Vaja 6: Redukcija dimenzije in razpoznavanje

Laboratorijske Vaje

31. december 2011

V naslednjih nalogah bomo v praksi preizkusili metodo glavnih komponent (angl., principal component analysis – PCA), ki ste ga obravnavali na predavanjih. Najprej bomo obravnavali preprost 2D primer direktne metode PCA, ki si jo lažje vizualiziramo. Ta primer boste tudi rešili na papir, hkrati pa si boste v Octave/Matlab vizualizirali kar računate. V nadaljevanju boste preizkusili *dualni* postopek PCA na primeru kompaktnega zapisa obrazov ter preizkusili nekaj lepih lastnosti takega opisa. Teorijo glede PCA in njeno uporabo za klasificiranje lahko preberete v [1] (poglavje 22.3.1, strani 507–515).

1 Direktna metoda PCA

Primarni namen PCA je redukcija dimenzije, kar pomeni, da želimo poiskati tako linearno transformacijo vhodnih podatkov, ki bo naše podatke zapisala v nizkodimenzionalnem ortogonalnem prostoru (imenovanem tudi podprostor), ob tem pa bomo minimalno izgubili informacijo potrebno za rekonstrukcijo podatkov. Podprostor napenjajo tako imenovani bazni vektorji. Najprej na kratko ponovimo direktni izračun baznih vektorjev z metodo PCA. Predpostavljamo, da so vhodni podatki stolpični vektorji $\mathbf{x}$ velikosti $m \times 1$ (imajo m vrstic):

Algorithm 1 : Algoritem direktna PCA.

1: Zgradimo matriko $\mathbf{X}$ velikosti $m \times N$ tako, da N vhodnih podatkov zložimo zaporedno: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$

2: Izračunamo srednjo vrednost podatkov: $\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$

3: Podatke centriramo: $\mathbf{X}_d = [\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_N - \mu]$

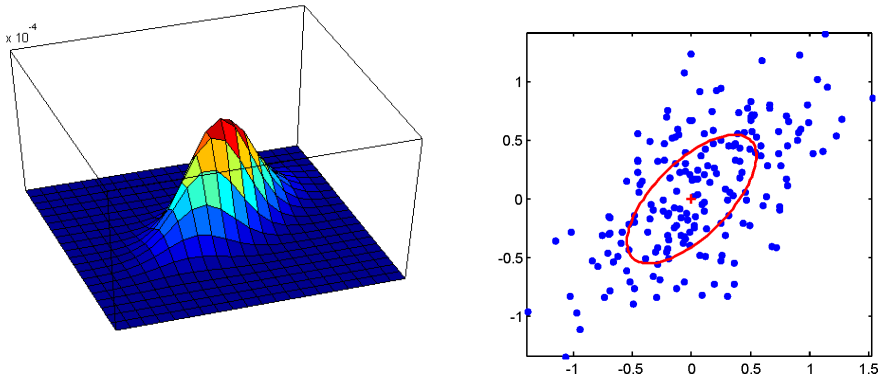
4: Izračunamo kovariančno matriko: $\mathbf{C} = \frac{1}{N-1} \sum_{i=1}^{N} (\mathbf{x_i} - \mu)(\mathbf{x_i} - \mu)^T = \frac{1}{N-1} \mathbf{X}_d \mathbf{X}_d^T$

5: Izračunamo SVD (angl., singular value decomposition) kovariančne matrike: $\mathbf{C} = \mathbf{USV}^T$ Matrika $\mathbf{U}$ je velikosti $m \times m$, njeni stolpci pa predstavljajo lastne vektorje matrike $\mathbf{C}$. Pripadajoče lastne vrednosti so shranjene v diagonalni matriki $\mathbf{S}$ in so urejene od največje do najmanjše. Stolpci $\mathbf{U}$ torej predstavljajo ortogonalno bazo (podprostor), lastne vrednosti pa nam povejo raztros vhodnih podatkov vzdolž posameznega lastnega vektorja.

6: Nov vhodni podatek $\mathbf{x}_q$ projiciramo v PCA prostor (podprostor) tako, da ga najprej centriramo z μ, nato pa pomnožimo z matriko lastnih vektorjev: $\mathbf{y} = \mathbf{U}^T(\mathbf{x}_q - \mu)$.

7: Transformacija s podprostora v originalni prostor sledi obratni enačbi: $\mathbf{x}_q = \mathbf{U}\mathbf{y} + \mu$.

