SADRŽAJ

1. Uvod

Strojno učenje je od relativno mlade, i široj publici nepoznate discipline, u nekoliko godina postalo jako popularno i dinamično znanstveno područje. Danas smo u interakciji s algoritmima strojnog učenja svakdonevno. Primjerice planetarno popularni *chatGPT* predstvlja veliki iskorak u polju *dubokog učenja* (jedna skupina algoritama strojnog učenja). Drugi primjer bi mogao biti *Google Maps* [1], koji spaja optimzacijski problem s predikcijama temeljenim na povijesnim podatcima i generira "optimalan" put od polazišta do odredišta. No, što je to strojno učenje? Na ovo pitanje se može dati puno odogvora i povesti puno metafizičkih rasprava o značenju riječi učenje [2]. Mi ćemo se zadržati dalje od takvih rasprava i reći ćemo da je strojno učenje područje umjetne inteligencije koje proučava algoritme koji na temelju viđenih podataka mogu donositi zaključke o neviđenima podatcima. Konkretno, algoritam strojnog učenja prvo prolazi period učenja s dostupnim podatcima. Primjenom nekog optimizacijskog algoritma, algoritam strojnog učenja "uči" zakonitosti između podataka. Zatim taj algoritam te zakonitosti može primjeniti na neviđenim podatcima.

Primjer 1.0.1. Zanima nas predikcija cijena kuća u nekom gradu [3] kako bi smo mogli donositi bolje odluke o kupnji nekretnina. Na raspolaganju nam je skup podatka koji se sastoji od kvadrature nekretnine, godina izgradnje i cijene po kojima su prodane. Želimo, u budućnosti, predviđati koliko je cijena neke nekretnine na tržištu manja ili veća od očekivane. Na tim neviđenim primjerima cijena po kojoj je nekretnina prodana bit će nepoznata, zato što nas upravo zanimaju nekretnine koje nisu prodane, te mi upravo tu cijenu i pokušavamo predvidjeti.

Primjer 1.0.2. Košarkaškog skauta zanima fizički potencijal juniora. Pošto postoji puno anatomskih i fizioloških atributa, skautu je teško pamtiti koje su vrijednosti i kombinacije vrijednosti povoljne. Primjerice visoki koordinirani igrač koji posjeduje veliku brzinu i visinu skoka, je zasigurno dobar potencijal, no postavlja se pitanje je li primjerice bolji brzi igrač srednje visine, ili spori visoki igrač. Na raspolaganju imamo skup podataka koji se sastoji od anatomskih i fizioloških karakteristika juniorskih igrača i podatka o tome jesu li

[1] Googlov blog o Google mapsu

[2] Alan Turing o umjetnoj inteligenciji.

[3] Skup podatka, *dataset*, o nekretninama u Bostonu.

uspjeli zaigrati kao seniori. Na temelju tih podataka gradimo algoritam koji će skautu pomoći u donošenju odluka, na način da će računati vjerojatnost da igrač s određenim karakteristika uspije, to jest vjerojatnost pripadnosti igrača *klasi* uspješnih igrača.

Rekli smo da je strojno učenje dio umjetne inteligencije, no koja je razlika? Umjetna inteligencija je šire područje koje istražuje načine da računala pokazuju inteligentno ponašanje. Primjer umjetne inteligencije koja nije strojno učenje su ekspertni sustavi. To je sustav u koji je ugrađena velika količina domenskih (*ekspertnih*) pretpostavki u obliku *ako-onda* pravila. Ekspertni sustav [4] na temelju tih pravila i logičkih algoritama donosi odluke. Pošto ovdje nema učenja, to jest nemamo skup podatka na temelju kojeg ugađamo algoritam, ne možemo govoriti o strojnom učenju, ali možemo govoriti o "inteligentnom" ponašanju.

Strojno učenje je nastalo kao spoj statistike i optimizacije. Statistika se kao grana razvila iz matematičkog područja vjerojatnosti, gdje se na temelju *uzorka* procjenjuju (to jest generaliziraju) parametri cijele populacije. Optimizacija se pak bavi pronalaženjem optimuma funkcija. U strojnom učenju nastojimo optimizirati model, tako da će tu biti govora o optimizaciji. Za razliku od čiste optimizacije, u strojnom učenju nam je bitna generalizacija nad neviđenim primjerima.

[4] Ekspertni sustav Dendral, koji pomaže u identifikaciji organskih spojeva

2. Osnovni pojomvi

U ovom poglavlju upoznat ćemo se s osnovnim pojomvima strojnog učenja koji će biti prisutni u svim algoritmima strojnog učenja. Algoritama stvarno ima mnogo, no svi se sastoje od 3 glavna dijela; modela, funkcije pogreške i optimizacijskog algoritma. Također, pošto govorimo o algoritmima postoje ulaz i izlaz svakog algoritma strojnog učenja. Ulazi su *primjeri*, dok izlaz može biti bilo koji entitet koji predviđamo (primjerice klasa nekog primjera, ali i samm element prostora primjera ako imamo generativni model). Pa krenimo redom.

