SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ante Buterin

VERIFIKACIJA POTPISA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, rujan, 2019.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Za majku Vedranu, koja je jedva čekala ovaj dan

Sadržaj

Uvod

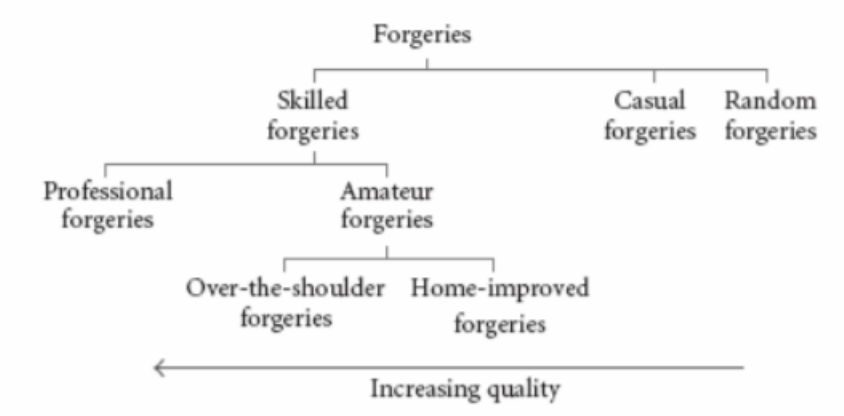
Verifikacija potpisa jedan je od čestih zadataka u forenzičkoj analizi dokumenata. Problem najčešće rješavaju posebno trenirani ispitivači dokumenata, koji odgovaraju na pitanje: Podudara li se dani potpis Q s već poznatim potpisima K osobe A. Trening takvih ispitivača traje godinama te jedino oni mogu donositi zaključke direktno promatrajući već poznate potpise nekog subjekta. U ovom diplomskom promatrat će se mogućnost automatizacije tog procesa, koja bi, na kraju, trebala dati veću preciznost nego čovjek. Brzina treninga algoritma je daleko veća nego u ljudi, a sam algoritam bi trebao biti u mogućnosti sagraditi više slojeva apstrakcije za razumijevanje problema verifikacije potpisa. Također, automatizacijom možemo dodavati nove potpise te time povećati skup poznatih potpisa nekog subjekta, čime će se povećati preciznost predloženih modela.

Postoji opće prihvaćena podjela metoda verifikacije na dvije kategorije prema načinu skupljanja podataka: *Online* i *Offline*. *Online* podaci bilježe pokrete olovke tijekom kreiranja potpisa, iz kojih se izvlače razne značajke, kao što su lokacija, brzina, akceleracija i pritisak olovke, kao funkcije u ovisnosti o vremenu. *Offline* podatak je dvodimenzionalna slika potpisa. Obrada sirove slike je teži pristup, budući da nam nedostaju stabilne dinamičke značajke *Online* podataka. Ovaj rad će se fokusirati samo na *Offline* podatke, budući da nemamo pristup alatima za skupljanje dinamičkih značajki.

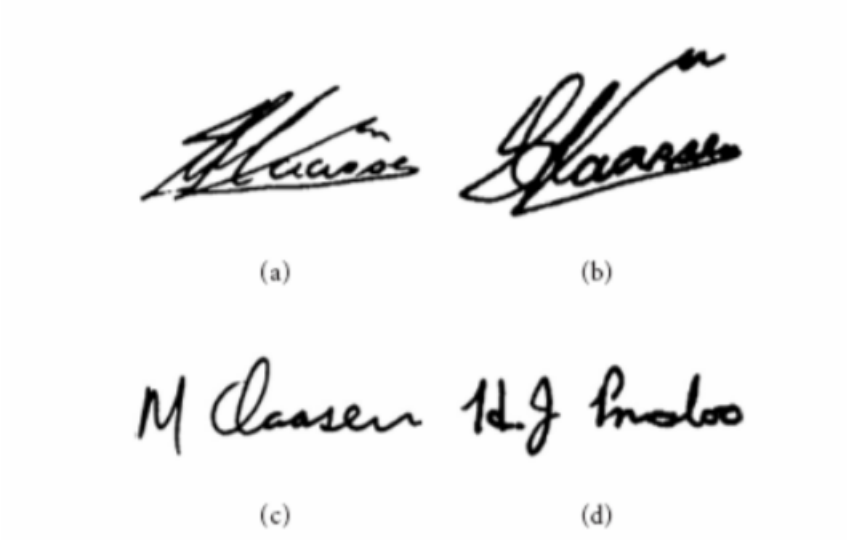
Problem verifikacije je, također, otežan činjenicom da postoji varijacija među potpisima iste osobe. Glavni uzroci su dob, fizičko i psihološko stanje potpisatelja. Potrebno je izgraditi robustan model koji će biti sposoban uzeti te faktore u obzir, ali i prepoznati krivotvorine. Model ne smije biti ni prestrog ni preblag. Formalnije, potrebno je naći ravnotežu između tzv. *false positive* i *false negative* metrika modela.

Pristupi koje ćemo iskušati se razlikuju u načinu ekstrakcije značajki. Prvi pristup se temelji na ekstrakciji manjeg broja značajki, koje su specifične za potpise, te treniranja jednostavnijih modela na takvom skupu za treniranje. U drugom pristupu će značajke biti intenzitet svakog piksela na slici te treniranje kompleksnijih modela na takvom skupu. Za razliku od klasičnih radova iz područja strojnog učenja, detaljno ćemo obraditi implementaciju svakog korištenog modela.

Pogledajmo kakvim krivotvorinama ćemo se pozabaviti u radu. Krivotvorine se generalno dijele na 3 vrste: iskusne (*skilled*), opuštene (*casual*) i nasumične (*random*).



Slika 0.1: Kategorizacija krivotvorina



Slika 0.2: Primjer (a) pravog potpisa, (b) iskusne krivotvorine, (c) opuštene krivotvorine, (d) nasumične krivotvorine

Poglavlje 1

Verifikacija izvlačenjem specifičnih značajki

Pristup kojeg obrađujemo u ovom poglavlju je sličan onom kojeg ljudski ispitivači primjenjuju u forenzici. Promatramo značajke na slici potpisa, koje se razlikuju među rukopisima, primjerice, kut potpisa, površina koju zauzima potpis, centar potpisa, broj vrhova i sjecišta i druge. Prvo dajemo postupak za obradu slike i izvlačenje značajki. Potom opisujemo svaki model koji će, na temelju danih značajki, procijeniti ispravnost danog potpisa. Na poslijetku iznosimo detalje eksperimenta i uspoređujemo rezultate između danih modela.

1.1 Obrada slika

Sirova slika potpisa može sadržavati razne vrste šumova koji trebaju biti uklonjeni, da bismo pravilno izvukli značajke iz nje. Na samom početku, pretvaramo sliku iz RGB u *Greyscale* format, budući da boja potpisa nije bitna karakteristika za potpis te daje nepotrebne podatke, koji mogu smetati u ekstrakciji značajki. Naša slika je reprezentirana matricom dimenzija $m \times n$, čije su vrijednosti u segmentu $[0, 1]$ i reprezentiraju intenzitet crne boje u danom pikselu. Preciznije, uvodimo oznaku za početnu sliku $I_0 \in M_{m,n}([0, 1])$. Potom **invertiramo** sliku i dobivamo novu sliku I_{inv}, u kojoj pozadina postaje crne boje, a sam potpis bijele:

$$I_{\text{inv}}(i, j) = I_{0,\max} - I_0(i, j),$$
$$I_{0,\max} := \max\{I_0(i, j) \mid (i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}\}.$$

Pozadina bi sada, idealno, trebala biti crna. Međutim, to ne mora biti slučaj ako postoje neki pikseli van potpisa koji imaju boju između boje potpisa (bijele) i boje pozadine (crne). Takve piksele **neutraliziramo**, računajući prosjek vrijednosti intenziteta piksela po redovima. Sve piksele koji imaju intenzitet ispod prosjeka reda kojem pripadaju neutraliziramo,

tj. postavljamo im intenzitet na nulu. Time dobivamo sliku s oznakom I_{ra}:

$$I_{\text{r}}(i,j) = I_{\text{inv}}(i,j) - \frac{1}{m}\sum_{k=1}^{m} I_{\text{inv}}(k,j),$$

$$I_{\text{ra}}(i,j) = \begin{cases} I_{\text{r}}(i,j), & \text{ako je } I_{\text{r}}(i,j) > 0, \\ 0, & \text{inače.} \end{cases}$$

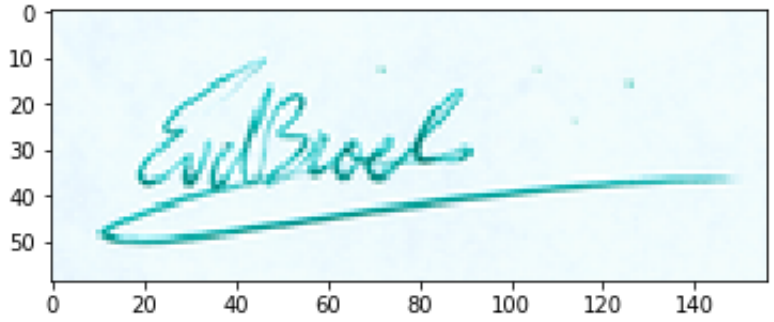
Potom uklanjamo dodatni preostali šum **izglađivanjem**, koje svakom pikselu računa prosjek, zajedno sa svih njegovih 8 susjeda, čime se dobiva slika I_{a}:

$$I_{\text{a}}(i,j) = \frac{1}{9}\sum_{k=i-1}^{i+1}\sum_{l=j-1}^{j+1} I_{\text{ra}}(k,l).$$

