

UNIVERSIDAD SIMÓN BOLÍVAR

Ingeniería de la Computación

Sistema para la Detección de Caras en Imágenes de Video

por

Susan Benzaquen Nahon

Proyecto de Grado

Presentado ante la Ilustre Universidad Simón Bolívar
como Requisito Parcial para Optar al Título de
Ingeniero en Computación

Sartenejas, Julio del 2006

UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE COMPUTACIÓN
ACTA FINAL DE PROYECTO DE GRADO
**Sistema para la Detección de Caras en Imágenes de
Video**

Presentado por

Susan Benzaquen Nahon

Este proyecto de grado ha sido aprobado en nombre de la Universidad Simón Bolívar por el siguiente
jurado examinador:

Prof. Osberth C. De Castro C.
Jurado

Prof. Víctor S. Theoktisto
Jurado

Prof. Carolina Martínez (Tutor Académico)

Prof. Jesús Miguel Ferrer (Profesor Guía)

Sartenejas, Julio de 2006

Sistema para la Detección de Caras en Imágenes de Video

por

Susan Benzaquen Nahon

Resumen

Las Máquinas de Vectores Soporte (SVM) constituyen una técnica de clasificación de patrones desarrollada por Vapnik-Bosen-Goyon [4], y han sido utilizadas para la resolución de problemas relacionados con el reconocimiento de voz, clasificación de documentos, reconocimiento de escritura y reconocimiento de patrones de caras. La idea principal de esta técnica consiste en separar el conjunto de datos etiquetados entrantes por medio de un hiperplano, de forma tal que se maximize el margen de separabilidad. Este hiperplano óptimo será utilizado para clasificar un nuevo patrón. Las SVM también pueden aplicarse aún cuando los datos de entrada no sean separables linealmente, utilizando para ello funciones kernel.

El trabajo que a continuación presentamos comprende el desarrollo de un sistema computarizado que nos permita entrenar a una Máquina de Vectores Soporte utilizando la Librería SvmFu 3.1 y aplicarlo en la detección de patrones de caras humanas en imágenes estáticas y/o provenientes de una cámara web, en tiempo real.

Índice general

Resumen	II
1. Introducción	1
2. Marco Teórico	4
2.1. Programación No Lineal	4
2.1.1. Formulación de un problema no lineal	5
2.1.2. Método de los Multiplicadores de Lagrange	5
2.1.3. Programación Cuadrática (QP)	6
2.2. Support Vector Machine	9
2.2.1. ¿Qué es una Máquina de Aprendizaje? (Machine Learning)	9
2.2.2. ¿Qué son las Máquinas de Vectores Soporte (SVM) ?	11
2.3. Método de Reducción de Burges	23
2.4. Algoritmos de Detección del Color Piel	24
2.5. Descripción de la Librería SvmFu 3	26
2.6. Manejo de la Cámara de Video	28
3. Diseño del Sistema	30
3.1. Esquema de la aplicación	30
3.2. Procesos Batch o Estáticos	31
3.3. Procesos a tiempo real	40
4. Implementación	45
4.1. Requerimientos del sistema	45
4.2. Operaciones sobre las imágenes	46
4.3. Resultados obtenidos	49
5. Conclusiones	57
A. Plataforma requerida	59
A.1. Librería SvmFu 3	59
A.2. Driver Spca5xx	59
B. Detección de Patrones en Imágenes Estáticas: resultados	61

C. Resultados de los procesos a tiempo real	64
Bibliografía	66

Índice de figuras

2.1. Caso Separable	13
2.2. Hiperplanos paralelos a H , H_1 y H_2 . Definición del margen M	14
2.3. Hiperplano de Margen Suave	18
2.4. Transformación de datos a una dimensión superior mediante el uso de $\Phi(\mathbf{x})$	21
3.1. Diagrama de los procesos batch	32
3.2. Ejemplos de patrones de caras y no-caras de la base de datos	33
3.3. Algoritmo de corrección del gradiente de iluminación	34
3.4. Algoritmo de ecualización del histograma	35
3.5. Algoritmo del programa de rotación	36
3.6. Algoritmo del programa de espejo	36
3.7. Algoritmo de bootstrapping	38
3.8. Proceso y resultados del Bootstrapping	39
3.9. Algoritmo de detección de imágenes estáticas	40
3.10. Diagrama de los procesos a tiempo real	41
3.11. Algoritmo del sistema de detección de piel	42
3.12. Algoritmo del sistema de detección de patrones de caras en imágenes de video	43
4.1. Ejemplos de aplicación de la operación: Máscara	47
4.2. Generación de patrones o ejemplos virtuales	48
4.3. Ejemplos de aplicación de la operación: Escala	48
4.4. Curva ROC	52
4.5. SVM vs. SVM-Baprox	55
B.1. Detección de Patrones en Imágenes Estáticas: Imagen 1	61
B.2. Detección de Patrones en Imágenes Estáticas: Imagen 2	62
B.3. Detección de Patrones en Imágenes Estáticas: Imagen 3	62
B.4. Detección de Patrones en Imágenes Estáticas: Imagen 4	63
B.5. Detección de Patrones en Imágenes Estáticas: Imagen 5	63
C.1. Detección del color piel en imágenes dinámicas	64
C.2. Detección de patrones de caras en imágenes dinámicas	65

Capítulo 1

Introducción

“Una imagen dice más que mil palabras” . Un video es una composición en tiempo de cientos de imágenes por lo que la información que posee es proporcionalmente mayor que la contenida en una imagen. Las nuevas tecnologías han permitido un acceso masivo a los mismos, convirtiéndolos en una fuente de información y medio de expresión para muchas personas.

Para estudiar un video es necesario descomponerlo en su elementos básicos, las imágenes. Es el estudio de las mismas lo que nos ha llevado a señalar que dentro de cada imagen existe siempre un conjunto de rasgos esenciales, un patrón. Es en ese estudio donde radica la importancia de la *detección de patrones*.

El objetivo de la detección de patrones es poder “clasificar en base a un conocimiento a priori. Los patrones a clasificar suelen ser grupos de medidas u observaciones, definiendo puntos en un espacio multidimensional apropiado.” [19].

La detección de patrones tiene numerosas aplicaciones en diversas ramas del conocimiento, tales como, el reconocimiento de voz, la clasificación de documentos, el reconocimiento de escritura, la detección y el reconocimiento de caras humanas y muchas más.

El proceso de identificación facial comprende dos fases: la detección y el reconocimiento. La primera corresponde a la localización de las caras que existen en una imagen o en una secuencia de imágenes. La segunda tarea busca asociar el patrón detectado a la persona correspondiente cuyos rasgos han sido previamente almacenadas en una base de datos.

La investigación contenida en este trabajo se enfoca principalmente en la *detección de caras en secuencias de imágenes de video*. La detección de patrones faciales en imágenes de video así como la identificación de características faciales en general ha recibido una especial atención en los últimos años debido al desarrollo de nuevas tecnologías de video y multimedia, lo que ha propiciado un uso mayor de cámaras en distintos espacios públicos y privados (Novar [7]). La detección e identificación de caras es utilizada para el reconocimiento de personas en aplicaciones policiales, en aplicaciones de bioinformática, y adicionalmente se está usando en un área nueva que es para aplicaciones de interfaz/usuario/máquina donde la detección de elementos principales en la cara como ojos, nariz y boca pueden definir un localizador 3D que es interpretado directamente por la máquina para mover objetos en escenas tridimensionales (Huang y Blanz 2003 [17]).

El objetivo de este trabajo es el desarrollo de un Sistema computarizado que permita la detección de caras humanas en una secuencia de imágenes de video, utilizando para ello la librería de Máquinas de Vectores Soporte (Support Vector Machines - SVM) SvmFu 3.1 desarrollada en el MIT por Ryan Rifkin en año 2000. [24].

En esta investigación se implementó un detector de caras en imágenes de video utilizando la teoría de Máquinas de Vectores Soporte (SVM). Las Máquinas de Vectores Soporte (SVM) representan una

técnica de clasificación desarrollada por Vapnik-Bosen-Goyon [4]. Una tarea de clasificación generalmente involucra a conjuntos de datos de entrenamiento etiquetados y datos de prueba. Cada dato del conjunto de entrenamiento contiene un valor de clasificación, etiqueta o valor objetivo y numerosos atributos. El logro principal de SVM es construir un modelo que permita predecir el valor objetivo para elementos del conjunto de prueba utilizando únicamente los valores de los atributos.

La principal idea geométrica detrás de la técnica SVM es separar las diferentes clases correspondientes a los valores objetivos con una superficie multidimensional que maximize, en cierta forma, el “margen” de separabilidad entre ellas y utilizar esta superficie para clasificar un nuevo punto.

Las Máquinas de Vectores Soporte (SVM) tienen numerosas aplicaciones, como por ejemplo: el reconocimiento de escrituras (Cortes y Vapnik 1995 [8]; Schölkopf, Burges y Vapnik 1995, 1996 [26, 27]; Schölkopf y Burges 1997 [5]), el reconocimiento de objetos (Blanz 1996 [3]), la identificación del interlocutor (Schmidt 1996 [25]), la categorización de textos (Joachims 1997 [18]) y la detección de patrones de caras (Edgar Osuna, Robert Freund y Federico Girosi [21]; Kah-Kay Sung y Tomaso Poggio [29]; S.M. Bileschi, B. Heisele 2002 [2]; J. Huang, V. Blanz, B. Heisele 2002 [16]; B. Heisele, P. Ho y T. Poggio 2001 [14]) entre otros.

Este trabajo está estructurado de la siguiente manera: en el Capítulo 2 presentamos el marco teórico donde se explican los fundamentos matemáticos en los que se basa la teoría de SVM y el área en la que se desarrolla: *Machine Learning*. En el Capítulo 3 explicamos los detalles de implementación de la aplicación así como sus resultados. En el Capítulo 4 presentamos las conclusiones y observaciones.

Capítulo 2

Marco Teórico

En este capítulo discutiremos la teoría matemática y las nociones básicas relevantes para el desarrollo de la aplicación. Para ello dividiremos esta sección en cuatro partes. Comenzaremos en la sección 2.1 resumiendo los resultados principales de la Programación Cuadrática (QP), técnica de optimización utilizada por las Máquinas de Vectores Soporte (SVM). En la sección 2.2 presentaremos, en detalle la teoría matemática de las Máquinas de Vectores Soporte (SVM). La sección 2.3 se refiere al Método de Reducción de Burges. En la sección 2.4 discutiremos el Algoritmo de Detección de Piel. Posteriormente en la sección 2.5 introduciremos la librería SvmFu 3.1 utilizada en la aplicación desarrollada, y por último en la sección 2.6 describiremos el dispositivo de entrada de video que se utiliza en la aplicación.

2.1. Programación No Lineal

En esta sección haremos una breve introducción a la programación no lineal, planteando los problemas generales de optimización y cómo resolverlos utilizando las distintas técnicas desarrolladas para tales fines. También presentaremos brevemente los problemas de programación cuadrática (QP) y sus características más resaltantes. Un estudio más detallado sobre programación no lineal y programación cuadrática (QP) pueden ser consultado en [1].

2.1.1. Formulación de un problema no lineal

Un problema de programación lineal corresponde a la optimización de una función objetivo lineal sujeta a un conjunto de restricciones definidas por funciones lineales. Frecuentemente, en las distintas aplicaciones nos encontramos con problemas de optimización en los que la función objetivo y/o las restricciones no son lineales. Esto nos permite formular el problema de optimización no lineal.

De una manera general, un problema de programación no lineal consiste en encontrar $\mathbf{x} = \{x_1, \dots, x_n\}$ de manera tal que:

PROBLEMA 1

$$\begin{aligned} & \text{maximizar/minimizar}_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \\ & \text{sujeto a: } h_k(\mathbf{x}) = 0, \quad k = 1, \dots, \ell \\ & \qquad g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \end{aligned}$$

donde $f(\mathbf{x})$, $g_i(\mathbf{x})$ y $h_k(\mathbf{x})$ son funciones reales de n variables.

2.1.2. Método de los Multiplicadores de Lagrange

Existen diversos métodos para resolver el Problema 1 de la sección 2.1.1, los cuales van a depender de las características específicas que posean las funciones $f(\mathbf{x})$, $g_i(\mathbf{x})$ y $h_k(\mathbf{x})$. Cuando tenemos problemas no lineales en los cuales las restricciones son de igualdad, el método más utilizado para resolver el problema es el de *multiplicadores de Lagrange*.

El método de Lagrange nos permite resolver el siguiente problema:

PROBLEMA 2

$$\begin{aligned} & \text{maximizar/minimizar}_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \\ & \text{sujeto a: } g_1(\mathbf{x}) = b_1 \\ & \qquad g_2(\mathbf{x}) = b_2 \\ & \qquad \dots \dots \\ & \qquad g_m(\mathbf{x}) = b_m \end{aligned}$$

con $b_i \in \mathbb{R}$. Para resolver el Problema 2 (2.1.2) utilizamos una función conocida como la *función de Lagrange* que se construye asociando a cada restricción i una constante negativa desconocida λ_i llamada multiplicador de Lagrange, que hallaremos posteriormente.

La función de Lagrange viene definida por:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i (g_i(\mathbf{x}) - b_i) \quad (2.1)$$

El punto óptimo $\bar{\mathbf{x}}$ del Problema 2 (2.1.2) se encuentra entre los puntos críticos de la función $L(\mathbf{x}, \boldsymbol{\lambda})$. Para hallarlos tenemos que resolver el siguiente sistema de $n + m$ ecuaciones con $n + m$ incógnitas resultantes de igualar las derivadas parciales de $L(\mathbf{x}, \boldsymbol{\lambda})$ con respecto a todas las variables a cero:

$$\begin{aligned} \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial x_i} &= 0 \quad i = 1, \dots, n \\ \frac{\partial L(\mathbf{x}, \boldsymbol{\lambda})}{\partial \lambda_i} &= 0 \quad i = 1, \dots, m \end{aligned} \quad (2.2)$$

Este sistema nos permite hallar los valores óptimos de $\bar{\mathbf{x}}$ y de $\bar{\boldsymbol{\lambda}}$, donde $\bar{\mathbf{x}}$ representa la solución del Problema 2 (ver 2.1.2).

2.1.3. Programación Cuadrática (QP)

El problema de programación cuadrática (QP) es un problema de optimización no-lineal que ha sido centro de estudios en los últimos años y para el cual se han demostrado diversas propiedades

características que nos permiten resolverlos.

Los algoritmos con los que se construyen las Máquinas de Vectores Soporte (SVM), que veremos en la sección 2.2, utilizan los problemas de programación cuadrática (QP) para derivar los hiperplanos óptimos que permiten clasificar a las imágenes.

Formulación de un problema cuadrático

Un problema de Programación Cuadrática es un problema de optimización en el cual la función objetivo es de forma cuadrática y las restricciones son lineales, de esta forma podemos *formular un problema cuadrático*, sin pérdida de generalidad como:

$$\text{PROBLEMA 3} \quad \text{minimizar } f(x) := \frac{1}{2} x^t Q x - c^t x$$

$$\text{sujeto a: } A x = b$$

donde, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$ es el vector de incógnitas, $A \in \mathbb{R}^{m \times n}$ es una matriz de rango completo y $Q \in \mathbb{R}^{n \times n}$ es simétrica y positiva definida, lo que implica que Q es convexa.