2.1. Primjer

Ulaz u algoritam strojnog učenja, bilo za vrijeme treniranja ili samog korištenja algoritma, zove se primjer (engl. *example*). Primjer je jedan podatak (jedan juniorski košarkaš, jedna kuća u Bostonu) nad kojim radimo predikciju. U slučaju da se predviđa klasa (primjerice je li košarkaš potencijal ili nije) govorimo o klasifikaciji, a u slučaju da se predviđa brojčana vrijednost (primjerice cijena stana) govorimo o regresiji. Primjer ćemo predstaviti vektorom značajki. Naime teško je za očekivati da algoritam radi primjerice sa cijelim čovjekom, a i ne znamo kako bi smo to programski ostvarili. Značajke su neke (nadamo se) bitne karakteristike primjera. To mogu biti visina, kvadratura, temperatura i tako dalje. Premda značajke nisu uvijek numeričke, primjerice kada govorimo o spolu osobe, mi ćemo ih radi jednostavnosti tako prikazivati. Muški spol možemo poistovijetiti s 0, a ženski s 1. Pošto je strojno učenje znanstveno područje, neke stvari moramo formalizirati. Dakle, svaki primjer je vektor značajki definiran kao:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$$

Ovdje su x_i elementi $\mathbb{R}$ te ih ima ukupno n. Svi primjeri imaju isti broj značajki. Primjeri "žive" u prostoru primjera (engl. *input space*) $\mathcal{X}$. Možemo pretpostaviti da je to prostor istovjetan vektorskom prostoru nad realnim brojevima dimenzije n tj. $\mathbb{R}^n$, ali to nije uvijek baš tako. Značajka koja predstavlja godine može biti samo pozitivna, dok stanje računa može biti i pozitivno i negativno. Te razlike su samo semantičke, dok je našem algoritmu ustvari svejedno koje raspone značajke mogu poprimiti.

Primjeri mogu biti označeni i neoznačeni. Označeni primjeri imaju svoju oznaku (engl. *label*) i nad njima se provodi nadzirano (engl. *supervised*) strojno učenje. U pravilu oznake označavamao s y i to je gotovo uvijek vrijednost koju ćemo na neoznačenim primjerima predviđati. Skup primjera na kojem ćemo učiti algoritam, dok primjere za koje predviđamo (npr. dok je sustav u produkciji) oznaka naravno nije poznata, inače predikcija nebi bila potrebna. Nad neoznačenim primjerima radimo nenadzirano (engl. *unsupervised*) strojno učenje. Glavni primjer takvih algoritama su algoritmi grupiranja (engl. *clustering*) poput *k-means* algoritma.

Skup primjera je skup svih primjera koje imamo na raspologanju, i preferabilno je da je broj primijera veći (u pravilu što veći to bolji) od broja značajki. Broj značajki označavat ćemo s n, a broj primjera s N. Skup primjera zajedno s pripadnim oznakama zovemo označeni skup primjera (engl. *labeled dataset*) s oznakom $\mathcal{D}$. Formalno:

$$\mathcal{D} = \{(\mathbf{x}^{(i)}), y^{(i)})\}_{i=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$$

"Prijevod" ove definicije na hrvatski jezik glasi: Skup označenih primjera $\mathcal{D}$ je skup uređenih parova vektora značajki i oznake indeksiran od 1 do N gdje je N broj primjera. Čest obrazac je zapisivanje neoznačenog skupa podataka u matričnom obliku kojeg nazivamo matrica dizajna (engl. *design matrix*) $\boldsymbol{X}$ i vektor oznaka $\boldsymbol{y}$.

$$\boldsymbol{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \dots & x_n^{(N)} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

Za primjere kažemo da dolaze iz neke distribucije p koju nazivamo distribucija podataka ili distribucija iza podataka. Uz samu distribuciju, na oznaku podatka utječe i neki šum koji može proizaći iz raznih razloga (primjerice greška kod mjerenja, slučajnost i druge). Najčešće pretpostavljamo da šum dolazi iz normalne distribucije s očekivanjem nula. U strojnom učenju trudimo se "naučiti" distribuciju podatka, a šum zanemariti.

2.2. Hipoteza

Kod nadziranog strojnog učenja cilj je naučiti hipotezu. Hipoteza je funkcija koja primjerima iz $\mathcal{X}$ pridružuje oznaku y iz $\mathcal{Y}$. Hipotezu označavmo s h te vrijedi:

$$h : \mathcal{X} \to \mathcal{Y}$$

Hipoteza h svakom primjeru iz $\mathcal{X}$ dodjeljuje oznaku klase ili brojčanu vrijednost. Kod binarne klasifikacije (to jest kada postoje samo dvije klase), klase ćemo označavati s O i 1[1], to jest $y = \{0, 1\}$. te za funkciju h vrijedi:

$$h : \mathcal{X} \to \{0, 1\}$$

Za ovu funkciju možemo reći da ulazni prostor $\mathcal{X}$ dijeli na dva podprostora, to jest podprostor primjera s oznakom 0 i podprostor primjera s oznakom 1. U praksi mi nikada ne znamo stvarnu hipotezu odmah, nego do nje moramo doći. Mi ćemo imati neku pretpostavku o tome kako bi hipoteza mogla izgledati, te ćemo odabrati neku *familiju funkcija*. To nije ništa drugo nego skup funkcija koje su po nečemu slične. Primjerice funkcije oblika $f(x) = ax^2$ možemo nazvati familijom kvadratnih funkcija indeksiranih po *parametru* a. Na isti način mi ćemo za h odabrati jednu od funkcija iz familije hipoteza indeksiranih po parametrima θ. Formalno to zapisujemo kao $h(\boldsymbol{x}; \theta)$. To jest predikcija za neki primjer ne ovisi samo o primjeru već i o odabranim parametrima. Hipoteza je dakle jedan specifični odabir parametara za neku familiju funkcija. Parametara naravno može biti i više te tada govorimo o vektoru parametara. Cilj strojnog učenja jest upravo pronaći optimalne parametre θ.

2.3. Model

Više smo puta spomenuli *modele* kod strojnog učenja. Model je ništa drugo nego skup hipoteza. Model je familija hipoteza indeksiranih po parametrima modela. Formalno:

$$\mathcal{H} = \{h(\boldsymbol{x}; \boldsymbol{\theta})\}_{\boldsymbol{\theta}}$$

Hrvatski prijevod glasi: model je skup svih hipoteza definiranih parametrima. Primjetite kako jedan vektor parametara $\boldsymbol{\theta}$ definira točno jednu hipotezu. Sada učenje modela možemo definirati kao pretragu skupa hipoteza s ciljem pronalaska najbolje. U praksi je skup hipoteza prevelik za iscrpno pretraživanje. Zbog toga posežemo za *heurističkim* metodama pretrage. Heuristika je korištenje informiranih pretpostavki kako bi se samnjio prostor pretraživanja.

Primjer 2.3.1. Želimo rasporediti poslove između dva servera. Dobra heuristika za odabir servera kojem će se proslijediti novi posao je odabir servera koji ima manji broj poslova. Ovo ne mora biti optimalno. Server A može obrađivati 5 kratkih poslova, a server B 3 duga posla za koja treba više vremena nego za 5 kratkih. Očekujemo da će raspored dugih i kratkih poslova biti balansiran između poslužitelja, pa očekujemo da se ova heuristika u prosjeku dobro ponaša.

[1]Ovo nije potpuno istina, nekada će nam radi matematičkog zapisa i optimiziacije algoritma biti korisno koristiti -1 i 1.

Bitno je da je model, to jest skup hipoteza familija. Kada to nebi bio slučaj, to jest kada bi model bio skup proizvoljnih funkcija, nebi mogli pretraživati prostor parametara radi odabira optimalne hipoteze, zato što parametri nebi sepcificirali točno jednu hipotezu. Nameće se važno pitanje, naime što znači da je neka hipoteza optimalna, i što generalno mjeri kvalitetu hipoteze?

2.4. Funkcija pogreške i gubitka

Kao što smo već natuknuli, htjeli bi smo, na neki numerički način, vrednovati svaku od hipoteza. Naravno nikada nemamo pristup svim primjerima, već samo nekom skupu za učenje. Ako nije naznačeno drukčije, pretpostavite da raspolažemo i primjerima i njihovim oznakama. Ideja je da na tom, nama dostupnom skupu, izmjerimo kvalitete naših hipoteza. Ako je skup reprezentativan (to jest ako su primjeri slučajno odabrani iz prostora primjera), onda očekujemo da će hitpozea koja je bolja na dostupnim primjerima biti bolja i na neviđenima. *Funkcija pogreške* (engl. *cost function*) $\mathcal{E}$ je funkcija koja mjeri kvalitetu hipoteze nad skupom primjera. U pravilu su nam svi primjeri jednako vrijedeni, pa je kvaliteta hipoteze prosječna kvaliteta hipoteze nad pojedinačnim primjerima. Kvalitetu hipoteze nad pojedinačnim primjerima nazivamo gubitkom (engl. *loss*) $\mathcal{L}$. Formalno:

$$\mathcal{L}_h(h(\boldsymbol{x}), y) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$$

$$\mathcal{E}(h|\mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(h(x^{(i)}), y^{(i)})$$

Funkcija gubitka kao ulaz prima dvije oznake; jednu stvarnu, a drugu onu koju je hipoteza dodijelila. Izlaz funkcije gubitka je uvijek nenegativan broj, naime mi model nikada ne želimo nagraditi, već za točnu klasifikaciju https://en.wikipedia.org/wiki/CoverNažalost, nekada je distribucija podatka presložena za hipoteze koje ćemo koristiti te nema garancije da će model moći pogrešku svesti na nulu.

2.5. Složenost

Složenost ili *kapacitet* je pojam koji ćemo često susretati u strojnom učenju. Kako smo već natuknuli, postoji mogućnost da model nema hipotezu za koju vrijedi $\mathcal{E}(h|\mathcal{D}) = 0$. U tom slučaju za model kažemo da je premalog kapaciteta. Naivno bi smo možda u tom slučaju htjeli jednostavno povećati kapacitet modela. Iako na prvi pogled ta ideja izgleda dobro treba se prisjetiti da je cilj strojnog učenja generalizacija. Naime pretpostavljamo da u podatcima postoji neki šum, neka neobjašnjiva greška. Jedan vrlo svjetovan primjer toga je visina i veličina stopala. Te dvije vrijednosti su pozitivno korelirane, to jest viši

ljudi imaju veća stopala, ali to se ne može izraziti jednostavnom matematičkom formulom. Postoji vjerojatnost da će niža osoba imati veće stopalo od više osobe. Kada bi odabrali model velikog kapaciteta, to jest složenost, model bi umjesto da nauči povezanosti značajki i oznaka, naučio i taj šum. Pošto taj šum ovisi o konkretnom uzorku, složeniji model može se lošije ponašati od manje složenog modela nad cijelom populacijom primjera. Situacija kada je neki model prejednostavan za stvarnu distirbuciju podatka nazivamo *podnaučenost*, a kada je presložen nazivamo *prenaučenost*. Podnaučeni model će imati veliku pogrešku na skupu za učenje i na neviđenim podatcima, dok će presložen model imati nestvarno malu pogrešku na skupu za učenje, a veliku na neviđenim podatcima.

