Reconocimiento de letras escritas por computadora

Rodriguez Mendoza Angel Rafael [1], Sánchez Max A. [2]

——————————  ——————————

————————————————

* [1] E-mail: angel\_r\_m\_zool@hotmail.com
* [2] E-mail: sanchezmaxar@gmail.com

El reconocimiento de letras escritas es un gran avance para la rama de la ciencia de la computación de inteligencia artificial y el aprendizaje máquina. El reconocimiento óptico de de caracteres (ROC) es un proceso de procesamiento de imágenes para la digitalización de archivos escritos en texto que la máquina, en los últimos años la digitalización de textos ha sido ampliamente usada para evitar la digitalización por parte de recursos humanos, esto mejora la calidad de ciertos servicios y la productividad.

Los principales problemas de la digitalización son:

- La introducción de niveles de gris durante la digitalización de la imagen (ruido)

- La distancia que separa a los caracteres no es la misma así que es posible que genere errores.

- La traslapación de caracteres también producirá errores durante el reconocimiento.

El ROC se basa en 4 etapas

1.- Binarización

2.- Fragmentación o segmentación de la imagen

3.- Adelgazamiento de los componentes

4.- Comparación de patrones

Entre sus aplicaciones se encuentran

- Reconocimiento de texto manuscrito

- Reconocimiento de matrículas

-Digitalización de documentos introducidos por el usuario

La historia de ROC empieza en 1914 con Emanuel Goldberg quien desarrollo una máquina que convierte el texto introducido en código telegráfico, al mismo tiempo Edmund Fournier d’Albe desarrolla el optofono un dispositivo desarrollado con el fin de producir determinados sonidos cuando se detecte cierto carácter, años más tarde Emanuel Goldberg desarrolla otra máquina estadistica con el fin de buscar archivos de microfilme usando reconocimiento óptico.

# 2 Objetivos

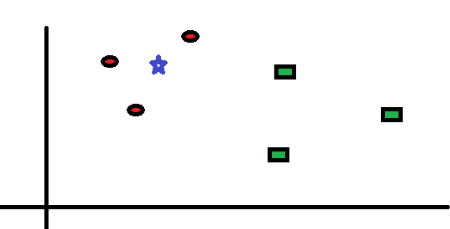
Crear un programa que reconozca las letras escritas a computadora con la opción de reconocer letras escritas a mano.

# 3 Desarrollo

El primer acercamiento al problema fue el uso de knn

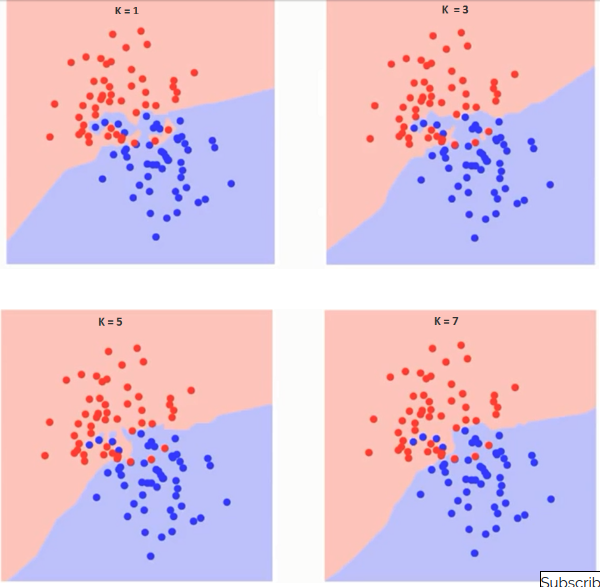
**3.1 Uso de KNN**

El algoritmo de knn consiste en el uso de un ‘plano’ en el que las características de una clase se ubican, entonces se considera que se le dará a la muestra actual de entrada al clasificador la clase de los k vecinos más cercanos, esto se refiere a que se comenzará a buscar en el espacio de muestras de entrenamiento las k muestras que esten más cerca. Observando la siguiente figura, se ilustra mejor la disposición en un espacio de lo que es el algoritmo knn:

Fig 1 Extraida de [1]

En la figura de arriba se simboliza la clase 1 con ovalos y la clase 2 con rectángulos, y la estrella simboliza la muestra de prueba, se puede ver que considerando una k de cualquier tamaño, la clasificación que se le de a la muestra va a ser siempre de la clase 1, ya que para toda k de 1...3 las clases de los vecinos más cernanos son de la clase 1.

