MESTERSÉGES INTELLIGENCIA

HÁZI FELADAT

Juhász-Nagy János

Vuchetich Bálint

Konzulens: Hadházi Dániel

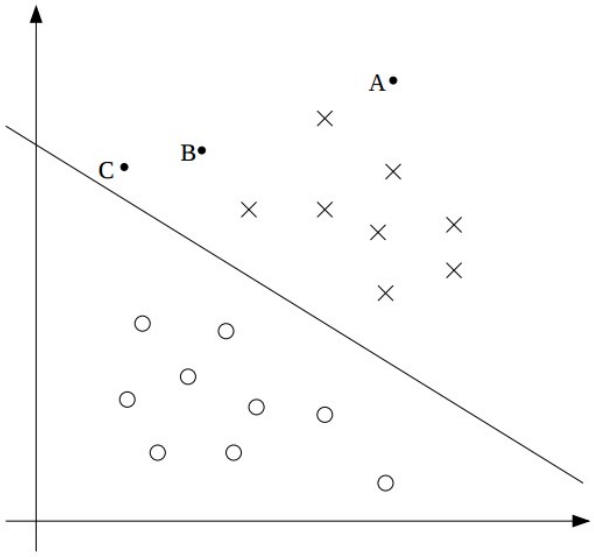
Tartalomjegyzék

Szakirodalom áttekintés

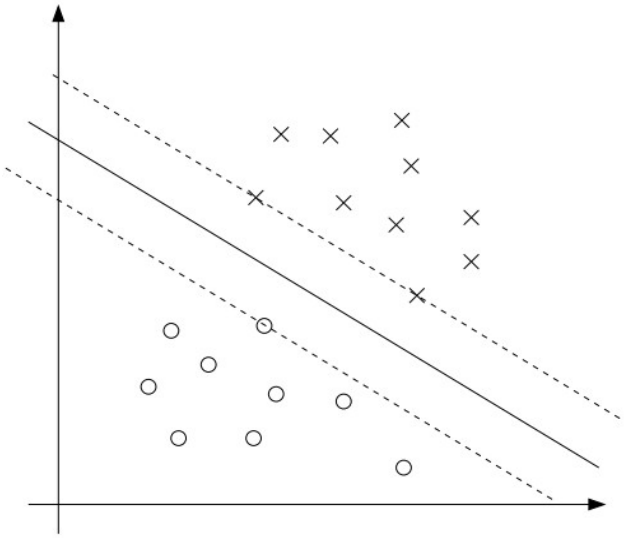
A szakirodalomban az SVM témában írt jegyzetek rendszerint szabadkozással kezdődnek, miszerint a szerző sajnálja a kedves olvasót, de a pontos matematikai háttér ismertetése nékül nem tudja tisztességesen elmagyarázni a módszert – és kezdenek bele a többtízoldalas elméleti összefoglalókba, aminek a végére eljutnak az alapötletekig.
Mivel ennek a házi feladat dokumentációnak nem célja, hogy ilyen mélységben áttekintse ezt a témakört, ezért jobb híján igyekszünk csak a megértéshez elengedhetetlenül fontos fogalmakat és összefüggéseket ismertetni, az egyszerűség és a tömörség miatt néha megengedve a felületesnek és pongyolának ható megfogalmazásokat.

Support Vector Machines alapok

Az alapprobléma a következő: két halmazt próblunk elválasztani egymástól lehetőleg minél jobban.



Látjuk, az egyenes szeparálja a két halmazt, ám úgy érezzük az A-ra megbízhatóbb besorolát tud adni, mint C-re, hiszen ez utóbbi kicsiny megváltozása esetén már átkerülhetne a másik osztályba. Így az egyik célunk, hogy olyan egyenest (ami magasabb dimenziókban hipersík) találjunk, ami a lehető legtávolabb van az egyes pontosztályoktól, azaz a szétválasztó biztonsági sáv minél szélesebb legyen.