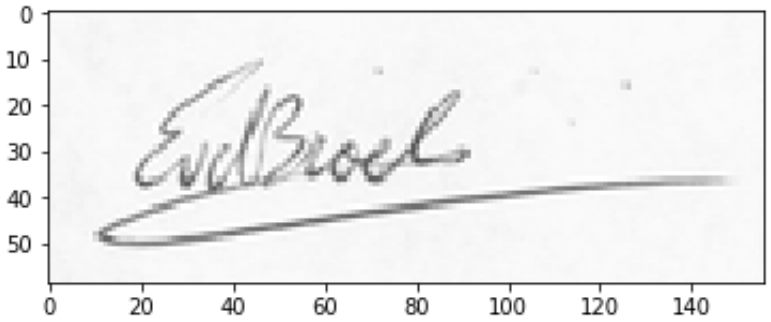
Na poslijetku, pretvaramo *Greyscale* sliku I_{a} u **binarnu** sliku $I_{\text{bin}} \in M_{m,n}(\{0,1\})$, sljedećim algoritmom:

1. Pronađi piksele s najmanjim i najvećim intenzitetom na slici te postavi prag T kao prosjek te 2 vrijednosti.
2. Podijeli piksele na 2 skupa s obzirom na to prelaze li njihovi intenziteti prag T ili ne.
3. Odredi prosječne vrijednosti μ_1, μ_2 intenziteta piksela u prethodno definiranim skupovima te promijeni vrijednost praga na $T = (\mu_1 + \mu_2)/2$.
4. Vrati se na korak (2) ako je udaljenost između novog i starog praga veća od prethodno definirane konstante $\varepsilon = 0.01$.
5. Intenzitete piksela koji prelaze konačni prag T postavi na 1, a ostale postavi na 0.

Na sljedećim slikama možete vidjeti kako se jedan primjer potpisa mijenja u *preprocessing pipeline*-u:

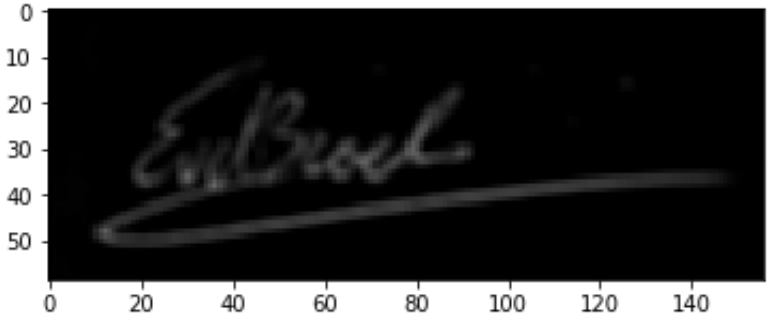


Slika 1.1: Originalna RGB slika potpisa

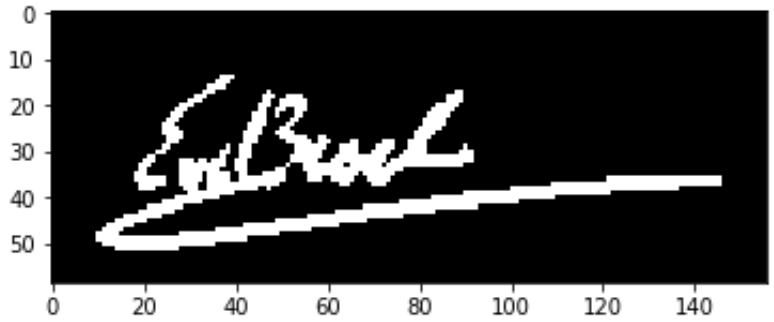


Slika 1.2: *Greyscale* slika potpisa

Slika 1.3: Invertirana slika potpisa s neutraliziranim šumom



Slika 1.4: Izglađena slika potpisa



Slika 1.5: Konačna binarna slika potpisa, spremna za ekstrakciju značajki

1.2 Izvlačenje značajki

Sljedeći korak je izvlačenje značajki iz slike, za koje pretpostavljamo da su jedninstvene za potpisivača. Popisujemo značajke i samu implementaciju njihova izvlačenja.

Bazni kut potpisa

Baza potpisa je zamišljeni pravac na kojem potpis "leži". Kut tog pravca prema horizontali zovemo *bazni kut potpisa*, u oznaci Θ. To je kut koji maksimizira omjer maksimalne horizontalne projekcije potpisa i širine same projekcije. Neka je I_θ slika zarotirana za kut $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Definiramo:

$$\begin{aligned} P_H(i,\theta) &= \sum_{j=1}^{n} I_\theta(i,j), \\ H(\theta) &= \max\{P_H(i,\theta) \mid i \in \{1,\ldots,m\}\}, \\ W(\theta) &= \text{broj netrivijalnih elemenata u } P_H(i,\theta), \\ \rho(\theta) &= \frac{H(\theta)}{W(\theta)}. \end{aligned}$$

Sada definiramo Θ kao vrijednost kuta za koju se postiže maksimalna vrijednost funkcije ρ. Vidimo da će vrijednost te funkcije biti manja za bilo koji drugi kut. Algoritam provodimo tako što gledamo potencijalne kutove θ iz segmenta $[-\frac{\pi}{2}, \frac{\pi}{2}]$ te računamo vrijednost funkcije ρ za svaki cjelobrojni kut (u stupnjevima) iz tog segmenta. Na kraju, za Θ uzmemo cjelobrojni kut koji je dao maksimalnu vrijednost gore spomenute funkcije.

Omjer slike

Prvo lako odredimo granični okvir slike (*bounding box*). Potom definiramo omjer slike (*aspect ratio*) kao omjer širine i visine graničnog okvira.

Normalizirana površina potpisa

Također se jednostavno definira kao omjer površine potpisa i ukupne površine graničnog okvira. Površina potpisa se lako dobije sumiranjem vrijednosti intenziteta svih piksela na slici potpisa.

Centar intenziteta

Neka je Δ površina graničnog okvira potpisa, te neka su, redom, P_H i P_V horizontalna i vertikalna projekcija. *Centar intenziteta* je dvodimenzionalna točka (X, Y) s koordinatama:

$$X = \frac{1}{\Delta} \sum_{j=1}^{n} P_V(j) \cdot j,$$

$$Y = \frac{1}{\Delta} \sum_{i=1}^{m} P_H(i) \cdot i.$$

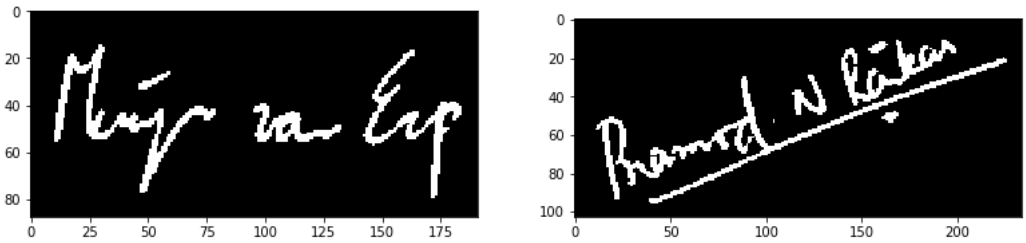
Kut nagiba između centara intenziteta dviju polovica slika

Ovo je najtrivijalnija značajka za implementaciju, budući da koristi cijelu funkcionalnost prethodne, samo na horizontalnim polovicama graničnog okvira potpisa.

Broj rubnih točaka i sjecišta

Rubnu točku definiramo kao piksel intenziteta 1, kojem je samo 1 susjed piksel intenziteta 1. *Sjecište* se definira kao piksel intenziteta 1, kojem su 3 ili više susjeda pikseli intenziteta 1.

Primjer 1.2.1. *Pokažimo na primjeru dvije slike koje su vrijednosti njihovih značajki.*



Slika 1.6: Primjer dva potpisa

Značajka	Slika 1	Slika 2
Bazni kut	2	-23
Omjer slike	2.703	2.548
Normalizirana površina	0.112	0.115
Centar inteziteta	(91.633, 45.962)	(111.157, 50.080)
Kut nagiba centara polovica	10.995	-4.884
Broj rubnih točaka	304	467
Broj sjecišta	1881	2902

Tablica 1.1: Vrijednosti značajki prikazanih slika

1.3 Korišteni modeli

Sada smo spremni izgraditi skupove za treniranje te ih proslijediti modelima za trening. Proučavamo 3 modela: model temeljen na euklidskoj udaljenosti (ukratko, *euklidski*, u nastavku rada), logistička regresija i stabla odlučivanja.

Euklidski model

Jedan od problema, na kojeg smo naišli na samom početku, je varijacija potpisa iste osobe. Ideja ovog modela je izgraditi prosječni (očekivani) potpis dane osobe. Prvo provodimo ekstrakciju značajki svih potpisa te računamo aritmetičku sredinu i standardnu devijaciju, za svaku od značajki koje spremamo kao referentne podatke o očekivanom potpisu. Sada se proces verifikacije prirodno nameće. Za svaki potpis kojeg verificiramo, računamo sumu normaliziranih udaljenosti između značajki danog i očekivanog potpisa. Time smo dobili mjeru ”sličnosti” među potpisima. Prelazi li ona preddefinirani prag, potpis nije valjan, inače jest.

Preciznije, neka su μ_i i σ_i aritmetička sredina i standardna devijacija i-te značajke, te neka je dano $n \in \mathbb{N}$ značajki. Neka su F_i vrijednosti značajki proizvoljnog potpisa. Udaljenost (sličnost) potpisa od očekivanog potpisa definiramo formulom:

$$\delta = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{F_i - \mu_i}{\sigma_i} \right)^2 .$$

Preostaje pitanje izbora praga. Nedovoljno velik prag može uzrokovati odbacivanje valjanih potpisa, dok nedovoljno malen prag može uzrokovati prihvaćanje krivotvorina. Za svaki od potpisa dane osobe, računamo udaljenost od očekivanog potpisa te, na temelju toga, lako dolazimo do prosjeka i standardne devijacije udaljenosti validnih potpisa od očekivanog potpisa. Označimo s μ i σ tako dobivenu prosječnu udaljenost i njezinu standardnu devijaciju. U eksperimentu ćemo testirati performanse modela na pragovima iz skupa $\{\mu + \sigma, \mu + 2\sigma, \mu + 3\sigma\}$.