Condiciones de optimalidad

Las condiciones de optimalidad de un problema de programación no lineal son aquellas condiciones sobre las función objetivo y/o restricciones que permiten garantizar la existencia de una solución óptima, ya sea esta correspondiente a un mínimo o máximo según sea el caso. Las mismas pueden ser *necesarias y/o suficientes*, y con esto nos referimos a que si un punto \mathbf{u} cumple las condiciones necesarias asociadas a un problema, entonces ese punto será el óptimo, pero a veces existen problemas tan

complicados que las condiciones necesarias son difíciles de verificar, y para esto tenemos las condiciones de suficiencia, que si son cumplidas y verificadas entonces podemos decir que el punto \mathbf{u} es el óptimo sin tener que comprobar las condiciones necesarias.

En el caso de la programación cuadrática (QP) los algoritmos de resolución del problema 2.1.1 se basan en la determinación de un punto en el que se cumplan las condiciones de Karush, Kuhn y Tucker (KKT) [1] las cuales son definidas como:

DEFINICIÓN 1 (Condiciones Karush-Khun-Tucker) *El vector $\bar{\mathbf{x}} \in \mathbb{R}^n$ satisface las condiciones de Karush-Khun-Tucker para el Problema 1 (2.1.1) si existe un par de vectores $\mu \in \mathbb{R}^m$ y $\lambda \in \mathbb{R}^\ell$ tales que:*

$$\begin{aligned}\nabla f(\bar{\mathbf{x}}) + \sum_{k=1}^{\ell} \lambda_k \nabla h_k(\bar{\mathbf{x}}) + \sum_{j=1}^m \mu_j \nabla g_j(\bar{\mathbf{x}}) &= 0 \\ g_j(\bar{\mathbf{x}}) &\leq 0, \quad j = 1, \dots, m \\ h_k(\bar{\mathbf{x}}) &= 0, \quad k = 1, \dots, \ell \\ \mu_j g_j(\bar{\mathbf{x}}) &= 0, \quad j = 1, \dots, m \\ \mu_j &\geq 0, \quad j = 1, \dots, m\end{aligned}$$

Utilizando la función de Lagrange $L(\mathbf{x}, \mu, \lambda) = f(\mathbf{x}) + \lambda^t \mathbf{h}(\mathbf{x}) + \mu^t \mathbf{g}(\mathbf{x})$, las condiciones KKT se escriben como:

$$\nabla_{\mathbf{x}} L(\bar{\mathbf{x}}, \mu, \lambda) = 0$$

$$\nabla_{\lambda} L(\bar{\mathbf{x}}, \mu, \lambda) = 0$$

$$\nabla_{\mu} L(\bar{\mathbf{x}}, \mu, \lambda) \leq 0$$

$$\mu^t \nabla_{\mu} L(\bar{\mathbf{x}}, \mu, \lambda) = 0$$

$$\mu \geq 0$$

Dado que las restricciones para el problema QP son lineales y, por consiguiente, convexas, si la

función objetivo es convexa, como sucede en el modelo QP planteado, (2.1.3), dichas condiciones son condiciones suficientes para garantizar la existencia de un punto óptimo; es decir, si se encuentra un punto en que satisfaga las condiciones KKT, se puede asegurar que este punto es la solución es óptima.

Resolviendo el Problema 3 (2.1.3) de programación cuadrática (QP), presentado anteriormente, utilizando las condiciones KKT para obtener el punto óptimo \bar{x} vemos que tenemos que resolver:

$$Q\bar{x} - c = A^t \lambda$$

$$A\bar{x} = b$$

donde λ es el vector de los multiplicadores de Khun-Tucker asociado a las restricciones del Problema 3 (2.1.3), y \bar{x} representa el punto óptimo.

2.2. Support Vector Machine

2.2.1. ¿Qué es una Máquina de Aprendizaje? (Machine Learning)

Son muchos los autores que tratan de explicar de manera sencilla y práctica *¿qué es el aprendizaje?*. El aprendizaje según la Real Academia Española no es más que el “proceso de adquirir conocimiento a través del estudio, la experiencia o la enseñanza”[10].

Basándonos en este concepto podemos pasar a explicar *¿qué es una máquina de aprendizaje?*. Una definición cualitativa de las máquinas de aprendizaje viene dada según Mitchell [20] por:

DEFINICIÓN 2 (Máquinas de Aprendizaje) *Programas que mejoran con la experiencia, dada una tarea y alguna medida de rendimiento*

Donde con ‘programas’ nos referimos a algoritmos y con ‘experiencia’ nos referimos a datos. Entonces tenemos que mientras más datos utilicen los algoritmos, mejores estos se comportarán. Es decir,

“son aquellas máquinas en las cuales un programa P es mejorado con respecto a alguna medida de rendimiento R basados en la data D ”.

El aprendizaje tiene que ocurrir en el contexto de alguna tarea, y tenemos que tener una idea clara del resultado del aprendizaje. Para ello necesitamos una función objetivo que nos permita medir el desempeño del aprendizaje obtenido. Esto nos lleva a la definición cuantitativa de las máquinas de aprendizaje, en particular de las supervisadas.[20]

DEFINICIÓN 3 (Máquinas de Aprendizaje Supervisado)

Dado:

- *Un universo de datos X*
- *Un conjunto de muestras S donde $S \subset X$*
- *Alguna función objetivo (la tarea o etiqueta) $f: X \rightarrow Y$*
- *Ejemplos de entrenamiento etiquetados D , donde $D = \{< s, f(s) > \mid s \in S\}$*

Determine la función $f': X \rightarrow Y$ usando D tal que $f'(x) \cong f(x)$ para toda $x \in X$.

A este tipo de aprendizaje lo llamamos aprendizaje supervisado por la necesidad de que el usuario suministre un conjunto de datos etiquetados. De acuerdo a la definición el aprendizaje puede ser visto como el problema de encontrar una aproximación f' de f dados los ejemplos de entrenamiento. Es importante notar que usualmente la función o la tarea f es desconocida para los puntos $x \in X - S$.

Todas estas nociones nos muestran un proceso inductivo, donde dada una cantidad finita de data tratamos de inducir, valga la redundancia, la función que aproxima el fenómeno original sobre toda la data del universo. Es justamente este detalle lo característico de las máquinas de aprendizaje y lo que nos genera la hipótesis inductiva [20]:

HIPÓTESIS 1 (Aprendizaje Inductivo) *Cualquier función encontrada que aproxime a la función objetivo adecuadamente sobre un conjunto suficientemente grande de ejemplos de entrenamiento, también aproximará correctamente la función objetivo sobre los ejemplos no observados.*

Las máquinas de aprendizaje supervisado permiten construir una función a partir de cierto conjunto de datos con el fin de estimar la clasificación de una nueva muestra, siguiendo la hipótesis inductiva de aprendizaje y haciendo uso del conjunto de entrenamiento etiquetado.

Un ejemplo de una máquina de aprendizaje supervisado lo constituye las Máquinas de Vectores Soporte (SVM) que pasaremos a estudiar detalladamente en la proxima sección.

2.2.2. ¿Qué son las Máquinas de Vectores Soporte (SVM) ?

Para estudiar la teoría relacionada con las SVM, comenzemos presentando los conceptos básicos y el objetivo de las mismas a través de un enfoque gráfico-matemático. Después, desarrollaremos en detalle las bases teóricas que sustentan las SVM.

Conceptos Básicos

La Máquina de Vectores Soporte fué originalmente concebida para la resolución de problemas de clasificación binaria en los que las clases eran linealmente separables (Vapnik y Lerner 1963 [30]). Posteriormente los resultados se extendieron para la resolución de problemas de clasificación multiclasa y utilizando como funciones de decisión superficies tanto lineales como no lineales. En este trabajo usamos solamente la clasificación binaria y superficies de decisión tanto lineales (conjunto de muestras separable linealmente) como no-lineales (conjunto de muestras no separables linealmente).

La idea geométrica de la SVM consiste en que dado un conjunto de puntos de entrenamiento en \mathbb{R}^n etiquetados en dos clases diferentes, entonces si estos son linealmente separables podemos hallar un hiperplano que separe los puntos de las clases. Esto es, todos los puntos de una misma clase quedarían en un mismo lado del hiperplano de separación. Es obvio observar que si existe un hiperplano de separación, este no es único. De manera que tendremos que escoger entre todos los hiperplanos de separación aquel que satisfaga algunas condiciones de optimalidad. Una vez obtenido el hiperplano de separación óptimo, lo utilizamos para clasificar un nuevo punto del universo de datos dependiendo del “lado” del hiperplano en el cual quede ubicado. Si los datos de entrenamiento originales no son separables linealmente en \mathbb{R}^n entonces podemos transformar, a través de funciones Kernel, los puntos a un espacio de dimensión mayor en el cual si sean separables. Esto permite clasificar un nuevo punto mediante la clasificación lineal en el espacio transformado de su imagen funcional.

Estudiemos estos conceptos más formalmente. Para ello, sea $D = \{(x_1, y_1), \dots, (x_\ell, y_\ell)\}$ una muestra de datos etiquetados donde $x_i \in \mathbb{R}^n$ con $i = 1, \dots, \ell$ y la etiqueta viene dada por y_i donde:

$$y_i = f(x_i) = \begin{cases} 1 & \text{si } x_i \in \text{Clase positiva} \\ -1 & \text{si } x_i \in \text{Clase negativa} \end{cases}$$

queremos construir una función $f(\mathbf{x})$ que nos permita clasificar al punto $\mathbf{x} \in \mathbb{R}^n$ de acuerdo al entrenamiento recibido del conjunto de muestras D . Usualmente $f(\mathbf{x})$ está asociada a una superficie $H \subset \mathbb{R}^n$ que llamamos clasificador.

Las Máquinas de Vectores Soporte (SVM) distinguen tres situaciones diferentes para determinar la superficie óptima de clasificación según utilizemos un clasificador lineal o bien no-lineal, estas tres opciones son:

1. S linealmente separable en \mathbb{R}^n y H lineal. (Caso Separable)
2. S no es linealmente separable en \mathbb{R}^n y H lineal. (Hiperplano de margen suave)

3. S no es linealmente separable y H no es lineal. (Máquinas de Vectores Soporte no-lineal)

donde $S = \{x_1, \dots, x_\ell\} \subset \mathbb{R}^n$.

Caso Separable

En esta situación el conjunto de datos S es separable linealmente en \mathbb{R}^n , esto es, existe al menos un hiperplano H en \mathbb{R}^n que separa los puntos de S de acuerdo a su etiqueta (ver Figura 2.1), de forma tal que todos los puntos de S de una misma clase quedan a un mismo lado del hiperplano H . Obviamente como observamos hay muchos hiperplanos que separan los puntos de S , pero ¿cuál es mejor? Para ello, sea $H : \langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ uno de tales hiperplanos de separación donde \mathbf{w} es el vector director ortogonal al plano H y b es una constante. Los puntos que están sobre H satisfacen la ecuación $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$. Los que están a un “lado” satisfacen que $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b > 0$ y los otros $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b < 0$.

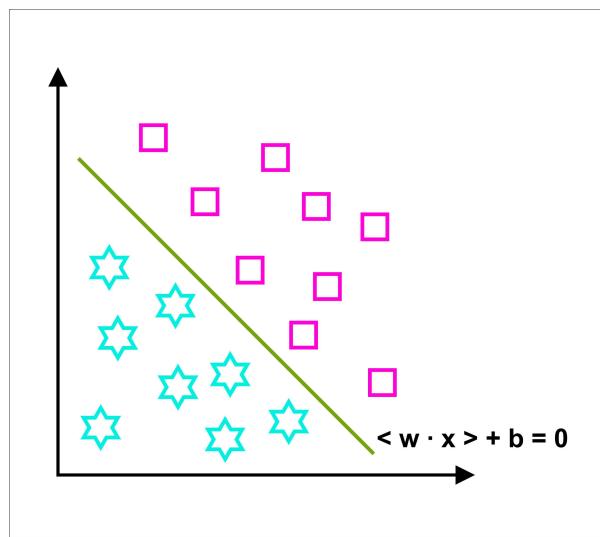


Figura 2.1: Caso Separable

Sin pérdida de generalidad y utilizando una normalización para \mathbf{w} y b , podemos suponer que:

$$\langle \mathbf{w} \cdot x_i \rangle + b \geq 1 \quad \text{si } y_i = 1$$

$$\langle \mathbf{w} \cdot x_i \rangle + b \leq -1 \quad \text{si } y_i = -1$$

o equivalentemente:

$$y_i(\langle \mathbf{w} \cdot x_i \rangle + b) - 1 \geq 0 \quad i = 1, \dots, \ell$$

Sean ahora H_1 y H_2 los hiperplanos paralelos a H que pasan por los puntos más cercanos a H de la clase positiva y la clase negativa respectivamente. Ver Figura 2.2.

DEFINICIÓN 4 (Margen de un hiperplano de separación) *Se define el margen M de un hiperplano de separación H como la distancia Euclídea o geométrica entre H_1 y H_2 .*

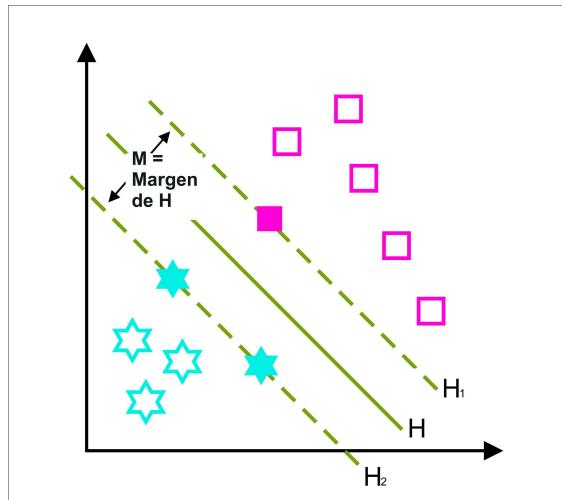


Figura 2.2: Hiperplanos paralelos a H , H_1 y H_2 . Definición del margen M

Calculemos esta distancia.

Sea \mathbf{s} un punto de H_1 y sea \mathbf{t} el punto de H_2 más cercano a \mathbf{s} , esto es la proyección perpendicular de \mathbf{s} sobre H_2 . Esto significa que \mathbf{s} y \mathbf{t} están sobre una línea perpendicular a los planos H_1 y H_2 , y por

lo tanto $\mathbf{s} = \mathbf{t} + \alpha \mathbf{w}$ (ecuación paramétrica de la recta) ya que el vector director de la recta que une a \mathbf{s} y a \mathbf{t} es paralelo al vector normal del plano (\mathbf{w}). De aquí tenemos que en margen de H sería $M = |\mathbf{s} - \mathbf{t}|$ que junto con las otras expresiones tenemos que:

$$\langle \mathbf{w} \cdot \mathbf{s} \rangle + b = 1 \quad (2.3)$$

$$\langle \mathbf{w} \cdot \mathbf{t} \rangle + b = -1 \quad (2.4)$$

$$\mathbf{s} = \mathbf{t} + \alpha \mathbf{w} \quad (2.5)$$

$$M = |\mathbf{s} - \mathbf{t}| \quad (2.6)$$

Sustituyendo (2.5) en (2.3) obtenemos lo siguiente:

$$\langle \mathbf{w} \cdot (\mathbf{t} + \alpha \mathbf{w}) \rangle + b = 1$$

$$\langle \mathbf{w} \cdot \mathbf{t} \rangle + \alpha \langle \mathbf{w} \cdot \mathbf{w} \rangle + b = 1$$

Pero por (2.4):

$$-1 + \alpha \langle \mathbf{w} \cdot \mathbf{w} \rangle = 1$$

$$\alpha = \frac{2}{\langle \mathbf{w} \cdot \mathbf{w} \rangle}$$

De aqui que el margen M del plano H venga dado por:

$$M = |\mathbf{s} - \mathbf{t}| = |\alpha \mathbf{w}| = \frac{2}{\langle \mathbf{w} \cdot \mathbf{w} \rangle} \|\mathbf{w}\| = \frac{2}{\|\mathbf{w}\|}$$

Luego un criterio para optimizar la selección del hiperplano de separación sería seleccionar el plano H como aquel que maximize el margen M , lo cual podríamos plantear como el problema de optimización:

PROBLEMA 4

$$\text{minimizar}_{\mathbf{w} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{sujeto a: } y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, \dots, \ell$$

Resolvamos el Problema 4 (2.2.2) mediante los multiplicadores de Lagrange para lo cual construimos la función de Lagrange definida como:

$$\begin{aligned} L(\mathbf{w}, b) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \lambda_i [y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1] \\ L(\mathbf{w}, b) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \lambda_i y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) + \sum_{i=1}^{\ell} \lambda_i \end{aligned} \quad (2.7)$$

Debemos minimizar $L(\mathbf{w}, b)$ con respecto a \mathbf{w} y a b anulando su vector gradiente. Además las derivadas parciales con respecto a todos los λ_i deben ser anuladas, y se tiene que satisfacer que $\lambda_i \geq 0$.