V zgornjem algoritmu vidimo, da v prvih korakih metode PCA izračunamo srednjo vrednost in kovariančno matriko podatkov. To sta ravno dva parametra, ki definirata Gaussovo porazdelitev. Ker se bomo na nadaljevanju sklicevali na Gaussovo porazdelitev se ji na tem mestu še malo posvetimo. Gaussovo porazdelitev ste že spoznali pri Vaji 2, kjer smo uporabljali posebno



(a) (b)

Slika 1: Leva slika prikazuje tabeliran primer 2D Gaussove porazdelitve, desna slika pa prikazuje primer porazdelitve kjer izrišemo samo vrednosti $\mathbf{x}$ ob konstantni predpisani verjetnosti.

obliko take porazdelitve za Gaussovo jedro. Tista oblika je imela srednjo vrednost enako nič in diagonalno kovariančno matriko z enakima kovariancama vzdolž x in y osi. V splošnem pa je formula za Gaussovo porazdelitev:

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^d |\mathbf{C}|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{C}^{-1}(\mathbf{x}-\mu)) \tag{1}$$

Slika 1(a) prikazuje primer Gaussove porazdelitve, kjer je na osi y izrisana verjetnost $p(\mathbf{x})$, v Sliki 1(b) pa vidimo alternativno ilustracijo, kjer izrišemo mejo pri kateri je znotraj elipse 68% vse gostote verjetnosti. Slednji prikaz bomo uporabljali v naši nalogi za vizalizacijo lastnih vrednosti in komponent. V desni sliki so prikazane točke s katerih izračunamo srednjo vrednost in kovarianco, ti pa definirata izrisano Gaussovo porazdelitev.

1.1 Kako analitično računamo lastne vektorje in lastne vrednosti?

Na kratko ponovimo kako analitično izračunamo lastne vrednosti in lastne vektorje kvadratne matrike $\mathbf{C}$, ki je velikosti $m \times m$. Zamislimo si matriko $\mathbf{Z}$, ki jo dobimo, če z neko konstanto pomnožimo enotsko matriko velikosti $m \times m$, npr., $\mathbf{Z} = \lambda\mathbf{I}$. Če to matriko odštejemo od $\mathbf{C}$ in je determinanta razlike nič, pravimo, da smo transformirali matriko $\mathbf{C}$ v singularno:

$$|\mathbf{C} - \lambda\mathbf{I}| \equiv 0. \tag{2}$$

Vrednosti λ, ki zadovoljijo zgornjo enačbo imenujemo *lastne vrednosti* matrike $\mathbf{C}$. Za $m \times m$ matriko izračunamo m lastnih vrednosti λ_i. Produkt lastnih vrednosti je enak determinanti matrike: $|\mathbf{A}| = \prod_{i=1}^{m} \lambda_i$. Vsaki lastni vrednosti λ_i pripada *lastni vektor* $\mathbf{x}_i$, lastni vektorji pa so si med sabo pravokotni (skalarni produkt je nič).

Za lastno vrednost λ_i izračunamo lastni vektor $\mathbf{x}_i$ velikosti $1 \times m$ tako, da nastavimo spodnjo enačbo:

$$(\mathbf{C} - \lambda_i\mathbf{I})\mathbf{x}_i \equiv 0. \tag{3}$$

Rešitev te enačbe nam da elemente lastnega vektorja do skale natančno. Torej, zgornjo enačbo reši neskončno mnogo vektorjev, ki se med seboj razlikujejo le po skali. V praksi za prvi element vektorja izberemo 1, nato izračunamo vse ostale elemente lastnega vektorja. Na koncu nastavimo skalo vsakega lastnega vektorja tako, da je njegova dolžina 1 (enotski vektor). Recimo, da $\mathbf{x}_i$ reši zgornjo enačbo. Enotski lastni vektor za λ_i je definiran kot $\mathbf{e}_i = \mathbf{x}_i / ||\mathbf{x}_i||$.

Nekoliko daljšo razlago in primere izračuna lastnih vrednosti najdete na:
`http://www.miislita.com/information-retrieval-tutorial/matrix-tutorial-3-eigenvalues-eigenvectors.html`.

2 Preprost primer direktne PCA

(a) S spletne strani predmeta si prenesite `vaja6.zip`, vsebino odpakirajte v vaše delovno okolje in zaženite `vaja6_naloga2a_f()`. Funkcija vam bo naložila in vizualizirala 2D podatke z datoteke `data/vaja6_2D_tocke.txt`. V to funkcijo boste pisali naslednje korake naloge.