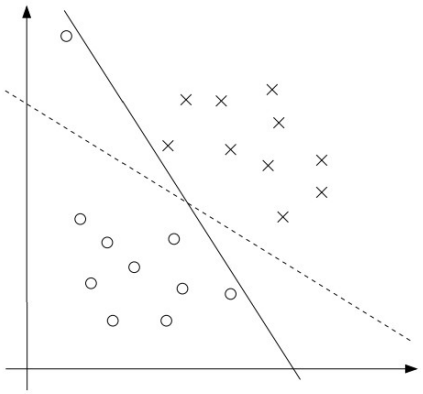
Ezt a problémát lehet formalizálni, amire a kvadratikus programozás módszereivel kereshetünk megoldásokat.
Hosszas és bonyolult átalakítások után a távolságfüggvény a határoló hipersíktól a következő alakban is felírható:

$$\omega^T x+b=\sum_{i=1}^{m}\alpha_i y^{(i)}\langle x^{(i)}|x\rangle+b$$

Ebből annyit érdemes megjegyezni és megérteni, hogy a jobb oldalon csak a bemenetek belső szorzatai állnak, és maguk a mennyiségek nem, ez még később hasznos lesz.

Soft margin SVM

Gyakran kerülhetnek bele zajos adatok a forrásainkba, amiket szeretnénk kiszűrni, mert torzítja a tanító algorimusunk futását. Például:



Látható, ha a bal felső pont nem lenne ott, akkor egy szélesebb szétválasztó biztonsági sávot kapnánk, ami globálisan biztosabb besorolást biztosítana, ezért határozzuk meg úgy a hipersíkot, hogy megengedjük, néhány outlinernek tekinthető pont belelógjon a sávba, vagy akár átlógjon a másik oldalra, de cserébe szélesebb legyen a sáv. Ezt gyakorlatban úgy lehet megoldani, a minimalizálandó célfüggvényhez hozzáveszünk egy ξ lazító változót, és ennek egy C konstans szorzójával lehet beálllítani mennyire szigorúan büntetjük az átlógást.

Vegyük észre, ezzel a kiegészítéssel sikerült (legalább valamilyen) megoldást adnunk akár lineárisan szigorúan nem szeparálható esetekre is.

Kernel függvények

Tegyük fel, egy egyszerű regressziós feladatnál adott egy x bemeneti változó, és ahhoz hogy kiszámítsuk az eredmény (ami mondjuk egy harmadfokú függvény), szükségünk van x-re, x^2-re és x^3-re is. Hogy megkülönböztessük ezeket, az x-et nevezzük **attribútum**nak, a x, x^2, x^3 tagokat pedig **feature**-öknek, és az ezeket előállító $\varphi()$ függvényt pedig **feature map**-nek. Ez esetben:

$$\varphi(x)=\begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

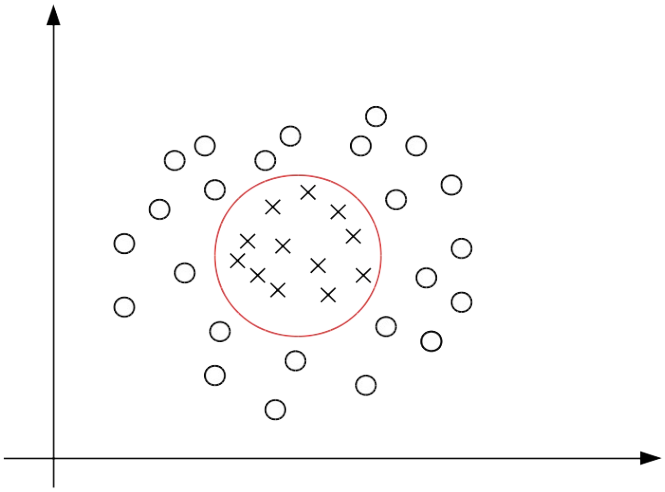
Ezt követően a tanító algoritmusunkat az eredeti attribútumok helyett alkalmazzuk a feature-ökre, ami annyit jelent, hogy a fenti képletben az x-eket $\varphi(x)$-re kell cserélni. Mint említettük a képletben csak $\langle x|z\rangle$ belső szorzatok szerepelnek, így azokból $\langle\varphi(x)|\varphi(z)\rangle$ lesz, ami a következő speciális alakot ölti: $K(x,z)=\varphi(x)^T\varphi(z)$.

