Procesado de Imagen y Visión por Computador

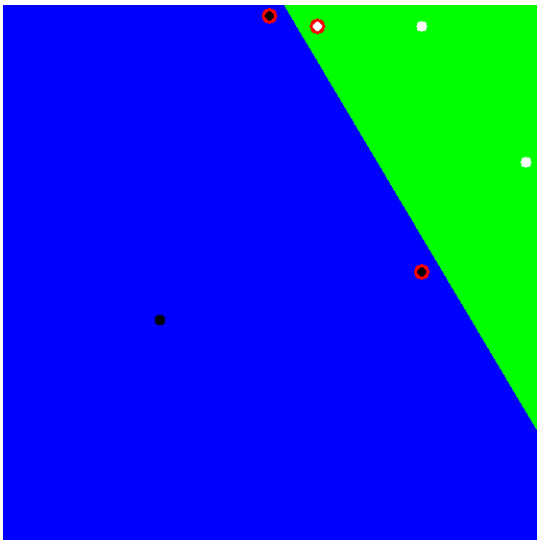
Clasificación. Herramientas básicas

1.- Clasificación mediante SVM

En este apartado vamos a realizar un "Toy example" con SVM similar al que puede encontrar en la página http://www.csie.ntu.edu.tw/~cjlin/libsvm. En este caso lo realizaremos en Python y con OpenCV a partir del código ya implementado en el fichero *SVM.py*. Partiendo de ese código se pide:

a) Edite el script y analice el código hasta entender los pasos fundamentales del mismo.

b) Ejecute el script. El resultado debería ser una imagen como la siguiente:



c) Responda a las siguientes preguntas: ¿Qué representan las regiones verde y azul? ¿Qué son los puntos blancos y negros? ¿Qué representan los puntos marcados en rojo?

d) Modifique el código para que en vez de ser una frontera lineal la frontera de decisión sea curva. Para ello cambiaremos el tipo de kernel a RBF en la definición de nuestra SVM. Solamente es necesario cambiar cv.ml.SVM_LINEAR por cv.ml.SVM_RBF. Pruebe con distintos valores de Gamma y C. Guarde los resultados en imágenes con distintos nombres.

e) Añada más puntos al problema de forma que este no sea linealmente separable (Fichero *Puntos.txt*). Con 4 puntos puede ser suficiente pero puede añadir más. Realice la clasificación con el kernel Lineal y con el RBF. Pruebe con distintos valores de Gamma y C. Guarde los resultados en imágenes con distintos nombres.

Procesado de Imagen y Visión por Computador

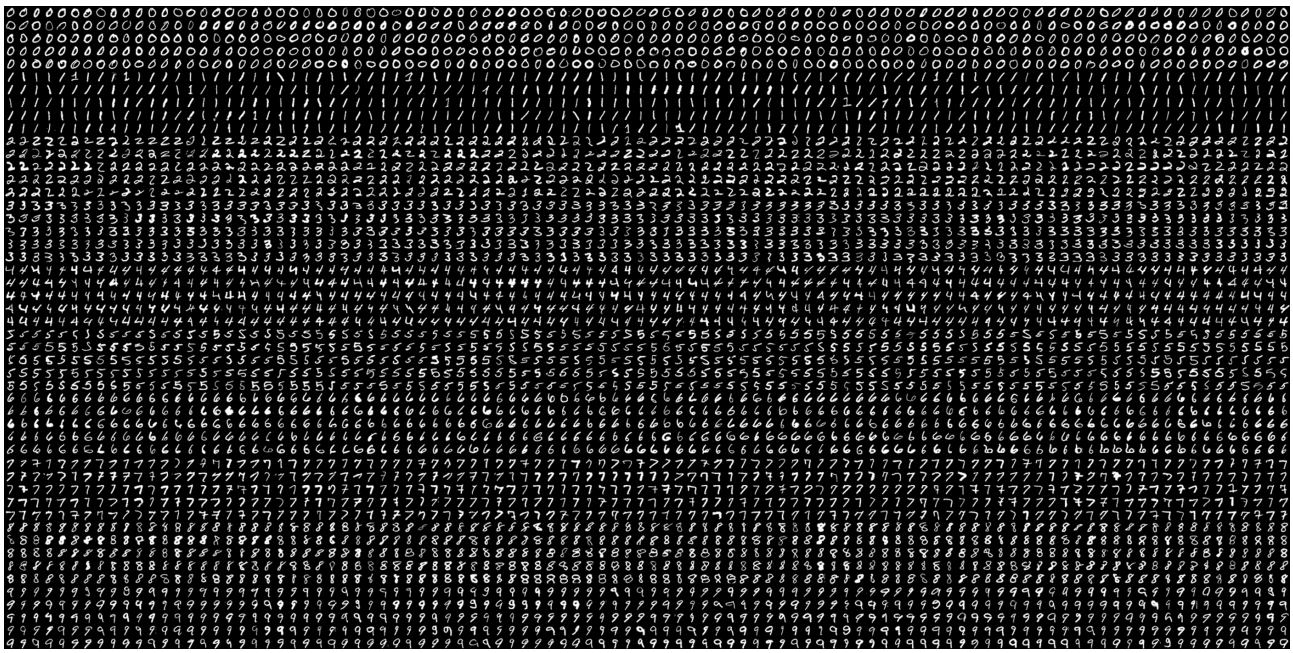
Clasificación. Herramientas básicas

2.- Reconocimiento de dígitos.

