Projektni zadatak iz Umjetne inteligencije

Validacija k-means i knn algoritama za prepoznavanje rukom pisanih brojeva

Jure Šiljeg

Siječanj, 2014.

UVOD :

Cilj projeknog zadatka je bio testirati uspješnost 2 algoritma za prepoznavanje rukom pisanih znamenki. Ta 2 dobro poznata algoritma za ovakvu (ili sličnu) problematiku su : k-means algoritam & knn algoritam. Također ćemo pokušati proširiti malo samu projektnu temu i igrati se raznim normama, k- ovima, vremenima izvršavanja, razlozima zašto bi nešto moglo biti baš takvo kakvo jest, ako je loše kako možda poboljšati (ili barem gdje se može o tome informirati dodatno), tokom i procesom rješavanja i problemima koji su se našli na putu i slično. Za kraj ćemo se kritički osvrnuti na samu uspješnost algoritama.

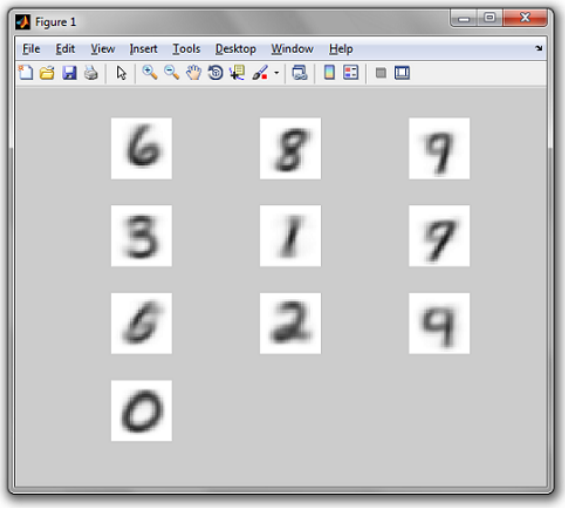
PROBLEMI I RJEŠENJA:

Za početak je zadatak bio koristiti implementirani k-means algoritam (s prošlih vježbi), testirati ga na 60k[1] podataka o rukom pisanim brojevima tako što ćemo najprije podijeliti skup zapisa `X` u 10 klastera pomoću `mykmeans` i to spremiti u matricu `idx`, a sam algoritam vidljiv je iz datoteke `mykmeans.m` (u nastavku ćemo `.m` datoteke samo nazivati fileovi). Potom je bilo potrebno "nacrtati" sve te podatke nekako, odnosno testirati je li naš k-means podijelio podatke u 10 klastera onako kako mi to želimo, jer nam je potrebno 10 klastera koji donekle odgovaraju onome s čim se u stvarnosti i susrećemo (nama je potrebno da unutar tih dodijeljenih 10 klastera bude moguće prepoznati znamenke 0,...,9). Naravno, kako je i bilo za očekivati, `mykmeans` neće to uvijek činiti onako kako mi to želimo, pa je stoga potrebno više puta (čitaj 6 ili 7 puta po cca 10 min) pokrenuti program dok konačno ne dođemo do željenog rezultata. Najprije je želja bila implementirati

[1] Radi se o podacima s ove stranice : http://yann.lecun.com/exdb/mnist/

potpuno jednostavan `kameans`[2] (radi se o istoimenom fileu), a još nam tu koristi i file `distMatrix` koja računa matricu udaljenosti), ali se ubrzo dalo naslutiti kako vrijeme izvršavanja neće ići "na ruku", pa je sljedeća misao bila implementirati po pseudokodu sa stranice za prošlu vježbu i to je rezultiralo sa 6 ili 7 pokušaja da dobijem ovakve rezultate :