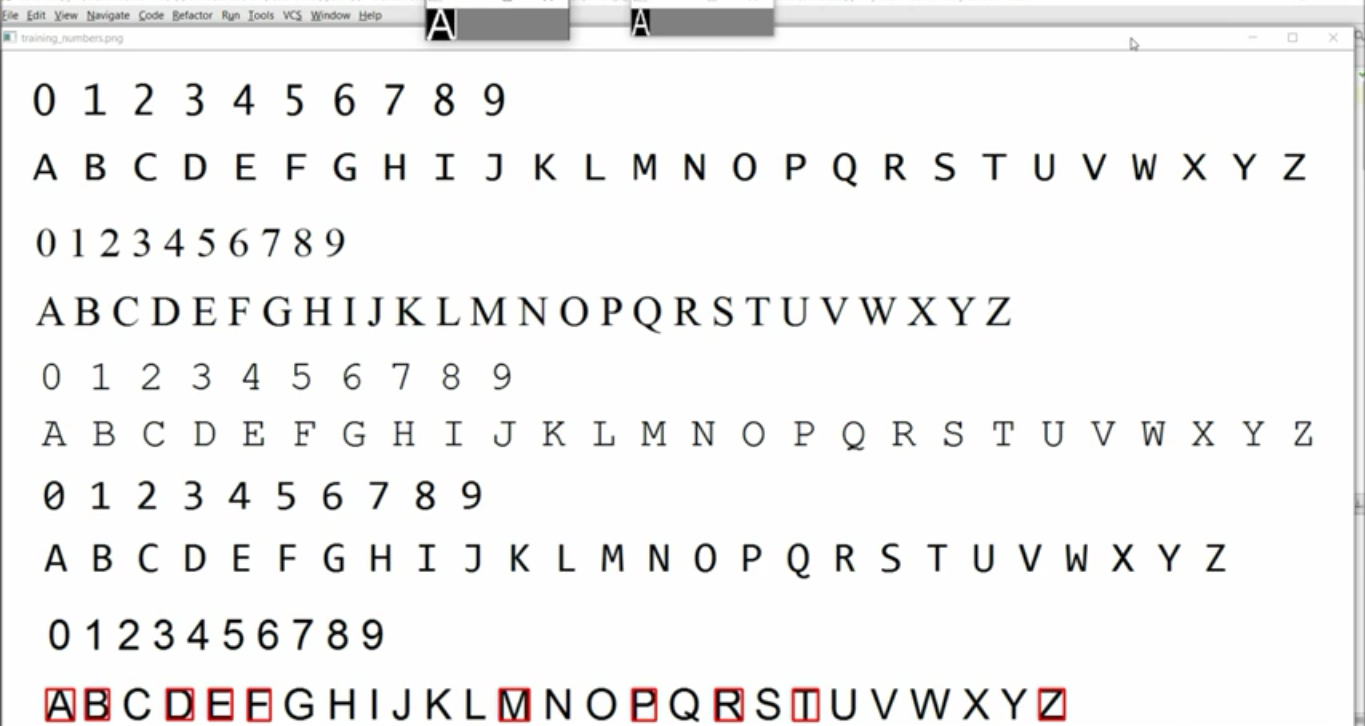
Entonces aquí entra la pregunta: ¿Qué valor de k es mejor para mi clasificador?, bueno según la experiencia esto lo determinará las pruebas que se hagan, tal vez para un caso específico es mejor con k=1 y en otro con k=10, en la figura siguiente se puede ver una comparativa entre un k de 1,3,5 y 7:

Fig 2 Extraida de [1]

El area roja es la clase que se predecirá para una muestra dada y la azul para otra muestra, como venimos viendo se clasificará según los k vecinos más cernanos, aquí se puede ver como al principio el area roja y azul compartian una parte equitativa del espacio y conforme va creciendo el area de la clase azul va disminuyendo. Cabe recalcar que el uso de la k para un problema dado se sujeta a la particularidad de las muestras.