Napomena 1.3.1. *Motivacija za izbor takvih pragova dolazi od empirijskog pravila, još poznatog kao pravilo* 68–95–99.7. *Neka je X normalno distribuirana slučajna varijabla s očekivanjem μ i standardnom devijacijom σ. Tada vrijedi sljedeće:*

$$Pr\,(\mu - \sigma \leq X \leq \mu + \sigma) \approx 0.6827,$$
$$Pr\,(\mu - 2\sigma \leq X \leq \mu + 2\sigma) \approx 0.9545,$$
$$Pr\,(\mu - 3\sigma \leq X \leq \mu + 3\sigma) \approx 0.9973.$$

Pretpostavimo li normalnu distribuiranost udaljenosti valjanih potpisa od očekivanog, vidimo da će većina njih biti u segmentima $[0, \mu + \sigma]$, $[0, \mu + 2\sigma]$ *ili* $[0, \mu + 3\sigma]$.

Logistička regresija

Logistička regresija je jedan od najčešće korištenih modela u području strojnog učenja s nadzorom. Za razliku od prethodnog modela, koji je praktički osmišljen u svrhu ovog rada i implementiran od temelja, već gotove implementacije logističke regresije postoje. Međutim, mi ćemo ovdje obraditi matematičku pozadinu samog modela, radi boljeg razumijevanja utjecaja na proces verifikacije.

Neka je $n \in \mathbb{N}$ broj značajki svakog podatka $x \in \mathbb{R}^n$. Podatke koje trebamo aproksimirati logističkom regresijom spremamo u kompaktnom obliku matrice $X \in M_{m,n}(\mathbb{R})$, gdje je m broj podataka koje aproksimiramo. U našem slučaju, svaki podatak reprezentira uređenu n-torku značajki spomenutih na početku poglavlja. Dodajemo novu informaciju na svaki podatak, koju ćemo zvati *oznaka* (*label*). Ako dani podatak predstavlja valjani potpis, dajemo mu oznaku 1, inače 0. Formiramo vektor-stupac oznaka $y \in M_{m,1}(\{0, 1\})$, gdje oznaka y_i odgovara podatku iz reda i u matrici X.

Kao što vidimo, cilj nam je pronaći funkciju $f : \mathbb{R}^n \to \mathbb{R}$ koja najbolje aproksimira odnos između podataka i njihovih oznaka. U slučaju logističke regresije, to je *logistička* funkcija.

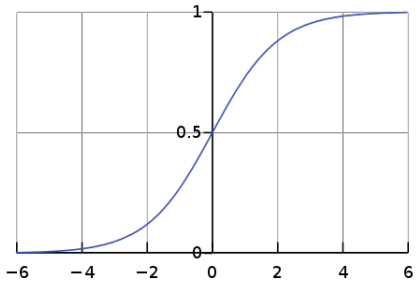
Definicija 1.3.2. *Logistička (sigmoidna) funkcija je $f : \mathbb{R} \to \mathbb{R}$ definirana kao:*

$$f(x) = \frac{1}{1 - e^{-x}}.$$

Tražena aproksimacijska funkcija bit će kompozicija jednostavne linearne regresijske funkcije i logističke funkcije. Neka su $w_1, \ldots, w_n \in \mathbb{R}$. Sada definiramo:

$$z := g(x_1, \ldots, x_n) = w_1 x_1 + \cdots + w_n x_n.$$

Konačna aproksimacijska funkcija je $h := f \circ g$. Primijetimo da ta funkcija poprima vrijednosti u segmentu $[0, 1]$. Također, na grafu vidimo da je logistička funkcija neparna oko točke $\frac{1}{2}$. To nas navodi na interpretaciju vrijednosti funkcije kao vjerojatnosti da danom podatku oznaka iznosi 1, u našem slučaju, da je to valjan potpis. Sada lako vidimo kako algoritam funkcionira. Ako je vrijednost aproksimacijske funkcije iznad $\frac{1}{2}$, deklariramo potpis kao valjan, a inače ga deklariramo kao krivotvorinu.



Slika 1.7: Graf logističke funkcije

Sada treba naći težine $w_1, \ldots, w_n$ s kojima h najbolje aproksimira dane podatke. Prije svega, definiramo funkciju greške.

Definicija 1.3.3. *Neka je $m \in \mathbb{N}$ broj podataka koje aproksimiramo, h aproksimacijska funkcija i $y \in M_{m,1}(\{0, 1\})$ vektor-stupac oznaka. Također, uvodimo oznaku x_i za i-ti stupac matrice X. Definiramo funkciju greške (cost function) $J : \mathbb{R}^n \to \mathbb{R}$ kao:*

$$J(w_1, \ldots, w_n) = \frac{1}{m} \sum_{i=1}^{m} \left(-y_i \log(h(x_i)) - (1 - y_i) \log(1 - h(x_i))\right). \tag{1.1}$$

Na prvi pogled, ne vidimo zašto ova funkcija daje mjeru odstupanja aproksimacijske funkcije od stvarnih oznaka. Međutim, promotrimo slučajeve za oznaku y_i. Ako je $y_i = 1$, od sumanda nam preostaje $-\log(h(x_i))$. Znamo da je $h(x_i) \leq 1$, stoga je sumand, zapravo, pozitivan, što je dobro, obzirom da aproksimacijska greška mora biti pozitivna. Također, vidimo da se sumand povećava, kako se $h(x_i)$ približava nuli. To u potpunosti ima smisla, jer se, zapravo, aproksimirana vrijednost $h(x_i)$ udaljava od prave vrijednosti y_i. Ista analiza vrijedi i za slučaj kad je $y_i = 0$.

Sada se problem pronalaženja težina aproksimacijske funkcije sveo na problem minimizacije funkcije greške J. Pronalaženje stacionarnih točaka na klasičan način ne dolazi u obzir, zbog kompleksnosti same funkcije i utjecaja na brzinu izvršavanja algoritma. Stoga primjenjujemo poznatu metodu nelinearne optimizacije koja se zove *gradijentni spust*. Algoritam kratko opisujemo sljedećim pseudokodom:

1. Na nasumičan način izaberi početni vektor težina $w := (w_1, \ldots, w_n)$.
2. Izračunaj gradijent $\nabla J(w)$ koji predstavlja smjer najbržeg rasta funkcije J u točki w.
3. Ažuriraj težine sljedećom formulom: $w = w - \alpha \cdot \nabla J(w)$, $\alpha \in \mathbb{R}^+$. Ovime se, zapravo, pomičemo u smjeru najbržeg pada funkcije J u točki w.
4. Ako udaljenost između stare i nove težine padne ispod preddefinirane konstante ε, završi algoritam. Inače se vrati na korak (2).

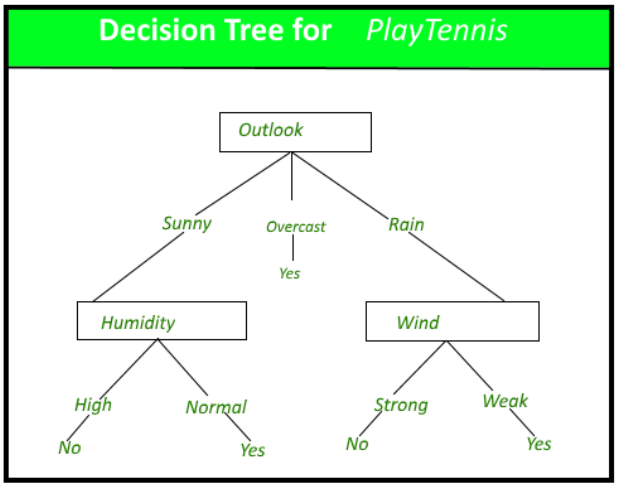
Napomena 1.3.4. *Primjetimo da izbor koeficijenta α za algoritam gradijentnog spusta najviše utječe na performansu algoritma. Izaberemo li preveliki α, postoji mogućnost da ćemo "promašiti" minimum funkcije i pobjeći iz "udoline" u kojoj smo se nalazili. Obratno, ako je α premalen, algoritmu će trebati više vremena da dođemo do optimalnog rješenja. U realnim primjenama, izbor koeficijenta α se provodi grid search algoritmom, kojim se gradijentni spust pokreće s raznim vrijednostima α te se, na kraju, bira ona na kojoj je spust konvergirao u najmanju vrijednost funkcije J.*

Stabla odlučivanja

Stabla odlučivanja su, kao i logistička regresija, popularan izbor za probleme binarne klasifikacije, posebice zbog visoke interpretabilnosti. Opet ćemo obraditi teoretsku pozadinu iza modela i vidjeti kako je povezana s našim problemom.

Neka je $n \in \mathbb{N}$ broj značajki, $m \in \mathbb{N}$ broj podataka koji su kompaktno spremljeni u matrici $X \in M_{m,n}(\mathbb{R})$, te neka je $y \in M_{m,1}(\{0, 1\})$ vektor-stupac oznaka. Mjesto funkcije koja aproksimira oznake podataka zauzima stablasta struktura, koja hijerarhijski, na temelju vrijednosti značajki, određuje oznaku podatka. Svaki čvor reprezentira značajku na

temelju koje dijelimo podatke, dok bridovi reprezentiraju uvjet na značajku iz čvora koji je izvor brida.



Slika 1.8: Primjer stabla odlučivanja za odluku o igranju tenisa

Sada možemo izvući primitivnu verziju algoritma za stvaranje stabla odlučivanja. U svakoj iteraciji izaberemo neku od značajki te podijelimo primjere za učenje prema vrijednostima te značajke. Potom stvaramo podstabla tog čvora na isti način, rekurzivno. Proces se zaustavlja kada svi primjeri u podstablu imaju istu oznaku. Formalnije:

1. Kreiraj osnovni čvor stabla. Ako su svi primjeri iste oznake, tada postavi stablo kao osnovni čvor s tom oznakom. Ako su sve značajke iskorištene, vrati osnovni čvor s najbrojnijom oznakom.

