Master: Ciencia de Datos e Ingeniería de Computadores

Curso: Visión por Computador

Rosa Mª Rodríguez Sánchez

Dpto. Ciencias de la Computación e I. A.

E.T.S. de Ingenierías Informática y de Telecomunicaciones

Universidad de Granada

Clasificacion y Segmentación de personas andando

Índice de contenido

1. Introducción

Esta práctica se compone de dos bloque fundamentales:

1. Crear algoritmos de clasificación que reconozcan a peatones. Para llevar a cabo este ejercicio analizaremos dos aproximaciones:
    1. Aplicar un clasificador SVM (Support Vector Machine). Para ello caracterizaremos las imágenes usando el histograma de orientación de los gradientes (HOG). Asi para cada imagen obtendremos un vector de HOG. Se pide al alumno/a que implemente la función HOG y no use ninguna librería para ello.
    2. Aplicar un clasificador basado en redes neuronales de convolución para clasificar imágenes en el grupo que contiene peatones o en el grupo que no contiene peatones.
2. Crea una red neuronal de convolución para aplicar segmentación de peatones en imágenes.

2. Base de Datos para la Clasificación.

Nuestro objetivo es detectar si sobre una imagen existe una persona andando (peatón) o no. Para ello vamos a trabajar con una base de datos de imágenes donde aparecen imágenes con personas andando y otra base donde no aparecen personas andando.

La idea subyacente es dado un conjunto de imágenes con personas andando y otro conjunto de imágenes que no contienen personas andando, debemos crear un clasificador que aprenda sobre estos dos conjuntos las características que deben de darse en la imagen para que aparezca una persona andando.

2.1. Preparar los datos para las imágenes con personas andando

La base de datos de imágenes con personas andando se componen de 900 imágenes. Para entrenar nuestro clasificador usaremos solamente 400, escogidas de forma aleatoria. Cada una de estas imágenes tiene 128 filas por 64 columnas

2.2. Prepara los datos para imágenes sin personas andando

Las imágenes sin personas andando tienen diferentes dimensiones de las dimensiones de las imágenes con personas andando. Para obtener una muestra de esta imágenes lo que haremos es escoger aleatoriamente un conjunto, hacer un resize de estas imágenes a 512x512 y a continuación escoger de esta imagen un trozo con dimensiones 128x64 (dimensiones de las imágenes con personas andando).

3. Clasificación usando SVM

Para aplicar el clasificador SVM primero debemos caracterizar cada imagen. Con tal fin, vamos a caracterizar cada imagen usando el Histograma de Orientación de Gradiente.

3.1. Histogramas de Orientación (HOG)

Para aplicar HOG en primer lugar debemos obtener las imágenes gradiente

Para ello podemos aplicar diferentes técnicas. Una forma simple es convolucionar la imagen con los siguientes filtros

$$[-1,0,1]\, y\, [-1,0,1]^T$$

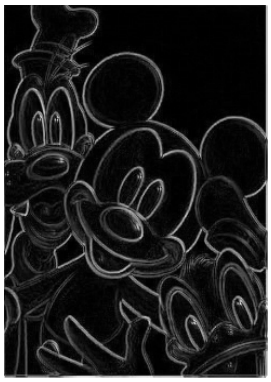
Obteniendo aristas en la dirección vertical y horizontal.

Gx

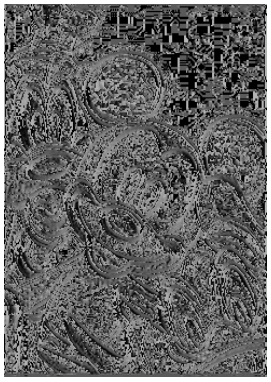
Gy

La magnitud ($\sqrt{G_x^2+G_y^2}$) y orientación ($atan(\frac{G_y}{G_x})$ sumándole a este resultado 90 grados para obtener la orientación de la arista):