Este problema de programación cuadrática (QP) es convexo, ya que la función objetivo es convexa y el conjunto de restricciones es convexo por ser la intersección de conjuntos convexos ya que cada restricción lineal es convexa en sí.

Igualando el gradiente de $L(\mathbf{w}, b)$ con respecto a \mathbf{w} y b a cero obtenemos las siguientes condiciones:

$$\mathbf{w} = \sum_{i=1}^{\ell} \lambda_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{\ell} \lambda_i y_i = 0$$

Sustituyendo estas ecuaciones en (2.7) obtenemos el siguiente problema dual (Fletcher 1987 [12]):

PROBLEMA 5

$$\begin{aligned} & \text{maximizar}_{\lambda} \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \lambda_i \lambda_j y_i y_j \langle x_i \cdot x_j \rangle = \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2} \| \mathbf{w} \|^2 \\ & \text{sujeto a: } \sum_{i=1}^{\ell} \lambda_i y_i = 0 \end{aligned}$$

$$\lambda_i \geq 0, \quad i = 1, \dots, \ell$$

y de aquí obtenemos que el \mathbf{w} óptimo es $\bar{\mathbf{w}} = \sum_{i=1}^{\ell} \bar{\lambda}_i y_i x_i$ con $\bar{\lambda}_i$ las soluciones del Problema 5 (2.2.2). En base a esto podemos observar las siguientes características de la solución óptima:

1. El hiperplano óptimo de separación $\langle \bar{\mathbf{w}} \cdot \mathbf{x} \rangle + \bar{b} = 0$ puede ser escrito como una combinación lineal de los vectores de entrenamiento x_1, \dots, x_ℓ , esto es:

$$\sum_{i=1}^{\ell} \bar{\lambda}_i y_i \langle x_i \cdot \mathbf{x} \rangle + \bar{b} = 0 \quad (2.8)$$

2. Aquellos vectores de entrenamiento para los cuales $\bar{\lambda}_i > 0$ son los *Vectores Soporte y son los que definen al hiperplano óptimo* (SV).
3. El hiperplano de separación sólo depende de los Vectores Soporte.
4. Dado un nuevo dato z que pertenece a \mathbb{R}^n para ser etiquetado, calculamos el clasificador como:

$$f(\mathbf{z}) = \text{signo}(\langle \bar{\mathbf{w}} \cdot \mathbf{z} \rangle + \bar{b}) = \text{signo}\left(\sum_{i=1}^{\ell} \bar{\lambda}_i y_i \langle x_i \cdot \mathbf{z} \rangle + \bar{b}\right)$$

Si $f(\mathbf{z}) > 0$ etiquetamos a \mathbf{z} como positivo, en caso contrario como negativo.

5. El problema de optimización (Problema 4) de las Máquinas de Vectores Soporte (SVM) satisface las condiciones necesarias y suficientes del Teorema de Karush-Kuhn-Tucker -descrito en la sección 2.1.3- ya que la función objetivo es convexa así como la región definida por las restricciones.

Hiperplano de margen suave

Para extender el algoritmo anterior para manejar situaciones con conjuntos de entrenamiento no separables linealmente y utilizando un hiperplano como clasificador, debemos relajar las condiciones del Problema 4 (2.2.2) introduciendo unas variables positivas de holgura (Cortes y Vapnik en [8]), ξ_1, \dots, ξ_ℓ transformando el problema tal como se muestra en la Figura 2.3. Sea $\xi_i = 0$ si no hay error al ser clasificado el punto x_i y sea $\xi_i = \text{distancia de } x_i \text{ al hiperplano } H$, en otro caso.

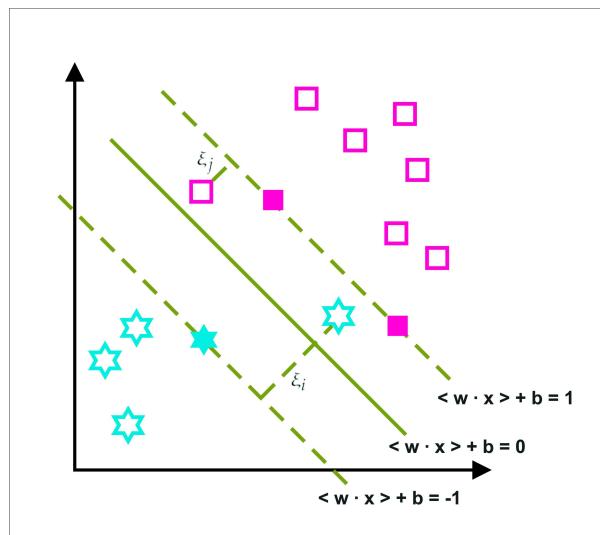


Figura 2.3: Hiperplano de Margen Suave

De esta forma, problema quedaría de la siguiente manera:

PROBLEMA 6

$$\text{minimizar}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + c \sum_{i=1}^{\ell} \xi_i$$

$$\text{su}jeto\ a: y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad i = 1, \dots, \ell$$

$$\xi_i \geq 0$$

Este problema en su expresión dual adquiere la siguiente forma:

PROBLEMA 7

$$\begin{aligned} & \text{maximizar}_{\lambda} \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \lambda_i \lambda_j y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \\ & \text{su}jeto\ a: \sum_{i=1}^{\ell} \lambda_i y_i = 0 \end{aligned}$$

$$0 \leq \lambda_i \leq c, \quad i = 1, \dots, \ell$$

Y el punto óptimo para \mathbf{w} tendría la forma de $\bar{\mathbf{w}} = \sum_{i=1}^{\ell} \bar{\lambda}_i y_i \mathbf{x}_i$. La única diferencia con respecto al caso separable es que existe una cota superior positiva c sobre todos los $\bar{\lambda}_i$, y en la práctica este valor puede ser determinado utilizando el método de validación cruzada junto con los otros parámetros presentes en el kernel, es decir, se van probando distintos valores de los parámetros desconocidos (inclusive c) hasta obtener aquellos que mejor se ajusten al problema que se está trabajando, esto significa que minimize el error de validación. El parámetro c representa el costo unitario de penalización para los puntos mal clasificados. Es bueno señalar que valores muy grandes de c pueden producir un sobreajuste de la superficie de clasificación, aumentando el error de validación. El aumento del error de validación también ocurre para valores muy pequeños de c .

El Problema 7 (2.2.2) satisface las condiciones necesarias y suficientes de Karush-Kuhn-Tucker (ver [2.1.3]), y la solución óptima viene dada por $\bar{\mathbf{w}} = \sum_{i=1}^{\ell} \bar{\lambda}_i y_i \mathbf{x}_i$. El hiperplano de separación óptimo $\langle \bar{\mathbf{w}} \cdot \mathbf{x} \rangle + \bar{b} = 0$ permite etiquetar un nuevo punto, al igual que en el caso anterior, de acuerdo al clasificador $f(\mathbf{z}) = \text{signo}(\langle \bar{\mathbf{w}} \cdot \mathbf{z} \rangle + \bar{b})$. En este caso el hiperplano $\langle \bar{\mathbf{w}} \cdot \mathbf{x} \rangle + \bar{b} = 0$ recibe el

nombre de *Hiperplano de Margen Suave*.

Análogamente al caso anterior, para decidir la etiqueta de un nuevo punto de dato \mathbf{z} usamos:

$$f(\mathbf{z}) = \text{signo}(\langle \bar{\mathbf{w}} \cdot \mathbf{z} \rangle + \bar{b}) = \text{signo}\left(\sum_{i=1}^{\ell} \bar{\lambda}_i y_i \langle x_i \cdot \mathbf{z} \rangle + \bar{b}\right)$$

Máquinas de Vectores Soporte no-lineales

Las Máquinas de Vectores Soporte no-lineales se refieren al uso de una superficie de decisión no lineal al momento de clasificar o etiquetar un nuevo valor \mathbf{z} , y generalmente van a ser usadas en los casos en que el conjunto de entrenamiento no permite una separación lineal.

Boser, Guyon y Vapnik [4] probaron que este problema se puede resolver proyectando los datos de entrenamiento $x_1, \dots, x_\ell \in \mathbb{R}^n$ sobre otro espacio euclidiano \mathcal{H} a través de una función $\Phi : \mathbb{R}^n \rightarrow \mathcal{H}$ (ver Figura 2.4).

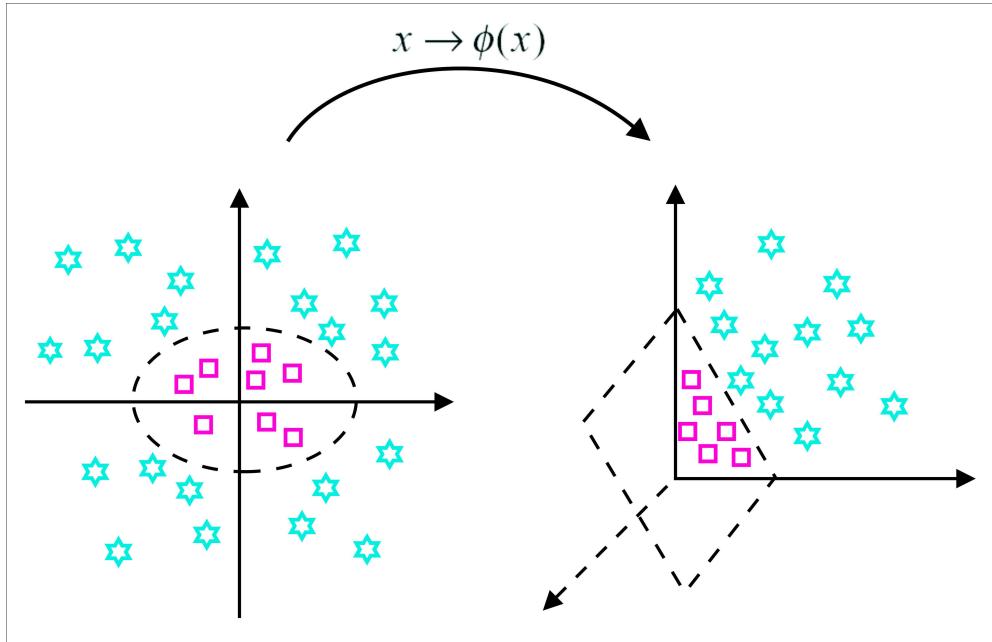


Figura 2.4: Transformación de datos a una dimensión superior mediante el uso de $\Phi(\mathbf{x})$

La dimensión de \mathcal{H} usualmente es mayor que n e inclusive podría ser infinita. A \mathcal{H} se le llama *Espacio de características*.

La idea de esta máquina de entrenamiento no lineal es separar en \mathcal{H} los puntos de entrenamiento transformados, $\Phi(x_1), \dots, \Phi(x_\ell)$ a través del hiperplano de separación óptimo en \mathcal{H} , y en función de esta superficie de decisión clasificar un punto \mathbf{z} mediante el uso de $\Phi(\mathbf{z})$. Para ello observemos que en el espacio transformado \mathcal{H} la función de decisión o el hiperplano óptimo de separación se construiría por medio de los vectores transformados $\Phi(x_1), \dots, \Phi(x_\ell)$.

Sustituyendo en 2.8 el vector \mathbf{x} por el vector $\Phi(\mathbf{x})$ o equivalentemente utilizando los vectores $\Phi(x_1), \dots, \Phi(x_\ell)$ obtenemos el hiperplano de separación óptimo en \mathcal{H} con $\mathbf{u} \in \mathcal{H}$ como:

$$\sum_{i=1}^{\ell} \bar{\lambda}_i y_i \langle \Phi(x_i) \cdot \mathbf{u} \rangle + \bar{b} = 0$$

y el clasificador para un nuevo punto $\mathbf{z} \in \mathbb{R}^n$ sería:

$$f(\mathbf{z}) = signo \left(\sum_{i=1}^{\ell} \bar{\lambda}_i y_i \langle \Phi(x_i) \cdot \Phi(\mathbf{z}) \rangle + \bar{b} \right)$$

Podemos observar que para un punto $\mathbf{z} \in \mathbb{R}^n$ la decisión de la máquina (SVM) depende del conocimiento de los productos escalares:

$$\langle \Phi(x_i) \cdot \Phi(\mathbf{z}) \rangle \quad i = 1, \dots, \ell$$

De hecho solo depende de los productos escalares de los Vectores Soporte $\Phi(x_i) \in \mathcal{H}$ con $\bar{\lambda}_i > 0$, de manera que la máquina de entrenamiento no necesita conocer la función de transformación $\Phi(\mathbf{z})$, sino que basta con los productos internos $\langle \Phi(x_i) \cdot \Phi(\mathbf{z}) \rangle$ para los Vectores Soporte en \mathcal{H} .

Ahora si existiera una función Kernel $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ tal que $K(x_i, x_j) = \langle \Phi(x_i) \cdot \Phi(x_j) \rangle$ entonces el algoritmo de entrenamiento solo dependería del conocimiento de la función Kernel $K(\mathbf{x}, \mathbf{y})$ y no de la transformación $\Phi(\mathbf{x})$.

No toda función $K(\mathbf{x}, \mathbf{y}) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ puede ser escrita como una función Kernel, el Teorema de Mercer (Vapnik, 1995 [31]) establece las condiciones matemáticas para que una función $K(\mathbf{x}, \mathbf{y})$ pueda ser una función Kernel, y por lo tanto hacer uso de ella en el entrenamiento de una SVM.

Algunas funciones Kernel usadas para la identificación y reconocimiento de patrones son las siguientes ([28, 9, 21]):

1. Lineal: $K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle$
2. Polinomial: $K(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x} \cdot \mathbf{z} \rangle + c)^d$
3. Sigmoidal: $K(\mathbf{x}, \mathbf{z}) = \tanh(\kappa \langle \mathbf{x} \cdot \mathbf{z} \rangle + \Theta)$

$$4. \text{ Gaussiano: } K(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/(2\sigma^2))$$

El kernel *polinomial* presenta una clasificación polinomial de grado d . El kernel *gaussiano* nos presenta el clasificador Gaussiano RBF y por último el kernel *sigmoidal* presenta un clasificador Perceptron de múltiples capas.

2.3. Método de Reducción de Burges

El número esperado de vectores soporte N_s en el entrenamiento de una Máquina de Vectores Soporte vienen dados por $(\ell - 1)E(p)$, donde $E(p)$ es el valor esperado de la probabilidad del error de prueba y ℓ es el número de datos utilizados en el entrenamiento (Burges [6]). Esto significa que el número N_s es de orden ℓ .