Otro punto interesante de knn es que se puede adaptar a más esquemás de clasificación, por ejemplo se puede ponderar la métrica en lo que se podría considerar una característica muy importante, también se puede crear un árbol de decisión dependiendo de la zona en la que se encuentre, esto usando diferentes formas de crearlo, por ejemplo si los datos se agrupan más cerca de un cierto punto, darle una k de un valor más pequeño o si estan cerca de una frontera se puede considerar una k de mayor tamaño, esto se llama edit knn. Una de las ventajas de usar knn es la complejidad de su algoritmo que puede varias de O(log(n)) hasta O(n) dependiendo de la implementación, sin embargo una de sus desventajas es que si no se tiene un set de entrenamiento suficientemente grande, el clasificador no hará bien su trabajo.

En un principio al usar KNN, nos dimos cuenta que nuestro clasificador dependía en gran medida a que tan precisos éramos seleccionando el área que le pertenecía a cada carácter. Decidimos pues, intentar automatizar el trabajo usando uso de un tipo de reconocimiento por modelos. Con la ayuda del modulo de OpenCV [5] de Python, hicimos que las imágenes tuvieran bordes más gruesos y definidos y así creamos un modelo que permite reconocer áreas automáticamente alrededor de cada carácter.

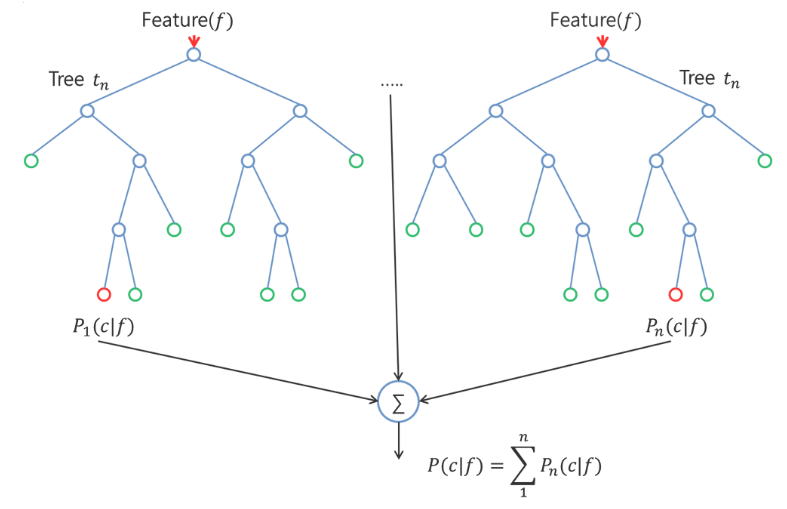


Este preprocesamiento fue muy útil porque nos ahorra mucho tiempo a la hora de generar el set de pruebas ya que ahora solo tenemos que presionar la tecla del carácter que se delinea. Así generamos nuestro conjunto de imágenes y las etiquetas respectivas a cada imagen.

**3.2 Random Forest**

Otro acercamiento que se tuvo fue el uso de random forest como forma de usar aprendizaje. Decidimos usar esta técnica porque encontramos que nuestro clasificador KNN no era muy bueno reconociendo letras manuscritas debido a las limitaciones que tenemos con el set pruebas.

Random Forest consiste en un algoritmo con el que se crearán árboles de decisión en base a un subset de entrenamiento escogido aleatoriamente. Veasé la figura 3

Fig 3 Extraida de [3]

Al crear estos árboles se debe generar un índice que separe las características, para esto se pueden usar diferentes métricas, una de las más usadas es el Coeficiente de Gini el cual es una medida de desigualdad, esto nos dice que valor marca la división entre una clase y otra, en resumidas palabras.

A partir de los árboles de decision creados se somete a votación para decidir que clase se tomará y se clasifica como la que tenga más votos.

Algunos de los parametros que impactan en el desempeño del random forest son:

El número de características que se tomarán no es óptimo tomar todos las características ya que esto pegaría en el tiempo de procesamiento, ya que no manejaría tan bien la multi dimensionalidad, sin embargo entre más se pueda tomar mejor.

El número de árboles que se generarán, teniendo más árboles se tomará una decisión mejor, pero al igual que el número de características esto pega en desempeño.

