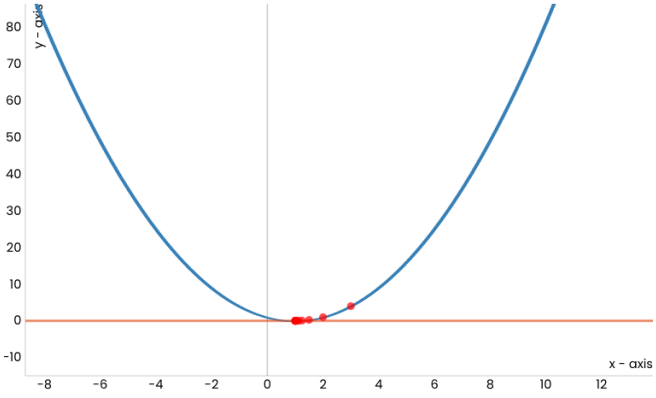
Posmatrajmo sada kvadratnu funkciju $f(x) = (x-1)^2$ čiji je grafik prikazan na donjoj slici i pokušajmo da tehnikom spusta stignemo do njenog minimuma - on je u tački x=1 i iznosi 0.



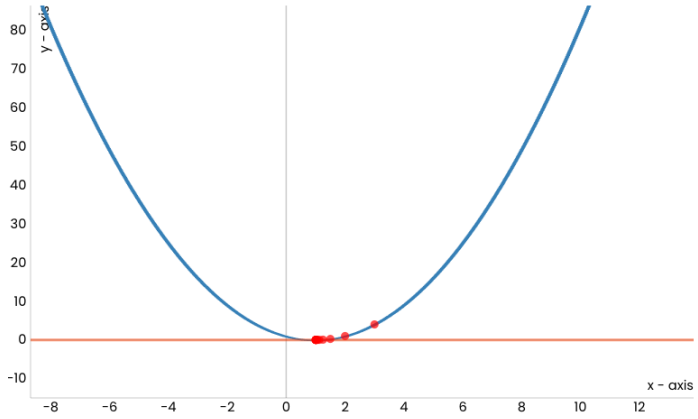
Uočimo i crvenu tačku koja odgovara vrednosti x=3 (nasumično smo je odabrali) i koja označava startnu poziciju kretanja ka minimumu ove funkcije. Deluje da je narandžastom linijom obeležen najstrmiji pravac duž koga možemo da započnemo spust. Zanimljivo je da ova linija zapravo predstavlja tangentu naše funkcije u tački x=3. Ako duž ovog pravca napravimo korak, naći ćemo se u novoj tački. Obeležimo i njenu vrednost crvenom bojom i prikažimo je na grafiku. Ona je malo bliže očekivanom minimumu.



Sada možemo da ponovimo postupak: nacrtajmo tangentu u novoj tački, a potom i napravimo korak duž tog pravca.



Nakon određenog broja koraka ovaj postupak će nas dovesti do minimuma funkcije, tj. do tačke x = 1.



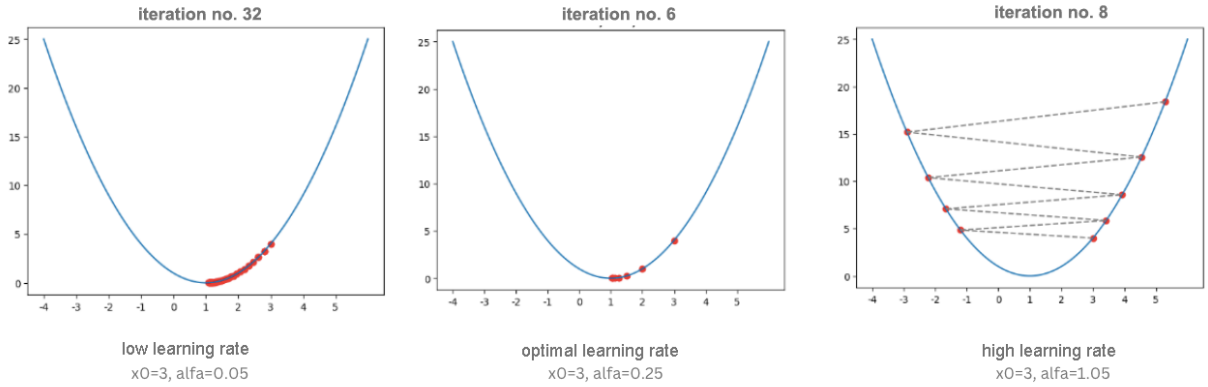
Ova sekcija je uparena sa Jupyter sveskom 05-2-gradient_descent.ipynb. Da bi mogao da pratiš sadržaj dalje, klikni na link, a potom i na dugme Open in Colab da bi se sadržaj otvorio u okruženju *Google Colab*. Ukoliko sveske pregledaš na lokalnoj mašini, među sadržajima pronađi svesku sa istim imenom i pokreni je. Za detaljnije instrukcije pogledaj sekciju *Hands-on zona* i lekciju *Jupyter sveske za vežbu.*

U svesci koja prati ovaj materijal možeš i sam da pokreneš animaciju i uveriš se da je tako.

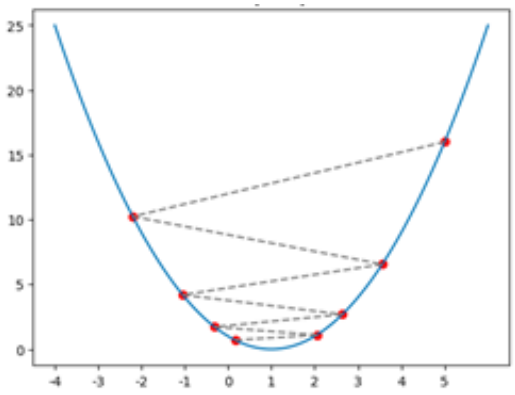
Pre nego što detaljnije prođemo kroz postupak koji smo opisali, podsetimo se kakve su to prave tangente. Za neku fiksiranu tačku x koeficijent pravca tangente u tački x jednak je vrednosti prvog izvoda funkcije u tački x. Prvi izvod naše funkcije je funkcija $f'(x) = 2x - 2$ i u početnoj tački $x = 3$ vrednost izvoda je $f'(x) = 4$. To znači da tangenta ima jednačinu $y = 4x - 8$ (broj -8 smo dobili iz uslova da ova prava mora da sadrži tačku (3, 4)). Zato možemo i da kažemo da tangenta ima pravac koji odgovara izvodu funkcije u nekoj tački, a za samo kretanje u tom pravcu da je kretanje duž pravca izvoda u toj tački. Sada je dilema da li se krećemo uz ili niz, tj. da li pratimo pravac izvoda ili njemu suprotan pravac? Pa, pošto želimo da se spuštamo ka minimumu, treba da pratimo pravac suprotan pravcu izvoda funkcije.

Ako sada sa x_0, obeležimo početnu tačku, novu tačku x_1 dobili smo tako što smo napravili korak duž pravca izvoda funkcije u tački x_0. Ako sa α obeležimo dužinu koraka, vrednost nove tačke x_1 izračunavamo kao $x_1 = x_0 - \alpha f'(x_0)$. Pošto postupak ponavljamo, vrednost tačke x_2 izračunavamo kao $x_2 = x_1 - \alpha f'(x_1)$ i nastavljamo redom sa izračunavanjima $x_3 = x_2 - \alpha f'(x_2), x4 = x_3 - \alpha f'(x_3)$, … Postupak ponavljamo sve dok za dve uzastopne vrednosti, recimo za x_{34} i x_{35}, vrednosti funkcije nisu dovoljno blizu, tj. dok apsolutna vrednost razlike $f(x_{35}) - f(x_{34})$ nije manja od neke unapred zadate tačnosti, recimo 0,001. Tako računski možemo da se približimo pojmu konvergencije u matematici.

Vrednost α koju smo uveli se zove **korak učenja** (eng. *learning rate*) i predstavlja vrlo važan parametar algoritma koji smo opisali. Ukoliko su vrednosti za α veoma male, trebaće nam mnogo vremena da stignemo do minimuma. Sa druge strane, ako su vrednosti za α veoma velike, može se desiti da preskočimo minimum ili zapadnemo u cikcak zamku stalnim skakutanjima oko njega! Pogledaj donju sliku!



Uticaj izbora koraka učenja



Cikcak zamka

Oba ova ponašanja obavezno proveri i sam u pratećoj svesci koristeći različita podešavanja za korak učenja u animaciji.

Algoritam koji smo opisali se zove **gradijentni spust** (eng. *gradient descent*) i uprkos svojoj jednostavnosti predstavlja jedan od najvažnijih algoritma u mašinskom učenju jer omogućava pronalaženje najmanje vrednosti funkcije greške. Postoji mnogo detalja u vezi sa ovim algoritmom u koje mi nećemo zalaziti, a koji se tiču osobina funkcija na koje ovaj algoritam može uspešno da se primeni, numeričkog izračunavanja izvoda i izbora koraka učenja. Svi oni se moraju razmotriti prilikom praktične primene algoritma.

Sam algoritam nije neugodno isprogramirati pa ćemo se upustiti u avanturu. Potrebna nam je funkcija f, koja će da računa vrednost zadate funkcije, i funkcija f_izvod, koja će da izračunava vrednost izvoda zadate funkcije. Potrebno je da definišemo i korak učenja alfa i zaustavne kriterijume: postupak ćemo obustaviti kada vrednosti funkcije u dvema uzastopnim iteracijama budu dovoljno blizu (razlika njihovih vrednosti je manja od neke unapred zadate tačnosti epsilon) ili kada dostignemo neki konačan broj iteracija max_broj_iteracija (moramo da se osiguramo i u slučajevima nepodesnih izbora koraka učenja).

```
def gradient_descent(f, f_derivative, x, alpha, epsilon, max_iterations):
    # set the initial value for x
    x_old = x
    # in each iteration...
    for i in range(0, max_iterations):
        # calculate the current value for x
        x_new = x_old - alpha * f_derivative(x_old)
        # and then check if the stopping criterion is met
        if np.abs(f(x_new) - f(x_old)) < epsilon:
            break
        # if the criterion is not met, prepare x for the next iteration
        x_old = x_new
    # at the end of the whole process, prepare a report with information:
    # whether the algorithm stops,
    # how many iterations it lasted,
    # and what value of x was found
    report = {}
    report['stops'] = i != max_iterations
    report['number_of_iterations'] = i
    report['x_min'] = x_old

    return report
```

Funkciju koju smo razmatrali i njen izvod možemo definisati sledećim Python blokovima:

```
def f(x):
   return (x-1)**2
def f_izvod(x):
   return 2*x-2
```

Nakon pokretanja funkcije gradijentni_spust za vrednosti argumenata x0 = 3, alfa = 0.1, epsilon = 0.001 i max_broj_iteracija = 100 dobijamo da je minimum funkcije broj 1,0048 što možemo i da potvrdimo. Kôd možeš i sam da izvršiš i uveriš se da se dobija baš ovaj rezultat. Ne propusti da ispitaš i kako se rezultati menjaju ukoliko se odaberu druge vrednosti argumenata.

Sada možemo da se vratimo i na problem pronalaženja parametara β_0 i β_1 linearne regresije za koju vrednost srednjekvadratne greške treba da ima najmanju vrednost. Funkcija srednjekvadratne greške je funkcija dveju promenljivih - zavisi i od vrednosti parametra β_0 i od vrednosti parametra β_1. Kada radimo sa funkcijama više promenljivih, u opštem slučaju sa n promenljivih $x_1, x_2, x_3, \ldots, x_n$, izvod koji smo koristili u algoritmu gradijentnog spusta uopštavamo vektorom parcijalnih izvoda - za svaku od promenljivih

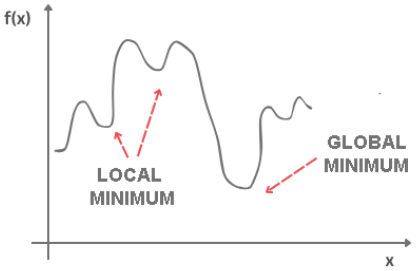
izračunavamo pojedinačno izvode. Recimo, za funkciju $\frac{1}{2}(x_1^2 + 10x_2^2)$, izvod po promenljivoj x_1 se dobija tako što se promenljiva x_2 proglasi konstantom pa potom primene standardna pravila za računanje izvoda koja nas dovode do $\frac{1}{2} \cdot 2 \cdot x_1 = x_1$. Sa druge strane, izvod po promenljivoj x_2 se računa tako što se promenljiva x_1 proglasi konstantom pa primene standardna pravila za računanje izvoda. Sada dobijamo $\frac{1}{2} \cdot 10 \cdot 2 \cdot x_2 = 10 \cdot x_2$. Sada dobijamo da je vektor izvoda po pojedinačnim promenljivama (takve izvode zovemo parcijalnim) vektor $[x_1, 10 \cdot x_2]$. U matematici, pa i u mašinskom učenju, ovi vektori se zovu **gradijenti** pa otuda dolazi i ime samog algoritma. Za obeležavanje gradijenata se koristi simbol trouglić nadole, ∇ koji se zove nabla. Tako bi precizan zapis gradijenta polazne funkcije f' bio $\nabla f(x_1, x_2) = [x_1, 10 \cdot x_2]$ i omogućavao bi nam da pratimo duž kojih pravaca izvoda pojedinačno treba da se krećemo prilikom spusta.

Ostali koraci algoritma gradijentnog spusta se ne razliku mnogo za slučaj funkcija više promenljivih: očekujemo da se algoritam zaustavi nakon što se ostvari željena tačnost ili nakon što se izvrši određeni broj iteracija.

Sada kada razumemo i kako gradijentni spust funkcioniše za funkcije više promenljivih vratimo se na izračunavanje parametara β_0 i β_1. Rekli smo da je jednačina srednjekvadratne greške $\frac{1}{N}\sum_{i=1}^{N}(yi - (\beta_0 + \beta 1 xi))^2$. Pošto je to funkcija za koju treba da pronađemo minimum, ako zasučemo rukave pa proverimo, dobićemo da je izvod srednjekvadratne funkcije po β_0 baš $\frac{2}{N}\sum_{i=1}^{N}(\beta_0 + \beta_1 x_i - y_i)$ i izvod po $\frac{1}{N}\sum_{i=1}^{N}(\beta_0 + \beta_1 x_i - y_i) \cdot x_i$. Ovi izvodi nam ukazuju duž kojih pravaca treba da se krećemo i koliko treba da korigujemo vrednosti za β_0 i β_1 u svakom koraku iteracije gradijentnog spusta.

U svesci možeš da vidiš i kako se ove vrednosti izračunavaju kroz kôd, a potom i da prođeš kroz ceo postupak prilagođenog gradijentnog spusta. Za skup o nekretninama koji smo uveli, stići ćemo do vrednosti $\beta_0 = 2.056$ i $\beta_0 = 1.198$.

Rekli smo da postoje određeni preduslovi koje funkcija treba da zadovolji da bi njen minimum mogao da se pronađe tehnikom gradijentnog spusta (potrebno je da funkcija bude diferencijabilna). Važno je da znaš i da se u opštem slučaju na ovaj način dostiže neki *lokalni minimum*. Recimo, funkcija na donjoj slici ima nekoliko lokalnih minimuma i samo jedan *globalni minimum*. U nekim slučajevima, recimo kada je funkcija konveksna, lokalni i globalni minimum se poklapaju pa uvek stižemo do željenog rešenja, globalnog minimuma. Funkcija srednjekvadratne greške je konveksna po parametrima β_0 i β_1.



Lokalni i globalni minimum

Oblast matematike koja se bavi pronalaženjem maksimalnih i minimalnih vrednosti funkcija (jednim imenom ih zovemo optimumima) zove se **matematička optimizacija**. Gradijentni spust je samo jedan algoritam iz palete ove oblasti.

3.3 Polynomial regression

Šta ako su vaši podaci zapravo složeniji od jednostavne prave linije? Iznenađujuće, zapravo možete koristiti linearni model da biste se uklopili u nelinearne podatke. Jednostavan način da to uradite je da dodate ovlašćenja svake funkcije kao nove funkcije, a zatim trenirate linearni model na ovom proširenom skupu funkcija. Ova tehnika se zove polinomijalna regresija.

Polinomijalna regresija je regresijski algoritam koji modelira odnos između zavisne (i) i nezavisne varijable (k) kao n-tog stepena polinoma. Jednačina polinomske regresije je data u nastavku:

$$y = \beta_0 + \beta_1 x_1^1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \cdots \ldots \beta_n x_1^n$$

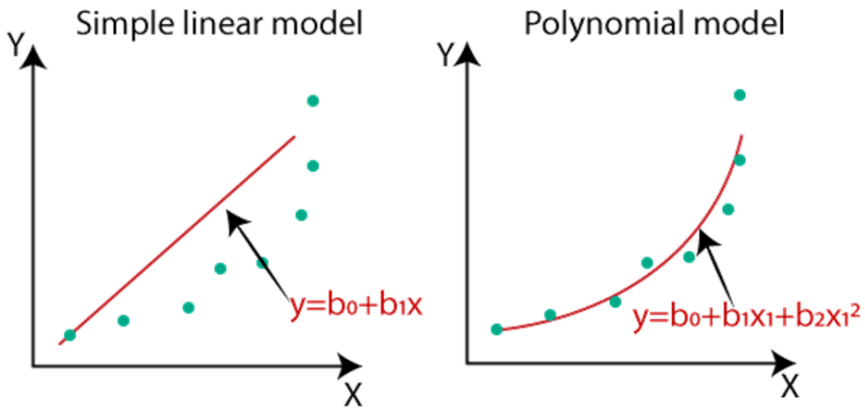
Takođe se naziva posebnim slučajem višestruke linearne regresije u ML. Zato što dodajemo neke polinomske termine u višestruku linearnu regresijsku jednačinu da bismo je pretvorili u polinomnu regresiju.
To je linearni model sa nekim modifikacijama kako bi se povećala tačnost.
Skup podataka koji se koristi u polinomskoj regresiji za obuku je nelinearne prirode.
Koristi linearni regresijski model kako bi se uklopio u komplikovane i nelinearne funkcije i skupove podataka.

Dakle, *"U polinomijalnoj regresiji, originalne karakteristike se pretvaraju u polinomne karakteristike potrebnog stepena (2,3,..,n), a zatim se modeliraju pomoću linearnog modela."*

Potreba polinomske regresije u ML može se razumeti u sledećim tačkama:

- Ako primenimo linearni model na **linearni skup podataka**, onda nam daje dobar rezultat kao što smo videli u jednostavnoj linearnoj regresiji, ali ako primenjujemo isti model bez ikakvih modifikacija na **nelinearnom skupu podataka**, onda će proizvesti drastičan izlaz. Zbog čega će se funkcija gubitka povećati, stopa grešaka će biti visoka, a tačnost će se smanjiti.

- Dakle, za takve slučajeve, **gde su tačke podataka raspoređene na nelinearan način, potreban nam je model polinomske regresije**. Možemo ga razumeti na bolji način koristeći dijagram poređenja ispod linearnog skupa podataka i nelinearnog skupa podataka.



- Na gornjoj slici uzeli smo skup podataka koji je raspoređen nelinearno. Dakle, ako pokušamo da ga pokrijemo linearnim modelom, onda možemo jasno videti da jedva pokriva bilo koju tačku podataka. S druge strane, kriva je pogodna da pokrije većinu tačaka podataka, što je polinomijalni model.
- Stoga, *ako su skupovi podataka raspoređeni na nelinearan način, onda bi trebalo da koristimo model polinomske regresije umesto jednostavne linearne regresije.*

Napomena: Algoritam polinomne regresije se takođe naziva polinomna linearna regresija jer ne zavisi od varijabli, već zavisi od koeficijenata, koji su raspoređeni linearno.

Jednostavna linearna regresijska jednačina	$y = \beta_0 + \beta_1 x_1$
Višestruka linearna regresijska jednačina	$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + + \cdots \ldots \beta_n x_n$
Polinomijalna regresijska jednačina	$y = \beta_0 + \beta_1 x_1^1 + \beta_2 x_1^2 + \cdots \ldots \beta_n x_1^n$

Implementacija polinomske regresije pomoću Pythona:

Ovde ćemo implementirati polinomsku regresiju koristeći Python. Mi ćemo to razumeti upoređivanjem modela polinomske regresije sa modelom jednostavne linearne regresije. Dakle, prvo, hajde da shvatimo problem za koji ćemo izgraditi model.

Opis problema: Postoji kompanija za ljudske resurse, koja će zaposliti novog kandidata. Kandidat je rekao svoju prethodnu platu 160K godišnje, a HR mora da proveri da li govori istinu ili blefira. Dakle, da bi to identifikovali, oni imaju samo skup podataka njegove prethodne kompanije u kojoj se pominju plate prvih 10 pozicija sa njihovim nivoima. Proverom dostupnog skupa podataka, otkrili smo da postoji **nelinearna veza između nivoa pozicije i plata**. Naš cilj je da izgradimo model **regresije detektora blefiranja**, tako da HR može zaposliti poštenog kandidata. U nastavku su navedeni koraci za izgradnju takvog modela.

Position	Level(X-variable)	Salary(Y-Variable)
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Koraci za polinomijalnu regresiju:

Glavni koraci uključeni u polinomijalnu regresiju su dati u nastavku:

- Predobrada podataka
- Izgradite model linearne regresije i uklopite ga u skup podataka
- Izgradite model polinomske regresije i prilagodite ga skupu podataka
- Vizuelizujte rezultat za model linearne regresije i polinomne regresije.
- Predviđanje izlaza.

Korak pre-obrade podataka:

Korak predobrade podataka će ostati isti kao u prethodnim regresijskim modelima, osim nekih promena. U modelu polinomske regresije nećemo koristiti skaliranje funkcija, a takođe nećemo podeliti naš skup podataka u skup za obuku i testiranje. Ima dva razloga:

- Skup podataka sadrži vrlo manje informacija koje nisu pogodne za podelu u skup testova i obuke, inače naš model neće moći da pronađe korelacije između plata i nivoa.
- U ovom modelu želimo vrlo precizna predviđanja za platu, tako da model treba da ima dovoljno informacija.

Kod po korak pre obrade je dat u nastavku:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
#importing datasets
data_set= pd.read_csv('Position_Salaries.csv')
#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

Kreiranje linearnog regresijskog modela:

Sada ćemo kreirati i uklopiti model linearne regresije u skup podataka. U izgradnji polinomske regresije, uzećemo model linearne regresije kao referencu i uporediti oba rezultata. Kod je dat ispod:

```
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

U gornjem kodu, kreirali smo Simple Linear model koristeći **lin_regs** objekat klase **LinearRegression** i uklopili ga u promenljive skupa podataka (x i y).

Output:

Out[5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

Kreiranje modela polinomske regresije:

Sada ćemo izgraditi model polinomske regresije, ali će biti malo drugačiji od jednostavnog linearnog modela. Jer ovde ćemo koristiti **PolinomialFeatures** klasu **preprocesiranje** biblioteke. Koristimo ovu klasu da dodamo neke dodatne funkcije našem skupu podataka.

```
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 2)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

U gornjim linijama koda, koristili smo **poly_regs.fit_transform(x)**, jer prvo pretvaramo našu matricu karakteristika u matricu polinomnih karakteristika, a zatim je uklapamo u model polinomske regresije. Vrednost parametra (stepen = 2) zavisi od našeg izbora. Možemo ga izabrati u skladu sa našim polinomskim karakteristikama.

Nakon izvršenja koda, dobićemo još jedan matrični **x_poly**, koji se može videti pod varijabilnom opcijom istraživača:

	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25
5	1	6	36
6	1	7	49
7	1	8	64
8	1	9	81
9	1	10	100

Zatim smo koristili još jedan LinearRegression objekat, naime **lin_reg_2**, kako bismo uklopili naš **x_poly** vektor u linearni model.