(b) Po Algoritmu 1 izračunajte PCA (lastne vektorje in lastne vrednosti) podatkov v `vaja6_2D_tocke.txt`. Za dekompozicijo kovariančne matrike $\mathbf{C}$ uporabite dekompozicijo po singularnih vrednostih `[U,S,V]=svd(C)`. Kovariančno matriko in srednjo vrednost prikažite na grafu, kamor ste izrisali točke. Za izris uporabite funkcijo `plotgauss2d(Mu, C, 'r', 1 )`.

(c) Matrika $\mathbf{U}$ je sestavljena iz lastnih vektorjev, ki predstavljajo bazo našega podprostora. Na sliko, ki ste jo izrisali v prejšnji točki si izrišite lastne vektorje matrike $\mathbf{C}$ z izhodiščem v srednji vrednosti podatkov μ. Lastna vektorja sta dolžine ena. Za boljšo vizualizacijo vsakega pomnožite s pripadajočo lastno vrednostjo[1], preden jih premaknete v izhodišče μ. Prvi lastni vektor izrišite z zeleno barvo, drugega pa z rdečo. Komentirajte kar opazite.

(d) Na predavajih ste že slišali, da lastne vrednosti predstavljajo rekonstrukcijsko napako, ki jo storimo, če pripadajoči lastni vektor zavržemo. Izrišite kumulativni graf lastnih vrednosti in ga normalizirajte tako, da bo največja vrednost v grafu 1. Odčitatjte z grafa koliko odstotkov rekonstrukcijske informacije obdržimo, če obdržimo le prvi lastni vektor, drugega pa zavržemo. Rečeno drugače, *kolikšen odstotek variance* je pojasnjen samo s prvim lastnim vektorjem? Za izračun kumulativnih vrednosti uporabite funkcijo `cumsum()`.

(e) Sedaj bomo vhodnim podatkom odstranili *smer najmanjše variacije* – podatke bomo projicirali v podprostor prvega lastnega vektorja. To storimo tako, da najprej podatke transformiramo v PCA prostor, postavimo na nič vrednosti tistih koordinatnih osi, ki jim želimo odstraniti varacijo in nato podatke transformiramo nazaj v originalni prostor. Tako rekonstruirane podatke si izrišite na sliko s točke (c) zgoraj. Za izris lahko uporabite funkcijo `izrisiRekonstrukcije2D(P_orig,P_rec)`, ki vzame za vhod originalne in rekonstruirane podatke `P_orig`, `P_rec` in izriše med njih povezave za boljšo vizualizacijo projekcije. Kaj opazite? Kam se (geometrijsko gledamo) projicirajo vhodne točke?

(f) Za točko $q_{\text{point}} = [3, 6]^T$ izračunajte najbližjo točko med vhodnimi točkami po evklidski razdalji. Katera točka je to? Sedaj projicirajte to točko, kakor tudi vhodne točke v podprostor, ki mu odstranite variacijo v smeri drugega lastnega vektorja. Kateri vhodni točki je v tem podprostoru najbližja q_{point}? Komentirajte rezultat zakaj je tako. Projekcijo (rekonstrukcijo) točke q_{point} si tudi izrišite na sliko s prejšnje alineje.

(g) Sedaj še analitično izračunajte lastne vektorje, lastne vrednosti in projekcijo vhodnih točk na prvi lastni vektor. Ali dobite enake rezultate kot pri vaši implementaciji PCA? Rešeno nalogo napišite na list in prinesite na zagovor.

3 Dualna metoda PCA

V primerih, ko imamo opravka z zelo visoko dimenzionalnimi podatki in je dimenzija podatkov višja kot pa število vzorcev, direktna metoda za izračun lastnih vektorjev, ki smo jo uporabili v

[1]Kot zanimivost: Lastna vrednost pove varianco podatkov vzdolž lastnega vektorja. Torej, če pomnožite lastni vektor s korenom lastne vrednosti, si boste vizualizirali standardno deviacijo vzdolž lastnega vektorja.

prejšnji nalogi ni primerna. Če imamo na primer, 10304 dimenionalne podatke, bo kovariančna matrika dimenzije 10304×10304, pri čemer že lahko naletimo na omejitve računalniškega pomnilnika. Ker pa je število podatkov/vzorcev N signifikantno manjše, lahko uporabimo tako imenovano *metodo dualne* PCA. Metoda se od direktne zgoraj razlikuje le v četrtem in petem koraku (primerjajte Algoritem 1 in Algoritem 2).

