Detección de manipulación en vídeos con formato MPEG

Proyecto final de Máster

Autora: Beatriz López Ruiz

Director: Javier Pórtela García-Miguel

Máster en Minería de Datos e Inteligencia de Negocios, 2016-2017
Universidad Complutense de Madrid
Facultad de Estudios Estadísticos

Agradecimientos

Este Trabajo Fin de Máster ha sido posible gracias al apoyo y ayuda de muchas personas a las que me gustaría dar mi agradecimiento.

En primer lugar, he de dar las gracias a todos y cada uno de mis profesores de este máster que, sin duda alguna, me han hecho crecer intelectualmente y, aún más, me han mostrado la infinita cantidad de saberes que puedo aprender todavía, para seguir creciendo, aportando lo que pueda y, ojalá, ayudando mínimamente a crecer a otros.

A mis padres, por su apoyo y presencia incondicional, tanto en buenos como en malos momentos. Por confiar en mí en todo momento, y darme el empujón que a veces me faltaba.

A Alberto, porque estás ahí, a mi lado, porque me alegro de compartir mi vida contigo y porque todos los días aprendo algo nuevo gracias a ti.

A mi tutor, Javier Pórtela, por confiar en mí, y darme la oportunidad de desarrollarme en este tema tan apasionante, y poder continuar mi investigación.

Tabla de contenido

Índice de ilustraciones

Índice de tablas

1. Introducción

En la última década las cámaras digitales se han vuelto tan populares que gran cantidad de las fotografías y vídeos son tomados por fotógrafos aficionados.

Hoy en día debido al progreso y desarrollo de técnicas de edición de vídeos, que se pueden utilizar, es difícil garantizar la autenticidad de un vídeo. Lamentablemente, los vídeos tomados por aficionados no están protegidos de la manipulación. Así que, si dichos vídeos se utilizan como una prueba en un tribunal de justicia, ¿Cómo podemos distinguir la verdadera evidencia de la falsa?

En el pasado, la marca de agua digital fue la contramedida principal contra el uso ilegal de contenidos digitales [1]. Sin embargo, la mayoría de fotos y vídeos no tenían marcas de agua incorporadas, por lo que se subían a internet y las marcas de agua podían ser incorporadas posteriormente. Esto hacía que dichas marcas no fueran eficaces ya que, incluso estando incrustada después no podríamos saber si el contenido ha sido manipulado o no. Por lo tanto, la marca de agua digital se encontraba limitada en su capacidad de asegurar la autenticidad.

Si nos centramos únicamente en la falsificación de vídeos, el software debe primero decodificar los vídeos comprimidos para después en el dominio sin comprimir ser re-codificado y guardado en formato comprimido. Por lo tanto, los artefactos de doble compresión, pueden revelar una alteración.

2. Estado del arte

En respuesta a las limitaciones de la marca de agua, hubo numerosas técnicas de detección de falsificaciones. Las primeras se enfocan en técnicas pasivas más prácticas, ya que utilizaban las características intrínsecas del vídeo en vez de los datos. Como, por ejemplo, las inconsistencias en la iluminación y la aberración cromática de Johnson [2] y

[1] Lee, S.-J., Jung, S.-H.: A survey of watermarking techniques applied to multimedia. In: Proc. of IEEE International Symposium on Industrial Electronics, vol. 1, pp. 272–277 (2001).

[2] MK Johnson y H. Farid, "Exposing digital. Falsificaciones mediante la detección de inconsistencias en la iluminación ", en *Proc. Taller sobre Multimedia y Seguridad*, 2005, pp.1-10.

Farid[3]. Sin embargo, las técnicas anteriores nos ayudan a estimar agresivamente las imágenes digitales. Por lo tanto, todo el objetivo de la detección de falsificaciones de vídeo se centraba en las características sin tener en cuenta la variación de los datos.

A continuación, Wang y Farid[4] realizaron un estudio sobre la duplicación, ya que esta produce una alta correlación entre los marcos originales o regiones y clonados. En el estudio demostraban que podemos encontrar sustituciones de otro marco en la misma secuencia de vídeo. Sin embargo, su método propuesto tiene una seria limitación, debido a que sólo puede detectar; copiar y pegar la adulteración de la misma secuencia de vídeo, no se puede utilizar, por ejemplo, para saber si hay superposición causada por inserción de objetos.

Actualmente, hay diversos resultados alentadores para el campo de doble detección de compresión en imágenes, como, por ejemplo, T. Pevny and J. Fridrich [5] o C.-H. Chen, Y.-Q. Shi, y W. Su [6]. Sin embargo, por el momento, hay muy poco trabajo para detectar la doble compresión en vídeo.

3. Ámbito y objetivo

3.1. Dominio y alcance

El objetivo principal de este trabajo es detectar la doble compresión de un vídeo en formato MPEG-1, y con ello la manipulación de dicho vídeo.

Los objetivos concretos son los siguientes:

- Entender el proceso mediante el cual se comprime un vídeo.
- Conocer cómo se produce la doble compresión de un vídeo.

[3] M. Johnson y H. Farid, "Exponiendo falsificaciones digitales a través de la aberración cromática", en Proc. *Int. Multimedia Conf.*, 2006, páginas 48-55.

[4] Wang, W., Farid, H.: Exposing digital forgeries in video by detecting duplication. In: Proc. of Workshop on Multimedia & security in International Multimedia Conference, pp. 35–42 (2007).

[5] T. Pevny and J. Fridrich, “Detection of double-compression in JPEG images for applications in steganography,” IEEE Trans. Inf. Forensics Secur., vol. 3, no. 2, pp. 247–258, jun. 2008.

[6] C.-H. Chen, Y.-Q. Shi, and W. Su, “A machine learning based scheme for double JPEG compression detection,” en Proc. Int. Conf. Pattern Recognit. (ICPR), Dec. 2008, pp. 1814–1817.

- Tener el background suficiente para obtener la matriz de Cuantización de cada frame.
- Aplicar un algoritmo predictivo para mejorar la detección de la manipulación.

3.2. Hipótesis de trabajo

Vamos a utilizar el estándar MPEG ya que es el más conocido, en concreto MPEG-1. Las secuencias de vídeo utilizadas tendrán una resolución en píxeles de 256x256 (ancho x alto). Vamos a comprimir y realizar la doble compresión usando VBR (Variable Bit Rate) y CBR (Constant Bit Rate). Los coeficientes DCT que vamos a obtener serán calculados con el DCT-1, para después obtener la matriz de Cuantización. La tabla estándar de Cuantización que usaremos será JPG. Durante todo el trabajo nos centraremos en la modificación de las imágenes del vídeo, no tendremos en cuenta el audio.

4. Desarrollo del trabajo y principales resultados

4.1. Métodos de manipulación de vídeos

Dentro de la falsificación de vídeos algunas de las técnicas más comunes son:

1) **Falsificación del contenido**, es decir, copiar y pegar regiones modificando de esta manera una escena, o crear una escena a partir de dos imágenes.
2) **Cambiar la marca de agua**, quitándole la información y volviendo a introducirla en el vídeo.
3) **Modificación intravídeo**; reemplazar regiones o marcos con duplicados del mismo vídeo. Para ocultar objetos desfavorables en una escena mediante la sobreescritura de otros segmentos del mismo vídeo.
4) **Modificación intervídeo**; quitar partes del vídeo, como objetos o personas.

4.2. Fase de compresión de un vídeo

Las técnicas de compresión de vídeo consisten en reducir y eliminar datos redundantes del vídeo para que el archivo se pueda enviar a través de la red o almacenar en discos informáticos, manteniendo una buena calidad. Hay dos tipos de compresión de vídeo:

- **Compresión sin pérdidas:** se refiere a los métodos de compresión en los que la calidad de la señal decodificada es, al menos, igual a la calidad de la señal de la fuente.

- **Compresión con pérdidas:** En la práctica del manejo de imágenes y sonido, son relativamente pocas las aplicaciones en que es necesaria una reconstrucción absolutamente fiel de la información de la fuente ya que intervienen aspectos perceptuales que son aprovechables para reducir la cantidad de información que debe transmitirse y que, se admite, puede ser reconstruida aproximadamente por el decodificador. Por consecuencia, en los sistemas de compresión con pérdidas es admisible la pérdida de cierta cantidad de información que no es relevante para el observador final. En estas condiciones, se pretende que la señal tenga la máxima calidad subjetiva, es decir que resulte aceptable al observador.
  El proceso de compresión con pérdidas es irreversible, es decir que no es posible recuperar la información original a partir de la información comprimida. Tal irreversibilidad da lugar a una rápida degradación de la calidad de señal si la compresión se aplica de forma concatenada, es decir, la realización de compresiones sucesivas sobre señales decodificadas.

Para entender el proceso de compresión es importante conocer las diferentes redundancias presentes entre los parámetros de una señal de vídeo:

- **Espacial**: La redundancia espacial ocurre cuando los pixeles cercanos tienen un grado de correlación, es decir, cuando son muy parecidos. Por ejemplo, en una imagen con un prado y un cielo azul, veremos cómo los píxeles del prado serán muy parecidos entre ellos.

- **Temporal**: La redundancia temporal ocurre cuando los píxeles en cuadros consecutivos de una señal también están correlacionados, es decir si la señal de vídeo fuera un recorrido por el prado, entre una imagen y la siguiente habría una alta correlación.

- **Psicovisual**: La redundancia psicovisual consiste en que el ojo no trata toda la información visual con igual sensibilidad, por ejemplo, el ojo es más sensible a cambios en la luminancia que en la crominancia.
  La **luminancia** contiene toda la información relacionada con la mayor o menor luminosidad de la imagen y no contiene ninguna información acerca de los colores de la misma. Reproduce, por lo tanto, la imagen en blanco y negro en todas sus tonalidades de grises intermedios.
  La **crominancia** contiene todo lo relacionado con el color de los objetos, separada en los tres colores básicos. Las señales de crominancia se obtienen de las llamadas señales diferencia de color: rojo menos luminancia (R-Y), azul menos luminancia (B-Y) y verde menos luminancia (G-Y). De estas tres señales sólo necesitamos dos, al poder obtener la otra mediante la combinación de las anteriores.

- **Codificación**: Consiste en que no todos los parámetros ocurren con la misma probabilidad en una imagen. Con lo que no todos necesitarán el mismo número de bits para codificarlos.

A día de hoy, la mayoría de los proveedores de vídeo utilizan técnicas estándar de compresión para sus vídeos, esto asegura la compatibilidad e interoperabilidad. Los tres estándares más utilizados son: JPEG, para fotos y MPEG, y H264, para vídeo. Este último es el estándar que se utiliza actualmente para la compresión de vídeo.

El proceso de compresión consiste en aplicar un algoritmo (H264 o MPEG-4) al vídeo original para crear un archivo comprimido y listo para ser transmitido o guardado. Los diferentes estándares de compresión utilizan métodos distintos para reducir los datos y, en consecuencia, los resultados en cuanto a frecuencia de bits y latencia son diferentes. Existen dos tipos de algoritmos de compresión: intrafotograma y interfotograma.

4.2.1. Compresión de imágenes: Intrafotograma

La compresión de intrafotograma es aquella que utiliza una técnica de codificación que consiste en explotar la redundancia espacial que existe entre una imagen y otra mediante un análisis frecuencial. Los datos se reducen a un fotograma [7] de imagen con el fin de eliminar la información innecesaria que puede ser imperceptible para el ojo humano. JPEG es un ejemplo de este tipo de estándar de compresión. En una secuencia JPEG, las imágenes se codifican o comprimen como imágenes JPEG individuales.

Ilustración I: Ejemplo JPEG

En este ejemplo, las tres imágenes de la secuencia se codifican y se envían como imágenes únicas y separadas, sin que dependan unas de otras.

4.2.2. Comprensión de vídeo: Interfotograma

Los algoritmos de compresión de vídeo como el MPEG-4 y el H264 utilizan la predicción interfotograma para reducir los datos de vídeo entre las series de fotogramas. Ésta consiste en técnicas como la codificación diferencial, en la que un fotograma se compara con un fotograma de referencia y sólo se codifican los píxeles que han cambiado con respecto al fotograma de referencia. De esta forma, se reduce el número de valores de píxeles codificados y enviados. Cuando se visualiza una secuencia codificada de este modo, las imágenes aparecen como en la secuencia de vídeo original.

[7] RAE: Un fotograma es cada una de las imágenes que se suceden en una película.

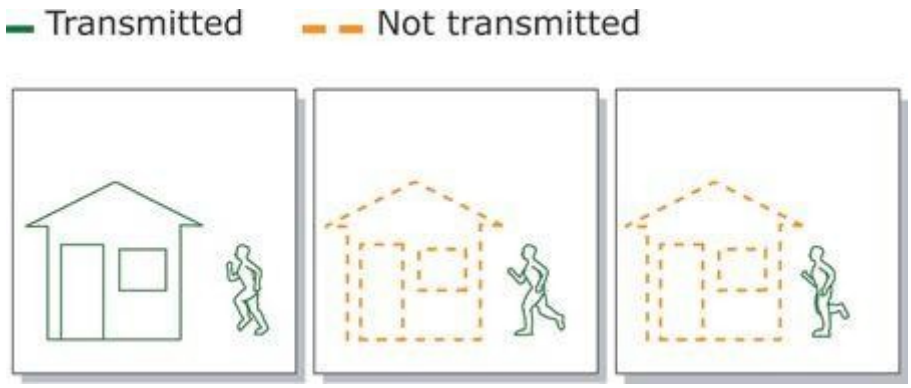


Ilustración II: Ejemplo de transmisión de secuencia

Con la codificación diferencial sólo la primera imagen (fotograma I) se codifica en su totalidad. En las dos imágenes siguientes (fotogramas P) existen referencias a la primera imagen en lo que se refiere a elementos estáticos, como la casa. Sólo se codifican las partes en movimiento (el hombre que corre) mediante vectores de movimiento, reduciendo así la cantidad de información que se envía y almacena.

Para reducir aún más los datos, se pueden aplicar otras técnicas como la compensación de movimiento basada en bloques. La compensación de movimiento basada en bloques tiene en cuenta que gran parte de un fotograma nuevo está ya incluido en el fotograma anterior, aunque quizás en un lugar diferente del mismo. Esta técnica divide un fotograma en una serie de macrobloques (bloques de píxeles). Se puede componer o "predecir" un nuevo fotograma bloque a bloque, buscando un bloque que coincida en un fotograma de referencia. Si se encuentra una coincidencia, el codificador codifica la posición en la que se debe encontrar el bloque coincidente en el fotograma de referencia. La codificación del vector de movimiento, como se denomina, precisa de menos bits que si hubiera de codificarse el contenido real de un bloque.

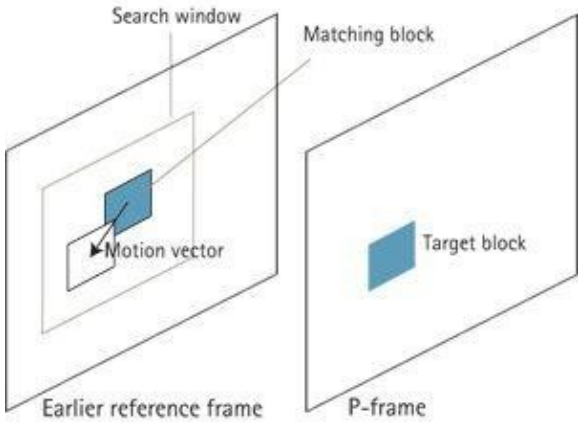


Ilustración III: Compensación de movimiento basada en bloques

Con la predicción interfotograma, cada fotograma de una secuencia de imágenes se clasifica como un tipo de fotograma concreto, como un fotograma I, P o B.

En la codificación de vídeo, un grupo de imágenes, o estructura de GOP, especifica el orden en el que se organizan las tramas intra e inter. El GOP es una colección de imágenes sucesivas dentro de un flujo de vídeo codificado. Cada secuencia de vídeo codificada consiste en sucesivos GOP, a partir de los cuales se generan los fotogramas visibles. Encontrar un nuevo GOP en un flujo de vídeo comprimido significa que el decodificador no necesita ninguna trama previa para decodificar las siguientes, y permite una búsqueda rápida a través del vídeo. Un GOP puede contener los siguientes tipos de fotogramas:

Un frame I, o intrafotograma, es una imagen autónoma que se puede codificar de forma independiente sin hacer referencia a otras imágenes, de ahí el nombre de intra. La primera imagen de una secuencia de vídeo es siempre un fotograma I. Los fotogramas I sirven como puntos de inicio en nuevas visualizaciones o como puntos de re sincronización si la transmisión de bits resulta dañada. Los fotogramas I se pueden utilizar para implementar funciones de avance o retroceso rápido o de acceso aleatorio. La desventaja de este tipo de fotogramas es que consumen muchos más bits, pero por otro lado no generan demasiados defectos provocados por los datos que faltan.

Un frameP (de interfotograma Predictivo), hace referencia a partes de fotogramas I o P anteriores para codificar el fotograma. Los fotogramas P suelen requerir menos bits que los fotogramas I, pero con la desventaja de ser muy sensibles a la transmisión de errores, debido a la compleja dependencia con fotogramas P o I anteriores.

Un frameB, o interfotograma Bipredictivo, es un fotograma que hace referencia tanto a fotogramas anteriores como posteriores. El uso de fotogramas B aumenta la latencia.

Un frameD o D (imagen directa codificada DC) sirve como una representación de acceso rápido de una imagen para robustez de pérdida o avance rápido. Las imágenes D sólo se utilizan en vídeo MPEG-1.

Un marco I indica el comienzo de un GOP. Después se siguen varios marcos P y B. En los diseños más antiguos, la ordenación y la estructura de referencias permitidas son relativamente limitadas.