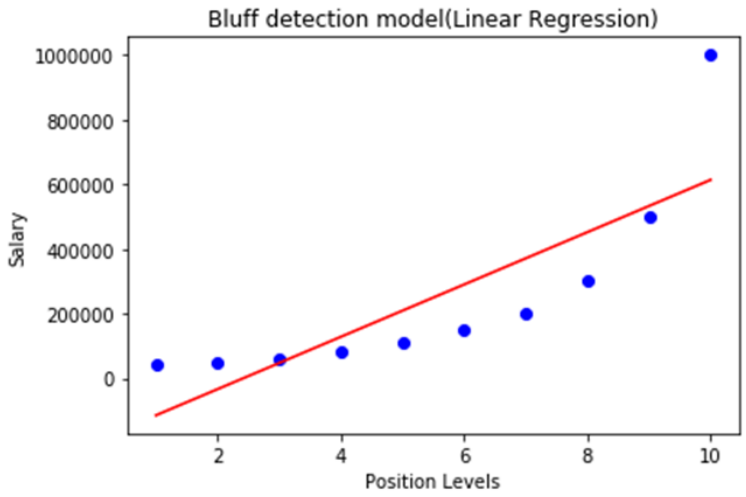
Izlaz:

Out[11]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

Vizualizacija rezultata za linearnu regresiju:

Sada ćemo vizualizovati rezultat za model linearne regresije kao što smo uradili u jednostavnoj linearnoj regresiji. Ispod je kod za to:

```
#Visulaizing the result for Linear Regression model
mtp.scatter(x,y,color="blue")
mtp.plot(x,lin_regs.predict(x), color="red")
mtp.title("Bluff detection model(Linear Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



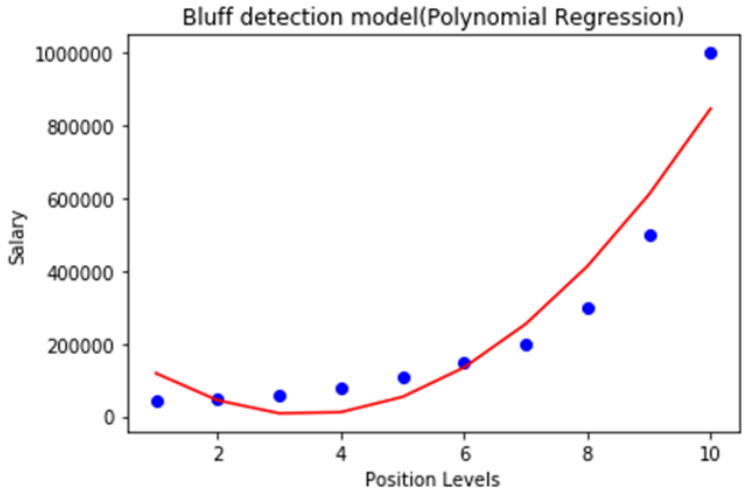
Na gornjoj izlaznoj slici možemo jasno videti da je regresijska linija toliko daleko od skupova podataka. Predviđanja su u crvenoj pravoj liniji, a plave tačke su stvarne vrednosti. Ako uzmemo u obzir ovaj izlaz da predvidi vrednost generalnog direktora, to će dati platu od cca. 600000 $, što je daleko od stvarne vrednosti.

Dakle, potreban nam je zakrivljeni model koji bi se uklopio u skup podataka osim prave linije.

Vizualizacija rezultata za polinomijalnu regresiju

Ovde ćemo vizualizovati rezultat polinomijalnog regresijskog modela, kod za koji se malo razlikuje od gore navedenog modela.

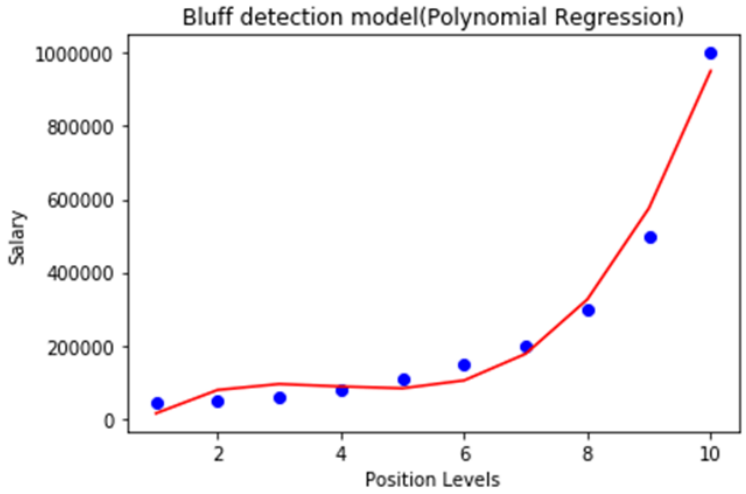
U gornjem kodu, uzeli smo lin_reg_2.predict(poly_regs.fit_transform(x), umesto x_poly, jer želimo objekat linearnog regresora da predvidi matricu polinomskih karakteristika.



Kao što možemo videti na gornjoj izlaznoj slici, predviđanja su blizu stvarnim vrednostima. Gornji zaplet će se razlikovati jer ćemo promeniti stepen.

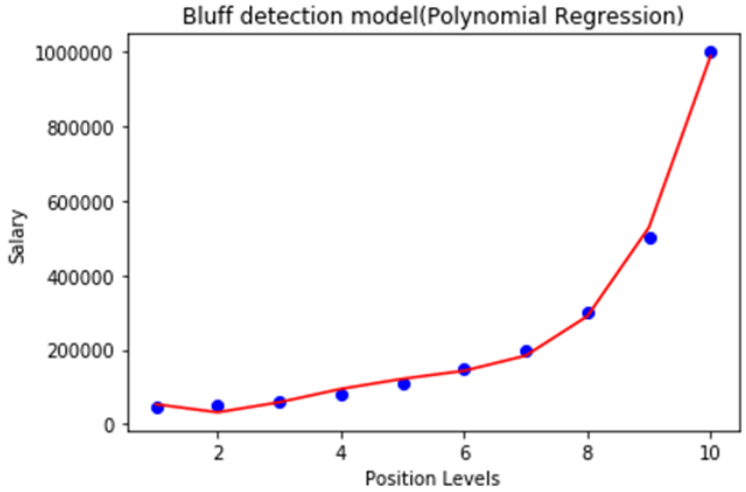
Za stepen = 3:

Ako promenimo stepen = 3, onda ćemo dati tačniji zaplet, kao što je prikazano na slici ispod.



Dakle, kao što možemo videti ovde na gornjoj izlaznoj slici, predviđena plata za nivo 6.5 je blizu 170K$-190k$, što izgleda da budući zaposleni govori istinu o svojoj plati.

Stepen = 4: Hajde da ponovo promenimo stepen u 4, i sada ćemo dobiti najtačniji zaplet. Stoga možemo dobiti preciznije rezultate povećanjem stepena polinoma.



Predviđanje konačnog rezultata sa modelom linearne regresije:

Sada ćemo predvideti konačni izlaz koristeći model linearne regresije da vidimo da li zaposleni govori istinu ili blef. Dakle, za ovo, mi ćemo koristiti **metod predict()** i da će proći vrednost 6.5. Ispod je kod za to:

```
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)
```

Izlaz:

[330378.78787879]

Predviđanje konačnog rezultata sa modelom polinomske regresije:

Sada ćemo predvideti konačni izlaz koristeći model polinomske regresije za poređenje sa linearnim modelom. Ispod je kod za to:

```
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

Izlaz:

[158862.45265153]

Kao što vidimo, predviđeni izlaz za polinomnu regresiju je [158862.45265153], što je mnogo bliže realnoj vrednosti - dakle, možemo reći da budući zaposleni kaže istina.

3.4 Višestruka linearna regresija

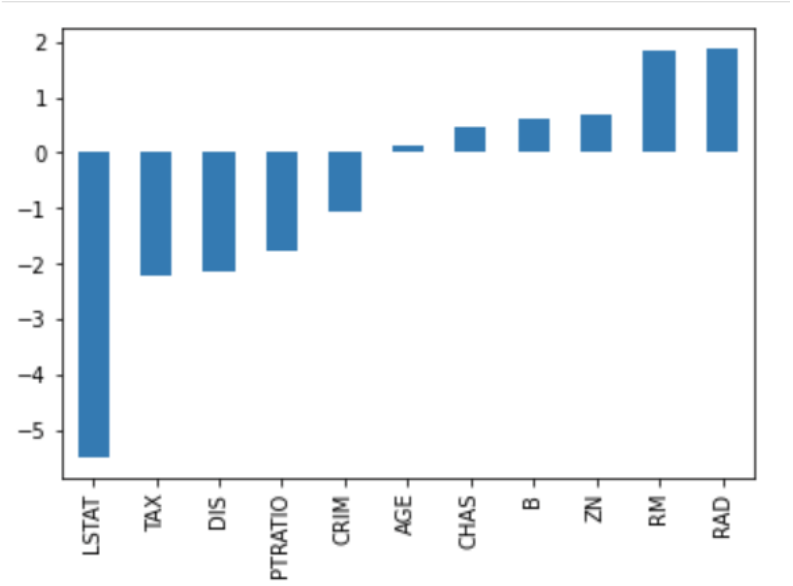
U uvodnoj priči o skupovima podataka videli smo da se koristi veći broj atributa. Ipak, u priči o linearnoj regresiji smo koristili samo jedan atribut (kvadraturu nekretnine). Verovatno se pitaš šta radimo kada imamo više atributa i da li tada možemo da primenimo model linearne regresije.

Model linearne regresije koji je prilagođen ovom scenariju se zove **višestruka** linearna regresija i ima oblik $y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_n X_n$. Nemoj da te ovaj dugački izraz zbuni - sada vrednosti $X_1, X_2, X_3, \ldots, X_n$ predstavljaju pojedinačne atribute a parametri $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ …, parametre modela. Iza ovog uopštavanja je opet ideja o linearnoj zavisnosti između pojedinačnih atributa i ciljne promenljive.

Cilj učenja je da odredimo vrednosti parametara $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ i tako steknemo predstavu o zavisnostima. Do njih dolazimo na isti način kao i kod linearne regresije koju smo upoznali (za nju kažemo i da je prosta): minimizacijom srednjekvadratne greške na skupu podataka za treniranje. Tehnika gradijentnog spusta se može uopštiti tako da odgovara i ovoj postavci zadatka i može nam pomoći da nađemo baš skup vrednosti $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ za koji je srednjekvadratna greška najmanja.

U slučaju modela linearne regresije sa jednim atributom mogli smo da zamislimo i smisao koji nose parametri β_0 i β_1: određivali su pomeraj i nagib prave koja prolazi kroz skup podataka. Tako su nam ukazivali na jačinu linearne zavisnost između ulazne i izlazne promenljive, tj. na to koliko se vrednost izlazne promenljive y promeni kada atribut x promenimo za 1. Sada, kada imamo više parametara, prirodno je da se pitamo kakvo značenje možemo da im damo. Pa, i oni modeluju istu vrstu zavisnosti. Ako zamislimo da su samo β_0 i β_2: parametri koji su različiti od nule, onda je veza između ciljne promenljive y i atributa X_2 predstavljena jednačinom $y = \beta_0 + \beta_2 X_2$, tj. linearna i isto nam ukazuje koliko će se promeniti vrednost za ciljnu promenljivu y i u kom smeru kada vrednost za X_2 promenimo za 1.

S obzirom na to da parametri za nas sumiraju znanja iz skupa podataka, u slučaju višestruke linearne regresije, veće vrednosti parametara ukazuju na veći značaj nekog atributa na vrednost ciljne promenljive. Da bismo mogli da ispratimo ovo svojstvo, vrednosti izračunatih parametara obično iscrtavamo grafikonom sa stubićima. Na donjoj slici prikazane su vrednosti parametara jednog modela koji koristi realni skup podataka za predviđanje cena nekretnina (popularni Boston skup podataka o nekretninama). Bez mnogo ulaženja u detalje ovog skupa, odmah možemo primetiti da atribut LSTAT utiče najviše, i to negativno, na vrednost ciljne promenljive, dok atributi RM i RAD imaju pozitivan uticaj, i to skoro podjednako. Grafike ovog tipa, koji mogu da nam daju neku ideju o uticaju atributa, nazivamo **graficima važnosti atributa** (eng. *feature importance graph*).



Grafik važnosti atributa višestruke regresije

Još jedan detalj koji treba na naglasimo, da te kasnije ne bi iznenadio, tiče se linearnosti. Model linearne regresije je **linearan po parametrima.** To znači da bi se i model čiji je oblik $y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$ u kojem figurišu stepeni vrednosti atributa, vodio kao linearni model. Slično je i za model $y = \beta_0 + \beta_1 log(X)$, u kojem figuriše logaritam vrednosti atributa. O ovim možda neočekivanim ulogama atributa možeš da razmišljaš kao transformacijama koje popravljaju linearnu zavisnost između atributa i ciljne promenljive.

3.5 Klasifikacija, vrste klasifikacije i matrica konfuzije

Verovao ili ne, do sada si se već mnogo puta susreo sa zadatkom klasifikacije. Kada sređuješ sobu pa razdvajaš papiriće koje ćeš da zadržiš ili bacиš ili kada razvrstavaš svoje fotografije na one sa ekskurzije, tetkinog rođendana ili izleta sa prijateljima, ti u stvari obavljaš zadatak klasifikacije: imaš na umu neke grupe i kada pogledaš papirić ili fotografiju, odlučuješ o tome kojoj grupi pripada. I mnogi programi sa kojima se susrećeš, obavljaju zadatak klasifikacije. Recimo, tvoj imejl klijent razlikuje poželjnu i nepoželjnu poštu i zahvaljujući ovoj njegovoj osobini uspevaš da izbegneš mnoge zamke i prevare na internetu. Takođe, na društvenim mrežama često dobijaš preporuke za povezivanje sa novim osobama - program koji je u pozadini društvene mreže aktivno procenjuje da li ti je neka osoba potencijalni prijatelj ili ne (obično prati prijatelje tvojih prijatelja i dobija ideje). Pošto ne sumnjamo da si ekspert u sređivanju sobe i fajlova na računaru, hajde da naučimo kako ove veštine usvajaju programi!

Vrste klasifikacije

Na samom početku je važno naglasiti da nisu sve klasifikacije iste. Zato ćemo prvo saznati koje klasifikacije postoje i šta ih to karakteriše. Primeri razvrstavanja papirića ili razvrstavanja pošte su primeri **binarne klasifikacije** zato što imamo samo dve grupe: papiriće za bacanje i papiriće za čuvanje, tj. poželjnu i nepoželjnu poštu. Grupe u svetu mašinskog učenja zovemo **klasama** pa ćemo se nadalje držati tog termina. Da bi klase mogli da razlikujemo, pridružujemo im imena koja približavaju šta zapravo one sadrže. Na primer, ”papirići za bacanje” i ”nepoželjna pošta” su dovoljno jasna imena. Imena su često određena i labelama, koje se pojavljuju u skupu podataka nad kojim se primenjuje zadatak klasifikacije.

Ako imamo više od dve klase, govorimo o zadatku **višeklasne klasifikacije.** Na primer, takav je zadatak razvrstavanja fotografija po događajima gde svaki događaj može da predstavlja jednu klasu. Možemo napraviti tri direktorijuma, tj. tri klase, dati im imena”ekskurzija”, ”tetkin rođendan”i ”izlet”, a potom svaku od fotografija pridružiti jednoj od ovih klasa tako što ćemo je staviti u odgovarajući direktorijum.

O različitim vrstama klasifikacije, možemo razmišljati i na osnovu kriterijuma pripadnosti. Na primer, jedan imejl može biti ili poželjan ili nepoželjan, ne može pripadati istovremeno i klasi poželjnih i klasi nepoželjnih imejlova. Slično je i sa fotografijama i klasama koje smo uveli. Sa druge strane, jedan novinski članak može da bude istovremeno i na temu kulture, putovanja i hrane, pa ga možemo pridružiti većem broju klasa - onoj koja predstavlja kulturu, onoj koja predstavlja putovanja i onoj koja predstavlja hranu. Kako u ovom slučaju instance imaju više obeležja, tj. labela, ovu vrstu klasifikacije nazivamo **višelabelarnom klasifikacijom**. Iako je vrlo zanimljiva i korisna, višelabelarnu klasifikaciju nećemo pokriti daljim sadržajima već ćemo se usredsrediti na binarnu i višeklasnu klasifikaciju.

Šta misliš, kojoj vrsti klasifikacije pripadaju sledeći zadaci:

- razvrstavanje đubreta za reciklažu,
- utvrđivanje ispravnosti programa,
- određivanje jezika dokumenta,
- provera validnosti bankarske transakcije,
- predlog sledeće reči pri kucanju SMS porukice?

Klasifikacija iz ugla mašinskog učenja

Kada o zadatku klasifikacije razmišljamo iz ugla mašinskog učenja, interesuju nas diskretna preslikavanja, tj. preslikavanja koja ulaznim promenljivama mogu da pridruže jednu od konačno mnogo vrednosti. Najčešće je broj klasa manji, izražen jednocifrenim brojem, ali se možeš prisetiti i skupa *ImageNet* i takmičenja klasifikacije slika u kojem se koristi 1000 klasa. Promenljive koje mogu da uzmu konačan broj vrednosti smo nazivali kategoričkim pa o klasifikaciji možemo da govorimo kao o preslikavanju koje karakteriše kategorička ciljna promenljiva.

$$F(x) = f(x) = \begin{cases} 0, & x < 0 \\ \frac{1}{2}, & 0 \leq x < 1 \\ 1, & x > 1 \end{cases}$$

Primer jedne diskretne funkcije

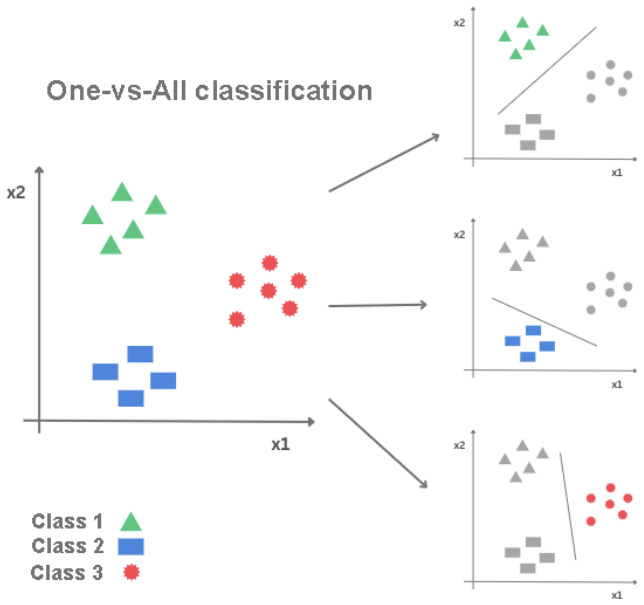
Kako je klasifikacija zadatak nadgledanog mašinskog učenja, skup podataka koji se koristi za obučavanje modela sadrži parove ulaza i očekivanih izlaza. Ulazi mogu biti slike, tekstualne poruke ili tabelarni podaci i za njihovu pripremu važe sve smernice koje smo diskutovali u lekciji o pripremama skupa podataka. Izlaz uvek predstavlja ime klase. Iako smo imena klasa uveli sa ciljem da lakše pratimo zadatak klasifikacije, kada stignemo do dela pripreme podataka, moramo i njih da transformišemo u numeričke vrednosti. Tu se možemo voditi pripremama o kojima smo diskutovali za rad sa kategoričkim atributima: preslikavanjem skupa vrednosti ili *one-hot* kodiranjem.

Ako je reč o binarnoj klasifikaciji, obično imena klasa preslikamo u vrednosti 0 i 1. Na primer, pojavljivanje imena klase ”nepoželjna pošta” zamenimo vrednošću 0, a pojavljivanje imena”poželjna pošta” vrednošću 1. Često se za instance koje imaju labelu 0 kaže da pripadaju **negativnoj klasi,** a za instance koji imaju labelu 1 da pripadaju **pozitivnoj klasi.**

Kada je u pitanju višeklasna klasifikacija, za pripremu ciljne promenljive koristimo *one-hot* kodiranje. Na primer, za zadatak razvrstavanja fotografija po događajima transformisaćemo izlaze u vektore dužine tri jer imamo tačno tri klase: ”ekskurzija”, ”tetkin rođendan”i ”izlet”. Dalje ćemo svakoj od ovih vrednosti pridružiti vektor koji na odgovarajućoj poziciji ima jedinicu, a na svim preostalim pozicijama nule. To će, redom, biti vrednosti (1, 0, 0), (0, 1, 0) i (0, 0, 1). Ovde je važno da se dosledno pridržavamo odabranog redosleda klasa.

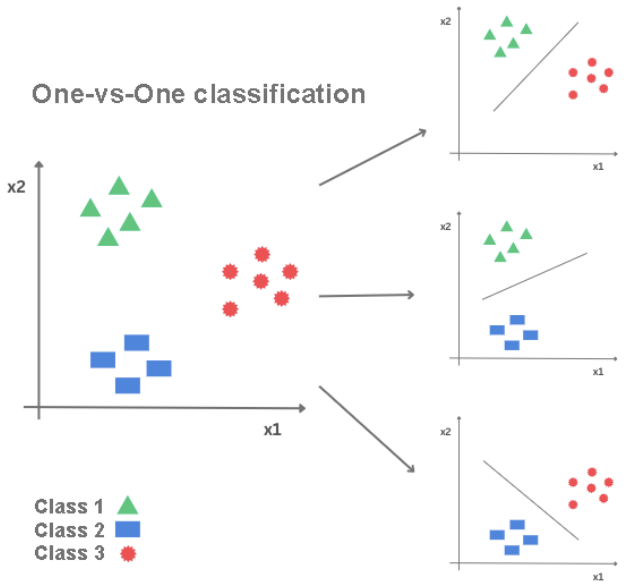
U nastavku ćemo upoznati dva algoritma koji se koriste za zadatak binarne klasifikacije. Rešavanju zadatka višeklasne klasifikacije možemo prići kroz posebno dizajnirane algoritme, ali i kroz više udruženih binarnih klasifikatora. Približićemo dve takve tehnike koje se zovu ”jedan protiv svih” i ”jedan na jedan”.

Zamislimo da raspolažemo trima klasama: crvenom, zelenom i plavom. Pristup ”jedan protiv svih” podrazumeva da treba da naučimo tri binarna klasifikatora: jedan koji razlikuje zelenu klasu od preostalih (unije crvene i plave klase), jedan koji razlikuje plavu klasu od preostalih (unije zelene i crvene klase) i jedan koji razlikuje crvenu klasu od preostalih (unije zelene i plave klase). Kada treba da klasifikujemo novu instancu, pokrećemo svaki od tri binarna klasifikatora i nad dobijenim rezultatima primenjujemo princip najveće pouzdanosti: instanca se pridružuje klasi čiji je klasifikator najsigurniji. Videćemo uskoro kako se procenjuje sigurnost klasifikatora.



Pristup „jedan protiv svih”

Zamislimo opet da raspolažemo trima klasama: crvenom, zelenom i plavom. Pristup ”jedan na jedan” podrazumeva da obučimo binarne klasifikatore koji mogu da razlikuju svaki od parova klasa: crvenu i zelenu, zelenu i plavu, i crvenu i plavu. U opštem slučaju, ako imamo n klasa, broj binarnih klasifikatora koje treba da obučimo je $\frac{n\cdot(n-1)}{2}$. Kada treba da klasifikujemo novu instancu, pokrećemo svaki od naučenih klasifikatora i nad dobijenim rezultatima primenjujemo princip većinskog glasanja: instanca se pridružuje klasi za koju glasa najveći broj klasifikatora.



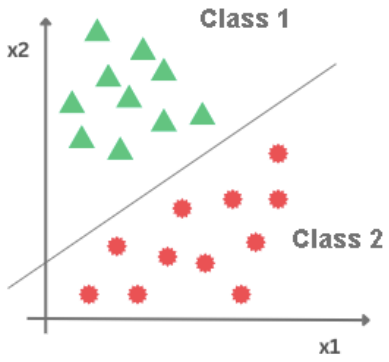
Pristup ”jedan na jedan”

3.6 Logistička regresija

Logistička regresija je poznati algoritam koji se koristi za kreiranje modela binarne klasifikacije. On nam uz informaciju o tome kojoj klasi pripada instanca izračunava i verovatnoću pripadnosti toj klasi.

Logistička regresija je poznati algoritam koji se koristi za kreiranje modela binarne klasifikacije. On nam uz informaciju o tome kojoj klasi pripada instanca izračunava i verovatnoću pripadnosti toj klasi.