En este apartado vamos a probar, en un problema real, la potencia de los algoritmos de clasificación. En concreto vamos a comprobar el funcionamiento de los algoritmos KNN (K-Nearest Neighbour[1]) y SVM (Support Vector Machines[2]) que se han visto en clase.

El problema que vamos a trabajar es el de reconocimiento de números escritos a mano (Hand-written Digits). Este es un problema complejo en el que la variabilidad de los datos es muy grande y se necesita, no sólo un buen clasificador, sino también un procesado previo de las imágenes y una base de datos de dígitos ya guardados para poder realizar el entrenamiento.

En nuestro caso nos basaremos en unos programas de ejemplo que se pueden encontrar en la distribución de OpenCV[3] entre otras muchas aplicaciones. Estos programas usan para el entrenamiento dígitos obtenidos de la base de datos MNIST[4]. En concreto nosotros usaremos un subconjunto de dicha base que se muestra en la imagen siguiente:



Los programas que vamos a usar son los siguientes:

- *digits.py*. Este programa tiene dos funciones, por una parte realiza el entrenamiento tanto en el caso de KNN como en el de SVM (en este caso guarda además el modelo de entrenamiento obtenido) y, por otra parte, prueba dicho entrenamiento devolviendo la matriz de confusión[5], el porcentaje de error cometido al testear y una imagen con los dígitos correctamente reconocidos y los que no lo han sido.
- *digits_adjust.py*. Este programa nos va a permitir buscar de manera automática los mejores parámetros para el entrenamiento de la SVM o de KNN. En el caso de SVM ya hemos visto en el apartado anterior como al usar un kernel RBF el ajuste de los parámetros C y γ es fundamental para obtener la mejor frontera de decisión posible.

1 https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html#knn-understanding

2 https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_svm/py_svm_basics/py_svm_basics.html#svm-understanding

3 https://github.com/opencv/opencv/tree/master/samples/python

4 http://yann.lecun.com/exdb/mnist/

5 https://es.wikipedia.org/wiki/Matriz_de_confusi%C3%B3n

- *digits_video.py*. Nos va a permitir probar nuestro entrenamiento en tiempo real usando para ello la cámara de nuestro ordenador a la que le mostraremos una serie de dígitos escritos a mano por nosotros mismos.

La realización de la práctica en este apartado consistirá en los siguientes puntos:

a) Entender qué procesos se están realizando en la ejecución de los programas. Por ello empezaremos por abrir el código correspondiente a *digits.py*. Tras cargar las imágenes de los dígitos y sus etiquetas (línea 152), se realiza un preprocesado de las imágenes que incluye un proceso de *deskew* y la extracción de las características HOG. Para entender el efecto de *deskew,* descomente las líneas 159 y 161. De este modo visualizará los dígitos antes y después del proceso de deskew (ventanas 'digits' y 'digits_deskew', respectivamente). Responda: ¿Cuál es la finalidad del proceso de *deskew*?

b) En las líneas 164-169 se realiza la configuración de los conjuntos de entrenamiento y de test. ¿Qué porcentaje de muestras se utiliza para el entrenamiento y cuál para el test?

c) Al ejecutar el script visualizará en la consola de Spyder el porcentaje de error obtenido con las dos técnicas: kNN y SVM, así como sus correspondientes matrices de confusión. ¿Qué representan y cómo se interpretan estas matrices? ¿cómo se relacionan con los porcentajes de error? ¿Cuál de los dos métodos parece más seguro? ¿Qué representan los dígitos en rojo en la visualización de las tablas de números?

d) Si ejecuta el script más de una vez, puede comprobar que los resultados de las diferentes ejecuciones son distintos. Esto se debe al código de las líneas 156-158. ¿Cuál es el motivo?

e) El programa *digits.py* no tiene los valores óptimos de los parámetros C y de las SVM y tampoco el parámetro *k* de KNN, por lo que los resultados obtenidos no serán los mejores posibles. Para determinar los valores óptimos use el programa *digits_adjust.py*. Por defecto realiza la optimización de los parámetros de la SVM, pero podemos también realizar la optimización del parámetro *k* de KNN. Para ello debe pasar como argumento de entrada '--model knearest'. Responda a las siguientes preguntas: ¿Qué representan las salidas sucesivas por consola en la ejecución? ¿Cuáles son los parámetros óptimos para cada uno de los métodos? Una vez obtenidos los valores óptimos cambie el valor de los correspondientes parámetros en *digits.py* y compruebe su efecto. ¿Cómo ha cambiado el error sobre el obtenido en el apartado c?

Una vez llegados a este punto, estamos en condiciones de probar el funcionamiento en tiempo real con nuestros propios dígitos. Para ello usaremos el programa *digits_video.py.* Escriba unos cuantos números en un papel y, durante la ejecución, muéstrelo a la cámara para comprobar su funcionamiento. Por defecto se usará SVM pero también puede probar con KNN pasando como argumento la cadena de texto 'knn'. Modifique el código para poder grabar el resultado de la detección en un vídeo.

Procesado de Imagen y Visión por Computador
Clasificación. Herramientas básicas

Material a entregar

1. Apartado 1: memoria explicativa con la respuesta a las preguntas planteadas, resultados generados y conclusiones obtenidas.
2. Apartado 2: memoria explicativa con la respuesta a las preguntas planteadas y vídeo de resultados.