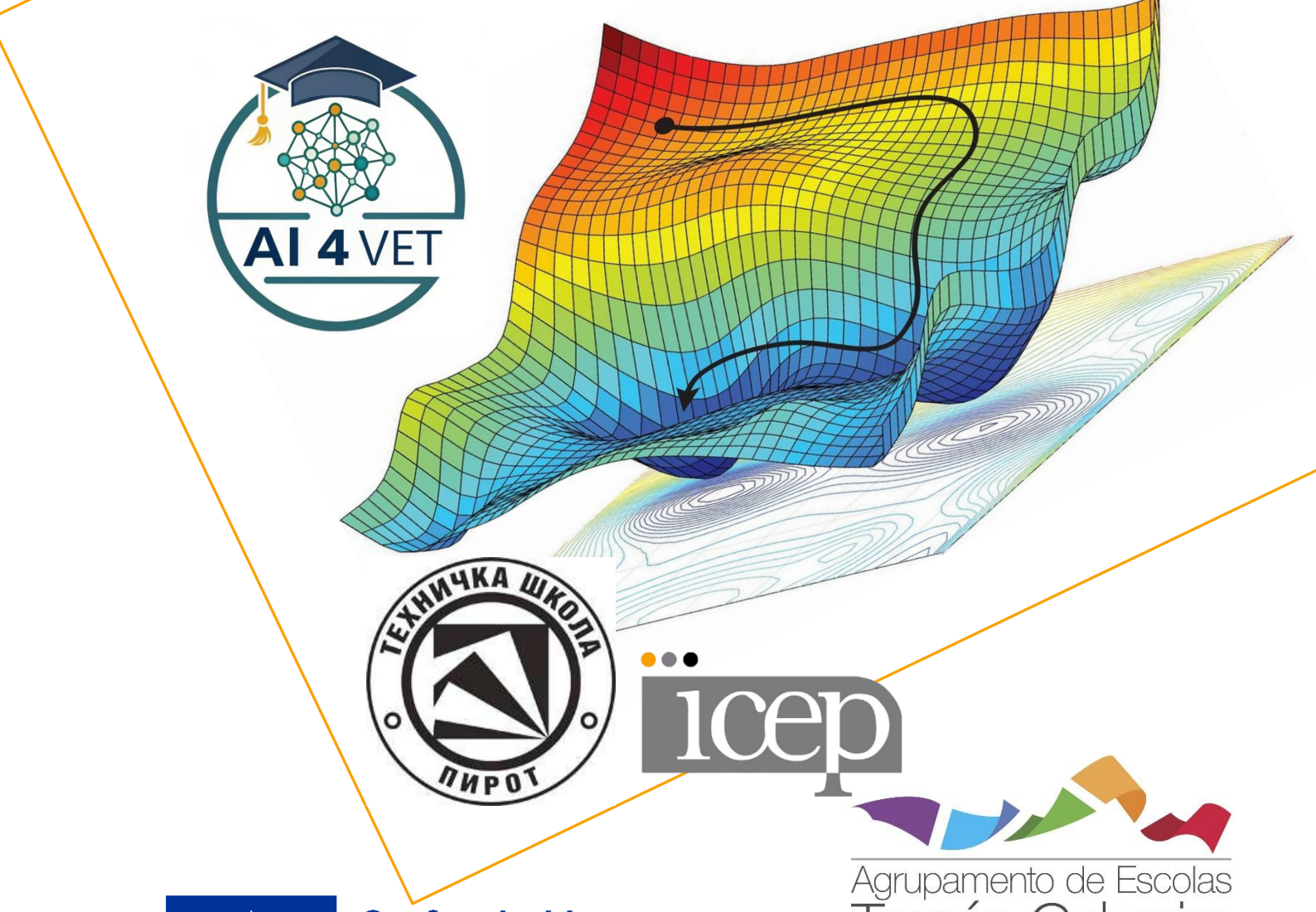


Co-funded by the European Union

ZÁKLADY UMELEJ INTELIGENCIE A STROJOVÉHO UČENIA

Technická škola Pirot

KA220-VET – Partnerstvá v oblasti odborného vzdelávania a prípravy

Nástroje umelej inteligencie pre školy odborného vzdelávania a prípravy

Dátum dokumentu: jún 2025

Tento materiál bol zostavený a pripravený na účely projektu Erasmus:
Technická škola Pirot (autor: Boban Blagojević, spoluautori: Bojan Ćirić, Aleksandar Madić)
ICEP (autor: Ladislav Mariš, spoluautor: Adelaida Fanfarova)
Agrupamento de Escolas Tomas Cabreira (autor: Sandra Nobre, spoluautori: Rui Dias, Gilherme Mota, Carla Lima, Maria Torrinha)

Web: https://book.tsp.edu.rs

OBSAH

1. UMELÁ INTELIGENCIA

Vitajte v téme **Umelá inteligencia (AI)**! V tejto časti budeme skúmať fascinujúci svet AI, ktorý zahŕňa vytváranie systémov, ktoré dokážu vykonávať úlohy, ktoré zvyčajne vyžadujú ľudskú inteligenciu. Dozviete sa o histórii a vývoji AI, základných pojmoch a etických otázkach týkajúcich sa jej používania. Budeme tiež diskutovať o rôznych typoch AI, vrátane úzkej, všeobecnej a superinteligentnej, a o tom, ako AI mení rôzne aspekty nášho každodenného života.

1.1 Pojem umelá inteligencia

Umelá inteligencia (*AI*) je disciplína, ktorá sa zaoberá vývojom programov, ktoré svojimi schopnosťami vytvárajú dojem inteligentného správania. Tieto programy sa vyznačujú schopnosťou rozpoznať zložité vzťahy a na ich základe vyvodiť závery. Uvidíme, že tieto činnosti majú základ v disciplínach, ako je matematika, informatika, počítačové vedy a robotika. Vzhľadom na svoje širšie uplatnenie sa táto oblasť týka aj všetkých ostatných disciplín, ktoré sa zaoberajú pochopením inteligencie, ako sú neurovedy, filozofia a umenie, a jej vplyvom na spoločnosť, ako sú sociológia, právo a etika.

Je dôležité zdôrazniť, že nie každý program, ktorý má nejakú formu *inteligentného* správania, musí byť založený na umelej inteligencii. Pozrime sa na program, ktorý otvára dvere pri vstupe do budovy. Táto funkcia môže byť aktivovaná pomocou senzorov priblíženia, ktoré detekujú prítomnosť alebo vyžadujú zadanie kódu, ktorý by mal zodpovedať očakávanému kódu. Oba tieto scenáre by sme mohli pokryť klasickými programovacími technikami porovnaním vzdialenosti meranej senzorom s nejakou hranicou vzdialenosti, t. j. zadám kód so správnym kódom. Na druhej strane, ak je na vstup do budovy potrebné, aby sprievodná kamera rozpoznala našu tvár, budeme, ako čoskoro uvidíme, potrebovať pomoc umelej inteligencie.

História umelej inteligencie

História umelej inteligencie (AI) je poznačená významnými míľnikmi, ktoré odrážajú vývoj technológie a ľudské chápanie inteligencie. Od svojich počiatkov až po súčasné aplikácie prešla AI rôznymi premenami ovplyvnenými prelomovými objavmi vo výskume, spoločenskými potrebami a technologickým pokrokom.

Počiatky (50. roky 20. storočia)

Cesta AI začala v 50. rokoch, desaťročí, ktoré položilo základy pre túto oblasť. V roku 1950 Alan Turing publikoval svoju prelomovú prácu „Computing Machinery and Intelligence" (Počítačové stroje a inteligencia), v ktorej predstavil Turingov test, ktorého cieľom bolo posúdiť schopnosť stroja preukázať inteligentné správanie, ktoré nie je možné odlíšiť od správania človeka. Nasledujúci rok Marvin Minsky a Dean Edmonds vytvorili SNARC, prvú umelú neurónovú sieť (ANN), ktorá simulovala sieť neurónov pomocou vákuových trubíc. V roku 1956 na workshope organizovanom Johnom McCarthym, Marvinom Minskym, Nathanielom Rochesterom a Claudom Shannonom bol vytvorený termín „umelá inteligencia". Táto udalosť je široko považovaná za moment založenia umelej inteligencie ako samostatnej oblasti.

Vzostup strojového učenia (60. – 70. roky)

V 60. rokoch došlo k vývoju prvých programov umelej inteligencie, ako napríklad Eliza, chatbot schopný viesť jednoduché konverzácie, a Shakey, prvý mobilný robot s funkciami umelej inteligencie. Toto obdobie však prinieslo aj výzvy. Obmedzenia raných neurónových sietí boli zdôraznené v roku 1969, keď Marvin Minsky a Seymour Papert publikovali *Perceptrons*, čo viedlo k úpadku výskumu neurónových sietí v prospech symbolických prístupov umelej inteligencie. Sedemdesiate roky znamenali „zimu umelej inteligencie", charakterizovanú znížením financovania a záujmu v dôsledku nesplnených očakávaní. Kľúčová správa Jamesa Lighthilla z roku 1973 kritizovala výskum umelej inteligencie v Spojenom kráľovstve, čo viedlo k významnému zníženiu vládnej podpory.

Oživenie a expanzia (80. – 90. roky)

AI zažila renesanciu v 80. rokoch s komercializáciou strojov Lisp a obnoveným záujmom o expertné systémy. V tomto období došlo k pokroku v oblasti reprezentácie znalostí a techník uvažovania, čo umožnilo sofistikovanejšie aplikácie AI. Zavedenie viacvrstvových algoritmov trénovania ANN ďalej oživilo výskum neurónových sietí. V 90. rokoch sa AI začala integrovať do praktických aplikácií, ako je rozpoznávanie reči a spracovanie videa. Deep Blue od IBM sa dostal na titulné stránky novín, keď v roku 1997 porazil svetového šachového majstra Garryho Kasparova, čím demonštroval potenciál umelej inteligencie v strategickom myslení.

Moderná éra (2000 – súčasnosť)

V 21. storočí došlo k explozívnemu nárastu schopností umelej inteligencie vďaka pokroku v oblasti strojového učenia, najmä hlbokého učenia. Technológie ako IBM Watson, osobní asistenti ako Siri a Alexa, systémy rozpoznávania tváre a generatívne modely ako GPT sa stali neoddeliteľnou súčasťou každodenného života. Vzostup veľkých dát a zvýšený výpočtový výkon umožnili týmto systémom učiť sa z obrovského množstva informácií, čo viedlo k významnému zlepšeniu výkonu v rôznych oblastiach. Dnes diskusie o umelej inteligencii zahŕňajú aj etické hľadiská a vplyv na spoločnosť. S rastúcou popularitou systémov umelej inteligencie sa čoraz viac skúmajú otázky súvisiace so súkromím, predsudkami a zodpovednosťou.

Kľúčové míľniky

- **1950**: Alan Turing navrhuje Turingov test.
- **1956**: Konferencia v Dartmouthe ustanovuje umelú inteligenciu ako vedný odbor.
- **1966**: Vytvorenie ELIZA, jedného z prvých programov NLP.
- **1997**: Deep Blue od IBM porazil Garryho Kasparova.
- **2012**: Prerazenie v oblasti hlbokého učenia, keď AlexNet vyhráva súťaž ImageNet.
- **2020**: Umelá inteligencia sa stáva neoddeliteľnou súčasťou rôznych odvetví, čo vyvoláva etické a regulačné obavy.

1.2 Turingov test

Turingov test, navrhnutý Alanom Turingom (1950), bol navrhnutý ako myšlienkový experiment, ktorý by obišiel filozofickú nejasnosť otázky „Môže stroj myslieť?" Počítač prejde testom, ak ľudský vyšetrovateľ po položení niekoľkých písomných otázok nedokáže rozlíšiť, či písomné odpovede pochádzajú od človeka alebo

od počítača. Kapitola 28 rozoberá podrobnosti testu a otázku, či by počítač bol skutočne inteligentný, ak by testom prešiel. Zatiaľ konštatujeme, že naprogramovanie počítača tak, aby prešiel prísne uplatňovaným testom, si vyžaduje veľa práce. Počítač by potreboval nasledujúce schopnosti:

- spracovanie prirodzeného jazyka, aby mohol úspešne komunikovať v ľudskom jazyku;
- reprezentácia vedomostí na ukladanie toho, čo vie alebo počuje;
- automatizované uvažovanie, aby mohol odpovedať na otázky a vyvodzovať nové závery;
- strojové učenie, aby sa prispôsoboval novým okolnostiam a aby dokázal rozpoznávať a extrapolovať vzory.

Turing považoval fyzickú simuláciu človeka za zbytočnú na preukázanie inteligencie. Iní výskumníci však navrhli úplný Turingov test, ktorý vyžaduje interakciu s objektmi a ľuďmi v reálnom svete. Na úspešné absolvovanie úplného Turingovho testu bude robot potrebovať

- počítačové videnie a rozpoznávanie reči na vnímanie sveta;
- robotiku na manipuláciu s objektmi a pohyb.

Týchto šesť disciplín tvorí väčšinu umelej inteligencie. Výskumníci umelej inteligencie však venovali málo úsilia úspešnému absolvovaniu Turingovho testu, pretože verili, že je dôležitejšie študovať základné princípy inteligencie. Hľadanie „umelého letu" bolo úspešné, keď inžinieri a vynálezcovia prestali napodobňovať vtáky a začali používať veterné tunely a študovať aerodynamiku. Texty z oblasti leteckého inžinierstva nedefinujú cieľ svojej oblasti ako výrobu „strojov, ktoré lietajú tak presne ako holuby, že dokážu oklamať aj ostatné holuby".

Môže stroj myslieť?

Zaujímavé je, že vývoj umelej inteligencie bol plný prekážok.

Otázka znie: „Môže stroj myslieť?" V roku 1950 anglický matematik Alan Turing signalizoval začiatok vývoja oblasti, ktorá je dnes známa ako umelá inteligencia. Niekoľko rokov po tomto skvelom nápade, v roku 1956, sa významní vedci John McCarthy, Marvin Minsky, Nathaniel Rochester a Claude Shannon zišli na konferencii v Dartmouthe, ktorá trvala celý mesiac, s cieľom definovať výskumné ciele a protokoly v tejto oblasti. Vtedy dostala tento odbor svoj oficiálny názov a po prvýkrát bol skutočne prezentovaný ako umelá inteligencia.

V čase vzniku umelej inteligencie boli počítače veľmi odlišné od tých dnešných. Mali oveľa menšiu kapacitu a rýchlosť a boli oveľa drahšie. Výskum v tejto oblasti si preto vyžadoval odlišné konštrukčné riešenia a závisel od oficiálnej podpory a stabilných zdrojov financovania.

V prvej vlne vývoja umelej inteligencie, ktorá trvala až do začiatku sedemdesiatych rokov 20. storočia, sa objavilo mnoho zaujímavých programov. Prvým z nich bol *program Logic Theorist* napísaný v roku 1956, ktorý skúmal schopnosti matematickej logiky a odvodzovania záverov. Tento program dokázal dokázať 38 z prvých 52 viet uvedených v slávnej knihe *Principles of Mathematics.* Existuje aj program *ELIZA*, ktorý dokázal simulovať konverzáciu s používateľmi podľa jednoduchých pravidiel a konštrukcií v angličtine. Motiváciou pre výskum v oblasti komunikácie bol návrh Alana Turinga vyhlásiť inteligentné stroje, ktoré dokážu komunikovať tak, že nevytvárajú dojem, že človek hovorí so strojom. Tento test je dnes známy ako Turingov test.

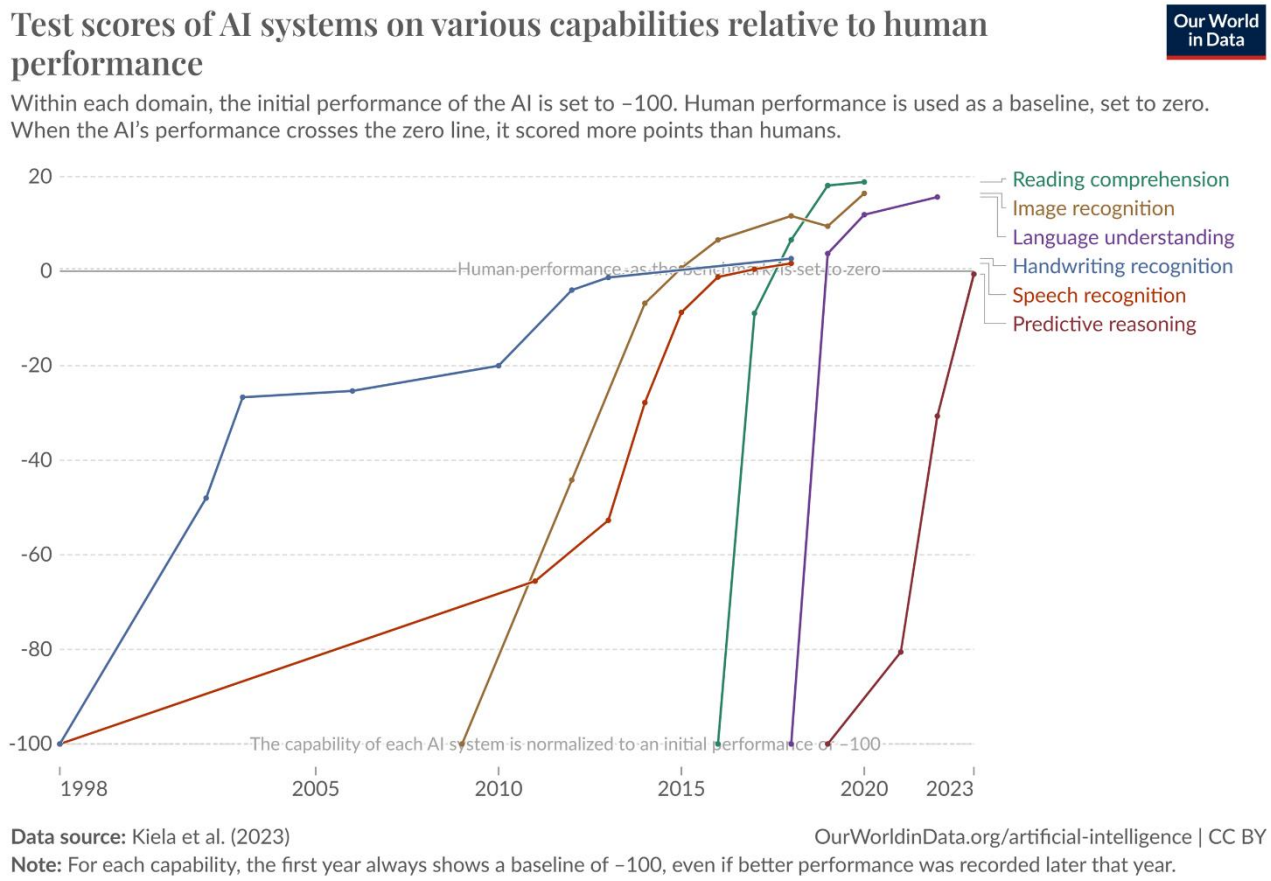
V období až do začiatku 70. rokov 20. storočia vzniklo mnoho nápadov, ktoré sa neskôr použili na prelomové objavy v modernej umelej inteligencii. Jedným z nich je idea perceptronu, základu dnešných neurónových sietí, ktorú v roku 1957 predstavil Frank Rosenblatt. V optimistických vyhláseniach tohto výskumníka mal perceptron schopnosť učiť sa, rozhodovať sa a prekladať jazyky, ale potvrdenie tejto skutočnosti trvalo dlho.

V dôsledku nedostatku finančných prostriedkov nasledovala po prvej vlne vývoja umelej inteligencie takzvaná prvá zima umelej inteligencie. Tento stav je čiastočne spôsobený ambicióznymi projektmi, ktorých výsledky chýbali kvôli obmedzeným kapacitám počítačov a nedostatku dostupných údajov.

Jedna zo zaujímavých a podnetných udalostí v histórii umelej inteligencie sa odohrala v roku 1997, keď sa *systému* DeepBlue *spoločnosti IBM* podarilo poraziť svetového šachového veľmajstra Garryho Kasparova. *Systém DeepBlue* je predstaviteľom triedy takzvaných expertných systémov, systémov, ktoré na základe databázy obsahujúcej množstvo pravidiel typu „ak-potom" a uplatňovaním logických pravidiel dokážu napodobniť uvažovanie odborníkov v danej oblasti a poskytnúť správny výsledok. *Systém AlphaGo* spoločnosti Google DeepMind *mal podobný vplyv na vývoj umelej inteligencie* takmer 20 rokov neskôr, v roku 2016, keď porazil svetového šampióna Lee Sedola v hre Go.

V roku 2011 demonštroval schopnosť strojov nájsť odpoveď na otázku položenú v angličtine systém *Watson* od IBM v kvíze *Jeopardy*! *Watson* porazil svojich dvoch súperov, víťazov predchádzajúcich ročníkov kvízu, tým, že najrýchlejšie poskytol správne odpovede na položené otázky. Zdroje, ktoré o tejto udalosti písali, uvádzali, že *Watson* bol schopný spracovať 500 GB obsahu za sekundu, t. j. približne milión kníh.

Na druhej strane, schopnosť strojov rozpoznávať a rozlišovať objekty na obrázkoch demonštroval v roku 2012 tím *Google X*, ktorý vytvoril program schopný rozpoznávať mačky na obrázkoch. Tento program za 3 dni prezrel viac ako 10 miliónov obrázkov a naučil sa rozpoznávať mačky. K dnešnému dňu sa schopnosti rozpoznávacích systémov výrazne zlepšili a v mnohých aplikáciách poskytujú presnejšie odpovede ako väčšina ľudí. Na obrázku nižšie môžete vidieť vývojové trendy systémov na rozpoznávanie ručne písaného textu, rozpoznávanie reči, rozpoznávanie obrázkov a dva najnovšie výsledky s pozoruhodným nárastom schopností súvisia s úlohami porozumenia jazyka.



Obrázok je prevzatý z https://ourworldindata.org/brief-history-of-ai

Tieto úspechy boli tiež predzvesťou oveľa sľubnejšieho pokračovania vývoja umelej inteligencie, a to jednak vďaka dostupnosti internetu, webu a väčšiemu množstvu údajov, ako aj vďaka počítačom, ktorých výpočtový výkon je neporovnateľne väčší ako výkon počítačov v 50. rokoch 20. storočia. To viedlo aj k zmene paradigmy, ktorá dominuje v tejto oblasti, a k prechodu od systémov založených na logike k systémom založeným na štatistike.

Príbeh vývoja umelej inteligencie súvisí aj s robotmi. Nielen v sci-fi románoch a filmoch, ale aj pokiaľ ide o vzhľad skutočných robotov. V roku 1950 americký vedec Claude Shannon navrhol myš, ktorá vedela nájsť cestu a dostať sa z bludiska. V duchu gréckej mytológie bola myš pomenovaná Theseus. V roku 1966 začal tím vedcov zo Stanfordského výskumného inštitútu pracovať na vývoji robota Shakey, prvého robota schopného pohybovať sa a vyvodzovať závery o prostredí. Prvé autonómne vozidlo ALVINN (skratka pre Autonomous *Land Vehicle In a Neural Network),* na ktorom pracoval tím výskumníkov z Carnegie Mellon University, bolo skonštruované v roku 1989 a úspešne prešlo 145 km rýchlosťou 110 km za hodinu medzi ostatnými autami.

1.3 Oblasti umelej inteligencie

V tejto lekcii sa budeme venovať niektorým oblastiam umelej inteligencie. Hranice medzi nimi nie sú striktné a techniky používané na riešenie problémov v jednej oblasti môžu byť často užitočné pri riešení problémov v inej oblasti. Skutočná sila umelej inteligencie bude spočívať v prepojení všetkých oblastí.

Počítačové videnie

Počítačové videnie je oblasť umelej inteligencie, ktorá sa zaoberá vývojom algoritmov a nástrojov, ktoré dávajú počítačom schopnosť rozumieť vizuálnemu svetu tak ako ľudia. Ide napríklad o úlohy rozpoznávania objektov na obrázkoch, pochopenia ich vzťahov, rozpoznávania farieb a textúr, a potom rozpoznávania pohybov, akcií a ich charakteristík. Keďže táto oblasť sa zaoberá predovšetkým analýzou obrázkov a videí, zoznámime sa aj s niektorými najbežnejšími úlohami tejto oblasti.

Úloha **klasifikácie obrazu** sa používa na určenie typu objektu, ktorý sa nachádza na obrázku. Napríklad určenie, či sa na obrázku nachádza pes, je úlohou klasifikácie obrazov. **Detekcia objektov** je úloha lokalizácie objektov a odpovede na otázku, kde presne sa objekty nachádzajú na obrázku. Takouto úlohou je napríklad orámovanie psa a mačky, ktoré sa nachádzajú na obrázku nižšie. Úloha **segmentácie obrazu** sa používa na určenie presného tvaru objektov, ktoré sa nachádzajú na obrázku. Takže teraz je jemnejšie oddelenie kontúr psa a mačky na treťom obrázku príkladom segmentácie.

Tri základné úlohy počítačového videnia pri práci s obrázkami

Všetky tieto úlohy sú veľmi použiteľné v mnohých disciplínach, ako je autonómne riadenie, analýza lekárskych obrazov alebo analýza satelitných obrazov, a umožňujú nám ľahšie vyhľadávať a organizovať obrazy.

Do ktorých úloh patria nasledujúce problémy:
Zistiť, či sa na obrázku nachádza chodec.
Oddelenie kontúr semaforov, chodníka a chodcov na obrázku.
Chcete vedieť, kde sa na obrázku nachádza značka?

Pokiaľ ide o spracovanie videa, najbežnejšie úlohy sú sledovanie objektov, rozpoznávanie akcií a lokalizácia.

Úloha sledovania objektov, ako už názov napovedá, vám umožňuje sledovať objekty vo videu v reálnom čase. Príkladom sledovania objektov je napríklad sledovanie susedného auta počas autonómnej jazdy alebo sledovanie pohybov hráča počas zápasu.

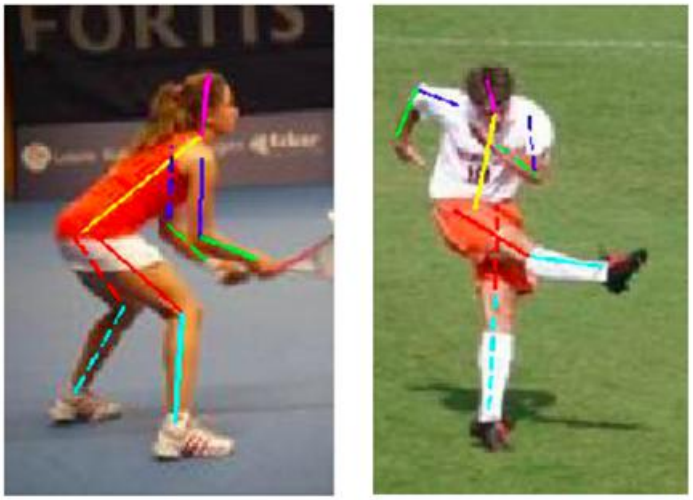
Sledovanie objektov

(obrázok prevzatý z https://docs.ultralytics.com/modes/track/)

Úloha ***rozpoznávania akcií*** je schopnosť rozpoznať a pomenovať akciu, ktorá sa vyskytuje vo videu, napríklad skok do vody alebo zatvorenie okna. Tieto úlohy nám pomáhajú lepšie porozumieť obsahu videa a efektívnejšie ho vyhľadávať.

Príklady rozpoznávania akcií vo videách

Odhad polohy je úloha, ktorá sa zaoberá rozpoznávaním postáv ľudí vo videách v reálnom čase a extrahovaním všetkých kľúčových bodov ich kostry. Ide o najbežnejšie choroidy očí, nosa, úst, ramien, lakťov, pása, rúk, kolien a nôh. Tieto úlohy nám pomáhajú pri interaktívnych animáciách, modelovaní rozšírenej reality a rôznych iných aplikáciách.



Úloha rozpoznávania polohy objektov: Obrázky z dátového súboru Leeds Sports Pose

Aké úlohy musíme vyriešiť, aby sme:
- analyzovať, či sedíme správne,
- rozpoznali východ do dvora pre domáce zvieratá,
- sledovať pohyb zákazníka v obchode?

Neskôr sa dostaneme k dátovým súborom používaným v úlohách počítačového videnia a konvolučných sietí, čo je špeciálny typ neurónovej siete používaný v úlohách spracovania obrazu a videa.

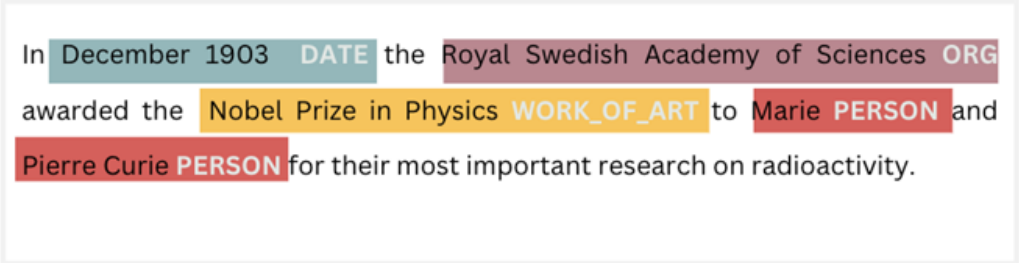
Spracovanie prirodzeného jazyka

Spracovanie prirodzeného jazyka *(NLP*) je oblasť umelej inteligencie, ktorá sa zaoberá úlohami súvisiacimi s porozumením a generovaním prirodzeného jazyka. Ako vieme, existuje viac ako 7 000 jazykov a každý z nich má svoje vlastné špecifiká, pokiaľ ide o slovnú zásobu, gramatické pravidlá a významy. Niektoré bežné úlohy, s ktorými sa stretávame pri spracovaní prirodzeného jazyka, budú opísané nižšie.

Rovnako ako pri úlohách klasifikácie obrazov, aj pri úlohách **klasifikácie** textu sa snažíme určiť, či text patrí do určitej kategórie, alebo nie. Napríklad, či ide o novinový článok na tému šport, či je napísaný v španielčine, či je pozitívny, t. j. obsahuje nejaký kompliment, či je pravdivý alebo nepravdivý a podobne.

Klasifikácia textu

Rozpoznávanie **menovaných entít** je úloha súvisiaca s rozpoznávaním niektorých častí textu, ktoré sú relevantné pre jeho ďalšiu analýzu. Zvyčajne ide o mená osôb, ktoré sa v ňom vyskytujú, dátumy, názvy geolokácií alebo v niektorých odborných textoch, napríklad v oblasti medicíny, symptómy alebo názvy chorôb. Tieto časti textu sa jednotne nazývajú entity.



Príklad označovania menovaných entít

Úlohou **prekladového stroja** je vyvíjať nástroje, ktoré nám umožňujú prekladať obsah z jedného jazyka do obsahu iného jazyka. Zhodneme sa, že táto úloha je základom úspešnej komunikácie a dostupnosti informácií, ale aj že je komplikovaná, pretože každý jazyk a každá kultúra, ktorú jazyk reprezentuje, má svoje vlastné špecifiká, ako sú frázy, idiomy, slang alebo sarkazmus, ktoré sú veľmi náročné na preklad (ako preložiť *„lámu si hlavu"?*).

Systémy odpovedí na otázky sa zaoberajú otázkou, ako nájsť konkrétnu odpoveď na danú otázku. Sú to zovšeobecnenia klasických systémov vyhľadávania informácií a umožňujú nám ľahšie získať potrebné informácie.

Zhrnutie všetkých dôležitých informácií z viacerých rôznych zdrojov sa nazýva úloha **zhrnutia**. Rovnako ako v predchádzajúcej úlohe, zhrnutia, ktoré sú súčasťou úloh zhrnutia, by nám mali uľahčiť prechádzanie väčším množstvom obsahu alebo nám pripomenúť dôležité informácie a podrobnosti obsahu, ktorý sme prečítali.

Zhrnutie

Okrem úloh súvisiacich s textom a textovým obsahom sa spracovanie prirodzeného jazyka zaoberá aj analýzou reči. Existujú najmä dve úlohy: *premena reči na text* a naopak, *premena textu na reč*. Tieto dve skupiny úloh sú obzvlášť dôležité pre vývoj osobných asistentov, programov ako *Siri, Cortana* alebo *Alexa*, ktoré dokážu rozumieť hlasovým správam a podľa toho vykonávať požadované úlohy, napríklad nastaviť budík alebo zavolať niekomu z telefónneho zoznamu.

Do ktorých úloh patria nasledujúce problémy:
Extrakcia názvu organizácie v texte,
Zistenie autora knihy,
Aký je význam slova Netizen?

Generatívna umelá inteligencia (AGI) je oblasť umelej inteligencie, ktorá sa zaoberá generovaním obsahu, ako sú obrázky, text, audio alebo video. Za posledných niekoľko rokov boli pokroky v tejto oblasti impozantné.

ChatGPT je program, ktorý dosiahol prelom v oblasti generovania textového obsahu. V súlade s pokynmi používateľa, tzv. podnetmi, dokáže generovať vhodný textový obsah. Treba mať na pamäti, že takto generované texty nemusia byť absolútne presné, môžu obsahovať nesprávne údaje, vymyslené odkazy alebo urážlivý obsah. Preto by ste pred použitím mali skontrolovať všetko, čo program vygeneroval. Ak si vytvoríte účet na chat.openai.com, môžete si sami vyskúšať, ako program *ChatGPT* funguje. Za programom *ChatGPT* stojí komunita *OpenAI*.

Program StableDiffusion, na rozdiel od *ChatGPT*, ktorý generuje text, generuje obrázky na základe pokynov. Napríklad všetky nižšie uvedené obrázky boli generované týmto programom. Je to open source a dá sa stiahnuť z oficiálneho repozitára GitHub spolu s priloženým kódom. Samotný model môžete otestovať na stránke https://stablediffusionweb.com/. Majte na pamäti, že túto službu používa veľké množstvo ľudí zadarmo a niekedy nie je k dispozícii. Názov samotného programu je populárnou technikou používanou v tejto oblasti.

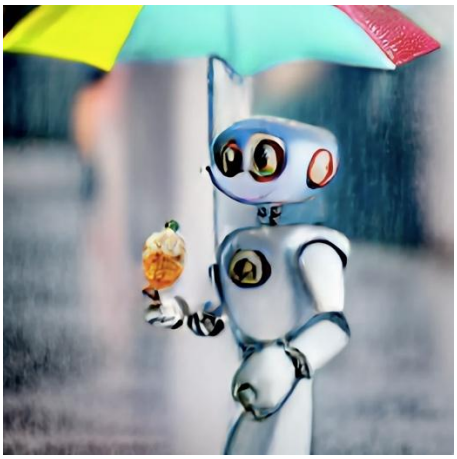
Príklady obrázkov generovaných programom StableDiffusion

Pri generovaní obrázkov je často možné vybrať aj požadovaný štýl nového obrázku. Táto technika sa nazýva **prenos štýlu**. Príklad môžete vidieť na obrázku nižšie.

Okrem obrázkov a textu môže umelá inteligencia generovať aj zvukový obsah. Na tomto odkaze môžete vyskúšať *program* Meta *MusicGen,* kde slovami opíšete, aký druh hudby chcete generovať, a prípadne zanecháte príklad pre prenos štýlu. Potom si môžete vypočuť svoj vlastný obsah. Môžete tiež vyskúšať programy, ktoré vykonávajú prenos štýlu pri generovaní hlasu (napodobňujú hlas inej osoby) alebo sami komponovať hudbu na základe toho, čo už „počuli" v údajoch. Jedným z takýchto projektov je Magenta. Odkaz https://magenta.github.io/listen-to-transformer/ vás naň presmeruje.

Požiadajte ChatGPT, aby vytvoril kvíz s otázkami o umelej inteligencii, a potom skontrolujte, na koľko otázok viete odpovedať.

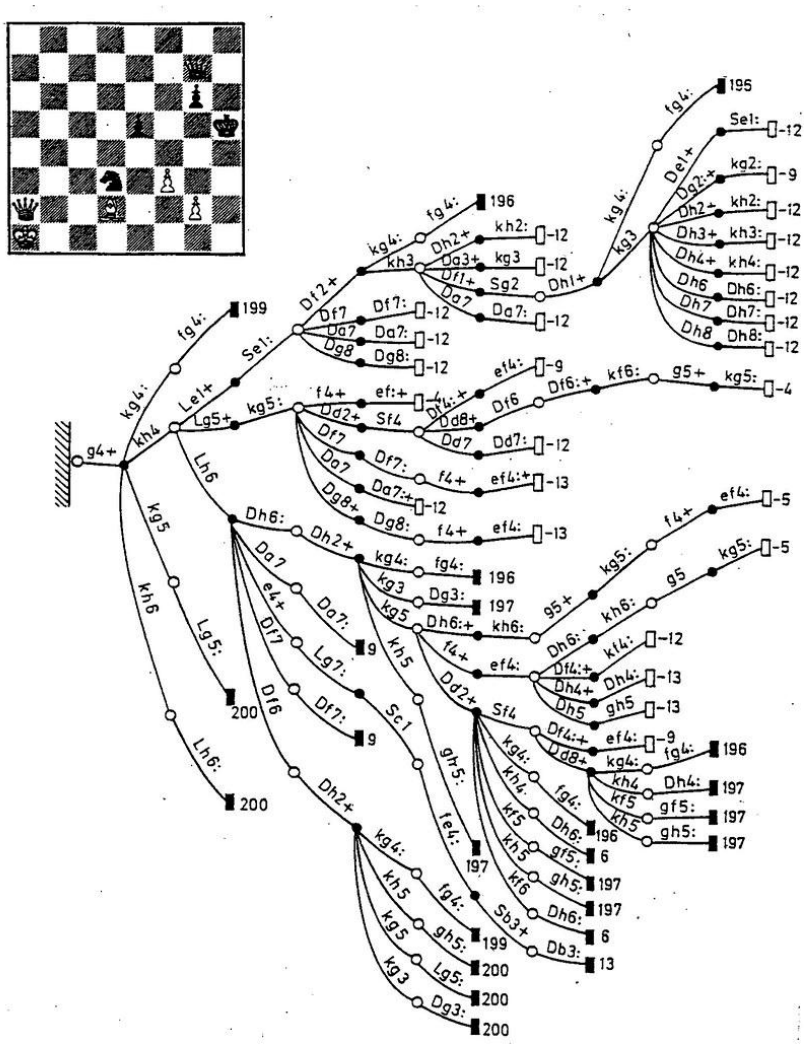
Skúste dať pokyny programom StableDiffusion alebo Dalle-mini, aby vygenerovali obrázok ako tento:



Program DALL-E od OpenAI tiež generuje obrázky na základe pokynov používateľa. Program dalle-mini je verejne dostupná verzia tohto programu. Je k dispozícii na [adrese https://huggingface.co/spaces/dalle-mini/dalle-mini](https://huggingface.co/spaces/dalle-mini/dalle-mini) .

Hranie hier

Jednou z prvých úloh, v ktorých bola umelá inteligencia vyskúšaná, je šachová hra. Vďaka víťazstvu nad veľmajstrom Garrym Kasparovom získala táto oblasť výskumu veľa sympatií a podpory zo strany umelej inteligencie. Hoci majú presný súbor pokynov a pravidiel, hry sa vyznačujú vlastnosťou kombinatorickej explózie – veľkým počtom možných volieb akcií po určitom počte krokov. To ďalej znamená, že hry neumožňujú nájsť riešenia pomocou bežných programovacích techník v primeranom čase. Okrem šachu sa umelá inteligencia preslávila aj v hre go s *programom AlphaGo,* potom v hraní videohier Atari a stratégií ako Dota, Starcraft a iných. Na konci kurzu sa dozviete viac o oblasti nazývanej posilňovacie učenie, ktorá sa v tejto oblasti aktívne uplatňuje.



Kombinatorická explózia v šachu

Skontrolujte, či hra, ktorú máte radi, používa umelú inteligenciu v niektorých svojich segmentoch.

Robotika

Umelá inteligencia často potrebuje zlepšiť správanie a schopnosti fyzických objektov, ako sú roboty, priemyselné stroje, autá, drony, domáce spotrebiče alebo zdravotnícke zariadenia. Informácie o svete sa k týmto objektom dostávajú prostredníctvom hlasových pokynov, záznamov z kamery alebo meraní senzorov a ich úlohou je spracovať ich a premeniť na rozhodnutia. Úlohou umelej inteligencie v tejto oblasti je zlepšiť schopnosti objektov a pomôcť im prijať inteligentné správanie.

1.4 Regulácia umelej inteligencie, právne a etické výzvy

Rýchly pokrok v oblasti technológií umelej inteligencie (AI) vyvolal naliehavé diskusie o ich regulácii. Vzhľadom na to, že systémy AI sú čoraz viac integrované do rôznych sektorov, vrátane zdravotníctva, financií a dopravy, potreba účinných právnych rámcov a etických usmernení sa stáva mimoriadne dôležitou. Tento text skúma zložitosť regulácie AI a zdôrazňuje právne a etické výzvy, ktoré vznikajú v tomto sa vyvíjajúcom prostredí.

Potreba regulácie

Technológie AI majú jedinečné vlastnosti, ktoré ich odlišujú od tradičných technológií. Často fungujú ako „čierne skrinky", kde ich rozhodovacie procesy nie sú transparentné ani pre ich vývojárov. Táto netransparentnosť vyvoláva značné obavy týkajúce sa zodpovednosti, zaujatosti a potenciálu spôsobiť škodu. Keďže systémy AI môžu ovplyvňovať kritické oblasti, ako sú rozhodnutia o zamestnaní, výsledky trestného súdnictva a verejná bezpečnosť, v stávke je efektívna regulácia.

Kľúčové hnacie sily regulácie:

- **Bezpečnosť a blaho**: Primárnym záujmom je zabezpečiť, aby systémy umelej inteligencie neohrozovali verejnú bezpečnosť ani individuálne práva. Incidenty týkajúce sa autonómnych vozidiel alebo zaujatých algoritmov v procesoch prijímania zamestnancov podčiarkujú potrebu regulačného dohľadu.
- **Zodpovednosť**: Vytvorenie jasných mechanizmov zodpovednosti je nevyhnutné na riešenie otázky, kto je zodpovedný, keď systémy umelej inteligencie spôsobia škodu alebo prijmú nesprávne rozhodnutia.
- **Etické používanie**: Keďže systémy umelej inteligencie môžu udržiavať predsudky prítomné v trénovacích dátach, regulácia musí zabezpečiť dodržiavanie etických noriem, aby sa zabránilo diskriminácii a nespravodlivému za

Právne výzvy pri regulácii umelej inteligencie

Právne prostredie týkajúce sa regulácie umelej inteligencie je komplexné a často zaostáva za technologickým pokrokom. Tradičné právne rámce nemusia dostatočne riešiť nuansy systémov umelej inteligencie.

1. Definovanie zodpovednosti

Jednou z najväčších výziev je určenie zodpovednosti v prípade, že systém umelej inteligencie spôsobí škodu. Vznikajú otázky, či zodpovednosť by mali niesť vývojári, používatelia alebo samotná umelá inteligencia. Napríklad, ak sa autonómne vozidlo zapletie do nehody, mal by byť zodpovedný výrobca alebo majiteľ?

2. Otázky duševného vlastníctva

S rastúcou popularitou obsahu generovaného umelou inteligenciou vznikajú otázky týkajúce sa práv duševného vlastníctva. Kto vlastní práva k dielam vytvoreným umelou inteligenciou? Súčasné zákony nemusia dostatočne pokrývať tieto scenáre, čo môže viesť k potenciálnym konfliktom týkajúcim sa vlastníctva a autorských práv.

3. Obavy týkajúce sa ochrany osobných údajov

Systémy umelej inteligencie sú vo veľkej miere závislé od údajov na účely školenia a prevádzky. Zber a používanie osobných údajov vyvoláva významné obavy o ochranu súkromia. Predpisy musia vyvažovať potrebu údajov na zlepšenie schopností umelej inteligencie s právami jednotlivcov na súkromie a ochranu údajov.

Etické výzvy pri regulácii umelej inteligencie

Okrem právnych hľadísk zohrávajú pri formovaní regulácie umelej inteligencie kľúčovú úlohu aj etické výzvy.

1. Predsudky a spravodlivosť

Systémy umelej inteligencie môžu neúmyselne udržiavať predsudky prítomné v trénovacích údajoch, čo vedie k nespravodlivým výsledkom. Regulačné orgány musia stanoviť usmernenia na zabezpečenie spravodlivosti a

zmiernenie predsudkov v algoritmoch umelej inteligencie. To zahŕňa nielen technické riešenia, ale aj záväzok dodržiava

2. Transparentnosť a vysvetliteľnosť

„Čierna skrinka" mnohých systémov umelej inteligencie komplikuje snahy o transparentnosť. Používatelia a dotknuté osoby často nemajú prehľad o tom, ako sa prijímajú rozhodnutia. Regulácia by mala podporovať vysvetliteľnosť a zabezpečiť, aby jednotlivci mohli pochopiť dôvody rozhodnutí prijatých systémami umelej inteligencie.

3. Informovaný súhlas

Keďže technológie umelej inteligencie majú čoraz väčší vplyv na osobné rozhodnutia, ako sú schvaľovanie úverov alebo žiadosti o zamestnanie, zabezpečenie informovaného súhlasu sa stáva kritickým. Jednotlivci by si mali byť vedomí toho, ako sa používajú ich údaje a ako umelá inteligencia ovplyvňuje rozhodovacie procesy, ktoré sa ich týkajú.

Prístupy k regulácii

Vzhľadom na tieto výzvy sa objavili rôzne prístupy k regulácii umelej inteligencie.

1. Rámce založené na riziku

Európska únia navrhla prístup k regulácii umelej inteligencie založený na riziku, ktorý kategorizuje aplikácie na základe úrovne rizika – od minimálneho po neprijateľné riziko. Tento rámec umožňuje prispôsobené regulácie, ktoré riešia konkrétne obavy spojené s rôznymi typmi aplikácií umelej inteligencie.

2. Neustály dohľad

Regulácia umelej inteligencie si vyžaduje neustálu ostražitosť vzhľadom na jej rýchlo sa meniacu povahu. Regulačné orgány sa musia prispôsobovať novým technologickým vývojom a neustále hodnotiť vplyv existujúcich predpisov na spoločnosť.

3. Spolupráca pri riadení

Účinná regulácia môže zahŕňať spoluprácu medzi vládami, zainteresovanými stranami z odvetvia a organizáciami občianskej spoločnosti. Zapojenie rôznych perspektív môže viesť k komplexnejším predpisom, ktoré zohľadňujú rôzne záujmy a etické hľadiská.

Záver

Regulácia umelej inteligencie predstavuje mnohostranné právne a etické výzvy, ktoré si vyžadujú starostlivé zváženie a proaktívne opatrenia. Keďže umelá inteligencia naďalej preniká do rôznych aspektov života, je nevyhnutné vytvoriť robustný regulačný rámec na ochranu verejného blaha, zabezpečenie zodpovednosti a dodržiavanie etických noriem. Riešením týchto výziev prostredníctvom spolupráce pri riadení a adaptívnych regulačných prístupov môže spoločnosť využiť výhody umelej inteligencie a zároveň účinne zmierniť jej riziká.

1.5 Úzka, všeobecná a superinteligencia

Teraz, keď poznáme oblasti uplatnenia umelej inteligencie a jej dosah, môžeme tiež predstaviť pojmy ako úzka a všeobecná umelá inteligencia, superinteligencia a singularita umelej inteligencie.

Videli sme, že uvedené príklady programov umelej inteligencie sa zameriavajú na riešenie jednej konkrétnej úlohy, napríklad hranie šachu, prekladanie jazykov, segmentácia obrázkov a podobne. Hovoríme, že takéto programy majú **úzku umelú inteligenciu (***narrow AI)*. Na rozdiel od umelej inteligencie existuje aj **všeobecná umelá inteligencia (***general AI)*. Programy tejto skupiny by sa mali viac približovať ľudskej inteligencii, aby umožnili riešenie množstva rôznych úloh. Podľa popredných výskumníkov v tejto disciplíne ( ) sú programy tohto typu stále v plienkach a v dohľadnej budúcnosti nebudú vyvinuté.

Superinteligencia je termín, ktorý označuje inteligenciu väčšiu ako ľudská. Takáto inteligencia by nám mohla pomôcť riešiť problémy, ako je globálne otepľovanie, zabezpečenie dostatku potravín alebo nájdenie lieku na rakovinu. Hypoteticky by sa mohla vymknúť spod kontroly, pokračovať v zdokonaľovaní a v niektorých ohľadoch ohroziť ľudstvo. **Singularita umelej inteligencie** je teoretický pojem označujúci dominanciu umelej inteligencie nad ľudskou inteligenciou a často sa vyskytuje ako téma v sci-fi filmoch a knihách.

Pohľad na umelú inteligenciu z tohto uhla je osobitne zaujímavý pre filozofov, historikov a výskumníkov v oblasti spoločenských vied. Izraelský historik Yuval Noah Harari a švédsky filozof Nick Bostrom o nich napísali. Mnoho z týchto príbehov nájdete na YouTube.

Skúste vymyslieť ďalší príklad uplatnenia superinteligentnosti. Aké problémy ľudia napriek rozvoju spoločnosti, vedy a techniky stále nevedia vyriešiť?

Preskúmajte význam pojmu *existenciálne riziko*. Ako ho vnímate?

Oblasť umelej inteligencie (AI) zahŕňa rôzne systémy a schopnosti, ktoré možno rozdeliť do troch základných typov: **úzka inteligencia**, **všeobecná inteligencia** a **superinteligencia**. Každá kategória predstavuje inú úroveň zložitosti a schopností systémov AI, čo odzrkadľuje neustály vývoj technológie a jej potenciálny vplyv na spoločnosť. Pochopenie týchto rozdielov je kľúčové pre pochopenie súčasného stavu AI a jej budúcich dôsledkov.

Úzka inteligencia (slabá AI)

Úzka inteligencia, často označovaná ako slabá AI, je navrhnutá na vykonávanie špecifických úloh alebo riešenie konkrétnych problémov. Tieto systémy AI vynikajú vo svojich určených funkciách, ale nemajú schopnosť fungovať mimo svojich vopred definovaných parametrov.

Charakteristiky úzkej inteligencie:

- **Funkčnosť špecifická pre danú úlohu**: Systémy úzkej umelej inteligencie sú navrhnuté na vykonávanie konkrétnych úloh, ako je rozpoznávanie obrazov, preklad jazykov alebo hranie hier, ako je šach. Fungujú efektívne v rámci svojho obmedzeného rozsahu, ale nemajú všeobecné schopnosti uvažovania.
- **Učenie založené na údajoch**: Tieto systémy sa pri trénovaní vo veľkej miere spoliehajú na veľké súbory údajov. Algoritmy strojového učenia analyzujú vzory v údajoch, aby na základe poskytnutých informácií mohli robiť predpovede alebo rozhodnutia.

- **Nedostatok sebauvedomenia**: Úzka umelá inteligencia nemá vedomie ani sebauvedomenie. Funguje na základe algoritmov a vopred definovaných pravidiel bez pochopenia kontextu alebo dôsledkov svojich akcií.

Príklady úzkej inteligencie:

- **Hlasoví asistenti**: Aplikácie ako Siri a Alexa dokážu vykonávať úlohy, ako je nastavovanie pripomienok alebo odpovedanie na otázky, ale nedokážu viesť konverzácie nad rámec svojich naprogramovaných schopností.
- **Odporúčacie systémy**: Platformy ako Netflix a Amazon používajú úzku umelú inteligenciu na navrhovanie obsahu na základe preferencií používateľov, pričom analyzujú minulé správanie, aby predpovedali budúce voľby.
- **Autonómne vozidlá**: Hoci autonómne vozidlá využívajú komplexné algoritmy na navigáciu po cestách, stále sú obmedzené na úlohy súvisiace s riadením a nedokážu vykonávať iné kognitívne funkcie podobné ľudským.

Všeobecná inteligencia (silná umelá inteligencia)

Všeobecná inteligencia, známa aj ako umelá všeobecná inteligencia (AGI), sa týka teoretickej formy umelej inteligencie, ktorá má schopnosť rozumieť, učiť sa a aplikovať vedomosti v širokej škále úloh na úrovni porovnateľnej s ľudskou inteligenciou.

Charakteristiky všeobecnej inteligencie:

- **Všestranné učenie**: AGI dokáže generalizovať vedomosti z jednej oblasti do druhej, čo jej umožňuje prispôsobiť sa novým situáciám bez potreby rozsiahleho preškolenia.
- **Uvažovanie a riešenie problémov**: Na rozdiel od úzkej umelej inteligencie, AGI dokáže uvažovať o zložitých problémoch, rozumieť abstraktným pojmom a robiť rozhodnutia na základe neúplných informácií.
- **Interakcia podobná ľudskej**: Systémy AGI by boli schopné viesť prirodzené konverzácie s ľuďmi, preukazovať emocionálne porozumenie a sociálne povedomie.

Súčasný stav všeobecnej inteligencie:

V súčasnosti zostáva AGI väčšinou v teoretickej rovine. Hoci sa dosiahol významný pokrok v oblasti strojového učenia a neurónových sietí, žiadny existujúci systém nevlastní celý rozsah kognitívnych schopností spojených s ľudskou inteligenciou. Výskumníci naďalej skúmajú cesty k dosiahnutiu AGI, pričom sa zameriavajú na vývoj algoritmov, ktoré sa dokážu učiť flexibilnejšie a autonómnejšie.

Superinteligencia

Superinteligencia sa týka hypotetickej formy umelej inteligencie, ktorá prekonáva ľudskú inteligenciu prakticky vo všetkých aspektoch. Tento koncept vyvoláva hlboké otázky o budúcnosti ľudstva a etických dôsledkoch vytvárania strojov, ktoré by mohli potenciálne prekonať svojich tvorcov.

Charakteristiky superinteligentnosti:

- **Exponenciálny rast**: Superinteligentné systémy by mali schopnosť rýchleho sebazdokonaľovania, čo by viedlo k explózii inteligencie, pri ktorej by stroje mohli posilniť svoje vlastné schopnosti nad rámec ľudského chápania.
- **Schopnosti riešiť problémy**: Takéto systémy by mohli riešiť komplexné globálne výzvy – od klimatických zmien po eradikáciu chorôb – efektívnejšie ako akýkoľvek človek alebo kolektívna skupina.

- **Etické hľadiská**: Vývoj superinteligentných systémov prináša významné etické dilemy týkajúce sa kontroly, bezpečnosti a potenciálnych dôsledkov pre spoločnosť. Zabezpečenie toho, aby superinteligentné systémy boli v súlade s ľudskými hodnotami, je kritickou otázkou.

Teoretické implikácie:

Diskusia o superinteligencii často zahŕňa scenáre o „singularite", bode, v ktorom sa technologický rast stáva nekontrolovateľným a nezvratným. Tento koncept vyvolal debaty medzi vedcami, etickými odborníkmi a futuristami o potenciálnych rizikách spojených s vytvorením entít, ktoré by mohli fungovať nezávisle od ľudského dohľadu.

Výzvy pri pokroku smerom k všeobecnej a superinteligencii

Hoci snaha o dosiahnutie všeobecnej a superinteligentnej umelej inteligencie ponúka vzrušujúce možnosti, je potrebné riešiť niekoľko výziev:

1. **Technické obmedzenia**: Súčasné systémy umelej inteligencie majú problémy so zovšeobecňovaním; vynikajú v úzkych úlohách, ale zlyhávajú, keď čelia neznámym kontextom alebo problémom mimo ich trénovacích dát.
2. **Etické obavy**: Vývoj AGI a superinteligentnej umelej inteligencie vyvoláva etické otázky týkajúce sa autonómie, rozhodovacej právomoci a potenciálu zneužitia vo vojenských alebo sledovacích aplikáciách.
3. **Bezpečnostné opatrenia**: Zabezpečenie bezpečného fungovania pokročilých systémov umelej inteligencie je mimoriadne dôležité. Výskumníci sa zasadzujú za robustné bezpečnostné protokoly a stratégie zosúlaďovania, aby sa predišlo ne
4. **Vnímanie verejnosti**: Nedorozumenia o schopnostiach umelej inteligencie môžu viesť k nerealistickým očakávaniam alebo obavám týkajúcim sa jej vplyvu na spoločnosť. Vzdelávanie verejnosti je nevyhnutné na podporu informovaných diskusií o technológiách umelej inteligencie.

Záver

Rozdiely medzi úzkou inteligenciou, všeobecnou inteligenciou a superinteligenciou poukazujú na rozmanitosť v oblasti umelej inteligencie. Kým úzka umelá inteligencia naďalej prosperuje v rôznych aplikáciách, snaha o dosiahnutie AGI zostáva ambicióznym cieľom výskumníkov po celom svete. Perspektíva superinteligentnosti vyvoláva nadšenie aj opatrnosť, keďže spoločnosť sa vyrovnáva s dôsledkami vytvorenia strojov, ktoré by mohli prekonať ľudský intelekt. Vzhľadom na pokračujúci pokrok bude riešenie etických otázok a zabezpečenie zodpovedného vývoja kľúčové pre využitie výhod umelej inteligencie a zároveň pre účinné zmiernenie jej rizík.

2. STROJOVÉ UČENIE

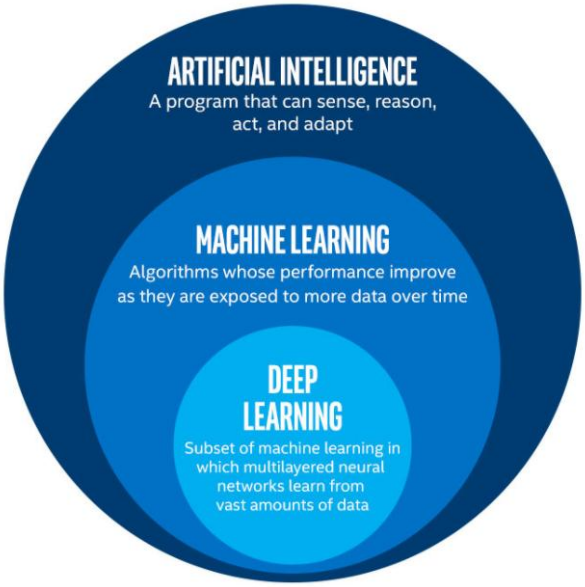
Vitajte v téme **strojového učenia (ML)**! Strojové učenie je podmnožinou umelej inteligencie, ktorá sa zameriava na vývoj algoritmov, ktoré umožňujú počítačom učiť sa z údajov a robiť na ich základe predpovede. V tejto časti získate hlboké pochopenie základných pojmov strojového učenia, rôznych typov učenia (supervidované, nesupervidované, polo-supervidované a posilňovacie učenie) a dôležitosti údajov v tomto procese. Zaoberať sa budeme aj krokmi potrebnými na vytvorenie a overenie modelov strojového učenia a spôsobmi, ako tieto modely možno použiť na riešenie reálnych problémov.

2.1 Vzťah medzi umelou inteligenciou a strojovým učením

Umelá inteligencia (AI) a strojové učenie (ML) sú dve prepojené oblasti, ktoré v posledných rokoch vzbudili značnú pozornosť vďaka svojmu transformatívnemu vplyvu na technológiu a spoločnosť. Hoci sa často používajú ako synonymá, v širšom kontexte výpočtovej inteligencie predstavujú odlišné pojmy. Porozumenie ich vzťahu je kľúčové pre pochopenie fungovania a vývoja moderných systémov umelej inteligencie.

1. **Umelá inteligencia (AI)** je široká oblasť, ktorá zahŕňa rôzne technológie zamerané na vytváranie systémov schopných vykonávať úlohy, ktoré zvyčajne vyžadujú ľudskú inteligenciu. Tieto úlohy zahŕňajú riešenie problémov, porozumenie prirodzenému jazyku, rozpoznávanie vzorov a rozhodovanie.

2. **Strojové učenie (ML)** je podmnožinou AI, ktorá sa zameriava na vývoj algoritmov, ktoré umožňujú počítačom učiť sa z údajov a robiť predpovede na ich základe. Namiesto toho, aby boli explicitne naprogramované na vykonávanie úlohy, algoritmy ML používajú štatistické techniky na identifikáciu vzorov v údajoch a zlepšovanie svojho výkonu v priebehu času.



Príklad:

- **Aplikácia umelej inteligencie**: Virtuálny asistent, ako je Siri alebo Alexa, používa umelú inteligenciu na porozumenie a reagovanie na hlasové príkazy.
- **Aplikácia ML**: Systém odporúčaní na Netflixe využíva ML na navrhovanie filmov a televíznych programov na základe vašej histórie sledovania.

Na vizualizáciu vzťahu:

- **AI** je všeobecný pojem označujúci stroje, ktoré sú schopné vykonávať úlohy spôsobom, ktorý by sme považovali za „inteligentný".
- **ML** je špecifický prístup k dosiahnutiu AI, pri ktorom stroje dostávajú údaje a učia sa samy.

Programovanie založené na údajoch

Programovanie založené na údajoch je paradigma, pri ktorej správanie programu riadia údaje, a nie pevne zakódovaná logika. Tento prístup umožňuje flexibilnejšie a prispôsobivejšie systémy. V programovaní založenom na údajoch údaje určujú tok programu, čo môže byť obzvlášť užitočné v aplikáciách, ako je spracovanie, transformácia a analýza údajov.

Príklady:

- **AWK** a **sed** na spracovanie textu: Tieto nástroje spracúvajú textové údaje na základe vzorov a pravidiel definovaných v samotných údajoch.
- **XSLT** na transformáciu dokumentov XML: XSLT používa dáta XML na definovanie toho, ako by mali byť transformované iné dokumenty XML.
- **Procmail** na filtrovanie e-mailov: Procmail používa pravidlá definované v konfiguračných súboroch na filtrovanie a spracovanie prichádzajúcich e-mailov.

Základné pojmy strojového učenia

Strojové učenie zahŕňa niekoľko kľúčových pojmov:

1. **Supervidované učenie**: Algoritmus je trénovaný na označených údajoch, čo znamená, že každý trénovací príklad je spárovaný s výstupnou značkou. Cieľom je naučiť sa mapovanie zo vstupov na výstupy, ktoré možno použiť na predikciu značiek pre nové údaje.
    - **Príklad**: Predpovedanie cien domov na základe charakteristík, ako je veľkosť, poloha a počet spální. Trénovacie údaje zahŕňajú domy so známymi cenami.

2. **Nekontrolované učenie**: Algoritmus dostane údaje bez explicitných pokynov, čo s nimi robiť. Snaží sa nájsť skryté vzory alebo vnútorné štruktúry vo vstupných údajoch.
    - **Príklad**: Segmentácia zákazníkov v marketingu, kde algoritmus zoskupuje zákazníkov na základe nákupného správania bez vopred definovaných označení.

3. **Posilňovacie učenie**: Algoritmus sa učí interakciou s prostredím a prijímaním spätnej väzby vo forme odmien alebo trestov. Jeho cieľom je naučiť sa stratégiu, ktorá maximalizuje kumulatívne odmeny.
    - **Príklad**: Trénovanie robota na navigáciu v bludisku, kde dostáva odmeny za dosiahnutie konca a tresty za narazenie do stien.

4. **Kľúčové prvky**:
    - **Úloha**: Problém, ktorý treba vyriešiť.
    - **Skúsenosti**: Dáta použité na učenie.
    - **Výkon**: Meradlo toho, ako dobre algoritmus rieši úlohu.

Príklad:

- **Úloha**: Klasifikácia e-mailov ako spam alebo nie spam.
- **Skúsenosti**: Dataset e-mailov označených ako spam alebo nie spam.
- **Výkonnosť**: Presnosť klasifikácie nových, doteraz neviditeľných e-mailov.

Prepojenie medzi AI a ML

Vzťah medzi umelou inteligenciou a strojovým učením je v podstate synergický. Hoci všetky strojové učenia sú formou umelej inteligencie, nie všetky systémy umelej inteligencie využívajú techniky strojového učenia. Takto vzájomne interagujú:

1. **Základy AI**: AI poskytuje teoretický rámec pre vytváranie inteligentných systémov, zatiaľ čo ML ponúka praktické techniky na implementáciu týchto systémov. Napríklad spracovanie prirodzeného jazyka (NLP), významná oblasť AI, sa vo veľkej miere spolieha na algoritmy ML, aby efektívne spracovávalo a rozumelo ľudskému jazyku.
2. **Mechanizmy učenia**: Tradičné systémy AI boli primárne založené na pravidlách a pri rozhodovaní sa spoliehali na vopred definované pravidlá a logiku. Tieto systémy však zápasili s neistotou a variabilitou údajov. ML zaviedlo schopnosť systémov AI učiť sa zo skúseností, čo im umožňuje prispôsobovať sa novým informáciám a robiť pravdepodobnostné rozhodnutia. Tento pokrok je obzvlášť dôležitý v dynamických prostrediach, ako sú autonómne vozidlá, ktoré musia navigovať v zložitých cestných podmienkach.
3. **Aplikácie v robotike**: Synergia medzi umelou inteligenciou a strojovým učením je evidentná v robotike. Inteligentné roboty využívajú umelú inteligenciu na kognitívne schopnosti a zároveň používajú algoritmy strojového učenia na navigáciu, vyhýbanie sa prekážkam a vykonávanie úloh. Napríklad skladové roboty môžu autonómne navigovať vo svojom prostredí pomocou počítačového videnia poháňaného technikami strojového učenia.
4. **Prediktívna analýza**: V obchodných aplikáciách prediktívna analýza využíva umelú inteligenciu aj strojové učenie na prognózovanie budúcich výsledkov na základe historických údajov. Umelá inteligencia poskytuje základný rámec pre vytváranie prediktívnych modelov, zatiaľ čo strojové učenie zlepšuje schopnosť týchto modelov učiť sa z údajov a generovať presné predpovede.

Kľúčové techniky v strojovom učení

Strojové učenie zahŕňa niekoľko techník, ktoré prispievajú k jeho účinnosti pri vylepšovaní aplikácií umelej inteligencie:

1. **Supervidované učenie**: V tomto prístupe sú algoritmy trénované na označených dátových súboroch, kde je známy správny výstup. Model sa učí mapovať vstupy na výstupy na základe týchto trénovacích dát, čo mu umožňuje robiť predikcie na nových, neznámych dátach.
2. **Nekontrolované učenie**: Na rozdiel od kontrolovaného učenia, nekontrolované učenie zahŕňa trénovanie algoritmov na neoznačených údajoch. Model identifikuje vnútorné vzory alebo štruktúry v údajoch bez explicitného usmernenia, čo má hľadať.
3. **Posilňované učenie**: Táto technika zahŕňa trénovanie agenta prostredníctvom interakcií s jeho prostredím formou pokusov a omylov. Agent dostáva spätnú väzbu vo forme odmien alebo trestov na základe svojich akcií, čo mu umožňuje postupne sa naučiť optimálne stratégie.
4. **Hlboké učenie**: Podskupina ML, ktorá využíva umelé neurónové siete s viacerými vrstvami (hlboké siete) na analýzu komplexných dátových súborov, ako sú obrázky alebo zvukové signály. Hlboké učenie je hnacou silou mnohých nedávnych pokrokov v oblastiach, ako je počítačové videnie a spracovanie prirodzeného jazyka.

Aplikácie v reálnom svete

Integrácia umelej inteligencie a strojového učenia viedla k významnému pokroku v rôznych odvetviach:

- **Zdravotníctvo**: Diagnostické nástroje poháňané umelou inteligenciou využívajú algoritmy strojového učenia na analýzu lekárskych snímok na účely detekcie chorôb alebo predpovedania výsledkov liečby pacientov na základe historických zdravotných údaj
- **Financie**: Vo finančnom sektore sa modely ML používajú na detekciu podvodov prostredníctvom analýzy transakčných vzorov v reálnom čase, čím poskytujú bankám nástroje na efektívne zmierňovanie rizík.
- **Autonómne vozidlá**: Samojazdiace autá využívajú kombináciu techník umelej inteligencie, ako je počítačové videnie a rozhodovanie, spolu s algoritmami strojového učenia, ako je hlboké učenie, na interpretáciu senzorických údajov z kamier a systémov LIDAR.
- **Maloobchod**: Platformy elektronického obchodu využívajú algoritmy ML na personalizované odporúčania na základe správania a preferencií používateľov, čím zlepšujú zákaznícku skúsenosť.

Výzvy vo vzťahu medzi umelou inteligenciou a ML

Napriek svojim potenciálnym výhodám vzťah medzi umelou inteligenciou a strojovým učením prináša aj výzvy:

1. **Kvalita údajov**: Účinnosť algoritmov ML vo veľkej miere závisí od kvality trénovacích údajov. Nekvalitné alebo skreslené údaje môžu viesť k nepresným predikciám alebo posilniť existujúce skreslenia v systémoch AI.
2. **Interpretovateľnosť**: Mnohé pokročilé modely ML, najmä siete hlbokého učenia, fungujú ako „čierne skrinky", čo používateľom sťažuje pochopenie toho, ako sa prijímajú rozhodnutia. Táto nedostatočná transparentnosť môže brániť dôvere v aplikácie AI.
3. **Etické otázky**: Vzhľadom na to, že systémy umelej inteligencie majú čoraz väčší vplyv na kritické oblasti, ako je nábor zamestnancov alebo presadzovanie práva, etické otázky týkajúce sa spravodlivosti, zodpovednosti a transparentnosti nadobúd

Záver

Vzťah medzi umelou inteligenciou a strojovým učením je základom pre pochopenie moderných technologických pokrokov. Hoci strojové učenie slúži ako výkonný nástroj v širšej oblasti umelej inteligencie, je dôležité si uvedomiť odlišné úlohy, ktoré každý z nich zohráva pri vývoji inteligentných systémov schopných transformovať odvetvia a zlepšovať životy. Keďže obe oblasti sa naďalej spoločne vyvíjajú, riešenie výziev súvisiacich s kvalitou údajov, interpretovateľnosťou a etikou bude kľúčové pre zodpovedné využitie ich plného potenciálu.

2.2 Programovanie založené na údajoch

Význam veľkého množstva údajov v strojovom učení

V oblasti strojového učenia (ML) sa údaje často označujú ako „palivo", ktoré poháňa algoritmy a modely. Účinnosť a presnosť systémov strojového učenia vo veľkej miere závisí od dostupnosti veľkého množstva vysokokvalitných údajov. Táto časť zdôrazňuje význam údajov v ML, predstavuje koncept programovania založeného na údajoch a vysvetľuje základné pojmy, ako sú dátové súbory, inštancie, atribúty, vstupné premenné a modely.

Úloha dát v strojovom učení

Dáta slúžia ako základ, na ktorom sú postavené modely strojového učenia. Bez dostatočného množstva dát nemôžu algoritmy strojového učenia efektívne učiť sa vzory ani robiť predpovede. Dôležitosť veľkých dátových súborov možno zhrnúť do nasledujúcich bodov:

1. **Učenie sa z príkladov**: Strojové učenie je v podstate o učení sa z príkladov. Veľké dátové súbory poskytujú širokú škálu inštancií, ktoré pomáhajú algoritmom identifikovať vzory a vzťahy v rámci údajov.
2. **Zlepšená presnosť**: Modely trénované na väčších dátových súboroch majú tendenciu lepšie generalizovať na neviditeľné dáta, čo vedie k zlepšenej presnosti predpovedí. To je obzvlášť dôležité v aplikáciách, ako je diagnostika v zdravotníctve alebo finančné prognózy, kde je presnosť životne dôležitá.
3. **Robustnosť**: Komplexný súbor údajov, ktorý zahŕňa rôzne scenáre, umožňuje modelom efektívnejšie riešiť okrajové prípady a anomálie. Táto robustnosť je nevyhnutná pre reálne aplikácie, kde môžu byť údaje nepresné alebo neúplné.
4. **Reprezentácia vlastností**: Veľké dátové súbory umožňujú extrakciu významných vlastností, ktoré môžu výrazne zlepšiť výkon modelu. S väčším množstvom údajov sa algoritmy môžu naučiť komplexné reprezentácie, ktoré zachytávajú základné trendy.
5. **Zmiernenie pretrénovania**: Disponovanie značným množstvom trénovacích dát pomáha zabrániť pretrénovaniu, pri ktorom sa model naučí šum namiesto základného rozdelenia. Väčší súbor dát poskytuje lepšiu aproximáciu skutočného rozdelenia dát.

Programovanie založené na údajoch

Programovanie založené na údajoch je prístup, ktorý kladie dôraz na využívanie údajov ako primárneho hnacieho motora pre rozhodovanie a vývoj algoritmov. V tomto paradigme sa programátori zameriavajú na zbieranie, analýzu a využívanie údajov na informovanie svojich postupov kódovania, namiesto toho, aby sa spoliehali výlučne na vopred definované pravidlá alebo logiku.

Prečo potrebujeme programovanie založené na údajoch

1. **Adaptabilita**: Programovanie založené na údajoch umožňuje systémom prispôsobovať sa meniacim sa podmienkam prostredníctvom neustáleho učenia sa z nových údajov.
2. **Vylepšené rozhodovanie**: Využitím veľkých dátových súborov môžu vývojári vytvárať informovanejšie algoritmy, ktoré vedú k lepším výsledkom.
3. **Automatizácia**: Prístupy založené na údajoch uľahčujú automatizáciu tým, že umožňujú strojom učiť sa z historických údajov a robiť predpovede bez ľudského zásahu.
4. **Škálovateľnosť**: Ako je k dispozícii viac údajov, systémy môžu škálovať svoje schopnosti bez potreby významných zmien v ich základnej architektúre.

Základné pojmy strojového učenia

1. **Dataset**

Dataset je štruktúrovaná zbierka údajov používaná na trénovanie modelov strojového učenia. Skladá sa z viacerých inštancií (dátových bodov) usporiadaných do riadkov a stĺpcov, kde každý stĺpec predstavuje atribút alebo vlastnosť.

2. **Inštancia**

Inštancia sa vzťahuje na jeden záznam alebo pozorovanie v rámci dátového súboru. Každá inštancia obsahuje hodnoty pre rôzne atribúty, ktoré opisujú jej charakteristiky.

3. **Atribút**

Atribúty sú jednotlivé premenné alebo vlastnosti, ktoré opisujú inštanciu v súbore údajov. Môžu byť numerické (napr. vek, plat) alebo kategorické (napr. pohlavie, farba). Atribúty sú kľúčové pre definovanie vstupného priestoru pre algoritmy strojového učenia.

4. **Vstupné premenné**

Vstupné premenné sú špecifické atribúty používané ako prediktory v modeli strojového učenia. Tieto premenné poskytujú potrebné informácie, aby sa model mohol naučiť vzory a robiť predikcie o výstupných premenných (cieľoch).

5. **Výstupné premenné**

Výstupné premenné sú cieľové hodnoty, ktoré model snaží predpovedať na základe vstupných premenných. Napríklad v modeli predpovedania cien nehnuteľností môžu vstupné premenné zahŕňať rozlohu a polohu, zatiaľ čo výstupnou premennou by bola odhadovaná cena.

Koncepcia modelu

V strojovom učení je model definovaný ako matematické znázornenie, ktoré mapuje dané vstupné premenné na výstupné premenné na základe vzorov naučených z trénovacích dát. Proces zahŕňa:

1. **Trénovanie**: Počas trénovania sa model učí z označených prípadov v dátovom súbore úpravou svojich parametrov s cieľom minimalizovať chyby predikcie.
2. **Mapovanie**: Po trénovaní môže model prijať nové vstupné premenné a generovať zodpovedajúce výstupné predikcie na základe naučených mapovaní.
3. **Hodnotenie**: Výkon modelu sa hodnotí pomocou metrík, ako sú presnosť, precíznosť, spomienka a skóre F1 na validačných alebo testovacích dátových súboroch.

Význam veľkého množstva vhodných údajov v strojovom učení nemožno preceňovať; je nevyhnutný pre trénovanie efektívnych modelov schopných robiť presné predikcie v reálnych aplikáciách. Programovanie založené na údajoch sa javí ako dôležitá metodika, ktorá využíva túto hojnosť údajov na informovanie o vývoji algoritmov a rozhodovacích procesoch. Porozumenie základným pojmom, ako sú súbory údajov, inštancie, atribúty, vstupné premenné, výstupné premenné a modely, vytvára základ pre ďalšie skúmanie techník strojového učenia a ich aplikácií v rôznych oblastiach. Keďže naďalej využívame silu údajov v strojovom učení, je nevyhnutné uprednostniť kvalitné postupy zberu a správy údajov, aby sa zabezpečili robustné a spoľahlivé výsledky v systémoch umelej inteligencie.

Tento príbeh začneme hlavolamom. Skúste z čísel v ľavom stĺpci (budeme ich nazývať **vstupy**) vyvodiť, ako súvisia s číslami v pravom stĺpci (budeme ich nazývať **výstupy)**.

Vstup	Výstup
1, 9, 5, 4	19

Vstup	Výstup
3, 8, 11	2
6, 7, 9, 2, 2	26
2, 3, 6	11

Predpokladáme, že táto úloha vás príliš nezamestnávala a že už v druhom alebo treťom riadku tabuľky ste prišli na to, že čísla v druhom stĺpci, t. j. výstup, predstavujú súčet čísel, ktoré sú v ľavom stĺpci, t. j. na vstupe. Pre túto úlohu viete tiež napísať algoritmus (a to aj so zažmúrenými očami!), ktorý vás dovedie k riešeniu. Napríklad v programovacom jazyku Python môžeme vypočítať súčet prvkov poľa [1, 9, 5, 4] tak, že najskôr deklarujeme súčet ako nulu a potom pridávame po jednom prvku, až kým nedosiahneme koniec poľa:

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

(Vstavaná funkcia sum to za nás rýchlo vypočíta a my ju v skutočnosti používame častejšie.)

Teraz si vyskúšajte ďalšiu hlavolam, v ktorom musíte zistiť, ako sú vstupy a výstupy prepojené. Vstupy sú teraz fotografie zvierat a výstupy sú čísla 0 alebo 1.

Vstup	Výstup
	1
	0
	0

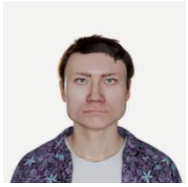
Vstup	Výstup
	1
	0

Určite máte veľa nápadov. A je veľmi pravdepodobné, že všetky sú správne! Keďže však jednotky sú vedľa fotografií mačiek a nuly vedľa ostatných fotografií zvierat, chceli sme, aby ste dospeli k záveru, že ide o úlohu, v ktorej musíte rozpoznať, či je na fotografii mačka, alebo nie. Ak je na vstupe fotografia mačky, na výstupe je 1, a naopak, ak na vstupe nie je fotografia mačky, na výstupe je 0. O niečo neskôr uvidíte, že táto úloha sa nazýva binárna klasifikačná úloha a je veľmi bežná v oblasti strojového učenia.

Je možné vytvoriť algoritmus na riešenie tohto problému? Určite súhlasíte, že je pre nás dôležité vedieť rozlíšiť, čo je na fotografiách, pretože tak ich môžeme ľahšie vyhľadávať a analyzovať. Môžete sa pokúsiť vytvoriť zoznam desiatok pravidiel, podľa ktorých určíte, či je na obrázku mačka alebo nie. Nezabudnite zohľadniť pozadie, osvetlenie, uhol pohľadu a skutočnosť, že existuje viac ako 50 rôznych druhov mačiek.

V skutočnosti existuje veľa podobných problémov, pri ktorých aj napriek veľkej snahe (určíme 1000 pravidiel) nie sme schopní napísať algoritmy, ktoré by ich presne riešili. Ďalšími príkladmi sú preklad z jedného jazyka do druhého a označovanie tvárí na fotografiách. Určite súhlasíte, že aj tieto úlohy sú pre nás dôležité, pretože nám umožňujú porozumieť obsahu napísanému v jazyku, ktorým nehovoríme, alebo zlepšiť bezpečnosť. Preto takéto problémy opisujeme **pomocou dátových súborov,** párov vstupov a očakávaných výstupov, a riešime ich pomocou techník programovania založených na dátach. V prípade úlohy rozpoznávania mačiek (a akýchkoľvek iných objektov) je možné vhodný súbor údajov usporiadať podobne ako náš, s obrázkami na vstupoch a hodnotami 0 alebo 1 na výstupe. V prípade úlohy prekladu to môžu byť páry viet v oboch jazykoch, zatiaľ čo v prípade úlohy označovania tvárí to môžu byť páry obrázkov, jeden bez označených tvárí ako vstup a jeden s označenými tvárami ako výstup. Tu je príklad.

Vstup



Výstup

Databázy, ktoré môžeme použiť na popis problémov, je možné vytvoriť prostredníctvom každodenných činností. Napríklad po vyšetrení pacienta v elektronickej zdravotnej dokumentácii lekár zaznamená informácie o pacientovi, ako je vek, pohlavie, symptómy, alergie na lieky (všetky tieto hodnoty sú vstupné údaje) a kód zodpovedajúci jeho diagnóze (výstupné údaje). Podobne na letiskách sú pre každý let známe informácie, ako

je čas odletu, letecká spoločnosť, typ lietadla atď. (všetky tieto hodnoty sú vstupné údaje) a informácie o tom, či bol let meškaný alebo nie (výstup).

Databázy údajov môžu byť vytvorené aj špeciálne na riešenie konkrétnej úlohy. Napríklad databáza údajov, ktorú sme použili na identifikáciu mačiek, mohla byť vytvorená tímom dobrovoľníkov, ktorí si prezreli naše obrázky a postupovali podľa našich pokynov, napríklad ak je na obrázku mačka, napíšte 1, inak napíšte 0, do stĺpca výstupu zadajte 1 alebo 0. V prípade dátových súborov v danej oblasti, napríklad pri rozpoznávaní zmien na röntgenových snímkach, by bolo potrebné najať lekárskych odborníkov, ktorí majú vhodné zručnosti a vedomosti na rozhodovanie. O niečo neskôr sa dozviete viac o tom, ako sa vytvárajú dátové súbory.

Pravdepodobne sa pýtate: ale ako sa naučíme súvislosť medzi vstupom a výstupom v dátovom súbore? Rovnako ako existuje oblasť, ktorá sa zaoberá vývojom klasických programovacích algoritmov a analýzou ich vlastností, existuje aj oblasť, ktorá sa zaoberá vývojom algoritmov založených na dátach a skúmaním ich vlastností. Nazýva sa **strojové učenie**. *Strojové učenie* je jadrom všetkých moderných oblastí umelej inteligencie, pretože úzko súvisí s dátami a spôsobmi získavania poznatkov z dát. V ďalšej lekcii strojové učenie odpovie na otázku, ktorá vás zaujíma.

Je dôležité zdôrazniť, že existujú aj iné oblasti, ktoré sa zaoberajú údajmi. Medzi nimi je najstaršou určite **štatistika**, odvetvie matematiky, ktoré sa zaoberá zbieraním údajov, ich popisom a analýzou, ako aj vyvodzovaním záverov z údajov. Štatistické techniky sú jadrom mnohých algoritmov strojového učenia. **Data Science** je disciplína, ktorá vznikla v dôsledku neschopnosti jednotlivých disciplín odpovedať na mnohé zaujímavé otázky. Napríklad každá spoločnosť čelí otázke, ako zlepšiť svoje služby. Na tento účel môže spoločnosť analyzovať komentáre používateľov na sociálnych médiách alebo predajných stránkach. Na spracovanie komentárov je potrebné ich zhromaždiť na jednom mieste a uložiť do databázy, potom ich usporiadať, napríklad oddelene pozitívne a negatívne komentáre, a potom analyzovať každú z týchto skupín podrobnejšie, aby sa určilo, čo používatelia hodnotia ako negatívne alebo pozitívne, napríklad konkrétny model produktu alebo niektorú funkciu. Tieto informácie by mali byť zdieľané s vedením spoločnosti, aby mohlo rozhodnúť, čo ďalej. Odpoveď na pôvodnú otázku je, ako si všimnete, dlhá a vyžaduje znalosti a prácu s nástrojmi na sťahovanie obsahu z webu (tzv. scrapers), prácu s databázami, spracovanie prirodzeného jazyka, ako aj techniky zobrazenia údajov, ktoré by boli pre odborníkov v danej oblasti najinformatívnejšie. V tomto a iných príkladoch aplikácie vedy o údajoch je strojové učenie neoddeliteľnou súčasťou cesty k poznaniu.

Než pôjdeme ďalej, zhrňme si, ako vyzerá riešenie problémov s klasickým programovaním a programovaním založeným na údajoch.

Keď riešime problém pomocou klasických programovacích techník, napríklad hľadanie najväčšieho prvku v sekvencii čísel, najprv premýšľame o probléme a priestorových a časových obmedzeniach, ktoré máme, potom navrhneme algoritmus na jeho riešenie (napríklad zlučovanie triedenia) a potom ho naprogramujeme v programovacom jazyku a uložíme kód. Skontrolujeme správnosť implementácie na niekoľkých náhodných záznamoch, kým sa nepresvedčíme, že všetko funguje presne tak, ako očakávame. Keď potrebujeme zoradiť nový reťazec, môžeme použiť program, ktorý sme napísali, spustiť ho a získať vhodné riešenie.

Keď sa spoliehame na programovanie založené na údajoch, spočiatku máme k dispozícii len údaje, napríklad tisíce párov vstupov a výstupov. Opäť je rozumné najprv premyslieť problém. Teraz to robíme tak, že sa zoznámime s dátovým súborom. To sa vykonáva pomocou techník exploratívnej analýzy, o ktorých budeme hovoriť neskôr v kurze a ktoré nám môžu dať predstavu o tom, akú formu riešenia hľadať. Potom namiesto vymýšľania algoritmu na riešenie problému **navrhneme algoritmus, ktorý sa naučí, ako problém riešiť.** To by znamenalo, že ak sa potrebujeme naučiť súvislosť medzi vstupom a výstupom a zmeniť súbor údajov, tento algoritmus môže opäť nájsť najlepšiu súvislosť medzi nimi. Súvislosť, ktorá existuje medzi vstupom a

výstupom, závisí od údajov (nie je statická), a preto sa ju musíme naučiť a nehovoríme to priamo. Keď navrhujeme a programujeme takýto algoritmus, musíme použiť údaje, aby sme zistili, ako dobre funguje. Ak nie sme spokojní s výsledkami, musíme urobiť krok späť a opraviť algoritmus alebo sa vrátiť na samý začiatok a skontrolovať, či v údajoch nie je niečo iné, čo by mohlo byť dôležité pre riešenie problému. Na rozdiel od klasického programovania je táto iteratívnosť veľmi prítomná v programovaní založenom na údajoch.

Táto vložená otázka je nesprávne nakonfigurovaná.

2.3 Základné pojmy strojového učenia

Pojmy, ktoré predstavíme v tejto lekcii, sú základné pojmy strojového učenia. Pomôžu vám sledovať témy, ktoré sa rozoberajú nižšie, a dozviete sa o nich viac.

V kontexte strojového učenia sa súbor údajov, ktorý máme k dispozícii, nazýva **dátový súbor**. Môžu to byť tabuľkové záznamy, podobné tým, s ktorými sa stretávame v databázach alebo súboroch *Excel*, ale aj skupina satelitných snímok alebo zvukových klipov. Jedna konkrétna časť dátového súboru sa nazýva **inštancia**. Príkladom inštancie je jeden konkrétny riadok v tabuľke protokolov alebo jedna konkrétna satelitná snímka. Počet inštancií v dátovom súbore môže ovplyvniť výber učiaceho sa algoritmu, pretože niektoré algoritmy vyžadujú viac údajov ako iné.

V prípadoch existujú **atribúty**, vlastnosti, ktoré používame na popis údajov. Ak si predstavíme, že ide o tabuľkový záznam výskytu zemetrasení, ako atribúty sa môžu objaviť dátum a čas výskytu, zemepisná šírka, zemepisná dĺžka, sila zemetrasenia, úroveň zničeniu a ďalšie dôležité údaje. Atribúty sa rovnako nazývajú **vlastnosti**. O niečo neskôr sa dozviete, aké druhy atribútov existujú a na čo si musíme dávať pozor. Atribúty, na základe ktorých sa musíme naučiť riešiť úlohu, sa nazývajú **vstupné premenné (**vstupy) a tie, ktoré je potrebné naučiť, **výstupné premenné.** Dátum a čas zemetrasenia, jeho zemepisné súradnice a jeho sila môžu byť vstupnými premennými v úlohe určenia rozsahu ničivých účinkov zemetrasenia. Rozsah ničivých účinkov zemetrasenia je tiež prítomný ako atribút v dátovom súbore, takže by to bola výstupná premenná. Niekedy použijeme menej formálne pojmy, ako sú **vstup** a **výstup.** Je dôležité poznamenať, že to je úloha, ktorá určuje, čo budú vstupné premenné a čo budú výstupné premenné.

Aké by mohli byť atribúty satelitného snímku?

***Odpoveď:** Určite súhlasíte, že v prípade satelitných snímok môžeme zaviesť atribúty, ako je poloha, dátum a čas vytvorenia. Môžeme tiež zaviesť atribúty, ktoré opisujú satelit, ktorý snímku zaznamenal. Žiaden z týchto atribútov však priamo neopisuje obsah satelitnej snímky. Zamyslite sa nad touto témou, kým sa nedostaneme k lekcii, ktorá sa jej venuje.*

Povedali sme, že cieľom algoritmov strojového učenia je určiť mapovanie daných vstupov na dané výstupy. Teraz môžeme byť presnejší a povedať, že cieľom strojového učenia je určiť mapovanie daných vstupných premenných na dané výstupné premenné. Tieto mapovania sa nazývajú modely.

Koncept, ktorý spájame s mapovaním, je funkcia. V matematike ste počuli veľa o funkciách, ako je mapovanie vstupov na výstupy. Napríklad funkcia jednej premennej$y = 2x + 4$ mapuje vstup$x = 5$ na hodnotu$y = 14$, zatiaľ čo funkcia viacerých premenných$y = 2x_1 - 3x_2 + x_3 + 5$ mapuje vstup$(x_1, x_2, x_3) = (1, -1, 3)$ na hodnotu$y = 13$. Premenné, ktoré sa objavujú vo funkciách, súvisia s hodnotami atribútu. Tak x v prvej funkcii môže predstavovať plochu nehnuteľnosti v štvorcových stopách, zatiaľ čox_1, x_2, x_3 v druhej funkcii môže predstavovať hodnoty atribútov, ako sú zemepisná šírka, zemepisná dĺžka a sila zemetrasenia. V hodine matematiky ste sa dozvedeli, že existujú rôzne triedy funkcií (lineárne, polynomiálne, trigonometrické, exponenciálne,

logaritmické) a že každá z nich sa vyznačuje niektorými špeciálnymi vlastnosťami, ako je spojitosť, monotónnosť alebo konvexnosť. Všetky tieto vedomosti sú vítané pri hľadaní správneho modelu.

Zložitosť funkcie je niečo, čo formálne nepredstavíme. Pochopíte, že niektoré funkcie sú jednoduchšie ako iné. Jednoduché funkcie sú výhodnejšie pre prácu a ľahšie pochopiteľné, ale nedávajú nám veľa slobody pri popisovaní niektorých nezvyčajnejších vzťahov medzi samotnými atribútmi a výstupmi. Na druhej strane, zložité funkcie sú zložité z určitého dôvodu, takže môže byť pre nás ťažké sledovať niektoré z ich matematických správaní, ktoré môžu ovplyvniť učenie. Snažíme sa nájsť rovnováhu medzi zložitosťou a tým, čo vieme o údajoch a čo sa chceme naučiť.

V modeloch, ako sme videli v úvodnom príklade oceňovania nehnuteľností, sa môžu objaviť **parametre** ako k a n. Takéto modely sa nazývajú **parametrické modely** a úloha určenia správneho modelu sa redukuje na úlohu určenia najlepších hodnôt parametrov. V lineárnom modeli sa v úlohe oceňovania nehnuteľností objavili len dva parametre, zatiaľ čo moderné modely, ktoré sú založené na neurónových sieťach, majú milióny alebo miliardy parametrov. Uvidíme, že existujú aj mierne odlišné **neparametrické modely,** ktorých formy sú vyjadrené odlišne.

Proces hľadania modelu sa nazýva **trénovanie modelu**. Ak sú v modeli neznáme parametre, musíme počas trénovania určiť ich hodnoty. To je náš cieľ.

V dátovom súbore používanom na trénovanie modelov sa môžu nachádzať aj nepresné alebo protichodné hodnoty. Preto modely nikdy nie sú absolútne správne. To nás privádza k ďalšiemu dôležitému pojmu v teórii strojového učenia: **funkcii straty**. Funkcia chyby nám hovorí, ako veľmi je model nesprávny. Jeho hodnoty aktívne využívame počas trénovania modelu a snažíme sa o také konfigurácie modelu, ktoré nás vedú k najnižšej hodnote funkcie chyby. V prípade parametrických modelov, ako to bolo v úvodnom príklade s nehnuteľnosťami, je cieľom určiť tie hodnoty parametrov, pre ktoré je hodnota funkcie chyby najnižšia.

Keď trénujeme model strojového učenia, musíme posúdiť, ako dobrý je v skutočnosti pre praktické použitie. Na to nám **slúžia** takzvané **meradlá kvality** – každé z nich je prispôsobené konkrétnej úlohe učenia a oblasti, v ktorej sa model bude používať. Je dôležité zdôrazniť, že funkcia chyby a meradlá kvality sa vo všeobecnosti líšia. Obe majú za cieľ poskytnúť nám informácie o tom, aký dobrý je model. Funkcia chyby to robí počas trénovania modelu, zatiaľ čo merania kvality to robia po trénovaní modelu. Funkcia chyby je úzko spätá s modelom, zatiaľ čo merania kvality sú navrhnuté tak, aby im rozumeli ako používatelia, tak aj odborníci v danej oblasti. Ak sa nezískajú správne hodnoty kvality, model je potrebné opraviť. Nižšie si povieme, čo to znamená a ako toho dosiahnuť. Celý proces testovania kvality modelu a výpočtu meradiel kvality sa nazýva **testovanie modelu.**

Hodnoty vypočítané a generované trénovaným modelom sa zvyčajne nazývajú **predikcie**. Cena novej nehnuteľnosti alebo odhad škôd spôsobených zemetrasením sú príkladmi predikcií modelu. Preto hovoríme o predikciách vo svete umelej inteligencie. Je vám jasné, že tieto predikcie nie sú v žiadnom prípade náhodné, ale veľmi dobre podložené a založené na údajoch. Aplikácia samotného modelu sa tiež nazýva **inferencia**.

Všetky zdôraznené pojmy sú dôležitými konceptmi strojového učenia a vždy sa vyskytujú v literatúre o strojovom učení a jeho aplikáciách. Preto je dôležité, aby ste im rozumeli a vedeli, akú úlohu zohrávajú vo vývoji modelu.

2.4 Proces strojového učenia

Proces strojového učenia je systematický prístup k vývoju algoritmov, ktoré umožňujú počítačom učiť sa z údajov a robiť predikcie alebo rozhodnutia bez toho, aby boli explicitne naprogramované. Tento proces zahŕňa

niekoľko fáz, z ktorých každá je kritická pre budovanie efektívnych modelov strojového učenia. Porozumenie týmto fázam pomáha odborníkom orientovať sa v zložitosti strojového učenia a zvyšuje pravdepodobnosť úspešných výsledkov.

1. Zber údajov

Prvým krokom v procese strojového učenia je zber údajov, ktorý zahŕňa zhromažďovanie relevantných údajov, ktoré sa použijú na trénovanie modelu. Kvalita a množstvo zhromaždených údajov priamo ovplyvňujú výkon modelu. Údaje môžu pochádzať z rôznych zdrojov, vrátane:

- **Verejné dátové súbory**: Repozitáre ako Kaggle, UCI Machine Learning Repository a vládne databázy ponúkajú vopred zhromaždené dátové súbory pre rôzne aplikácie.
- **Web Scraping**: Automatizované nástroje môžu extrahovať údaje z webových stránok, aby zhromaždili informácie, ktoré nie sú ľahko dostupné v štruktúrovaných formátoch.
- **Prieskumy a experimenty**: Zber údajov na mieru prostredníctvom prieskumov alebo kontrolovaných experimentov môže priniesť špecifické súbory údajov prispôsobené konkrétnemu problému.

Výsledkom tohto kroku je koherentná dátová sada, ktorá reprezentuje problémovú oblasť a bude slúžiť ako základ pre ďalšie kroky v procese strojového učenia.

2. Príprava údajov

Po zozbieraní údajov je potrebné ich pripraviť na analýzu. Táto fáza prípravy zahŕňa niekoľko dôležitých úloh:

- **Čistenie údajov**: Surové údaje často obsahujú chyby, duplikáty alebo chýbajúce hodnoty. Čistenie údajov je nevyhnutné na zabezpečenie presnosti a spoľahlivosti. Techniky zahŕňajú odstraňovanie duplikátov, opravu nekonzistencií a doplňovanie alebo odstraňovanie chýbajúcich hodnôt.
- **Transformácia údajov**: Tento krok môže zahŕňať normalizáciu alebo štandardizáciu údajov, aby sa zabezpečilo, že všetky vlastnosti rovnako prispievajú k výkonu modelu. Typy údajov môžu tiež vyžadovať konverziu (napr. konverziu kategorických premenných na numerické formáty).
- **Rozdelenie údajov**: Dataset sa zvyčajne delí na trénovacie a testovacie súbory. Bežný pomer rozdelenia je 80 % pre trénovanie a 20 % pre testovanie, čím sa zabezpečí, že model bude možné vyhodnotiť na základe neviditeľných údajov.

Príprava údajov je často jedným z najčasovo náročnejších aspektov procesu strojového učenia, ale je kľúčová pre vývoj efektívneho modelu.

3. Výber a úprava vlastností

Výber a inžinierstvo vlastností sú kľúčové kroky, ktoré určujú, ktoré atribúty údajov sa použijú pri trénovaní modelu:

- **Výber vlastností**: Zahŕňa identifikáciu najrelevantnejších vlastností, ktoré prispievajú k predikcii cieľovej premennej. Techniky ako korelačná analýza, rekurzívna eliminácia vlastností alebo použitie algoritmov ako LASSO môžu pomôcť pri výbere dôležitých vlastností a vyradení irelevantných.

- **Inžinierstvo vlastností**: V tejto fáze môžu byť z existujúcich vlastností vytvorené nové vlastnosti s cieľom zlepšiť výkon modelu. Môžu to byť polynomiálne vlastnosti, interakčné termíny alebo agregované vlastnosti založené na znalostiach danej oblasti.

Efektívny výber a inžinierstvo vlastností môže výrazne zvýšiť prediktívnu silu modelu tým, že zabezpečí, aby sa zameriaval na najinformatívnejšie aspekty údajov.

4. Výber modelu

S pripravenými údajmi a vybranými vlastnosťami je ďalším krokom výber vhodného algoritmu strojového učenia. Výber algoritmu závisí od viacerých faktorov:

- **Typ problému**: Rôzne algoritmy sú vhodné pre rôzne úlohy – klasifikáciu (napr. logistická regresia, rozhodovacie stromy), regresiu (napr. lineárna regresia), zhlukovanie (napr. k-means) alebo posilňovacie učenie.

- **Charakteristiky údajov**: Povaha údajov (napr. veľkosť, dimenzionalita) ovplyvňuje výber algoritmu. Napríklad modely hlbokého učenia môžu byť vhodnejšie pre veľké súbory údajov so zložitými vzormi.

Výber vhodného modelu vytvára základ pre efektívne trénovanie a hodnotenie.

5. Tréning modelu

Trénovanie modelu zahŕňa vkladanie pripravených dát do vybraného algoritmu, aby sa mohol naučiť vzory v rámci dátového súboru:

- **Proces trénovania**: Počas trénovania algoritmus upravuje svoje interné parametre na základe vstupných charakteristík a zodpovedajúcich cieľových výstupov. Tento iteratívny proces pokračuje, kým nie je splnené kritérium zastavenia (napr. stanovený počet epoch alebo konvergencia).

- **Metriky hodnotenia**: Pred začatím trénovania je nevyhnutné definovať metriky úspešnosti. Bežné metriky zahŕňajú presnosť pre klasifikačné úlohy, strednú kvadratickú chybu pre regresné úlohy a F1-skóre pre nevyvážené dátové súbory.

Cieľom trénovania je vyvinúť model, ktorý sa dobre generalizuje na nové dáta, a nie len jednoducho zapamätať si trénovací súbor.

6. Hodnotenie modelu

Po tréningu je kľúčové vyhodnotiť výkon modelu na neviditeľných testovacích údajoch:

- **Testovanie**: Model sa hodnotí pomocou testovacej dátovej sady, ktorá nebola použitá pri tréningu. Toto hodnotenie poskytuje informácie o tom, ako dobre sa model generalizuje na nové prípady.

- **Metriky výkonu**: V závislosti od typu problému sa môžu použiť rôzne metriky:

    - Pre klasifikačné úlohy: presnosť, precíznosť, spomienka, F1-skóre.

    - Pre regresné úlohy: R-kvadrát, priemerná absolútna chyba (MAE), priemerná kvadratická chyba (MSE).

Toto hodnotenie pomáha identifikovať oblasti, v ktorých model vyniká alebo potrebuje zlepšenie.

7. Ladenie hyperparametrov

Ladenie hyperparametrov zahŕňa optimalizáciu parametrov, ktoré riadia proces trénovania, ale nie sú priamo naučené z dát:

- **Grid Search a Random Search**: Tieto techniky systematicky skúmajú kombinácie hyperparametrov, aby našli optimálne nastavenia, ktoré zlepšujú výkon modelu.
- **Krížová validácia**: Implementácia krížovej validácie počas ladenia hyperparametrov pomáha zabezpečiť, aby zlepšenia výkonu boli konzistentné v rôznych podmnožinách údajov.

Jemné ladenie hyperparametrov môže viesť k významnému zlepšeniu presnosti a robustnosti modelu.

8. Nasadenie

Po vytrénovaní a vyhodnotení uspokojivého modelu ho možno nasadiť do produkcie:

- **Integrácia**: Konečný model je potrebné integrovať do existujúcich systémov, kde bude poskytovať predpovede alebo informácie na základe údajov v reálnom čase.
- **Monitorovanie**: Neustále monitorovanie po nasadení je nevyhnutné na zabezpečenie toho, aby model zachoval svoj výkon v priebehu času, keď budú k dispozícii nové údaje.

Nasadenie znamená prechod od vývoja k praktickému uplatneniu, s dôrazom na reálnu využiteľnosť.

Záver

Proces strojového učenia zahŕňa sériu štruktúrovaných krokov, ktoré vedú odborníkov od počiatočného zberu údajov až po nasadenie prediktívnych modelov. Každá fáza – zber údajov, príprava, výber funkcií, výber a trénovanie modelu, hodnotenie, ladenie hyperparametrov a nasadenie – hrá dôležitú úlohu pri vývoji efektívnych riešení strojového učenia. Dôkladným pochopením tohto procesu môžu odborníci zlepšiť svoju schopnosť vytvárať robustné modely, ktoré poskytujú cenné informácie v rôznych aplikáciách v dnešnom svete založenom na údajoch.

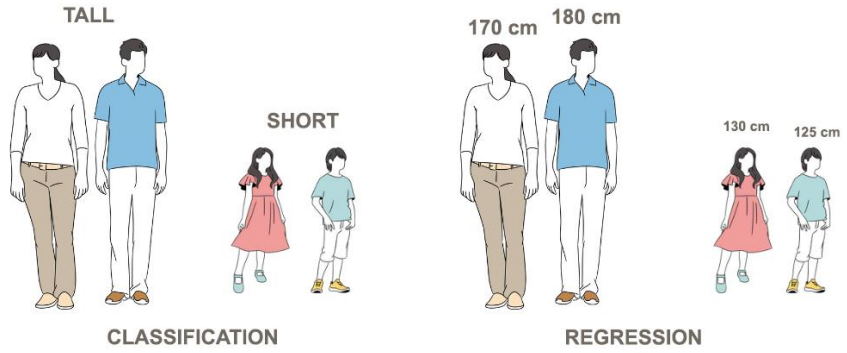
2.5 Typy strojového učenia

V tejto lekcii predstavíme základné typy strojového učenia: strojové učenie s dohľadom, strojové učenie bez dohľadu a posilňovacie učenie. Všetky sa skrývajú za úlohami, ako je predpovedanie zrážok, odporúčanie filmov na pozretie alebo hranie hier. Každá z týchto oblastí bude podrobnejšie rozoberaná neskôr v kurze. Budeme tiež hovoriť o niektorých súčasných oblastiach výskumu, ako je učenie prostredníctvom prenosu vedomostí.

Supervízované strojové učenie

Väčšina príkladov, ktoré sme doteraz videli, sú vlastne príklady **supervidovaného strojového učenia**. Supervidované strojové učenie zahŕňa algoritmy, ktoré dokonale zapadajú do toho, o čom sme doteraz hovorili, a pomáhajú nám naučiť sa, ako priradiť jednu sadu hodnôt k druhej. Preto je potrebné, aby sme v dátovej sade, na ktorú sa uplatňujú, okrem hodnôt atribútu poznali aj hodnoty cieľovej premennej.

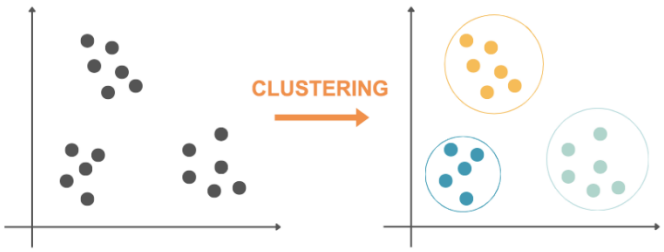
Dve hlavné úlohy strojového učenia pod dohľadom sú **regresia** a **klasifikácia.** V úlohách regresie aj klasifikácie sa chceme naučiť predpovedať hodnoty, ale v úlohách regresie môžu byť hodnoty ľubovoľné a v prípade klasifikácie môžu pochádzať z vopred definovanej konečnej množiny hodnôt. Úlohy regresie sú teda vhodné na predpovedanie hodnoty teploty, ceny produktu (úloha určenia ceny nehnuteľnosti, s ktorou sme sa stretli v úvodnej časti, je príkladom úlohy regresie), škôd spôsobených zemetrasením a podobne. Na druhej strane, určenie, či je pošta nežiaduca alebo žiaduca, alebo určenie žánru filmu, sú klasifikačné úlohy, pretože množina hodnôt, ktorú máme na druhej strane, je konečná – pošta môže byť buď žiaduca, alebo nežiaduca (dve hodnoty), zatiaľ čo žáner môže byť napríklad komédia, dráma, akčný film alebo thriller (štyri hodnoty). O niečo neskôr predstavíme presnejšiu definíciu každej z týchto úloh.



Klasifikácia a regresia. Určenie, či je niekto vysoký alebo nízky, je úlohou klasifikácie. Určenie presnej výšky je úlohou regresie.

Nekontrolované strojové učenie

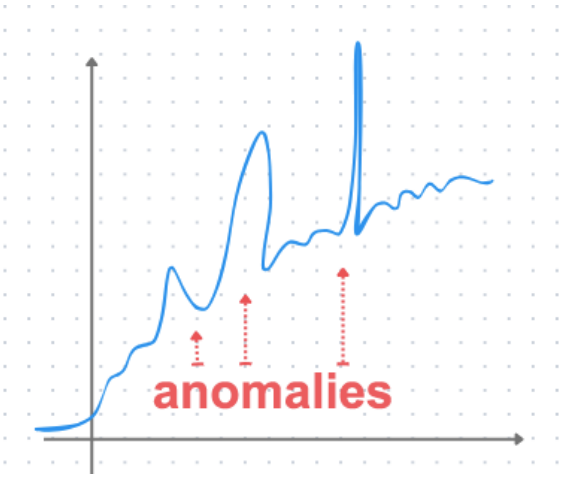
Nekontrolované strojové učenie používame v úlohách, ktoré vyžadujú preskúmanie štruktúry súboru údajov. Ak napríklad analyzujeme nákupy spotrebiteľov v jednom obchode, môže byť zaujímavé všimnúť si produkty, ktoré sa často kupujú spolu, aby sme ich mohli v obchode lepšie rozmiestniť, zlepšiť ponuku, ale aj zisky. Rovnakým spôsobom je možné analyzovať a zoskupiť komentáre používateľov a získať tak prehľad o službách alebo funkciách, o ktorých používatelia hovoria. Úlohy tohto typu, pri ktorých chceme vidieť skupiny medzi údajmi, sa nazývajú **zhlukovanie.** Neskôr v kurze sa dozviete o algoritme k-mean, najznámejšom algoritme zhlukovania.



Zhlukovanie

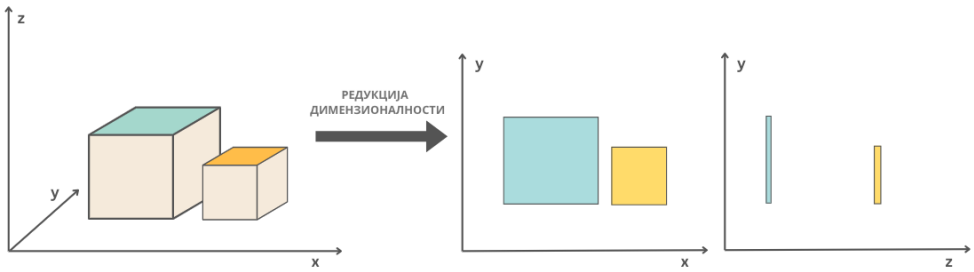
Zisťovanie prípadov údajov, ktoré sa nejakým spôsobom líšia od ostatných, tiež patrí medzi úlohy neriadeného strojového učenia. Zisťovanie atypických meraní senzorov v továrni môže byť signálom na spustenie dodatočných bezpečnostných postupov. Podobne, zisťovanie atypických bankových transakcií, napríklad zo

vzdialenej lokality alebo v nezvyčajnej výške, môže byť náznakom podvodu. Táto úloha neřízeného strojového učenia sa nazýva **detekcia anomálií**.



Detekcia anomálií

Nekontrolované strojové učenie sa zaoberá aj úlohami **znižovania dimenzionality.** Často je na účely grafického znázornenia údajov potrebné prejsť z väčšieho počtu atribútov na menší počet atribútov, napríklad dva alebo tri. Je zrejmé, že počas tejto transformácie sa stratia niektoré informácie z pôvodného súboru údajov, ale na druhej strane sa získa schopnosť zobrazovať údaje a možno aj lepší prehľad o niektorých zákonitostiach. Menšia dimenzionalita údajov (menej atribútov) je žiaduca, pretože umožňuje rýchlejšie vykonávanie algoritmov a menšiu zložitosť pamäte, čo môže byť obzvlášť dôležité, ak máme k dispozícii obmedzené zdroje na prácu. Niektoré z najčastejšie používaných algoritmov na redukciu dimenzionality sú *analýza hlavných komponentov (PCA)* a t-SNE.



Význam redukcie dimenzionality: Dva kvádre a ich projekcie z trojrozmerného do dvojrozmerného priestoru

Zaujímavé je, že pri úlohách strojového učenia bez dohľadu nie je potrebné poznať hodnoty cieľovej premennej. Zhlukovanie, detekcia anomálií a redukcia dimenzionality sa vykonávajú iba na základe hodnôt atribútov.

Posilňovacie učenie

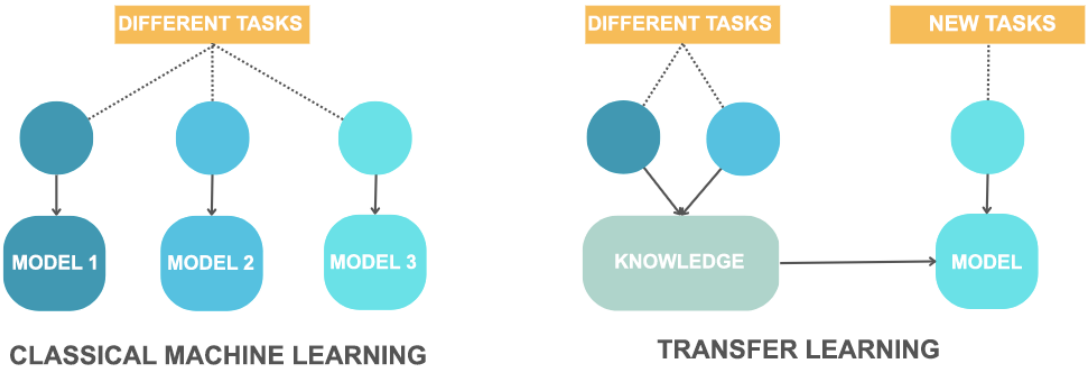
Určite ste už mnohokrát videli, ako sa cvičí pes. Keď dostane úlohu, napríklad priniesť loptu z druhého konca dvora, odmena v podobe sušienky, keď ju prinesie, motivuje psa, aby túto úlohu nabudúce vykonal ešte úspešnejšie a s väčšou radosťou. Táto myšlienka je tiež základom učenia sa posilňovaním. **Posilňovacie učenie** je oblasť strojového učenia, ktorá sa používa pri úlohách, ako je hranie hier alebo autonómne riadenie. Vyznačuje sa existenciou prostredia, ktoré má svoje vlastné stavy, agenta, ktorý môže vykonávať určitý súbor akcií, a konceptu odmeny. Cieľom je, aby agent v danom prostredí, ktorého stavy sa menia, zvolil (naučil sa) postupnosť akcií, ktorá mu umožní získať najväčšiu odmenu. V kontexte úvodného príkladu je dvor prostredím.

Jeho stavmi môžu byť lopta na konci dvora alebo susedova mačka na strome. Pes je agent a súbor akcií, ktoré môže vykonať, je behať, sedieť, ísť spať. Odmenou môže byť niekoľko sušienok alebo nič. Ak pes zvolí správny sled akcií (behať, nájsť a vrátiť) na zmenu prostredia, napríklad na objav loptu, bude môcť získať najväčšiu odmenu.

Viac o tomto type učenia sa dozviete na konci kurzu.

Nové smery učenia

Keď sa potrebujeme naučiť novú úlohu, napríklad jazdu na kolobežke, nezačíname od nuly. Všetky vedomosti a zručnosti, ktoré sme nadobudli pri iných úlohách, napríklad pri hraní basketbalu, jazde na bicykli, a dokonca aj vytrvalosť a trpezlivosť pri úlohách, ktoré nepatrili medzi naše obľúbené, ako napríklad upratovanie pivnice, nám pomáhajú lepšie sa to naučiť. Táto myšlienka je základom **transferového učenia**. Preto často počujete ľudí hovoriť o modeloch, ktoré boli použité ako základ pre vývoj iného modelu. Takéto modely sú najskôr trénované na niektorých všeobecných dátových súboroch a úlohách a potom preškolené, t. j. môžu byť použité aj na riešenie veľmi špecifických úloh. Napríklad jazykový model *GPT* bol použitý ako základ pre vývoj *modelu ChatGPT,* ktorý predtým dosahoval dobré výsledky v úlohách generovania súhrnov, skrátených verzií textu a odpovedaní na otázky.



Myšlienka učenia prostredníctvom prenosu vedomostí

Techniky prenosu vedomostí možno kombinovať so všetkými vyššie uvedenými typmi učenia. Sú pre nás obzvlášť dôležité, keď sú trénovacie dátové súbory pre konkrétnu úlohu nedostatočné alebo keď vyvíjame model pre konkrétnu oblasť.

2.6 Dáta v strojovom učení

V komunite umelej inteligencie často počujete dve porekadlá: „dáta sú nové zlato" a „ak sú vstupné dáta nekvalitné, výstupné dáta budú tiež nekvalitné". Tieto porekadlá nám pripomínajú hodnotu dát pre pochopenie a modelovanie javov a dôležitosť vytvárania vysokokvalitných dátových súborov. Pozrime sa na tieto témy bližšie.

Dnes takmer všetky oblasti činnosti generujú veľké množstvo dát: informácie o videách, ktoré sme sledovali online, produktoch, ktoré sme zakúpili, priateľoch, s ktorými sme sa spojili na sociálnych médiách, ako aj informácie o návštevách u lekára, poveternostných podmienkach v našom meste alebo dopravnej situácii zaznamenanej príslušnými inštitúciami. Všetky tieto dáta možno použiť na lepšie pochopenie prostredia, v ktorom vznikajú.

Rovnako ako v prípade databáz, aj v strojovom učení opisujeme dôležité entity a udalosti, ktorých správanie chceme modelovať pomocou atribútov (tiež nazývaných vlastnosti). Napríklad film možno opísať podľa jeho názvu, žánru, roku uvedenia do kín, produkčnej spoločnosti, rozpočtu, zisku, synopsie, mena režiséra a mien hlavných hercov. Výber správnych atribútov, ktoré treba sledovať a zaznamenávať pri zbere údajov, nie je ľahká úloha, pretože vopred nevieme, ktoré atribúty budú najužitočnejšie pre úlohu, ktorú chceme v budúcnosti riešiť. Napríklad, ak chceme použiť údaje na predpovedanie zisku filmu (regresná úloha), informácie o hercoch a produkčnej spoločnosti môžu byť užitočnejšie, zatiaľ čo na určenie žánru filmu (klasifikačná úloha) môže byť užitočnejší obsah. V zložitejších oblastiach sú tieto voľby spojené s ešte väčšími dilemami a výzvami.

Vzhľadom na potrebu využívať údaje pre širokú škálu aplikácií môžeme zvážiť zber čo najväčšieho počtu atribútových hodnôt. Hoci táto myšlienka je v niektorých situáciách platná, vo všeobecnosti musíme mať na pamäti, že veľké množstvá údajov vyžadujú vhodné úložisko, hardvér na podporu ich spracovania a tím odborníkov s potrebnými zručnosťami a znalosťami na vykonávanie týchto úloh. Preto môžu byť takéto voľby nákladné a vyžadujú si špeciálne plánovanie. Je tiež dôležité poznamenať, že analýza a pochopenie veľkého množstva údajov je náročné a vyžaduje si vhodné technické kompetencie, ako sú techniky vizualizácie údajov. Okrem toho mnoho oblastí, ktoré zahŕňajú súkromné a citlivé údaje, musí dôsledne dodržiavať predpisy a etické usmernenia týkajúce sa zberu údajov, čo kladie ďalšie obmedzenia na výber atribútov a možnosti ukladania. Úloha zberu údajov a vytvárania vysokokvalitných dátových súborov je teda náročná a vyžaduje si starostlivú organizáciu.

V nasledujúcich lekciách uvidíme, že každý atribút je definovaný svojím typom a sadou hodnôt a tieto vlastnosti ovplyvňujú spôsob, akým údaje pripravujeme. Algoritmy strojového učenia sa nakoniec dajú použiť len na numerické hodnoty. Počet atribútov a ich vlastnosti tiež ovplyvňujú výber algoritmu strojového učenia.

Pokročilé algoritmy strojového učenia, ako sú neurónové siete, dokážu samy identifikovať atribúty dôležité pre riešenie úlohy. To nás oslobodzuje od premýšľania o výbere a kombináciách atribútov. To je obzvlášť užitočné pri práci so zložitými údajmi, ako sú obrázky alebo textový obsah, kde definovanie a extrakcia atribútov nie je vždy intuitívna. Tieto algoritmy dokážu pracovať s surovými údajmi.

- Čo považujete za náročné na zbere údajov v oblasti, ktorá vás zaujíma? Môže to byť šport, vedecká disciplína, sociálny fenomén alebo čokoľvek iné.
- Máte nejaké obavy alebo výhrady týkajúce sa zberu a spracovania údajov?
- Čo je pre vás osobne najdôležitejšie v procese zberu údajov?

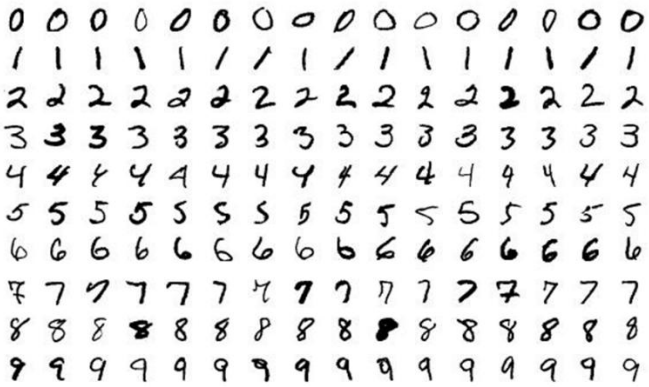
Populárne dátové súbory

Možno vás to prekvapí, ale aj dátové súbory môžu byť populárne! Niektoré z nich sú známe tým, že sa používajú v prvých úlohách strojového učenia, zatiaľ čo iné dosiahli svoju popularitu vďaka vytrvalému úsiliu komunity o ich rozširovanie a dopĺňanie. Keďže rôzne dátové súbory sledujú rôzne oblasti umelej inteligencie, použijeme to ako kritérium na ich zoskupovanie a zobrazovanie. Konkrétne sa zoznámime so súbormi, ktoré obsahujú obrázky, textové údaje, zvukové archívy a videá. Veľké množstvo knižníc používaných v oblasti strojového učenia umožňuje rýchlo a ľahko načítať súbory, o ktorých budeme diskutovať.

Počítačové videnie

MNIST

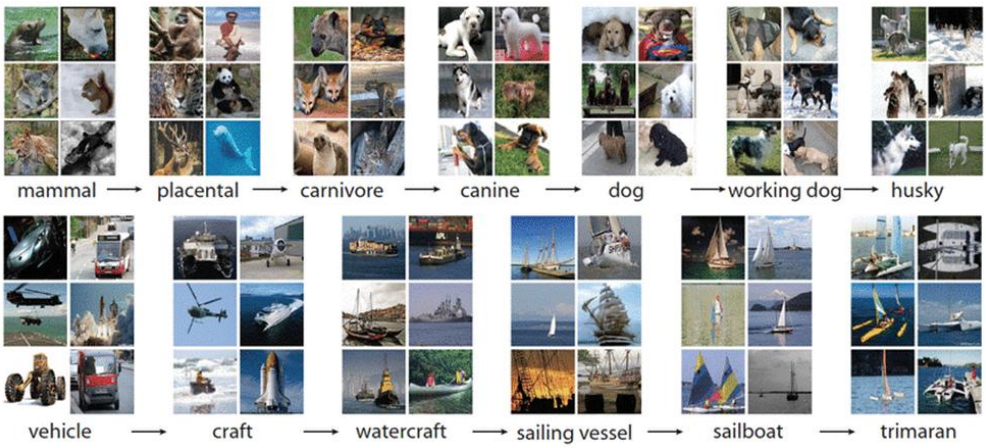
Jedným z najpopulárnejších súborov v oblasti počítačového videnia je určite **MNIST**, súbor obrázkov ručne písaných číslic. Jeho vývoj začal americký Národný inštitút pre štandardy a technológie *(NIST)* už v roku 1998. Všetky obrázky majú rozmery 28x28 pixelov, sú čiernobiele a je ich celkom 70 000: 60 000 obrázkov tvorí trénovaciu sadu a 10 000 obrázkov tvorí testovaciu sadu. Na obrázku môžete vidieť niektoré číslice z tejto dátovej sady.



Niektoré čísla z MNIST

Súbor MNIST sa používa na trénovanie viac triednych klasifikátorov, najčastejšie v kombinácii s konvolučnými neurónovými sieťami, o ktorých sa dozviete viac neskôr v kurze.
Pre každú číslicu súboru MNIST je k dispozícii jedna trieda. Zamyslite sa nad tým, ktoré číslice môžu byť potenciálne problematické na rozlíšenie (napríklad číslice 1 a 7 sa môžu navzájom podobať), a potom sa pokúste nájsť niektoré príklady na webe.

ImageNet
Obrázky v súbore **ImageNet** predstavujú obrázky všeobecných objektov: počítače, okná, lietadlá, sadence, tropické zvieratá a rôzne iné entity. Zaujímavé je, že tieto obrázky sú usporiadané do súvisiacich skupín (tzv. synsets), medzi ktorými platí vzťah rodič-dieťa. Napríklad všetky plachetnice patria do jednej skupiny (jedného synsetu), v hierarchii pod nimi sú skupiny klzákov a trimaranov, zatiaľ čo v hierarchii nad nimi sú skupiny vodných plavidiel, lodí a vozidiel. Na obrázku v spodnom riadku môžete sledovať túto hierarchiu: v spodnej časti sú trimarány a v hornej časti vozidlá. V hornom riadku sú synsety, ktoré sa týkajú psov a niektorých ich kategorizácií.



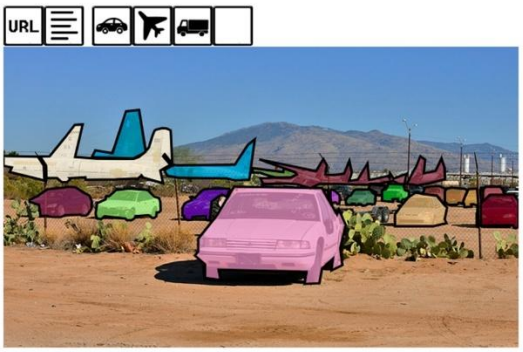
Príklad obrázku ImageNet

Kolekcia v súčasnosti obsahuje približne 14 miliónov obrázkov a viac ako 21 000 synsetov. Používa sa v rôznych úlohách klasifikácie obrázkov a detekcie objektov v obrázkoch.
Oficiálna webová stránka konferencie ImageNet je https://www.image-net.org/index.php . Na jej vývoji aktívne pracujú výskumníci zo Stanfordovej a Princetonskej univerzity.

Skúste zistiť, do ktorej skupiny v súbore ImageNet patrí počítač a ktoré skupiny sa nachádzajú v hierarchii pod ním a nad ním.

COCO

Dataset **COCO** (skratka pre *Common Objects in Context*) sa používa v úlohách detekcie objektov, segmentácie obrázkov a automatického priraďovania názvov k obrázkom. Vytvorila ho spoločnosť Microsoft a v roku 2015 ho zdieľala s komunitou.



Obrázok súboru COCO s označenými rozpoznanými objektmi: lietadlá, nákladné automobily a autá

Súbor si môžete interaktívne prezrieť na oficiálnej webovej stránke: pri každom obrázku je uvedená URL adresa, z ktorej bol obrázok prevzatý, niekoľko názvov spojených s obrázkom a séria ikon zodpovedajúcich rozpoznaným objektom. Dataset obsahuje 330 000 obrázkov a 80 kategórií objektov s viac ako 1,5 miliónmi prípadov. Odkaz na sekciu vyhľadávania na stránke je https://cocodataset.org/#explore.

Spracovanie prirodzeného jazyka

IMDB

Ak radi pozeráte filmy a televízne programy, bude vás zaujímať dataset **IMDB**, ktorý obsahuje recenzie používateľov z populárnej platformy IMDB. Pre každé zobrazenie v tomto datasetu je tiež známe, či je pozitívne alebo negatívne, t. j. či obsahuje predovšetkým niečo chvályhodné a dobré o filme, alebo nejakú kritiku a

námietky. Pokiaľ ide o dataset, ktorý obsahuje textový obsah, je vždy dôležité zdôrazniť, v akom jazyku je napísaný. Dataset IMDB obsahuje recenzie v anglickom jazyku s celkovým počtom 50 000 recenzií, z toho 25 000 pozitívnych a 25 000 negatívnych. Nižšie môžete vidieť pozitívny a negatívny záznam v tomto datasetu.

Review	Sentiment 0-negative 1-positive
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

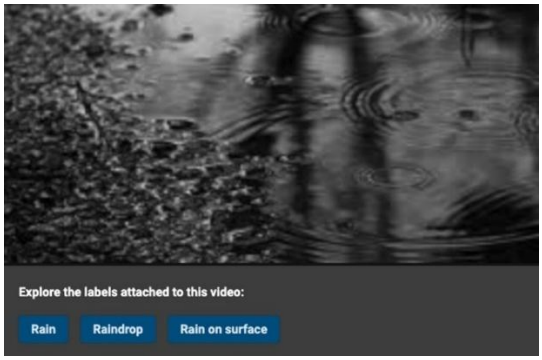
Príklady pozitívnych a negatívnych recenzií zo súboru IMDB

Dataset IMDB sa používa v úlohách analýzy sentimentu – pripomeňme, že ide o úlohy, v ktorých je potrebné rozpoznať emóciu alebo postoj prítomný v texte. Keďže súbor obsahuje iba informácie o tom, či je recenzia pozitívna alebo negatívna, úloha analýzy sentimentu v súbore IMDB sa rieši ako problém binárnej klasifikácie. Vo všeobecnosti môže byť stupnica sentimentu jemnejšia a zahŕňať hodnotenia ako veľmi pozitívne, pozitívne, neutrálne, negatívne alebo veľmi negatívne.

Spracovanie zvuku

AudioSet

AudioSet je súbor údajov, ktorý obsahuje 10-sekundové úryvky videí z YouTube. Každý z týchto úryvkov je spojený s charakteristikami zvukov, ktoré v nich zaznievajú. Súbor bol vytvorený spoločnosťou Google a obsahuje viac ako 2 milióny klipov s celkovou dĺžkou 5,8 tisíc hodín.



Príklad videoklipu s príslušnými zvukovými poznámkami, ktoré obsahuje

Oficiálna webová stránka konferencie poskytuje prehľad príkladov a informácie o organizácii konferencie. Používa sa 632 rôznych kategórií, ako napríklad zvuky hudobných nástrojov, zvuk vetra, zvuk človeka, hluk atď. Môžete navštíviť adresu https://research.google.com/audioset/index.html a vypočuť si ďalšie príklady. Samotná konferencia bola vytvorená s cieľom podporiť vývoj algoritmov rozpoznávania zvuku.

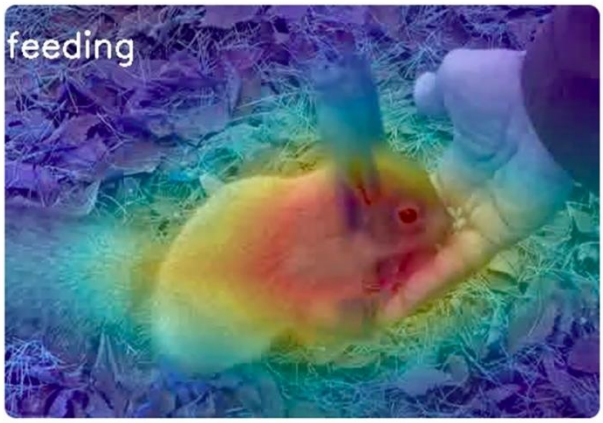
Spracovanie videa

Momenty v čase

Moments in Time je súbor údajov, ktorý sa vyvíja s cieľom pomôcť systémom umelej inteligencie naučiť sa rozpoznávať akcie a udalosti. Tento súbor momentálne obsahuje jeden milión videí s dĺžkou 3 sekundy, v

ktorých sú označené aktivity. Videá obsahujú ľudí, zvieratá, predmety a prírodné javy. Medzi udalosti, ktoré sú zahrnuté, patria napríklad tanec, cvičenie, lezenie na strom, skákanie do vody a spanie.

Zbierku Moments in Time vyvíja tím z Massachusetts Institute of Technology (MIT) a na oficiálnej webovej stránke projektu si môžete pozrieť ďalšie príklady videí a rozpoznaných činností. Odkaz na oficiálnu webovú stránku je http://moments.csail.mit.edu/.

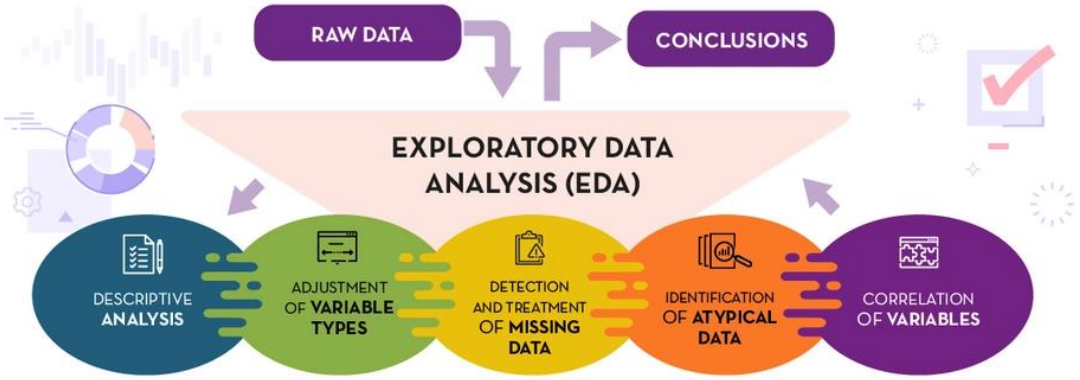


Video, v ktorom je rozpoznaný muž kŕmiaci králika

2.7 Exploratívna analýza dát (EDA)

Stretnutie s novým súborom údajov je ako výlet na nové miesto. Musíte ho starostlivo preskúmať, zistiť, kde sa čo nachádza a aké súvislosti existujú medzi jednotlivými časťami. V tejto časti sa naučíte niekoľko techník, ktoré vám pomôžu v našom dobrodružstve s údajmi.

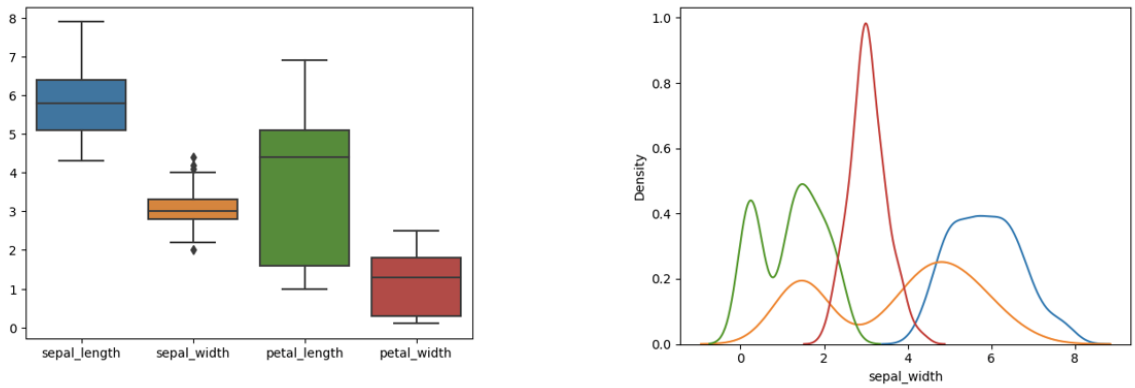
Každú úlohu strojového učenia začíname spoznávaním súboru údajov. Ak používame tabuľkové údaje, zaujíma nás, aké atribúty sa objavujú, aké majú hodnoty a či niektoré z nich môžu súvisieť. Keď pracujeme s inými typmi údajov, napríklad s textom, zvyčajne nás zaujíma, či sú všetky texty napísané v rovnakom jazyku a aká je ich dĺžka. Keďže žiadny súbor údajov nie je dokonalý, pri analýze sa snažíme nájsť potenciálne duplikáty a niektoré atypické záznamy. Všetky tieto úlohy sa nazývajú **exploratívna analýza súboru údajov.** *Exploratívna analýza dát (*EDA). Jej cieľom je pomôcť nám prostredníctvom rôznych úloh lepšie spoznať súbor dát a prijať ďalšie informovanejšie rozhodnutia týkajúce sa prípravy dát. Vzhľadom na dôležitosť dát v ďalších krokoch (spomeňte si na príslovie „garbage in, garbage out" – čo tam vložíte, to tam máte) sa snažíme venovať dostatok času exploratívnej analýze súboru dát a prejsť k ďalšiemu kroku až vtedy, keď sme si istí, že dátam rozumieme.



Úlohy exploratívnej analýzy dát

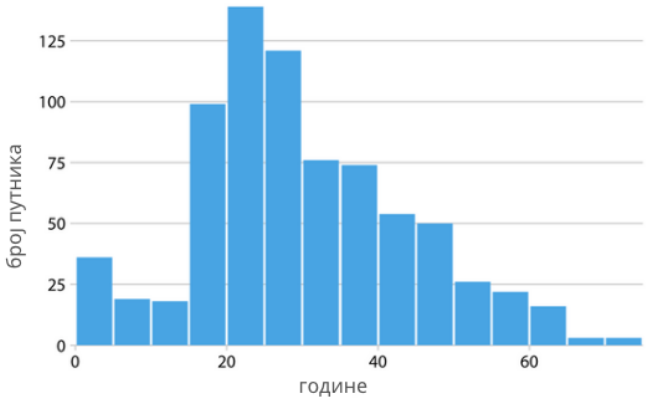
Analýza atribútov

Keďže atribúty sa používajú na opis širokej škály vlastností, ich typy a rozsahy hodnôt sa líšia. Dve veľké skupiny atribútov, s ktorými sa stretávame, sú **numerické** (kvantitatívne) a **kategorické** (kvalitatívne) atribúty. Číselné atribúty majú, ako už názov napovedá, číselné hodnoty. Takými sú napríklad výška hráča, vzdialenosť od letiska, počet domácich zvierat, vonkajšia teplota, počet predaných zmrzlín, koncentrácia glukózy v krvi a mnoho ďalších. Pri týchto atribútoch sa zvyčajne pozeráme na rozsahy hodnôt, najvyššie a najnižšie hodnoty, priemernú hodnotu, medián, ako aj samotné rozdelenie. Všetky tieto **analýzy** nazývame **deskriptívnymi analýzami,** pretože nám pomáhajú opísať množstvo, s ktorým je atribút spojený.



Príklady niektorých deskriptívnych analýz atribútov dátového súboru Iris

Kategórie atribútov sú typom atribútov, ktoré môžu mať konečnú sadu hodnôt. Takými atribútmi sú napríklad farba auta, typ oblečenia, pohlavie pacienta, aktuálne ročné obdobie a iné. Tieto atribúty sú zvyčajne reprezentované reťazcami alebo ekvivalentnými číselnými kódmi. Napríklad mesiac v roku môže byť uvedený ako názov „február" alebo ako číslo „dva" (pretože február je druhý mesiac v roku). Je dôležité poznamenať, že aj keď na reprezentáciu týchto atribútov používame numerické kódy, nemá zmysel počítať hodnoty ako priemerná alebo maximálna, pretože tieto hodnoty nie sú svojou podstatou numerické. V ich prípade zvyčajne analyzujeme, aké hodnoty môžu nadobúdať a ako často sa vyskytujú, a tieto závery zobrazujeme pomocou grafov.



Príklad analýzy atribútu „rok" v sade Titanic

Zjednotenie hodnôt

Pri analýze údajov môžeme zistiť, že hodnoty atribútov nie sú jednotne nastavené. Napríklad názvy farieb môžu byť písané nejednotne, niekedy malými písmenami a niekedy veľkými písmenami, alebo dátumy môžu byť uvedené v rôznych formátoch, napríklad deň-mesiac-rok a rok/mesiac/deň. Aby sme mohli správne vykonať úlohu analýzy, je žiaduce zjednotiť tieto hodnoty, t. j. zredukovať ich na rovnaký spôsob reprezentácie. Zvyčajne existuje spôsob, ktorý je žiadúcejší alebo užitočnejší, ale stáva sa aj to, že voľby sú úplne rovnaké.

Chýbajúce hodnoty

Pri analýze súboru údajov môžeme zistiť, že hodnoty niektorých atribútov chýbajú. Môže to byť spôsobené nepozornosťou pri vkladaní údajov alebo jednoducho nedostupnosťou informácií. Takéto hodnoty v súbore údajov sa nazývajú chýbajúce hodnoty.

name and surname	year	work experience	revenues
Marco	48	22	83
Ivan	67	30	110
Ana	34	10	
Peter		4	
Milan	21		85

Príklad súboru s chýbajúcimi hodnotami

Najjednoduchším krokom, ktorý môžeme urobiť, keď si všimneme chýbajúce hodnoty, je vymazať buď atribúty (stĺpce súboru údajov), alebo inštancie (typy súboru údajov), v ktorých sa vyskytujú. Napríklad, ak nepoznáme hodnotu atribútu pre viac ako 50 % inštancií súboru údajov, má zmysel ho odstrániť. Ak naopak máme len niekoľko inštancií, v ktorých chýba hodnota atribútu, je najlepšie odstrániť inštancie a ponechať atribút. Tieto rozhodnutia však nie sú vždy jednoduché. Môže sa napríklad stať, že v rôznych inštanciách chýbajú rôzne hodnoty atribútov, takže takto odstránime a ignorujeme značný počet inštancií, čo môže byť problematické, ak nemáme veľký súbor údajov. Preto má zmysel zvážiť ďalšie možnosti pri práci s chýbajúcimi hodnotami.

Ak chýbajúci atribút je číselný, napríklad vzdialenosť od letiska alebo výška hráča, môžeme chýbajúce hodnoty nahradiť priemernou hodnotou známych hodnôt. Argumentom pre tento výber je, že použijeme informácie, ktoré už existujú v dátovom súbore, a že veľa nezmeníme na niektorých ďalších vlastnostiach atribútov. Na druhej strane, ak hovoríme o kategorických atribútoch, ako je farba auta alebo krajina výroby, ktoré môžu mať konečný súbor hodnôt, môžeme chýbajúcu hodnotu nahradiť najbežnejšou hodnotou. Ďalšou možnosťou, ktorá je platná pre numerické aj kategorické atribúty, je použitie náhodných hodnôt – chýbajúcu farbu môžeme nahradiť náhodnou farbou z možnej množiny farieb a chýbajúcu výšku hráča hodnotou z rozsahu najmenšej a najvyššej výšky v množine. Vo všetkých prípadoch musíme byť opatrní, pretože zmeny v údajoch môžu ovplyvniť úspešnosť modelu a výsledky, ktoré získame. Veľmi dôležité je aj to, v ktorom bode tieto opravy vykonáme. O tom si povieme neskôr.

Duplikáty

Prítomnosť duplikátov v dátovom súbore môže ovplyvniť generalizačnú schopnosť modelu. Preto je vždy vhodné skontrolovať, či sa v dátovom súbore nenachádzajú opakujúce sa alebo veľmi podobné údaje. V prípade tabuľkových údajov je možné duplikáty nájsť priamym porovnaním hodnôt atribútov. Pri práci s rôznymi typmi údajov zvyčajne potrebujeme pokročilejšie techniky. Napríklad duplikáty obrázkov môžu byť symetrické, ako v zrkadle, buď horizontálne, alebo vertikálne. To isté platí pre textové údaje. Dve spravodajské správy môžu obsahovať rovnaké oznámenie (uverejnené niektorými spravodajskými agentúrami) s mierne odlišnými nadpismi, takže z hľadiska priameho porovnania znakov sú odlišné, ale zároveň rovnaké.

Hľadanie výnimiek

Všimnutie si údajov, ktoré sa nejakým spôsobom líšia od ostatných, nám umožňuje odhaliť chyby v údajoch alebo objaviť nové, atypické správanie. Takéto údaje sa nazývajú výnimky alebo *odľahlé hodnoty*. Vzdialenosť od letiska, ktorá je -1,2 km, by bola nezrovnalosťou, pretože očakávame, že vzdialenosť bude kladnou hodnotou. Takýmto spôsobom by sme mohli chybu odhaliť a opraviť. Na druhej strane, teplota 45 °C je tiež neobvyklá hodnota, ale reálna v dôsledku klimatických zmien a možno veľmi užitočná ako informácia pre prijatie určitých krokov a opatrení.

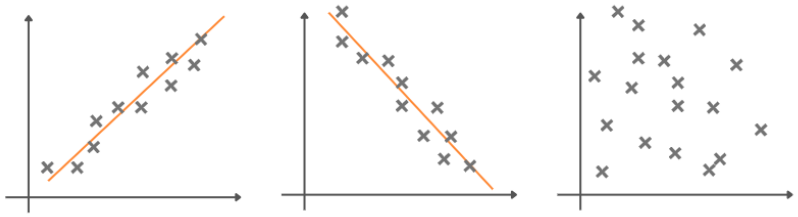
Grafické znázornenie odchýlky

Rozpory môžu ovplyvniť aj výsledky algoritmov strojového učenia. Preto je dôležité, aby sa po ich zistení a spracovaní rozhodlo, či sa majú zachovať alebo vymazať.

Korelácia atribútov

Atribúty môžu súvisieť medzi sebou. Súvislosť môžeme vidieť, ak nakreslíme graf, ktorý má hodnotu jedného atribútu na osi x a hodnotu iného atribútu na osi y. Môžeme napríklad sledovať páry atribútov vonkajšia teplota a počet predaných zmrzlín, vonkajšia teplota a spotreba elektrickej energie a vonkajšia teplota a počet kníh v knižnici. Nech každý z týchto párov zodpovedá grafu, ako je na obrázku nižšie. Môžeme si všimnúť, že nárast

teploty sprevádza nárast počtu predaných zmrzlín. Ak nárast hodnoty jedného atribútu nasleduje po náraste hodnoty iného atribútu, hovoríme, že sú **pozitívne korelované.** Na grafe môžeme tiež pozorovať, že táto závislosť je lineárna, t. j. že sleduje imaginárnu líniu, ktorá prechádza súborom bodov. Na druhej strane sa zdá, že situácia s vonkajšou teplotou a spotrebou elektriny je trochu iná, t. j. že pokles teploty sprevádza vyššia spotreba elektriny, pravdepodobne v dôsledku používania ohrievačov. Atribúty, pri ktorých nárast hodnoty jedného atribútu nasleduje pokles hodnoty druhého atribútu, sa považujú za **negatívne korelované.** Z grafu môžeme opäť usúdiť, že tento druh korelácie je lineárny. Tretí graf, ktorý zobrazuje vonkajšiu teplotu a počet kníh v knižnici, neukazuje žiadnu, aspoň nie zrejmú, pravidelnosť medzi atribútmi. Môžeme určite usúdiť, že tieto atribúty nie sú lineárne korelované.



Grafy asociácie atribútov

Na meranie lineárneho vzťahu atribútov môžeme použiť aj rôzne typy koeficientov, ktoré sú stanovené v oblasti matematicko-štatistickej analýzy. Jedným z takýchto koeficientov je Pearsonov korelačný koeficient. Jeho hodnoty sa pohybujú od -1 do 1 a indikujú smer aj silu spojenia. Hodnoty koeficientu bližšie k -1 indikujú negatívnu koreláciu, hodnoty koeficientu bližšie k 1 indikujú pozitívnu koreláciu a hodnoty okolo nuly indikujú absenciu lineárnej korelácie.

Hodnoty korelačných koeficientov medzi atribútmi sa zvyčajne zobrazujú graficky vo forme takzvanej tepelnej mapy. Každý štvorček na tejto mape zodpovedá jednej dvojici atribútov a jeho farba je prispôsobená hodnote korelačného koeficientu. Stĺpec umiestnený na boku tejto mapy spája hodnoty a odtiene farieb. Pozorovaním tejto mapy môžeme ľahko vidieť korelácie v údajoch. Na obrázku nižšie sú zobrazené páry atribútov jednej dátovej sady, ktorá kombinuje informácie o zamestnancoch. Hoci o tejto sade vieme málo, môžeme usúdiť, že skúsenosti a počet rokov (atribút vek) najlepšie sledujú *hodnoty platov.* Môžeme tiež vidieť, že existuje korelácia medzi počtom rokov (atribút vek) a atribútom skúsenosti.

Teplotná mapa s hodnotami korelačného koeficientu

Zistenie atribútov, ktoré súvisia, nám v prvom rade umožňuje lepšie pochopiť oblasť, na ktorú sa údaje vzťahujú. Niektoré súvislosti sa dajú očakávať, zatiaľ čo iné nám môžu priniesť nové poznatky. Odstránením atribútov, ktoré sú prepojené, môžeme znížiť dimenzionalitu súboru údajov. Týmto spôsobom môžeme urýchliť prácu niektorých algoritmov a ľahšie pochopiť výsledky. Existujú aj algoritmy strojového učenia, ktoré nefungujú dobre, ak sú v súbore údajov asociácie – odstránenie atribútov, na ktoré sa to vzťahuje, môže zlepšiť úspešnosť algoritmu.

Táto lekcia je spojená s notebookom Jupyter 03-exploratory-data-analysis.ipynb. Ak chcete vyskúšať úlohy, ktoré sme opísali, kliknite na odkaz a potom na tlačidlo „Open in Colab " (Otvor v *Google Colab)*, aby sa obsah otvoril *v prostredí Google Colab*. Ak si notebooky prezeráte na svojom lokálnom počítači, vyhľadajte medzi obsahom notebook s rovnakým názvom a spustite ho. Podrobnejšie pokyny nájdete v časti *Praktická zóna* a v lekcii *Jupyter exercise notebooks.*

V zošite Jupyter boli pomocou funkcií *knižnice Pandas* analyzované údaje zo súboru Titanic. Tento súbor obsahuje informácie o cestujúcich, ktorí sa nachádzali na slávnej lodi Titanic, keď v roku 1912 pri plavbe Atlantickým oceánom narazila na ľadovec a stroskotala.

2.8 Vytvorenie reprezentácie dátového súboru

Po exploratívnej analýze dátového súboru môžeme rozhodnúť, ktoré atribúty a inštancie vyradíme. Zostávajúci dátový súbor je potrebné pripraviť tak, aby sme naň mohli aplikovať niektoré algoritmy strojového učenia. V závislosti od typu atribútu a súboru hodnôt, ktoré obsahuje, si nižšie môžete prečítať o niektorých technikách prípravy. V ďalšej lekcii sa dozviete, kedy je na to ideálny čas.

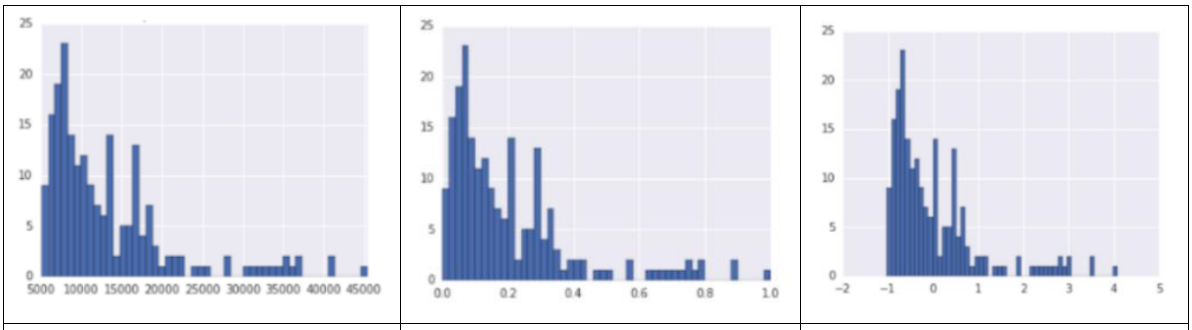
Príprava numerických atribútov

Pri práci s numerickými atribútmi sa stretávame s veličinami, ktoré sú vyjadrené v rôznych hodnotových škálach. Napríklad v jednej sade lekárskych údajov môžu byť laboratórne analýzy s hodnotami od 0 do 1 a informácie o výške pacienta vyjadrené v centimetroch od 100 do 250, čo je výrazne vyššia hodnota. Mnohé modely strojového učenia sú citlivé na prítomnosť týchto atribútov, a preto trvá dlhšie, kým nájdu riešenie. Nie je ľahké pochopiť výsledky, ktoré prináša tento druh práce. Preto pri práci s numerickými údajmi používame normalizačné techniky, ktoré nám pomáhajú zjednotiť súbor hodnôt atribútov do rovnakých rozsahov hodnôt.

Jednou z takýchto normalizácií je *min-max* normalizácia. Aby sme objasnili, ako sa vykonáva, predpokladajme, že potrebujeme normalizovať hodnotu atribútu X, aby sme vyjadrili výšku pacientov. Nech Xmax = 180 predstavuje maximálnu výšku pacientov a Xmin = 110 najmenšiu. Normalizácia *min-max* sa vykonáva použitím vzorca (x – Xmin)/(Xmax – Xmin) na každú hodnotu atribútu x. Ak x = 165, nová normalizovaná hodnota bude x'= (165 – 110)/(180 – 110) = 0,786. Týmto spôsobom sa hodnota atribútu zníži na rozsah od 0 do 1.

Jedným z najdôležitejších aspektov normalizácie je štandardizácia. Zahŕňa centrovanie hodnoty atribútu okolo nuly a škálovanie na jednotkovú varianciu. Aby sme objasnili, ako sa to vykonáva, môžeme opäť predpokladať, že potrebujeme normalizovať hodnotu atribútu X, ktorý vyjadruje výšku pacientov. Nech teraz Xmean = 153,2 je priemerná výška v dátovom súbore a σ = 40,23 je štandardná odchýlka. Štandardizácia sa vykonáva

použitím vzorca (x − Xmean)/σ na každú hodnotu atribútu x. Nová štandardizovaná hodnota pre pacienta, ktorého výška je x = 165, je teraz x'= (165 − 153,2)/40,23 = 0,293.

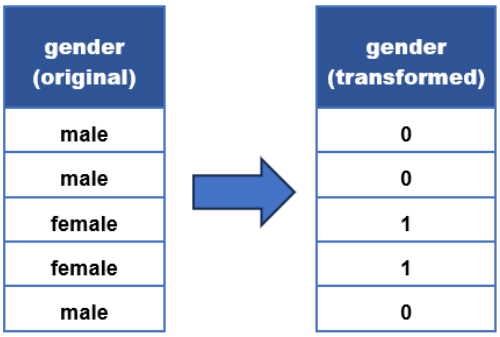


Pôvodný súbor	min -max normalizácia	Štandardizácia

Vplyv normalizácie a štandardizácie na súbor údajov

Príprava kategorických atribútov

Keďže algoritmy strojového učenia možno aplikovať iba na čísla, kategorické atribúty vyžadujú špeciálnu prípravu. Uviedli sme, že predstavujú veličiny, ktoré majú konečný počet hodnôt a často sa vyskytujú vo forme reťazca. Niektoré z príkladov, ktoré sme spomenuli, sú názov farby, pohlavie pacienta a mesiac v roku.
Ak má atribút len dve hodnoty, napríklad pohlavie pacienta, jeho hodnoty sa zvyčajne priraďujú k číslam 0 alebo 1. Napríklad hodnota „žena" sa môže priradiť k číslu 1 a hodnota „muž" k číslu 0. Tieto atribúty sa inak nazývajú binárne atribúty.



Príklad priradenia hodnôt

Pre atribúty, ktoré môžu mať viacero hodnôt, používame kódovanie *one-hot*. Aby sme objasnili jeho význam, môžeme sa pozrieť na atribút, ktorý reprezentuje farbu, ktorá môže mať tri hodnoty: červená, žltá a zelená. Myšlienka spočíva v tom, že predvolený atribút farby reprezentujeme pomocou troch nových atribútov, z ktorých každý bude zodpovedať jednej z hodnôt, ktoré farba môže mať: červená, žltá a zelená (pozrite sa na obrázok, bola to zložitá veta). To ďalej znamená, že každú z hodnôt pôvodného atribútu premeníme na trojicu hodnôt, konkrétne hodnotu *červenej* na trojicu 1*,* 0*,* 0*,* hodnotu *žltej* na trojicu 0*,* 1*,* 0 *a* hodnotu *zelenej* na trojicu *0,* 0*,* 1*.* Trojice, ako vidíme, pozostávajú z núl a presne jednej jednotky v stĺpci, ktorý zodpovedá hodnote atribútu.

COLOR
red
red
yellow
green
yellow

RED	YELLOW	GREEN
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Príklad jednorazového mapovania

Reprezentácia dátového súboru

Po kroku transformácie atribútov dospejeme k konečnej podobe údajov, ktorú môžeme použiť na spustenie algoritmov učenia. Táto konečná podoba sa nazýva **reprezentácia súboru údajov.** V doterajšom príbehu sme sa najskôr venovali tomu, ako dospieť k reprezentácii tabuľkových dát. A pre všetky ostatné typy dát, ako sú obrázky, zvuky, text, video obsah, ale aj zložité štruktúry, ako sú grafy, musíme vytvoriť vhodné reprezentácie. V časti o neurónových sieťach sa dozvieme o niektorých ďalších spôsoboch vytvárania reprezentácií.

2.9 Trénovacie, validačné a testovacie súbory

V tejto lekcii sa dozviete o trénovacích, validačných a testovacích súboroch. Trénovací súbor si môžete predstaviť ako literatúru, z ktorej sa model strojového učenia učí, zatiaľ čo testovací súbor si môžete predstaviť ako kontrolnú úlohu, ktorá overuje, ako dobre sa model naučil a pochopil požadovaný materiál. Validačná sada sa používa v procese učenia modelu a môžete si ju predstaviť ako súbor pomocných testov, ktoré kontrolujú, ako je model pripravený na kontrolu, a na základe ktorých výsledkov môže model zlepšiť spôsob a úspešnosť učenia.

Po analýze údajov a výbere vhodných inštancií a atribútov sa súbor všetkých údajov rozdelí na **trénovací súbor** (trénovateľný) a **testovací súbor**. Ako možno odhadnúť z názvu súboru, trénovací súbor sa používa na trénovanie samotného modelu strojového učenia. Naň sa aplikuje vybraný algoritmus, ktorý vytvorí samotný model. Testovací súbor sa používa na testovanie modelu, t. j. na výpočet vhodných meradiel kvality modelu. Vďaka nemu je možné objektívne posúdiť, ako dobre sa model naučil potrebnú úlohu. Zvyčajne používame aj časť údajov zo základného súboru údajov na vytvorenie **validačnej sady.** Validačná sada sa používa na sledovanie procesu trénovania modelu a určenie niektorých konfigurácií modelu, ktoré vedú k lepším meradlám kvality. Tieto témy budú rozoberané neskôr v kurze.

Rozdelenie dátového súboru na trénovací súbor, validačný súbor a testovací súbor

Trénovacie, validačné a testovacie súbory sa zvyčajne vytvárajú náhodným rozdelením základného súboru údajov. Najskôr definujeme, aké veľké by mali byť tieto súbory, a potom náhodne vyberieme inštancie, ktoré sa v každom z nich nachádzajú. Trénovací súbor je zvyčajne najväčší, zatiaľ čo testovací a validačný súbor sú menšie, pretože chceme mať dostatok údajov na trénovanie modelu, ale aj dostatok údajov na adekvátne hodnotenie jeho výkonu. V praxi sa pomer veľkostí týchto súborov vyjadruje v pomere. Napríklad pomer trénovacej sady k testovacej sade sa často uvádza ako 2:1, čo znamená, že dve tretiny počiatočnej sady tvoria trénovaciu sadu a jedna tretina testovaciu sadu. Podobne pomer 2:1:1 by znamenal, že dve štvrtiny (t. j. polovica) základnej sady by sa použili ako trénovacia sada a jedna štvrtina ako validačná a testovacia sada.

Hoci je výhodné, že trénovacie, validačné a testovacie súbory sú vytvárané náhodne, stále by to znamenalo určitý pohľad na to, ako sa toto rozdelenie vykonalo. Napríklad, keď chceme replikovať experiment alebo umožniť iným, aby ho vykonali sami (toto je dôležitá vlastnosť experimentov a nazýva sa reprodukovateľnosť), je výhodnejšie použiť rovnaké trénovacie súbory, validáciu a testovanie. Podobne, keď riešime problém, nie sme si hneď istí, čo je najlepšie urobiť, takže vyskúšame väčší počet algoritmov a vytvoríme väčší počet modelov. V záujme spravodlivosti porovnania by to znamenalo, že vytvoríme všetky modely nad rovnakou tréningovou sadou a vyhodnotíme ich nad rovnakou testovacou sadou. Preto je dobré nastaviť parameter na úrovni knižnice, ktorý ovplyvňuje náhodnosť rozdelenia (zvyčajne sa nazýva *náhodné semeno* a má rovnaký účel ako nastavenie semien v generátore náhodných čísel), alebo jednoducho rozdeliť dáta od začiatku a pokračovať v ich konzistentnom používaní . Niektoré bežne používané dátové súbory majú tieto preddefinované rozdelenia na súbor pre trénovanie, validáciu a testovanie (napríklad si môžete pozrieť súbor *MNIST*).

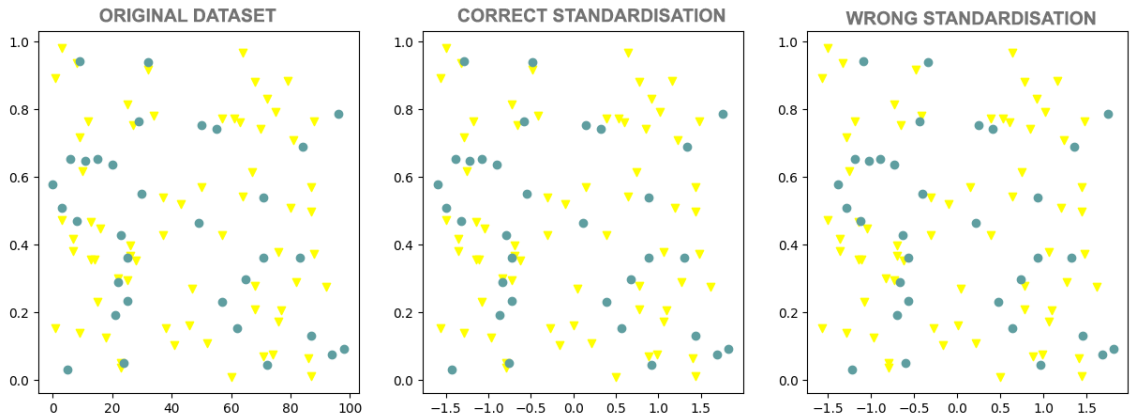
Dôležitou vlastnosťou, ktorú musia spĺňať súbory na trénovanie, validáciu a testovanie, je to, že musia byť disjunktné. To znamená, že každá inštancia zdrojového súboru údajov pri vytváraní súborov na trénovanie, validáciu a testovanie musí patriť presne do jedného z týchto súborov a nesmie dochádzať k prekrývaniu a zdieľaniu inštancií. Treba mať na pamäti, že od modelov strojového učenia sa očakáva, že budú dobre generalizovať, t. j. že sa budú správne správať v prípade nových inštancií, s ktorými sa model nemal možnosť stretnúť v trénovacej sade. Ak sa trénovacie a testovacie sady prekrývajú, nebudeme môcť objektívne posúdiť, či sa model skutočne učí, alebo si pamätá informácie z trénovacej sady. To isté platí pre vzťah medzi trénovacou sadou a validačnou sadou: funkcia validačnej sady je pomôcť vybrať konfigurácie, ktoré zabezpečia čo najúspešnejšie učenie. Ak sa tieto sady prekrývajú, nebudeme môcť objektívne a nestranne vyhodnotiť správanie modelu a vybrať vhodné konfigurácie.

Požiadavka, aby boli trénovacie, validačné a testovacie súbory disjunktné, znamená tiež, že informácie z jedného súboru sa nesmú prenášať do ostatných súborov. To je obzvlášť dôležité pri aplikovaní techník predspracovania a prípravy súborov. Pozrime sa na nasledujúci príklad. Spomenuli sme, že vzhľadom na citlivosť modelu strojového učenia na hodnotu atribútov sa často vykonáva štandardizácia numerických atribútov. K tejto transformácii možno pristupovať tak, že sa vypočíta priemer a štandardná odchýlka na základe celej sady a potom sa použije na štandardizáciu trénovacích a testovacích sád. Aby nedošlo k pretečeniu informácií, správny postup týchto krokov je v skutočnosti nasledovný:

- rozdelenie dátového súboru na trénovaciu a testovaciu sadu,
- výpočet priemeru a štandardnej odchýlky iba na trénovacej sade,
- transformácia trénovacej sady pomocou vypočítaných hodnôt,
- transformácia testovacej sady pomocou hodnôt vypočítaných na trénovacej sade.

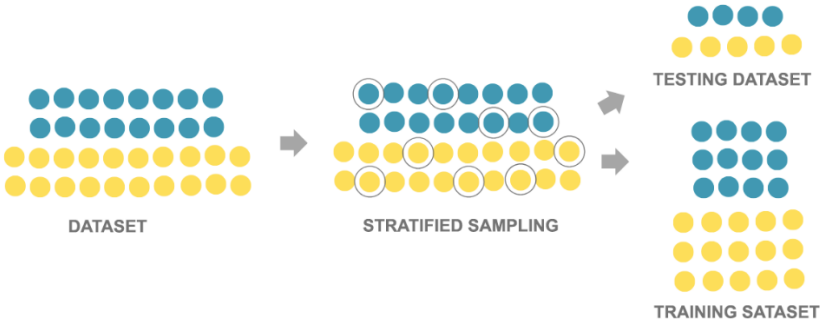
Z dôvodu snahy vyhnúť sa chybám by sa dalo uvažovať aj takto: po rozdelení pôvodného súboru údajov na trénovací a testovací súbor vykonám samostatnú štandardizáciu trénovacieho a testovacieho súboru. Tento prístup je síce opatrnejší, ale nie je správny, pretože vedie k modifikácii testovacieho súboru. Na obrázku

vľavo dole žlté trojuholníky predstavujú inštancie trénovacieho súboru a modré kruhy predstavujú inštancie testovacieho súboru. Obrázok uprostred predstavuje tieto inštancie po správnej štandardizácii (môžete starostlivo porovnať obrázky a usporiadanie bodov – mierka pozdĺž osi x sa zmenila v dôsledku štandardizácie, všetko ostatné zostalo rovnaké). Na obrázku vpravo vidíte inštancie po tom, čo bola štandardizácia vykonaná samostatne na trénovacej a testovacej sade – priestorové usporiadanie sa teraz dosť zmenilo.



Príklady správnej a nesprávnej štandardizácie

Pri delení základnej dátovej sady by bolo ideálne zachovať pomery vzhľadom na hodnoty atribútov a hodnotu cieľovej premennej. Napríklad, ak je pomer mužských a ženských pacientov v lekárskej dátovej sade 4:5, bolo by ideálne, aby po rozdelení bol pomer pacientov v trénovacej sade a v testovacej sade približne 4:5. Techniky, ktoré umožňujú tento typ rozdelenia, sa nazývajú **stratifikačné** techniky. Vzhľadom na počet atribútov a ich kombinácií však v praxi nie je táto požiadavka často realistická, preto sa najčastejšie trvá na proporcionalite vo vzťahu k hodnotám cieľovej premennej. Túto tému budeme diskutovať samostatne v kontexte úlohy klasifikácie.



Stratifikované trénovacie a testovacie súbory

3. TRÉNINGOVÉ MODELY

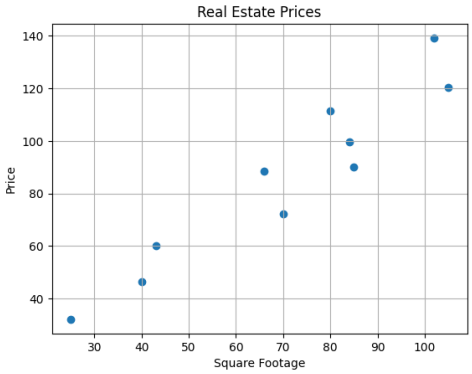
Vitajte v téme **Tréningové modely**! Táto časť sa zaoberá rôznymi modelmi a technikami používanými v strojovom učení na riešenie rôznych typov problémov. Dozviete sa o lineárnej regresii, klasifikácii, rozhodovacích stromoch a algoritme k-najbližších susedov, okrem iného. Budeme diskutovať o charakteristikách, výhodách a nevýhodách každého modelu a o dôležitosti validácie modelu na zabezpečenie presnosti a spoľahlivosti. Prostredníctvom praktických cvičení budete tieto modely aplikovať na reálne dátové súbory a interpretovať výsledky.

3.1 Lineárna regresia

Vráťme sa k príkladu, ktorý sme použili na predstavení paradigmy programovania založeného na údajoch. V súbore údajov sme mali informácie o rozlohe nehnuteľností a ich cenách a našou úlohou bolo zistiť súvislosť medzi týmito hodnotami, aby sme mohli odhadnúť ceny nových nehnuteľností.
Pre jednoduchosť použijeme trénovací súbor, ktorý obsahuje 10 prípadov, ktoré sú uvedené v tabuľke nižšie:

Plocha (m^2)	Ceny nehnuteľností (1000 €)
43	60
25	32,1
66	88,4
80	111,4
105	120,32
70	72,1
40	46
85	90,1
84	99,6
102	139,2

Tieto údaje môžeme zobraziť aj graficky. Na osi x nastavíme hodnoty plochy v štvorcových stopách, na osi y hodnoty cien nehnuteľností a páry hodnôt označíme modrými kruhmi.



Vyberme pre model funkciu, ktorá spája plochu nehnuteľností x a ceny nehnuteľností y rovnicou$y = \beta_0 + \beta_1 x$, kdeβ_0 iβ_1 predstavujú neznáme parametre. Ide o takzvaný **lineárny model** a keďže ho používame na riešenie regresného problému, nazývame ho aj **lineárnym regresným modelom.** Všimnite si, že ide vlastne o rovnicu priamky$y = kx + n$, kde koeficient priamky je označený akoβ_1 a voľný člen je označený akoβ_0 . Motiváciou pre zavedenie tohto modelu je skutočnosť, že body sledujú imaginárnu diagonálu kvadrantu, možno mierne znížený.

Táto časť je spojená s Jupyter Volume 05-1-linear_regression.ipynb. Ak chcete pokračovať v čítaní obsahu, kliknite na odkaz a potom na tlačidlo Open in Colab , aby sa obsah otvoril v *Google Colab.* Ak si prezeráte zošity na svojom lokálnom počítači, vyhľadajte zošit s rovnakým názvom medzi obsahom a spustite ho. Podrobnejšie pokyny nájdete v časti *Praktická zóna* a v lekcii *Jupyter Exercise Notebook*.

Vašou úlohou je načítať súbor údajov o nehnuteľnostiach do sprievodného zošitu a vybrať hodnoty parametrovβ_0 aβ_1 , ktoré podľa vás najlepšie zodpovedajú týmto údajom. Môžete ich upraviť posúvaním posuvníkov doľava a doprava. Zapamätajte si hodnoty, ktoré ste zvolili, a myšlienky, ktorými ste sa riadili pri určovaní parametrov.

Pravdepodobne ste sa snažili nájsť správnu hodnotu, ktorá sa čo najviac približuje daným bodom a má čo najmenej odchýlok. S niektorými voľbami parametrov ste boli rovnako spokojní, zatiaľ čo iné boli naozaj zlé. Z hľadiska strojového učenia sa snažíme nájsť hodnoty parametrovβ_0 aβ_1 , pre ktoré dosiahneme najmenšiu chybu, ale musíme presne definovať, čo táto chyba vlastne je. Tu je postup, ako to urobíme.

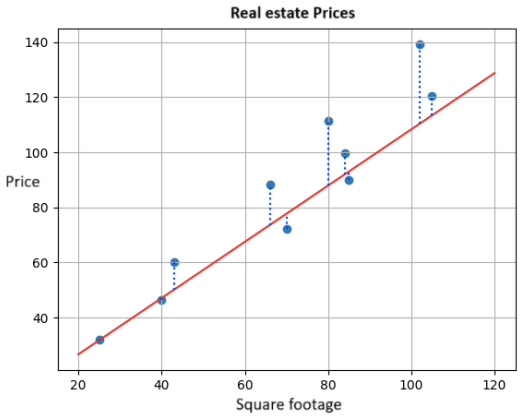
Predpokladajme, že vybrané hodnoty sú$\beta_0 = 6.3$ i$\beta_1 = 1.02$. Teraz rozšírime tabuľku údajov o stĺpec s hodnotami vypočítanými týmto lineárnym regresným modelom pre hodnoty kvadratúry, ktoré máme. Z hľadiska modelu ich reprezentujeme veľkosťou x.

Plocha v štvorcových stopách (m^2)	Ceny nehnuteľností (1000 €)	Modelová cena 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
40	46,3	47
85	90,1	93
84	99,6	91,98
102	139,2	110,34

Rozdiel medzi očakávanými hodnotami (známymi v súbore údajov) a hodnotami, ktoré sme vypočítali (nezabudnite ich nazvať predikciami), je chyba. Teraz vypočítame všetky chyby a zaznamenáme ich do tabuľky.

Plocha v štvorcových stopách (m^2)	Ceny nehnuteľností (1000 €)	Modelová cena y=6,3+1,02x (1000 €)	Chyba modelu
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

Aby bolo ľahšie sledovať správanie chýb, na obrázku nižšie sú ich hodnoty znázornené modrými bodkovanými čiarami.



Aby sme získali predstavu o celkovej chybe modelu, nie je rozumné sčítať jednotlivé chyby, pretože niektoré hodnoty chýb sú kladné a niektoré záporné. Preto ich môžeme umocniť na druhú a sčítať – tým získame presnejšie informácie o veľkosti chyby, bez ohľadu na to, či je kladná alebo záporná. Ak výsledný súčet vydelíme počtom inštancií v sade, zistíme priemernú chybu modelu. V našom prípade je to:

$$(9.84^2 + 0.32^2 + 14.782^2 + 23.52^2 + 6.92^2 + (-5.6)^2 + (-0.8)^2 + (-2.9)^2 + 7.62^2 + 28.86^2)/10 = 184.687$$

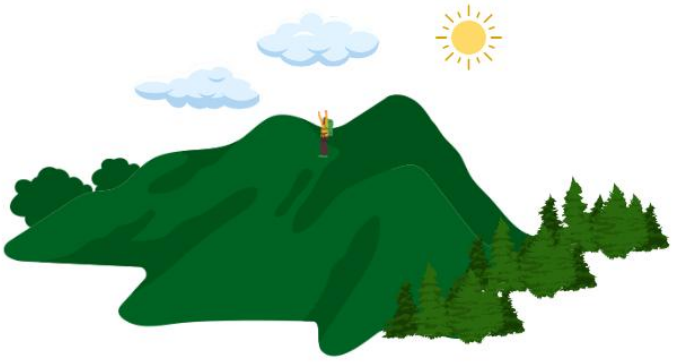
Chyba lineárneho regresného modelu vypočítaná týmto spôsobom sa nazýva **stredná kvadratická chyba (***MSE)*. Pre pevné hodnoty parametrovβ_0 aβ_1 možno postup výpočtu, ktorý sme opísali, skrátiť vzorcom$\frac{1}{N}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))^2$. V ňom páryx_i, y_i zodpovedajú jednotlivým inštanciám, štvorcová plocha nehnuteľnostíx_i a ich cenyy_i a číslo n označuje celkový počet inštancií. V našom prípade je to 10. Výraz uvedený v súčte predstavuje rozdiel medzi očakávanými hodnotami yi a vypočítanými hodnotami$\beta_0 + \beta_1 x_i$.

Kvadratická chyba je chyba, ktorú vždy spájame s lineárnym regresným modelom a ktorú chceme čo najviac minimalizovať výberom správnych parametrovβ_0 aβ_1 . Zo skúseností s nastavovaním parametrov ste zistili, že to nie je veľmi ľahká úloha. Našťastie existujú matematické techniky, ktoré nám v tom môžu pomôcť. Aby sme zistili, ako to urobiť, prejdime k ďalšej lekcii o gradientnom zostupe.

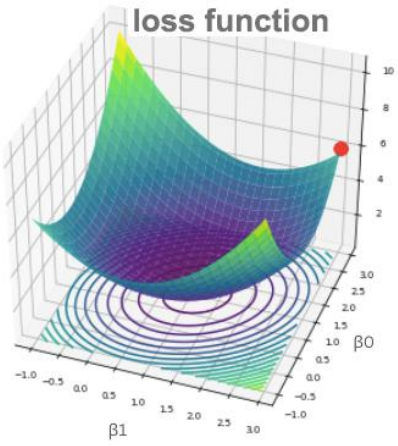
3.2 Gradientný zostup

V tejto lekcii sa naučíme o gradientovom zostupe, technike, ktorá nám pomáha nájsť parametre, pre ktoré má funkcia strednej kvadratickej chyby najmenšiu hodnotu. Táto technika sa za určitých podmienok dá použiť aj na iné funkcie.

Zase si budete musieť niečo predstaviť – tentoraz ste na vrchole krásnej hory. To nie je také ťažké! Problém je, že teraz nasleduje úloha: rýchlo sa dostať dolu! Jedným zo spôsobov, ako to urobiť, je najprv sa rozhliadnuť a zistiť, v ktorom smere je hora vo vašej oblasti najstrmšia – majte na pamäti, že musíte zísť veľmi rýchlo! Potom môžete urobiť opatrný krok tým smerom, zastaviť sa a znova sa rozhliadnuť. Znovu sa môžete pozrieť, v ktorom smere je hora vo vašej oblasti najstrmšia, urobiť krok tým smerom a zastaviť sa. Je vám jasné, že môžete pokračovať v opakovaní tohto postupu pozorovania, výberu smeru a výpadu, kým nedosiahnete úpätie. Za úspešne splnenú úlohu na vás čaká občerstvenie!



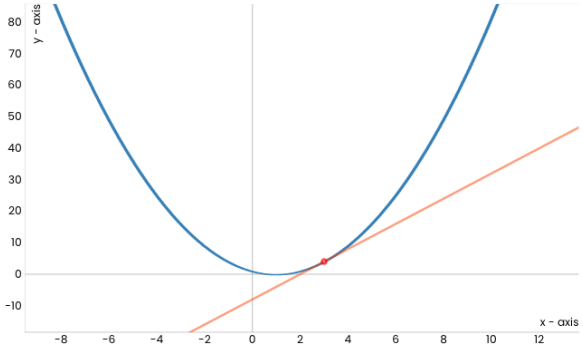
Trochu fyzickej aktivity uprostred príbehu o lineárnej regresii neuškodí, ale cítite, že je tu ešte niečo iné. Funkcia strednej kvadratickej chyby závisí od výberu parametrovβ_0 aβ_1 – pre rôzne kombinácie hodnôtβ_0 aβ_1 dostaneme rôzne hodnoty chyby. Ak nakreslíme graf tejto funkcie, napríklad pozdĺž osi x zaznamenáme hodnoty β_0 , pozdĺž osi y zaznamenáme hodnotyβ_1 a pozdĺž osi z zaznamenáme hodnoty chyby, dostaneme graf, ktorý vyzerá ako ten na obrázku nižšie. Ak označíme náhodný výber parametrovβ_0 aβ_1 červenou bodkou, aby sme sa dostali do bodu, v ktorom je hodnota chyby najmenšia, musíme skutočne zostúpiť až na dno tejto plochy. Preto je „technika", ktorú sme vyvinuli v predchádzajúcom príklade, veľmi relevantná. Musíme len zistiť, ako nájsť najstrmšie smery zostupu. Funkcie nám v tom pomôžu.



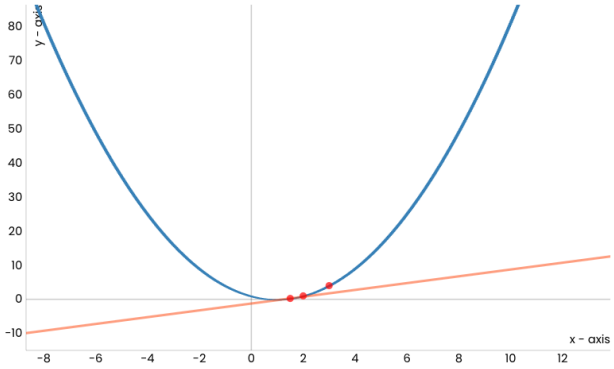
Graf funkcie strednej kvadratickej chyby

Najmenšia hodnota funkcie sa nazýva minimum.

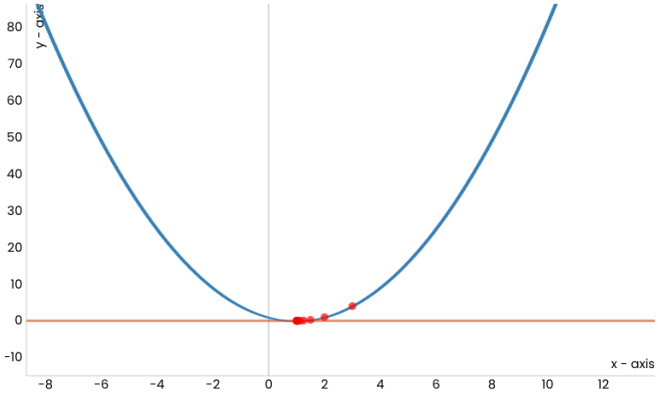
Teraz sa pozrime na kvadratickú funkciu$f(x) = (x-1)^2$, ktorej graf je zobrazený na obrázku nižšie, a pokúsme sa dosiahnuť jej minimum pomocou techniky zostupu – nachádza sa v bode x=1 a je rovné 0.



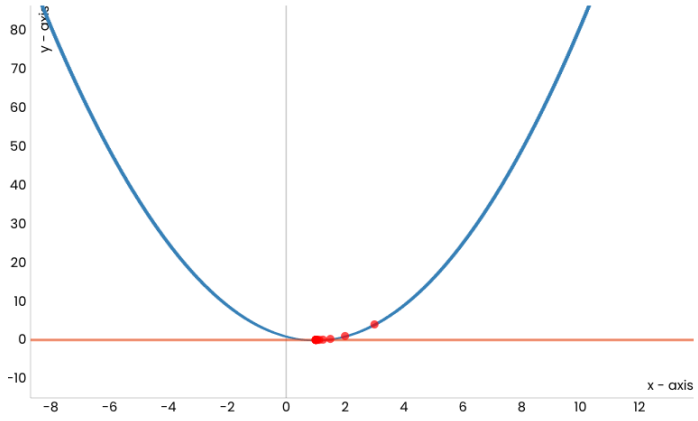
Všimnite si červenú bodku zodpovedajúcu x=3 (náhodne zvolenú), ktorá označuje počiatočnú pozíciu pohybu smerom k minimu tejto funkcie. Zdá sa, že oranžová čiara označuje najstrmší smer, po ktorom môžeme začať zostup. Zaujímavé je, že táto čiara v skutočnosti predstavuje dotyčnicu našej funkcie v bode x=3. Ak urobíme krok po tejto čiare, ocitneme sa v novom bode. Označme jeho hodnotu červenou farbou a zobrazme ju na grafe. Je o niečo bližšie k očakávanému minimu.



Teraz môžeme proces zopakovať: nakreslíme dotyčnicu v novom bode a potom urobíme krok pozdĺž tejto priamky.



Po určitom počte krokov nás tento postup privedie k minimálnej funkcii, t. j. k bodu x = 1.



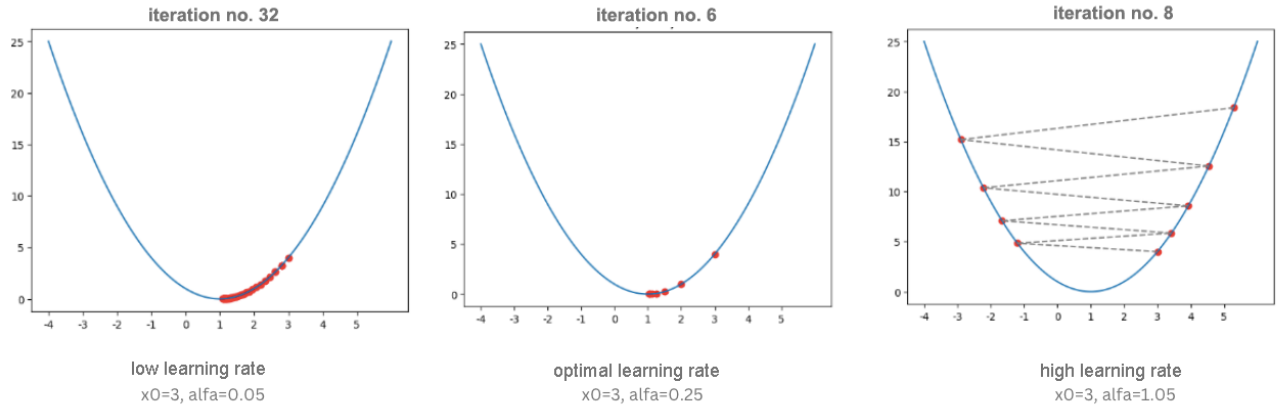
Táto časť je spojená s Jupyter Notebook 05-2-gradient_descent.ipynb. Ak chcete pokračovať v sledovaní obsahu, kliknite na tlačidlo Open in Colab d, aby sa obsah otvoril v *Google Colab.* Ak si prezeráte zošity na svojom lokálnom počítači, nájdite zošit s rovnakým názvom medzi obsahom a spustite ho. Podrobnejšie pokyny nájdete v časti *Praktická zóna* a v lekcii *Jupyter Exercise Notebook*.

V zošite, ktorý je súčasťou tohto materiálu, môžete animáciu spustiť sami a uistiť sa, že je to tak.

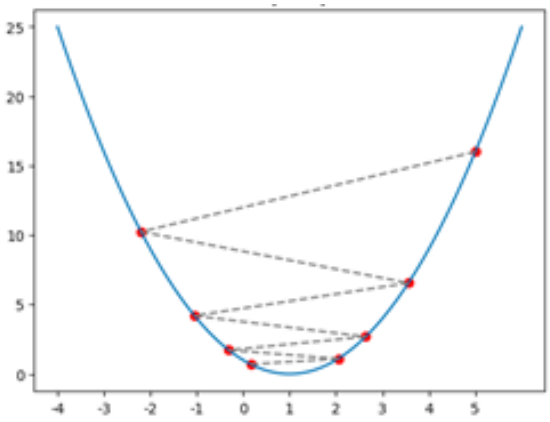
Než sa pustíme do podrobnejšieho opisu postupu, pripomeňme si, čo sú to vlastne skutočné dotyčnice. Pre pevný bod x je koeficient smeru dotyčnice v bode x rovný hodnote prvej derivácie funkcie v bode x. Prvá derivácia našej funkcie je funkcia$f'(x) = 2x - 2$ a v počiatočnom bode x = 3 je hodnota deriváci$f'(x) = 4$. To znamená, že dotyčnica má rovnicu$y = 4x - 8$ (číslo -8 sa získava z podmienky, že táto priamka musí obsahovať bod (3, 4)). Preto môžeme tiež povedať, že dotyčnica má smer zodpovedajúci derivácii funkcie v určitom bode a pre samotný pohyb v tomto smere, že pohyb v smere derivácie je v tomto bode. Teraz je dilema, či sa pohybujeme nahor alebo nadol, t. j. ideme v opačnom smere, alebo ideme v opačnom smere? Keďže chceme ísť nadol k minimu, musíme sledovať smer opačný k smeru derivácie funkcie.

Ak teraz označíme počiatočný bod akox_0 , dostaneme nový bodx_1 tým, že urobíme krok v smere derivácie funkcie v bodex_0 . Tým, že urobíme krok v smere derivácie funkcie v bodex_1 , vypočítame hodnotu nového bodu$x_1 = x_0 - \alpha f'(x_0)$. Keďže postup opakujeme, vypočítame hodnotu bodux_2 ako$x_2 = x_1 - \alpha f'(x_1)$ a pokračujeme vo výpočtoch$x_3 = x_2 - \alpha f'(x_2), x4 = x_3 - \alpha f'(x_3)$, … Postup opakujeme, kým pre dve po sebe idúce hodnoty, napríkladx_{34} ax_{35} , nie sú hodnoty funkcie dostatočne blízke, t. j. kým absolútna hodnota rozdielu$f(x_{35}) - f(x_{34})$ nie je menšia ako určitá vopred stanovená presnosť, napríklad 0,001. Z hľadiska výpočtov sa tak môžeme priblížiť k pojmu konvergencie v matematike.

Hodnota α, ktorú sme zaviedli, sa nazýva **rýchlosť učenia** a predstavuje veľmi dôležitý parameter algoritmu, ktorý sme opísali. Ak sú hodnoty α veľmi malé, dosiahnutie minima nám bude trvať dlho. Na druhej strane, ak sú hodnoty α veľmi vysoké, môže sa stať, že minimum preskočíme alebo sa dostaneme do cikcakovej pasce tým, že budeme neustále skákať okolo neho! Pozrite sa na obrázok nižšie!



Vplyv výberu krokov učenia



Cikcaková pasca

Nezabudnite si obe tieto správania overiť sami v sprievodnom zošite pomocou rôznych nastavení kroku učenia v animácii.

Algoritmus, ktorý sme opísali, sa nazýva **Gradient Descent** a napriek svojej jednoduchosti je jedným z najdôležitejších algoritmov v strojovom učení, pretože umožňuje nájsť najmenšiu hodnotu chybovej funkcie. Existuje mnoho detailov o tomto algoritme, ktorými sa nebudeme zaoberať, ako sú vlastnosti funkcií, na ktoré sa tento algoritmus dá úspešne aplikovať, numerický výpočet derivácie a výber krokov učenia. Všetky tieto detaily je potrebné zohľadniť pri praktickej aplikácii algoritmu.

Samotný algoritmus nie je nepríjemný na programovanie, takže sa pustíme do dobrodružstva. Potrebujeme funkciu f, ktorá vypočíta hodnotu danej funkcie, a funkciu f, ktorá vypočíta hodnotu derivácie danej funkcie. Musíme definovať ako alfa učiaci krok, tak aj kritériá zastavenia: postup pozastavíme, keď budú hodnoty funkcie v dvoch po sebe idúcich iteráciách dostatočne blízke (rozdiel medzi ich hodnotami je menší ako nejaká vopred stanovená epsilonová presnosť) alebo keď dosiahneme konečný počet iterácií (musíme sa uistiť aj v prípadoch nevhodného výberu učiacich krokov).

Postupujte podľa bloku kódu. Algoritmus začal nastavením počiatočného bodu. Keďže bod, v ktorom sa pohybujeme pomocou algoritmu gradientového zostupu, je počiatočným bodom ďalšieho kroku, použijeme značky x_old a x_new na ich označenie v po sebe idúcich krokoch. Správa, ktorú vytvoríme na konci funkcie,

obsahuje informácie o tom, či sa algoritmus zastavil, koľko krokov je potrebných, t. j. potrebných iterácií, a akú hodnotu našiel. Nezabudnite importovať knižnicu numpy.

```
def gradient_descent(f, f_derivative, x, alpha, epsilon, max_iterations):
    # nastavte počiatočnú hodnotu pre x
    x_old = x
    # v každej iterácii...
    for i in range(0, max_iterations):
        # vypočítaj aktuálnu hodnotu pre x
        x_new = x_old - alpha * f_derivative(x_old)
        # a potom skontroluj, či je splnené kritérium zastavenia
        if np.abs(f(x_new) - f(x_old)) < epsilon:
            break
        # ak nie je kritérium splnené, priprav x pre ďalšiu iteráciu
        x_old = x_new
    # na konci celého procesu pripravte správu s informáciami:
    # či sa algoritmus zastaví,
    # koľko iterácií trval,
    # a aká hodnota x bola nájdená
    report = {}
    report['zastaví sa'] = i != max_iterations
    report['počet_iterácií'] = i
    report['x_min'] = x_old

    return report
```

Funkcia a jej odvodenie môžu byť definované nasledujúcimi blokmi Pythonu:

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

Po spustení funkcie gradient_descent pre hodnoty argumentov x0 = 3, alpha = 0,1, epsilon = 0,001 a max_number_of_iterations = 100 dostaneme, že minimum funkcie je 1,0048, čo môžeme potvrdiť. Kód si môžete spustiť aj sami a overiť si, či dostanete rovnaký výsledok. Nezabudnite preskúmať, ako sa výsledky menia, ak sú vybrané iné hodnoty argumentov.

Teraz sa môžeme vrátiť k problému nájdenia parametraβ_0 iβ_1 lineárnej regresie, pre ktorú by hodnota strednej kvadratickej chyby mala mať najmenšiu hodnotu. Funkcia strednej kvadratickej chyby je funkciou dvoch premenných – závisí od hodnoty parametraβ_0 aj od hodnoty parametraβ_1 . Pri práci s funkciami viacerých premenných, všeobecne s n premennými$x_1, x_2, x_3, \ldots, x_n$, je derivácia, ktorú sme použili v algoritme gradientného zostupu, zovšeobecnená vektorom parciálnych derivácií – pre každú z premenných vypočítame derivácie individuálne. Napríklad pre funkciu$\frac{1}{2}(x_1^2 + 10x_2^2)$ sa derivácia premennejx_1 získa tak, že premennáx_2 sa deklaruje ako konštanta a potom sa uplatnia štandardné pravidlá pre výpočet derivácie, ktoré nás dovedú k$\frac{1}{2} \cdot 2 \cdot x_1 = x_1$. Na druhej strane, derivácia premennej x_2 sa vypočíta tak, že premenná x_1 sa deklaruje ako konštanta a uplatnia sa štandardné pravidlá pre výpočet derivácie. Teraz dostaneme$\frac{1}{2} \cdot 10 \cdot 2 \cdot x_2 = 10 \cdot x_2$. Teraz

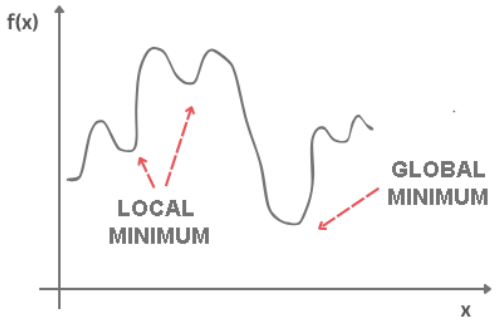
dostaneme, že vektor derivácie podľa jednotlivých premenných (takéto derivácie sa nazývajú parciálne) je vektor$[x_1, 10 \cdot x_2]$. V matematike a dokonca aj v strojovom učení sa tieto vektory nazývajú **gradienty,** od čoho pochádza aj názov samotného algoritmu. Na označenie gradientov používame symbol trojuholník smerom nadol,∇ nazývaný nabla. Presný zápis gradientu počiatočnej funkcie f' by teda bol$\nabla f(x_1, x_2) = [x_1, 10 \cdot x_2]$ a umožnil by nám sledovať, ktorými smermi by sme sa mali pohybovať individuálne počas zostupu.

Ostatné kroky algoritmu gradientného zostupu sa v prípade viacerých funkcií príliš nelíšia: očakávame, že algoritmus sa zastaví po dosiahnutí požadovanej presnosti alebo po vykonaní určitého počtu iterácií.

Teraz, keď už chápeme, ako funguje gradientný zostup pre funkcie viacerých premenných, vrátime sa k výpočtu parametrovβ_0 iβ_1 . Povedali sme, že rovnica strednej kvadratickej chyby je$\frac{1}{N}\sum_{i=1}^{N}(yi - (\beta_0 + \beta 1xi))^2$. Keďže ide o funkciu, pre ktorú potrebujeme nájsť minimum, ak si vyhrnieme rukávy a skontrolujeme to, zistíme, že derivácia strednej kvadratickej funkcie podľaβ_0 je presne$\frac{2}{N}\sum_{i=1}^{N}(\beta_0 + \beta_1 x_i - y_i)$ a derivácia podľa$\frac{1}{N}\sum_{i=1}^{N}(\beta_0 + \beta_1 x_i - y_i) \cdot x_i$. Tieto derivácie udávajú, ktorým smerom by sme sa mali pohybovať a o koľko by sme mali korigovať hodnotyβ_0 iβ_1 v každom kroku iterácie gradientného zostupu.

V zošite môžete tiež vidieť, ako sa tieto hodnoty počítajú prostredníctvom kódu, a potom prejsť celým procesom vlastného gradientového zostupu. Pre súbor nehnuteľností, ktorý sme predstavili, dospejeme k hodnotám$\beta_0 = 2.056$ a $\beta_1 = 1.198$.

Povedali sme, že existujú určité predpoklady, ktoré musí funkcia spĺňať, aby bolo možné nájsť jej minimum pomocou techniky gradientového zostupu (funkcia musí byť diferencovateľná). Je tiež dôležité vedieť, že *týmto spôsobom sa* vo všeobecnosti *dosiahne lokálne minimum*. Napríklad funkcia na obrázku nižšie má niekoľko lokálnych minim a iba jedno *globálne minimum.* V niektorých prípadoch, napríklad keď je funkcia konvexná, sa lokálne a globálne minimum zhodujú, takže vždy dospejeme k požadovanému riešeniu, globálnemu minimu. Funkcia štvorcovej chyby je konvexná podľa parametrovβ_0 aβ_1 .



Lokálne a globálne minimá.

Oblasť matematiky, ktorá sa zaoberá hľadaním maximálnych a minimálnych hodnôt funkcií (nazývame ich optimá), sa nazýva **matematická optimalizácia.** Gradientný zostup je len jeden z algoritmov z palety tejto oblasti.

3.3 Polynomiálna regresia

Čo ak sú vaše údaje v skutočnosti zložitejšie ako jednoduchá priamka? Prekvapivo, na prispôsobenie nelineárnych údajov môžete skutočne použiť lineárny model. Jednoduchým spôsobom, ako to urobiť, je pridať mocniny každej vlastnosti ako nové vlastnosti a potom trénovať lineárny model na tejto rozšírenej sade vlastností. Táto technika sa nazýva polynomiálna regresia.

Polynomiálna regresia je regresný algoritmus, ktorý modeluje vzťah medzi závislou (y) a nezávislou premennou (x) ako polynóm n-tého stupňa. Rovnica polynomiálnej regresie je uvedená nižšie:

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

V ML sa nazýva aj špeciálny prípad viacnásobnej lineárnej regresie. Pretože do rovnice viacnásobnej lineárnej regresie pridáme niektoré polynomiálne členy, aby sme ju previedli na polynomiálnu regresiu.

Ide o lineárny model s určitými úpravami s cieľom zvýšiť presnosť.

Dátový súbor použitý v polynomiálnej regresii na trénovanie má nelineárnu povahu.

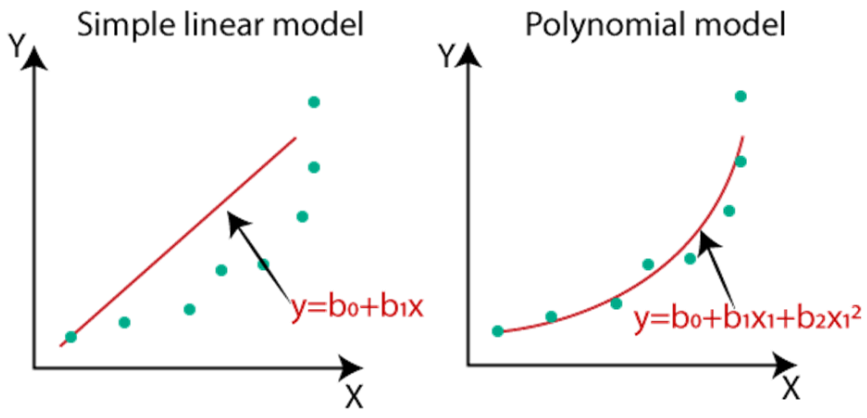
Využívajú lineárny regresný model na prispôsobenie zložitých a nelineárnych funkcií a dátových súborov.

Preto *„v polynomiálnej regresii sa pôvodné vlastnosti prevádzajú na polynomiálne vlastnosti požadovaného stupňa (2,3,..,n) a potom sa modelujú pomocou lineárneho modelu*".

Potrebu polynomiálnej regresie v ML možno pochopiť na základe nasledujúcich bodov:

- Ak použijeme lineárny model na **lineárny súbor údajov**, poskytne nám dobrý výsledok, ako sme videli v jednoduchej lineárnej regresii, ale ak použijeme ten istý model bez akýchkoľvek úprav na **nelineárny súbor údajov**, bude mať drastický výstup. V dôsledku čoho sa zvýši strata funkcie, miera chyby bude vysoká a presnosť sa zníži.

- Preto v takýchto prípadoch, **keď sú dátové body usporiadané nelineárnym spôsobom, potrebujeme model polynomiálnej regresie**. Môžeme to lepšie pochopiť pomocou nižšie uvedeného porovnávacieho diagramu lineárneho a nelineárneho dátového súboru.



- Na obrázku vyššie sme použili dátovú sadu, ktorá je usporiadaná nelineárne. Ak sa ju pokúsime pokryť lineárnym modelom, jasne vidíme, že pokrýva len veľmi málo dátových bodov. Na druhej strane, krivka je vhodná na pokrytie väčšiny dátových bodov, čo je model polynómovej regresie.

- *Ak sú* teda *dátové súbory usporiadané nelineárnym spôsobom, mali by sme namiesto jednoduchého lineárneho regresného modelu použiť polynomiálny regresný model.*

Poznámka: Algoritmus polynomiálnej regresie sa tiež nazýva polynomiálna lineárna regresia, pretože nezávisí od premenných, ale od koeficientov, ktoré sú usporiadané lineárne.

Rovnica jednoduchého lineárneho regresného modelu	$y = \beta_0 + \beta_1 x_1$
Rovnica viacnásobnej lineárnej regresie	$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + + \cdots \ldots \beta_n x_n$
Rovnica polynomiálnej regresie	$y = \beta_0 + \beta_1 x_1^1 + \beta_2 x_1^2 + \cdots \ldots \beta_n x_1^n$

Implementácia polynomiálnej regresie pomocou jazyka Python:

Tu implementujeme polynomiálnu regresiu pomocou jazyka Python. Porozumieme tomu porovnaním modelu polynomiálnej regresie s modelom jednoduchého lineárneho regresného modelu. Najprv si teda vysvetlíme problém, pre ktorý budeme model vytvárať.

Popis problému: Existuje personálna agentúra, ktorá sa chystá prijať nového kandidáta. Kandidát uviedol, že jeho predchádzajúci plat bol 160 000 ročne, a personálna agentúra musí overiť, či hovorí pravdu, alebo blafuje. Na overenie tejto informácie má k dispozícii iba súbor údajov o jeho predchádzajúcej spoločnosti, v ktorom sú uvedené platy 10 najvyšších pozícií spolu s ich úrovňami. Pri kontrole dostupného súboru údajov sme zistili, že **medzi úrovňami pozícií a platmi** existuje **nelineárny vzťah.** Naším cieľom je vytvoriť **regresný** model **na odhalenie blafovania**, aby personálne oddelenie mohlo prijať čestného uchádzača. Nižšie sú uvedené kroky na vytvorenie takéhoto modelu.

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

Kroky pre polynomiálnu regresiu:

Hlavné kroky polynomiálnej regresie sú uvedené nižšie:

- Predspracovanie údajov
- Vytvorenie lineárneho regresného modelu a jeho prispôsobenie dátovému súboru
- Vytvorte model polynomiálnej regresie a prispôsobte ho súboru údajov
- Vizualizujte výsledok lineárneho regresného modelu a polynomiálneho regresného modelu.
- Predikcia výstupu.

Krok predspracovania údajov:
Krok predspracovania údajov zostane rovnaký ako v predchádzajúcich regresných modeloch, s výnimkou niektorých zmien. V modeli polynomiálnej regresie nebudeme používať škálovanie vlastností a tiež nebudeme rozdeľovať náš súbor údajov na trénovací a testovací súbor. Má to dva dôvody:

- Súbor údajov obsahuje veľmi málo informácií, ktoré nie sú vhodné na rozdelenie na testovaciu a tréningovú sadu, inak náš model nebude schopný nájsť korelácie medzi platmi a úrovňami.
- V tomto modeli chceme veľmi presné predpovede platov, takže model by mal mať dostatok informácií.

Kód pre predspracovanie je uvedený nižšie:

```
# importovanie knižníc
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
#importovanie dátových súborov
data_set= pd.read_csv('Position_Salaries.csv')
#Extrahovanie nezávislej a závislej premennej
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

Vytvorenie lineárneho regresného modelu:

Teraz vytvoríme a prispôsobíme lineárny regresný model k dátovému súboru. Pri vytváraní polynomiálnej regresie použijeme lineárny regresný model ako referenciu a porovnáme oba výsledky. Kód je uvedený nižšie:

```
#Prispôsobenie lineárnej regresie dátovému súboru
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

V uvedenom kóde sme vytvorili jednoduchý lineárny model pomocou objektu **lin_regs** triedy **LinearRegression** a prispôsobili ho premenným dátového súboru (x a y).

Výstup:

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

Vytvorenie polynomiálneho regresného modelu:

Teraz vytvoríme polynomiálny regresný model, ktorý sa však bude trochu líšiť od jednoduchého lineárneho modelu. Pretože tu použijeme triedu **PolynomialFeatures** knižnice pre **predspracovanie**. Túto triedu používame na pridanie niektorých ďalších funkcií do nášho súboru údajov.

```
#Prispôsobenie polynomiálnej regresie dátovému súboru
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)
```

V uvedených riadkoch kódu sme použili **poly_regs.fit_transform(x)**, pretože najskôr prevádzame našu maticu vlastností na polynomiálnu maticu vlastností a potom ju prispôsobujeme polynomiálnemu regresnému

modelu. Hodnota parametra (degree= 2) závisí od nášho výberu. Môžeme ju zvoliť podľa našich polynomiálnych vlastností.

Po vykonaní kódu dostaneme ďalšiu maticu **x_poly**, ktorú môžeme vidieť v možnosti prehliadača premenných:

	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

Ďalej sme použili ďalší objekt LinearRegression, konkrétne **lin_reg_2**, aby sme prispôsobili náš vektor **x_poly** lineárnemu modelu.

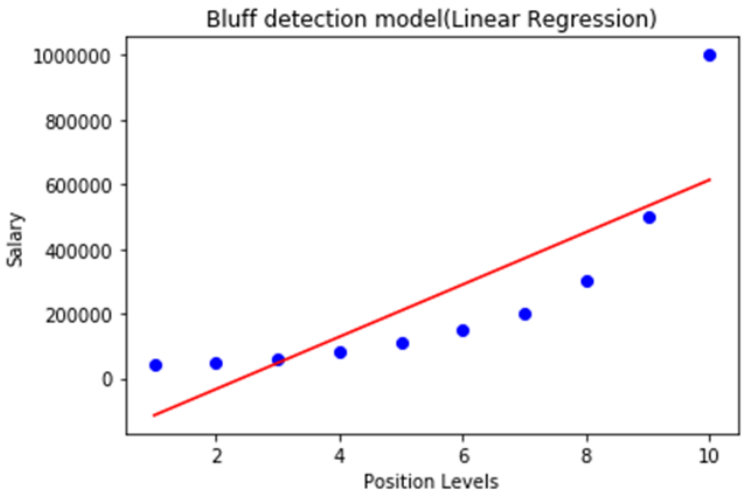
Výstup:

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

Vizualizácia výsledku lineárnej regresie:

Teraz vizualizujeme výsledok lineárneho regresného modelu, ako sme to urobili v prípade jednoduchého lineárneho regresného modelu. Nižšie je uvedený kód na to:

```
#Vizualizácia výsledku lineárneho regresného modelu
mtp.scatter(x,y,color="blue")
mtp.plot(x,lin_regs.predict(x), color="red")
mtp.title("Model detekcie blafovania (lineárna regresia)")
mtp.xlabel("Pozícia")
mtp.ylabel("Plat")
mtp.show()
```



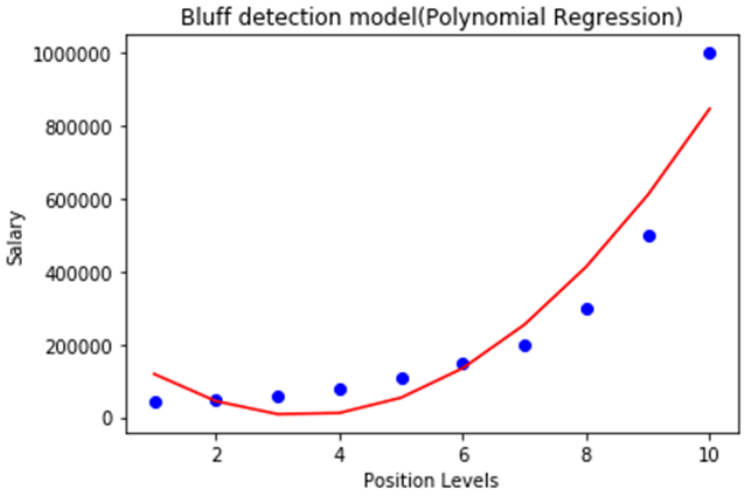
Na vyššie uvedenom výstupnom obrázku jasne vidíme, že regresná čiara je veľmi vzdialená od dátových súborov. Predikcie sú znázornené červenou priamkou a modré body sú skutočné hodnoty. Ak vezmeme tento výstup do úvahy pri predikcii hodnoty CEO, dostaneme plat približne 600 000 $, čo je veľmi vzdialené od skutočnej hodnoty.

Potrebujeme teda zakrivený model, ktorý bude lepšie zodpovedať dátam ako priamka.

Vizualizácia výsledku polynomiálnej regresie

Tu vizualizujeme výsledok polynomiálneho regresného modelu, ktorého kód sa mierne líši od vyššie uvedeného modelu.

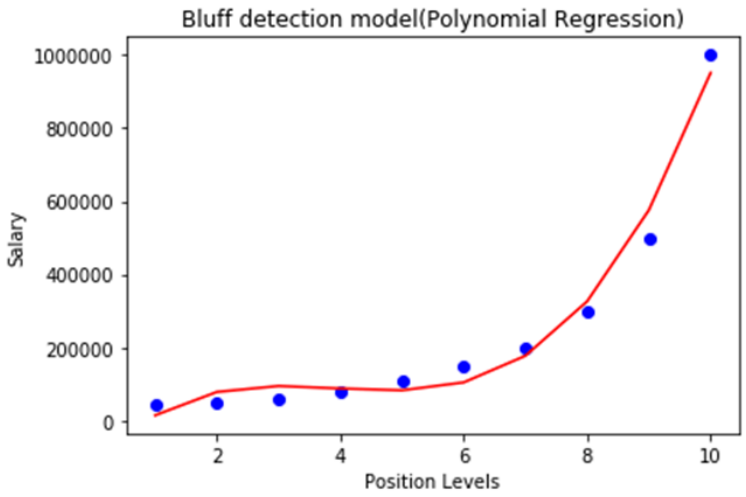
V uvedenom kóde sme použili lin_reg_2.predict(poly_regs.fit_transform(x) namiesto x_poly, pretože chceme, aby objekt lineárneho regresora predpovedal maticu polynomiálnych vlastností.



Ako vidíme na vyššie uvedenom výstupnom obrázku, predikcie sa blížia skutočným hodnotám. Vyššie uvedený graf sa bude meniť, keď zmeníme stupeň.

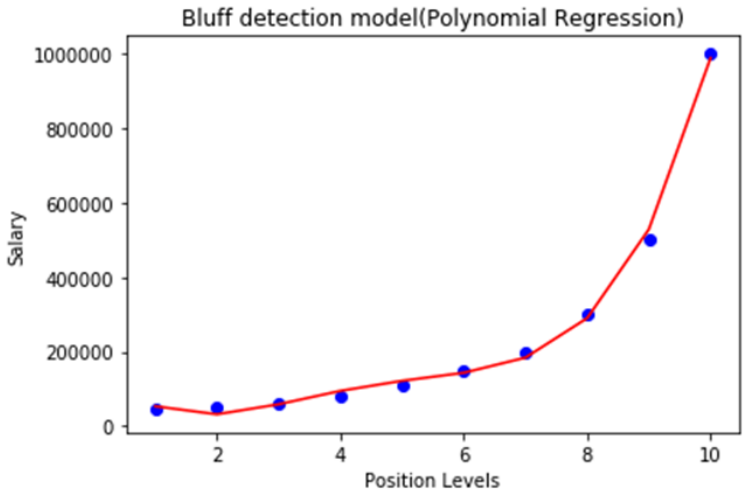
Pre stupeň = 3:

Ak zmeníme stupeň = 3, dostaneme presnejší graf, ako je znázornené na obrázku nižšie.



Ako vidíme na vyššie uvedenom výstupnom obrázku, predikovaný plat pre úroveň 6,5 sa blíži k 170 000 – 190 000 USD, čo naznačuje, že budúci zamestnanec hovorí pravdu o svojom plate.

Stupeň = 4: Zmeňme opäť stupeň na 4 a teraz dostaneme najpresnejší graf. Zvýšením stupňa polynómu teda môžeme získať presnejšie výsledky.



Predikcia konečného výsledku pomocou modelu lineárnej regresie:

Teraz predpovedáme konečný výstup pomocou modelu lineárnej regresie, aby sme zistili, či zamestnanec hovorí pravdu alebo blafuje. Na to použijeme metódu **predict()** a zadáme hodnotu 6,5. Nižšie je uvedený kód:

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

Výstup:

[330378.78787879]

Predikcia konečného výsledku pomocou modelu polynomiálnej regresie:

Teraz budeme predpovedať konečný výstup pomocou modelu polynomiálnej regresie, aby sme ho mohli porovnať s lineárnym modelom. Nižšie je uvedený kód pre tento účel:

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

Výstup:

[158862.45265153]

Ako vidíme, predpokladaný výstup pre polynomiálnu regresiu je [158862,45265153], čo je oveľa bližšie k skutočnej hodnote, preto môžeme povedať, že budúci zamestnanec hovorí pravdu.

3.4 Viacnásobná lineárna regresia

V úvodnom príbehu o dátových súboroch sme videli, že sa používa väčší počet atribútov. V príbehu o lineárnej regresii sme však použili len jeden atribút (plocha nehnuteľnosti v štvorcových stopách). Pravdepodobne sa pýtate, čo robíme, keď máme viacero atribútov, a či môžeme v takom prípade použiť model lineárnej regresie.

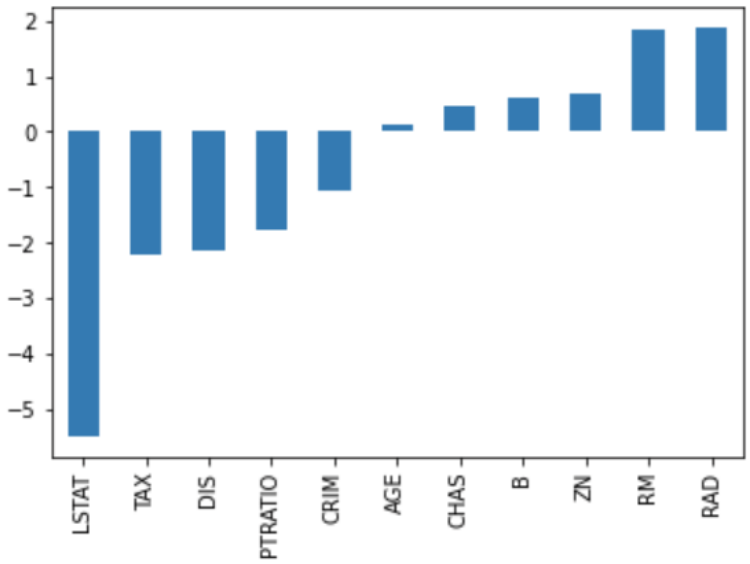
Lineárny regresný model, ktorý je prispôsobený tomuto scenáru, sa nazýva **viacnásobná** lineárna regresia a má tvar $y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_n X_n$. Nenechajte sa zmýliť týmto dlhým výrazom – hodnoty $X_1, X_2, X_3, \ldots, X_n$ predstavujú jednotlivé atribúty a parametre $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ … sú parametre modelu. Za touto generalizáciou je opäť myšlienka lineárneho vzťahu medzi jednotlivými atribútmi a cieľovou premennou.

Cieľom učenia je určiť hodnoty parametrov $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ a získať tak predstavu o závislostiach. K nim sa dostaneme rovnakým spôsobom ako v prípade lineárnej regresie, ktorú už poznáme (hovoríme tiež, že je jednoduchá): minimalizovaním strednej kvadratickej chyby na trénovacom súbore údajov. Technika gradientového zostupu sa dá zovšeobecniť tak, aby vyhovovala tejto úlohe, a môže nám pomôcť nájsť súbor hodnôt $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$, pre ktoré je stredná kvadratická chyba najmenšia.

V prípade lineárneho regresného modelu s jedným atribútom si môžeme predstaviť význam parametrov β_0 a β_1 : určovali posun a sklon priamky prechádzajúcej dátovým súborom. Ukázali nám tak silu lineárnej závislosti medzi vstupnými a výstupnými premennými, t. j. o koľko sa zmení hodnota výstupnej premennej y, keď zmeníme atribút x o 1. Teraz, keď máme viac parametrov, je prirodzené, že sa pýtame, aký význam im môžeme priradiť. Majú rovnaký druh závislosti. Ak si predstavíme, že iba β_0 a β_2 : sú parametre odlišné od nuly, potom vzťah medzi cieľovou premennou y a atribútom X2 je reprezentovaný rovnicou $y = \beta_0 + \beta_2 X_2$, tj. Lineárna a rovnaká nám hovorí, o koľko sa zmení hodnota cieľovej premennej y a v akom smere, keď zmeníme hodnotu X_2 o 1.

Vzhľadom na to, že parametre pre nás sumarizujú poznatky z dátového súboru, v prípade viacnásobnej lineárnej regresie väčšie hodnoty parametrov naznačujú väčší význam atribútu na hodnotu cieľovej premennej. Aby sme mohli sledovať túto vlastnosť, zvyčajne vynášame hodnoty vypočítaných parametrov do stĺpcového grafu. Na obrázku nižšie sú znázornené hodnoty parametrov modelu, ktorý používa reálny dátový súbor na predikciu cien nehnuteľností (populárne bostonské nehnuteľnosti). Bez toho, aby sme sa podrobne zaoberali týmto súborom, môžeme hneď zistiť, že atribút LSTAT má najväčší a negatívny vplyv na hodnotu cieľovej premennej, zatiaľ čo atribúty RM a RAD majú takmer rovnaký pozitívny vplyv. Grafické znázornenia

tohto typu, ktoré nám môžu poskytnúť určitú predstavu o vplyve atribútov, sa nazývajú **grafy dôležitosti atribútov.**



Graf dôležitosti atribútov viacnásobnej regresie

Ďalším detailom, ktorý by mal byť zdôraznený, aby vás neskôr neprekvapil, je linearita. Lineárny regresný model je **lineárny z hľadiska parametrov.** To znamená, že model, ktorého forma je $y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$, v ktorom sa počíta stupeň hodnôt atribútov, by sa spúšťal ako lineárny model. Podobne je to v prípade modelu $y = \beta_0 + \beta_1 log(X)$, v ktorom sa počíta logaritmus hodnoty atribútu. Tieto možno neočakávané úlohy atribútov si môžete predstaviť ako transformácie, ktoré fixujú lineárny vzťah medzi atribútom a cieľovou premennou.

3.5 Klasifikácia, typy klasifikácie a matica zmätkov

Verte tomu alebo nie, túto otázku ste už dostali mnohokrát. Keď upratujete izbu a triediť papiere, ktoré si necháte alebo vyhodíte, alebo keď triediť fotografie z výletu, narodenín tety alebo výletu s priateľmi, v skutočnosti vykonávate klasifikačnú úlohu: máte na mysli skupiny a keď sa pozriete na papier alebo fotografiu, rozhodujete, do ktorej skupiny patríte. A mnoho programov, s ktorými sa stretávate, vykonáva úlohu klasifikácie. Napríklad váš e-mailový klient rozlišuje medzi žiaducou a nevyžiadanou poštou a vďaka tejto funkcii sa vám darí vyhnúť sa mnohým pascám a podvodom na internete. Aj na sociálnych médiách často dostávate odporúčania na nadviazanie kontaktu s novými ľuďmi – program, ktorý stojí za sociálnou sieťou, aktívne hodnotí, či je daná osoba vaším potenciálnym priateľom alebo nie (zvyčajne sleduje priateľov vašich priateľov a získava nápady). Keďže nepochybujeme, že ste expertom na organizovanie priestoru a súborov vo vašom počítači, naučme sa, ako tieto zručnosti zvládajú programy!

Typy klasifikácie

Na začiatku je dôležité zdôrazniť, že nie všetky klasifikácie sú rovnaké. Preto najskôr zistíme, aké klasifikácie existujú a čo ich charakterizuje. Príklady triedenia papierov alebo triedenia pošty sú **príkladmi binárnej klasifikácie**, pretože máme len dve skupiny: papiere, ktoré sa majú vyhodiť, a papiere, ktoré sa majú uchovať, t. j. žiaduca a nežiaduca pošta. Skupiny vo svete strojového učenia sa nazývajú **triedy**, takže budeme naďalej používať tento termín. Aby sme mohli rozlišovať medzi triedami, priraďujeme im názvy, ktoré približne vystihujú ich skutočný obsah. Napríklad „papierové lístky" a „nevyžiadaná pošta" sú dostatočne jasné názvy. Názvy sa často špecifikujú pomocou štítkov, ktoré sa objavujú v dátovom súbore, na ktorý sa klasifikačná úloha vzťahuje.

Ak máme viac ako dve triedy, hovoríme o úlohe **klasifikácie viacerých tried**. Takou úlohou je napríklad triedenie fotografií podľa udalostí, kde každá udalosť môže predstavovať jednu triedu. Môžeme vytvoriť tri adresáre, t. j. tri triedy, dať im názvy „výlet", „narodeniny tety" a „cesta" a potom každú z fotografií priradiť k jednej z týchto tried tak, že ju umiestnime do príslušného adresára.

Môžeme uvažovať o rôznych typoch klasifikácie na základe kritérií príslušnosti. Napríklad jeden e-mail môže byť buď žiaduci, alebo nežiaduci, nemôže patriť súčasne do triedy žiaducich aj nežiaducich e-mailov. Podobne je to s fotografiami a triedami, ktoré sme predstavili. Na druhej strane, jeden novinový článok môže byť súčasne o téme kultúry, cestovania a jedla, takže ho môžeme priradiť k väčšiemu počtu tried – tej, ktorá reprezentuje kultúru, tej, ktorá reprezentuje cestovanie, a tej, ktorá reprezentuje jedlo. Keďže v tomto prípade majú inštancie viac vlastností, t. j. štítkov, tento typ **klasifikácie** nazývame **viacštítková klasifikácia.** Hoci je veľmi zaujímavá a užitočná, viac sa ňou nebudeme zaoberať, ale zameriame sa na binárnu a viac triedovú klasifikáciu.

K akému typu klasifikácie podľa vás patria nasledujúce úlohy:

- triedenie odpadu na recykláciu,
- určenie správnosti programu,
- určenie jazyka dokumentu,
- kontrola platnosti bankovej transakcie.
- navrhovanie ďalších slov pri písaní SMS správy

Klasifikácia z hľadiska strojového učenia

Keď uvažujeme o úlohe klasifikácie z hľadiska strojového učenia, zaujímajú nás diskrétne mapovania, t. j. mapovania, ktoré môžu priradiť jednu z konečných hodnôt vstupným premenným. Najčastejšie je počet tried menší, vyjadrený jednociferným číslom, ale môžete si spomenúť aj na súbor *ImageNet* a súťaž v klasifikácii obrázkov, v ktorej sa používa 1000 tried. Premenné, ktoré môžu nadobúdať konečný počet hodnôt, sa nazývajú kategorické, takže môžeme hovoriť o klasifikácii. Je to ako mapovanie, ktoré sa vyznačuje kategorickou cieľovou premennou.

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

Príklad diskrétnej funkcie.

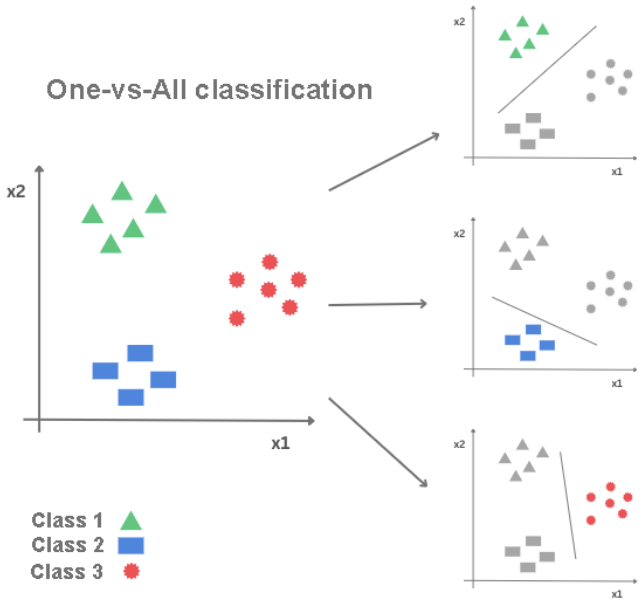
Keďže klasifikácia je úloha strojového učenia pod dohľadom, súbor údajov použitý na trénovanie modelu obsahuje páry vstupov a očakávaných výstupov. Vstupy môžu byť obrázky, textové správy alebo tabuľkové údaje a na ich prípravu sa vzťahujú všetky pokyny uvedené v lekcii o príprave súboru údajov. Výstupom je vždy názov triedy. Hoci sme zaviedli názvy tried, aby sme uľahčili sledovanie klasifikačnej úlohy, keď sa dostaneme k časti prípravy údajov, musíme ich tiež transformovať na číselné hodnoty. Tu sa môžeme riadiť prípravami, ktoré sme diskutovali pre prácu s kategorickými atribútmi: mapovanie súboru hodnôt alebo *one-hot* kódovanie.

V prípade binárnej klasifikácie zvyčajne mapujeme názvy tried na hodnoty 0 a 1. Napríklad výskyt názvu triedy „junk mail" je nahradený hodnotou 0 a výskyt názvu „junk mail" hodnotou 1. Často sa hovorí, že inštancie, ktoré majú označenie 0, patria do **negatívnej triedy**, a inštancie, ktoré majú označenie 1, patria **do pozitívnej triedy.**

Pokiaľ ide o viac triednu klasifikáciu, *na prípravu cieľovej premennej* používame *one-hot kódovanie.* Napríklad pri úlohe triedenia fotografií podľa udalostí transformujeme výstupy na vektory dĺžky tri, pretože máme presne tri triedy: „výlet", „narodeniny tety" a „cesta". Ďalej priradíme každej z týchto hodnôt vektor, ktorý má jednotku na príslušnej pozícii a nulu na všetkých ostatných pozíciách. To budú v poradí hodnoty (1, 0, 0), (0, 1, 0) a (0, 0, 1). Tu je dôležité dôsledne dodržiavať zvolené poradie tried.

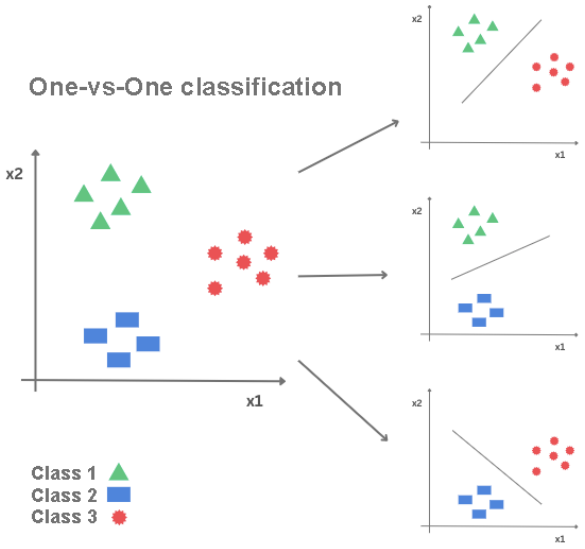
Nižšie sa zoznámime s dvoma algoritmami používanými na binárnu klasifikáciu. Problém viacnásobnej klasifikácie možno riešiť pomocou špeciálne navrhnutých algoritmov, ale aj pomocou viacerých súvisiacich binárnych klasifikátorov. Predstavíme si dve takéto techniky nazývané „jeden proti všetkým" a „jeden na jedného".

Predstavme si, že máme tri triedy: červenú, zelenú a modrú. Prístup „jeden proti všetkým" znamená, že sa musíme naučiť tri binárne klasifikátory: jeden, ktorý odlišuje zelenú triedu od ostatných (zjednotenie červenej a modrej triedy), jeden, ktorý odlišuje modrú triedu od ostatných (zjednotenie zelenej a červenej triedy), a jeden, ktorý odlišuje červenú triedu od ostatných (zjednotenie zelenej a modrej triedy). Keď potrebujeme klasifikovať novú inštanciu, spustíme každý z troch binárnych klasifikátorov a na získané výsledky uplatníme princíp najvyššej spoľahlivosti: inštancia sa pripojí k triede, ktorej klasifikátor je najbezpečnejší. Čoskoro uvidíme, ako sa hodnotí bezpečnosť klasifikátora.



Klasifikácia jeden proti všetkým

Znovu si predstavte, že máme tri triedy: červenú, zelenú a modrú. Prístup jeden k jednému zahŕňa trénovanie binárnych klasifikátorov, ktoré dokážu rozlíšiť každú z dvojíc tried: červenú a zelenú, zelenú a modrú a červenú a modrú. Všeobecne platí, že ak máme n tried, počet binárnych klasifikátorov, ktoré musíme trénovať, je $\frac{n\cdot(n-1)}{2}$. Keď potrebujeme klasifikovať novú inštanciu, spustíme každý z naučených klasifikátorov a na získané výsledky aplikujeme princíp väčšinového hlasovania: inštancia sa pripojí k triede, pre ktorú hlasuje najväčší počet klasifikátorov.

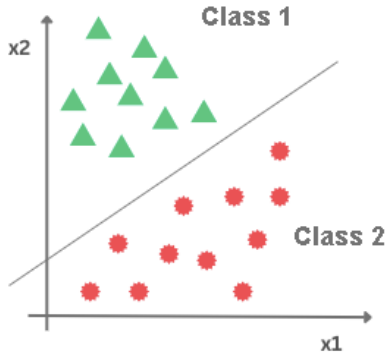


Klasifikácia jeden proti jednému

3.6 Logistická regresia

Logistická regresia je dobre známy algoritmus používaný na vytváranie binárnych klasifikačných modelov. Okrem toho, že vie, do akej triedy inštancia patrí, vypočíta aj pravdepodobnosť, že do tej triedy patrí.

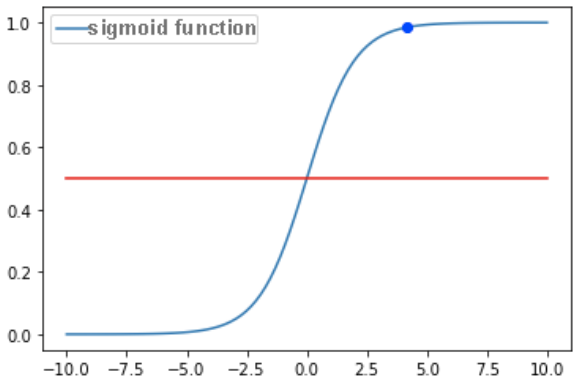
Predstavme si, že máme súbor údajov s dvoma atribútmi, X_1 a X_2 , a že inštancie tohto súboru sú zobrazené ako na obrázku nižšie. Na osi x je znázornený atribút X_1 , na osi y atribút X_2 a farba bodov označuje triedu, do ktorej každá z týchto inštancií patrí. Určite súhlasíte, že lineárny model, ktorý určuje priamku v rovine, by nám mohol pomôcť vyriešiť problém klasifikácie oddelením tried – jednej pod touto priamkou a druhej nad ňou. Aby sme mohli dospieť k takémuto záveru, využijeme sigmoidnú funkciu.



Sigmoidná funkcia je populárna funkcia v oblasti strojového učenia. Je určená rovnicou

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

Jej graf vyzerá ako na obrázku nižšie.



Graf sigmoidnej funkcie

Hneď si môžeme všimnúť, že táto funkcia má rozsah hodnôt od 0 do 1. Čím menšie sú hodnoty x, tým je hodnota tejto funkcie bližšie k 0 a podobne, čím väčšia je hodnota x, tým je hodnota sigmoidnej funkcie bližšie k 1. Pre x = 0 je hodnota sigmoidnej funkcie 0,5. Ak túto hodnotu vyhlásime za prahovú hodnotu a zavedieme pravidlá:

1. Ak je hodnota sigmoidnej funkcie väčšia alebo rovná 0,5, priraďte x k pozitívnej triede
2. Ak je hodnota sigmoidnej funkcie menšia ako prahová hodnota 0,5, priraďte x k negatívnej triede

dostaneme funkciu vhodnú pre klasifikačnú úlohu.

Zdá sa nám tiež, že čím vyššie sú hodnoty x, tým presvedčivejšie je rozhodnutie priradiť x k pozitívnej triede, pretože výrazne prekračujeme prahovú hodnotu. Zdá sa tiež, že čím menšie sú hodnoty x, tým je pravdepodobnejšie rozhodnutie priradiť x k negatívnej triede, pretože sme výrazne pod prahovou hodnotou. Pre hodnoty x, ktoré sú okolo nuly, sú tieto argumenty slabšie. Preto sigmoidná funkcia môže byť tiež spojená s interpretáciou pravdepodobnosti príslušnosti k triede.

Ak spojíme sigmoidnú funkciu a rovnicu lineárneho modelu, dostaneme rovnicu logistického regresného modelu, ktorá vo všeobecnosti je

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

Argumenty$X_1, X_2, \dots, X_n$ označujú atribúty v dátovom súbore, pričom ich hodnoty uu sa pohybujú od 0 do 1 a, ako sme videli, majú zmysel pre klasifikačnú úlohu. K tejto rovnici môžeme pridať aj nasledujúcu geometrickú interpretáciu: dáta sú klasifikované buď pod alebo nad „čiarou", ktorú určuje rovnica lineárneho vzťahu, ktorú sme si pôvodne predstavili.

Ak máme presne jeden atribút, „správna" strana, na ktorú sa odvolávame, je skutočná vec. Ak máme presne dva atribúty, „správna" strana je v skutočnosti plochá v priestore. Ak máme viac ako dva atribúty, sú „pravdivé", v matematickom jazyku, hyperplochy.

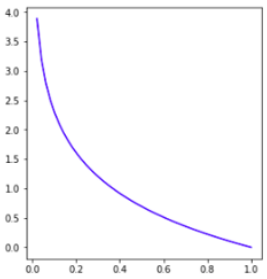
Krížová entropia

Chybová funkcia, ktorá charakterizuje logistickú regresiu, sa nazýva **krížová entropia**. Najprv si vysvetlíme intuíciu, ktorá stojí za touto funkciou, a potom sa zoznámime s jej matematickou formou.

Povedali sme, že hodnotu vypočítanú logistickým regresným modelom interpretujeme ako pravdepodobnosť príslušnosti k jednej z tried a že sa riadime pravidlom, že ak táto hodnota prekročí prah 0,5, interpretujeme ju ako príslušnosť k pozitívnej triede, a ak je táto hodnota menšia ako 0,5, interpretujeme ju ako príslušnosť k negatívnej triede. Ak je hodnota pravdepodobnosti 0,5, interpretuje sa ako príslušnosť k pozitívnej triede.

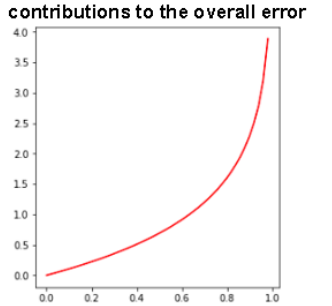
Vypočítame chybovú funkciu na trénovacej sade. V nej vieme pre každý prípad, aké sú presné charakteristiky, takže ich môžeme vždy porovnať s charakteristikami, ktoré vypočítal, t. j. pripojil som sa k modelu.

Predpokladajme, že pre tri inštancie patriace do pozitívnej triedy logistický regresný model vypočítal hodnoty 0,94, 0,56 a 0,3. V prvom prípade je hodnota blízka jednotke, takže naznačuje určité rozhodnutie modelu. V druhom prípade je táto hodnota menšia a bližšia k prahu klasifikácie, ale dostatočná na dobré rozhodnutie modelu. V treťom prípade je hodnota pod prahom a spôsobila by, že model by urobil chybu. Pri navrhovaní chybovej funkcie chceme viac penalizovať výpočty modelu, ktoré sa viac odchyľujú od hodnoty 1 pre pozitívne inštancie, t. j. aby ich príspevok k celkovej chybe modelu bol väčší. Jednou z takých funkcií, ktorá spĺňa požadovanú vlastnosť, je$-log(x)$, ktorej graf je zobrazený na obrázku nižšie. Potrebujeme znamienko mínus, aby chyba mala kladnú hodnotu, pretože logaritmus je záporný pre hodnoty argumentu funkcie, ktoré sú od 0 do 1. Na grafe môžeme tiež vidieť, že hodnoty funkcie sú malé pre argumenty bližšie k 1, t. j. že hodnoty funkcie sú väčšie pre argumenty, ktoré sú bližšie k nule. Teraz budú príspevky k celkovej chybe extrahovaných inštancií v poradí$-log(0.94) = 0.062, -log(0.56) = 0.579, -log(0.3) = 1.203$, čo je presne pomer veľkostí, ktorý sme chceli. Môžeme ich tiež zaznamenať do tabuľky, tak ako sme to urobili v úlohe lineárnej regresie. Do prvého stĺpca umiestnime značku triedy (presnú hodnotu), do druhého stĺpca pravdepodobnosť p vypočítanú modelom a do tretieho stĺpca zadáme hodnotu$-log(p)$. Všimnite si, že názov stĺpca je$-y * log(p)$, ale keďže$y = 1$ je to rovnaké ako$-log(p)$.



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

Teraz vyberme tri prípady negatívnej triedy a diskutujme o očakávaniach, ktoré máme od funkcie chyby v ich prípade. Nech sú pravdepodobnosti vypočítané logistickým regresným modelom 0,03, 0,48 a 0,74. V prvom prípade je hodnota modelu blízka nule, čo naznačuje určité rozhodnutie patriť do negatívnej triedy. V druhom prípade je táto hodnota blízka klasifikačnému prahu, ale je pod ním, takže opäť stačí, aby model rozhodol o negatívnej triede. V treťom prípade je hodnota pravdepodobnosti nad prahom, takže model sa pomýli a klasifikuje prípad ako pozitívny. Od funkcie chyby pre negatívne prípady očakávame, že ich podiel na celkovej chybe bude čo najvyšší, čím ďalej sú od nuly. Jednou z funkcií, ktorá spĺňa túto vlastnosť, je $-log(1-p)$ a jej graf je zobrazený na obrázku nižšie. Opäť používame funkciu so znamienkom mínus, aby bola hodnota chyby kladná. Teraz môžeme hodnoty tejto funkcie zapísať do tabuľky. Prvý stĺpec teraz obsahuje prípady s hodnotou 0, druhý stĺpec obsahuje pravdepodobnosti p, ktoré vypočítal model, a posledný stĺpec obsahuje hodnoty funkcií chyby $-log(1-p)$. Keďže y = 0 pre všetky prípady, symbol v názve stĺpca $-(1-y)*(1-p)$ nič nemení.



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

Celková hodnota funkcií krížovej entropie sa získa, keď sa spočítajú príspevky chýb všetkých kladných a všetkých záporných inštancií (podobne ako v prípade lineárnej regresie a problému strednej kvadratickej chyby). Zapíše sa to vo forme

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

kde prvý faktor sčituje príspevky chýb kladných inštancií a druhý faktor prispieva chybami záporných inštancií. Hodnota yi je presnou charakteristikou triedy v trénovacej sade a pi je pravdepodobnosť vypočítaná logistickým regresným modelom. Táto chyba sa nazýva **binárna krížová entropia**.

Hodnoty neznámych parametrov β v logistickom regresnom modeli sa nájdu výberom hodnoty parametra, pre ktorú je krížová chybová funkcia najmenšia. V tomto prípade nám môže pomôcť aj technika gradientového zostupu.

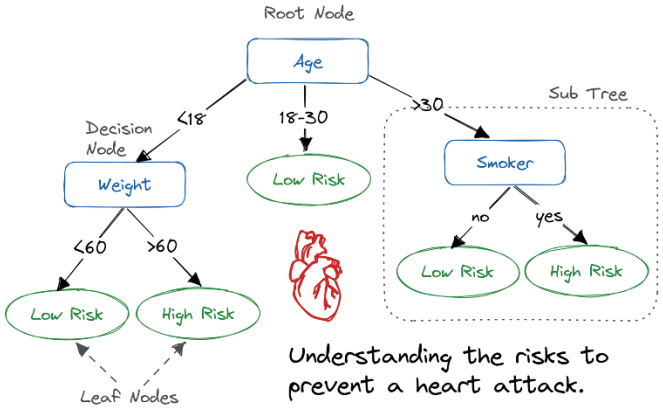
Teraz sa zoznámime s trochu odlišným klasifikačným algoritmom.

3.7 Rozhodovací strom

Rozhodovacie stromy sú univerzálne algoritmy strojového učenia, ktoré môžu vykonávať klasifikačné aj regresné úlohy a dokonca aj úlohy s viacerými výstupmi. Sú to veľmi výkonné algoritmy, schopné prispôsobiť sa komplexným dátovým súborom.

Rozhodovací strom je stromová štruktúra podobná diagramu, kde vnútorný uzol predstavuje vlastnosť (alebo atribút), vetva predstavuje rozhodovacie pravidlo a každý listový uzol predstavuje výsledok.

Najvyšší uzol v rozhodovacom strome sa nazýva koreňový uzol. Naučí sa rozdeľovať na základe hodnoty atribútu. Rozdeľuje strom rekurzívnym spôsobom, ktorý sa nazýva rekurzívne rozdeľovanie. Táto štruktúra podobná diagramu vám pomáha pri rozhodovaní. Je to vizualizácia podobná diagramu, ktorá ľahko napodobňuje ľudské myslenie. Preto sú rozhodovacie stromy ľahko zrozumiteľné a interpretovateľné.



Rozhodovací strom je algoritmus strojového učenia typu „biela skrinka". Má spoločnú internú logiku rozhodovania, ktorá nie je k dispozícii v algoritmoch typu „čierna skrinka", ako napríklad v neurónových sieťach. Jeho tréningový čas je rýchlejší v porovnaní s algoritmom neurónovej siete.

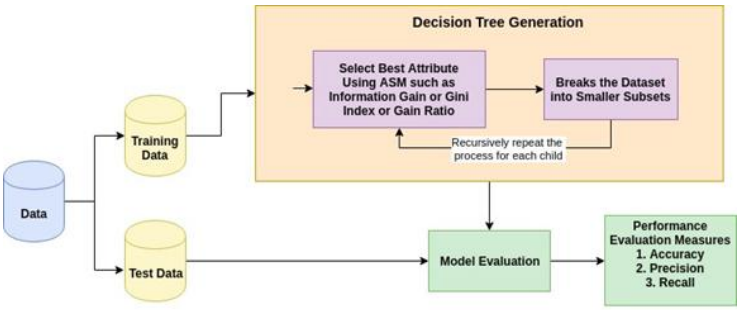
Časová zložitosť rozhodovacích stromov je funkciou počtu záznamov a atribútov v daných údajoch. Rozhodovací strom je metóda bez distribúcie alebo neparametrická metóda, ktorá nezávisí od predpokladov pravdepodobnostnej distribúcie. Rozhodovacie stromy dokážu spracovať vysokorozmerné údaje s dobrou presnosťou.

Ako funguje algoritmus rozhodovacieho stromu?

Základná myšlienka akéhokoľvek algoritmu rozhodovacieho stromu je nasledovná:

1. Vyberte najlepší atribút pomocou meradiel výberu atribútov (ASM) na rozdelenie záznamov.
2. Urobte z tohto atribútu rozhodovací uzol a rozdeľte dátovú sadu na menšie podmnožiny.
3. Začnite budovať strom opakovaním tohto procesu rekurzívne pre každé dieťa, kým sa nesplní jedna z podmienok:
    - Všetky tuple patria k rovnakej hodnote atribútu.
    - Nezostali žiadne ďalšie atribúty.

- Neexistujú žiadne ďalšie inštancie.



Merania výberu atribútov

Meranie výberu atribútov je heuristika na výber kritéria rozdelenia, ktoré rozdeľuje dáta najlepším možným spôsobom. Je tiež známe ako pravidlá rozdelenia, pretože nám pomáha určiť body zlomu pre tuple na danom uzle. ASM poskytuje hodnotenie každej vlastnosti (alebo atribútu) vysvetlením daného súboru dát. Ako atribút rozdelenia bude vybraný atribút s najlepším skóre. V prípade atribútu s nepretržitou hodnotou je potrebné definovať aj body rozdelenia pre vetvy. Najpopulárnejšie merania výberu sú informačný zisk, pomer zisku a Giniho index.

Zisk informácií

Claude Shannon vynájsol pojem entropia, ktorý meria nečistotu vstupnej množiny. Vo fyzike a matematike sa entropia označuje ako náhodnosť alebo nečistota v systéme. V informačnej teórii sa týka nečistoty v skupine príkladov. Informačný zisk je pokles entropie. Informačný zisk vypočíta rozdiel medzi entropiou pred rozdelením a priemernou entropiou po rozdelení dátovej sady na základe daných hodnôt atribútov. Algoritmus rozhodovacieho stromu ID3 (Iterative Dichotomiser) používa informačný zisk.

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

Kde Pi je pravdepodobnosť, že ľubovoľný tuple v D patrí do triedy Ci.

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

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

Kde:

- Info(D) je priemerné množstvo informácií potrebných na identifikáciu triedy tuple v D.
- |Dj|/|D| funguje ako váha j-tej partície.
- InfoA(D) je očakávaná informácia potrebná na klasifikáciu tuple z D na základe rozdelenia podľa A.

Atribút A s najvyšším informačným ziskom, Gain(A), je zvolený ako atribút rozdelenia v uzle N().

Pomer zisku

Zisk informácií je zaujatý voči atribútu s mnohými výsledkami. To znamená, že uprednostňuje atribút s veľkým počtom odlišných hodnôt. Napríklad zvážte atribút s jedinečným identifikátorom, ako je ID zákazníka, ktorý má nulovú hodnotu info(D) kvôli čistému rozdeleniu. To maximalizuje zisk informácií a vytvára zbytočné rozdelenie.

C4.5, vylepšenie ID3, používa rozšírenie informačného zisku známe ako pomer zisku. Pomer zisku rieši problém zaujatosti normalizáciou informačného zisku pomocou Split Info. Java implementácia algoritmu C4.5 je známa ako J48, ktorá je k dispozícii v nástroji na ťažbu dát WEKA.

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

Kde:

- $\frac{|D_j|}{|D|}$ pôsobí ako váha j-tej partície.
- v je počet diskrétnych hodnôt v atribúte A.
- Pomer zisku možno definovať ako

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

Atribút s najvyšším pomerom zisku je zvolený ako atribút rozdelenia (zdroj).

Giniho index

Ďalší algoritmus rozhodovacieho stromu CART (Classification and Regression Tree) používa Giniho metódu na vytvorenie bodov rozdelenia.

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

Kde pi je pravdepodobnosť, že tuple v D patrí do triedy Ci.

Giniho index zohľadňuje binárne rozdelenie pre každý atribút. Môžete vypočítať vážený súčet nečistoty každej partície. Ak binárne rozdelenie atribútu A rozdelí dáta D na D1 a D2, Giniho index D je:

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

V prípade atribútu s diskrétnymi hodnotami sa ako atribút rozdelenia vyberie podmnožina, ktorá poskytuje minimálny Giniho index pre daný atribút. V prípade atribútov s kontinuálnymi hodnotami je stratégiou vybrať každú dvojicu susedných hodnôt ako možný bod rozdelenia a ako bod rozdelenia sa vyberie bod s menším Giniho indexom.

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

Atribút s minimálnym Giniho indexom sa vyberie ako atribút rozdelenia.

Vytváranie klasifikátora rozhodovacieho stromu v Scikit-learn

Importovanie požadovaných knižníc

Najskôr načítajme potrebné knižnice.

```
# Načítanie knižníc
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import klasifikátora rozhodovacieho stromu
z sklearn.model_selection import train_test_split # Import funkcie train_test_split
from sklearn import metrics #Importujte modul scikit-learn metrics pre výpočet presnosti
```

Načítanie údajov

Najskôr načítajme požadovaný súbor údajov o cukrovke u indiánov kmene Pima pomocou funkcie read CSV v pandas. Súbor údajov Kaggle si môžete stiahnuť a sledovať postup.

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

	pregnant	glukóza	bp	koža	inzulín	BMI	rodokmeň	vek	označenie
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

Výber vlastností

Na pochopenie výkonu modelu je dobré rozdeliť súbor údajov na trénovací súbor a testovací súbor.

Rozdelíme dátovú sadu pomocou funkcie train_test_split(). Musíte zadať tri parametre: vlastnosti, cieľ a veľkosť test_set.

```
# Rozdelenie dátového súboru na trénovací súbor a testovací súbor
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
# 70 % trénovanie a 30 % testovanie
```

Vytvorenie modelu rozhodovacieho stromu

```
# Vytvorenie objektu klasifikátora rozhodovacieho stromu
clf = DecisionTreeClassifier()
# Trénovanie klasifikátora rozhodovacieho stromu
clf = clf.fit(X_train,y_train)
#Predikcia odpovede pre testovaciu dátovú sadu
y_pred = clf.predict(X_test)
```

Hodnotenie modelu

Odhadnime, ako presne dokáže klasifikátor alebo model predpovedať typ kultivarov.
Presnosť sa dá vypočítať porovnaním skutočných hodnôt testovacej sady a predpovedaných hodnôt.

```
# Presnosť modelu, ako často je klasifikátor správny?
print("Presnosť:",metrics.accuracy_score(y_test, y_pred))
```

Presnosť: 0,6753246753246753

Dostali sme klasifikačnú mieru 67,53 %, čo sa považuje za dobrú presnosť. Túto presnosť môžete zlepšiť vyladením parametrov v algoritme rozhodovacieho stromu.

Vizualizácia rozhodovacích stromov

Na zobrazenie stromu v notebooku Jupyter môžete použiť funkciu *export_graphviz* z balíka Scikit-learn. Na vykreslenie stromu je potrebné nainštalovať aj balíky graphviz a pydotplus.

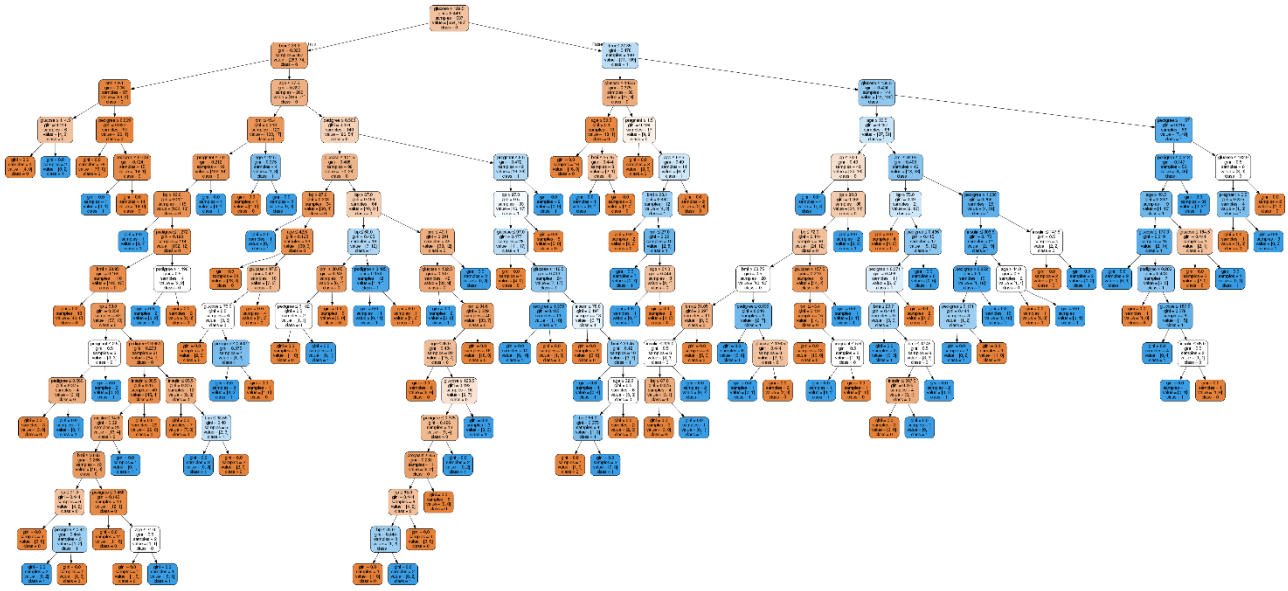
pip install graphviz

pip install pydotplus

Funkcia *export_graphviz* prevádza klasifikátor rozhodovacieho stromu do súboru dot a pydotplus prevádza tento súbor dot do formátu png alebo do formy, ktorú je možné zobraziť v Jupyter.

```
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())
```



V rozhodovacom strome má každý vnútorný uzol rozhodovacie pravidlo, ktoré rozdeľuje dáta. Gini, označovaný ako Giniho koeficient, meria nečistotu uzla. Uzol je čistý, ak všetky jeho záznamy patria do tej istej triedy. Takéto uzly sa nazývajú listové uzly.

V tomto prípade je výsledný strom neupravený. Takýto neupravený strom je nevysvetliteľný a ťažko pochopiteľný. V ďalšej časti ho optimalizujeme pomocou úpravy.

Optimalizácia výkonu rozhodovacieho stromu

- **kritérium: voliteľné (predvolené = „gini") alebo Vyberte mieru výberu atribútu.** Tento parameter nám umožňuje použiť rôzne miery výberu atribútu. Podporované kritériá sú „gini" pre Giniho index a „entropia" pre informačný zisk.

- **splitter: reťazec, voliteľné (predvolené = „best") alebo stratégia rozdelenia.** Tento parameter nám umožňuje vybrať stratégiu rozdelenia. Podporované stratégie sú „best" na výber najlepšieho rozdelenia a „random" na výber najlepšieho náhodného rozdelenia.

- **max_depth : int alebo None, voliteľné (predvolené=None) alebo Maximálna hĺbka stromu.** Maximálna hĺbka stromu. Ak je None, uzly sa rozširujú, kým všetky listy neobsahujú menej ako min_samples_split vzoriek. Vyššia hodnota maximálnej hĺbky spôsobuje pretrénovanie a nižšia hodnota spôsobuje podtrénovanie (Zdroj).

V Scikit-learn sa optimalizácia klasifikátora rozhodovacieho stromu vykonáva iba predbežným prerezávaním. Maximálna hĺbka stromu sa môže použiť ako kontrolná premenná pre predbežné prerezávanie. V nasledujúcom príklade môžete nakresliť rozhodovací strom na rovnakých údajoch s max_depth=3. Okrem parametrov predbežného prerezávania môžete vyskúšať aj iné merania výberu atribútov, ako napríklad entropiu.

```
# Vytvorte objekt klasifikátora rozhodovacieho stromu
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
# Trénovanie klasifikátora rozhodovacieho stromu
clf = clf.fit(X_train,y_train)
#Predikcia odpovede pre testovaciu dátovú sadu
y_pred = clf.predict(X_test)
# Presnosť modelu, ako často je klasifikátor správny?
print("Presnosť:",metrics.accuracy_score(y_test, y_pred))
```

Presnosť: 0,7705627705627706

No, miera klasifikácie sa zvýšila na 77,05 %, čo je lepšia presnosť ako v predchádzajúcom modeli.

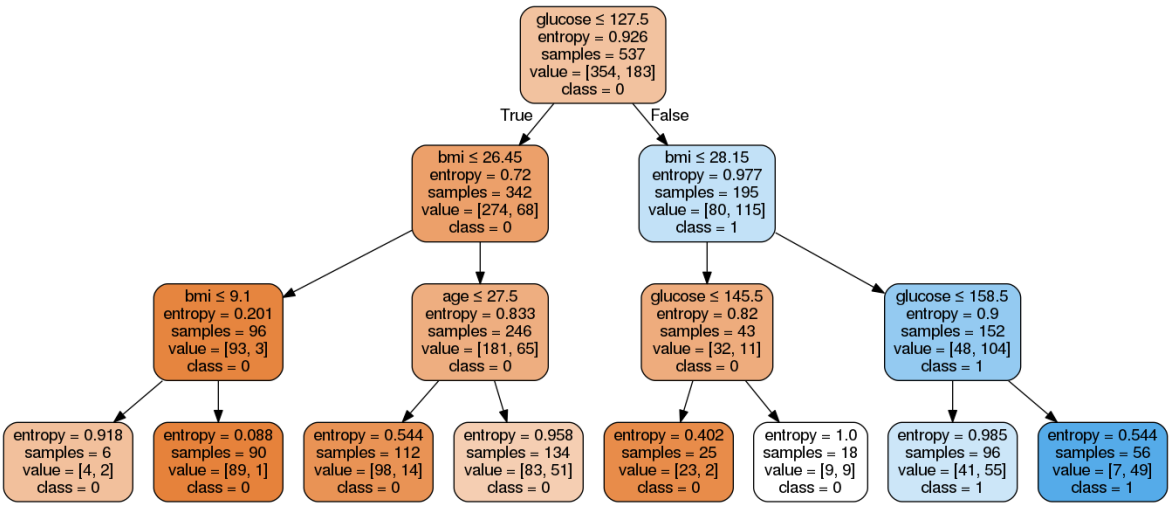
Vizualizácia rozhodovacích stromov

Pomocou nasledujúceho kódu urobíme náš rozhodovací strom trochu zrozumiteľnejším:

```
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())
```

Tu sme vykonali nasledujúce kroky:

- Importovali sme potrebné knižnice.
- Vytvorili sme objekt `StringIO` s názvom `dot_data,` ktorý obsahuje textové znázornenie rozhodovacieho stromu.
- Exportovali sme rozhodovací strom do formátu `dot` pomocou funkcie `export_graphviz` a zapísali sme výstup do vyrovnávacej pamäte `dot_data`.
- Vytvorili sme grafový objekt `pydotplus` z reprezentácie rozhodovacieho stromu vo formáte `dot` uloženej v vyrovnávacej pamäti `dot_data`.
- Zapísali sme vygenerovaný graf do súboru PNG s názvom „diabetes.png“.
- Zobrazenie generovaného PNG obrázku rozhodovacieho stromu pomocou objektu `Image` z modulu `IPython.display`



Ako vidíte, tento prispôsobený model je menej zložitý, zrozumiteľnejší a ľahšie pochopiteľný ako predchádzajúci graf modelu rozhodovacieho stromu.

Výhody rozhodovacieho stromu

- Rozhodovacie stromy sa ľahko interpretujú a vizualizujú.
- Môžu ľahko zachytiť nelineárne vzory.
- Vyžadujú menej predspracovania údajov zo strany používateľa, napríklad nie je potrebné normalizovať stĺpce.
- Môžu sa použiť na inžinierstvo vlastností, ako je predikcia chýbajúcich hodnôt, vhodné pre výber premenných.
- Rozhodovací strom nemá žiadne predpoklady o distribúcii kvôli neparametrickej povahe algoritmu. (Zdroj)

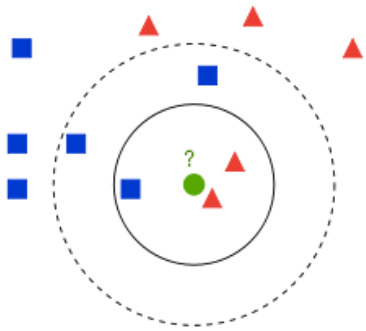
Nevýhody rozhodovacieho stromu

- Je citlivý na šumové údaje. Môže nadmerne prispôsobovať šumové údaje.
- Malá variácia (alebo rozptyl) v údajoch môže viesť k odlišnému rozhodovaciemu stromu. To je možné znížiť pomocou algoritmov bagging a boosting.
- Rozhodovacie stromy sú ovplyvnené nevyváženým súborom údajov, preto sa odporúča vyvážiť súbor údajov pred vytvorením rozhodovacieho stromu.

3.8 Algoritmus K-najbližších susedov (kNN)

Existujú aj neparametrické modely strojového učenia. Model získaný pomocou algoritmu k-najbližších susedov je práve taký. Zistime, ako funguje!

Nech naša trénovacia množina pozostáva z párov čísel(x_1, x_2) a zodpovedajúcich názvov tried. Páry môžu byť reprezentované ako body v rovine, kde prvá súradnicax_1 označuje hodnotu na osi x a druhá súradnicax_2 označuje hodnotu na osi y. V praxi sú hodnotyx_1 ax_2 vždy spojené s niektorými špecifickými atribútmi, napríklad teplotou a vlhkosťou, ale teraz ich môžeme považovať za nejaké všeobecné hodnoty. Každá dvojica čísel patrí do jednej z dvoch tried: červené trojuholníky alebo modré štvorce. Keďže existujú len dve triedy, môžete predpokladať, že ide o binárnu klasifikáciu. Teraz si predstavte, že zelený kruh predstavuje novú inštanciu, novú dvojicu čísel, pre ktorú musíme určiť, do ktorej triedy patrí: či je to červený trojuholník alebo modrý štvorec.



Trénovacia sada

Algoritmus k-najbližších susedov je klasifikačný algoritmus, ktorý hovorí, že najprv stanovíme počet susedov (okolité inštancie) k na konkrétnu hodnotu a potom určíme, koľko z k-najbližších susedov je červených a koľko modrých: červený sused je inštancia patriaca do červenej triedy a modrý sused je inštancia patriaca do modrej triedy. Napríklad, ak nastavíme číslo k na 3, tri najbližšie susedy zeleného kruhu sa nachádzajú vnútri plného kruhu. Sú tam dva červené trojuholníky a jeden modrý štvorec.

Algoritmus k-najbližších susedov ďalej hovorí, že nová inštancia, nový pár bodov, sa pridá do triedy početnejšieho suseda: ak je červených susedov viac, hovoríme, že nová inštancia patrí do červenej triedy, a podobne, ak je modrých susedov viac, hovoríme, že nová inštancia patrí do modrej triedy. Môžete si to predstaviť ako príslovie „poznáš človeka podľa spoločnosti", ktoré platí aj vo svete strojového učenia.

V našom príklade, keď je hodnota k stanovená na 3, dochádzame k záveru, že by sme mali zelený kruh priradiť k červenej triede, pretože máme dvoch červených susedov a jedného modrého suseda.

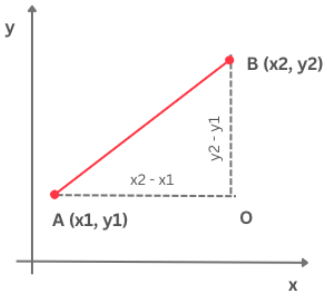
Pozrime sa, čo sa stane, ak nastavíme číslo k na 5. Na obrázku je toto susedstvo znázornené prerušovanou kružnicou. Keďže teraz máme tri modré štvorce a dva červené trojuholníky, záverom by bolo, že zelený kruh by mal byť priradený k modrej triede.

Táto časť je spojená s cvičením 8 a Jupyter Notebook 08-k-nearest_neighbors.ipynb. Ak chcete pokračovať v prezeraní obsahu, kliknite na odkaz a potom na tlačidlo „Open in Colab" (Otvor v *Google Collab*), aby sa obsah otvoril v *Google Collab*. Ak si prezeráte zošity na svojom lokálnom počítači, vyhľadajte zošit s rovnakým názvom medzi obsahom a spustite ho. Podrobnejšie pokyny nájdete v časti *Praktická zóna* a v *lekcii Jupyter Notebook Practice Lesson.*

Sprievodný materiál obsahuje spomínaný súbor bodov a aplikáciu, v ktorej môžete preskúmať, čo sa stane, ak zvolíte inú hodnotu čísla k. Keďže algoritmus musí rozhodnúť, ktorí susedia sú početnejší, je rozumné zvoliť nepárne hodnoty čísla k.

Všimnite si, že okrem počtu susedov k závisí výsledok algoritmu aj od toho, ako meriame vzdialenosti k susedom! Aby sme našli najbližších susedov, musíme nejako zmerať vzdialenosť k nim.

Doteraz sme sa stretli s vzdialenosťou nazývanou euklidovská vzdialenosť. Euklidovská vzdialenosť medzi bodmi A a B sa počíta ako dĺžka úsečiek spájajúcich body A a B. Napríklad pre body A =(0,0) a B =(3,4) sa euklidovská vzdialenosť počíta ako $\sqrt{(3-0)^2+(4-0)^2}=5$



Euklidovská vzdialenosť

Existuje aj mnoho iných vzdialeností. Napríklad vás môže zaujať manhattanská vzdialenosť. Na rozdiel od euklidovskej vzdialenosti, ktorá počíta „preponu" trojuholníka definovaného bodmi A, B a O (ak vychádzame z predchádzajúceho obrázku), manhattanská vzdialenosť počíta súčet „odsekov" tohto trojuholníka. Pre body A a B by hodnota manhattanskej vzdialenosti bola $|3-0|+|4-0|=7$.

Ktorú vzdialenosť zvolíme, závisí od povahy úlohy a významu atribútov, s ktorými pracujeme. Vo všeobecnosti môžeme vyskúšať viacero vzdialeností a vybrať tú, s ktorou dosiahneme najlepšie výsledky. O tom si povieme neskôr. Je dôležité poznamenať, že funkcia musí spĺňať určité matematické vlastnosti, aby mohla byť vyhlásená za vzdialenosť, takže nie každá funkcia môže byť užitočná.

Rovnako ako iné algoritmy strojového učenia, algoritmus x-najbližších susedov je trénovaný na trénovacej sade. Je zaujímavé poznamenať, že fáza učenia v tomto algoritme je v skutočnosti zredukovaná len na ukladanie dátového súboru. V iných algoritmoch, ako je lineárna regresia alebo logistická regresia, sme videli, že v tejto fáze sa hodnoty niektorých parametrov, ktoré sa objavujú v modeli, počítajú hľadaním minimálnej chyby funkcie. Algoritmus x-najbližších susedov taký nie je. Mapovanie, ktoré sa učíme, nie je o konkrétnej funkcii, ale o samotných dátach a krokoch, ktoré je potrebné podniknúť. Preto sa modely, ktoré majú túto vlastnosť, bežne nazývajú **neparametrické modely**.

Algoritmus k-najbližších susedov vykonáva všetku prácu počas aplikácie, t. j. rozhoduje, do akej triedy nová inštancia patrí. Keď potrebujeme klasifikovať novú inštanciu, najskôr vypočítame vzdialenosť novej inštancie od všetkých inštancií v trénovacom dátovom súbore. Ďalej tieto vzdialenosti zoradíme od najmenšej po najväčšiu. Zachováme prvých k vzdialeností (pretože sú to vzdialenosti k k najbližším susedom) a vyberieme inštancie z trénovacieho súboru, na ktoré sa vzťahujú. Naďalej sledujeme, čo sa deje v priestore ich orientačných bodov, a hľadáme najpočetnejšie orientačné body, t. j. najväčšiu triedu. Ako sme videli v úvodnom príklade, nová inštancia by mala byť priradená k triede, ktorá je najpočetnejšia.

Tento algoritmus je ľahko implementovateľný, takže si vyhrňme rukávy a pustime sa do práce!

Predstavme si, že pracujeme so súborom údajov, ktoré sme doteraz používali, a že každá inštancia má formu (x1, x2), kde značka je hodnota 0 pre červenú alebo 1 pre modrú.

Na meranie vzdialenosti medzi inštanciami použijeme funkciu euclidean_distance, ktorá je definovaná nasledujúcim blokom kódu:

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

Samotný algoritmus x-najbližších susedov je reprezentovaný nasledujúcim blokom kódu:

```
def kNN(k, instances, new_instance, classes={0: 'red', 1: 'blue'}):
# najskôr vypočítajte vzdialenosti medzi novou inštanciou a všetkými inštanciami v dátovom súbore
  vzdialenosti = [euclidean_distance(inštancia, nová_inštancia) pre inštanciu v inštanciách]
  # potom zoradiť vzdialenosti, extrahovať k najmenších a zodpovedajúce inštancie
  # deklarujte ich ako susedov
  susedovia = np.argsort(vzdialenosti)[0:k]
  # potom prečítajte štítky susedov a spočítajte ich
  označenia_susedov = [prípady[sused][2] pre suseda v susedoch]
  počet_označení = np.bincount(označenia_susedov)
  # označenie novej inštancie bude označením najčastejšieho suseda
  označenie = np.argmax(počet_označení)
  vráť triedy[označenie]
```

V ňom, ako sme už spomínali, vykonávame nasledujúce kroky:

1. Vypočítame vzdialenosť novej inštancie od všetkých inštancií v dátovom súbore.
2. Potom ich umiestnime uprostred noci a potom ich umiestnime do tmy.
3. A my sme tí, ktorí majú právo byť susedmi.
4. V súbore izolovaných susedov spočítame najpočetnejších
5. Dospeli sme k záveru, že nová inštancia patrí do triedy najpočetnejšieho suseda.

3.9 Hyperparametre

Videli sme, že v algoritme x-najbližších susedov je potrebné vopred stanoviť hodnotu čísla k a že rôzne voľby vedú k rôznym záverom. Ako vieme, ktorú hodnotu zvoliť? Táto otázka sa týka všetkých ostatných algoritmov strojového učenia, v ktorých sa objavujú niektoré hodnoty, ktoré musíme vopred definovať. Takéto hodnoty sa nazývajú **hyperparametre** alebo **metaparametre.**

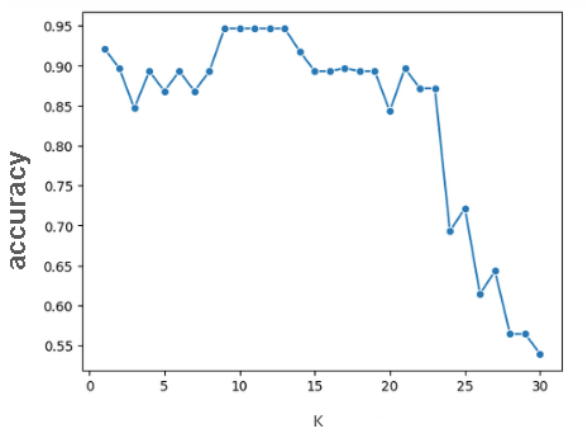
Spomenuli sme, že pri delení dátového súboru vždy vyberáme trénovací súbor, testovací súbor a validačný súbor. Validačný súbor sme doteraz nepoužili. V skutočnosti ho potrebujeme vždy, keď sa v našom algoritme učenia nachádzajú nejaké hyperparametre, ktorých optimálnu hodnotu musíme určiť. Príbeh, ktorý vám budeme rozprávať, platí pre všetky algoritmy, ale budeme naďalej používať algoritmus k-najbližších susedov.

Vráťme sa k otázke, ako vybrať najlepšiu hodnotu hyperparametra k. Je prirodzené myslieť si: vyskúšame viacero hodnôt, napríklad všetky čísla od 1 do 10, a potom vyberieme najlepšiu hodnotu! Urobíme to, ale budeme veľmi opatrní pri skúšaní, ako dobrý je náš výber. Ak to urobíme na testovacej sade, porušíme zlaté pravidlo strojového učenia o prísnom oddelení testovacej sady a vývoja modelu: testovaciu sadu použijeme

na rozhodnutie, aká je najlepšia hodnota hyperparametra k , a potom, keď budeme trénovať model, opäť použijeme testovaciu sadu na vyhodnotenie, aký je dobrý! Určite súhlasíte, že to nedáva veľký zmysel!

Správne je postupovať takto: otestujeme, ktoré hodnoty hyperparametrov sú najlepšie na validačnej sade. Táto sada nezdieľa informácie ani s trénovacou sadou, ani s testovacou sadou, takže prispeje k objektívnosti našich záverov. Teraz, keď sme si to ujasnili, môžeme sa pustiť do práce a určiť najlepšiu hodnotu hyperparametra k.

Pre každú z hodnôt hyperparametra k, ktorú chceme otestovať, budeme samostatne trénovať model na trénovacej sade a vypočítať jeho mieru kvality na validačnej sade. V tomto prípade je to presné. Získané hodnoty môžeme graficky znázorniť tak, že umiestnime rôzne hodnoty parametra k na os x a hodnoty presnosti na os y. Hodnota hyperparametra k, pre ktorú získame najlepšiu hodnotu merania kvality na validačnej sade, je hodnota hyperparametra, ktorú hľadáme. V grafe sa to zvyčajne prejavuje ako oblasť, kde sú hodnoty najvyššie.



Demonštrácia presnosti modelu na validačnej sade

Na základe predchádzajúceho grafu vidíme, že optimálne hodnoty hyperparametra k sú v skutočnosti 9, 10, 11, 12 a 13, pretože všetky vedú k rovnakej, najvyššej presnosti modelu.

Podobné grafy možno nakresliť pre hodnoty hyperparametrov a chybové funkcie. Potom nastavíme rôzne hodnoty hyperparametra pozdĺž osi x a hodnoty chybovej funkcie pozdĺž osi y. Teraz je dôležité si všimnúť hodnoty hyperparametra, pre ktoré je chybová funkcia najmenšia.

Keď sa v učiacom algoritme nachádza viacero hyperparametrov, cieľom je nájsť najlepšiu kombináciu hyperparametrov. Určujeme ju tiež na základe validačnej sady sledovaním úspešnosti modelu a hľadaním kombinácie, ktorá poskytuje najlepšiu hodnotu meradla kvality (alebo rovnako sledovaním chyby modelu a hľadaním kombinácie, ktorá poskytuje najmenšiu hodnotu chyby). Problémom je, že tento proces môže byť pomerne pomalý a výpočtovo náročný pre veľký počet hyperparametrov: napríklad, ak chceme preskúmať 10 rôznych hodnôt k a 3 rôzne funkcie vzdialenosti, máme v skutočnosti 10 x 3 = 30 rôznych kombinácií, takže musíme trénovať a vyhodnocovať 30 rôznych modelov.

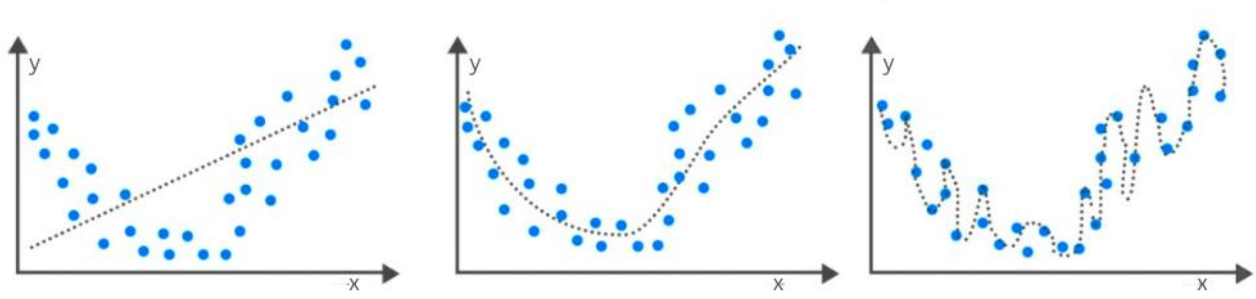
3.10 Generalizácia, nedostatočná adaptácia a nadmerná adaptácia

V tejto časti sa dozvieme o pojmoch generalizácia, nadmerná adaptácia a nedostatočná adaptácia, s ktorými sa často stretávame v oblasti strojového učenia.

Predstavte si, že Mike, Ana a Luka sa pripravujú na matematickú skúšku a všetci používajú rovnakú zbierku. Mike sa flákal a úlohy riešil len povrchne, Ana bola usilovná a celý týždeň sa dôkladne a komplexne pripravovala, zatiaľ čo Luka sa rozhodol úlohy naspamäť naučiť. Viete uhádnuť, kto v teste dosiahol najlepšie výsledky? Samozrejme, Ana!

Zbierku úloh si môžeme predstaviť ako abstraktný súbor údajov pozostávajúci zo vstupov (texty úloh) a výstupov (riešenia). Model strojového učenia sa, podobne ako Pera, môže naučiť len niekoľko spojení v údajoch a v praxi robiť veľa chýb. Takáto vlastnosť modelu sa nazýva **nedostatočné prispôsobenie**. Model môže tiež preháňať úroveň detailov, ako Luke, a stratiť schopnosť spracovávať niektoré nové údaje. Takáto vlastnosť modelu sa nazýva **nadmerné prispôsobenie.** Najlepšie by bolo, keby model prijal správne informácie a mohol, ako Anna, úspešne riešiť známe aj niektoré nové úlohy. Táto vlastnosť modelu sa nazýva **generalizácia.**

Príklad nedostatočnej adaptácie a nadmernej adaptácie možno ilustrovať na nasledujúcom obrázku. Predstavte si, že na osi x sú hodnoty atribútu, na osi y sú hodnoty cieľovej premennej a prerušovaná čiara znázorňuje model. Model naľavo nie je vzhľadom na usporiadanie bodov najlepšou voľbou, zdá sa príliš jednoduchý. Dáta vyzerajú skôr ako „sklo", takže lepší riešením by mohol byť štvorcový model, ktorý má túto formu. Vidíme ho na strednom obrázku. Na obrázku vpravo vidíme model, ktorý konzistentne sleduje každý bod v dátovom súbore a je k nemu úplne prispôsobený.



Príklad nedostatočnej adaptácie a nadmernej adaptácie

Úloha nájsť optimálny model a vyvážiť nedostatočné a nadmerné prispôsobenie nie je ľahká. Našťastie oblasť strojového učenia definuje protokoly a techniky, ktoré môžeme použiť na sledovanie každej z týchto situácií. Napríklad veľké rozdiely vo výkone modelu v trénovacom súbore a testovacom súbore naznačujú, že model bol prispôsobený. To je zvyčajne spôsobené výberom zložitejšieho modelu, ako je potrebné (ako na obrázku vpravo hore), alebo dlhším trénovaním modelu. Na druhej strane, nízke hodnoty meradiel kvality v trénovacom aj testovacom súbore naznačujú, že model sa z dát nenaučil dosť, je príliš jednoduchý (ako na obrázku vľavo hore) alebo potrebuje viac atribútov.

Dobrá generalizácia je vlastnosť, ktorá umožňuje úspešné uplatnenie modelov strojového učenia v praxi. Na ich trénovanie sa používa len malá časť dostupných údajov, a napriek tomu očakávame, že sa budú správne správať počas aplikácie a pri nových údajoch, s ktorými sa nikdy nestretli. Preto je dôležité, aby boli súbory

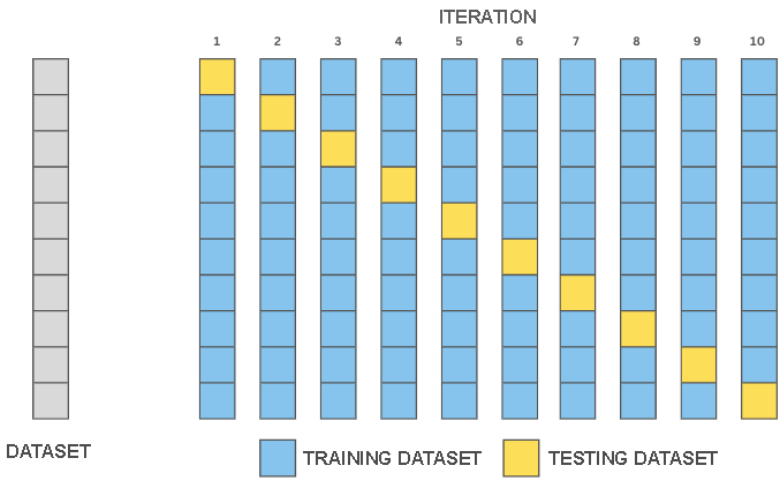
údajov reprezentatívne, t. j. aby boli dostatočne bohaté a rôznorodé, aby vyhovovali riešenému problému, ako aj starostlivé monitorovanie možných nedostatočných a nadmerných adaptácií modelu.

3.11 Validácia, krížová validácia

Už mnohokrát sme spomínali, že pred aplikáciou algoritmu strojového učenia sa údaje rozdelia na trénovaciu a testovaciu sadu (v prípade potreby pridáme aj validačnú sadu). Spomenuli sme tiež, že proces rozdelenia je náhodný. Možno ste sa zamýšľali nad tým, či by niektoré iné rozdelenia v porovnaní s tými, ktoré sme zvolili, viedli k odlišným výsledkom práce modelu. Možno práve pri konkrétnom rozdelení súboru údajov dostaneme optimistickejšie výsledky alebo naopak výrazne horšie. A to je akási úprava.

Ak to veľkosť dátových súborov a zvolené algoritmy dovoľujú, je žiaduce vykonať viacero rozdelení počiatočného dátového súboru na trénovací súbor a testovací súbor, aby každá inštancia v dátovom súbore mala možnosť byť nájdená v oboch súboroch. Jeden takýto postup, ktorý opíšeme, sa nazýva **krížová validácia**. V príklade použijeme algoritmus lineárnej regresie, ale príbeh je všeobecný a platí pre všetky algoritmy.

Rozdelíme dátový súbor na 10 častí, ako je znázornené na obrázku nižšie. V prvom kroku extrahujeme prvú časť testovacej sady a zvyšných deväť častí ponecháme na trénovanie. Aby ste to mohli ľahšie sledovať, testovacia sada je na obrázku vyfarbená žltou farbou a trénovacie sady sú vyfarbené modrou farbou. Teraz trénujeme prvý lineárny regresný model na trénovacej sade a vypočítame hodnotu jeho strednej kvadratickej chyby na testovacej sade. Výslednú hodnotu môžeme označiť ako $MSE_{(1.)}$ V druhom kroku oddelíme druhú časť testovacej sady a zvyšných deväť častí ponecháme na trénovanie. Na obrázku je teraz druhá časť vyfarbená žltou farbou a zvyšné časti sú vyfarbené modrou farbou. Preškolíme lineárny regresný model na trénovacej sade (teraz je to druhý model) a vypočítame hodnotu jeho strednej kvadratickej chyby na testovacej sade. Teraz označme túto hodnotu ako $MSE_{(2.)}$ Pokračujme v tomto procese, kým sa nedostaneme k poslednej, desiatej časti: teraz ju ponecháme ako testovú sadu a zvyšné časti použijeme na trénovanie modelu. Na nej trénujeme desiaty lineárny regresný model a potom vypočítame strednú kvadratickú chybu $MSE_{(10)}$ na testovacej sade.



Krížová validácia s 10 vrstvami

Keďže máme 10 rôznych rozdelení dátového súboru, máme aj 10 rôznych hodnôt strednej kvadratickej chyby. Priemer získaných hodnôt (MSE_1 + MSE_2 +... + MSE_{10})/10 v skutočnosti najlepšie ukazuje, ako sa náš model správa, a pomáha nám vyriešiť dilemy, ktoré sme mali na začiatku, pokiaľ ide o vplyv rozdelenia na úspešnosť

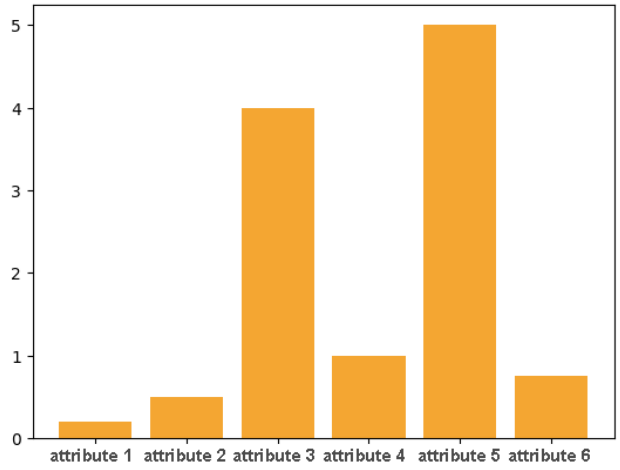
modelu. Nie je však celkom jasné, ktorý z 10 rôznych modelov, ktoré máme k dispozícii, by sme mali zvoliť. Ten s najmenšou chybou alebo nejaký iný? V skutočnosti by sme teraz mali vytrénovať nový model na celom súbore údajov a pokračovať v jeho používaní, priblížiť sa k jeho správaniu a ohodnotiť ho na základe správania každého z 10 vytrénovaných modelov.

Tento proces sa nazýva 10-vrstvová *krížová validácia (10-fold cross validation*). V praxi sa používajú aj 3- a 5-vrstvové delenia a výber závisí od veľkosti súboru údajov a typu použitých algoritmov. Existuje aj delenie, pri ktorom počet vrstiev zodpovedá počtu inštancií v súbore údajov, nazývané *leave-one-out krížová validácia*.

3.12 Regularizácia

Regularizácie sú ďalšou skupinou techník, ktoré možno použiť na kontrolu prispôsobenia modelov. Ich hlavným cieľom je zabrániť tomu, aby sa komplexné modely, ktoré nám pomáhajú naučiť sa bohatšiu sadu závislostí v údajoch, stali nadmerne prispôsobenými.

Regularizáciu predstavíme na príklade lineárneho regresného modelu. Predpokladajme, že sme model vytrénovali a získali sme hodnoty parametrov, ktorých grafické znázornenie vyzerá ako na obrázku.



Parametre, ktoré sú najväčšie (absolútne) z hľadiska svojej hodnoty, sú tiež najdôležitejšie pre predikcie modelu. Na obrázku sú to parametre, ktoré zodpovedajú atribútom 3 a 5, a ich hodnoty, ako vidíme, sú výrazne vyššie ako hodnoty ostatných parametrov. V tomto zmysle môžu tieto atribúty ignorovať vplyv ostatných atribútov na predikčné hodnoty, takže toto správanie modelu môžeme interpretovať ako formu prispôsobenia údajov.

Preto je do určitej miery žiaduce obmedziť hodnoty parametrov – chceme, aby sa model naučil parametre a aby odrážali vlastnosti údajov, ale zároveň chceme monitorovať ich hodnotu, aby sme zabránili nadmernému prispôsobeniu. Táto technika sa nazýva **regularizácia.** (niekde uvádzaná ako *regulácia*) V kontexte lineárnej regresie to môžeme urobiť pridaním súčtu štvorcov parametrov k strednej kvadratickej chybe modelu:

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

Hodnota λ vo výraze je hyperparametr, ktorý ovplyvňuje silu regularizácie. Ak je jej hodnota 0, regularizácia nebude mať žiadny účinok. Zadávaním niektorých hodnôt odlišných od nuly vyvažujeme učenie určené strednou kvadratickou chybou a prispôsobenie merané hodnotami súčtu štvorcov parametrov. Štvorec je tu z technických dôvodov, najprv aby sa zabránilo potlačeniu hodnôt koeficientov voči sebe a potom aby sa zachovali vlastnosti chybovej funkcie pre aplikáciu optimalizačného algoritmu. Takáto rozšírená forma lineárnej regresie doplnená o regularizačný termín sa nazýva ridge regresia.

O niečo neskôr sa vrátime k príbehu regularizácie, keď predstavíme neurónové siete. Sú to veľmi zložité modely, takže ich často možno prispôsobiť údajom. Uvidíme tiež, ako to môžeme sledovať.

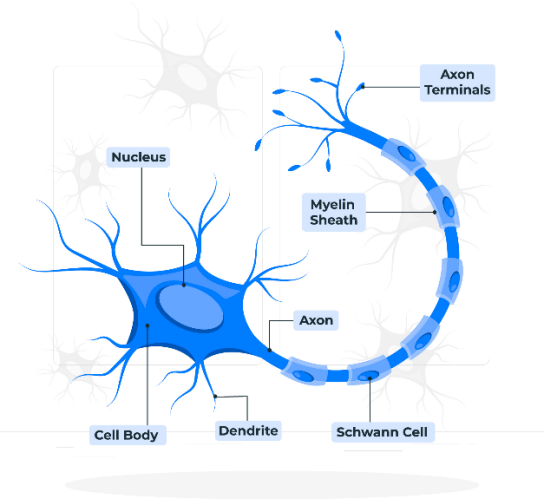
4.1 NEURÓNOVÉ SIETE

Vitajte v téme **Neurónové siete**! Neurónové siete sú kľúčovou súčasťou hlbokého učenia, podmnožiny strojového učenia, ktorá napodobňuje štruktúru a funkciu ľudského mozgu. V tejto časti sa dozviete o základných prvkoch neurónových sietí, vrátane neurónov, vrstiev a aktivačných funkcií. Preskúmame rôzne typy architektúr neurónových sietí, ako sú konvolučné a rekurentné neurónové siete, a ich aplikácie pri riešení zložitých problémov. Získate tiež praktické skúsenosti s trénovaním a testovaním neurónových sietí pomocou populárnych nástrojov a rámcov.

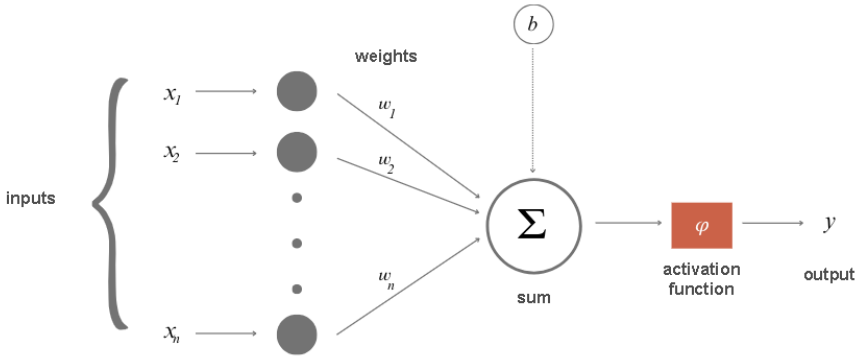
4.1 Neurónové siete

V tejto lekcii sa dozviete o neurónových sieťach, špeciálnej skupine algoritmov strojového učenia. Vďaka nim sme dosiahli mnoho zaujímavých prelomov vo svete umelej inteligencie.

Z hodín biológie viete, že bunka je základnou jednotkou štruktúry a funkcie všetkých živých organizmov. Z hľadiska umelej inteligencie a učenia sú najzaujímavejšie mozgové bunky. Tieto bunky sa nazývajú neuróny. Neuróny sa skladajú z tela s jadrom a dlhších a kratších výbežkov, ktoré sa nazývajú axóny a dendrity. Výbežky umožňujú neurónom spájať sa s inými neurónmi. Tieto spojovacie body neurónov sa nazývajú synapsie. Umožňujú prenos signálov, t. j. elektrických impulzov generovaných jedným neurónom, do iného neurónu. Zaujímavé je, že jeden neurón môže byť prepojený s miliónmi iných neurónov. To znamená, že prijíma a spracováva signály prichádzajúce z množstva iných neurónov a na základe svojich vnútorných mechanizmov jemne ladí signál, ktorý vysiela do iných neurónov. Tento stav sa bežne nazýva stavom neurónovej aktivácie. Trvá len zlomok sekundy, ale umožňuje vykonávať jemné výpočty a generuje signál, ktorý sa prenáša celým nervovým systémom.



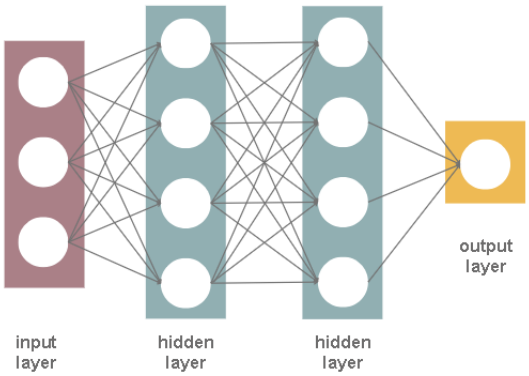
Neurón, s ktorým sa stretávame v umelej inteligencii, je matematickou abstrakciou neurónov mozgu. Je popísaný ako funkcia viacerých premenných$f(x_1, x_2, \ldots, x_n)$, kde každá z premenných$x_1, x_2, \ldots, x_n$ zodpovedá jednému signálu, ktorý dosiahne neurón. Keďže nie všetky signály sú rovnako dôležité pre neurónovú aktivitu, sú spojené váhami$w_1, w_2, \ldots, w_n$, ktoré by mali indikovať ich dôležitosť. Vyššie hodnoty týchto čísel naznačujú, že signál je dôležitejší, a nižšie hodnoty naznačujú, že signál je menej dôležitý. Celková stimulácia neurónov teda zodpovedá súčtu váh$w_1x_1 + w_2x_2 + \cdots + w_nx_n$. Aby bolo možné ovplyvniť ďalšie neurónové správanie, k tomuto súčtu sa pridá jeden voľný člen b, takže celková stimulácia neurónu je v skutočnosti$w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$. Táto hodnota sa potom odovzdá takzvanej aktivačnej funkcii φ, ktorej úlohou je vypočítať výstup neurónu. V závislosti od voľby aktivačnej funkcie budú závisieť aj získané výstupné hodnoty. Ak teraz všetko systematicky zapíšeme, dostaneme, že pre prijaté signály$x_1, x_2, \ldots, x_n$ je výstup neurónu$y = \varphi(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$. Postup, ktorý sme opísali, môžete sledovať aj na nižšie uvedenej ilustrácii



Matematická abstrakcia neurónov

Pozrime sa bližšie na význam parametra b. Prirodzený neurón je charakterizovaný takzvanou aktivačnou prahovou hodnotou – ak je celkový signál prijatý neurónom vyšší ako hodnota aktivačnej prahovej hodnoty, neurón sa aktivuje, spracuje signál a výsledok spracovania odovzdá ďalej iným neurónom. Podobnú úlohu v matematickom modeli neurónov zohráva parameter b. Ak je celkový signál vyšší ako aktivačná prahová hodnota b, t. j. ak je$w1x1 + w2x2 + \cdots + wnxn > b$, neurón sa aktivuje. Parameter b nám teda ponecháva možnosť ovplyvňovať ďalšie správanie neurónov. Výraz$w_1x_1 + w_2x_2 + \cdots + w_nx_n > b$ možno zapísať aj ako$w_1x_1 + w_2x_2 + \cdots + w_nx_n - b > 0$, a v tomto zmysle je parameter b tiež integrálnou súčasťou súčtu.

Keď sú neuróny navzájom prepojené, máme **neurónovú sieť**. Neurónová sieť sa zvyčajne skladá z **vrstiev**, najmä spriaznených skupín neurónov.



Vrstvy neurónovej siete

Vstupná vrstva je vrstva, ktorá sa nachádza na vstupe neurónovej siete. Vstupné signály x1, x2,..., xn tejto vrstvy súvisia s hodnotami atribútov, ktoré máme v dátovom súbore, a tak sa približujeme k praktickému využitiu neurónových sietí. Napríklad, ak máme dátovú sadu obsahujúcu tri atribúty, teplotu, vlhkosť a atmosférický tlak, vstupná vrstva bude mať tri neuróny: prvý bude zodpovedať prvému atribútu, teplote, druhý bude zodpovedať druhému atribútu, vlhkosti, a tretí neurón tretieho atribútu, t. j. atmosférickému tlaku. Pre jeden konkrétny prípad dátového súboru s hodnotami teploty, vlhkosti a atmosférického tlaku, ktoré dosahujú hodnoty 19 °C, 77 % a 1011,2 mb, budeme mať hodnoty signálu x1 = 19, x2 = 77 a x3 = 1011,2. V duchu predchádzajúceho príbehu prvý neurón vstupnej vrstvy prijíma a spracováva iba signál x1, ktorý prechádza bez akejkoľvek úpravy (to je možné pri výbere aktivačnej funkcie φ(x)= x a hodnote w_1 = 1 a b = 0). Ostatné dva neuróny a ich signály x_2 a x_3 sú tiež platné. To by znamenalo, že vstupná vrstva nám umožňuje vstúpiť do siete.

Výstupná vrstva je vrstva, ktorá sa nachádza na výstupe neurónovej siete. Ako môžete tušiť, umožňuje nám čítať výsledky, ktoré pre nás neurónová sieť vypočítala. V závislosti od riešenej úlohy bude závisieť aj počet neurónov v tejto vrstve.

V regresných úlohách, keďže ako výsledok očakávame jednu číselnú hodnotu (množstvo zrážok alebo niečo podobné), stačí jeden neurón. Jeho výsledok by mal zodpovedať predpovedi, ktorú očakávame. V prípade klasifikačnej úlohy zvážme binárnu klasifikáciu a viac triednu klasifikáciu samostatne. Keďže binárna klasifikácia očakáva dve hodnoty, 0 alebo 1, vaša prvá myšlienka môže byť, že potrebujeme dva neuróny. Ak sa však nad tým zamyslíte, zistíte, že stačí aj jeden neurón: ak jeho výstup prekročí prahovú hodnotu, vopred definovanú hodnotu, môžeme to považovať za výsledok 1, alebo naopak za výsledok 0. V prípade viacklasovej klasifikácie môžeme mať niekoľko tried, takže je praktické zaviesť jeden neurón pre každú triedu.

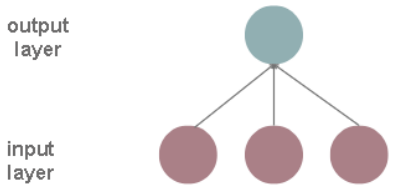
Určite súhlasíte, že pri úlohe klasifikácie viacerých tried očakávame, že všetky výstupy neurónov výstupnej vrstvy budú 0, okrem jedného, ktorý má hodnotu 1 – tak budeme presne vedieť, o akú triedu ide.

Vrstvy neurónovej siete medzi vstupnou a výstupnou vrstvou nazývame **skryté vrstvy**. Neurónové siete, ktoré majú viac ako jednu skrytú vrstvu, sa bežne označujú ako **hlboké neurónové siete**. Odtiaľ pochádza názov **hlboké učenie**. *Hlboké učenie* je oblasť strojového učenia, ktorá sa nimi zaoberá. Názov **plytké učenie sa používa** pre klasickejšie formy učenia.

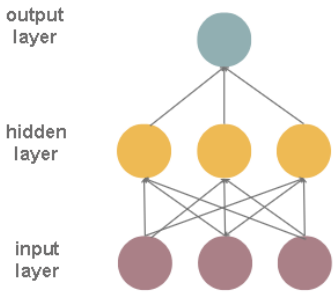
Plne prepojené neurónové siete sú siete, v ktorých je každý neurón predchádzajúcej vrstvy prepojený s každým neurónom nasledujúcej vrstvy. Obrázok znázorňujúci vrstvy neurónovej siete tiež ukazuje jednu plne prepojenú neurónovú sieť, pretože všetky neuróny vstupnej vrstvy sú prepojené so všetkými neurónmi prvej skrytej vrstvy, potom sú všetky neuróny prvej skrytej vrstvy prepojené so všetkými neurónmi druhej skrytej

vrstvy a nakoniec sú všetky neuróny druhej skrytej vrstvy prepojené so všetkými neurónmi (na našom obrázku je len jeden) výstupnej vrstvy. Spôsob, akým sú neuróny vrstiev prepojené medzi sebou, určuje architektúru neurónových sietí a niektoré špecifické vlastnosti sietí, ktoré ďalej určujú, v ktorých oblastiach môžu byť použité. V ďalšej lekcii sa s niektorými z nich zoznámime.

Teraz sa zamyslime nad tým, čo sme vlastne získali zo zavedenia neurónov a neurónových sietí. Predpokladajme, že máme tri atribúty$x_1, x_2 \; and \; x_3$. Lineárny vzťah medzi atribútom a cieľovou premennou je matematicky opísaný rovnicou$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$. Ak namiesto parametrov β napíšeme w a namiesto β0 napíšeme b a presunieme ho na koniec, dostaneme vlastne súčet váh$w_1 x_1 + w_2 x_2 + w_3 x_3 + b$, ktorý vypočíta jeden neurón pre signály, ktoré prijíma. To znamená, že ak by neexistovala aktivačná funkcia, \(φ\) a neurón by modelovali lineárny vzťah medzi atribútmi (signálmi) a výstupmi. Tento vzťah ( ) možno graficky znázorniť sieťou pozostávajúcou iba zo vstupnej vrstvy s tromi neurónmi a výstupnej vrstvy s jedným neurónom, ako je znázornené na obrázku nižšie



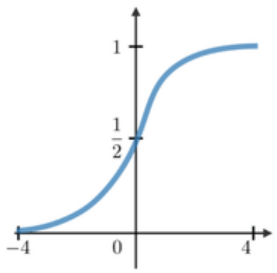
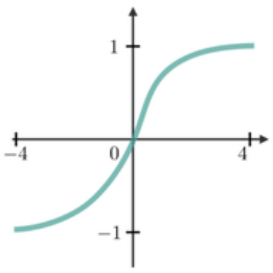
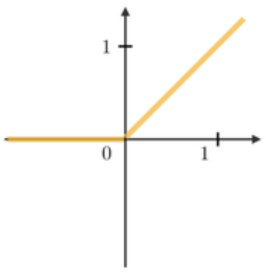
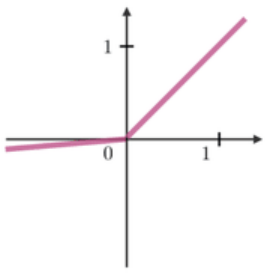
Ak by aktivačná funkcia neexistovala, malo by z hľadiska modelovania závislosti pridanie novej skrytej vrstvy nejaký význam? Nech je to vrstva žltej farby na nasledujúcom obrázku.



Teraz každý neurón skrytej vrstvy vypočíta nejakú lineárnu kombináciu atribútov a neurón výstupnej vrstvy vypočíta nejakú lineárnu kombináciu hodnôt skrytej vrstvy. To by znamenalo, že neurón našej výstupnej vrstvy opäť vypočíta nejakú lineárnu kombináciu atribútov a že sme sa príliš nepohli od reprezentácie zložitejších vzťahov medzi atribútmi a výstupmi. Okrem toho by sme sa neposunuli ani pridaním 100 skrytých vrstiev – vždy by sme modelovali lineárnu závislosť.

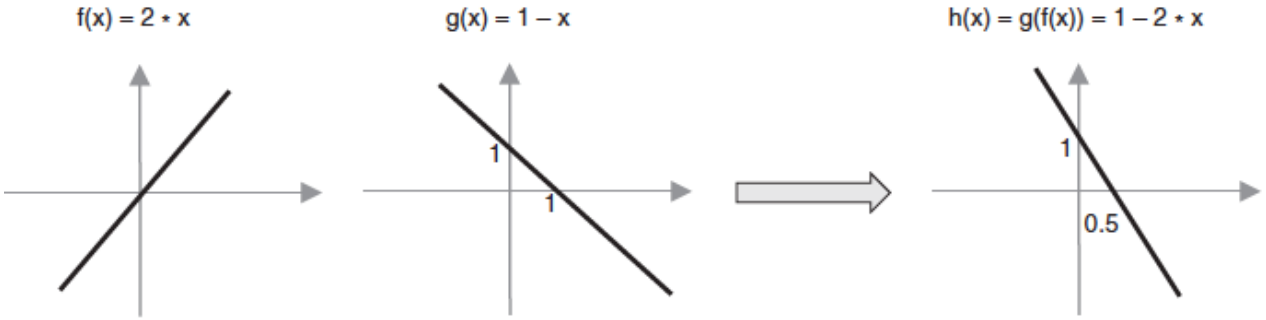
Preto zaradenie aktivačnej funkcie do výpočtov neurónov výrazne mení súbor možností, ktoré máme k dispozícii. Ak použijeme nelineárnu aktivačnú funkciu, budeme môcť modelovať niektoré nelineárne vzťahy medzi atribútom a cieľovou premennou. Existencia nelineárnej aktivačnej funkcie v skrytej vrstve z predchádzajúceho príkladu tak umožňuje neurónu výstupnej vrstvy vypočítať niektorú nelineárnu kombináciu atribútov. V tomto svetle má pridanie nových vrstiev oveľa väčší zmysel. Kombináciou nelinearít viacerých vrstiev môžeme modelovať zložité vzťahy medzi atribútmi a výstupmi.

Aby sme mohli všetky kocky poskladať dohromady, zostáva ešte diskutovať, aké nelineárne aktivačné funkcie sú populárne v strojovom učení. Sú to sigmoidná funkcia, ktorú sme spoznali v príbehu o logistickej regresii, hyperbolické tenges, *Rectified Linear Unit (ReLU)* a Leaky Rectified Linear Unit (Leaky ReLU). Vzorce, podľa ktorých sa tieto funkcie počítajú, a ich grafy sú zobrazené na obrázku nižšie. Ako vidíte, tieto funkcie nie sú skutočne lineárne – ich grafy nie sú reálne.

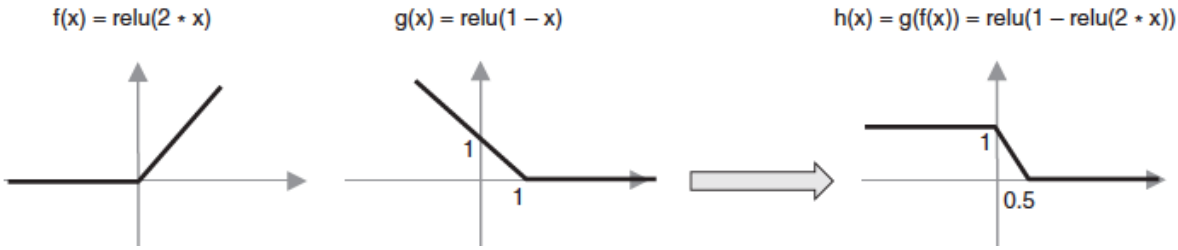
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Najbežnejšie voľby aktivačných funkcií

Aby sme doplnili príbeh o kombinovaní rôznych aktivačných funkcií, pozrime sa na funkcie f(x)= 2x a g(x)= 1 − x. Vidíme, že obe funkcie lineárnej funkcie sú premenné. Ich kombináciou, zložením funkcií, získame funkciu g(f(x))= 1 − 2x, ktorá je tiež lineárnou funkciou premennej. Grafické znázornenie všetkých troch funkcií môžete vidieť na obrázku nižšie.



Teraz sa pozrime na funkcie f(x)= ReLU(2x) a g(x)= ReLU(1 − x), ktoré sa od predchádzajúcich funkcií líšia tým, že obsahujú aktivačnú funkciu usmernenej lineárnej jednotky. Preto sú obe funkcie nelineárne. Ich kombináciou, t. j. zložením, dostaneme funkciu g(f(x))= ReLU(1 − ReLU(2x)), ktorá je tiež nelineárna a má novú „formu": umožňuje nám vyjadriť mierne odlišný vzťah medzi vstupnou premennou a výstupom.



Voľba vhodnej aktivačnej funkcie závisí od povahy úlohy a niektorých vlastností, ktoré by mala neurónová sieť mať počas trénovania. Ako sa to robí, vysvetlíme v ďalšej lekcii.

4.2 Trénovanie neurónových sietí

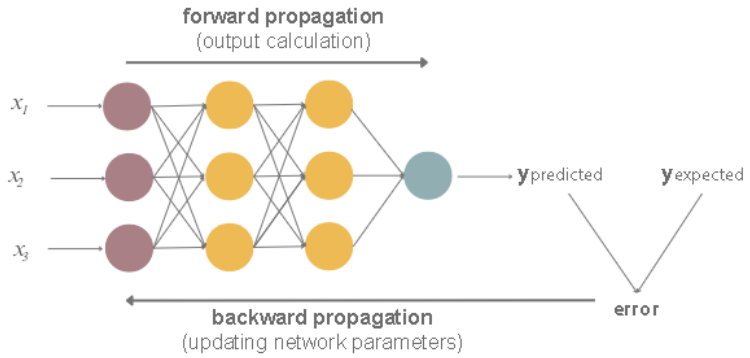
Ako sme videli, každý neurón neurónovej siete je nejakým spôsobom prepojený s ostatnými neurónmi v sieti. Tieto prepojenia sú popísané váhami w, ktoré v skutočnosti predstavujú parametre neurónovej siete, ktoré je potrebné naučiť počas trénovania. Počet parametrov v neurónovej sieti je zvyčajne veľký. Povedzme, že pre plne prepojenú neurónovú sieť, ktorá má 5 neurónov vo vstupnej vrstve, jednu skrytú vrstvu s 10 neurónmi a výstupnú vrstvu s 3 neurónmi, je počet parametrov, ktoré je potrebné naučiť, 93. V praxi majú neurónové siete tisíce a milióny parametrov, dokonca miliardy! Preto je na ich trénovanie potrebné veľké množstvo údajov.

Je počet parametrov v plne prepojenej neurónovej sieti, ktorá má 5 neurónov vo vstupnej vrstve, jednu skrytú vrstvu s 10 neurónmi a výstupnú vrstvu s 3 neurónmi, 93?

Vstupná vrstva neobsahuje neznáme parametre – iba prenáša dáta do siete. Každý neurón skrytej vrstvy je prepojený s každým neurónom vstupnej vrstvy, čo znamená, že každé z týchto prepojení má 5 parametrov a jeden voľný člen. To je spolu 10x5 + 10x1 = 60 parametrov. Každý neurón výstupnej vrstvy je prepojený s každým neurónom skrytej vrstvy, čo znamená, že každé z týchto prepojení má 10 parametrov a jeden voľný člen. To je spolu 3x10 + 3x1 = 33 parametrov. Keď sčítame tieto dve hodnoty, dostaneme 93 parametrov.

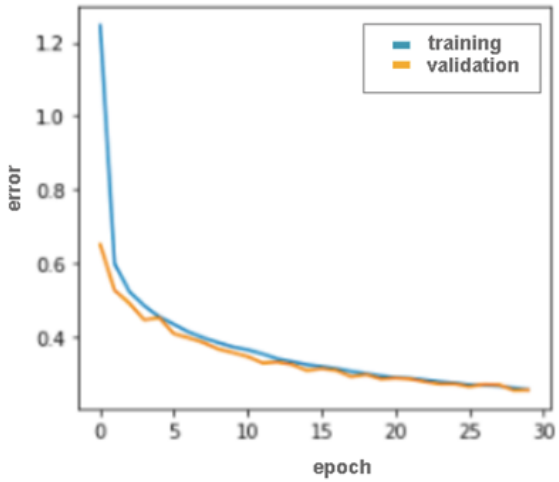
Neurónové siete, podobne ako iné modely, sú trénované na trénovacej sade a hodnotené na testovacej sade. Keďže neurónové siete sú komplexné modely, ktoré sa môžu naučiť komplexné vzťahy medzi atribútmi a výstupmi, dajú sa ľahko prispôsobiť dátam. Preto počas trénovania siete vždy používame validačnú sadu. Pomáha nám to podrobnejšie sledovať priebeh trénovania a skôr si všimnúť nadmerné úpravy a iné nežiaduce vlastnosti modelu.

V úvode kurzu sme uviedli, že neznáme parametre modelu sa určujú definovaním chybovej funkcie a následným použitím niektorých optimalizačných techník (medzi ktoré patrí gradientný zostup) s cieľom nájsť hodnoty parametrov, pre ktoré je chybová funkcia najmenšia. Tento protokol sleduje príbeh neurónových sietí, ale hodnoty chýb sa nepočítajú pre jednotlivé inštancie, ale pre skupiny inštancií. Motiváciou pre tento návrh je v prvom rade práca s veľkým množstvom údajov a potreba paralelizovať a urýchliť celý proces. Preto sa všetky údaje v trénovacom súbore najskôr rozdelia na **pakety (***dávky)* rovnakej veľkosti. Pakety sa potom postupne prechádzajú sieťou a pre ne sa vypočíta hodnota chybovej funkcie porovnaním očakávaných a získaných hodnôt cieľovej premennej. Potom sa v pomere k ich príspevkom aktualizujú hodnoty chyby neurónovej siete spätným prechodom sieťou. Opísaný proces aktualizácie parametrov siete sa nazýva **spätné šírenie** a umožňuje nám v iteráciách vylepšovať hodnoty parametrov a dosiahnuť optimálne hodnoty parametrov. Inak začneme od začiatku.



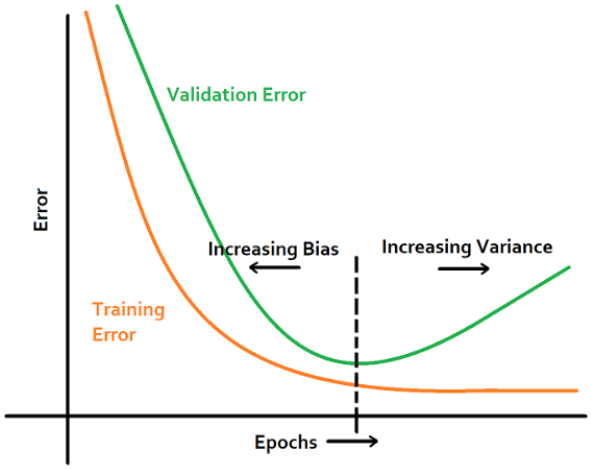
Jedno prechádzanie celým dátovým súborom, t. j. jedno spracovanie všetkých paketov trénovacieho súboru, sa nazýva **epocha.** Neurónové siete sú trénované v niekoľkých epochách. Po dokončení jednej epochy sa dáta „zamiešajú", potom sa opäť rozdelia na pakety a prejdú sieťou. V ktorej epoche bude model trénovaný, závisí od úspešnosti trénovania a dostupných zdrojov. Vzhľadom na prácu s veľkým množstvom dát potrebujú siete špecializovaný hardvér, ktorý dokáže paralelizovať výpočty (napríklad grafické karty alebo tenzorové karty), takže trénovanie sietí je často nákladné a časovo náročné.

Tento spôsob trénovania siete prostredníctvom epok nám umožňuje sledovať priebeh trénovania podrobnejšie. Na konci každej epochy sa vypočíta chyba modelu na trénovacom súbore a chyba modelu na validačnom súbore. Tieto dve hodnoty sa potom zobrazia na grafe, ktorý zobrazuje poradové číslo epochy na osi x a hodnotu chyby na osi y. Jeden takýto graf môžete vidieť na obrázku nižšie. Dobré trénovanie sa vyznačuje komparatívnym poklesom týchto hodnôt na uspokojivú hodnotu chyby – čím sme bližšie k nule, tým je model lepší. Pripomíname, že tento záver vychádza zo skutočnosti, že validačná sada obsahuje údaje, ktoré sú oddelené od trénovacej sady a ktoré sieť vidí po prvýkrát.



Ak zistíme, že hodnoty chybovej funkcie v trénovacej sade klesajú a vo validačnej sade stúpajú, usudzujeme, že model sa prispôsobuje, a trénovanie zastavíme. Ďalej máme dve možnosti. Ak boli hodnoty funkcie chyby modelu v epohe pred pozorovaným prispôsobením modelu uspokojivé, môžeme túto verziu modelu ponechať na ďalšie testovanie na testovacej sade (zvyčajne sa počas trénovania siete ukladá niekoľko verzií modelu s cieľom použiť ich na tento účel alebo v prípade, že je potrebné zastaviť a pokračovať v procese trénovania). V opačnom prípade musíme vyskúšať mierne odlišnú architektúru siete alebo mierne odlišnú sadu jej hyperparametrov. Vzhľadom na to, že každá vrstva siete má svoje vlastné nastavenia ( ) (počet neurónov,

aktivačná funkcia, počiatočná sada parametrov), že vrstvy môžu byť prepojené rôznymi spôsobmi, že musíme súčasne monitorovať všetky nastavenia optimalizačného algoritmu, napríklad gradientný zostup a jeho učebný krok, a že je potrebné splniť niektoré očakávania v oblasti merania kvality, je trénovanie siete náročnou a komplexnou úlohou. Preto sa hovorí, že predstavuje *umenie trénovania.*



Monitorovanie readaptácie neurónovej siete na základe grafov hodnôt chybovej funkcie na trénovacom súbore a validačnom súbore

4.3 Konvolučné neurónové siete (CNN)

Naučiť sa reprezentovať dáta

Neurónové siete nám môžu pomôcť izolovať niektoré abstraktné atribúty v údajoch a naučiť sa reprezentácie, ktoré sú vhodné na riešenie problémov.

V príkladoch, ktoré sme doteraz použili, sme sa najčastejšie spoliehali na existenciu určitej sady atribútov v dátovej sade. Veľké množstvo domén generuje dáta, ktoré majú túto formu, s atribútmi v stĺpcoch a inštanciami v jednotlivých riadkoch. Ako sme videli v úvodnej časti príbehu o príprave dát, aj keď máme tieto atribúty, nie je najintuitívnejšie rozhodnúť, ktoré atribúty zvoliť na vytvorenie modelu. To nás postavilo do pozície, v ktorej sme museli vyskúšať rôzne kombinácie alebo vymyslieť techniky, ktoré nám pomôžu pri výbere atribútov. Vďaka zložitosti funkcií, ktoré modelujú, sa neurónové siete môžu pochváliť schopnosťou naučiť sa filtrovať a zoskupovať atribúty, ktoré sú dôležité.

Táto vlastnosť neurónových sietí je obzvlášť dôležitá pri práci s netabulkovými údajmi – mnohokrát sme sa zaoberali otázkou, ako napríklad reprezentovať obrázky, textové údaje alebo zvukové nahrávky. Hoci máme znalosti o týchto formátoch, je pre nás ťažké presne opísať, čo obsahujú, stručným a použiteľným spôsobom. To nás okrem iného motivovalo k tomu, aby sme uplatnili paradigmu programovania založeného na údajoch. Neurónové siete sa môžu (a čoskoro uvidíme) naučiť niektoré abstraktné atribúty z údajov v ich zdrojovej forme, ktoré sú užitočné pre úspešné riešenie problémov.

Nižšie predstavíme konvolučné neurónové siete, ktoré sa používajú predovšetkým pri práci s obrázkami a videami a na učenie sa vizuálnych atribútov vstupných údajov, a potom rekurentné neurónové siete a transformátory, typy neurónových sietí, ktoré sa používajú na učenie sa atribútov sekvenčných údajov, ako sú text alebo zvuk.

Konvolučné neurónové siete

Konvolučné neurónové siete sú typom neurónovej siete, ktorý sa používa predovšetkým v oblasti počítačového videnia na prácu s obrazmi a video obsahom.

Čiernobiele obrázky sú reprezentované maticami pixelov. Počet typov tejto matice zodpovedá výške obrázku, zatiaľ čo počet stĺpcov tejto matice zodpovedá jeho šírke. Hodnoty jednotlivých pixelov sú čísla v rozmedzí od 0 do 255, kde 0 je čierna a 255 je biela. Všetky hodnoty medzi týmito hodnotami predstavujú nejaký odtieň šedej. Viete uhádnuť, čo sa skrýva za obrázkom, ktorý je reprezentovaný pixelovou maticou nižšie? Ak sa dostatočne vzdialite a nakreslíte kontúry pozdĺž tmavších odtieňov, možno budete schopní uhádnuť, čo je na obrázku. Pomáha to, že v komunite strojového učenia sú obrázky mačiek pomerne bežnou voľbou.

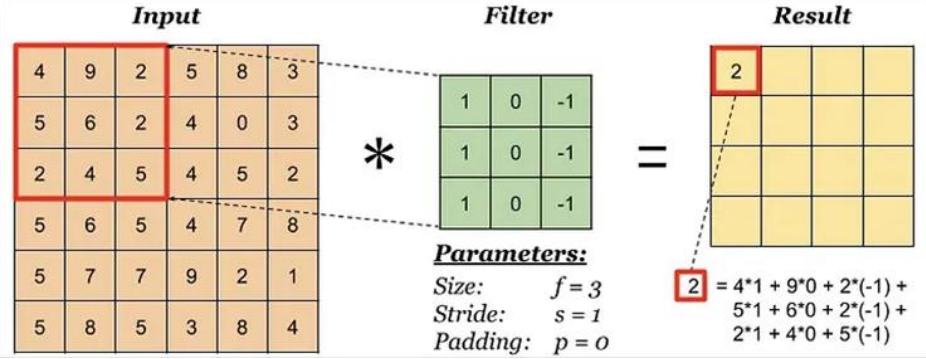
```
[[  9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
 [  3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
 [  1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
 [  3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
 [  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]]
```

Môžete tiež skontrolovať ďalší blok a pozrieť sa, čo je na obrázku.

Rovnako ako je pre nás ľudí ťažké pochopiť, čo je na obrázku reprezentovanom skupinou čísel, tak je to aj s konvolučnými neurónovými sieťami. Svoju analýzu obrázku začínajú najskôr rozpoznaním niektorých jednoduchých prvkov, ako sú horizontálne a vertikálne línie, a potom ich ďalej kombinujú a pridávajú zložitosť, až kým nedospejú k nejakým komplexným popisom obrázku, ktoré im pomôžu vyriešiť danú úlohu. Teraz je prirodzenou otázkou, ako začať od jednoduchých kontúr a na nich stavať, aby sme získali zložitejšie štruktúry. Odpoveď nám dá **konvolučný** operátor.

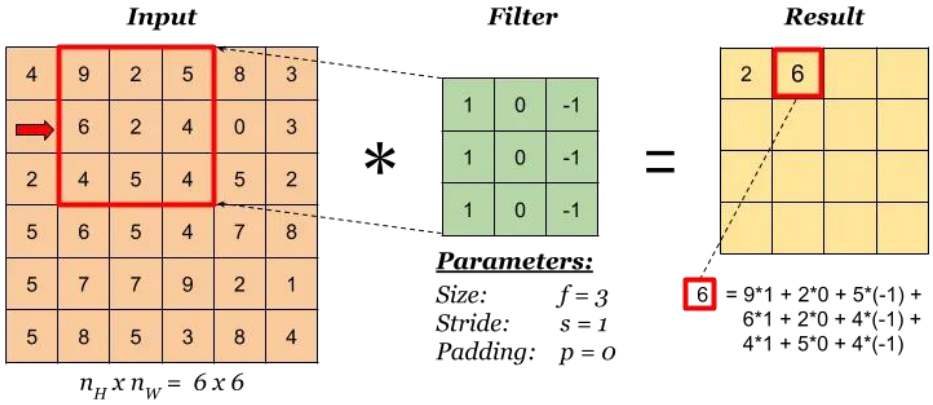
Najjednoduchší spôsob, ako predstaviť konvolučný operátor, je prostredníctvom ilustrácií. Predstavme si, že máme na vstupe maticu 6x6 pixelov, ktorá reprezentuje obraz, a malú maticu, takzvaný filter, s rozmermi 3x3 pixely. Nech sú to matice zobrazené na obrázku nižšie. Začneme aplikovať konvolučný operátor na obrázok (označíme ho symbolom ✳) tak, že prekrývame časť obrázka v ľavom hornom rohu filtrom, potom vynásobíme

jednotlivé hodnoty a súčet hodnôt zapíšeme do novej matice. Táto matica bude výsledkom aplikácie konvolučného operátora.



Konvolúcia – krok 1

Budeme pokračovať v aplikovaní konvolučného operátora: prekrývame filter s časťou obrázku umiestnenou v ľavom hornom rohu, ale tak, aby bol teraz posunutý o jeden pixel od ľavého okraja, t. j. v porovnaní s predchádzajúcou polohou. Opäť vynásobíme jednotlivé hodnoty, sčítame ich a zapíšeme do výslednej matice.



Konvolúcia – krok 2

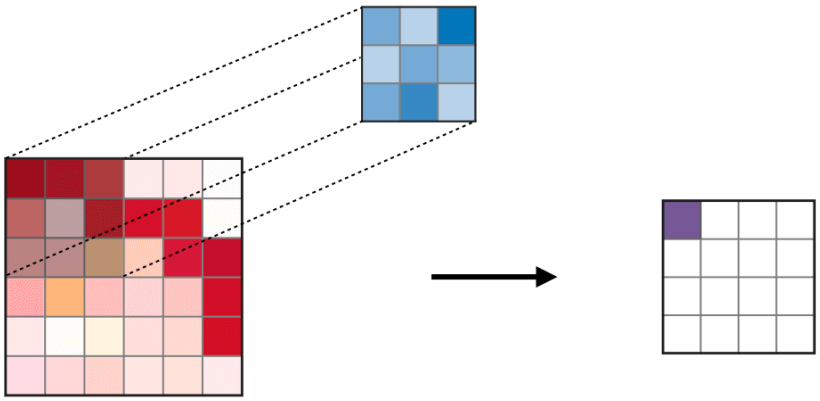
Filter môžeme posunúť o jednu pozíciu doprava až k okraju. Potom ho musíme posunúť o jednu pozíciu nadol a umiestniť späť hneď vedľa okraja. Potom môžeme pokračovať v procese, kým nedosiahneme pravý dolný roh. Výsledkom tejto operácie bude matica s rozmermi 4x4 pixely, ktorej hodnoty sú zobrazené na obrázku nižšie.

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

Konvolúcia – výsledok

O koľko sa filter posunie v každej iterácii, je definované hyperparametrom nazývaným scroll. *stride*). V našom prípade malo posunutie hodnotu 1, pretože sme filter posunuli o jednu pozíciu doprava, t. j. keď mal byť posunutý o jednu pozíciu nadol. Aby sme mohli ovplyvniť rozmery výslednej matice pri aplikovaní konvolučnej operácie (zvyčajne chceme zachovať rozmery, ktoré zodpovedajú vstupnej matici), môžeme okolo počiatočného obrázka pridať rámček. Zvyčajne ide o blok núl alebo jednotiek alebo čísel, ktorých hodnoty zodpovedajú hodnotám najbližšieho pixelu v obraze. Nazýva sa to rozšírenie. *padding)* a jeho šírka sa vždy zdôrazňuje pri aplikovaní konvolučnej operácie. Je to pre nás obzvlášť dôležité, ak sú charakteristiky, ktoré chceme, aby sa náš model naučil, blízko okraja obrazu.

Animácia nižšie ilustruje celý proces aplikovania filtrov na obrázok, t. j. na jej maticu. Ako vidíte, použilo sa posunutie veľkosti 1 a rozšírenie veľkosti 0.



Animácia konvolučnej operácie

Ak použijeme filter z predchádzajúceho príkladu pomocou konvolučnej operácie na počiatočnom obrázku mačiatka, dostaneme obrázok nižšie. Vidíte, že všetky vertikálne línie, ktoré sa objavujú na obrázku, sú zvýraznené.

Extrakcia vertikálnych okrajov

Prekvapuje vás, že horný filter detegoval vertikálne hrany?

Tu je vysvetlenie. Pozrite sa na obrázok zľava doprava. Keď sa prvýkrát presuniete zo svetlej časti obrázka do tmavej časti obrázka, uvidíte vertikálny okraj. Teraz použijeme filter zľava doprava s konvolučnou operáciou. Najvyšší výsledok iterácie konvolúcie bude vtedy, keď pravá strana nášho filtra (stĺpec s číslom -1) bude umiestnená presne na vertikálnom okraji. Keďže okraj je tmavý, hodnoty, ktoré zodpovedajú tejto farbe, sú malé, pretože čierna farba je reprezentovaná nulou. Hodnoty vľavo od okraja sú svetlé, takže čísla, ktoré zodpovedajú týmto farbám, sú väčšie (hodnota pre bielu farbu je 255). Keď sa malé hodnoty zodpovedajúce čiernej farbe okrajov vynásobia -1, t. j. s pravou časťou filtra, a veľké hodnoty, ktoré zodpovedajú svetlým farbám vľavo od okraja, sa vynásobia 0 a 1, t. j. Výsledkom je vyššia hodnota, ako keby sa filter nachádzal kdekoľvek inde, kde nie je filter.

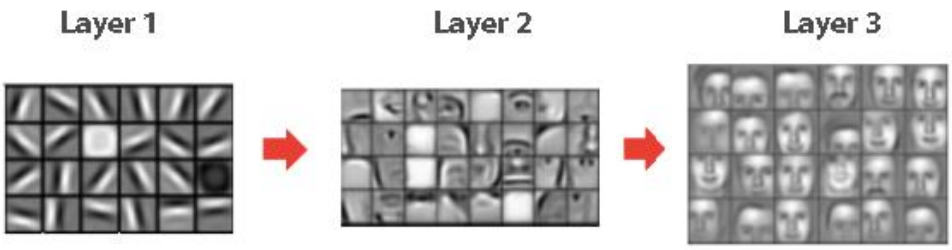
Ako by ste oddelili horizontálne okraje na obrázku? Ktorý filter by ste použili?

Stačí otočiť filter, ktorý oddeľuje vertikálne okraje! Dáva vám to zmysel?

Teraz, keď vieme, ako oddeľovať vertikálne a horizontálne okraje, môžeme kombinovať rôznymi spôsobmi, aby sme vybrali línie, ktoré nie sú len horizontálne a vertikálne. Ďalším kombinovaním týchto výsledkov môžeme dokonca extrahovať sférické kontúry. To sme mali na mysli, keď sme hovorili, že začíname s jasnými, ľahko naučiteľnými charakteristikami a potom krok za krokom budujeme zložitosť charakteristík, ktoré sa môžeme naučiť.

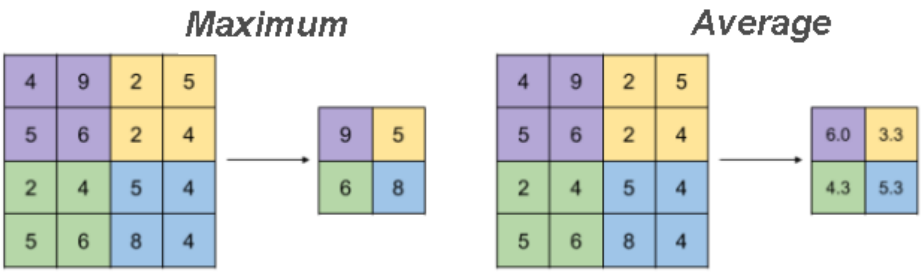
Vrstvy neurónovej siete charakterizované použitím konvolučného operátora sa nazývajú **konvolučné vrstvy**. Kým priekopníci v oblasti počítačového videnia vytvárajú filtre ručne, cieľom trénovania konvolučných neurónových sietí je naučiť sa samy hodnoty, ktoré sa v nich nachádzajú.

Na obrázku nižšie môžete vidieť znázornenie naučených filtrov na vrstvách konvolučnej neurónovej siete, ktorá rozpoznáva tváre. Na najnižšej vrstve sú to horizontálne, vertikálne a diagonálne čiary, na druhej vrstve sú to už obrysy, ktoré zodpovedajú častiam tváre, ako je nos, oči a ústa, zatiaľ čo na tretej vrstve sú to filtre, ktoré zodpovedajú kontúram tváre.



Proces klasifikácie prechodom cez po sebe idúce vrstvy obrazu

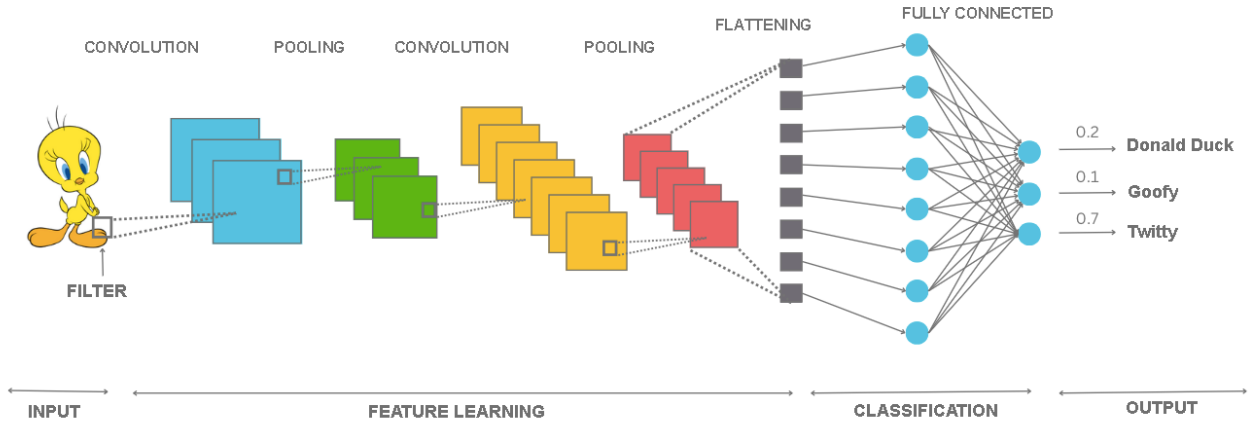
Okrem konvolučnej operácie sa konvolučné siete vyznačujú aj ***zlučovaním***. Ako naznačuje názov, cieľom tejto operácie je zlučovať, t. j. konsolidovať vstupy. Na obrázku nižšie vidíte dva typy zlučovacích operátorov: jeden, ktorý používa maximum, a druhý, ktorý používa priemer na zlučovanie informácií. Rovnako ako konvolučný operátor, aj tento operátor sa aplikuje na bloky vstupov tak, že si všimne blok a vykoná na ňom potrebné výpočty. Hodnota získaná týmto spôsobom sa zadá do novej matice. Na obrázku sú obidva operátory aplikované na bloky 2x2. Intuitívne povedané, maximami zdôrazňujeme najdominantnejšiu časť, zatiaľ čo výpočtom priemeru zohľadňujeme príspevok všetkých častí.



Uplatnením zlučovacej operácie získavame schopnosť znížiť rozmer vstupu, ale zároveň zachovať časť obsiahnutých informácií. Na obrázku vidíte, že použitím operátora zlučovania sme zredukovali maticu z 4x4 na 2x2. Prečo potrebujeme zníženie rozmerov? Výsledkom je, že sieť nám musí dať konkrétnu odpoveď, napríklad či je na obrázku mačka alebo nie, na čo potrebujeme malý počet neurónov.

Vrstvy neurónovej siete, ktoré sa vyznačujú použitím operátora zlučovania, sa nazývajú *vrstvy zlučovania*. Neobsahujú žiadne ďalšie parametre, ktoré by sieť musela naučiť, ale ako sme videli, pomáhajú nám kontrolovať rozmery matíc, s ktorými pracujeme.

Teraz, keď vieme, aké sú stavebné bloky konvolučnej neurónovej siete, pozrime sa, ako ich môžeme spojiť a získať funkčný model, ktorý nám pomôže vyriešiť úlohu klasifikácie. Predstavme si, že potrebujeme vyriešiť úlohu klasifikácie viacerých tried, v ktorej musíme pre každý obrázok kreslenej postavičky určiť, či ide o Tweetyho, Goofyho alebo Daciu Duck. Pozrime sa na ilustráciu architektúry hlbokej konvolučnej neurónovej siete, ktorú sme si vybrali na riešenie tejto úlohy, a diskutujme o motivácii jej vytvorenia.



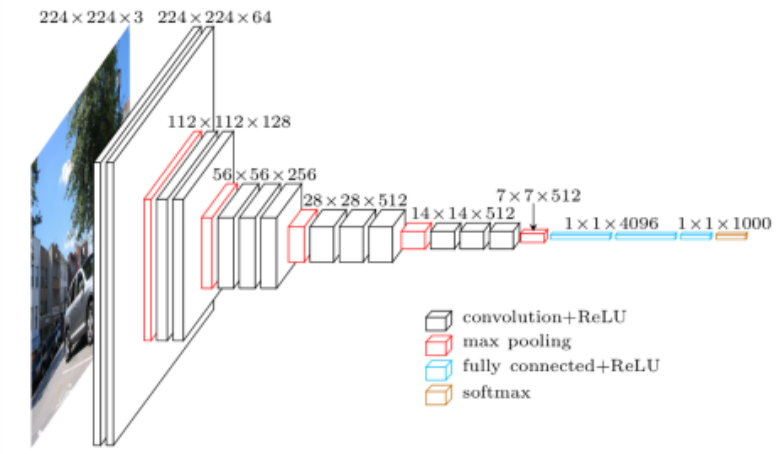
Na začiatku siete je vstupná vrstva, ktorá obsahuje pixely obrázku. Nasleduje konvolučná vrstva (modrý blok). Cieľom tejto vrstvy je použiť konvolučný operátor na extrakciu prvej sady abstraktných atribútov. Potom umiestnime zlučovaciu vrstvu (zelený blok), za ktorou nasleduje ďalšia konvolučná vrstva (oranžový blok) a zlučovacia vrstva (červený blok). V praxi sú konvolučné a zlučovacie vrstvy kombinované a často umiestnené vedľa seba, pretože zlučovacie vrstvy sú ďalej zlučované, t. j. sumarizujú to, čo sa naučili konvolučné vrstvy. Druhá konvolučná vrstva nám umožňuje aplikovať druhý konvolučný operátor na atribúty, ktoré už boli extrahované prvou konvolučnou vrstvou, a vytvoriť komplexnejšie obrazové atribúty. Za týmto blokom vrstiev nasleduje vrstva (sivý blok), ktorej úlohou je „opravovať" maticu (alebo presnejšie tenzor), ktorú sme doteraz získali, a prebaliť jej hodnoty tak, aby boli všetky vedľa seba v jednom poli. Vrstvy na tento účel sa nazývajú vyrovnávacie vrstvy ( ). Po korekcii môžeme ďalej stavať na plne prepojenej neurónovej sieti. Táto sieť bude mať teraz ako vstupy abstraktné atribúty, ktoré sa pre ňu naučí kombinácia konvolučných vrstiev a zlučovacích vrstiev. Okrem korigovanej vstupnej vrstvy vidíme v obrázku aj skrytú vrstvu, ako aj výstupnú vrstvu, v ktorej

sú presne tri neuróny – jeden pre každú z kreslených postavičiek. Výstupné hodnoty týchto neurónov zodpovedajú pravdepodobnosti, že vstupný obrázok patrí do triedy, ktorú reprezentujú. V prípade obrázku z Twitteru, ktorý máme na vstupe, môžeme pozorovať, že výstupná hodnota tretieho neurónu, ktorý presne zodpovedá tejto triede, je najväčšia a je 0,7.

Teraz môžeme vidieť aj to, ako vyzerá architektúra siete *VGGNet*, populárnej konvolučnej siete, ktorá sa aktívne používa v praxi.

VGGNet je hlboká konvolučná neurónová sieť vyvinutá oxfordským tímom Visual *Geometry Group* (odtiaľ názov *VGGNet*). Na prestížnej súťaži Large *Scale Visual Recognition Challenge*, ktorá sa konala v roku 2014, sa táto sieť ukázala ako najlepšia v riešení problému lokalizácie objektov v obraze a ako druhá v poradí v probléme klasifikácie objektov z obrazu. Na klasifikačnú úlohu súboru ImageNet, ktorý sme spoznali, a klasifikáciu do 1000 možných tried bolo použitých viac ako 1,2 milióna obrázkov. Trénovanie tejto siete trvalo 15 až 20 dní s použitím 4 grafických kariet (v tom čase najlepších) *NVIDIA Titan Black.* Predtým bola v týchto úlohách najlepšia *sieť AlexNet,* ktorá je známa zavedením praxe používania grafických kariet na trénovanie neurónových sietí a umožnením ďalšieho rozvoja hlbokého učenia.

Architektúra siete *VGG-16*, verzie siete s 16 vrstvami, je znázornená na obrázku nižšie. Ako vidíme, na vstupe sa očakáva farebný obrázok s rozmermi 244x244 pixelov a sieť kombinuje konvolučné vrstvy a zlučovacie vrstvy (s maximom) a výsledkom je plne prepojená neurónová sieť s 1000 neurónmi na výstupe, kde každý neurón zodpovedá jednej konkrétnej triede súboru *ImageNet*. Samotná sieť má 138 miliónov parametrov a na ich uloženie vyžaduje približne 500 MB pamäte.



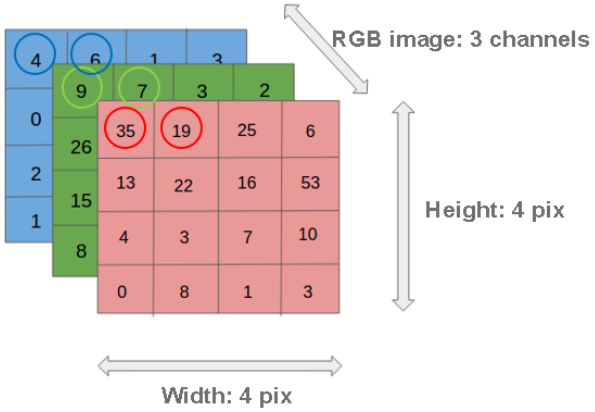
Zobrazenie siete VGG-16

Zvýšenie počtu vrstiev konvolučnej časti siete zvyčajne prináša v praxi lepšie výsledky. Počet vrstiev však nemožno zvyšovať donekonečna. Nielen kvôli obmedzeniam zdrojov, času a nákladov, ale aj kvôli matematickým vlastnostiam hlbokých neurónových sietí, ktoré ďalej sťažujú aplikáciu algoritmu spätného šírenia a trénovanie samotnej siete.

Ak vás tento matematický problém zaujíma, môžete si prečítať viac o miznúcich a explodujúcich gradientoch, najmä v časti konvolučných sietí a reziduálnych spojení.

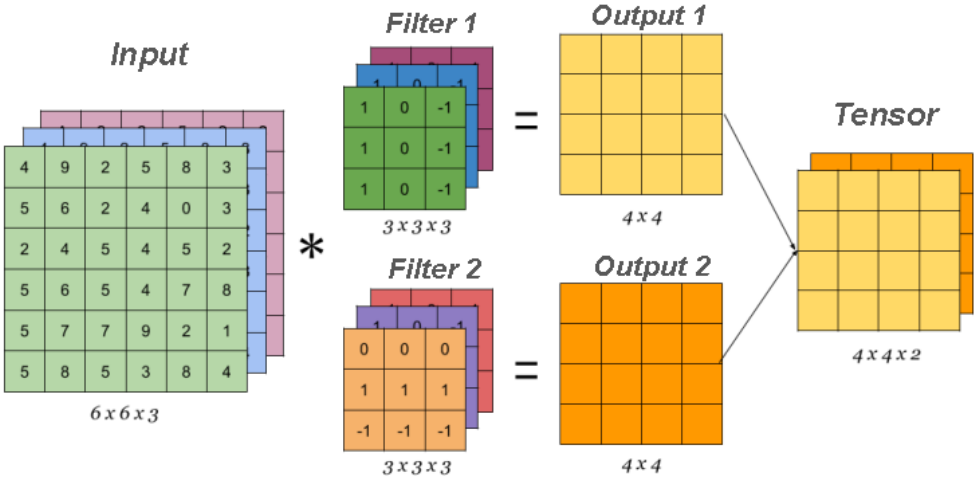
Než sa pustíme do úlohy práce s konvolučnými neurónovými sieťami, pozrime sa na otázku práce s farebnými obrázkami. Zatiaľ sme sa o nich nezmienili.

Keď potrebujeme zobraziť farebný obrázok, ktorý používa farebný formát RGB a zobrazuje všetky farby ako kombinácie červenej, zelenej a modrej, používame tri matice. Pre každú farbu je k dispozícii jedna matica. Počet matíc, ktoré používame na zobrazenie obrázkov, sa nazýva **kanály**. Čiernobiele obrázky majú teda len jeden kanál, zatiaľ čo farebné obrázky majú tri kanály.



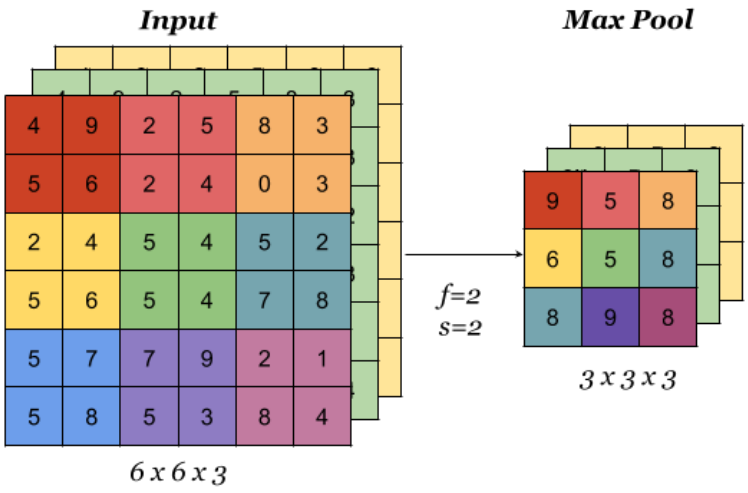
Zobrazenie obrázkov, ktoré používajú farebný formát RGB

Prítomnosť farieb ovplyvňuje výkon konvolučnej operácie úpravou počtu kanálov filtra na počet kanálov obrázku, na ktorý ho aplikujeme. Ďalej je potrebné spárovať každý z kanálov filtra s kanálom obrázka (červený s červeným, modrý s modrým a zelený so zeleným) a vykonať konvolučnú operáciu, ako keby ste pracovali s jedným kanálom. Potom musíme pridať matice, ktoré takto získame, a výslednú maticu vyhlásiť za konečný výsledok. Na obrázku nižšie vidíme dva filtre v konvolučnej vrstve, ktoré sa aplikujú na farebný vstupný obraz. Výsledkom aplikácie každého z týchto filtrov sú samostatné matice, ktoré po „zlúčení" predstavujú konečný výsledok konvolučnej vrstvy. V praxi sa na úrovni jednej konvolučnej vrstvy zvyčajne umiestňuje viacero filtrov, takže výsledkom sú tenzory. Konvolučné operácie sa potom aplikujú na tieto tenzory rovnakým spôsobom – dbá sa len na to, aby počet kanálov filtra zodpovedal dimenzii tenzora (napríklad pre ďalšiu aplikáciu v príklade, ktorý sme zvažovali, by to bolo číslo 2) a aby bol príslušný vstupný kanál spárovaný s príslušným kanálom filtra.



Uplatňovanie konvolučného operátora na viackanálové vstupy

Pokiaľ ide o operáciu zlučovania, tá sa aplikuje na každý kanál vstupného obrazu. Ak má napríklad vstupný obraz 3 kanály, operácia zlučovania sa aplikuje na každý kanál samostatne. To tiež znamená, že operácia zlučovania zachováva počet kanálov v čase nasadenia. Na obrázku nižšie je znázornený tento proces.



Uplatňovanie zlučovacích operátorov na viackanálové vstupy

Táto časť je spojená s Jupyter Notebook 11-VGG-16_network_and_classification.ipynb. Ak chcete pokračovať v čítaní obsahu, kliknite na odkaz a potom na tlačidlo „Open in Colab" (Otvor v *Google Colab)*, aby sa obsah otvoril v *Google Colab.* Ak si prezeráte notebooky na svojom lokálnom počítači, vyhľadajte notebook s rovnakým názvom medzi obsahom a spustite ho. Podrobnejšie pokyny nájdete v časti *Praktická zóna* a v *lekcii Jupyter Notebooks na precvičenie.*

Teraz si vyskúšajme, ako skutočne funguje *konvolučná sieť VGG-16*! Nezabudnite sledovať zošit s kódom počas čítania lekcie.

Po vytrénovaní je možné model neurónovej siete zdieľať s komunitou rozdelením parametrov, ktoré sa v ňom nachádzajú. V tomto príklade použijeme model, ktorý je k dispozícii v knižnici *Keras*. Knižnica *Keras* je open source knižnica široko používaná v komunite strojového učenia. Aby sme mohli využiť *model siete VGG-16,* musíme vykonať nasledujúce dva príkazy:

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

Informácie o modeli, ktorý sme načítali, môžeme prečítať pomocou funkcie model.summary(). Jej výsledkom je popis vrstiev siete, za ktorým nasledujú informácie o veľkostiach vstupov, ktoré tieto vrstvy očakávajú. Teraz môžete vykonať príkaz:

```
model.summary()
```

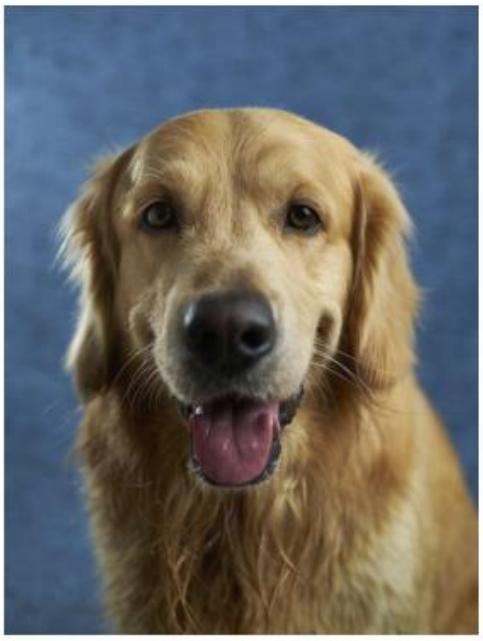
Nenechajte sa zmýliť, ak nerozumiete všetkým detailom, ktoré sa zobrazia po vykonaní tohto príkazu. Je dôležité vedieť, že vstupom má byť obrázok s rozmermi 224x224 pixelov v farbe (preto je vedľa vstupnej vrstvy uvedené 224, 224, 3) a že na výstupe máte jednu z 1000 tried. Tlač môžete tiež porovnať s obrázkom *VGG-16*, o ktorom sme hovorili, aby ste získali viac informácií.

Je dôležité zdôrazniť, že sieť *VGG-16 nebudeme trénovať* – použijeme iba trénovaný model. Preto nesmieme počas prevádzky meniť parametre modelu – každý z nich má svoj vlastný prínos. Celkový počet parametrov modelu, ktoré môžeme prečítať v súhrne siete, je nieco viac ako 138 miliónov.

Myšlienka je taká, že obrázok, na ktorom budeme model testovať, bude ľubovoľný obrázok z webu. Na to použijeme niekoľko štandardných knižníc jazyka Python. Na predvolené nastavenie URL nám pomôže funkcia upload_image, pomocou ktorej môžeme pretiahnuť požadovaný obrázok.

```
def load_image(url_path):
        response = request.urlopen(url_path, context=ssl_context).read()
        return Image.open(BytesIO(response))
```

Na testovanie sme vybrali obrázok zlatého retrievera z adresy https://unsplash.com/photos/x5oPmHmY3kQ, ktorý je voľne použiteľný. Môžete si vybrať obrázok, ktorý chcete! Je dôležité mať na pamäti, že trieda objektu na obrázku musí byť modelu známa. Keďže *model VGG-16 bol* trénovaný na viac ako 1,2 milióna obrázkov, pozná veľa tried, až 1000 rôznych. Zlatý retriever je jednou z nich. Ak modelu poskytneme obrázok s objektom, ktorý nepozná, poskytne nám predpovede tried, ktorých obrázky sa najviac podobajú na náš. Na konci uvidíme, ktoré triedy sa podobajú zlatému retrieverovi.



Hrdina príbehu modelu VGG-16

Keďže obrázok, ktorý treba poslať modelu, musí byť špeciálne pripravený, urobíme nasledovné:

nastavíme jeho rozmery na 224x224 a nariadime mu, aby používal tri farebné kanály RGB:

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

Premeníme obrázok na vhodný maticový formát:

```
matrix_form_test_image = image.img_to_array(test_image)
```

vytvorte balík, ktorý obsahuje náš obrázok:

```
batch = np.expand_dims(matrix_form_test_image, axis=0)
```

Vykonajte numerické predspracovanie obrázku vo forme normalizácie:

```
test_image_batch = preprocess_input(batch)
```

Iba takto môžeme odovzdať model na klasifikáciu. Funkcia, ktorá nám pomôže, sa (ako sa dalo očakávať) nazýva predict.

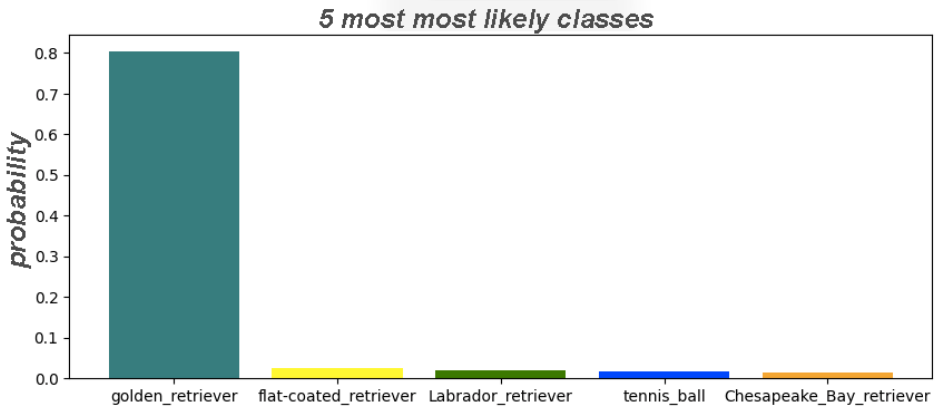
```
model_predictions = model.predict(test_image_batch)
```

Premenná model_predictions, v ktorej ukladáme predikcie modelu, je pole dĺžky 1000 a obsahuje pravdepodobnosti príslušnosti k každej z 1000 tried, ktoré model rozpoznáva. Na extrakciu triedy, do ktorej náš obrázok patrí, môžeme použiť funkciu decode_predictions, ktorá vráti pravdepodobnosti a názvy 5 najpravdepodobnejších tried. To nám poskytne prehľad o tom, ako bezpečný je model pri klasifikácii. Po vykonaní nasledujúceho príkazu získame informácie o najpravdepodobnejších triedach.

```
most_likely_classes = decode_predictions(model_predictions)[0]
```

Keď tieto predikcie graficky znázorníme pomocou kódu uvedeného nižšie, dostaneme graf s pruhmi, ktorý nám umožní ľahšie analyzovať výsledky.

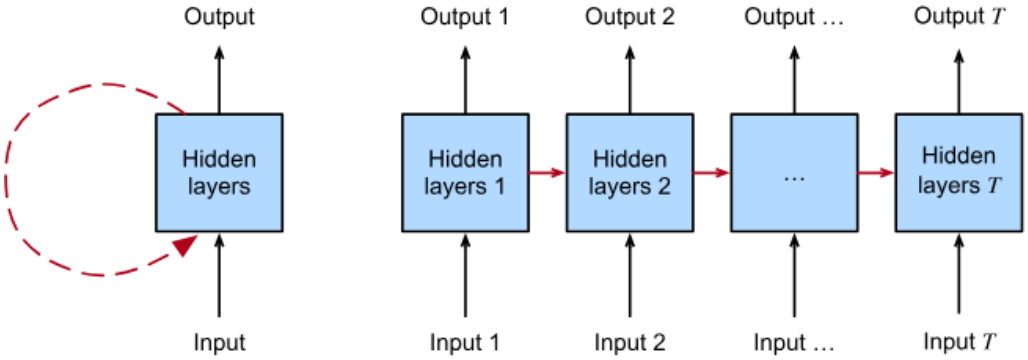
```
class_names = [item[1] for item in most_likely_classes]
class_probabilities = [item[2] for item in most_likely_classes]
plt.figure(figsize=(10, 4))
plt.bar(class_names, class_probabilities, color=['teal', 'yellow', 'green', 'blue',
'orange'])
plt.title('Top 5 najpravdepodobnejších tried')
plt.ylabel('Pravdepodobnosť')
plt.show()
```



Ako vidíme, model s veľkou istotou (pravdepodobnosť je 0,804) predpovedal, že obrázok, ktorý sme vybrali, je obrázok zlatého retrievera. Niektoré ďalšie triedy, ktoré model zohľadnil, sú iné druhy retrieverov. Zvláštne je, že v zozname výsledkov sa objavila aj tenisová loptička. Pravdepodobne preto, že v trénovacom súbore sú aj obrázky, na ktorých retrievery behajú za tenisovými loptičkami. Toto správanie modelu by sa malo ďalej skúmať v praxi.

4.4 Recurentné neurónové siete

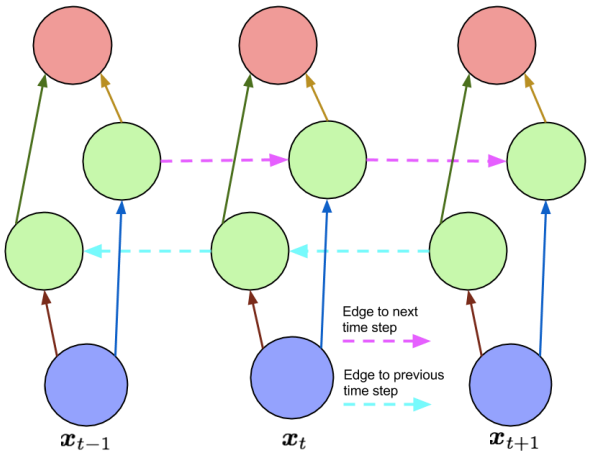
Recurrent Neural Networks (neurónové siete) sú typom neurónových sietí, ktoré sa používajú predovšetkým na spracovanie sekvenčných dát. Sekvenčné dáta alebo sekvencie sa skladajú z prvkov, ktoré na seba nadväzujú. Takými sú napríklad textové dáta (prvky sú jednotlivé slová), zvukové nahrávky (prvky sú jednotlivé vzorky), časové rady (prvky sú jednotlivé merania), genetické sekvencie (prvky sú jednotlivé nukleotidy) a mnohé ďalšie. Recurentné siete spracúvajú sekvenciu prvok po prvku. Aby bolo možné spracovať prvok v pozícii t, musia byť spracované všetky prvky, ktoré mu predchádzajú, a aby boli prvky sekvencie prepojené do jedného celku, hodnoty skrytých vrstiev sú rozdelené medzi spracovanie po sebe idúcich vstupných prvkov. To sa zvyčajne graficky znázorňuje ako na obrázku nižšie.



Recurentná neurónová sieť

(obrázok prevzatý z https://d2l.ai/chapter_recurrent-neural-networks/index.html)

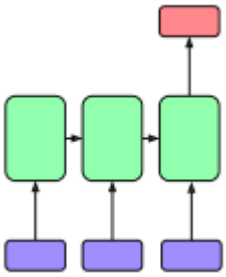
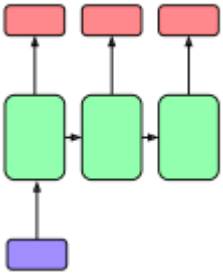
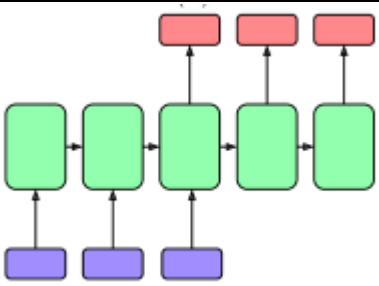
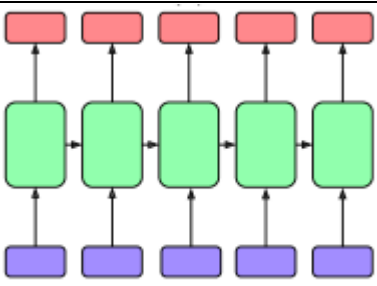
Recurentné neurónové siete sú hypoteticky schopné spracovávať nekonečne dlhé sekvencie: prvok po prvku. Pri trénovaní takýchto sietí sa však zistilo, že zabúdajú. Ak sú sekvencie príliš dlhé, sieť začne zabúdať, čo videla na začiatku, a ukladá nedávno videné informácie na úrovni skrytých vrstiev. Toto pozorovanie viedlo k návrhu špeciálnych neurónov nazývaných LSTM (LSTM – *Long Short Term Memory)* a GRU (*Gated Recurrent Unit),* ale kvôli ich zložitosti sa nimi nebudeme podrobne zaoberať. Riešením tohto problému sú obojsmerné rekurentné neurónové siete (*Bidirectional Recurrent Neural Networks)*. V týchto sieťach sa sekvencia spracováva na jednej strane od začiatku do konca a na druhej strane od konca do začiatku. Vstupná reprezentácia jednotlivých prvkov predstavuje súvislé reprezentácie týchto pasáží, ako je znázornené na obrázku nižšie.



Obousměrná rekurentná neurónová sieť – sekvenčné prvky

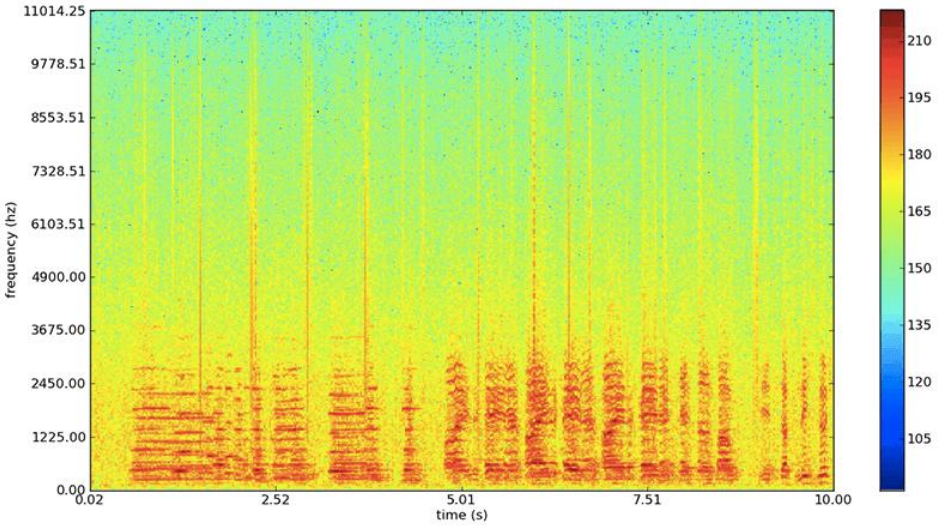
(obrázok prevzatý z https://www.arxiv-vanity.com/papers/1506.00019/)

Existuje niekoľko populárnych architektúr rekurentných neurónových sietí. V tabuľke nižšie stručne prejdeme niektoré z najpopulárnejších príkladov, ktoré graficky znázorníme v ľavom stĺpci a v pravom stĺpci opíšeme sieť a oblasti použitia.

ARCHITEKTÚRA	VYSVETLENIE A PRÍKLADY APLIKÁCIÍ
	Tento typ siete zodpovedá úlohám, v ktorých je vstupom sekvencia a výstupom je vektorové znázornenie s pevnou dĺžkou. Siete tohto typu sa nazývajú kodéry a získané vektory s pevnou dĺžkou podľa kontextu. Úlohy, v ktorých sa stretávame s týmto typom sietí, sú rôzne klasifikačné úlohy, ako napríklad klasifikácia zvukových stôp alebo klasifikácia textu.
	Na rozdiel od predchádzajúceho príkladu je vstupom pre tento typ siete vektorové znázornenie s pevnou dĺžkou a výstupom je sekvencia. Tento typ siete sa nazýva **dekodéry**). Úlohy, v ktorých sa stretávame s dekodérmi, sú generovanie názvov obrázkov.
	Tento typ siete je kombináciou predchádzajúcich dvoch typov a nazýva sa architektúra **kodér-dekodér**. Úlohou kodéra je vytvoriť reprezentáciu (kontext) na základe vstupnej sekvencie, ktorú dekodér môže použiť na generovanie novej výstupnej sekvencie. Tento typ siete sa vyskytuje v úlohách strojového prekladu alebo generovania abstraktov.
	Tento typ siete umožňuje generovanie výstupov pre každý prvok vstupu. Ako vidíte, sekvencie sa nachádzajú na vstupe aj na výstupe. Úlohy, v ktorých sa stretávame s týmto typom siete, sú napríklad úlohy označovania (markovania) jednotlivých prvkov.

Jednou z hlavných nevýhod rekurentných neurónových sietí je nemožnosť paralelizácie: aby bolo možné spracovať prvok v pozícii *t*, musia byť spracované všetky prvky, ktoré mu predchádzajú. Preto trénovanie neurónových sietí vyžaduje oveľa viac času a zdrojov ako trénovanie konvolučných neurónových sietí, ktoré sme spoznali v predchádzajúcej lekcii. Tieto okolnosti viedli k vzniku mechanizmu pozornosti a transformátorov, neurónových sietí, ktoré budú podrobnejšie rozoberané v nasledujúcej lekcii.

Audio nahrávky je možné spracovávať aj pomocou konvolučných neurónových sietí. Audio nahrávku je možné rozdeliť na fragmenty, kratšie úseky trvajúce niekoľko sekúnd, a potom je možné vytvoriť spektrogramy pre každú časť. Spektrogram je grafické znázornenie všetkých frekvencií zvuku prítomných v audio nahrávke. Výsledné obrázky je potom možné odovzdať ako vstupy do konvolučných neurónových sietí a použiť na analýzu zvuku.

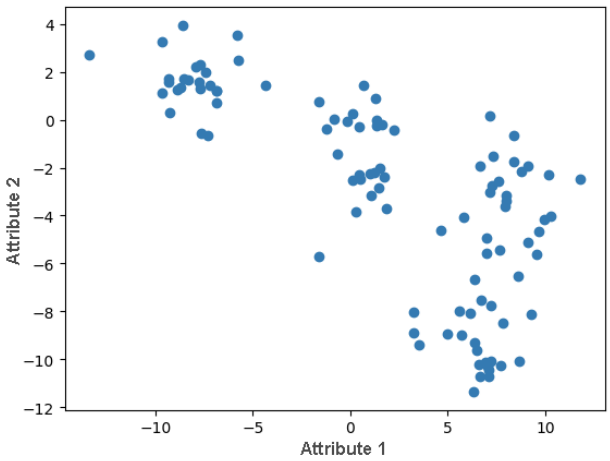


Príklad spektrogramu

4.5 Algoritmus K-Means

K-mean je nezvyčajný názov pre algoritmus. Čítajte ďalej a zistite, čo sa skrýva za touto voľbou.

Aby bolo ľahšie sledovať príbeh algoritmu x-mean, použijeme dátový súbor zobrazený na obrázku nižšie. Skladá sa zo 100 párov bodov, predstavte si, že ide o hodnoty dvoch číselných atribútov.



Táto časť je spojená s Jupyter Notebook 12-k-means.ipynb. Ak chcete pokračovať v čítaní obsahu, kliknite na odkaz a potom na tlačidlo „ Open in Colab " (Otvorené v *Google Colab*), aby sa obsah otvoril v *prostredí Google Colab*. Ak si prezeráte notebooky na svojom lokálnom počítači, vyhľadajte medzi obsahom notebook s rovnakým názvom a spustite ho. Podrobnejšie pokyny nájdete v časti *„Hands-on Zone*" *(Praktická zóna)* a v lekcii *„Jupyter Exercise Notebook*" *(Cvičebný notebook Jupyter*).

V sprievodnom materiáli si môžete všetky obrázky a animácie vygenerovať sami.

Algoritmus **k-mean** by mal nájsť k zhlukov v dátovom súbore. Zhluky, ktoré tento algoritmus hľadá, sú **určené centroidom,** inštanciou, ktorá predstavuje stred zhluku.

Počiatočným krokom algoritmu je inicializačný krok. V ňom musíme náhodne vybrať *x* centroidov. Potom musíme opakovať nasledujúce kroky:

Pre každú inštanciu musíme vypočítať euklidovskú vzdialenosť od každého *z x* centroidov a potom priradiť inštanciu ku klastru, ktorého centroidu je najbližšie. Akonáhle sme usporiadali všetky inštancie, prejdeme k ďalšiemu kroku.

Pre každú z *x* zhlukov musíme vybrať nové centroidy. Urobíme to tak, že vypočítame priemer inštancií, ktoré sa nachádzajú v každom zo zhlukov, a túto hodnotu vyhlásime za nový centroid. Potom sa vrátime späť k kroku 1.

Algoritmus zhlukovania končí, keď sa hodnoty centroidov zhlukov stabilizujú. To by znamenalo, že v dvoch po sebe idúcich iteráciách dostaneme centroidy, ktoré sa od seba líšia len veľmi málo, menej ako určitá vopred stanovená presnosť.

Samotný algoritmus k-mean nie je nepríjemný na programovanie, takže to urobíme spoločne. Predtým si však pozrime niektoré technické detaily:

Jedna inštancia dátového súboru je pár čísel, napríklad (2, -3). To znamená, že centroid bude mať tiež pár čísel a bude mať dve súradnice;

Ak (10, 2) a (4, -4) sú dve inštancie dátového súboru, vypočítame inštanciu reprezentujúcu ich priemer ako (10 + 42, 2 − 42) = (7, -1);

Ak (0, 0) a (3, 4) sú dve inštancie dátového súboru, vypočítame euklidovskú vzdialenosť medzi nimi ako (3 − 0)2 + (4 − 0)2.

V dátovom súbore budeme hľadať štyri zhluky. O niečo neskôr sa zamyslíme nad tým, prečo sme si vybrali práve tento problém. Teraz sa pripravme na naprogramovanie algoritmu.

Premenná k označuje počet klastrov. K = 4.

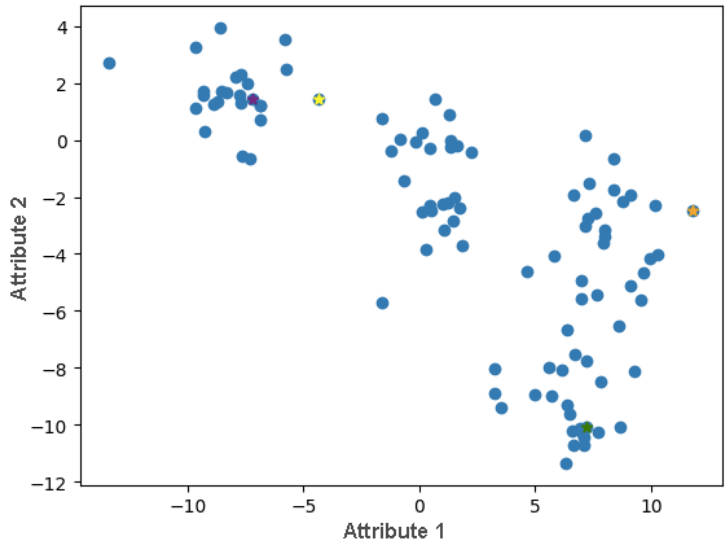
Centroidy klastra označíme premennou centroid. Keďže máme *k* klastrov, bude to pole dĺžky *k*. Jeden centroid je, ako sme už povedali, jeden pár čísel, takže prvky tohto poľa budú páry čísel.

V procese zhlukovania musíme sledovať, ktorý zhluk spájame s ktorou inštanciou. Preto budeme používať štítky na označenie zhlukov, podobne ako pri klasifikačných úlohách. Môžu to byť hodnoty 0, 1, 2 a 3. Všeobecne niektoré hodnoty 0, 1, 2,..., k-1. Všetky štítky budeme uchovávať v sérii štítkov zhlukov.

Teraz zavedieme funkciu generate_centroids(X, k), ktorá generuje počiatočné centroidy. Jej argumentmi sú množina inštancií X a počet zhlukov k, a samotná funkcia náhodne vyberie k čísel z intervalu od 0 do 100 a vráti inštancie, ktoré sa nachádzajú v týchto pozíciách.

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

Na obrázku nižšie sú zobrazené generované centroidy. Každý z nich má farbu klastra, ktorý reprezentuje.

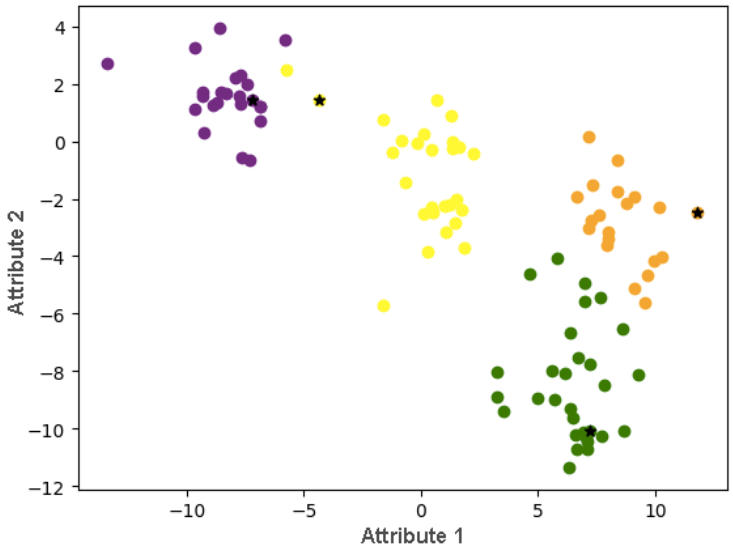


Počiatočné hodnoty centroidov

Teraz napíšeme funkciu divide_data(X, centroids, k) na rozdelenie súboru inštancií do klastrov. Táto funkcia má ako argumenty súbor inštancií X, aktuálne centroidy centroids a počet klastrov k. Pre každú inštanciu vypočítame hodnotu vzdialenosti od každého centroidu, potom vyberieme najbližší centroid a usúdime, že inštancia patrí do klastra, ktorý špecifikuje.

```
def divide_data(X, centroids, k):
  # inicializovať zoznam označení klastrov
  cluster_labels = []
  # prechádzanie dátového súboru inštancia po inštancii
  for x in X:
    # inicializovať zoznam vzdialeností od centroidov
    distances_to_centroids = []
    # potom pre každý centroid ...
    pre centroid v centroidoch:
      # ... vypočítaj vzdialenosť medzi inštanciou a centroidom
      d = calculate_distance(x, centroid)
      #... a pridajte ju do zoznamu vzdialeností
      distances_to_centroids.append(d)
    # keď sme navštívili všetky centroidy,
    # vyberte centroid najbližší k inštancii x
    label = np.argmin(distances_to_centroids)
    # usúdiť, že inštancia patrí do klastra
    # určenému týmto centroidom
    cluster_labels.append(label)
  # výsledkom funkcie je pole klastrových štítkov
  vráť np.array(cluster_labels)
```

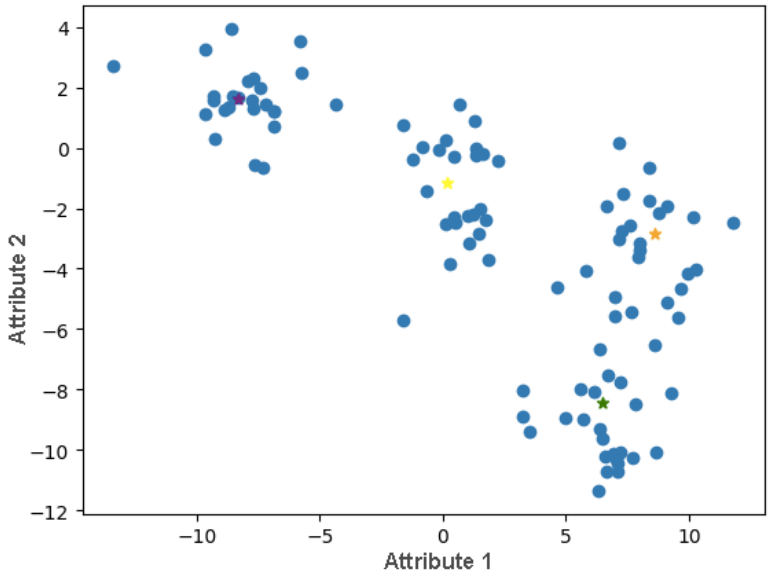
Na obrázku nižšie vidíte prvú iteráciu rozdelenia inštancií do klastrov.



Teraz napíšeme funkciu calculate_new_centroids(X, cluster_labels, k), ktorá dokáže vypočítať hodnoty nových centroidov na základe aktuálneho rozdelenia inštancií do klastrov. Jej argumentmi sú množina inštancií X, aktuálne inštancie label_cluster a počet klastrov k. Pre každý z klastrov by táto funkcia mala extrahovať inštancie, ktoré do neho patria, a potom vypočítať ich priemer.

```
def calculate_new_centroids(X, cluster_labels, k):
  # inicializovať zoznam nových centroidov
  new_centroids = []
  # pre každý klaster
  for i in range(0, k):
    #... extrahovať inštancie, ktoré k nemu patria
    instance_indices = cluster_labels == i
    instances_in_cluster = X[instance_indices]
    # potom vypočítaj novú hodnotu centroidu
    # zpriemerovaním všetkých inštancií v klastri
    new_centroid = np.average(instances_in_cluster, axis=0)
    # pridajte vypočítané nové centroidy do zoznamu všetkých centroidov
    nové_centroidy.append(nový_centroid)
  # výsledkom funkcie je pole nových centroidov
  return np.array(new_centroids)
```

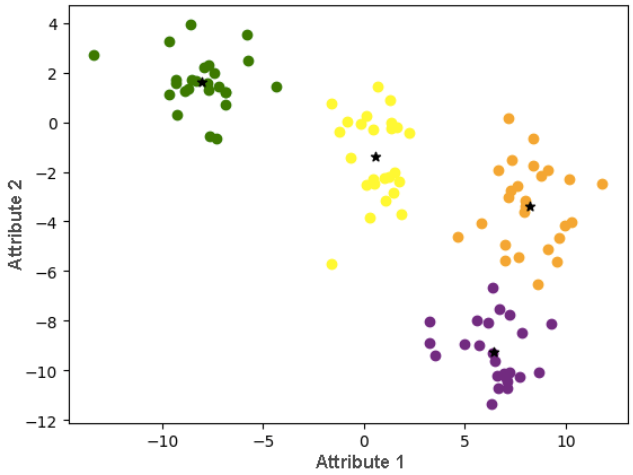
Nové centroidy sú teraz zobrazené na obrázku nižšie. Všimnete si, že centroidy žltých a fialových klastrov sa „oddelili".



Zostáva konsolidovať úlohy jednotlivých krokov do funkcie, ktorá ich bude opakovať dostatočný počet krát. Bude to funkcia execute_clustering (X, k, epsilon = 1e-4, number_of_ iterations = 300), v ktorej X predstavuje množinu inštancií, k počet klastrov, epsilon blízkosť, ktorú musia splniť centroidy klastra, aby sa algoritmus zastavil. Existuje tiež maximálny počet iterácií max_iterations, ktorý dodatočne poskytuje kritérium zastavenia.

```
def execute_clustering(X, k, epsilon=1e-4, max_iterations=300):
  # krok inicializácie centroidov
  centroids = generate_centroids(X, k)
  # v každej iterácii slučky
  for i in range(0, max_iterations):
    # krok 1: rozdelenie inštancií do klastrov
    cluster_labels = divide_data(X, centroids, k)
    # krok 2: výpočet nových centroidov
    new_centroids = calculate_new_centroids(X, cluster_labels, k)
    # kontrola kritérií zastavenia
    # ak sú splnené, zastavíme algoritmus
    if np.linalg.norm(new_centroids - centroids) < epsilon:
      break
    # inak prejdeme k ďalšej iterácii
    centroids = new_centroids.copy()
  # výsledkom funkcie sú konečné označenia klastrov a hodnoty centroidov
  vrátiť cluster_labels, new_centroids
```

Vykonaním tejto funkcie sa dostaneme k finálnemu rozdeleniu súboru inštancií do klastrov, ktoré je znázornené na obrázku nižšie.



V sprievodnej knihe kódu môžete vidieť aj animáciu, ktorá sleduje toto rozdelenie. Niektoré kroky sa opierajú o náhodné rozhodnutia (napríklad ak je inštancia rovnako blízko viacerých centroidov), takže sa nenechajte zmýliť, ak sa niektoré hodnoty mierne líšia.

Referencie

Curriculum: Fundamentals of Artificial Intelligence and Machine Learning

https://petlja.org/sr-Latn-RS/kurs/11203/0

https://scikit-learn.org/

„Machine Learning and Artificial Intelligence“, Ameet V Joshi

„Deep Learning“, Ian Goodfellow Yoshua Bengio Aaron Courville

„Machine Learning For Absolute Beginners“, Oliver Theobald

„Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow“, Aurélien Géron

"Artificial Intelligence A Modern Approach", Stuart J. Russell and Peter Norvig

„Explorations in Artificial Intelligence and Machine Learning“, A CRCPress FreeBook

"The Hundred-Page Machine Learning Book", Andriy Burkov

"Machine Learning in Action", Peter Harrington

"Machine Learning A Probabilistic Perspective", Kevin P. Murphy

"Introduction to Machine Learning with Python", Andreas C. Müller and Sarah Guido

"Machine Learning Yearning", Andrew Ng

"Deep Learning with TensorFlow 2 and Keras", Antonio Gulli, Amita Kapoor, Sujit Pal

"NeuralNetworksandDeepLearning", CharuC.Aggarwal