Kratak sadržaj

POGLAVLJE 1
Principi veštačke inteligencije 1

POGLAVLJE 2
AI u tehnikama pretrage i igrama 29

POGLAVLJE 3
Regresija 81

POGLAVLJE 4
Klasifikacija 137

POGLAVLJE 5
Upotreba stabala za prediktivnu analizu 167

POGLAVLJE 6
Uvod u klasterizaciju 197

POGLAVLJE 7
Duboko učenje pomoću neuronskih mreža 223

DODATAK A
Duboko učenje pomoću neuronskih mreža 257

INDEKS 307

Sadržaj

Uvod XI

POGLAVLJE 1

Principi veštačke inteligencije 1

Uvod 2
Kako AI rešava probleme iz realnog sveta? 2
Raznovrsnost disciplina 3
Primena veštačke inteligencije 4
Simuliranje inteligencije - Turing Test 7
AI alatke i modeli učenja 7
Klasifikacija i predviđanje 8
Modeli učenja 8
Uloga Pythona u veštačkoj inteligenciji 9
Zašto je Python dominantan u mašinskom učenju, nauci o podacima i AI-i? 9
Anaconda u Pythonu 10
Python biblioteke za veštačku inteligenciju 11
Kratak uvod u NumPy biblioteku 12
Vežba 1: Operacije matrice upotrebom NumPy biblioteke 14
Python za AI igrice 16
Inteligentni agenti u igrama 16
Breadth First Search i Depth First Search 17
Istraživanje prostora stanja igre 20
Vežba 2: Procena broja mogućih stanja u Tic-Tac-Toe igri 22
Vežba 3: Kreiranje AI-e nasumično 23
Aktivnost 1: Generisanje svih mogućih sekvenci koraka u Tic-Tac-Toe igri 26
Rezime 27

POGLAVLJE 2

AI u tehnikama pretrage i igrama 29

Uvod 30
Vežba 4: Učenje agenta da pobedi 30
Aktivnost 2: Učenje agenta da shvati situacije kada se brani od gubitka 32
Aktivnost 3: Ispravljanje prvog i drugog poteza AI igrača da bismo ga učinili nepobedivim 33
Heuristika 35
Neinformisana i informisana pretraga 35
Kreiranje heuristike 35
Dopustiva i nedopustiva heuristika 36
Heuristička evaluacija 36
Vežba 5: Tic-Tac-Toe statička evaluacija pomoću heurističke funkcije 39
Upotreba heuristike za informisanu pretragu 41
Tipovi heuristike 41
Pronalaženje putanje pomoću A* algoritma 42
Vežba 6: Pronalaženje najkraće putanje za postizanje cilja 44
Vežba 7: Pronalaženje najkraće putanje upotrebom BFS-a 45
Predstavljanje A* algoritma 47
A* pretraga u praksi upotrebom simpleai biblioteke 60
AI igre sa Minmax algoritmom i Alfa-Beta odsecanjem 63
Algoritmi pretrage za multiplejer igre zasnovane na potezu 64
Minmax algoritam 65
Optimizacija Minmax algoritma Alpha-Beta odsecanjem 70
Upotreba DRY koda za Minmax algoritam - NegaMax Algoritam 73
Upotreba EasyAI biblioteke 74
Aktivnost 4: Connect Four 77
Rezime 79

POGLAVLJE 3

Regresija 81

Uvod 82
Linearna regresija jedne promenljive 82
Šta je regresija? 83
Funkcije i oznake 86
Skaliranje funkcije 87
Unakrsna provera pomoću trening i test podataka 89
Podešavanje modela podataka pomoću biblioteke scikit-learn 90
Linearna regresija pomoću NumPy nizova 92
Uklapanje modela pomoću NumPy Polyft biblioteke 97
Predviđanje vrednosti pomoću linearne regresije 104
Aktivnost 5: Predviđanje stanovništva 105
Linearna regresija više promenljivih 106
Višestruka linearna regresija 106
Proces linearne regresije 107

Importovanje podataka iz izvora podataka ... 108
Učitavanje cena akcija pomoću Yahoo Financea ... 108
Učitavanje fajlova pomoću biblioteke pandas ... 109
Učitavanje cena akcija pomoću Quandla ... 109
Vežba 8: Upotreba Quandla za učitavanje cena akcija ... 110
Priprema podataka za predikciju ... 111
Izvršavanje i validacija linearne regresije ... 116
Predviđanje budućnosti ... 118
Polinomna regresija i regresija potpornih vektora ... 122
Polinomna regresija jedne promenljive ... 123
Vežba 9: 1, 2. i 3. stepen polinomne regresije ... 124
Polinomna regresija više promenljivih ... 127
Support Vector regresija ... 129
Support Vector Machines sa polinomijalnim kernelom 3. stepena ... 132
Aktivnost 6: Predikcija cena akcija pomoću kvadratne i kubne linearne polinomne regresije više promenljivih ... 134
Rezime ... 135

POGLAVLJE 4

Klasifikacija ... 137

Uvod ... 138
Osnove klasifikacije ... 138
Vežba 10: Učitavanje skupova podataka ... 139
Obrada podataka ... 142
Vežba 11: Pretprocesiranje podataka ... 145
Minmax skaliranje kolone goal ... 147
Identifikovanje funkcija i oznaka ... 148
Unakrsna provera pomoću biblioteke scikit-learn ... 148
Aktivnost 7: Priprema kreditnih podataka za klasifikaciju ... 149
Klasifikator k-najbližih suseda ... 149
Predstavljanje k-nearest neighbor algoritma ... 150
Funkcije rastojanja ... 150
Vežba 12: Ilustracija algoritma klasifikatora k-najbližeg suseda ... 152
Vežba 13: Klasifikacija k-najbližeg suseda u biblioteci scikit-learn ... 155
Vežba 14: Predikcija pomoću klasifikatora k-najbližih suseda ... 156
Parametarizacija klasifikatora k-najbližih suseda u scikit-learn biblioteci ... 156
Aktivnost 8: Povećanje tačnosti kreditnog bodovanja ... 157
Klasifikacija pomoću metoda potpornog vektora ... 157
Šta su klasifikatori metoda potpornog vektora ... 158
Razumevanje metoda potpornog vektora ... 159
Metod potpornih vektora u biblioteci scikit-learn ... 162
Parametri scikit-learn SVM-a ... 163
Aktivnost 9: Optimizacija metoda potpornog vektora u biblioteci scikit-learn ... 163
Rezime ... 165

POGLAVLJE 5

Upotreba stabala za prediktivnu analizu 167

Uvod u stabla odlučivanja 168
Entropija 172
Vežba 15: Izračunavanje entropije 174
Informacioni dobitak 176
Gini Impurity 177
Izlazni uslov 178
Izgradnja klasifikatora stabla odlučivanja pomoću biblioteke scikit-learn 179
Evaluacija performanse klasifikatora 180
Vežba 16: Preciznost i odziv 182
Vežba 17: Izračunavanje F1 Scorea 183
Matrica konfuzije 186
Vežba 18: Matrica konfuzije 186
Aktivnost 10: Klasifikacija podataka automobila 188
Klasifikator Random Forest 189
Konstruisanje slučajne šume 190
Klasifikacija slučajne šume pomoću biblioteke scikit-learn 191
Parametarizacija klasifikatora slučajne šume 192
Važnost funkcije 192
Extremly Randomized Trees 193
Aktivnost 11: Klasifikacija slučajne šume za kompaniju za iznajmljivanje automobila 193
Rezime 195

POGLAVLJE 6

Uvod u klasterizaciju. 197

Uvod u klasterizaciju 198
Definisanje problema klasterizacije 198
Pristupi klasterizacije 201
Algoritmi klasterizacije koje podržava biblioteka scikit-learn 201
k-means algoritam 202
Vežba 19: k-means u scikit-learn biblioteci 203
Parametarizacija k-means algoritma u scikit-learn biblioteci 208
Vežba 20: Preuzimanje centralnih tačaka i oznaka 208
k-means klasterizacija podataka prodaje 209
Aktivnost 12: k-means klasterizacija podataka o prodaji 209
Mean Shift algoritam 210
Vežba 21: Ilustracija mean shift algoritma u 2D-u 211
Mean Shift algoritam u biblioteci scikit-learn 216
Obrada slike u Pythonu 218
Aktivnost 13: Prepoznavanje oblika pomoću Mean Shift algoritma 220
Rezime 222