Ezt a függvényt nevezik **kernel függvény**nek. Végeredményben tehát mindenhol lecseréltük az $\langle x|y\rangle$ attribútumok belső szorzatát a $K(x,z)$ kernel függvényre.

Adott $\varphi()$ feature map-re könnyedén megkaphatjuk $K(x,z)$-t úgy, hogy kiszámoljuk $\varphi(x)$-et és $\varphi(z)$-t, majd belső szorozzuk őket. Ám ami igazán érdekes, hogy $K(x,z)$-t kiszámolni gyakran nagyon gyorsan lehet, még akkor is, ha $\varphi()$-t csak lassan. Ily módon például magasabb dimenziókban is hatékonyan tudunk SVM-eket számolni anélkül, hogy expilcit módon $\varphi(x)$-et bármikor is ki kéne számolnunk. Ennek alkalmazása az úgynevezett **kernel trükk**.

Nemlineáris szeparálás

És mi a teendő akkor, ha a feladat még csak nagyjából sem lineárisan szétválasztható?



A csel a következő: a bemeneti attribútumokra alkalmazunk egy $\varphi()$ nemlineáris leképzést, ami egy olyan térbe képezi le, ahol már lineárisan szeparálható lesz a feladat, és itt keressük a lehető legszélesebb margót.

Nahát! Nem véletlen, hogy ezt a függvényt is $\varphi()$-nek neveztük, hiszen éppen az előző részben leírt kernel függvény előnyös tulajdonságait szeretnénk kihasználni. Tehát alkalmazva a kernel trükköt a lépések ezek lesznek:

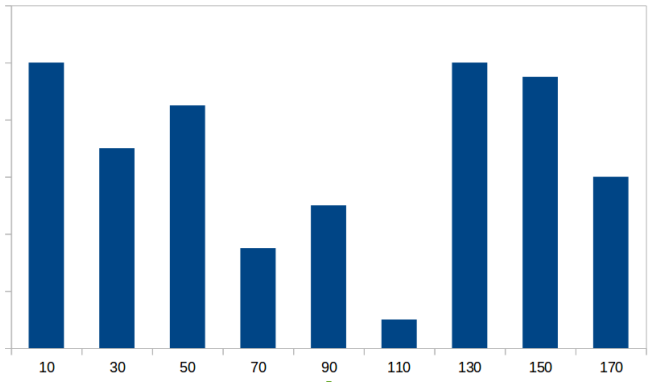
1. lépés: meghatározzuk a nemlineáris függvényt, ami linearizálja a feladatunkat, ebből kiszámoljuk a kernel függvényünket;
2. lépés: a kernel térben alkalmazzuk a tanító algoritmusunkat, ehhez a következőket érdemes megjegyezni: egyrészt a kernel teret a $\varphi()$ kiszámítása nélkül is megkaphatjuk, másrészt a kernel tér véges dimenziójú, így hatékonyabban tudunk számolni, mint a feature térben;
3. lépés: a kernel függvényből és a support vektorokból bármilyen bemenetet le tudunk képezni a kernel térbe, így tudjuk alkalmazni a betanított algoritmust.

Histogram of oriented gradients (HOG)

Ez a módszer a képek globális jellemzésére szolgál – bár nyilvánvaló a kapcsolat az SVM-hez – a tömörség miatt önmagában, csupán az algoritmust ismertetjük.

- Tegyük fel, adott egy 64*128 pixeles kép.
- Osszuk fel 16*16-os blokkokra 50%-os átfedéssel, azaz 7*15= 105 blokkunk lesz.
- Minden blokk tartalmazzon 2*2 cellát, amik 8*8 pixelesek.
- Számítsuk ki a grandienseket minden cellára, és osszuk őket 9 csatornába:
  - minden pixelre kiszámítjuk az X és Y irányú paricális deriváltakat: s_x és s_y ;
  - ezekből megkapjuk a gradiens nagyságát: $s = \sqrt{(s_x^2 + s_y^2)}$

    és irányát: $\theta = \arctan\left(\frac{s_y}{s_x}\right)$;
  - ezeket irányuk szerint 9 csatornába osztjuk (0°-20°, 20°-40°, …, 160°-180°), minden pixel szavazatának erősségét a gradiens nagysága adja meg.
    Például ilyen hisztogramot kapunk:



- Fűzzük össze az így kapott hisztogramokat, így 105 blokk * 4 cella * 9 csatorna = 3'780 feature-t kapunk.