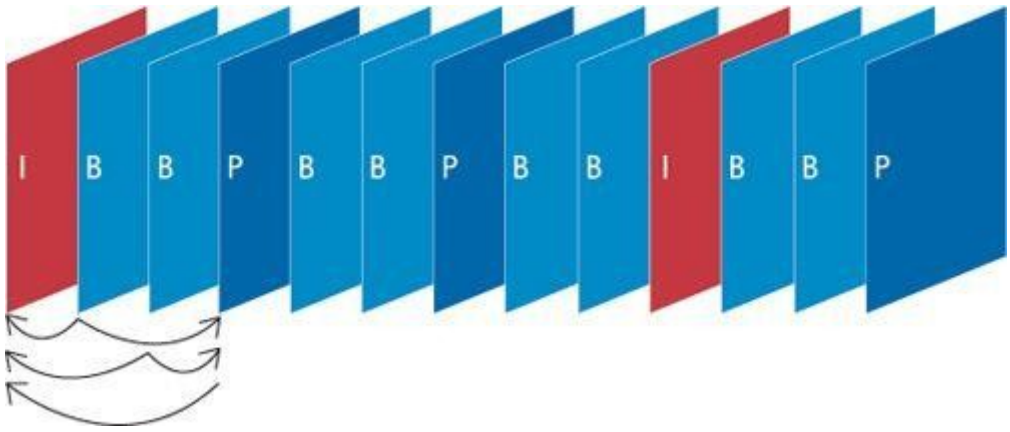


Ilustración IV: Estructura GOP

Un fotograma P sólo puede hacer referencia a fotogramas I o P anteriores, mientras que un fotograma B puede hacerlo a fotogramas I o P tanto anteriores como posteriores.

Para la detección de los cambios entre tomas se usa la tasa de transmisión. El primer frame de una nueva escena tendrá gran cantidad de macrobloques intracodificados, los cuales ocupan más, provocando de esta manera que el primer frame

de una nueva escena tenga una tasa de transmisión mucho mayor que los frames anteriores. Diversos estudios se han realizado sobre este parámetro, como el de Feng et al. en 1996[8]. Además, este parámetro se puede utilizar en combinación con otros métodos, como hacen Divakaran et al. en 1999[9] y Boccignone et al. en 2000[10], logrando una detección de cambio de toma de manera más robusta.

La transformada del coseno o DCT es una implementación específica de la transformada de Fourier donde la imagen es transformada de su representación espacial a su frecuencial equivalente. Cada elemento de la imagen se representa por ciertos coeficientes de frecuencia. Las zonas con colores similares se representan con coeficientes de baja frecuencia y las imágenes con mucho detalle con coeficientes de alta frecuencia. La información resultante son 64 coeficientes DCT.

El DCT reordena toda la información y la prepara para la Cuantización. El proceso de Cuantización es la parte del algoritmo que causa pérdidas y también ayuda en el control de velocidad, por ejemplo, permitiendo al codificador producir un flujo de bits a una determinada velocidad. Los coeficientes DCT son codificados y escaneados siguiendo un patrón en zigzag para crear una secuencia de una dimensión. La secuencia resultante usualmente contiene un gran número de ceros debido a la naturaleza del espectro DCT y del proceso de cuantificación. Cada coeficiente diferente de cero se asocia con un par de apuntadores. Primero, su posición en el bloque que se indica por el número de ceros entre él y el coeficiente anterior diferente de cero y el segundo, su valor.

[8] Feng, J., Lo, K.T., Mehrpour, H., 1996. “Scene change detection algorithm for MPEG video sequence”. In: Proc. IEEE Internat. Conf. on Image Processing, Lausanne, Switzerland.
[9] Divakaran, A., Ito, H., Sun, H., Poon, T., 1999. “Scene change detection and feature extraction for MPEG-4 sequences”. In: Proc. SPIE Conf. on Storage and Retrieval for Image and Video Databases VII, San Jose, CA, January, pp. 545–551.
[10] Boccignone, G., De Santo, M., Percannella, G., 2000. “Analgorithm for video cut detection in MPEG sequences”. In: Proc. SPIE Conf. on Storage and Retrieval of Media Databases 2000, San Jose, CA, January, pp. 523–530.

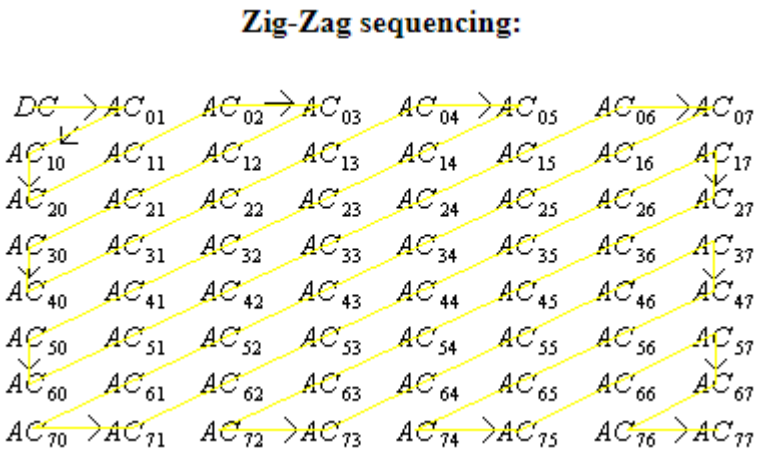


Ilustración V: Patrón zigzag

El patrón zigzag está basado en estos dos apuntadores, se le asigna un código de longitud variable en función de una tabla predeterminada. Este proceso se realiza de tal forma que las combinaciones con una alta probabilidad consiguen un código con pocos bits, mientras que los poco habituales obtienen un código mayor. Adoptando esta codificación sin pérdidas, el número total de bits disminuye. Sin embargo, ya que la redundancia espacial es limitada, las imágenes **I** sólo proporcionan una compresión moderada. Estas imágenes son muy importantes para acceso aleatorio utilizado para fines de edición. La frecuencia de imágenes **I** es normalmente de una cada 12 o 15 cuadros o frames. Un GOP está delimitado por dos cuadros **I**.

En las imágenes P y B es donde se proporciona su máxima eficiencia en compresión. Esto lo consigue mediante la compensación de movimiento, que como ya hemos explicado explota la redundancia temporal. Ya que los cuadros están relacionados, podemos asumir que una imagen puede ser modelada como una translación de la imagen en el instante anterior.

Entonces, es posible representar de manera precisa o predecir los valores de un cuadro basándonos en los valores del cuadro anterior, estimando el movimiento. Este proceso disminuye considerablemente la cantidad de información. En las imágenes P, cada macrobloque de tamaño 16x16 se predice a partir de un macrobloque de la anterior imagen I. Ya que, los cuadros son instantáneos en el tiempo de un objeto en movimiento, los macrobloques en los dos cuadros pueden no corresponder a la misma localización espacial, por lo tanto, se debe proceder a buscar en el cuadro I para

encontrar un macrobloque que coincida lo máximo posible con el macrobloque que se está considerando en el cuadro P.

La diferencia entre los dos macrobloques es el error de predicción. Este error puede codificarse como tal o en el dominio DCT. La DCT del error consigue pocos coeficientes de alta frecuencia, que tras la cuantificación requieren un número menor de bits para su representación. Las matrices de cuantificación para los bloques de error de predicción son diferentes de las utilizadas en los intra bloques, debido a la distinta naturaleza de sus espectros. La distancia en las direcciones horizontal y vertical del macrobloque coincidente con el macrobloque estimado se denomina vector de movimiento.

Los vectores de movimiento representan la translación de las imágenes de los bloques entre cuadros. Estos vectores se necesitan para la reconstrucción y son codificados de forma diferencial en el flujo de datos. Se utiliza codificación diferencial ya que reduce el total de bits requeridos para transmitir la diferencia entre los vectores de movimiento de los cuadros consecutivos. La eficiencia de la compresión y la calidad de la reconstrucción de la señal de vídeo dependen de la exactitud en la estimación del movimiento.

El método para este cálculo no se especifica en el estándar de compresión y por lo tanto está abierto a diferentes implementaciones y diseños, aunque evidentemente existe una relación directa entre la exactitud de la estimación de movimiento y la complejidad de su cálculo. Para los cuadros B, se utiliza la predicción de la compensación de movimiento y los cuadros de referencia (I y P) que se encuentran antes y después.

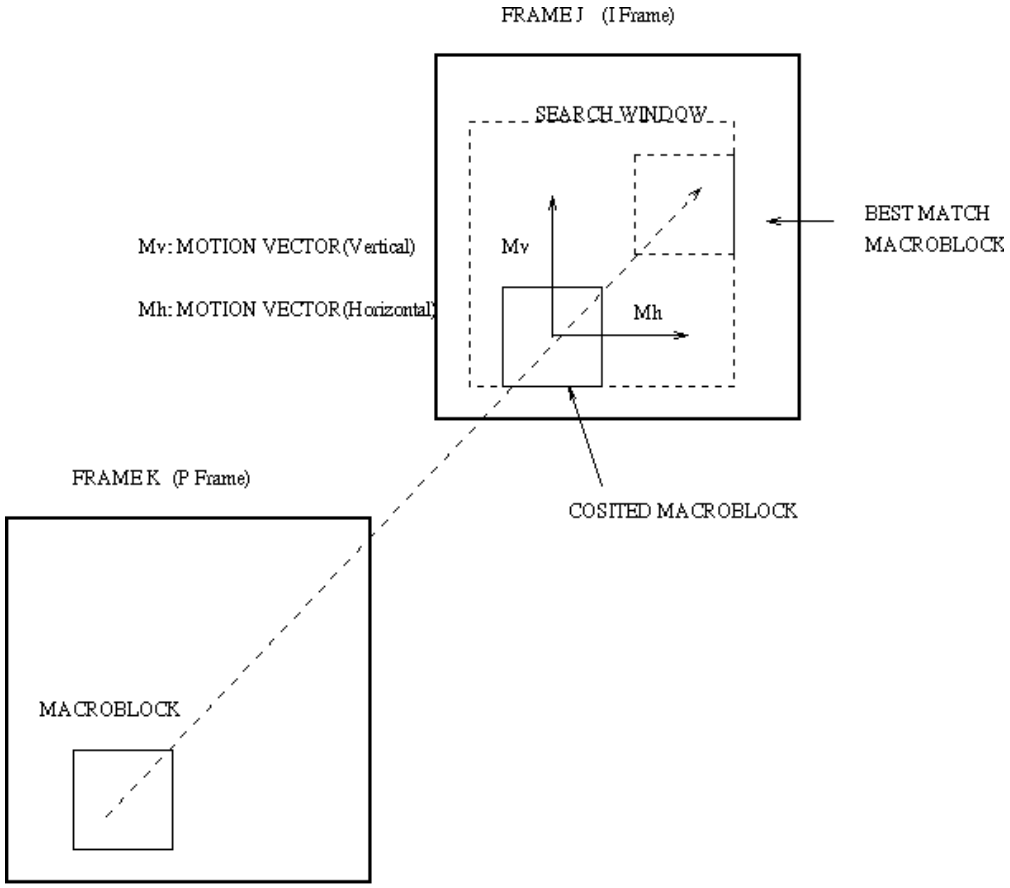


Ilustración VI: Proceso de predicción por compensación del movimiento

Además de la codificación diferencial y la compensación del movimiento, se pueden emplear otros métodos avanzados para reducir aún más los datos y mejorar la calidad de vídeo. El H264, por ejemplo, admite técnicas avanzadas como los esquemas de predicción para codificar fotogramas I, la compensación de movimiento mejorada con una precisión inferior a un píxel y el filtro de eliminación de bloques en bucle para suavizar los bordes de los bloques (defectos).

4.2.3. Evolución de los estándares de compresión

Los métodos de compresión de vídeo con perdida más utilizados son:

- **M-JPEG** (Motion JPEG): es una versión extendida del algoritmo JPEG que comprime imágenes. Básicamente consiste en tratar al vídeo como una secuencia de imágenes estáticas independientes a las que se aplica el proceso de compresión del algoritmo JPEG una y otra vez para cada imagen de la secuencia de vídeo. Existen cuatro modos de operación para el JPEG: secuencial, progresiva, sin pérdida, y jerárquica. Normalmente se utiliza el modo secuencial.

La ventaja es que se puede realizar en tiempo real e incluso con poca inversión en hardware. El inconveniente de este sistema es que no se puede considerar como un estándar de vídeo pues ni siquiera incluye la señal de audio. Otro problema es que el índice de compresión no es muy grande.

JPEG utiliza una técnica de compresión espacial, la intracuadros o DCT. El sistema JPEG solamente utiliza la compresión espacial al estar diseñado para comprimir imágenes individuales.

Motion-JPEG es el método elegido para las aplicaciones donde se envía la misma información a todos los usuarios, las broadcast.

- **MPEG:** Es una de las técnicas de vídeo y audio más conocidas; el estándar denominado MPEG (iniciado por el Motion Picture Experts Groups a finales de los años 80). El algoritmo que utiliza además de comprimir imágenes estáticas compara los fotogramas presentes con los anteriores y los futuros para almacenar sólo las partes que cambian.

  MPEG aplica la compresión temporal y la espacial. En primer lugar, se aplica una transformada de coseno discreta, seguida de una Cuantización para finalmente comprimir mediante un algoritmo Run Length Encoding (RLE). Los bloques de imagen y los de predicción de errores tienen una gran redundancia espacial, que se reduce gracias a la transformación de los bloques desde el dominio del espacio al dominio de frecuencia. Otro aspecto importante de MPEG es el modo en el que se usa el ancho de banda disponible. En la mayoría de los sistemas MPEG es posible seleccionar si el ratio de bits debe ejecutarse en modo CBR (constante) o VBR (variable). La selección óptima depende de la aplicación y de la infraestructura de red disponible.

  Existen diferentes opciones dependiendo del uso:

  - **MPEG-1** fue presentado en 1993 y está dirigido a aplicaciones de almacenamiento de vídeo digital en CD. Por esta circunstancia, la mayoría de los codificadores y decodificadores MPEG-1 precisan un ancho de banda de aproximadamente 1.5 Mbit/segundo a resolución CIF (352x288 píxeles).MPEG-1 guarda una imagen, la compara con la siguiente y almacena sólo las diferencias. Se alcanzan así grados de compresión muy elevados. Define tres tipos de fotogramas:

Fotogramas **I**, **P** y **B**. Consigue el mayor grado de compresión a costa de un mayor tiempo de cálculo. Estándar escogido por Vídeo-CD: calidad VHS con sonido digital.

- **MPEG-2** fue aprobado en 1994 como estándar y fue diseñado para vídeo digital de alta calidad (DVD), TV digital de alta definición (HDTV), medios de almacenamiento interactivo (ISM), retransmisión de vídeo digital (Digital Vídeo Broadcasting, DVB) y Televisión por cable (CATV). MPEG-2 se centra en ampliar la técnica de compresión MPEG-1 para cubrir imágenes más grandes y de mayor calidad con un nivel de compresión menor y un consumo de ancho de banda mayor. También proporciona herramientas adicionales para mejorar la calidad del vídeo consumiendo el mismo ancho de banda, con lo que se producen imágenes de muy alta calidad cuando lo comparamos con otras tecnologías de compresión.

- **MPEG-3** cuyo lanzamiento inicial fue en 1993. Fue diseñado originalmente para HDTV (Televisión de Alta Definición), pero abandonado posteriormente a favor de MPEG-2.

- **MPEG-4** cuya primera parte fue publicada en 1999. Es el estándar de transmisión de vídeo clásico, también denominado MPEG-4 Visual. Incorpora muchas más herramientas para reducir el ancho de banda preciso en la transmisión para ajustar una cierta calidad de imagen a una determinada aplicación o escena de la imagen. Es importante destacar, no obstante, que la mayoría de las herramientas para reducir el número de bits que se transmiten son sólo relevantes para las aplicaciones en tiempo no real. Esto es debido a que algunas de las nuevas herramientas necesitan tanta potencia de proceso que el tiempo total de codificación/decodificación (por ejemplo, la latencia) lo hace impracticable para otras aplicaciones que no sean la codificación de películas, codificación de películas de animación y similares. De hecho, la mayoría de las herramientas en MPEG-4 que pueden ser usadas en

aplicaciones en tiempo real son las mismas herramientas que están disponibles en MPEG-1 y MPEG-2.

- **H.264:** O MPEG-4 parte 10, es un códec digital de alta compresión estándar. La intención del proyecto H.264/AVC fue crear un estándar que sea capaz de proveer de una buena calidad de imagen con Bit Rates substancialmente menores que los estándares previos. Además de no incrementar la complejidad para que el diseño sea impracticable (demasiado caro) de implementar. Otro objetivo fue que el estándar fuera lo suficientemente flexible para ser aplicado a una gran variedad de aplicaciones y para trabajar correctamente en una gran variedad de redes y sistemas.

4.2.4. Comparación de estándares

Al comparar los rendimientos de los estándares MPEG como el MPEG-4 y H.264, es importante tener en cuenta que los resultados pueden variar entre codificadores que usen el mismo estándar. Esto se debe a que el diseñador de un codificador puede elegir implementar diferentes conjuntos de herramientas definidas por un estándar. Siempre que los datos de salida de un codificador se ajusten al formato de un estándar, se pueden realizar implementaciones diferentes. De ahí que un estándar MPEG no pueda garantizar una frecuencia de bits o calidad determinadas, del mismo modo que no se puede realizar una comparación como es debido sin definir primero cómo se han implementado los estándares en un codificador. Un decodificador, a diferencia de un codificador, debe implementar todas las partes necesarias de un estándar para descodificar una transmisión de bits compatible. Un estándar especifica exactamente la forma en la que el algoritmo de descompresión debe restaurar cada bit de un vídeo comprimido.