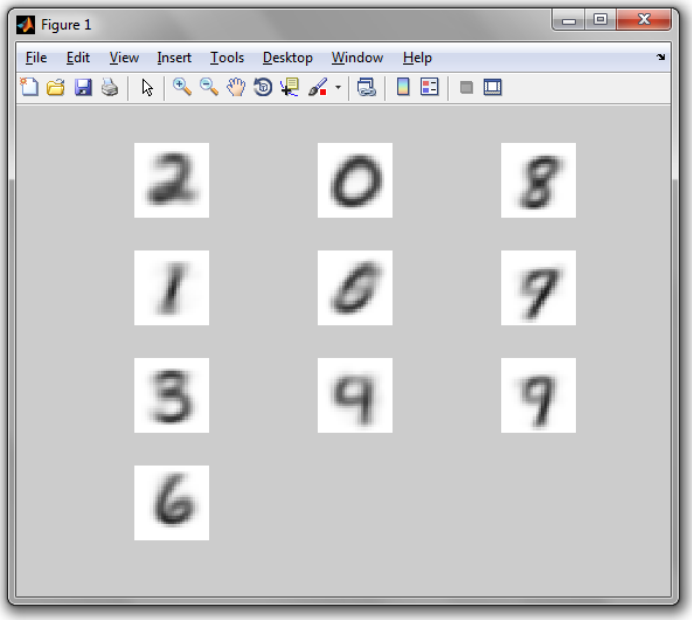
```
>> fid = fopen('train-images.idx3-ubyte', 'r', 'b');
header = fread(fid, 1, 'int32');
count = fread(fid, 1, 'int32');
h = fread(fid, 1, 'int32');
w = fread(fid, 1, 'int32');
offset=0;
readDigits=60000;
imgs = zeros([h w readDigits]);
for i=1:readDigits
    for y=1:h
        imgs(y,:,i) = fread(fid, w, 'uint8');
    end
end

X=reshape(imgs,h*w,readDigits);

%%%%%%%%%%%%
>> idx=mykmeans(X,10);
>> pridijeli
>>
```



Sa same slike je očit gornji dio koda gdje se učitava gore spomenuta datoteka i nakon toga pokušava izračunati `mykmeans` i nacrtati rezultate koji se već nekako daju interpretirati. Naravno, tuga ulovi čovjeka zašto mu prošli algoritam ne valja, pa za slučaj da ima vremena (a ima) onda krene testirati algoritam koji je prvi bio implementiran i pusti mu na volju da potroši svo vrijeme svijeta i dobije nešto ovakvo :

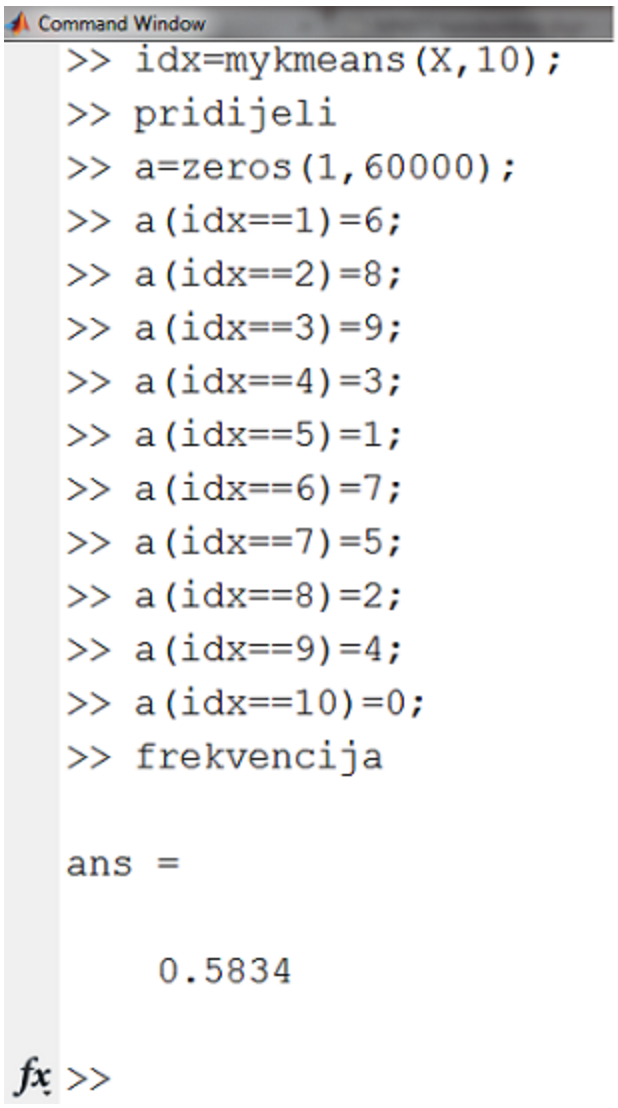
[2] Ideja dobivena na inače odličnim stranicama Dr. Kardija Tekmona : http://people.revoledu.com/kardi/

```
Command Window
>> tic;
idxx=kameans(X',10,1);
toc;
Elapsed time is 10609.933025 seconds.
>> edit pridijeli
>> pridijeli
fx >>
```