Képmanipulációs eljárások

Gaussian Blur

Ez az eljárást a képeken lévő zaj kiszűrésére lehet használni; minden pixelt önmaga és a környezetének súlyozott átlagára cserélünk (minden színcsatornában).

A kétdimenziós izotróp Gauss-eloszlás képlete a következő:

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

Egy $\sigma=1.0$ -ás szórással és 5*5-ös kerettel ezek a súlyok adódnak:

$$\frac{1}{273}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}$$

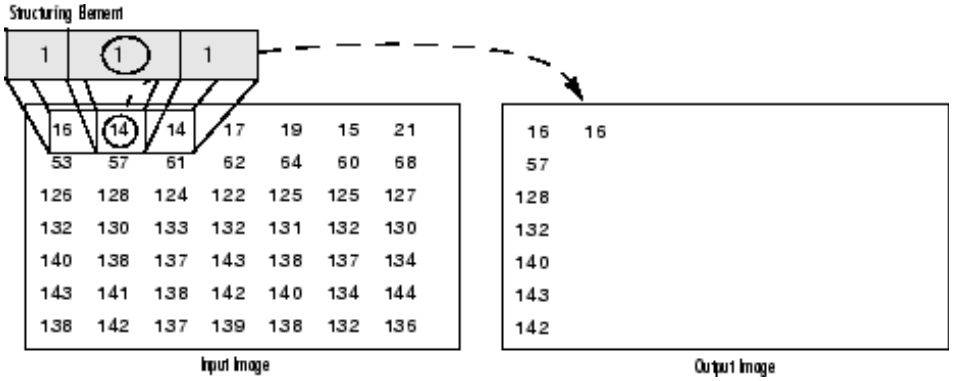
Példa:

Vastagítás (dilate)

A zajszűrő és a maratás (erosion) alkalmazása után általában visszavastagításra van szükség, mert bár a fehér zajokat kiszűrtük, kicsit össze is zsugorodott az alakzatunk.

A vastagítást a következőképpen érjük el. Minden pixelhez definiáljuk a környezetét (ezt a CV2 függvényben a kernel paraméterrel tudjuk állítani), és utána lecseréljük a környezetében lévő

maximális értékre.



Példa:

eredeti

vastagított

Az alkalmazott algoritmus ismertetése

Szeretnénk előre leszögezni, hogy a kód nagyrésze Bikramjot Singh Hanzra egyik blogbejegyzéséből származik (lásd [5]), ám ő egy egyszerűbb előre beépített tanító adathalmazzal (MNIST database of handwritten digits) doglozott, így a mi adathalmazunk előfeldogozása a saját átdolgozásunk.

Két fő lépésből áll: az első, a tanító adathalmazból előállítjuk a betanított algoritmus szükséges adatait (ez egy fájlt jelent), a második pedig, hogy alkalmazzuk az algoritmust, és felismertetünk vele karaktereket.

A feladatot Python nyelven oldottuk meg a NumPy kiegészítő csomag használatával. Az egyszerűség kedvéért tekintsük a kódot sorról sorra, és a magyarázatokat ott adjuk meg. A teljes kódok megtalálhatóak a GitHubon:
https://github.com/vuchetichbalint/machine_learning/tree/master/digitRecognition

1. lépés: a tanítás (generateClassifier.py)

Beimportáljuk a szükséges könyvtárakat.

```
#!/usr/bin/env python
import cv2
from sklearn.externals import joblib
from skimage.feature import hog
from sklearn.svm import LinearSVC
import numpy as np
from collections import Counter
from os import listdir
from os.path import isfile, join
```

Megadjuk neki a tanítóhalmaz elérhetőségét, és a fájlneveket eltároljuk a paths válozóba.

```
mypath = '/home/balint/workspace/svm/digitRecognition/data'
paths = [f for f in listdir(mypath) if isfile(join(mypath, f))]
```

Inicializáljuk a változókat, amikbe a labeleket és a feature-öket fogjuk eltárolni.

```
list_hog_fd = []
labels = []
i = 1
no_images = len(paths)
```