Magnitud

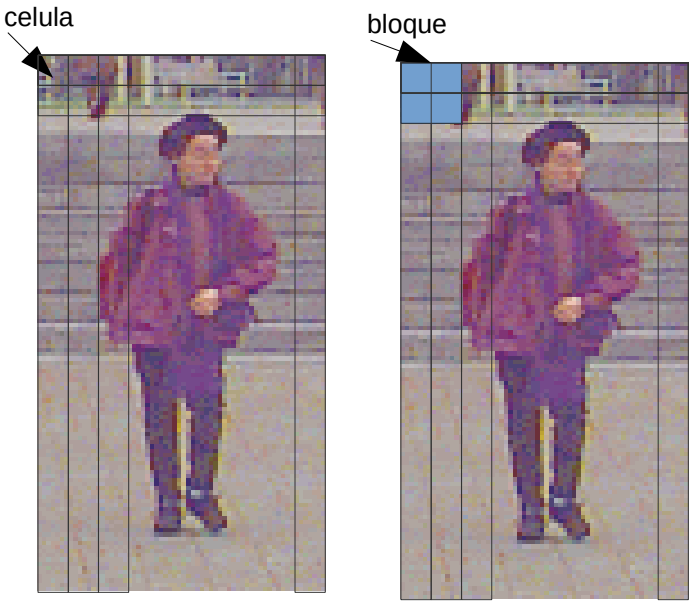


Orientación



Definición de las células y bloques

El siguiente paso es dividir la imagen en un conjunto de células (cuadrados vecinos) y a su vez las células vecinas se agrupan en bloques:



Una célula puede pertenecer a diferentes bloques. Los bloque por lo tanto se solapan. Un ejemplo de división sería tener células de tamaño 8x8 y bloques de tamaño 16x16. De forma que cada bloque se compone de 4 células. Dos bloques vecinos comparten dos células.

Obtener los histogramas de orientación del gradiente

Sobre cada célula vamos a obtener un histograma de orientación del gradiente. Para ello vamos a analizar orientaciones en el rango 0 grados a 180 grados. Así sea **h** el histograma, en **h(i)** acumula la magnitud de los pixeles en una célula que tiene orientación **i**:

$$h(i)=\sum_{\forall\ pixel\ p\ en\ la\ celula\ y\ atan(p)=i} magnitud(p)$$

Por ejemplo si el rango de orientación [0,180] lo dividimos en 9 partes, calcularemos **h** para {10,30,50,70,90,110,130,150,170,180} (orientaciones).

Para que no haya un efecto de aliasing en el cálculo del histograma sería interesante realizar una interpolación bilineal cuando se incrementa el histograma en el *bin* **i**. Es decir suponer que la orientación del pixel *p* es 20. Realmente no tenemos ese *bin* pero se encuentra entre 10 y 30 por lo tanto en h(10) incrementaremos con el valor **magnitud(p)*(30-20)/(30-10**) y en h(30) incrementaremos con el valor **magnitud(p)*(20-10)/(30-10).**

Con este paso caracterizamos cada célula con un vector de nueve elementos dados por *{h(10),h(30),h(50),h(70),h(90),h(110),h(130),h(150),h(170),h(180)}*.

Estos histogramas se agrupan como un vector del bloque (4 histogramas cada uno por una célula en el bloque). Y a continuación se normaliza como : $bloque = \frac{bloque}{\sqrt{||bloque||^2 + e}}$.
Siendo ***e*** una constante pequeña para evitar divisiones por cero (e=0.01)

Para la imagen de disney podemos ver en la imagen de la derecha que orientación de arista domina en cada célula.

Obtener el vector de descriptores

Con el conjunto de datos obtenido en cada bloque este se agrupa en lo que vamos a denominar vector de descriptores HOG. Este vector igualmente debe ser normalizado:

$$HOG = \frac{HOG}{\sqrt{||HOG||^2 + e_2}}$$

Además si HOG en una posición adopta un determinado valor por encima de un valor este se satura a un valor máximo. Por ejemplo si HOG(i)>0.2 este se modifica a 0.2. Esto evita que determinadas regiones sobresalga con mucha magnitud frente a otras que también son importantes.

3.2. Obtener el clasificador SVM

Según vimos en la sección 2, nuestra base de datos tiene imágenes con peatones (imágenes positivas) y tiene imágenes sin peatones (imágenes negativas). Sobre estas imágenes debemos obtener el vector de descriptores HOG. Para las imágenes positivas obtendremos ***HOG_X_pos*** y ***y_pos*** el valor de clase para las imágenes positivas será 1.