Prva reakcija : "Hm, čitljivo (a i nije potrošio svo vrijeme svijeta, izgleda)". Sljedeća reakcija : " Treba mu ipak stotinu gladnih godina[3], ali iz prve je dao ono što ovaj prošli nije ni iz više pokušaja !" Primijetiti valja kako je u prvom slučaju matrica X kao ulazni parametar prošla "bez transponiranja", dok to nije bio slučaj kod drugog (prvog!) pokušaja (čisto razlika u implementaciji). Bilo kako bilo, daljnji testovi će se vršiti za prvu sliku i algoritam `mykmeans`. Sljedeće što treba napraviti jest to da dodijelimo "ručno" kategorijama koje je `mykmeans` odredio neka malo konkretnija "imena", pa smo tako u sljedećoj situaciji (naravno, podaci trebaju odgovarati "stvarnom" stanju) :

[3] Točnije, trebalo mu je nešto manje od 3 sata što je sitnica obzirom na ostatak testiranja koji slijede. Naravno, autor ovih redaka to tada nije znao



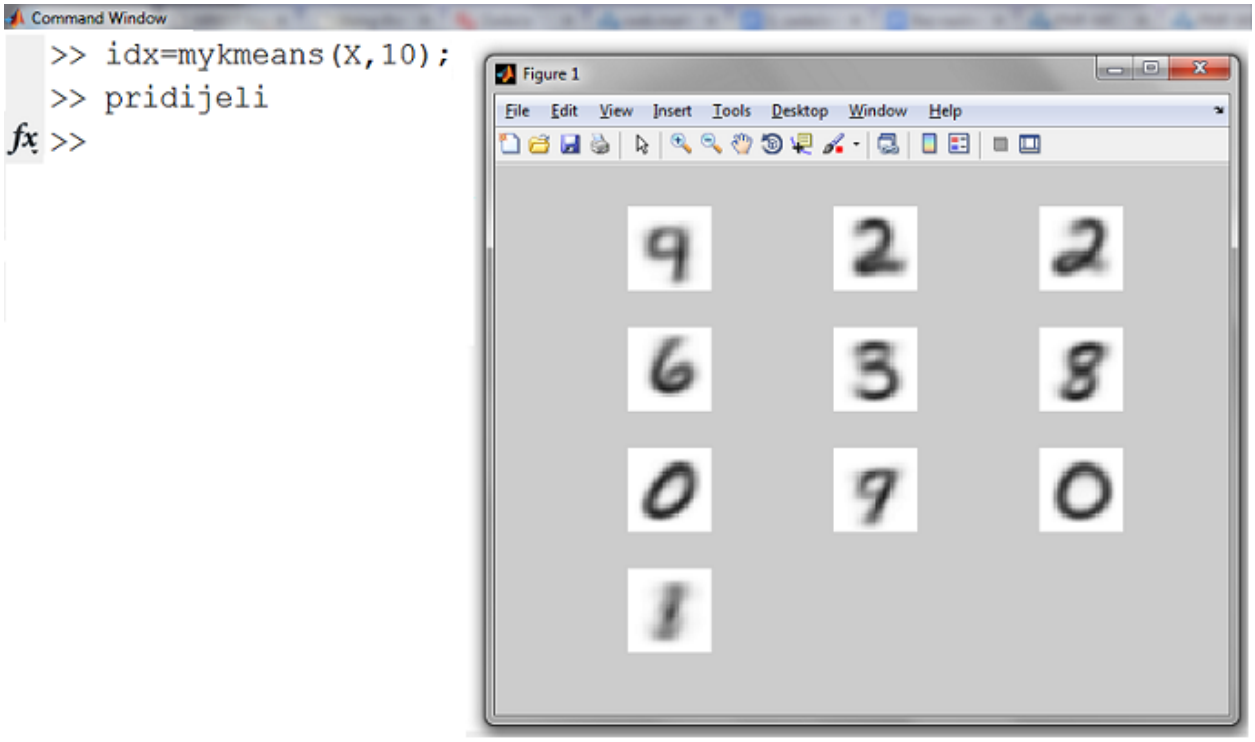
Ovdje se da primijetiti kako smo zapravo opisali stvarno stanje s dobivene slike naredbom `pridijeli`.