El número mínimo de hojas, entre más hojas mejor ya que esto hará que el árbol ignore casos de ruido.

# 4 Resultados

# 4.1 KNN

Los resultados fueron lo que nos esperábamos en un principio. Como el set de entrenamiento que usamos fueron caracteres generados por computadora con diferentes fuentes, esperábamos que pudiera reconocer sin problemas otros caracteres generados por computadora.

También probamos variar la K con nuestras imágenes de prueba. Resulta que no hay mucha diferencia entre K igual a 1, 2, 3 y 4. Con K mayor a 4, el clasificador comienza a ser más impreciso.

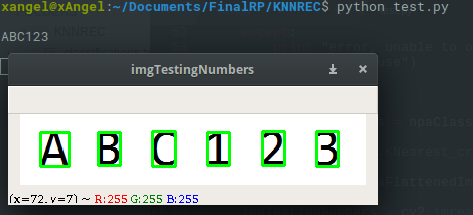


Imagen de prueba test1.png encontrada en [6]

Los rectángulos verdes significan que el programa reconoció que en esa área hay un carácter, y en la consola aparecen los caracteres que el clasificador cree que son.

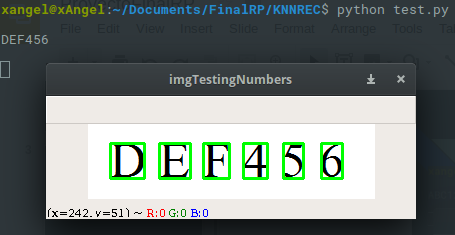


Imagen de prueba test2.png encontrada en [6]

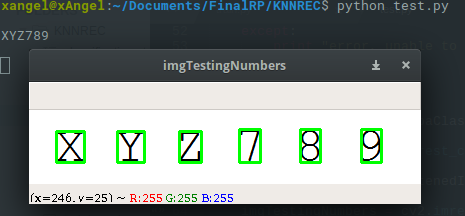


Imagen de prueba test3.png encontrada en [6]

El problema surge cuando intentamos ingresar letras manuscritas. En primer lugar, el programa falla en reconocer todas las áreas en las que hay un carácter. En segundo lugar, cuando si es capaz de reconocer un área, el clasificador es muy impreciso.

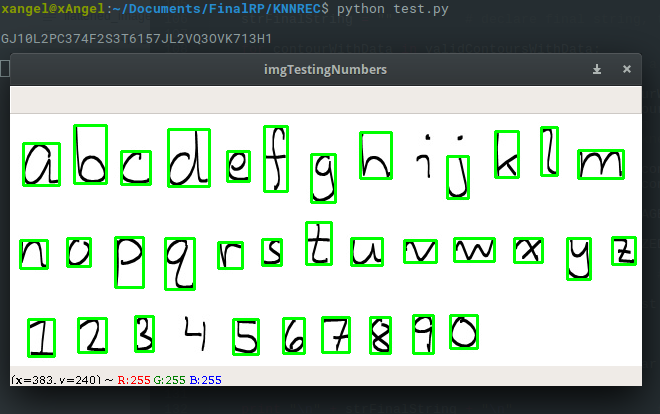


Imagen de prueba manuscrita.png encontrada en [6]

Para poder resolver estos problemas, nosotros deberíamos tener un set de entrenamiento muy grande que costaría mucho tiempo poder reunir.

**4.2 Random Forest**

Esta solución es un poco más rudimentaria. Solo tomamos un set de imágenes con letra manuscrita que ya está dentro del módulo de sklearn[5] en Python. Podemos ver que el método es más efectico, pero sigue estando limitado al conjunto de prueba que tenemos a nuestra disposición.