POGLAVLJE 7

Duboko učenje pomoću neuronskih mreža 223

Uvod 224
TensorFlow biblioteka za Python 224
Instaliranje biblioteke TensorFlow u Anaconda Navigator 225
Tensor Flow operacije 226
Vežba 22: Upotreba osnovnih operacija i TensorFlow konstanti 226
Rezervisana mesta i promenljive 227
Inicijalizator globalnih promenljivih 228
Uvod u neuronske mreže 229
Biasi 233
Primeri za veštačke neuronske mreže 234
Aktivacione funkcije 235
Vežba 23 Aktivacione funkcije 238
Propagacija unapred i unazad 240
Konfigurisanje neuronske mreže 241
Importovanje TensorFlow skupa podataka cifara 241
Modelovanje funkcija i oznaka 242
TensorFlow modelovanje za više oznaka 244
Optimizovanje promenljivih 245
Obučavanje TensorFlow modela 247
Upotreba modela za predikciju 247
Testiranje modela 248
Randomizacija veličine uzorka 249
Aktivnost 14: Detekcija pisanih cifara 250
Duboko učenje 251
Dodavanje slojeva 251
Konvolutivne neuronske mrže 252
Aktivnost 15: Detekcija pisanih cifara pomoću dubokog učenja 253
Rezime 255

DODATAK A

Poglavlje 1, „Principi veštačke inteligencije“ 258
Aktivnost 1: Generisanje svih mogućih sekvenci koraka u Tic-Tac-Toe igri 258
Poglavlje 2, „AI u tehnikama pretrage i igrama“ 261
Aktivnost 2: Učenje agenta da shvati situacije kada se brani od gubitka 261
Aktivnost 3: Ispravljanje prvog i drugog poteza AI igrača da bismo ga učinili nepobedivim 263
Aktivnost 4: Connect Four 265
Poglavlje 3, „Regresija“ 269
Aktivnost 5: Predviđanje stanovništva 269
Aktivnost 6: Predikcija cena akcija pomoću kvadratne i kubne linearne polinomne regresije više promenljivih 271
Poglavlje 4: Klasifikacija 276

Aktivnost 7: Priprema kreditnih podataka za klasifikaciju ... 276
Aktivnost 8: Povećanje tačnosti kreditnog bodovanja ... 280
Aktivnost 9: Optimizacija metoda potpornog vektora u biblioteci scikit-learn ... 280
Poglavlje 5, „Upotreba stabala za prediktivnu analizu“ ... 282
Aktivnost 10: Klasifikacija podataka automobila ... 282
Aktivnost 11: Klasifikacija slučajne šume za kompaniju za iznajmljivanje automobila ... 285
Poglavlje 6, „Klasterizacija“ ... 291
Aktivnost 12: k-means klasterizacija podataka prodaje ... 291
Aktivnost 13 Prepoznavanje oblika pomoću Mean Shift algoritma ... 293
Poglavlje 7, „Duboko učenje pomoću neuronskih mreža“ ... 298
Aktivnost 14: Detekcija pisanih cifara ... 298
Aktivnost 15: Detekcija pisanih cifara pomoću dubokog učenja ... 302

INDEKS ... 307

UVOD

O uvodu

U ovom odeljku ćemo kratko predstaviti autora, šta obuhvata knjiga, tehničke veštine koje će vam biti potrebne za početak rada i hardverske i softverske zahteve potrebne za izvršavanje svih uključenih aktivnosti i vežbi.

O KNJIZI

Mašinsko učenje i neuronske mreže brzo postaju stubovi na kojima možete da gradite inteligentne aplikacije. Na početku knjige predstavljen je Python i opisana je upotreba algoritama AI pretrage. Učićete o matematički teškim temama, kao što su regresiona analiza i klasifikacija, koje su ilustrovane Python primerima.

U nastavku knjige fokusiraćemo se na naprednije AI tehnike i koncepte i koristićemo skupove podataka iz realnog sveta da bismo formirali stablo odlučivanja i klastere. Predstavićemo vam neuronske mreže, moćan alat koji ima koristi od Murovog zakona koji je primenjen na računarsku moć 21.veka. Kada završite čitanje knjige bićete nestrpljivi da gradite sopstvene AI aplikacije, koristeći novostečene veštine.

O autoru

Zsolt Nagy (Žolt Nađ) je inženjerski rukovodilac u jednoj tehničkoj kompaniji za nauku o podacima. Nakon što je stekao master diplomu na temu zaključka o ontologijama, uglavnom je koristio AI za analizu online poker strategija da bi pomogao profesionalnim igračima pokera u donošenju odluka. Nakon toga, trud je uložio u svoj stručni profil u rukovodstvu i softverskom inženjeringu.

Ciljevi

- **razumevanje** važnosti, principa i polja AI-a
- **učenje** kako se upotrebljava Python za implementiranje osnovne veštačke inteligencije za igrice
- **implementiranje** vežbi regresije i klasifikacija u Pythonu, predstavljenih primerima iz realnog sveta
- **izvršavanje** prediktivne analize u Pythonu upotrebom stabala odlučivanja i random foresta
- **izvršavanje** klasterovanja u Pythonu upotrebom k-means i mean shift algoritama
- **razumevanje** osnova dubokog učenja pomoću praktičnih primera

Auditorijum

Ova knjiga je namenjena onim programerima koji smatraju da im je budućnost unosnija kao istraživačima podataka ili koji žele da upotrebe mašinsko učenje da bi obogatili svoje aktuelne lične i profesionalne projekte. Prethodno poznavanje AI-e nije potrebno, ali je potrebno poznavanje bar jednog programskog jezika (po mogućstvu, Pythona) i matematike za srednje škole. Iako je ovo knjiga za početni nivo AI-e, napredniji studenti će moći da poboljšaju poznavanje Pythona implementiranjem praktičnih aplikacija, koristeći svoje osnovno znanje o AI-i.

Pristup

U ovoj knjizi upotrebljen je praktičan pristup učenju o veštačkoj inteligenciji i mašinskom učenju upotrebom Pythona. Ona sadrži više aktivnosti koje koriste stvarne scenarije za vežbu i primenu novih veština u visokorelevantnom kontekstu.

Minimalni hardverski zahtevi

Za optimalno iskustvo učenja preporučujemo sledeću hardversku konfiguraciju:

- **procesor:** Intel Core i5 ili ekvivalent
- **memorija:** 8 GB RAM
- **skladište:** 35 GB slobodnog prostora

Softverski zahtevi

Takođe treba da imate instaliran sledeći softver:

- **OS:** Windows 7 SP1 64-bit, Windows 8.1 64-bit ili Windows 10 64-bit, UbuntuLinux ili najnoviju verziju macOS-a
- **pretraživač:** Google Chrome (najnovija verzija)
- **Anaconda** (najnovija verzija)
- **IPython** (najnovija verzija)

Konvencije

Reči koda u tekstu, nazivi tabela baze podataka, nazivi direktorijuma, nazivi fajlova, ekstenzije fajlova, nazivi putanja, lažni URL-ovi, korisnički unos i Twitter postovi prikazani su na sledeći način: „Najčešće upotrebljavane funkcije za aktivaciju su sigmoid i tanh (Hyperbolic tangent funkcije).“

Blok koda je prikazan na sledeći način:

```
from sklearn.metrics.pairwise import euclidean_distances
points = [[2,3], [3,7], [1,6]]
euclidean_distances([[4,4]], points)
```

Novi termini i važne reči ispisani su zadebljanim slovima. Reči koje vidite na ekranu - na primer, u menijima i okvirima za dijalog, prikazane su u tekstu na sledeći način: „Optimalni razdelnik koji nalaze metodi potpornih vektora naziva se **best separatinghyperplane**."

Instalacija i postavka

Pre nego što započnete čitanje ove knjige, treba da imate instalirane Python 3.6 i Anacondu. Korake za instalaciju ćete pronaći na sledećim lokacijama:

Instaliranje Pythona

Instalirajte Python verziju 3.6, prateći instrukcije na linku

http://bit.ly/2OXWuHo

Instaliranje virtuelnog okruženja

Anaconda je važan program za izbegavanje konflikta paketa i uštedu vremena/energije izbegavanjem frustrirajućih grešaka.

Da biste instalirali Anaconda program, kliknite na link

http://bit.ly/2MO2nnR

Izaberite operativni sistem i izaberite najnoviju verziju Pythona. Kada je paket preuzet, pokrenite ga.

Nakon što kliknete na Next, videćete ugovor o licenciranju. Kada kliknete na dugme **I Agree**, možete da izaberete da li želite da instalirate Anaconda program za sebe ili za sve korisnike na računaru. Ako želite da instalirate za sve korisnike, potrebna su vam prava administratora. Izaberite opciju **Just Me**.

Zatim, treba da izaberete direktorijum u koji želite da instalirate Anaconda program. Uverite se da u nazivu direktorijuma nema razmaka i dugih Unicode karaktera. Uverite se da imate najmanje 3 GB prostora na računaru i da imate internet konekciju dovoljno brzu za preuzimanje fajla.