En el procesamiento de imágenes es necesario clasificar una gran cantidad de sub-imágenes, lo cual representa un aumento considerable del tiempo necesario para clasificar la imagen original, sobre todo cuando tenemos que recorrer ampliamente una imagen para la búsqueda de patrones de caras en imágenes más pequeñas.

Burges [6] propone en su trabajo, un algoritmo de reducción del número de vectores soporte que para el caso de Máquinas de Vectores Soporte con kernel polinomial de orden 2, resulta en una reducción importante del número de vectores soporte sin aumento del error de validación. En este caso el número de vectores soporte se puede reducir a la dimensión del espacio del conjunto de entrenamiento, lográndose de esta forma una reducción sustancial del número de operaciones a realizar en la clasificación de un punto del espacio de muestras.

La aplicación del método de Burges permite pre-establecer el número de vectores N_z del conjunto reducido, para luego calcular los nuevos vectores z_1, \dots, z_{N_z} y las constantes $\gamma_1, \dots, \gamma_{N_z}$ y clasificar a un punto dado $\mathbf{x} \in \mathbb{R}^n$ a través de:

$$f(\mathbf{x}) = \sum_{i=1}^{N_z} \gamma_i K(\mathbf{z}_i, \mathbf{x}) + b$$

Para kernel polinomial de grado 2, con $N_z = \dim \mathbf{x}$ el clasificador de Burges produce exactamente los mismos resultados que la SVM original, pero con una reducción significativa del número de vectores soporte a utilizar.

Si fijamos $N_z < \dim \mathbf{x}$ entonces el clasificador obtenido por el método de Burges va a representar una aproximación al clasificador original de la SVM, cuyo error debemos de valorar a la hora de utilizar esta técnica.

2.4. Algoritmos de Detección del Color Piel

El uso de la información del color de los pixels de una imagen para identificar la presencia del color piel, ha sido incorporado entre las técnicas de localización de patrones de caras humanas en una imagen. Esto es posible ya que la piel humana contiene una distribución especial de colores que permiten construir un modelo para su representación.

Los algoritmos que utilizan los colores de los pixels permiten un procesamiento rápido de las imágenes digitales. También estos algoritmos son robustos con respecto a cambios geométricos o de posición de los patrones presentes en una imagen.

Para construir un algoritmo que use el color como una característica para detectar la presencia de caras humanas, debemos inicialmente seleccionar el espacio de colores en el que vamos a representar la información de los pixels, y posteriormente definir un modelo de representación del color piel que nos permita clasificar un pixel como color piel o color no-piel independientemente de los pixels vecinos.

RGB es un espacio de colores en el cual cada color es representado como una combinación de tres rayos de colores (R, G, B) rojo, verde y azul. Es uno de los espacios más utilizados para el procesamiento y almacenamiento de imágenes.

Un método para construir un clasificador de piel es definir explícitamente un cluster o sub-conjunto en el espacio de colores RGB que permita clasificar un pixel como piel dependiendo si sus componentes (R, G, B) pertenecen al cluster.

En este trabajo, vamos a utilizar como cluster en el espacio RGB el propuesto por Peer [22], que establece lo siguiente:

Si

$$R > 95, G > 40, B > 20, \max(R, G, B) - \min(R, G, B) > 15,$$

$$| (R - G) | > 15, R > G \text{ y } R > B$$

Entonces

el pixel (R, G, B) se clasifica como color piel.

No todo pixel que el algoritmo detecte como piel realmente representa un punto de piel en la imagen real. Este algoritmo puede ser utilizado para descartar aquellos puntos cuyos colores no corresponden al color piel. A aquellos puntos detectados como piel le aplicaremos, a continuación otro algoritmo,

como por ejemplo un clasificador SVM de patrones de caras y así decidir su clasificación. De esta forma usaremos el clasificador SVM solamente para aquellos puntos detectados previamente por el clasificador de piel, reduciéndose significativamente el tiempo de procesamiento de una imagen.

2.5. Descripción de la Librería SvmFu 3

La librería SvmFu 3 es una paquete que implementa las Máquinas de Vectores Soporte que se describieron anteriormente (ver 2.2.2). Fue escrito por Ryan Rifkin en el MIT. Esta librería está escrita en C++, y no necesita ninguna optimización adicional.

En la descripción que la librería nos presenta [24], observamos que el algoritmo de optimización utilizado es de tipo secuencial, en el cual en cada iteración es solucionado un problema de optimización con h patrones del conjunto de entrenamiento, tal como describimos en el Problema 7 (ver 2.2.2), de manera de solucionar pequeños sub-problemas en lugar de uno solo con muchos patrones de entrada. Esta librería maneja únicamente problemas de etiquetado o clasificación. En el Apendice A se encuentran las instrucciones de instalación de la librería y sus requerimientos.

Esta librería trabaja en dos partes: primero están los programas de entrenamiento de la Máquina de Vectores Soporte y posteriormente los programas de pruebas de la máquina. Los programas de entrenamiento leen un conjunto de vectores o datos de entrenamiento y almacenan la Máquina de Vectores Soporte en un archivo. Los programas de prueba leen un conjunto de datos de prueba y usan la Máquina de Vectores Soporte almacenada previamente.

Los datos que maneja la librería SvmFu 3, ya sea en las componentes de los vectores de entrenamiento, de los vectores de prueba o los valores de los parámetros de las funciones kernel pueden ser enteros, flotantes o dobles.

Los conjuntos de patrones de entrenamiento y prueba están formados por vectores de $n + 1$ componentes, donde las primeras n componentes representan al vector de entrenamiento y la última componente la clasificación asociada a ese vector. Estos conjuntos son leídos por la librería SvmFu 3.1 como un archivo de texto ASCII en cualquiera de los siguientes formatos: dense, sparseN o sparse01. Para todos los formatos, los datos consisten en números (int, float o double) separados por espacios en blanco.

El formato “dense” viene dado a partir de un archivo descrito por:

- * El número de datos
- * La dimensión de los datos
- * Los datos normalizados, uno tras otro, con su respectiva etiqueta al final.

Por ejemplo,

```
3 5  
0.1 0 0.5 0 0.9 -1  
0.2 0.4 0 0 0 1  
0 0 0.7 0.6 1 1
```

Los formatos sparseN, sparse01 se utilizan cuando la mayoría de las componentes de los vectores de entrenamiento son ceros, o bien ceros y unos respectivamente.

Adicionalmente es necesario suministrarle al programa de entrenamiento de la Máquina de Vectores Soporte los valores de los parámetros siguientes: el costo lineal por unidad c para el caso del Hiperplano

de Margen Suave; la tolerancia en el cumplimiento de las condiciones Karush-Khun-Tucker y la función kernel.

La clasificación de un nuevo vector de datos utilizando la Máquina de Vectores Soporte previamente entrenada, viene dado por la evaluación de la función *outputAtDataPt* del programa *svmftest* de la librería SvmFu 3.1.

2.6. Manejo de la Cámara de Video

Uno de los objetivos más relevantes de esta aplicación es la implementación de un programa que permita la captura y detección de patrones de caras mediante el uso de imágenes provenientes de secuencias de video en tiempo real.

La cámara web que utilizamos en esta aplicación es *Genius WebCam Express*, la cual instalamos en el sistema operativo, utilizando el driver *Spca5xx* cuyas instrucciones de instalación se encuentran en el Apéndice A.

El driver *Spca5xx* posee varias estructuras las cuales permiten el manejo de la cámara web. Entre ellas merece la pena señalar la estructura denominada *spca5xx_frame* que nos permite obtener en un momento determinado la imagen que está captando la cámara y sus características, tales como:

1. Ancho de la imagen en pixels.
2. Alto de la imagen en pixels.
3. Profundidad del color de la imagen en bytes por pixel.

4. Conjunto de pixels que definen la imagen como un vector de Alto*Ancho*3 valores de intensidades correspondientes al sistema de colores RGB.

El manejo de la cámara web también hace uso de la librería < **SDL.h** > ubicada en el sistema operativo Ubuntu. Una de las estructuras utilizadas de la librería es *SDL_Surface*. *SDL_Surface* representa áreas de memoria gráfica, esto es memoria que puede ser dibujada. Las funciones SDL que utilizamos son las siguientes:

1. *SDL_SetVideoMode*: permite especificar el modo de la superficie SDL señalando su ancho, alto y profundidad.
2. *SDL_Flip*: permite intercambiar los buffers y actualizar la pantalla.
3. *SDL_LockSurface*: permite bloquear una superficie de acceso directo.
4. *SDL_UnlockSurface*: permite desbloquear una superficie bloqueada.
5. *SDL_ShowCursor*: permite ver el cursor sobre la superficie *SDL_Surface*.
6. *SDL_WM_SetCaption*: permite establecer el nombre de la ventana que contiene la superficie *SDL_Surface*.

Capítulo 3

Diseño del Sistema

Un detector de caras consiste en un sistema o programa que permite identificar y localizar patrones de caras humanas en una imagen.

El sistema desarrollado esta dividido en dos capas, la primera de ellas consiste en un conjunto de programas, procedimientos y funciones desarrollados en lenguaje C para ser utilizados en forma *batch* o *estática*. La segunda capa comprende el desarrollo de una aplicación *online* o *dinámica* que permite la detección y seguimiento del color piel en una imagen dada al igual que la detección de patrones de caras utilizando una Máquina de Vectores Soporte.

El nombre de los programas desarrollados en este proyecto sigue la convención de comenzar con el prefijo *SDPC*, que significa Sistema de Detección de Patrones de Caras.

3.1. Esquema de la aplicación

Basándonos en diversos sistemas de detección de caras, como los presentados por Osuna-Freund-Girosi [21], por Heisele-Ho-Wu-Poggio [15] y por Pontil-Verri [23], se implementa un sistema de detección de patrones de caras en el que se realizan los siguientes procesos: batch y online.

Los procesos batch o estáticos, son aquellos procesos que conducen a la producción de cantidades finitas de datos a partir de un conjunto de datos de entrada. Estos datos de entrada pasan por un conjunto de actividades de procesamiento en un periodo de tiempo finito. Mientras que los procesos online producen una cantidad infinita de datos en un periodo de tiempo continuo.

Las siguientes fases constituyen los pasos principales realizados durante el desarrollo de nuestra aplicación:

1. Obtención de la base de datos de imágenes para el entrenamiento de la SVM.
2. Pre-procesamiento de imágenes.
3. Uso de la librería SvmFu 3.
4. Bootstrapping.
5. Detector de patrones de caras en imágenes estáticas.
6. Implantación de un algoritmo de detección del color piel.
7. Captura y clasificación de las imágenes provenientes de la cámara web.

Las fases 1, 2, 3, 4 y 5 están desarrolladas para ser ejecutadas en forma *batch*, mientras que las fases 6 y 7 en forma *online*.

3.2. Procesos Batch o Estáticos

Mediante el uso de procesos batch, buscamos obtener una Máquina de Vectores Soporte entrenada para la detección de patrones de caras en imágenes estáticas. Para ello desarrollamos un conjunto de

módulos, tanto de pre-procesamiento del conjunto de datos entrantes: patrones de imágenes de 19×19 etiquetadas como caras o no-caras, como de prueba de la SVM entrenada: un detector de patrones de caras en imágenes estáticas. En la Figura 3.1 ilustramos los procesos batch realizados.

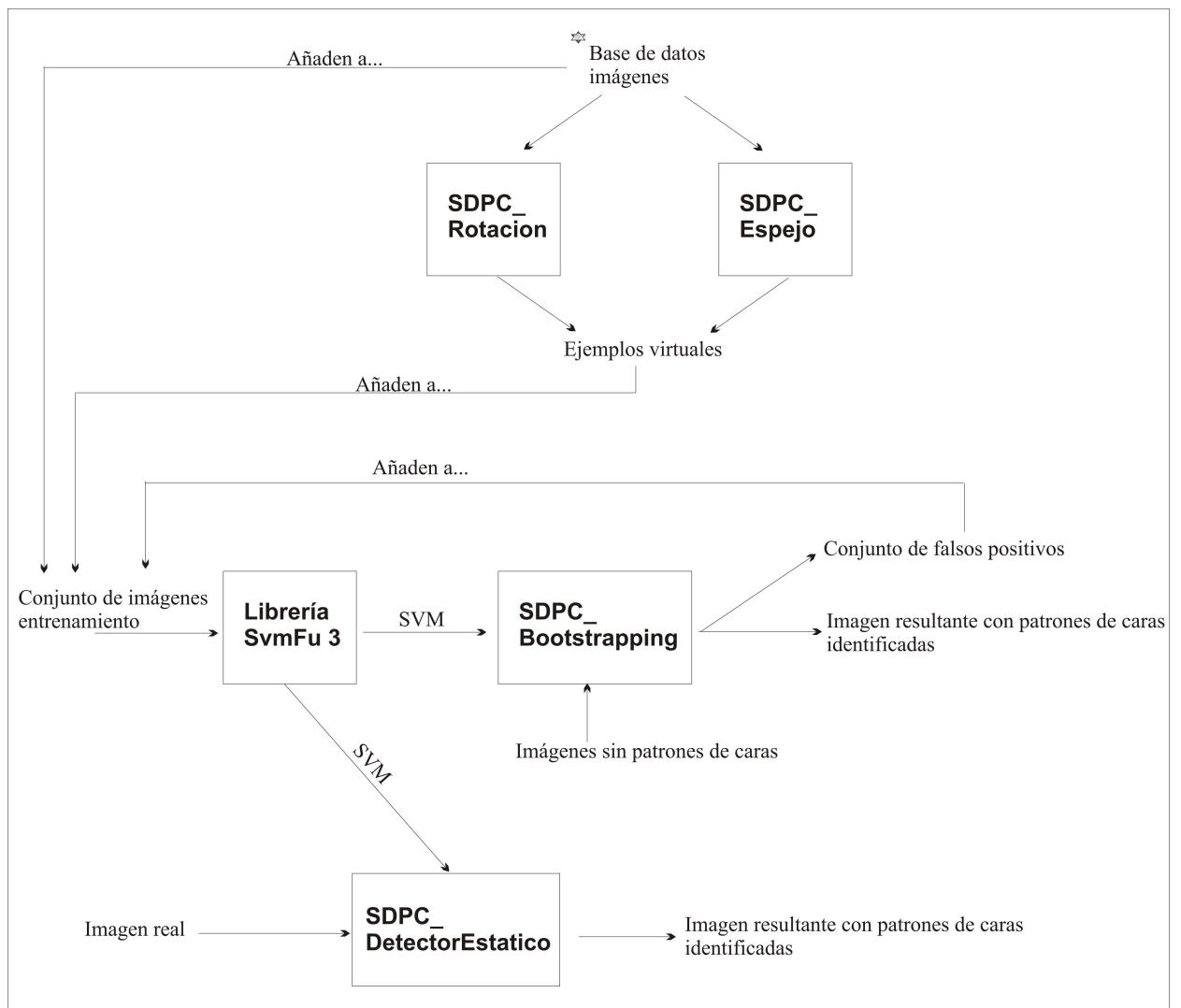


Figura 3.1: Diagrama de los procesos batch

Observamos en la Figura 3.1 que para la obtención de la Máquina de Vectores Soporte entrenada, el conjunto de datos inicial debe pasar por las siguientes fases:

1. Obtención de la base de datos de imágenes para el entrenamiento de la SVM

Para generar la base de datos de vectores de entrenamiento de la Máquina de Vectores Soporte empleada para detectar patrones de caras, utilizamos una base de datos construida por imágenes en formato PGM de 19×19 pixels clasificadas previamente como caras o no-caras. Esta base de datos la obtuvimos de la página web del MIT del “Center for Biological and Computational Learning”, y consiste en 2429 patrones clasificados como caras y 4548 patrones clasificados como no-caras. Ver Figura 3.2.