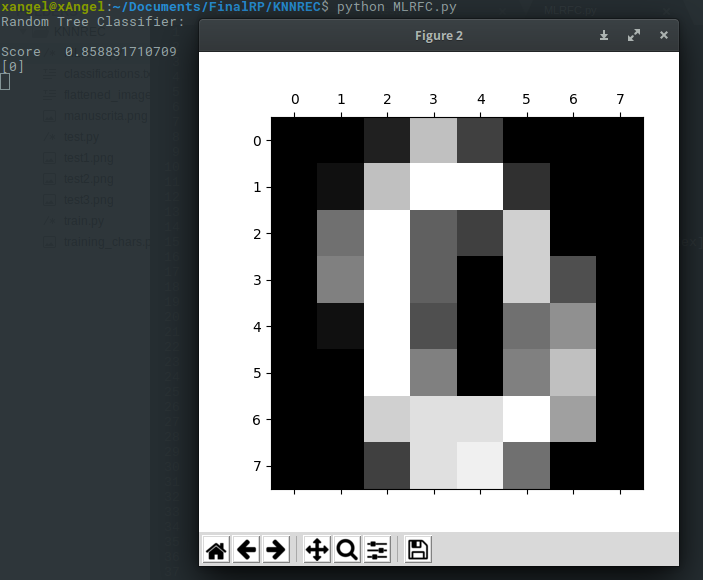


Imagen de prueba encontrada en el módulo sklearn[5]

La imagen que se muestra ha pasado por un preprocesamiento en donde se invierten los colores y se le aplica un filtro gausiano para que sea más homogénea. El dígito que esta entre los corchetes es el carácter que el clasificador decidió que es.

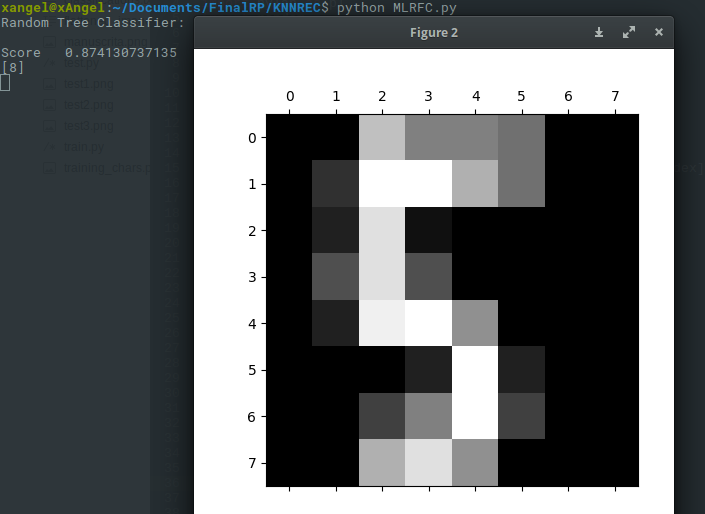


Imagen de prueba encontrada en el módulo sklearn[5]

Sin embargo, podemos ver que el sistema no es lo suficientemente preciso, la imagen corresponde a un 5 y el clasificador la reconocer como un 8.

# 5 Implementación

**5.1 Implementación de KNN[6]**.

**## Archivo train.py**

import sys

import numpy as np

import cv2

import os

# modula el nivel de las variables ##########################################################################

MIN\_CONTOUR\_AREA = 100

RESIZED\_IMAGE\_WIDTH = 20

RESIZED\_IMAGE\_HEIGHT = 30

###################################################################################################

def main():

imgTrainingNumbers = cv2.imread("training\_chars.png") # lee la imagen con los caracteres de prueba

if imgTrainingNumbers is None: # si el archivo no se lee correctamente

# manda un mensaje de error en la salida estandar

raw\_input("Error la imagen puede ser leida del archivo") # pausa para que el usuario vea el mensaje

return # sale del programa

# end if

imgGray = cv2.cvtColor(imgTrainingNumbers, cv2.COLOR\_BGR2GRAY) # obtiene la imagen en escala de grises

imgBlurred = cv2.GaussianBlur(imgGray, (5,5), 0) # se difumina la imagen

# filtra la imagen de escala de grises a blanco y negro

imgThresh = cv2.adaptiveThreshold(imgBlurred, # imagen de entrada

255, # convierte a los pixeles que pasan el umbral en blancos

cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, # usa un filtro gausiano, que la media, parece tener mejores resultados

cv2.THRESH\_BINARY\_INV, # invierte el fondo por el primer plano (fondo negro)

11, # tamanio de un pixel vecino para calcular el umbral

2) # constante sustraida de la media o de la media ponderada

cv2.imshow("imgThresh", imgThresh) # muestra la imagen umbral para referencia

imgThreshCopy = imgThresh.copy() # hace una copia de la imagen umbral, es necesario ya que findContours modifica la imagen