Na sledećem ekranu možete da izaberete da li želite da dodate Anaconda program u **PATH** promenljivu okruženja. Nemojte da izaberete ovu opciju, jer ćete moći da pokrenete Anaconda iz **Start** menija.

Kliknite na **Install**. Potrebno je nekoliko minuta da se Anaconda instalira na računar. Nakon što je instalacija završena, možete da izaberete da naučite više o Anaconda Cloudu i Anaconda Supportu ili možete da isključite ta polja i završite instalaciju.

Pokretanje Anaconda programa

Instalirani program Anaconda pronaći ćete u Start meniju. Ako već imate instaliran program pre početka čitanja ove knjige, možete da izaberete da ga nadgradite na Python 3. Najčistiji način da to uradite je da ga deinstalirate i ponovo instalirate.

Anaconda Navigator obezbeđuje pristup za većinu alatki koje će vam biti potrebne za ovu knjigu. Pokrenite **IPython**, tako što ćete kliknuti na opciju gore-desno.



Jupyter Notebook je mesto na kojem ćete izvršiti Python kod za ovu knjigu.

Dodatni resursi

Kompletan izvorni kod za knjigu možete da preuzmete sa našeg sajta:

http://bit.ly/2Bg9vnR

Osim toga, paket koda za ovu knjigu pronaći ćete i na GitHubu:

http://bit.ly/2VKdoLf

1

Principi veštačke inteligencije

Ciljevi učenja

Do kraja ovog poglavlja moći ćete da:

- opišete različita polja AI-e
- objasnite glavne modele učenja upotrebljene u AI-i
- objasnite zašto je Python popularan jezik za AI projekte
- modelujete pretragu prostora u AI-i za odgovarajuću igru

U ovom poglavlju ćete učiti o svrsi, poljima i oblicima primene AI-e, a videćete i kratak rezime funkcija koje ćemo upotrebiti u Pythonu.

UVOD

Pre nego što počnemo razmatranje različitih AI tehnika i algoritama, opisaćemo osnove veštačke inteligencije i mašinskog učenja i objasniti nekoliko osnovnih definicija. Zatim ćemo nastaviti navođenjem različitih primera. Primeri iz stvarnog sveta će biti upotrebljeni za predstavljanje osnovnih koncepata veštačke inteligencije na jasan način.

Ako želite da budete stručnjak u nečemu, treba da imate veoma dobru osnovu. Pa, započnimo sada definicijom veštačke inteligencije:

Definicija: Veštačka inteligencija (AI) je nauka koja se koristi za konstruisanje inteligencije upotrebom hardverskih i softverskih rešenja.

Inspirisana je obrnutim inženjerstvom - na primer, na način na koji funkcionišu neuroni u ljudskom mozgu. Naš mozak se sastoji od malih jedinica koje se nazivaju neuroni, a njihova mreža se naziva neuronska mreža. Osim neuronskih mreža, postoje mnogi drugi modeli u neuronauci koji mogu da se upotrebe za rešavanje problema iz stvarnog sveta u veštačkoj inteligenciji.

Mašinsko učenje je termin koji se često meša sa veštačkom inteligencijom. Prvi ga je definisao Arthur Lee Samuel 1959. godine.

Definicija: Mašinsko učenje je polje nauke koje se bavi pružanjem mogućnosti računarima da uče bez eksplicitnog programiranja.

Tom Mitchell je predložio matematički precizniju definiciju mašinskog učenja.

Definicija: Kaže se da kompjuterski program uči iz iskustva E, vezanog za zadatak T i meru performanse P, ukoliko se njegove performanse u T, merene metrikama P, unapređuju iskustvom E.

Iz ove dve definicije možemo zaključiti da je mašinsko učenje način za postizanje veštačke inteligencije. Međutim, možete da imate veštačku inteligenciju bez mašinskog učenja. Na primer, ako eksplicitno zadate pravila i stabla odlučivanja ili primenite tehnike pretraživanja, kreirate agenta veštačke inteligencije, čak i ako vaš pristup nema nikakve veze sa mašinskim učenjem.

Kako AI rešava probleme iz realnog sveta?

Veštačka inteligencija automatizuje ljudsku inteligenciju na osnovu načina na koji ljudski mozak obrađuje informacije.

Kad god rešimo problem ili komuniciramo sa ljudima, prolazimo kroz proces. Kad god ograničimo opseg problema ili interakcije, ovaj proces često može da bude modelovan i automatizovan.

AI čini da računari razmišljaju kao ljudi.

Ponekad, izgleda kao da AI „zna“ šta nam je potrebno. Samo razmislite o personalizovanim kuponima koje primate nakon online kupovine. Do kraja ove knjige razumećete zašto, ako želite da izaberete najuspešnije proizvode, treba da vam se pokaže kako da maksimizujete porudžbine - to je relativno jednostavan zadatak. Međutim, često mislimo da računari „znaju“ sve što nam je potrebno.

AI izvršavaju računari koji izvršavaju instrukcije nižeg nivoa.

Iako se rešenje možda čini inteligentnim, mi pišemo kod isto kao i za bilo koje drugo softversko rešenje. Čak i ako simuliramo neurone, jednostavan mašinski kod i računarski hardver izvršavaju proces „razmišljanja“.

Većina AI aplikacija ima jedan primarni cilj. Kada komuniciramo sa AI aplikacijom, čini se kao da je ljudska, jer može da ograniči domen problema na primarni cilj. Prema tome, možemo da razdvojimo složene procese i simuliramo inteligenciju pomoću računarskih instrukcija nižeg nivoa.

AI može da stimuliše ljudska čula i procese razmišljanja za specijalizovane oblasti.

Treba da simuliramo ljudska čula i razmišljanje i ponekad da prevarimo AI da veruje da komuniciramo sa drugim ljudskim bićem. U posebnim slučajevima možemo čak da poboljšamo i naša čula.

Slično tome, kada komuniciramo sa chatbotom, očekujemo da nas bot razume. Očekujemo da chatbot ili čak sistem za prepoznavanje govora obezbede interfejs između računara i čoveka. Da bi ispunili naša očekivanja, računari treba da simuliraju procese ljudskog razmišljanja.

Raznovrsnost disciplina

Automobil koji se kreće samostalno i ne može da „oseti“ da se i drugi automobili kreću na istom autoputu bio bi veoma opasan. AI agent treba da obradi i „oseti“ šta se dešava oko njega da bi mogao da vozi automobil. Međutim, to, samo po sebi, nije dovoljno. Bez razumevanja fizike objekata koji se kreću, vožnja automobila u normalnom okruženju bila bi skoro nemoguć, da ne kažem smrtonosan, zadatak.

Da bismo kreirali upotrebljivo AI rešenje, uključene su različite discipline. Na primer:

- **robotika** - za pomeranje objekata u prostoru
- **teorija algoritama** - za konstruisanje efikasnih algoritama
- **statistika** - za izvođenje korisnih rezultata, predviđanje budućnosti i analizu prošlosti

- **psihologija** - za modelovanje načina funkcionisanja ljudskog mozga
- **softversko inženjerstvo** - za kreiranje rešenja koja se mogu održavati i koja mogu izdržati test vremena
- **računarska nauka ili računarsko programiranje** - za implementiranje softverskih rešenja u praksi
- **matematika** - za izvršavanje složenih matematičkih operacija
- **teorija kontrole** - za kreiranje feed-forward i feedback sistema
- **teorija informacije** - za predstavljanje, kodiranje, dekodiranje i kompresovanje informacija
- **teorija grafikona** - za modelovanje i optimizovanje različitih tačaka u prostoru i za predstavljanje hijerarhija
- **fizika** - za modelovanje realnog sveta
- **računarska grafika i obrada slika** - za prikaz i obradu slika i filmova

U ovoj knjizi ćemo opisati neke od ovih disciplina. Ne zaboravite - mi se sada fokusiramo na visok nivo razumevanja veštačke inteligencije.

PRIMENA VEŠTAČKE INTELIGENCIJE

Sada, kada znate šta je veštačka inteligencija, istražićemo različite oblasti u kojima se ona primenjuje.

Simulacija ljudskog ponašanja

Ljudi imaju pet osnovnih čula: vida, sluha, dodira, mirisa i ukusa. Međutim, da bismo bolje razumeli kako da kreiramo inteligentne mašine, možemo da razdvojimo discipline na sledeći način:

- slušanje i govor
- razumevanje jezika
- pamćenje
- razmišljanje
- vid
- pokret

Neke od ovih disciplina su nam u ovom razmatranju van opsega, jer je cilj u ovoj knjizi da razumete osnove. Na primer, kada želimo da pomerimo robotsku ruku, treba da poznajemo složenu matematiku visokog nivoa da bismo razumeli šta se dešava.