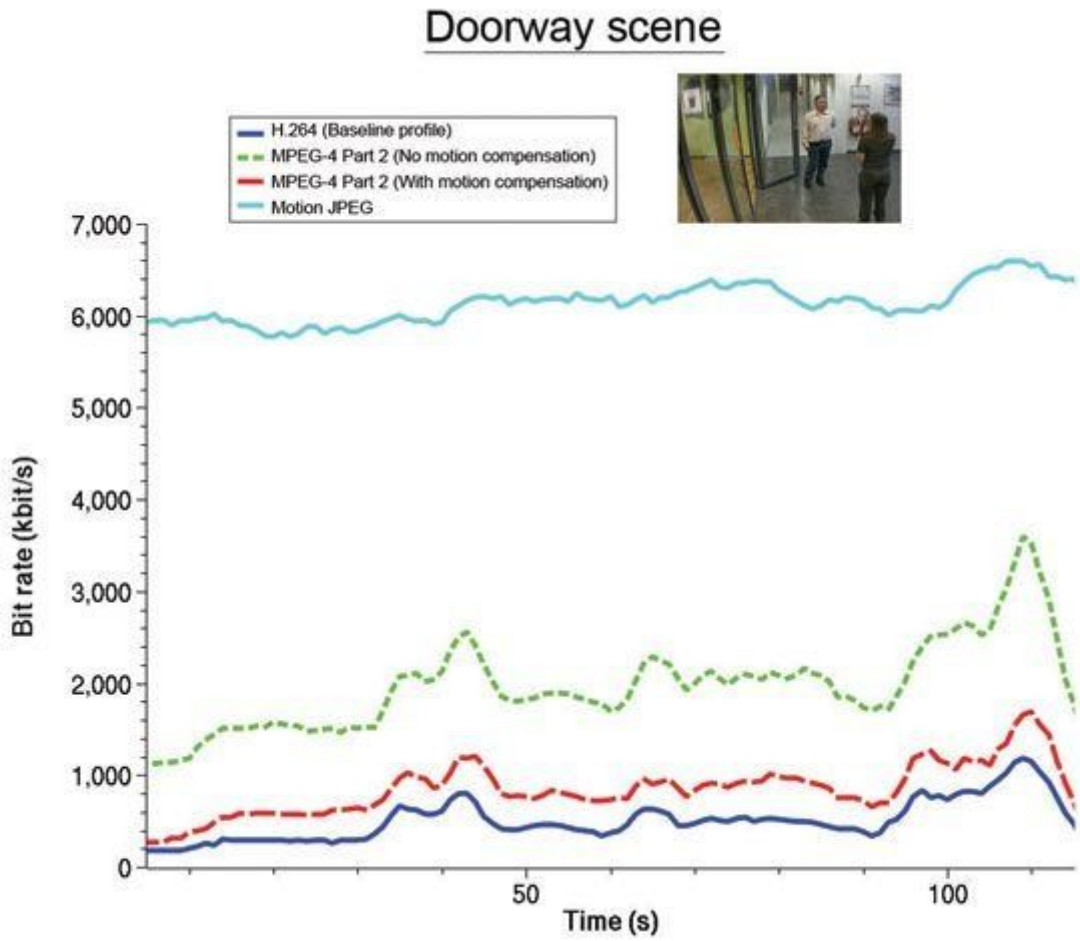


Ilustración VII: Gráfico comparación de estándares

El gráfico siguiente compara la frecuencia de bits, partiendo de la misma calidad de imagen, entre los siguientes estándares de vídeo: Motion JPEG, MPEG-4 Parte 2 (sin compensación de movimiento), MPEG-4 Parte 2 (con compensación de movimiento) y H.264 (perfil de base).

4.2.5. VBR vs CBR

Un medio de codificación VBR es el cuantificador fijo o la codificación de calidad fija. Por lo general, es una codificación de un solo pase. El usuario especifica un valor de calidad subjetivo dado, y el codificador asigna bits según sea necesario para alcanzar el nivel de calidad dado. Esto asegura que el flujo de salida tendrá una calidad consistente en todo. Un nivel de calidad generalmente tiene un rango de velocidad de bits asociado.

La codificación de tasa de bits constante significa que la velocidad a la que se deben consumir los datos de salida de un códec es constante. CBR es útil para transmitir contenido multimedia en canales de capacidad limitada, ya que es la velocidad máxima de bits lo que importa, no el promedio, por lo que CBR se utilizará para aprovechar toda la capacidad. CBR no sería la opción óptima para el almacenamiento, ya que no asignaría suficientes datos para secciones complejas (lo que daría como resultado una calidad degradada) mientras desperdiciaba datos en secciones simples.

4.3. Doble compresión

En muchas aplicaciones, los vídeos digitales están disponibles en formato comprimido, tales como MPEG. Sin embargo, las operaciones de manipulación se realizan a menudo en el dominio, como se muestra en la siguiente figura.

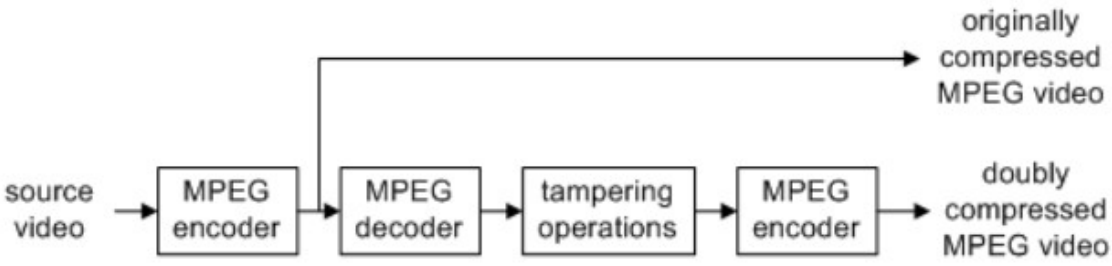


Ilustración VIII: Operaciones de manipulación de vídeo MPEG

Debido a su gran tamaño, el vídeo siempre necesita ser re-codificado y vuelto a guardar en el formato de compresión. Como resultado, se produce la doble compresión. Si los parámetros de cuantificación en el segundo codificador son diferentes de los del primer codificador, la compresión doble puede revelar la ocurrencia de alteración. Por lo tanto, la detección de doble compresión se puede utilizar para autenticar vídeo.

Se ha demostrado con JPEG que las distribuciones de probabilidad de los primeros dígitos de los coeficientes cuantificados no nulos se perturban si la imagen está doblemente comprimida, excepto para el movimiento de compensación. MPEG utiliza de alguna manera el esquema de compresión tipo JPEG para un marco de vídeo en términos de metodología. La similitud entre el MPEG y el JPEG nos inspira a investigar la distribución del primer digito de coeficientes cuantificados de MPEG no

nulos. Explotando la perturbación estadística en la distribución del primer dígito, se propone una nueva detección. Nos centraremos en la detección de doble compresión en vídeos comprimidos en CBR de MPEG.

4.3.1. Dataset de secuencias de vídeos

Vamos a seleccionar como secuencias de vídeo de origen varias secuencias de la fuente y4m[11], estas secuencias se cogerán en crudo, es decir, sin comprimir. Se utilizará el formato CIF (Common Intermediate Format), que define una secuencia de vídeo con una resolución en píxeles de 256x256 (ancho x alto). Estas secuencias incluyen los videos que vemos a continuación. El contenido y la complejidad de movimientos de las secuencias varían. Las secuencias de fotogramas se descargarán con extensión y4m, es un archivo de vídeo creado en el formato YUV4MPEG. Se almacena una secuencia de imágenes sin comprimir que componen el vídeo fotograma a fotograma, que se utiliza como un formato en bruto.

Akiyo (300 frames)

Bowing (300 frames)

City (600 frames)

Crew (600 frames)

[11] https://media.xiph.org/video/derf/

Football (260 frames)

Galeon (360 frames)

Garden (115 frames)

Harbour (600 frames)

Ice (480 frames)

Intros (360 frames)

Mis_am (150 frames)

Mobile calendar (360 frames)

Ilustración IX: Secuencias de vídeo y4m

4.3.2. Vídeos en formato MPEG

Una vez que tenemos las secuencias de vídeo en bruto vamos a comprimirlas en MPEG para después descomprimir el vídeo, y volver a comprimirlo. De esta forma vamos a realizar el mismo proceso que llevaríamos a cabo si quisiéramos realizar alguna modificación en las secuencias.

Para la compresión de un vídeo en y4m a MPG/MPEG vamos a usar el programa "Total Vídeo Audio Converter" [12]. Una vez descargado accedemos a opciones para configurar el vídeo y usaremos una resolución en píxeles de 256x256. Para la primera compresión utilizaremos CBR (Constant Bit Rate) con un Bit Rate de 10.000 kbps.

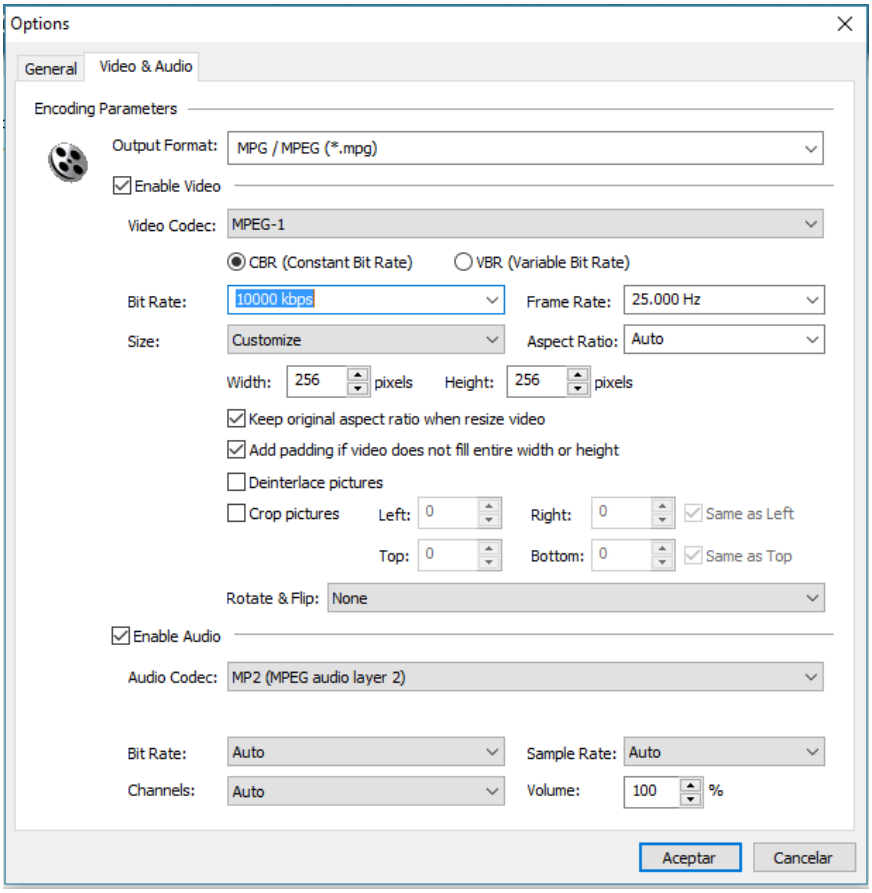


Ilustración X: Configuración de opciones CBR

Después de configurar las opciones introduciremos el vídeo y lo comprimiremos en MPEG.

[12] http://www.hootech.com/formats/mpeg/convert-y4m-to-mpeg.htm

Ilustración XI: Compresión de secuencias y4m a MPEG

Vamos a realizar el mismo proceso pero para VBR, con una calidad de seis:

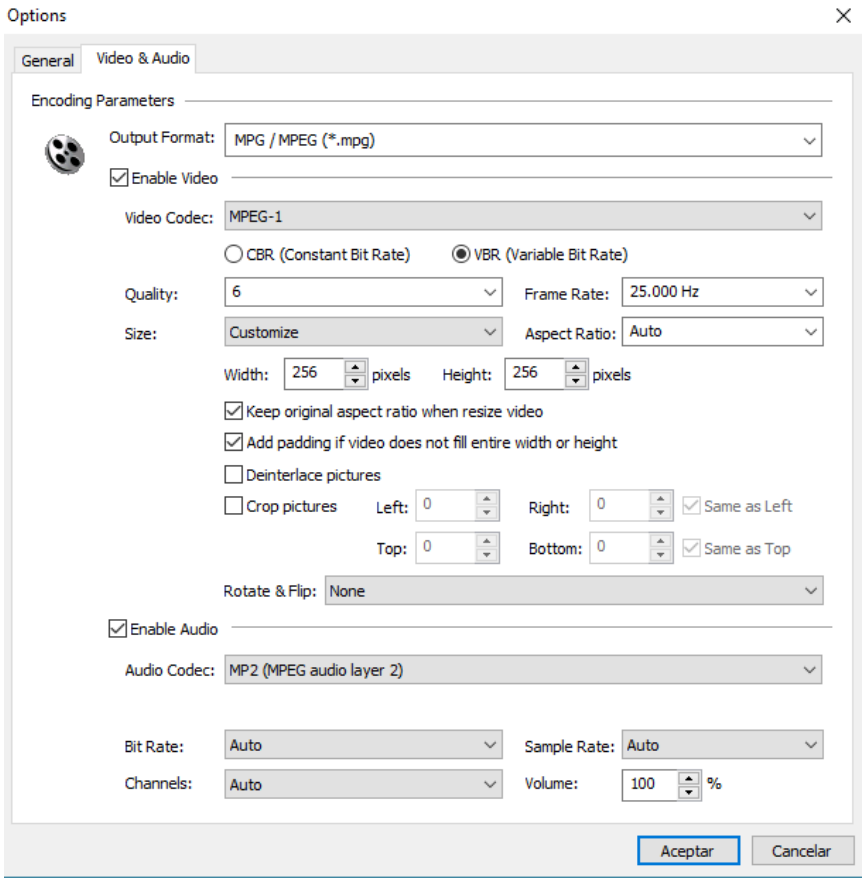


Ilustración XII: Configuración de opciones VBR

A continuación, para conseguir una doble compresión en nuestro vídeo vamos a descomprimirlo y volver a comprimirlo. Para la descompresión y doble compresión de vídeo utilizaremos un códec/decodec llamado VirtualDub [13].

Ilustración XIII: Configuración de opciones CBR

Con este programa vamos a realizar la decodificación del vídeo. Además, podemos ver el número total de frames que tiene el vídeo una vez que hemos realizado la primera compresión, como podemos ver, el vídeo de Akiyo consta de ciento veinticinco frames comprimido en MPEG.

Realizaremos este mismo proceso con el resto de secuencias y4m.

4.3.3. Distribución del primer digito en vídeos MPEG

4.3.3.1. Background

1) Espacio de color

Antes de codificar vídeo a MPEG-1, el espacio de color se transforma en Y'CbCr (Y '= Luma, Cb = Chroma Blue, Cr = Chroma Red). Luma (brillo y resolución) se almacena separadamente del croma (color, tonalidad y fase) y aún más separados en componentes rojos y azules. El croma es también sub-muestreado a 4:2:0, lo que significa que se reduce a la mitad verticalmente y la mitad horizontalmente, a sólo un cuarto de la resolución del vídeo. Este algoritmo de software también tiene analogías en

[13] http://www.virtualdub.org/download.html

hardware, como la salida de un filtro de patrón Bayer, común en cámaras digitales a color.

Debido a que el ojo humano es mucho más sensible a los pequeños cambios de brillo (el componente Y) que al color (los componentes Cr y Cb), el submuestreo cromático es una forma muy eficaz de reducir la cantidad de datos de vídeo que se deben comprimir.

Debido al submuestreo, el vídeo Y'CbCr siempre debe almacenarse usando dimensiones pares (divisible por 2), de lo contrario se producirá un desajuste cromático ("fantasmas"), y aparecerá como si el color estuviera delante o detrás del resto de la pantalla. Y'CbCr es nuestro archivo y4m.

2) Resolución de vídeo

La resolución que se usará para los vídeos que vamos a analizar será de 256x256.

3) Frames

MPEG-1 tiene varios tipos de frame que sirven para diferentes propósitos. El más importante, aunque más sencillo, es I-frame.
La compresión de I-frame es muy rápida, pero produce tamaños de archivo muy grandes, más grande que el vídeo MPEG-1 normalmente codificado, dependiendo de cuán complejo temporalmente sea un vídeo específico.

4) DCT

Cada bloque 8x8 se codifica aplicando primero una transformada de coseno discreta directa (FDCT) y luego un proceso de cuantificación. El proceso FDCT (por sí mismo) es teóricamente sin pérdidas, y puede revertirse aplicando un DCT Inverso (IDCT) para reproducir los valores originales

El proceso FDCT convierte el bloque 8x8 de valores de píxeles no comprimidos (valores de diferencia de brillo o color) en una matriz indexada 8x8 de valores de coeficiente de frecuencia. Uno de ellos es el coeficiente DC (estadísticamente alto en varianza), que representa el valor promedio de todo el bloque 8x8. Los otros 63 coeficientes son los coeficientes AC más pequeños, que son valores positivos o negativos, cada uno representando desviaciones sinusoidales del valor de bloque plano representado por el coeficiente DC.

Dado que el valor del coeficiente DC está correlacionado estadísticamente de un bloque al siguiente, se comprime usando la codificación DPCM. La modulación por código de impulsos (PCM) es un método utilizado para representar digitalmente señales analógicas muestreadas. Es la forma estándar de audio digital en computadoras, discos compactos, telefonía digital y otras aplicaciones de audio digital. En una corriente PCM, la amplitud de la señal analógica se muestrea regularmente a intervalos uniformes, y cada muestra se cuantifica al valor más próximo dentro de un intervalo de pasos digitales.

Sólo la cantidad (menor) de diferencia entre cada valor DC y el valor del coeficiente DC en el bloque a su izquierda necesita ser representada en el flujo de bits final.

A continuación, podemos ver la fórmula matemática para calcular del DCT-1.

$$X_k = \frac{1}{2}(x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi}{N-1} nk\right] \qquad k = 0, \ldots, N-1.$$

Ilustración XIV: Fórmula DCT-1

Vamos a ver un ejemplo de los coeficientes DCT de una imagen con su matriz 8x8 en escala de grises. La imagen que usaremos como ejemplo será la siguiente:

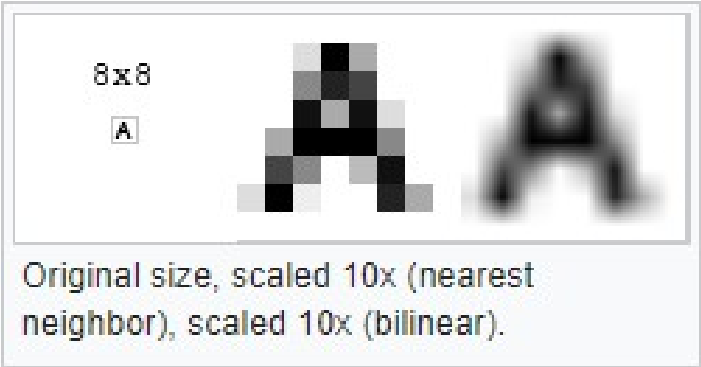
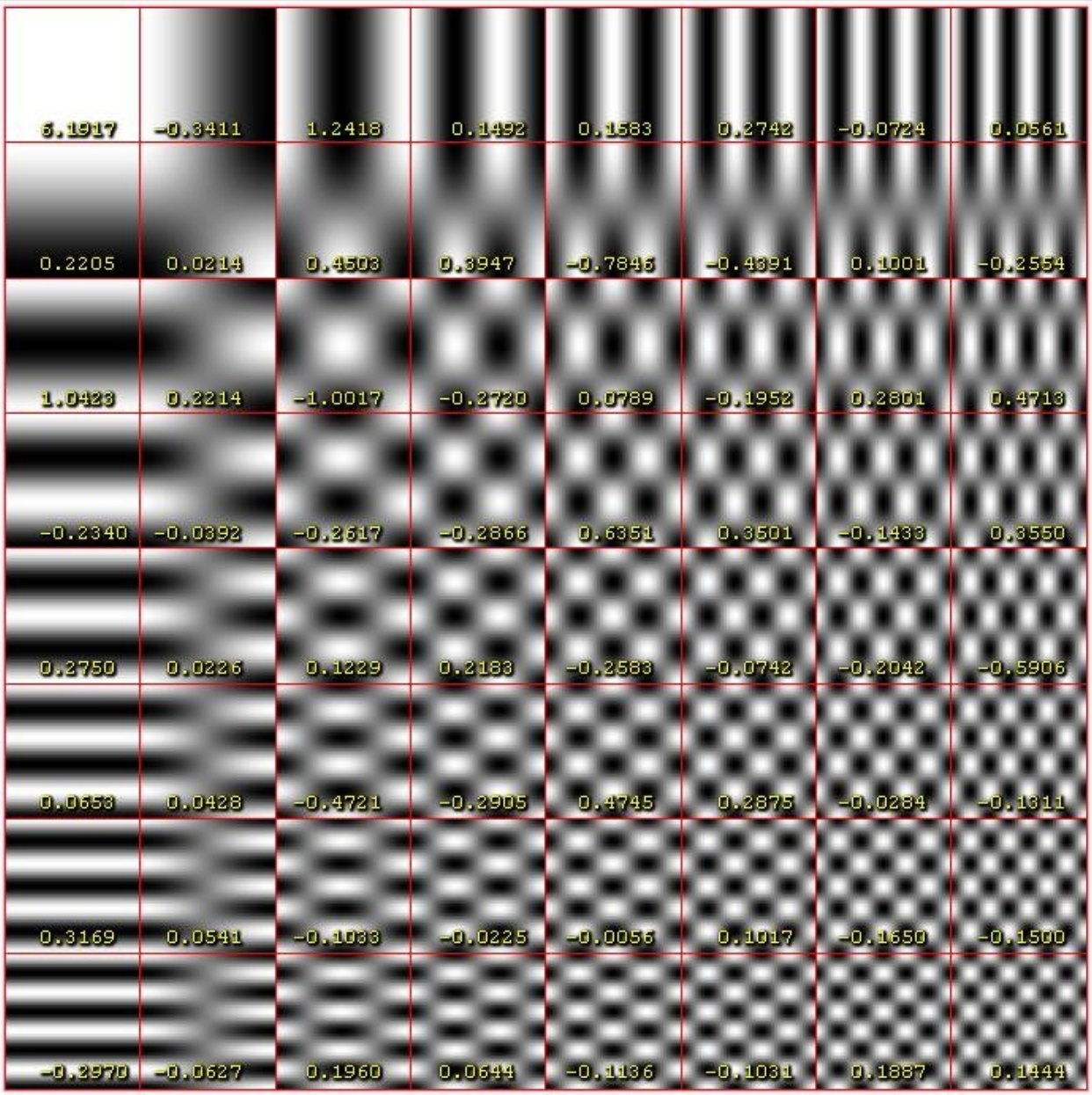


Ilustración XV: Imagen de ejemplo DCT-1



Basis functions of the discrete cosine transformation with corresponding coefficients (specific for our image).

$$\text{DCT of the image} = \begin{bmatrix} 6.1917 & -0.3411 & 1.2418 & 0.1492 & 0.1583 & 0.2742 & -0.0724 & 0.0561 \\ 0.2205 & 0.0214 & 0.4503 & 0.3947 & -0.7846 & -0.4391 & 0.1001 & -0.2554 \\ 1.0423 & 0.2214 & -1.0017 & -0.2720 & 0.0789 & -0.1952 & 0.2801 & 0.4713 \\ -0.2340 & -0.0392 & -0.2617 & -0.2866 & 0.6351 & 0.3501 & -0.1433 & 0.3550 \\ 0.2750 & 0.0226 & 0.1229 & 0.2183 & -0.2583 & -0.0742 & -0.2042 & -0.5906 \\ 0.0653 & 0.0428 & -0.4721 & -0.2905 & 0.4745 & 0.2875 & -0.0284 & -0.1311 \\ 0.3169 & 0.0541 & -0.1033 & -0.0225 & -0.0056 & 0.1017 & -0.1650 & -0.1500 \\ -0.2970 & -0.0627 & 0.1960 & 0.0644 & -0.1136 & -0.1031 & 0.1887 & 0.1444 \end{bmatrix}$$

Ilustración XVI: Frecuencia de dominio DCT

5) Cuantificación

La cuantificación (de datos digitales) es, esencialmente, el proceso de reducir la precisión de una señal, dividiéndola en un tamaño de paso mayor (es decir, encontrando el múltiplo más cercano y descartando el resto).

El cuantificador de nivel de trama determina la cantidad de información que se eliminará de una trama dada. El cuantificador de nivel de trama se selecciona dinámicamente por el codificador para mantener una determinada velocidad de bits especificada por el usuario o (mucho menos comúnmente) directamente especificada por el usuario.

Contrariamente a la creencia popular, un cuantificador de nivel de marco fijo (establecido por el usuario) no proporciona un nivel constante de calidad. En su lugar, es una métrica arbitraria que proporcionará un nivel variable de calidad, dependiendo del contenido de cada fotograma. Dado que dos archivos de tamaños idénticos, el codificado en una tasa de bits promedio debe verse mejor que la codificada con un cuantificador fijo (tasa de bits variable). Sin embargo, la codificación de cuantificador constante puede utilizarse para determinar con precisión los débitos mínimos y máximos posibles para codificar un vídeo dado.

Una matriz de cuantificación es una cadena de 64 números (0-255) que le dice al codificador cuán relativamente importante o poco importante es cada pieza de información visual. Cada número en la matriz corresponde a un cierto componente de frecuencia de la imagen de vídeo.

La cuantificación se realiza tomando cada uno de los 64 valores de frecuencia del bloque DCT, dividiéndolos por el cuantificador a nivel de trama, dividiéndolos luego por sus valores correspondientes en la matriz de cuantificación. Finalmente, el resultado se redondea hacia abajo. Esto reduce significativamente, o elimina completamente la información en algunos componentes de frecuencia de la imagen. Típicamente, la información de alta frecuencia es menos importante visualmente, y por lo tanto las frecuencias altas son mucho más fuertemente cuantificadas (drásticamente reducidas). En realidad MPEG-1 utiliza dos matrices de cuantificación separadas, una para bloques intra-bloques (I-bloques) y otra para interbloqueo (bloques P y B), por lo que la cuantificación de diferentes tipos de bloques puede hacerse independientemente y

este proceso de cuantificación generalmente reduce un número significativo de los coeficientes de CA a cero (conocidos como datos escasos) que pueden ser comprimidos más eficientemente mediante la codificación de entropía (compresión sin pérdida) en la siguiente etapa.

La Cuantización elimina una gran cantidad de datos y es el principal paso de procesamiento con pérdidas en la codificación de vídeo MPEG-1. Esta es también la fuente primaria de la mayoría de los artefactos de compresión de vídeo MPEG-1, como bloqueo, bandas de color, ruido, timbre, decoloración y otros. Esto ocurre cuando el vídeo se codifica con una velocidad de bits insuficiente, por lo que el codificador se ve forzado a utilizar cuantificadores de alto nivel de cuadro (Cuantización fuerte) a través de gran parte del vídeo.

4.3.3.2. Proceso con Matlab

Para obtener los coeficientes de cuantificación vamos a usar Matlab. Comenzaremos extrayendo los coeficientes DCT de cada frame que contiene el vídeo, para finalmente obtener los coeficientes de Cuantización.

Primero vamos a separar el vídeo en imágenes, y extraeremos de la primera imagen un cuadrado de 8 por 8 píxeles, tendrá las mismas dimensiones que la matriz de coeficientes DCT. Además, permite que las regiones altamente correlacionadas sean comprimidas más eficientemente. Lo siguiente que vamos a hacer con el cuadrado es separar la luminancia de la crominancia, es decir, le quitaremos el color a la imagen y nos quedaremos con la imagen en blanco y negro.

Después se convertirá cada imagen del vídeo en escala de grises (8-bit grayscale). Es decir, cada pixel está compuesto de una determinada cantidad de luz. Por lo tanto, se obtiene una matriz bidimensional de elementos con un rango de 0 a 255 (256 números).

El DCT es una función coseno, que puede tomar valores de 1 a -1. Lo que hacemos en el eje x es ir de 0 a π y a 2*π, en grados sería 180º en π y 360 en 2*π.

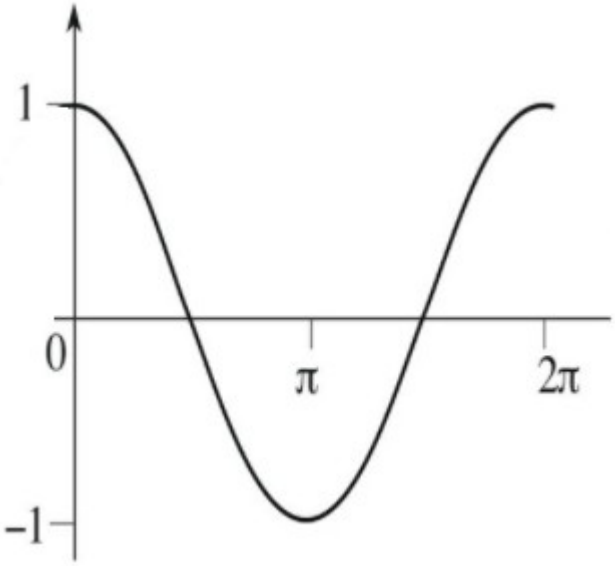


Ilustración XVII: Función coseno

Cada onda representa una parte constituyente de la salida. En la siguiente imagen podemos ver el significado de la función coseno en una imagen, cuando la función toma el valor 1 tendremos más luz y cuando la función toma el valor -1 tendremos más negro.

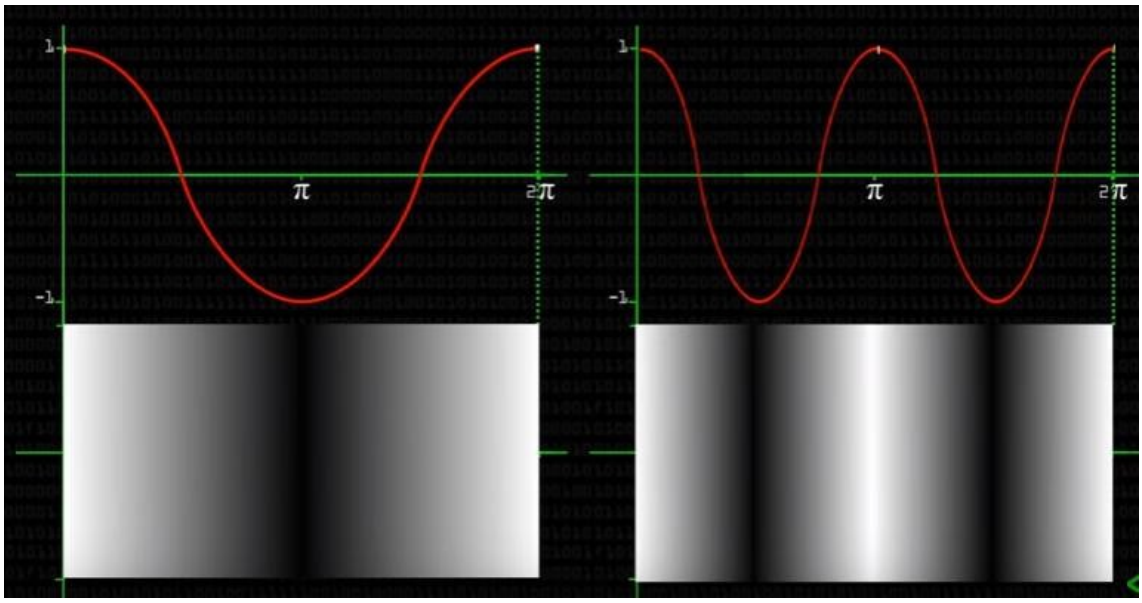


Ilustración XVIII: Función coseno con equivalencia en una imagen

Cuando mayor sea la frecuencia de la onda, mayor será la parte de frecuencia de la señal.

Cada onda esta ponderada con un coeficiente, que es un número que representa la contribución de cada uno de estos bloques individuales con el todo. Así, por ejemplo, si un coeficiente vale cero, entonces no hay parte de coseno en esa imagen de 8 por 8.

Lo que haremos con la transformada de coseno será calcular ese coeficiente para cada onda, aplicando la fórmula del DCT.

Puesto que tenemos una matriz bidimensional de elementos con un rango de 0 a 255 (256 números).Dado que la onda de coseno va de 1 a -1, vamos a desplazar cada uno de nuestros valores, restándoles 128. De esta manera obtendremos la misma imagen, pero esta vez centrada en torno a cero.

Después de esto aplicamos la formula DCT-1, y obtendremos de esta manera los coeficientes DCT. Generalmente el primer coeficiente de la matriz, será el mayor, esto se debe a que la imagen es plana y el coseno representa la intensidad general de ese bloque de imagen en particular. Por lo que este coeficiente se llama DC o coeficiente de corriente continua, el resto de coeficientes están en corriente alterna, son coeficientes AC.

El siguiente paso después de calcular los coeficientes DCT es realizar el proceso de Cuantización para convertir una señal con un rango de valores X, en otra señal con un reducido rango de valores Y, donde el menor número de posibles valores de la señal cuantizada permite representarla con menos bits que la señal original, produciendo un efecto de compresión de datos.

Para realizar el proceso de Cuantización debemos elegir una tabla estándar de cuantificación y su calidad, en nuestro caso hemos elegido la tabla de Cuantización JPEG con una calidad del 50%. La mayoría de los programas usan las mismas tablas estándar de Cuantización, excepto Photoshop que utiliza sus propias tablas de Cuantización.

Para calcular los coeficientes de Cuantización dividiremos la matriz de coeficientes DCT entre la tabla estándar JPEG. Una vez que lo tengamos dividido redondearemos los coeficientes al entero más cercano. Lo que obtendremos será una matriz cuyos coeficientes situados en la esquina inferior derecha serán cero y todos los valores se encontraran en la parte superior izquierda, dichos valores serán los únicos que tengan algún efecto sobre nuestra imagen. Con solo estos pocos coeficientes podemos

conseguir casi la misma imagen de nuevo, pero con menos calidad y un par de pixeles hacia arriba o hacia abajo en intensidad.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Ilustración XIX: Tabla estándar de Cuantización JPEG con calidad 50%

Una vez tenemos las matrices de coeficientes, realizaremos este mismo proceso con cada cuadrado de 8 por 8 de cada frame, y almacenaremos en un archivo de texto todas las matrices obtenidas de cada frame. Por lo que tendremos un archivo de texto por cada frame del vídeo.

4.3.3.3. Frames I, P, B

Para ver los frames I, P, B que tiene cada vídeo, hemos usado el programa VirtualDub, en el que nos muestra los frames totales, la media, el mínimo, el máximo, y el total. Vamos a comparar la información que obtenemos con un vídeo comprimido una vez y con un vídeo que tiene doble compresión.

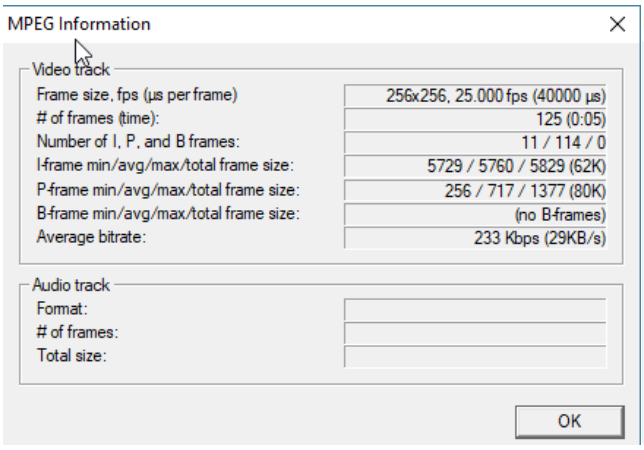


Ilustración XX: Información vídeo una compresión

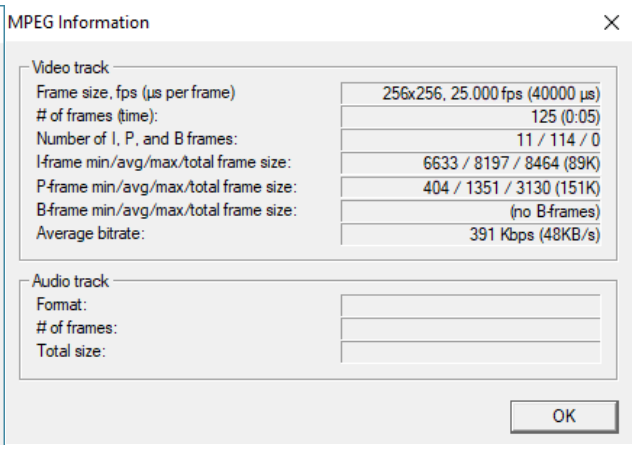


Ilustración XXI: Información vídeo con doble compresión

Si comparamos un vídeo con una sola compresión y otro vídeo con doble compresión podemos ver que el número de frames no varía. Sin embargo, el mínimo, el máximo, la media y el total de cada frame I y P, sí que varían de una sola compresión a dos, por lo que podemos demostrar de esta forma que los frames I, P y B varían en la doble compresión.

4.3.3.4. Test de comparación visual

Si una imagen JPEG dada, se comprime solo una vez, se ha observado por Fu, D., Shi, Y.Q., Su, W.[14] que la distribución de probabilidad de los coeficientes de los primeros dígitos de AC cuantificados no nulos siguen la ley logarítmica paramétrica (también llamada Ley de Bendford).

$$p(x) = Nlog_{10}(1 + \frac{1}{s + x^q}) , x = 1, 2, \cdots, 9$$

Ilustración XXII: Ley de Benford

Donde x representa los primeros dígitos de coeficientes de AC distintos de cero, N, s y q son parámetros para describir con precisión la distribución. Por otra parte, se muestra que la distribución del primer dígito violará la ley cuando la doble compresión ocurre.

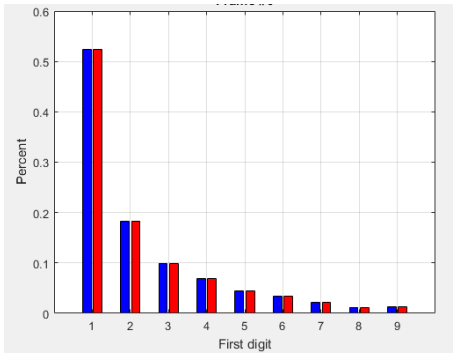
Como dijimos anteriormente MPEG usa de algún modo un esquema de compresión similar a JPEG. La similitud entre MPEG y JPEG nos anima a investigar cómo los primeros dígitos de los coeficientes de AC cuantificados MPEG distintos de cero siguen la ley logarítmica.

La primera prueba que vamos a llevar a cabo es una comparación visual de la distribución real del primer dígito y la distribución ajustada del primer dígito. Para ello hemos usado Matlab, en concreto, Curve Fitting para encontrar la distribución ajustada.

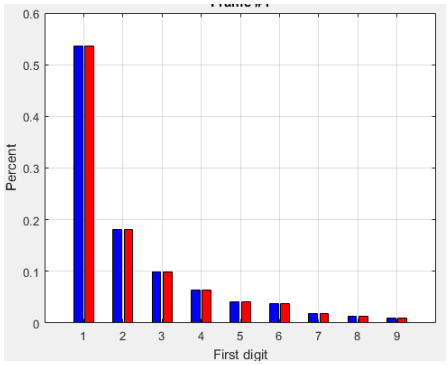
Las comparaciones visuales se demuestran a través de la secuencia de video de Akiyo y Football. La razón por la que hemos elegido Akiyo y Football, es que contienen una complejidad de movimiento bastante diferente, es decir, la primera tiene menos movimiento, mientras que el segundo tiene un movimiento rápido.

[14] Fu, D., Shi, Y.Q., Su, W.: A generalized Benford's law for JPEG coefficients and its application in image forensics. In: IS&T/SPIE 19TH Annual Symposium, San Jose, California, USA (2007)

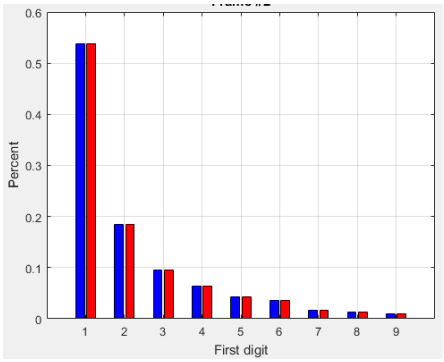
A continuación, vemos representada la distribución del primer dígito, en color azul, y la distribución ajustada del primer dígito en color rojo. Vamos a verlo con CBR con un Bit Rate de 10.000.



Frame I

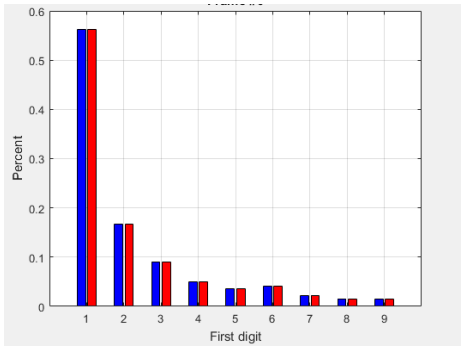


Frame P

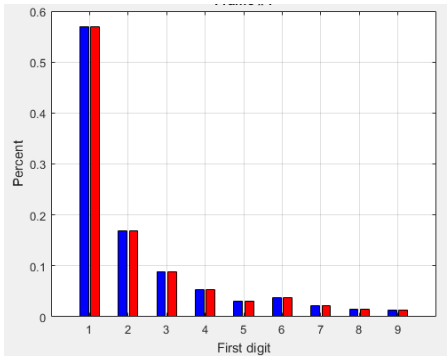


Frame B

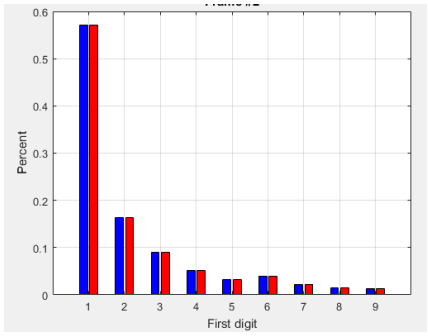
Ilustración XXIII: La primera distribución media de video originalmente comprimido con MPEG (Akiyo) CBR



Frame I



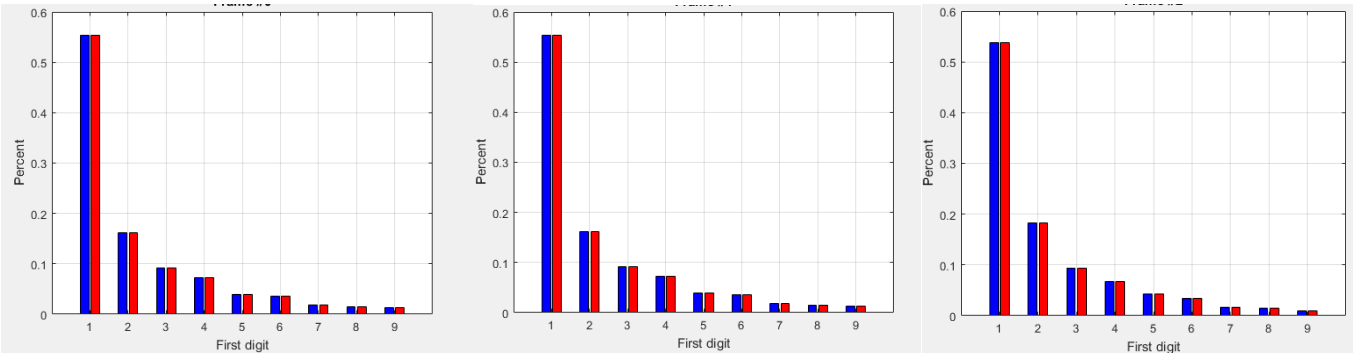
Frame P



Frame B

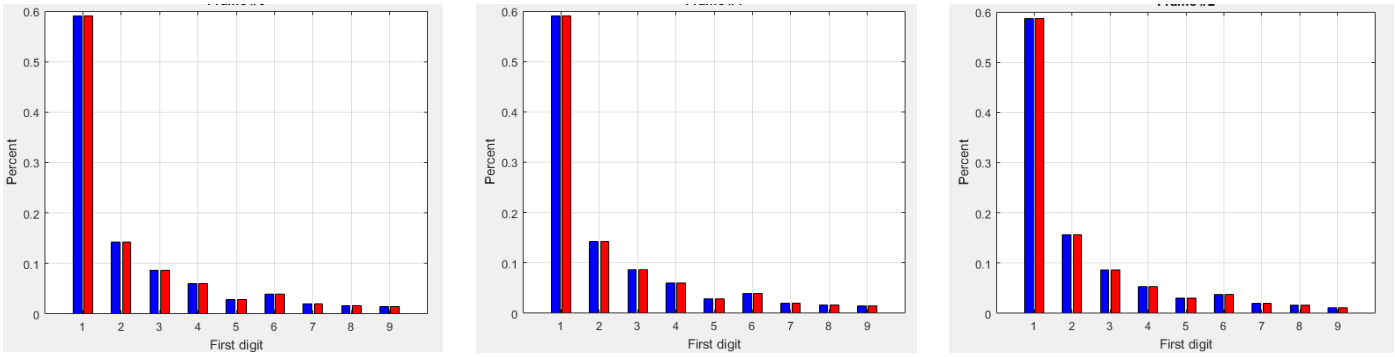
Ilustración XXIV: La primera distribución media de video originalmente comprimido con MPEG (Football) CBR

A continuación vamos a ver la distribución con VBR con una calidad de seis.



Frame I Frame P Frame B

Ilustración XXV: La primera distribución media de video originalmente comprimido con MPEG (Akiyo) VBR



Frame I Frame P Frame B

Ilustración XXVI: La primera distribución media de video originalmente comprimido con MPEG (Football) VBR

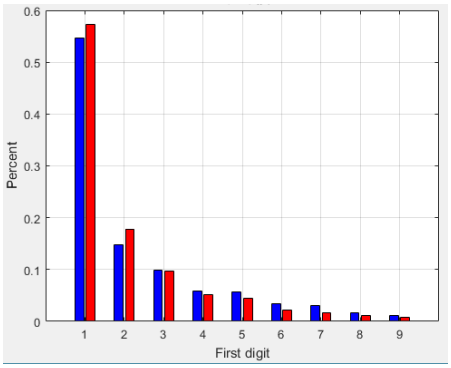
Las primeras distribuciones de dígitos de la secuencia de video comprimida originalmente MPEG utilizando el modo CBR se ilustran en la figura anterior. Allí los ejes horizontales representan los primeros dígitos con valores de 1 hasta 9 y en los ejes verticales representan la frecuencia relativa real y ajustada del primer dígito.

Como se muestra en las dos ilustraciones anteriores, los coeficiente AC no nulos de los videos comprimidos en MPEG están bien modelados por la ley logarítmica en ambos casos de VBR y CBR.

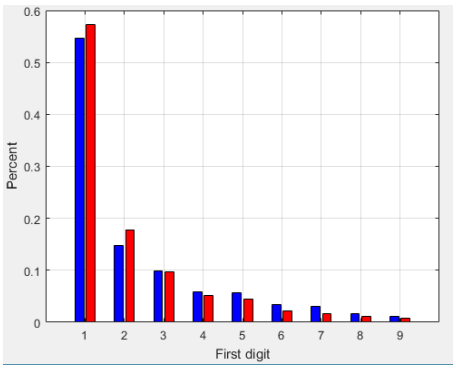
Para demostrar la distribución del primer dígito de video comprimido MPEG doble, una vez más, Akiyo y Football se usan como ejemplos. En el caso del control de VBR, originalmente los videos MPEG con calidad igual a seis se descomprimieron y luego se volvieron a comprimir para generar un video comprimido MPEG doble. La

calidad para la recompresión es diferente de 6, ya que cuando realizas la doble compresión con VBR no siempre usas la misma calidad inicial que tenía el vídeo.

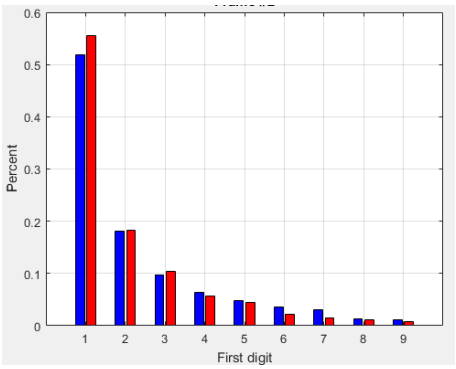
Por ello vamos a comprobar que pasa con la ley de Benford si la calidad es mayor a seis (calidad de 9) o menor a seis (calidad de 4), con VBR. Con el color azul representaremos un vídeo con una compresión y con el color rojo un vídeo con doble compresión. Vamos a comenzar primero con Akiyo.



Frame I

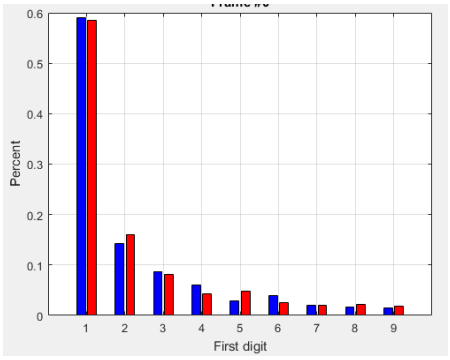


Frame P

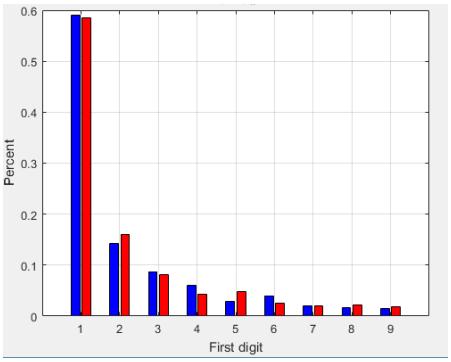


Frame B

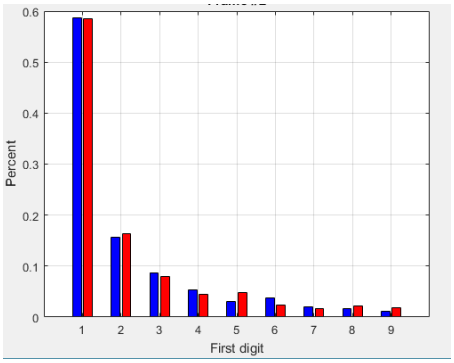
Ilustración XXVII: Akiyo calidad 9



Frame I



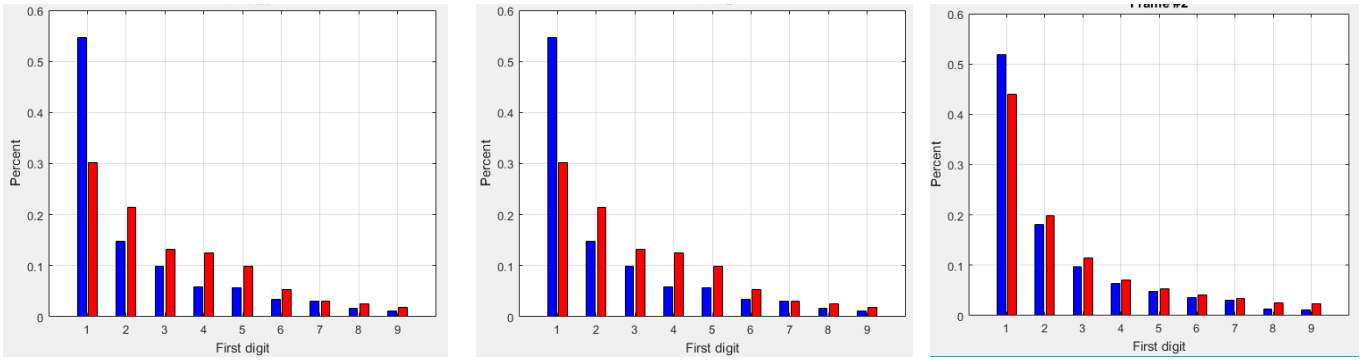
Frame P



Frame B

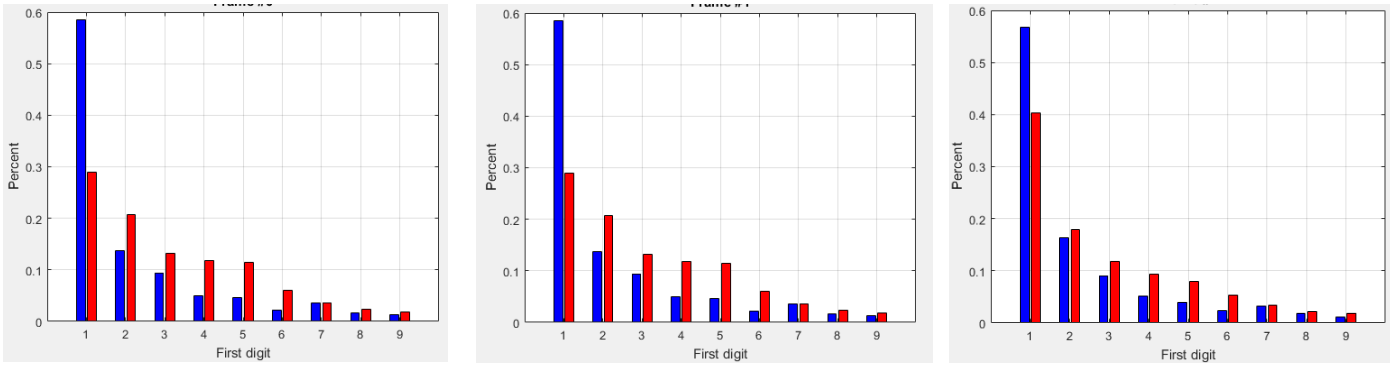
Ilustración XXVIII: Football calidad 9

Vamos a probar ahora con calidad cuatro.



Frame I Frame P Frame B

Ilustración XXIX: Akiyo calidad 4



Frame I Frame P Frame B

Ilustración XXX: Fooball calidad 4

Para el caso en que la calidad para recompresión es menor que 6, como en la ilustración veintisiete y veintiocho, la violación de la ley no es visiblemente obvia en marcos I, P o B.

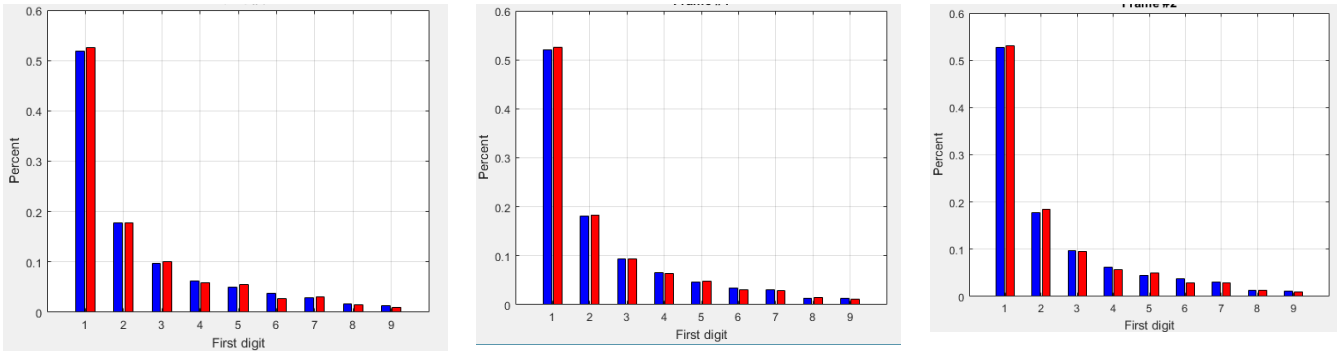
Para el caso en que la calidad de la recompresión es mayor que 6, como en la ilustración veintinueve y treinta, la violación de la ley no es visiblemente obvia en marcos I, P o B. A partir de las observaciones, se concluye que la doble compresión se puede identificar examinando visualmente la diferencia del primer dígito de coeficientes cuantificados AC MPEG distintos de cero en I frames si la calidad para la recompresión es menor que la calidad para la compresión original. Pero la doble compresión es difícil detectar visualmente si la calidad para la recompresión es mayor que la calidad para la compresión original.

De las observaciones, se concluye que la doble compresión puede ser identificada al examinar visualmente la diferencia del primer dígito AC no nulo de MPEG en los frames I si la calidad para recompresión es mayor que la calidad para la

compresión original, pero la doble compresión es difícil de detectar visualmente si la calidad para la recompresión es menor que la calidad para la compresión original.

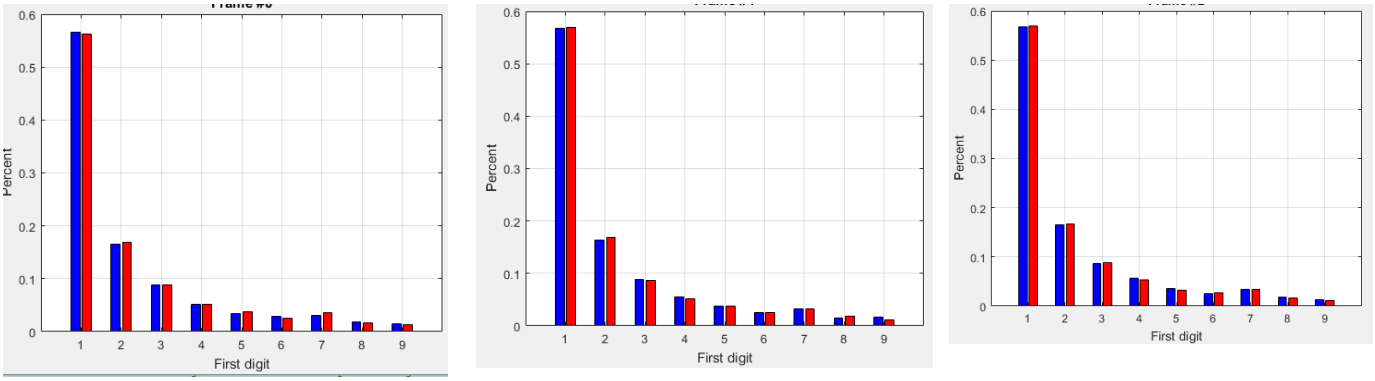
En el caso del control de CBR, la tasa de bits objetivo utilizada para la recompresión es más pequeño o mayor que 10.000 bits/s con el control de CBR, la calidad puede variar en el nivel de macrobloque dentro de cada marco de modo que la tasa de bits de salida permanezca constante. Si la tasa de bits objetivo utilizada para la recompresión es diferente de la compresión original, el control de velocidad de bits elegirá una calidad diferente de la que tenga el vídeo original comprimido MPEG.

Ahora vamos a realiza la doble compresión pero para videos en CBR, variando la tasa de bits.



Frame I Frame P Frame B

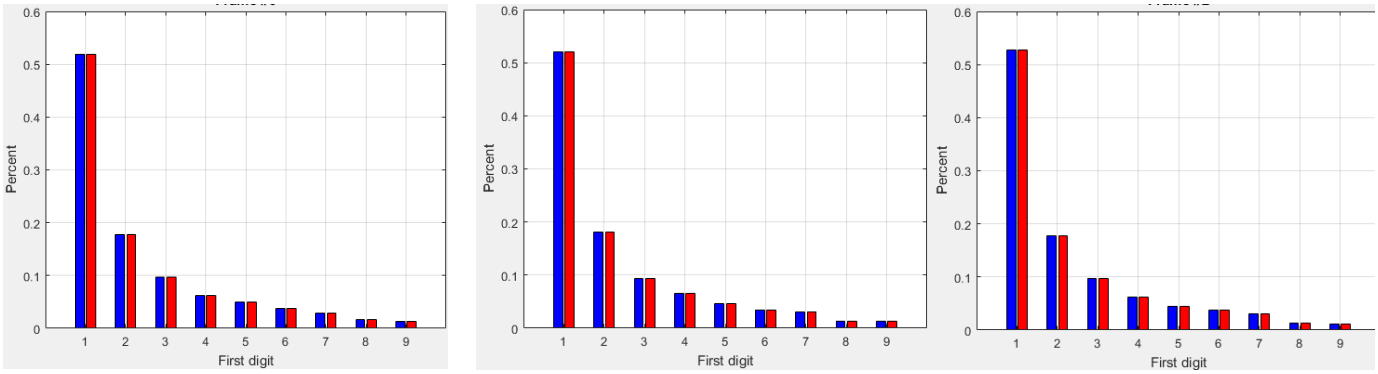
Ilustración XXXI: Akiyo CBR menor a 10.000



Frame I Frame P Frame B

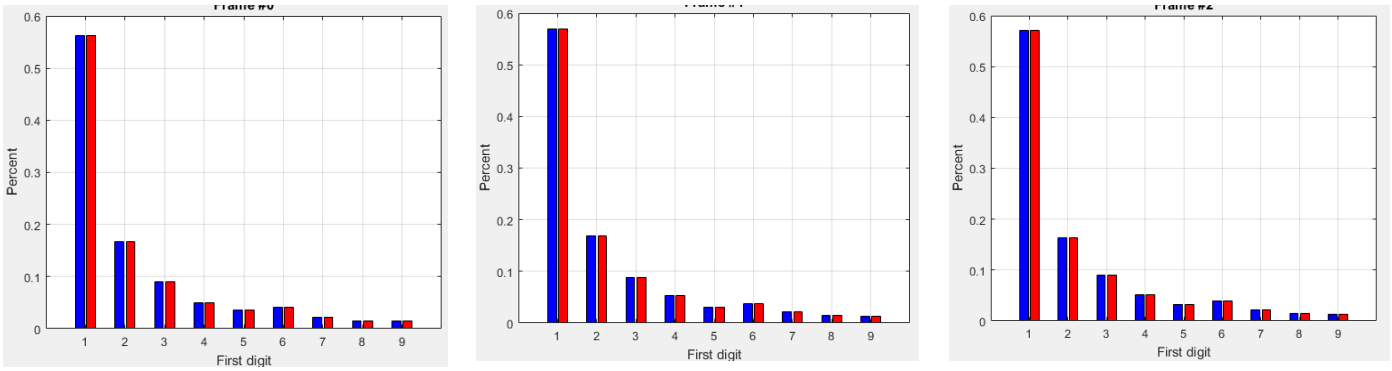
Ilustración XXXII: Football CBR menor a 10.000

Vamos a probar ahora con tasa de bits mayor a 10.000.



Frame I Frame P Frame B

Ilustración XXXIII: Akiyo CBR mayor a 10.000



Frame I Frame P Frame B

Ilustración XXXIV: Football CBR mayor a 10.000

Para el caso donde la tasa de bits objetivo para la recompresión es mayor a 10.000 bits / s, como se demuestra en la ilustración treinta y tres y treinta y cuatro, la complejidad del movimiento parece tener una influencia en la distribución del primer dígito. Para secuencia de video con movimiento rápido en marcos vecinos como el fútbol, es obvio que las distribuciones de primer dígito en los cuadros I, P y B no están modelados por la ley logarítmica. Para secuencia de video con cámara lenta en marcos vecinos como Akiyo, la violación de la ley logarítmica no es tan visualmente obvia.

Para el caso donde la tasa de bits objetivo para la recompresión es menor que 10.000 bits/s, se muestran las distribuciones del primer dígito de videos doblemente comprimidos MPEG en la treinta y uno y treinta y dos. Aunque parece que los primeros dígitos tienen tendencia a seguir la ley logarítmica, una mirada más de cerca a la figura puede encontrar la anormalidad en cierta dígitos, por ejemplo, las probabilidades de los

dígitos 8 y 9 en el fútbol están muy cerca a ceros, y las probabilidades de los dígitos 7 y 8 son casi iguales en Akiyo.

4.3.3.5. Test Chi-cuadrado

La prueba de chi-cuadrado es una forma más formal de llevar a cabo la comparación anterior. En la prueba de chi-cuadrado, consideramos la hipótesis nula de que los primeros dígitos de los coeficientes de CA cuantificados MPEG no nulos en cada trama siguen la ley logarítmica. Aquí podemos ver la formula chi-cuadrado.

$$\chi^2 = \sum_i \frac{(\text{observada}_i - \text{teorica}_i)^2}{\text{teorica}_i}$$

Ilustración XXXV: Chi-cuadrado

El nivel significativo se establece en el 1% y el grado de libertad está configurado para ser 8. Las entradas en la Tabla 1 representan cuántos porcentajes de marcos no rechazan la hipótesis nula. Para cada índice de inteligencia o tasa de bits objetivo, hay un total de 164 marcos I, 613 marcos P y 1523 marcos B en el original o clips de vídeo doblemente comprimidos MPEG, respectivamente.

	I		P		B	
	Originally	**Doubly**	**Originally**	**Doubly**	**Originally**	**Doubly**
VBR	0.57	0.09	0.58	0.10	0.6	0.12
CBR mín.	0.53	0.13	0.50	0.19	0.54	0.20
CBR máx.	0.50	0.21	0.48	0.20	0.49	0.19

Tabla I: Porcentaje de frames que siguen la ley logarítmica

En el caso de VBR, aproximadamente el 57% de los frames I, el 58% de los frames P y el 60% de los frames B, los fotogramas originalmente comprimidos en MPEG no rechazan la hipótesis nula. Sin embargo, solo aproximadamente el 9% de los cuadros I, el 10% de los cuadros P y el 12% de los cuadros B en videos doblemente comprimidos MPEG no rechazan la hipótesis nula.

En el caso de CBR min, aproximadamente el 53% de los frames I, el 50% de los frames P y el 54% de B, los fotogramas originalmente comprimidos en MPEG no rechazan la hipótesis nula. Sin embargo, solo alrededor del 13% de los cuadros I, el

19% de los cuadros P y el 20% de los cuadros B en el video comprimido MPEG doble no rechaza la hipótesis nula.

En el caso de CBR máx., aproximadamente el 50% de los frames I, el 48% de los frames P y el 49% de frames, los fotogramas originalmente comprimidos en MPEG no rechazan la hipótesis nula. Sin embargo, solo el 21% de los fotogramas I, el 20% de los fotogramas P y el 19% de los fotogramas B en videos doblemente comprimidos MPEG no rechazan la hipótesis nula.

Los resultados de la prueba de chi-cuadrado indican que la mayoría de los frames I, P y B en los videos MPEG originales no rechazan la hipótesis nula, pero solo unos pocos frames I, P y B en videos doblemente comprimidos MPEG no rechazan la nula hipótesis, que es consistente con la comparación visual.

4.3.3.6. Detección de la doble compresión en MPEG

De la sección anterior extraemos que la distribución del primer digito en videos MPEG doblemente comprimidos no siguen la ley de Benford (tal y como hemos visto los videos originales si la cumplen). La alteración en la distribución del primer digito es un buen indicador para detectar la doble compresión, es decir, que el video se encuentra manipulado.

Por comparativa visual se puede detectar que un video esta doblemente comprimido examinando la distribución de los primeros dígitos. Existen casos en los cuales detectar que un video está manipulado no es seguro por los análisis que se han realizado en la sección anterior. Para detectar la doble compresión en los videos hemos considerado implementar un Support Vector Machines (SVM). Es una algoritmo de aprendizaje supervisado que construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad alta que puede ser utilizado en problemas de clasificación o regresión.

Como elemento de detección de doble compresión se va a utilizar el GOP, con esto podemos detectar si hay una alteración estadística que pueda aparecer en los frames I, P o B.

Para la obtención de los GOP de cada vídeo se han usado dos programas por un lado, se ha usado para compilarlo desde Matlab Ffmpeg, es un programa en el que introduces un video y obtienes un fichero con todo el background del video como vemos en la siguiente ilustración:

```
[FRAME]
media_type=video
stream_index=0
key_frame=1
pkt_pts=48600
pkt_pts_time=0.540000
pkt_dts=48600
pkt_dts_time=0.540000
best_effort_timestamp=48600
best_effort_timestamp_time=0.540000
pkt_duration=3600
pkt_duration_time=0.040000
pkt_pos=27
pkt_size=6653
width=256
height=256
pix_fmt=yuv420p
sample_aspect_ratio=1:1
pict_type=I
coded_picture_number=0
display_picture_number=0
interlaced_frame=0
top_field_first=0
repeat_pict=0
color_range=tv
color_space=unknown
color_primaries=unknown
color_transfer=unknown
chroma_location=center
[SIDE_DATA]
side_data_type=AVPanScan
[/SIDE_DATA]
[SIDE_DATA]
```

Ilustración XXXVI: Ejemplo obtención tipo de frame

Por otro lado hemos usado VirtualDub como hemos explicado en el apartado anterior para compararlo con los tipos de frame obtenidos con Ffmpeg y cerciorarnos de que eran correctos.

La detección de la doble compresión de cada GOP es el punto clave para una clasificación correcta de un vídeo MPEG doblemente comprimido o no.

Para simplificar los tiempos de entrenamiento, reducir el sobreentrenamiento y simplificar el modelo para ser más fácilmente interpretado por otras personas se va a realizar una selección de features. Las variables que se van a introducir son las siguientes:

- Para el frame I se va a incluir la distribución de los primeros dígitos de los coeficientes cuantificados AC y tres pruebas de bondad de ajuste (RSE, RMSE y el Rsquare).
- Para los frames P y B se va a incluir la media de la distribución de los primeros dígitos de los coeficientes cuantificados AC , y tres pruebas de bondad de ajuste (RSE, RMSE y el Rsquare) tanto de los frames P como B.

La dimensión de los datos de entrada va a ser de 36, es decir, 12 elementos por cada tipo de frame. Las tres pruebas de bondad que se van a realizar son las siguientes:

$$SSE = \sum_{i=1}^{n} \frac{(p_i - \hat{p}_i)^2}{p_i}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (p_i - \hat{p}_i)^2}{n}}$$

$$R - square = 1 - \frac{\sum_{i=1}^{n} \frac{(p_i - \hat{p}_i)^2}{p_i}}{\sum_{i=1}^{n} \frac{(p_i - \bar{p}_i)^2}{p_i}}$$

Ilustración XXXVII: SSE, RMSE y R-square

Si el SSE tiene un valor cercano a cero indicara que el modelo tiene un menor componente de error aleatorio y por lo tanto será más útil para poder predecir.

Para el RMSE debemos saber que es la raíz cuadrada del MSE y si este tiene un valor cercano de cero indica que es un modelo más útil para predecir.

R-square puede tomar cualquier valor entre cero y uno, este valor estadístico mide cual bueno es el modelo en relación a la variación de los datos.

4.3.4. Aplicación del algoritmo

4.3.4.1. Elección del algoritmo

El problema que nos atañe, es un problema de clasificación, ya que queremos clasificar cada vídeo como originalmente comprimido MPEG o comprimido doblemente MPEG.

El algoritmo que vamos a usar es Support Vector Machine (SVM) ya que tiene una buena performance en clasificación de imágenes y vídeos.

Dado un conjunto de ejemplos de entrenamiento, cada uno marcado como perteneciente a una u otra de las dos categorías, un algoritmo de entrenamiento SVM construye un modelo que asigna nuevos ejemplos a una u otra categoría, convirtiéndolo en un clasificador lineal binario no probabilístico. Un modelo SVM es una representación de los ejemplos como puntos en el espacio, mapeados de manera que los ejemplos de las categorías separadas estén divididos por un espacio libre lo más ancho posible. A continuación, se mapean nuevos ejemplos en ese mismo espacio y se predice que pertenecen a una categoría según el lado del espacio en el que caen.

Un algoritmo de entrenamiento de SVM encuentra el clasificador representado por el vector normal **w** y sesgo **b** del hiperplano. Este hiperplano (límite) separa las diferentes clases por un margen tan amplio como sea posible. El problema se puede convertir en un problema de optimización restringida:

$$\text{Minimizar } \mathbf{w}\ ||\mathbf{w}|| \text{ sujeto a } \mathbf{Y_i\ (w^T\ X_i - b) \geq 1,\ i = 1, \ldots ,\ n}.$$

Un algoritmo de entrenamiento SVM encuentra el clasificador representado por el vector normal y el sesgo del hiperplano. Este hiperplano (límite) separa las diferentes clases por un margen tan amplio como sea posible, como podemos ver la siguiente ilustración.

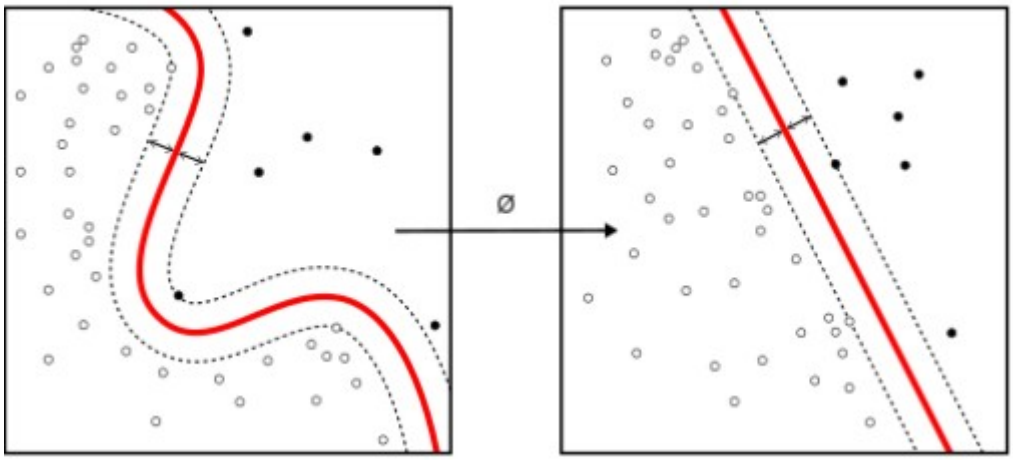


Ilustración XXXVIII: SVM

4.3.4.2. Implementación del algoritmo

Para la implementación del algoritmo Support Vector Machine (SVM) se va a utilizar el software: Matlab R2017a. Matlab es una potente herramienta que incluye en el paquete "Statistics and Machine Learning Toolbox" un conjunto de funciones y aplicaciones que serán de gran utilidad en este apartado.

En lo que se refiere a Support Vector Machine (SVM), Matlab proporciona la función `fitcsvm`. Esta función sirve para entrenar un modelo para dos clases (binario) de Support Vector Machine (SVM). También se puede utilizar para realizar validación cruzada (la cual será utilizada en este trabajo para entrenar al modelo).

La validación cruzada será utilizada para poder determinar si un modelo predice mejor que otro. En este caso, se va a realizar una validación cruzada de K iteraciones. El valor que suele tener K es de 10, que va a ser el valor que utilizaremos en esta implementación.

De forma general, la función objetivo es una función que trata de maximizar o minimizar un valor determinado. En el caso de Support Vector Machine el objetivo es minimizar el error de generalización (Generalization error, también conocido como "out-of-sample error"). El error de generalización es una medida usada para evaluar si un modelo predice valores de salida de forma correcta de datos que no ha visto anteriormente, y disminuye si se evita el sobreentrenamiento en el modelo. Por lo tanto, para elegir qué modelo predice mejor se escogerá el que tenga menor error de generalización, es decir, el que tenga un menor valor de la función objetivo.

Si obtenemos un modelo con un error de generalización de 0.05 (5%) significa que el modelo, para datos que no ha visto anteriormente, predice de forma errónea el 5% de esos datos de entrada.

En el caso de que queramos saber si con nuestro modelo un video está manipulado o no, los datos de entrada que necesita nuestro Support Vector Machine serán el conjunto de GOP que forma ese video. Por ejemplo, si introducimos un video formado por 5 GOP, el modelo predecirá si el GOP está manipulado o no. Si el video tiene un gran número de GOP manipulados, se sabrá con certeza que el video está manipulado.

El modelo de Support Vector Machine va a tener dos clases:

- ORIGINAL. Esta clase será utilizada cuando un GOP sea identificado como original. Es decir, que no ha sufrido edición o manipulación del mismo.
- TAMPERING. Esta clase indicará que el GOP ha sido manipulado.

```
>> SVMModel.ClassNames

ans =

  2×1 cell array

    'ORIGINAL'
    'TAMPERING'
```

Ilustración XXXIX: Nombre de las clases del SVM

La función 'fitcsvm' necesita recibir como parámetros dos variables. La primera a la que hemos llamado X, que contendrá las matrices de coeficientes de los GOP de cada vídeo y la segunda a la que hemos llamado Y, que contendrá las etiquetas de los GOP de cada vídeo. Estas etiquetas serán "original" o "tampering" dependiendo si el GOP pertenece a un vídeo que ha sido manipulado o no.

```
SVMModel = fitcsvm(X,Y,'Standardize',true,'KernelFunction','RBF',...
    'KernelScale','auto');
```

Ilustración XLI: SVM función Matlab

La imagen que mostramos arriba es nuestra función 'fitcsvm' con los parámetros que vamos a entrenar.

A continuación aplicaremos la validación cruzada, usando la siguiente función que vemos a continuación.

```
CVSVMModel = crossval(SVMModel);
```

Ilustración XLII: SVM validación cruzada

Para saber cuál es el error de generalización de nuestro modelo haremos uso de la función 'kfoldLoss'. En cada iteración, se mostrará en la tercera columna como la función objetivo minimiza el error de generalización. El modelo se quedará con la iteración que tenga un menor error de generalización.

4.3.4.3. Optimización de los híper parámetros del algoritmo

1) Función de optimización lower-confidence-bound:

Vamos a comenzar usando la función de optimización de los híper parámetros del SVM, para poder entrenar el algoritmo mejor.

Los híper parámetros son parámetros cuyos valores son asignados antes de comenzar el proceso de aprendizaje. La función de optimización que vamos a usar es lower-confidence-bound.

A continuación vamos a analizar los resultados obtenidos.

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	BoxConstraint	KernelScale
1	Best	0.41114	0.097717	0.41114	0.41114	0.0045943	444.59
2	Accept	0.41114	0.14752	0.41114	0.41114	5.3686	496.94
3	Accept	0.41114	0.11828	0.41114	0.41114	0.046955	1.6411
4	Best	0.32891	0.1275	0.32891	0.33959	11.915	0.59373
5	Best	0.31565	0.14639	0.31565	0.31881	12.966	0.44959
6	Best	0.28912	0.15105	0.28912	0.28913	19.787	0.1795
7	Accept	0.36074	28.849	0.28912	0.28913	967.88	0.0063535
8	Best	0.27321	2.3438	0.27321	0.27323	958.48	0.14885
9	Best	0.25995	13.445	0.25995	0.25997	988.57	0.062905
10	Accept	0.2626	18.598	0.25995	0.25993	927.38	0.03855
11	Accept	0.40584	0.12824	0.25995	0.25991	0.01001	0.036489
12	Accept	0.27586	7.0183	0.25995	0.25994	167.96	0.051323
13	Accept	0.2626	16.891	0.25995	0.26087	994.51	0.055231
14	Accept	0.26525	10.309	0.25995	0.26192	979.93	0.064064
15	Accept	0.2626	18.317	0.25995	0.26209	998.78	0.04862
16	Accept	0.2626	18.905	0.25995	0.26208	998.04	0.051804
17	Accept	0.2626	18.623	0.25995	0.26217	995.93	0.048674
18	Accept	0.25995	20.551	0.25995	0.26174	966.52	0.045636
19	Accept	0.25995	20.655	0.25995	0.26147	987.47	0.046757
20	Accept	0.25995	17.206	0.25995	0.26127	983.51	0.043133
21	Accept	0.25995	21.034	0.25995	0.26104	994.64	0.046048
22	Accept	0.25995	17.941	0.25995	0.26089	978.05	0.04316
23	Accept	0.25995	18.64	0.25995	0.26078	967.35	0.044985
24	Accept	0.2626	18.622	0.25995	0.26089	985.18	0.039665
25	Accept	0.25995	20.645	0.25995	0.26081	999.65	0.047663
26	Accept	0.26525	18.54	0.25995	0.26114	989.78	0.048316
27	Accept	0.25995	18.1	0.25995	0.26104	967.28	0.044847
28	Accept	0.25995	18.407	0.25995	0.26097	998.86	0.040801
29	Accept	0.25995	16.856	0.25995	0.26091	990.38	0.043118
30	Accept	0.25995	17.017	0.25995	0.26085	971.48	0.042609

Ilustración XLIII: SVM iteraciones

En la tabla anterior podemos ver el número total de iteraciones que realiza el algoritmo, que son un total de treinta. En la evaluación de resultados podemos observar como cataloga con la etiqueta de "best" aquellos que considera mejores. En la siguiente columna podemos ver la función objetivo correspondiente a cada iteración. El objetive runtime es el tiempo que tarda en realizar cada iteración. El BestSoFar es el mejor valor de la función objetivo que ha encontrado hasta ahora, con las iteraciones realizadas. El Box Constraint es un parámetro que controla la penalización máxima impuesta en las observaciones que exceden los márgenes y ayuda a prevenir el sobreajuste (regularización). El kernelScale divide todos los elementos de la matriz de predicción X por el valor de KernelScale. Luego, el software aplica la norma de Kernel apropiada para calcular la matriz de Gram.

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 444.7405 seconds.
Total objective function evaluation time: 398.4281

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    988.57           0.062905

Observed objective function value = 0.25995
Estimated objective function value = 0.26085
Function evaluation time = 13.4449

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    994.64           0.046048

Estimated objective function value = 0.26085
Estimated function evaluation time = 19.0712
```

Ilustración XLIV: SVM resultados

En este resumen final, podemos ver el mejor resultado obtenido de las treinta iteraciones que se han realizado, tenemos el mejor valor de la función objetivo que sería 0.26085, como para el BoxConstraint y el kernelScale. Además del tiempo total que tarda el algoritmo en ejecutarse.

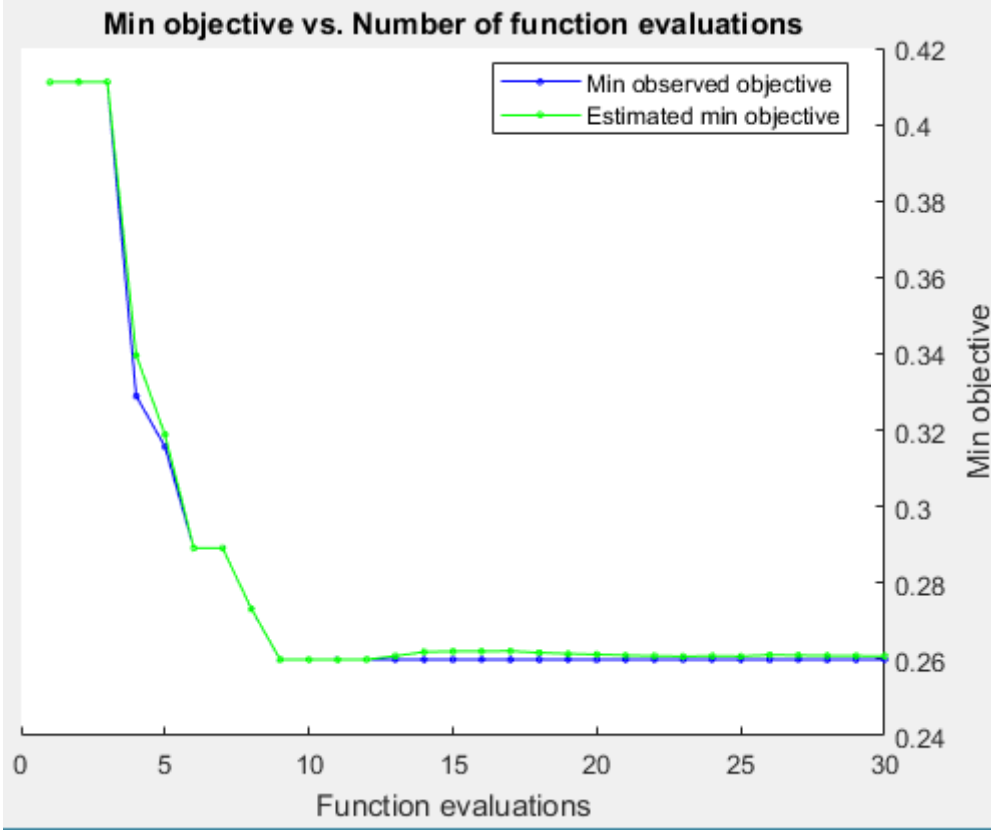


Ilustración XLV: SVM min. objetive vs number of function evaluations

En este gráfico podemos observar el número total de iteraciones evaluadas, que han sido treinta y el número mínimo observado. Podemos ver como se representa con una línea de puntos verde el mínimo objetivo estimado y con una línea de puntos azul el mínimo objetivo observado. En la gráfica podemos analizar como en las primera iteraciones el número mínimo es el más alto del gráfico y como a partir de la iteración nueve disminuye, al más bajo de la gráfica.

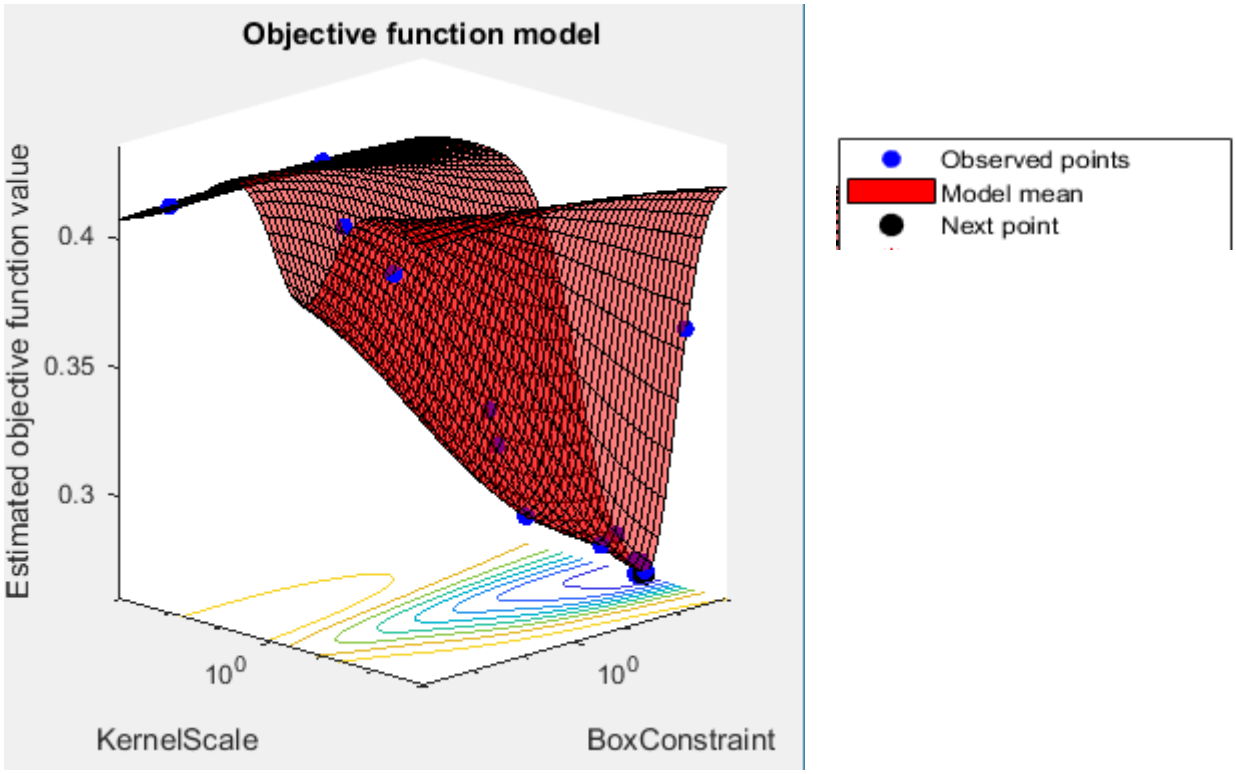


Ilustración XLVI: SVM grafico

En esta gráfico vemos dibujada nuestra función objetivo y como se adapta a los datos. Tenemos en el eje x el kernelScale: el software divide todos los elementos de la matriz de predicción X por el valor de KernelScale. Luego, el software aplica la norma de Kernel apropiada para calcular la matriz de Gram.

En el eje Z tenemos Box Constraint, que es un parámetro que controla la penalización máxima impuesta en las observaciones que exceden los márgenes y ayuda a prevenir el sobreajuste (regularización).

Si aumenta la restricción de cuadro, el clasificador SVM asigna menos vectores de soporte. Sin embargo, aumentar la restricción de caja puede conducir a tiempos de entrenamiento más largos.

En el eje Y tenemos el valor estimado de la función objetivo.

Si analizamos el gráfico podemos ver la evolución de la función objetivo estimada del modelo. Podemos ver donde se sitúan los puntos azules que son los puntos observados y donde se situarían los puntos negros que son los puntos siguientes.

2) Función de optimización expected-improvement-plus

Vamos a usar otra función de optimización para comprobar si podemos mejorar los datos obtenidos con la anterior. La función de optimización que vamos a usar es la **expected-improvement-plus**.

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	BoxConstraint	KernelScale
1	Best	0.41114	0.64049	0.41114	0.41114	0.0045943	444.59
2	Accept	0.41114	0.18161	0.41114	0.41114	5.3686	496.94
3	Accept	0.41114	0.10926	0.41114	0.41114	0.046955	1.6411
4	Best	0.32891	0.10811	0.32891	0.33959	11.915	0.59373
5	Accept	0.34218	0.14583	0.32891	0.33669	11.658	0.61194
6	Best	0.31565	0.13545	0.31565	0.31681	13.209	0.48393
7	Accept	0.41114	0.13032	0.31565	0.32963	14.424	872.78
8	Best	0.27586	0.202	0.27586	0.27609	27.121	0.18661
9	Accept	0.41645	0.15509	0.27586	0.27773	0.0076063	0.055576
10	Accept	0.27586	0.24707	0.27586	0.27503	99.953	0.17591
11	Accept	0.41114	0.09148	0.27586	0.27604	0.0012204	0.24008
12	Best	0.2679	0.33381	0.2679	0.26884	66.142	0.10815
13	Best	0.25995	0.90328	0.25995	0.26148	56.074	0.057598
14	Best	0.25729	11.921	0.25729	0.25957	130.47	0.023178
15	Accept	0.27321	5.4595	0.25729	0.26072	38.841	0.016739
16	Accept	0.26525	21.584	0.25729	0.26008	999.42	0.031838
17	Accept	0.27056	6.5712	0.25729	0.26137	178.9	0.042012
18	Accept	0.34483	28.749	0.25729	0.26068	997.95	0.0027825
19	Accept	0.28117	4.7852	0.25729	0.26446	102.74	0.044161
20	Accept	0.27056	6.4869	0.25729	0.26488	536.43	0.074761
21	Accept	0.28382	0.23591	0.25729	0.26625	0.0010225	0.0010073
22	Accept	0.27586	5.9449	0.25729	0.26627	0.085255	0.0010076
23	Accept	0.2626	19.181	0.25729	0.26453	244.57	0.026
24	Accept	0.2626	17.929	0.25729	0.26372	331.5	0.022956
25	Best	0.25464	21.407	0.25464	0.26088	347.77	0.020753
26	Accept	0.26525	24.061	0.25464	0.26144	651.65	0.018152
27	Accept	0.41114	0.091213	0.25464	0.2615	988.21	28.274
28	Accept	0.41114	0.095598	0.25464	0.26158	0.0010044	23.552
29	Accept	0.41114	0.10206	0.25464	0.26167	975.97	280.31
30	Accept	0.3687	0.11596	0.25464	0.26178	0.0010038	0.0068816

Ilustración XLVII: SVM iteraciones Optimización II

En la tabla anterior podemos ver los mismos datos que en la tabla que teníamos para la anterior función de optimización. Vamos a analizar los resultados ahora:

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 214.6225 seconds.
Total objective function evaluation time: 178.1033

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    347.77           0.020753

Observed objective function value = 0.25464
Estimated objective function value = 0.26178
Function evaluation time = 21.4067

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    331.5            0.022956

Estimated objective function value = 0.26178
Estimated function evaluation time = 20.3834
```

Ilustración XLVIII: SVM resultados Optimización II

En este resumen final, podemos ver como los resultados han empeorado comparado con la función usada anteriormente. En el Best observed podemos ver como el Box Constraint ha disminuido notablemente de 988.57 a 347.77, el kernelScale tambien ha disminuido de 0.062 a 0.020, y vemos que el valor de la función objetivo observada es casi parecido en el anterior un 0.0.259 y hemos mejorado a 0.254.

En cuanto a los datos estimados, ha pasado igual, hemos disminuido en Box Constraint y en kernelScale, y aumentado el valor de la función objetivo.