Kreirali smo nulmatricu `a` potrebnih dimenzija i u nju redom spremili ono što se sa slike dalo iščitati.

Sljedeći korak u zadaći je bio izvijestiti o frekvenciji uspješnosti. U tu svrhu je stvorena nova skripta `frekvencija` koja je za početak učitavala točne rezultate (njih 60k) iz one datoteke s početka u kojoj su bili "sadržani" podaci o rukom pisanim brojevima i trebalo se onda procijeniti koliko ih je algoritam "pogodio". Rezultat je bio (očekivanih) `58.34%`.

Svi primjeri do sada pokazani za `mykmeans` su rađeni za euklidsku udaljenost (2 - normu) što se da vidjeti u fileu `udaljenost`, pa je zanimljivo bilo vidjeti što se događa ako pokušamo neku drugu normu postaviti i pokušati dobiti raspoznatljivu sliku. Za početak je vrijedilo iskušati 1-normu[4] (još se u literaturi spominje kao Taxicab norma ili Manhattan norma) koja se vidi u fileu `udaljenost2`:

[4] Više o samoj normi se može pročitati ovdje :
http://en.wikipedia.org/wiki/Norm_(mathematics)#Taxicab_norm_or_Manhattan_norm



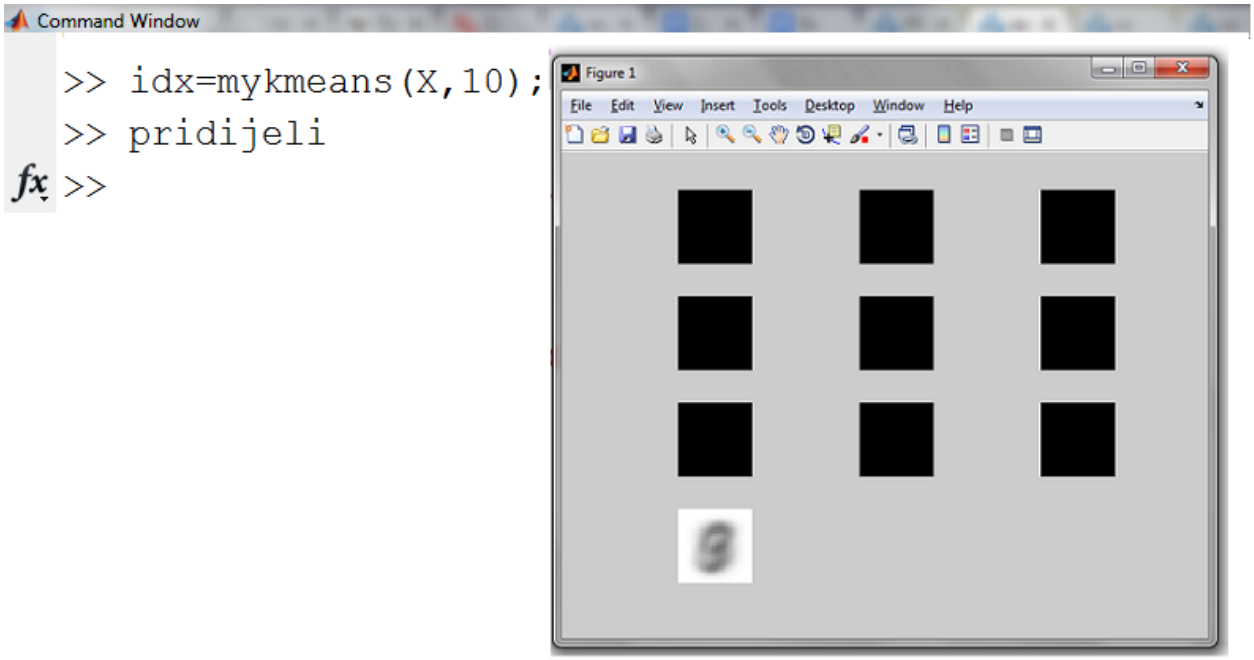
U početku je bilo čudno vidjeti par puta loše rezultate, ali nakon što su se nastavili gomilati, bilo je očito da nešto nije kako treba. Tako, zapravo, i jest jer `mean` (aritmetička sredina iz funkcije `pridijeli` koja služi za prikazivanje ovih "sličica brojeva") nekako "odgovara" zapravo euklidskoj normi, pa je i za očekivati postalo da ako želimo čitljive brojeve u slikama iznad da trebamo koristiti nešto drugo umjesto `mean`-a (trebalo bi nekako minimizirati sumu udaljenosti u ovisnosti o metrici što je trenutno nešto što nije predmet ovog projekta). Također, da se primijetiti i da je npr. Hamming udaljenost[5] loš izbor (čak i postoji kao izbor za takvu udaljenost u regularnom MATLAB-ovom algoritmu za kmeans[6]). To je također za očekivati jer su metrike pomrdane i vjerojatno bi trebalo malo dublje to analizirati[7], a primjer je vidljiv u fileu `udaljenost1.`