Slušanje i govor

Upotrebom sistema za prepoznavanje govora AI može da sakuplja informacije. Upotrebom sinteze govora može da pretvori interne podatke u razumljive zvukove. Tehnike prepoznavanja govora i sinteze govora prepoznaju i konstruišu zvukove koje ljudi emituju ili koje ljudi mogu da razumeju.

Zamislite da ste na putovanju u državi u kojoj se govori jezikom koji vi ne razumete. Možete da govorite u mikrofon svog telefona, očekujući da telefon „razume" ono što vi kažete, a zatim da to prevede na drugi jezik. Isto se dešava i obratno - kada lokalni stanovnik govori, a AI prevodi zvukove na jezik koji razumete. Prepoznavanje govora i sinteza govora to omogućavaju.



Primer sinteze govora je Google Translate. Kliknite na https://translate.google.com/ i na dugme sa zvučnikom ispod prevedene reči da bi prevodilac glasno izgovorio reči na jeziku koji nije englski.

Razumevanje jezika

Možemo da razumemo prirodni jezik ako ga obradimo. Ova oblast se naziva **obrada prirodnog jezika**, ili, skraćeno, NLP.

Kada je reč o obradi prirodnog jezika, želimo da naučimo jezike na osnovu **statističkog učenja**.

Pamćenje

Treba svetu da predstavimo nešto što poznajemo. Za to kreiramo **baze znanja** i hijerarhijske reprezentacije pod nazivom **ontologije**. Ontologije kategorišu elemente i ideje u našem svetu i sadrže veze između ovih kategorija.

Razmišljanje

Naš AI sistem treba da bude ekspert u određenom domenu upotrebom **ekspertskog sistema**, koji može da bude zasnovan na matematičkoj logici na deterministički način, kao i na nejasan, nedeterministički način.

Baza znanja ekspertskog sistema je predstavljena upotrebom različitih tehnika. Kako problem domena raste, kreiramo hijerarhijske ontologije.

Možemo da repliciramo ovu strukturu modelovanjem mreže na gradivnim blokovima mozga. Ovi gradivni blokovi se nazivaju neuroni, a sama mreža se naziva **neuronska mreža**.

Postoji još jedan ključni termin koji treba da povežemo sa neuronskim mrežama - **duboko učenje**. To učenje je duboko, jer prevazilazi obrazac prepoznavanja i kategorizacije. Ono je utisnuto u neuronsku strukturu mreže. Na primer, specijalni zadatak dubokog učenja je **prepoznavanje objekta** pomoću **računarskog vida**.

Vid

Treba da komuniciramo sa realnim svetom pomoću naših čula. Do sada smo govorili samo o čulu sluha, o prepoznavanju i sintezi govora. Šta se dešava ako treba da vidimo nešto? U tom slučaju treba da kreiramo tehnike **računarskog vida** za učenje o okruženju. Prepoznavanje lica je korisno i većinom su ljudi u tome eksperti.

Računarski vid zavisi od obrade slike. Iako obrada slike nije direktno AI disciplina, ona je potrebna disciplina za AI.

Pokret

Pokret i dodir su prirodni za nas ljude, ali su veoma složeni zadaci za računare. Pokret se vrši pomoću **robotike**. To je tema koja je matematički zahtevna.

Robotika je zasnovana na **teoriji kontrole**, gde kreiramo povratnu petlju i kontrolišemo pokret objekta na osnovu sakupljenih povratnih informacija. Interesantno je da teorija kontrole ima primenu i u drugim oblastima koje apsolutno nemaju nikakve veze sa pokretanjem objekata u prostoru. Razlog je činjenica da su potrebne povratne petlje slične onima koje su modelovane u ekonomiji.

Simuliranje inteligencije - Turing Test

Alan Turing, pronalazač Turing mašine, apstraktnog koncepta koji se koristi u teoriji algoritma, predložio je način za testiranje inteligencije. Ovaj test se u AI literaturi naziva **Turing test**.

Koristeći tekstualni interfejs, ispitivač četuje sa čovekom i chatbotom. Zadatak chatbota je da prevari ispitivača do tačke da ne može da kaže da li je on čovek ili ne.

Koje discipline su nam potrebne da bismo uspešno završili Turing test?

Pre svega, treba da razumemo govorni jezik da bismo znali šta ispitivač govori, što ćemo postići upotrebom **Natural Language Processinga (NLP)**. Takođe je neophodno da ispitivaču odgovorimo.

Treba da budemo stručnjaci za ono za šta je ljudski mozak zainteresovan. Takođe treba da izgradimo **ekspertski sistem** čovečanstva, uključujući taksonomiju objekata i apstraktna razmišljanja u našem svetu, kao i istorijske događaje, pa, čak, i emocije.

Uspešno završavanje Turing testa je veoma teško. Trenutna predviđanja su da nećemo moći da kreiramo sistem dovoljno dobar da prođe Turing test do kraja naredne decenije. Međutim, to nije sve - možemo da pređemo i na Total Turing Test koji takođe uključuje pokret i vid.

AI ALATKE I MODELI UČENJA

U prethodnim odeljcima otkrili smo osnove veštačke inteligencije. Jedan od osnovnih zadataka za tu inteligenciju je učenje.

Inteligentni agenti

Kada rešavamo AI probleme, kreiramo u okruženju aktera koji može da sakuplja podatke iz okruženja i utiče na svoje okruženje. Ovaj akter se naziva inteligentni agent.

Inteligentan agent je:

- autonoman
- Posmatra svoje okruženje pomoću senzora.
- Deluje u svom okruženju upotrebom aktuatora.
- Usmerava svoje aktivnosti ka postizanju ciljeva.

Agent može i da uči i ima pristup bazi znanja.

Agenta možemo da zamislimo kao funkciju koja mapira percepcije na akcije. Ako agent ima internu bazu znanja, percepcije, akcije i reakcije mogu da promene bazu znanja.

Akcije mogu da budu nagrađene ili kažnjene. Postavljanje tačnog cilja i implementiranje primera nekih situacija pomaže agentu da uči. Ako je cilj tačno podešen, agenti imaju mogućnost da pobede često mnogo složeniji ljudski mozak, jer je prvi cilj ljudskog mozga opstanak, bez obzira na igru koju igramo, a prvi motiv agenta je da sam dostigne cilj. Prema tome, inteligentni agenti se neće osramotiti kada izvrše nasumični pokret bez ikakvog znanja.

Klasifikacija i predviđanje

Različiti ciljevi zahtevaju različite procese. Istražimo sada najpopularnije tipove AI rasuđivanja - klasifikaciju i predviđanje.

Klasifikacija je proces za otkrivanje kako objekat može da bude definisan u odnosu na drugi objekat. Na primer, otac je muškarac koji ima jedno dete ili više dece. Ako je Džejn roditelj deteta, a Džejn je žensko, onda je ona majka. Osim toga, ona je ljudsko biće, sisar i živi organizam. Znamo da ima nacionalnost i datum rođenja.

Predviđanje je proces koji se vrši na osnovu obrazaca i verovatnoće. Na primer, ako kupac u standardnom supermarketu kupi organsko mleko, isti kupac će kupiti organski jogurt verovatnije nego prosečan kupac.

Modeli učenja

Proces AI učenja može da se izvrši na nadgledan i nenadgledan način. Nadgledano učenje je zasnovano na obeleženim podacima i zaključivanju funkcija iz podataka za trening. Linearna regresija je jedan primer. Nenadgledano učenje je zasnovano na neobeleženim podacima i često funkcioniše na analizama klastera.

ULOGA PYTHONA U VEŠTAČKOJ INTELIGENCIJI

Da bismo primenili osnovne AI koncepte u praksi, potreban nam je programski jezik koji podržava veštačku inteligenciju. U ovoj knjizi izabrali smo Python. Postoji nekoliko razloga zbog kojih je on dobar izbor za AI:

- Python je **programski jezik visokog nivoa**. To znači da ne treba da brinemo o dodeli memorije, pokazivačima ili mašinskom kodu uopšte. Možemo da pišemo kod na pogodan način i da se oslonimo na robusnost Pythona. Python je takođe kompatibilan sa različitim platformama.
- Snažan naglasak na **doživljaj programera** čini Python veoma popularnim izborom među softverskim programerima. U stvari, prema istraživanju programera svih starosnih grupa za 2018. godinu na sajtu https://www.hackerrank.com, Python je rangiran na prvom mestu kao omiljeni jezik softverskih programera, zato što je čitljiv i jednostavan. Prema tome, Python je odličan za brzo razvijanje aplikacija.
- Bez obzira što je interpretirani jezik, Python je uporediv sa drugim jezicima koji se koriste u nauci o podacima, kao što je R. Glavna prednost je **efikasnost memorije**, jer Python može da rukuje velikim bazama podataka u memoriji.