# imagen de entrada, asegurese de usar una copia ya que la funcion modificara esta imagen el en curso de enccontrar los contornos

imgContours, npaContours, npaHierarchy = cv2.findContours(imgThreshCopy,

cv2.RETR\_EXTERNAL, # regresa el contono mas externo

cv2.CHAIN\_APPROX\_SIMPLE) # comprime horizontalmente, verticalmente y diagonalmente segmentos y deja unicamente sus puntos finales

# declara un arreglo numpy vacio, se usara despues para escribir en un archivo

# cero renglones, suficientes columnas para mantener toda la informacion de la imagen

npaFlattenedImages = np.empty((0, RESIZED\_IMAGE\_WIDTH \* RESIZED\_IMAGE\_HEIGHT))

# declara un lista de clasificaciones vacia esta sera nuestra lista de como clasificaremos nuestros caracteres de entrada del usuario, se escribira un archivo al final

intClassifications = []

#posibles caracteres en los que estamos interesados son digitos de 0 a 9,etc

intValidChars = [ord('0'), ord('1'), ord('2'), ord('3'), ord('4'), ord('5'), ord('6'), ord('7'), ord('8'), ord('9'),

ord('A'), ord('B'), ord('C'), ord('D'), ord('E'), ord('F'), ord('G'), ord('H'), ord('I'), ord('J'),

ord('K'), ord('L'), ord('M'), ord('N'), ord('O'), ord('P'), ord('Q'), ord('R'), ord('S'), ord('T'),

ord('U'), ord('V'), ord('W'), ord('X'), ord('Y'), ord('Z')]

for npaContour in npaContours: # para cada contorno

if cv2.contourArea(npaContour) > MIN\_CONTOUR\_AREA: # si el contorno es suficientemente grande para considerarlo

[intX, intY, intW, intH] = cv2.boundingRect(npaContour) # se obtiene y encierra en un rectangul

# se encierra en un rectangulo y se le pregunta al usuario por el caracter

cv2.rectangle(imgTrainingNumbers, # se dibuja un rectangulo en la imagen de entrenamiento original

(intX, intY), # incia en la esquina superior izquierda

(intX+intW,intY+intH), # hasta la esquina inferior derecha

(0, 0, 255), # rojo

2) # el grueso

imgROI = imgThresh[intY:intY+intH, intX:intX+intW] # se corta la imagen

imgROIResized = cv2.resize(imgROI, (RESIZED\_IMAGE\_WIDTH, RESIZED\_IMAGE\_HEIGHT)) # se aumenta el tamanio de la imagen

cv2.imshow("imgROI", imgROI) #muestra el caracter en una nueva ventana para referencia

cv2.imshow("imgROIResized", imgROIResized) # muestra la image con el tamanio modificado

cv2.imshow("training\_numbers.png", imgTrainingNumbers) # muestra la imagen de entrenamiento con el rectangulo dibujado

intChar = cv2.waitKey(0) # se presiona una tecla

if intChar == 27: # si se oprimio esc

sys.exit() # se sale del programa

elif intChar in intValidChars: # si el caracter esta en la lista de caracteres de busca

intClassifications.append(intChar) # se apendiza el caracter entero en el arreglo de caracteres, este se conertira a float despues

npaFlattenedImage = imgROIResized.reshape((1, RESIZED\_IMAGE\_WIDTH \* RESIZED\_IMAGE\_HEIGHT)) # se aplana la imagen para apendizarla al arreglo numpy

npaFlattenedImages = np.append(npaFlattenedImages, npaFlattenedImage, 0) # se agrega la imagen aplanada a l arreglo numpy

# end if

# end if

# end for

fltClassifications = np.array(intClassifications, np.float32) # se convierte la lista de clasificaciones a un arreglo de flotante

npaClassifications = fltClassifications.reshape((fltClassifications.size, 1)) # se aplana el arreglo para escribirlo en un archivo despues

print "\n\nEntrenamiento completo !!\n"

np.savetxt("classifications.txt", npaClassifications) # se escriben las imagenes aplanadas en el archivo

np.savetxt("flattened\_images.txt", npaFlattenedImages) #

cv2.destroyAllWindows() # remueve todas las ventanas

return

###################################################################################################

if \_\_name\_\_ == "\_\_main\_\_":

main()