Figura 3.2: Ejemplos de patrones de caras y no-caras de la base de datos

Para cada uno de los patrones se generaron dos nuevos patrones obtenidos a través de las operaciones de espejo y rotación de 180 grados para la imagen original, tal como se muestra en la Figura 4.2. Este proceso permitió ampliar la base de datos de caras a 7287 imágenes.

2. Pre-procesamiento de imágenes y generación de ejemplos virtuales de imágenes

Todas las imágenes poseen ciertas fuentes de variaciones y para compensar estas variaciones antes de ser utilizadas en el entrenamiento de la SVM hacemos un pre-procesamiento de la data, aplicándoles a las mismas las operaciones de corrección del gradiente de iluminación, ecualización del histograma, máscara y normalización.

Además, como mencionamos en el párrafo anterior, para cada una de las imágenes caras existentes en la base de datos, generamos nuevas imágenes caras virtuales mediante las operaciones de rotación y espejo.

Las operaciones anteriores fueron implementadas a través de los siguientes programas desarrollados en lenguaje C:

a) *SDPC_Iluminacion.c*

Dada una imagen en formato PGM, *nombreImagen*, este programa corrige el gradiente de iluminación de la misma y devuelve la imagen modificada, *nombreImagenResultante*, en formato PGM. El pseudocódigo de este programa viene dado por:

```

Cargar la imagen inicial, nombreImagen, en un vector fila
Normalizar nombreImagen.
Para i desde 0 hasta nombreImagen.longitudImagen
    promedio + =  $\frac{\text{intensidad}[i]}{\text{longitudImg}}$ 
Calcular la constante de ajuste:
    constante = 0.5 - promedio
Calcular las intensidades de nombreImagenResultante en función de las
intensidades de nombreImagen:
    Para i desde 0 hasta nombreImagen.longitudImagen, hacer:
        Si  $0 < \text{intensidad}[i] + \text{constante} < 1$ , entonces:
            intensidad[i] = intensidad[i] + constante
Escribir la imagen resultante, nombreImagenResultante, en formato
PGM.

```

Figura 3.3: Algoritmo de corrección del gradiente de iluminación

b) *SDPC_Histograma.c*

Dada una imagen en formato PGM, *nombreImagen*, este programa ecualiza el histograma de la misma y devuelve la imagen modificada, *nombreImagenResultante*, en formato PGM.

El pseudocódigo de este programa viene dado por:

Cargar la imagen inicial, *nombreImagen*, en un vector fila.
 Normalizar *imagen*.

Calcular las ocurrencias de las intensidades de *imagen*:

Para i desde 0 hasta *imagen.valorMaximoGrises*, hacer:

$$\begin{aligned} desplazamiento &= intensidad[i] \\ ocurrencia[desplazamiento] &+= 1 \end{aligned}$$

Calcular el histograma acumulado de *imagen*:

$$\begin{aligned} acumulado[0] &= ocurrencia[0] \\ \text{Para i desde 1 hasta } &imagen.valorMaximoGrises, \text{ hacer:} \\ acumulado[i] &= acumulado[i-1] + ocurrencia[i] \end{aligned}$$

Ajustar las intensidades de *imagen*:

Para i desde 0 hasta *imagen.longitudImagen*

$$intensidad[i] = \frac{acumulado[i]}{acumulado[longitudImagen]}$$

Calcular el máximo *M* y el mínimo *m* acumulado.

Calcular las intensidades de la *imagenResultante* en función de las intensidades de la *imagen*:

Para i desde 0 hasta *imagen.longitudImagen*

$$intensidad[i] = \frac{intensidad[i] - m}{M - m}$$

Escribir la imagen resultante, *imagenResultante*, en formato PGM.

Figura 3.4: Algoritmo de ecualización del histograma

c) SDPC_Rotacion.c

Dado un conjunto de imágenes en formato PGM, *listaNombreImagen*, este programa genera un conjunto de imágenes rotadas un ángulo β con respecto al centro de la imagen. Para ello se define el centro de la imagen y se transforma la imagen con las operaciones de rotación.

Se considera la posibilidad de que al girar la imagen un determinado grado, algunas regiones de la imagen queden sin valores, a estos pixels se le coloca el color negro. Esto ocurre la mayoría de las veces en las esquinas.

El pseudocódigo de este programa viene dado por:

Abrir el archivo *listaNombreImagen*.

Mientras no se haya alcanzado el final del archivo, hacer:

Cargar de la imagen inicial, *imgin*, que esta siendo apuntada por en archivo en una estructura.

Calcular el seno y coseno de β .

Calcular los límites superior-inferior y derecha-izquierda.

Definir el centro de rotacion.

Para i de 0 a *imgin.anchoImagen*

 Para j de 0 a *imgin.altoImagen*

 Calcular el valor de la transformada, *trans*.

 Calcular el valor donde debería ir la transformada al concluir la rotación.

 Si la parte entera de la transformada está en los límites de la imagen,
entonces:

$$\text{imgout.pixel}[i,j] = \text{trans}$$

 Si no,

$$\text{imgout.pixel}[i,j] = \text{colorNegro}$$

Enmascara la imagen resultante *imgout* en un vector *imgMascara*.

Escribir *imgMascara* en el *archivoSaliente*

Cerrar el archivo *listaNombreImagen*

Figura 3.5: Algoritmo del programa de rotación

d) *SDPC_Espejo.c*

Dado un conjunto de imágenes en formato PGM, *listaNombreImagen*, este programa genera el conjunto de imágenes espejo de las mismas. El pseudocódigo de este programa viene dado por:

Abrir archivo *listaNombreImagen*.

Mientras no se haya alcanzado el final del archivo, hacer:

Cargar de la imagen inicial, *imgin*, que esta siendo apuntada por en archivo en un vector fila.

Generar la imagen espejo resultante:

$$\text{imgout.intensidad}[i, j] = \text{imgin.intensidad}[i, \text{imgin.anchoImagen}-j]$$

Enmascarar *imgout* en un vector fila, *imgMascara*.

Escribir *imgMascara* en el *archivoSaliente*

Cerrar del archivo *listaNombreImagen*.

Figura 3.6: Algoritmo del programa de espejo

Adicionalmente, se tiene el programa *SDPC_manejoArchivoTrain.c* el cual lee la base de datos

original de caras y no-caras, y la base de datos de ejemplos virtuales generados anteriormente y construye un archivo de tipo “dense” en el cual están contenidos todos los datos muestrales a ser entrenados por la librería SvmFu 3.

3. Uso de la librería SvmFu 3

Entrenamos una Máquina de Vectores Soporte utilizando la librería SvmFu 3.1 y usando como conjunto de entrenamiento la base de datos de imágenes del MIT.

Para el entrenamiento de la Máquina de Vectores Soporte utilizamos el programa *svmtrain* de la librería SvmFu 3. Los parámetros de entrada del programa *svmtrain* son: una función kernel con sus respectivos parámetros, la base de datos a utilizar en formato “dense” y el nombre de la máquina resultante. Esta rutina devuelve en el nombre de la máquina resultante el conjunto de vectores soporte, sus respectivos autovalores, el kernel utilizado y el valor de b resultante.

Una vez obtenida la Máquina de Vectores Soporte podemos calcularle el error de validación sobre un conjunto de imágenes pruebas, provenientes de una base de datos de 24045 patrones previamente etiquetados como cara o no-cara, mediante la rutina *svmfutest* que se encuentra en la librería SvmFu 3. Este valor de error nos da la proporción de patrones que resultaron correctamente clasificados por la Máquina de Vectores Soporte entrenada.

4. Bootstrapping

Una vez realizado el entrenamiento y obtenida la superficie de decisión separable, pasamos al paso de *Bootstrapping* que consiste en tomar imágenes que no posean caras y analizarlas con el sistema. Todas aquellas imágenes que resulten clasificadas como “caras”, conocidas como los falsos positivos, serán almacenadas y utilizadas como datos negativos (no-caras) en el próximo

entrenamiento de la máquina. Este paso es muy importante en el proceso de aprendizaje de la máquina debido a la gran cantidad de posibles muestras negativas que existen y la necesidad de poseer la mayoría de ellas como ejemplos negativos para el uso de un clasificador binario óptimo. El proceso de bootstrapping lo elaboramos a través del programa *SDPC_Bootstrapping.cpp* el cual se implementa con el siguiente algoritmo de bootstrapping:

Datos de entrada: imagen estática en formato PGM de cualquier tamaño (*inimg*), una Máquina de Vectores Soporte entrenada, una lista de nombres de archivos salientes, el nombre de la imagen resultante (*outimg*) y el nombre del archivo de datos resultantes. Para *escala* > 0.3, hacer:

```

    Desde i = 0 hasta inimg.ancho
        Desde j = 0 hasta inimg.alto
            Extraer de la imagen ventana de 19x19 pixels con extremo superior izquierdo en (i,j).
            Ajustar la iluminación de la imagen ventana
            Ecualizar ventana
            Enmascar ventana en un vector ventanaMascara
            Clasificar ventanaMascara con la Máquina de Vectores Soporte entrenada:
                 $\phi_i = \text{testSVM.outputAtDataPt}(\text{ventanaMascara})$ 
                Si  $\phi_i > 0$ 
                    Dibujar un cuadrado escalado alrededor de la ventana en la imagen resultante.
                    Almacenar la ventana en formato PGM, según el nombre indicado en la lista de nombres proporcionada.
                    Escribir la ventanaMascara en el archivo de datos resultante en formato “dense”.
                Actualizar el valor de j
            Actualizar el valor de i
            Actualizar el valor de escala.
            Escalar la imagen inimg.
            Escribir la imagen global resultante en formato PGM.
            Reportar el tiempo de ejecución y los resultados de caras y no-caras obtenidos.

```

Figura 3.7: Algoritmo de bootstrapping

El programa retorna el número de caras y no-caras detectadas, el tiempo de ejecución y la imagen original con las caras detectadas enmarcadas con un cuadrado. Cada uno de los patrones detectados como cara son falsos positivos. La Figura 3.8 muestra el proceso de bootstrapping,

algunas de las imágenes utilizadas y sus resultados.

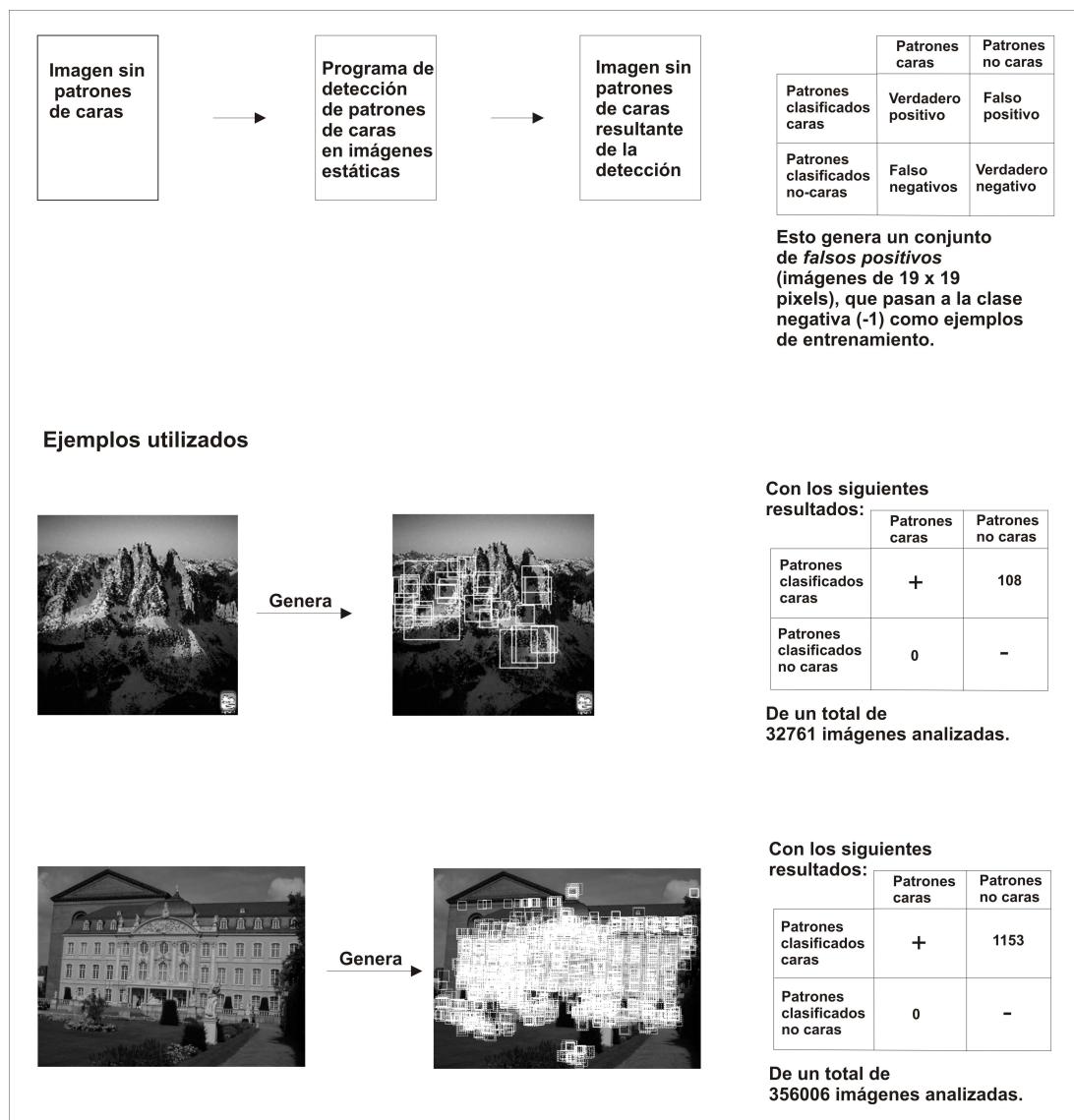


Figura 3.8: Proceso y resultados del Bootstrapping

5. Detector de caras en imágenes estáticas

Una vez completada la fase de entrenamiento utilizando las imágenes del proceso de bootstrapping, usamos el programa *SDPC_DetectorEstatico.cpp* para reconocer y ubicar los patrones de

caras sobre la imagen, el cual opera de la siguiente manera:

Datos de entrada: imagen estática en formato PGM de cualquier tamaño (*inimg*), una Máquina de Vectores Soporte entrenada y el nombre de la imagen resultante.
 Para *escala* > 0.3, hacer:

```

    Desde i = 0 hasta inimg.ancho
      Desde j = 0 hasta inimg.alto
        Extraer la imagen ventana de 19x19 pixels con extremo superior izquierdo en (i,j).
        Ajustar la iluminación de la imagen ventana
        Ecualizar ventana
        Enmascarar ventana en un vector ventanaMascara
        Clasificar ventanaMascara con la Máquina de Vectores Soporte entrenada:
           $\phi_i = \text{testSVM.outputAtDataPt}(\text{ventanaMascara})$ 
          Si  $\phi_i > 0 \rightarrow$  Dibujar un cuadrado escalado alrededor de la ventana en la imagen resultante.
        Actualizar el valor de j
      Actualizar el valor de i
    Actualizar el valor de escala.
    Escalar la imagen inimg.
    Escribir la imagen global resultante en formato PGM.
    Reportar el tiempo de ejecución y los resultados de caras y no-caras obtenidos.
```

Figura 3.9: Algoritmo de detección de imágenes estáticas

El programa retorna el número de caras y no-caras detectadas, el tiempo de ejecución y la imagen original con las caras detectadas enmarcadas con un cuadrado. Este programa fue utilizado para evaluar el rendimiento de la Máquina de Vectores Soporte obtenida, en el Apéndice B están mostrados algunos de estos resultados.