Python je višenamenski jezik. Može da se upotrebi za kreiranje desktop aplikacija, aplikacija baze podataka, mobilnih aplikacija i igara. Takođe treba pomenuti njegove funkcije za programiranje mreže. Štaviše, Python je odlična alatka za kreiranje prototipa.

Zašto je Python dominantan u mašinskom učenju, nauci o podacima i AI-i?

Da biste razumeli dominantnu prirodu Pythona u mašinskom učenju, nauci o podacima i AI-i, uporedićemo Python sa drugim jezicima koji se takođe koriste u ovim oblastima.

Jedna od glavnih alternativa je jezik R. Python je u prednosti u odnosu na R, zato što je jezik opšte namene i praktičniji.

Pisanje programa u Pythonu je značajno brže nego u jezicima Java i C++. Python takođe obezbeđuje visok stepen fleksibilnosti.

Postoje neki jezici koji su slični po prirodi kada je u pitanju fleksibilnost i pogodnost, kao što su Ruby i JavaScript. Python ima prednost u odnosu na ove jezike zbog AI ekosistema koji je dostupan za njega. U bilo kojoj oblasti podrška otvorenog koda i biblioteka iz drugih izvora uveliko određuju uspeh određenog jezika. Pythonova podrška za AI biblioteke iz drugih izvora je odlična.

Anaconda u Pythonu

Već smo instalirali Anaconda program u uvodu. On će biti izabrana alatka kada je u pitanju eksperimentisanje sa veštačkom inteligencijom.

Ova lista biblioteka je nepotpuna, jer postoji više od 700 biblioteka koje su dostupne u Anacondi. Ako poznajete ove biblioteke, imate dobru početnu tačku za rad, jer ćete moći da implementirate osnovne AI algoritme u Python.

Anaconda sadrži na jednom mestu pakete, IDE-je, biblioteku za vizuelizaciju podataka i alatke visoke performanse za paralelno računarstvo. Ona skriva probleme sa konfiguracijom i složenost održavanja steka za nauku o podacima, mašinsko učenje i veštačku inteligenciju. Ova funkcija je posebno korisna u Windowsu, gde se nepoklapanje verzije i problemi sa konfiguracijom javljaju najčešće.

Anaconda ima IPython konzolu, u kojoj možemo da pišemo kod i komentare u stilu dokumentacije. Kada eksperimentišemo sa AI funkcijama, tok ideja podseća na interaktivni vodič u kojem pokrećemo svaki korak koda.



IDE je skraćenica za Integrated Development Environment. Iako editor za tekst obezbeđuje neke funkcionalnosti za isticanje i formatiranje koda, IDE prevazilazi funkcije editora za tekst obezbeđivanjem alatki za automatsku preradu, testiranje, ispravljanje grešaka, pakovanje, pokretanje i raspoređivanje koda.

Python biblioteke za veštačku inteligenciju

Lista biblioteka predstavljenih ovde nije kompletna, jer postoji više od 700 dostupnih biblioteka u Anacondi. Međutim, ove specifičnost će vam pomoći da započnete učenje, jer će vam pružiti dobru osnovu da možete da implementirate osnovne AI algoritme u Pythonu:

- **NumPy** - NumPy je biblioteka za analizu podataka za Python. Pošto Python nema ugrađenu strukturu niza podataka, treba da upotrebimo biblioteku za efikasno modelovanje vektora i matrica. U nauci o podacima potrebne su nam ove strukture podataka da bismo izvršili jednostavne matematičke operacije. U budućim modelima ćemo često koristiti biblioteku NumPy.
- **SciPy** - SciPy je napredna biblioteka koja sadrži algoritme koji se koriste u nauci o podacima. To je odlična komplementarna biblioteka za NumPy, jer pruža sve napredne algoritme koji su nam potrebni, bez obzira da li su to algoritmi linearne algebre, alatka za obradu slika ili operacija nad matricama.
- **pandas** – Obezbeđuje brze, fleksibilne i ekspresivne strukture podataka, kao što su jednodimenzionalne serije i dvodimenzionalni DataFrames. Efikasno učitava formate i obrađuje složene tabele različitog tipa.
- **scikit-learn** - Pythonova glavna biblioteka za mašinsko učenje je scikit-learn. Zasnovana je na NumPy i SciPy bibliotekama. Obezbeđuje funkcionalnost koja je potrebna za izvršavanje klasifikacije i regresije, obradu podataka i za nadgledano i nenadgledano učenje.
- **NLTK** - Mi se nećemo u ovoj knjizi baviti obradom prirodnog jezika, ali NLTK biblioteku ipak vredi pomenuti, jer je glavni skup alatki za prirodni jezik za Python. Možemo da izvršimo klasifikaciju, deljenje u tokene, pretraživanje od korena, označavanje, raščlanjavanje, semantičko rezonovanje i mnoge druge usluge upotrebom ove biblioteke.
- **TensorFlow** - TensorFlow je „Googleova“ biblioteka neuronske mreže i savršena je za implementiranje dubokog učenja veštačke inteligencije. Fleksibilna osnova TensorFlow biblioteke može da se upotrebi za rešavanje različitih problema numeričkog izračunavanja. Neki oblici primene ove biblioteke u realnom svetu uključuju „Googleovo“ prepoznavanje glasa i identifikaciju objekta.

Kratak uvod u NumPy biblioteku

NumPy biblioteka će imati glavnu ulogu u ovoj knjizi, pa ćemo je detaljnije istražiti.

Nakon pokretanja IPython konzole, možemo jednostavno da importujemo NumPy na sledeći način:

```
import numpy as np
```

Kada je NumPy biblioteka importovana, možemo da joj pristupimo koristeći njen alijas. Na primer, NumPy sadrži efikasnu implementaciju nekih struktura podataka, kao što su vektori i matrice. Python nema ugrađenu strukturu niza, pa je niz NumPy biblioteke koristan. Pogledajte kako možemo da definišemo vektore i matrice:

```
np.array([1,3,5,7])
```

Rezultat je sledeći:

```
array([1, 3, 5, 7])
```

Možemo da deklarišemo matricu, koristeći sledeću sintaksu:

```
A = np.mat([[1,2],[3,3]])
A
```

Rezultat je sledeći:

```
matrix([[1, 2],
[3, 3]])
```

Metod niza kreira strukturu niza podataka, dok metod mat kreira matricu.

Možemo da izvršimo mnogo operacija sa matricama. To uključuje sabiranje, oduzimanje i množenje:

Sabiranje u matricama:

```
A + A
```

Rezultat je sledeći:

```
matrix([[2, 4],
        [6, 6]])
```

Oduzimanje u matricama:

```
A - A
```

Rezultat je sledeći:

```
matrix([[0, 0],
        [0, 0]])
```

Množenje u matricama:

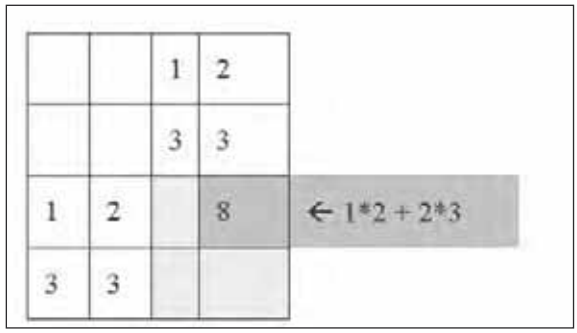
```
A * A
```

Rezultat je sledeći:

```
matrix([[ 7,    8],
        [12, 15]])
```

Sabiranje i oduzimanje matrica funkcioniše ćeliju po ćeliju.

Množenje matrica funkcioniše u skladu sa pravilima linearne algebre. Za ručno izračunavanje množenja matrica treba da usaglasimo dve matrice kao na sledećoj slici.



Slika 1.1 Izračunavanje množenja sa dve matrice

Da bismo dobili (i,j)th element matrice, izračunaćemo skalarni proizvod ith reda matrice sa jth kolonom. Skalarni proizvod dva vektora je zbir proizvoda njihovih odgovarajućih koordinata.

Još jedna operacija česta za matrice je determinant matrice. To je je broj koji je povezan sa kvadratnim matricama. Njegovo izračunavanje pomoću NumPy alatke je jednostavno:

```
np.linalg.det( A )
```

Rezultat je `-3.0000000000000004`.

Tehnički, determinant može da bude izračunat kao 1*3 – 2*3 = -3. Vidite da NumPy izračunava determinant korišćenjem aritmetike sa pokretnim zarezom, zbog čega tačnost rezultata nije savršena. Greška se javlja zbog načina na koji su brojevi za pokretnim zarezom prikazani u većini programskih jezika.