Por otro lado, para las imágenes negativas también tendremos su vector de descriptores HOG que denominamos ***HOG_X_neg*** y ***y_nega*** el valor de clase que será en este caso -1. A continuación creamos una matriz **X** que se construye como la concatenación de ***HOG_X_pos*** y ***HOG_X_neg***. Y el vector **y** como la concatenación de y_pos e y_neg.

Sobre **X** e **y** obtenemos una división en un conjunto de entrenamiento **X_train** e **y_train** y un conjunto test como **X_test** e **y_test**. Una posible división es dejar el 80% de total de imágenes para entrenamiento y el 20% para test.

Para hacer esto podemos ejecutar las siguientes sentencias:

```
X=np.concatenate([HOG_X_pos,HOG_X_neg])#Concatenación de HOG
y=np.concatenate([y_pos,y_neg]); #Concatenación de valor de la clase
ntrain= round(0.8*len(y));#cardinal del conjunto entrenamiento
ntest = len(x)-ntrain; #cardinal del conjunto test
```

```
idx = np.random.permutation(np.arange(len(y))) #obtenemos una permutación
X_train= X[idx[0:ntrain],:]; #Obtenemos barajado el conjunto entrenamiento
y_train=y[idx[0:ntrain]] #igual para el valor de clase
X_test= X[idx[ntrain+1:len(y)],:];#Obtenemos barajado el conjunto test
y_test=y[idx[ntrain+1:len(y)]]; #igual para el valor de clase
```

Una vez obtenida esta división aplicamos el clasificador SVM (ver sección 7) con la siguiente sentencia:

```
from sklearn.svm import SVC

m=SVC();

 m.fit(X_train,y_train)
```

Aplicar el clasificador y obtener la bondad del mismo

Para aplicar el clasificador sobre nuestro conjunto de datos de entrenamiento ejecutaremos las sentencias:

```
y_pred_train=m.predict(X_train);
```

y de igual forma para el conjunto test sería

```
y_pred_test=m.predict(X_test);
```

Una vez calculadas estas predicciones de clase se obtiene la matriz de confusión como

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train,y_pred_train)  y
confusion_matrix(y_test,y_pred_test)
```

4. Clasificación usando CNN

En este parte de la práctica el alumno/a diseñará una arquitectura de red neuronal de convolución para dada una imagen la clasifique como que contiene peatones o que no contiene peatones. Para este fin usaremos transferencia de aprendizaje. De esta forma partiendo de una arquitectura existente la modificaremos para el fin que tenemos entre manos. El objetivo es analizar que arquitectura obtiene la mayor precisión en la clasificación. Las arquitecturas que debemos probar son: VGG16, ResNet18 y Xception.

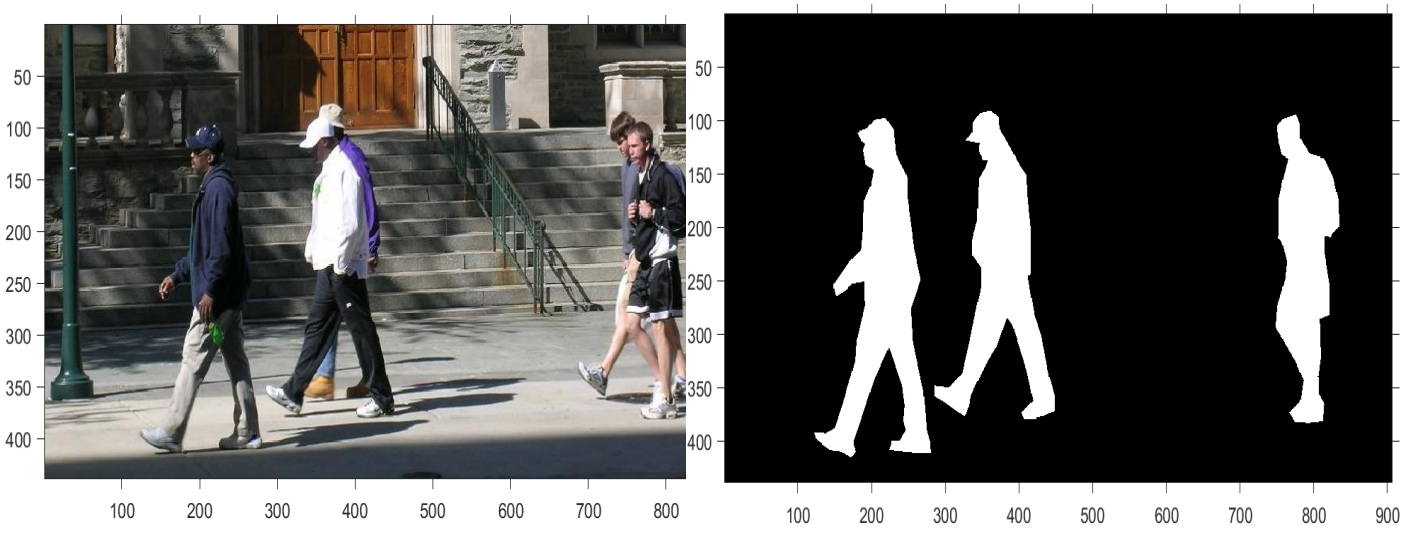
Os recuerdo que en la práctica 6 hemos obtenido diferentes clasificadores y se uso transferencia de aprendizaje usando como partida la arquitectura Xception.

5. Segmentación Semántica.

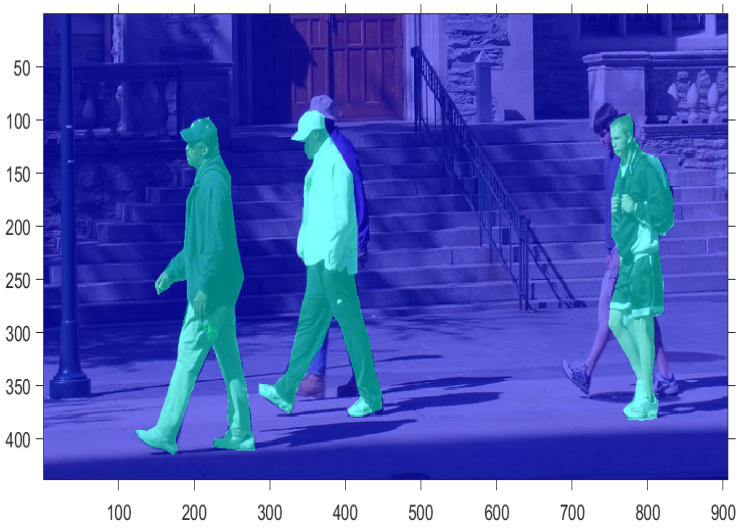
El objetivo en este apartado es que el alumno/a establezca un diseño de red neuronal (basado en una arquitectura autocodificadora) para dada una imagen donde aparezca peatones nos marque donde se encuentran los peatones.

5.1. Base de Datos

La base de datos para aplicar segmentación sintáctica se encuentra dividida en un conjunto de entrenamiento y un conjunto de test. Para ambos conjuntos tenemos el conjunto de imágenes además la máscara que establece que pixeles pertenecen a un peatón. En las siguientes imágenes se puede ver un conjunto de peatones y a la derecha la máscara correspondiente. En esta imagen solamente se ha señalado 3 peatones.



A partir de la imagen original podemos distinguir lo que es peatón, según la máscara, de lo que no es peatón. La siguiente imagen ilustra este resultado:



El alumno/a debe tener en cuenta que las imágenes tienen diferentes dimensiones tanto en el conjunto de entrenamiento como en el conjunto test. Por lo tanto un paso previo será normalizar las imágenes a la misma dimensión.

5.2. Ejemplo de arquitectura

Una arquitectura base podría ser la siguiente:

15x1 Layer array with layers:

1	'imageinput'	Image Input	214x214x3 images with 'zerocenter' normalization
2	'conv_1'	Convolution	64 3x3x3 convolutions with stride [1 1] and padding 'same'
3	'relu_1'	ReLU	ReLU
4	'maxpool_1'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
5	'conv_2'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding 'same'

```
 6 'relu_2'           ReLU                ReLU
 7 'maxpool_2'        Max Pooling         2x2 max pooling with stride [2 2] and padding [0  0  0  0]
 8 'conv_3'           Convolution         64 3x3x64 convolutions with stride [1  1] and padding
'same'
 9  'relu_3'                ReLU                          ReLU
 10 'conv_4'                 Convolution                   64 3x3x64 convolutions with stride [1  1]
and padding 'same'
 11 'transposed-conv_1'      Transposed Convolution        64 4x4x64 transposed convolutions with
stride [2  2] and cropping 'same'
 12   'transposed-conv_2'    Transposed Convolution       64 4x4x64 transposed convolutions with
stride [2  2] and cropping [0  0  0  0]
 13   'conv_5'               Convolution                  2 1x1x64 convolutions with stride [1  1]
and padding [0  0  0  0]
 14   'softmax'              Softmax                      softmax
 15   'classoutput'          Pixel Classification Layer   Cross-entropy loss with classes 'fondo' and
'peaton'
```

Fijaros que en esta arquitectura tienes una zona codificadora que va desde la capa 2 hasta la capa 10. Esta arquitectura exige que las dimensiones de las imágenes sean 214x214x3 (ver capa 1).

En la capa 2 tenemos una capa de convolución con 64 filtros de dimensiones 3x3x3. Por lo tanto la salida de esta capa tendrá dimensiones 214x214x64. En la siguiente capa se aplica una rectificación no lineal usando ReLu. En la capa 4 aplica max-pooling lo que obtiene una salida de 107x107x64. Estos tres bloques se repiten respectivamente en la capa 5,6 y 7. Las capas 8 y 9 son de convolución y relu. Y la capa 10 aplica una nueva capa de convolución.

Las capas 11 y 12 es la parte decodificadora. Estas capas aplica un proceso de deconvolución con una capa de convolución traspuesta. Esta aplica sobremuestreo al doble aplicando una convolución a continuación.

La capa 13 es una capa de convolución que aplana las salidas de la capa 12 a dos salidas (en nuestro caso peatón no peatón). A continuación aplicamos una capa softmax con función de perdida entropia cruzada para obtener la clases fondo o peatón. La capa 15 es una capa de clasificación de pixeles. En este caso a dos clases fondo y peatón.

5.3. Ejercicio

Tomar como base la arquitectura comentada en la sección anterior. Modificarla (cambiando las capas, numero,etc). Realizar un análisis de las arquitecturas estudiadas:

- Obtener en este análisis el promedio de IoU (intersection over union) a lo largo del conjunto test sobre las diferentes arquitecturas estudiadas. Obtener la agudeza
- Tiempo de aprendizaje
- Dimensión de la representación latente

6. Ejercicio

Los ejercicios a realizar son:

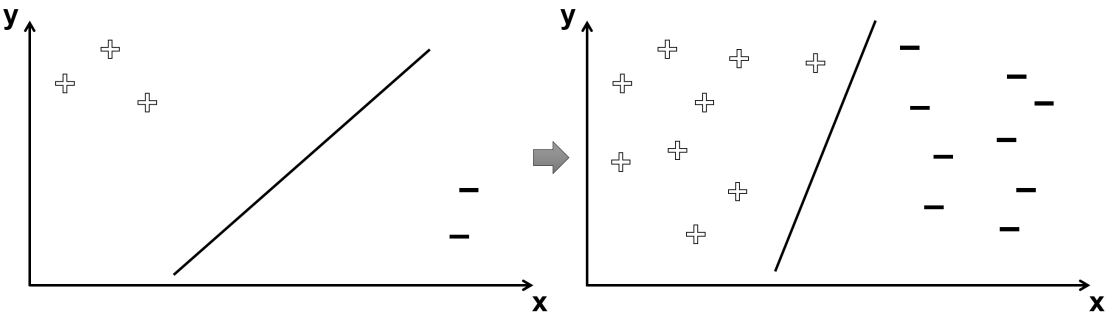
1. Sobre la base de datos de peatones para clasificación.
    1. Aplicar un clasificador SVM. Se deberá implementar HoG (no usar una función ya implementada).Obtener resultados de bondad del clasificador
    2. Aplicar un clasificador usando una red neuronal de convolución. Se deberán reusar y modificar las arquitecturas VGG16, ResNet18 y Xception para tal fin. Obtener una comparativa de bondad usando predictores como agudeza, F1, recall, precision.
    3. Compara la mejor red neuronal obtenida en el punto 2 con el clasificador SVM.
2. Sobre la base de datos de peatones para segmentación semántica. Partiendo de la arquitectura base probar diferentes modificaciones de esta. Realizar un análisis de las arquitecturas estudiadas:
    - Obtener en este análisis el promedio de IoU (intersection over union) a lo largo del

conjunto test sobre las diferentes arquitecturas estudiadas. Obtener la agudeza. Tiempo de aprendizaje. Dimensión de la representación latente.

Se permiten implementaciones en Matlab y Python.

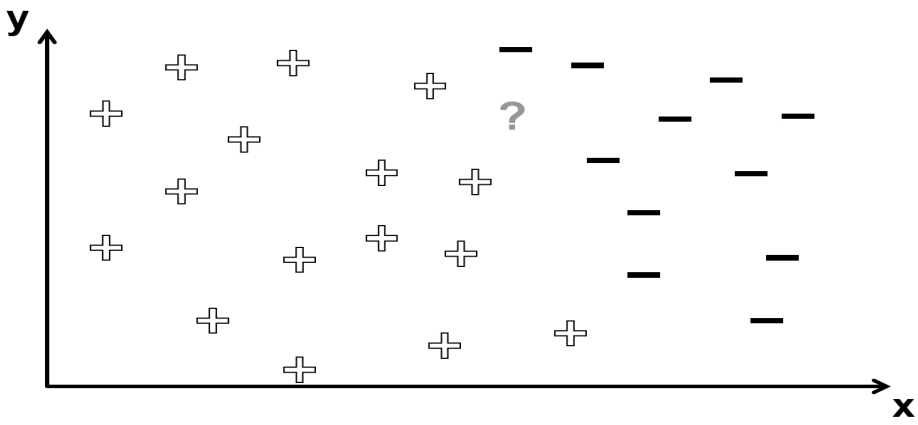
7. Principios del Clasificador Support Vector Machine (SVM)

Consideremos un conjunto de datos que debemos clasificar con solamente dos rasgos (x e y) y una etiqueta objetivo (1 o -1). En clasificación binaria, una frontera de decisión es una línea que divide el conjunto de entrenamiento en dos subconjuntos, uno por cada clase. Una frontera de decisión óptima divide los datos con -1 a la izquierda de la frontera de decisión y los objetos con etiqueta 1 esta a la derecha de la frontera de decisión. Conforme el progreso continua, el clasificador ve más y mas muestras, de esta forma modifica la frontera en cada paso. Este proceso se ilustra en la siguiente figura:



La frontera de decisión se modifica constantemente durante la fase de entrenamiento

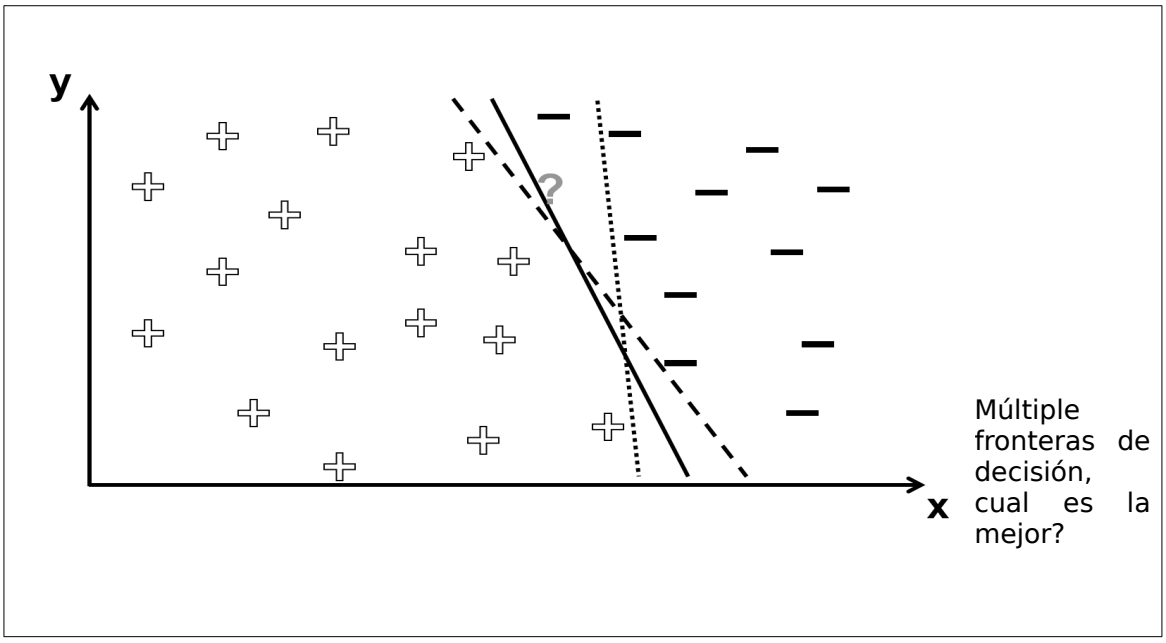
Durante la fase de test queremos establecer cual es la clase de ?, dada una nueva muestra,



Con la frontera aprendida podemos predecir la clase de un nuevo dato ?

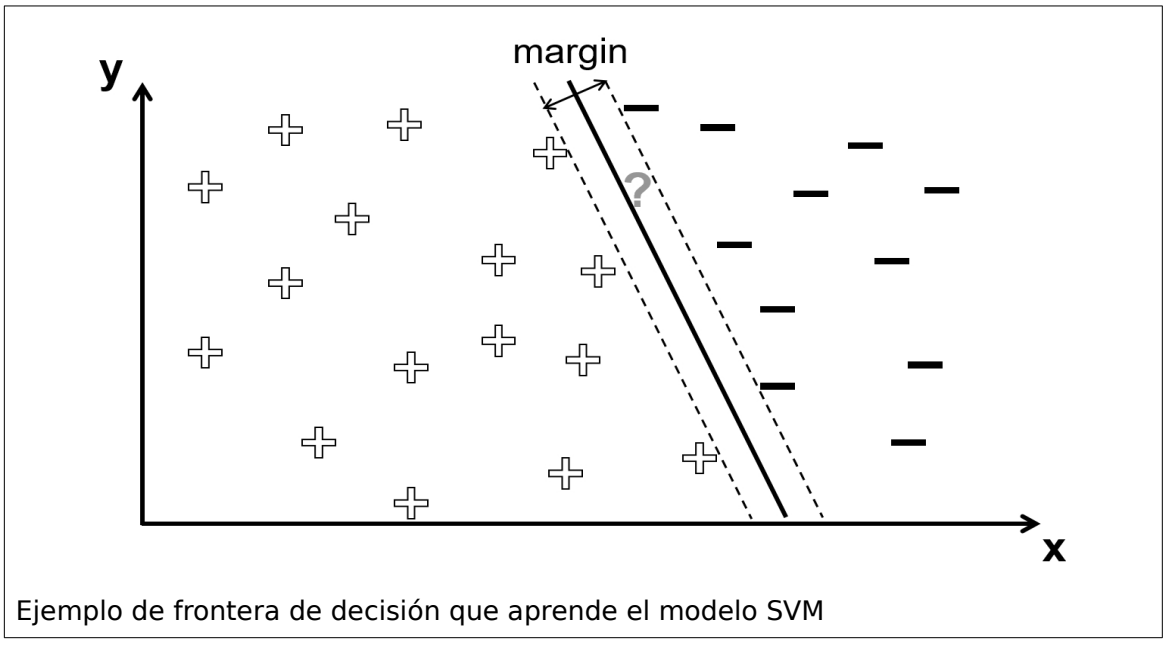
usando la frontera de decisión,

De todas formas podemos tener problemas en cuanto a la clasificación del nuevo dato. Si la localización de ? está más a la izquierda tendríamos certeza de que la clase es +,Por otro lado, tenemos diferentes formas de definir la frontera de decisión, cumpliendo que todas las + quedan a la izquierda y todos los – quedan a la derecha como se puede ver en la siguiente figura:



Múltiple fronteras de decisión, cual es la mejor?

Por lo tanto, ¿cuál es la mejor frontera de decisión?. Un clasificador SVM escogería la línea sólida como frontera, ya que esta frontera de decisión maximiza el margen entre los puntos de los datos + y -.



Ejemplo de frontera de decisión que aprende el modelo SVM

Referencias:

- Histograms of Oriented Gradients for Human Detection. Navneet Dalai and Bill Triggs
- Practical Image and Video Processing Using Matlab. Oge Marques. Ed. Wiley.