2. Dok još postoje značajke koje nismo iskoristili u stablu:

   a) Odredi značajku F_i koja **najbolje dijeli** skup podataka.

   b) Za svaku vrijednost f_j značajke F_i, dodaj novu granu koja odgovara uvjetu $F_i = f_j$ i rasporedi podatke koji zadovoljavaju taj uvjet u to podstablo. Ako vrijednosti značajke F_i nisu diskretne, tada provodimo diskretizaciju podataka. Zapravo, jednostavno dijelimo vrijednosti s pomno odabranim pragom, koji najbolje dijeli podatke.

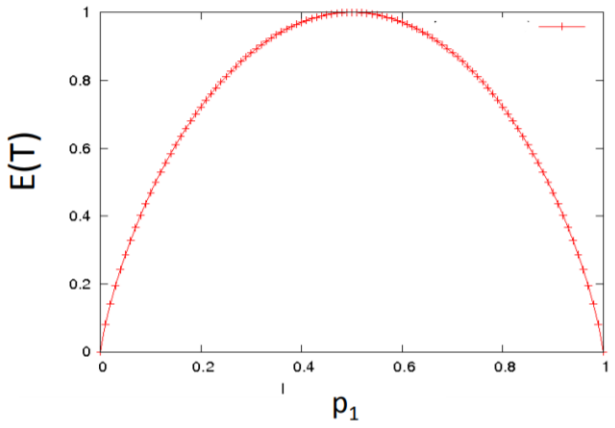
Većina algoritma je jednostavna za razumijevanje i implementaciju. Međutim, jedna stvar iskače i nije dovoljno dobro objašnjena — kako izabrati značajku koja najbolje dijeli skup podataka. Definiramo mjeru za porast informacije korištenjem značajke i po njoj možemo dobro urediti značajke. No, prije same definicije, uvodimo pojam entropije podataka.

Definicija 1.3.5. *Neka je* $X \in M_{m,n}(\mathbb{R})$ *skup primjera i* $y \in M_{m,1(\{0,1\})}$ *skup odgovarajućih oznaka. Definiramo entropiju skupa podataka X kao:*

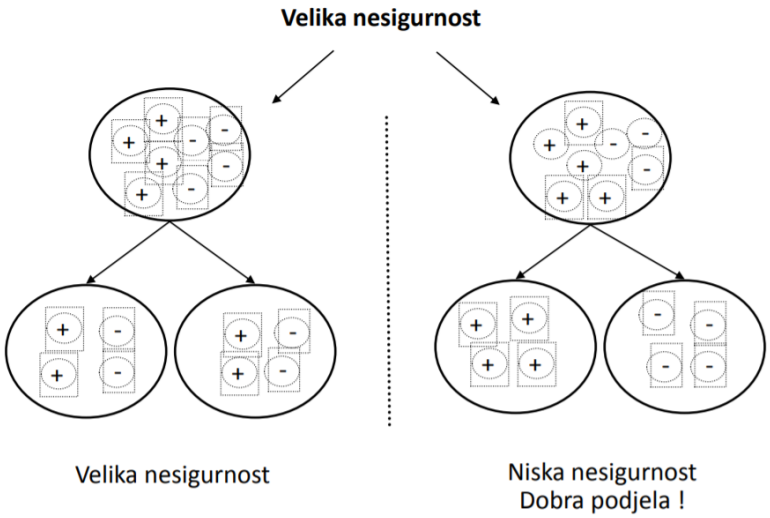
$$E(p_1, p_0) = -p_1 \log_2(p_1) - p_0 \log_2(p_0),$$

gdje p_1 *udio primjera oznake* 1 *u skupu X, dok je* p_0 *udio primjera oznake* 0*.*

Napomena 1.3.6. *Primijetimo da se maksimalna entropija postiže za* $p_1 = p_0 = \frac{1}{2}$*, a najmanja za* $p_1 = 1, p_0 = 0$ *ili* $p_1 = 0, p_0 = 1$*.*



Slika 1.9: Entropija u ovisnosti o udjelu oznaka 1 u skupu



Slika 1.10: Ideja rezultata kojeg želimo postići dobrim izborom značajki

Sada možemo definirati mjeru porasta informacije sa značajkom.

Definicija 1.3.7. *Neka vrijede iste pretpostavke kao u definiciji entropije. Neka je N_F broj mogućih vrijednosti značajke F, N broj podataka u čvoru kojeg dijelimo, te n_i broj podataka koji zadovoljavaju uvjet $F = f_i$, za $i \in \{1, \ldots, N_F\}$. Porast informacije sa značajkom F na skupu podataka X definiramo kao:*

$$InfoGain(F) = E(p_1, p_0) - \frac{1}{N} \sum_{i=1}^{N_F} n_i \cdot E(p_{1,i}, p_{0,i}),$$

gdje su $p_{1,i}$ i $p_{0,i}$ udjeli podataka s oznakom 1 i 0 u i-toj grani.

Napomena 1.3.8. *Računanje mjere porasta informacije može pomoći pri korištenju modela koji nisu nužno stabla odlučivanja. Ako imamo velik broj značajki i želimo ga smanjiti zbog prevelike kompleksnosti modela, na ovaj način možemo odrediti koliko koja značajka pridonosi modelu.*

1.4 Eksperiment i rezultati

Prikupili smo skup potpisa za eksperiment s Kaggle stranice. Sastoji se od ukupno 300 slika potpisa koji pripadaju 30 ljudi. Imamo 10 potpisa po svakoj osobi, 5 pravih i 5 krivotvorenih. Više detalja o implementaciji eksperimenta se nalazi u sljedećem github repozitoriju: `https://github.com/tubibuto/Offline-Signature-Verifier`.

Euklidski model je skupio podatke i izračunao prosjeke i standardne devijacije samo iz pravih potpisa te odredio pragove sličnosti na temelju empirijskog pravila. Uspoređujemo performanse na 3 praga i prilažemo rezultate u sljedećoj tablici. Primijetimo da je odnos *false negative* i *false positive* metrika negativno koreliran, kao što smo i predvidjeli na početku. Najbolje peformanse se postižu za prag $\mu + 2\sigma$, koji nije ni prestrog ni preblag.

Prag	$\mu + \sigma$	$\mu + 2\sigma$	$\mu + 3\sigma$
FN	0.233	0.0	0.0
FP	0.033	0.066	0.111
Preciznost	0.734	0.934	0.889

Tablica 1.2: Metrike euklidskog modela u odnosu na izabrani prag prihvaćanja. FN označava *false negative*, a FP *false positive*.

Modeli logističke regresije i stabla odlučivanja su trenirani na sličan način jedan drugom te dijele gotovo cijelu implementaciju. Stvaramo model za svaku osobu te, za trening

i test skup, dijelimo skup potpisa osobe u 2 omjera: 50%–50% i 80%–20%. Ponavljamo tu podjelu i trening 10 puta te uzimamo prosjek metrika svih modela, radi vjerodostojnosti na malom skupu podataka (10 potpisa po osobi). Rezultate prilažemo u donjoj tablici.

	Log 50	Log 80	DT 50	DT 80
FN	0.128	0.075	0.158	0.063
FP	0.09	0.096	0.115	0.117
Preciznost	0.782	0.829	0.72	0.85

Tablica 1.3: Metrike logističke regresije (Log) i stabla odlučivanja (DT). Brojevi uz imena modela označavaju udio podataka u postotcima u trening skupu.

Prva stvar koju primjećujemo je veći porast u preciznosti s povećanjem broja podataka kod stabla odlučivanja, u odnosu na logističku regresiju. Također, generalno vidimo da se oba modela slabije nose s prepoznavanjem lažnih potpisa, budući da im se *false positive* metrika ne poboljšava s povećanjem broja podataka u treningu. Logistička regresija, generalno, bolje prepoznaje krivotvorine, međutim, stabla odlučivanja imaju znatno veće poboljšanje u prepoznavanju validnih potpisa s povećanjem broja podataka.

Možemo zaključiti puno korisnih informacija uspoređujući modele koji su trenirani na specifičnim značajkama za potpis. Euklidska metoda, koja je najmanje kompleksna, ima najveću preciznost i najbolje *false positive* i *false negative* metrike, nakon ugađanja praga prihvaćanja. Međutim, modeli logističke regresije i stabla odlučivanja imaju potencijal mjeriti se s euklidskom metodom, ako povećamo broj podataka za treniranje po svakoj osobi. To se lako vidi u povećanju preciznosti i smanjenju *false negative* metrika kod oba modela. Problem smanjenja *false positive* metrike kod prethodna dva modela ostaje otvoreno pitanje.

Poglavlje 2

Verifikacija temeljem cijele slike

Prethodno poglavlje se fokusiralo na pristup izvlačenja malog broja ključnih značajki te treniranje modela manje kompleksnosti. Također, svakoj osobi smo dodijelili po jedan model, koji prepoznaje njezine potpise. Za takve sustave verifikacije potpisa kažemo da su *ovisni o korisniku* (*writer-dependent*).

Sljedeći pristup kojeg promatramo je verifikacija potpisa na temelju cijele slike. Proces izvlačenja značajki je redundantan, budući da će intenzitet svakog piksela na slici biti jedna značajka. Očito ćemo na kraju završiti s velikim brojem značajki te će bilo koji model, kojeg koristimo, postati prekompleksan i podložan prenaučenosti (*overfit*). Ovom pojavom se detaljnije bavimo u nastavku poglavlja. Obrada slike prije treniranja je identična kao u prethodnom poglavlju, stoga počinjemo s modelima koje koristimo za verifikaciju, gdje detaljno obrađujemo njihovu matematičku pozadinu.