[5] Općenito o Hamming distance pročitati na sljedećem linku : http://en.wikipedia.org/wiki/Hamming_distance

[6] Primjeri i dodatne informacije se mogu vidjeti na MATLAB-ovim stranicam (npr. ovoj : http://www.mathworks.com/help/stats/kmeans.html)

[7] Jedan lijep primjer korištenja Hamming distance baš kod MNIST database-a uz dosta matematičke pozadine, minimizacije i aproksimacije greški se može vidjeti na linku : http://www.tecacet.com/articles/SSN.pdf

Evo kako to kaotično izgleda kada se gleda Hamming distance, ali se "crtaju" sličice gdje se gledaju aritmetičke sredine pomoću `mean` u funkciji `proslijedi` :



Naravno, ovo je samo za ovakav slučaj kada gledamo `mean` u `pridijeli` (naravno da se može podesiti drukčije "crtanje" , pa da se dobiju čitljiviji brojevi za što je lijep primjer onaj iz fusnote 6).

Sljedeći zadatak je bio primijeniti knn[8] za klasifikaciju 10k znamenki također sa stranice gdje je MNIST-ova baza s podacima o rukom pisanim znamenkama (naši algoritmi će se zvati `knntest`, `knntest1`, `knntest_inf` u ovisnosti o normi koju izaberemo). Algoritam se može vidjeti ovdje[9], ali je malo modificiran da služi svrsi. Sam algoritam bi trebao promatrati $k \in \mathbb{N}$ najbližih susjeda nekog elementa i većinsku kategoriju tih istih susjeda dodijeliti našem promatranom elementu (što

[8] Više o samome algoritmu, primjenama i slično se može pročitati ovdje :http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[9] http://www.mathworks.com/matlabcentral/fileexchange/29004-feature-points-in-image-keypoint-extraction/content/FPS_in_image/FPS%20in%20image/Help%20Functions/SearchingMatches/kNearestNeighbors.m

je poprilično jednostavno za implementirati, zapravo). Većinska kategorija je ona za koju vrijedi da je broj elemenata koji pripadaju toj kategoriji najveći u odnosu na broj elemenata koji pripadaju bilo kojoj drugoj kategoriji unutar tih promatranih k susjeda. Samu klasifikaciju nekog elementa u kategoriju bi trebalo obaviti tako da uzmemo npr. jedan od testnih 10k elemenata , prođemo po svim testnim primjerima od početnih 60k i pronađemo tom elementu k najbližih susjeda, ali nam je za to potrebna i matrica `a` koju smo koristili u `mykmeans`-u (potrebna je za klasifikaciju čvora jer u `a` pišu zapravo naše "procijenjene" znamenke). Nakon što knn procijeni koja znamenka bi mogla biti u pitanju, usporedit ćemo rezultat sa unaprijed znanim točnim rješenjima (također iz MNIST database-a, a koristili smo i dva filea[10] koja su pomogla pri učitavanju datoteka) i promatrati frekvenciju uspješnosti. U knn-u očito možemo varirati i sami k, a možemo i udaljenosti (pokazat će se da su procjene dosta solidne i za razne druge metrike i pripadne k- ove). Kako smo počeli `mykmeans` s euklidskom udaljenosti, tako ćemo prvo testirati i knn za tu udaljenost i , za početak, $k=2$. Dodatno, knn ćemo testirati za 3 metrike i za svaku od 3 metrike po 3 različita k da vidimo što ćemo dobiti. Mjerit će se i vrijeme izvršavanja (za to nam služi MATLAB-ova naredba `tic; [kod onoga čije vrijeme izvšavanja želimo mjeriti] toc;` Za početak, trebamo učitati primjere nad kojima ćemo testirati knn (`y` predstavlja matricu "znamenki" za koje želimo knn-om procijeniti koje su, dok `labelss` predstavlja točne odgovore iz MNIST baze):