Za naše potrebe je bistveno izračunati pravilno lastne vektorje, lastne vrednosti pa le do skalirnega faktorja natančno. Zato implementirajte dualni postopek po Algoritmu 2 in ga testirajte na podatkih `vaja6_2D_tocke.txt` iz točke 2b. Preverite ali dobite enake lastne vektorje (matrika $\mathbf{U}$) kot pri postopku z Algoritmom 1. Prva dva lastna vektorja bi morala biti enaka kot pri Algoritmom 1, medtem, ko pri postopku (Algoritmu 2) dobite večjo matriko $\mathbf{U}$, vendar so vsi lastni vektorji razen prvih dveh enaki nič. Za samo projekcijo v podprostor in iz njega lahko matriko $\mathbf{U}$ pustite kot je. Kot končni preizkus, projicirajte podatke iz točke 2b v PCA prostor z izračunano matriko $\mathbf{U}$, nato pa podatke projicirajte nazaj v originalni prostor. Če ste projekcije izvedli pravilno, morate dobiti (do numerične napake natančno) vhodne podatke.

Algorithm 2 : Algoritem dualna PCA.

1: Zgradimo matriko $\mathbf{X}$ velikosti $m \times N$ iz stolpičnih podatkov $\mathbf{x}_i$ dimenzije $1 \times m$ tako, da N vhodnih podatkov zložimo zaporedno: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$
2: Izračunamo srednjo vrednost podatkov: $\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$
3: Podatke centriramo: $\mathbf{X}_d = [\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_N - \mu]$
4: Izračunamo dualno kovariančno matriko: $\tilde{\mathbf{C}} = \frac{1}{m-1} \mathbf{X}_d^T \mathbf{X}_d$. Pozor: normalizacija z vrednostjo $\frac{1}{m-1}$ je tu zgolj zaradi definicije kovariančne matrike, vendar ta vrednost ne vpliva na izračun lastnih vektorjev in jo lahko tudi izpustite.
5: Izračunamo SVD (angl., singular value decomposition) dualne kovariančne matrike, $\tilde{\mathbf{C}} = \tilde{\mathbf{U}}\tilde{\mathbf{S}}\tilde{\mathbf{V}}^T$, in preračunamo bazo lastnih vektorjev: $\mathbf{U} = \mathbf{X}_d \tilde{\mathbf{U}} \tilde{\mathbf{S}}^{-1/2}$. Pozor, pričakovati je, da bo od nič različnih le prvih m ali N stolpcev lastnih vektorjev (tista vrednost, ki je nižja).
6: Nov vhodni podatek $\mathbf{x}_q$ projiciramo v PCA prostor (podprostor) tako, da ga najprej centriramo, nato pa pomnožimo z matriko lastnih vektorjev po enačbi: $\mathbf{y} = \mathbf{U}^T(\mathbf{x}_q - \mu)$.
7: Transformacija s podprostora v originalni prostor sledi obratni enačbi: $\mathbf{x} = \mathbf{U}\mathbf{y} + \mu$.

4 Dekompozicija slike in umetno generiranje osvetlitve

V nadaljevanju bomo implementacijo dualne metode PCA preizkusili na problemu iz računalniškega vida. V podatkih, ki ste jih dobili skupaj z navodili so vam na voljo tri serije slik. Vsaka serija vsebuje 64 slik obraza pod različnimi orientacijami in intenzivnostimi osvetlitve. Vaša naloga je, da za posamezno serijo slik analizirate s pomočjo metode PCA.

(a) **Priprava podatkov**: Prva stvar, ki jo moramo narediti je, da problem formuliramo v obliki, primerni za metodo PCA. Kot ste spoznali v prejšnjih nalogah, PCA deluje nad točkami v prostoru. Sivinsko sliko velikosti $m \times n$ lahko predstavimo kot točko v mn-dimenzionalnem prostoru tako, da sivinsko sliko po stolpcih razvijemo v vektor. Napišite funkcijo, ki slike ene izmed treh zbirk (številko zbirke dobi funkcija kot vhodni parameter) prebere v pomnilnik, jih po potrebi transformira v sivinske, ter razvije v vrstične vektorje (z metodo `reshape()`). Vektorje nato zloži po vrsticah skupaj v matriko velikosti $64 \times mn$. Slednjo matriko funkcija vrne kot rezultat. V nadaljnih nalogah kot referenčno serijo uporabljajte prvo serijo slik, seveda pa lahko naloge preizkusite tudi z drugima serijama slik.

(b) **Uporaba dualne PCA**: Nad vektorji, ki ste jih pridobili iz slik, uporabite metodo dualne PCA. Metodo zapišite v obliki funkcije, ki ji na vhod podate matriko vektrojev, funkcija pa vrne lastne vektorje PCA podprostora ter srednjo vrednost podatkov.

Pri koraku 5 v Algoritmu 2 bodite pozorni na izračun inverza $\tilde{\mathbf{S}}$, saj so lahko nekatere lastne vrednosti zelo blizu 0. Pri inverzu tako lahko deljenje z nič povzroči numerične napake. Upoštevati morate, da je matrika $\tilde{\mathbf{S}}$ diagonalna in mora imeti po diagonali elemente različne od nič. En način kako rešiti ta numerični problem je, da vsem vrednostim diagonale prištejete neko zelo majhno konstanto, na primer 10^{-15}. Numerično stabilen izračun inverza $\tilde{\mathbf{S}}$ lahko implementirate takole:

```
s = diag(S) + 1e-15; % Diagonalo spremenimo v vektor in mu prištejemo konstanto
Si = diag(1 ./ s); % Izračunamo inverz in spremenimo v matriko
```

Prvih pet lastnih vektorjev s pomočjo funkcije `reshape` preoblikujte nazaj v matriko in jih prikažite na zaslonu kot slike (za prikaz je priporočljiva uporaba funkcije `imagesc`). Kako si razlagate rezultat? Projicirajte prvo sliko iz serije v PCA prostor ter nato nazaj. Je rezultat enak? Kaj opazite, ko spremenite eno izmed komponent vektorja v prostoru slike (na primer komponento z indeksom 4074) na 0 in sliko prikažite na zaslonu? Nato podobno operacijo naredite za vektor v PCA prostoru (projicirajte vektor slike v PCA prostor, spremenite eno izmed prvih petih komponent, recimo na nič, vektor projicirajte nazaj v prostor slike ter sliko prikažite). Kakšna je razlika? Na koliko slikovnih elementov ima vpliv prva sprememba in na koliko druga? Svojo kodo pišite v skripto `vaja6_naloga4b.m`.

(c) **Vpliv števila komponent na rekonstrukcijo:** Sliko z zaporedno številko 32 projicirajte v PCA prostor. Nato vektor v PCA prostoru spremenite tako, da ohranite le 32 prvih komponent, ostale pa postavite na 0. Vektor nato preslikajte nazaj v prostor slike in sliko prikažite. Ponovite postopek še za prvih 16, 8, 4, 2 in en lastni vektor. Rezultate transformacij prikažite skupaj v enem oknu. Kaj opazite? Kodo pišite v skripto `vaja6_naloga4c.m`.

(d) **Informacijska vsebina posamezne komponente**: Ta del naloge naredite na drugi bazi slik. Vzemite povprečno sliko, ki jo izračunate iz vseh slik izbrane baze[2]. Sliko preslikajte v PCA prostor. Nato vektorju v PCA prostoru spremenite vrednost druge ali tretje komponente (dodajte ali odvzamite 2) ter vektor preslikajte nazaj v slikovni prostor. Prikažite izvorno povprečno sliko in preoblikovano sliko. Kaj opazite? Nato spreminjanje druge in tretje komponente implementirajte še v zanki, kjer se druga komponenta spreminja po formuli $5 \cdot sin(x)$, tretja komponenta pa po formuli $5 \cdot cos(x)$. Variirajte vrednosti x od -20 po koraku 20 do 500. Vsako spremembo v PCA prostoru preslikajte nazaj v slikovni prostor in slike prikažite v animirani obliki. Vsako spremenjeno sliko posebej prikažite v istem oknu za 0.1 sekundo, nato pa prikažite naslednjo sliko. Tako boste dobili animacijo in boste lažje opazovali kako posamezna komponenta vpliva na sliko. Za izhodišče in primer animiranja rezultatov uporabite spodnjo kodo:

```
function vaja6_naloga4d()
% Tukaj zračunate PCA
for x = -20:20:500

    % Tukaj zračunate preslikavo, jo spremenite
```

[2]Vrednost vsakega slikovnega elementa je enaka povprečni vrednosti soležnih slikovnih elementov vseh slik. V primeru vektorjev slik pa gre preprosto za njihovo povprečje.

```
    % in sliko shranite v spremenljivko image.

    figure(1); imshow(uint8(image)) ;
    pause(0.1);
end;
```

Literatura

[1] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 2002.