# end if

**## Archivo test.py**

# GenData.py

import sys

import numpy as np

import cv2

import os

# modula el nivel de las variables ##########################################################################

MIN\_CONTOUR\_AREA = 100

RESIZED\_IMAGE\_WIDTH = 20

RESIZED\_IMAGE\_HEIGHT = 30

###################################################################################################

def main():

imgTrainingNumbers = cv2.imread("training\_chars.png") # lee la imagen con los caracteres de prueba

if imgTrainingNumbers is None: # si el archivo no se lee correctamente

# manda un mensaje de error en la salida estandar

raw\_input("Error la imagen puede ser leida del archivo") # pausa para que el usuario vea el mensaje

return # sale del programa

# end if

imgGray = cv2.cvtColor(imgTrainingNumbers, cv2.COLOR\_BGR2GRAY) # obtiene la imagen en escala de grises

imgBlurred = cv2.GaussianBlur(imgGray, (5,5), 0) # se difumina la imagen

# filtra la imagen de escala de grises a blanco y negro

imgThresh = cv2.adaptiveThreshold(imgBlurred, # imagen de entrada

255, # convierte a los pixeles que pasan el umbral en blancos

cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, # usa un filtro gausiano, que la media, parece tener mejores resultados

cv2.THRESH\_BINARY\_INV, # invierte el fondo por el primer plano (fondo negro)

11, # tamanio de un pixel vecino para calcular el umbral

2) # constante sustraida de la media o de la media ponderada

cv2.imshow("imgThresh", imgThresh) # muestra la imagen umbral para referencia

imgThreshCopy = imgThresh.copy() # hace una copia de la imagen umbral, es necesario ya que findContours modifica la imagen

# imagen de entrada, asegurese de usar una copia ya que la funcion modificara esta imagen el en curso de enccontrar los contornos

imgContours, npaContours, npaHierarchy = cv2.findContours(imgThreshCopy,

cv2.RETR\_EXTERNAL, # regresa el contono mas externo

cv2.CHAIN\_APPROX\_SIMPLE) # comprime horizontalmente, verticalmente y diagonalmente segmentos y deja unicamente sus puntos finales

# declara un arreglo numpy vacio, se usara despues para escribir en un archivo

# cero renglones, suficientes columnas para mantener toda la informacion de la imagen

npaFlattenedImages = np.empty((0, RESIZED\_IMAGE\_WIDTH \* RESIZED\_IMAGE\_HEIGHT))

# declara un lista de clasificaciones vacia esta sera nuestra lista de como clasificaremos nuestros caracteres de entrada del usuario, se escribira un archivo al final

intClassifications = []

#posibles caracteres en los que estamos interesados son digitos de 0 a 9,etc

intValidChars = [ord('0'), ord('1'), ord('2'), ord('3'), ord('4'), ord('5'), ord('6'), ord('7'), ord('8'), ord('9'),

ord('A'), ord('B'), ord('C'), ord('D'), ord('E'), ord('F'), ord('G'), ord('H'), ord('I'), ord('J'),

ord('K'), ord('L'), ord('M'), ord('N'), ord('O'), ord('P'), ord('Q'), ord('R'), ord('S'), ord('T'),

ord('U'), ord('V'), ord('W'), ord('X'), ord('Y'), ord('Z')]

for npaContour in npaContours: # para cada contorno

if cv2.contourArea(npaContour) > MIN\_CONTOUR\_AREA: # si el contorno es suficientemente grande para considerarlo

[intX, intY, intW, intH] = cv2.boundingRect(npaContour) # se obtiene y encierra en un rectangul

# se encierra en un rectangulo y se le pregunta al usuario por el caracter

cv2.rectangle(imgTrainingNumbers, # se dibuja un rectangulo en la imagen de entrenamiento original