Ezek után minden képre végrehajtuk a következő műveleteket: egyrészt beolvassuk az im változóba, másrészt a fájlnév alapján meghatozzuk a labeljét a képnek. (Például img004-xxx.png egy hármast ábrázol, a img005-xxx.png egy négyest stb...) Az im változó egy szélesség*magasság*3 hosszúságú tömb, minden egyes pixelre megadja az BGR értékeket.

```
for path in paths:
      im = cv2.imread(mypath + '/' + path)
      label = int(path[-10:-8])-1
```

A képet átalakítjuk színesről szürkeárnyalatossá.

```
      im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
```

Majd alkalmazunk rá egy 5*5-ös Gauss-simítást, hogy kiszűrjük az esetleges zajokat.

```
      im_gray = cv2.GaussianBlur(im_gray, (5, 5), 0)
```

A threshold() függvény egy fekete-fehér képet ad vissza a bináris inverz paraméterrel használva.

```
      ret, im_th = cv2.threshold(im_gray, 90, 255, cv2.THRESH_BINARY_INV)
```

A findContours() függvény két struktúrát ad vissza: az egyik a megtalált alakzat befoglaló szokszögének sarkainak koordinátái, a másik pedig az alakzatok egymáshoz képest elfoglalt pozíciója (a kép hierarchiája) – ez utóbbi nekünk nem fog kelleni.

```
      ctrs, hier = cv2.findContours(im_th.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

A befoglaló sokszögből befoglaló téglalapot csinálunk.

```
      rects = [cv2.boundingRect(ctr) for ctr in ctrs]
```

Élünk a feltételezéssel, hogy a tanító halmazban egy képen egy számjegy van, de azért a biztonság kedvéért a 4*4 pixelnél kisebb alakzatokat eldobjuk.

```
      for rect in rects:
            if ( rect[2] > 4 and rect[3] > 4  ):
                  break;
```

Az uniformizálás miatt a kicsit kitágítjuk a befoglaló téglalapot, így megkapjuk a ROI-t, azaz a »region of interest«-et.

```
      leng = int(rect[3] * 1.05)
            pt1 = int(rect[1] + rect[3] // 2 - leng // 2)
            pt2 = int(rect[0] + rect[2] // 2 - leng // 2)
            roi = im_th[pt1:pt1+leng, pt2:pt2+leng]
```

Átméretezzük 28*28-asra a téglapunkat.

```
      roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)
```

A számjegyeket megvastagítjuk.

```
      roi = cv2.dilate(roi, (3, 3))
```

Kiszámoljuk a HOG gradienseket 9 színcsatornával 14*14-es cellákkal cellánként 1-1 blokkal.

```
      fd = hog(roi, orientations=9, pixels_per_cell=(14, 14),
cells_per_block=(1, 1), visualise=False)
```

Eltároljuk a megkapott feature-öket az eddigiekhez.

```
      list_hog_fd.append(fd)
      labels.append(label)
```

Miután a fenti műveletekkel végeztünk, elkezdjuk a tanítást, ehhez először átalakítjuk az adatstruktúránkat NumPy formátumra.

```
hog_features = np.array(list_hog_fd, 'float64')
labels = np.array(labels, 'int')
```

Példányosítunk egy lineáris SVM osztályozót, és betanítjuk.

```
clf = LinearSVC()
clf.fit(hog_features, labels)
```

Majd a végén kimentjük a kapott fájlt.

```
joblib.dump(clf, "digits_cls.pkl", compress=3)
```

2. lépés: az alkalmazás (performRecognition.py)

Az alkalmazás során szintén a tanításnál leírt előfeldogozást használjuk, így azt nem ismeretnénk újra, és csak az érdekesebb kódrészeket mutatnánk be.

Így olvassuk be az osztályozót.

```
clf = joblib.load("digits_cls.pkl")
```

És így alkalmazzuk. A clf.predict() egy labellel tér vissza.

```
      nbr = clf.predict(np.array([roi_hog_fd], 'float64'))
```

A megkapott számot kiiratjuk a képre.