Primjer 2.5.1. Želimo procijeniti isplativost rudarskog okna širine $1m^2$ temeljenog na njegovoj dubini. Svakim metrom dubine, izrudarimo jedan kubni metar rudače. Očito je količina rudače kubno proporiconalna s dubinom okna. Svako okono ima neke početne troškove te postoji minimalna dubina prije no što okono postane isplativo. Također što je okno dublje postaje skuplje za održavanje. Vidimo da dubina na razne načine utječe na isplativost, ali da sigurno imamo jednu kubnu ovisnost. Kada bi smo za model odabrali familiju kvadratnih funkcija, jasno je da bi model bio podnaučen za taj problem. S druge pak strane ako bi za model odabrali familiju polinoma dvadestog stupnja onda bi taj model očito bio prekompleksan. Tako složen model bi naučio šum u podatcima te bi jako loše generalizirao. Idealna složenost je negdje između te dvije krajnosti.

U pravilu su složeniji modeli *nelinearniji*. U smislu regresije to znači da funkcija kojom aprkosimiramo podatke postaje sve kompleksnija krivulja. Kod klasifikacije nelinarniji modeli mogu prostor primjera razdvojiti na kompleksnije podprostore.

2.6. Odabir modela

Očito je odabir modela prave složenosti od velike važnosti. Na isti način kako je model familija hipoteza, možemo napraviti i familiju modela. Da se ne izgubimo u matematičkim formalizmima dat ćemo primjer familije:

Primjer 2.6.1. Defniniramo familiju modela u obliku polinoma stupnja α. Kada bi α primjerice bio 2, tada bi smo odabrali model kvadratnih funkcija. To su funkcije oblika:

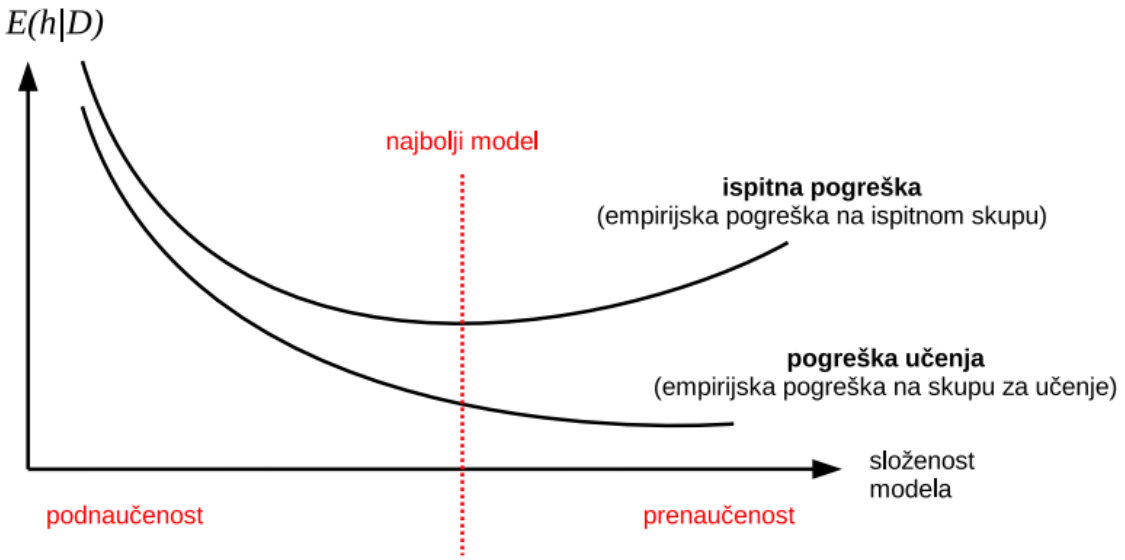
$$y = ax^2 + bx + c$$

Ovdje je vektor parametara $\boldsymbol{\theta}$ jednak vektoru $\begin{bmatrix} a & b & c \end{bmatrix}^T$. Vidimo da α specifcira točno jedan skup hipoteza, to jest familiju kvadratnih funkcija. Primjetite da što je α veći to je odabrana familija nelinearnija. Također model definiran s $\alpha = n + 1$ sadrži sve hipoteze modela definiranog s $\alpha = n$.

Stupanj polinoma α u prethodnom primjeru je primjer *hiperparametra*. Važno je primjetiti da hiperparametri specificraju model u familiji modela, dok parametri specificiraju hipotezu unutar točno jednog modela. Odabirom hiperparametara mi direktno ugađamo složenost našeg modela. Očito je da prije pronalska optimalnih parametara moramo odabrati optimalne hiperparametre. Hiperparametre određujemo mi to jest ljudi koji razvijaju model, dok parametre pronalazi optimizacijski algoritam. Odogovrnost algoritma strojnog učenja je pronalazak najboljih parametara, dok je odabir najboljih hiperparametara van ovalsti i mogućnosti algoritma. Ovo sad zvuči malo razočaravjuće, naime algoritam ne može odabraiti hiperparametre, a mi ustvari ne znamo optimalne hiperparametre. Ovom problemu možemo doskočiti na sljedeći način. Odabaremo skup potencijalnih kombinacija hiperparametara. Za svaki elemnt tog skupa natreniramo model i ocijenimo njegove sposobnosti. Zatim jednostavno odabremo najbolji model, to jest one hiperparametre koji odabiru najbolji model.

2.7. Unakrsna provjera

Unakrsa provjera je najpoznatija strategija za procjenu genralizacijske sposobnosti modela. Ona je u načelu prilično jednostavna. Umjesto da naš algoritam uči nad svim dostupnim primjerima mi ćemo skup primjera podijeliti na *skup za učenje* (engl. *train dataset*) i *skup za provjeru* (engl. *test dataset*). Sada će skup za provjeru upravo simulirati neviđene primjere, to jest možemo generalizaciju modela mjeriti u kontroliranim uvjetima. Zašto nam je to bitno? Mi modele možemo birati iz familije modela indeksirane hiperparametrima. Što je veća složenost odabranog modela, model može bolji naučiti, ali će u jednom trenutku početi gubiti sposobnost generalizacije. Mi bi smo htjeli odabrati model one složenosti tik prije nego generalizacijska spobonost počne opadati. Jasno ovo podrazumijeva da možemo mijenjati hiperparametre na način da krećemo od jednostavnijih prema slozenijim modelima. Tada bi smo, barem teoretski, trebali imati optimalno odabrani model, to jest model koji najbolje generalizira. Očekujemo da će pogreška na skupu za treniranje konzistentno padati. Kada to nije slučaj, znamo da smo negdje ozbiljno pogriješili.



2.8. Optimizacijski postupak

Optimizacijski postupak nastoji odabrati one parametre modela koji minimiziraju danu funkciju pogreške. Pošto funkcija pogreške može biti rahttps://en.wikipedia.org/wiki/Cover

3. Linearna regresija

Vrijeme je da upoznamo naš prvi model strojnog učenja i to je upravo linearna regresija. Linearna regresija je algoritam koji predviđa numeričku oznaku (očito, ipak se radi o regresiji zar ne?). To je algoritam nadziranog strojnog učenja, to jest potrebne su nam oznake za svaki primjer. Zašto se linearna regresija zove linearnom? Zato što je oznaka koja se predviđa linearna kombinacija značajki. To jest vrijedi:

$$h(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Primjetite da se umjesto vektora $\boldsymbol{\theta}$ koristi oznaka $\boldsymbol{w}$. Ovo je samo konvencije radi, ali i jedno i drugo su parametri. $\boldsymbol{w}$ dolazi od engleskog *weight*, zato što taj vektor definira težinu svake značajke u linearnoj kombinaciji. Pošto je riječ o linearnoj kombinaciji riječ je o pravcu u dvije dimenzije, ravnini u tri, te hiperravnini u višedimenzionalnom prostoru .Obratite pozornost na težinu w_0, to je takozvani *bias* koji omogućuje da da hiperravninu pomaknemo od ishodišta. Jednostavna ilustracije zašto nam je to potrebno je primjerice procjena visine na temelju godina. Čak i kada netko ima 0 godina, to jest promatramo novorođenče, nemoguće je da je visine 0.

Nakon kratkog uvoda pogledajmo jedan elementaran primjer.

Primjer 3.0.1. Predviđamo cijenu nekretnine na temelju kvadrature. Imamo skup za učenje koji se sastoji od primjera (u ovom slučaju samo kvadratura. Odabrali smo jednodimenzionalan primjer radi lakše ilustracije.) i njegove oznake (to jest cijene stana). Pošto radimo običnu linearnu regresiju model je skup hipoteza oblika

$$h(\boldsymbol{x}, \boldsymbol{w}) = w_0 + x_1 w_1$$

Želimo pronaći pravac koji najbolje odgovara našim primjerima. Jasno je da je jako mala vjerojatnost da su primjeri kolinearni, to jest zbog raznih faktora (kvart, stanje nekretnine i sličnih) cijene neće biti savršena funkcija kvadrature. Cilj je onda pronaći takav pravac da točke (*data points*) najmanje odstupaju od njega.

U primjeru smo nažalost ostali razočarani, naime stvarno ne postoji garancija da možemo pronaći takvu hiperravninu na kojoj će ležati sve naše točke. Postavlja se pitanje kako numerički vrednovati svaku hipotezu, to jest koju funkciju gubitka (ili pogreške) koristiti. Naivno

možemo učiniti sljedeće. Jednostavno za svaki primjer izračunamo koliko se predikcija razlikuje od stvarne oznake to jest:

$$\mathcal{L}_h(\boldsymbol{x}, y) = y - h(\boldsymbol{x})$$
$$\mathcal{E}(h|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} y^{(i)} - h(\boldsymbol{x}^{(i)})$$

Obaj pristup nažalost neće biti dobar. Naime neki predikcije će biti prevelike, a neke premale te će se takve greške međusobno poništavati. Jedna od ideja bi mogla biti uzeti apsolutnu razliku ove vrijednosti. To bi stvarno riješilo ovaj problem, ali ta funkcija nebi bila derivabilna. Mi funkciju greške želimo minimizirati, to jest naći njen minimum. Iz analize znamo da derivabilne funkcije minimum (a i maksimum) postižu u točkama gdje je derivacija jednaka nuli. Tako da funkcije greške koje nemaju derivaciju moramo optimizirati iterativno. Rješenje ovog problema je sljedeće; nećemo gledati razliku, već kvadrat razlike. Pošto je kvadrat broja uvijek nenegativan, nećemo imati problem s poništavanjem gubitaka. Također derivaciju kvadrata znamo izračunati. Formalno:

$$\mathcal{L}_h(\boldsymbol{x}, y) = (y - h(\boldsymbol{x}))^2$$
$$\mathcal{E}(h|\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N} (y^{(i)} - h(\boldsymbol{x}^{(i)}))^2$$

Pitate se zašto ne dijelimo s N već sa 2? Radi pojednostavljivanja optimizacije. Bitno je primjetiti da se skaliranjem funkcije točka njenog optimuma ne mijenja, tako da nam je "svejedno" koju od te dvije funkcije minimiziramo. Posljedica ovog kvadrata je i da se točke, čija je predikcija daleko od stvarne oznake, kažnjavaju više, dok se točke čije su predikcije blizu stvarne oznake kažnjavamo proporcionalno manje. Kako pronalzimo minimum ove funkcije? Izjednačavnjem derivacije s nulom. Pozorni čitatelj će se zapatitati kako znamo da će pronađena stacionarna točka stvarno biti minimum? Odgovor se krije u tome što je funkcija gubitka, a posljedično i funkcija greške [1]. Pošto deriviramo funkciju po parametrima modela, riječ je o funkciji realne varijable s više argumenata. Tako da umjesto nul-derivacije tražimo nul-gradijent.

Jednostavnosti radi, izvod ćemo raditi s jednodimenzionalnim podatcime, te će mo ga zatim generalizirati na višedimenzionalne.

$$\nabla_{w_0,w_1}\mathcal{E}(h|d) = \mathbf{0}$$

Računamo parcijalne derivacije po w_0 i w_1 te ih izjednačavamo s nulom.

$$\frac{\partial}{\partial w_0}\left[\frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - h(w_1 x^{(i)} + w_0)^2\right] = 0$$

[1] zbroj konveksnih funkcija je konveksna funkcija

$$\frac{\partial}{\partial w_1}\left[\frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - h(w_1 x^{(i)} + w_0)^2\right] = 0$$

Nakon pojednostavljivanja izraza [2] dobivamo sljedeće forumle:

$$w_0 = \bar{y} - w_1\bar{x}$$
$$w_1 = \frac{\sum_i^n x^{(i)}y^{(i)} - N\bar{x}\bar{y}}{\sum_i^N (x^{(i)})^2 - N\bar{x}^2}$$

Primjetite da se optimizacija svodi na uvrštavanje brojeva u formule, to jest imamo analitičko rješenje našeg optimizacijskog problema. Primjetite kompleksnost izraza za w_1. Kada bi smo imali višedimenzionalnije primjere, izrazi za težine bi postajali sve nezgrapniji. Željeli bi smo imati optimizacijski postupak koji je neovisan o broju paramatera. Jasno je da je to nemoguće u punom smislu te riječi, već da je algoritmaski nezavisan. Tu u priču ulazi matrica dizajna. Linearnu regresiju s više značajki zovemo *višestruka regresija*.

3.1. Postupak najmanjih kvadrata

Raspolažemo skupom primjera:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_n^{(N)} \end{bmatrix}_{N\times(n+1)}$$

Dodane jedinice su takozvane *dummy* značajke i one služe kako bi *bias* jednostavno uključili u zbroj. Također imamo i vektor oznaka $\boldsymbol{y}$.

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}_{N\times 1}$$

U idealnom slučaju htjeli bi smo naći vektor težina $\boldsymbol{w}$ takav da vrijedi:

$$\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$$

Svaki redak matrice $\boldsymbol{X}_i$ skalarno bi se množio vektorom težina $\boldsymbol{w}$ i dao bi točno traženu oznaku y_i. U tom slučaju optimalne težine bi jednostavno pronašli sljedećom operacijom:

$$\boldsymbol{w} = \boldsymbol{X}^{-1}\boldsymbol{y}$$

[2] pogledajte drugi dokument

Nažalost ovo je preoptimistično, naime potreban nam je inverz matrice dizajna. Taj inverz ne postoji čim matrica nije kvadratna (a idelno niti neće biti zato što bi htjeli imati više primjera od značajki), a čak kad bi i bila kvadratna nema garancije da inverz postoji. Radi toga posežemo za *pseudoinverzom*[3]. Pseudoinverz je matrica koja nije pravi inverz, ali od svih matrica te dimenzije najbolje odgovara inverzu. Ako matrica ima inverz, njen pseudoinverz je upravo inverz. Pseudoinverz matrice $\boldsymbol{X}$ označavamo s $\boldsymbol{X}^+$. Optimalne težine pronalazimo:

$$\boldsymbol{w} = \boldsymbol{X}^+ \boldsymbol{y}$$

Potrebno je još samo saznati kako se računa pseudinverz. Prepostavlimo da je matrica "visoka" to jest da ima više primjera od značajki[4]. Pseudoinverz je dan s:

$$\boldsymbol{X}^+ = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T$$

Ovo je poznati postupak najmanjih kvadrata (engl. *least squares* zato što minimizira kvadratna odstupanja predikcija od stvarnih oznaka.

3.2. Nelinearni podatci

Iskustveno znamo da nisu sve zakonitosti linearne, već postoje kubne, kvadratne a i neke ovisnosti koje se ne mogu izraziti elementrarnim funkcijama. Spomenuli smo da je u linearnoj regresiji model ustvari hiperploha, to jest višedimenzionalna generalizacija pravca, odnosno linearne funkcije. Postavlja se pitanje kako možemo iskoristiti ovaj jednostavan i moćan model linearne regresije na podatke koji nisu linearni. Ideja je jednostavna, a vrlo moćna: model nećemo dirati, on će i dalje modelirati hiperplohe, već ćemo podataka preslikat nelinearnim funkcijama preslikavanja u prostor viših dimenzija. Pogledajmo primjer

Primjer 3.2.1. Imamo podatke oblika

$$\boldsymbol{x} = [x_0, x_1]^T$$

Sada ćemo ga preslikati pomoću funkcija polinoma drugog stupnja. Konkretno to je funkcija oblika $\phi(\boldsymbol{x}) = \mathbb{R}^2 \to \mathbb{R}^6$.

$$\phi(x_1, x_2) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$$

Sada za neki stvarni podatak ta transformacija izgleda ovako:

$$\boldsymbol{x^{(i)}} = [2, 1]^T$$

$$\phi(\boldsymbol{x^{(i)}}) = [1, 2, 1, 2, 4, 1]^T$$

[3] Izvod za znatiželjne biti će u zasebnom dokumentu.

[4] Kada to nije slučaj pseudoinverz računamo singularnom dekompozicijom.

Primjetimo par ključnih stvari iz prethodnog primjera. Dimenzionalnost vektora se povećala, po Coverovom teoremu povećanjem dimenzionalnosti raste vjerojatnost da su podatci linearno separabilni. Mi radimo regresiju, ali posljedica tog teorema je da su veze između podataka manje nelinarne. Također primjetite da je *dummy* značajka u polinomijalnom preslikavanju dodana automatski to jest ona predstavlja x_1^0 i x_2^0. U primjeru smo koristili polinomijalno preslikavanje koje je najčesće, iako se koriste i neka druga.

Primjer 3.2.2. Modeliramo jednoddimenzionalne podatke kvadratnih ovisnosti tj. oznaka y se može izraziti kao

$$y = ax^2 + bx + c + \epsilon$$

gdje je ϵ neki šum prisutan u podatcima. Odabiremo polinomijalno preslikavanje drugog stupnja. Naš model linearne regresije će modelirati funkciju

$$h(x) = w_2x^2 + w_1x + w_0$$

Vidimo da su ove funkcije izrazito slične, to jest našem modelu samo fali šum, što je i poželjno, zato što želimo naučiti samo distribuciju podatka, a ne i stohastički šum. Kada bi smo odabrali polinom trećeg stupnja kao funkciju preslikavanja, dobili bi smo model koji bi imao prevelik kapicete, to jest postoji mogućnost da bi krenuo učiti šum.

Rezimirajmo ovu genijalnu ideju, razvili smo model koji izvrsno radi s linearnim podatcima, i sada umjesto promjene modela, mi nelinearne podatke pokušavamo transformirati u linearne.

Uvođenjem višedimenzionalnog preslikavanja ϕ naš model linearne regresije postaje

$$h(x) = \boldsymbol{w}^T\phi(\boldsymbol{x}) = \sum_{i=1}^{n} w_i\phi(\boldsymbol{x})_i$$

Također, optimizacijski postupak se mijenja samo u notaciji, umjesto da koristimo matricu dizajna $\boldsymbol{X}$, koristimo $\boldsymbol{\Phi}$, gdje je svaki redak $\boldsymbol{\Phi}_i = \phi(\boldsymbol{X}_i)$.

Dotaknimo se složenosti modela. Složeniji modeli su nelinearniji. Pošto je model linearne regresije linearan u parametrima, nelinarnost se povećava nelinearnijim preslikavanjima. Konkretno, za model koji kao transformaciju koristi polinom drugog stupnja, možemo reći da je *složeniji* od modela koji koristi polinom prvog stupnja. Konkretno, kvadratni model sadrži sve hipoteze linearnog modela. To jednostavno možemo dokazati na način da fiksiramo sve težine (koeficjente) koji stoje uz nelinearne značajkena nulu. Pošto stupanj nelinearnosti, to jest u našem slučaju stupanj polinoma, direktno određuju složenost modela riječ je o *hiperparametrima*.

3.3. Regularizacija

Postavlja se pitanje, za neke podatke čija distribucija nije poznata, kako odabrati optimalnu složenost. Također postavlja se problem, da nam možda treba neka "međusloženost", to jest neki "međustupanj" polinoma. Zato je osmišljen postupak regularizacije. Prije nego što ga objasnimo obratimo pažnju na sljedeću propoziciju. Za neki model linearne regresije koji koristi polinomijalnu funkciju preslikavanja. Što su težine modela veće to je hipoteza nelinearnija. Primjerice, kvadratna funkcija s parametrom $a = 2$ je nelinearnija od one s parametrom $a = 0.1$. Regularizacija je postupak kažnjavanja modela na temelju magnitude težina. To jest u funkciju pogreške ugradit ćemo kaznu temeljenu na normi[5] vektora težina tog modela. Što smo time dobili? Dobili smo mogućnost modelu dopustiti preslikavanja većeg stupnja, uz svojervrsnu garanaciju da se model neće prenaučiti. Naime ako bi model krenuo koristiti nelinearnost kako bi naučio šum, očekujemo da će se kvadratno odstupanje samnjivati u mjeri manjoj od rasta kazne za povećanje magnitude. Pogledajmo formalnu definiciju regularizirane funkcije pogreške:

$$\mathcal{E}_R(\boldsymbol{w}|\mathcal{D}) = \mathcal{E}(\boldsymbol{w}|\mathcal{D}) + \lambda\Omega(\boldsymbol{w})$$

Vidimo da je regularizirana funkcija pogreške linearna kombinacija funkcije pogreške i *regularizacijskog izraza*, a *regularizacijskim faktorom* λ ugađamo omjere. Pošto λ direktno utječe na složenost modela, riječ je o *hiperparametru*. Nameće se pitanje, kako odabrati regularizacijski izraz? Već smo prije natuknuli da želimo da izraz mjeri magnitudu težina. Kao dva (glavna) kandidata nameću se *manhattan udaljenost*(L^1 norma) i *euklidska udaljenost*(L^2 norma):

$$L^1(\boldsymbol{w}) = \sum_{i=1}^{n} |w_i| = \|\boldsymbol{w}\|_1$$

$$L^2(\boldsymbol{w}) = \sqrt{\sum_{i=1}^{n} w_i^2} = \|\boldsymbol{w}\|_2$$

Odabirom L^1 norme govorimo o *lasso* regularizaciji, a odabirom L^2 norme o hrbatnoj regularizaciji (engl. *Ridge regression*) o kojoj će biti riječ u nastavku. Hrbatna regularizacija ima rješenje u zatvorenoj formi dok *lasso* regularizacija nema. Ona se optimizira iterativno, te je njena bitna karakteristika, da, u prosjeku, više parametara "pritegne" na nulu od hrbatne.

Također trebamo izuzeti težinu w_0 iz regularizacije iz dva razloga. Prvi je da ona uopće ne doprinosi složenosti, a drugi je da ona omugućava pomicanje naše hipoteze što nam je vrlo bitno svojstvo.

[5]Norma vektora mjeri duljinu vektora.

3.3.1. Hrbatna regularizacija

Kod hrbatne regresije, nova funkcija pogreške koju nastojimo minimizirati glasi

$$\mathcal{E}(\boldsymbol{w}|\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N} \boldsymbol{w}^T\phi(\boldsymbol{x}^{(i)}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

Slično kao i kod obične regresije, radi matematičke jednostavnosti, koristimo $\frac{1}{2}$ umjesto $\frac{1}{N}$. Primjetite da je λ zamijenjen s $\frac{\lambda}{2}$. Minum ove funkcije ostvaruje se s izrazom[6]

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}$$

Primjetite da je u odnosu na neregulariziranu regresiju dodan samo član $\lambda\boldsymbol{I}$ koji "erodira" težine, to jest regularizira izraz. Jedan detalj na koji još nismo obratili pažnju je taj da težinu w_0 izuzimamo iz regularizacije. To ostvarujemo postavljanjem prvog elementa dijagonale matrice $\lambda\boldsymbol{I}$ u nulu.

[6]Pogledati izvod u drugom dokumentu

4. Logistička regresija

U prethodnom poglavlju upoznali smo se s osnovnim modelom koji se primjenjuje na regresivne probleme. U ovom poglavlju upoznat ćemo se klasifikacijskim modelom koji je dugo vremena bio *defacto* standard za klasifikaciju. Za složenije probleme, poput klasifikacije slike i teksta, ovaj model nije dovoljno složen, ali jako dobro radi na manje složenim problemima.

Logističku regresiju, unatoč imenu, koristimo kod klasifikacijskih problema. Kod binarne klasifikacije, postoje samo dvije klase pa su i oznake binarne, to jest može vrijediti ili $y \in \{0, 1\}$ ili $y \in \{-1, 1\}$. Ovisno o algoritmu optimizacije, i matematičke jednostavnosti biramo koje vrijednosti ćemo koristiti. Postavlja se pitanje zašto jednostavno ne bi radili linearnu regresiju, na način da jednostavno predviđamo oznaku. Iako se na prvu ovaj pristup čini validan, zbog kvdartnog gubitka, primjeri za koje će model biti "jako siguran" da su na primjer pozitivni, to jest nalaziti će se "dublje" u području pozitivnih primjera biti će jako puno kažnjavani. Vidimo da kvadratni gubitak nikako nije dobar za klasifikaciju, zato što će kažnjavati i pozitivno klasificirane primjere. Model logističke regresije se s tim problemom bori na dva načina, prvi je da se mijenja model, a drugi je promjena funkcije pogreške. Pa krenimo od modela.

4.1. Model logističke regresije

Sjetimo se da je model ništa drugo nego skup hipoteza indeksiran parametrima. Stoga, dovoljno nam je opisati kako izgledaju hipoteze. Vektor parametara $\boldsymbol{theta}$ ponovno ćemo označavati s $\boldsymbol{w}$. Hipoteze modela logističke regresije izgledaju ovako:

$$h(\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{1 + exp(-\boldsymbol{w}^T \phi(\boldsymbol{x}))} = \sigma(\boldsymbol{w}^T \phi(\boldsymbol{x}))$$

Primjetimo dvije stvari, prva je da i dalje koristimo funkciju nelinearnog višedimenzionalnog preslikavanja ϕ. Druga je da smo ustvari uzeli običan linearan model