3.3. Procesos a tiempo real

Obtenida la Máquina de Vectores Soporte entrenada en el proceso batch, pasamos a los procesos a tiempo real cuyo objetivo consiste en obtener una secuencia de imágenes de video procesadas a tiempo real a partir de una secuencia de imágenes de video incial capturada por la cámara web.

Dividiremos los procesos a tiempo real en dos partes, primero implementaremos un módulo de detección de color piel y posteriormente implementaremos el de detección de patrones de caras en una secuencia de imágenes de video. Estos procesos vienen ilustrados en la Figura 3.10.

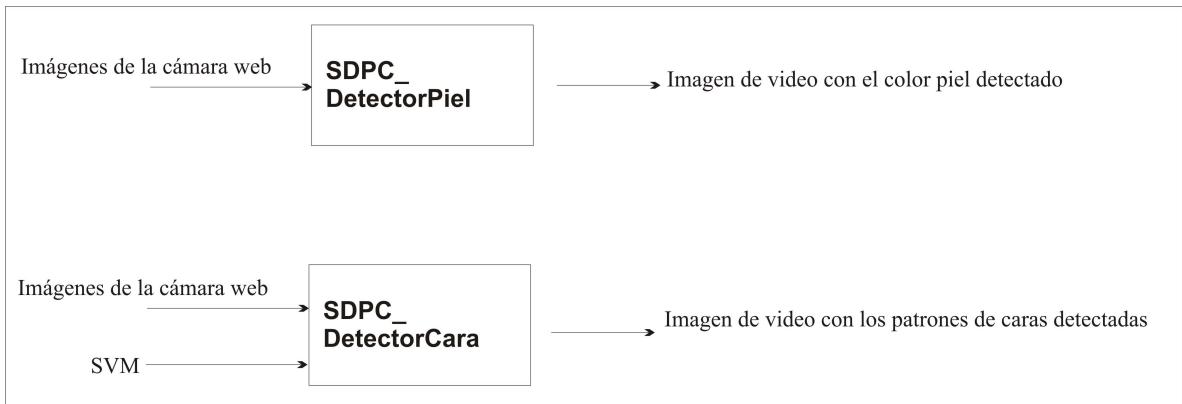


Figura 3.10: Diagrama de los procesos a tiempo real

Analizando el diseño e implementación de ambos módulos por separado tenemos lo siguiente:

1. *SDPC_DetectorPiel.c*

Esta programa online incorpora una cámara web al sistema y utilizando el algoritmo de detección del color piel señalado en el marco teórico, se implementa una aplicación que realiza las siguientes operaciones:

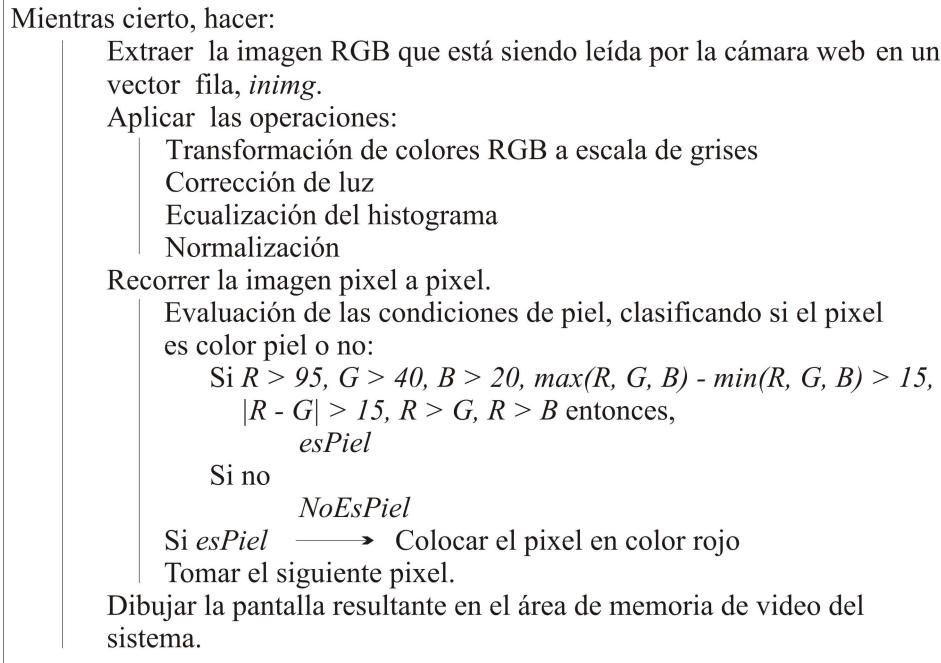


Figura 3.11: Algoritmo del sistema de detección de piel

Como resultado de este programa tenemos una aplicación online en la que podemos detectar el color piel y su movimiento. Dado que el algoritmo de detección del color piel es sumamente rápido, esto permite que el número de frames procesados por segundo sea prácticamente igual al número de frames capturados por la cámara web. Producíendose de esta forma una continuidad en el movimiento de las imágenes. En el Apéndice C se presentan algunos de los frames resultantes de este algoritmo.

2. *SDPC_DetectorCara.c*

Esta aplicación online permite la detección de patrones de caras en imágenes de video, utilizando para ello una Máquina de Vectores Soporte previamente entrenada. Este programa recorrer todos los pixels de la imagen buscando patrones de caras en ventanas de 19×19 pixels.

El proceso que sigue este programa es el siguiente:

Mientras cierto, hacer:

 Extraer la imagen que está siendo leída por la cámara web en un vector fila, *inimg*
 Aplicar las operaciones:

 Transformación de colores RGB a escala de grises
 Corrección de luz
 Normalización.

Mientras escala > 0.3, hacer:

 Recorrer la imagen comenzando por el pixel (i,j) del extremo superior izquierdo y con un vector fila, *ventana*, de 19x19 pixels, avanzando pixel a pixel hacia la derecha y hacia abajo.

 Si el pixel (i,j) es clasificado como color piel.

 Ecualizar *ventana*

 Enmascarar *ventana* en un vector *ventanaMascara*

 Clasificar *ventanaMascara* con la Máquina de Vectores Soporte entrenada:

$\phi = \text{testSVM.outputAtDataPt}(\text{ventanaMascara})$

 Si $\phi > 0 \rightarrow$ Dibujar un cuadrado escalado alrededor de la *ventana* en la imagen resultante.

 Si $\phi > -6$

 Si $\phi < 2 \rightarrow j$ se incrementa en 2.

 Si no $\rightarrow j$ se incrementa en 1.

 Si no

 Si $\phi > -8 \rightarrow j$ se incrementa en 4.

 Si no $\rightarrow j$ se incrementa en 9.

 Tomar el siguiente pixel (i,j).

 Actualizar el valor de *escala*

 Escalar la imagen *inimg*.

 Dibujar el resultado en el área de memoria de video del sistema.

 Reportar el tiempo de ejecución y los resultados de caras y no-caras obtenidos.

Figura 3.12: Algoritmo del sistema de detección de patrones de caras en imágenes de video

De esta manera tenemos un sistema de detección de patrones de caras en una secuencia de imágenes de video mediante el uso de una Máquina de Vectores Soporte.

El módulo *SDPC_DetectorCara.c* fué implementado de manera interactiva en lenguaje C mediante el uso de las librerías SDL. Al ejecutar la aplicación SDPC se presenta un menú de opciones y

una ventana que muestra las imágenes provenientes de la cámara web. El usuario podrá ajustar inicialmente el brillo, contraste y color según la iluminación presente. Después el usuario indicará el comienzo del proceso de detección de patrones de caras presionando el botón “cámara”. El programa captura la imagen presentada por la cámara web y la clasifica de acuerdo a la Máquina de Vectores Soporte previamente entrenada. El programa continua analizando y mostrando las imágenes de la cámara web hasta conseguir una imagen que presente patrones de caras de acuerdo a la SVM. A continuación el programa presenta la imagen obtenida dibujando un rectángulo alrededor de cada cara detectada. El tamaño de los rectángulos dibujados está relacionado con la escala utilizada.

El tiempo de respuesta de cada frame analizado dependerá de la máquina utilizada y de la cantidad de color piel presente en ella. En el Apéndice C se presentan algunos frames resultantes de este sistema.

Capítulo 4

Implementación

Esta aplicación como ya mencionamos anteriormente, implementa un detector de patrones de caras en imágenes estáticas y/o en imágenes provenientes de una secuencia de video utilizando un clasificador binario de Máquinas de Vectores Soporte, basado en el sistema de detección de caras propuesto por Osuna-Freund-Girosi [21]. Dividiremos este capítulo en tres secciones: en la primera señalaremos los requerimientos del sistema operativo para poder ejecutar la aplicación; en la segunda, describiremos las operaciones sobre imágenes y por último en la tercera mostraremos los resultados obtenidos del diseño del sistema presentado en el capítulo anterior.

4.1. Requerimientos del sistema

Para la compilación y ejecución del *Sistema de detección de patrones de caras en imágenes de video* que se está presentando es necesario disponer de un computador con *Sistema Operativo UNIX* y que reúna los siguientes requerimientos:

1. Compilador GCC 3.0
2. Compilador G++ versión mayor o igual que 2.95
3. Driver Spca5xx de la cámara web utilizada que se puede obtener en:

<http://mxhaard.free.fr/download.html>

4. Librería SvmFu 3, disponible en: <http://five-percent-nation.mit.edu/SvmFu/>
5. Una cámara web compatible con el driver Spca5xx.

Se pueden observar las cámaras web compatibles en:

<http://mxhaard.free.fr/spca5xx.html>.

4.2. Operaciones sobre las imágenes

Los objetos más relevantes de esta aplicación son las *imágenes*. Por ello es importante entender y conocer las operaciones que se pueden realizar sobre las mismas de manera tal que su manejo sea óptimo.

Existen diversas operaciones a realizar sobre las imágenes, así que trataremos aquellas relevantes para la aplicación que se está presentando. Estas operaciones son:

1. **Máscara:** esta operación es utilizada para eliminar algunos pixels no relevantes de la imagen, manteniendo visible el área de los ojos, la nariz y la boca. Para ello utilizamos una máscara binaria de 19×19 pixels, y se la aplicamos a una imagen de igual dimensión $n = 19 \times 19 = 361$ pixels que viene representada como un vector $\mathbf{x} \in \mathbb{R}^n$, generando así la imagen enmascarada. Esto nos permite realizar una reducción de la dimensión del espacio de entrada, de $19 \times 19 = 361$ a 265 pixels, eliminando así el ruido innecesario de la imagen y reduciendo significativamente la dimensión de los vectores de entrenamiento. La máscara utilizada en esta aplicación se basa en la máscara descrita por Sung-Poggio [29], y se muestra en la Figura 4.1.



Figura 4.1: Ejemplos de aplicación de la operación: Máscara

2. **Ecuallización del Histograma:** esta operación es aplicada para eliminar las variaciones de brillo y contraste que presenta la imagen. Este procedimiento parte de una imagen en escala de grises representada con un vector $\mathbf{x} \in \mathbb{R}^n$, $n = \text{alto} * \text{ancho}$. Para cada grado de gris i , con i entre 0 y 255, se calcula el número de ocurrencia de i ($\text{ocurrencia}[i]$) entre las componentes del vector \mathbf{x} que representa la imagen. Luego calculamos la suma de los elementos de todo el histograma, $\text{hist}[i] = \sum_{k=1}^i \text{ocurrencia}[k]$ y reemplazamos las intensidades viejas por las intensidades nuevas, mediante la relación:

$$x[i] = \left(\frac{1}{\text{alto} * \text{ancho}} \right) \frac{\text{hist}[i] - m}{M - m}, \quad i = 1, \dots, n.$$

Donde, M representa el máximo valor del histograma y la m el mínimo.

3. **Corrección del Gradiente de iluminación:** en esta operación el mejor plano de ajuste del brillo es substraído de la imagen desenmascarada de manera tal que las luces y las sombras fuertes sean reducidas. Esto ocurre calculando el promedio prom de las intensidades de los pixels de la imagen para poder obtener la constante de ajuste $\text{ctte} = 128 - \text{prom}$. Posteriormente se aplica la corrección incrementando ctte a cada intensidad.
4. **Transformación de colores RGB a escala de grises:** esta operación toma una imagen $I = (R, G, B)$ y la transforma en una imagen E en escala de grises, según la función de transformación geométrica presentada por David W. Fanning en [11], que viene dada por:

$$E = 0,3 R + 0,59 G + 0,11 B$$

5. **Espejo:** esta operación toma una imagen y genera su imagen espejo, como se muestra en la Figura 4.2. Esta operación nos permite generar ejemplos de entrenamiento virtuales a partir de patrones de entrenamientos conocidos.
6. **Rotación:** esta operación toma una imagen y la rota un ángulo β tomando como centro de rotación el centro de la imagen tal como se muestra en la Figura 4.2. Esta operación también nos permite generar ejemplos de entrenamiento virtuales.

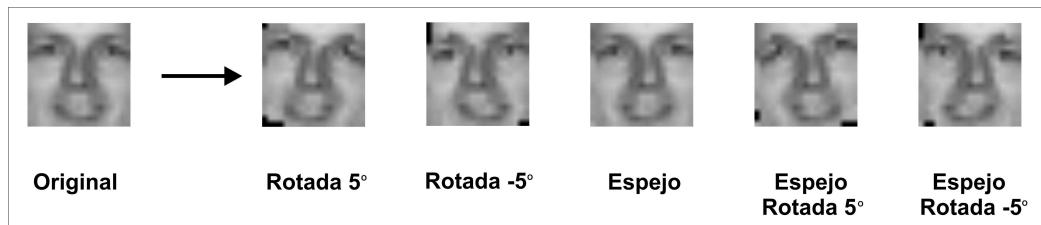


Figura 4.2: Generación de patrones o ejemplos virtuales

7. **Normalización:** esta operación lleva los valores de la imagen en escala de grises (de 0 a 255) a un rango de 0 a 1.
8. **Escalamiento:** esta operación toma una imagen inicial y la reduce según un factor de escalamiento prefijado. Ver Figura 4.3.

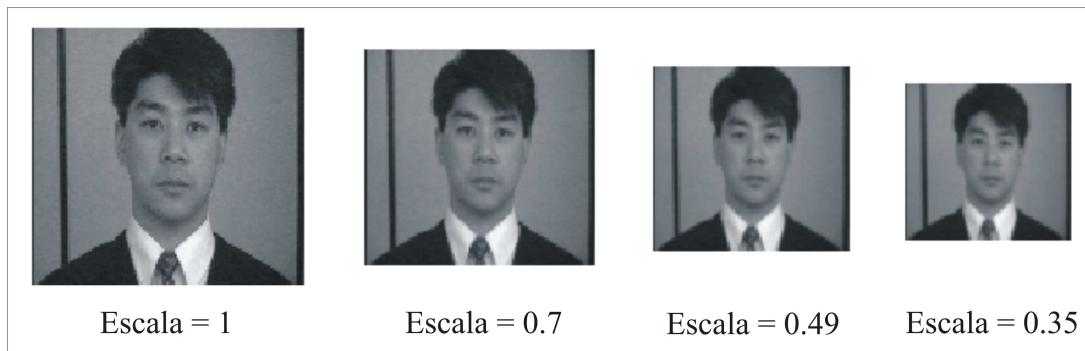


Figura 4.3: Ejemplos de aplicación de la operación: Escala

4.3. Resultados obtenidos

A continuación presentamos los resultados obtenidos en las diferentes pruebas realizadas sobre una Máquina de Vectores Soporte construida a partir del conjunto básico de caras y no-caras del MIT [13]. Dichas pruebas fueron realizadas tanto sobre imágenes estáticas como sobre imágenes de video capturadas en tiempo real. En ellas identificamos el número de caras y no-caras detectadas por la SVM construida y también por la SVM modificada con el método de reducción de Burges (Burges 1996 [6]), que para nuestro caso, por ser el kernel polinomial de grado 2, se producen los mismos resultados con una reducción sustancial del tiempo de procesamiento.