[10] Radi se o loadMNISTImages te loadMNISTLabels, a kako mogu poslužiti se može naučiti ovdje : http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset

```
Command Window
>>  fid = fopen('t10k-images.idx3-ubyte', 'r', 'b');
header = fread(fid, 1, 'int32');
count = fread(fid, 1, 'int32');
h = fread(fid, 1, 'int32');
w = fread(fid, 1, 'int32');
offset=0;
readDigits=10000;
imgs_test = zeros([h w readDigits]);
for i=1:readDigits
for y=1:h
imgs_test(y,:,i) = fread(fid, w, 'uint8');
end
end
y=reshape(imgs_test,28*28,10000);
labelss = loadMNISTLabels('t10k-labels.idx1-ubyte');
```

Potom, za ovako učitane matrice promatramo različite norme (euklidska norma, Manhattan/Taxicab norma, max norma[11]) i različite vrijednosti $k \in \{2, 5, 10\}$. Same implementacije normi se mogu vidjeti u fileovima : `knntest` (za euklidsku normu), `knntest1` (za Manhattan normu), `knntest_inf` (za max normu)

Euklidska udaljenost, k=2 :

[11] Pogledati 91. stranicu ovdje : https://www.math.ucdavis.edu/~hunter/book/ch5.pdf

```
Command Window
>> tic;
suma = 0;
for i=1:10000
if knntest ((y(:,i))',X',a,2)==labelss(i)
suma = suma+1;
end
end
frekvencija_drugi_knn = suma/10000
toc;

frekvencija_drugi_knn =

    0.6075

Elapsed time is 13663.928170 seconds.
fx >>
```

Euklidska udaljenost, k=5 :

```
tic;
suma = 0;
for i=1:10000
if knntest((y(:,i))',X',a,5)==labelss(i)
suma=suma+1;
end
end
frekvencija_knn = suma/10000
toc;

frekvencija_knn =

    0.6222

Elapsed time is 13461.139399 seconds.
fx >>
```

Euklidska udaljenost, k=20 :

```
>> tic;
suma = 0;
for i=1:10000
if knntest((y(:,i))', X', a, 20 ) == labelss (i)
suma = suma + 1;
end
end
frekvencija_treci_knn = suma / 10000
toc;

frekvencija_treci_knn =

    0.6297

Elapsed time is 17283.973499 seconds.
>>
```

Manhattan norma, k=2 :

Command Window

```
>>  tic;
suma=0;
for i=1:10000
if knntest1((y(:,i))',X',a,2)==labelss(i)
suma=suma+1;
end
end
frekvencija_knn_norm1_1 = suma/10000
toc;

frekvencija_knn_norm1_1 =

    0.6050

Elapsed time is 20311.189125 seconds.
>>
```

Manhattan norma, k=5 :

```
Command Window
>> tic;
suma = 0;
for i=1:10000
if knntest1((y(:,i))',X',a,5)==labelss(i)
suma=suma+1;
end
end
frekvencija_knn_norm1_2 = suma/10000
toc;

frekvencija_knn_norm1_2 =

    0.6174

Elapsed time is 21368.189210 seconds.
fx >>
```

Manhattan norma, k=20 :

```
Command Window
>> tic;
suma = 0;
for i=1:10000
if knntest1((y(:,i))', X', a, 20 ) == labelss(i)
suma = suma + 1;
end
end
frekvencija_knn_norm1_3 = suma / 10000
toc;

frekvencija_knn_norm1_3 =

    0.6226

Elapsed time is 20567.872657 seconds.
fx >>
```