```
      cv2.putText(im, str(int(nbr[0])), (rect[0],
rect[1]),cv2.FONT_HERSHEY_DUPLEX, 2, (255, 20, 20), 3)
```

Kiértékelés és validáció

Az adathalmazunkban minden számjegyről 55 képünk van, 50-nel tanítottunk, a kihagyott 5-tel pedig ellenőriztünk.

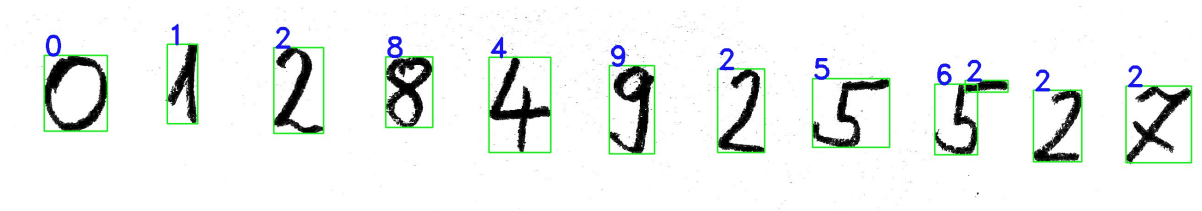
A legjobb eredményt 7*7 pixels per cell és 2*2 cells per block beállítással kaptuk, ennek confusion mátrixsza az alábbi.

		predicted									
		0	1	2	3	4	5	6	7	8	9
actual	0	5									
	1		4						1		
	2			4	1						
	3				5						
	4					4					1
	5						5				
	6							5			
	7					1			4		
	8									5	
	9										5

Ezzel szemben 14*14 pixels per cell és 1*1 cells per blockkal sokkal kevésbé megbízható besorolást nyertünk.

		predicted									
		0	1	2	3	4	5	6	7	8	9
actual	0	5									
	1		1						1		3
	2			4	1						
	3				5						
	4		1			3					1
	5						5				
	6							5			
	7								4		1
	8									5	
	9				1				1		3

Kipróbáltuk egy saját kezűleg írt képre, mennyire működik, látható, a többségét felismerte, viszont egyáltalán nem meglepő módon, a hetesünket elvétette – mivel a tanító halmazban lévő heteseknek nem hullámos a teteje.



Kitekintés

Az algoritmusnak nyilvánvalóan vannak hiányosságai, ennek egy része az implementációból ered, a másik pedig magából a módszerből.

A megvalósításból fakadóak közül kettőt emelnénk ki: egyrészt ha a kép szélén van az alakzat, akkor a befoglaló téglalap megnagyításakor a képen kívülre eshet a téglalap egyik sarka, ilyenkor a program egyszerűen lefagy; másrészt a findContours() függvény hívásakor tudnánk használni a hierarchiát, amivel megakadályozhatnánk, hogy egy alakzatot szétvágjon önkényesen (például a saját kézírás kilencedik karaktere).

Az alkalmazott előfeldolgozó eljárásokon is lehetne még sokat finomítani, pl. hatékonyabb zajszűrők, nem rögzített threshold-érték stb..., illetve a legegyszerűbb lineáris SVC helyett használhatnánk bonyolultabbakat (ez persze nem biztos, hogy javulást eredményezne).

Hivatkozások

[1] Stanford University CS 229 Machine Learning 2015. őszi kurzusának jegyzete (Andrew Ng) http://cs229.stanford.edu/notes/cs229-notes3.pdf
[2] „Histograms of Oriented Gradients (HOG)” youtube videó (Dr. Mubarak Shah) https://youtu.be/0Zib1YEE4LU
[3] Histograms of Oriented Gradients for Human Detection (Navneet Dalal, Bill Triggs) http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf
[4] Neurális hálózatok (Horváth G., Altrichter M., Horváth G., Pataki B., Strausz Gy., Takács G., Valyon J.,) Budapest, Panem Kiadó, 2006. – 6.3.-as fejezete
[5] Digit Recognition using OpenCV, sklearn and Python (Bikramjot Singh Hanzra) http://hanzratech.in/2015/02/24/handwritten-digit-recognition-using-opencv-sklearn-and-python.html
[6] Gauss-simítás: http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html#gaussian-filtering
[7] Vastagítás: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html
[8] Vastagítás: http://www.mathworks.com/help/images/morphology-fundamentals-dilation-and-erosion.html
[9] Használt adathalmaz: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/