Takođe možemo da transponujemo matricu na sledeći način:

```
np.matrix.transpose(A)
```

Rezultat je sledeći:

```
matrix([[1, 3],
        [2, 3]])
```

Kada izračunavamo transponovanje matrice, mi obrćemo njene vrednosti preko njene glavne dijagonale.

NumPy ima mnogo drugih važnih funkcija i stoga ćemo je upotrebiti u nekoliko poglavlja ove knjige.

Vežba 1: Operacije matrice upotrebom NumPy biblioteke

Da bismo uradili ovu vežbu, upotrebićemo IPython i sledeću matricu. Prvo ćemo objasniti NumPy sintaksu.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Slika 1.2 Jednostavna matrica

Koristeći alatku NumPy, izračunaćemo sledeće:

- kvadrat matrice
- determinant matrice
- transponovanje matrice

Započećemo sa NumPy operacijama matrice:

1. Importovaćemo NumPy biblioteku:

   ```
   import numpy as np
   ```

2. Kreiraćemo dvodimenzionalni niz koji čuva matricu:

   ```
   A = np.mat([[1,2,3],[4,5,6],[7,8,9]])
   ```

   Vidite `np.mat` konstrukciju. Ako smo kreirali `np.array`, umesto `np.mat`, rešenje za množenje niza će biti netačno.

3. NumPy podržava množenje matrice upotrebom zvezdice:

```
A * A
```

Rezultat je sledeći:

```
matrix([[ 30,    36,   42],
        [ 66,   81,   96],
        [102, 126, 150]])
```

Kao što možete da vidite iz sledećeg koda, kvadrat A je izračunat izvršavanjem množenja matrice. Na primer, gornji levi element matrice je izračunat na sledeći način:

```
1 * 1 + 2 * 4 + 3 * 7
```

Rezultat je 30.

4. Upotrebićemo np.linalg.det za izračunavanje determinanta matrice:

```
np.linalg.det( A )
```

Rezultat je `-9.51619735392994e-16`.

Determinant je skoro nula prema prethodnim izračunavanjima. Ova neefikasnost proističe iz aritmetike sa pokretnim zarezom. Stvarni determinant je nula.

Možemo da zaključimo to ručnim izračunavanjem determinanta:

```
1*5*9 + 2*6*7 + 3*4*8 - 1*6*8 - 2*4*9 - 3*5*7
```

Rezultat je 0.

Kad god koristimo NumPy, treba da vodimo računa o mogućnosti greške pri zaokruživanju aritmetike sa pokretnim zarezom, čak i ako koristimo cele brojeve.

5. Upotrebićemo `np.matrix.transpose` da bismo transponovali matricu:

```
np.matrix.transpose(A)
```

Rezultat je sledeći:

```
matrix([[1, 4, 7],
[2, 5, 8],
[3, 6, 9]])
```

Ako je T transponovanje matrice A, onda je `T[j][i]` jednak sa `A[i][j]`.

NumPy ima mnogo korisnih funkcija za vektore, matrice i druge matematičke strukture.

PYTHON ZA AI IGRICE

AI igrač je samo **inteligentni agent** koji ima jasan cilj: da pobedi u igri. U eksperimentima za veštačku inteligenciju postignuti su iznenađujući rezultati kada su u pitanju igre. U današnje vreme ljudsko biće ne može da pobedi AI u igri šaha.

Go je poslednja igra u kojoj su velemajstori mogli stalno da pobede računarskog igrača. Međutim, 2017.godine „Googleov" AI igrač je pobedio Go velemajstora.

Inteligentni agenti u igrama

Inteligentni agent igra u skladu sa pravilima igre. On može da „oseti" **aktuelno stanje** igre pomoću **senzora** i može da proceni **korisnost potencijalnih koraka**. Kada pronađe najbolji mogući korak, izvršava akciju pomoću svojih pokretača. On pronalazi najbolju moguću akciju za postizanje cilja na osnovu informacija koje ima. Akcije su ili **nagrađene** ili **kažnjene**. Šargarepa privezana za štap je odličan primer nagrade i kazne. Zamislite magarca ispred vaših kolica. Stavite šargarepu ispred očiju magarca, pa on kreće ka njoj. Čim se magarac zaustavi, vozač može da primeni kaznu pomoću štapa. To nije ljudski način kretanja, ali nagrade i kazna kontrolišu žive organizme do neke mere. Isto se dešava ljudima u školi, na poslu i u svakodnevnom životu. Umesto šargarepe i štapa, mi imamo prihode i zakonske kazne za oblikovanje našeg ponašanja.

U većini igara i gamificiranih aplikacija dobra sekvenca akcija dovodi do nagrade. Kada se igrač oseća nagrađenim, otpušta se hormon pod nazivom dopamin, koji se naziva i hemikalija nagrade. Kada čovek postigne cilj ili završi zadatak, oslobađa se dopamin. Ovaj hormon čini da se osećamo srećnim. Ljudi imaju tendenciju da se ponašaju na način koji maksimizira njihovu sreću. Ova sekvenca akcija se naziva **kompulsiona petlja**. Sa druge strane, inteligentni agenti su zainteresovani samo za svoje ciljeve, odnosno da maksimizuju nagradu i minimiziraju kaznu.

Kada modelujemo igrice, treba da odredimo njihov **prostor stanja**. Akcija dovodi do **prelaza stanja**. Kada istražimo posledice svih mogućih akcija, dobićemo **stablo odlučivanja**. Ovo stablo se produbljuje kada započnemo istraživanje mogućih budućih akcija svih igrača do kraja igre.

Moć AI-e je u izvršenju miliona mogućih koraka svake sekunde. Prema tome, AI igrice se često svodi na **vežbu pretrage**. Kada istražujemo sve moguće sekvence pokreta u igri, dobićemo **stablo stanja igre**.

Razmotrimo primer AI šaha. U čemu je problem sa procenjivanjem svih mogućih poteza izgradnjom stabla stanja, koje se sastoji od svih mogućih sekvenci poteza?

Šah je EXPTIME igra kada je u pitanju složenost. Broj mogućih poteza eksplodira kombinatorno.

Bela figura počinje sa 20 mogućih poteza: osam pešaka se može pomeriti za jedan ili dva polja, a dva skakača mogu da se pomere ili gore-gore-levo, ili gore-gore-desno. Zatim, crna figura može da izvrši bilo koji od ovih 20 poteza. Dakle, već postoji 20*20 = 400 mogućih kombinacija nakon samo jednog poteza po igraču.

Nakon drugog poteza, dobijamo 8.902 moguća sazvežđa na tabli i ovaj broj samo raste. Nakon povučenih sedam poteza, treba da pretražujete kroz 10.921.506 mogućih sazvežđa.

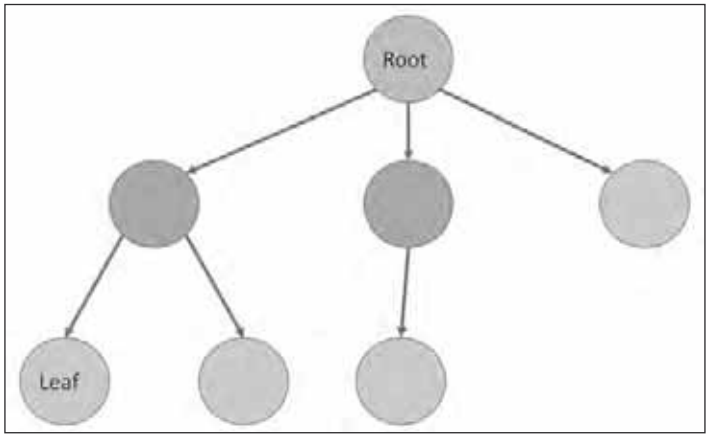
Prosečna dužina šahovske partije je oko 40 poteza. Neke izuzetne igre traju i više od 200 poteza.

Računar jednostavno nema vremena da istraži ceo prostor stanja. Prema tome, aktivnost pretrage treba da bude vođena odgovarajućim nagradama, kaznama i pojednostavljenjem pravila.

Breadth First Search i Depth First Search

Kreiranje AI-e za igru je često vežba pretraživanja. Prema tome, treba da su nam poznate dve primarne tehnike pretrage: Breadth First Search (BFS) i Depth First Search (DFS).

Ove tehnike pretrage su primenjene na **usmereno korensko stablo**. Stablo je struktura podataka koja ima čvorove - ivice povezuju ova čvorove, tako da su bilo koja dva čvora stabla povezana tačno jednom putanjom.