Max norma, k=2 :

```
>> tic;
suma = 0;
for i=1:10000
if knntest_inf((y(:,i))', X', a, 2 ) == labelss (i)
suma = suma + 1;
end
end
frekvencija_knn_inf3 = suma / 10000
toc;

frekvencija_knn_inf3 =

    0.5555

Elapsed time is 33708.103803 seconds.
>>
```

Max norma, k=5 :

```
>> tic;
suma=0;
for i=1:10000
if knntest_inf((y(:,i))',X',a,5)==labelss(i)
suma=suma+1;
end
end
frekvencija_knn_inf1 = suma/10000
toc;

frekvencija_knn_inf1 =

    0.5771

Elapsed time is 20406.259255 seconds.
>>
```

Max norma, k=20 :

```
>> tic;
suma = 0;
for i=1:10000
if knntest_inf((y(:,i))', X', a, 20 ) == labelss (i)
suma = suma + 1;
end
end
frekvencija_knn_inf2 = suma / 10000
toc;

frekvencija_knn_inf2 =

    0.5710

Elapsed time is 33700.587571 seconds.
fx >>
```

KRATKA STATISTIKA :

Podaci su pomalo i zanimljivi. Iako je za ovih 10-ak sličica utrošeno oko 194470s ili neka 54 sata (moglo se možda ubrzati da sve udaljenosti spremim u neku ogromnu matricu radi dobrog MATLAB-ovog baratanja vektorima i matricama, pa da onda preko nje pretražujem, no dobro), dobili smo , u prosjeku, da je euklidska udaljenost dala rezultate od visokih 61.98%, Manhattan norma prosjek od 61.5%, a max norma nešto niže, ali svejedno respektabilnih 56.79% (u totalu je testiranje dalo visokih 60.09% što se može smatrati donekle uspješnom analizom na ovakav način). Dakle, euklidska norma je polučila najbolje rezultate, ali je odstupanje rezultata od prosjeka jednako ±3.29% što govori da je rezultat prilično pouzdan. Valjalo bi spomenuti, kao i u svakoj iole ozbiljnijoj statistici, eventualne "rupe" u zaključivanju koliko smo, zapravo, dobili *dobre* rezultate. Jedino što se kritički i može gledati su rezultati jer su algoritmi egzaktni i tu se nema što posebno pričati. Bitno je naglasiti kako algoritmi inače mogu postići puno bolje rezultate [12], ali trebalo bi dosta više samog matematičkog znanja i aparata.

[12] Čist primjer je ovo : http://www.mathworks.com/help/stats/kmeans.html

Maknuti "šumove" iz velike matrice X i koristiti SVD[13] dekompoziciju su jedni od načina koji uspješnost mogu dogurati i do visokih 90%. To bi značilo da iz matrice X sa svim silnim podacima uzmemo samo one najbitnije (pokušava se smanjiti rang matrice npr.) i nad njima radimo k-means ili knn algoritam (tako da je prosjek od nekih 60% u odnosu na ovih 90% prilično i "mršav").

IZVORI :

- http://www.mathworks.com/help/stats/kmeans.html
- https://www.math.ucdavis.edu/~hunter/book/ch5.pdf
- http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset
- http://en.wikipedia.org/wiki/Hamming_distance
- http://www.mathworks.com/matlabcentral/fileexchange/29004-feature-points-in-image-keypoint-extraction/content/FPS_in_image/FPS%20in%20image/Help%20Functions/SearchingMatches/kNearestNeighbors.m
- http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- http://www.tecacet.com/articles/SSN.pdf
- http://www.mathworks.com/help/stats/kmeans.html
- http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf
- http://en.wikipedia.org/wiki/Norm_(mathematics)#Taxicab_norm_or_Manhattan_norm
- http://yann.lecun.com/exdb/mnist/
- http://people.revoledu.com/kardi/
- http://jeremykun.com/2012/08/26/k-nearest-neighbors-and-handwritten-digit-classification/
- http://courses.cs.tamu.edu/rgutier/cs790_w02/l8.pdf
- http://www.mathworks.com/help/pdf_doc/matlab/matlab_prog.pdf
- http://www.mathworks.com/help/matlab/

[13] Pročitati : http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf