INSTITUTO UNIVERSITARIO AERONAUTICO

DETECCIÓN DE OBJETOS POR COMPUTADOR

ANÁLISIS DE ALTERNATIVAS PARA EL DESARROLLO DE UN SISTEMA DETECTOR DE PERSONAS

Proyecto de Grado

Facultad de Ciencias de la Administración – Ingeniería de Sistemas

Autor: Rodolfo Martín Villanueva

Profesor Tutor: Ing. Juan Francisco Giro Martín

CONTENIDO

1 DEDICATORIA

A mi familia.

2 AGRADECIMIENTO

A Yoko.

A todas las personas que hicieron y hacen posible la educación a distancia en el IUA.

3 RESUMEN

Este trabajo tiene por fin desarrollar una aplicación de detección y reconocimiento de objetos. La visión por computador es un tema específico dentro de lo que se conoce como “Machine Learning” o “Aprendizaje Automático” en español. Investigando el tema se concluye que no existe una única técnica eficaz para detectar distintas clases de objetos en todas las situaciones. Existen técnicas que han probado su efectividad para algunas clases de objetos. La intención primaria de este trabajo es desarrollar y evaluar algunas de las principales técnicas de visión por computador que permiten detectar y reconocer personas en una imagen.

Por lo tanto, en este trabajo se profundizara en el tema de la detección de una determinada clase de objetos, en este caso de personas. Cuando en este documento se habla de detección de objetos, se hace referencia a utilizar técnicas de visión por computador. En principio se puede pensar en tener una sola imagen y detectar objetos dentro de esa imagen, (en este caso, personas) o se puede pensar en tener videos, es decir, una secuencia de imágenes, y detectar objetos en esa secuencia. Este trabajo se centrara en el caso de tener una sola imagen y tener que buscar los objetos en esa imagen. Se utilizara principalmente la apariencia visual de los objetos para desarrollar un ejemplo de aplicación de detección de personas, ya que hay otras técnicas que se basan en los colores de los pixeles, pero que no se desarrollara aquí. Siendo un poco más específico, se partirá de un modelo, y lo que se quiere, es saber en qué lugares aparece ese modelo dentro de la imagen.

La visión por computador no es un tema fácil. Como humanos, si se presenta una imagen y se pide buscar objetos en ella, se hace con relativa sencillez. Sin embargo, para un sistema automático no es tan sencillo. Solo en la última década se ha empezado a obtener resultados que pueden ser usados en ciertas aplicaciones. La detección de objetos sigue siendo un campo de investigación en la actualidad. Lo que se quiere, es poder distinguir una clase de objetos de otra. El modelo de las personas no puede ser igual que el modelo de cualquier otro objeto.

Por otra parte, los modelos tienen que presentar invariancia a la propia variabilidad que existe en las clases de objetos. Por ejemplo, si se busca personas en una imagen, los individuos en ella tendrán distinta ropa, con distintas localizaciones, con distintas posturas, existirán oclusiones, etc.

Además en esa imagen se tiene que tener en cuenta las condiciones distintas del entorno, por ejemplo, si el día es soleado o es nublado, es decir que los objetos que se busquen deben ser independientes de esas condiciones.

Por otro lado, se tiene la propia búsqueda de los objetos en la imagen, que proporciona todas las posibles ventanas que puedan contener los objetos que interesan y da todas las posibilidades para encontrar al objeto. Por otra parte esto tiene que ser un mecanismo de búsqueda lo suficientemente eficiente.

Por último, es necesario aclarar que la visión por computador es un tema en continuo desarrollo y con numerosas fuentes, que hace que se encuentren muchos temas con el mismo nombre, desarrollados con notorias diferencias, entre una fuente y otra. Esto obliga a desarrollar el marco teórico del presente documento en forma un poco más detallada, para poder aclarar, en forma precisa, a que se hace referencia.

4 MOTIVACIÓN

Como ya se ha expresado, dentro de la detección de objetos en general, en este trabajo se está interesado en desarrollar una aplicación que permita detectar personas, distinguiéndolas claramente de cualquier otro objeto. Por otra parte resulta necesario analizar y llegar a conclusiones respecto de cuales, entre las técnicas empleadas aquí, son las mejores. Existe una enorme cantidad de situaciones donde un sistema con estas capacidades sería muy útil. Se puede pensar, por ejemplo, en un perímetro amplio alrededor de una gran instalación. En este caso simples sensores de movimiento no son posibles porque darían falsas alarmas, mientras un sistema que detectara personas no fallaría en el caso de que una persona intente cruzar dicho perímetro.

5 ESTADO DEL ARTE

En la actualidad existen diversas técnicas de detección de objetos, cuyas efectividades están relacionadas a la clase de objetos a detectar y los objetivos finales que se tienen. Por otra parte, estás se combinan entre sí, para lograr resultados. Existen numerosas investigaciones científicas con relación a la visión por computador y en especial a la detección de personas: Analizando varios de ellos, se llega a la conclusión de que técnicas son las más usadas, o de referencia, actualmente, y ello justifica cuales se han elegido para este trabajo. Véase:

- “Pedestrian detection at 100 frames per second” [13]
- “Histograms of Oriented Gradients for Human Detection” [14]
- “Random Forests of Local Experts for Pedestrian Detection” [15]
- “An HOG-LBP Human Detector with Partial Occlusion Handling” [16]
- “Opponent Colors for Human Detection” [22]
- “An Extended Set of Haar-like Features for Rapid Object Detection” [23]

6 ALCANCE DEL TRABAJO

El alcance del presente trabajo es **analizar**, **desarrollar** y **evaluar** alternativas en reconocimiento de personas en imágenes. Se quiere desarrollar un módulo que demuestre que es posible la detección de personas. Porque de esta manera se demuestra que es posible desarrollar aplicaciones que usen la detección de objetos por

computadora. Por lo tanto es necesario investigar y comprender las técnicas de un sistema automático de detección y reconocimiento de objetos en imágenes. Analizar las diferentes técnicas de descripción, clasificación y localización de todas las instancias de un objeto en una imagen.

Esta fuera del alcance del presente trabajo desarrollar un sistema completo de detección de personas para ser usado para otro fin específico que no sea el ya mencionado.

Por otra parte, existen muchas otras técnicas de visión por computador, por lo cual es necesario aclarar que está fuera del alcance de este trabajo toda otra técnica que no sean alguna de las siguientes: **HOB** [4, 14, 7]**, LBP** [3,7]**, HOB+LBP, Ventana deslizante, Pirámide con ventana deslizante, (Pyramidal Sliding Window)** [9, 7, 13]**, SVM** [5, 12, 7] **y Regresión Logística** [6, 12 ,7].

Por otra parte, algunas técnicas muy extendidas, pero que no se han seleccionado para el presente trabajo porque los resultados de su aplicación no se ajustan al objetivo específico que se tiene aquí, son Template matching [1, 7], Filtros de Haar + Adaboost, [2, 7, 24]. Otras técnicas que no se desarrollaran, porque su complejidad excede por mucho el alcance del presente trabajo, son L-BFGS [10, 12] y Método del gradiente conjugado [11, 12].

6.1 TÉCNICAS QUE SE USARAN EN ESTE TRABAJO:

- Histogram of oriented gradients, (HOG) es un descriptor de toda la imagen, que captura la forma del objeto.
- Local Binary Patterns, (LBP) es un poderoso clasificador de texturas.
- (LBP y HOG son dos descriptores de imágenes que pueden ser combinados: han resultado ser unos eficientes descriptores de ciertas clases de objetos, **en particular de personas**. Su efectividad, particular o combinada, será analizada en el presente trabajo.)
- Support Vector Machines, (SVM) son algoritmos de aprendizaje supervisado. Dado un conjunto de ejemplos de entrenamiento, (de muestras), se puede etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase en una nueva muestra.
- Regresión Logística es un tipo de análisis de regresión utilizado para predecir el resultado de una variable que puede adoptar un número limitado de categorías, en función de las variables independientes.
- (Estas dos últimas técnicas son clasificadoras de imágenes, cuya efectividad será analizada en el presente trabajo.)

6.2 TÉCNICAS QUE NO SE USARAN EN ESTE TRABAJO

- Detector de objetos basado en “Template matching” [1, 7]: Traducido sería “correspondencia de patrón” y es una técnica para detectar dentro de una imagen, aquella ventana que más se parece a un “Template” o patrón dado, siendo este patrón el objeto que se quiere detectar. Se utiliza como parte del control de calidad en la industria, como una manera de navegar de un robot móvil, o como una forma de detectar bordes en imágenes.
- Detector de caras basado en Filtros de Haar + Adaboost [2, 7, 24] propuesto por Paul Viola y Michael Jones: El problema básico que intenta resolver es la detección de caras en una imagen.

- Algoritmos clasificadores: L-BFGS y L-BFGS-B [10, 12] son dos métodos de optimización de funciones con un gran número de parámetros. Permite obtener el mínimo de una función.
- Algoritmos clasificadores: Método del gradiente conjugado [11, 12] es un algoritmo para resolver numéricamente los sistemas de ecuaciones. El método del gradiente conjugado se utiliza para resolver los problemas de optimización sin restricciones como la minimización.

6.3 FUNCIONALIDAD DEL SISTEMA

El sistema deberá lograr la detección de personas dentro de una imagen. **Por lo tanto el input del sistema será una imagen y el output será esa misma imagen con un recuadro marcando el objeto detectado. Es decir, este sistema tendrá siempre una entrada y una salida. La entrada será una imagen y modelos del objeto que se quiere detectar. Y la salida será la misma imagen con una ventana que indica que se ha detectado el objeto indicado.** La base de datos con las imágenes modelos es pública y pertenece a **Inria, el Instituto Nacional Francés para la informática y las matemáticas aplicadas.** [8]

Por otra parte, para analizar las distintas técnicas, es necesario que el sistema procese un conjunto de imágenes y realice una estadística de los resultados obtenidos, con un modelo determinado, para así poder realizar una comparación sistémica entre los distintos modelos y así poder sacar una conclusión válida. Esto es el objetivo final del presente trabajo. Para realizar la mencionada comparación, el conjunto de imágenes posee un conjunto de archivos planos donde se establece la posición de los objetos a hallar.

6.3.1 INPUT DEL SISTEMA

6.3.1.1 BASE CON MODELOS (IMAGENES)

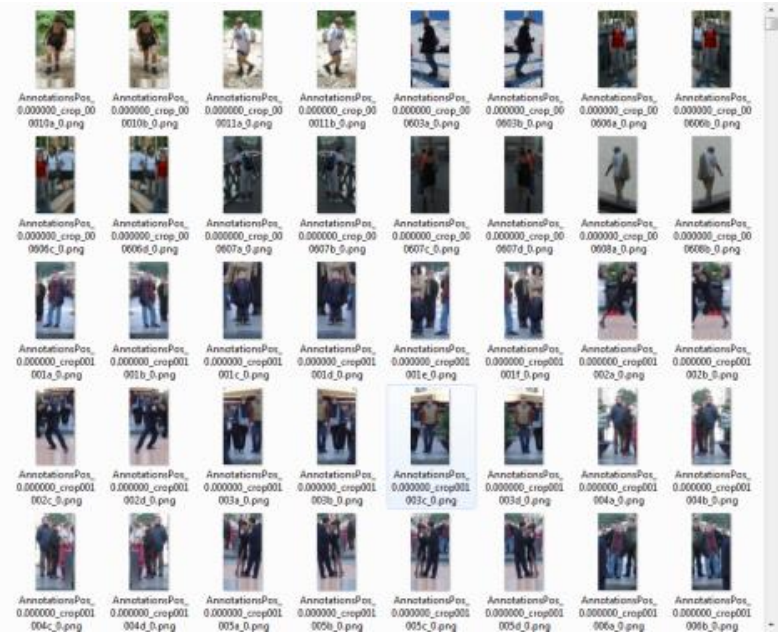


Ilustración 1: Base con Modelos [8, 17]

6.3.1.2 IMAGEN QUE INGRESA

Ilustración 2: Imagen que ingresa [17]

6.3.2 OUTPUT DEL SISTEMA

6.3.2.1 IMAGEN QUE SALE

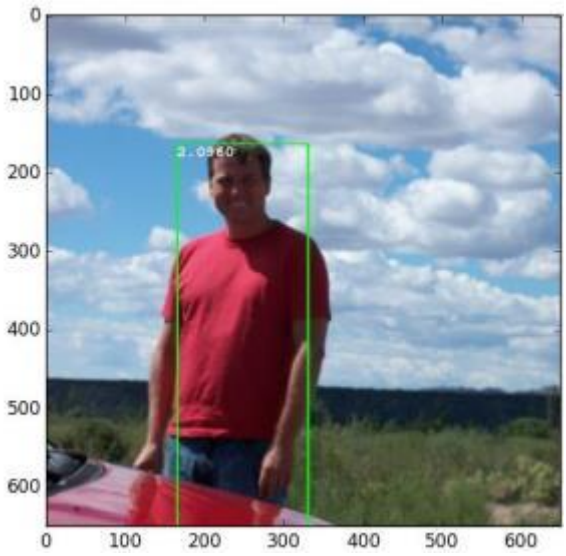


Ilustración 3: Imagen que sale [17]

El desarrollo teórico y de algoritmos representa un difícil desafío. Se hará uso de muchos algoritmos que ya están completamente desarrollados como módulos GNU, sin embargo, su comprensión, configuración y correcto uso implican un enorme desafío. Dichos algoritmos, estarán desarrollados en forma teórica, ya que se basan en conceptos matemáticos.

6.4 HERRAMIENTAS A SER USADAS

6.4.1 PYTHON

Para poder desarrollar este módulo es necesario usar herramientas de libre disponibilidad: Por una parte está el lenguaje de programación que mejor se adapta a los objetivos, que es Python. Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Es un lenguaje de programación multi-paradigma, ya que soporta orientación a objetos, programación imperativa y programación funcional.

Es administrado por la "Python Software Foundation" [18] y posee una licencia de código abierto, denominada Python Software Foundation License, compatible con la Licencia pública general de GNU.

6.4.2 PAQUETE PYTHON "SCIKIT-IMAGE" Y "SCIKIT-LEARN"

Incursionare en conceptos de formación de la imagen y por lo tanto es necesario usar rutinas de procesamiento de imágenes. Python cuenta con paquetes de libre acceso, (GNU), y para este proyecto, usare el paquete para Python "Scikit-image" y "Scikit-Learn". [19]

6.4.3 IDE PYCHARM

PyCharm es un entorno de desarrollo integrado (IDE) que se utiliza para la programación en Python. "PyCharm Community Edition" [20] se distribuye bajo licencia Apache. La licencia Apache es una licencia de software libre escrito por "Apache Software Foundation", (ASF). [21]

7 INTRODUCCIÓN

7.1 ESQUEMA GENERAL DE UN DETECTOR DE OBJETOS

Se puede plantear una secuencia de procesamiento concreta como la que se presenta en la ilustración 4. Como se observa allí, dentro de este esquema, se tiene una primera parte, donde se extraen las características y se luego se generan los candidatos: Cuando se entrena al sistema detector primero se generan los candidatos y luego se extraen las características; cuando el sistema intenta reconocer objetos, primero se extraen las características y luego se generan los candidatos.

Esos candidatos, con sus características, van a llegar a un módulo de clasificación, que va a decidir si los candidatos representan uno de los objetos buscados o no. En ocasiones, al hacer esa búsqueda de objetos en una imagen se tendrá incluso detecciones que son redundantes, y por lo tanto se tendrá que refinar el resultado.

Una vez que se tienen los resultados finales, se va a utilizar métodos de evaluación para saber si los modelos desarrollados están funcionando o no para la aplicación que se busca.

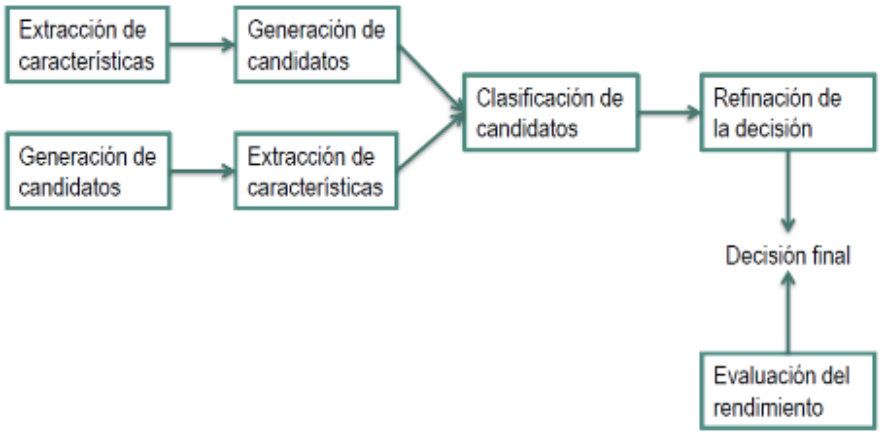


Ilustración 4: Esquema general de detección de objetos

7.2 EXTRACCIÓN DE CARACTERÍSTICAS: CÁLCULO DE DESCRIPTORES

Cuando se habla de extracción de características, en realidad se habla de extracción o cálculo de descriptores. Entonces en el contexto del aprendizaje automático se habla de lectores de características y dentro del contexto de la visión por computador se habla de descriptores de imagen o descriptores de ventana.

Con relación a estos descriptores de imagen, en este trabajo, se desarrollaran 3 técnicas conocidas:

- **LBP, "Local Binary Patterns"**[3, 7]
- **HOG, "Histograms of Oriented Gradients"**[4, 14, 7]
- **Combinación de ambas técnicas: HOG + LBP.**

7.3 GENERACIÓN DE CANDIDATOS

Se analizaran conceptos como el de la ventana deslizante, las pirámides y la eliminación de ventanas redundantes, todo esto en relación al concepto de búsqueda y refinación de objetos en imágenes, y se verá cómo evaluar los clasificadores y detectores de objetos que se diseñen.

Técnicas a desarrollar para la generación de candidatos:

- **Ventana deslizante (Sliding Window)** [9, 7]
- **Pirámide con ventana deslizante (Pyramidal Sliding Window).** [9, 7, 13]

7.4 CLASIFICACIÓN DE CANDIDATOS

Para evaluar a los candidatos se aplicaran los SVM, es decir las "Super Vector Machines", además se profundizara en el tema de la regresión logística, y en general el aprendizaje automático.

Técnicas a desarrollar para evaluar candidatos:

- **SVM** [5, 12, 7]
- **Regresión Logística** [6, 12, 7]

8 OBJETIVO GENERAL

El principal objetivo de este proyecto es desarrollar un detector de personas en entornos reales, de tal forma que no se centre únicamente en la detección de personas individuales, sino que pueda obtener resultados en distintos tipos de escenarios, con distintos grupos de personas y con oclusiones.

A modo de resumen se puede fijar el alcance y objetivos de este proyecto como sigue:

- Estudio del estado del arte en detección de personas y grupos de personas.
- Selección de los algoritmos base de detección de personas a usar en el presente trabajo.
- Selección de distintas combinaciones de los algoritmos base seleccionados, a los que llamaremos modelos.
- Implementación de los algoritmos base seleccionados en un detector de personas.
- Selección de una base de datos adecuada para la evaluación de los modelos propuestos.
- Definición de una métrica para la evaluación de los modelos.
- Exposición de los resultados de los modelos propuestos.
- Conclusiones del proyecto

9 OBJETIVOS ESPECÍFICOS

El objetivo específico del presente trabajo consiste en desarrollar un sistema de detección y reconocimiento de personas. Dicho sistema tendrá como salidas, (outputs), lo siguiente:

- Un estadístico que describen las *Curvas Tasa de error/FPPI*, (se ve más adelante), como forma de medir y comparar el detector de objetos.
- Un directorio con las imágenes de evaluación/validación, donde se identifican con una ventana roja el *Ground Truth*, (se ve más adelante), y con una ventana verde los hallazgos, (el objeto buscado).
- Cuenta con la opción de realizar la búsqueda en una única imagen. En este caso el sistema abre un archivo imagen con los hallazgos dentro de una ventana verde.

Dicho sistema debe contar con la posibilidad de seleccionar que método de *cálculo de descriptores* usa: **HOB, LBP, HOB+LBP.** Por otra parte deberá poder usar distintos métodos de *clasificación de candidatos*: **SVM y Regresión Logística.**

Además, implementara el método de **Pirámide con ventana deslizante,** a fin de realizar una *refinación de la decisión*. Determinar la base que se usara como conjunto de entrenamiento, con la que se entrenaran los distintos

modelos. Luego se determinara las *base de validación*, (se ve más adelante), y con ella se analizaran y evaluaran los resultados de cada una de las distintas combinaciones de las técnicas de *cálculo de descriptores y clasificación de candidatos*, dichas combinaciones son llamadas *modelos*.

Los modelos posibles son los siguientes:

1) LBP + SVM
2) LBP + Regresión Logística
3) HOB + SVM
4) HOB + Regresión Logística
5) HOB + LBP + SVM
6) HOB + LBP + Regresión Logística

Con ello se determinara el mejor *Factor de regularización,* (se ve más adelante), para cada uno de los modelos. Una vez logrado identificar los mejores factores de regularización, se puede con ellos, hacer una evaluación entre todos los modelos, usando la *base de evaluación*, (se ve más adelante). Finalmente el objetivo es llegar a una conclusión respecto de cuál es el modelo más apropiada para este caso, (la detección de personas). Así, entonces, por medio de este proceso estricto, se está en condiciones de llegar a conclusiones válidas.

10 MARCO TEÓRICO

Referencias [7], [8]

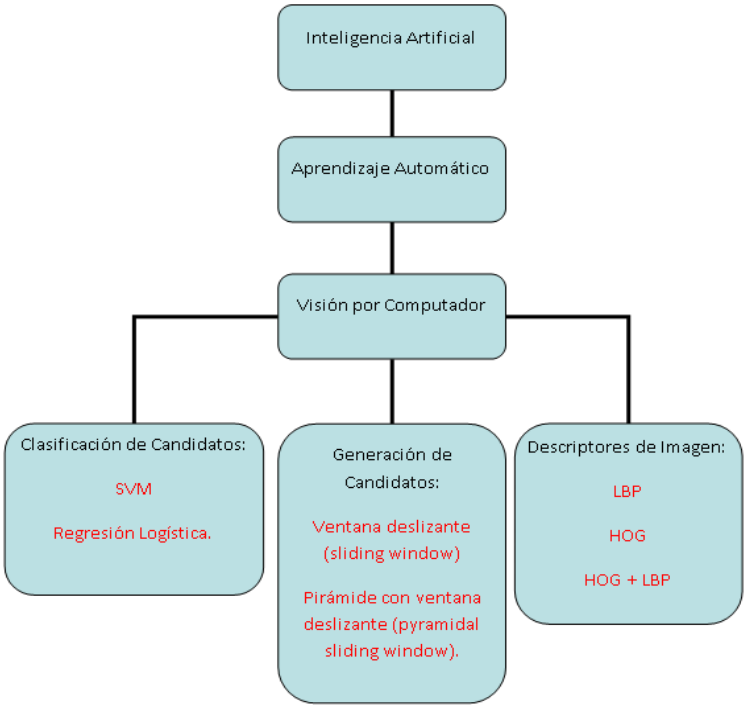


Ilustración 5: Esquema conceptual del marco teórico del trabajo

10.1 EXTRACCIÓN DE CARACTERÍSTICAS: DESCRIPTOR LOCAL BINARY PATTERNS

Dos ingredientes fundamentales de un clasificador son por un lado el descriptor de las ventanas que se han de clasificar y por otro lado la frontera en el espacio del descriptor que va a permitir distinguir los objetos de interés del resto. Se verá primero un descriptor muy concreto, conocido como Local Binary Patterns. Lo primero que se hace en este método es transformar la imagen color en una imagen con niveles de grises, en una escala de 0 a 255.

En la figuras 6 se ve a la izquierda una imagen en niveles de gris, en particular los niveles van de cero a 255. En esta imagen a la derecha, lo que se ve es el resultado, para cada uno de los pixeles de la imagen de la izquierda, de aplicar el método de Local Binary Patterns o abreviado LBP. Es decir, para cada uno de los pixeles que se tienen en la imagen en gris, se va a tener un código asociado de los posibles códigos que hay en LBP. Una particularidad de los LBP es que tiene distintas invariancias. Una de ellas es invariancia a cambios monotónicos de nivel de gris y otras es a la traslación. En la ilustración 7 se ve otro ejemplo de LBP con su histograma de los valores calculados LBP, para cada uno de sus pixeles, concepto que se desarrollara más adelante.

Invariancias:

- Cambios monotónicos del nivel de gris
- Traslación

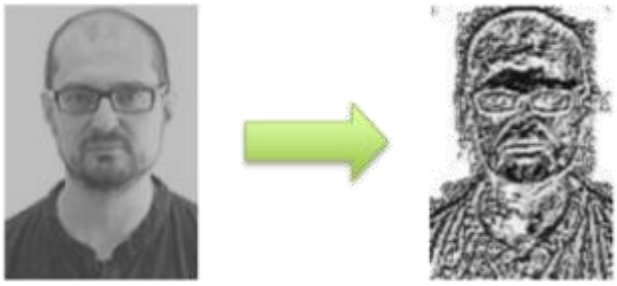


Ilustración 6: Invariancias en LBP

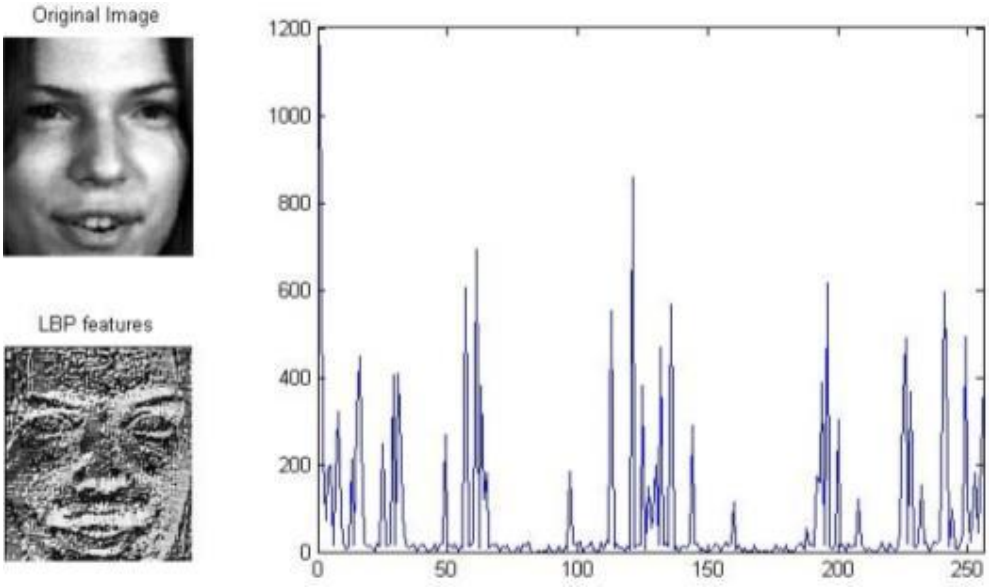


Ilustración 7 [29]: Ejemplo de LBP con histograma de los valores de los pixeles obtenidos

Se verá ahora cómo se calcula el código LBP para cada pixel de la imagen. Suponiendo que el cuadrado en la ilustración 8 es una imagen y cada uno de los cuadrados internos es un pixel, se va a considerar el cálculo del código LBP para este pixel central. Ese cálculo se basa en primero definir una vecindad del pixel y luego ir comparando los niveles de gris de este pixel central con el de los vecinos. Ver regla en ilustración 8.

En este caso como vecindad se ha definido a todos aquellos pixeles que tocan el pixel central, que serían todos los que le rodean. Los números que se ven en rojo, en tamaño pequeño, son los niveles de gris en este ejemplo. Bien, ahora lo que se tiene que hacer es, para cada pixel vecino se ha de aplicar la regla enunciada, (en la ilustración 8). Para eso se ha de escoger también un orden en relación a esos vecinos. El orden es arbitrario pero siempre se ha de respetar el mismo. En este ejemplo se ha decidido empezar por el pixel a la izquierda superior, luego continuar por su vecino de la derecha, luego el otro a la derecha y así sucesivamente. En este caso se ha de comparar el valor siete con el valor ocho. La regla dice que si el valor del nivel del gris del vecino es mayor o igual del nivel de gris del pixel central entonces a ese vecino se le asignara un uno y en otro caso se le asignara un cero. En este caso se asigna el valor uno. Así sucesivamente como se ve en la parte central de la ilustración 8. Como hay ocho vecinos, es un byte, donde cada bit tendrá un valor de cero o uno según el resultado de aplicar esta regla a cada uno de los vecinos. Como esto es un código binario, se puede traducir a un código decimal. En este caso el valor que se obtiene es 182. Por tanto, el código LBP de ese pixel central es 182.

Resumen de LBP Básico: El valor del bit es 1 si el valor del vecino >= valor central, y 0 en caso contrario. Es una forma de informar respecto de la relación de cada pixel con sus vecinos.

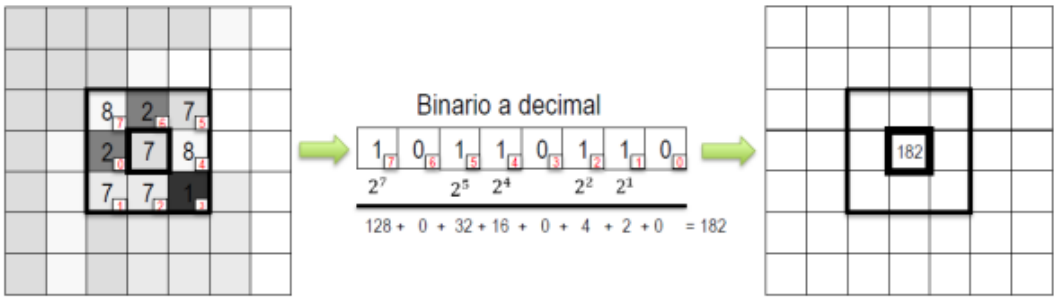


Ilustración 8: Cálculo de LBP, de binario a decimal

Se ve que se ha pasado de un valor en la imagen original de gris que va de cero a 255 a otro valor en la imagen de los LBP que en realidad también va de cero a 255. La diferencia lo que se codifica es la relación del nivel de gris que había en este pixel en la imagen original con el nivel de gris de los pixeles vecinos. Por tanto, la información es un poco más de alto nivel, no solo es información del propio pixel sino de cómo se relaciona con los pixeles vecinos.

Con ese algoritmo, dado una imagen de entrada, como se observan en las figuras 6 y 7, se puede producir una imagen de salida donde en cada pixel tiene un código LBP. En realidad lo que se hace es un barrido de la imagen calculando todos esos códigos LBP, ver ilustración 9. Una cuestión técnica es que en los bordes de la imagen hay problemas. Si se quiere calcular el valor LBP de una esquina entonces se puede tomar la decisión de extender en forma simétrica la imagen, solo para estos casos, o ignorar los bordes en el sentido que no se calculan los LBP para estos bordes.

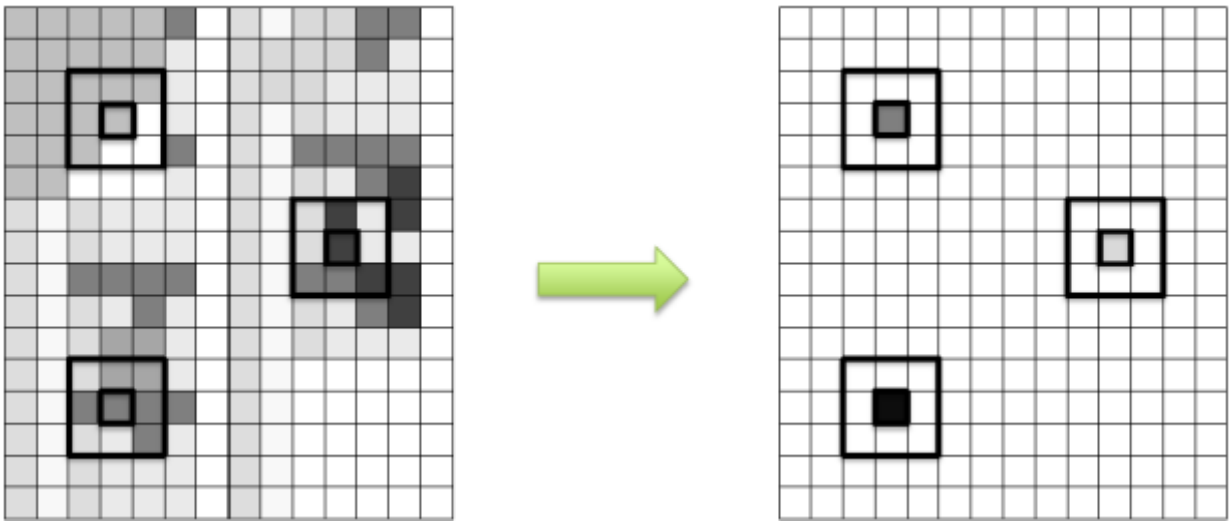


Ilustración 9: Transformación de una imagen calculando un valor LBP para cada pixel

10.1.1 LBP - IMÁGENES CON TEXTURA

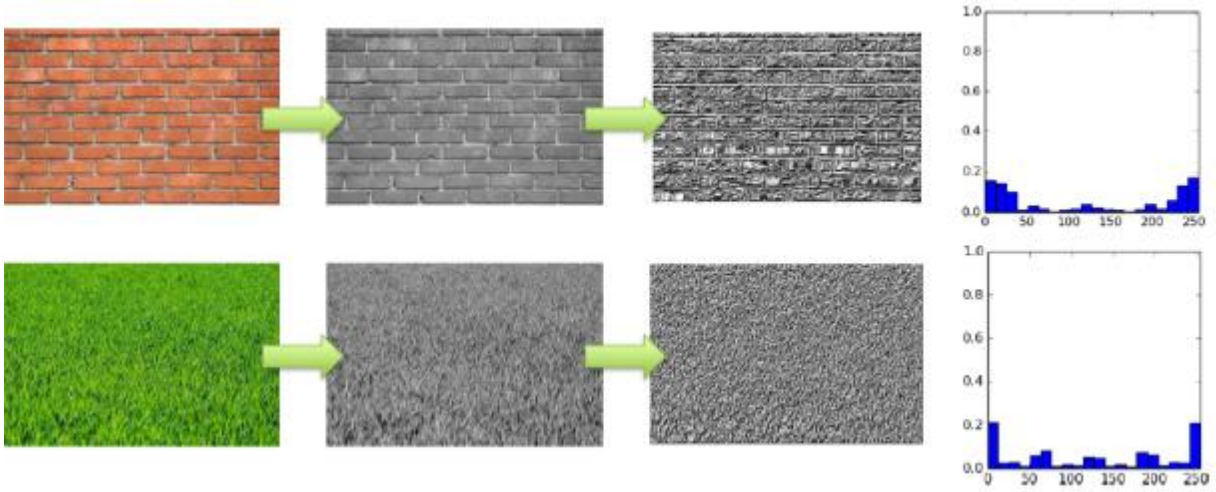


Ilustración 10: Transformación de imagines con textura y sus histogramas

Los Local Binary Patterns se definieron inicialmente para distinguir texturas, como por ejemplo las de la ilustración 10, una repetición de ladrillos y césped que tienen sus texturas particulares y sus histogramas característicos. Las dos primeras son imágenes de color como se puede ver, por tanto son imágenes con tres canales, el rojo, el verde y el azul, pero el LBP se define para imágenes en niveles de gris. Para arreglar este problema lo único que se hace es transformar las imágenes RGB en su correspondiente imagen de intensidad que es la que se ve en la segunda posición. Entonces a estas imágenes de intensidad se le aplica el LBP, como se ve en la tercera posición. Así, se pueden ver las imágenes LBP correspondientes a cada uno de los casos. Para describir estas imágenes lo que se hace es un histograma, un histograma normalizado, de las imágenes de códigos LBP. Por ejemplo en el primer histograma normalizado de esta imagen LBP se ve que ninguno llega al 20% pero hay bastantes pixeles con códigos LBP bajos y lo mismo pasa en el otro extremo, donde hay bastantes pixeles con código LBP alto. Se ve que cada

histograma tiene su forma particular. Entonces la idea es que esto es un descriptor de 256 dimensiones, es decir un espacio de 256 dimensiones. Por lo tanto se intenta poner un hiperplano que separe unos casos de otros.

10.1.2 LBP - INVARIANZAS: CAMBIOS DE NIVEL DE INTENSIDAD Y TRASLACIÓN

Ya se ha dicho que los LBP son invariantes a cambios monotónicos de nivel de gris y a traslaciones. Pero ¿Qué es un cambio de nivel de gris? En este caso se refiere a que se tiene unos niveles de gris de entrada, unos niveles de gris de salida y hay una transformación que lleva los niveles de gris de entrada a los de salida. Que esa transformación sea monotónica, lo que quiere decir es que tendrá un aspecto como el que se observa en las dos primeras imágenes de la ilustración 11, que pueden tener un nivel de gris distinto pero que obtienen la misma trasformación a LBP. Lo que sucede es que la diferencia entre de dos pixeles a y b pueden cambiar respecto a los mismos pixeles de la otra imagen, pero lo que no cambia es el orden relativo, por lo que si a es menor que b en una imagen, en la otra imagen sigue siendo a menor que b. Por tanto, como los Local Binary Patterns se basan en comparaciones entre niveles de gris, el local Binary Pattern de una imagen y el de la otra serán exactamente el mismo.

La invariancia respecto a la traslación se refiere que dado un marco como en la parte baja de la ilustración 11, si se desplaza el contenido, en realidad lo que le sucede a los local Binary Patterns de ese marco, es que simplemente se trasladan en una cantidad igual a la imagen original. O sea que estos patrones que son locales, no se ven afectados por este tipo de transformaciones globales en los ejes de coordenadas. Lo histogramas se mantienen iguales

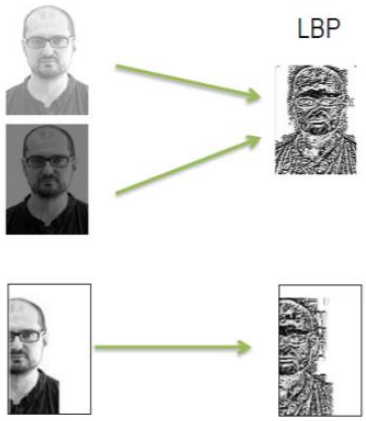


Ilustración 11: Invariancias en cambios monotónicos del nivel de gris y en translación en una imagen.

10.1.3 LBP - UNIFORME: VARIANTES DE LOCAL BINARY PATTERNS

Existen variantes sobre el concepto básico de LBP. La primera variante viene de la definición de vecindad. Para calcular el código LBP de un cierto pixel, se tiene que calcular o comparar el nivel de gris de ese pixel con el de sus vecinos. Se ha dicho que los vecinos son todos aquellos pixeles que tocan un determinado pixel, teniendo ocho vecinos. Sin embargo, se puede considerar otras vecindades donde las vecindades son circunferencias de un cierto radio r, ver ilustración 12. En esa circunferencia, a lo largo de ella se puede considerar más puntos o menos puntos. Ya sé hablo de como calcular el código LBP de un cierto pixel, donde cada uno de los vecinos ha de contribuir en un bit dentro del código LBP. Esto se puede reescribir y poner en esta comparación, un cierto umbral T, (threshold). Con esto se logra robustez a pequeñas variaciones en el nivel de gris debidas a ruido en la captura de la imagen.

Valor bit b = 1 si (valor pixel vecino – valor pixel central) >= T

Valor bit b = 0 en caso contrario

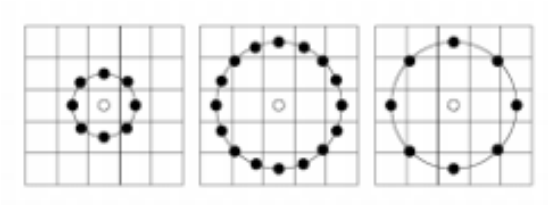


Ilustración 12: Variantes en la definición de vecindad en LBP

Otra variante de los códigos LBP corresponde al llamado código LBP uniforme. Hasta aquí se ha trabajado con códigos LBP cuyo valor va de 0 a 255, es decir que hay 256 posibles valores. Sin embargo, cuando se calculan códigos LBP en imágenes reales, hay muchos patrones que tienen muy poca probabilidad de darse, y por eso se decide reducir el número de patrones a solo aquellos que sí aparecen con regularidad y por tanto son informativos. Para ver cómo se obtienen esos patrones, se define una medida de uniformidad U, como el número de transiciones de cero a uno o de uno a cero en la representación binaria del código LBP: Véase ilustración 13.

Por ejemplo, en el primer cuadrado de la ilustración 13, cada uno de estos unos quiere decir que el nivel de gris del vecino era mayor o igual que el nivel del pixel central, en este caso se ve que no hay ninguna transición de tipo cero uno ni de tipo uno cero, por tanto U es igual a cero. Por otra parte, en el segundo cuadrado, cuando se tiene un cero quiere decir que el nivel de gris del pixel central era mayor que el del vecino, en este caso hay varias transiciones, se tiene que seguir el sentido anti horario y por tanto, hay dos transiciones. Dando finalmente un número de transiciones U igual a 2. Siguiendo el mismo procedimiento para el último cuadrado se tienen un número de transiciones U igual a cuatro.

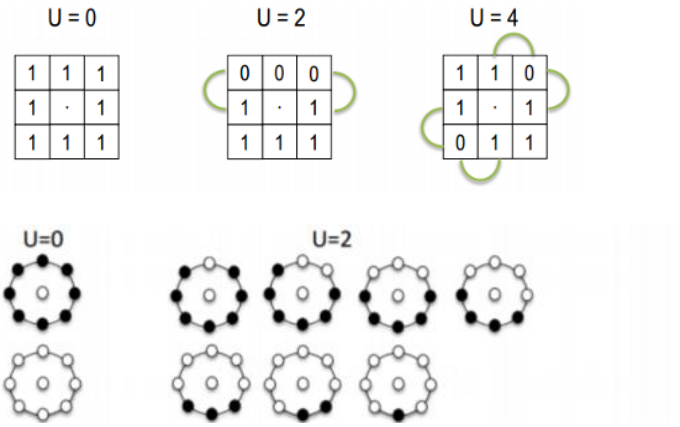


Ilustración 13: Medida de uniformidad

Los patrones uniformes se definen de la siguiente manera. A los patrones cuyo valor de U es dos o cero, simplemente se les reasigna un código nuevo, el código es indiferente siempre que se mantenga el mismo criterio.

Todos los patrones cuyo U no es ni cero ni dos, se les asigna el mismo código. También es un código arbitrario, pero es el mismo para todos. Por tanto, se pasa de 256 patrones en el LBP normal a 59 patrones del LBP uniforme.

Estos 58 salen de todos aquellos que tienen valor U igual a cero o U igual a dos y el patrón restante es el código que se ha reservado para el resto de patrones, es decir aquellos cuyo valor de U no era ni cero ni dos. En la ilustración 14 se ven como se forman los 58 patrones.

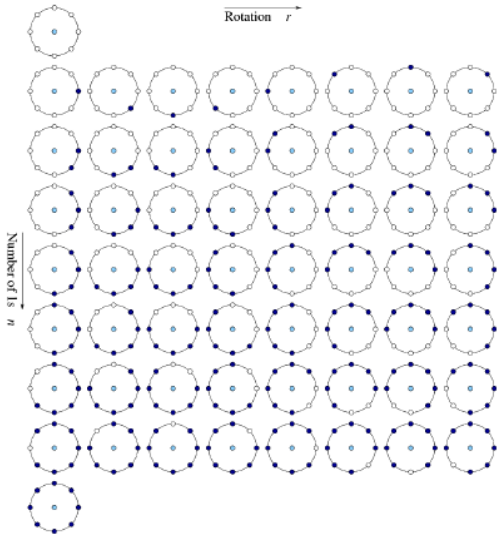


Ilustración 14 [26]: 58 Patrones correspondiente a U=0 y U=2

En la ilustración 15 se ve un ejemplo visual de la diferencia entre el código LBP visto hasta ahora y el LBP uniforme a partir de la misma imagen, con los correspondientes histogramas. Nótese el corrimiento a la izquierda del histograma del LBP uniforme.

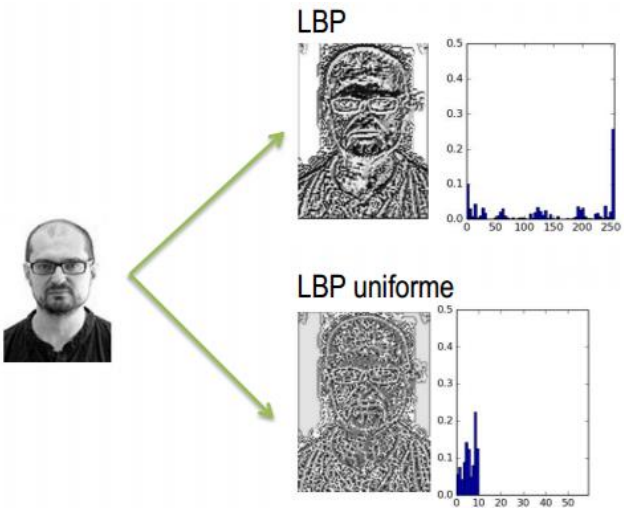


Ilustración 15: Comparación de histogramas entre LBP y LBP uniforme

10.1.4 LBP - HISTOGRAMA POR BLOQUES

Se han visto los LBP y los LBP uniformes y cómo se podían calcular para cada uno de ellos, los pixeles de una imagen. Ahora se verá cómo se utilizan esos códigos para definir descriptores de ventanas que posteriormente se utilizarán para clasificar las mismas.

10.1.4.1 EL PROBLEMA DE USAR DIRECTAMENTE EL CÓDIGO LBP COMO DESCRIPTOR

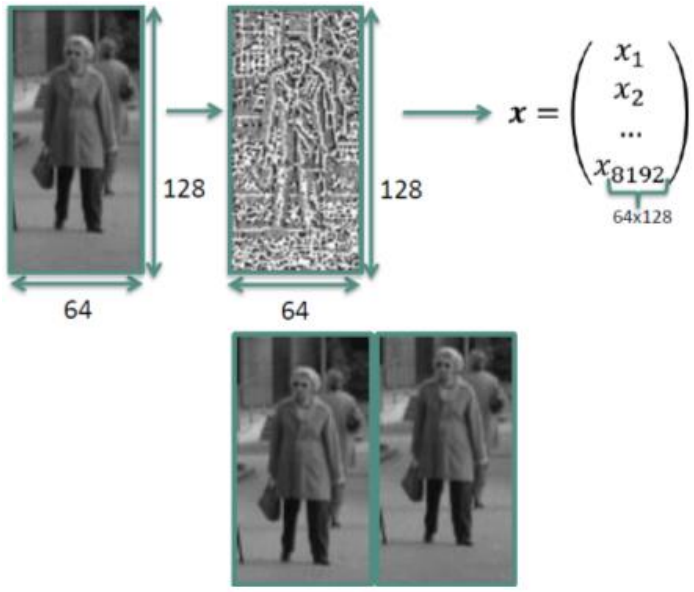


Ilustración 16: El alineamiento del objeto en la ventana es crítico, ya que cambia el vector de características x

En la ilustración 16 se tiene una ventana de 64 columnas y 128 filas y su correspondiente con los códigos LBP uniformes.

Se podría definir el descriptor x de la ventana mediante una serie de componentes, donde cada uno de esos componentes es un código LBP. Por ejemplo, el primer componente podría ser el código LBP del pixel de la esquina superior izquierda de la ventana y la última componente, el código LBP del pixel de la esquina inferior derecha de la misma ventana. Los componentes que hay entre estos dos pixeles simplemente saldrían de ir viajando por la ventana de códigos LBP. Como esta ventana es de 64 por 128 pixeles, la dimensión del descriptor es de 8192. El problema de este descriptor es el siguiente: en la esquina inferior derecha de la ilustración 16 se tienen dos ventanas que son prácticamente iguales, pero cada una tiene su vector. El contenido de las figuras es el mismo, tienen un objeto que interesa, que es una persona. Ahora se analizara que pasa cuando se toma una porción de la imagen en una forma semánticamente relevante del objeto que interesa: De la persona, por ejemplo, la cabeza. Los códigos LBP de esta zona irán a parar a cierta zona del primer descriptor y los códigos LBP de la otra imagen irán a parar a otra zona de descriptor de esa ventana. Los códigos en sí, son los mismos, porque los códigos LBP son invariantes a traslaciones, sin embargo su posición dentro de los descriptores es diferente y esto hace que estos descriptores en general sean bastante diferentes, cosa que no interesa porque el contenido de las imágenes es básicamente el mismo. Para resolver esto se suelen utilizar histogramas.

10.1.4.2 LBP UNIFORME: USO DE HISTOGRAMAS

Como se usa el LBP uniforme, la dimensión del histograma es de 59, es decir hay 59 entradas y por tanto el descriptor en lugar de tener 8192 dimensiones tiene solo 59. Así lo que interesa es que los descriptores o los histogramas, que corresponden a ventanas con personas sean muy diferentes al resto de ventanas que contienen

simplemente fondo y no contienen personas. Si estos histogramas entre sí son parecidos o no, da igual, lo importante es que sean diferentes de aquellos que corresponden a ventanas que sí contienen personas.

Y que si se tiene otra ventana con otra persona diferente, se quiere que su correspondiente histograma de códigos LBP uniforme sea muy similar al de las otras personas. No Igual, porque el fondo en el que esta persona aparece será distinto del resto y porque la ropa será distinta, etcétera, pero debe ser lo más parecido posible. Ver ilustración 17

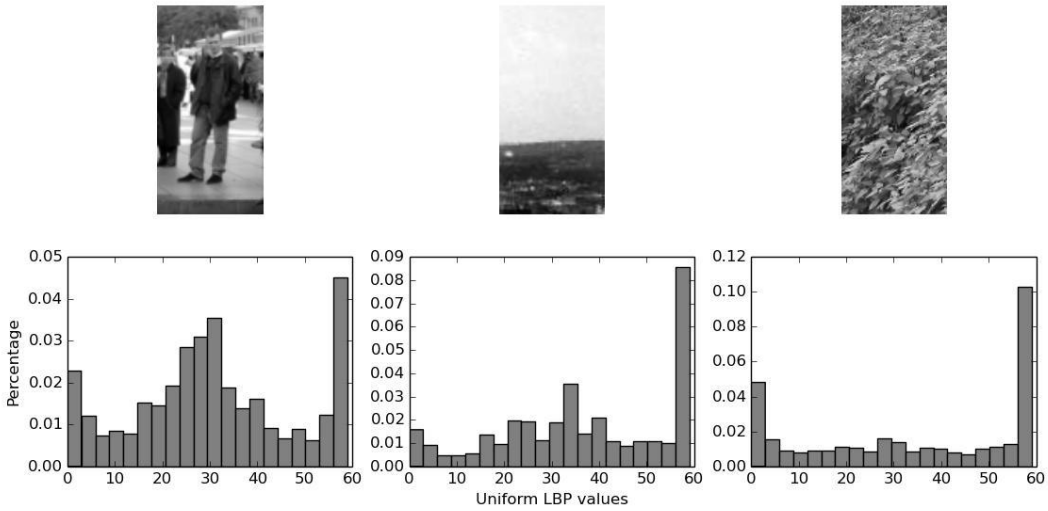


Ilustración 17: Histogramas de imágenes con personas y fondo

10.1.4.3 PROBLEMA DE USO DIRECTO DEL HISTOGRAMA DEL LBP

El problema ahora es que este descriptor en realidad no es suficientemente discriminativo. En la ilustración 18 se muestra una ventana que se la corta en medio y la parte de arriba se sitúa en esta nueva ventana en su parte de abajo, y la parte de abajo de la original, en la parte de arriba de la nueva.

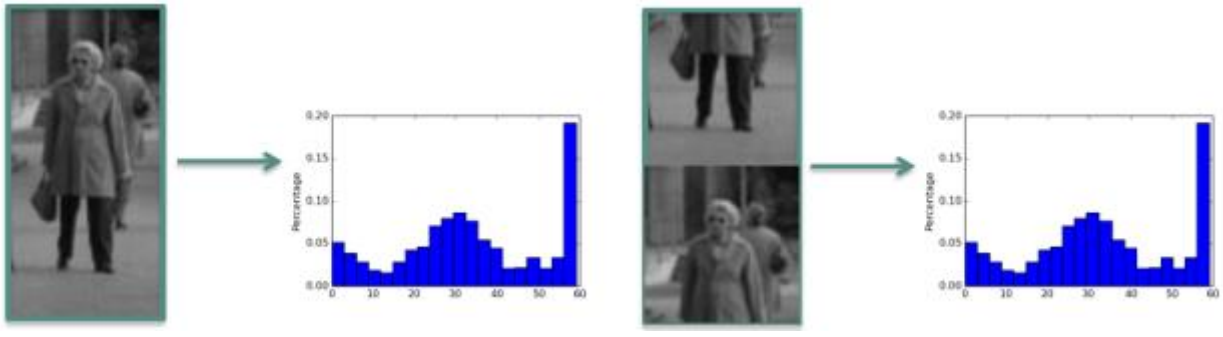


Ilustración 18: Problema de dos imágenes distintas que generan el mismo histograma

Como los LBP son invariantes a traslaciones los códigos LBP son iguales. Por tanto, los únicos códigos nuevos de la ventana son los que hay en la línea divisoria, pero estos son muy pocos porque el número de pixeles que hay ahí es menor que el uno por ciento del total de la ventana. Eso quiere decir que ambos descriptores son muy parecidos, y por tanto no es un buen descriptor porque estaría diciendo que ambas ventanas contienen exactamente lo mismo.

Para resolver este problema lo que se hace es dividir la ventana en bloques.

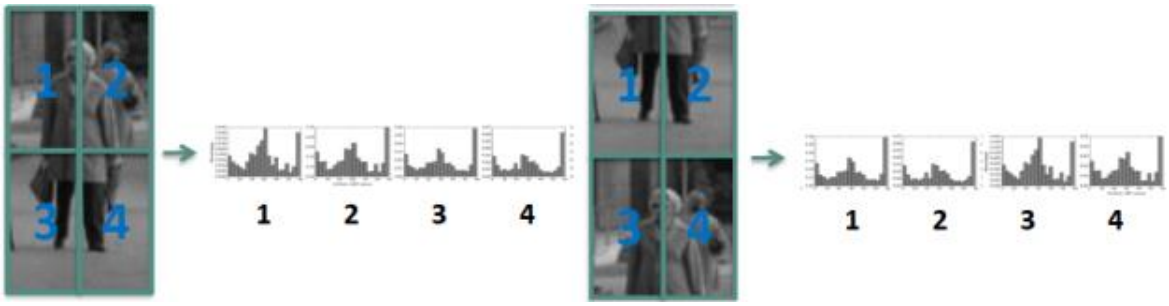


Ilustración 19: División de la ventana en bloques, para lograr mayor robustez ante variaciones locales en la ventana

En la ilustración 19 se divide la ventana en cuatro bloques y se hace un histograma para cada uno de los bloques por separado. Entonces el descriptor de toda la ventana consiste en concatenar todos estos histogramas. Por tanto, en este caso si se tiene n bloques, la dimensión del descriptor de la ventana es n por 59. En este caso lo que sucede es que si se hace lo mismo para la ventana que se ha creado artificialmente, se puede ver que a la hora de comparar la zona del bloque uno con la zona del bloque uno de la ventana anterior, estas componentes del descriptor entre una y otra ventana son muy diferentes porque ahora sí que corresponden a zonas muy diferentes de la persona. Por tanto ahora, el descriptor de la ventana izquierda y el descriptor de la ventana derecha son diferentes. Hasta aquí para simplificar se habló de histogramas. Pero en realidad hay que aclarar que son histogramas normalizados, (ver ilustración 20), es decir cada entrada tiene un valor entre cero y uno. Y la suma de todas las entradas del histograma darán el valor uno.

En la siguiente ilustración se observa un histograma LBP-U sin normalizar y luego de ser normalizado

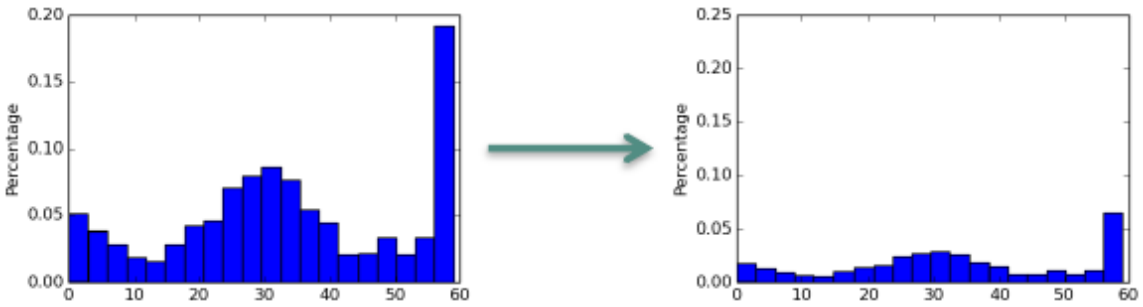


Ilustración 20: Normalización de un histograma

Cuando se tiene una ventana dividida en bloques, esa normalización se hace individualmente para cada bloque, hasta aquí se ha visto primero que para describir una ventana se usa un descriptor de 8192 dimensiones, luego otro de 59. La problemática del primero es que sus componentes son muy locales, es decir cada componente tenía información de cada pixel de la ventana. Y en el otro caso era un poco en sentido contrario, que era demasiado global a la ventana el descriptor, es decir en realidad el histograma no cambiaba mucho y por tanto ventanas que teniendo un contenido visualmente diferente tenían el mismo descriptor, cosa que es indeseada.

Como solución se ha introducido los bloques donde ahora el descriptor tendrá una dimensión de n por 59. Podría parecer que estos bloques tienen que ser disjuntos, pero en realidad para tener descriptores más robustos se

añade cierta redundancia, para ganar robustez. Lo que quiere decirse con esa redundancia es que se puede tener bloques solapados.

Ilustración 21: Solapamiento de bloques

Por ejemplo, para una rejilla de 64 por 128 pixel, el tipo de solapamiento que se usa es poner bloques que se van distribuyendo dentro de la ventana con un paso de ocho pixeles tanto en el eje x como en el eje y por lo tanto se llega a 105 bloques de 16 por 16 pixeles cada uno, lo cual da un descriptor de 6195 dimensiones, (59*105).

10.2 EXTRACCIÓN DE CARACTERÍSTICAS: DESCRIPTOR HOG (HISTOGRAM OFORIENTED GRADIENTS)

A diferencia de LBP, que es un descriptor que se basa en información de textura de la imagen, HOG es un descriptor que, que explota la información del gradiente de la imagen.

Características:

- Utiliza información del gradiente de la imagen, contornos de los objetos.
- Agregación de la información de la orientación del gradiente en histogramas locales obtenidos en celdas distribuidas por toda la imagen.
- Combinación de los histogramas en bloques de forma similar a LBP

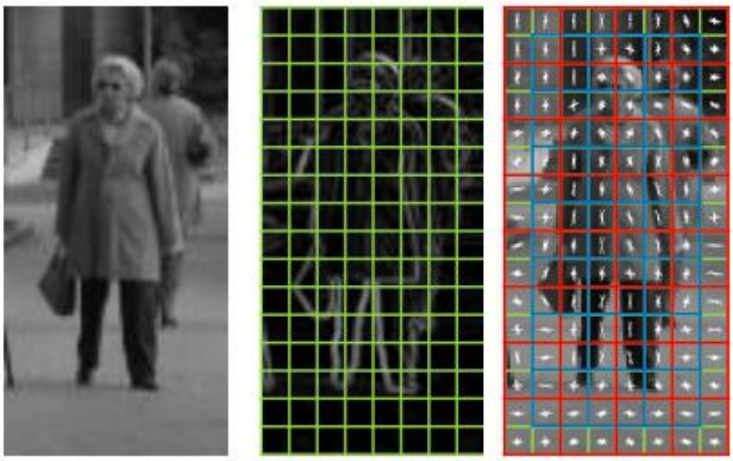


Ilustración 22: Descriptor HOG

Como se puede ver en la segunda imagen, (ilustración 22), el gradiente aporta información que puede ser relevante tanto para la detección como el reconocimiento de los objetos ya que valores altos del gradiente se corresponden con los contornos o silueta de los objetos. El descriptor HOG permite aprovechar de forma bastante eficiente la información del gradiente y combinar esta información en forma de histogramas locales que se calculan en celdas de pequeño tamaño que se distribuyen de forma uniforme por toda la imagen como se puede ver en la tercera imagen, (ilustración 22). También allí se puede ver un ejemplo de los histogramas, donde para cada una de las celdas se proporciona información de las orientaciones de los contornos que dominan cada una de las posiciones de la imagen. Así, se puede ver, por ejemplo, que en las piernas las orientaciones dominantes son las verticales y en cambio en la cartera de la mujer, la orientación dominante es la diagonal. Esta información permite distinguir la forma de los objetos presentes en una imagen y es una buena base para detectar y reconocer los objetos. Como último paso y de forma muy similar a lo que se realizaba en LBP, los histogramas que se calculan localmente en diferentes posiciones de la imagen y se agrupan en bloques de un tamaño un poco mayor. Estos bloques sirven para normalizar la representación final y para hacerla más invariante a cambios de iluminación y a distorsiones en la imagen. La representación final será la concatenación de la representación de todos estos bloques.

10.2.1 HOG - CÁLCULO DEL GRADIENTE

HOG es un descriptor de la imagen que utiliza el gradiente en cada uno de los píxeles como información básica. Primero se verá el concepto de gradiente y cómo se calcula el gradiente en el marco del HOG. El gradiente se define como un cambio de intensidad de la imagen en una cierta dirección, aquella dirección en la que el cambio de intensidad es máximo. Se calcula para cada uno de los píxeles de la imagen y queda definido para cada píxel dos valores, la dirección donde el cambio de intensidad es máximo y la magnitud del cambio en esa dirección. Así para cada píxel, estos dos valores permiten distinguir distintas situaciones de la configuración local alrededor del píxel en relación al cambio de contraste y a la forma local.

Por ejemplo, si se toman estas tres imágenes, y se analiza el píxel central de cada una de ellas.

Ilustración 23: Distintas direcciones de los gradientes

En la imagen de la izquierda, el gradiente tiene una dirección vertical porque es donde se produce el mayor cambio de intensidad y en la magnitud es elevada ya que el contraste también es elevado. En la imagen de la derecha, el contraste es menor. En este píxel el gradiente tendrá la misma orientación, sin embargo su magnitud será menor ya que el contraste también es menor. En la imagen central la orientación del gradiente es horizontal y su magnitud equivalente a la de la primera imagen.

El gradiente se puede calcular de varias formas diferentes. En el contexto del descriptor HOG este cálculo se realiza a partir de la diferencia de intensidad de los píxeles vecinos en dirección tanto horizontal como vertical.

Así, si se tiene una imagen, (ilustración 24), donde se representa para cada punto el valor de intensidad, el nivel de gris, y se quiere calcular el gradiente en un determinado punto x, este se calcula como la diferencia en la dirección horizontal de x, que se puede calcular como la diferencia de intensidad entre el píxel situado a la derecha de x, más uno menos el píxel situado a la izquierda x menos uno. Igualmente se puede calcular la diferencia en la dirección vertical como la diferencia en la intensidad entre el píxel situado arriba y el píxel situado debajo y menos uno.

$$dx = I(x+1,y) - I(x-1,y)$$
$$dy = I(x,y+1) + I(x,y-1)$$

0	0	255	255	255
0	0	255	255	255
0	0	0	255	255
0	0	0	0	0
0	0	0	0	0

Ilustración 24: Cálculo HOG, intensidades del nivel de gris

Es decir entre el píxel central, (0) y el píxel a la izquierda, la diferencia da 255. Y entre ese mismo píxel y el de arriba, da también 255. Estas diferencias en horizontal y en vertical se pueden calcular para todos los píxeles de la imagen y cada píxel dará información de la situación local alrededor del mismo.

A partir de las diferencias en horizontal y en vertical se puede calcular la orientación y la magnitud global del gradiente. Los valores dx y dy en el eje de coordenadas permiten definir al vector gradiente. La orientación es el ángulo que forma este vector con el eje horizontal y se puede calcular utilizando el arco tangente del cociente entre la diferencia en vertical y la diferencia en horizontal, mientras que la magnitud del gradiente es la longitud del vector. Se calcula la orientación y la magnitud del gradiente para cada uno de los píxeles de la imagen. Y esto da una información global de la imagen como se puede ver en este ejemplo, (ilustración 25).

Ilustración 25: HOG, gradiente

Combinando las dos magnitudes, de las direcciones de cada eje, se obtiene la magnitud global del gradiente para cada uno de los píxeles, en la cuarta imagen. Dan valores altos en todos aquellos píxeles de la imagen donde hay un alto cambio de intensidad en cualquier dirección. Básicamente estos altos cambios de intensidad se concentran alrededor de la silueta de la persona y dan información acerca de la forma de la persona que permitirá distinguirla de otros objetos de la imagen. Aquí solo se ha visualizado la magnitud del gradiente pero también se utiliza la orientación del gradiente en cada uno de los píxeles.

Hasta aquí se ha considerado que se trabaja siempre con imágenes en niveles de gris y por lo tanto el cálculo de diferencias es en relación a la intensidad en niveles de gris. Si se tienen imágenes en color, en el descriptor HOG el color se trata priorizando aquel color que domina para cada uno de los píxeles. Esto se consigue calculando para cada uno de los píxeles de la imagen el gradiente en cada uno de los tres canales de color, el rojo, el verde y el azul. Una vez calculados los tres canales por separado, se toma para cada píxel aquel gradiente de aquel canal que tiene una magnitud mayor. De esta forma se prioriza localmente el color que domina alrededor de un determinado píxel.

10.2.2 HOG - CÁLCULO DE LOS HISTOGRAMAS

La información del gradiente local a nivel de cada uno de los píxeles se puede agregar en forma de histogramas calculados en diferentes áreas de la imagen. El gradiente permite obtener información valiosa sobre la forma de un objeto, porque valores altos en la magnitud del gradiente se corresponden con el contorno de los objetos.

Mientras que la orientación proporciona información sobre la forma del contorno, de una manera que además es invariante a la localización del objeto en la imagen. El problema con esta representación es que puede ser muy sensible a pequeñas variaciones tanto en la forma como en la localización del objeto en la imagen.

Para solucionar esto, el descriptor HOG actúa en dos pasos. En primer lugar divide la imagen en un número fijo de celdas y para cada una de estas celdas se obtiene un histograma de las orientaciones de los gradientes. En un segundo paso se calculan los histogramas para todas las celdas y todos estos histogramas se combinan para obtener el vector de características, (ilustración 26).

Ilustración 26: HOG, división de la imagen en un número fijo de celdas

El primer parámetro a fijar es el tamaño de la celda, valores entre seis y ocho píxeles tanto en ancho como en alto suelen ser valores habituales. Otro aspecto a considerar es cómo se divide el rango de orientaciones en un número de intervalos fijo. Se considera la orientación del gradiente sin signo y de esta forma el rango de orientaciones va desde cero hasta 180. Con esta opción, dos gradientes con la misma dirección pero sentidos inversos se consideran equivalentes y quedan asignados al mismo intervalo.

Entonces para el para el cálculo del histograma de orientaciones en una celda, primero se divide el rango de orientaciones en un número de intervalos fijo. Luego se asigna a cada pixel de la celda a un intervalo en función de la orientación del gradiente y se acumula la magnitud del gradiente de todos los píxeles asignados a un intervalo.

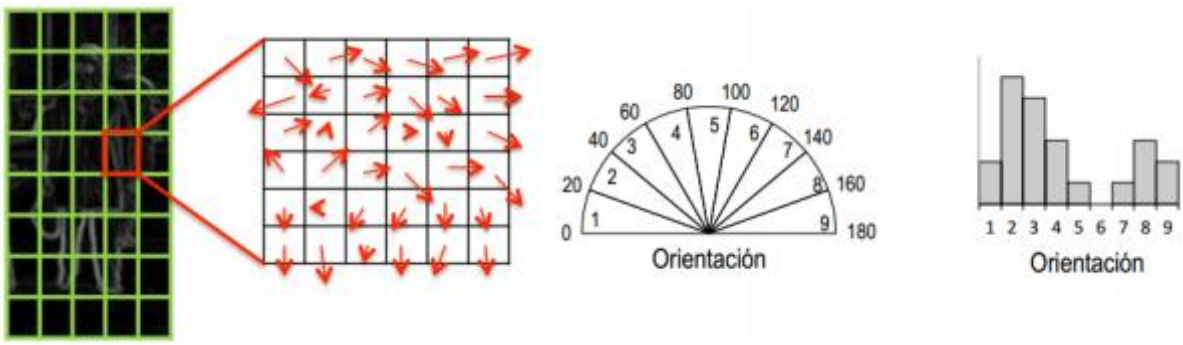


Ilustración 27: HOG, cálculo del histograma de orientaciones

Un parámetro que hay que fijar es en cuántos intervalos se dividen el rango de orientaciones. Aquí se usara una división en nueve intervalos, que suele ser también un valor bastante habitual. De esta forma, cada intervalo agrupa un rango de orientaciones de 20 grados. De todas formas se puede generalizar fácilmente a cualquier combinación de tamaño y de rango de orientaciones. Fijados los parámetros para el cálculo del histograma, cada uno de los gradientes de las celdas, cada uno de ellos con su orientación y su magnitud, quedará asignado a uno de los intervalos en función de la orientación del gradiente. De esta forma, cada intervalo acumula gradientes con una orientación dentro de los límites de ese intervalo. Finalmente, el valor de uno de estos intervalos en el histograma se obtiene acumulando la magnitud de todos estos gradientes asignados al intervalo.

Este cálculo del histograma se puede formalizar matemáticamente con la siguiente expresión que se tiene aquí.

$$w_k(x,y) = 1 \rightarrow (k-1)\partial\theta \leq \theta(x,y) < k\partial\theta$$

Caso Contrario: $w_k(x,y) = 0$

$$h(k) = \sum_{(x,y)\in C} w_k(x,y)g(x,y)$$

Para calcular el valor del histograma en una determinada posición k, este valor se obtiene como la suma, la acumulación de la magnitud de los gradientes ponderado por un factor que determina la asociación del gradiente con dicho intervalo k.

En un primer escenario simplificado, este factor de asignación se definirá de forma que para todos los gradientes valga uno, mientras que para todas aquellas orientaciones que estén fuera de los límites del intervalo, vale cero.

Este modelo de cálculo de los histogramas de orientaciones, parte del principio de asignar cada gradiente a un único intervalo. Es un modelo simple que puede presentar ciertos problemas: Gradientes con orientaciones muy similares pueden asignarse a intervalos diferentes y es sensible a pequeñas variaciones del gradiente. La solución es asignar cada gradiente a los dos intervalos más cercanos con un peso proporcional a la distancia de la orientación al centro de cada intervalo, según las siguientes formulas:

$$w_k(x,y) = \max(0, 1 - \frac{|\theta(x,y) - \theta_k|}{\partial\theta}$$

$$h(k) = \sum_{(x,y)\in C} w_k(x,y)g(x,y)$$

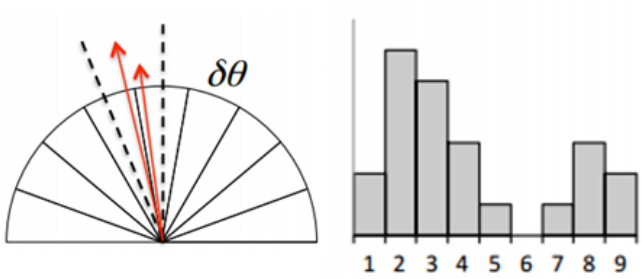


Ilustración 28: Interpolación en orientación

En la ilustración 28 se ve que gradientes con orientaciones muy similares han quedado asignados a intervalos diferentes. Esto puede provocar que pequeñas variaciones en la imagen de entrada acaben provocando variaciones significativas en la representación final. Entonces, como se dijo, la solución consiste en asignar cada píxel a los dos intervalos más cercanos, con un peso proporcional a la distancia de la orientación del gradiente a cada uno de los intervalos. Así, para asignar un gradiente a un determinado intervalo k se saca la distancia entre la orientación del gradiente y la orientación del centro del intervalo. Esta distancia se normaliza por el rango de cada uno de los intervalos del histograma. De esta forma, si la distancia es cercana a cero el factor de asignación será cercano a uno, mientras que si la distancia es cercana al rango del intervalo el factor de asignación, entonces, se acercará a cero. Por otro lado, la utilización del máximo en la fórmula hace que para distancias que sean mayores al rango del intervalo el factor de ponderación sea cero. Finalmente, este factor de ponderación se utiliza igual que antes para acumular la magnitud de todos los gradientes de la celda.

Con esta nueva formulación, cada gradiente contribuye no a un único intervalo sino a los dos intervalos más cercanos en función de su orientación. El cálculo del histograma se repite para todas las celdas en que se divide la imagen, de forma que para cada una de las celdas se tiene su correspondiente histograma.

Sin embargo, este esquema, puede presentar otro tipo de problema. Se ve en la ilustración 29 que dos píxeles muy cercanos en la imagen, pueden quedar asignados a celdas diferentes. Por lo tanto, pequeños cambios en la forma o en la localización del objeto pueden dar lugar a variaciones significativas en la representación final, porque puntos que son equivalentes en el contorno del objeto contribuirán a histogramas diferentes.

$$w_{ij}^{x}(x,y) = \max(0,1-\frac{d_{ij}^{x}}{\partial x})$$

$$w_{ij}^{y}(x,y) = \max(0,1-\frac{d_{ij}^{y}}{\partial y})$$

$$h_{ij}(k) = \sum_{(x,y)} w_{ij}^{x}(x,y) w_{ij}^{y}(x,y) w_{k}(x,y) g(x,y)$$

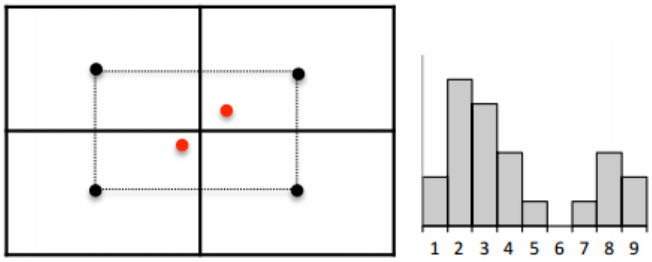


Ilustración 29: Interpolación espacial

Para solucionarlo, se utiliza la siguiente estrategia de interpolación. Cada píxel se asignará a las cuatro celdas más cercanas con un peso proporcional a la distancia del píxel al centro de la celda. Así se calcula para cada uno de los píxeles, la distancia de este píxel al centro de la celda en dirección x, y la distancia al centro de la celda en la dirección y. Estas distancias, de forma muy parecida a lo que se ha hecho con la orientación, se utilizan para calcular el factor de asignación de cada uno de los píxeles de las celdas normalizados por la distancia entre los centros de dos celdas, tanto en la dirección x como en la dirección y.

Estos factores de ponderación de asignación de un píxel a una celda en las direcciones x e y, se combinan con el factor de asignación de cada gradiente a cada uno de los intervalos del histograma y se obtiene la ponderación final que se aplica a cada una de las magnitudes de los gradientes de todos los píxeles de la imagen, para acabar obteniendo el valor concreto del histograma para cada uno de los intervalos.

10.2.3 HOG - CÁLCULO DEL DESCRIPTOR

El descriptor se va a basar en normalizar y agrupar los histogramas en forma de bloques de manera bastante similar a cómo se realiza en el descriptor LBP. El punto de partida para obtener la representación final del descriptor son los histogramas de orientación del gradiente, que se calculan para cada una de las celdas en las que se divide la imagen. Uno de los objetivos de cualquier descriptor debe ser conseguir la máxima invariancia posible a todas aquellas variaciones que se pueden producir en la imagen de entrada, sean de iluminación, posición,

escala, aspecto, etcétera. Al agrupar los gradientes en los histogramas se siguen ciertas estrategias que permiten obtener invariancia a algunas variaciones en la localización y el aspecto de los objetos en la imagen. Sin embargo, cuando se tienen cambios de iluminación como los que se producen en estas imágenes la intensidad del gradiente va a cambiar. Estos cambios en la intensidad del gradiente también se van a reflejar en los valores del histograma.

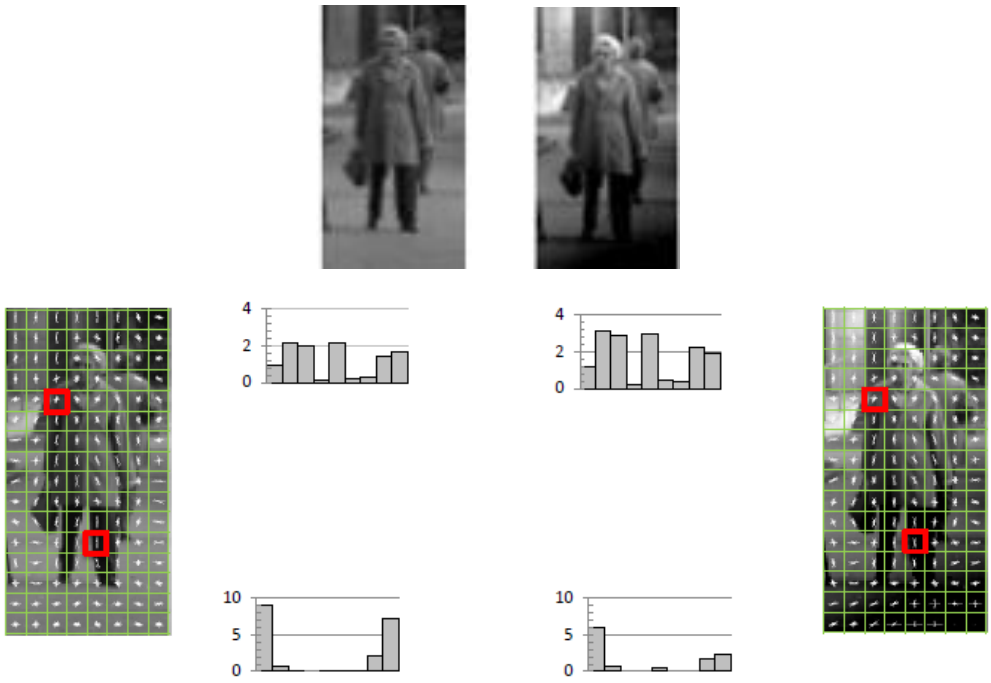


Ilustración 30: Cambios en la intensidad del gradiente

Sí se toman los histogramas de la celda que está situada más arriba en rojo, en una zona de la imagen donde el contraste es mayor en la segunda imagen que en la primera, se observa que también los valores del histograma son mayores para la segunda imagen que para la primera imagen. Este efecto es el contrario para la celda de abajo que está situada en una zona donde el contraste es mayor para la primera imagen que para la segunda y los valores del histograma para la primera imagen también son mayores que los valores del histograma para la segunda imagen.

Para minimizar estas diferencias en la descripción de las imágenes, es conveniente normalizar los valores de los histogramas, con el objetivo de conseguir que la magnitud global del gradiente sea similar en ambas imágenes. Sin embargo, como se vio en este ejemplo, los cambios de iluminación no son constantes a lo largo de toda la imagen, y no es suficiente aplicar una única normalización uniforme en toda la imagen, sino que va a ser preferible una normalización local adaptada a cada una de las zonas de la imagen. Es por ello que se introduce el concepto de bloque.

Un bloque es una agrupación de varias celdas vecinas. Una configuración habitual en el descriptor HOG es utilizar bloques de dos celdas en horizontal y dos celdas en vertical, como se ve en la ilustración 31.

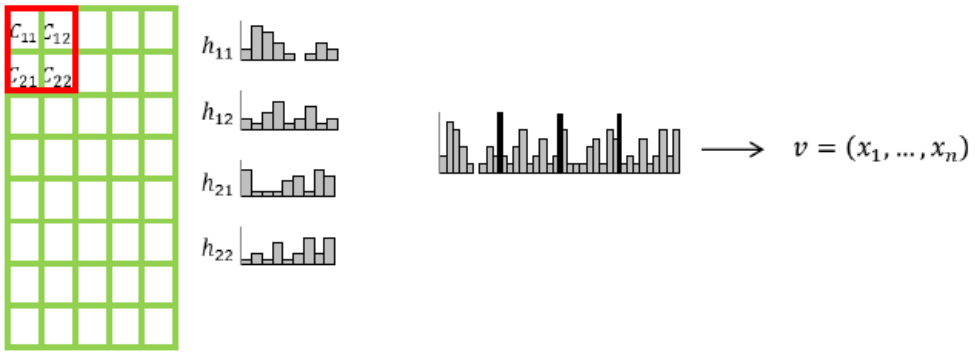


Ilustración 31: Bloque y normalización

Para cada bloque se toman los histogramas de cada una de las celdas y se concatenan para obtener el vector con la representación del bloque. La normalización de los histogramas se realiza a nivel de bloque, es decir, se normaliza el vector resultado de concatenar todas las celdas. La normalización se obtiene dividiendo cada uno de los componentes del vector por su norma. La norma de un vector es la raíz cuadrada de la suma de todas sus componentes al cuadrado. De esta forma se garantiza que la magnitud global del vector va a ser uno y se minimizan las diferencias debidas a variaciones locales en el contraste.

$$v' = \frac{v}{\sqrt{\|v\|^2 + \varepsilon}}$$

La constante épsilon de la fórmula es un valor muy pequeño y se introduce simplemente para evitar divisiones por cero, en aquellos casos de bloques dónde la intensidad sea constante a lo largo de todo el bloque y por lo tanto, la magnitud total del gradiente sea cero.

La normalización contribuye a reducir las diferencias en la representación final entre imágenes similares y por lo tanto, permite obtener un descriptor final mucho más robusto. Los bloques se definen de forma que tengan un cierto solapamiento entre ellos. La redundancia que se va a obtener con este solapamiento, va a ayudar a conseguir un descriptor más robusto ante deformaciones y variaciones en la forma del objeto.

Habitualmente, los bloques se colocan con una separación de una sola celda entre ellos, tanto en horizontal como en vertical, y la representación final del descriptor HOG se obtiene concatenando la representación normalizada de todos estos bloques solapados.

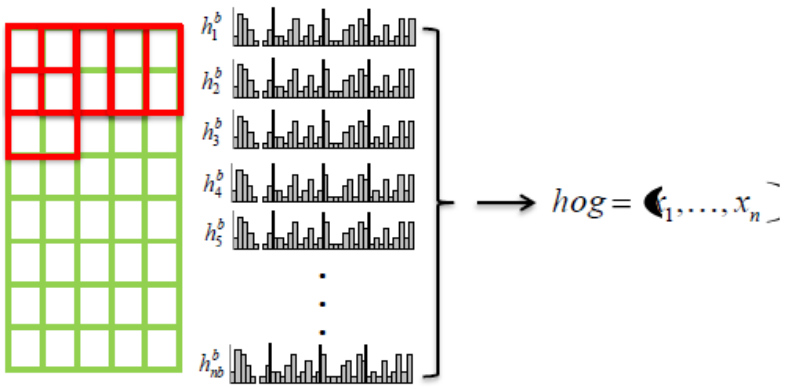


Ilustración 32: Solapamiento de bloques

Cada celda va a contribuir a la representación final de varios bloques. Tantos, como celdas se tengan en cada uno de los bloques. A pesar de que el histograma básico de la celda va a ser el mismo en todos los bloques, como cada bloque va a aplicar una normalización diferente que depende también del resto de las celdas del bloque, cada uno de los bloques va a ser diferente.

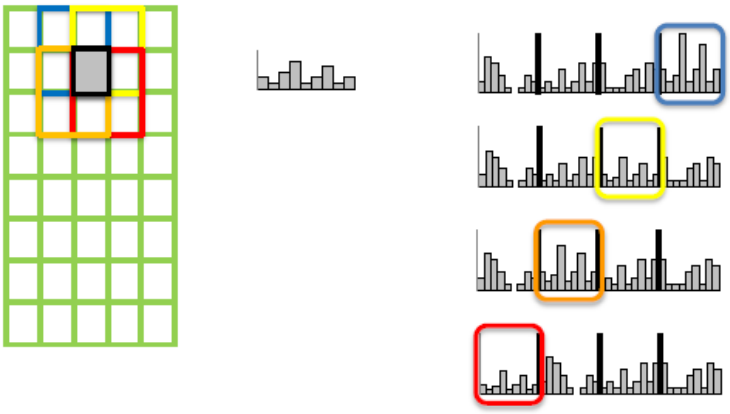


Ilustración 33: Descriptor HOG, cada celda contribuye a varios bloques

En el descriptor se tiene como parámetros el tamaño de cada una de las celdas y junto con el tamaño de la imagen de entrada, esto va a determinar el número de celdas que se pueden colocar en la imagen. Además se tienen un par de parámetros que afectan a cómo se construyen los histogramas. Por un lado, si se utiliza el gradiente con signo o sin signo y por otro lado el número de intervalos del histograma. Finalmente, se tiene el parámetro que fija cuántas celdas se colocan en cada uno de los bloques.

Resumiendo, parámetros del descriptor:

- Tamaño de la celda
- Signo del gradiente
- Número de intervalos del histograma de orientaciones
- Número de celdas en cada bloque

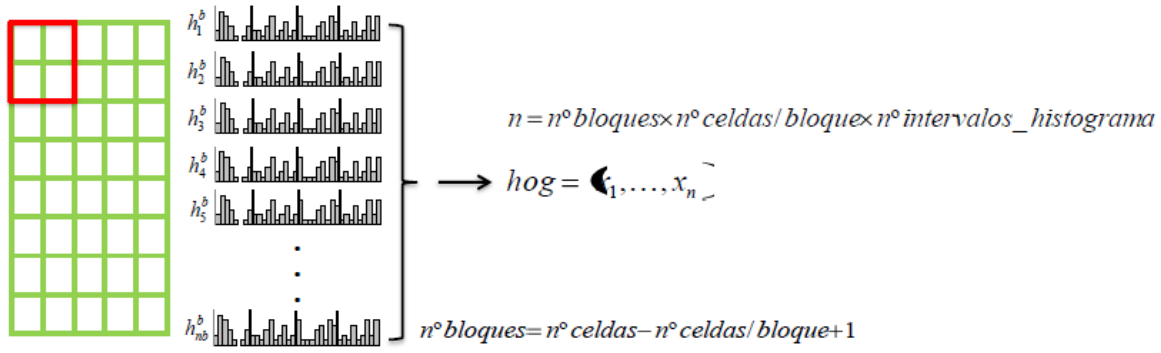


Ilustración 34: Parámetros del descriptor HOG, número de bloques

Algunos valores habituales para estos parámetros suelen ser un tamaño de celda de ocho por ocho pixeles, utilizar el gradiente sin signo, dividir la orientación del gradiente en nueve intervalos y utilizar bloques de dos por dos celdas. Evidentemente otras configuraciones son posibles y dependiendo de la aplicación concreta de detección de objetos a realizar.

Todos estos parámetros acaban determinando el número de dimensiones final del descriptor HOG, cuántos componentes tiene el vector y este número final de dimensiones se puede calcular a partir del número de bloques que se sea capaces de colocar en la imagen, multiplicado por el número de celdas que se tienen en cada uno de estos bloques y a su vez, multiplicado por el número de intervalos de los histogramas de cada una de las celdas.

Notar que la división de la imagen en bloques es muy similar a la que se realiza en el descriptor LBP. Esta coincidencia en la división en bloques, favorece que con la misma división de la imagen en bloques se puedan fácilmente calcular ambos descriptores, LBP y HOG. Esta facilidad permite diseñar métodos de detección que calculen los dos descriptores a la vez y posteriormente los combine. De esta forma habitualmente se mejoran los resultados de detección, porque se combinan las propiedades de textura y de contorno que proporcionan ambos descriptores.

10.3 CLASIFICACIÓN DE CANDIDATOS: INTRODUCCIÓN

Se verá primero en el concepto de clasificación. Con anterioridad se ha comentado que un detector de objetos se basa en una secuencia de procesamiento en la que intervienen distintos módulos. Uno de los módulos principales, el más importante probablemente, es el de clasificación. Así, a este módulo de clasificación le llegan una serie de candidatos para determinar si se corresponden con los objetos que interesan o no.

Se supondrá que un candidato consiste en una ventana dentro de una imagen, y que esas ventanas tienen un tamaño fijo o también llamado canónico, es decir que su anchura y su altura en el momento de ser clasificadas siempre son fijas. Así, la tarea del clasificador es decir si dentro de estas ventanas hay contenidos uno de los objetos que interesan o no, y además con un tamaño determinado. Por ejemplo, véase la ilustración 35, si el objeto es un peatón, para hacer un detector de peatones, el clasificador divide las ventanas que tienen el objeto buscado de las que no lo tienen. Cuando dice que no, en realidad, en este caso el clasificador no sabe qué contenido hay en la ventana, lo único que sabe es que ese contenido no se corresponde con el objeto de interés.

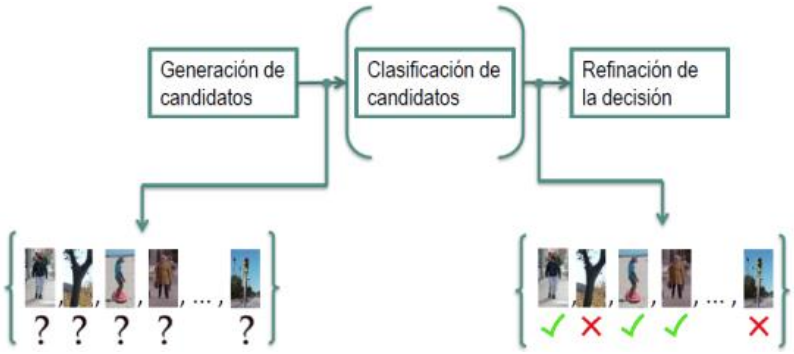


Ilustración 35: Clasificación de candidatos, ingresan ventanas con tamaño canónico.

Lo primero que hay que hacer es diseñar un descriptor de ventanas tal que el valor de ese descriptor sea similar, a otras de la misma clase, aunque las ventanas contengan peatones distintos, y a la vez sea muy diferente para ventanas que no contienen peatones, es decir, para ventanas que contienen fondo. Una vez conseguido eso, se coloca una frontera que separe peatones de objetos de fondo en el espacio del descriptor. Por tanto, los conceptos de interés aquí son descriptor de ventanas y frontera en el espacio del descriptor, (ilustración 36).

Ilustración 36: Conceptos de Descriptores y Frontera

Un descriptor es un vector columna donde la primera componente es un uno solo por cuestiones matemáticas, (que se verá más adelante), y donde el resto de las componentes son las que realmente describen la ventana, (Ilustración 37). Si se tiene una ventana con un objeto que interesa y otra ventana donde no hay ninguno de esos objetos, lo que se quiere es que sus respectivos descriptores sean muy diferentes y a su vez todas las ventanas que contengan objetos que interesan, entonces sus respectivos descriptores tengan valores muy similares.

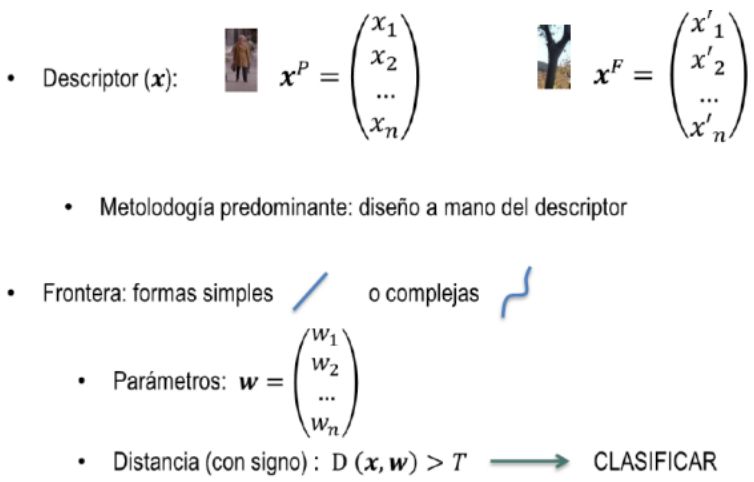


Ilustración 37: Descriptores y Fronteras

Por otro lado, las fronteras pueden consistir en formas simples como una recta en el espacio bidimensional o formas más complejas como una curva en ese mismo espacio. En tres dimensiones se habla de plano y de superficie y en n dimensiones de hiperplano y de hipersuperficie. En cualquiera de los casos estas fronteras se definen a partir de una serie de parámetros que también se agrupan en forma de vector columna y ese vector es al que se le llama modelo. Con los descriptores y los modelos y también a partir de un cierto umbral, (en inglés threshold), se define una distancia con signo que es la que en realidad va a permitir clasificar las ventanas. Por ejemplo en la ilustración 38 se tiene una frontera que es una línea en el espacio bidimensional y las muestras que caen a un lado de la frontera se clasifican como negativas y las que caen al otro lado como positivas, como conteniendo objetos que interesan o conteniendo simplemente fondo.

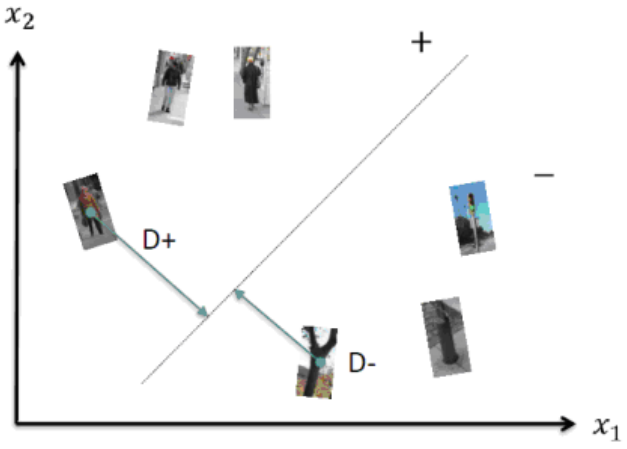


Ilustración 38: Frontera en un espacio bidimensional

También se puede ser un poco más permisivo y si se tiene esa frontera de referencia, en realidad se puede establecer un umbral diferente. Bien, la cuestión también es que dado un conjunto de muestras negativas y un conjunto de muestras positivas, en realidad hay muchas fronteras, véase ilustración 39. Allí hay varias líneas que separaran los dos conjuntos. Entonces, ¿Cuál de ellas es la mejor? Para obtener la frontera óptima es necesario introducir conceptos de aprendizaje automático, que viene del inglés machine Learning, desarrollados más adelante.

Ya se analizaron los descriptores Local Binary Patterns, y HOG y desde el punto de vista de las fronteras se verá el caso de las fronteras lineales que se aprenden en base a la regresión logística.

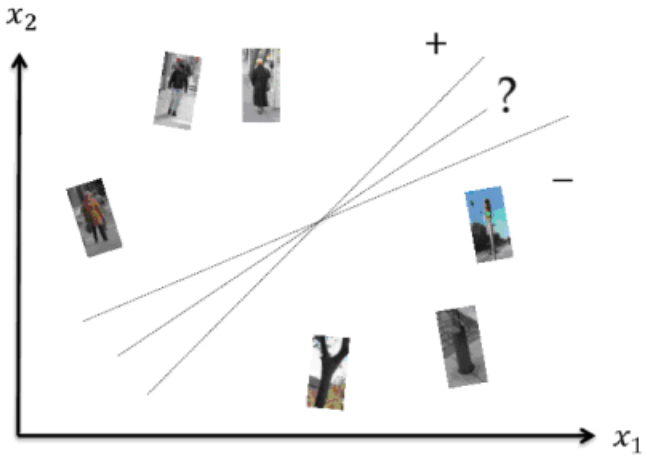


Ilustración 39: Aprendizaje computacional, varias posibles fronteras

10.3.1 REGRESIÓN LOGÍSTICA

Ya se vio que para clasificar las ventanas de una imagen se necesita un descriptor de las ventanas y una frontera en el espacio del descriptor. Con esos dos ingredientes se puede formar un clasificador. La función logística ayuda a definir ese clasificador. Para ello se verá fronteras lineales, que aunque son las más simples son muy utilizadas en la práctica porque dan buenos resultados. Esto quiere decir que si en un espacio bidimensional se tiene una línea, en un espacio tridimensional se tiene un plano, y en general en un espacio n dimensional se tiene lo que se conoce como hiperplano.

- La frontera de clasificación es un Hiperplano en $\Re^n$, cuya ecuación es:

  $$w_0 + \sum_{i=1}^{n} w_i x_i = 0$$

  Esto da una línea en R a la dos y un plano en R a la tres.

- El vector que define al hiperplano, es decir el modelo, es: $w = (w_0, w_1, ..., w_n)^T$

- Un punto en $\Re^{n+1}$, es decir el descriptor es: $x = (1, x_1, x_2, ..., x_n)^T$

Por lo tanto, una forma de escribir la primera ecuación como producto vectorial, en forma compacta es:

$$w^T x = 0$$

Se puede ver en la primera ecuación, que se define un hiperplano donde se tienen las componentes del vector w que define al hiperplano y las coordenadas del espacio donde ese hiperplano está definido.

Las coordenadas de X corresponden a las componentes de los descriptores de las ventanas donde también se añade artificialmente un 1 para poder expresar esta ecuación de forma compacta, como se ve en la última

ecuación. La t hace referencia a la traspuesta. Un ejemplo de clasificador binario con n=2 es el siguiente:

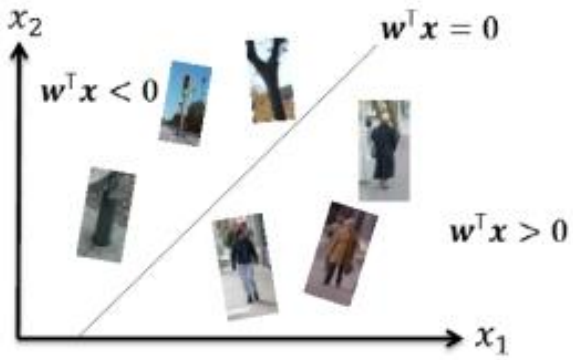


Ilustración 40: Clasificador binario lineal en un espacio n=2

Donde se cumple que:

$w^T x > 0 \rightarrow 1$, es decir, 1 es equivalente a asignar un peatón.

$w^T x < 0 \rightarrow 0$, es decir, 0 es equivalente a asignar fondo.

En $w^T x = 0$ se tendría que definir qué valor se quiere asignar, si fondo o peatón.

Por lo tanto, en la ilustración 40 se ve un ejemplo de clasificación para un caso de n igual a dos, es decir un espacio bidimensional. Se observa ejemplos de ventanas con peatones y contraejemplos o ejemplos negativos que son ventanas que contienen fondo. La línea en el medio es la frontera que se tiene definida por la ecuación $w^T x = 0$ donde aquí el descriptor para este caso es w. Con una cierta ventana, con un cierto descriptor X, dependiendo del signo de este producto escalar se dice que la ventana contiene fondo o contiene peatón. En caso de que fuese cero, se tendría que tomar una decisión de diseño y decir si es peatón o fondo o si simplemente se dice que no se puede decidir.

Con $w^T x > 0$ se ha asumido un cero para hacer estas comparaciones pero alternativamente se puede asumir cualquier otro umbral T para ser más permisivos o más restrictivos.

$w^T x > T \rightarrow 1$, es decir, 1 es equivalente a asignar un peatón.

$w^T x < T \rightarrow 0$, es decir, 0 es equivalente a asignar fondo.

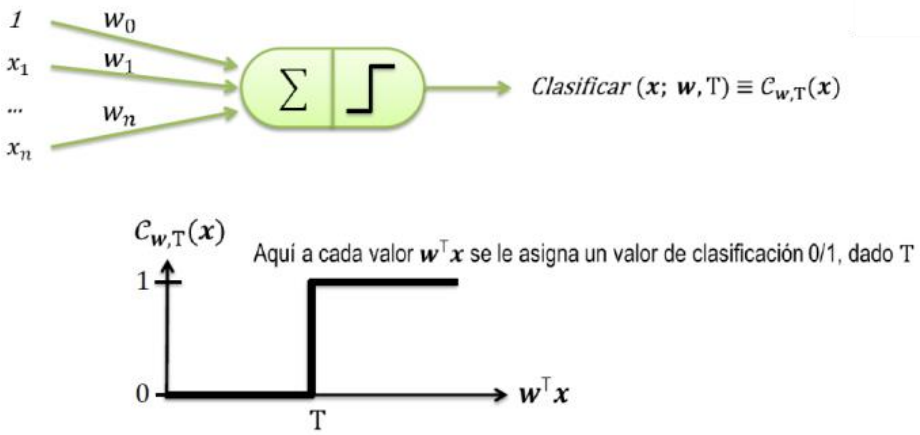


Ilustración 41: Función umbral

Por tanto esto se reduce a aplicar la función, que se observa en la ilustración 41, que es un umbral sobre el producto escalar y un dado valor T de umbral. Se tiene que la definición de esa función umbral es 0 si el valor de entrada es menor que el umbral y si es mayor, es 1. Aquí se tiene una unidad de proceso a la que le entran una serie de valores. Estos valores son los componentes del descriptor y cada uno va por un camino distinto. Estos caminos tienen asociados unos pesos que son los componentes del modelo. Cuando una componente de entrada pasa por un camino, queda multiplicada por el peso que tiene ese camino. Una vez hechas todas esas multiplicaciones, estas acaban siendo sumadas y se les aplica un umbral sobre el valor suma. Esto da el resultado de clasificar para ese descriptor en ese modelo y con un cierto umbral. Se ve que la función del clasificador va a dar un valor cero o uno dependiendo del valor de este producto escalar respecto al valor del umbral que se ha introducido, T. Por tanto, cuando el valor de este producto escalar sea menor que T el clasificador devolverá un cero y cuando sea mayor devolverá un uno. Por eso, esta función tiene esta forma de escalón. Un problema que se tiene es que este modelo vendrá de un proceso de entrenamiento. Puede pasar que el primer modelo aprenda con una serie de ejemplos y más adelante se tienen nuevos ejemplos y se quiere reaprender el modelo para que sea mejor. Si se hace eso, es posible que este umbral T, ajustado para el primer modelo que se aprendió, ahora no sea el mejor umbral para el segundo modelo que se ha aprendido y por tanto se tiene que volverlo a reajustar el umbral T. Desde luego, esto no es deseable, y para evitar eso en lugar de seleccionar un umbral directamente para el producto escalar, lo que se hace es definir una versión suave del producto escalar y hacer un umbral sobre esa versión suave. Y para ello se utiliza lo que se conoce como función sigmoidea o también llamada función logística. Por lo tanto, la alternativa es reconvertir los valores de $w^T x$ para tener una transición más suave. Una posibilidad es usar, como ya se dijo, la función logística: Logistic ($w^T x$), donde

$$y = w^T x$$

$$Logistic(y) = \frac{1}{1+e^{-y}}$$

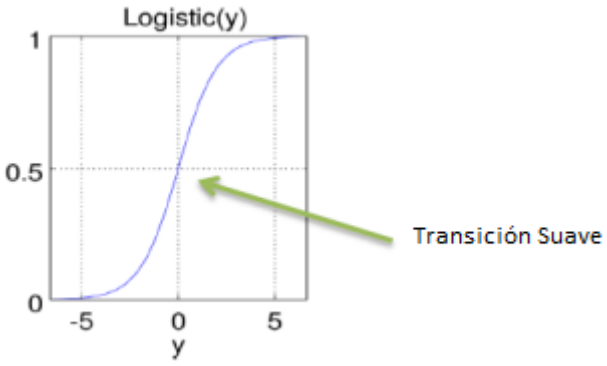


Ilustración 42: Función Logística

Ahora podemos utilizar métodos de umbralización más elaborados. Ahora $T \in [0,1]$ y no actúa sobre $w^T x$, sino sobre la versión suave dada por la función logística. En la ilustración 42 se tiene la definición de una función logística para un valor real cualquiera y donde se ve que también la función exponencial. Se ve la forma que tiene esa función logística: Cuando el valor y tiene a + infinito la función logística tiende a uno. Cuando el valor y tiende a – infinito la función logística tiende a cero. Y cuando y vale cero la función logística tiene el valor 0.5. Por tanto, esta función está acotada entre cero y uno y se pone el umbral en el eje vertical del clasificador. De esta manera aunque cambie el modelo, si ese cambio no es muy grande, se puede usar el mismo umbral que antes se tenía, y se puede interpretar mejor porque es un valor entre cero y uno, que es una forma más intuitiva.

Ilustración 43: Función logística y red neuronal

Si en lugar de la decisión de clasificación, simplemente devolvemos el valor de Logistic ($w^T x$), estaríamos ante el clásico modelo matemático de una neurona en el contexto de las redes neuronales. Como se ve en la ilustración 43, lo único que cambia con respecto al anterior es que tiene una función sigmoidea o logística, en lugar de un escalón. Esta unidad de proceso devuelve el valor de la función logística del producto escalar del modelo del descriptor y esto coincide con la unidad básica de procesamiento de las conocidas redes neuronales.

10.3.2 REGRESIÓN LOGÍSTICA – APRENDIZAJE

Anteriormente se vio cómo la función logística ayuda a definir un clasificador C y en esa definición también está involucrado un modelo w. Así se tiene un clasificador C que depende de un modelo w y un umbral T, el clasificador tiene como entrada el descriptor de una ventana, X, cuyo contenido se quiere clasificar. Si la ventana contiene el objeto que interesa devuelve el uno, si no lo contiene, devuelve un cero. Esto depende de una condición que dice

que la función h ha de ser menor que el umbral T. Esta función h del modelo y el descriptor consiste en la función logística del producto escalar del modelo y el descriptor.

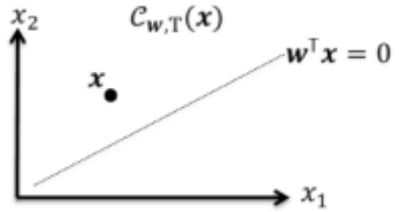


Ilustración 44: Clasificador C, modelo W, umbral T y un punto x, a ser clasificado

$$y = w^T x$$

$$h_w(x) = Logistic(y) = \frac{1}{1+e^{-y}}$$

$$C_{w,T}(x) = 1 \Rightarrow h_w(x) < T \Rightarrow w^T x < \ln\left(\frac{T}{1-T}\right)$$

$$C_{w,T}(x) = 0 \Rightarrow h_w(x) \geq T$$

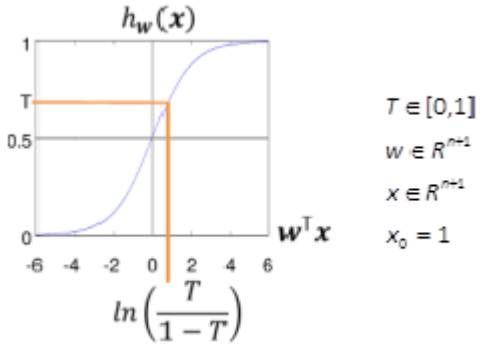


Ilustración 45: Regresión Logística

Por tanto, lo que queda saber es qué valor tiene el modelo w y para obtener ese valor se utilizara aprendizaje automático. Para realizar aprendizaje automático, se parte de un conjunto de entrenamiento que tiene m muestras. El modelo W se aprende a partir de un conjunto de muestras.

Conjunto de entrenamiento: $\partial = \left\{\left(x'^1, y^1\right)...\left(x'^M, y^M\right)\right\}$ donde:

- La j-esima muestra de entrenamiento es $\left(x'^j, y^j\right)$
- $x'^j \ni \Re^n$, Lo transformamos en $x^j \ni \Re^{n+1}$ donde $x^j = \left(1, x'^j\right)$
- $y^j \in \{0,1\}$ es la clasificación binaria y supervisada.

Ilustración 46: Regresión Logística, w obtenido minimiza el "coste de equivocarse" al clasificar x

Cada una de estas muestras tiene dos componentes. La primera componente la x prima, es un número en $\Re^n$ que corresponde al descriptor de una ventana usado para realizar el entrenamiento y se usan M ventanas de entrenamiento. En lugar de trabajar con las x primas se trabaja con una x, a la que se le añade la componente uno por delante, para poder trabajar con la notación compacta donde se multiplica el modelo por el descriptor directamente. La segunda componente de cada muestra es una etiqueta. Esta etiqueta dice que si el descriptor corresponde al objeto que interesa, el valor de y es uno, si no lo contiene, es cero. El aprendizaje automático consiste en que dado un conjunto de entrenamiento se obtiene un modelo w. Ese modelo es el que minimiza el coste de equivocarse al clasificar las muestras. El proceso de ingresar el conjunto de entrenamiento y de obtener el modelo es lo que se llama regresión logística. La palabra regresión hoy en día no sería muy adecuada, se la mantiene por cuestiones históricas porque siempre se ha llamado así a este proceso. Cuando y es un número real se habla de regresión, y cuando y es un número natural se habla de clasificación. En este caso, se debería hablar de clasificación logística. Por un lado se tiene la etiqueta de la muestra, por otro lado, si se tiene un cierto modelo y se aplica esta función logística del producto escalar del modelo y el descriptor, lo que va a devolver un número que estará entre cero y uno.

El coste de equivocarse al clasificar x^j se puede definir como $\left(y^j - h_w\left(x^j\right)\right)^2$, es decir, utilizando la denominada pérdida cuadrática (L_2). El vector $w^{'}$ que minimiza el coste de equivocarse para ∂ en su totalidad, se obtiene resolviendo lo siguiente:

$$w^* \leftarrow ArgMin_w\left(J(w)\right) \text{ Con } w \in \Re^{n+1}$$

Siendo $J(w) = \frac{1}{M}\sum_{j=1}^{M}\left(y^j - h_w\left(x^j\right)\right)^2$

Se eleva al cuadrado para tener una función de coste cuadrático. Se tiene que considerar todas las muestras y eso se hace mediante la sumatoria, donde M es el número de muestras, y se suma el coste de equivocarse de cada muestra y dividirlo por el número de muestras. Lo que interesa es encontrar el modelo que minimice esta expresión. La expresión J (w) del modelo es a lo que se llama la función de coste o función de pérdida. Así, lo que se quiere es minimizar la función de coste y para eso se tiene en cuenta lo siguiente:

- No hay una solución cerrada fácil para obtener w^*, es decir, no podemos despejar w^* de una manera sencilla a partir de J (w).
- Por lo tanto, se opta por utilizar un método de optimización, que a partir de un valor inicial de w, y unas ecuaciones de "actualización", nos lleve a la solución que se busca.
- Sin embargo, J (w) no es una función convexa respecto a w. Por lo tanto, es difícil de optimizar, en caso de minimizar.

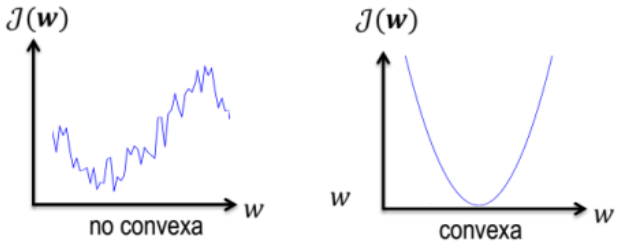


Ilustración 47: Funciones convexa y no convexa

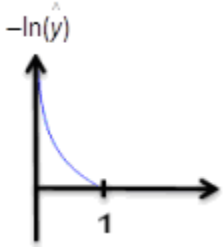
En primer lugar lo que se quiere es manipular J (w) de forma de despejar el modelo óptimo que se busca. Sin embargo, esto no es posible, entonces lo que queda es utilizar un método de optimización para buscar el mínimo de esta función. Es decir, a partir de un cierto valor inicial del modelo y unas ecuaciones de actualización, se aplican esas ecuaciones iterativamente hasta que se llega a una solución. Cuando se utiliza este tipo de procedimiento, es necesario tener funciones convexas, porque dado un punto inicial cualquiera y con las convenientes precauciones, se converge al mínimo. Pero si la expresión de función de coste no es convexa, como la de la izquierda de la ilustración 47, se puede acabar un mínimo local. Por ese motivo se utiliza lo siguiente:

$$\left(y^i - h_w\left(x^j\right)\right)^2 \sim Coste\left(h_w\left(x^j\right), y^j\right)$$

- $Coste\left(\hat{y}, y\right) = -\ln\left(\hat{y}\right) \forall y = 1$
- $Coste\left(\hat{y}, y\right) = -\ln\left(1 - \hat{y}\right) \forall y = 0$

Por otra parte, $\hat{y} \in [0,1]$

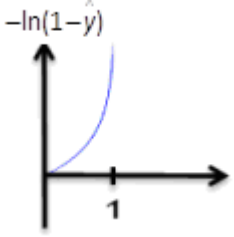
Con $-\ln(\hat{y})$



$\hat{y} \to 1 \Rightarrow coste \to 0$

$\hat{y} \to 0 \Rightarrow coste \to \infty$

Con $-\ln(1-\hat{y})$



$$\hat{y} \to 1 \Rightarrow coste \to \infty$$
$$\hat{y} \to 0 \Rightarrow coste \to 0$$

En lugar de utilizar la expresión cuadrática vista para medir el coste de equivocarse al clasificar la muestra, se usa el coste donde si la etiqueta tiene valor uno el coste es el menos el logaritmo neperiano de la estimación, mientras que si la etiqueta tiene valor cero el coste es menos el logaritmo neperiano de uno menos la estimación. En una estimación que viene de la función logística estará acotado entre cero y uno.

Una versión más compacta es la siguiente:

$$Coste\left(\hat{y},y\right) = -\left(y\ln\left(\hat{y}\right) + (1-y)\ln\left(1-\hat{y}\right)\right)$$

Ahora sí, J (w) es convexa respecto de w

Nos queda la fórmula de la regresión logística

$$J(w) = -\frac{1}{M}\sum_{j=1}^{M} y^j \ln\left(h_w\left(x^j\right)\right) + \left(1-y^j\right)\ln\left(1-h_w\left(x^j\right)\right)$$

$$w^* \leftarrow ArgMin_w\left(J(w)\right) \text{ con } w \in \Re^{n+1}$$

Así es esta la nueva función de coste y se puede ver que es una función de coste convexa respecto al modelo. Por lo tanto se puede utilizar con garantías de que el algoritmo de optimización la minimice.

10.3.2.1 RESUMEN

W se aprende de un conjunto de entrenamiento: $\partial = \left\{\left(x^1,y^1\right)\ldots\left(x^M,y^M\right)\right\}$ donde:

- La j-esima muestra de entrenamiento es $\left(x^j,y^j\right)$
- x^j ə $\Re^{n+1}$, descriptor de la ventana j
- $y^j \in \{0,1\}$, etiqueta de la ventana j

- $h_w(x^j) = Logistic(w^T x^j)$
- J (w) es convexa respecto de w

$$J(w) = -\frac{1}{M}\sum_{j=1}^{M} y^j \ln\left(h_w\left(x^j\right)\right) + \left(1-y^j\right)\ln\left(1-h_w\left(x^j\right)\right)$$

$$w^* \leftarrow ArgMin_w\left(J(w)\right) \text{ con } w \in \Re^{n+1}$$

Se ha visto que un clasificador de ventanas se basa en una frontera en el espacio del descriptor de esas ventanas. Esas fronteras quedan definidas por un conjunto de parámetros, llamado modelo y esos modelos se pueden aprender mediante el método de regresión logística. Hay que minimizar dicha función para obtener el modelo. El proceso utilizado para hacer esa minimización devolverá el modelo que se busca. La función de coste lo que hace es medir el error que se comete al clasificar las muestras de entrenamiento dado un cierto modelo y es una función convexa respecto al modelo. Como esta función de coste es convexa, para minimizarla se puede utilizar un algoritmo que se conoce como descenso del gradiente que reduce el cálculo de un mínimo local a una secuencia de problemas de búsqueda lineal.

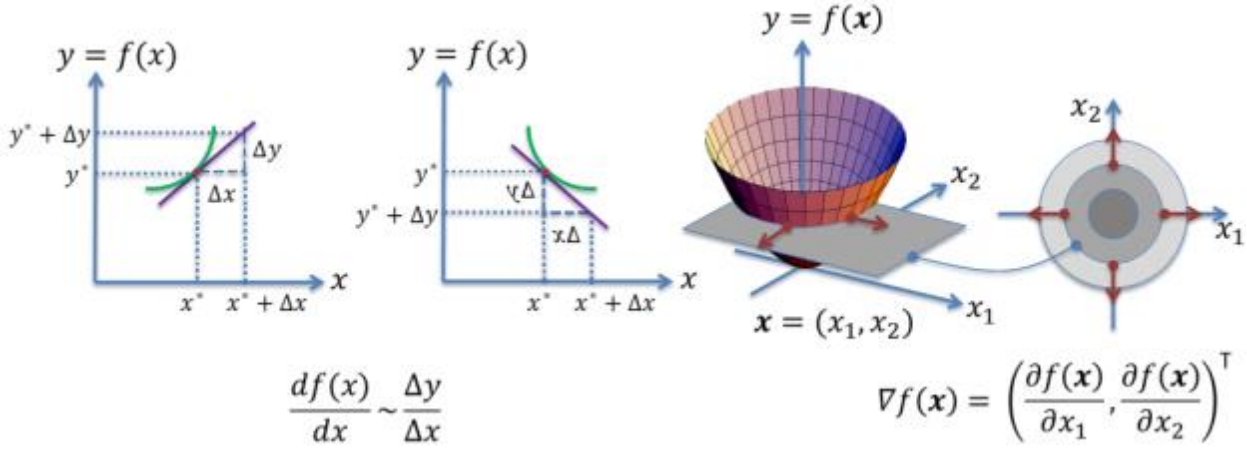


Ilustración 48: Descenso del gradiente

La derivada de la función en un cierto punto x asterisco consiste en la pendiente de la recta tangente a la función en ese punto. Dado un salto en x, se puede ver cuál es el salto en y que le corresponde desde el punto de vista de esa recta tangente. El cociente entre el salto Y con el salto en X es la aproximación a la derivada, como se marca en la ilustración 48 a la izquierda abajo.

Para una función de dos variables, el gradiente de la función tiene dos componentes, uno es la derivada parcial de la función respecto a la primera variable y la otra componente es la derivada parcial de la función respecto a la segunda variable. El gradiente viene determinado por dos números, uno es la dirección y el otro es la magnitud del gradiente. Uno es en qué dirección apunta y el otro es la longitud. La dirección es perpendicular a la curva de nivel en el punto y la magnitud corresponde a cuanto cambia la función en esa dirección, véase la ilustración 48 a la derecha abajo. El gradiente apunta siempre en la dirección de máximo cambio local de la función.

El algoritmo por descenso al gradiente permite minimizar el valor de la función de coste, que es convexa respecto al modelo y por lo que tiene un mínimo global. Por otra parte, este algoritmo necesita partir de un cierto punto. Es

decir, en un instante cero, tener un cierto valor inicial del modelo. Partiendo de ese valor inicial del modelo, se le resta el gradiente de la función de coste a ese valor inicial.

Descenso del gradiente, se repite lo siguiente:

- $w_0^{(k)} \leftarrow w_0^{(k-1)} - \alpha \frac{\delta}{\delta w_0} J(w^{(k-1)})$
- ….
- $w_n^{(k)} \leftarrow w_n^{(k-1)} - \alpha \frac{\delta}{\delta w_n} J(w^{(k-1)})$

Hasta converger, (se monitoriza J (w)).

$\alpha \in \Re^+$ Representa la velocidad, (ratio), de aprendizaje.

$$\nabla J(w) = \left(\frac{\partial J(w)}{\partial w_0}, \ldots, \frac{\partial J(w)}{\partial w_n} \right)^T$$

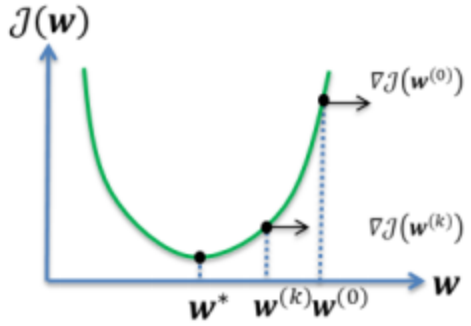


Ilustración 49: Descenso del gradiente

Lo que hace en cada iteración es dar un salto en este caso hacia atrás, para acercarnos al óptimo. En otras palabras, baja por la curva con cada nuevo punto, bajando paso a paso, para eventualmente llegar al mínimo. Se ha introducido un número real positivo, alfa, que se conoce como velocidad de aprendizaje. Esto puesto en forma compacta es:

$$\frac{\delta J(w)}{\delta w_i} = \frac{\delta}{\delta w_i} \left[-\frac{1}{M} \sum_{j=1}^{M} y^j \ln\left(h_w\left(x^j\right)\right) + \left(1 - y^j\right) \ln\left(1 - h_w\left(x^j\right)\right) \right]$$

$$= \frac{1}{M} \sum_{j=1}^{M} \left(h_w\left(x^j\right) - y^j\right) x^j$$

Repetir hasta converger, monitorizando J (w):

$$w_0^{(k)} \leftarrow w_0^{(k-1)} - \alpha \frac{1}{M} \sum_{j=1}^{M} \left(h_w \left(x^j \right) - y^j \right) x_0^j$$

…

$$w_n^{(k)} \leftarrow w_n^{(k-1)} - \alpha \frac{1}{M} \sum_{j=1}^{M} \left(h_w \left(x^j \right) - y^j \right) x_n^j$$

Alfa controla los saltos. Si alfa está entre cero y uno, se acortan los saltos que corresponden al módulo del gradiente. Si alfa es un número mayor que uno, se alargan dichos saltos. Esto sirve para controlar dos tipos de situaciones. Puede haber casos en que la función de coste sea muy plana y por tanto, intentar llegar al mínimo local a base de saltos basados en el módulo del gradiente da lugar a muchas iteraciones. En este caso conviene tener un alfa mayor que uno, relativamente grande. También se puede tener el caso contrario donde los gradientes son muy grandes y esto provoca ir dando saltos de un lado al otro de la función de coste, alrededor del mínimo. En estos casos conviene un alfa pequeño, algún número entre cero y uno. El problema es que no se sabe a priori cuál es el mejor alfa.

10.3.3 REGRESIÓN LOGÍSTICA - FRONTERAS NO LINEALES

Un descriptor x tiene una serie de componentes, que se conocen como características, y es en el espacio del descriptor donde se buscan las fronteras lineales que separan los ejemplos que interesan del resto. Estas fronteras lineales se expresan mediante el producto escalar de un modelo w por el descriptor x igualado a cero. Para usar esta notación compacta se añade en la primera componente del descriptor un uno. Mediante este formalismo también se pueden expresar fronteras no lineales.

Viendo la ilustración 50, se tiene que:

$$(x_1 - c_{x_1})^2 + (x_2 - c_{x_2})^2 = r^2$$

Esta ecuación se la puede expresar asimismo como:

$$w^T x = 0$$

Donde

$$w = (w_0, w_1, w_2, w_3, w_4, w_5)^T$$

$$x = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)^T$$

Deduciéndose el modelo óptimo

$$w = ((c_{x_1}^2 + c_{x_2}^2 - r^2), -2c_{x_1}, -2c_{x_1}, 1, 0, 1)^T$$

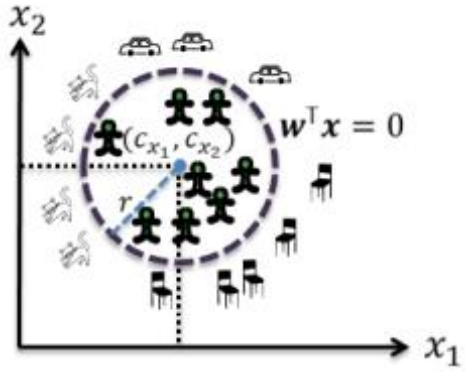


Ilustración 50: Fronteras no lineales

Arriba se ve un ejemplo de dos dimensiones donde lo que se ajusta como frontera es una circunferencia. Esta circunferencia tiene un centro (cx1, cx2) y un radio r. Los valores del modelo se aprenden, pero al descriptor es posible diseñarlo. Entonces, las primeras tres componentes coinciden exactamente con el modelo lineal, pero se le añade una serie de términos que son x1 al cuadrado, x2 al cuadrado y x1 por x2. El modelo óptimo w es el que ilustración más arriba, y se puede comprobar haciendo el producto escalar del modelo w por el descriptor x.

El problema es que si en el espacio lineal se tienen n componentes y si se hace la multiplicación de todos los componentes por todos los componentes, se pasa a tener (n cuadrado + n) /2. Si por ejemplo n es igual a 1000, el número de componentes es de 500.000, en el descriptor.

10.3.3.1 PROBLEMA DEL SOBREAJUSTE, "OVERFITTING"

Cuando se quiere aprender un modelo a partir de un conjunto de muestras, sucede que esas muestras tienen ruido o que el conjunto puede ser incompleto. Además el propio modelo es una aproximación porque no se sabe realmente qué forma tiene la frontera que separan los objetos que interesan del resto, eso hace que aunque se consiga un clasificador sin error en la clasificación de las muestras de entrenamiento, no se pueda garantizar la correcta clasificación de muestras nuevas, es decir no se puede garantizar la capacidad de generalización del clasificador. Por ejemplo, en la ilustración 51, se puede aprender un modelo que generase la frontera en azul y efectivamente todas las muestras de entrenamiento están clasificadas correctamente, pero da la sensación de que este clasificador es excesivamente complejo. Podría ser que en este ejemplo, que la línea roja sea la frontera que separa los objetos de interés del resto. Este problema se conoce como sobreajuste, es decir la frontera se ha ajustado excesivamente a los datos que se tienen de entrenamiento. Este problema tiende a manifestarse especialmente cuando el número de parámetros del modelo es muy alto respecto al número de muestras de entrenamiento.

Una manera de evitar este problema es, evidentemente, aumentar el número de muestras de entrenamiento. Esto no es siempre posible porque estas muestras muchas veces cuestan dinero de adquirir, y otras veces simplemente no es posible conseguir más. Otra posibilidad es bajar la dimensión del modelo, (la dimensión del descriptor), y esto es lo que se conoce como selección de características. Otra opción es la selección de modelo, es decir empezar con modelos simples e ir probando modelos cada vez más complejos y ver cuál de ellos es el que mejor resultado da. Finalmente queda el algoritmo de regularización del modelo.

Ilustración 51: Sobreajuste (Overfitting)

10.3.4 REGRESIÓN LOGÍSTICA – FACTOR DE REGULARIZACIÓN

Cuando se habla de regularizar un modelo, se refiere a que durante el proceso de aprendizaje de ese modelo se introduce un término en la función de coste para favorecer unos valores respecto a otros del modelo. En el caso de la regresión logística, consiste en la función que ya se tenía, más un término que es nuevo. Este término suma desde i igual a uno hasta n de los componentes al cuadrado del modelo multiplicado por una lambda divido 2M. Como esto se ha de minimizar, con la introducción de este término en la función de coste, se exige que los componentes del modelo no sean valores grandes. Con esta restricción todas las componentes de los descriptores tienen una relevancia similar. La constante lambda lo que hace es establecer el compromiso entre la función de coste que se tenía antes y este nuevo término. La función de coste que se tenía antes mide el error al clasificar las muestras de entrenamiento con el modelo. A esta nueva función se la puede introducir de forma sencilla en el algoritmo de descenso del gradiente, que se basa en las derivadas parciales respecto de los componentes del modelo, como se ve en la última ecuación, más abajo.

Regularización de la regresión logística:

$$J_R(w) \leftarrow J(w) + \frac{\lambda}{2M}\sum_{i=1}^{n} w_i^2$$

Repetir en paralelo

$$w_0^{(k)} \leftarrow w_0^{(k-1)} - \frac{\alpha}{M}\sum_{j=1}^{M} h_w(x^j) - y^j)x_0^j$$

$\forall i \in \{1...n\}$

$$w_0^{(k)} \leftarrow w_0^{(k-1)} - \frac{\alpha}{M}\left[\sum_{j=1}^{M} h_w(x^j) - y^j)x_i^j + \lambda w_i\right]$$

Hasta converger

$\lambda \in \Re^+$ Determina el grado de penalización

10.4 CLASIFICACIÓN DE CANDIDATOS: SUPPORT VECTOR MACHINES (SVM)

Referencia [30]

El objetivo de cualquier clasificador es encontrar una frontera que permita separar los ejemplos positivos de los ejemplos negativos. Igual que en la regresión logística, Support Vector Machine es un clasificador lineal, lo que quiere decir que la frontera de decisión, si se tiene un espacio bidimensional, será una línea; para tres dimensiones la solución va a ser un plano, y si se tiene un espacio de mayor dimensionalidad será un hiperplano. La diferencia entre la regresión logística y el Support Vector Machine es cómo se determina este hiperplano.

No existe una única manera de trazar estas líneas de separación, o hiperplano de separación, entre las dos clases.

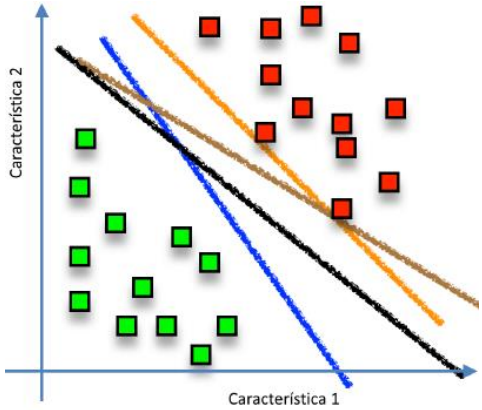


Ilustración 52: Múltiples Fronteras Lineales

En general se puede decir que existen dos grandes formas de atacar este problema dentro de lo que es la disciplina de reconocimiento de patrones. Por un lado se tiene lo que se conoce como modelos generativos que tratan de construir estas fronteras de separación entre clases a partir de estimar la función de densidad de probabilidad que se puede asociar a cada una de las clases.

Por otro lado se tiene lo que conoce como modelos discriminativos que tratan de clasificar las muestras sin tener que generar estas funciones de densidad de probabilidad de las clases, y por lo tanto encuentra la frontera a partir de un conjunto de entrenamiento que sea suficientemente representativo. Algunos ejemplos de estos métodos serían la regresión logística y las redes neuronales. Es dentro de este grupo que se puede ubicar a los Support Vector Machines.

Los SVM son sistemas de clasificación binarios, es decir, permiten distinguir entre dos clases. Los SVM son clasificadores lineales cuya solución se basa en encontrar el margen máximo entre las dos clases a partir de unos vectores determinados que se conocen como vectores de soporte.

El hiperplano de separación entre clases se obtiene a partir de la solución de un problema de optimización: distancia máxima entre los hiperplanos que contienen a los vectores de soporte de ambas clases, (margen máximo).

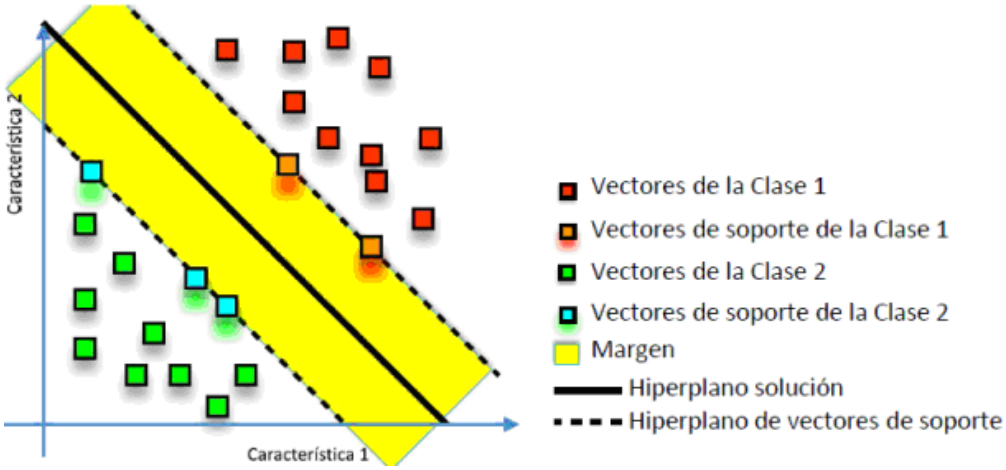


Ilustración 53: SVM, clasificador lineal basado en margen máximo a partir de los vectores de soporte

En el caso particular de dos dimensiones, por lo tanto de dos características, esta solución va a ser una línea recta. Se encuentra la frontera de separación a partir de las muestras de entrenamiento y para encontrar esta solución solo tiene en cuenta un número limitado de muestras del conjunto de entrenamiento con unas propiedades concretas. A estas muestras particulares se les denomina vectores de soporte. Y hay vectores de soporte para cada una de las clases que se tienen. Los vectores de soporte son elegidos de manera que la distancia del margen entre los planos, sea máxima, (en la imagen es la distancia entre las líneas punteadas con la línea central). Esta condición de margen máximo implica que se busca encontrar la región entre vectores de soporte más amplia posible que esté vacía de muestras de entrenamiento de cualquiera de las dos clases. El plano intermedio de esta región, será la solución del Support Vector Machines.

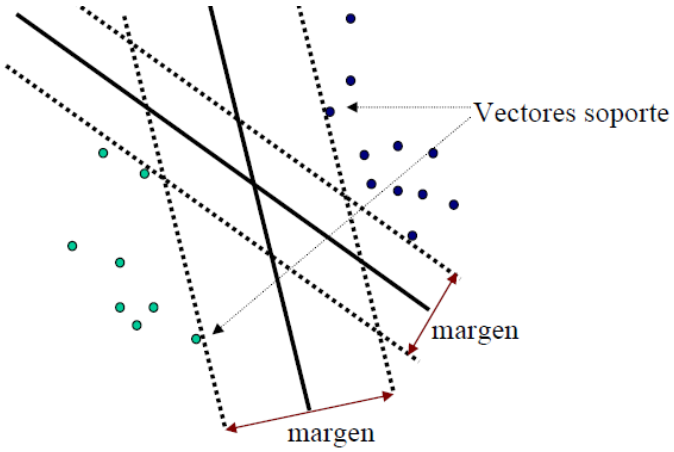


Ilustración 54: Distintas soluciones, con distintos márgenes

En la ilustración 54, observamos distintas soluciones, según sean seleccionados distintos vectores de soporte que dan distintos márgenes. De forma intuitiva se puede describir la solución como aquella que proporciona la región más amplia en el espacio de características que separa las dos clases y que además está vacía de muestras de entrenamiento. Cualquier elección alternativa de los vectores de soporte dará una solución alternativa y diferente

al hiperplano que permite separar las dos clases. Pero si no es una solución óptima, va a tener un margen más estrecho y por lo tanto un margen de seguridad menor en la clasificación, (ilustración 55).

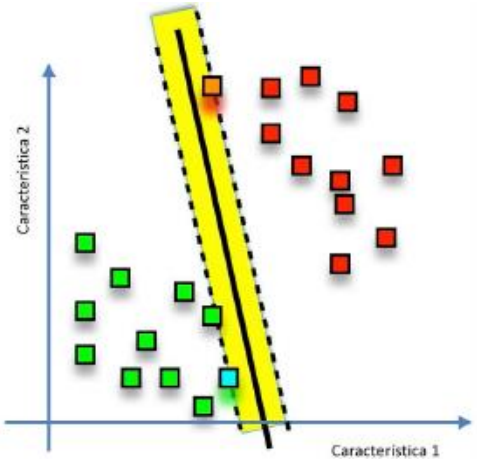


Ilustración 55: Solución no óptima

La manera que tiene el Support Vector Machines de encontrar la solución a partir del uso de vectores de soporte le proporciona algunas propiedades interesantes. Por un lado, debido a que la solución depende únicamente de la elección de los vectores de soporte, si se desplaza estos vectores de soporte, evidentemente la solución que se encuentra como resultado de la optimización del margen va a cambiar. Sin embargo, y debido a que los vectores de soporte son los únicos que contribuyen a la solución, si se desplaza las muestras que no son vectores de soporte la solución no va a cambiar y se tiene el mismo hiperplano como solución. Este hecho proporciona cierta robustez estadística y particularmente lo hace robusto a lo que se conoce como sobreajuste, (overfitting), que implica perder capacidad de generalización en el clasificador si se concentra demasiado en las muestras de entrenamiento. Por el contrario Support Vector Machines generaliza de manera robusta.

Hasta aquí se ha explicado lo que es el modelo básico del Support Vector Machines como clasificador lineal. Ahora se podría preguntar si este modelo no es demasiado restrictivo para poder obtener buenos resultados en conjuntos reales ya que tal como se ha expuesto, la aproximación que da sirve únicamente para clasificar conjuntos que sean linealmente separables y sin solapamiento entre clases.

No sería el caso del ejemplo que se presenta en la ilustración 56.

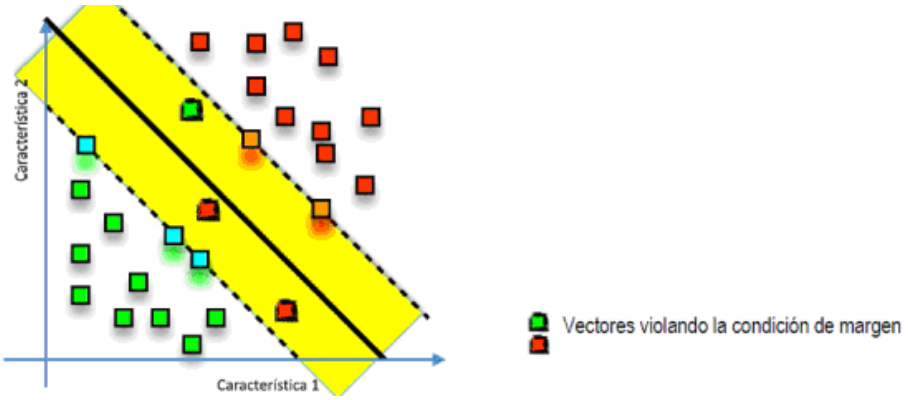


Ilustración 56: Conjuntos no linealmente separables: Margen Suave

Para que Support Vector Machines sea eficiente en este tipo de problemas que no son linealmente separables hay dos alternativas. Por un lado, se permite cierto solapamiento de clases a partir de relajar la condición del margen y por otro lado se extiende el modelo de forma que pueda trabajar en conjuntos que no sean linealmente separables mediante la transformación del espacio de características en otro que sí lo es, es lo que se conoce como el truco del kernel o el Kernel Trick.

La solución del margen suave implica añadir tolerancia a errores, es decir, vectores violando la condición de margen. La relajación de la condición del margen implica permitir que algunas muestras estén dentro de la región que define el margen entre las dos clases o incluso que queden erróneamente clasificadas. Esto permite que el Support Vector Machines pueda ser robusto al ruido asociado a estas muestras y permite tener cierta tolerancia a errores de clasificación. De forma que se puede trabajar con conjuntos que no sean separables linealmente. Esta tolerancia de errores se gestiona mediante lo que se conoce como variables de Slack y que lleva asociado consigo un cierto factor de regularización que busca encontrar el compromiso en esta tolerancia a errores.

Finalmente, el Support Vector Machines se puede aplicar a conjuntos que no sean linealmente separables. Para ello se mapea el conjunto de características, del espacio de características original en el que los datos no son separables linealmente a otro espacio de características de dimensión superior en el que los datos sí son linealmente separables, como se observa en la ilustración 57. Allí, conjuntos no linealmente separables pueden transformarse en linealmente separables en un espacio de dimensión superior.

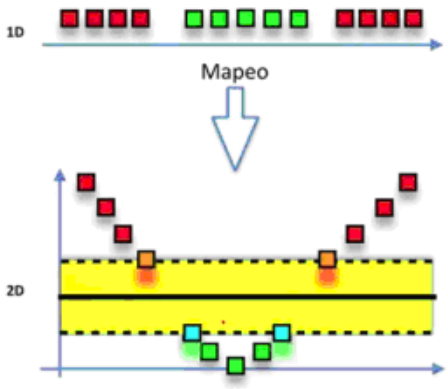


Ilustración 57: Conjunto no linealmente separable: Kernel Trick

10.4.1 SVM - DESARROLLO MATEMÁTICO: INTRODUCCIÓN

Que el Support Vector Machines sea un clasificador lineal implica que existe un hiperplano solución, cuya solución se puede expresar de esta forma:

$w^t x_i + b = 0$ Hiperplano solución W

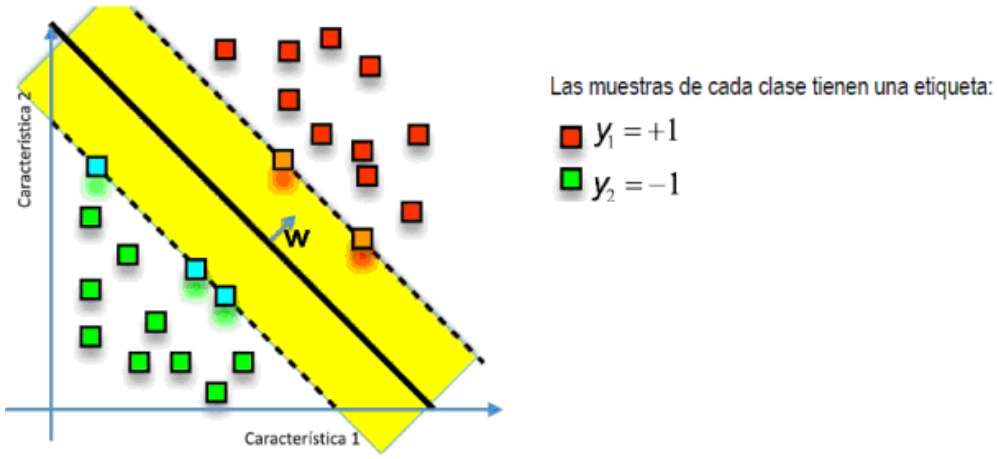


Ilustración 58: SVM, Hiperplano Solución W

La solución se da a partir de un vector w que es ortogonal al hiperplano solución y a un coeficiente de intersección b. Este hiperplano solución es el hiperplano medio entre otros dos hiperplanos h más y h menos, que son los que contienen los vectores de soporte de ambas clases respectivamente.

Hiperplanos de los vectores de soporte:

$h^+ \rightarrow w^t x_i + b = +1$
$h^- \rightarrow w^t x_i + b = -1$

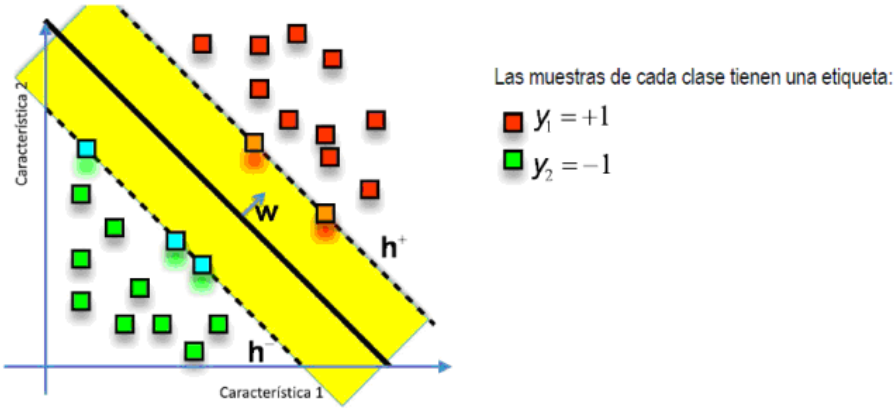


Ilustración 59: Hiperplanos de los vectores soporte

De esta forma, la condición de clasificación va a implicar que cuando se aplican los parámetros del hiperplano w a las muestras positivas, da un valor mayor que más uno, mientras que cuando se aplican a las muestras negativas da un valor inferior a menos uno.

Condición de Clasificación: $y_i(w^t x_i + b) \geq 1$

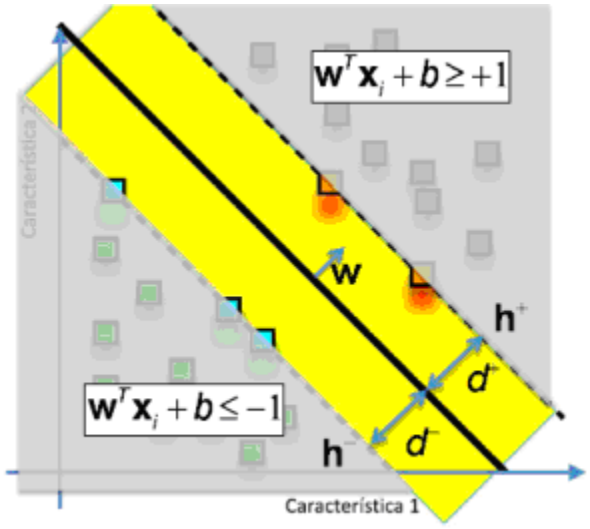


Ilustración 60: Condición de Clasificación

Estas dos condiciones se pueden expresar conjuntamente introduciendo la variable Yi que corresponde a la etiqueta de cada una de las muestras, siendo más uno para las muestras positivas y menos uno para las muestras negativas. De esta forma la condición de clasificación implica que todas las muestras clasificadas correctamente tienen que estar situadas en la región del espacio de características situada más allá del margen.

El margen corresponde a la región en amarillo en el gráfico y se puede calcular a partir de la distancia entre los dos hiperplanos h.

$$d^+ = d^- = \frac{|wx + b|}{\|w\|} = \frac{1}{\|w\|}$$

Margen = $d^+ + d^- = 2\frac{1}{\|w\|}$

Si se desarrolla matemáticamente, se observa que el margen depende únicamente de los parámetros w del hiperplano. Maximizar el margen es equivalente a minimizar el problema inverso que se expresa con esta función

Minimizar $\phi(w) = \frac{1}{2}\frac{1}{\|w\|^2}$

Sujeto a $y_i(w^t x_i + b) \geq 1$

Que va a depender únicamente del producto escalar entre w traspuesto y w.

$\|w\|^2 = w^t w$

Cuando a la minimización se le añade la condición de clasificación vista antes:

$$y_i(w^t x_i + b) \geq 1$$

Se comprueba que se ha transformado el problema de clasificación en un problema de optimización cuadrática estándar y que por lo tanto también podrá ser resuelto de manera estándar. La solución del Support Vector Machine consiste en obtener los valores w que van a dar la solución del hiperplano. Para ello se plantea el problema como la minimización de la función fi sujeta a la condición de clasificación.

Objetivo: Obtener W, b pertenecientes al hiperplano solución:

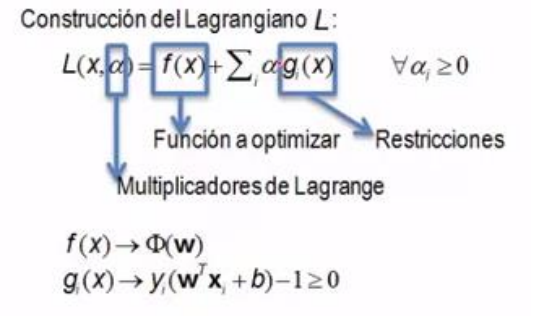
$$w^t x_i + b = 0$$

Se resuelve como un problema de optimización cuadrática:

Minimizar $\phi(w) = \frac{1}{2} w^t w$

Sujeto a $y_i(w^t x_i + b) \geq 1$

La resolución de este tipo de problemas se puede plantear con una función auxiliar que se denomina el Lagrangiano y que se construye a partir de la suma de la función a optimizar, en este caso la función fi más las restricciones a la que está sujeto el problema, en este caso, la condición de clasificación y estas restricciones se multiplican por unos factores alfa que son los que se denominan los multiplicadores de Lagrange.



De esta forma y de manera general, la solución al problema de optimización se obtiene como la minimización del Lagrangiano respecto a las variables de la función a optimizar, en este caso w y posteriormente maximizando respecto a los multiplicadores de Lagrange.

Se resuelve como problema de optimización cuadrática:

$$\left.\begin{array}{l} \underset{\mathbf{w},b}{\text{Minimizar}}\ \Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{Sujeto a:}\ \ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{array}\right\} \Rightarrow \quad \max_{\alpha}(\min_{\mathbf{w},b}(L(\mathbf{w},b,\alpha))$$

En este caso, el Lagrangiano, combinando la función a optimizar y las restricciones multiplicadas por los multiplicadores de Lagrange dan esta expresión.

El Lagrangiano L del problema SVM es:

$$L(w,b,\alpha)=\frac{1}{2}w^t w-\sum\alpha_i\left[y_i(w^t x_i+b)-1\right]$$

La minimización de esta expresión va a implicar realizar las derivadas parciales respecto a w y respecto a b igualando a cero. A partir de la primera derivada parcial con respecto a w se obtiene el valor óptimo de w como combinación lineal de las muestras de entrenamiento.

La minimización de L con respecto a w, b implica:

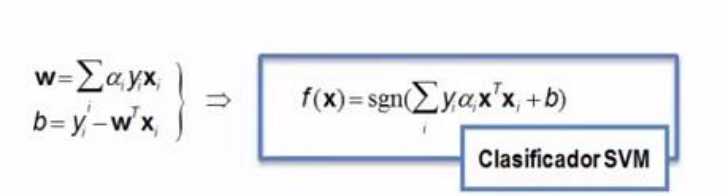
$$\frac{dl}{dw}=w-\sum\alpha_i y_i x_i=0\Rightarrow w=\sum\alpha_i y_i x_i$$

$$\frac{dL}{db}=-\sum\alpha_i y_i=0\Rightarrow\sum\alpha_i y_i=0\wedge\alpha_i\geq 0$$

Bajo condiciones de convexidad, se puede incluir esta solución de w en la fórmula de Lagrangiano y con un desarrollo matemático se llega a encontrar esta nueva formulación para el Lagrangiano.

$$L(w,b,\alpha_i)=\frac{1}{2}\left(\sum_i\alpha_i y_i x_i\right)^T\left(\sum_j\alpha_j y_j x_j\right)-\sum_i\alpha_i y_i\left(\sum_j\alpha_j y_j x_j\right)^T x_i+\sum_i\alpha_i=$$

$$=-\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j y_i y_j x_i^T x_j+\sum\alpha_i$$



De esta forma se obtiene una nueva función teta cuya maximización sujeta a la condición que se ha obtenido de la derivada respecto al parámetro b, va a dar la solución final del Support Vector Machine y es un nuevo problema de optimización que se denomina SVM dual.

SVM DUAL:

$$\underset{\alpha}{\text{Maximizar}}\ \Theta(\alpha)=-\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j y_i y_i\mathbf{x}_i^T\mathbf{x}_j+\sum_i\alpha_i$$

$$\text{Sujeto a: }\sum_i\alpha_i y_i=0\quad\alpha_i\geq 0$$

Así, a partir de la formulación original que se denomina SVM primal se tiene un nuevo problema de optimización cuya solución es la solución del Support Vector Machine y que se denomina SVM dual. La resolución de este nuevo problema de optimización se puede realizar con métodos estándar.

En primer lugar, se observa que la solución w, viene dada como combinación lineal de las muestras de entrenamiento.

$$w = \sum_i \alpha_i y_i x_i$$

Pero no de todas ellas, porque algunos multiplicadores de Lagrange van a ser igual a cero. Por lo tanto, solo va a ser resultado de la combinación lineal de aquellas muestras que tengan asociado un multiplicador de Lagrange mayor que cero. Estos multiplicadores de Lagrange mayor que cero, van a definir qué muestras constituyen los vectores de soporte.

También es interesante observar que en el problema dual las muestras aparecen como productos escalares. Este hecho se usa a continuación para poder adaptar el problema del Support Vector Machine a los casos en que no sean linealmente separables utilizando el concepto de Kernel. Finalmente la solución del Support Vector Machine que se ha obtenido de esta forma es no paramétrica, lo que quiere decir que no hay necesidad de ajustar ningún parámetro durante el entrenamiento.

Sin embargo, en el caso general cuando los datos no son linealmente separables y se tiene que introducir tolerancia a errores de clasificación, donde sí va a ser necesario ajustar algunos parámetros. La formulación dual del problema, ayuda a solucionar el caso en que los datos no son linealmente separables. Una estrategia consiste en definir una función de mapeo que convierta el espacio de características original, en un nuevo espacio de características de una dimensionalidad mayor y con este nuevo espacio de características los datos sí sean linealmente separables.

10.4.2 CASO NO LINEALMENTE SEPARABLE: TRUCO DEL KERNEL

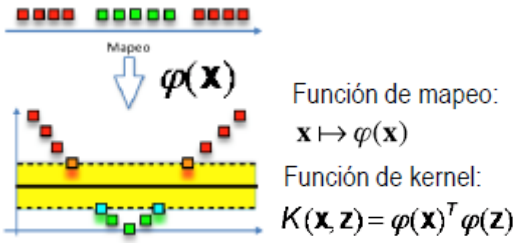
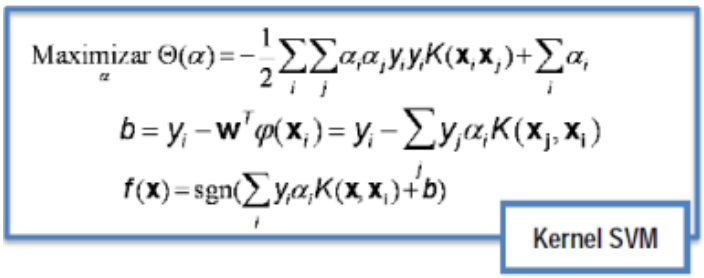


Ilustración 61: Truco del Kernel

Sin embargo, dado que en la formulación dual tiene únicamente productos escalares entre los vectores, no va a ser necesario definir explícitamente esta función de mapeo Ro sino que va a ser suficiente con definir un Kernel que defina el producto escalar en este nuevo espacio de características.

Este Kernel es por lo tanto un producto escalar que se puede integrar fácilmente en la formulación dual del problema.



Además otra de las ventajas va a ser que, no va a ser necesario definir para cada problema un nuevo kernel sino que ya existen diferentes tipos de kernel estándar que en la práctica resultan bastante útiles.

10.4.3 EJEMPLOS DE FUNCIÓN KERNEL

Entre ellos se tienen el kernel polinomial, el kernel de base radial, el kernel Sigmoide, Multi-cuadrático inverso o el kernel de intersección. Será importante tener en cuenta que estas funciones de kernel introducen algunos parámetros en su formulación y que estos parámetros van a ser necesario ajustarlos durante el proceso de entrenamiento utilizando variación cruzada con algún conjunto de validación.

Polinomial:	$K(\mathbf{x},\mathbf{z}) = \langle \mathbf{x},\mathbf{z}\rangle^d$
Funciones de base radial:	$K(\mathbf{x},\mathbf{z}) = e^{-\|\mathbf{x}-\mathbf{z}\|^2/2\sigma}$
Sigmoide:	$K(\mathbf{x},\mathbf{z}) = \tanh(\kappa\langle \mathbf{x},\mathbf{z}\rangle - \delta)$
Multi-cuadrático inverso:	$K(\mathbf{x},\mathbf{z}) = (\|\mathbf{x}-\mathbf{z}\|^{1/2} 2\sigma + c^2)^{-1}$
Kernel de intersección:	$K(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{n}\min(x(i), z(i))$

10.4.4 MARGEN BLANDO: FACTOR DE REGULARIZACIÓN

La segunda alternativa para tratar el caso de datos que no sean linealmente separables consiste en suavizar la condición del margen e introducir tolerancia a errores de clasificación. La suavización del margen se va a implementar en la formulación matemática definiendo un conjunto de variables, las que se conocen como variables de holgura, en inglés Slack Variables, que dan cuenta de aquellos vectores de soporte que violan la condición del margen. Estas variables de holgura se incorporan en la formulación del problema relajando la condición de clasificación y cómo un nuevo factor a tener en cuenta en la función a minimizar.

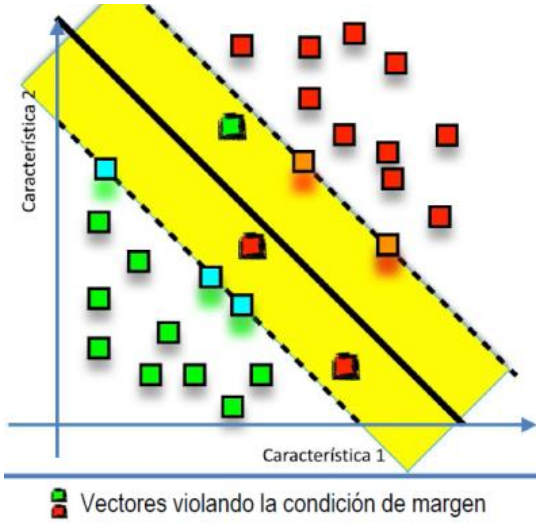


Ilustración 62: Conjuntos no linealmente separables: Margen suave

Variables de holgura: $\xi_i \geq 0$

SVM Primal:

Minimizar $\Phi(w) = \frac{1}{2} w^T w + C \sum_i \xi_i$

Sujeto a: $y_i(w^T \phi(w_i) + b) \geq 1 - \xi_i$

El impacto de las variables de holgura en la formulación queda modulado por el parámetro C y este parámetro puede ser interpretado como un parámetro de regularización que permitirá controlar el equilibrio entre maximizar el margen y minimizar el error en las muestras de entrenamiento. Cuando este parámetro es muy grande va a anular la aproximación de margen suave y va a tener que ser fijado de manera estándar con variación cruzada a partir de un conjunto de validación. En la formulación dual, las variables de holgura, van a implicar incluir una nueva inecuación que va a modular la contribución de los vectores con alfa distinto de cero. Se observa que en la solución final del clasificador, no influyen de manera explícita ni las variables de holgura ni el parámetro C.

SVM Dual:

Maximizar $\Theta(\alpha) = \frac{1}{2} \sum_i \sum_i \alpha_i \alpha_j y_i y_j K(x_i x_j) + \sum_i \alpha_i$

Sujeto a: $0 \leq \alpha_i \leq C \wedge \sum_i \alpha_i y_i = 0$

$b = y_i(1 - \xi_i) - \sum_j y_j \alpha_i K(x_j, x_i)$

$f(x) = \text{sgn}(\sum y_i \alpha_i K(x_j, x_i) + b)$

10.5 GENERACIÓN DE VENTANAS CANDIDATO

En la secuencia de procesamiento que da lugar a un detector de objetos se ha visto hasta ahora el módulo de clasificación. Dada una secuencia de ventanas, el módulo de clasificación determina si contenía o no contenía el objeto que interesa. Ahora se verá cómo se generan esos candidatos, es decir, dada una imagen cómo se obtienen esas ventanas que posteriormente se han de clasificar. Con el conjunto de generador de candidatos y el clasificador de los mismos se verá que dada una imagen y un objeto concreto, se va a obtener una serie de lo que se puede llamar pre-detecciones o potenciales detecciones. Entonces se necesitara un módulo que refine todas estas decisiones que han dicho que hay un objeto en la ventana que se ha examinado, de forma que finalmente para cada uno de los objetos que hay en la imagen como máximo, se tenga una sola detección. En otras palabras en la generación de ventanas candidato, es necesaria la refinación de múltiples pre-detecciones.

Ilustración 63: Refinación de la decisión

10.5.1 GENERACIÓN DE CANDIDATOS - VENTANA DESLIZANTE

Dada una ventana, que se llamara canónica, colocada en una cierta posición de la imagen, se tiene un determinado paso en el eje vertical y con ello se genera otra nueva ventana del mismo tamaño, pero que está desplazada según este paso en este eje. Y lo mismo se hace para el eje horizontal. Estos parámetros, el paso en x y el paso en y, van a servir para generar un patrón de desplazamiento de la ventana canónica mediante un algoritmo de desplazamiento, obteniéndose una serie de ventanas

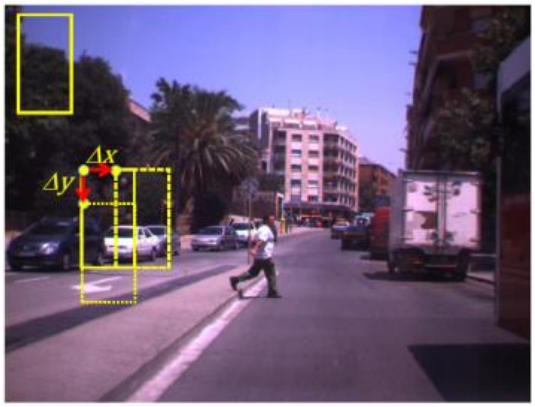


Ilustración 64: Ventana Deslizante, pasos en X e Y

Luego el clasificador las clasifica como conteniendo un peatón o no. Por ejemplo en el caso de abajo, tres de las ventanas examinadas, han sido clasificadas como conteniendo un peatón.

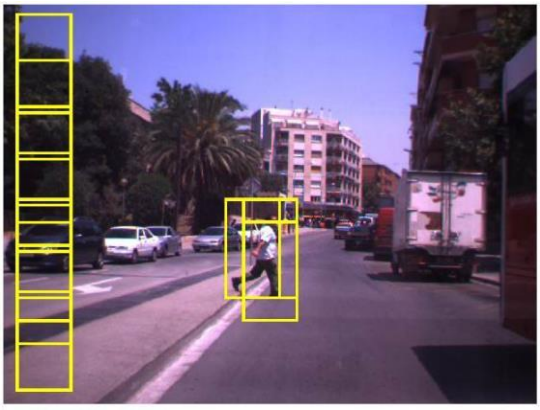


Ilustración 65: Ventana Deslizante, 3 ventanas conteniendo un peatón

Una cuestión importante aquí es evitar cálculos redundantes ya que para cada ventana se tendrá que calcular su descriptor para clasificarla. Cuando dos ventanas tienen un cierto solapamiento, en esa zona comparten la misma sub-imagen, por tanto, puede pasar que se esté calculando de forma redundante varias veces los mismos componentes del descriptor, por ejemplo códigos LBP de los píxeles de la zona donde las ventanas se solapan. Para evitar esos cálculos redundantes se modifica la secuencia de procesamiento. Suponiendo que se utilizan descriptores basados en códigos LBP. A partir de una imagen en color, se la transforma en intensidad de gris y luego se calcula el código LBP de todos los píxeles que hay en la imagen. Luego se aplican el procedimiento de ventana deslizante a la imagen con los códigos LBP. Por lo tanto, lo que llega al módulo de clasificación son ventanas de la imagen con los códigos LBP calculados.

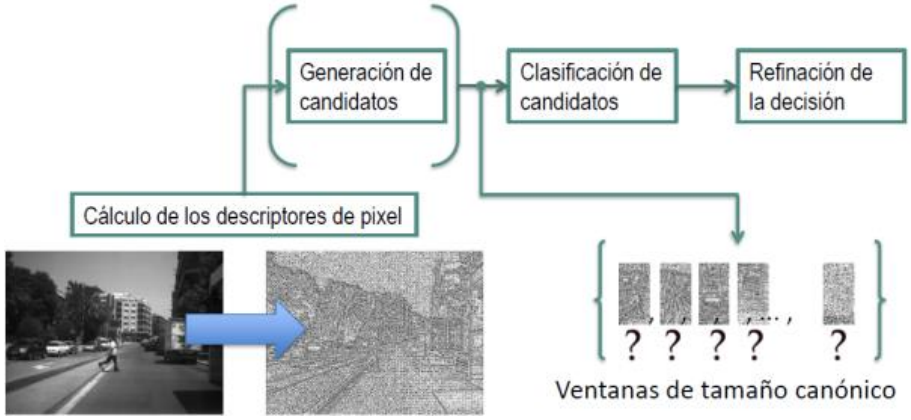


Ilustración 66: Generación de candidatos

Otra cuestión a tener en cuenta respecto a la ventana deslizante es que la altura y anchura de la ventana canónica determina el tamaño de los objetos que se detectan. En principio, solo se pueden detectar objetos que son de tamaños iguales o muy parecidos. Para clasificar objetos se necesita un modelo de esos objetos, y esos modelos, se aprenden a partir de un conjunto de muestras. Todas las ventanas han de tener el mismo tamaño que es justamente el tamaño canónico. En las imágenes originales los objetos de interés pueden haber tenido distintos tamaños por lo que se hace es una interpolación o re-escalado de cada una de esas imágenes.

El descriptor de una ventana consiste, en el caso de los LBP, en una serie de bloques donde para cada bloque se extrae el histograma de códigos LBP y luego son concatenados esos histogramas y esa concatenación de histogramas es el descriptor de la ventana.

Ilustración 67: Bloques LBP

Por lo tanto se puede buscar un paso en x y un paso en y del algoritmo de ventana deslizante que permita reaprovechar el cálculo de esos histogramas, no solo el cálculo de los códigos LBP de cada píxel, sino también el cálculo de los histogramas.

Así la nueva secuencia de procesamiento es esta:

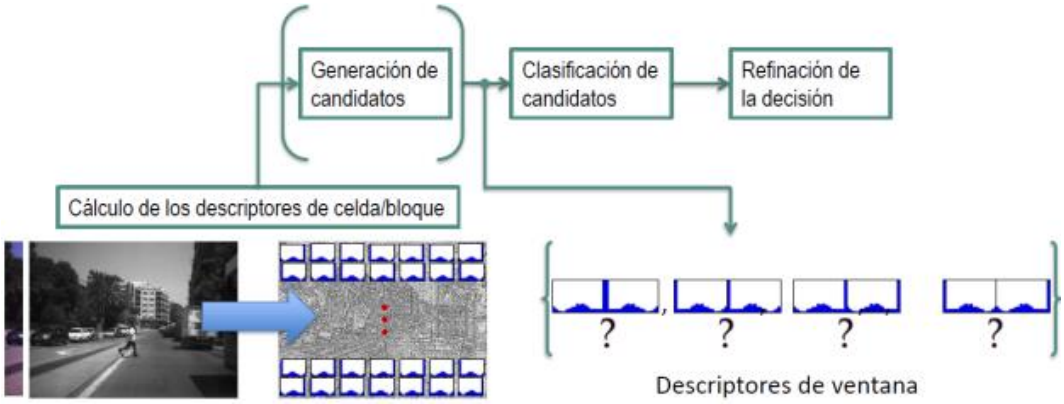


Ilustración 68: Generación de candidatos, secuencia de procesamiento

A partir de una imagen RGB, se la transformara a su imagen de intensidad, luego se calculan los códigos LBP de cada uno de los píxeles de la imagen, y también los histogramas de cada uno de los bloques de la imagen. El procedimiento de ventana deslizante concatena distintos bloques, o dicho de otra manera, distintos histogramas de códigos LBP. Así al clasificador de candidatos le llegan directamente los descriptores y se ahorra también el cálculo de histogramas de forma redundante.

10.5.2 GENERACIÓN DE CANDIDATOS - PIRÁMIDE

Se ha visto cómo a partir de una imagen se pueden generar distintas ventanas para su posterior clasificación. Estas ventanas tienen un tamaño fijo canónico y por tanto solo sirven para detectar objetos que aparezcan con el mismo tamaño dentro de una imagen. Ahora sé vera cómo se puede detectar objetos que aparecen a distintos tamaños dentro de la misma imagen. Una ventana canónica permite detectar objetos que tienen el mismo tamaño que esa ventana pero no otros que puedan tener un tamaño mayor o menor.

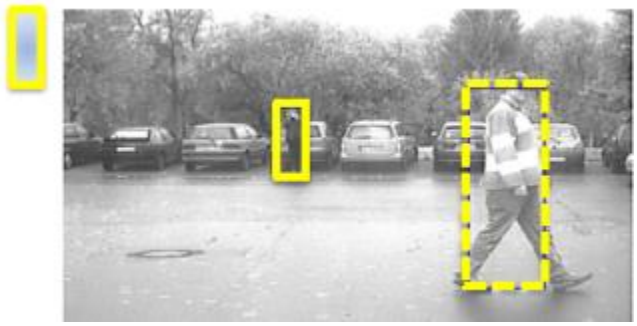


Ilustración 69: Ventana canónica y no canónica

Por lo tanto, es necesario simular la situación en la que se tienen ventanas canónicas que tienen distintos tamaños. En el contexto de la ventana deslizante eso sería primero hacer un barrido de la imagen utilizando un tamaño de ventana canónica y después hacer barridos con otros tamaños. Una solución consiste en que dada una imagen original, producir imágenes re-escaladas de ésta, cada vez de menor tamaño, de forma que todas juntas aparecen como una pirámide, teniendo una sola ventana de un tamaño canónico, y esa ventana es la que se desplaza por los distintos niveles de la pirámide. En cada uno de esos niveles se realiza el mecanismo de la ventana deslizante. El mecanismo se conoce como pirámide con ventana deslizante. En este mecanismo no se re-escalan las ventanas sino que se re-escala la imagen formando una pirámide, e introduce un nuevo parámetro, el grado de re-escalado: s^i con i = 0 para la imagen original

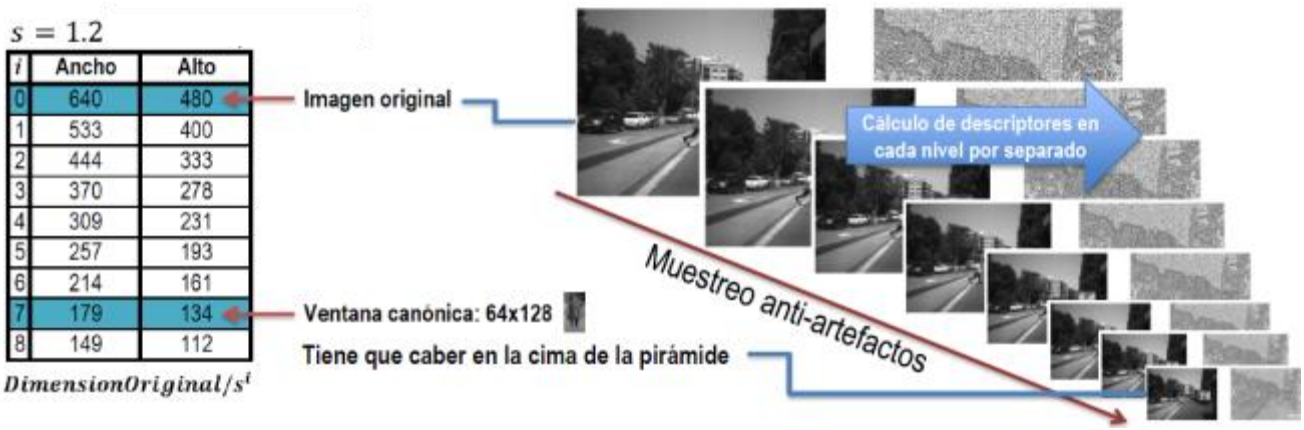


Ilustración 70: Pirámide con ventana deslizante

En este caso, aparte de los parámetros de la ventana deslizante, es decir, el paso en X y en Y, de los desplazamientos, se tendrá otro parámetro, S elevado a i, de los desplazamientos donde en el caso i igual a cero corresponde a la imagen original. Este S elevado a i es el que controla los niveles que habrá en la pirámide. Se parte de una imagen original, es decir i =0, de 640 por 480 píxeles y con un valor de S de 1.2.

En este caso, por ejemplo, como se ve aquí con este valor de S y esta imagen original el siguiente nivel de la pirámide tendrá una imagen de 533 por 400 píxeles, (640/1.2 y 480/1.2). En el siguiente 444 por 333 y así sucesivamente hasta que llega un punto en el la imagen es demasiado pequeña como para que siquiera la ventana canónica quepa en esa imagen. En este ejemplo concreto, se tienen los niveles desde el cero al siete de la pirámide, ocho niveles de pirámides.

Para cada una de estas versiones escaladas de la imagen, se calcula su correspondiente descriptor por píxel o su correspondiente descriptor de histogramas.

Ilustración 71: Pirámide deslizante, cálculo de descriptores en cada nivel

Con este mecanismo los objetos que están lejos, se detectan con la ventana canónica en la imagen original, es decir, en la base de la pirámide, y los objetos que están cerca se detectan en la cima de la pirámide. El tamaño de la ventana canónica determina el tamaño mínimo de los objetos detectables.

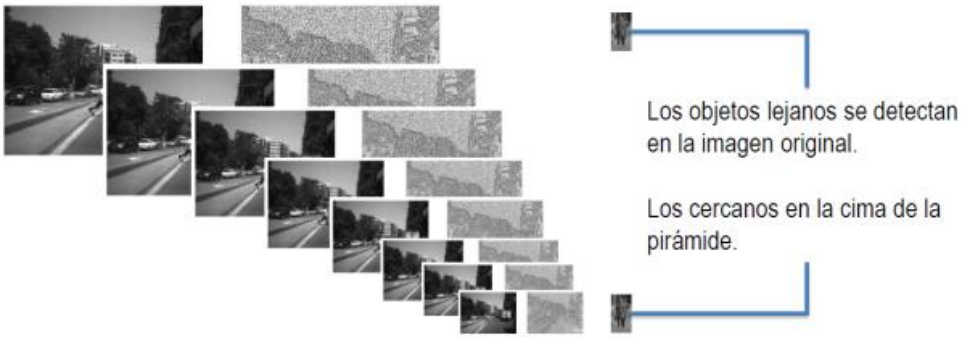


Ilustración 72: Pirámide deslizante, el tamaño de la ventana determina los objetos detectables

10.5.3 GENERACIÓN DE CANDIDATOS - REFINACIÓN

Con el mecanismo combinado de pirámide y ventana deslizante sucede que para un mismo objeto se pueden obtener más de una detección e interesa tener solo una.

Tomando de ejemplo la siguiente imagen:

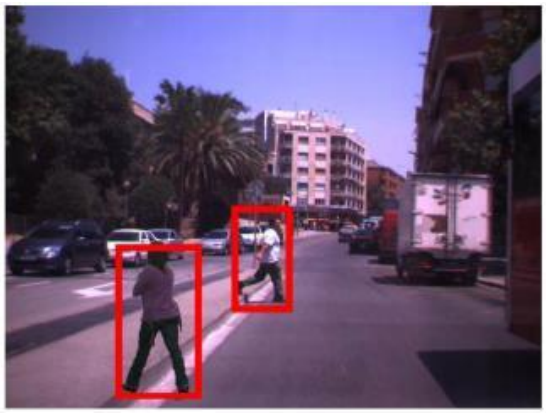


Ilustración 73: Refinación de la decisión

Dentro de la secuencia de procesamiento, el módulo de clasificación de candidatos determina que una serie de ventanas de la imagen contienen los objetos que interesan, pero el problema es que hay redundancia y algunas ventanas contienen el mismo objeto, por tanto la tarea del módulo de refinación es eliminar esa redundancia.

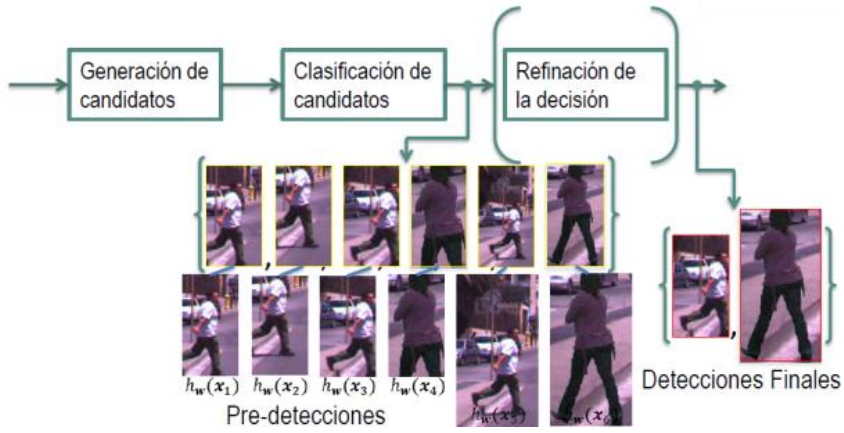


Ilustración 74: Refinación de la decisión

Por lo tanto, se quiere pasar de estas pre-detecciones a las detecciones finales donde sí se tiene una sola ventana por cada uno de los objetos.

Para lograr esto se usa un algoritmo, que en inglés conoce como **"Non-Maximum Suppression" (NMS)**, del que se verá un ejemplo. Suponiendo que se tiene una situación donde se han pre-detectado cuatro personas, cada una de ellas con distinto número de pre-detecciones como se ve en la siguiente imagen.

El primer paso del algoritmo es ordenar esas pre-detecciones de mayor a menor valor según su confidencia o probabilidad que viene asignado desde el módulo de clasificación: el valor que ha devuelto para cada una de esas ventanas la función logística. A ese conjunto de pre-detecciones, ordenadas de esa manera, se le llama P. El siguiente paso es crear una lista o un conjunto de grupos o en inglés clusters, llamado G y que inicialmente estará vacío.

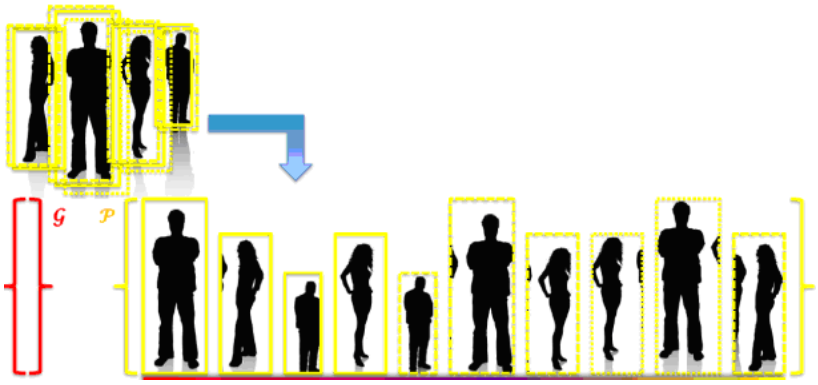


Ilustración 75: Primer paso del NMS

El tercer paso del algoritmo consiste en un procedimiento iterativo que se va a ir repitiendo hasta que el conjunto P esté vacío. Primero se toma la primera pre-detección del conjunto P y se mira si se solapa lo suficiente con algunos de los grupos que hay en G. Lo suficiente puede ser, por ejemplo, el 50% del área de esa pre-detección. Si no se solapa lo suficiente con ningún grupo de los que hay en G, entonces se crea un grupo nuevo con la pre-detección en G y se elimina de P.

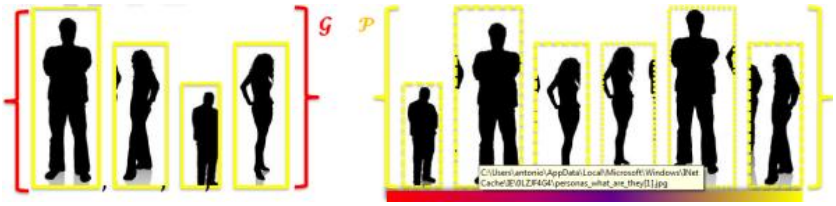


Ilustración 76: Segundo paso del NMS

Finalmente G contiene las detecciones finales del proceso.

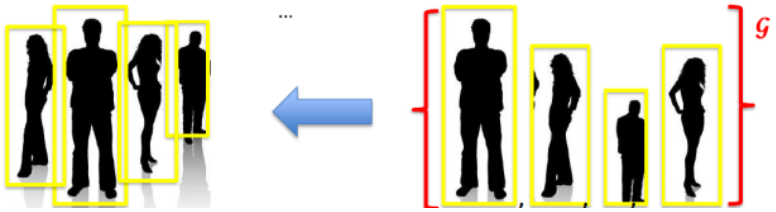


Ilustración 77: Fin del NMS

Este tipo de algoritmo se lo conoce como "Non-Maximum Suppression". Hay otros algoritmos más sofisticados, pero no parece haber alguno que sea el mejor en todos los casos. Algunos son más sofisticados pero también son más lentos en ejecución.

Otra cuestión es que no se ha tenido en cuenta el tamaño que tenían las pre-detecciones, dando lo mismo que fueran ventanas más grandes o más pequeñas. Puede ser que para algunas aplicaciones deba ser tenido en cuenta el tamaño de esas ventanas.

10.6 EVALUACIÓN DEL RENDIMIENTO DEL CLASIFICADOR

Es necesario evaluar el rendimiento de los sistemas detectores de objetos que se estén diseñando. Primero se evaluara el rendimiento única y exclusivamente del clasificador aplicado sobre un conjunto de ventanas candidatas. Así se definirá la matriz de confusión y sobre esta matriz de confusión se definirán dos medidas, la exactitud y la precisión. Si se parte del esquema general, el objetivo final será cuantificar cómo funciona el detector.

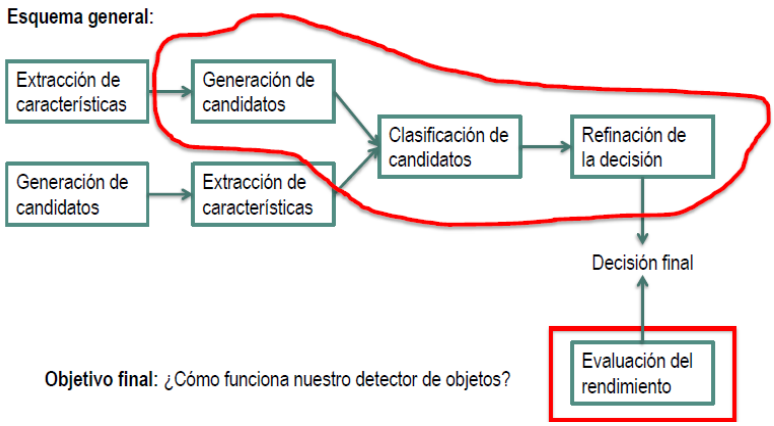


Ilustración 78: Esquema General del detector

Dado un conjunto de ventanas candidatas, la respuesta del clasificador podrá ser correcta o no. La pregunta es, ¿cómo se puede evaluar el resultado de un clasificador? Con un conjunto de candidatos, se representa el clasificador como una circunferencia y todos aquellos objetos que queden dentro de la circunferencia serán los clasificados como positivos por el clasificador, en este caso personas.

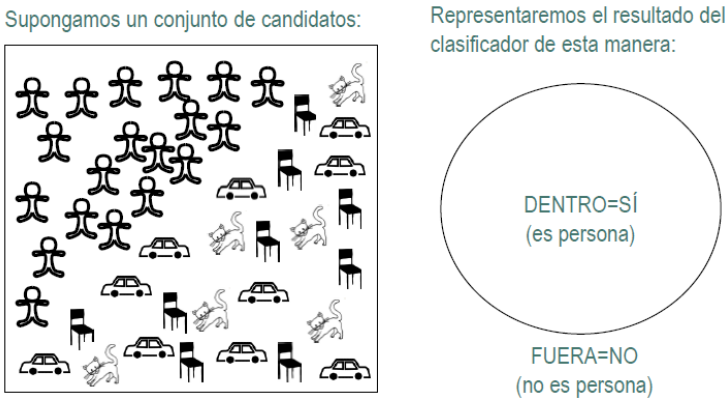


Ilustración 79: Conjunto de candidatos

Los candidatos que queden fuera de la circunferencia representarán los clasificados como negativos por el clasificador. Suponiendo el siguiente resultado:

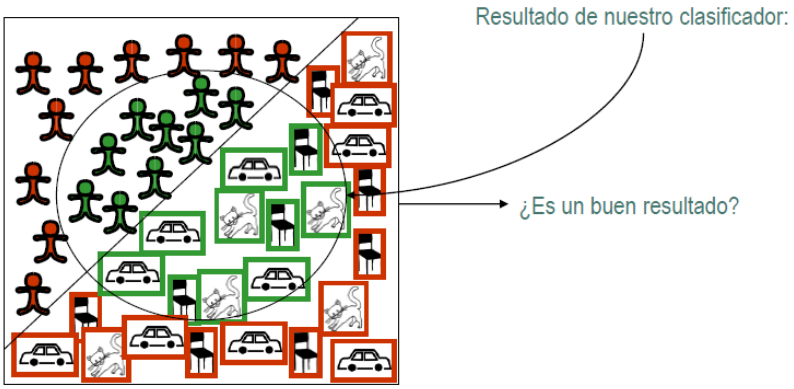


Ilustración 80: Conjunto de candidatos y resultado del clasificador

Las ventanas marcadas en verde han sido clasificadas como personas, las marcadas en rojo han sido clasificadas como no personas. “Ground Truth”, es un término que viene esencialmente de teledetección, se usa para referirse a la etiqueta que tiene cada punto de un terreno en imágenes satélites, indica el tipo de suelo que está representando ese punto. En aprendizaje computacional se usa para referirse al resultado correcto que debería dar un clasificador. El Ground Truth corresponderá al conjunto de ventanas de las imágenes correctamente etiquetadas. La Matriz de confusión es la herramienta básica para visualizar el rendimiento de un clasificador. Si se busca la clase persona, es como tratar dos clases, la de persona y la clase no persona. La matriz de confusión es una matriz de dos por dos que representa el comportamiento del clasificador sobre estas dos clases. Las filas de la matriz representan las instancias reales de las clases que vienen dadas por el Ground Truth. Las columnas de la matriz representan las predicciones del clasificador analizado, en este caso el clasificador de persona y por consiguiente el de no persona.

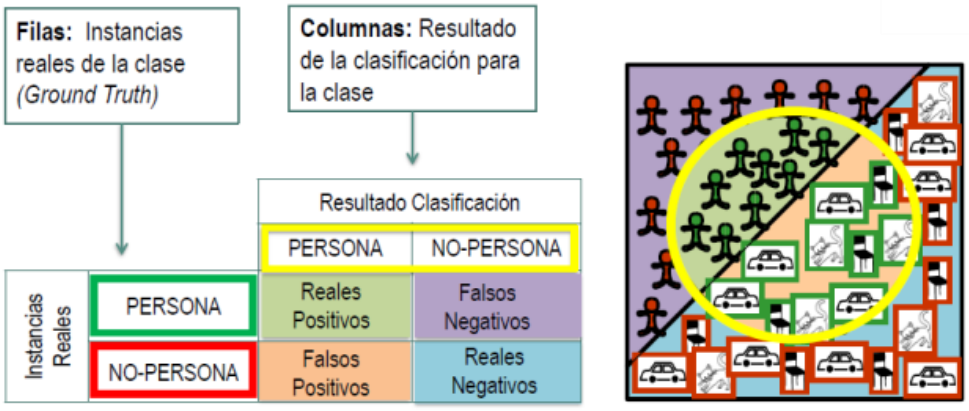


Ilustración 81: Matriz de confusión

En la primera columna se tienen las respuestas positivas que el clasificador ha clasificado como personas. En la primera fila de esta columna se tienen los reales positivos que representan los candidatos que han sido correctamente clasificados como personas y corresponden al fondo verde del dibujo. En la segunda fila de esta columna se tienen los falsos positivos que representan los candidatos que han sido incorrectamente clasificados como personas y corresponden al fondo naranja del dibujo. En la segunda columna se tienen las respuestas negativas del clasificador. En la primera fila de esta columna se tienen los falsos negativos que representan a los candidatos que son persona y que han sido incorrectamente clasificados como no personas, corresponden al fondo violeta. Mientras que en la segunda fila de esta columna se tienen a los reales negativos. Estos representan los candidatos que han sido correctamente clasificados como no personas y que corresponden al fondo azul del gráfico.

10.6.1 ANÁLISIS DE LA MATRIZ DE CONFUSIÓN: LA EXACTITUD Y LA PRECISIÓN

Para poder analizar la matriz de confusión se definen dos medidas, la exactitud y la precisión. La exactitud mide la proximidad entre el resultado global del clasificador y la clasificación perfecta. Se calcula cuantitativamente como el cociente entre la suma de reales positivos y reales negativos dividido por el número total de candidatos que han sido clasificados y que corresponden al número total de elementos que están dentro de la matriz de confusión.

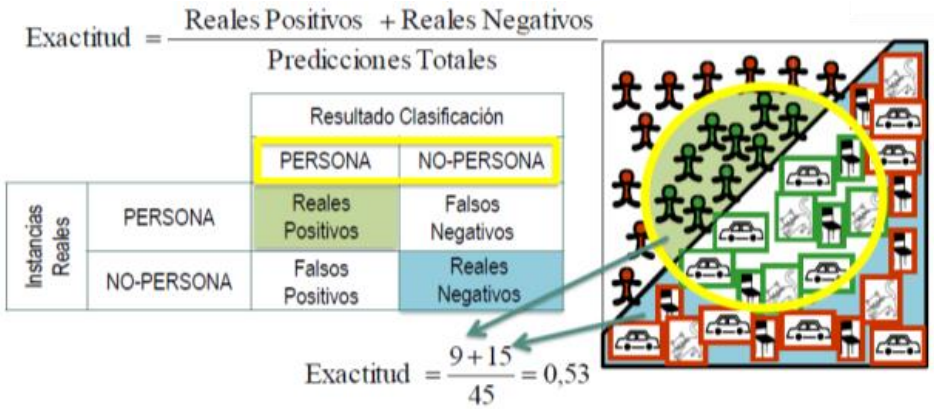


Ilustración 82: Exactitud, Proximidad entre el resultado y la clasificación exacta

La segunda medida es la precisión que mide la calidad en las respuestas positivas del clasificador. Se calcula cuantitativamente como el cociente entre reales positivos y la suma de todos los positivos dados por el clasificador, tanto los reales como los falsos. Analiza la primera columna de la matriz de confusión y corresponde al siguiente cálculo: personas bien clasificadas dividido por la suma de personas bien clasificadas más no personas clasificados como personas.

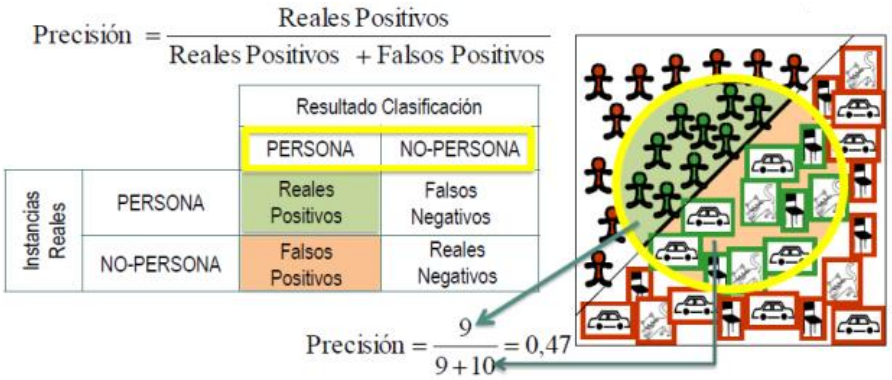


Ilustración 83: Precisión, calidad de la respuesta del clasificador

La precisión en este caso es de 0,47 y dice que el clasificador da más respuestas no correctas que correctas.

10.6.2 ANÁLISIS DE LA MATRIZ DE CONFUSIÓN: SENSIBILIDAD Y ESPECIFICIDAD

A partir de los resultados del clasificador, se construye la matriz de confusión y se definen dos medidas generales sobre el clasificador que ayuda a decidir los mejores parámetros para el clasificador.

La sensibilidad mide la eficiencia en la clasificación de todos los elementos que son de la clase. Se calcula con el cociente entre los reales positivos, dividido por la suma del total de instancias reales que se tienen en la clase. Por lo tanto, analiza la primera fila de la matriz de confusión.

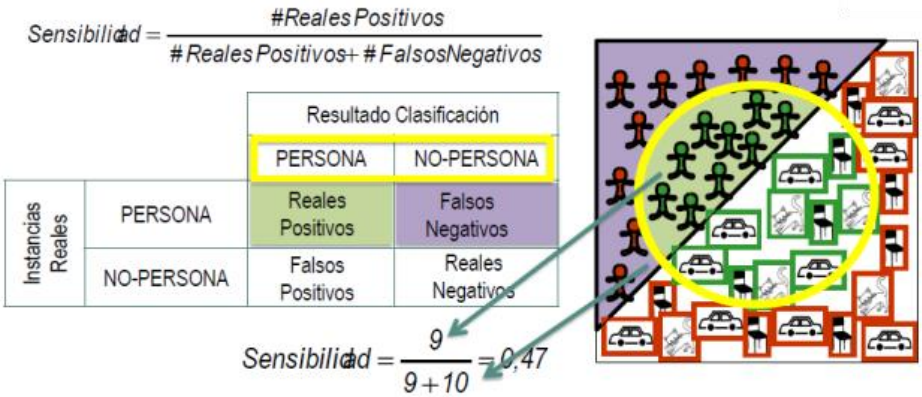


Ilustración 84: Sensibilidad, Eficiencia en la clasificación de los elementos que son de la clase

La especificidad, al igual que la sensibilidad, evalúa la eficiencia en la clasificación, pero en la clase complementaria. Es decir, la eficiencia en la clasificación de los elementos que no son de la clase. Se calcula como el cociente entre los reales negativos dividido por la suma del total de todas las instancias reales que se tienen en la otra clase. Por lo tanto, analiza la segunda fila de la matriz de confusión.

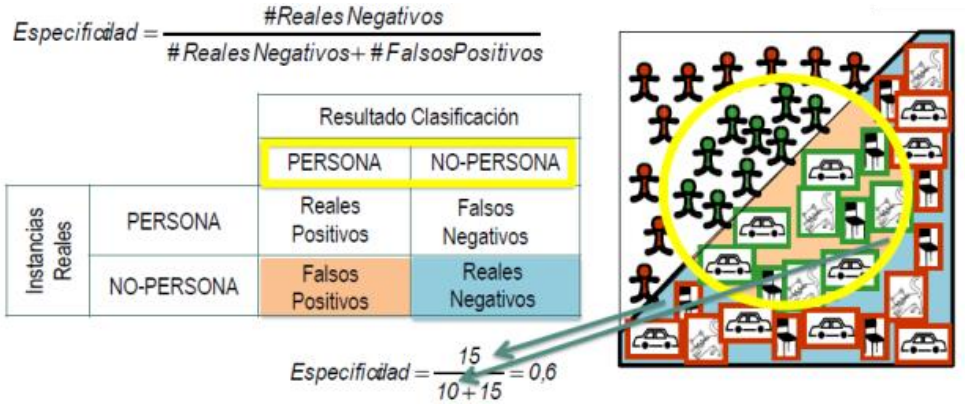


Ilustración 85: Especificidad, Eficiencia en la clasificación de todos los elementos que no son de la clase

La especificidad en este caso es de 0.6, que es mayor que 0.5, y por tanto indica que el número de no personas bien clasificadas como no personas, es mayor que las no personas que han sido incorrectamente clasificadas como personas.

Hay una relación entre la especificidad y sensibilidad con dos razones usadas habitualmente. Estas son:

- True Positive Rate: Tasa de Reales Positivos es la Especificidad
- False Positive Rate: Tasa de Falsos Positivos es (1 – Sensibilidad)

Ilustración 86: Resumen de medidas sobre la matriz de confusión

En este punto, la pregunta es, ¿Cómo usar el análisis de estos resultados para mejorar el clasificador? Cualquier clasificador presenta siempre algún parámetro que se relaciona con el umbral que va a determinar la decisión entre ser o no ser de la clase.

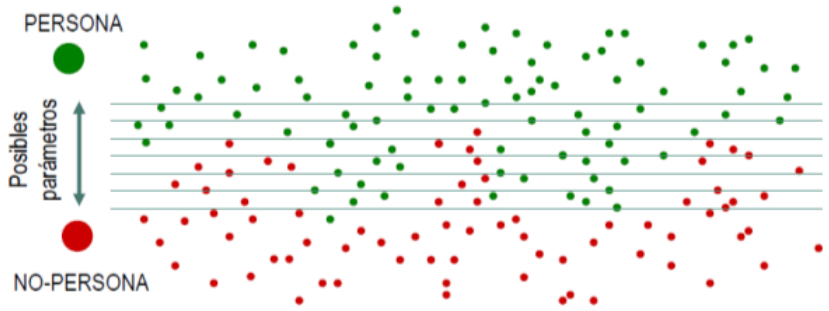


Ilustración 87: Ejemplo de una frontera entre dos clases, persona y no persona

En verde se tienen los elementos que pertenecen a la clase persona, y en rojo los elementos que pertenecen a la clase no persona. La variación de un parámetro, que provoque que se aproxime la frontera de decisión hacia la clase persona, hace que aumente la especificidad y disminuya la sensibilidad. En cambio, si se aleja la frontera de la decisión de la clase persona, entonces se disminuye la especificidad y se aumenta la sensibilidad. Cada umbral determinará, un clasificador diferente. Por lo tanto, la variación de estos parámetros determina una familia de clasificadores cuyos resultados pueden ser estudiados globalmente.

10.7 EVALUACIÓN DEL RENDIMIENTO DEL DETECTOR

Para evaluar el rendimiento del detector ser requiere de un conjunto de imágenes de evaluación con su Ground Truth correspondiente. El Ground Truth tiene un conjunto de ventanas por cada imagen y el sistema detector retorna una/s ventana/s dentro de la imagen ingresada.

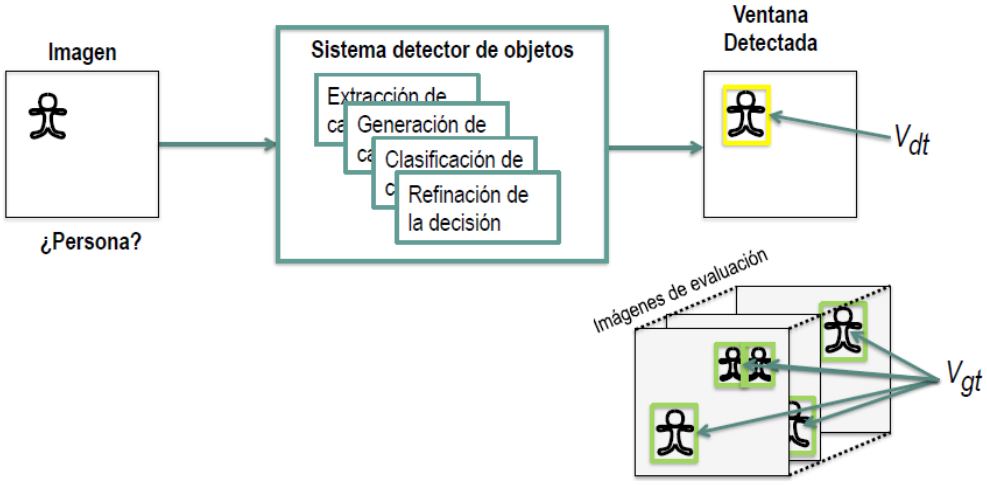


Ilustración 88: Evaluación de un sistema detector

El resultado de cualquier sistema detector es una o varias entradas dentro de la imagen dada, que se llaman ventanas detectadas Vdt. Para poder evaluar si este resultado es bueno, se necesita el Ground Truth sobre el

conjunto de imágenes de evaluación. El Ground Truth de detección será un conjunto de ventanas por cada imagen del conjunto de evaluación, estas se llaman ventanas Vgt por Ground Truth. Una ventana detectada será correcta si su grado de solapamiento con una ventana del Ground Truth es suficiente. A las dos ventanas con potencial de solapamiento Vdt, se las representa en verde y a las Vgt en amarillo, que es la ventana que ha retornado el detector.

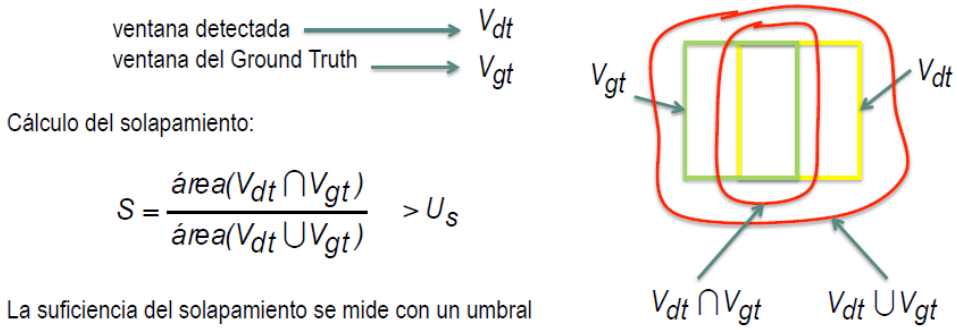


Ilustración 89: Umbral de solapamiento

El cálculo del solapamiento de estas dos ventanas vendrá dado por el cociente entre el área de intersección de las dos ventanas dividido por el área de la unión de las dos ventanas y para determinar si el solapamiento es suficiente se compara con un umbral. Este cociente es el llamado parámetro s. El umbral que se aplique sobre esta variable determinará los conjuntos de reales positivos y de falsos positivos.

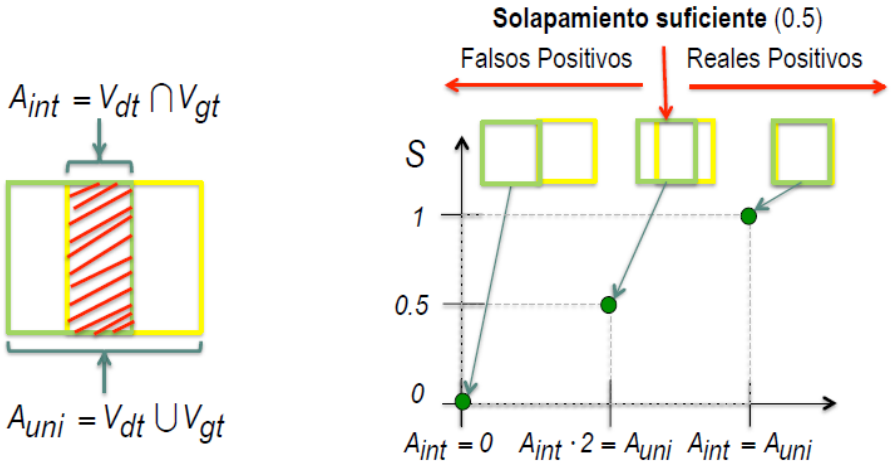


Ilustración 90: Umbral de solapamiento suficiente

Los falsos positivos más claros son aquellos en los que no existe solapamiento con el Ground Truth. Otros falsos positivos son aquellos en los que existe solapamiento, pero no suficiente. Finalmente se encuentra con el caso de múltiples detecciones sobre una misma ventana del Ground Truth, en las cuales, una de ellas da más solapamiento

y será considerada como real positivo, y el resto serán consideradas falsos positivos y por tanto penalizarán el comportamiento global del detector.

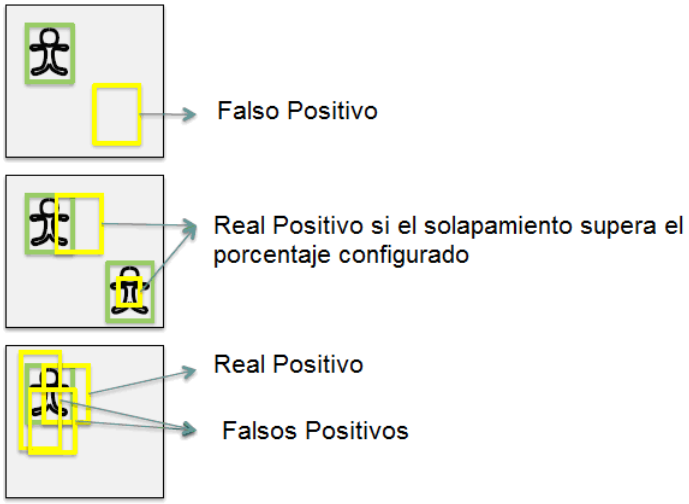


Ilustración 91: Criterios para reales y falsos positivos

Cada muestra de testeo tiene su correspondiente archivo plano donde se indican, en una serie de líneas, 4 números. Esto significa que cada línea representa una ventana y los 4 números enteros son el centro de la ventana, (x, y) y su a ancho y altura (W, H). Por lo tanto, en cada muestra se tiene marcada la ventana donde se encuentra cada una de las personas que allí aparecen. En este trabajo se toma como criterio que el objetivo buscado es detectar a la persona y no el realizar un marco exacto alrededor de la persona. Por lo tanto, si en un caso el programa se detecta, por ejemplo, un marco que abarca solo las piernas de una persona, se considera una detección aceptada. Se considera error si detectara otro marco con otra parte de la misma persona. Esto último se penaliza en las estadísticas. En otras palabras se busca que el programa marque una ventana que encierre alguna parte de la persona y solo exista una ventana por persona. Para lograr este efecto, los parámetros de solapamiento fueron fijados a valores bajos e iguales para todos los modelos.

Ahora se definirá una serie de medidas que cuantifique estas detecciones. Primero se define la tasa de detección, que es la razón entre el número de buenas detecciones y el número total de objetos dados por el Ground Truth. Esta medida se calcula globalmente sobre las ventanas.

Se define también la tasa de error, que se calcula como uno menos la tasa de detección. Finalmente se define la tasa de falsos positivos por imagen, que es la razón entre el número total de falsos positivos detectados en todo el conjunto de evaluación dividido por el número total de imágenes que se tiene en este conjunto de evaluación.

Resumiendo, las medidas para la evaluación del comportamiento global del detector son:

Tasa de detección: Es la razón entre el número de objetos detectados, (Reales Positivos), y el número de objetos en el Ground Truth. Tasa de detección=#Reales Positivos/#Objetos Ground Truth

Tasa de error es el complemento de la tasa de detección. Tasa de error = 1 – Tasa de detección

Falsos Positivos por imagen, (FPPI), son las detecciones incorrectas por imagen. FPPI = #Falsos Positivos/#Imágenes en Ground Truth

Todas las medidas se calculan a partir de un umbral de solapamiento que se ha fijado a priori. Si se compara el rendimiento de diferentes detectores, la comparación habitual es la tasa de error versus la tasa de falsos positivos por imagen.

Para realizar una comparación de detectores, se representa la tasa de error versus el FPPI

Ilustración 92: Comparación de detectores

En esta gráfica se introduce el efecto de los parámetros del clasificador, donde para cada posible umbral del clasificador se introduce el punto correspondiente a la tasa de error y la tasa de falsos positivos por imagen. Siempre para un mismo umbral de solapamiento igual para todos los clasificadores. De esta manera se construirá una curva que es decreciente, a medida que se va variando el umbral de clasificación, la tasa de error varía inversamente a la tasa de falsos positivos por imagen. Una buena medida del comportamiento global del clasificador se puede dar calculando el área bajo esta curva. Valores bajos de esta área indicaran un buen comportamiento del clasificador. En la evaluación por imagen está incluido el comportamiento del detector en su conjunto, por lo tanto, se está incluyendo el comportamiento de los módulos de generación de candidatos, de clasificación y de refinamiento de la decisión. Mientras que cuando se realiza una evaluación por ventana se está evaluando única y exclusivamente el comportamiento del clasificador. Estas evaluaciones son significativamente diferentes y evidentemente la que es relevante para la detección de objetos es la que se realiza por imagen, desarrollada aquí.

Resumiendo, en la evaluación por imagen se está evaluando el detector completo con todos sus componentes:

1. La generación de candidatos
2. La clasificación de los candidatos ← En la Evaluación por ventana se evalúa exclusivamente el clasificador
3. El refinamiento de la decisión

10.8 CONJUNTOS DE ENTRENAMIENTO, EVALUACIÓN Y VALIDACIÓN

En este punto se verá cómo utilizar las muestras para evaluar la capacidad de generalización del detector y también para optimizar algunos de los parámetros que están involucrados en el desarrollo de ese detector. Dado un conjunto de muestras, se quiere aprender un modelo a partir de ellas y también evaluar el clasificador asociado a ese modelo o incluso el detector completo. Sin embargo, si se usan las mismas muestras para aprender el modelo y evaluar el clasificador o el detector asociado lo único que se obtiene es el error de clasificación, es decir el error que se ha minimizado. Por lo tanto, lo que interesa realmente es conocer la capacidad de generalización del clasificador asociado al modelo que se ha aprendido. Y para esto los conjuntos tienen que ser diferentes.

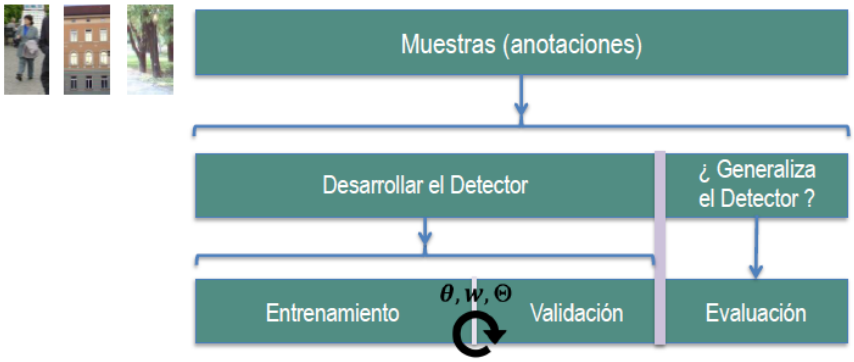


Ilustración 93: Conjuntos de entrenamiento, validación y evaluación

Aquí se ve un esquema general de cómo utilizar las muestras anotadas para desarrollar un detector de objetos y evaluar su capacidad de generalización, es decir lo bien que detecta los objetos en imágenes que no ha visto durante su desarrollo. Por lo tanto del conjunto de muestras se reserva una parte para la fase de evaluación. Durante el desarrollo del detector, a las muestras también se las divide en dos partes, unas de entrenamiento y otras de validación. Esto es así porque en el detector de objetos están involucrados diversos parámetros a los que se les tiene que asignar el mejor valor. Para saber cuál es ese mejor valor se tiene que probar diversos valores y tomar aquellos que dan mejor resultado. Por ejemplo, en el caso del descenso del gradiente, en regresión logística, o el caso del parámetro alfa, (la velocidad de aprendizaje). También sirven para ajustar parámetros del descriptor, por ejemplo, en los bloques de histogramas de LBP, el tamaño de los bloques

Por lo tanto, se tienen los conjuntos de entrenamiento y validación, por una parte, y evaluación, por otra. El conjunto de entrenamiento no necesita su correspondiente Ground Truth, ya que son muestras positivas o negativas. Pero los conjuntos de validación y evaluación si tienen que tener su Ground Truth. Para realizar esta división entre validación y evaluación se analizaran 2 métodos.

10.8.1 MÉTODO HOLD-OUT

Uno de los métodos que se puede utilizar para usar muestras distintas al entrenar modelos y evaluar es el que en inglés se conoce como Hold-Out: En este caso se trata de tener un cierto porcentaje de las muestras para validar y el resto se usa para evaluar. Usualmente el porcentaje es de un 50%. Por otra parte, se tiene que utilizar algún tipo de mecanismo de selección de muestras aleatoriamente. A este procedimiento de entrenar y evaluar se lo puede realizar un número de veces, por ejemplo k veces, y promediar el resultado de las k evaluaciones.

10.8.2 MÉTODO CROSS-VALIDATION

Otro método es el que se conoce como validación cruzada o en inglés Cross-Validation: En este caso se parte de un conjunto de m muestras y dado un número entero k, se divide ese conjunto en k subconjuntos. Se utilizan todos esos subconjuntos, menos 1, en la evaluación del clasificador que se está entrenando o el detector global que se está utilizando. Es decir, se tienen k -1 subconjuntos en la primera ronda, y el restante para entrenar. Así sucesivamente hasta completar k rondas. Luego se promedia el resultado de las evaluaciones, las que están basadas en una alguna métrica. Así el resultado de ese promedio informa de la capacidad de generalización que tiene el clasificador o globalmente el detector que se está entrenando.

Con el método de validación cruzada, todas las muestras se utilizan tanto para validar como evaluar, aunque en rondas distintas. Con el método de Hold-put, esto no se garantiza. El método de hold-out puede también utilizarse una o varias veces, promediando luego los resultados.

10.8.3 MÉTODO SELECCIONADO

En este trabajo se usó el método Hold Out, con 50%, y la separación aleatoria fue simplemente ordenar por nombre e ir poniendo una muestra para cada grupo, tomando de a dos nombres por vez.

11 METODOLOGÍA EXPERIMENTAL

11.1 DISEÑO Y DESARROLLO DEL PROGRAMA

El código implementa las fases de un detector de objetos explicado a lo largo del presente documento:

- Generación de candidatos
- Clasificación de candidatos
- Refinación de la decisión.
- Elaboración de métricas para poder seleccionar los mejores parámetros de las distintas técnicas durante la validación
- Elaboración de métricas para evaluar cuál es la mejor combinación de técnicas para un detector de objetos.

Se trata de una implementación que incluye todas las partes de un detector y que tiene el objetivo de poder realizar la evaluación de los resultados. Y ese es precisamente el objetivo de este trabajo, evaluar las distintas técnicas y sus combinaciones para poder llegar a una conclusión válida respecto de la elección y configuración óptima. Es por ello que no cuenta con interfaces para usuarios finales, ya que como se dijo, no se busca que un usuario maneje un software detector de personas, sino que se llegue a conclusiones válidas sobre el mejor uso de las técnicas y sus configuraciones.

Para la generación de candidatos se utiliza una pirámide de imágenes a diferentes escalas, combinada con el método de ventana deslizante. Para la clasificación de candidatos el código incorpora dos descriptores alternativos (LBP y HOG) y dos métodos de clasificación (regresión logística y SVM).

El código se puede utilizar con cualquier combinación de descriptor y clasificador, incluso combinando LBP y HOG como descriptor. Para la refinación de decisión, se ha implementado el método de Non Maximal Suppression, ya explicado. Están incluidos también los modelos de los dos clasificadores (regresión logística y SVM) entrenados con los dos descriptores (LBP y HOG) con la bases de datos de imágenes que proporciona Inria, el Instituto Nacional Francés para la informática y las matemáticas aplicadas. Estos modelos ya entrenados se pueden utilizar directamente sobre imágenes de test para comprobar el funcionamiento del detector. Como ya se dijo, se permite entrenar modelos con diferentes configuraciones del descriptor y el clasificador.

11.2 INSTALACIÓN Y CONFIGURACIÓN

Para poder utilizar el código, son necesarios los siguientes pasos:

1. Descomprimir el fichero Zip en algún directorio de la unidad C. Este fichero contiene el código en Python y los modelos pre-entrenados.

2. Instalar en el ordenador Python, las librerías adicionales que se utilizan en el código y un IDE para trabajar con Python.

i. Python 2.7.x

ii. Paquetes Python:

- Scikit-image
- scikit-learn
- PIL

iii. Descargar e instalar Mini-Conda: http://conda.pydata.org/miniconda.html

iv. Descargar e instalar PyCharm Community Edition: https://www.jetbrains.com/pycharm/download/

v. Descargar e instalar Python 2: https://www.python.org/downloads/

vi. Abrir una consola de windows, (cmd), y ejecutar:

- conda install Scikit-image
- conda install scikit-learn
- conda install PIL

3. Descomprimir el fichero Datasets.zip, que contiene las bases de datos de imágenes.

11.3 ESTRUCTURA DEL CÓDIGO

La mayoría de pasos (extracción de características, aprendizaje del clasificador, generación de la pirámide) se implementan a partir de llamadas a las funciones de las librerías Scikit-image y scikit-learn. El programa básicamente ejecuta dos módulos, validación y evaluación, con los cuales genera resultados, valores y estadísticas a partir de ellos, que sirven para llegar a conclusiones.

El código está estructurado a partir de los siguientes ficheros:

1. config.py: Contiene la definición de todos los parámetros de configuración del detector en todas sus fases (tipo y parámetros del descriptor y clasificador, parámetros de la ventana deslizante, directorios de imágenes, etc.).

2. validación.py: Modulo que ejecuta las pruebas de validación para hallar el factor de regularización donde se configuran distintas corridas de distintos modelos, con distintos parámetros de configuración. Modifica en tiempo de ejecución los parámetros del config.py. Genera asimismo una salida estadística y las imágenes de evaluación con los recuadros del Ground Truth y los hallados para cada una de las validaciones. Hace llamadas a los módulos, (en el siguiente orden): extrae_descriptores.py, entrena_modelo.py, prueba_directorio.py, muestra_resultados.py y evalua_resultados.py. Es decir realiza la extracción de características, el aprendizaje del clasificador, aplica el detector a todo un directorio de imágenes, guarda las imágenes resultantes en otro directorio y evalúa resultados, con cada modelo y configuración.

3. evalua.py: Igual al módulo de validación, con la diferencia que ejecuta las pruebas de evaluación donde se configuran distintas corridas, de los distintos modelos, con los parámetros seleccionados durante la etapa de validación.

4. extrae_descriptores.py: Al ejecutar este módulo se calcula un determinado descriptor (LBP o HOG) para un conjunto de imágenes de entrenamiento. El descriptor y las imágenes a utilizar se especifican en el fichero Principal.py. El resultado se guarda en una subcarpeta “Descriptores”, para poderlo utilizar al entrenar el modelo de clasificador.

5. entrena_modelo.py: Este módulo entrena un modelo de clasificador con un conjunto de características ya pre-calculadas. Tanto el tipo y configuración del clasificador (regresión logística o SVM) como el tipo de descriptor utilizado (HOG o LBP) se especifican en el fichero config.py. Las características deben estar pre-calculadas en la subcarpeta “Descriptores”, como resultado de ejecutar el módulo extrae_descriptores.py. El modelo del clasificador aprendido se guarda en una subcarpeta “Modelos”, para ser utilizado al aplicar el detector en una imagen de test.

6. prueba_una_imagen.py: Este módulo aplica el detector sobre una imagen completa, aplicando la generación de candidatos con pirámide y non Maximal Suppression. Tanto los parámetros del detector, como el tipo de descriptor y el modelo de clasificador que se utilizan se especifican en el fichero config.py. El modelo del clasificador debe estar pre-calculado en la subcarpeta Modelos, como resultado de ejecutar el módulo entrena_modelo.py. El resultado de la detección se muestra en una nueva imagen, mostrando las ventanas de detección con la confidencia de cada una de ellas.

7. prueba_directorio.py: Este módulo aplica el detector sobre un conjunto de imágenes que se encuentran en una subcarpeta determinada, especificada en el fichero config.py. El modelo del clasificador debe estar pre-calculado en la subcarpeta “Modelos”, como resultado de ejecutar el módulo entrena_modelo.py. El resultado de la detección se guarda en la subcarpeta “Resultados”.

8. muestra_resultados.py: Este módulo toma todos los resultados guardados en una subcarpeta de Resultados y genera, para cada uno de las imágenes de test, una nueva imagen, superponiendo a la imagen original los rectángulos con el resultado de la detección.

9. evalua_resultados.py: A partir de los resultados guardados en una subcarpeta de Resultados, genera la gráfica con la evaluación del rendimiento que muestra la curva tasa de error (miss rate) vs. FFPI y sus valores.

10. borra_archivos.py: modulo accesorio para borrado de archivos.

11. separa_validacion_evaluacion.py: modulo que se usó para separar en forma aleatoria el conjunto que se tenía de muestras con el Ground Truth en un 50% al conjunto evaluación y el otro 50% al conjunto validación.

11.3.1 PARÁMETROS DE CONFIGURACIÓN DEL DETECTOR DE OBJETOS

Principales parámetros del detector que se fijan en el fichero de configuración config.py como en el archivo de arranque:

- featuresToExtract: Determina qué descriptor se utiliza. Puede tomar el valor de uno de los dos descriptores, ['HOG'] o ['LBP'], o especificar que se calculen los dos descriptores a la vez ['HOG', 'LBP'], concatenándolos en una única descripción final.

- Parámetros utilizados para el cálculo de LBP:

  - ✓ lbp_win_shape: determina el tamaño de los bloques y

  - ✓ lbp_win_step: Determina la separación entre bloques.

- Parámetros utilizados en el cálculo del descriptor HOG:

  - ✓ nº de intervalos del histograma de orientaciones

  - ✓ nº de píxeles en cada celda

  - ✓ nº de celdas por bloque.

- datasetRoot: Ruta del directorio donde se encuentra la base de datos con las imágenes que se utilizarán en el aprendizaje de los modelos.

- positive_folder, negative_folder: Subcarpetas del directorio de la base de datos donde se encuentran las imágenes positivas y negativas para el aprendizaje.

- positiveFeaturesPath, negativeFeaturesPath: Directorio donde se guardan las características de las imágenes positivas y negativas utilizadas para entrenar el modelo.
- model: Método de clasificación que se utiliza. Puede tomar los valores 'SVM' o 'LogisticRegression'
- modelPath: Directorio donde se guarda el modelo de clasificador que se ha aprendido.
- SVM.LinearSVC parameters: Parámetros utilizados para entrenar un clasificador SVM. El más relevante será svm_C, que indica el factor de regularización.
- LogisticRegression parameters: Parámetros utilizados para entrenar un clasificador con regresión logística. El más relevante será logReg_C que permite especificar el factor de regularización.
- window_shape: Tamaño de la ventana de detección al aplicar la ventana deslizante.
- window_step: Desplazamiento de la ventana deslizante entre paso y paso.
- decision_threshold: Umbral que se fija para determinar si la evaluación de una ventana se debe dar como detección positiva. Se puede fijar un umbral diferente para cada combinación de descriptor y clasificador.
- downScaleFactor: Factor de escalado al generar la pirámide de imágenes.
- nmsOverlapThresh: Factor de solapamiento para considerar equivalentes dos ventanas al aplicar el non-Maximal Suppression.
- testFolderPath: Directorio que contiene las imágenes que se van a utilizar como test.
- annotationsFolderPath: Directorio que contiene la anotación con la localización correcta de las imágenes de test, para poder realizar la evaluación.
- resultsFolder: Subcarpeta donde se guardan los resultados de las detecciones y las imágenes que se generan con las ventanas detectadas.

11.4 BASES DE DATOS

De la base de datos Pedestrians – INRIA, se obtuvieron los siguientes conjuntos de muestras. Es necesario aclarar que todos los conjuntos formados son disjuntos entre sí.

11.4.1 CONJUNTO DE ENTRENAMIENTO

Conjunto de imágenes de peatones preparadas para poder realizar el aprendizaje de los modelos. Contiene imágenes ya recortadas todas al mismo tamaño, (64x128), con tamaño aproximado de 12 a 21 KB, usadas para extraer los descriptores y entrenar los modelos. Se divide en un subconjunto de imágenes de resultados positivos, (2416 muestras), y un subconjunto de resultados negativos, (2990 muestras).

11.4.2 CONJUNTO DE EVALUACIÓN

Conjunto de imágenes de test. Se usara para realizar una comparación estadística entre distintos detectores de personas. Contiene imágenes completas (extraídas del conjunto de test de INRIA). Son las imágenes a utilizar para comprobar el funcionamiento del detector con parámetros ya definidos. Consta de 144 imágenes y sus respectivos archivos planos donde se indican las ventanas positivas, el Ground Truth. Dicha notación está hecha marcando el centro de la ventana con dos números, y con otros dos números su altura y ancho. Dicha información es procesada y comparada con las ventanas identificadas como positivas por el programa. Con esta información, finalmente, se realiza las estadísticas correspondientes.

11.4.3 CONJUNTO DE VALIDACIÓN

Conjunto de imágenes donde aparecen uno o más peatones en distintas circunstancias y a diferentes distancias, es decir con distintos tamaños. Se usara para hallar el factor de regularización apropiado para cada detector de personas. Son 144 muestras positivas con tamaños grandes y variados. Al igual que el conjunto de evaluación, consta de sus respectivos archivos planos donde se indican las ubicaciones de las ventanas positivas.

12 PROCEDIMIENTO EXPERIMENTAL

Como ya se ha dicho, el alcance del presente trabajo es **analizar**, **desarrollar** y **evaluar** alternativas en reconocimiento de una clase en particular, las personas, en imágenes. Para lo cual se desarrolla un módulo de detección de personas que sirve de marco para evaluar las distintas combinaciones de técnicas aquí propuestas. A estas combinaciones, serán llamados, de ahora en más, “Modelos”. Por lo tanto el objetivo es analizar los diferentes modelos, para llegar a una conclusión, sobre cuál es el mejor, para esta clase de objetos en particular. Estas conclusiones se representan con claridad por medio de una gráfica, que es la curva formada entre la tasa de error y la tasa de falsos positivos por imagen, donde el área debajo de dicha curva representa una buena medida de comparación entre distintos detectores, es decir, entre distintos “modelos”.

Antes de realizar esta comparativa, es necesario, primero entrenar los distintos modelos y hallar los mejores parámetros, y el umbral a usar, ya que para cada selección de parámetros y modelo, existirá una curva tasa de error/tasa falsos positivos por imagen distinta. La mayoría de los parámetros a usar son los usuales, e iguales para todos los modelos; pero otro, el factor de regularización será probado con el conjunto de validación para cada modelo.

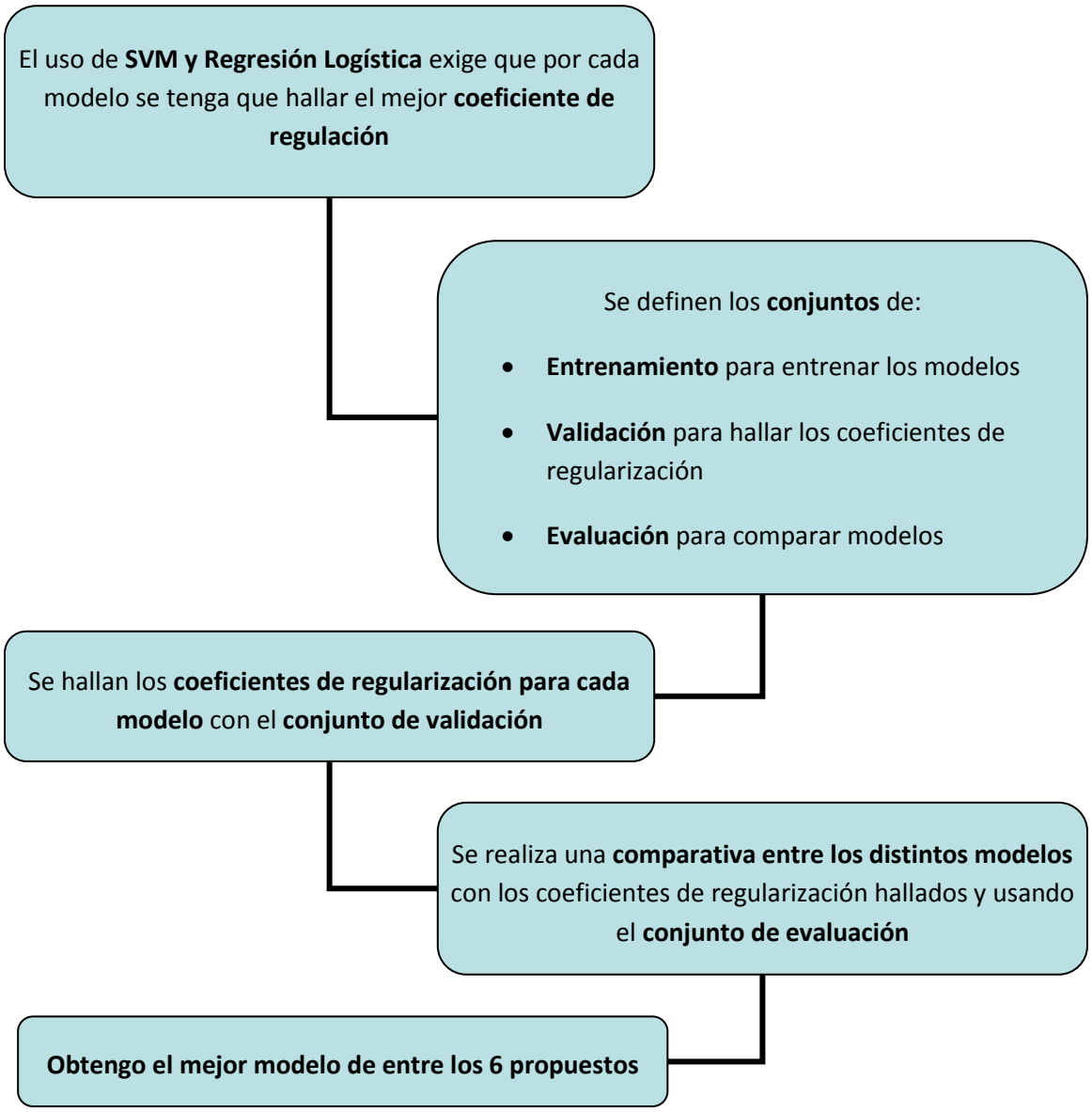
Resumiendo, el experimento se realiza en dos partes, la primera consiste en hallar el factor de regularización de cada modelo usando el conjunto de validación, y la segunda parte consiste en comparar los resultados de los distintos modelos respecto al conjunto de evaluación.

Las imágenes generadas por el programa se pueden encontrar en la carpeta “Resultados”, donde los recuadros en rojos son el Ground Truth y los recuadros verdes indican el hallazgo de una persona, según el valor de umbral configurado.

12.1 DEFINICIÓN DE UN CASO DE ESTUDIO

Primero se encontrara cual es el mejor factor de regularización para todos los modelos a evaluar usando el conjunto de validación. A partir de allí es posible realizar una comparativa entro los modelos usando el conjunto de evaluación. Tanto para hallar el factor de regularización como para comparar modelos se usan las gráficas Tasa de Error / FPPI.

12.1.1 MAPA CONCEPTUAL



12.1.2 MODELOS A EVALUAR

1. **LBP + SVM**
2. **LBP + Regresión Logística**
3. **HOB + SVM**
4. **HOB + Regresión Logística**
5. **HOB + LBP + SVM**
6. **HOB + LBP + Regresión Logística**

Por otra parte es necesario definir, cuáles serán los parámetros a configurar correspondientes a las técnicas usadas.

12.1.3 CONFIGURACIÓN DE PARÁMETROS PARA HOB

Parámetros utilizados en el cálculo del descriptor HOG que no se modifican durante la etapa de validación:

- ✓ Nº de intervalos del histograma de orientaciones: hog_orientations = 9
- ✓ Nº de píxeles en cada celda: hog_pixels_per_cell = (8, 8)
- ✓ Nº de celdas por bloque: hog_cells_per_block = (2, 2)
- ✓ Valores normalizados: hog_normalise = True

12.1.4 CONFIGURACIÓN DE PARÁMETROS PARA LBP

Parámetros utilizados en el cálculo del descripto LBP que no se modifican durante la etapa de validación:

- ✓ Determina el tamaño de los bloques: lbp_win_shape = (16, 16) ->Nótese que los tamaños de las imágenes en el conjunto entrenamiento son múltiplos de 16, tanto la altura, como el ancho.
- ✓ Determina la separación entre bloques: lbp_win_step = lbp_win_shape[0]/2
- ✓ Radio del descriptor LBP: lbp_radius = 1
- ✓ Número de puntos que definen la vecindad en LBP: lbp_n_points = 8 * lbp_radius
- ✓ Define como descriptor a LBP Uniforme: lbp_METHOD = 'nri_uniform'
- ✓ Número de patrones en el LBP Uniforme: lbp_n_bins = 59

12.1.5 CONFIGURACIÓN DE PARÁMETROS PARA SVM

- ✓ Factor de regularización, parámetro de penalización C del término de error, ya descripto en este documento: svm_C. Para este parámetro se realizaran pruebas con los siguientes valores 0.001, 0.01, 0.1, 1, 10, 100, 1000. [31]

Los valores de los siguientes parámetros son los usuales y están referidos a la formulación matemática del SVM, ver referencia [29]. Estos parámetros permanecen fijos durante las pruebas

- ✓ svm_penalty = 'l2'
- ✓ svm_dual = False
- ✓ svm_tol = 0.0001
- ✓ svm_fit_intercept = True
- ✓ svm_intercept_scaling = 100

12.1.6 CONFIGURACIÓN DE PARÁMETROS PARA REGRESIÓN LOGÍSTICA

- ✓ Factor de regularización, parámetro de penalización C del término de error, ya descripto en este documento: logReg_C. Para este parámetro se realizaran pruebas con los siguientes valores 0.001, 0.01, 0.1, 1, 10, 100, 1000.

Los valores de los siguientes parámetros son los usuales y están referidos a la formulación matemática de la regresión logística, ver referencia [29]. Estos parámetros permanecen fijos durante las pruebas.

- ✓ logReg_penalty = 'l2'
- ✓ logReg_dual = False
- ✓ logReg_tol = 0.0001
- ✓ logReg_fit_intercept = True
- ✓ logReg_intercept_scaling = 100

12.1.7 CONFIGURACIÓN DE VENTANA DESLIZANTE Y PIRAMIDE

Estos parámetros quedan fijos para todas las pruebas

- ✓ Tamaño de la ventana de detección al aplicar la ventana deslizante : window_shape = (128, 64)
- ✓ Margen que se añade a la ventana si es positiva, es decir si hallo el objeto buscado: window_margin = 16
- ✓ Desplazamiento de la ventana deslizante entre paso y paso: window_step = 32
- ✓ Factor de escalamiento de la pirámide: downScaleFactor = 1.2

12.1.8 CONFIGURACIÓN DE EVALUACIÓN DE PRUEBAS

- ✓ Ancho del borde de relleno dado a las imágenes de prueba: padding = 16
- ✓ Porcentaje de solapamiento entre dos ventanas para ser consideradas la misma en Non Maximum Suppression: nmsOverlapThresh = 0.05
- ✓ Porcentaje de solapamiento con Ground Truth para ser considerado bueno: annotation_min_overlap = 0.05

12.1.9 CONFIGURACIÓN DEL UMBRAL DE DECISIÓN

- ✓ Umbral de decisión, para cada modelo se elige uno distinto: decision_threshold

  Los siguientes valores se usan para realizar las comparativas entre modelos por medio de las curvas Tasa de error/FPPI, e indican que los umbrales a tomar van desde -2 a 30 con una distancia entre un umbral y el siguiente de 0.1. Por ejemplo, empieza con -2; -1.9; -1.8….así hasta llegar a 30. Para cada uno de estos umbrales se hallara la tasa de error y la tasa de falsos positivos por imagen, que representa un punto en la curva Tasa de error/FPPI, tema ya explicado.

- ✓ decision_threshold_min = -2
- ✓ decision_threshold_max = 30
- ✓ decision_threshold_step = 0.1

12.2 PRUEBAS

Cada muestra de testeo tiene su correspondiente archivo plano donde se indican, en una serie de líneas, 4 números. Esto significa que cada línea representa una ventana y los 4 números enteros son el centro de la ventana, (x, y) y su a ancho y altura (W, H). Por lo tanto, en cada muestra se tiene marcada la ventana donde se encuentra cada una de las personas que allí aparecen. **En este trabajo se toma como criterio que el objetivo buscado es detectar a la persona y no el realizar un marco exacto alrededor de la persona. Por lo tanto, si en un caso el programa se detecta, por ejemplo, un marco que abarca solo las piernas de una persona, se considera una detección aceptada. Se considera error si detectara otro marco con otra parte de la misma persona. Esto último se penaliza en las estadísticas.** En otras palabras se busca que el programa marque una ventana que encierre alguna parte de la persona y solo exista una ventana por persona. Para lograr este efecto, los siguientes parámetros fueron fijados a valores bajos e iguales para todos los modelos.

- ✓ nmsOverlapThresh
- ✓ annotation_min_overlap

12.2.1 PRUEBAS DE VALIDACIÓN PARA HALLAR EL FACTOR DE REGULARIZACIÓN

El programa calcula estadísticos que generan las curvas Tasa de error/FPPI, donde cada elemento de la curva representa un valor de umbral determinado. Como ya se dijo en el punto "EVALUACIÓN DEL RENDIMIENTO DEL DETECTOR", para poder comparar el rendimiento entre distintos detectores es necesario hallar la curva Tasa de error/FPPI de cada uno de los modelos. Por lo que cada punto de dicha curva representa puntos que van desde -2 a 30, con un paso de 0.1. Para cada uno de estos puntos se calcula la tasa de error por medio de la fórmula 1 – {TP/(TP+FN)}, y también para cada uno de los umbrales, se calculan los falsos positivos por imagen. Los FPPI se calculan directamente como el total de FP dividido el número de imágenes del conjunto. Póngase especial atención donde la tasa de error alcanza su mínimo, siendo que a partir de allí los valores de FPPI solo aumentan.

Se realizaron pruebas con valores de C = 0.001, 0.01, 0.1, 1, 10, 100, 1000 para todos los modelos, (C = Factor de regularización). La curva más próxima al eje de coordenadas es la mejor, por tener al mismo tiempo baja tasa de error y baja cantidad de falsos positivos. Se quiere buscar un mínimo de FPPI y un mínimo de la tasa de error al mismo tiempo, para cada uno de los valores de C, con un modelo en particular, y seleccionar el mejor valor de C para dicho modelo. ***Con dichos factores de regularización se realizara luego la evaluación entre los modelos.***

12.2.2 PRUEBAS DE VALIDACIÓN PARA HALLAR EL FACTOR DE REGULARIZACIÓN: MODELO LBP + SVM

En el anexo A, (el archivo "Curvas Tasa de error vs FPPI del modelo LBP SVM.xlsx"), se encuentran los valores que forman cada una de estas gráficas, la tasa de error, FPPI y los valores umbrales. El mejor desempeño se atribuye a la curva que se encuentre más desplazada hacia el centro de coordenadas.

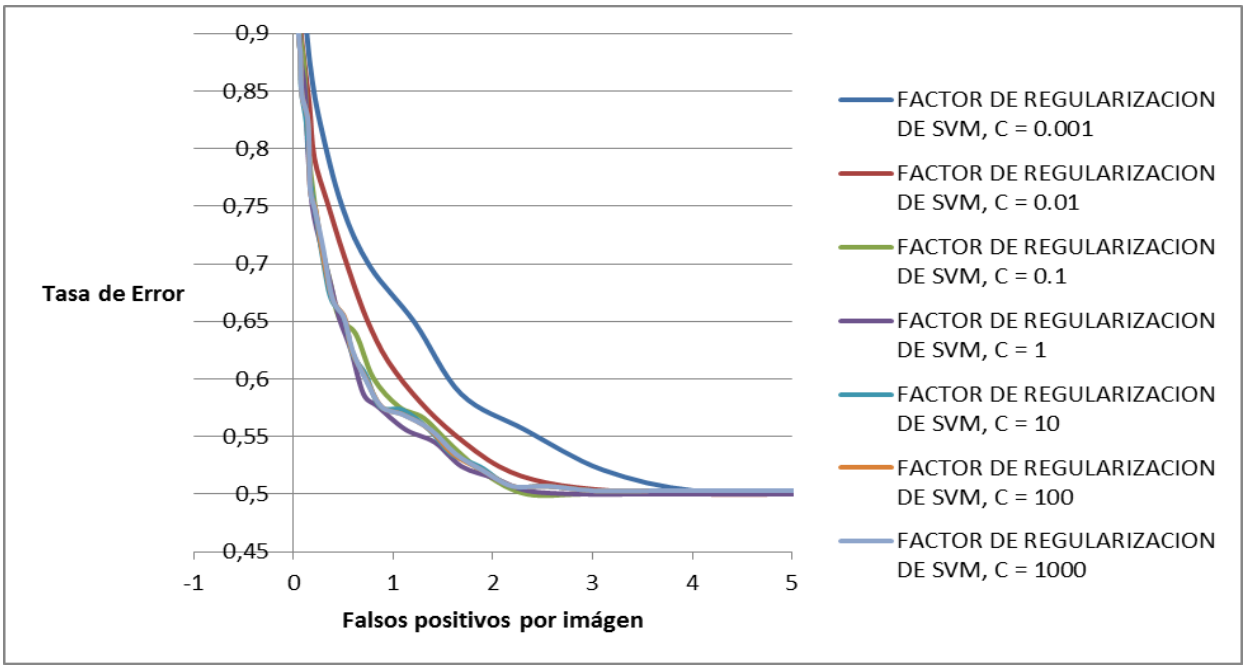


Ilustración 94: Curvas Tasa de error/FPPI con distintos factores de regularización del modelo LBP + SVM

12.2.3 CONCLUSIONES SOBRE VALIDACIÓN EN EL MODELO LBP + SVM

Observando las curvas de tasa de error vs FPPI se concluye a simple vista que la curva obtenida con el factor de regularización C=1, es la más óptima, con menores valores de tasa de error y falsos positivos por imagen. Un factor de regularización igual a 1 nos habla que no existe un problema de "overfitting" para este modelo, pero que tampoco se puede despreciar su peso en la fórmula de minimización del costo.

Por otra parte es interesante analizar los datos mostrados por el gráfico, como se dijo antes, cada punto de la curva representa un umbral, si se ven los datos en la planilla Excel, se comprueba que los umbrales van de izquierda a derecha de 30 a -2. Estos valores son el rango que se marcó en la configuración del programa

El gráfico pareciera tener una tendencia a un piso en la tasa de error igual a 0.5, cuando el umbral tiende a -2, (y los FPPI tienden a aumentar). La razón de porque sucede, está relacionada a la cantidad de verdaderos positivos que hay en la totalidad de las muestras, (288), y la cantidad de muestras mismas, (144). La tasa de error se calcula como:

$$T.E. = 1 - \frac{TP}{TP + FN}$$

Cuando el umbral T tiende a -2, lo que sucede es que todas las ventanas que se evalúan con el método de pirámide con ventana deslizante tienden a dar positivos erróneamente. Así, la ventana que cubre toda imagen también da positivo y el algoritmo de "Non-Maximum Suppression" hace que se marque como True Positive solo esta ventana desechando el resto, ya que están solapadas con la misma. Por lo tanto lo que tenemos es un sistema que tiende a devolver solo una ventana que abarca toda la imagen, que la da positiva y que por lo tanto, si hubiera más de una anotación positiva del Ground Truth, las marca como False Negative. Nos queda un total de True Positive similar al total de muestras y un total de False Negative similar al total de marcaciones positivas del Ground Truth menos el total de muestras. En nuestro caso el total de marcaciones positivas del Ground Truth es igual a 288, (es decir personas anotadas en las imágenes). El total muestras es 144, (cantidad de imágenes). Esto quiere decir que:

$$0.5 = 1 - \frac{144}{144 + 288 - 144}$$

Por otra parte, cuando el umbral tiende a 30, ocurre que el sistema no detecta ningún peatón. Queda entonces que la cantidad total de True Positive es 0. En la formula anterior nos queda que la tasa de error tiende a 1, como se comprueba en el gráfico, al mismo tiempo que el Falso Positivo Por Imagen, FPPI, tiende a 0.

La elección del umbral es una decisión subjetiva a los objetivos. Arbitrariamente se podría tomar como ejemplo un valor umbral de 1.6, con lo que da una tasa de error del 0.556, y daría 2.319 de falsos positivos por imagen. Vemos algunos ejemplos, donde el rojo es el Ground Truth, el verde el hallazgo del sistema. El valor escrito es la distancia entre el hiperplano w y el vector de características, que se compara con el valor umbral, en este caso 1.6

Ilustración 95: Ejemplos del detector LBP + SVM con C = 1 y umbral = 1.6, en el conjunto de validación

12.2.4 PRUEBAS DE VALIDACIÓN PARA HALLAR EL FACTOR DE REGULARIZACIÓN: MODELO LBP + REGRESIÓN LOGÍSTICA

En el anexo B, (el archivo "Curvas Tasa de error vs FPPI del modelo LBP RL.xlsx") se encuentran los valores que forman cada una de las gráficas: la tasa de error, FPPI y los valores umbrales

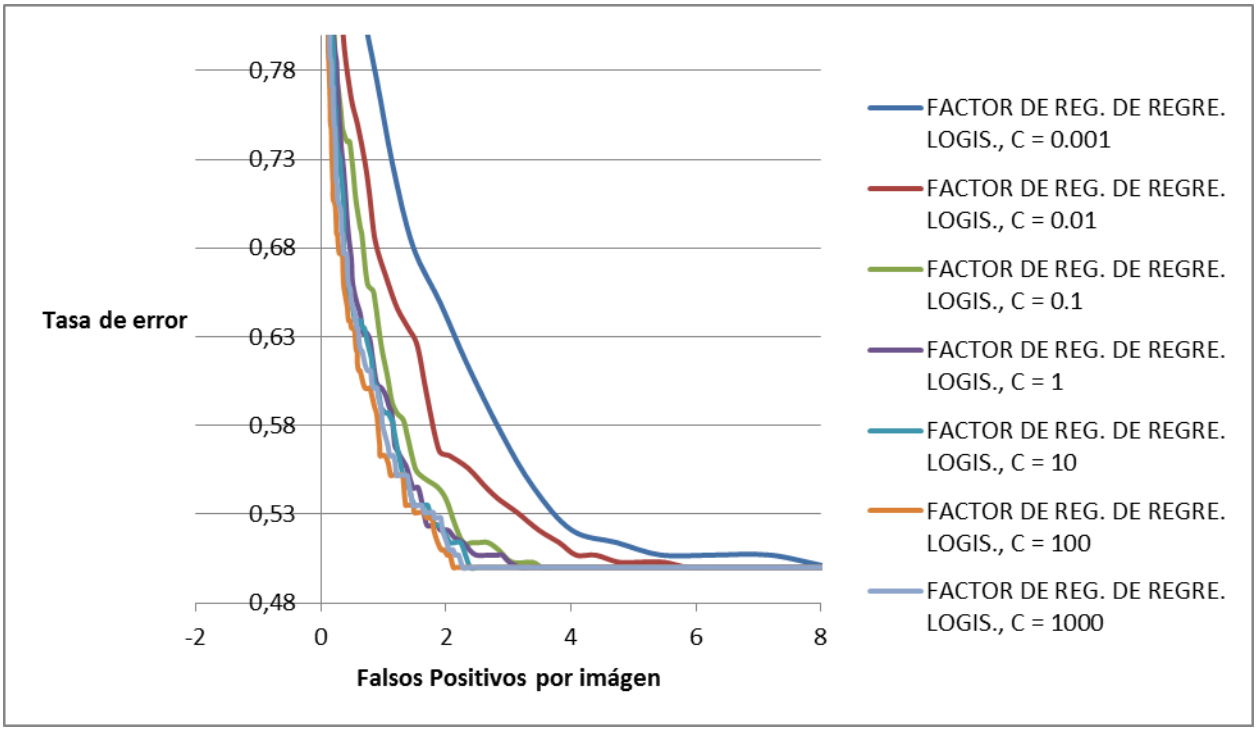


Ilustración 96: Curvas Tasa de error/FPPI con distintos factores de regularización del modelo LBP + Regresión Logística

12.2.5 CONCLUSIONES SOBRE VALIDACIÓN EN EL MODELO LBP + REGRESIÓN LOGÍSTICA

Las conclusiones son similares a los del punto 12.2.3 en cuanto a la forma de las curvas. Los resultados indican que el parámetro óptimo es C=100, valor que muestra que existe un problema de overfitting que el factor de regularización corrige. Arbitrariamente se podría tomar un valor umbral de 9.8 que da una tasa de error = 0.55.

12.2.6 PRUEBAS DE VALIDACIÓN PARA HALLAR EL FACTOR DE REGULARIZACIÓN: MODELO HOG + SVM

En el anexo C, (el archivo "Curvas Tasa de error vs FPPI del modelo HOG SVM.xlsx") se encuentran los valores que forman cada una de estas gráficas, la tasa de error, FPPI y los valores umbrales.

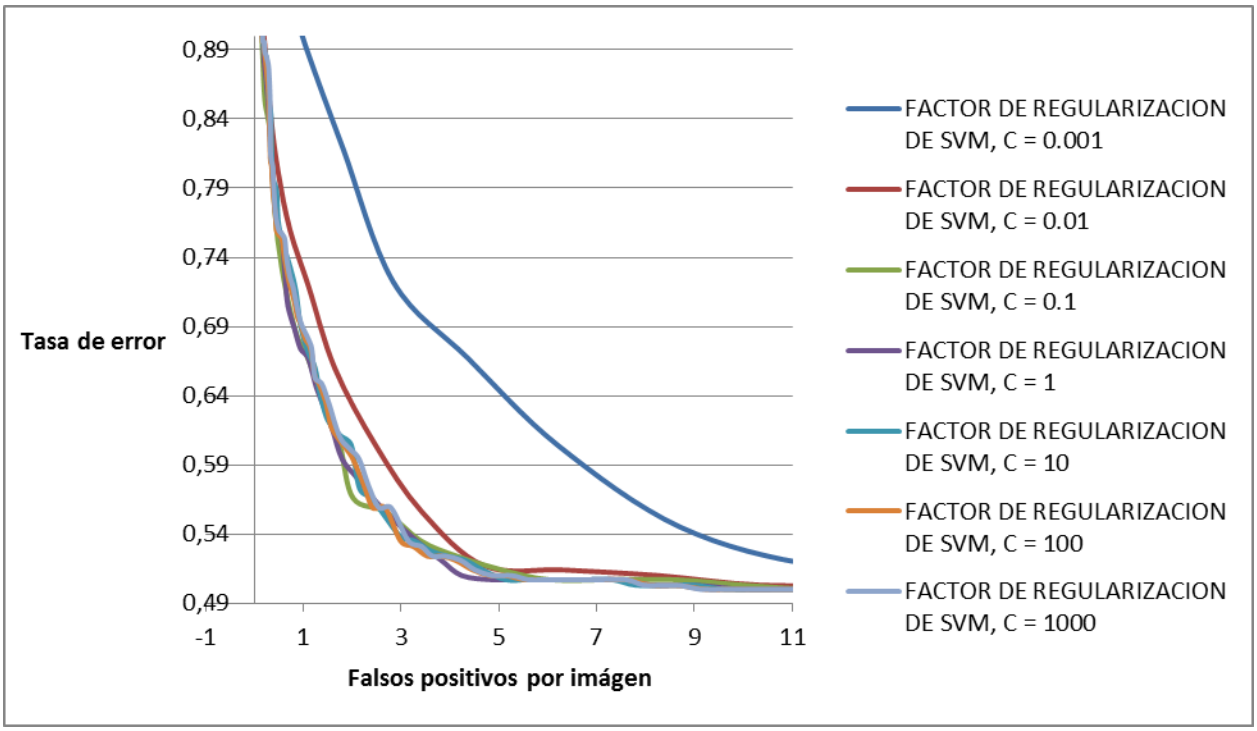


Ilustración 97: Curvas Tasa de error/FPPI con distintos factores de regularización del modelo HOG + SVM

12.2.7 CONCLUSIONES SOBRE VALIDACIÓN EN EL MODELO HOG + SVM

Las conclusiones son similares a los del punto 12.2.3 en cuanto a la forma de las curvas. Debido a la proximidad de las curvas, la elección es difícil. Los resultados indican que el parámetro óptimo es C=1. Como ejemplo, un valor con umbral = 1.3 obtiene una tasa de error = 0.59

12.2.8 PRUEBAS DE VALIDACIÓN PARA HALLAR EL FACTOR DE REGULARIZACIÓN: MODELO HOG + REGRESIÓN LOGÍSTICA

En el anexo D, (el archivo "Curvas Tasa de error vs FPPI del modelo HOG RL.xlsx") se encuentran los valores que forman cada una de las gráficas: la tasa de error, FPPI y los valores umbrales

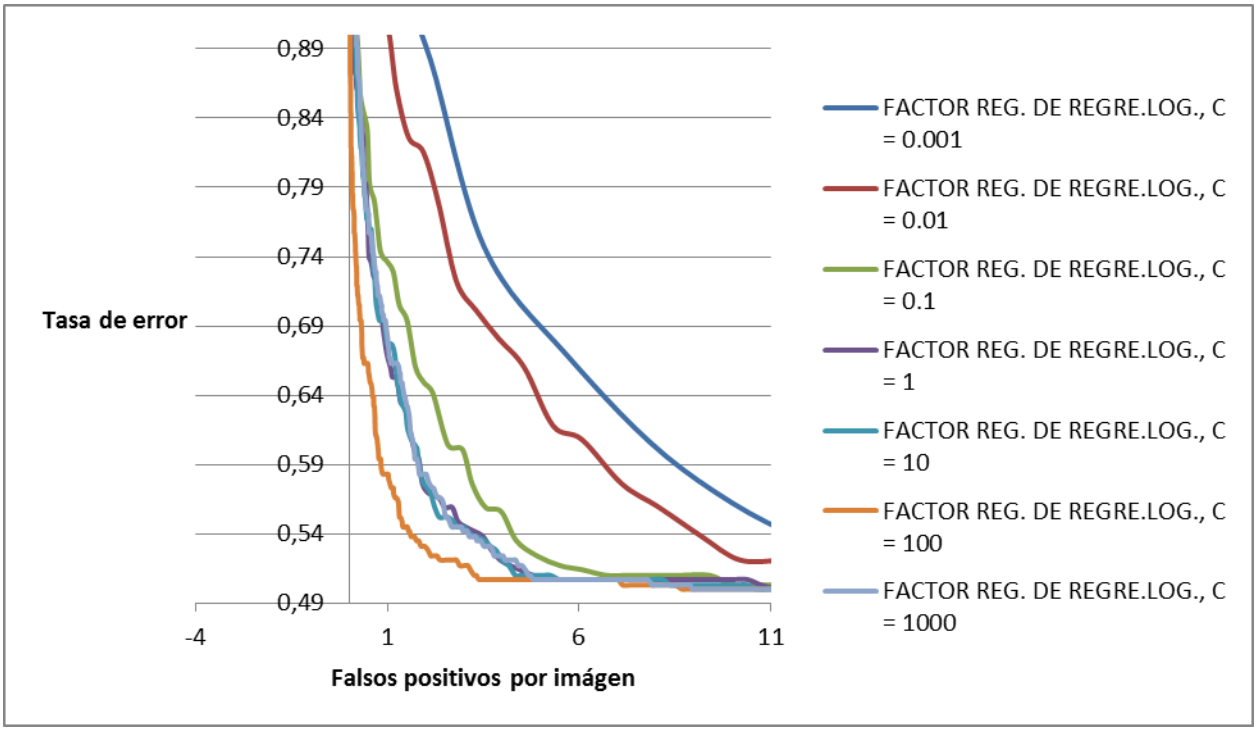


Ilustración 98: Curvas Tasa de error/FPPI con distintos factores de regularización del modelo HOG + Regresión Logística

12.2.9 CONCLUSIONES SOBRE VALIDACIÓN EN EL MODELO HOG + REGRESIÓN LOGÍSTICA

Las conclusiones son similares a los del punto 12.2.3 en cuanto a la forma de las curvas. Los resultados indican que el parámetro óptimo es C=100. Con un umbral = 4.4 se obtendría una tasa de error de 0.53

12.2.10 PRUEBAS DE VALIDACIÓN PARA HALLAR EL FACTOR DE REGULARIZACIÓN: MODELO LBP + HOG + SVM

En el anexo E, (el archivo "Curvas Tasa de error vs FPPI del modelo LBP HOG SVM.xlsx") se encuentran los valores que forman cada una de estas gráficas, la tasa de error, FPPI y los valores umbrales.

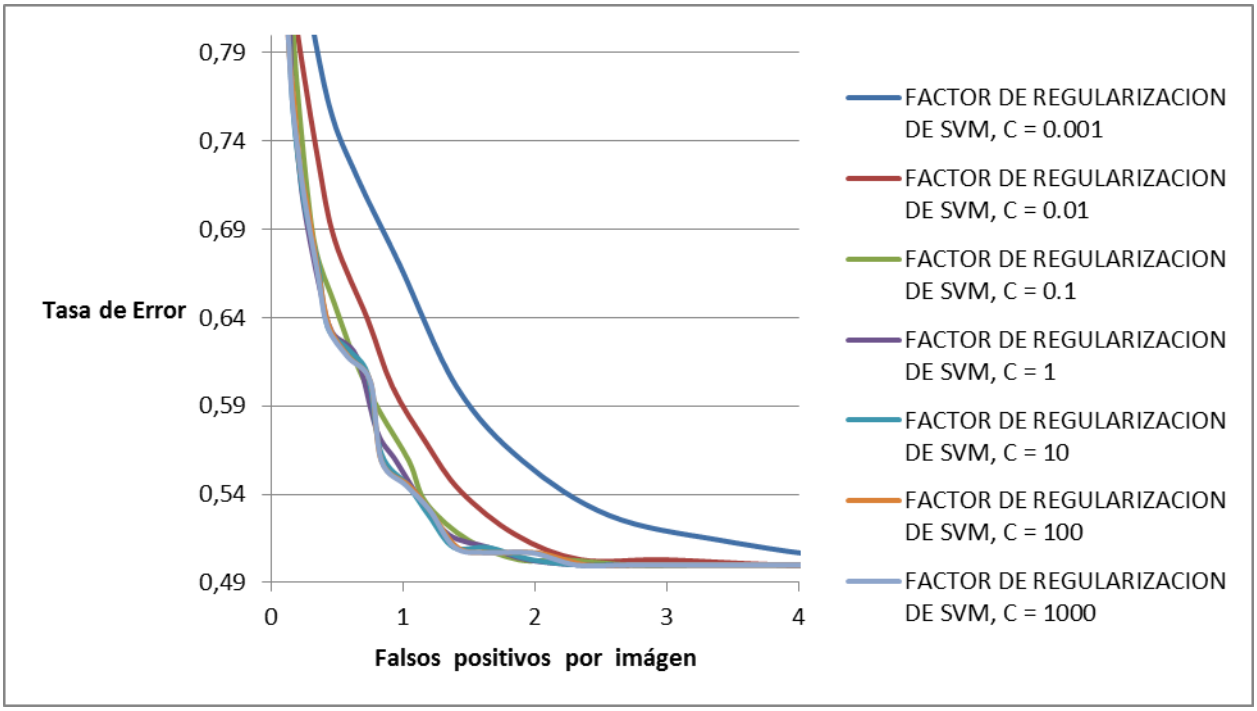


Ilustración 99: Curvas Tasa de error/FPPI con distintos factores de regularización del modelo LBP + HOG + SVM

12.2.11 CONCLUSIONES SOBRE VALIDACIÓN EN EL MODELO LBP + HOG + SVM

Las conclusiones son similares a los del punto 12.2.3 en cuanto a la forma de las curvas. Los resultados indican que el parámetro óptimo es C=1000 que muestra que es necesario corregir un tema de overfitting. Con umbral =1.6, se obtiene una tasa de error = 0.51

12.2.12 PRUEBAS DE VALIDACIÓN PARA HALLAR EL FACTOR DE REGULARIZACIÓN: MODELO LBP + HOG + REGRESIÓN LOGÍSTICA

En el anexo F, (el archivo “Curvas Tasa de error vs FPPI del modelo LBP HOG RL.xlsx”) se encuentran los valores que forman cada una de las gráficas: la tasa de error, FPPI y los valores umbrales

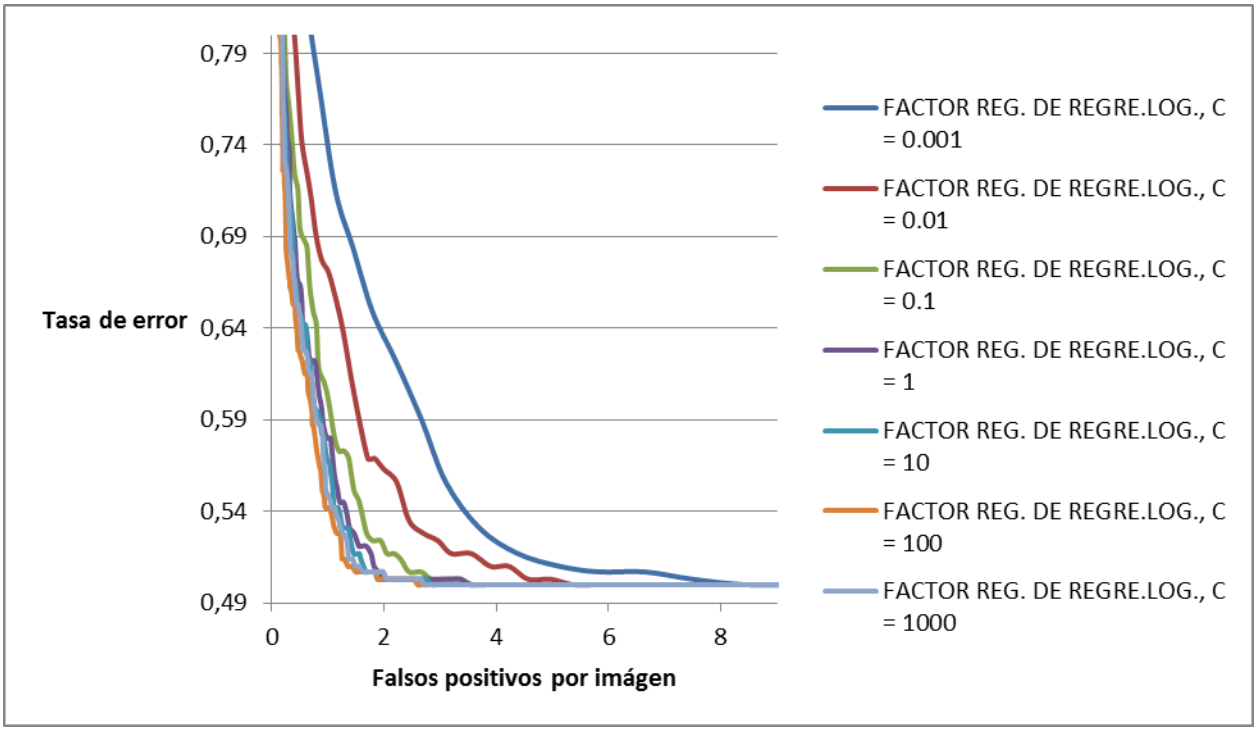


Ilustración 100: Curvas Tasa de error/FPPI con distintos factores de regularización del modelo LBP + HOG + Regresión Logística

12.2.13 CONCLUSIONES SOBRE VALIDACIÓN EN EL MODELO LBP + HOG + REGRESIÓN LOGÍSTICA

Las conclusiones son similares a los del punto 12.2.3 en cuanto a la forma de las curvas. Los resultados indican que el parámetro óptimo es C=100, que corrige un problema de overfitting. Como ejemplo, con un umbral =10 tendríamos una tasa de error= 0.51.

13 RESULTADOS

13.1 PRUEBAS DE EVALUACIÓN DE TODOS LOS MODELOS JUNTOS

Luego de realizar las pruebas de todos los modelos con los factores de regularización encontrados para cada uno de ellos se llega a estos resultados, en el anexo G, (el archivo “Curvas Tasa de error vs FPPI de todos los modelos con su mejor factor de regularización.xlsx”)

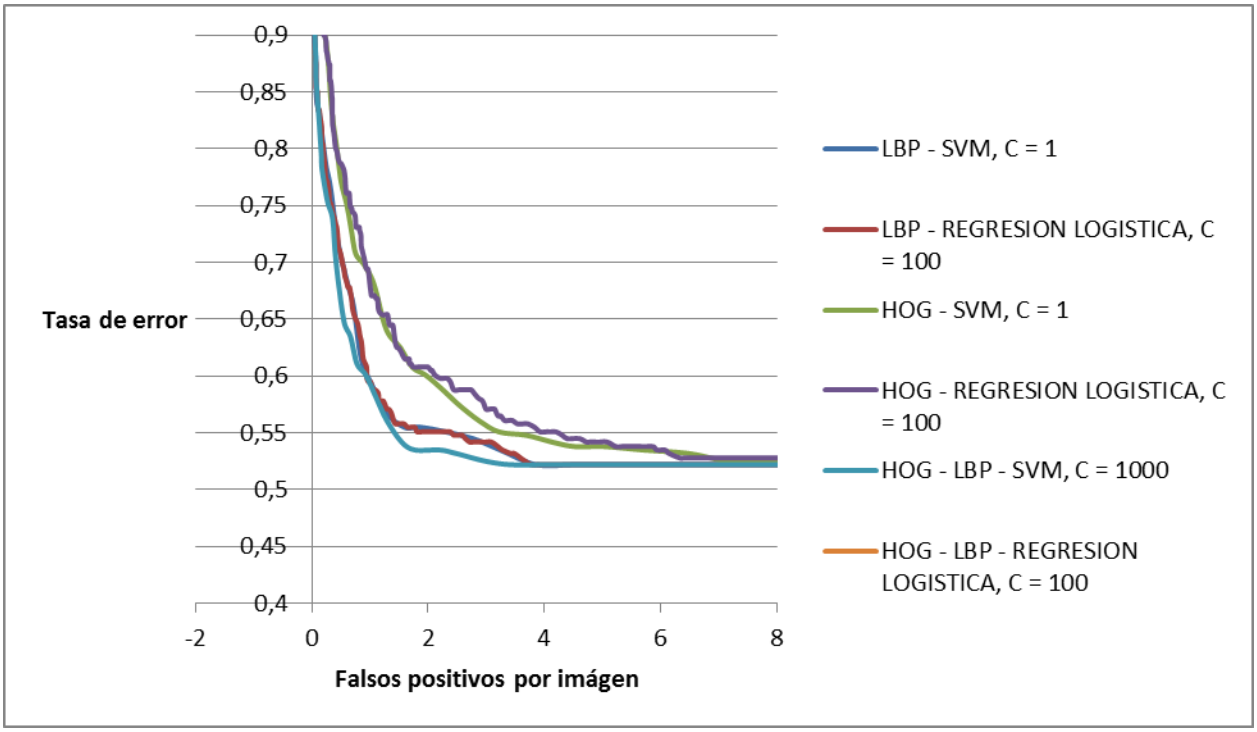


Ilustración 101: Curvas Tasa de error vs FPPI de todos los modelos con su mejor factor de regularización

13.1.1 DISCUCIÓN DE RESULTADOS

Primero fue necesario hallar para cada modelo su mejor factor de regularización, conseguido ello, se pudo realizar una comparación entre los distintos modelos de detectores de personas. Como primera observación se puede decir que las conclusiones son similares a los del punto 12.2.3 en cuanto a la forma de las curvas, aunque teniendo en cuenta que la cantidad de marcas positivas en el Ground Truth ahora es de 301, mientras que la cantidad de muestras coincide en 144 imágenes, entonces se tiene que cuando el FPPI crece, es decir el valor umbral tiende a -2, la tasa de error tiende a 0.521:

$$T.E. = 1 - \frac{TP}{TP + FN}$$

$$0.521 = 1 - \frac{144}{144 + 301 - 144}$$

Como segunda observación, se ve claramente que los mejores resultados se obtienen con un modelo HOG - LBP - SVM, con un factor de regularización de 1000. Esto es así ya que la eficiencia de un detector, en iguales condiciones, se puede medir en términos del que posea la menor área bajo la curva del gráfico Tasa de error/FPPI.

Nuevamente se tiene que aclarar que la elección del umbral es una decisión subjetiva a los objetivos. Arbitrariamente se podría tomar como ejemplo un valor umbral de 1.6, con lo que da una tasa de error del 0.542, y daría 1.542 de falsos positivos por imagen. Vemos algunos ejemplos, donde el rojo es el Ground Truth, el verde el

hallazgo del sistema. El valor escrito es la distancia entre el hiperplano w y el vector de características, que se compara con el valor umbral, en este caso 1.6

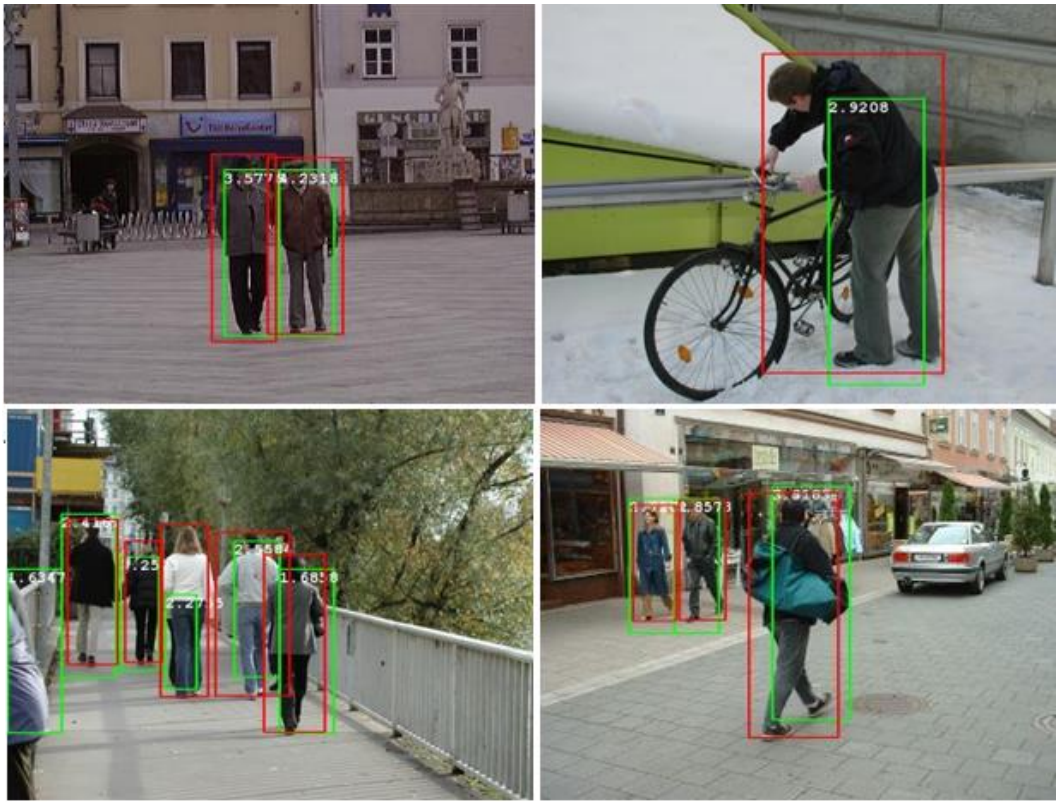


Ilustración 102: Ejemplos del detector HOG - LBP - SVM con C = 1000 y umbral = 1.6, en el conjunto de evaluación

14 CONCLUSIONES

Se ha comenzado este proyecto exponiendo, en el capítulo 3, la problemática derivada del análisis de imágenes y las dificultades que los algoritmos de detección de objetos presentan. En el capítulo 4 se expone claramente que el interés principal de este trabajo es desarrollar un detector de objetos de una clase en particular, "personas", y que el mismo sería de gran utilidad en muchas circunstancias, por lo que en el capítulo 8 se fijó como principal objetivo de este proyecto su desarrollo. Este detector tendría que garantizar resultados óptimos. Para lo cual, en el capítulo 9, se detallaron una serie de objetivos específicos a ser alcanzados, todos los cuales fueron conseguidos. En el capítulo 5 se realizó un estudio detallado del estado del arte en detección de personas. De este estudio se extrajeron las principales características y limitaciones que presenta cada uno de los enfoques analizados. En la sección 6.1, se seleccionaron algunos de los algoritmos base de detección de personas más utilizadas actualmente y se desecharon otros algoritmos por no ajustarse su uso a los objetivos, o porque su complejidad excedía por mucho los alcances del presente trabajo. Por otra parte, se propuso un esquema general de un detector de personas, en la sección 7.1, y se avanzó con el desarrollo de sus partes.

En la sección 10.7 se definió con claridad la forma de evaluar el rendimiento del detector en su conjunto por medio de las curvas Tasa de error/FPPI, cuyos datos son generados automáticamente por el sistema.

En la sección 11.4 se analizó la forma de dividir la base de datos en subconjuntos de entrenamiento, validación y evaluación, lo cual permitió realizar las validaciones de cada uno de los modelos propuestos, (sección 12.2). Con los resultados de las validaciones se pudo entonces avanzar con la evaluación del mejor modelo, (sección 13.1).

A la vista de los resultados podemos concluir que el objetivo de desarrollar un detector de personas confiable fue alcanzado con éxito, ya que se ha conseguido medir correctamente la eficiencia de los distintos modelos analizados, llegando a poder realizar la siguiente afirmación basada en un proceso experimental sólido:

El mejor desempeño se atribuye a la curva que se encuentre más desplazada hacia el centro de coordenadas. Es decir, aquella que tiene los menores porcentajes de tasa de error y falsos positivos por imagen. Se concluye que el mejor detector de objetos que fue posible entrenar con el conjunto de datos que se posee es el que corresponde al modelo HOG - LBP - SVM, con un factor de regularización de 1000.

Por otra parte, luego de las pruebas realizadas se puede llegar a la conclusión de que el umbral elegido es un factor crítico que debe ser evaluado cuidadosamente en cada aplicación en particular.

15 TRABAJOS FUTUROS

En relación a trabajos futuros que derivan del presente, se puede hablar de dos puntos importantes:

- El primero es que el desarrollo específico de una aplicación que necesite de un detector de personas tiene ya acotado su principal factor de riesgo, ya que se ha probado que un detector de personas funciona y se conoce su grado de eficiencia.
- El segundo punto a destacar es que en el presente trabajo se desarrolló un de detector de objetos para la clase “personas”, pero el mismo funcionaría también con cualquier otra clase de objetos. Simplemente se tendría que cambiar la base de datos y realizar el mismo proceso de entrenamiento y configuración aquí realizado.

16 REFERENCIAS

[1] Template matching, https://en.wikipedia.org/wiki/Template_matching

[2] Detector de caras basado en Filtros de Haar + Adaboost, http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf

[3] Local Binary Patterns, (LBP), https://en.wikipedia.org/wiki/Local_binary_patterns

[4] Histogram of oriented gradients, (HOG)
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

[5] Support Vector Machines, (SVM),
https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte

[6] Regresión Logística, https://es.wikipedia.org/wiki/Regresi%C3%B3n_log%C3%ADstica

[7] Curso Detección de Objetos: https://www.coursera.org/course/deteccionobjetos, Universidad Autónoma de Barcelona: http://www.uab.cat/

[8] Fuente de la base de datos de modelos, (personas): Inria, Instituto Nacional Francés para la informática y las matemáticas aplicadas: http://www.inria.fr/

[9] Object Category Detection: Sliding Windows - Computer Vision - CS 543 - University of Illinois https://courses.engr.illinois.edu/cs543/sp2011/lectures/Lecture%2019%20-%20Sliding%20Window%20Detection%20-%20Vision_Spring2011.pdf

[10] L-BFGS https://es.wikipedia.org/wiki/L-BFGS

[11] Método del gradiente conjugado https://es.wikipedia.org/wiki/M%C3%A9todo_del_gradiente_conjugado

[12] Curso "Machine Learning". https://www.coursera.org/learn/machine-learning - Curso CS229 - Universidad de Stanford http://cs229.stanford.edu/materials.html

[13] Benenson, R., Mathias, M., Timofte, R., & Van Gool, L. (2012, June). Pedestrian detection at 100 frames per second. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on (pp. 2903-2910). IEEE.

[14] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on (Vol. 1, pp. 886-893). IEEE.

[15] Marin, J., Vázquez, D., López, A. M., Amores, J., & Leibe, B. (2013, December). Random forests of local experts for pedestrian detection. In Computer Vision (ICCV), 2013 IEEE International Conference on (pp. 2592-2599). IEEE.

[16] Wang, X., Han, T. X., & Yan, S. (2009, September). An HOG-LBP human detector with partial occlusion handling. In Computer Vision, 2009 IEEE 12th International Conference on (pp. 32-39). IEEE.

[17] Fuente de las imágenes 2, 3 y 4: Propias.

[18] Python Software Foundation. https://www.python.org/psf/

[19] "Scikit-image" http://scikit-image.org/

[20] PyCharm Community Edition https://www.jetbrains.com/pycharm/

[21] Apache Software Foundation http://www.apache.org/

[22] Anwer, R. M., Vázquez, D., & López, A. M. (2011). Opponent colors for human detection. In Pattern Recognition and Image Analysis (pp. 363-370). Springer Berlin Heidelberg.

[23] Lienhart, R., & Maydt, J. (2002). An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on (Vol. 1, pp. I-900). IEEE.

[24] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on (Vol. 1, pp. I-511). IEEE.

[25] "Face and Periocular Biometrics" http://www.cmeri.res.in/rnd/srlab/cvision/face%20pericular.php

[26] "Uniform LBP Features and Spatial Histogram Computation" http://www.codeproject.com/Articles/741559/Uniform-LBP-Features-and-Spatial-Histogram-Computa

[27] Local Binary Pattern for texture classification http://scikit-image.org/docs/dev/auto_examples/plot_local_binary_pattern.html

[28] HOG Dr. Mubarak Shah https://www.youtube.com/watch?v=0Zib1YEE4LU

[29] sklearn.svm.LinearSVC http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[30] CS229 Lecture notes, Andrew Ng, Support Vector Machines http://cs229.stanford.edu/notes/cs229-notes3.pdf

[31] A User's Guide to Support Vector Machines http://pyml.sourceforge.net/doc/howto.pdf

17 INDICE DE ILUSTRACIONES