Sažimajuće očitavanje

Marjanović, Mario

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:272465

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Mario Marjanović

SAŽIMAJUĆE OČITAVANJE

Diplomski rad

Voditelj rada:
Ivica Nakić

Zagreb, Rujan 2022.

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. _____

Sadržaj

Uvod

Sažimajuće očitavanje je uzbudljivo, brzo rastuće područje koje je pronašlo veliku primjenu u elektrotehnici, primjenjenoj matematici, statistici, strojnom učenju i računarstvu. Područje je bazirano na rezultatu objavljenom 2004. godine koji su objavili Emmanuel Candes, Justin Romberg, Terence Tao i David Donoho. Nakon objave tog rada [5] došlo je do velikog broja novih rezultata kako praktičnih tako i teorijskih. Ideja ovog rada je objasniti problem sažimajućeg očitavanja, uvesti potrebne pojmove te najvažnije rezultate. Nakon toga ćemo implementirati sustav za prepoznavanje lica koristeći razvijenu teoriju sažimajućeg očitavanja.

U prvom poglavlju obradit ćemo uvodni dio u sažimajuće očitavanje gdje ćemo precizno definirati sve pojmove koji se pojavljuju u teoriji i opisati polazni problem te potrebne uvjete kada ima smisla koristiti teoriju sažimajućeg očitavanja.

U drugom poglavlju opisati ćemo nekoliko algoritama pomoću kojih na efikasan način rješavamo problem, također ćemo za svaki nabrojati neke prednosti i nedostatke te dati primjere situacija u kojim je svaki od njih najbolje koristiti.

U trećem poglavlju dajemo jedan od mnogih primjera problema gdje koristimo teoriju sažimajućeg očitavanja. Uz neke prirodne pretpostavke, problem klasifikacije svodimo na problem sažimajućeg očitavanja, posebno koristimo neke od algoritama iz prethodnog poglavlja pri samoj klasifikaciji.

U četvrtom poglavlju opisujemo implementaciju jednog sustava za klasifikaciju koji je osnovan na teoriji sažimajućeg očitavanja. Specifično, implementirati ćemo sustav za prepoznavanje lica.

Poglavlje 1

Uvod u sažimajuće očitavanje

1.1 Uvod

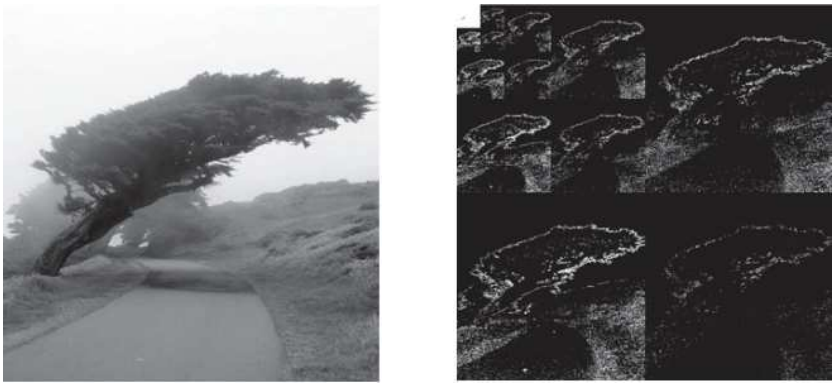
Često u obradi signala ili slika dolazimo do problema gdje želimo rekonstruirati signal iz izmjerenih podataka. Kada je proces prikupljanja informacija linearan, problem se svodi na rješavanje linearnog sustava jednadžbi. Matematički, na taj poblem možemo gledati tako da izmjerene podatke promatramo kao vektor $y \in \mathbb{C}^m$ a originalan signal koji mjerimo kao vektor $x \in \mathbb{C}^N$ pa zapravo imamo sustav jednadžbi:

$$Ax = y. \tag{1.1}$$

Matrica $A \in \mathbb{C}^{m\times N}$ predstavlja model linearnih mjerenja procesa. Kada bi bez dodatnih pretpostavki o vektoru x pokušali riješiti sustav (1.1) za broj mjerenja m morali bi pretpostaviti da je veći ili jednak od N (broju komponenata od x). Ta činjenica zapravo je osnova velikog broja uređaja za mjerenje kao radar, uređaji za medicinsko snimanje i uređaji za mobilnu komunikaciju. Pa tako vrijedi da ako je $m < N$ bez dodatnih pretpostavki o x ne možemo iz jednadžbe (1.1) rekonstruirati x.

Međutim uz dodatnu pretpostavku da je vektor x u (1.1) *rijedak* dobivamo nove, blaže uvjete na m za koje će sustav imati rješenje. Područje istraživanja povezano s ovim fenomenom postalo je poznato kao *sažimajuće očitavanje*.

Za vektor x kažemo da je rijedak ako je većina njegovih komponenti jednaka nuli. Mnogo signala koje promatramo su kompresibilni u smislu da ih možemo dobro aproksimirati rijetkim signalima, često je signal rijedak u nekoj specijalnoj bazi, ali to nam ne predstavlja problem. Veliki broj tehnika kompresiranja oslanjaju se na rijetkost signala, neke od njih su JPEG, MPEG i MP3. Na primjer JPEG se oslanja na rijetkost uslikanih slika u *bazi diskretne valićke transformacije* (BDVT) i ostvaraje sažimanje tako da sprema samo najveće koeficijente u zapisima signala u bazi. Za primjer imamo sliku 1.1



(a) prikaz slike u kanonskoj bazi (b) prikaz slike u BDVT bazi

Slika 1.1: Veliki koeficijenti označeni su sa svijetlim pikselima, dok su mali sa tamnijim pikselima. Iz slika je očito kako je slika prikazana u (b) rijetka.

Glavno pitanje je na koji način možemo iskoristiti činjenicu da je vektor x rijedak, a cilj nam je smanjiti broj mjerenja m što više. To je zapravo osnovni cilj sažimajućeg očitavanja. Potrebno je naglasiti da je najteže odrediti koje su komponente vektora x različite od nule, inače bi samo reducirali matricu A na stupce čiji indeksi odgovaraju indeksima komponenti različitih od nule u vektoru x.

Budući da je broj komponenti različitih od nule signala kojeg možemo sažeti puno manji od duljine signala N, intuitivno će i potreban broj mjerenja signala biti manji. Kada promatramo problem sažimajućeg očitavanja gdje pokušavamo rekonstruirati rijedak vektor $x \in \mathbb{C}^N$ s neodređenim mjerenjem $y = Ax \in \mathbb{C}^m, m \leq N$, postavljamo si dva pitanja:

- Za koje matrice $A \in \mathbb{C}^{m\times N}$ postoji rješenje?
- Ako rješenje postoji, kakve efikasne algoritme imamo za rekonstruiranje vektora x?

Prije nego što odgovorimo na sva pitanja, potrebno je precizno definirati neke od pojmova koje smo do sad spomenuli.

Rijetkost

Želimo precizno definirati pojam *rijetkog* vektora koji smo do sada koristili. Često ćemo koristiti skup od prvih N prirodnih brojeva pa zato uvodimo oznaku $[N]$ koja označava skup $\{1, 2, ..., N\}$.

Definicija 1.1.1. *Oslonac vektora $x \in \mathbb{C}^N$ je skup indeksa u kojima je komponenta od x različita od nule, odnosno:*

$$\text{supp}(x) := \{j \in [N] \mid x_j \neq 0\}$$

Definicija 1.1.2. *Neka je* $x = (x_1, x_2, ..., x_n) \in \mathbb{C}^n$. *Definiramo preslikavanje* $\|\cdot\|_0 : \mathbb{C}^n \to \mathbb{N}$ *dano s:*

$$\|x\|_0 = \#(\text{supp}(x)) \text{ za svaki } x \in \mathbb{C}^N.$$

Drugim riječima, $\|x\|_0$ označava broj komponenti od x različitih od nule.

Definicija 1.1.3. *Za vektor* $x \in C^n$ *kažemo da je s-rijedak ako ima najviše s elemenata različitih od nule, odnosno ako vrijedi:*

$$\|x\|_0 \leq s.$$

1.2 Minimalan broj mjerenja

Sad kada imamo precizno definiran pojam s-rijetkog vektora, možemo preći na problem određivanja minimalnog broja mjerenja koji intuitivno ovisi o rijetkosti vektora $x \in \mathbb{C}^N$ i eventualno o dimenziji N vektorskog prostora u kojem se x nalazi. Problem sažimajućeg očitavanja svodi se na problem rekonstruiranja s-rijetkog vektora $x \in \mathbb{C}^N$ iz jednadžbe:

$$y = Ax,$$

gdje je $A \in \mathbb{C}^{m \times N}$ matrica mjerenja. Ako je $m \leq N$ taj sustavi linearnih jednadžbi je neodređen, odnosno ne mora imati jedinstveno rješenje, ali ako pretpostavimo da je vektor x rijedak to nam može pomoći pri rješavanju sustava. U ovom potpoglavlju odrediti ćemo minimalan broj linearnih mjerenja m za koje možemo egzaktno rekonstruirati s-rijedak vektor x pomoću tih mjerenja. Prije nego dokažemo nužne i dovoljne uvjete za m, uvodimo još nekoliko oznaka:

- A_S koja za matricu $A \in \mathbb{C}^{m \times N}$ i skup $S \subset [N]$ označava podmatricu od A koja se sastoji samo od stupaca čiji indeks se nalazi u S.
- Na sličan način za $x \in \mathbb{C}^N$ uvodimo oznaku x_S što ovisno o kontekstu označava ili podvektor u $\mathbb{C}^{\#(S)}$ čije su komponente redom komponente od vektora x čiji indeksi se nalaze u S, ili označava vektor koji u komponentama čiji indeksi se nalaze u S identičan vektoru x, a u svim ostalima ima vrijednost nula, odnosno za $i \in [N]$ vrijedi:

$$(x_S)_i = \begin{cases} x_i & i \in S, \\ 0 & i \notin S. \end{cases}$$

Sljedeći teorem govori nam o uvjetima koje mora zadovoljavati matrica mjerenja A za koju će sustav uvijek imati jedinstveno rješenje uz dodatnu pretpostavku da je $x \in \mathbb{C}^N$ s-rijedak.

Teorem 1.2.1. *Za zadanu matricu* $A \in C^{m \times N}$, *sljedeće tvrdnje su ekvivalentne:*

(i) *Svaki s-rijedak vektor* $x \in \mathbb{C}^N$ *ima jedinstveno s-rijetko riješenje za* $Az = Ax$ *odnosno, ako je* $Az = Ax$ *i vrijedi da su* x *i* z *s-rijetki, tada vrijedi i* $x = z$.

(ii) $\operatorname{Ker} A$ *ne sadrži nijedan 2s-rijedak vektor osim nul vektora, odnosno* $\operatorname{Ker} A \cap \{z \in \mathbb{C}^N \mid \|z\|_0 \leq 2s\} = \{0\}$.

(iii) *Za svaki* $S \subset [N]$ *takav da vrijedi* $\#(S) \leq 2s$, *podmatrica* A_S *je injekcija sa* $\mathbb{C}^{\#(S)}$ *na* $\mathbb{C}^m$.

(iv) *Svaki podskup od 2s stupaca iz A je linearno nezavisan.*

Dokaz. $(i) \implies (ii)$ Pretpostavimo da za svaki s-rijedak vektor $x \in \mathbb{C}^N$ imamo $\{z \in \mathbb{C}^N \mid Az = Ax, \|z\|_0 \leq s\} = \{x\}$. Neka je $v \in \operatorname{Ker} A$ 2s-rijedak. Možemo zapisati $v = x - z$ za s-rijetke vektore x i z takve da je $\operatorname{supp}(x) \cap \operatorname{supp}(z) = \varnothing$. Tada je $Av = A(x - z) = 0$ pa je i $Ax = Az$, po pretpostavci vrijedi $x = z$. Iz $\operatorname{supp}(x) \cap \operatorname{supp}(z) = \varnothing$ slijedi da je $x = z = 0$ dakle slijedi $v = 0$

$(ii) \implies (i)$ Neka su x i z s-rijetki i vrijedi $Ax = Az$. Tada je $x - z$ $2s$-rijedak vektor i $A(x - z) = 0$. Ako jezgra ne sadrži nijedan $2s$-rijedak vektor različit od nule, tada slijedi da je $x - z = 0$ iz čega slijedi $x = z$.

$(ii) \implies (iii)$ Pretpostavimo suprotno, neka $\operatorname{Ker} A$ ne sadrži nijedan $2s$-rijedak vektor osim nul vektora i da za neki $S \subset [N]$ podmatrica A_S nije injekcija. Tada postoje $x, y \in \mathbb{C}^{\#(S)}$ takvi da vrijedi $x \neq y$ i $Ax = Ay$, iz čega slijedi da je $A_S x - A_S y = A_S(x - y) = 0$. Označimo vektor $z = x - y \neq 0$, za prošireni vektor z_S očito vrijedi: $z_S \neq 0, \|z\|_0 \leq 2s, Az_S = 0$, što je u kontradikciji s početnom pretpostavkom

$(iii) \implies (iv)$ Pretpostavimo suprotno, neka za svaki $S \subset [N]$ takav da $\#(S) \leq 2S$ vrijedi da je A_S injekcija i da postoji neki podskup indeksa S od 2s članova takav da je skup stupaca $V = \{v_i \mid v_i$ je i-ti stupac od A, $i \in S\}$ linearno zavisan. Kada bi taj skup bio linearno zavisan iz definicije linearne zavisnosti dobili bi koeficijente pomoću kojih tvorimo vektor $z \neq 0$ za kojeg će vrijediti da je $A_S z = 0$ što je u kontradikciji s pretpostavkom da je A_S injekcija.

$(iv) \implies (ii)$ Kada bi svaka $2s$ stupca u A bili linearno nezavisni, jezgra od A ne bi sadržavala niti jedan $2s$-rijedak vektor osim nul vektora, u suprotnom kad bi postojao neki $2s$-rijedak vektor x u jezgri različit od nul vektora bi skup stupaca od A kojima indeksi odgovaraju pripadajućim indeksima unutar $S = \operatorname{supp}(x)$ imali svojstvo da je $A_S x_S = 0$. Kako je x_S različit od 0 očito bi i stupci bili linearno zavisni pa dolazimo do kontradikcije. □

Iz prethodnog teorema vidimo da ako je moguće rekonstruirati svaki s-rijedak vektor $x \in \mathbb{C}^N$ iz izmjerenog vektora $y = Ax \in \mathbb{C}^m$, tada očito vrijedi (i), ali i po prethodnom teoremu, vrijedi i (iv). Očito slijedi i da je $\operatorname{rang}(A) \geq 2s$, uz već poznatu ogradu $\operatorname{rang}(A) \leq m$ dobijamo donju ogradu za broj mjerenja m:

$$m \geq 2s.$$

Zapravo možemo dokazati i jaču tvrdnju. Dovoljno nam je točno $m = 2s$ mjerenja kako bi mogli rekonstruirati s-rijedak vektor. Dokaz kako uvijek možemo pronaći matricu mjerenja $A \in \mathbb{C}^{2s \times N}$ koja zadovoljava uvjete teorema 1.2.1 može se pronaći u [7].

Iako imamo dovoljan uvjet koji nam govori kada polazni sustav ima jedinstveno rješenje, nemamo efektivan algoritam koji će nam egzaktno rekonstruirati x. Štoviše možemo dokazati da je taj problem NP-težak. Dokaz za tu tvrdnju može se pronaći u [7].

Poglavlje 2

Algoritmi rekonstrukcije rijetkog signala

U ovom poglavlju pokriti ćemo dvije najraširenije familije algoritama za rješavanje problema rekonstrukcije rijetkog signala x. Formalno, za $A \in \mathbb{C}^{m \times N}$ i $y \in \mathbb{C}^m$ želimo rekonstruirati rijedak signal $x \in \mathbb{C}^N$ iz sustava jednadžbi:

$$Ax = y. \tag{2.1}$$

Prvo ćemo opisati dvije najraširenije familije algoritama: algoritmi l_1 minimizacije i pohlepni algoritmi, za svaki ćemo dati barem jedan konkretan primjer algoritma i neke od njih implementirati kao glavnu komponentu u sustavu prepoznavanja lica.

2.1 l_1 minimizacija

Ako imamo izmjereni vektor y i poznatu činjenicu da je originalan vektor x rijedak, idealno bi bilo kad bi postojao efikasan algoritam koji rješava sljedeći problem optimizacije

$$\hat{x} = \arg\min_{z \in B(y)} \|z\|_0, \tag{2.2}$$

gdje $B(y)$ označava skup rješenja koji su nam prihvatljivi. Ako na primjer tražimo egzaktna rješenja, tada je $B(y) = \{z \mid Az = y\}$. Ono što se u praksi češće koristi je $B(y) = \{z \mid \|Az - y\|_2 \leq \epsilon\}$ za neki $\epsilon \in \mathbb{R}$.

Napomena 2.1.1. *U (2.2) imamo inicijalnu pretpostavku da je vektor x rijedak u kanonskoj bazi, ali često promatramo slučaje kada je x rijedak u nekoj drugoj bazi, odnosno za matricu prijelaza* Φ *vrijedi* $x = \Phi c$*, gdje je c rijedak vektor, uz malu modifikaciju problema*

(2.2) imamo

$$\hat{c} = \arg\min_{z \in B(y)} \|z\|_0 \tag{2.3}$$

gdje je $B(y) = \{z \mid A\Phi z = y\}$ *ili* $B(y) = \{z \mid \|A\Phi z - y\|_2 \leq \epsilon\}$. *Ako uvedemo oznaku* $\hat{A} = A\Phi$ *vidi se da su problemi (2.2) i (2.3) zapravo identični i u ovom radu ćemo uvijek pretpostavljati da je vektor x rijedak u kanonskoj bazi, odnosno* $\Phi = I$.

No kako smo zaključili na kraju prethodnog poglavlja, mi zapravo nemamo efikasan algoritam za rješavanje (2.2). On je općenito za matrice A NP-težak [7].

Jedan način da si olakšamo inicijalan problem (2.2) je umjesto da minimiziramo po nekonveksnoj funkciji $\|\cdot\|_0$, zamijenimo ju sa njenom konveksnom aproksimacijom $\|\cdot\|_1$. Drugim riječima promatramo novi problem:

$$\hat{x} = \arg\min_{z \in B(y)} \|z\|_1. \tag{2.4}$$

Ako je $B(y)$ konveksan skup, postoji dobro razvijena teorija van okvira sažimajućeg očitavanja koja se bavi rješavanjem tog problema. Posebno ako je $B(y) = \{z \mid Az = y\}$, na (2.4) možemo gledati kao problem linearnog programiranja.

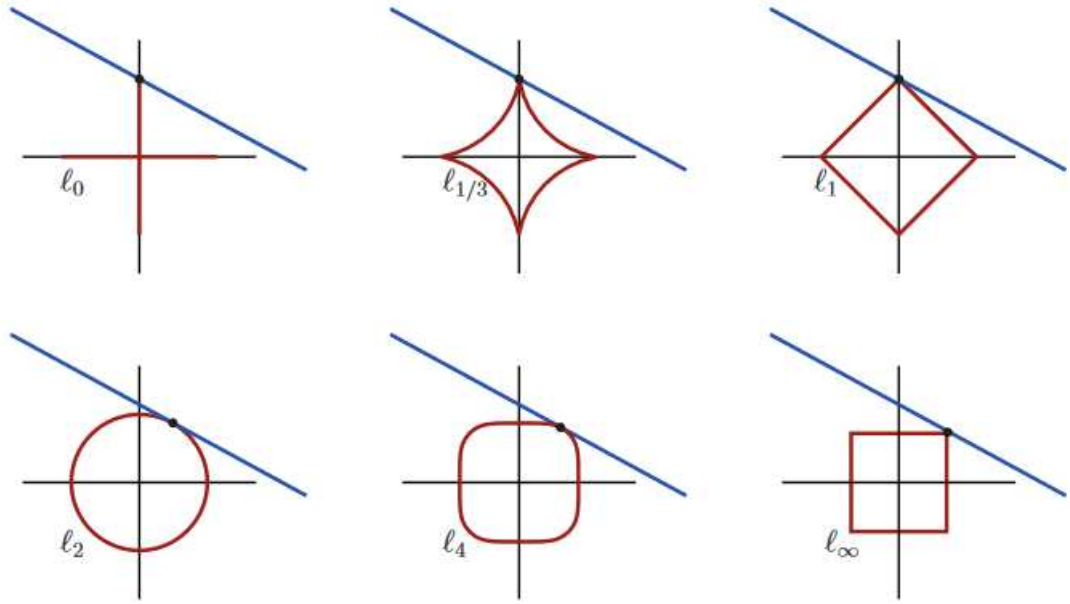
Jasno je da promatranjem (2.4) umjesto (2.2) imamo problem za koji postoje efektivni algoritmi koji ga riješavaju, ali možda nije očito zašto bi l_1 minimizacijom dobili vektor x koji je rijedak. Slika 2.1 nam daje intuiciju zašto za $p \leq 1$, l_p minimizacijom dobivamo rijetka rješenja. Vidimo da za sve manji p vrhovi kugla su sve oštriji prema koordinatnim osima i povećavanjem radijusa kugle velika je vjerojatnos da ćemo s kuglom dodirnuti skup rješenja nekom točkom kugle koja se nalazi na koordinatnoj osi. Više o tome zašto l_1 minimizacija potiče rijetkost može se pronaći u [5].

Problem (2.4) možemo riješiti kao problem linearnog programiranja s visokom preciznošću koristeći metodu unutrašnje točke [17], ali on nam nije dovoljno dobar za velike dimenzije m vektora y jer se svaka iteracija odvija u kubnom vremenu ovisno o m. Kako se razvija potreba za korištenje tehnika sažimajućeg očitavanja tako se razvijaju i brzi algoritmi za problem l_1 minimizacije s nešto manjom preciznošću.

Opisati ćemo jedan algoritam l_1 minimizacije koji se često koristi u sažimajućem očitavanju i implementirati ga u sustav za prepoznavanje lica.

Prošireni Lagrangeov Multiplikator

Prošireni Lagrangeov Multiplikator (PLM) algoritam se koristi u sustavima za prepoznavanje lica jer on osim što minimizira vektor x, ujedno minimizira i grešku e što je korisno jer u većini slučajeva ne možemo dobiti egzaktno rješenje u takvim sustavima pa imamo problem koji se malo razlikuje od standardnog. Želimo pronaći rješenje jednadžbe:



Slika 2.1: Prikazane su točke s najmanjom normom koje pripadaju nekom skupu za različite l_p norme. Plava linija prikazuje rješenja nekog neodređenog sustava jednadžbi, a crvene krivulje najmanje kugle p norme koji dodiruje skup rješenja

$$\min\left(\|x\|_1 + \|e\|_1\right) \text{ tako da vrijedi } y = Ax + e. \tag{2.5}$$

Algoritam se oslanja na metodu Lagrangeovih multiplikatora [3], ali i koristi ubrzani gradijentni algoritam [2] za rješavanje potproblema. Uz to koristi i neke osnovne rezultate teorije l_1 minimizacije.

Jedno od ključnih svojstava l_1 norme koje nam daje mnoge efikasne algoritme je što imamo efikasno rješenje za problem proksimalne minimizacije:

$$S_\lambda[z] = \arg\min_c \lambda\|c\|_1 + \frac{1}{2}\|c - z\|_2^2,$$

gdje su $c, z \in \mathbb{R}^n$ i $\lambda > 0$. Lako se pokaže da rješenje problema proksimalne minimizacije nad $\mathbb{R}$ dobijemo pozivom funkcije *mekog praga*, koja je za skalar $z \in \mathbb{R}$ definirana kao:

$$S_\lambda[z] = \begin{cases} z - \lambda, & z > \lambda, \\ z + \lambda, & z \leq -\lambda, \\ 0, & |z| \leq \lambda. \end{cases} \tag{2.6}$$

Funkcija se proširuje na vektore i matrice tako što se primjenjuje po elementima (u algoritmu 1 funkciju mekog praga označavamo sa *shrink*). Jednostavna je za implementirat, izvodi se u linearnom vremenu i korak je u mnogim algoritmima za l_1 minimizaciju. PLM se u više koraka iteracije oslanja na funkciju mekog praga.

Metoda Lagrangeovih multiplikatora je popularan alat u konveksnoj optimizaciji, osnovna ideja je eliminirati uvjete jednakosti dodavanjem pripadne kazne funkciji cilja za nedostižne točke. Ideja je da riješimo novi problem bez uvjeta jednakosti za koji imamo algoritme koji ga efikasno rješavaju. Za naš problem formiramo Lagrangeovu funkciju na sljedeći način:

$$L_\mu(x, e, \nu) := \|x\|_1 + \|e\|_1 + \langle \nu, y - Ax - e\rangle + \frac{\mu}{2}\|y - Ax - e\|_2^2$$

gdje je $\mu > 0$ i ν je vektor Lagrangeovih multiplikatora. Lagrangeova funkcija je konveksna u c i e i ako imamo optimalno rješenje (c^*, e^*) za originalan problem (2.5) tada za dovoljno velik parametar μ postoji ν^* takav da vrijedi:

$$(x^*, e^*) = \arg\min_{c,e} L_\mu(x, e, \nu^*)$$

Problem je naravno što nam a priori nisu poznati ν^* i μ. To riješavamo tako što ν^* iterativno računamo zajedno sa x i e, a μ monotono povećavamo na kraju svake iteracije. Tako u $k+1$ iteraciji PLM algoritma računamo:

$$\begin{aligned}(x_{k+1}, e_{k+1}) &= \arg\min_{x,e} L_{\mu_k}(x, e, \nu_k),\\ \nu_{k+1} &= \nu_k + \mu_k(y - Ax_{k+1} - e_{k+1}).\end{aligned}$$

Koristeći *metodu alternirajućih smjerova multiplikatora* [9], prethodnu iteraciju možemo zapisati kao:

$$\begin{aligned}e_{k+1} &= \arg\min_{e} L_{\mu_k}(x_k, e, \nu_k),\\ x_{k+1} &= \arg\min_{x} L_{\mu_k}(x, e_{k+1}, \nu_k),\\ \nu_{k+1} &= \nu_k + \mu_k(y - Ax_{k+1} - e_{k+1}).\end{aligned}$$

Koristeći definiciju funkcije L_μ, lako se pokaže da vrijedi:

$$e_{k+1} = S_{\frac{1}{\mu_k}}\left[\frac{1}{\mu_k}\nu_k + y - Ax_k\right].$$

Nažalost za računanje izraza x_{k+1} nemamo zatvorenu formulu. Ali koristeći činjenicu da se $L_\mu(x, e_{k+1}, \nu_k)$ može rastaviti kao suma dvije funkcije: $\|x\|_1 + \|e_{k+1}\|_1 + \langle \nu_k, y - Ac - e\rangle$ koja je konveksna i neprekidna u x i funkciju $\frac{\mu}{2}\|y - Ax - e\|_2^2$ koja je konveksna, pripada klasi C^∞ i ima Lipschitz neprekidan gradijent, možemo koristiti FISTA [2] algoritam kako

bi efikasno izračunali x_{k+1}. U pseudokodu PLM algoritma 1 u unutrašnjoj petlji vidimo kako koristimo FISTA algoritam pri računanju izraza x_{k+1}, gdje nam γ označava najveću svojstvenu vrijednost matrice A^TA. Iako se algoritam sastoji od dvije petlje, prosječno vrijeme izvršavanja algoritma nije veliko jer u praksi unutarnja petlja konvergira nakon nekoliko iteracija.

Algorithm 1 Prošireni Lagrangeov Multiplikator (PLM)

Input: $y \in \mathbb{R}^m, A \in \mathbb{R}^{m\times n}$.
Initilize: $x_1 = 0, e_1 = y, \nu_1 = 0$
while not converged (k = 1, 2, ...) **do**
  $e_{k+1} \leftarrow \text{shrink}(y - Ax_k + \frac{1}{\mu_k}\nu_k, \frac{1}{\mu_k})$
  $t_1 \leftarrow, z_1 \leftarrow x_k, w_1 \leftarrow x_k$
  while not converged ($l = 1, 2, ...$) **do**
    $w_{l+1} \leftarrow \text{shrink}\left(z_l + \frac{1}{\gamma}A^T\left(y - Az_l - e_{k+1} + \frac{1}{\mu_k}\nu_k\right), \frac{1}{\mu_k\gamma}\right)$
    $t_{l+1} \leftarrow \frac{1}{2}\left(1 + \sqrt{1 + 4t_l^2}\right)$
    $z_{l+1} \leftarrow w_{l+1} + \frac{t_{l-1}}{t_{l+1}}(w_{l+1} - w_l)$
  end while
  $x_{k+1} \leftarrow w_l$
  $\nu_{k+1} \leftarrow \nu_k + \mu_k(y - Ax_{k+1} - e_{k+1})$
end while
Output $x^* \leftarrow x_k, e^* \leftarrow e_k,$

2.2 Pohlepni algoritmi

Sažimajuće očitavanje često je sinonim za l_1 optimizaciju, ali kada odabiremo algoritam za rješavanje nekog specifičnog problema moramo procijeniti koja svojstva želimo da algoritam ima te odrediti koji je optimalan. Neka od svojstava koja nas često zanimaju su brzina, prostorna složenost, težina implementacije i fleksibilnost. Za primjer dajemo neke od alternativnih algoritama koji su brži od algoritama l_1 minimizacije, ali imaju slabiju teorijsku preciznost.

U ovom potpoglavlju promatramo pohlepne algoritme za rekonstrukciju signala x iz jednadžbe (2.2). Takve metode možemo definirati kao skup metoda gdje iterativno konstruiramo aproksimaciju od x počevši od nul-vektora. Ovim metodama pokušavamo odrediti skup indeksa komponenata u kojima je x različit od nule tako što iterativno dodajemo novu

komponentu za koju smatramo da je različita od nule. Uz pohlepnu selekciju još radimo i korak procjene gdje optimiziramo same vrijednosti komponenti za koje smatramo da su različite od nule. Ove metode nam često daju vrlo brze algoritme koje možemo koristiti za probleme s velikim dimenzijama m i N, ali teorijske performanse su u većini slučajeva slabije nego za neke druge metode.

Prije nego opišemo neke konkretne pohlepne algoritme, prvo ćemo opisati fundementalne korake koje imaju svi pohlepni algoritmi u okviru sažimajućeg očitavanja. Općenito u ovim metodama na početku inicijaliziramo aproksimaciju vektora x sa $\hat{x}_0 = 0$, početni ostatak sa $\hat{r}_0 = y - A\hat{x}_0 = y$ i skup oslonaca (skup indeksa elemenata od $\hat{x}$ koji su različiti od nule) sa $T_0 = \varnothing$. Glavna ideja ovih algoritama je da nakon svake iteracije na neki način dodajemo novi element u skup oslonaca T i dajemo novu, bolju aproksimaciju $\hat{x}$ takav da je $\text{supp}(\hat{x}) = T$ dok same vrijednosti elemenata ovise o konkretnom algoritmu. Na kraju svake iteracije ažuriramo i ostatak r.

Za prvi primjer dajemo jedan od najjednostavnijih algoritama iz familije pohlepnih algoritama - Matching Pursuit (MP).

Algorithm 2 Matching Pursuit (MP)

Input: $y \in \mathbb{R}^m, A \in \mathbb{R}^{m\times n}, k \in \mathbb{N}$
Initilize: $r_0 = y, \hat{x}_0 = 0, T_0 = \varnothing$
for $i = 1, i := i + 1, i \leq k$ **do**
  $g_i \leftarrow A^T r_{i-1}$
  $j_i \leftarrow \arg\max_j \left|g_j\right| / \|A_j\|_2$
  $T_i \leftarrow T_{i-1} \cup j_i$
  $(\hat{x}_i)_{j_i} \leftarrow (\hat{x}_{i-1})_{j_i} + \left|(g_i)_j\right| / \|A_j\|_2^2$
  $r_i = r_{i-1} - (g_i)_{j_i} * A_{j_i} / \|A_{j_i}\|_2^2$
end for
Output $\hat{x}_k$

U MP algoritmu aproksimaciju $\hat{x}$ računamo postepeno tako što prvo odabiremo jedan stupac matrice A i u svakoj iteraciji mijenjamo samo koeficijent unutar vektora x koji odgovara odabranom stupcu matrice. Koeficijent komponente ažuriramo formulom:

$$(\hat{x}_i)_{j_i} = s(\hat{x}_{i-1})_{j_i} + \left|(g_i)_j\right| / \|A_j\|_2^2 \tag{2.7}$$

koja minimizira funkciju cilja $\|y - A\hat{x}_i\|_2^2$ gdje je jedino dozvoljeno mijenjati komponentu s indeksom j koju smo odredili na početku iteracije. Oznaka A_j u formuli (2.7) i u ostatku ovog poglavlja označava j-ti stupac matrice A. Važno je napomenuti da je u MP algoritmu dozvoljeno više puta odabrati isti indeks stupca j matrice A ako ćemo ažuriranjem pripadajuće komponente pohlepno dobiti najbolju moguću aproksimaciju vektora x. Jedno snažno svojstvo ovog algoritma je da $\|r_i\|_2$ linearno konvergira u nulu kad stupci od A razapinju

$\mathbb{R}^m$, dokaz za tu tvrdnju može se pronaći u [14]. Ali u praksi češće algoritam završavamo nakon k broja iteracija ili ako nakon neke iteracije norma ostatka dovoljno mala, odnosno ako vrijedi $\|r_i\|_2 < \epsilon$ za neki $\epsilon \in \mathbb{R}$.

Za svaku iteraciju algoritma MP potrebno je izvršiti matrično množenje s matricom A^T, to je korak koji najviše utječe na kompleksnost algoritma. Iz tog razloga MP se najčešće koristi za matrice gdje su stupci od matrice A rijetki. Za takve matrice imamo iznimno brze implementacije algoritma MP [11].

Algorithm 3 Orthogonal Matching Pursuit (OMP)

In $y \in \mathbb{R}^m, A \in \mathbb{R}^{m\times n}, k \in \mathbb{N}$
Initilize: $r_0 = y, \hat{x}_0 = 0, T_0 = \varnothing$
for $i = 1, i := i + 1, i \leq k$ **do**
  $g_i \leftarrow A^T r_{i-1}$
  $j_i \leftarrow \arg\max_j \left|(g_i)_j\right| / \|A_j\|_2$
  $T_i \leftarrow T_{i-1} \cup j_i$
  $\hat{x}_i \leftarrow \arg\min_z \|y - A_{T_i} z\|_2$
  $r_i = y - A\hat{x}_i$
end for
Output $\hat{x}_k$

Nešto složeniji algoritam od prethodnog je Orthogonal Matching Pursuit (OMP). U OMP-u nakon svake iteracije algoritma dobivamo aproksimaciju $\hat{x}$ koja je ortogonalna na sve stupce iz A čiji indeksi se nalaze u skupu oslonaca T za tu iteraciju algoritma, upravo po tom svojstvu algoritam dobiva i ime. Tako osiguravamo da na kraju svake iteracije OMP algoritam minimzira $\|y - A\hat{x}\|_2$ za sve $\hat{x}$ tako da vrijedi $\text{supp}(x) \subseteq T$ (2.8). Najveća razlika u odnosu na MP algoritam je što u koraku minimizacije ažuriramo sve komponente od $\hat{x}$ čiji indeksi se nalaze unutar T, a ne samo komponentu čiji indeks smo odabrali na početku iteracije. Baš zbog toga, u OMP algoritmu nikada u početku iteracije ne odabiremo indeks koji se već nalazi u skupu oslonaca T, dakle

$$x_{T_i} = \arg\min_{z_{T_i}} \|y - A_{T_i} z_{T_i}\|_2^2. \tag{2.8}$$

Za općenite matrice A, kao i u MP algoritmu vremenskoj složenosti najviše doprinosi množenje vektora i matrice, ali u praksi se algoritam primjenjuje za matrice gdje se taj korak može implementirati na efikasan način. Zapravo u praksi kritična točka algoritma je ortogonalizacija (2.8) odnosno problem najmanjih kvadrata koji rješavamo QR faktorizacijom [19] ili Cholesky faktorizacijom [18]. Iako je OMP algoritam vremenski složeniji od MP algoritma, u praksi ima znatno bolje performanse, posebno u kontekstu sažimajućeg očitavanja.

Dva su glavna problema sa primjenom OMP algoritma za velike matrice A. Prvi je što je prostorna i vremenska složenost jedne iteracije algoritma velika i za rekreiranje jednog vektora x algoritam će imati k iteracija pa je algoritam za veliku rijetkost k nepraktičan. OMP algoritam ima razne varijacije koje pokušavaju riješiti ova dva problema, više o njima nalazi se u [12] i [20].

Poglavlje 3

Sažimajuće očitavanje u sustavima za prepoznavanje lica

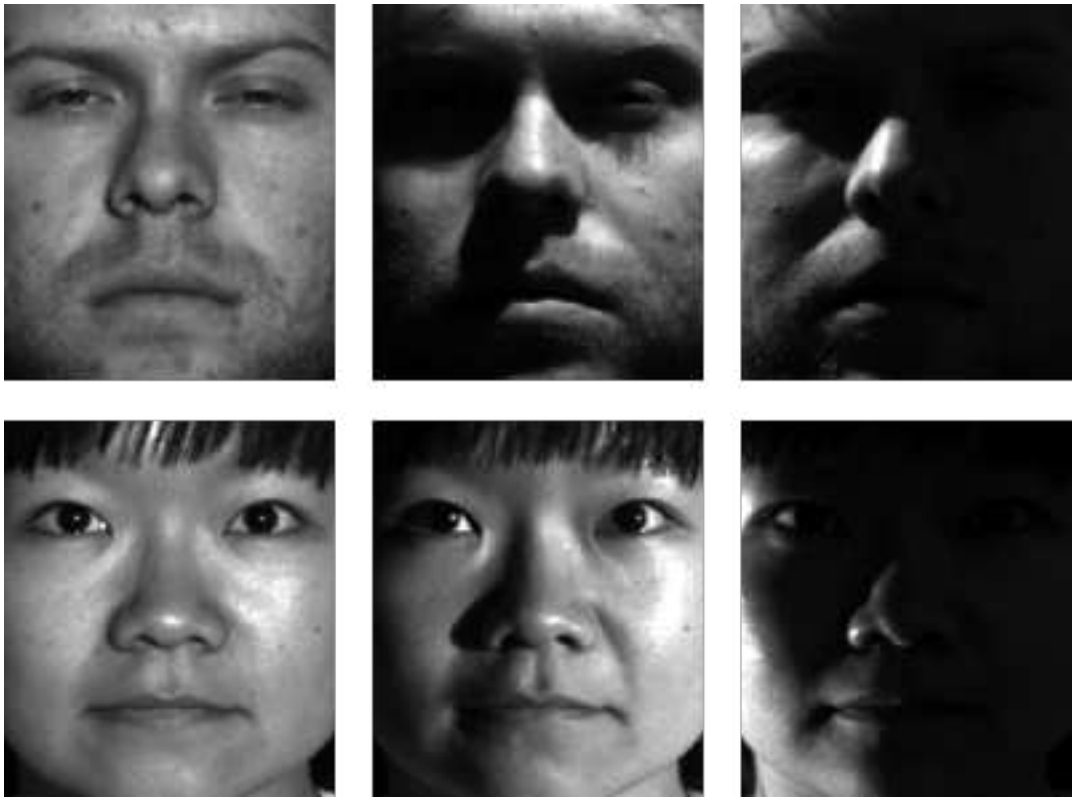
U ovom poglavlju pokazujemo kako riješiti dobro poznati problem prepoznavanja lica koristeći teoriju i algoritme sažimajućeg očitavanja. Ovaj pristup zove se *klasifikacija lica pomoću rijetke reprezentacije* (KLPRR) i bazira se na tome da sliku lica po kojoj radimo upit izrazimo kao linearnu kombinaciju malog broja slika s kojima smo trenirali sustav. U posljednjem koraku ćemo pomoću dobivene linearne kombinacije odrediti koja osoba se nalazi na slici.

3.1 Klasifikacija lica pomoću rijetke reprezentacije

Ilustriramo osnovnu ideju iza KLPRR-a u idealnom scenariju gdje su slike korištene za treniranje i testiranje uvijek slikane u crno-bijeloj tehnici iz istog kuta za svaku osobu i jedina značajna razlika između slika osoba je osvjetljenje. Vidjeti ćemo kako na vrlo jednostavan način problem prepoznavanja osobe možemo postaviti kao problem rekonstruiranja signala rijetkog vektora i riješiti ga pomoću prethodno opisanih algoritama.

Općenito sustavi za prepoznavanje lica rade tako da ih prvo treniramo označenim slikama za treniranje $\{\phi_i, l_i\}$ koje pripadaju nekima od C osoba koje ćemo pri testiranju identficirati. Tu je $\phi_i \in \mathbb{R}^m$ vektorska reprezentacija digitalne slike, gdje je $m = W \times H$ broj piksela slike koji ovisi o njeno širini W i visini H, a $l_i \in [C]$ označava koja se osoba od njih C se nalazi na slici i. Zadaća sustava je da za potpuno novu sliku reprezentiranu vektorom $y \in \mathbb{R}^m$ odredi kojoj ona osobi pripada, ili da odbije sliku ako ne pripada ni jednoj od C osoba koje smo koristili za treniranje sustava.

Napomena 3.1.1. *Kako su slike crno-bijele, vektorska reprezentacija digitalne slike je vektor kojem je svaka komponenta prirodni broj u intervalu* $[0, 255]$ *i predstavlja nijansu*



Slika 3.1: Primjeri slika koje su korištene pri treniranju sustava za prepoznavanje lica.

sive pojedinog piksela gdje 0 predstavlja potpuno crni piksel, a 255 potpuno bijeli piksel.

Sustavi za prepoznavanje lica koji koriste teoriju sažimajućeg očitavanja baziraju se na pretpostavci da uz dovoljnu količinu kvalitetnih slika s raznim osvjetljenima novu digitalnu sliku osobe k reprezentiranu kao vektor y možemo aproksimirati kao linearnu kombinaciju vektora slika koje smo koristili za treniranje sustava:

$$y \approx \sum_{\{j \in [N] | l_j = k\}} \phi_j x_j := \Phi_k x_k,$$

gdje je $\Phi_k \in \mathbb{R}^{m \times n_i}$ matrica čiji su stupci svi vektori koji reprezentiraju slike osobe k koje smo koristili pri treniranju sustava, a $x_i \in \mathbb{R}^{n_i}$ odgovarajući vektor koeficijenta linearne kombinacije.

Naravno, pri testiranju sustava nije prethodno poznato kojoj osobi k slika pripada. Najbolje što možemo je riješiti sljedeći sustav jednadžbi:

$$y = [\Phi_1, \Phi_2, ...\Phi_C]\, x = \Phi x \in \mathbb{R}^m, \tag{3.1}$$

gdje je $x \in \mathbb{R}^n$. Sustav ćemo riješiti nekim od poznatih algoritama za rekonstrukciju rijetkog vektora gdje očekujemo da će većina komponenti u rješenju x koje su različite od nule biti one koje se množe s pripadajućim stupcima iz Φ_k.

Upravo to je glavna ideja SRC-a, postavljanje problema prepoznavanja lica kao poznati sustav jednadžbi

$$y = \Phi x,$$

gdje pretpostavljamo da je x rijedak vektor. Na osnovu rekonstruiranog vektora x sustav donosi odluku koja osoba je na novoj slici (ako se na slici uopće nalazi neka od C osoba). Ovdje zapravo možemo i pretpostaviti koliko je x zapravo rijedak. U prosjeku očekujemo da će $\frac{n}{C}$ njegovih komponenti biti različite od nule jer je toliko slika svake osobe korišteno pri treniranju sustava.

U praksi se rijetko događa da su rekonstruiranom vektoru x sve komponente koje su različite od nule vezane uz jednu osobu k pa je potreban neki algoritam koji nam govori kada za sliku upita možemo reći da je ona slika osobe k, a kada se radi o slici koja ne predstavlja ni jednu od C osoba. Opisujemo jedan jednostavan algoritam koji ne ovisi o algoritmu koji smo koristili za rekonstrukciju vektora x. Ako vektor x zapišemo kao

$$x = [x_1^T, x_2^T, ..., x_C^T]^T$$

te za osobu k definiramo

$$\alpha_k := \frac{||x_k||_1}{||x||_1} \tag{3.2}$$

i pronalaskom $z \in [C]$ takav da vrijedi

$$z = \arg\max_{k \in [C]} \alpha_k,$$

dobili smo osobu za koju imamo najveću vjerojatnost da se nalazi na slici vektora y. Konačna odluka još ovisi o tome je li zadovoljen uvjet $\alpha_z \geq \xi$ za neki $\xi \in \langle 0, 1 \rangle$. Ako je uvjet zadovoljen, sustav vraća odgovor da se na slici vektora y nalazi osoba z, u suprotnom sustav odgovara kako ne prepoznaje osobu sa slike. Neke kompleksnije metode klasifikacije ovisne o rijetkim koeficijentima u x mogu se pronaći u [21] i [22].

3.2 Smanjivanje uzorkovanja (Downsampling)

Jedan problem koji nismo spomenuli je ogromna veličina podataka većih sustava. Dimenzija jedne slike može biti u milijinuma, a potreban broj slika za treniranja sustava proporcionalan je sa dimenzijom slike i brojem osoba koje želimo dodati u sustav. Naravno, uz veće dimenzije povećava se i vrijeme izvršavanja algoritama za rekonstrukciju vektora. To je problem koji se često pojavljuje kada prepoznajemo uzorke nekih slika i najčešće se

rješava tako što se originalne reprezentacije slika iz $\mathbb{R}^m$ projiciraju u $\mathbb{R}^d$ gdje je $d \ll m$. Projekcijom pokušavamo sačuvati sva važna svojstva originalne slike.

Ne postoji jedinstvena projekcija koja je najbolja u svim slučajevima, za svaki problem tražimo onu koja nam najviše odgovara. Ako za projekciju uzmemo neki linearni operator $A \in \mathbb{R}^{d \times m}$, originalan problem možemo postaviti na sljedeći način:

$$\hat{y} := Ay \approx A\Phi x = \Psi x \in \mathbb{R}^d.$$

I tako ponovno imamo problem rekonstrukcije rijetkog signala. Pri odabiru projekcije potrebno je paziti da ne uzmemo premalu dimenziju d jer je moguće da rješenje jednadžbe $A\Phi x = \hat{y}$ ne bude jedinstveno. Iako će algoritmi za rekonstrukciju rijetkog vektora uvijek vraćati jedinstveno, *najrjeđe* rješenje, niska dimenzionalnost vektorskog prostora sa sobom donosi i određenu nestabilnost jer uz manju dimenziju slike se ujedno i manje razlikuju jedna od druge i tako češće dolazi do pogrešne klasifikacije.

Neizbježno je da projekcijom na manji vektorski prostor gubimo preciznost sustava, ali uz pametan odabir matrije proekcije A, dimenzija d može i dalje biti puno manja od originalne dimenzije m uz minimalan gubitak preciznosti. Potrebno je još naglasiti da nam je ponekad vrlo važna preciznost sustava, a ne brzina kojom sustav prepoznaje osobu, tada ako imamo dovoljno memorije možemo tražiti rješenje za originalan problem (3.1).

Poglavlje 4

Implementacija sustava za prepoznavanje lica

U ovom poglavlju opisujemo postupak izrade aplikacije koja implementira sustav za prepoznavanje lica baziranom na teoriji sažimajućeg očitavanja. Prvo ćemo navesti korištene tehnologije pri izradi aplikacije, a zatim detaljno proći kroz korake implementacije koju dijelimo na dva dijela:

- treniranje sustava,
- testiranje sustava.

Pri treniranju sustava cilj nam je pomoću određenog skupa slika dobiti *matricu treniranja* koja u klasičnom problemu rekonstrukcije rijetkog vektora (2.1) predstavlja matricu A, ali u ovom poglavlju označavamo ju sa Φ. Vektorsku reprezentaciju slike upita označavamo s y i iz jednadžbe (2.1) računamo rijetko rješenje x te pomoću njega određujemo koja osoba se nalazi na slici upita (ako nam je poznata).

Programski kôd aplikacije napisan je u programskom jeziku Java i javno je dostupan na GitHub repositoriju SparseRepresentationClassification [15]. Za razvoj aplikacije korišteni su:

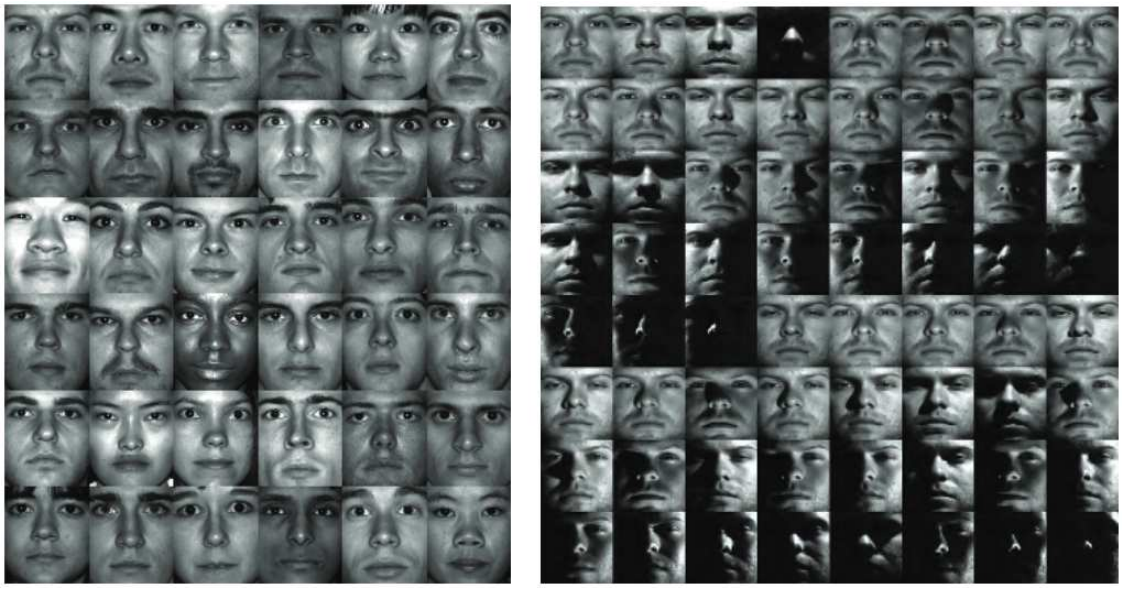
- Java SE Development Kit (JDK) 11 [16],
- sustav za izgradnju aplikacija Gradle 7 [1],
- ND4J biblioteka za matrične operacije [4],
- razvojno okruženje Intellij IDEA Community [10].

4.1 Treniranje sustava

Baza podataka slika

Za treniranje i testiranje sustava korištena je javno dostupna Yale B Extended baza podataka [8]. U njoj se nalazi po pedeset slika lica izbliza za dvadeset različitih osoba (vidi sliku 4.1). Dimenzije svih slika su 168×192.

Slike su originalno preuzete kao .mat (MATLAB) datoteka u kojoj se nalazi matrica čiji su elementi slike, zatim su pomoću python skripte 4.1 slike izvučene iz matrice i spremljene kao crno-bijela slika u .png formatu kako bi ih mogli koristiti u Java aplikaciji. Od ukupno pedeset slika, trideset su korištene za treniranje sustava, a dvadeset za testiranje.



Slika 4.1: Na lijevoj strani nalaze se pojedine slike svih osoba u Yale B bazi podataka, a na desnoj strani imamo sve slike jedne osobe.

Listing 4.1: extract_images.ipynb

```
import matplotlib.pyplot as plt
import os
import scipy.io
from skimage.transform import resize
from PIL import Image


mat = scipy.io.loadmat(os.path.join('..','DATA','allFaces.mat'))
```

```
X = mat['faces']
nfaces = mat['nfaces'].reshape(-1)
n = mat['n']
n = int(n)
m = mat['m']
m = int(m)

nTrain = 30
nTest = 20
nPeople = 20
Train = np.zeros((X.shape[0],nTrain*nPeople))
Test = np.zeros((X.shape[0],nTest*nPeople))

for k in range(nPeople):
    baseind = 0
    if k > 0:
        baseind = np.sum(nfaces[:k])
    inds = range(baseind,baseind+nfaces[k])
    Train[:,k*nTrain:(k+1)*nTrain] = X[:,inds[:nTrain]]
    Test[:,k*nTest:(k+1)*nTest] = X[:,inds[nTrain:(nTrain+nTest)]]
    for i in range (nTrain):
        im = Image.fromarray(np.reshape(Train[:,k*nTrain + i], (m,n)).T)
        im = im.convert('L')
        im.save('Train/train_subject_' + str(k) + '_sample_' + str(i) +
            '.png')
    for i in range (nTest):
        im = Image.fromarray(np.reshape(Test[:,k*nTest + i], (m,n)).T)
        im = im.convert('L')
        im.save('Test/test_subject_' + str(k) + '_sample_' + str(i) +
            '.png')
```

Smanjenje uzoraka

Iduća stavka koju je potrebno implementirati je metoda koja će smanjiti dimenzije originalne slike. U ovom sustavu implementiran je jednostavan algoritam koji originalne sliku dimenzija 168×192 skalira na sliku dimenzija 10×12 koristeći metodu *getScaledInstance* paketa java.awt.Image. Pri pozivu metode tražimo da koristi zadani algoritam za skaliranje gdje je nova vrijednost pojeding piksela prosječna vrijednost područja kojeg je pokrivao prije skaliranja.

Listing 4.2: Metoda downsampleImage

```
private static BufferedImage downsampleImage(BufferedImage
    originalImage, int targetWidth, int targetHeight) {
    Image resultingImage = originalImage.getScaledInstance(targetWidth,
        targetHeight, Image.SCALE_DEFAULT);
    BufferedImage outputImage = new BufferedImage(targetWidth,
        targetHeight, BufferedImage.TYPE_BYTE_GRAY);
    outputImage.getGraphics().drawImage(resultingImage, 0, 0, null);
    return outputImage;
}
```

Slika 4.2: Učinak metode downsampleImage na jednoj slici.

Pri treniranju i testiranju sustava nećemo koristiti same slike već njihove vektorske reprezentacije, zato implementiramo metodu grayscaleImageToColumnVector koja uzima crno-bijelu sliku, spljošti ju i kreira vektor kojem svaka komponenta odgovara jednom pikselu slike. Vrijednost pojedine komponente je cijeli broj u intervalu $[0, 255]$, gdje 0 označava da je pripadajući piksel potpuno crn, a vrijednost 255 označava da je pripadajući piksel potpuno bijel.

Listing 4.3: Metoda grayscaleImageToColumnVector

```
private static INDArray grayscaleImageToColumnVector(BufferedImage
    image) {
    INDArray array = Nd4j.zeros(image.getWidth() * image.getHeight(),
        1);
    Raster raster = image.getData();

    for (int i = 0; i < image.getWidth(); i++) {
        for (int j = 0; j < image.getHeight(); j++) {
            array.put(i * image.getHeight() + j, 0, raster.getSample(i,
                j, 0));
        }
```

```
        }
        return array;
    }
```

Matrica treniranja

Za potrebe treniranja sustava implementirano je gotovo sve što je potrebno. Još je potrebno iz skupa slika za treniranje kreirati matricu čiji su stupci vektorske reprezentacije slika sa smanjenim dimenzijama. Kako imamo metode downsampleImage i grayscaleImageToColumnVector, dovoljno je da za svaku sliku koju koristimo za treniranje učitamo, redom pozovemo te dvije metode te konstruiramo matricu tako što spojimo sve dobijene vektore. Upravo to radi funkcija concatImageColumns. Budući da pri testiranju sustava koristimo implementaciju PLM algoritma 1, za matricu treniranja Φ je potrebno izračunati dominatnu svojstvenu vrijednost δ (svojstvena vrijednost s najvećom apsolutnom vrijednosti) matrice $\Phi^T\Phi$. Broj δ aproksimiramo tako što implementiramo *metodu potencija* [6] s metodom findLargestEigenvalue. Matricu koju dobijemo pozivom metode concatImageColumns nazivamo *matrica treniranja* i označavamo ju sa Φ.

Listing 4.4: Metoda concatImageColumns

```
public static INDArray concatImageColumns() throws URISyntaxException,
    IOException {
    INDArray theta = Nd4j.zeros(ImageProcessor.columnSize,
        numberOfSamples * numberOfSubjects);
    for (int i = 0; i < numberOfSubjects; i++) {
        for (int j = 0; j < numberOfSamples; j++) {
            theta.putColumn(i * numberOfSamples + j,
                ImageProcessor.transformTrainImageToColumn(i, j));
        }
    }
    return theta;
}
```

Listing 4.5: Metoda findLargestEigenvalue

```
public static double findLargestEigenvalue(INDArray matrix) {
    INDArray vector = Nd4j.ones(matrix.rows(), 1);
    INDArray shownVector;
    double a = 1, aPrev = 1;
    for(int i = 0; i < 20; i++) {
        vector = matrix.mmul(vector);
```

```
            aPrev = a;
            a *= vector.normmax(0).getDouble(0,0);
            vector = vector.mul(1 / vector.normmax(0).getDouble(0,0));
        }
        return a / aPrev;
    }
```

4.2 Testiranje sustava

Sustav testiramo tako što mu dajemo novu sliku na upit koju nismo koristili pri njegovom treniranju i očekujemo da nam odgovori koja osoba se nalazi na slici. U sustavima za prepoznavanje lica baziranim na sažimajućem očitavanje pri testiranju potrebno je redom provesti korake:

- odrediti vektorsku reprezentaciju y slike upita sa smanjenom dimenzijom,
- za matricu treniranja Φ proizvoljnim algoritmom pronaći rijetko rješenje sustava $y = \Phi x$,
- ovisno o rješenju sustava x odgovoriti koja osoba se nalazi na slici upita.

Već pri treniranju sustava koristili smo metode downsampleImage i greyscaleImageToColumnVector kako bi smanjili dimenziju slike i izračunali njenu vektorsku reprezentaciju, iste dvije metode koristimo i na slici upita.

Implementacija algoritama za rekonstrukciju rijetkog vektora

Za potrebe sustava implementirana su dva algoritma, Prošireni Lagrangeov Multiplikator 1 i Orthogonal Matching Pursuit 3. Naknadno ćemo usporediti performanse oba algoritma kako bi odredili koji je pogodniji za ovaj problem i koji ćemo ultimativno koristiti u sustavu.

Prije nego implementiramo PLM algoritam, potrebno je implementirati *shrink* funkciju. Za proizvoljan $x \in \mathbb{R}^n$ i $\lambda \in \mathbb{R}$, shrink (x, λ) definiran je kao poziv funkcije *mekog praga* S_λ (2.6) na sve komponente vektora x. Implementacije softThreshold i shrink su vrlo prirodne i jednostavne.

Listing 4.6: Metoda softThreshold

```
public static double softThreshold(double x, double lambda) {
    return Math.signum(x) * Math.max(Math.abs(x) - lambda, 0);
}
```

Listing 4.7: Metoda shrink

```
public static INDArray shrink(INDArray array, double lambda) {
    INDArray newArray = Nd4j.zeros(array.rows(), 1);
    for(int i = -1; i < newArray.rows() - 1; i++) {
        newArray.put(i, 1, softThreshold(array.getDouble(i,1), lambda));
    }
    return newArray;
}
```

Metoda algorithmALM prati pseudokod PLM algoritma 1, jedino je važno napomenuti da je za uvijet konvergencije odabran fiksan broj iteracija za unutarnju i vanjsku petlju. Vanjsku petlju izvodimo 1000 puta dok unutarnju 5 puta, a za niz μ_k uzeli smo konstantan niz jer algoritam nije bio konzistentan za nizove koji se obično koriste u PLM algoritmu pa je $\forall k \in \mathbb{N}, \mu_k = \frac{1}{10000}$.

Listing 4.8: Metoda algorithmALM

```
public static INDArray algorithmALM(INDArray A, INDArray b) {
        INDArray x = Nd4j.zeros(A.columns(), 1);
        INDArray e;
        INDArray theta = Nd4j.zeros(b.rows(), 1);
        INDArray xPrev;
        INDArray z, w, wPrev;
        double t, tPrev;
        double L = ThetaUtil.findLargestEigenvalue(A.mmul(A.transpose()));
        double epsilon = 0.00001;
        int i;
        int k = 0;

        do {
            e = ShrinkOperatorUtil.shrink(b.sub(A.mmul(x)).add(theta.mul(1
                / epsilon)), 1 / epsilon);
            t = 1;
            z = x.dup();
            w = x.dup();
            i = 0;
            do {
                wPrev = w.dup();
                w = ShrinkOperatorUtil.shrink(z.add(A.transpose().mul(1 /
                    L).mmul(b.sub(A.mmul(z)).sub(e).add(theta.mul(1 /
                    epsilon)))), 1 / (L * epsilon));
                tPrev = t;
```

```
            t = 0.5 * (1 + Math.sqrt(1 + 4 * t * t));
            z = w.add(w.sub(wPrev).mul((tPrev - 1) / t));
        } while (i++ < 10);
        xPrev = x.dup();
        x = w.dup();
        theta = theta.add(b.sub(A.mmul(x)).sub(e).mul(epsilon));
    } while(++k < 1000);
    return x;
}
```

Pri implementiranju OMP algoritma potrebno je donijeti dvije odluke:

- Kako efikasno riješiti problem ortogonalizacije (2.8)?
- Koji je uvjet konvergencije?

Za problem ortogonalizacije korištena je QR faktorizacija [13] čija implementacija postoji u biblioteci ND4J. Glavna ideja je da zbog ortogonalnosti matrice Q, QR faktorizacijom problem ortogonalizacije možemo svesti na problem rješavanja sustava $R_{T_i}x_{T_i} = Q_{T_i}^T y$ koji možemo brzo riješiti jer je $R_{T_i} \in \mathbb{R}^{i \times i}$ gornje-trokutasta matrica. Za rješavanje sustava ponovno koristimo ND4J biblioteku.

Kao uvjet konvergencije odabran je fiksan broj k iteracija OMP algoritma. Za k je odabran broj slika pojedine osobe korištenih pri treniranju sustava, u našem slučaju je $k = 30$.

Listing 4.9: Metoda algorithmOMP

```
public static INDArray algorithmOMP(INDArray A, INDArray b) {
    Set<Integer> omega = new HashSet<>();
    Set<Integer> omegaC = new HashSet<>();
    for (int i = 0; i < A.columns(); i++) {
        omegaC.add(i);
    }
    int maxIndex;
    INDArray r = b.dup();
    INDArray[] decompositionQR;
    ArrayList<Integer> usedIndexesOrderList = new ArrayList<>();
    INDArray AOmega = Nd4j.zeros(A.rows(), 1);
    INDArray x = Nd4j.zeros(A.columns(), 1);
    for (int i = 0; i < ThetaUtil.numberOfSamples; i++) {
        maxIndex = findMaxIndex(A, r, omegaC);
        omega.add(maxIndex);
        omegaC.remove(maxIndex);
```

```
            usedIndexesOrderList.add(maxIndex);
            if (usedIndexesOrderList.size() == 1) {
                AOmega = convertToColumn(A.getColumn(maxIndex));
            } else {
                AOmega = Nd4j.hstack(AOmega,
                    convertToColumn(A.getColumn(maxIndex)));
            }
            NDLinalg linAlg = new NDLinalg();
            decompositionQR = linAlg.qr(AOmega);
            INDArray xOmega = linAlg.solve(decompositionQR[1],
                decompositionQR[0].transpose().mmul(b));
            for (int j = 0; j < usedIndexesOrderList.size(); j++) {
                x.put(usedIndexesOrderList.get(j), 0, xOmega.getDouble(j,
                    0));
            }

            r = b.sub(AOmega.mmul(xOmega));

        }
        return x;
    }
```

Očitavanje rješenja

Nakon što rekonstruiramo rijetki vektor x, potrebno je još pomoću njega odrediti nalazi li se na slici upita neka osoba iz $[C]$, i ako da, koja točno osoba je na slici. Postoji više algoritama koji se mogu koristiti i najčešće se biraju ovisno o algoritmu rekonstrukcije, ali kako radimo pojednostavljeni sustav, dovoljno dobar se pokazuje jednostavan algoritam opisan u prethodnom poglavlju.

Metoda *evaluateSubjectProbabilities* računa parametre $\alpha_k, \forall k \in [C]$ iz jednadžbe (3.2), dok metoda *evaluateSolution* ju koristi kako bi izračunala koja osoba se najvjerojatnije nalazi na slici i koliko je sustav siguran u tu odluku. Metoda rješenje vraća u obliku insance klase *SolutionModel*. U implementaciji klase vidi se da je za parametar praga ξ odabrana vrijednost 0.2.

Listing 4.10: Metoda evaluateSubjectProbabilities

```
    public static double[] evaluateSubjectProbabilities(INDArray x) {
        double xL1 = x.norm1(0).getDouble(0,0);
        double[] evaluations = new double[ThetaUtil.numberOfSubjects];
        for (int i = 0; i < ThetaUtil.numberOfSubjects; i++) {
```

```
            double sum = 0;
            for (int j = 0; j < ThetaUtil.numberOfSamples; j++) {
                sum += Math.abs(x.getDouble(i * ThetaUtil.numberOfSamples +
                    j));
            }
            evaluations[i] = sum / xL1;
        }
        return evaluations;
    }
```

Listing 4.11: Metoda evaluateSolution

```
    public static SolutionModel evaluateSolution(INDArray x) throws
        Exception {
        double[] subjectProbabilities = evaluateSubjectProbabilities(x);
        if(subjectProbabilities.length == 0) {
            return null;
        }
        int maxIndex = 0;
        double maxValue = subjectProbabilities[0];
        for(int i = 1; i < subjectProbabilities.length; i++) {
            if(subjectProbabilities[i] > maxValue) {
                maxIndex = i;
                maxValue = subjectProbabilities[i];
            }
        }
        return new SolutionModel(maxIndex, maxValue);
    }
```

Listing 4.12: Klasa SolutionModel

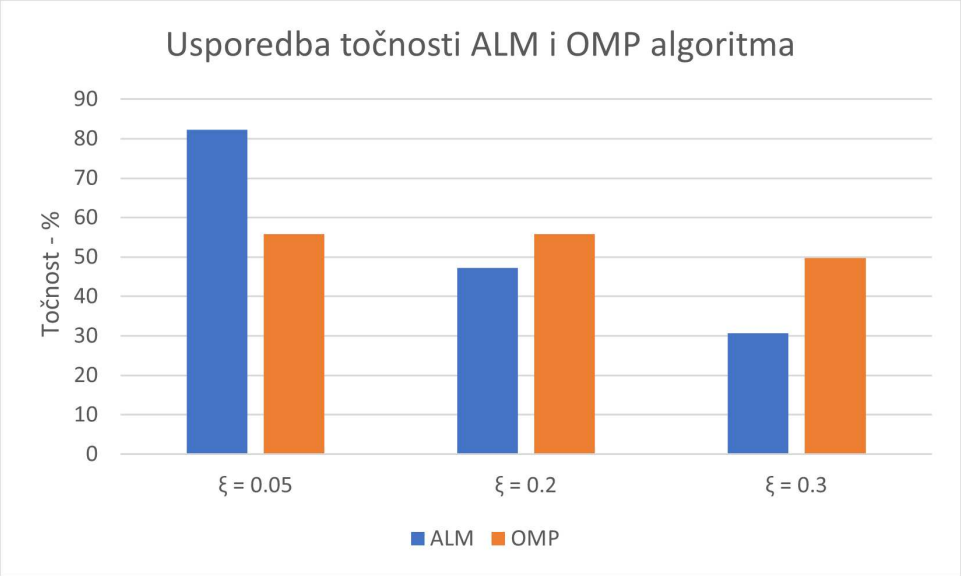
```
    public class SolutionModel {
        private int subject;
        private double probability;
        private boolean isCertain;
        private static final double xi = 0.2;

        public SolutionModel(int subject, double probability) {
            this.subject = subject;
            this.probability = probability;
            this.isCertain = xi < probability;
        }
```

```
    public int getSubject() {
        return subject;
    }

    public double getProbability() {
        return probability;
    }

    public boolean isCertain() {
        return isCertain;
    }
}
```

4.3 Usporedba algoritama

Algoritme OMP i PLM ćemo uspoređivati za tri različite vrijednosti parametra $\xi = 0.05$, 0.2, 0.3. Budući da sustav treniramo s dvadeset osoba, za $\xi = 0.05$ sustav će uvijek potvrditi da se na slici upita nalazi neka osoba iz $[C]$ i odgovor će biti osoba z s najvećom vrijednosti α_z. Sustav testiramo sa ukupno 400 slika, po dvadeset slika lica od dvadeset različitih osoba čije slike su korištene pri treniranju sustava.

Iz grafa 4.3 je očito da se za smislene parametre ξ OMP pokazuje kao bolji algoritam za sustave prepoznavanja lica, ali u slučaju da smo sigurni da se na slici upita nalazi neka osoba iz [C] PLM algoritam je znatno bolji. Naravno, puno je korisniji sustav koji može prepoznati da se na slici upita ne nalazi niti jedna poznata osoba pa za konačnu aplikaciju odabiremo OMP algoritam kao zadani. Za moguću nadogradnju sustava potrebno je istražiti alternativni algoritam za očitavanje rješenja koji bi koristili zajedno s PLM algoritmom. Idealan algoritam iskoristio bi dobru intuiciju PLM algoritma, ali ujedno bi sa sigurnošću odbio sliku upita koja ne pripada ni jednoj osobi u sustavu.



Slika 4.3: Prikazane su točnosti oba algoritma na testnom skupu slika za razne parametre ξ.

Bibliografija

[1] Apache, *Gradle 7*, `https://gradle.org/whats-new/gradle-7/`.

[2] A. Beck i M. Teboulle, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, (2009).

[3] D. Bertsekas, *Nonlinear Programming*, 2003.

[4] Deeplearning4j, *ND4J*, `https://deeplearning4j.konduit.ai/ND4J/tutorials`.

[5] Terence Tao Emmanuel Candes, Justin Romberg i David Donoho, *Stable Signal Recovery from Incomplete and Inaccurate Measurements*, (2005).

[6] James F. Epperson, *An Introduction to Numerical Methods and Analysis*, 2013.

[7] Simon Foucart i Holger Rauhut, *A Mathematical Introduction to Compressive Sensing*, 2013.

[8] A.S. Georghiades, P.N. Belhumeur i D.J. Kriegman, *From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose*, IEEE Trans. Pattern Anal. Mach. Intelligence **23** (2001), br. 6, 643–660.

[9] R. Glowinski i A. Marrocco, *Sur l'approximation par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet nonlinéaires. Rev Franc d'Automat Inform, Recherche Opérationnelle*, (1975).

[10] JetBrains, *Intellij IDEA Community*, `https://www.jetbrains.com/idea/download`.

[11] S. Krstulovic i R. Gribonval, *MPTK: Matching pursuit made tractable*, (2006).

[12] Yonina C. Eldar Kutynok i Gitta Kutyniok, *Compressed Sensing*, 2012.

[13] John Lambert, *Solving Least-Squares with QR*, `https://johnwlambert.github.io/least-squares/#qr-for-lstsqr`.

[14] S. Mallat i Z. Zhang, *Matching pursuits with time-frequency dictionaries*, 1993.

[15] M. Marjanović, *SparseRepresentationClassification*, `https://github.com/rioma101/SparseRepresentationClassification`, 2022.

[16] Oracle, *Java SE 11*, `https://www.oracle.com/java/technologies/javase/jdk11-archive-downloads.html`, 2019.

[17] Marodi Petra, *Metoda unutrašnje točke u kvadratičnom programiranju*, 2019.

[18] B. D. Rao S. F. Cotter, J. Adler i K. Kreutz-Delgado, *Forward sequential algorithms for best basis selection*, (1999).

[19] G. Davis S. Mallat i Z. Zhang, *Adaptive time-frequency decompositions*, (1994).

[20] Joel A. Tropp i Stephen J. Wright, *Computational Methods for Sparse Solution of Linear Inverse Problems*, (2010).

[21] A. Wagner, J. Wright, A. Ganesh, Z. Zhou i Y. Ma., *Toward a practical automatic face recognition system: robust pose and illumination via sparse representation*, (2009).

[22] J. Wright, A. Ganesh A. Yang, S. Sastry i Y. Ma., *Robust face recognition via sparse representation*, (2009).

Sažetak

U ovom radu dan je uvod u sažimajuće očitavanje gdje su objašnjeni osnovni pojmovi, rezultati i neke najpoznatije primjene ovog aktualnog, brzo rastućeg područja matematike. Pokazano je kakvi uvjeti moraju biti zadovoljeni ako želimo računati egzaktno rješenje problema i zašto zapravo rijetko računamo egzaktno rješenje.

Analizirane su dvije najzastpuljenije familije algoritama za aproksimiranje rješenja, dani su i neki konkretni primjeri algoritama te navedene neke njihove prednosti i nedostatci. Posebni naglasak stavljen je na algoritme Orthogonal Matching Pursuit (OMP) i Prošireni Lagrangeov Multiplikator (PLM) koji su naknadno implementirani i čije perfromanse su uspoređene za problem prepoznavanja lica.

Detaljnije je obrađena primjena sažimajućeg očitavanja u problemu klasifikacije, specifično u sustavima za prepoznavanje lica. Opisano je kako problem prepoznavanja lica postaviti kao klasični problem sažimajućeg očitavanja.

Na kraju rada, opisana je aplikacija koja implementira sustav za prepoznavanje lica baziranom na sažimajućem očitavanju. Za treniranje i testiranje sustava korištena je extended Yale B baza podataka lica, a sama aplikacija napisana je u programskom jeziku Java koristeći biblioteku ND4J za brze matrične operacije.

Summary

This paper gives an introduction to compressed sensing where the basic terms, results and some of the most famous applications of this rapidly growing field of mathematics are explained. It is shown which conditions must be met if we want to calculate the exact solution to the problem and why we rarely calculate the exact solution.

The two most common families of algorithms for approximating solutions are analyzed, some specific examples of algorithms are given, and some of their advantages and disadvantages are listed. Special emphasis was placed on the Orthogonal Matching Pursuit (OMP) and Augmented Lagrange Multiplier (ALM) algorithms, which were subsequently implemented and whose performances were compared for the face recognition problem.

The application of compressed sensing in the problem of classification, specific to facial recognition systems, is treated in more detail. It is described how the problem of face recognition is posed as a classic problem of compressed sensing.

At the end of the paper, an application that implements a face recognition system based on summarizing reading is described. The extended Yale B face database is used for training and testing the system, and the application itself is written in the Java programming language using the ND4J library for faster matrix operations.

Životopis

Rođen sam 1. kolovoza 1998. godine u Vinkovcima. Pohađao sam Osnovnu školu Stjepana Antolovića Privlaka u Privlaci. Nakon toga upisujem Tehničku školu Ruđera Boškovića Vinkovci, smjer strojarski računalni tehničar. 2017. godine upisujem preddiplomski sveučilišni studij Matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu. Njega završavam 2020. godine te iste godine upisujem smjer Računarstvo i matematika.