Utilizando los modulos SDPC desarrollados especialmente para esta investigación, se obtienen los siguientes resultados, tanto en la obtención de la SVM óptima como en optimizaciones del tiempo de procesamiento de las imágenes:

1. El conjunto de entrenamiento básico está constituido por la base de datos inicial de 2429 patrones de caras y 4548 patrones de no-caras. El conjunto de prueba o test esta compuesto por 24045 imágenes de 19×19 pixels previamente clasificadas. Estas bases de datos junto con ejemplos virtuales e imágenes provenientes del proceso de bootstrapping, nos permitieron entrenar una Máquina de Vectores Soporte con kernel polinomial de grado 2 y con las siguientes características:

Tabla 1: Características de la Máquina de Vectores Soporte

SVM	Caras	No-caras	Boots	Virtuales	c	Precisión	VS
1	2434	6091	Si	No	1	0.9796	1035
2	2434	6091	Si	No	50	0.9799	1136
3	2434	6091	Si	No	200	0.9695	1147

En la tabla anterior la columna Boots representa si la máquina se entrenó con bootstrapping, la columna Virtuales representa si en la base de datos a entrenar se incluyeron los ejemplos virtuales de espejos y rotaciones. La columna Precisión representa el porcentaje de imágenes clasificadas correctamente y por último la columna VS representa el número de vectores soporte de la máquina entrenada.

El parámetro c del kernel se determina minimizando el error de validación ($1 - \text{Precisión}$) para un mismo conjunto de entrenamiento y en diferentes máquinas con el mismo tipo de kernel. Un valor muy grande de c produce un sobreajuste de la data y un valor muy pequeño aumenta el error de entrenamiento. Para los datos de la tabla anterior, tomamos $c = 50$. De manera que todos los resultados que se mencionan a continuación van a hacer referencia a la SVM 2 de la tabla anterior.

2. El proceso de Bootstrapping aplicado a la Máquina de Vectores Soporte señalada anteriormente, produjo una base de datos muestrales constituida por un total de 8525 imágenes, de las cuales 2434 son patrones positivos y 6091 son patrones negativos. La Figura 3.8 muestra el proceso de bootstrapping, identificando las cantidades de falsos positivos y falsos negativos obtenidos para las imágenes allí presentadas. Las ventanas de 19×19 pixels de falsos positivos se incorporaron a la base de datos principal aumentando el número de patrones negativos en la cantidad señalada.

3. La curva ROC nos permite verificar la bondad del clasificador en cuanto a minimizar los falsos positivos y a maximizar los verdaderos positivos. Para elaborar esta curva construimos la siguiente tabla obtenida de procesar con la Máquina de Vectores Soporte un conjunto de 8 imágenes

estáticas. Las columnas de esta tabla representan: el número de la imagen, la dimensión de la imagen, el porcentaje de falsos positivos FP (no-caras clasificadas como caras), el porcentaje de verdaderos positivos TP (verdaderas caras clasificadas como caras), el tiempo en segundos que tardó el programa en procesar la imagen y el tiempo utilizando la reducción de Burges sobre la SVM. En el Apendice B se presentan algunas de estas imágenes.

Tabla 2: Diferentes imágenes analizadas para detectar patrones de caras

Imagen	Dimensión	FP	TP	Tiempo (seg)	Tiempo Burges (seg)
1	239x255	0.04	94	51	12
2	1280x960	0.06	87	1287	389
3	182x257	0.021	86	41	9
4	282x289	0.0	66	72	16
5	640x438	0.062	86	311	73
6	300x220	0.022	83	58	14
7	694x1024	0.17	68	807	197
8	450x295	0.038	75	1905	472

La Figura 4.4 ilustra una curva ROC de los datos anteriores, observándose que esta curva está muy por encima de la recta de diagonal del cuadrado, de tal forma que la Máquina de Vectores Soporte utilizada es un buen clasificador.

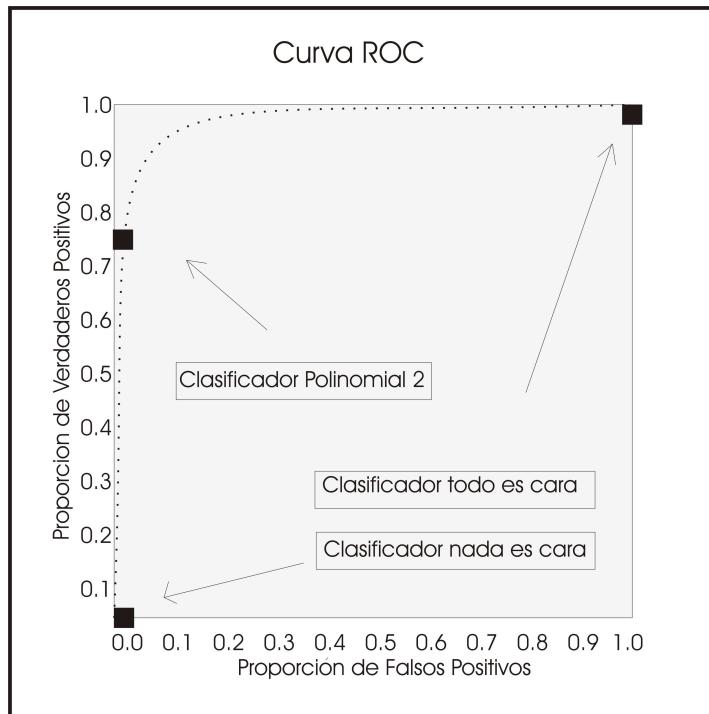


Figura 4.4: Curva ROC

4. El algoritmo de detección del color piel utilizado produjo una mejora en el tiempo de respuesta en la detección de patrones de caras en el sistema online de video, reduciendo significativamente la cantidad de imágenes de 19×19 pixels a ser clasificadas por la librería SvmFu 3. En promedio el tiempo de respuesta del sistema online entre imagen e imagen es de aproximadamente 25 segundos usando la SVM original y de 6 segundos utilizando la SVM reducida por el método de Burges. Este es el tiempo que tarda la librería en analizar las imágenes a clasificar.

La distancia entre el hiperplano óptimo separable y la imagen a clasificar, Φ , la usamos para disminuir el tiempo de respuesta, mediante saltos en las columnas de la imagen de 9, 4, 2, 1 dependiendo de la distancia Φ . Así por ejemplo si un punto es rechazado con un Φ muy negativo, es poco probable que los pixels siguientes puedan tener un patrón de cara, en esos casos ocurre

un salto de 9 en las columnas evitando tener que clasificar los puntos intermedios.

La siguiente tabla compara el tiempo de procesamiento de seis imágenes de 320x240 pixels tomadas en diferentes momentos y en forma online directamente desde la cámara de video. La tabla pone en evidencia la importancia del algoritmo de detección de piel y del método de reducción de Burges en cuanto al tiempo de procesamiento.

Tabla 3: Tiempos de detección con el clasificador de color piel

Imagen	Sin piel (seg)	Con piel (seg)	Con piel y Burges (seg)
1	61	11	3
2	53	5	1
3	57	14	3
4	70	17	4
5	68	16	3
6	61	11	2

5. Para $N_z < 265$, el método de Burges produce un conjunto de vectores z_1, \dots, z_{N_z} , y constantes $\gamma_1, \dots, \gamma_{N_z}$ que definen un clasificador aproximado. Sea SVM el clasificador original de la máquina, sea SVM-B el clasificador exacto de Burges (kernel polinomial de segundo grado) y sea SVM-Baprox el clasificador aproximado de Burges utilizando N_z vectores.

La tabla siguiente nos muestra la cantidad resultados de positivos obtenidos en tres imágenes de pruebas, las cuales fueron clasificadas utilizando las SVM-Baprox con valores de N_z desde 30 a 120. La fila correspondiente a $N_z = 265$ corresponde al clasificador SVM-B, con el cual se

obtienen los mismos resultados que con el clasificador SVM debido a que estamos utilizando un kernel polinomial de grado 2.

Tabla 4: Comparación de SVM-Baprox sobre el número de caras detectadas

N_z	Imagen 1	Imagen 2	Imagen 3
30	55	340	148
40	36	136	97
50	39	123	84
60	32	95	72
70	33	104	73
80	30	91	57
90	27	90	58
100	29	91	59
110	28	91	55
120	27	90	56
265	26	88	54

Como observamos el número de caras detectadas va convergiendo al número de caras detectadas por el clasificador original, en la medida en que N_z va creciendo en valores. A partir de $N_z = 120$, en nuestros ejemplos, la diferencia entre los clasificadores se hace imperceptible.

Otra comparación que puede establecerse, es medir la proporción de clasificaciones diferentes obtenidas entre un clasificador SVM-Baprox y el clasificador original, SVM-B. Para ello consideremos la base de datos de pruebas constituida por los 24045 patrones de imágenes. Para distintos valores de N_z calculemos h como el número de imágenes de la base de datos de pruebas clasificadas en forma diferente por las SVM y SVM-Baprox.

Calculamos la proporción p como:

$$p = \frac{h}{24045}$$

La Figura 4.5 muestra los resultados de p obtenidos para diferentes valores de N_z , observándose que si permitimos una tolerancia de error del 5 % entonces podríamos seleccionar $N_z = 130$ aproximadamente, con este valor de N_z definimos un nuevo clasificador que permitirá una reducción del tiempo necesario para clasificar un patrón.

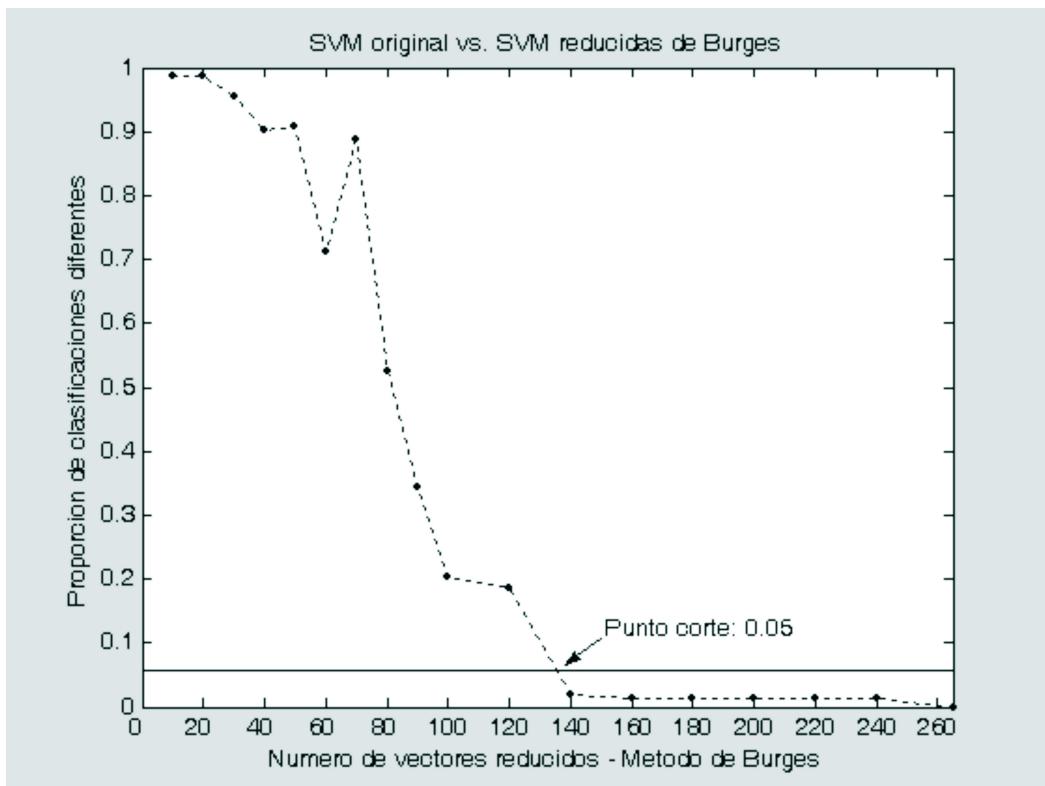


Figura 4.5: SVM vs. SVM-Baprox

La siguiente tabla resume el análisis de los tiempos promedios de clasificación de una imagen de video de 320x240 pixels extraída de la cámara web:

Tabla 5: Análisis del tiempo de clasificación de una imagen de 320x240

Clasificador	Tiempo (seg)
Solo SVM	65
Algoritmo piel + SVM	25
Algoritmo piel + SVM-B	6
Algoritmo piel + SVM-Bapprox ($N_z = 130$)	3

Capítulo 5

Conclusiones

En este trabajo hemos logrado evidenciar las potencialidades del uso de la metodología de Máquinas de Vectores Soporte para la detección de patrones de caras en imágenes de video. Nuestra implementación fué realizada en una arquitectura de bajo costo, se utilizó una cámara web Genius y una computadora Laptop Toshiba con un procesador Intel Celeron 1.6, logrando en tiempo real analizar imágenes obtenidas por dicha cámara, clasificando en cada una de ellas la posible existencia de patrones de caras mediante una Máquina de Vectores Soporte especialmente entrenada para esta aplicación.

También en nuestro trabajo logramos localizar mediante el algoritmo de detección de color piel, la presencia y movimiento del color piel en una secuencia de video totalmente dinámica tal como describimos previamente.

Una dificultad que se puede señalar, en la implementación del sistema online, es el tiempo que demora el programa en procesar una imagen hasta ser clasificada en su totalidad con la Máquina de Vectores Soporte para la detección de patrones de caras. Aplicando el algoritmo de detección de color piel y los saltos Φ descritos anteriormente, obtuvimos que el tiempo promedio para la clasificación de una imagen es de aproximadamente de 6 segundos, dependiendo de la cantidad de pixeles con color

piel detectados. Estos segundos son utilizados en su totalidad por la librería SvmFu 3 en el cálculo de los productos internos necesarios para la clasificación de una imagen.

En nuestro desarrollo, la modificación de las imágenes es el resultado de la clasificación positiva por una SVM, o bien por una clasificación positiva por medio de un algoritmo diseñado para un fin específico, como el algoritmo de detección del color piel.

Este proyecto ha comprobado la factibilidad de implementar sistemas de detección en tiempo real móviles. Esto abre un gran potencial de aplicaciones en tiempo real, tales como identificación de personas para aplicaciones policiales, militares, bioinformática y domóticas.

Queda abierto el problema de corregir los efectos de iluminación sobre los patrones de caras en el espacio tridimensional, tal como propone Pontil-Verril [23]. Mediante la formulación de un nuevo modelo de patrones detectados y tomando a partir de ellos nuevas imágenes con iluminación controlada, podríamos generar nuevos vectores a ser utilizados en el entrenamiento de una SVM.

Apéndice A

Plataforma requerida

A.1. Librería SvmFu 3

La librería SvmFu 3 versión 3.1 se puede obtener en: <http://fpn.mit.edu/SvmFu/SvmFu-3.1.tar.gz>.

Una vez obtenida la librería, se descomprime en una carpeta vacía y desde el terminal se ejecutan los siguientes comandos:

```
./configure
```

```
make
```

El segundo comando hay que ejecutarlo desde el terminal del administrador, para que la instalación sea exitosa. Adicionalmente necesitamos el compilador *GNU g++* para compilar la librería SvmFu 3. La versión de *GNU g++* que utilizamos en esta aplicación es la 2.95 y la versión del *GNU gcc* 3.0.

A.2. Driver Spca5xx

El driver *Spca5xx* disponible en <http://mxhaard.free.fr/index.html> permite el manejo de cámaras web sobre sistemas operativos UNIX. Este driver se instala de acuerdo a las siguientes instrucciones:

```
sudo apt-get install linux-headers-`uname -r` linux-restricted-modules-`uname -r`
```

```
build-essential gcc-3.4
```

```
wget http://mxhaard.free.fr/spca50x/Download/spca5xx-20060501.tar.gz  
tar xvfvz spca5xx-20060501.tar.gz  
cd spca5xx-20060501  
sudo make CC=gcc-3.4  
sudo modprobe -r spca5xx  
sudo rm -rf /lib/modules/`uname -r`/kernel/drivers/usb/media/spca5xx*  
sudo make install  
sudo modprobe spca5xx
```

Las instrucciones anteriores se obtuvieron del foro de Ubuntu, en la dirección web:

<http://www.ubuntuforums.org/showthread.php?t=75284>

Apéndice B

Detección de Patrones en Imágenes Estáticas: resultados



Figura B.1: Detección de Patrones en Imágenes Estáticas: Imagen 1

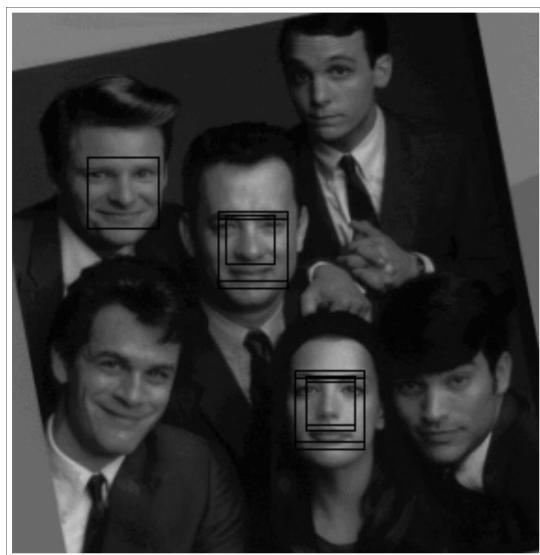


Figura B.2: Detección de Patrones en Imágenes Estáticas: Imagen 2

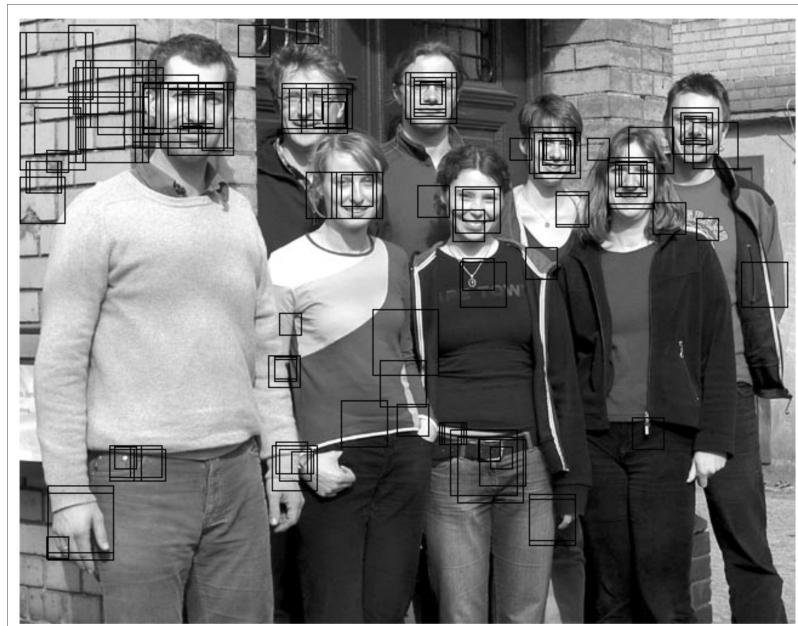


Figura B.3: Detección de Patrones en Imágenes Estáticas: Imagen 3

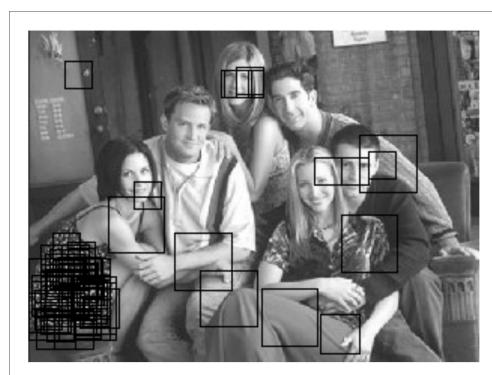


Figura B.4: Detección de Patrones en Imágenes Estáticas: Imagen 4



Figura B.5: Detección de Patrones en Imágenes Estáticas: Imagen 5

Apéndice C

Resultados de los procesos a tiempo real

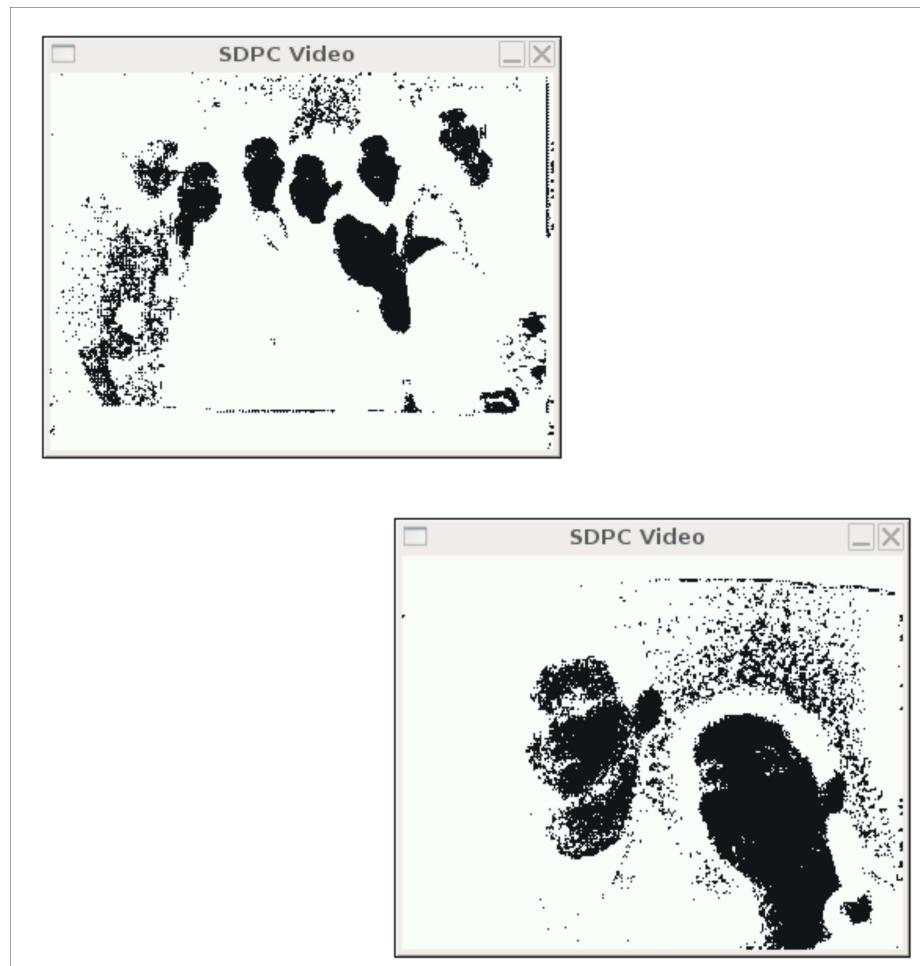


Figura C.1: Detección del color piel en imágenes dinámicas

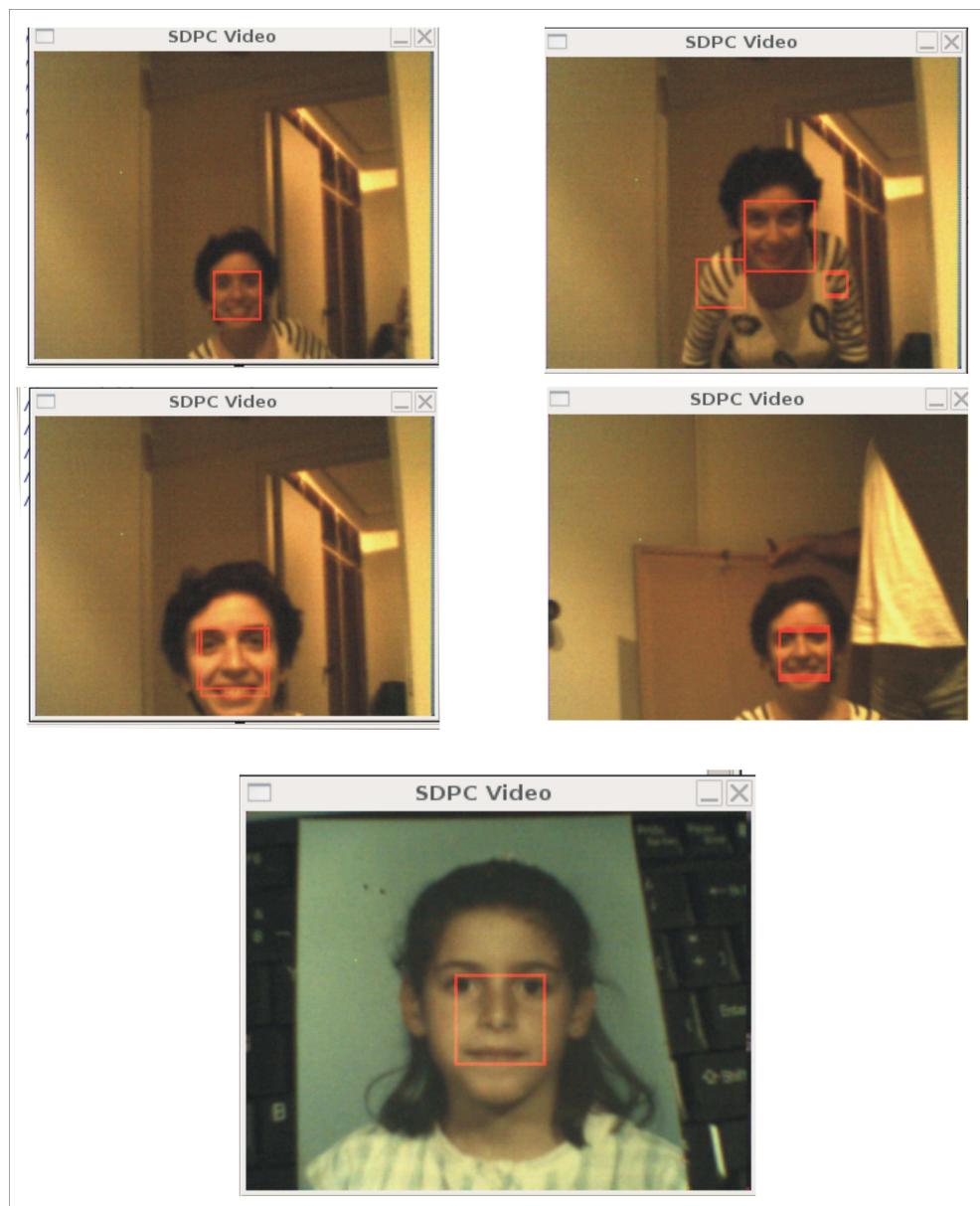


Figura C.2: Detección de patrones de caras en imágenes dinámicas

Bibliografía

- [1] M. Bazaraa and C.M. Shetty. *Nonlinear programming*. John Wiley, New York, 1979.
- [2] S.M. Bileschi and B. Heisele. Advances in component-based face detection. in: Proceedings of pattern recognition with support vector machines. *First International Workshop, SVM 2002, Niagara Falls, Canada, S.-W. Lee and A. Verri (eds.), Lecture Notes in Computer Science, Springer LNCS 2388*, pages 135–143, 2002.
- [3] V. Blanz, B. Schölkopf, H. Bulthoff, C. Burges, V.N. Vapnik, and T. Vetter. Comparation of view-based object recognition algorithms using realistic 3d models. Proc of ICANN'96, *LNCS*, 1112:251–256, 1996.
- [4] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Computational Learing Theory*, pages 144–152, 1992.
- [5] Chris J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 375. The MIT Press, 1997.
- [6] Christopher J. C. Burges. Simplified support vector decision rules. In *International Conference on Machine Learning*, pages 71–77, 1996.
- [7] Novar Technology Centers. Novar airport technologies - integrated airport control, biometric identification and automated dicking systems, 2006.

-
- [8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
 - [9] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines (and other kernel-based learning methods)*. Cambridge University Press, 2000.
 - [10] Real Academia Española. Aprendizaje, 2006.
 - [11] David W. Fanning. Convert rgb image to greyscale, 2002.
 - [12] R. Fletcher. *Practical Methods of Optimization*. Chichester: John Wiley Sons, Inc, 1987.
 - [13] Center for Biological and Computational Learning at MIT. Face data, 2000.
 - [14] B. Heisele, P. Ho, and Tomaso Poggio. Face recognition with support vector machines: Global versus component-based approach. *International Conference on Computer Vision (ICCV'01), Vancouver, Canada*, 2:688–694, 2001.
 - [15] B. Heisele, P. Ho, J. Wu, and T. Poggio. Face recognition: component-based versus global approaches, 2003.
 - [16] J. Huang, J. V. Blanz, and B. Heisele. Face recognition using component-based svm classification and morphable models. in: Proceedings of pattern recognition with support vector machines. *First International Workshop, SVM 2002, Niagara Falls, Canada, S.-W. Lee and A. Verri (eds.), Lecture Notes in Computer Science, Springer 2388*, pages 334–341, 2002.
 - [17] Jennifer Huang and Volker Blanz. Face recognition with support vector machines and 3D head models, January 02 2003.
 - [18] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical Report 23, Universität Dortmund, 1997.

- [19] Wikipedia: La Enciclopedia Libre. Lectura de patrones, 2006.
- [20] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [21] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection, 1997.
- [22] P. Peer, J. Kovac, and F. Solina. Human skin colour clustering for face detection, 2003.
- [23] Massimiliano Pontil and Alessandro Verri. Support vector machines for 3d object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(6):637–646, 1998.
- [24] Ryan Rifkin. Svmfu 3, 2000.
- [25] M. Schmidt. Identifying speaker whit support vector networks, 1996.
- [26] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task, 1995.
- [27] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines, 1996.
- [28] Bernhard Schölkopf. Svm and kernel methods. *Tutorial given at the NIPS Conference. Max-Planck-Institut für biologische Kybernetik. 72076 Tübingen, Germany*, 2001.
- [29] Kah Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51, 1998.
- [30] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.
- [31] Vladimir Vapnik. The nature of statistical learning theory, 1995.