(intX, intY), # incia en la esquina superior izquierda

(intX+intW,intY+intH), # hasta la esquina inferior derecha

(0, 0, 255), # rojo

2) # el grueso

imgROI = imgThresh[intY:intY+intH, intX:intX+intW] # se corta la imagen

imgROIResized = cv2.resize(imgROI, (RESIZED\_IMAGE\_WIDTH, RESIZED\_IMAGE\_HEIGHT)) # se aumenta el tamanio de la imagen

cv2.imshow("imgROI", imgROI) #muestra el caracter en una nueva ventana para referencia

cv2.imshow("imgROIResized", imgROIResized) # muestra la image con el tamanio modificado

cv2.imshow("training\_numbers.png", imgTrainingNumbers) # muestra la imagen de entrenamiento con el rectangulo dibujado

intChar = cv2.waitKey(0) # se presiona una tecla

if intChar == 27: # si se oprimio esc

sys.exit() # se sale del programa

elif intChar in intValidChars: # si el caracter esta en la lista de caracteres de busca

intClassifications.append(intChar) # se apendiza el caracter entero en el arreglo de caracteres, este se conertira a float despues

npaFlattenedImage = imgROIResized.reshape((1, RESIZED\_IMAGE\_WIDTH \* RESIZED\_IMAGE\_HEIGHT)) # se aplana la imagen para apendizarla al arreglo numpy

npaFlattenedImages = np.append(npaFlattenedImages, npaFlattenedImage, 0) # se agrega la imagen aplanada a l arreglo numpy

# end if

# end if

# end for

fltClassifications = np.array(intClassifications, np.float32) # se convierte la lista de clasificaciones a un arreglo de flotante

npaClassifications = fltClassifications.reshape((fltClassifications.size, 1)) # se aplana el arreglo para escribirlo en un archivo despues

print "\n\nEntrenamiento completo !!\n"

np.savetxt("classifications.txt", npaClassifications) # se escriben las imagenes aplanadas en el archivo

np.savetxt("flattened\_images.txt", npaFlattenedImages) #

cv2.destroyAllWindows() # remueve todas las ventanas

return

###################################################################################################

if \_\_name\_\_ == "\_\_main\_\_":

main()

# end if

**5.2 Implementación Random Forest[6].**

**## Archivo MLRFC.py**

import matplotlib.pyplot as plt

from sklearn.datasets import load\_digits

from sklearn import ensemble

import pylab as pl

import random

digits = load\_digits()

# define variables

n\_samples = len(digits.images)

x = digits.images.reshape((n\_samples, -1))

y = digits.target

#crea indices aleatorios

sample\_index=random.sample(range(len(x)),len(x)/5) #entre 20 y 80

valid\_index=[i for i in range(len(x)) if i not in sample\_index]

# imagenes muestra y validas

sample\_images=[x[i] for i in sample\_index]

valid\_images=[x[i] for i in valid\_index]

# etiqueta las muestras validas y de prueba

sample\_target=[y[i] for i in sample\_index]

valid\_target=[y[i] for i in valid\_index]

# usando random forest classifier

classifier = ensemble.RandomForestClassifier()

# ajusta el modelo a los datos muestra

classifier.fit(sample\_images, sample\_target)

# intenta predecir datos de validacion

score=classifier.score(valid\_images, valid\_target)

print 'Random Tree Classifier:\n'

print 'Puntaje\t'+str(score)

print classifier.predict([x[i]])

i=800

pl.gray()

pl.matshow(digits.images[i])

pl.show()

# 6 Referencias

# [1] TAVISH SRIVASTAVA. (2014). Introduction to k-nearest neighbors : Simplified. 30 de Noviembre del 2017, de Analytics Vidhya Sitio web: <https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/>

# [2]j2kun. (2012). K-Nearest-Neighbors and Handwritten Digit Classification. 30 de Noviembre del 2017, de jeremykun Sitio web: <https://jeremykun.com/2012/08/26/k-nearest-neighbors-and-handwritten-digit-classification/>

# [3]TAVISH SRIVASTAVA. (2015). Tuning the parameters of your Random Forest model. 30 de Noviembre del 2017, de Analytics Vidhya Sitio web: <https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/>

# [4]https://www.investopedia.com/terms/g/gini-index.asp

# [5] La lista de módulos usados que necesitan instalación son: cv2, scipy, tk-inter, matplotlib, pylab y sklearn. Todos para Python 2.7

# [6] <https://github.com/Ax-Angel/LetterPatternRecognition>

# [7] <https://en.wikipedia.org/wiki/Optical_character_recognition>