2.1 Neuronske mreže

Neuronske mreže su danas najpopularniji izbor za rješavanje problema u području računalnog vida (*Computer Vision*). Njihovo proučavanje je postalo toliko rašireno, da bi se i jedan diplomski rad mogao baviti samo njima. Mi ćemo dati kratki, ali potpuni pregled, radi potpunog razumijevanja od strane čitatelja koji nisu upoznati s područjem.

Perceptroni

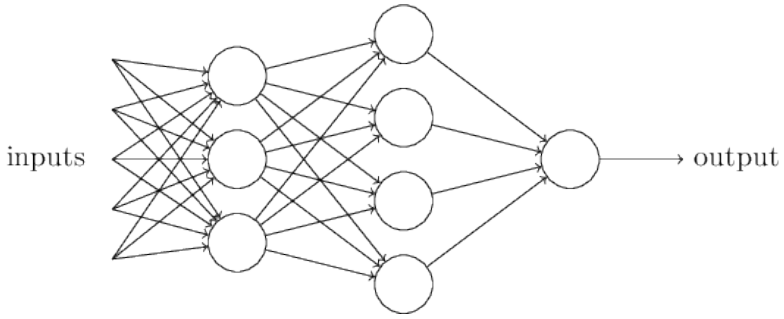
Započinjemo s jednostavnijim primjerom *neurona*, zvanim perceptron.

Definicija 2.1.1. *Neka je $n \in \mathbb{N}$, te $x_1, \ldots, x_n \in \mathbb{R}$, $w_1, \ldots, w_n \in \mathbb{R}$ i $t \in \mathbb{R}$. Perceptron je funkcija $p : \mathbb{R}^n \to \mathbb{R}$ definirana kao:*

$$p(x_1, \ldots, x_n) = \begin{cases} 1, & za \ \sum_{i=1}^{n} w_i x_i \leq t, \\ 0, & za \ \sum_{i=1}^{n} w_i x_i > t. \end{cases}$$

Vrijednosti $x_1, \ldots, x_n$ zovemo ulaznim varijablama perceptrona, $w_1, \ldots, w_n$ su težine perceptrona, dok je t prag prihvaćanja perceptrona.

Perceptron možemo interpretirati kao funkciju koja donosi odluku temeljem ulaznih podataka, od kojih svaki ima određenu težinu u odluci. Međutim, ta odluka je jednostavna, budući da se možemo u jednom trenutku odlučiti za samo jednu kombinaciju težina. Sofisticiranije odluke se mogu donositi ako povežemo više instanci perceptrona u mrežu. Time dobivamo više kombinacija težina za jednu odluku, čije rezultate možemo nanovo kombinirati s dodatnom kombinacijom težina. Sljedeća slika savršeno vizualizira taj odnos. Prvi sloj od 3 perceptrona prima ulazne podatke te donosi 3 jednostavne odluke. Potom šalju rezultate perceptronima u drugom sloju. Oni donose odluku koja je kompleksnija i na višoj razini apstrakcije. Dodavanjem više slojeva možemo donositi još kompleksnije odluke.



Slika 2.1: Vizualizacija mreže perceptrona

Vrijeme je da pojednostavnimo način na koji definiramo perceptron. Označimo vektore ulaznih podataka i težina, redom, kao $x := (x_1, \ldots, x_n)$ i $w := (w_1, \ldots, w_n)$. Također, definiramo *pristranost* (*bias*) kao $b := -t$. Sada se zapis perceptrona mijenja:

$$p(x) = \begin{cases} 1, & \text{za } w \cdot x + b \leq 0, \\ 0, & \text{za } w \cdot x + b > 0. \end{cases}$$

Glavni cilj, nakon uvođenja perceptrona, je osmisliti algoritam kojim bismo pronašli optimalne vektore težina i pristranosti za sve perceptrone u mreži.

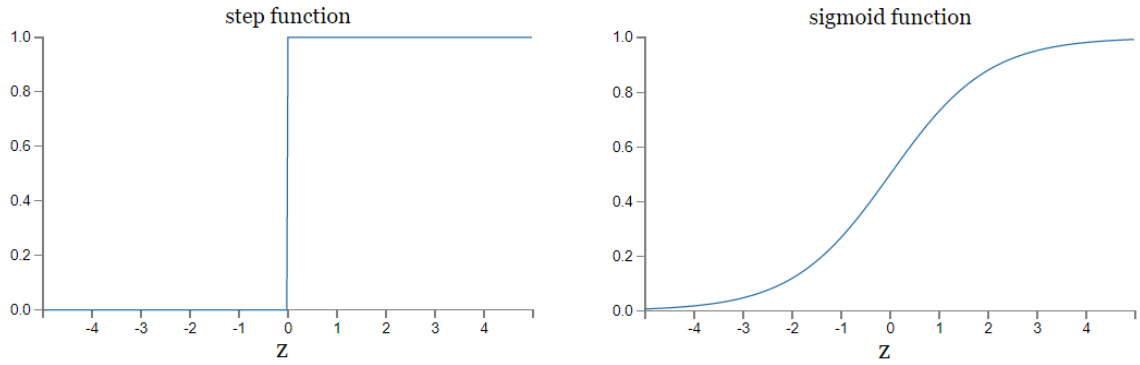
Sigmoidni neuroni

Nelinearna optimizacija gradijentnog spusta se oslanja na korištenje gradijenta funkcije. Perceptroni stvaraju veliki problem pri takvoj optimizaciji, budući da oni nisu neprekidni na cijeloj domeni, čime nisu ni diferencijabilni na cijeloj domeni. Stvara se potreba za uvođenjem novog neurona koji je diferencijabilan i zadržava intuiciju samog perceptrona.

Definicija 2.1.2. *Neka su* $n \in \mathbb{N}$ *i* $x, w \in \mathbb{R}^n$. *Sigmoidni neuron je funkcija* $f : \mathbb{R}^n \to \mathbb{R}$ *definirana kao:*

$$f(x) = \frac{1}{1 + \exp(-w \cdot x - b)} = \sigma(w \cdot x + b).$$

Napomena 2.1.3. *Primijetimo da je sigmoidni neuron, zapravo, sigmoidna funkcija (ovdje označena sa* σ*), s argumentom* $w \cdot x + b$. *Također, pogledamo li grafove perceptrona i sigmoidnog neurona, vidimo da smo dobili izglađenu step funkciju.*



Slika 2.2: Graf perceptrona i sigmoidnog neurona

Treniranje neuronskih mreža

Neka je $n \in \mathbb{N}$ broj značajki svakog podatka $x \in \mathbb{R}^n$. Podatke koje aproksimiramo neuronskom mrežom spremamo u matricu $X \in M_{m,n}(\mathbb{R})$, gdje je $m \in \mathbb{R}$ broj podataka. Neka je $y \in M_{m,1}(\mathbb{R})$ vektor-stupac oznaka. Neka neuronska mreža ima 3 sloja. Prvi sloj zovemo *ulazni* sloj, zadnji zovemo *izlazni* sloj, dok ostale zovemo *skriveni* slojevi. U nastavku promatramo jednostavni primjer mreže s 3 sloja, međutim, postupak treninga se lako generalizira i na veći broj slojeva.

Neka su $l_1, l_2, l_3 \in \mathbb{N}$ brojevi neurona u svakom od slojeva. Označimo s h funkciju koja reprezentira danu neuronsku mrežu. Očito vrijedi $h : \mathbb{R}^{l_1} \to \mathbb{R}^{l_3}$. Neka je $x = (x_1, \ldots, x_n)$. Od sada ćemo vektore reprezentirati matricama. Kao što znamo, srednji sloj ima l_2 neurona, stoga mora dati i toliko rezultata. Također, svaki neuron u tom sloju ima l_1 težina,

budući da mu ih šalju neuroni iz prethodnog sloja. Sve težine možemo kompaktno spremiti u matricu težina $W \in M_{l_2,l_1+1}(\mathbb{R})$. Primjetimo da je dodatna težina, zapravo, *pristranost (bias)* svakog neurona. Također, svaki red matrice težina odgovara svakom neuronu. Vrijednosti koje daju neuroni srednjeg sloja se sada računaju kao:

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{l_2} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{1,1} & \dots & w_{1,l_1+1} \\ \vdots & \ddots & \vdots \\ w_{l_2,1} & \dots & w_{l_2,l_1+1} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_{l_1} \end{bmatrix} \right).$$

Na sličan način se računaju i vrijednosti koje daju neuroni u izlaznom sloju. Označimo s x vektor-stupac ulaznih podataka, s a vektor-stupac vrijednosti koje daje srednji sloj i s W_1, W_2 matrice težina srednjeg i izlaznog sloja. Sada lako izlazi formula za funkciju h:

$$h(x) = \sigma\left(W_2 \begin{bmatrix} 1 \\ a \end{bmatrix}\right) = \sigma\left(W_2 \begin{bmatrix} 1 \\ \sigma\left(W_1 \begin{bmatrix} 1 \\ x \end{bmatrix}\right) \end{bmatrix} \right).$$

Vrijednosti koje daje zadnji sloj su u segmentu $[0, 1]$ te ih možemo interpretirati kao vjerojatnosti da dani primjer x pripada klasi $k \in \{1, \dots, l_3\}$. Primjer deklariramo da pripada onoj klasi čiji izlazni neuron je dao najveću vrijednost.

Napomena 2.1.4. *Bitno je napomenuti da suma vrijednosti koje daju izlazni neuroni ne iznosi nužno* 1. *Međutim, možemo se opredijeliti da te izlazne vrijednosti dodatno normaliziramo, tako da jednostavno izračunamo omjer same vrijednosti i sume svih vrijednosti. U literaturi nalazimo tu funkciju pod imenom softmax.*

Uvodimo funkciju greške koja je ista kao i (1.1). Sada još treba naći matrice težina neuronske mreže koje minimiziraju danu funkciju greške. Kao i prije, korištenje analize i traženje analitičkog rješenja bi bio mukotrpan posao, budući da je broj varijabli funkcije greške jednostavno prevelik. Koristimo gradijentni spust za traženje minimuma.