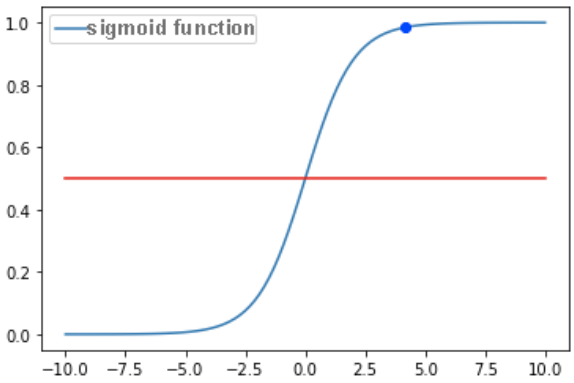
Zamislimo da raspolažemo skupom podataka sa dva atributa X_1 i X_2 i da su instance ovog skupa prikazane kao na donjoj slici. Duž x-ose je predstavljen atribut X_1, duž y-ose atribut X_2, dok boja tačaka označava klasu kojoj svaka od ovih instanci pripada. Složićeš se da bi neki linearni model koji određuje pravu u ravni mogao da nam pomogne u rešavanju zadatka klasifikacije tako što bi razdvojio klase - jedna bi se našla ispod ove prave, a druga iznad. Da bi mogli da zaključujemo na ovaj način, biće nam od koristi sigmoidna funkcija.



Sigmoidna funkcija je popularna funkcija u priči o mašinskom učenju. Određena je jednačinom

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

i njen grafik izgleda kao na donjoj slici.



Grafik sigmoidne funkcije

Odmah možemo da primetimo da ova funkcija uzima raspon vrednosti od 0 do 1. Što su vrednosti broja x manje, to je vrednost ove funkcije bliža 0 i, slično, što je vrednost broja x veća, to je vrednost sigmoidne funkcije bliže 1. Za x=0 vrednost sigmoidne funkcije je 0,5. Ukoliko ovu vrednost proglasimo pragom i uvedemo pravila:

1. ako je vrednosti sigmoidne funkcije veća ili jednaka od 0,5, pridruži x pozitivnoj klasi i
2. ako je vrednost sigmoidne funkcije manja od praga 0,5 pridruži x negativnoj klasi

dobićemo jednu funkciju podesnu za zadatak klasifikacije.

Deluje nam i da što su vrednosti broja x, veće to je odluka da se x pridruži pozitivnoj klasi uverljivija jer značajno prelazimo iznad vrednosti praga. Deluje i da što su vrednosti broja x manje da je odluka da se x pridruži negativnoj klasi uverljivija jer značajno prelazimo ispod vrednosti praga. Za vrednosti broja x, koje su oko nule, ovi argumenti su slabiji. Zato sigmoidnoj funkciji možemo pridružiti i interpretaciju verovatnoće pripadnosti nekoj klasi.

Ukoliko povežemo sigmoidnu funkciju i jednačinu linearnog modela, dobićemo jednačinu modela logističke regresije koja u opštem slučaju glasi

$$y = \sigma(X_1, X_2, \dots, X_n) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_n X_n)}}.$$

Argumenti $X_1, X_2, \dots, X_n$ označavaju atribute u skupu podataka, dok su njene vrednosti uu u rasponu od 0 do 1 i kao što smo videli smislene za zadatak klasifikacije. Ovoj jednačini možemo da pridružimo i sledeću geometrijsku interpretaciju: podaci se klasifikuju ili ispod ili iznad ”prave” koja je određena jednačinom linearne veze koju smo u startu i zamislili.

Ukoliko imamo tačno jedan atribut, ”prava” koju pominjemo je zaicta prava. Ako imamo tačno dva atributa, ”prava” je zapravo ravan u prostoru. Ako imamo više od dva atributa, ”prave” su, matematičkim jezikom, hiperravni.

Unakrsna entropija

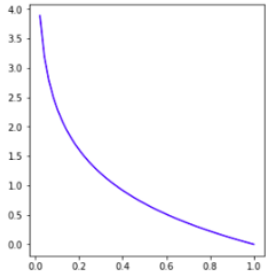
Funkcija greške koja karakteriše logističku regresiju se zove **unakrsna entropija.** Upoznajmo prvo intuiciju koja leži iza ove funkcije, a potom upoznajmo i njen matematički oblik.

Rekli smo da vrednost koju nam izračunava model logističke regresije tumačimo kao verovatnoću pripadnosti jednoj od klasa i da se vodimo pravilom da ako ta vrednost pređe prag 0,5 to protumačimo kao pripadnost pozitivnoj klasi, a ukoliko ta vrednost bude manja od 0,5 to protumačimo kao pripadnost negativnoj klasi. Ukoliko vrednost verovatnoće bude baš 0,5, to tumačimo kao pripadnost pozitivnoj klasi.

Funkciju greške izračunavamo na skupu za treniranje. U njemu za svaku instancu znamo koja su tačna obeležja pa uvek možemo da ih upoređujemo sa obeležjima koja je izračunao, tj. pridružio model.

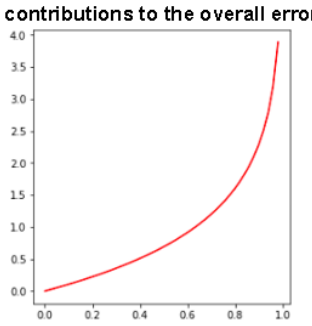
Pretpostavimo da je za tri instance koje pripadaju pozitivnoj klasi model logističke regresije redom izračunao vrednosti 0,94, 0,56 i 0,3. U prvom slučaju je vrednost bliska jedinici pa označava sigurnu odluku modela. U drugom slučaju je ova vrednost manja i bliže pragu klasifikacije, ali dovoljna za dobru odluku modela. U trećem slučaju je vrednost ispod praga pa bi navela model da pogreši. Prilikom dizajniranja

funkcije greške želimo da više kaznimo izračunavanja modela koja za pozitivne instance više odstupaju od vrednosti 1, tj. da učinimo da njihovi doprinosi ukupnoj grešci modela budu veći. Jedna takva funkcija koja zadovoljava traženo svojstvo je $-log(x)$ čiji je grafik prikazan na donjoj slici. Predznak minus nam je potreban da bi greška dobila pozitivnu vrednost jer je logaritam negativan za vrednosti argumenta funkcije koje su od 0 do 1. Na grafiku možemo i da vidimo da su vrednosti funkcije male za argumente bliže 1, tj. da su vrednosti funkcije veće za argumente koji su bliže nuli. Tako će sada, redom, doprinosi ukupnoj grešci izdvojenih instanci biti $-log(0.94) = 0.062, -log(0.56) = 0.579, -log(0.3) = 1.203$ i baš odnosa veličina koji smo želeli. Možemo ih zabeležiti i u tabeli, na način na koji smo to radili i u zadatku linearne regresije. U prvoj koloni ćemo smestiti obeležje klase (tačnu vrednost), u drugoj koloni verovatnoću p koju je izračunao modela, dok ćemo u trećoj koloni upisati vrednost $-log(p)$. Primetimo da u imenu kolone stoji $-y * log(p)$, no kako je $y = 1$ ovo je isto kao $-log(p)$.



class marker	probability p	-y*log(p)
1	0.94	0.062
1	0.56	0.579
1	0.3	1.203

Odaberimo sada tri instance negativne klase i prodiskutujmo očekivanja koja imamo od funkcije greške u njihovom slučaju. Neka su, redom, verovatnoće koje je izračunao model logističke regresije 0,03,0,48 i 0,74. Sada je u prvom slučaju vrednost modela bliska nuli pa označava sigurnu odluku o pripadnosti negativnoj klasi. U drugom slučaju ova vrednost je blizu pragu klasifikacije, ali jeispod njega, pa je opet dovoljna da model odluči o negativnoj klasi. U slučaju treće instance, vrednost verovatnoće je preko praga pa će model pogrešiti i instancu klasifikovati kao pozitivnu. Ono što očekujemo od funkcije greške za negativne instance je da njihov udeo u ukupnoj grešci bude što veći što su one dalje od nule. Jedna takva funkcija koja zadovoljava ovo svojstvo je $-log(1-p)$ i njen grafik je prikazan na slici ispod. Opet koristimo funkciju sa predznakom minus kako bi vrednost greške bila pozitivna. Možemo sada zapisati i vrednosti ove funkcije u tabeli. Sada su u prvoj koloni obeležja instanci sa vrednošću 0, u drugoj koloni verovatnoće p koje je model izračunao, dok su u poslednjoj koloni vrednosti funkcije greške $-log(1-p)$. S obzirom na to da je $y = 0$ za sve instance, obeležje u imenu kolone $-(1-y) * (1-p)$ ništa ne menja.



class marker	probability p	-(1-y)*log(1-p)
0	0.03	0.030
0	0.48	0.654
0	0.74	1.347

Ukupna vrednost funkcija unakrsne entropije se dobija kada se saberu doprinosi grešaka svih pozitivnih i svih negativnih instanci (slično kao što smo radili u zadatku linearne regresije i srednjekvadratne greške). To skraćeno zapisujemo u obliku

$$-\sum_{i=1}^{N}\left(y_i \cdot log(p_i) + (1-y_i) \cdot log(1-p_i)\right)$$

gde zapravo prvi faktor sumira doprinose grešaka pozitivnih instanci a drugi faktor doprinose grešaka negativnih instanci. Vrednost yi je tačno obeležje klase iz skupa za treniranje a pi verovatnoća koju je izračunao model logističke regresije. Ova greška se zove **unakrsna entropija** (eng. *binary crossentropy*).

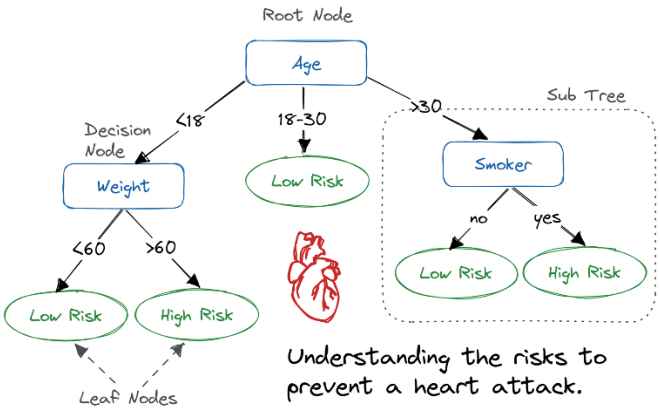
Vrednosti nepoznatih parametara β u modelu logističke regresije se pronalaze tako što se bira ona vrednost parametara za koju je funkcija unakrsne greške najmanja. Tehnika gradijentnog spusta nam može pomoći i u ovom slučaju.

3.7 Stablo odlučivanja (Decision tree)

Stabla odlučivanja su svestrani algoritmi mašinskog učenja koji mogu obavljati i zadatke klasifikacije i regresije, pa čak i zadatke sa više izlaza. Oni su veoma moćni algoritmi, sposobni da uklope složene skupove podataka.

Stablo odlučivanja je struktura stabla nalik dijagramu toka gde unutrašnji čvor predstavlja funkciju (ili atribut), grana predstavlja pravilo odlučivanja, a svaki čvor lista predstavlja ishod.

Najviši čvor u stablu odlučivanja poznat je kao korenski čvor. Uči da deli na osnovu vrednosti atributa. On particioniše stablo na rekurzivni način koji se zove rekurzivna particija. Ova struktura nalik dijagramu toka pomaže vam u donošenju odluka. To je vizualizacija poput dijagrama toka koji lako oponaša razmišljanje na ljudskom nivou. Zato su stabla odlučivanja lako razumljiva i interpretirana.



Stablo odlučivanja je tip bele kutije ML algoritma. Deli unutrašnju logiku donošenja odluka, koja nije dostupna u algoritmima crne kutije, kao što je neuronska mreža. Vreme obuke je brže u poređenju sa algoritmom neuronske mreže.

Vremenska složenost stabala odlučivanja je funkcija broja zapisa i atributa u datim podacima. Stablo odlučivanja je distribucija bez ili ne-parametarski metod koji ne zavisi od pretpostavki distribucije verovatnoće. Stabla odlučivanja mogu da rukuju visokodimenzionalnim podacima sa dobrom preciznošću.

Kako funkcioniše algoritam stabla odlučivanja?

Osnovna ideja iza bilo kog algoritma stabla odlučivanja je sledeća:

1. Izaberite najbolji atribut koristeći Attribute Selection Measures (ASM) da biste podelili zapise.
2. Napravite taj atribut čvor odluke i razbija skup podataka u manje podskupove.
3. Započnite izgradnju stabla ponavljajući ovaj proces rekurzivno za svako dete dok se jedan od uslova ne podudara:
    - Sve torke pripadaju istoj vrednosti atributa.
    - Nema više preostalih atributa.
    - Nema više slučajeva.

Mere za izbor atributa

Mera izbora atributa je heuristika za odabir kriterijuma razdvajanja koji particionira podatke na najbolji mogući način. Poznat je i kao pravila razdvajanja jer nam pomaže da odredimo tačke prekida za torke na datom čvoru. ASM obezbeđuje rang za svaku funkciju (ili atribut) objašnjavajući datu skup podataka. Atribut najboljeg rezultata će biti izabran kao atribut razdvajanja. U slučaju atributa kontinuirane vrednosti, podelne tačke za grane takođe treba da definišu. Najpopularnije mere selekcije su informacije Dobitak, Gain Ratio, i Gini indeks.

Dobitak informacija

Claude Shannon je izmislio koncept entropije, koji meri nečistoću ulaznog skupa. U fizici i matematici, entropija se naziva slučajnost ili nečistoća u sistemu. U teoriji informacija, to se odnosi na nečistoću u grupi primera. Informacioni dobitak je smanjenje entropije. Dobitak informacija izračunava razliku između

entropije pre podele i prosečne entropije nakon podele skupa podataka na osnovu datih vrednosti atributa. ID3 (Iterativni Dichotomiser) algoritam stabla odlučivanja koristi dobitak informacija.

$$Info(D) = -\sum_{i=1}^{m} pi \cdot log_2 pi$$

Gde je Pi verovatnoća da proizvoljna torka u D pripada klasi Ci.

$$Info_A(D) = \sum_{j=1}^{V} \frac{Dj}{D} XInfo(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

Gde:

- Info(D) je prosečna količina informacija potrebnih za identifikaciju oznake klase torke u D.
- |Dj|/|D| deluje kao težina JTH particije.
- InfoA(D) je očekivana informacija potrebna za klasifikaciju torke od D na osnovu podele od strane A.

Atribut A sa najvećim dobitkom informacija, Gain(A), izabran je kao atribut razdvajanja na čvoru N().

Odnos pojačanja

Dobitak informacija je pristrasan za atribut sa mnogim ishodima. To znači da preferira atribut sa velikim brojem različitih vrednosti. Na primer, razmotrite atribut sa jedinstvenim identifikatorom, kao što je customer_ID, koji ima nula info(D) zbog čiste particije. Ovo maksimizira dobijanje informacija i stvara beskorisno particioniranje.

C4.5, poboljšanje ID3, koristi proširenje za dobijanje informacija poznato kao odnos dobitka. Gain odnos bavi pitanje pristrasnosti normalizacijom informacioni dobitak koristeći Split Info. Java implementacija C4.5 algoritma je poznat kao J48, koji je dostupan u VEKA rudarstvu podataka alat.

$$Info_A(D) = \sum_{j=1}^{V} \frac{Dj}{D} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

Gde:

- $\frac{|D_j|}{|D|}$ deluje kao težina J-te particije.
- v je broj diskretnih vrednosti u atributu A.

Odnos pojačanja može se definisati kao

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

Atribut sa najvećim odnosom pojačanja je izabran kao atribut razdvajanja (Izvor).

Gini indeks

Još jedan algoritam stabla odlučivanja CART (Klasifikacija i regresijsko stablo) koristi Gini metod za kreiranje podelnih tačaka.

$$Gini(D) = 1 - \sum_{i=1}^{m} Pi^2$$

Gde je pi verovatnoća da torka u D pripada klasi Ci.

Gini indeks razmatra binarni podelu za svaki atribut. Možete izračunati ponderisanu sumu nečistoće svake particije. Ako binarna podela na atribut A podeljuje podatke D u D1 i D2, Gini indeks D je:

$$\text{Gini}_A(D) = \frac{|D_1|}{|D|}\text{Gini}(D_1) + \frac{|D_2|}{|D|}\text{Gini}(D_2)$$

U slučaju atributa diskretne vrednosti, podskup koji daje minimalni gini indeks za taj izabrani je izabran kao atribut razdvajanja. U slučaju atributa kontinuirane vrednosti, strategija je da se svaki par susednih vrednosti izabere kao moguća tačka razdvajanja, a tačka sa manjim gini indeksom je izabrana kao tačka razdvajanja.

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

Atribut sa minimalnim Gini indeksom je izabran kao atribut razdvajanja.

Kreiranje klasifikatora stabla odlučivanja u Scikit-learn

Uvoz potrebnih biblioteka

Hajde da prvo učitamo potrebne biblioteke.

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

Učitavanje podataka

Hajde da prvo učitamo potreban Pima indijski dijabetes skup podataka koristeći pande 'čitati CSV funkciju. Možete preuzeti skup podataka Kaggle da biste pratili.

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age',
'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
pima.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Izbor funkcija

Da bi razumeli performanse modela, podela skupa podataka u skup za obuku i test skup je dobra strategija.

Hajde da podelimo skup podataka pomoću funkcije train_test_split(). Potrebno je da prođe tri parametra funkcije; cilj, i test_set veličina.

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
# 70% training and 30% test
```

Kreiranje modela stabla odlučivanja

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Evaluacija modela

Hajde da procenimo koliko tačno klasifikator ili model može da predvidi vrstu sorti.

Tačnost se može izračunati upoređivanjem stvarnih vrednosti testa i predviđenih vrednosti.

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.6753246753246753

Dobili smo stopu klasifikacije od 67,53%, što se smatra dobrom tačnošću. Možete poboljšati ovu tačnost podešavanjem parametara u algoritmu stabla odlučivanja.

Vizualizacija stabala odlučivanja

Možete koristiti funkciju export_graphviz Scikit-learn za prikaz stabla u Jupiter notebooku. Za crtanje stabla, takođe morate da instalirate graphviz i pidotplus.

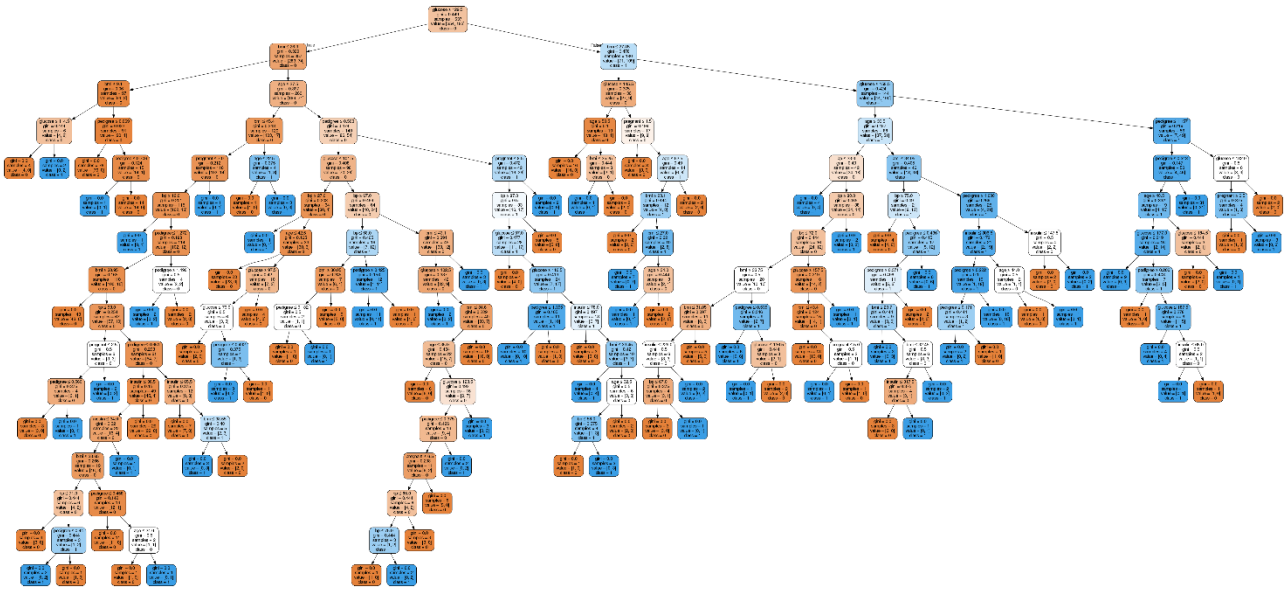
pip install graphviz

pip install pydotplus

Funkcija *export_graphviz* pretvara klasifikator stabla odlučivanja u tačkastu datoteku, a pidotplus pretvara ovu tačkastu datoteku u png ili prikazni oblik na Jupyter-u.

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names =
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



U grafikonu stabla odlučivanja, svaki unutrašnji čvor ima pravilo odluke koje deli podatke. Gini, koji se naziva Gini odnos, meri nečistoću čvora. Možete reći da je čvor čist kada svi njegovi zapisi pripadaju istoj klasi, takvi čvorovi poznati kao čvor lista.

Ovde, rezultujuće drvo je neorezano. Ovo neobrezano drvo je neobjašnjivo i nije lako razumeti. U sledećem odeljku, hajde da ga optimizujemo obrezivanjem.

Optimizacija performansi stabla odlučivanja

- **kriterijum : opciono (default = "gini") ili Izaberite meru za izbor atributa.** Ovaj parametar nam omogućava da koristimo različitu meru izbora atributa. Podržani kriterijumi su "gini" za Gini indeks i "entropija" za dobijanje informacija.
- **splitter : string, opciono (default = "best") ili Split Strategi.** Ovaj parametar nam omogućava da izaberemo strategiju razdvajanja. Podržane strategije su "najbolji" da izaberete najbolju podelu i "slučajno" da izaberete najbolju slučajnu podelu.
- **max_depth : int ili None, opciono (podrazumevano = Ništa) ili Maksimalna dubina stabla.** Maksimalna dubina drveta. Ako nema, onda se čvorovi proširuju sve dok svi listovi ne sadrže manje od min_samples_split uzoraka. Veća vrednost maksimalne dubine izaziva overfitting, a niža vrednost izaziva underfitting (Izvor).

U Scikit-learn, optimizacija klasifikatora stabla odlučivanja vrši samo pre-rezidba. Maksimalna dubina stabla može se koristiti kao kontrolna varijabla za pred-orezivanje. U sledećem primeru, možete iscrtati stablo odlučivanja na istim podacima sa max_depth=3. Osim parametara pre orezivanja, Takođe možete probati i druge mere selekcije atributa, kao što je entropija.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.7705627705627706

Pa, stopa klasifikacije porasla je na 77,05%, što je bolja tačnost od prethodnog modela.

Vizualizacija stabala odlučivanja

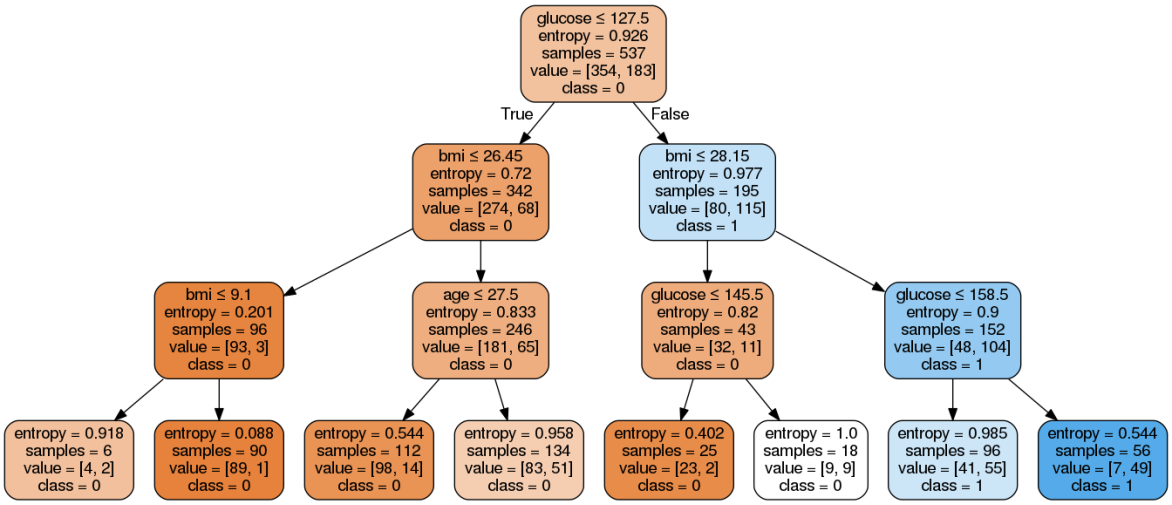
Hajde da učinimo naše stablo odlučivanja malo lakšim za razumevanje pomoću sledećeg koda:

```
from six import StringIO from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

```
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names =
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Ovde smo završili sledeće korake:

- Uvezene potrebne biblioteke.
- Kreiran StringIO objekat pod nazivom dot_data da drži tekstualnu reprezentaciju stabla odlučivanja.
- Izvozi stablo odlučivanja u format tačke pomoću funkcije export_graphviz i upisivanje izlaza u bafer dot_data.
- Kreiran je pitotplus graf objekat iz tačkastog formata reprezentacije stabla odlučivanja koje se čuva u dot_data baferu.
- Napisao je generisani grafikon u PNG datoteku pod nazivom "diabetes.png".
- Prikazana je generisana PNG slika stabla odlučivanja pomoću objekta Slika iz modula IPython.display.



Kao što vidite, ovaj orezani model je manje složen, objašnjiviji i lakši za razumevanje od prethodnog modela stabla odlučivanja.

Prednosti stable odlučivanja

- Stabla odlučivanja su jednostavna za tumačenje i vizualizaciju.
- Lako može da uhvati nelinearne obrasce.
- To zahteva manje predobrade podataka od korisnika, na primer, nema potrebe za normalizacijom kolona.

- Može se koristiti za inženjering karakteristika, kao što je predviđanje nedostajućih vrednosti, pogodnih za izbor varijabli.
- Stablo odlučivanja nema pretpostavke o distribuciji zbog neparametarske prirode algoritma. (Izvor)

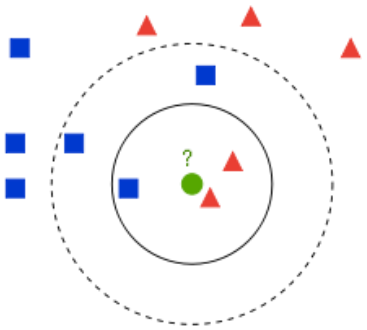
Mane stable odlučivanja

- Osetljiv na bučne podatke. Može da pretera bučne podatke.
- Mala varijacija (ili varijansa) u podacima može dovesti do različitog stabla odlučivanja. Ovo se može smanjiti pakovanjem i pojačavanjem algoritama.
- Stabla odlučivanja su pristrasna sa skupom podataka neravnoteže, pa se preporučuje da se uravnoteži skup podataka pre kreiranja stabla odlučivanja.

3.8 Algoritam k-najbližih suseda

Pomenuli smo da postoje i neparametarski modeli mašinskog učenja. Model koji se dobija primenom algoritma k-najbližih suseda je baš takav. Otkrijmo kako on funkcioniše!

Neka se naš skup za obučavanje sastoji od parova brojeva (x_1, x_2) i odgovarajućih imena klasa. Parove možemo da prikažemo kao tačke u ravni gde prva koordinata x_1 označava vrednost na x-osi, a druga koordinata x_2 vrednost na y-osi. U praksi vrednosti x_1 i x_2 uvek vežemo za neke konkretne atribute, na primer, temperaturu i vlažnost vazduha, ali sada o njima možemo da razmišljamo kao o nekim uopštenim vrednostima. Neka svaki od parova brojeva pripada jednoj od dveju klasa: crvenim trouglovima ili plavim kvadratima. Kako imamo samo dve klase, zaključuješ da je reč o binarnoj klasifikaciji. Zamisli sada da zeleni krug predstavlja novu instancu, novi par brojeva, za koji treba da odredimo kojoj klasi pripada: da li je to crveni trougao ili plavi kvadrat.



Skup za obučavanje

Algoritam k-najbližih suseda je algoritam klasifikacije koji kaže da prvo fiksiramo broj suseda (okolnih instanci) k na neku konkretnu vrednost i da zatim odredimo koliko među k-najbližih suseda ima crvenih i plavih: crveni sused je instanca koja pripada crvenoj klasi, a plavi sused instanca koji pripada plavoj klasi. Ako, na primer, broj k fiksiramo na vrednost 3, tri najbliža suseda zelenog kruga se nalaze unutar pune kružnice. To su dva crvena trougla i jedan plavi kvadrat.

Dalje, algoritam k-najbližih suseda kaže da novu instancu, tj. novi par tačaka, pridružujemo klasi brojnijeg suseda: ako su crveni susedi brojniji, za novu instancu ćemo reći da pripada crvenoj klasi i, slično, ako su plavi susedi brojniji, za novu instancu ćemo reći da pripada plavoj klasi. Ovaj vid zaključivanja možeš da razumeš i kao izreku ”s kim si, takav si” u svetu mašinskog učenja.

U našem primeru, kada je vrednost broja k fiksirana na 3, zaključićemo da zeleni krug treba da pridružimo crvenoj klasi jer imamo dva crvena suseda i jednog plavog.

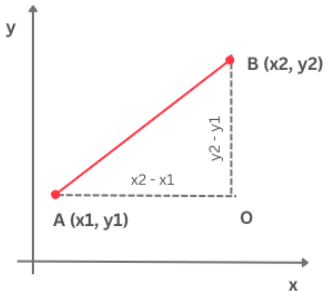
Hajde da vidimo šta će se dogoditi ako broja k fiksiramo na vrednost 5. Na slici je ovo susedstvo prikazano isprekidanom kružnicom. Kako se sada tu nalaze tri plava kvadrata i dva crvena trougla, zaključak bi bio da zeleni krug treba da pridružimo plavoj klasi.

Ova sekcija je uparena sa Jupyter sveskom 08-k-nearest_neighbors.ipynb. Da bi mogao da pratiš sadržaj dalje, klikni na link, a potom i na dugme Open in Colab da bi se sadržaj otvorio u okruženju *Google Colab.* Ukoliko sveske pregledaš na lokalnoj mašini, među sadržajima pronađi svesku sa istim imenom i pokreni je. Za detaljnije instrukcije pogledaj sekciju *Hands-on zona* i lekciju *Jupyter sveske za vežbu.*

Prateći materijal sadrži pomenuti skup tačaka i aplikaciju u kojoj možeš da ispitaš šta će se dogoditi ako odabereš neku drugu vrednost broja k. S obzirom na to da algoritam treba da odluči kojih suseda ima više, mudro je da biraš neparne vrednosti broja k.

Primetimo da osim od broja suseda k, rezultat algoritma zavisi i od načina na koji merimo udaljenosti do suseda! Da bismo pronašli najbliže susede, moramo nekako da izmerimo rastojanje do njih.

Do sada smo se na časovima matematike susretali sa rastojanjem koje se zove euklidsko. Podsetimo se, euklidsko rastojanje između tačaka A i B se računa kao dužina duži koja spaja tačke A i B. Na primer, za tačke A =(0,0) i B =(3,4) euklidsko rastojanje se računa kao $\sqrt{(3-0)^2+(4-0)^2}=5$



Euklidsko rastojanje

Postoje i mnoga druga rastojanja. Na primer, može ti biti zaniljivo Menhetn rastojanje. Za razliku od euklidskog rastojanja koje računa”hipotenuzu” trougla određenog tačkama A i B i O (ako pratimo prethodnu sliku), Menhetn rastojanje računa zbir ”kateta” ovog trougla. Za tačke A i B vrednost Menhetn rastojanja bi iznosila $|3-0|+|4-0|=7$.

Koje rastojanje ćemo odabrati zavisi od prirode zadatka i smisla koji imaju atributi sa kojima radimo. U opštem slučaju možemo da oprobamo više rastojanja i odaberemo ono za koje dobijamo najbolje rezultate. O tome ćemo još govoriti u nastavku. Važno je naglasiti da funkcija mora da zadovoljava neka određena

matematička svojstva da bi je proglasili rastojanjem pa zato ne može baš svaka funkcija da nam bude od pomoći.

Baš kao i drugi algoritmi mašinskog učenja, algoritam k-najbližih suseda se obučava nad skupom za treniranje. Zanimljivo je primetiti da se faza učenja u ovom algoritmu zapravo svodi samo na čuvanje skupa podataka. U drugim algoritmima, kao što je linearna regresija ili logistička regresija, videli smo da se u ovoj fazi izračunavaju vrednosti nekih parametara koji se pojavljuju u modelu tako što se traži minimum funkcije greške. Algoritam k-najbližih suseda nije takav. Preslikavanje koje učimo nije određeno nekom konkretnom funkcijom već samim podacima i koracima koje treba da sprovedemo. Zato je uobičajeno da modele koji imaju ovo svojstvo zovemo **neparametarskim modelima.**

Algoritam k-najbližih suseda ceo posao realizuje u toku primene, tj. odlučivanja o tome kojoj klasi pripada nova instanca. Kada treba klasifikovati novu instancu, prvo izračunamo rastojanje nove instance od svih instanci u skupu podataka za treniranje. Zatim sortiramo ova rastojanja od najmanjeg do najvećeg. Prva k rastojanja zadržavamo (jer su to rastojanja do k najbližih suseda) i biramo instance iz skupa za treniranje na koje se odnose. Dalje pratimo šta se događa u prostoru njihovih obeležja i tražimo najbrojnije obeležje, tj. najbrojniju klasu. Kao što smo videli u uvodnom primeru, novu instancu treba da pridružimo klasi koja je najbrojnija.

Ovaj algoritam je jednostavno i implementirati pa zasučimo rukave i počnimo!

Zamislićemo da radimo sa skupom podataka koji smo do sada koristili i da svaka instanca ima oblik (x1, x2, obelezje gde obelezje ima vrednost 0 za crvenu boju ili 1 za plavu.

Za merenje rastojanja između instanci koristićemo funkciju euklidsko_rastojanje, koja je definisana sledećim blokom koda:

```
def euclidean_distance(instance1, instance2):
  return np.sqrt((instance1[0]-instance2[0])**2 + (instance1[1]-instance2[1])**2)
```

Sam algoritam k-najbližih suseda je predstavljen sledećim blokom koda:

```
def kNN(k, instances, new_instance, classes={0: 'red', 1: 'blue'}):
# first, calculate the distances between the new instance and all instances in the dataset
  distances = [euclidean_distance(instance, new_instance) for instance in instances]
  # then sort the distances, extract the k smallest ones and the corresponding instances
  # declare them as neighbors
  neighbors = np.argsort(distances)[0:k]
  # then read the labels of the neighbors and count them
  neighbor_labels = [instances[neighbor][2] for neighbor in neighbors]
  label_counts = np.bincount(neighbor_labels)
  # the label of the new instance will be the label of the most frequent neighbor
  label = np.argmax(label_counts)
  return classes[label]
```


U njemu, kao što smo diskutovali, sprovodimo sledeće korake:

1. izračunavamo rastojanje od nove instance do svih instanci u skupu podataka,
2. zatim sortiramo rastojanja i izdvajamo k najmanjih,
3. instance kojima odgovaraju izdvojena rastojanja proglašavamo susedima,
4. u skupu izdvojenih suseda prebrojavamo najbrojnije,
5. zaključujemo da nova instanca pripada klasi najbrojnijeg suseda.

Funkciju kNN možeš da probaš u pratećoj svesci. Na engleskom jeziku se algoritam k-najbližih suseda zove *k-nearest-neighbours* pa se često susreće skraćeno ime *k-NN.* Otuda i ime funkcije.

Ostalo je još da naučimo kako to da odaberemo baš najbolju vrednost broja k. O tome ćemo govoriti u sledećoj lekciji.

Da li algoritam k-najbližih suseda može da se primeni u zadacima višeklasne klasifikacije?

Da, samo će biti više različitih suseda pa moramo da budemo pažljiviji prilikom prebrojavanja.

Da li algoritam k-najbližih suseda može da se primeni u regresionim zadacima?

Da. Samo će sada vrednosti ciljne promenljive najbližih suseda biti neke realne vrednosti pa nema mnogo smisla da ih prebrojavamo i tražimo najčešće. Treba da uradimo nešto što je smisleno za zadatak regresije, recimo da uprosečimo (izračunamo aritmetičku sredinu) sve vrednosti.

3.9 Hiperparametri

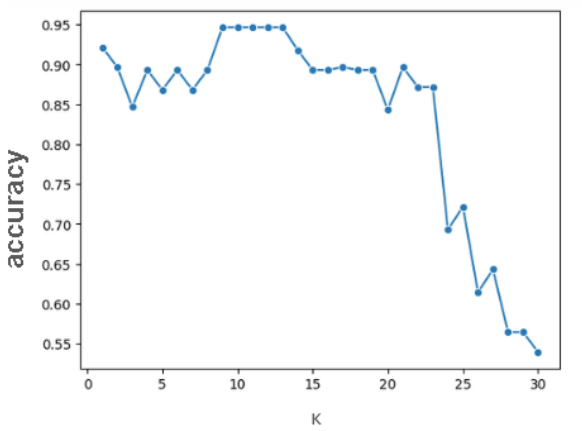
Videli smo da je u algoritmu k-najbližih suseda potrebno da unapred fiksiramo vrednost broja k, kao i da različiti izbori dovode do različitih zaključaka. Kako da znamo koju vrednost baš da odaberemo? Ovo pitanje prati i sve druge algoritme mašinskog učenja u kojima se pojavljuju neke vrednosti koje unapred treba da definišemo. Takve vrednosti zovemo **hiperparametrima** ili **metaparametrima**.

Pomenuli smo da prilikom podele skupa podataka uvek izdvajamo skup za treniranje, skup za testiranje i skup za validaciju. Skup za validaciju do sada nismo koristili. On nam je zapravo potreban kadgod u našem algoritmu učenja figurišu neki hiperparametri čiju najbolju vrednost treba da odredimo. Priča koju ćemo podeliti važi za sve algoritme, ali ćemo nastaviti da koristimo algoritam k-najbližih suseda.

Vratimo se na pitanje kako da odaberemo najbolju vrednost hiperparametra k. Prirodno je da pomislimo: probaćemo više vrednosti, na primer, sve brojeve od 1 do 10, pa ćemo odabrati najbolju vrednost! Ovo ćemo zapravo i uraditi ali ćemo veoma voditi računa o tome gde oprobavamo koliko je naš izbor dobar. Ako to budemo radili nad skupom za testiranje, ogrešićemo se o zlatno pravilo mašinskog učenja o strogoj razdvojenosti skupa za testiranje i razvoja modela: upotrebićemo skup za testiranje da odlučimo koja je najbolja vrednost hiperparametra k, a onda ćemo, kada obučimo model, opet iskoristiti skup za testiranje da ocenimo koliko je dobar! Složićeš se da to i nema baš mnogo smisla!

Korektno je da uradimo sledeće: isprobavaćemo koje vrednosti hiperparametara su najbolje na skupu za validaciju. Taj skup ne deli informacije ni sa skupom za treniranje ni sa skupom za testiranje pa će doprineti objektivnosti naših zaključaka. Sada kada smo to ustanovili, možemo da se bacimo na posao određivanja najbolje vrednosti hiperparametra k.

Za svaku od vrednosti hiperparametra k koju želimo da probamo, posebno ćemo obučiti model na skupu za treniranje i izračunati njegovu meru kvaliteta na validacionom skupu. Neka to u našem slučaju bude tačnost. Dobijene vrednosti možemo da prikažemo grafički tako što ćemo duž x-ose postaviti različite vrednosti parametra k, a duž y-ose vrednosti tačnosti. Vrednost hiperparametra k za koju dobijamo najbolju vrednost mere kvaliteta na validacionom skupu je vrednost hiperparametra koju tražimo. To se na grafiku obično vidi kao region gde su vrednosti najveće.



Prikaz tačnosti modela na skupu za validaciju

Na osnovu prethodnog grafika vidimo da su optimalne vrednosti hiperparametra k zapravo 9, 10, 11, 12 i 13 jer sve rezultiraju istom, najvećom tačnošću modela.

Slični grafici se mogu crtati i za vrednosti hiperparametra i funkciju greške. Tada duž x-ose postavljamo različite vrednosti hiperparametra, a duž y-ose vrednosti funkcije greške. Sada je važno uočiti vrednosti hiperparametra za koju je funkcija greške najmanja.

Kada u algoritmu učenja figuriše više hiperparametara cilj je pronaći najbolju kombinaciju hiperparametara. Nju, takođe, određujemo na osnovu skupa za validaciju prateći uspešnost modela i loveći kombinaciju koja daje najbolju vrednost mere kvaliteta (ili ravnopravno, prateći grešku modela i loveći kombinaciju koja daje najmanju vrednost greške). Nevolja je što ovaj postupak za veliki broj hiperparametara može da bude dosta spor i računski zahtevan: recimo, ako želimo da ispitamo 10 različitih vrednosti broja k i 3 različite funkcije rastojanja, imamo zapravo 10 x 3 = 30 različitih kombinacija pa moramo da obučimo i ocenimo 30 različitih modela.

3.10 Generalizacija, potprilagođavanje i preprilagođavanje

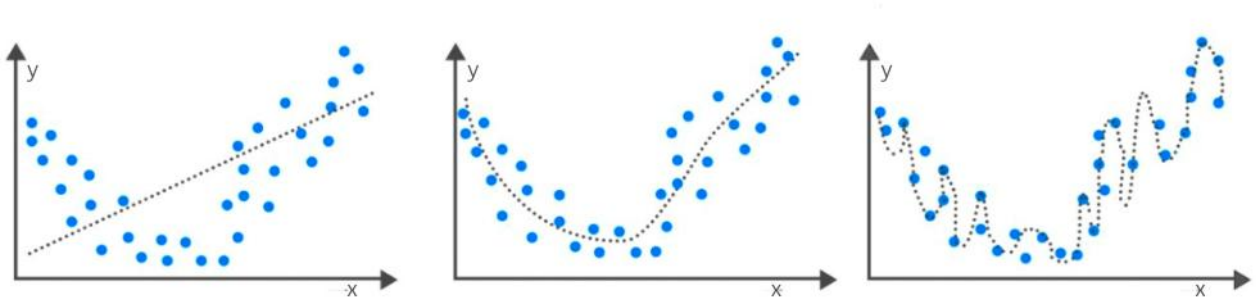
U ovoj svesci upoznaćemo pojmove generalizacije, preprilagođavanja i potprilagođavanja koji se često susreću u priči o mašinskom učenju.

Zamisli da se Pera, Ana i Luka pripremaju za kontrolni iz matematike i da svi koriste istu zbirku. Pera je zabušavao i samo je ovlaš provežbao zadatke, Ana je bila vredna i cele nedelje pažljivo i sa razumevanjem

vežbala, dok se Luka odlučio da zadatke nauči napamet. Da li možeš da pogodiš ko je najbolje prošao na kontrolnom? Naravno, Ana!

Na zbirku zadataka možemo da gledamo kao na neki apstraktni skup podataka koji se sastoji od ulaza (tekstova zadataka) i izlaza (rešenja). Model mašinskog učenja može, poput Pere, da nauči tek poneku vezu u podacima i dosta greši u praksi. Takvo svojstvo modela zovemo **potprilagođavanje** (eng. *underfitting*). Model može i da pretera sa nivoom detalja, poput Luke, i izgubi moć da se snađe sa nekim novim podacima. Takvo svojstvo modela zovemo **preprilagođavanje** (eng. *overfitting*). Najbolje bi bilo kada bi model usvojio prave informacije i mogao, kao Ana, da uspešno reši i poznate i neke nove zadatke. To svojstvo modela zovemo **dobra generalizacija** (eng. *generalisation*).

Primer potprilagođavanja i preprilagođavanja možemo da ilustrujemo i sledećom slikom. Zamisli da su duž x-ose navedene vrednosti nekog atributa, duž y-oce vrednosti ciljne promenljive i da je isprekidanom linijom prikazan model. Model na levoj slici nije baš najbolji izbor s obzirom na raspored tačaka, deluje previše jednostavno. Podaci više liče na neku"čašu" pa bi neki kvadratni model, koji ima tu formu, mogao da bude bolje rešenje. Njega možemo da vidimo na srednjoj slici. Na desnoj slici vidimo model koji dosledno prati svaku tačku u skupu podataka i koji mu je sasvim prilagođen.



Primer potprilagođavanja i preprilagođavanja

Zadatak pronalaženja optimalnog modela i balansiranja između potprilagođavanja i preprilagođavanja nije jednostavan. Srećom, oblast mašinskog učenja definiše protokole i tehnike koje možemo da koristimo da ispratimo svaku od ovih situacija. Tako, recimo, velike razlike u uspešnosti modela na skupu za treniranje i skupu za testiranje ukazuju da se model preprilagodio. To je obično posledica izbora kompleksnijeg modela nego što je potrebno (kao na gornjoj desnoj slici) ili dužeg treniranja modela. Sa druge strane, male vrednosti mera kvaliteta i na skupu za treniranje i na skupu za testiranje ukazuju da model nije naučio dovoljno na osnovu podataka, da je previše jednostavan (kao na gornjoj levoj slici) ili da mu je potrebno više atributa.

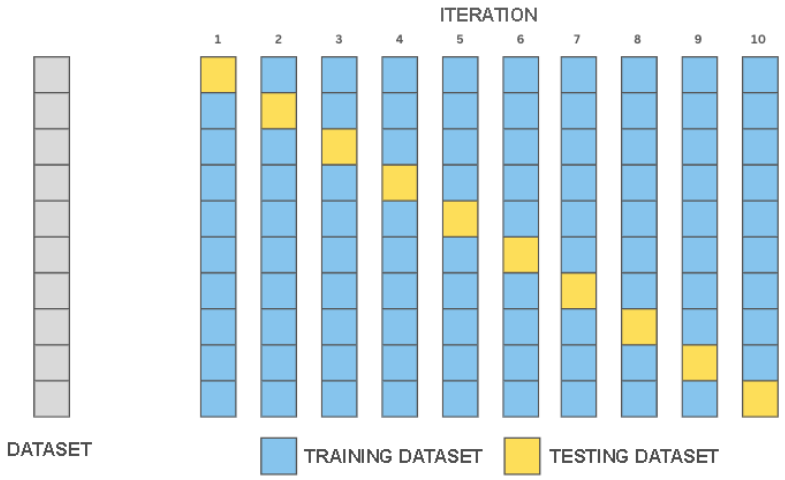
Dobra generalizacija je svojstvo koje modelima mašinskog učenja omogućava uspešnu primenu u praksi. Za njihovo obučavanje se koristi samo jedan mali deo podataka koji je dostupan, a opet, od njih očekujemo u toku primene da se dobro ponašaju i nad novim podacima koje nisu nikada susreli. Zato je važno da skupovi podataka budu reprezentativni, tj. da budu i dovoljno bogati i raznovrsni i da odgovaraju problemu koji se rešava, kao i da se pažljivo proprate moguća potprilagođavanja i preprilagođavanja modela.

3.11 Unakrsna validacija

Mnogo puta do sada smo ponovili da se podaci pre primene algoritma mašinskog učenja dele na skup za treniranje i skup za testiranje (validacioni skup uključujemo onda kada nam je to neophodno). Pomenuli smo i da je postupak podele nasumičan. Možda si se pitao da li bi neke drugačije podele, u odnosu na one koje smo odabrali, dovele do drugačijih rezultata rada modela. Možda baš za neku konkretnu podelu skupa podataka dobijamo optimističnije rezultate ili drastično lošije. I to je neka vrsta prilagođavanja.

Kadgod veličine skupova podataka i odabrani algoritmi dozvoljavaju, poželjno je zapravo izvršiti više podela polaznog skupa podataka na skup za treniranje i skup za testiranje tako da svaka instanca u skupu podataka dobije priliku da se nađe i u jednom i u drugom skupu. Jedan takav postupak koji ćemo opisati se zove **unakrsna validacija** (eng. *cross validation)*. U primeru ćemo koristiti algoritam linearne regresije, ali je priča opšta i važi za sve algoritme.

Podelimo skup podataka na 10 delova kao na donjoj slici. U prvom koraku izdvojimo prvi deo skupa za testiranje a preostalih devet delova zadržimo za treniranje. Da bi lakše mogao da pratiš, skup za testiranje je na slici obojen žutom bojom, a skupovi za treniranje plavom. Obučimo sada prvi model linearne regresije na skupu za treniranje i izračunajmo vrednost njegove srednjekvadratne greške na skupu za testiranje. Dobijenu vrednost možemo obeležiti sa MSE_1. U koraku dva izdvojimo drugi deo skupa za testiranje, a preostalih devet delova za skup za treniranje. Sada je na slici drugi deo obojen žutom bojom, a preostali delovi plavom. Ponovo obučimo model linearne regresije na skupu za treniranje (to je sada drugi model) i izračunajmo vrednost njegove srednjekvadratne greške na skupu za testiranje. Obeležimo sada ovu vrednost sa MSE_2. Nastavimo ovaj postupak sve dok ne stignemo do poslednjeg, desetog, dela: sada ćemo njega zadržati kao skup za testiranje, a preostale delove ćemo iskoristiti za treniranje modela. Nad njime ćemo obučiti deseti po redu model linearne regresije a potom i izračunati srednjekvadratnu grešku MSE_{10} na skupu za testiranje.



Unakrsna validacija sa 10 slojeva

Pošto imamo 10 različitih podela skupa podataka imamo i 10 različitih vrednosti srednjekvadratne greške. Prosek dobijenih vrednosti ($MSE_1 + MSE_2 + \ldots + MSE_{10}$)/10 zapravo najbolje ukazuje kako se ponaša naš model i pomaže nam da razrešimo dileme koje smo imali u startu u vezi sa uticajem podele na uspešnost

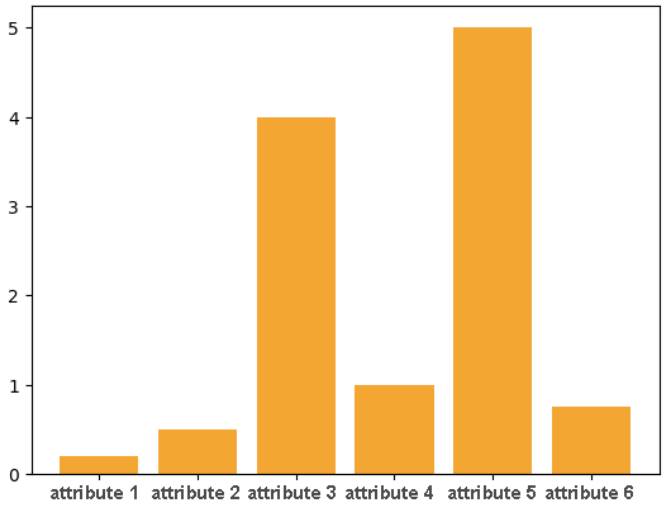
rada modela. Ono što nije baš najjasnije je koji to od 10 različitih modela kojima raspolažemo treba da odaberemo. Da li onaj čija je greška najmanja ili neki drugi? Zapravo, sada bi trebalo da obučimo novi model nad celim skupom podataka i dalje ga koristimo - njegovo ponašanje smo aproksimirali i ocenili ponašanjima svakog od 10 obučenih modela.

Opisani postupak se zove unakrsna validacija sa 10 slojeva (eng. *10 -fold cross validation)*. U praksi se koriste i podele sa 3 i 5 slojeva, a izbori zavise od veličine skupa podataka i vrste algoritama koji se koriste. Takođe, postoji i podela u kojoj broj slojeva odgovara broju instanci u skupu podataka, takozvana *izostavi jednu instancu* unakrsna validacija (eng. *leave-one -out cross validation)*.

3.12 Regularizacija

Regularizacije predstavljaju još jedan skup tehnika koje se mogu koristiti za kontrolu preprilagođavanja modela. Njihov osnovi cilj je da spreče kompleksne modele, koji nam pomažu da naučimo bogatiji skup zavisnosti u podacima, da se previše prilagode.

Regularizaciju ćemo uvesti na primeru modela linearne regresije. Pretpostavimo da smo obučili model i da smo dobili vrednosti parametara čiji grafički prikaz izgleda kao na slici.



Parametri koji su po svojoj (apsolutnoj) vrednosti najveći su ujedno i najznačajniji za predikcije modela. Na slici su to parametri koji odgovaraju atributima 3 i 5 i njihove vrednosti su, kao što možemo da primetimo, znatno veće od vrednosti preostalih parametara. U tom smislu, ovi atributi mogu da zanemare uticaj preostalih atributa na vrednosti predikcija pa ovo ponašanje modela možemo da protumačimo i kao vid preprilagođavanja podacima.

Zato je poželjno, u nekoj meri, ograničiti vrednosti parametara - želimo da model nauči parametre i da oni oslikavaju svojstva podataka, ali želimo i da pratimo njihovu vrednost kako bi predupredili preprilagođavanje. Ova tehnika se zove **regularizacija** (eng. *regularisation*). U kontekstu linearne regresije to možemo uraditi dodavanjem sume kvadrata parametara srednjekvadratnoj grešci modela:

$$\frac{1}{N}\sum_{i=1}^{N}\left(y_i - (\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)\right)^2 + \lambda(\beta_1^2 + \beta_2^2 + \cdots + \beta_n^2)$$

Vrednost λ koja figuriše u izrazu je hiperparametar kojim utičemo na jačinu regularizacije. Ako je njegova vrednost 0, regularizacija neće imati nikakvog efekta. Zadavanjem nekih ne-nula vrednosti balansiramo učenje određeno srednjekvadratnom greškom i preprilagođavanje mereno vrednostima sume kvadrata parametara. Kvadrati su tu iz tehničkih razloga, prvo da bi onemogućili da se vrednosti koeficijenata između sebe potiru, a potom i da bi se očuvala svojstva funkcije greške za primenu algoritma optimizacije. Ovako prošireni oblik linearne regresije dopunjen regularizacionim članom naziva se **grebena linearna regresija** (eng. *ridge regression*).

Nešto kasnije ćemo se vratiti na priču o regularizaciji kada budemo uveli neuronske mreže. One su veoma kompleksni modeli pa se često mogu preprilagoditi podacima. Videćemo i kako to možemo da pratimo.

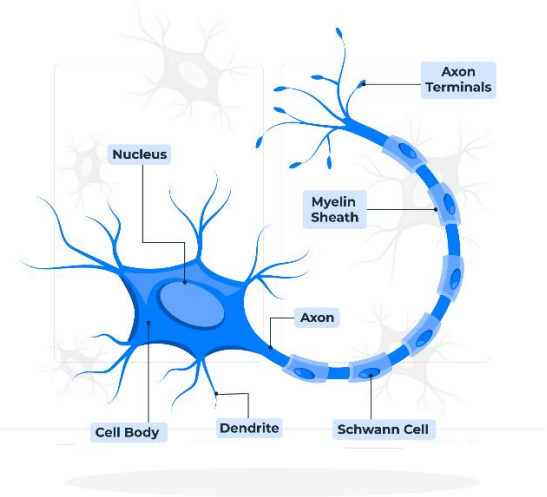
4.1 NEURONSKE MREŽE

Dobrodošli na temu **neuronskih mreža**! Neuronske mreže su ključna komponenta dubokog učenja, podskup mašinskog učenja koji oponaša strukturu i funkciju ljudskog mozga. U ovom odeljku ćete naučiti o osnovnim elementima neuronskih mreža, uključujući neurone, slojeve i aktivacijske funkcije. Istražićemo različite tipove arhitektura neuronskih mreža, kao što su konvolucijske i rekurentne neuronske mreže, i njihove primene u rešavanju složenih problema. Takođe ćete steći praktično iskustvo u obuci i testiranju neuronskih mreža koristeći popularne alate i okvire.

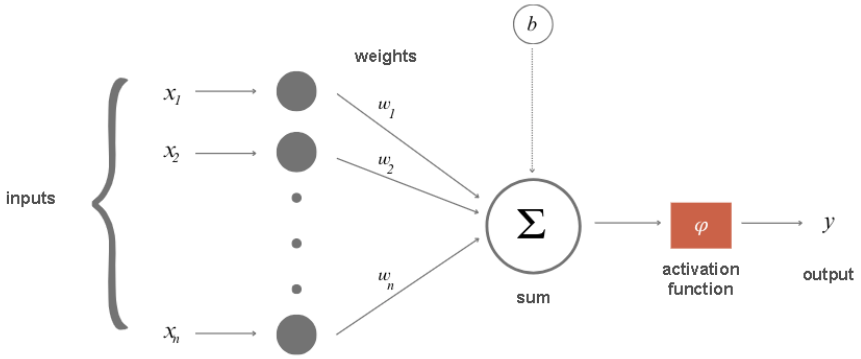
4.1 Neuronske mreže

U ovoj lekciji ćemo upoznati neuronske mreže, posebnu grupu algoritama mašinskog učenja. Njima dugujemo mnoge zanimljive proboje u svetu veštačke inteligencije.

Sa časova biologije ti je poznato da je ćelija osnovna jedinica građe i funkcije svih živih bića. Iz ugla veštačke inteligencije i učenja, najzanimljivije su nam ćelije mozga. One se zovu neuroni. Neuroni se sastoje iz tela u kojem je jezgro i dužih i kraćih nastavaka, koji se zovu aksoni i dendriti. Nastavci neuronima omogućavaju da se povežu sa drugim neuronima. Te tačke povezivanja neurona se nazivaju sinapsama. One omogućavaju da se signali, tj. električni impulsi koji generiše jedan neuron, prenesu do drugog neurona. Zanimljivo je da jedan neuron može biti povezan sa milionima drugih neurona. To znači da on prima i obrađuje signale koji stižu od mnoštva drugih neurona i na osnovu svojih internih mehanizama fino proračunava signal koji dalje šalje drugim neuronima. Uobičajeno je da se ovo stanje naziva stanje aktivacije neurona. Ono traje tek delić sekunde, ali omogućava da se izvrše suptilne kalkulacije i generiše signal koji se prenosi kroz ceo nervni sistem.



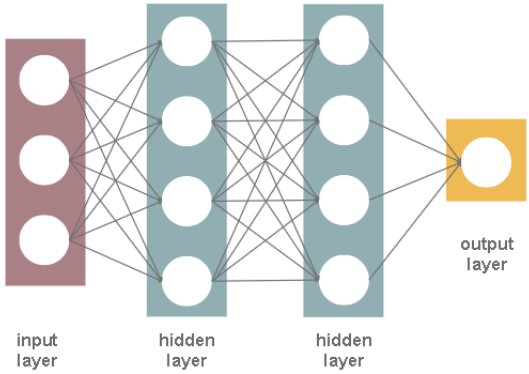
Neuron koji susrećemo u veštačkoj inteligenciji je matematička apstrakcija neurona mozga. Njega opisujemo funkcijom više promenljivih $f(x_1, x_2, \dots, x_n)$ gde svaka od promenljivih $x_1, x_2, \dots, x_n$ odgovara po jednom signalu koji stiže do neurona. Kako nisu svi signali podjednako važni za aktivnosti neurona, pridružuju im se težine $w_1, w_2, \dots, w_n$ koje treba da ukažu na njihov značaj. Veće vrednosti ovih brojeva ukazuju da je signal važniji, a manje vrednosti da je signal manje važan. Tako, ukupna stimulacija neurona odgovara težinskoj sumi $w_1x_1 + w_2x_2 + \cdots + w_nx_n$. Da bi moglo da se utiče na dodatna ponašanja neurona, ovoj sumi se dodaje i jedan slobodan član b, tako da ukupna stimulacija neurona zapravo iznosi $w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$. Ona se dalje prosleđuje takozvanoj aktivacionoj funkciji φ, koja ima zadatak da izračuna izlaz neurona. U zavisnosti od izbora aktivacione funkcije zavisiće i vrednosti izlaza koje se dobijaju. Ako sada sve sistematično zapišemo, dobijamo da je za primljene signale $x_1, x_2, \dots, x_n$ izlaz neurona $y = \varphi(w\_1x\_1 + w\_2x\_2 + \cdots + w\_nx\_n + b$. Postupak koji smo opisali možeš da ispratiš i na donjoj ilustraciji.



Matematička apstrakcija neurona

Približimo dodatno smisao parametra b. Prirodni neuron karakteriše takozvani prag aktivacije - ukoliko je ukupan signal koji neuron primi veći od vrednosti praga aktivacije, on se aktivira, obrađuje signal i prosleđuje rezultat obrade dalje drugim neuronima. Sličnu ulogu u matematičkom modelu neurona ima i parametar b. Ukoliko je ukupni signal veći od praga aktivacije b, tj. ako je $w1x1 + w2x2 + \cdots + wnxn > b$, neuron će se aktivirati. Stoga nam parametar b ostavlja mogućnost da utičemo na dodatna ponašanja neurona. Izraz $w_1x_1 + w_2x_2 + \cdots + w_nx_n > b$ se može zapisati i kao $w_1x_1 + w_2x_2 + \cdots + w_nx_n - b > 0$, pa je u tom smislu parametar b i sastavni deo sume.

Kada neurone povežemo među sobom, dobijamo **neuronsku mrežu** (eng. *neural network*). Neuronska mreža se po pravilu sastoji od **slojeva** (eng. *layer*), posebno udruženih grupa neurona.



Slojevi neuronske mreže

Ulazni sloj (eng. *input layer)* je sloj koji se nalazi na ulazu neuronske mreže. Ulazne signale x1, x2,..., xn ovog sloja povezujemo sa vrednostima atributa koje imamo u skupu podataka i tako prilazimo praktičnoj primeni neuronskih mreža. Na primer, ako raspolažemo skupom podataka u kojem se nalaze tri atributa, temperatura, vlažnost vazduha i atmosferski pritisak, ulazni sloj će imati tri neurona: prvi će odgovarati prvom atributu, temperaturi, drugi će odgovarati drugom atributu, vlažnosti vazduha, a treći neuron trećem atributu, tj. atmosferskom pritisku. Za jednu konkretnu instancu skupa podataka sa vrednostima temperature, vlažnosti vazduha i atmosferskog pritiska koji iznose, redom, 19℃, 77% i 1011,2 mb imaćemo vrednosti signala x1 = 19, x2 = 77 i x3 = 1011,2. U duhu prethodne priče, prvi neuron ulaznog sloja prima i obrađuje samo signal x_1 i to tako što ga propušta bez bilo kakve modifikacije (to je moguće za izbor aktivacione funkcije φ(x)= x i vrednost w_1 = 1 i b = 0). Slično važi i za preostala dva neurona i njihove signale x_2 i x_3. To bi značilo da nam ulazni sloj omogućava da podaci uđu u mrežu.

Izlazni sloj (eng. *output layer)* je sloj koji se nalazi na izlazu neuronske mreže. Kao što naslućuješ, on nam omogućava da očitamo rezultate koje je neuronska mreža izračunala za nas. U zavisnosti od zadatka koji se rešava, zavisiće i broj neurona koji se nalazi u ovom sloju.

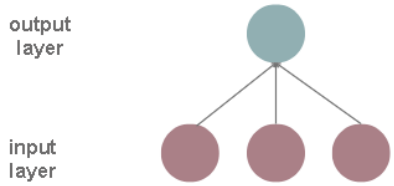
U zadacima regresije, pošto očekujemo jednu brojčanu vrednost kao rezultat (količinu padavina ili nešto slično), dovoljan nam je jedan neuron. Njegov izlaz treba da odgovara predikciji koju očekujemo. Za zadatak klasifikacije razmotrimo posebno binarnu klasifikaciju i višeklasnu klasifikaciju. Kako kod binarne klasifikacije očekujemo dve vrednosti, 0 ili 1, možda će ti prva pomisao biti da su nam potrebna dva neurona. Ipak, ako bolje razmisliš, primetićeš da je dovoljan čak i jedan neuron: ako njegov izlaz pređe neki prag, neku unapred definisanu vrednost, to možemo voditi kao rezultat 1, ili, u suprotnom, kao rezultat 0. U slučaju višeklasne klasifikacije možemo da imamo više klasa pa je praktično da za svaku klasu uvedemo po jedan neuron.

Složit ćete se da u zadatku klasifikacije više klasa očekujemo da svi izlazi neurona izlaznog sloja budu 0, osim onog koji ima vrednost 1 - tako da ćemo tačno znati koja je to klasa.

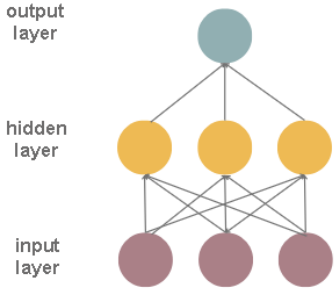
Slojeve neuronske mreže koji se nalaze između ulaznog i izlaznog sloja nazivamo **skrivenim slojevima** (eng. *hidden layers*). Uobičajeno je da se neuronske mreže koje imaju više od jednog skrivenog sloja nazivaju **dubokim neuronskim mrežama** (eng. *deep neural networks*). Odatle dolazi i **duboko učenje**

(eng. *deep learning)* za oblast mašinskog učenja koja ih izučava i ime **plitko učenje** (eng. *shallow learning*) za klasičnije forme učenja.

Razmotrimo sada šta smo zapravo dobili uvođenjem neurona i neuronskih mreža. Pretpostavimo da imamo tri atributa $x_1, x_2 i x_3$. Linearnu zavisnost između atributa i ciljne promenljive smo matematički opisivali jednačinom $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$. Ukoliko umesto parametara β zapišemo w a umesto β0 zapišemo b i prebacimo ga na kraj, dobijamo zapravo težinsku sumu $w_1 x_1 + w_2 x_2 + w_3 x_3 + b$ koju izračunava jedan neuron za signale koje prima. To znači da, kada ne bi bilo aktivacione funkcije φ i neuron bi modelovao linearnu zavisnost između atributa (signala) i izlaza. Ovo možemo grafički prikazati i mrežom koja se sastoji samo od ulaznog sloja sa tri neurona i izlaznog sloja sa jednim neuronom, kao na donjoj slici.



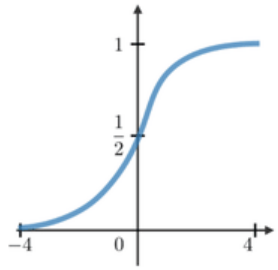
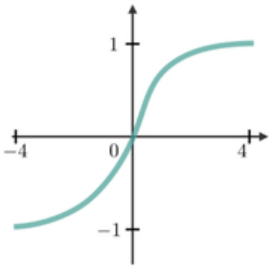
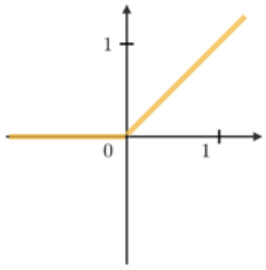
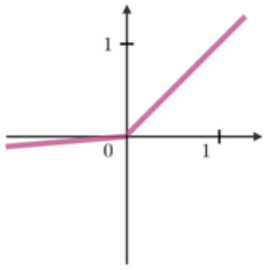
Ako aktivaciona funkcija ne bi postojala, da li bi iz ugla modelovanja zavisnosti nešto promenilo dodavanje novog skrivenog sloja? Neka to bude sloj žute boje na sledećoj slici.



Sada svaki neuron skrivenog sloja izračunava neku linearnu kombinaciju atributa, a neuron izlaznog sloja neku linearnu kombinaciju vrednosti skrivenog sloja. To bi značilo da naš neuron izlaznog sloja opet izračunava neku linearnu kombinaciju atributa i da se nismo mnogo pomerili od predstavljanja nekih složenijih zavisnosti između atributa i izlaza. Dodatno, ne bismo se pomerili čak ni dodavanjem 100 skrivenih slojeva - uvek bismo modelovali linearnu zavisnost.

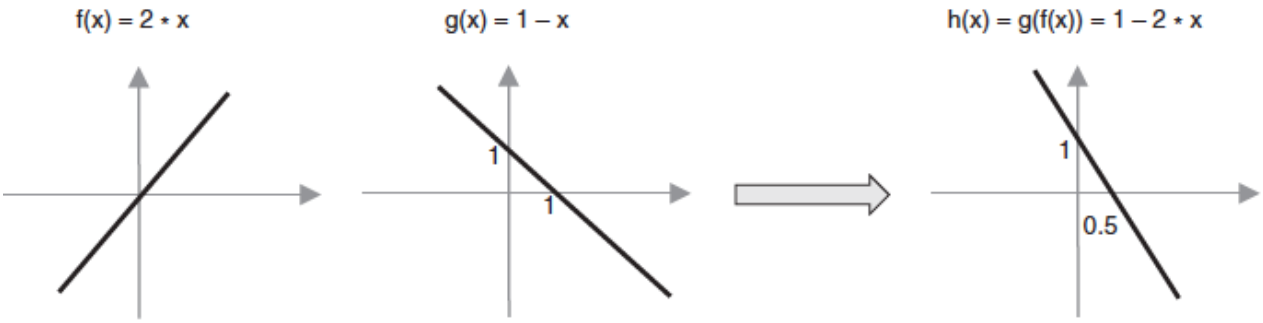
Zato uključivanje aktivacione funkcije u izračunavanja neurona značajno menja skup mogućnosti koje imamo. Ukoliko iskoristimo neku nelinearnu aktivacionu funkciju, moći ćemo da modelujemo i neke nelinearne zavisnosti između atributa i ciljne promenljive. Tako postojanje nelinearne aktivacione funkcije u skrivenom sloju iz prethodnog primera omogućava da neuron izlaznog sloja sada izračunava neku nelinearnu kombinaciju atributa. U ovom svetlu, dodavanje novih slojeva ima mnogo više smisla. Kombinujući nelinearnosti većeg broja slojeva možemo da modelujemo kompleksne zavisnosti između atributa i izlaza.

Da bi se sve kockice uklopile, ostaje još da prodiskutujemo koje su to nelinearne aktivacione funkcije koje su popularne u mašinskom učenju. To su sigmoidna funkcija koju smo upoznali u priči o logističkoj regresiji, hiperbolički tenges, ispravljena linearna jedinica (eng. *rectified linear unit, ReLU)* i nakošena ispravljena linearna jedinica (eng. *leaky rectified linear unit, leaky ReLU)*. Formule po kojima se ove funkcije izračunavaju i njihovi grafici prikazani su na donjoj slici. Kao što možeš da primetiš, ove funkcije zaista nisu linearne - njihovi grafici nisu prave.

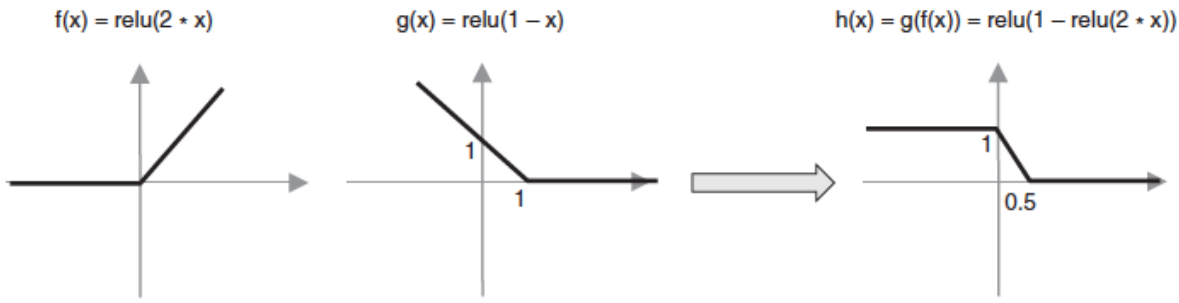
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Najčešći izbori aktivacionih funkcija

Da bismo upotpunili priču o kombinovanju različitih aktivacionih funkcija, posmatrajmo funkcije f(x)= 2x i g(x)= 1 − x. Možemo da primetimo da su obe funkcije linearne funkcije jedne promenljive. Njihovim kombinovanjem, kompozicijom funkcija, dobijamo funkciju g(f(x))= 1 − 2x, koja je takođe linearna funkcija jedne promenljive. Grafike sve tri funkcije možemo da vidimo i na donjoj slici.



Posmatrajmo sada funkcije f(x)= ReLU(2x) i g(x)= ReLU(1 − x), koje se od prethodnih funkcija razliku po tome što u njima figuriše aktivaciona funkcija ispravljena linearna jedinica. Zato su obe funkcije nelinearne. Njihovim kombinovanjem, tj. njihovom kompozicijom, dobijamo funkciju g(f(x))= ReLU(1 − ReLU(2x)), koja je takođe nelinearna i koja ima novi ”oblik”: omogućava nam da izrazimo nešto drugačiju zavisnost između ulazne promenljive i izlaza.



Izbor odgovarajuće aktivacione funkcije zavisi od prirode zadatka i nekih svojstava koje neuronska mreža treba da ima u toku obučavanja. Kako se to radi, objasnićemo u sledećoj lekciji.

4.2 Obučavanje neuronskih mreža

Kao što smo videli, svaki neuron neuronske mreže je na neki način povezan sa drugim neuronima u mreži. Te veze su opisane težinama w koje zapravo predstavljaju parametre neuronske mreže koje je potrebno naučiti u toku treniranja. Broj parametara u neuronskoj mreži je uglavnom veliki. Recimo, za potpuno povezanu neuronsku mrežu koja ima 5 neurona u ulaznom sloju, jedan skriveni sloj sa 10 neurona i izlazni sloj sa 3 neurona, broj parametara koje treba naučiti je 93. U praksi, neuronske mreže imaju hiljade i milione parametara, čak i milijarde! Zato je potrebna velika količina podataka za njihovo obučavanje.

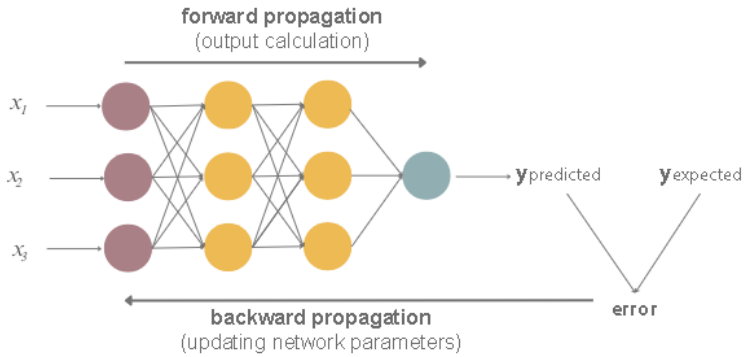
Da li je zaista broj parametara u potpuno povezanoj neuronskoj mrežu koja ima 5 neurona u ulaznom sloju, jedan skriveni sloj sa 10 neurona i izlazni sloj sa 3 neurona 93?

Ulazni sloj ne sadrži nepoznate parametre - on samo propušta podatke u mrežu. Svaki neuron skrivenog sloja je povezan sa svakim neuronom ulaznog sloja, što znači da svaka od tih veza ima po 5 parametara i jedan slobodan član. To je ukupno 10x5 + 10x1 = 60 parametara. Svaki neuron izlaznog sloja je povezan sa svakim neuronom skrivenog sloja što znači da svaka od tih veza ima po 10 parametara i jedan slobodni član. To je ukupno 3x10 + 3x1 = 33 parametra. Kada saberemo obe vrednosti, to je 93 parametra.

Neuronske mreže se poput drugih modela obučavaju na skupu za treniranje i ocenjuju na skupu za testiranje. Pošto neuronske mreže predstavljaju kompleksne modele koji mogu da nauče složene zavisnosti između atributa i izlaza, one se lako mogu preprilagoditi podacima. Zato u toku obučavanja mreža uvek koristimo i validacioni skup. On nam pomaže da finije ispratimo tok treniranja i ranije primetimo preprilagođavanje i druga nepoželjna svojstva modela.

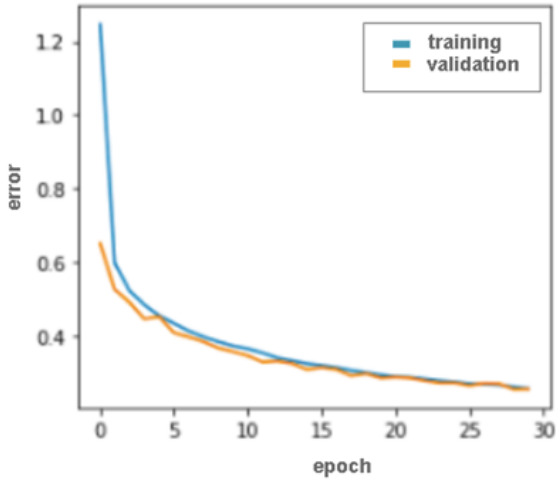
U uvodu kursa smo rekli da se nepoznati parametri modela određuju tako što se definiše funkcija greške, a zatim primeni neka tehnika optimizacije (u koje spada i gradijentni spust) sa ciljem da se pronađu one vrednosti parametra za koje je funkcija greške najmanja. Ovaj protokol prati i priču o neuronskim mrežama s tim što se vrednosti greške ne izračunavaju za pojedinačne instance već za grupe instanci. Motivacija za ovakav dizajn je, pre svega, rad sa velikom količinom podataka i potreba da se ceo proces paralelizuje i ubrza. Zato se prvo svi podaci u skupu za treniranje podele u **paketiće** (eng. *batch*) jednakih veličina. Paketići se dalje, jedan po jedan, propuštaju kroz mrežu i za njih se izračunava vrednost funkcije greške tako što se uporede očekivane i dobijene vrednosti ciljne promenljive. Zatim se srazmerno svojim

doprinosima vrednosti greške parametri neuronske mreže ažuriraju prolaskom kroz mrežu unazad. Opisani postupak ažuriranja parametara mreže se naziva **propagacija unazad** (eng. *backpropagation*) i omogućava nam da u iteracijama profinimo vrednosti parametara i stignemo do optimalnih vrednosti parametara. Njih, inače, u startu nasumično inicijalizujemo.

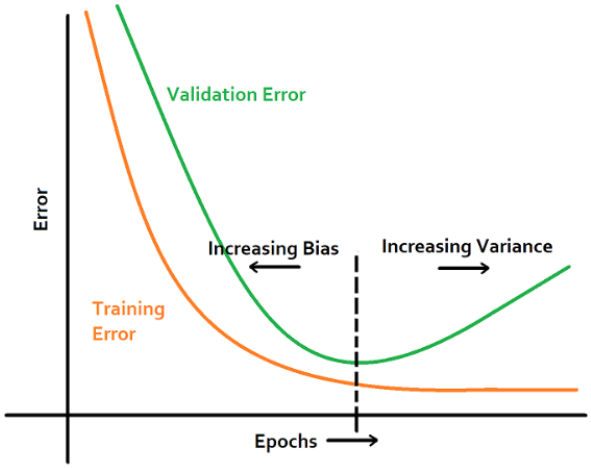


Jedan prolazak kroz ceo skup podataka, tj. jedna obrada svih paketića skupa za treniranje, naziva se **epoha.** Neuronske mreže se treniraju u više epoha. Nakon što se završi jedna epoha, podaci se ”promešaju”, potom ponovo podele u paketiće i propuste kroz mrežu. U koliko će se epoha trenirati model zavisi od uspešnosti treniranja i raspoloživih resursa. Zbog rada sa velikom količinom podataka, mrežama je potreban specijalizovan hardver koji može da paralelizuje izračunavanja (na primer, grafičke kartice ili tenzorske kartice) pa je samo treniranje mreža često i skupo i dugotrajno.

Ovakav način treniranja mreže kroz epohe nam omogućava da finije pratimo tok treniranja. Na kraju svake epohe se po pravilu izračuna greška modela na skupu za treniranje i greška modela na skupu za validaciju. Zatim se ove dve vrednosti prikažu na grafiku koji duž x-ose prikazuje redni broj epohe, a duž y-ose vrednost greške. Jedan takav grafik možeš da vidiš na donjoj slici. Dobro treniranje karakteriše uporedni pad ovih vrednosti do zadovoljavajuće vrednosti greške - što smo bliži nuli, to je model bolji. Podsetimo se da je ovaj zaključak utemeljen na tome što se u validacionom skupu nalaze podaci koji su razdvojeni od skupa za treniranje i koje mreža vidi prvi put.



Ukoliko primetimo da se vrednosti funkcije greške na skupu za treniranje smanjuju a na validacionom skupu rastu, zaključujemo da se model preprilagođava i zaustavljamo obučavanje. Dalje imamo dve opcije. Ako su vrednosti funkcije greške modela u epohi pre primećenog preprilagođavanja modela bile zadovoljavajuće, možemo da zadržimo tu verziju modela za dalje testiranje na skupu za testiranje (obično se u toku treniranja mreže sačuva nekoliko verzija modela sa idejom da se iskoriste za ovakve svrhe ili da se iskoriste ukoliko treba zaustaviti pa nastaviti proces obučavanja). U suprotnom, moramo da oprobamo nešto drugačiju arhitekturu mreže ili nešto drugačiji skup njenih hiperparametara. S obzirom na to da svaki sloj mreže ima svoja podešavanja (broj neurona, aktivacionu funkciju, inicijalni skup parametara), da slojeve možemo povezati na različite načine, da uporedo moramo pratiti sva podešavanja optimizacionog algoritma, recimo gradijentnog spusta i njegovog koraka učenja, i da treba zadovoljiti i neka očekivanja u pogledu mera kvaliteta, treniranje mreže je izazovan i kompleksan zadatak. Zato se za njega kaže da predstavlja *umetnost treniranja*.



Praćenje preprilagođavanja neuronske mreže na osnovu grafika vrednosti funkcije greške na skupu za treniranje i skupu za validaciju

4.3 Konvolutivne neuronske mreže (CNN)

Učenje reprezentacije podataka

Neuronske mreže nam mogu pomoći da izdvojimo neke apstraktne atribute u podacima i naučimo reprezentacije koje su podesne za rešavanje zadataka.

U primerima koje smo do sada koristili najčešće smo se oslanjali na postojanje nekog skupa atributa u skupu podataka. Uistinu govoreći, veliki broj domena generiše baš podatke koji su ovog oblika, sa atributima u kolonama i instancama u pojedinačnim redovima. Kao što smo videli u uvodnom delu u priči o pripremi podataka, čak i kada raspolažemo ovakvim atributima nije baš najintuitivnije odlučiti koje atribute treba da odaberemo za kreiranje modela. To nas je stavljalo u poziciju da oprobavamo različite kombinacije ili osmišljavamo tehnike koje nam mogu pomoći u selekciji atributa. Usled kompleksnosti funkcija koje modeluju, neuronske mreže se mogu pohvaliti svojstvom da mogu lepo da nauče da filtriraju i grupišu atribute koji su bitni.

Ovo svojstvo neuronskih mreža je posebno važno u radu sa podacima koji nisu tabelarni - pokrenuli smo već mnogo puta pitanje kako predstaviti, na primer, slike, tekstualne podatke ili audio-zapise. Iako imamo znanja o ovim formatima, teško nam je da opišemo šta tačno oni sadrže na neki koncizan i upotrebljiv način. To nas je, između ostalog, i motivisalo za primenu paradigme programiranja vođenog podacima. Neuronske mreže mogu (u to ćemo se uskoro uveriti) na osnovu podataka u izvornom obliku da nauče neke apstraktne atribute koji su korisni za uspešno rešavanje zadataka.

U nastavku ćemo upoznati konvolutivne neuronske mreže koje se koriste primarno u radu sa slikama i videom i za učenje vizuelnih atributa ulaza, a zatim i rekurentne neuronske mreže i transformere, tipove neuronskih mreža koje se koriste za učenje atributa sekvencijalnih podataka kao što su tekst ili zvuk.

Konvolutivne neuronske mreže

Konvolutivne neuronske mreže su vrsta neuronskih mreža koje se primarno koriste u oblasti računarskog vida za rad sa slikama i video-sadržajima.

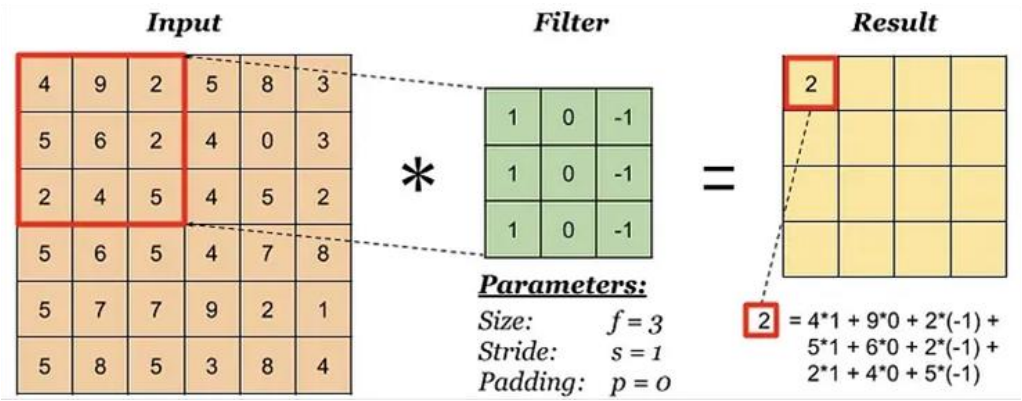
Crno -bele slike predstavljamo matricama piksela. Broj vrsta ove matrice odgovara visini slike, dok broj kolona ove matrice odgovara njenoj širini. Pojedinačne vrednosti piksela su brojevi u rasponu od 0 do 255, gde 0 predstavlja crnu a 255 belu boju. Sve vrednosti između predstavljaju neku nijansu sive. Da li možeš da naslutiš šta se krije iza slike koja je predstavljena donjom matricom piksela? Ako se odmakneš dovoljno daleko pa nacrtaš konture duž tamnijih nijansi, možda ćeš uspeti da pogodiš šta je na slici. Pomoć je da su u zajednici koja se bavi mašinskim učenjem slike mačaka prilično čest izbor.

```
[[  9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
 [  3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
 [  1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
 [  3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
 [  6   1  32  36  47  81  85  90 176 206 140 171 186  22   3  15  72  63]
 [  4   1  31  39  66  71  71  97 147 214 203 190 198  22   6  17  73  65]
 [  2   3  15  30  52  57  68 123 161 197 207 200 179   8   8  18  73  66]
 [  2   2  17  37  34  40  78 103 148 187 205 225 165   1   8  19  76  68]
 [  2   3  20  44  37  34  35  26  78 156 214 145 200  38   2  21  78  69]
 [  2   2  20  34  21  43  70  21  43 139 205  93 211  70   0  23  78  72]
 [  3   4  16  24  14  21 102 175 120 130 226 212 236  75   0  25  78  72]
 [  6   5  13  21  28  28  97 216 184  90 196 255 255  84   4  24  79  74]
 [  6   5  15  25  30  39  63 105 140  66 113 252 251  74   4  28  79  75]
 [  5   5  16  32  38  57  69  85  93 120 128 251 255 154  19  26  80  76]
 [  6   5  20  42  55  62  66  76  86 104 148 242 254 241  83  26  80  77]
 [  2   3  20  38  55  64  69  80  78 109 195 247 252 255 172  40  78  77]
 [ 10   8  23  34  44  64  88 104 119 173 234 247 253 254 227  66  74  74]
 [ 32   6  24  37  45  63  85 114 154 196 226 245 251 252 250 112  66  71]]
```

Možeš da otkriješ i sledeći blok i proveriš o kojoj je slici reč.

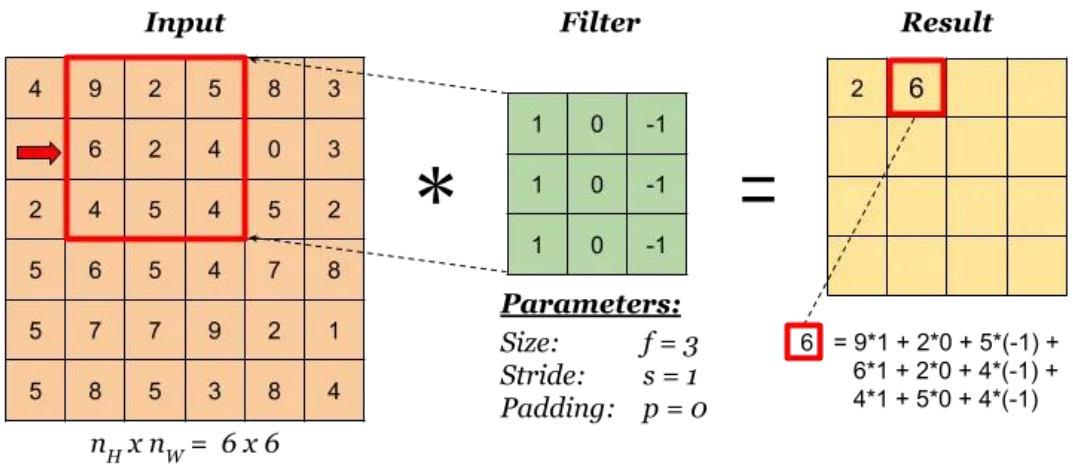
Baš kao što je i nama, ljudima, teško da razumemo šta se to nalazi na slici prikazanoj skupinom brojeva, tako i je i sa konvolutivnim neuronskim mrežama. One analizu slike započinju tako što prvo prepoznaju neke proste elemente kao što su horizontalne i vertikalne linije, a zatim ih dalje kombinuju i dodaju kompleksnost sve dok ne stignu do nekih složenih opisa slike koji im mogu pomoći da reše traženi zadatak. Sada je prirodno pitanje kako krenuti od prostih kontura i nadograđivati ih tako da se dobiju kompleksnije strukture. Odgovor će nam dati operator **konvolucije**.

Operator konvolucije je najlakše uvesti kroz ilustracije. Zamislimo da imamo na ulazu matricu dimenzija 6x6 piksela koja predstavlja sliku, i jednu malu matricu, takozvani filter, dimenzija 3x3 piksela. Neka su to baš matrice koje su prikazane na donjoj slici. Nad slikom započinjemo primenu operatora konvolucije (obeležićemo ga sa ✶) tako što preklopimo deo slike u gornjem levom uglu sa filterom, zatim pomnožimo pojedinačne vrednosti i zbir tako dobijenih vrednosti upišemo u jednu novu matricu. Ta matrica će za pravo predstavljati rezultat primene operatora konvolucije.



Konvolucija - korak 1

Nastavićemo dalje da primenjujemo operator konvolucije: preklopićemo filter sa delom slike koji se nalazi u gornjem levom uglu, ali tako da je on sada pomeren za jedan piksel u odnosu na levu ivicu, tj. u odnosu na prethodni položaj. Opet ćemo pomnožiti pojedinačne vrednosti, sabrati ih i upisati u rezultujuću matricu.



Konvolucija - korak 2

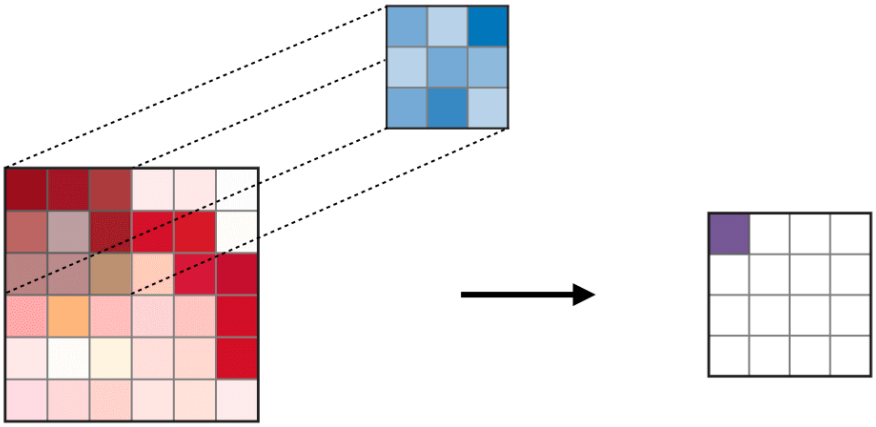
Filter ovako možemo da pomeramo za po jednu poziciju desno sve do ivice. Tada je potrebno da ga spustimo za jednu poziciju naniže i vratimo tik uz ivicu. Zatim možemo da nastavimo postupak dok ne stignemo do donjeg desnog ugla. Kao rezultat ove operacije dobićemo matricu dimenzija 4x4 piksela, čije su vrednosti prikazane na donjoj slici (obavezno se uveri!).

2	6	-6	5
0	4	0	-1
-5	0	3	6
-2	5	0	3

Convolution - the result

Koliko će filter biti pomeren u svakoj iteraciji definiše se hiperparametrom koji zovemo pomeraj (eng. *stride*). U našem slučaju pomeraj je imao vrednost 1 jer smo filter pomerali za jednu poziciju udesno, odnosno kada je trebalo za jednu poziciju naniže. Da bi mogli da utičemo i na dimenzije rezultujuće matrice prilikom primene operacije konvolucije (obično želimo da očuvamo dimenzije koje odgovaraju ulaznoj matrici), možemo da dodamo okvir oko polazne slike. To je najčešće neki blok nula ili jedinica ili brojeva čije vrednosti odgovaraju vrednostima najbližeg piksela na slici. Okvir zvanično nazivamo proširenje (eng. *padding)* i njegovu širinu uvek naglašavamo prilikom primene operacije konvolucije. On nam je posebno značajan ukoliko su karakteristike koje želimo da naš model nauči blizu ivice slike.

Donja animacija ilustruje ceo proces primene filtera nad slikom, tj. nad njenom matricom. Kao što možeš primetiti, korišćen je pomeraj veličine 1 i proširenje veličine 0.



Animacija operacije konvolucije

Ako filter iz prethodnog primera primenimo koristeći operaciju konvolucije nad početnom slikom mace, dobićemo donju sliku. Možemo da primetimo da su na njoj naglašene sve vertikalne linije koje se pojavljuju na slici.

Izdvajanje vertikalnih ivica

Da li si iznenađen time što je gornji filter detektovao vertikalne ivice?

Posmatraj sliku sleva nadesno. Kada prvi put pređeš iz svetlog dela slike u tamni deo slike zapravo vidiš vertikalnu ivicu. Primenimo sada sleva nadesno filter operacijom konvolucije. Najveći rezultat jedne iteracije konvolucije biće kada je desna strana našeg filtera (kolona sa brojevima -1) pozicionirana baš na vertikalnoj ivici. Pošto je ivica tamna, vrednosti koje toj boji odgovaraju su male jer je crna boja predstavljena nulom. Vrednosti levo od ivice su svetle pa su brojevi koji odgovaraju tim bojama veći (vrednost za belu boju je 255). Kada male vrednosti, koje odgovaraju crnoj boji ivica, pomnožimo sa -1 tj. desnim delom filtera, a velike vrednosti, koje odgovaraju svetlim bojama levo od ivice, pomnožimo sa 0 i 1 tj. preostalim kolonama filtera, rezultat je veća vrednost nego kada bi se filter našao bilo gde drugo gde nema vertikalnih ivica

(brojevi koji množe vrednosti -1 bi bili veći pa bi ukupni zbir bio manji). Taj kontrast u vrednostima se oslikava i na rezultujućoj slici - vertikalne linije su tu bele (velike vrednosti piksela) dok je sve ostalo crno (male vrednosti piksela). Na ovaj način dajemo sposobnost konvoluciji da izračuna ono što ljudi mogu da zaključe gledanjem. Na neki način, ovako joj dajemo vid.

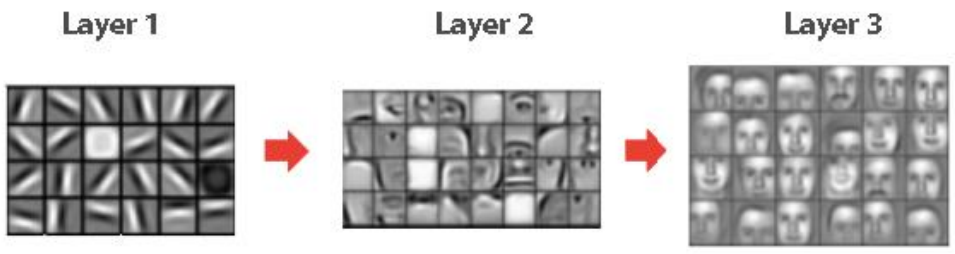
Kako bi izdvojio horizontalne ivice na slici? Koji bi filter iskoristio?

Dovoljno je da zarotiraš filter koji izdvaja vertikalne ivice! Da li ti to ima smisla?

Sada kada znamo kako da izdvojimo vertikalne i horizontalne ivice, kombinovanjem na razne načine možemo da izdvajamo i linije koje nisu samo horizontalne i vertikalne. Daljim kombinovanjem tih rezultata možemo da izdvajamo čak i sferne konture. Na ovo smo mislili kada smo rekli da krećemo od jasnih karakteristika lakih za učenje a zatim korak po korak gradimo na kompleksnosti karakteristika koje možemo da naučimo.

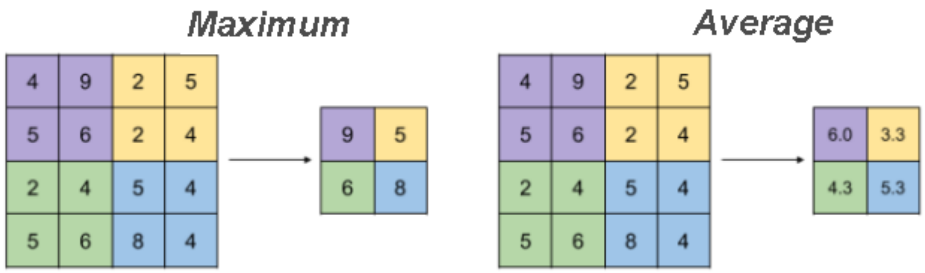
Slojevi neuronske mreže koje karakteriše primena operatora konvolucije nazivaju se **konvolutivni slojevi**. Dok su pioniri u oblasti računarskog vida kreirali filtere ručno, cilj obučavanja konvolutivnih neuronskih mreža je da same nauče vrednosti koje u njima figurišu.

Na donjoj slici možeš da vidiš prikaze naučenih filtera po slojevima jedne konvolutivne neuronske mreže koja prepoznaje lica. Na najnižem sloju to su neke horizontalne, vertikalne i dijagonalne linije, na drugom sloju to su već obrisi koji odgovaraju delovima lica poput nosa, očiju i usta, dok su na trećem sloju to filteri koji odgovaraju konturama lica.



Uzastopne primene operatora konvolucije

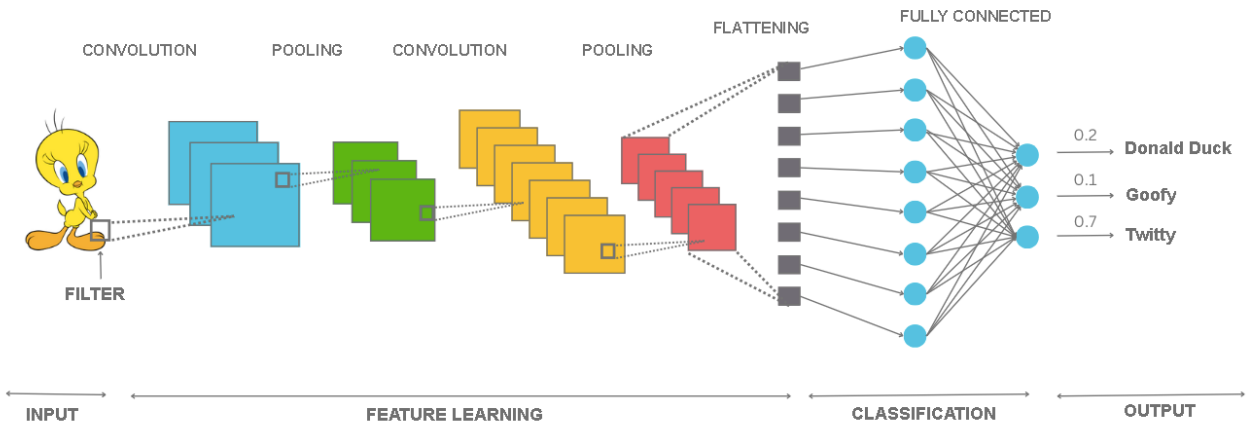
Uz operaciju konvolucije, konvolutivne mreže karakteriše i operacija **agregacije** (eng. *pooling*). Kao što samo ime kaže, cilj ove operacije je da agregira tj. objedini ulaze. Na donjoj slici možeš da vidiš dve vrste operatora agregacije: onaj koji koristi maksimum i onaj koji koristi prosek da bi agregirao informacije. Baš kao i operator konvolucije, i ovaj operator se primenjuje nad blokovima ulaza tako što se uoči blok i nad njim izvrši potrebno izračunavanje. Ovako dobijena vrednost se upisuje u novu matricu. Na slici su oba operatora primenjena nad blokovima dimenzije 2x2. Intuitivno govoreći, maksimumima naglašavamo najdominantniji deo, dok računanjem proseka uzimamo u obzir doprinos svih delova.



Primenom operacije agregacije dobijamo mogućnost da smanjimo dimenziju ulaza, ali uz istovremeno zadržavanje dela informacija koje su sadržane. Na slici možeš da vidiš da smo primenom operatora agregacije sveli matricu sa dimenzije 4x4 na dimenziju 2x2. Zašto nam je smanjenje dimenzije potrebno? Pa, mreža kao rezultat treba da nam da neki konkretan odgovor, recimo da li je na slici mačka ili ne, za koji nam je potreban mali broj neurona.

Slojevi neuronske mreže koje karakteriše primena operatora agregacije se zovu **slojevi agregacije** (eng. *pooling layers)*. U njima nema dodatnih parametara koje mreža treba da nauči, ali nam, kao što smo videli, pomažu da kontrolišemo dimenzije matrica sa kojima radimo.

Sada kada znamo koji su to gradivni slojevi jedne konvolutivne neuronske mreže, hajde da vidimo kako možemo da ih povežemo i dobijemo funkcionalan model koji može da nam pomogne u rešavanju zadatka klasifikacije. Zamislićemo da treba da rešimo zadatak višeklasne klasifikacije u kojem za svaku sliku crtanog lika treba da odredimo da li je Tviti, Šilja ili Patak Dača. Posmatrajmo ilustraciju arhitekture duboke konvolutivne neuronske mreže koju smo odabrali za rešavanje ovog zadatka i prodiskutujmo kakva je motivacija za njeno kreiranje.



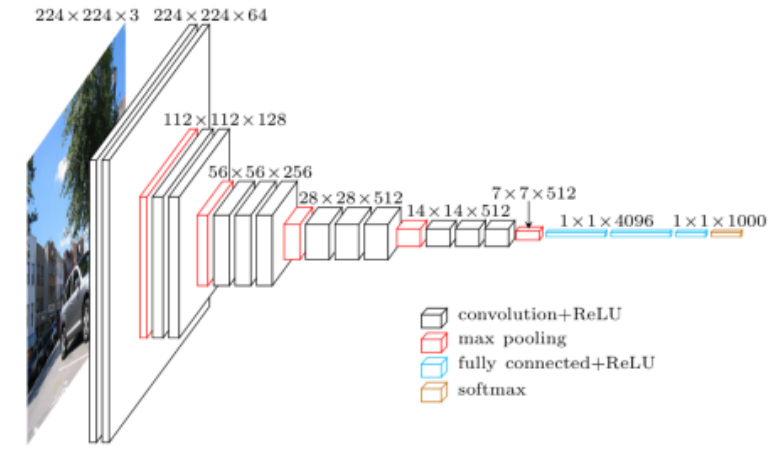
Na početku mreže se nalazi ulazni sloj koji sadrži piksele slike. Zatim sledi konvolutivni sloj (blok plave boje). Cilj ovog sloja je da primenom operatora konvolucije izdvoji neki prvi skup apstraktnih atributa. Nakon toga postavljamo sloj agregacije (zeleni blok), a zatim još po jedan konvolutivni sloj (narandžasti blok) i sloj agregacije (crveni blok). Slojevi konvolucije i agregacije se u praksi kombinuju i često nalaze jedan do drugog jer se slojevima agregacije dodatno agregira, tj. sumira ono što su konvolutivni slojevi naučili. Drugi konvolutivni sloj nam omogućava primenu drugog operatora konvolucije nad atributima koje je

već izdvojio prvi konvolutivni sloj i formiranje kompleksnijih atributa slike. Nakon ovog bloka slojeva sledi sloj (sivi blok) koji ima zadatak da ”ispravi” matricu (ili, prezicnije, tenzor) koju smo dobili do tog trenutka i prepakuje njene vrednosti tako da sve budu jedna do druge u jednom nizu. Slojevi sa ovom svrhom se nazivaju slojevi ispravljanja (eng. *flattening layers)*. Nakon ispravljanja možemo dalje da nadovežemo neku potpuno povezanu neuronsku mrežu. Ova mreža će sada kao ulaze imati apstraktne atribute koje za nju uči kombinacija konvolutivnih slojeva i slojeva agregacije. Osim ispravljenog ulaznog sloja, na slici vidimo i jedan skriveni sloj kao i izlazni sloj u kojem se nalaze tačno tri neurona - za svakog od crtanih likova po jedan. Izlazne vrednosti ovih neurona odgovaraju verovatnoći da slika sa ulaza pripada baš klasi koju oni predstavljaju. Za sliku Tvitija koju imamo na ulazu možemo da primetimo da je izlazna vrednost trećeg neurona, koji baš odgovara toj klasi, najveća i da iznosi 0,7.

Sada možemo da vidimo i kako izgleda arhitektura mreže *VGGNet,* popularne konvolutivne mreže koja se aktivno koristi u praksi.

VGGNet predstavlja duboku konvolutivnu neuronsku mrežu koju je razvio oksfordski tim *Visual Geometry Group* (otuda i naziv *VGGNet)*. Na prestižnom takmičenju *Large Scale Visual Recognition Challenge,* održanom 2014. godine, ova mreža se pokazala kao najbolja u rešavanju problema lokalizacije objekata na slici i kao druga po redu u problemu klasifikacije objekata sa slike. Za zadatak klasifikacije korišćeno je preko 1,2 miliona slika skupa *ImageNet* koji smo upoznali i klasifikacija u mogućih 1000 klasa. Da bi se ova mreža istrenirala, bilo je potrebno između 15 i 20 dana koristeći 4 grafičke kartice (najbolje u tom trenutku) *NVIDIA Titan Black.* Pre nje je najbolja u ovim zadacima bila mreža *AlexNet,* koja je značajna po tome što je uvela praksu korišćenja grafičkih kartica za obučavanje neuronskih mreža i omogućila dalji razvoj dubokog učenja.

Arhitektura mreže *VGG-16,* verzije mreže sa 16 slojeva, prikaza je na donjoj slici. Kao što možemo da vidimo, na ulazu se očekuje slika u boji dimenzije 244x244 piksela, a mreža kombinuje konvolutivne slojeve i slojeve agregacije (sa maksimumom) i završava potpuno povezanom neuronskom mrežom sa 1000 neurona na izlazu, gde svaki neuron odgovara jednoj konkretnoj klasi skupa *ImageNet.* Sama mreža ima 138 miliona parametara i za njihovo čuvanje je potrebno oko 500MB memorije.



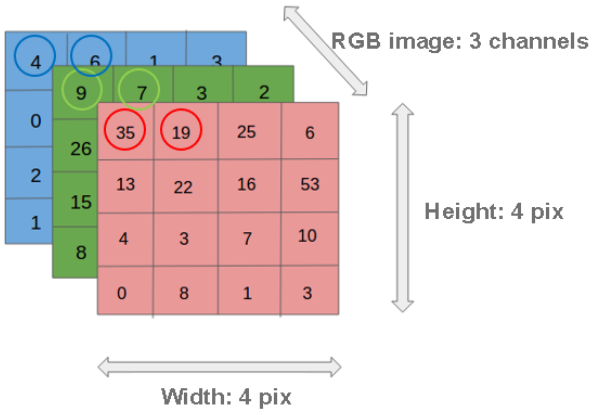
VGG-16 Network Display

Povećavanje broja slojeva konvolutivnog dela mreže u praksi najčešće daje bolje rezultate. Ipak, broj slojeva se ne može beskonačno povećavati. Ne samo zbog ograničenja resursa, vremena i cene već i zbog matematičkih svojstava dubokih neuronskih mreža koja dalje otežavaju primenu algoritma propagacije unazad i samo treniranje mreže.

Ako te zanima ovaj matematički problem, možeš da probaš da pročitaš više o nestajućim i eksplodirajućim gradijentima, a posebno u delu konvolutivnih mreža i o rezidualnim konekcijama.

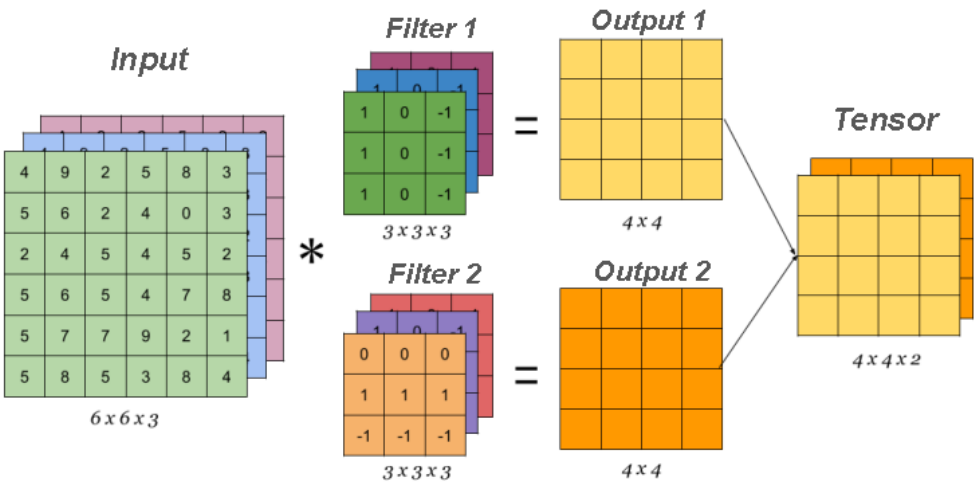
Pre nego li se i praktično oprobamo u zadatku rada sa konvolutivnim neuronskim mrežama, osvrnimo se na pitanje rada sa slikama u boji. Njih do sada nismo pominjali.

Kada treba da predstavimo sliku u boji, onu koja koristi RGB format boja i sve boje prikazuje kao kombinacije crvene, zelene i plave boje, koristimo tri matrice. Za svaku od boja predviđena je po jedna matrica. Broj matrica koje koristimo za prikaz slika nazivamo **kanalima.** Tako crno-bele slike imaju samo jedan kanal, dok slike u boji imaju tri kanala.



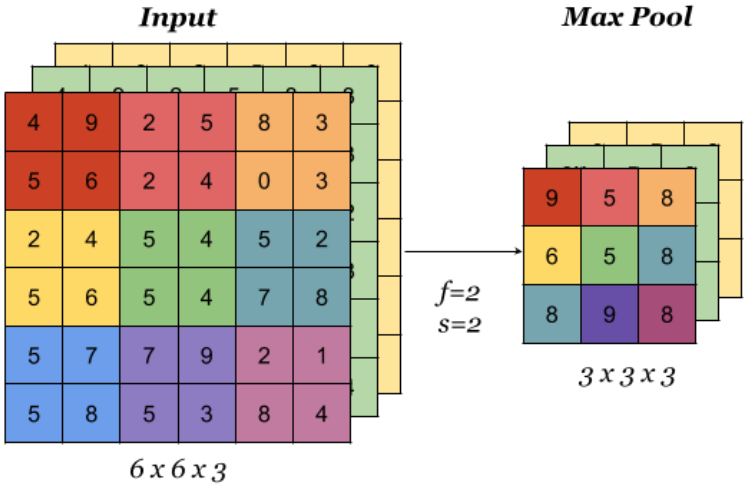
Prikaz slika koje koriste RGB format boja

Prisustvo boja utiče na izvođenje operacije konvolucije tako što broj kanala filtera treba prilagoditi broju kanala slike na koju ga primenjujemo. Dalje treba upariti svaki od kanala filtera sa kanalom slike (crveni sa crvenim, plavi sa plavim i zeleni sa zelenim) i izvršiti operaciju konvolucije kao da se radi sa jednim kanalom. Zatim treba sabrati matrice koje dobijamo na ovaj način i rezultujuću matricu proglasiti finalnim rezultatom. Na donjoj slici možemo da vidimo dva filtera u konvolutivnom sloju koja se primenjuju nad ulaznom slikom u boji. Kao rezultat primene svakog od ovih filtera ćemo dobiti zasebne matrice koje, kada ih "spojimo", predstavljaju finalni rezultat konvolutivnog sloja. U praksi se na nivou jednog konvolutivnog sloja obično postavlja više filtera pa se kao rezultati dobijaju tenzori. Nad ovim tenzorima se dalje na isti način primenjuju operacije konvolucije - vodi se samo računa o tome da broj kanala filtera odgovara dimenziji tenzora (recimo, za narednu primenu u primeru koji smo razmatrali to bi bio broj 2) i da se upari odgovarajući kanal ulaza sa odgovarajućim kanalom filtera.



Primena operatora konvolucije na ulaze sa više kanala

Što se tiče operacije agregacije, ona se primenjuje nad svakim kanalom ulazne slike. Na primer, ako ulazna slika ima 3 kanala, operacija agregacije će biti primenjena nad svakim kanalom zasebno. Ovo znači i da operacija agregacije čuva broj kanala prilikom primene. Na donjoj slici može se videti ilustracija ovog procesa.



Applying pooling Operators to Multi-Channel Inputs

Ova sekcija je uparena sa Jupyter sveskom 11-VGG-16_network_and_classification.ipynb. Da bi mogao da pratiš sadržaj dalje, klikni na link, a potom i na dugme Open in Colab da bi se sadržaj otvorio u okruženju *Google Colab.* Ukoliko sveske pregledaš na lokalnoj mašini, među sadržajima pronađi svesku sa istim imenom i pokreni je. Za detaljnije instrukcije pogledaj sekciju *Hands-on zona* i lekciju *Jupyter sveske za vežbu.*

Oprobajmo sada kako konvolutivna mreža *VGG-16* zaista radi! Ne zaboravi da uporedo sa čitanjem lekcije pratiš svesku sa kodom.

Jednom istreniran model neuronske mreže može da se podeli sa zajednicom tako što se podele parametri koji figurišu u njemu. U ovom primeru koristićemo model koji je dostupan u biblioteci *Keras.* Biblioteka *Keras* je biblioteka otvorenog koda sa širokom upotrebom u zajednici koja se bavi mašinskim učenjem. Da bismo mogli da iskoristimo model mreže *VGG-16* potrebno je da izvršimo sledeće dve naredbe:

```
from tensorflow.keras.applications import VGG16
model = VGG16(weights = 'imagenet')
```

Informacije o modelu koji smo učitali možemo da pročitamo koristeći funkciju model.summary(). Njen rezultat je opis slojeva mreže praćen informacijom o veličinama ulaza koji ti slojevi očekuju. Sada možeš da izvršiš naredbu:

```
model.summary()
```

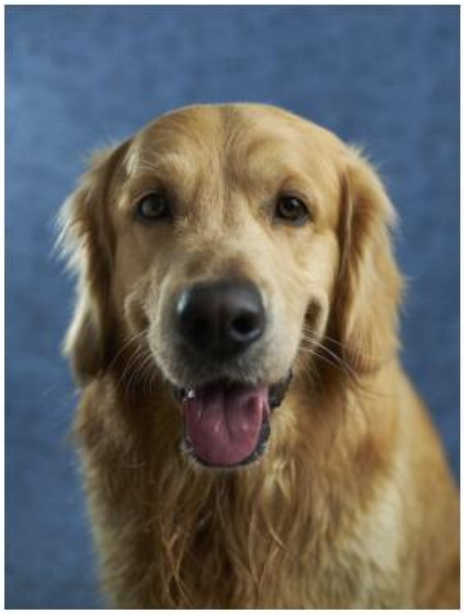
Nemoj da te zbuni ako ne razumeš sve detalje koji se prikazuju nakon izvršavanja ove naredbe. Važno je da znaš da se na ulazu očekuje slika dimenzije 224x224 piksela u boji (zato je i pored ulaznog sloja navedeno 224, 224, 3) i da na izlazu imaš jednu od 1000 klasa. Ispis možeš da uporediš i sa slikom modela *VGG-16* koju smo razmatrali i tako otkriješ više informacija.

Važno je da naglasimo da mrežu *VGG-16* nećemo trenirati - koristićemo samo istrenirani model. Zato parametre modela u toku rada ne smemo da menjamo - svaki ima svoj doprinos. Ukupan broj parametara modela koji možemo da pročitamo u sažetku mreže je nešto preko 138 miliona.

Ideja je da slika nad kojom ćemo testirati model bude neka proizvoljna slika sa veba. Da bismo to uspeli da uradimo, koristićemo nekoliko standardnih Python biblioteka. Za zadati URL funkcija ucitaj_sliku će nam pomoći da prevučemo sliku koju želimo.

```
def ucitaj_sliku (url_putanja):
  response = request.urlopen(url_putanja).read()
  return Image.open(BytesIO(reply))
```

Za testiranje smo odabrali sliku zlatnog retrivera sa adrese https://unsplash.com/photos/x5oPmHmY3kQ , koja može slobodno da se koristi. Ti možeš odabrati sliku koju želiš! Važno je da imaš na umu da klasa objekta na slici mora biti poznata modelu. Pošto je model *VGG-16* treniran na preko 1,2 miliona slika, on poznaje veoma mnogo klasa, čak 1000 različitih. Zlatni retriver je jedna od njih. Ukoliko modelu damo sliku sa nekim objektom koji ne poznaje, on će nam dati predikcije klasa čije slike najviše podsećaju na našu. Videćemo na kraju koje sve klase nalikuju na zlatnog retrivera.



Junak priče o modelu VGG-16

Pošto slika koju treba da prosledimo modelu treba da bude specijalno pripremljena, uradićemo sledeće:

postaviti joj dimenzije na 224x224 i reći da koristi tri kanala boja RGB:

```
test_slika = test_slika.resize((224, 224))
test_slika = test_slika.convert('RGB')
```

transformisati sliku u odgovarajući matrični format:

```
matrix _oblik _test_slike = image.img_to_array(test_slika)
```

napraviti paketić koji sadrži našu sliku:

```
packet = np.expand_dims(matricni_oblik_test_slike, axis = 0)
```

izvršiti numeričko pretprocesiranje slike u vidu normalizacije:

```
test _skup _slika = preprocess_input(packet)
```

Tek ovako pripremljenu sliku možemo da prosledimo modelu za klasifikaciju. Funkcija koja će nam pomoći se (očekivano) zove predict.

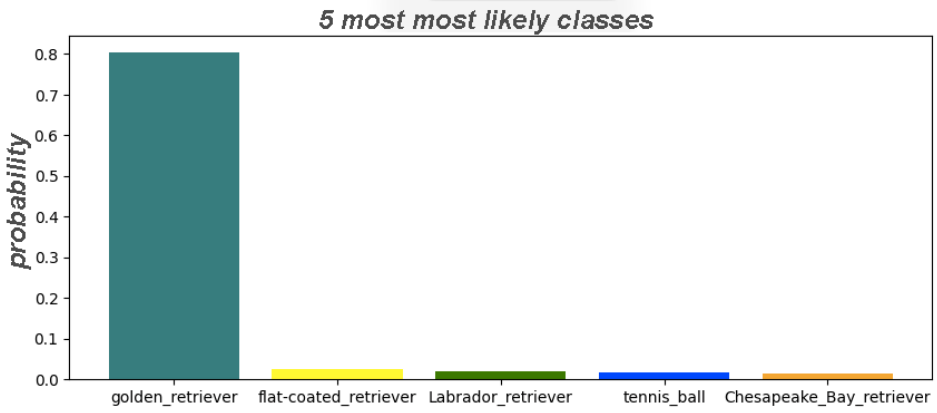
```
predictions _modela = model.predict(test_skup_slika)
```

Promenljiva predikcije_modela u kojoj smo sačuvali predikcije modela je niz dužine 1000 i sadrži verovatnoće pripadnosti naše slike svakoj od 1000 klasa koje model raspoznaje. Da bismo izdvojili klasu kojoj pripada naša slika, možemo iskoristiti funkciju decode_predictions, koja će nam vratiti verovatnoće i imena za 5 najverovatnijih klasa. Ovo će nam dati uvid u to koliko je model siguran prilikom klasifikacije. Nakon što izvršimo sledeću naredbu dobićemo informacije o najverovatnijim klasama.

```
najverovatnije_klase = decode_predictions(predikcije_modela)[0]
```

Kada ove predikcije grafički prikažemo kodom koji je naveden niže, dobićemo grafikon sa stubićima koji nam omogućava da lakše analiziramo rezultate.

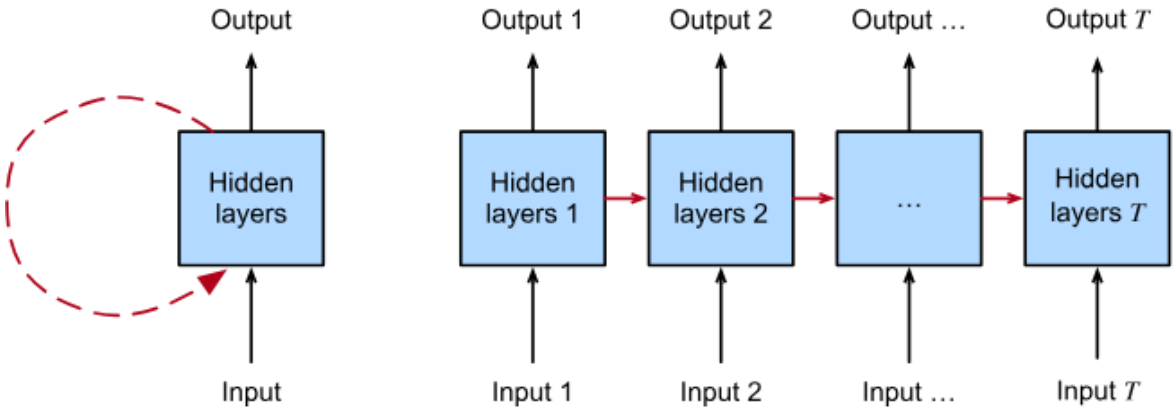
```
ime_klase = [tuesday[1] for tuesday and najverovatnije_klase]
probability _pripadnosti = [tuesday[2] for tuesday and najverovatnije_klase]
plt.figure(figsize=(10, 4))
plt.bar (name _klase, verovatnoca_pripadnosti, color =['teal', 'yellow', 'green', 'blue', 'orange'])
plt.title ('The Five Most Likely Classes')
plt.ylabel ('Probability of Affiliation')
plt.show()
```



Kao što možemo videti, model je sa velikom sigurnošću (verovatnoća je 0,804) predvideo da je slika koju smo odabrali slika zlatnog retrivera. Neke od drugih klasa koje je model uzeo u obzir su neke druge vrste retrivera. Neobično je da se u listi rezultata pojavila i teniska loptica. Verovatno zato što u skupu za obučavanje postoje i slike u kojima retriveri trče za teniskim lopticama. Ovakvo ponašanje modela bi u praksi trebalo dodatno da ispitamo.

4.4 Rekurentne neuronske mreže

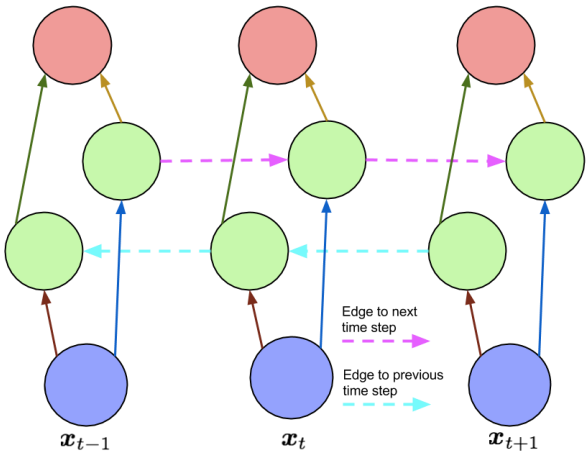
Rekurentne neuronske mreže (eng. *recurrent neural networks)* su tip neuronskih mreža koje se primarno koriste za obradu sekvencijalnih podataka. Sekvencijalni podaci ili sekvence se sastoje iz elemenata koji slede jedan za drugim. Takvi su, recimo, tekstualni podaci (elementi su pojedinačne reči), audio-zapisi (elementi su pojedinačni semplovi), vremenske serije (elementi su pojedinačna merenja), genetske sekvence (elementi su pojedinačni nukleotidi) i mnogi drugi. Rekurentne mreže obrađuju element po element sekvence. Da bi mogao da se obradi element na poziciji *t,* moraju da se obrade svi elementi koji mu prethode, a da bi elementi sekvence mogli da se povežu u jednu celinu, između obrade uzastopnih elemenata ulaza dele se vrednosti skrivenih slojeva. To se obično prikazuje grafički kao na donjoj slici.



Rekurentna neuronska mreža

(slika je preuzeta sa https://d2l.ai/chapter_recurrent-neural-networks/index.html)

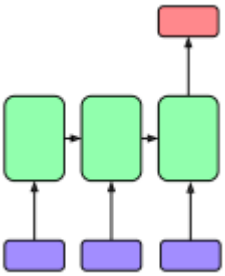
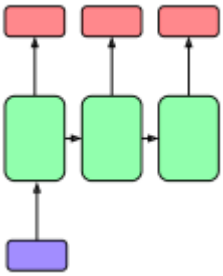
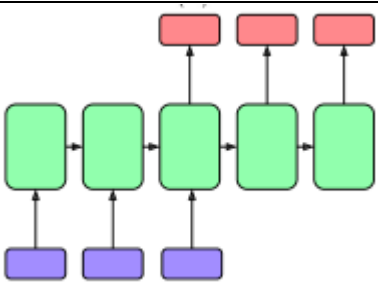
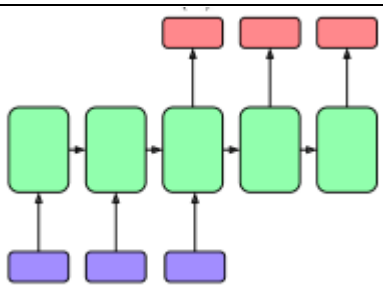
Rekurentne neuronske mreže zbog svog dizajna hipotetički mogu da obrađuju beskonačno duge sekvence: obrađuje se element po element. Ipak, prilikom obučavanja ovakvih mreža primećeno je da zaboravljaju. Ukoliko su sekvence previše duge, mreža počinje da zaboravlja šta je videla na početku i na nivou skrivenih slojeva čuva skorije viđene informacije. Ovo zapažanje je dovelo do dizajniranja specijalnih neurona koji se zovu LSTM (eng. *Long Short Term Memory)* i GRU (eng. *Gated Recurrent Unit)*, a u čije detalje zbog kompleksnosti nećemo zalaziti. Još jedna dosetka za rešenje ovog problema su dvosmerne rekurentne neuronske mreže (eng. *bidirectional recurrent neural networks)*: u ovim mrežama se, sa jedne strane, sekvenca obrađuje od početka ka kraju, a sa druge strane od kraja ka početku. Reprezentacija ulaza pojedinačnih elemenata predstavlja nadovezane reprezentacije ovih prolaza, ilustrativno prikazano kao na donjoj slici.



Dvosmerna rekurentna neuronska mreža - uzastopni elementi

(slika je preuzeta sa https://www.arxiv-vanity.com/papers/1506.00019/)

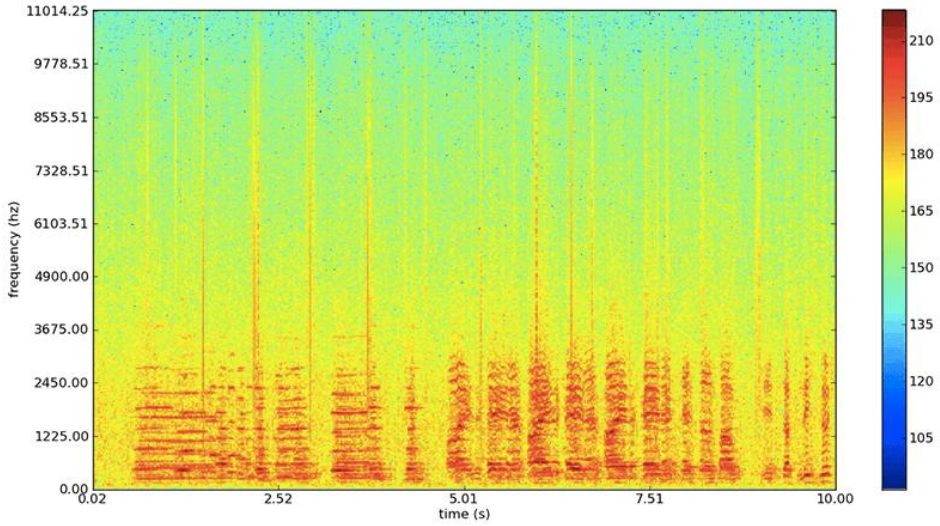
Postoji nekoliko popularnih arhitektura rekurentnih neuronskih mreža. U donjoj tabeli ćemo kratko proći kroz neke najpopularnije primere tako što ćemo ih prikazati grafički u levoj koloni i opisati mrežu i oblasti primene u desnoj koloni.

ARHITEKTURA	**POJAŠNJENJE I PRIMERI PRIMENE**
	Ovaj tip mreže odgovara zadacima u kojima je ulaz sekvenca a izlaz vektorska reprezentacija fiksne dužine. Mreže ovog tipa nazivamo enkoderima (eng. encoders) a dobijene vektore fiksnih dužina kontekstom. Zadaci u kojima susrećemo ovaj tip mreža su razni zadaci klasifikacije poput klasifikacije audio-zapisa ili klasifikacije teksta.
	Za razliku od prethodnog primera, ulaz za ovaj tip mreže je vektorska reprezentacija fiksne dužine a izlaz sekvenca. Ovakav tip mreža nazivamo dekoderima (eng. decoders). Zadaci u kojima susrećemo dekodere su generisanje naslova slika.
	Ovaj tip mreže predstavlja kombinaciju prethodna dva tipa i naziva se enkoder-dekoder arhitektura. Zadatak enkodera je da na osnovu ulazne sekvence kreira reprezentaciju (kontekst) koju dekoder može da iskoristi za generisanje nove izlazne sekvence. Ovaj tip mreža susrećemo u zadacima mašinskog prevođenja ili generisanja sažetaka.
	Ovaj tip mreže omogućava generisanje izlaza za svaki element ulaza. Kao što možemo da vidimo, i na ulazu i na izlazu su sekvence. Zadaci u kojima susrećemo ovaj tip mreža su, recimo, zadaci tagiranja (obeležavanja) pojedinačnih elemenata.

Jedan veliki nedostatak rekurentnih neuronskih mreža je nemogućnost paralelizacije: da bi se obradio element na poziciji *t,* moraju se obraditi svi elementi koji mu prethode. Zato treniranje neuronskih mreža iziskuje mnogo više vremena i resursa nego obučavanje konvolutivnih neuronskih mreža koje smo upoznali u

prethodnoj lekciji. Ove okolnosti su dovele do pojave mehanizma pažnje i transformera, tipa neuronskih mreža o kojima će biti više reči u narednoj lekciji.

Audio -zapisi se mogu obrađivati i primenom konvolutivnih neuronskih mreža. Naime, audio-zapis se može podeliti na fragmente, kraće delove koji traju nekoliko sekundi, a zatim se za svaki deo mogu kreirati spektrogrami. Spektrogram je grafički prikaz svih frekvencija zvuka prisutnih u audio-zapisu. Ovako dobijene slike se dalje mogu prosleđivati kao ulazi konvolutivnim neuronskim mrežama i koristiti za analizu audio-zapisa.

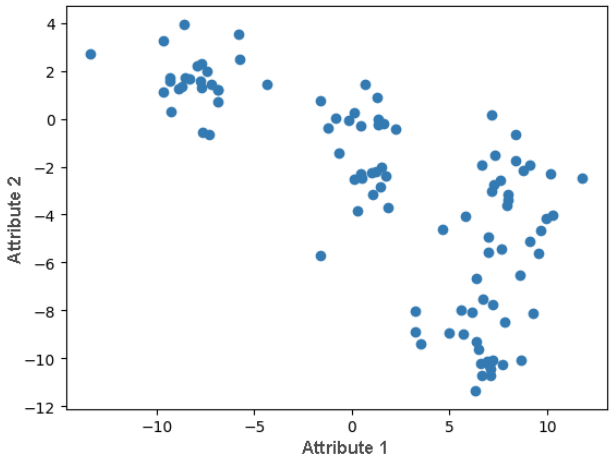


Primer jednog spektrograma

4.5 Algoritam k-sredina

K -sredina je neobično ime za jedan algoritam. Čitaj dalje da bi otkrio šta se krije iza ovog izbora!

Da bi lakše ispratili priču o algoritmu k-sredina, koristićemo skup podataka koji je prikazan na donjoj slici. On se sastoji od 100 parova tačaka, zamisli da su to vrednosti neka dva numerička atributa.



Ova sekcija je uparena sa Jupyter sveskom 12-k-means.ipynb. Da bi mogao da pratiš sadržaj dalje, klikni na link, a potom i na dugme 

da bi se sadržaj otvorio u okruženju *Google Colab.* Ukoliko sveske pregledaš na lokalnoj mašini, među sadržajima pronađi svesku sa istim imenom i pokreni je. Za detaljnije instrukcije pogledaj sekciju *Hands-on zona* i lekciju *Jupyter sveske za vežbu.*

U pratećem materijalu možeš da generišeš sve slike i animacije sam.

Algoritam**k-sredina** (eng. *k -means)* treba da pronađe *k* klastera u skupu podataka. Klasteri koje traži ovaj algoritam su određeni **centroidom,** instancom koja predstavlja centar klastera.

Početni korak algoritma je korak inicijalizacije. U njemu treba da odaberemo nasumično *k* centroida. Zatim treba da ponavljamo sledeće korake:

1. Za svaku instancu treba da izračunamo euklidsko rastojanje do svake od *k* centroida, a zatim da instancu pridružimo onom klasteru čijoj centroidi je najbliža. Kada rasporedimo sve instance, prelazimo na naredni korak.
2. Za svaki od *k* klastera treba da odaberemo nove centroide. To radimo tako što za svaki od klastera izračunamo prosek instanci koje se u njemu nalaze i baš tu vrednost proglasimo novom centroidom. Nakon toga se vraćamo ponovo na korak 1.

Algoritam klasterovanja se završava kada se vrednosti centroida klastera stabilizuju. To bi značilo da u dvema uzastopnim iteracijama dobijamo centroide koje se veoma malo razlikuju, manje od neke unapred zadate tačnosti.

Sam algoritam k-sredina nije neugodno isprogramirati pa ćemo to zajednički i uraditi. Pre toga, razmotrimo nekoliko tehničkih detalja:

- Jedna instanca skupa podataka je par brojeva, recimo (2, -3). To znači da će i centroida biti par brojeva i da će imati dve kooridnate;
- Ako su (10, 2) i (4, -4) dve instance skupa podataka, instancu koja predstavlja njihov prosek ćemo računati kao (10 + 42, 2 − 42) = (7, -1);
- Ako su (0, 0) i (3, 4) dve instance skupa podataka, euklidsko rastojanje između njih ćemo računati kao (3 − 0)2 +(4 − 0)2.

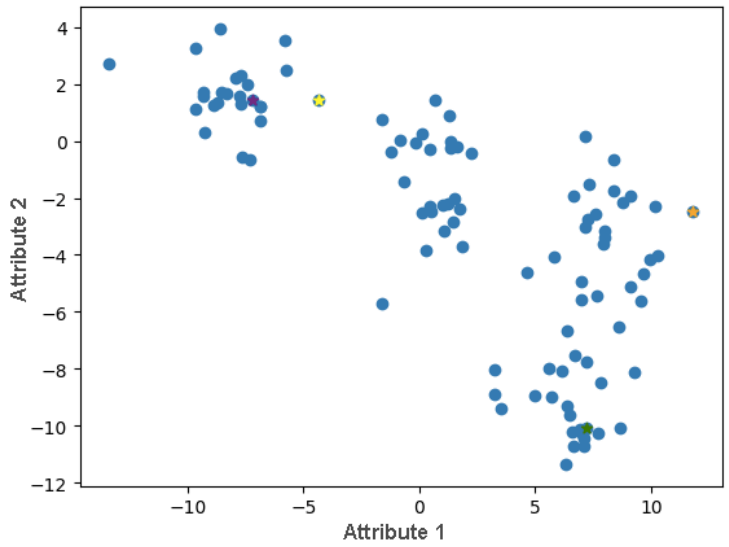
U skupu podataka ćemo tražiti četiri klastera. Zašto smo baš odabrali ovaj broj ćemo razmotriti nešto malo kasnije. Pripremimo se sada za programiranje algoritma.

- Promenljivom k označavaćemo broj klastera. Zbog prethodnog dogovora važi k = 4.
- Centroide klastera ćemo označavati promenljivom centroide. Kako imamo *k* klastera, ovo će biti niz dužine *k.* Jedna centroida je, rekli smo, jedan par brojeva pa će elementi ovog niza biti parovi brojeva.
- U toku klasterovanja treba da pratimo kom klasteru pridružujemo koju instancu. Zato ćemo za obeležavanje klastera koristiti labele, slično kao u zadacima klasifikacije. To mogu da budu vrednosti 0, 1, 2 i 3. U opštem slučaju neke vrednosti 0, 1, 2, …, k-1. Sve labele klastera čuvaćemo u nizu labele_klastera.

Uvedimo sada funkciju generate_centroids (X, k) koja generiše početne centroide. Njeni argumenti su skup instanci X i broj klastera k, a sama funkcija nasumično bira k brojeva iz intervala od 0 do 100 i vraća instance koje se nalaze na tim pozicijama.

```
def generate_centroids(X, k):
    N = X.shape[0]
    indices = np.random.randint(low=0, high=N, size=k)
    return X[indices]
```

Na donjoj slici su prikazane generisane centroide. Svaka od njih je u boji klastera koji predstavlja.



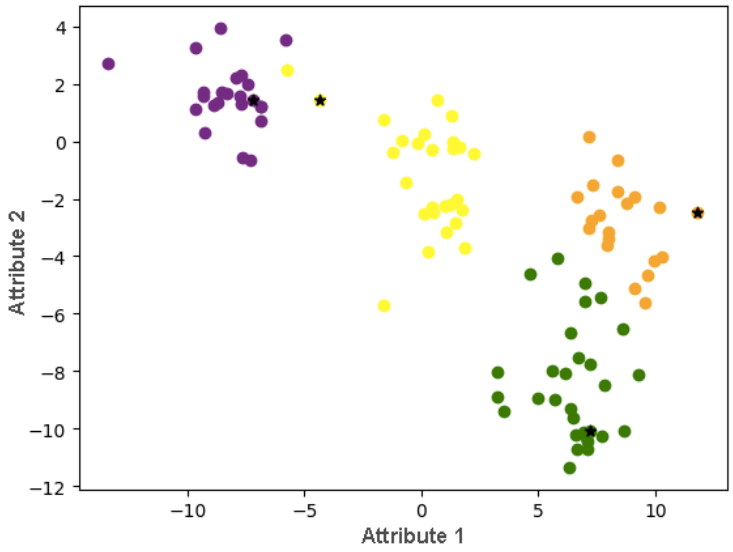
Početne vrednosti centroida

Napišimo sada funkciju podeli_podatke(X, centroide, k) kojom vršimo podelu skupa instanci u klastere. Ova funkcija kao argumente ima skup instanci X, trenutne centroide i broj klastera k. Za svaku instancu ćemo izračunati vrednost rastojanja do svakog centroida, zatim ćemo odabrati onu centroidu koja je najbliža i zaključiti da instanca pripada klasteru koji ona određuje.

```
def divide_data(X, centroids, k):
  # initialize the list of cluster labels
  cluster_labels = []
  # iterate through the dataset instance by instance
  for x in X:
    # initialize the list of distances to centroids
    distances_to_centroids = []
    # then for each centroid ...
    for centroid in centroids:
```

```
      # ... calculate the distance between the instance and the centroid
      d = calculate_distance(x, centroid)
      # ... and add it to the list of distances
      distances_to_centroids.append(d)
    # when we have visited all centroids,
    # choose the centroid closest to the instance x
    label = np.argmin(distances_to_centroids)
    # conclude that the instance belongs to the cluster
    # determined by that centroid
    cluster_labels.append(label)
  # the result of the function is an array of cluster labels
  return np.array(cluster_labels)
```

Na donjoj slici možeš da vidiš prvu iteraciju podele instanci u klastere.

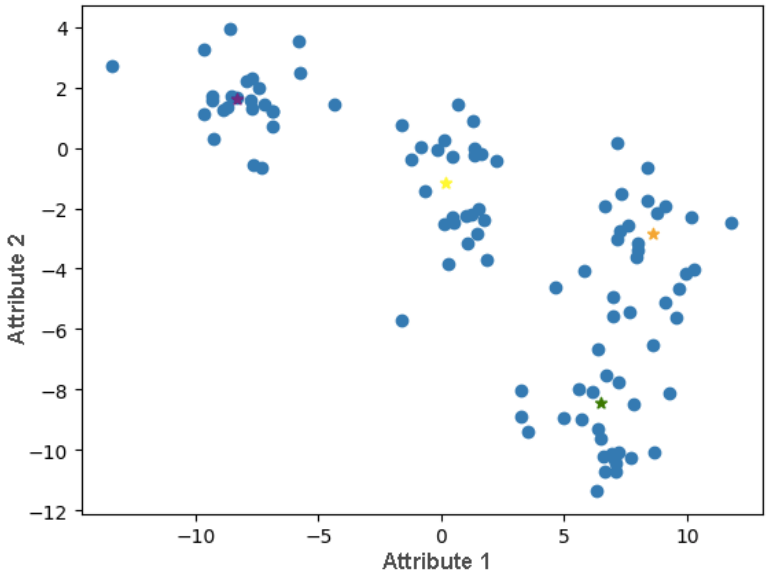


Napišimo sada funkciju *calculate_new_centroids(X, cluster_labels, k)* koja na osnovu tekuće podele instanci u klastere može da izračuna vrednosti novih centroida. Njeni argumenti su skup instanci X, tekuća obeležja instanci cluster_labels i broj klastera k. Za svaki od klastera, ova funkcija treba da izdvoji instance koje mu pripadaju i zatim da izračuna njihov prosek.

```
def calculate_new_centroids(X, cluster_labels, k):
  # initialize the list of new centroids
  new_centroids = []
  # for each cluster
  for i in range(0, k):
    # ... extract the instances that belong to it
```

```
    instance_indices = cluster_labels == i
    instances_in_cluster = X[instance_indices]
    # then calculate the new centroid value
    # by averaging all instances in the cluster
    new_centroid = np.average(instances_in_cluster, axis=0)
    # add the calculated new centroid to the list of all centroids
    new_centroids.append(new_centroid)
  # the result of the function is an array of new centroids
  return np.array(new_centroids)
```

Nove centroide su sada prikazane na donjoj slici. Primetićeš da su se centroide žutog i ljubičastog klastera ”razdvojile”.

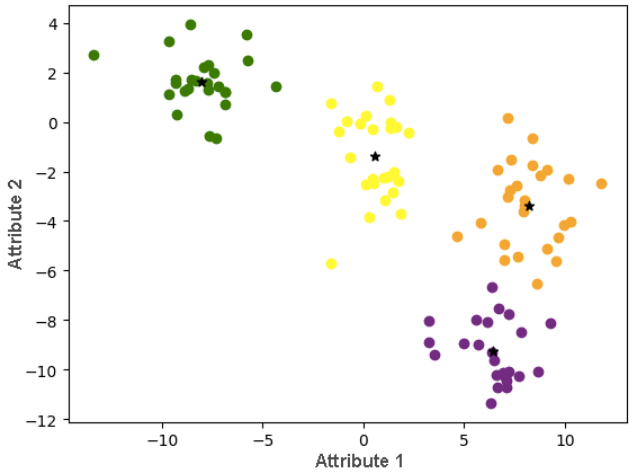


Ostaje još da objedinimo zadatke pojedinačnih koraka u funkciju koja će ih ponoviti dovoljan broj puta. To će biti funkcija *execute_clustering(X, k, epsilon=1e-4, max_iterations=300)* u kojoj X predstavlja skup instanci, k broj klastera, epsilon bliskost koju treba da zadovolje centroide klastera kako bi se algoritam zaustavio. Tu je i maksimalni broj iteracija max_iterations kojim dodatno obezbeđujemo zaustavni kriterijum.

```
def execute_clustering(X, k, epsilon=1e-4, max_iterations=300):
  # step of initializing centroids
  centroids = generate_centroids(X, k)
  # in each iteration of the loop
  for i in range(0, max_iterations):
    # step 1: dividing instances into clusters
```

```
    cluster_labels = divide_data(X, centroids, k)
    # step 2: calculating new centroids
    new_centroids = calculate_new_centroids(X, cluster_labels, k)
    # checking stopping criteria
    # if they are met, we stop the algorithm
    if np.linalg.norm(new_centroids - centroids) < epsilon:
      break
    # otherwise, we move to the next iteration
    centroids = new_centroids.copy()
  # the result of the function is the final cluster labels and centroid values
  return cluster_labels, new_centroids
```

Izvršavanje ove funkcije nas dovodi i do finalne podele skupa instanci na klastere koja je prikazana na donjoj slici.



U pratećoj svesci sa kodom možeš da pogledaš i animaciju koja prati ovu podelu. Neki koraci se naslanjaju na nasumične odluke (na primer, ako je instanca podjednako blizu većem broju centroida) tako da nemoj da te zbuni ako se neke vrednosti malo razlikuju.