Slika 1.3 Usmereno korensko stablo

Kada je stablo ukorenjeno, postoji specijalni čvor u stablu koji se naziva korenski, u kojem započinjemo put kroz dijagram. Usmereno stablo je stablo gde se ivice mogu kretati samo u jednom smeru. Čvorovi mogu da budu **interni** ili **čvorovi listovi**. Interni čvorovi imaju najmanje jednu ivicu pomoću koje možemo da napustimo čvor. Čvor list nema ivice koja pokazuju van čvora.

U AI pretrazi koren stabla je početno stanje. Mi se krećemo od ovog stanja generisanjem čvorova naslednika stabla pretrage. Tehnike pretrage se razlikuju, u zavisnosti od toga u kojem redosledu posećujemo ove čvorove naslednike.

Pretpostavimo da imamo stablo definisano po njegovom korenu i funkciju koja generiše sve čvorove naslednike iz korena. U ovom primeru svaki čvor ima vrednost i dubinu. Počinjemo od 1 i možemo povećavati vrednost za 1 ili 2. Naš cilj je da dostignemo vrednost 5.

```
root = {'value': 1, 'depth': 1}
def succ(node):
    if node['value'] == 5:
        return []
    elif node['value'] == 4:
        return [{'value': 5,'depth': node['depth']+1}]
    else:
        return [
            {'value': node['value']+1, 'depth':node['depth']+1},
            {'value': node['value']+2, 'depth':node['depth']+1}
        ]
```

Prvo ćemo izvršiti DFS pretragu u ovom primeru:

```
def bfs_tree(node):
    nodes_to_visit = [node]
    visited_nodes = []
    while len(nodes_to_visit) > 0:
        current_node = nodes_to_visit.pop(0)
        visited_bodes.append(current_node)
        nodes_to_visit.extend(succ(current_node))
    return visited_nodes


bfs_tree(root)
```

Rezultat je sledeći:

```
[{'depth': 1, 'value': 1},
 {'depth':  2, 'value':  2},
 {'depth':  2, 'value':  3},
 {'depth':  3, 'value':  3},
 {'depth':  3, 'value':  4},
 {'depth':  3, 'value':  4},
 {'depth':  3, 'value':  5},
 {'depth':  4, 'value':  4},
 {'depth':  4, 'value':  5},
 {'depth':  4, 'value':  5},
 {'depth':  4, 'value':  5},
 {'depth':  5, 'value':  5}]
```

Vidite da BFS pronalazi najkraću putanju prvo do čvora lista, jer nabraja sve čvorove u redosledu povećanja dubine.

Ako bi trebalo da prelistamo graf, umesto usmerenog korenskog stabla, BFS bi izgledala drugačije: kad god posetimo čvor, treba da proverimo da li je čvor već ranije posećen. Ako je čvor bio ranije posećen, jednostavno ga ignorišemo.

U ovom poglavlju upotrebićemo samo **Breadth First Traversal** u stablima. **Depth First Search** je iznenađujuće slična pretrazi **Breadth First Search**. Razlika između **Depth First Traversals** i BFS pretrage je sekvenca u kojoj pristupamo čvorovima. BFS posećuje sve potomke čvora pre nego što poseti druge čvorove, a DFS se kreće u dubinu stabla:

```
def dfs_tree(node):
    nodes_to_visit = [node]
    visited_nodes = []
    while len(nodes_to_visit) > 0:
        current_node = nodes_to_visit.pop()
        visited_nodes.append(current_node)
        nodes_to_visit.extend(succ(current_node))
    return visited_nodes

dfs_tree(root)
```

Rezultat je sledeći:

```
[{'depth': 1, 'value': 1},
 {'depth':  2, 'value':  3},
 {'depth':  3, 'value':  5},
 {'depth':  3, 'value':  4},
 {'depth':  4, 'value':  5},
 {'depth':  2, 'value':  2},
 {'depth':  3, 'value':  4},
 {'depth':  4, 'value':  5},
 {'depth':  3, 'value':  3},
 {'depth':  4, 'value':  5},
 {'depth':  4, 'value':  4},
 {'depth':  5, 'value':  5}]
```

Kao što možete da vidite, DFS algoritam brzo prolazi u dubinu. Neće obavezno prvo pronaći najkraću putanju, ali će sigurno pronaći čvor list pre istraživanja druge putanje.

U AI igrici BFS algoritam je često bolji za procenu stanja igre, jer se DFS može izgubiti. Zamislite pokretanje šahovske igre u kojoj se DFS algoritam može lako izgubiti u pretrazi.

Istraživanje prostora stanja igre

Istražićemo prostor stanja jednostavne igre Tic-Tac-Toe.

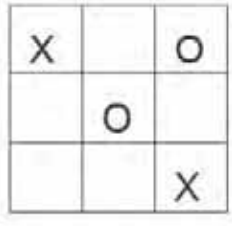
U igri Tic-Tac-Toe postoji tabla za igru 3x3. Dva igrača igraju ovu igru. Jedan koristi znak X, a drugi znak O. Igrač X započinje igru i igrači izvršavaju korake jedan posle drugog. Cilj igre je da igrač dobije tri znaka horizontalno, vertikalno ili dijagonalno.

Označimo sada ćelije Tic-Tac-Toe table kao na sledećoj slici.



Slika 1.4 Tic-Tac-Toe tabla

U sledećem primeru igrač X započinje u poziciji 1. Igrač O je odmakao na poziciju 5, a igrač X se pomerio na poziciju 9, a zatim se igrač O pomerio na poziciju 3.



Slika 1.5 Tic-Tac-Toe tabla sa oznakama X i O

Drugi igrač je napravio grešku, jer je sada igrač X prisiljen da postavi znak u polje 7, što dovodi do dva buduća scenarija za pobedu u igri. Nije važno da li se igrač O brani pomeranjem na polje 4 ili 8 - igrač X pobeđuje u igri selektovanjem druge slobodne ćelije.



Možete da isprobate igricu na adresi http://www.half-real.net/tictactoe/.

Zbog jednostavnosti primera, istražili smo samo prostor stanja koje pripada slučajevima kada AI igrač započinje igru. Započećemo igru sa AI igračem nasumično, postavljajući znak u praznu ćeliju. Nakon igranja sa ovim AI igračem, kreiraćemo kompletno stablo odlučivanja. Kada generišemo sva moguća stanja igre, suočićemo se sa eksplozijom njihove kombinatorike. Pošto je naš cilj da ove kompleksnosti učinimo jednostavnim, upotrebićemo nekoliko različitih tehnika da bismo AI igrača učinili pametnijim i da bismo smanjili veličinu stabla odlučivanja. Do kraja ovog eksperimenta imaćemo stablo odlučivanja koje je manje od 200 različitih završetaka igre i, kao bonus, AI igrač nikada neće izgubiti u igri.

Da bismo izvršili nasumičan potez, treba da znamo kako da izaberemo nasumičan element iz liste upotrebom Pythona. Upotrebićemo funkciju `choice` biblioteke random:

```
from random import choice
choice([2, 4, 6, 8])
```

Rezultat je 6.

Rezultat funkcije choice je nasumičan element liste.



Upotrebićemo faktorijel notaciju u sledećoj vežbi. Faktorijel je označen uzvičnikom „!“, prema definiciji, 0! = 1, a n! = n*(n-1)!. U našem primeru, 9! = 9* 8! = 9*8*7! = … = 9*8*7*6*5*4*3*2*1.

Vežba 2: Procena broja mogućih stanja u Tic-Tac-Toe igri

Izvršićemo grubu procenu broja mogućih stanja na svakom nivou prostora stanja Tic-Tac-Toe igre:

- U proceni se nećemo zaustaviti sve dok sve ćelije na tabli ne budu popunjene. Igrač može da pobedi pre kraja igre, ali, zbog jednostavnosti, mi ćemo nastaviti igru.
- Prvi igrač će izabrati jedno od devet polja. Drugi igrač će izabrati jedno od preostalih osam polja. Prvi igrač će, zatim, izabrati jedno od sedam preostalih polja. I ovo se nastavlja dok jedan od igrača ne pobedi u igri ili dok prvi igrač ne bude prisiljen da izvrši deveti i poslednji korak.
- Broj mogućih sekvenci odlučivanja je, prema tome, 9! = 362880. Neke od ovih sekvenci su nevalidne, jer igrač može da pobedi u igri u manje od devet poteza. Potrebno je najmanje pet poteza da se pobedi u igri, jer prvi igrač treba da se pomeri tri puta.

- Da bismo izračunali tačnu veličinu prostora stanja, treba da izračunamo broj igara koje su završene pobedom u pet, šest, sedam i osam koraka. Ovo izračunavanje je jednostavno, ali, zbog njegove opširne prirode, nećemo ga primeniti u ovoj knjizi. Prema tome, mi ćemo se zadovoljiti veličinom prostora stanja.



Nakon generisanja svih mogućih Tic-Tac-Toe igara, istraživači su izbrojali 255.168 mogućih igara: u 131.184 je pobedio prvi igrač, u 77.904 je pobedio drugi igrač, a 46.080 igara je završeno nerešeno. Pogledajte http://www.half-real.net/tictactoe/allgamesoftictactoe.zip da biste preuzeli sve moguće TicTac-Toe igre.

Čak i jednostavna igra, kao što je Tic-Tac-Toe, ima veliki broj stanja. Samo zamislite koliko bi bilo teško započeti istraživanje svih mogućih igara šaha. Prema tome, možemo da zaključimo da je gruba pretraga retko idealna.

Vežba 3: Kreiranje AI-e nasumično

U ovom odeljku ćemo kreirati radni okvir za Tic-Tac-Toe igru za eksperimentisanje. Modelovaćemo igru na pretpostavci da AI igrač uvek započinje igru. Kreiraćemo funkciju koja štampa internu reprezentaciju i omogućava protivniku da nasumično unese korak. Odredićemo da li je igrač pobedio. Da bismo bili sigurni da se sve dešava korektno, treba prvo da zvršimo prethodne vežbe:

1. Importovaćemo funkciju choice iz biblioteke random:

```
from random import choice
```

2. Modelovaćemo devet ćelija u jednostavnom znakovnom nizu. Python znakovni niz dužine devet karaktera čuva ove ćelije u sledećem redosledu: „123456789“. Sada ćemo odrediti indeksne trojke koje moraju da sadrže odgovarajuće znakove da bi igrač pobedio u igri:

```
combo_indices = [
    [0,   1,  2],
    [3,   4,  5],
    [6,   7,  8],
    [0,   3,  6],
    [1,   4,  7],
    [2,   5,  8],
    [0,   4,  8],
    [2,   4,  6]
]
```

3. Definisaćemo sign konstante za prazne ćelije, AI i protivničkog igrača:

```
EMPTY_SIGN = '.'
AI_SIGN = 'X'
OPPONENT_SIGN = 'O'
```

4. Kreiraćemo funkciju koja štampa tablu. Dodaćemo prazan red pre i posle table da bismo lako mogli da čitamo stanje igre:

```
def print_board(board):
    print(" ")
    print(' '.join(board[:3]))
    print(' '.join(board[3:6]))
    print(' '.join(board[6:]))
    print(" ")
```

5. Opisaćemo potez ljudskog igrača. Uneti argumenti su table, brojevi reda od 1 do 3 i brojevi kolone od 1 do 3. Vraćena vrednost ove funkcije je tabla koja sadrži novi potez:

```
def opponent_move(board, row, column):
    index = 3 * (row - 1) + (column - 1)
    if board[index] == EMPTY_SIGN:
        return board[:index] + OPPONENT_SIGN + board[index+1:]
    return board
```

6. Vreme je da definišemo nasumični potez AI igrača. Generisaćemo sve moguće poteze pomoću funkcije `all_moves_from_board`, a zatim ćemo selektovati nasumičan potez iz liste mogućih poteza:

```
def all_moves_from_board_list(board, sign):
    move_list = []
    for i, v in enumerate(board):
        if v == EMPTY_SIGN:
            move_list.append(board[:i] + sign + board[i+1:])
    return move_list


def ai_move(board):
    return choice(all_moves_from_board(board, AI_SIGN))
```

7. Nakon definisanja poteza, treba da odredimo da li je igrač pobedio u igri:

```
def game_won_by(board):
    for index in combo_indices:
        if board[index[0]] == board[index[1]] == board[index[2]] !=
        EMPTY_
SIGN:
            return board[index[0]]
    return EMPTY_SIGN
```

8. Na kraju ćemo kreirati petlju igre da bismo mogli da testiramo interakciju između računarskog igrača i ljudskog igrača. Izvršićemo detaljnu pretragu u sledećim primerima:

```
def game_loop():
    board = EMPTY_SIGN * 9
    empty_cell_count = 9
    is_game_ended = False
    while empty_cell_count > 0 and not is_game_ended:
        if empty_cell_count % 2 == 1:
            board = ai_move(board)
        else:
            row = int(input('Enter row: '))
            col = int(input('Enter column: '))
            board = opponent_move(board,            row, col)
        print_board(board)
        is_game_ended = game_won_by(board) != EMPTY_SIGN
    empty_cell_count = sum(
            1 for cell in board if cell == EMPTY_SIGN
        )
    print('Game has been ended.')
```

9. Upotrebićemo funkciju `game_loop` za pokretanje igre:

```
game_loop()
```

Kao što možete da vidite, čak i igrač koji igra nasumično može da pobedi s vremena na vreme ako protivnik napravi grešku.

Aktivnost 1: Generisanje svih mogućih sekvenci koraka u Tic-Tac-Toe igri

Ova aktivnost će istražiti eksploziju kombinatorike koja je moguća kada dva igrača igraju nasumično. Upotrebićemo program građen na prethodnim rezultatima, koji generiše sve moguće sekvence poteza između računara i igrača. Pretpostavimo da ljudski igrač može da izvrši bilo koji mogući potez. U ovom primeru, pošto računar igra nasumično, istražićemo pobede, gubitke i nerešene rezultate:

1. Kreiraćemo funkciju koja mapira `all_moves_from_board` funkciju na svakom elementu liste prostora table/kvadrata. U tom slučaju imaćemo sve čvorove stabla odlučivanja.
2. Stablo odlučivanja započinje sa `[ EMPTY_SIGN * 9 ]` i širi se nakon svakog poteza. Kreiraćemo funkciju `filter_wins`, koja izvlači završene igre iz liste poteza i dodaje ih u niz koji sadrži stanja table koje je osvojio AI igrač i protivnički igrač.
3. Zatim ćemo upotrebiti funkciju `count_possibilities`, koja štampa broj listova stabla odlučivanja koja su završena nerešenim rezultatom, pobedom prvog igrača ili pobedom drugog igrača.
4. Imamo do devet koraka u svakom stanju. U 0., 2., 4., 6. i 8. iteraciji AI igrač izvršava korak. U svim drugim iteracijama, pokreće se protivnik. Kreiraćemo sve moguće poteze u svim koracima i izvući završene igre iz liste poteza.
5. Zatim ćemo izvršiti broj mogućnosti da bismo iskusili eksploziju kombinatorike.

Kao što možete da vidite, stablo stanja table se sastoji od 266.073 lista. U stvari, funkcija `count_possibilities` implementira BFS algoritam za pregled svih mogućih stanja igre. Brojimo ova stanja više puta, jer postavljanje X-a u gornji desni ugao u koraku 1 i njegovo postavljanje u gornji levi ugao u koraku 3 dovode do sličnih mogućih stanja, pošto počinjemo sa gornjim levim uglom, a zatim postavljamo X u gornji desni ugao. Ako implementiramo stanja detekcije duplikata, treba da proverimo manje čvorova. Međutim, u ovoj fazi, zbog ograničene dubine igre, izostavićemo taj korak.

Stablo odlučivanja je veoma slično strukturi podataka koju ispituje funkcija count_possibilities. U tom stablu istražujemo korisnost svakog poteza ispitivanjem svim mogućih budućih koraka do određene granice. U našem primeru možemo da izračunamo korisnost prvih poteza posmatranjem broja pobeda i gubitaka nakon nekoliko početnih koraka.



Koren stabla je početno stanje. Interno stanje stabla je stanje u kojem igra nije završena i mogući su potezi. List stabla sadrži stanje u kojem se igra završava.

Rešenje za ovu aktivnost pronaći ćete na 258. stranici.

REZIME

U ovom poglavlju ste naučili šta je veštačka inteligencija i koje su njene višestruke discipline.

Videli ste kako AI može da se upotrebi za poboljšanje ili zamenu moći ljudskog mozga, za slušanje, govor, razumevanje jezika, skladištenje i preuzimanje informacija, razmišljanje, vid i pokret. Nakon toga ste učili o inteligentnim agentima koji deluju u svojim okruženjima, rešavajući probleme na inteligentan način za postizanje prethodno određenog cilja. Oni uče na nadgledan i nenadgledan način. Možemo da ih upotrebimo za klasifikaciju ili predviđanje budućnosti.

Zatim smo predstavili Python i njegovu ulogu u oblasti veštačke inteligencije. Opisali smo nekoliko važnih Python biblioteka za razvoj inteligentnih agenata i pripremu podataka za te agente. Zaključili smo ovo poglavlje primerom u kojem smo upotrebili NumPy biblioteku za izvršavanje nekih operacija matrica u Pythonu. Takođe ste saznali kako se kreira prostor pretrage za Tic Tac Toe igricu. U sledećem poglavlju ćete naučiti kako inteligencija može da bude dodeljena pomoću prostora pretrage.