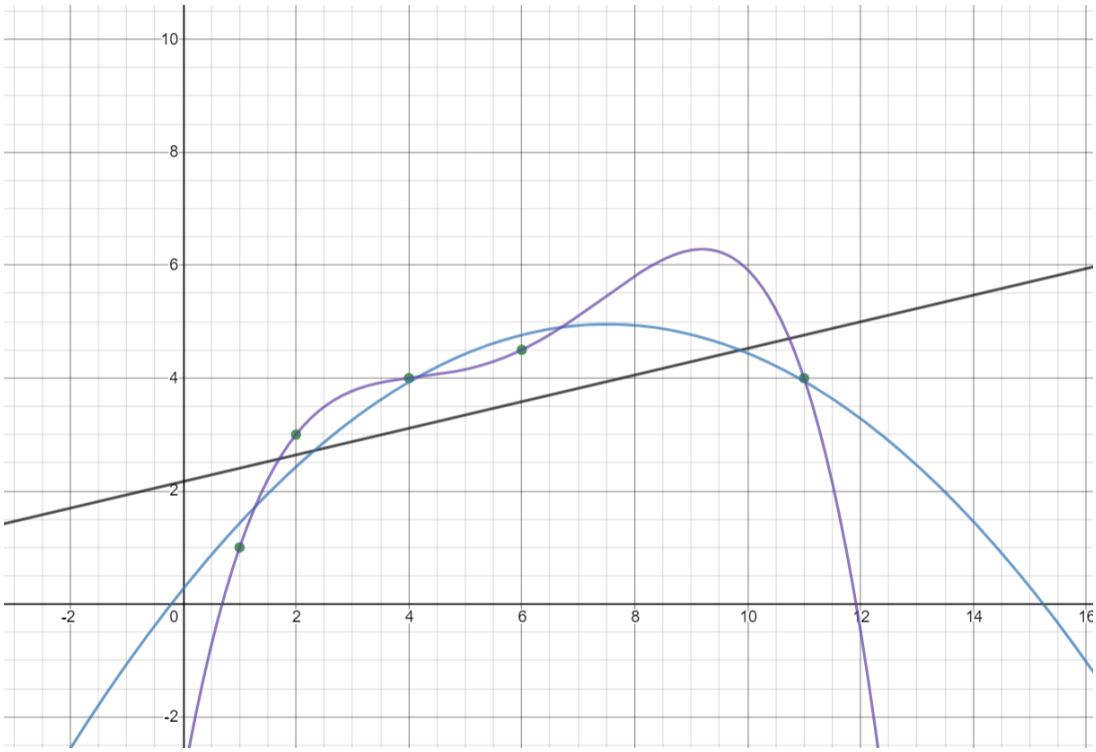
Problem prenaučenosti

Jedan od velikih problema neuronskih mreža i drugih kompleksnih modela je fenomen prenaučenosti (*overfit*). Ideja fenomena je prilično jednostavna. Kompleksnost modela raste s porastom broja značajki. To je očito, budući da je značajka jedan od parametara aproksimacijske funkcije koja reprezentira model. Imamo li velik broj parametara, moći ćemo reprezentirati gotovo svaki podatak iz skupa za treniranje. Dapače, za dovoljno velik broj parametara, moći ćemo naći aproksimaciju čija greška iznosi 0 na skupu za treniranje. Međutim, pokazuje se da na novim podacima, na kojima model nije bio treniran, performanse drastično padaju. Razlog te pojave ćemo objasniti kroz jednostavan primjer.

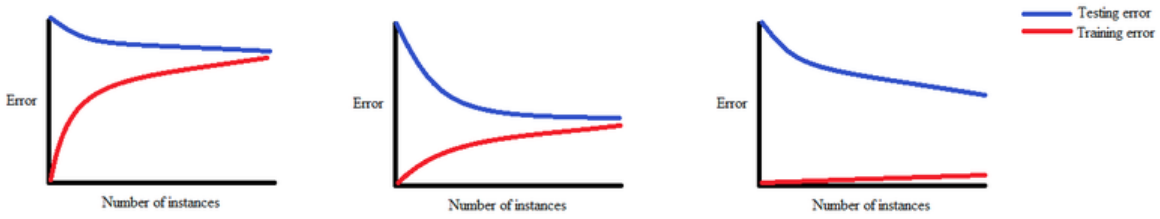
Primjer 2.1.5. *Neka su* $x_1, x_2, x_3, x_4, x_5 \in \mathbb{R}^2$. *Promotrimo sljedeće* 3 *aproksimacijske funkcije za zadanih* 5 *točaka:*

$$f_1(x) = w_0 + w_1 x,$$
$$f_2(x) = w_0 + w_1 x + w_2 x^2,$$
$$f_3(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4.$$

Na sljedećem grafu vidimo te funkcije s optimalnim izborom težina. Zelenim točkama su označene točke $x_1, \ldots, x_5$, *crnom bojom funkcija* f_1, *plavom* f_2 *i ljubičastom* f_3. *Prije svega, vidimo kako linearna funkcija* f_1 *ima najveću grešku na skupu za trening i ona će teško generalizirati podatke, jer je prejednostavna. Za nju kažemo da pati od pojave podnaučenosti (underfit). Funkcija* f_3, *koja je, zapravo, interpolacijski polinom za dane točke, savršeno aproksimira sve točke. Međutim, ona će loše generalizirati nove podatke iz istog izvora, budući da oni imaju kvadratnu ovisnost među sobom. Za nju kažemo da pati od pojave prenaučenosti (overfit). Kvadratna funkcija* f_2 *ima nešto veću grešku na trening skupu, međutim, ima bolju sposobnost generalizacije. Često se spominje usporedba prenaučenosti s "učenjem napamet", jer i ono, također, ne dovodi do dobre generalizacije.*



Slika 2.3: Usporedba grafova regresijskih funkcija



Slika 2.4: Usporedba krivulja učenja koje prikazuju ovisnost greške o veličini skupa za treniranje. Prva slika prikazuje model koji pati od podnaučenosti, druga prikazuje model koji ne pati od nijedne od pojava, dok treća predstavlja model koji pati od prenaučenosti.

Jedan očiti način borbe protiv prenaučenosti je povećanje skupa za treniranje. Međutim, to nije uvijek moguće, posebice na našem primjeru verifikacije potpisa, gdje se pretpostavlja da nećemo imati pristup više od 50 potpisa neke osobe. Druga dva načina dajemo u sljedeća dva potpoglavlja.

Regularizacija

Obradit ćemo najpoznatiji oblik regularizacije, *L2 regularizaciju*, još poznatu pod nazivom *weight decay*. Ideja je dodati novi član u funkciju greške, kojim bismo smanjili utjecaj kompleksnosti samog modela:

$$J(w_1, \ldots, w_n) = \frac{1}{m} \sum_{i=1}^{m} (-y_i \log(h(x_i)) - (1 - y_i) \log(1 - h(x_i))) + \frac{\lambda}{2m} \sum_{i=1}^{n} w_i^2. \tag{2.1}$$

Koeficijent $\lambda \in \mathbb{R}^+$ zovemo regularizacijski faktor. Njegovim povećanjem, povećava se i utjecaj regularizacije. Što je veći desni član funkcije greške, to će algoritam gradijentnog spusta preferirati manje težine, budući da će veće težine smanjiti cijelu funkciju cilja samo ako znatno smanje njezin lijevi sumand. Model s manjim težinama će biti manje osjetljiv na promjene i time će mu se kompleksnost smanjiti. Vrijedno je napomenuti da izbor prevelikog faktora λ može dovesti do podnaučenosti modela, budući da mu, u tom slučaju, drastično pada kompleksnost.

Napomena 2.1.6. *L2 regularizacija se može primijeniti na bilo koju drugu funkciju greške J_0 koju odaberemo. Tada je njezin generalni oblik:*

$$J(w) = J_0(w) + \frac{\lambda}{2m} \sum_{i=1}^{n} w_i^2.$$

Napomena 2.1.7. *Napomenimo još da se često koristi i L1 regularizacija, kojoj je oblik:*

$$J(w) = J_0(w) + \frac{\lambda}{m} \sum_{i=1}^{n} |w_i|.$$

Augmentacija skupa za treniranje

Spomenuli smo kako je jedan od načina borbe protiv prenaučenosti povećanje skupa za treniranje. Često je teško nabaviti nove podatke, stoga moramo pribjeći drugim metodama povećanja skupa za treniranje. Augmentacija skupa za treniranje je algoritam kojim radimo kopije podataka iz skupa te ih manipuliramo. Ideja je dobiti podatke koji imaju istu oznaku kao i original, ali je promjena dovoljno velika da ih možemo smatrati novim podacima.

Postoje razne tehnike manipulacija podacima, posebice u području računalnog vida, te ćemo nabrojiti najčešće korištene:

- Proizvoljne rotacije slika. Kod toga treba paziti, budući da bi velike rotacije mogle uzrokovati nesklad sa stvarnim oznakama podataka.
- Translacija slika.
- Iskrivljavanje slika (*skewing*).
- Dodavanje pozadinskih zvukova u problemu prepoznavanja zvuka.

2.2 Metoda potpornih vektora

Metoda potpornih vektora (*Support Vector Machine*, skraćeno, SVM) je još jedna od popularnih metoda koja se primjenjuje u području računalnog vida. Također, pripada modelu strojnog učenja s nadzorom.

Funkcija greške

Prisjetimo se logističke regresije i glavne hipoteze vezane uz njezino promatranje:

$$y = 1 \implies h(x) \approx 1 \text{ i } w \cdot x \gg 0,$$
$$y = 0 \implies h(x) \approx 0 \text{ i } w \cdot x \ll 0.$$

Također, prisjetimo se regularizirane funkcije greške J iz (2.1):

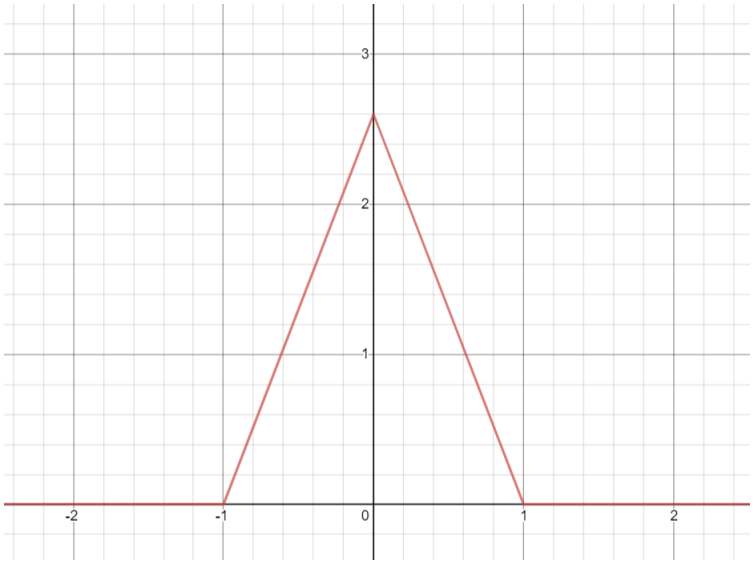
$$J(w_1, \ldots, w_n) = \frac{1}{m} \sum_{i=1}^{m} \left(-y_i \log(h(x_i)) - (1 - y_i) \log(1 - h(x_i))\right) + \frac{\lambda}{2m} \sum_{i=1}^{n} w_i^2.$$

Metoda potpornih vektora modificira funkciju greške tako što mijenja izraz

$$-\log(h(x_i)) = -\log\left(\frac{1}{1 + \exp(-w \cdot x)}\right).$$

Kada je $z := -w \cdot x$ veći od 1, postavljamo dani izraz na 0. Na sličan način modificiramo i drugi izraz, samo što kad je on manji od -1, postavljamo ga na 0. Također, modificiramo ostale slučajeve, tako da logaritamske krivulje zamijenimo pravcima. Formalnije, uzmemo proizvoljnu konstantu $k \in \mathbb{R}^+$, označimo te izraze s $cost_0(z)$ i $cost_1(z)$ i definiramo ih kao:

$$cost_0(z) = \max\{0, k(1+z)\},$$
$$cost_1(z) = \max\{0, k(1-z)\}.$$



Slika 2.5: Graf funkcija $cost_0$ i $cost_1$ koje se, redom, nalaze na lijevoj i desnoj strani

Nadalje uvodimo oznaku $C := 1/\lambda$ i skaliramo funkciju greške, tako da dobivamo:

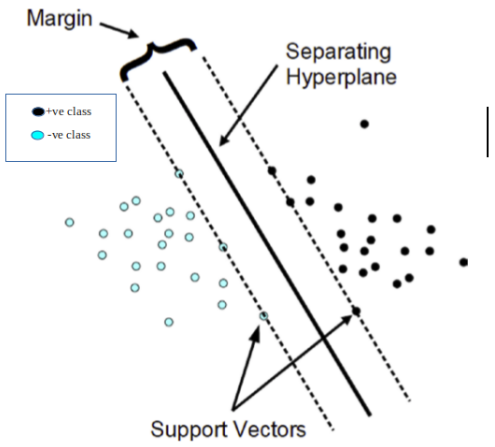
$$J(w) = C \sum_{i=1}^{m} (y_i \cdot cost_1(w \cdot x) + (1 - y_i) \cdot cost_0(w \cdot x)) + \frac{1}{2} \sum_{i=1}^{n} w_i^2.$$

Klasifikacija temeljem velike margine

Većinom se metodu potpornih vektora neformalno zove klasifikacijom temeljem velike margine (*Large Margin Classifier*). Dajemo sljedeću hipotezu na model:

$$y = 1 \implies w \cdot x \geq 1,$$
$$y = 0 \implies w \cdot x \leq -1.$$

Postavimo li faktor C na visoku vrijednost, novodefiniranu funkciju greške moći ćemo optimizirati jedino ako su svi sumandi, osim regularizacijskog, jednaki 0. Tada funkcija greške iznosi $J(w) = \frac{1}{2}\sum_{i=1}^{n} w_i^2$. Budući da je greška strogo pozitivna, vidimo da neće postojati fina granica odluke (*decision boundary*) između skupova primjera s oznakama 0 i 1, već će ona biti na najvećoj mogućoj udaljenosti od pozitivnih i negativnih primjera. Udaljenost granice odluke od najbližeg primjera zovemo *margina*.



Slika 2.6: Prikaz margine između pozitivnih i negativnih primjera

Preciznije, prisjetimo se drugog načina zapisa skalarnog produkta $w \cdot x = p \cdot \|w\|$, gdje je p duljina projekcije vektora x na vektor w. Sada hipoteza poprima oblik:

$$y = 1 \implies p \cdot \|w\| \geq 1,$$
$$y = 0 \implies p \cdot \|w\| \leq -1.$$

Vektor w je okomit na granicu odluke te, ako želimo da je hipoteza istinita, apsolutne vrijednosti projekcija moraju biti što je moguće veće. To je glavni uzrok "velikih" margina.

Jezgre

Glavna ideja je izračunati nove značajke iz originalnih.

Definicija 2.2.1. *Neka je $x \in \mathbb{R}^n$ neki primjer sastavljen od $n \in \mathbb{N}$ značajki i neka je $\sigma \in \mathbb{R}$. Neka su $l_1, \ldots, l_m \in \mathbb{R}^n$ proizvoljne točke u prostoru, koje zovemo orijentiri (landmarks). Gaussove jezgre su funkcije $f_i : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, za $i \in \{1, \ldots, m\}$, definirane kao:*

$$f_i(x, l) = \exp\left(-\frac{\|x - l_i\|^2}{2\sigma^2}\right).$$

Odmah primjećujemo par zanimljivih svojstava. Ako je x na jako maloj udaljenosti od točke l_i, vrijednost jezgre će biti jako blizu 1. Obratno, ako je x jako daleko od l_i, vrijednost jezgre će biti jako blizu 0. Zapravo, Gaussovu jezgru možemo interpretirati kao mjeru sličnosti između dviju točaka. Orijentiri se najčešće izaberu tako da se podudaraju s primjerima iz skupa za treniranje. Tada dobivamo, uz oznaku $f_i := f_i(x_i)$:

$$J(w) = C \sum_{i=1}^{m} (y_i \cdot cost_1(w \cdot f_i) + (1 - y_i) \cdot cost_0(w \cdot f_i)) + \frac{1}{2} \sum_{i=1}^{n} w_i^2.$$

Napomena 2.2.2. *Vrijedi prokomentirati izbor hiperparametara i jezgri za metodu potpornih vektora. Prije svega, prisjetimo se da je $C = 1/\lambda$. Izaberemo li veliki C, model će patiti od prenaučenosti. U suprotnom, ako izaberemo premali C, model će patiti od podnaučenosti. Izbor faktora σ je bitan, budući da utječe na glatkoću varijacije novih značajki. Premali σ će uzrokovati prenaučenost, dok će veliki stvoriti podnaučenost modela. Također, bitna je odluka između korištenja logističke regresije i SVM. SVM ćemo preferirati u slučaju velikog broja podataka, a za manji broj, prije se opredjeljujemo za manje kompleksan model logističke regresije.*

2.3 Eksperiment i rezultati

Skup potpisa, kojeg smo koristili u prethodnom poglavlju, jednostavno je premalen za kompleksne modele opisane u ovom poglavlju. Novi skup smo prikupili s *ICDAR 2009 Signature Verification Competition* stranice. Sadrži ukupno 1564 potpisa od 100 osoba, od kojih je 940 pravo, dok ih je 620 krivotvoreno. Napomenimo da, u ovom slučaju, treniramo globalni model za sve osobe (*writer-independent*), budući da trebamo izvršavati trening na što većem skupu. Oslanjamo se na pretpostavku da su neuronske mreže i metoda potpornih vektora dovoljno kompleksni modeli, da će izgraditi dodatni sloj apstrakcije, u kojem će moći razlikovati potpise ljudi i onda verificirati radi li se o validnom potpisu. Više detalja o implementaciji ovog eksperimenta se nalazi u sljedećem github repozitoriju: `https://github.com/tubibuto/Offline-Signature-Verifier`.

Glavni izazov kod neuronske mreže je izbor idealne topologije. Sve slike smo postavili na veličinu 50×50, stoga ulazni sloj ima 2500 neurona. Očito, izlazni sloj ima 1 neuron, koji daje vjerojatnost da je potpis valjan. Preostaje pitanje topologije skrivenih slojeva. Sljedeća tablica pokazuje preciznosti u ovisnosti o topologije mreže. Napomenimo da skup na kojem se provodio trening nije bio augmentiran, budući da nemamo resursa i vremena za treniranje modela na tako velikom skupu.

Topologija skrivenih slojeva	Preciznost
10	0.798
25	0.806
50	0.819
100	0.826
250	0.826
500	0.832
1000	0.838
100, 25	0.826
100, 100	0.819

Tablica 2.1: Ovisnost preciznosti o topologiji neuronske mreže. Svaki broj označava koliko ima neurona u skrivenom sloju. Par brojeva označava dva skrivena sloja te njihov broj neurona.

Zaključujemo da nam je najbolje odabrati mrežu koja u skrivenom sloju ima 100 neurona, budući da bilo koja kompleksnija mreža ne postiže znantno bolju preciznost. Kod toga smo imali na umu kako vrijeme treniranja mreže kvadratno raste s povećanjem broja neurona u srednjem sloju, budući da je izračunavanje funkcije koja reprezentira mrežu takve složenosti.

Također, koristimo metodu potpornih vektora na istom skupu. Optimizacija je jednostavnija, budući da radimo *grid search* na hiperparametrima modela. Kao što je i očekivano, najbolje performanse se postižu za Gaussovu jezgru. Koeficijent σ jezgre izabrali smo formulom $1/(n\text{Var}(X))$, što je dalo najveću preciznost.

Napomenimo da su oba modela koristila L2 regularizaciju. Konačno, provodimo augmentaciju skupa, provođenjem rotacija slika s kutovima iz skupa $\{-15, -10, -5, 5, 10, 15\}$. Time se skup podataka za treniranje povećao 7 puta. Dajemo konačne rezultate u sljedećoj tablici.

	Neuronska mreža	SVM
FN	0.03	0.046
FP	0.051	0.076
Preciznost	0.919	0.878

Tablica 2.2: Usporedba metrika neuronskih mreža i metode potpornih vektora

Primjećujemo dosta korisnih informacija iz promatranih metrika. Neuronske mreže imaju bolje performanse od metode potpornih vektora na testnom skupu. Slična situacija se oslikava i u sličnim problemima računalnog vida, gdje su većinom neuronske mreže u prednosti. Primjetimo da, kao i kod tablice 1.3, oba modela bolje verificiraju validne potpise, nego što odbijaju krivotvorine. Euklidski model se pokazao kao najbolji u kontroli *false negative* i *false positive* metrika, budući da se one mogu lako kontrolirati mijenjanjem praga prihvaćanja. Euklidski model, također, ima bolju preciznost na testnom skupu. Međutim, obje *false positive* metrike bolje izgledaju u slučaju neuronskih mreža.

Zaključak

Prethodna poglavlja su nam služila kao koristan pregled raznih metoda verifikacije potpisa. Euklidski model je imao uvjerljivo bolje performanse u slučaju verifikacije temeljem specifičnih značajki potpisa. Međutim, svaki korisnik je trebao imati svoju instancu modela. Korištenjem kompleksnijih modela izbjegavamo korak izvlačenja značajki. Također, možemo imati jedan model koji je treniran na čitavom skupu potpisa, neovisno o osobama kojima pripadaju. Neuronske mreže imaju nižu preciznost, ali bolju *false positive* metriku. Ohrabrujuća činjenica je što je augmentacija skupa za treniranje jednostavnim rotacijama dovela do povećanja točnosti od 9.3%. Potencijalni sljedeći koraci bi bili primjena ostalih metoda augmentacije, spominjanih na kraju odjeljka 2.1. Isto tako, vrijedi probati kompleksnije varijante neuronskih mreža, kao što su konvolucijske i rekurzivne, jer bi se, pravilnom augmentacijom, skup za treniranje mogao znatno povećati.

Bibliografija

[1] A. Dhawan i A. R. Ganesan, *Handwritten Signature Verification*, svibanj 2005, https://pdfs.semanticscholar.org/7244/79ac337d5c1262fb151281b900a4f4210426.pdf.

[2] L. G. Hafemann, Sabourin R. i Oliveira L. S., *Offline Handwritten Signature Verification - Literature Review*, 2017, https://arxiv.org/pdf/1507.07909.pdf.

[3] J. Mujahed, N. Al-Najdawi i S. Tedmori, *Offline handwritten signature verification system using a supervised neural network approach*, ožujak 2014, https://www.researchgate.net/publication/269308740_Offline_handwritten_signature_verification_system_using_a_supervised_neural_network_approach.

[4] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015, http://neuralnetworksanddeeplearning.com/.

Sažetak

Verifikacija potpisa je zadatak forenzičke analize dokumenata, kojeg rješavaju posebno trenirani ispitivači dokumenata. Glavno pitanje je provjera podudaranja potpisa s već poznatim potpisima dane osobe. Ovaj rad proučava mogućnost automatizacije tog procesa, budući da trening ispitivača traje godinama.

Prikupljamo *Offline* podatke o potpisu, dakle samo "sirove" slike potpisa, te na temelju piksela trebamo odrediti njihovu ispravnost. Prije svega opisujemo proces obrade slika. Pretvaramo ih u *greyscale* format, invertiramo boje, uklanjamo šumove te, na poslijetku, intenzitete piksela pretvaramo u binarne vrijednosti. Takve obrađene slike su spremne za izvlačenje značajki te njihovu predaju modelima na treniranje.

Sam rad se dijeli na dva poglavlja, od kojih svako daje svoj pristup prema izvlačenju značajki. Prvo uzimamo u obzir pristup gdje izvlačimo malen broj značajki, specifičnih za potpis, te treniramo manje kompleksne modele na njima. Te značajke su bazni kut potpisa, omjer slike, normalizirana površina potpisa, centar intenziteta, kut nagiba između centara intenziteta dviju polovica slika te broj rubnih točaka i sjecišta. Koristimo 3 modela: euklidski, logističku regresiju i stabla odlučivanja. Prvo navodimo matematičku pozadinu svakog, radi boljeg razumijevanja procesa učenja na podacima iz skupa za treniranje. Potom provodimo sam trening i uspoređujemo performanse na testnom skupu. Najbolje performanse ima euklidski model, kojemu je lako naći balans između *false positive* i *false negative* metrika ugađanjem praga prihvaćanja. Logistička regresija i stabla odlučivanja imaju slabije performanse. Međutim, malim povećanjem veličine skupa za treniranje dobivamo poboljšanja u preciznosti, stoga ih je vrijedno promatrati na većim skupovima podataka.

Prethodni pristup je ograničen činjenicom što svakoj osobi dajemo novu instancu modela. Uvođenjem kompleksnijih modela, kao što su neuronske mreže i metoda potpornih vektora, možemo provoditi treniranje na potpisima svih ljudi u eksperimentu. Javlja se potreba za većim skupom za treniranje. Međutim, to nije velik problem, kako skup sada nije ograničen na samo jednu osobu. Kao i u prethodnom poglavlju, opisujemo matematičku pozadinu oba modela. Nailazimo na problem prenaučenosti, kojeg rješavamo primjenom regularizacije i augmentacije skupa za treniranje. Provodimo trening te, nakon optimizacije topologije neuronske mreže i hiperparametara metode potpornih vektora, dolazimo do konačnih performansi. Očekivano, neuronske mreže imaju veću preciznost, ali ne i od

euklidskog modela. Euklidski model ima lošiju *false positive* metriku. Ta činjenica, kombinirana s velikim povećanjem preciznosti primjenom jednostavnih augmentacija skupa rotacijama, opravdava izbor neuronskih mreža. Daljnja istraživanja bi se mogla baviti isprobavanjem dodatnih metoda augmentacije te korištenjem kompleksnijih vrsta neuronskih mreža, koje bi imale dobre performanse na većim augmentiranim skupovima.

Summary

Signature Verification is a forensic analysis task, usually solved by trained experts. The main goal is to check whether a new signature belongs to a certain person, if we already have a set of his or her referent signatures. In this paper we will consider possibility of automating this process, since the training of human experts can take a long time.

We are collecting Offline data. These are just plain images of signatures and from pixels we have to determine their validity. First of all we describe the image processing step. RGB Images are converted to the greyscale format, followed by color inversion, removal of noise and converting pixel intensities to binary values. These processed images are now ready for feature extraction, followed by model training.

This paper is divided in two chapters, depending on the feature extraction approach. In the first approach we extract a low number of features which are specific for signatures, and we train low complexity models on top of them. The features are baseline slant angle, aspect ratio, normalized signature area, center of gravity, angle between centers of gravity of each half of the picture, number of edge and cross points. We are using 3 models: Euclidian, Logistic Regression and Decision Trees. First we introduce their mathematical background in order to fully understand their training process. After that we conduct the training and compare their performances. Euclidian model has the best performance. It is also easy to find a balance between false positive and false negative metrics by a simple fine tune of acceptance threshold. Logistic Regression and Decision Trees have lower performances. However, we see improvements by a slight increase of the training set.

This approach has a big restriction, since each person gets its own instance of the model. However, we can conduct training on the whole set of signatures, no matter to whom they belong, by introducing models of higher complexity, like Neural Networks and Support Vector Machine (SVM). We need a bigger training set, but this is not a problem, since we combine all people into a one big set of signatures. Like in the previous chapter, first we introduce the mathematical background of both models. We encounter the overfitting problem and solve it by using regularization and data set augmentation techniques. We conduct the training while optimizing network's topology and hyper-parameters of SVM. As expected, Neural Networks perform better than SVM. However, Euclidian model has better accuracy, but worse false positive metric. We also see network's performance dra-

matically improving, after introduction of data set augmentation by performing simple rotations. There is a big room for further research by using additional data set augmentation techniques and more complex variants of Neural Networks, which would perform better on bigger augmented sets.

Životopis

Ja, Ante Buterin, rođen sam 05. 11. 1995. godine u Zadru. Školovanje sam započeo 2002. godine u Osnovnoj školi Petra Preradovića u Zadru te nastavio 2010. godine upisom prirodoslovno-matematičkog smjera u Gimnaziji Jurja Barakovića. Potom sam upisao preddiplomski studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu, gdje sam 2017. godine stekao titulu sveučilišnog prvostupnika matematike, univ. bacc. math., nakon čega sam nastavio s diplomskim studijem Računarstva i matematike.

Tijekom diplomskog studija radio sam razne poslove u struci. Započeo sam radom u Hrvatskoj Lutriji, kao softverski inženjer pripravnik u rujnu 2017. godine, gdje sam radio na poboljšanju internog servisa za klađenje. Potom sam nastavio karijeru u *Game Development* kompaniji Croteam u veljači 2018. godine u Zagrebu, gdje sam radio kao softverski inženjer na izradi *indie* igara. Tijekom ljeta 2018. godine otišao sam na praksu u Facebook u Londonu, gdje sam radio u *Related Searches* timu kao softverski inženjer pripravnik. Od listopada 2019. godine počet ću raditi kao softverski inženjer u kompaniji R3 u Londonu.

Također sam sudjelovao na par projekata, od kojih bih istaknuo rad na *indie* igri koja je doživjela trenutni uspjeh na platformi *Google Play*, imena *Arrow Science*.