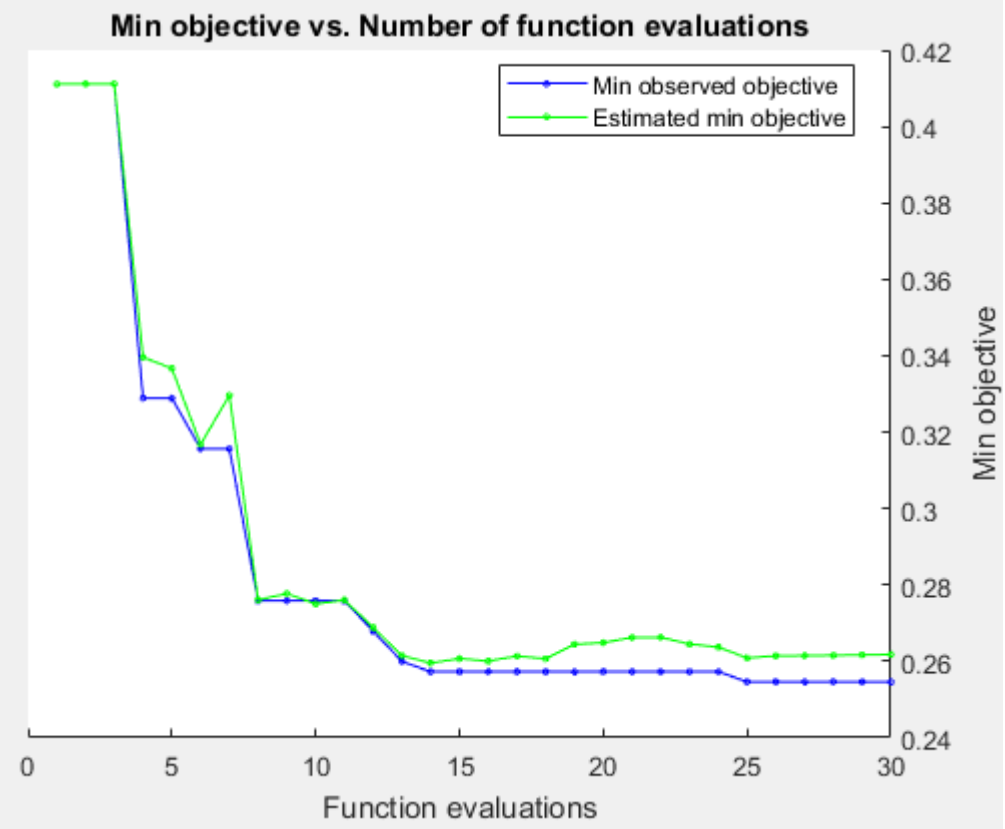


Ilustración XLIX: SVM min. objetive vs number of function evaluations Optimización II

En este gráfico podemos observar el número total de iteraciones evaluadas, que han sido treinta y el número mínimo observado. Podemos ver como se representa con una línea de puntos verde el mínimo objetivo estimado y con una línea de puntos azul el mínimo objetivo observado. En la gráfica podemos analizar como en las primera iteraciones el número mínimo es el más alto del gráfico y como a partir de la iteración seis se mantiene constante tanto el número mínimo observado como el número mínimo estimado.

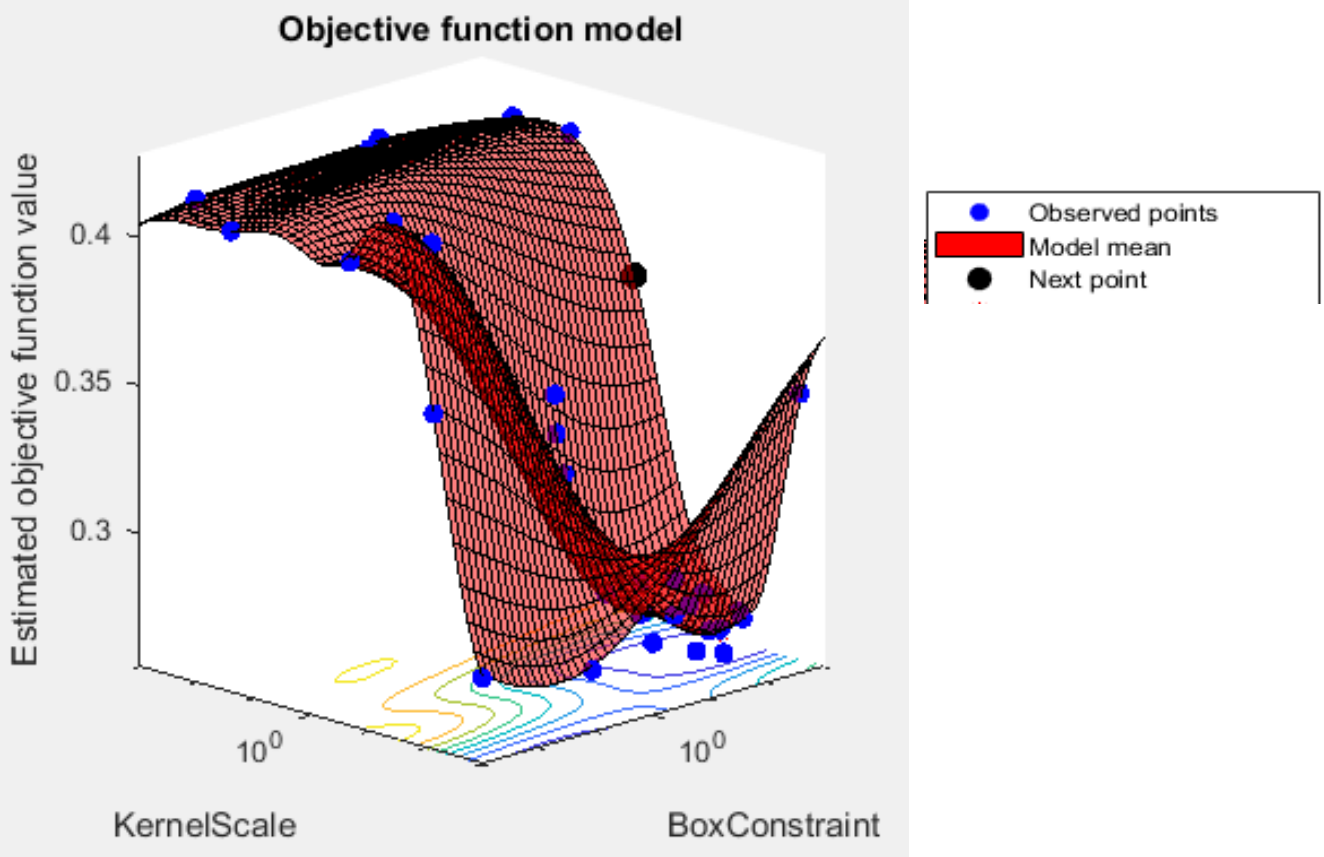


Ilustración L: SVM grafico Optimización II

En esta gráfico podemos ver la evolución de la función objetivo estimada del modelo. Podemos ver donde se sitúan los puntos azules que son los puntos observados y donde se situarían los puntos negros que son los puntos siguientes.

3) Elección de la función de optimización de los hiperparametros

Nos vamos a quedar con aquella función de optimización con la que obtengamos una función objetivo más baja, y un Box Constraint más bajo, en este caso con expected-improvement-plus.

4.3.4.4. Entrenamiento del algoritmo

Vamos a ver los parámetros que podemos entrenar dentro de nuestro SVM:

1) Función Kernel:

   Vamos a entrenar el algoritmo con distintas funciones de Kernel.

Kernel Function Name	Description	Formula
'gaussian' or 'rbf'	Gaussian or Radial Basis Function (RBF) kernel, default for one-class learning	$G(x_j,x_k) = \exp(-\|x_j - x_k\|^2)$
'linear'	Linear kernel, default for two-class learning	$G(x_j,x_k) = x_j'x_k$
'polynomial'	Polynomial kernel. Use 'PolynomialOrder',q to specify a polynomial kernel of order q.	$G(x_j,x_k) = (1 + x_j'x_k)^q$

Ilustración LI: SVM funciones Kernel

- Kernel polinomial

Vamos a comenzar con un kernel polinomial de orden dos. Vamos a ver los resultados:

```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 132.8121 seconds.
Total objective function evaluation time: 107.7814

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    0.022282         0.80965

Observed objective function value = 0.16446
Estimated objective function value = 0.16718
Function evaluation time = 0.14253

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    0.020828         0.80248

Estimated objective function value = 0.16718
Estimated function evaluation time = 0.14763
```

Ilustración LII: SVM Kernel polinomial resumen

Si analizamos los resultados obtenidos podemos ver como el valor de la función objetivo ha disminuido comparado con la optimización de los parametros. Vamos a ver ahora las otras dos gráficas obtenidas:

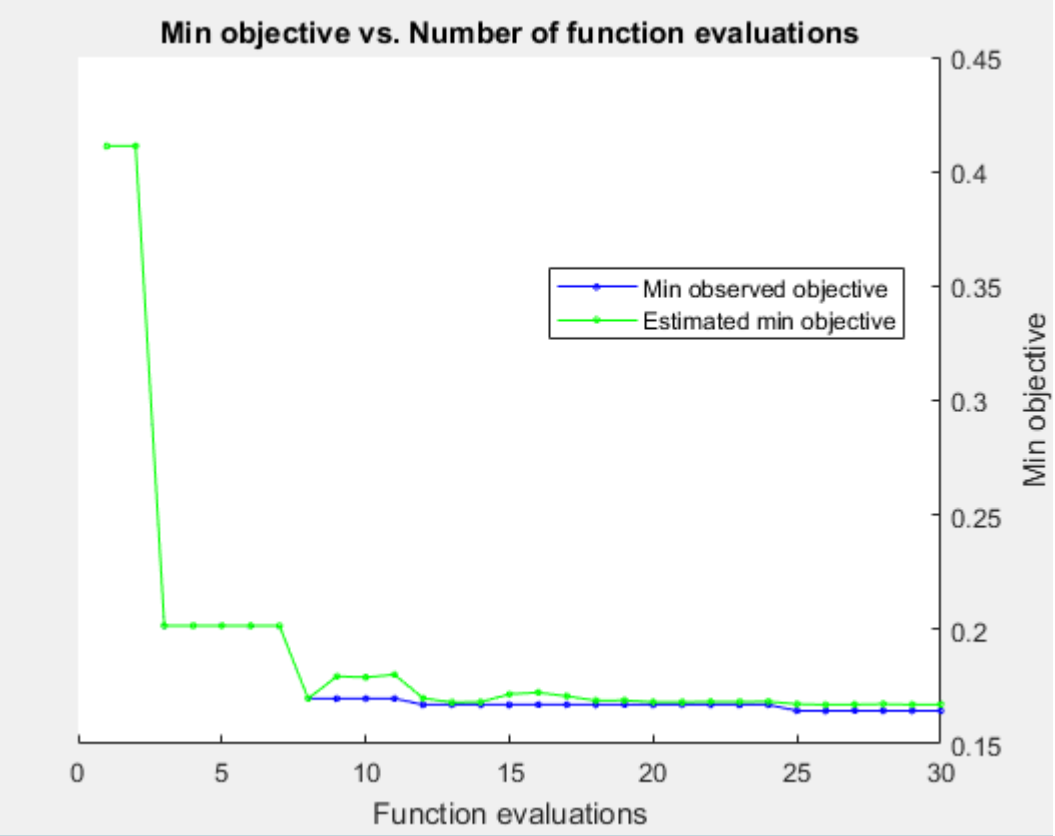


Ilustración LIII: SVM Kernel polinomial min y iteraciones

Vemos como se ha reducido muchísimo el mínimo objetivo, aplicando la optimización de los hiperparametros y la función kernel polinomial.

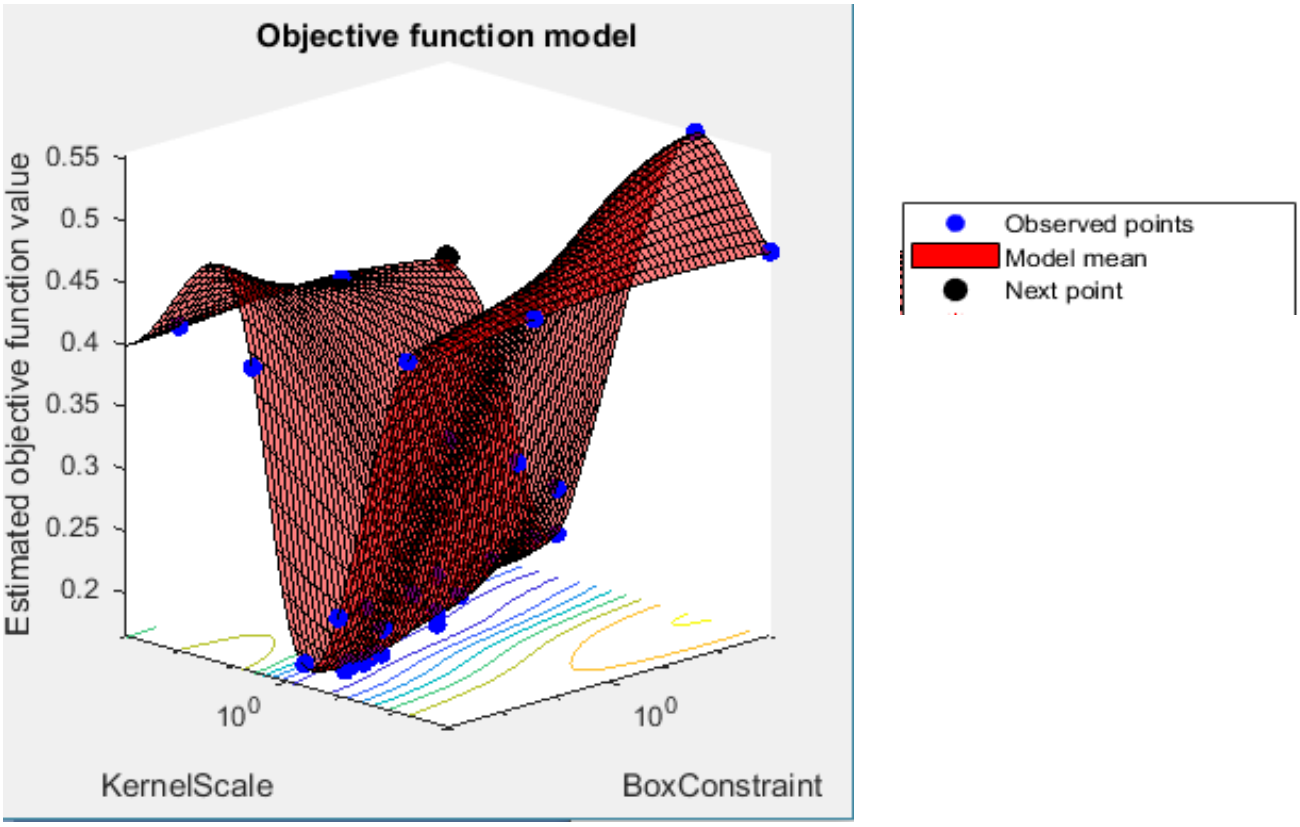


Ilustración LIV: SVM Kernel polinomial

Vemos como en esta gráfica el algoritmo se adapta mejor a los valores.

- Kernel lineal

Vamos a comprobar los resultados ahora con el Kernel lineal y vamos a analizar los resultados.

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 380.995 seconds.
Total objective function evaluation time: 350.702

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    0.0010124        0.0036076

Observed objective function value = 0.25199
Estimated objective function value = 0.2553
Function evaluation time = 12.4226

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    0.0010091        0.0021498

Estimated objective function value = 0.2553
Estimated function evaluation time = 17.8172
```

Ilustración LV: SVM Kernel linear resumen

Si comparamos los datos del kernel polinomial con los del kernel linear podemos ver como con el kernel polinomial tenemos un menor valor de la función objetivo, por lo que de momento el modelo con kernel polinomial de grado dos es mejor que el kernel linear.

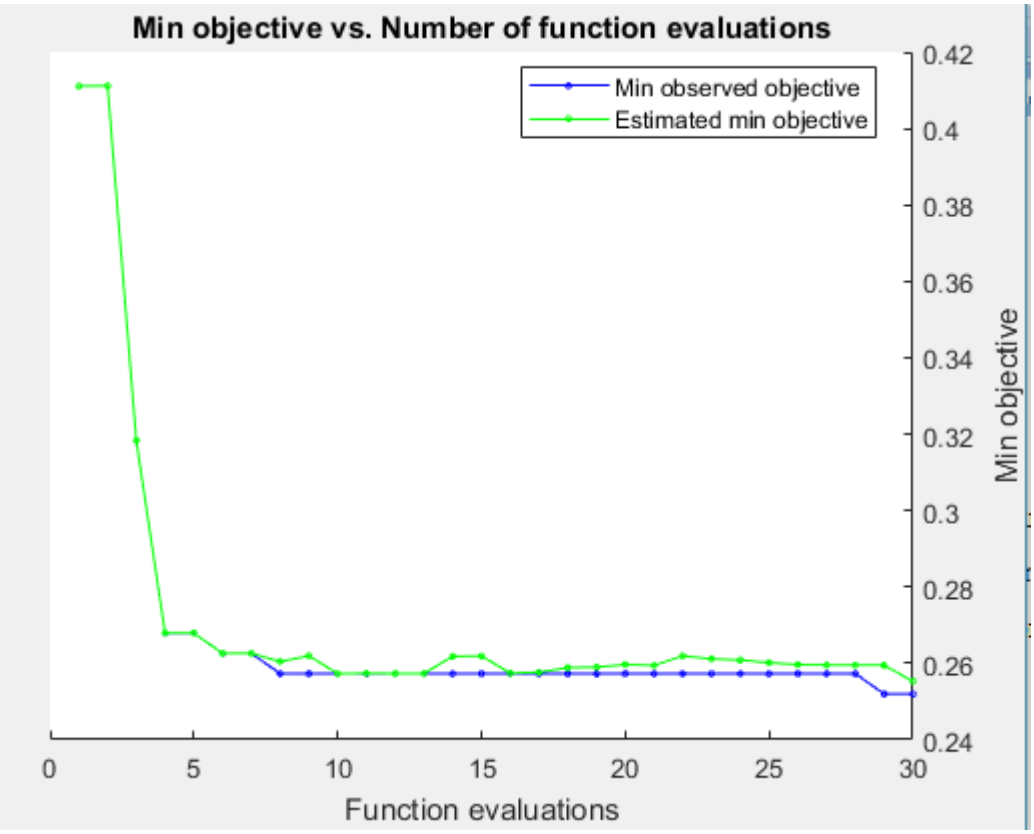


Ilustración LVI: SVM Kernel linear min e iteraciones

En esta gráfica podemos observar cómo ha aumentado el número mínimo objetivo, comparado con el kernel polinomial.

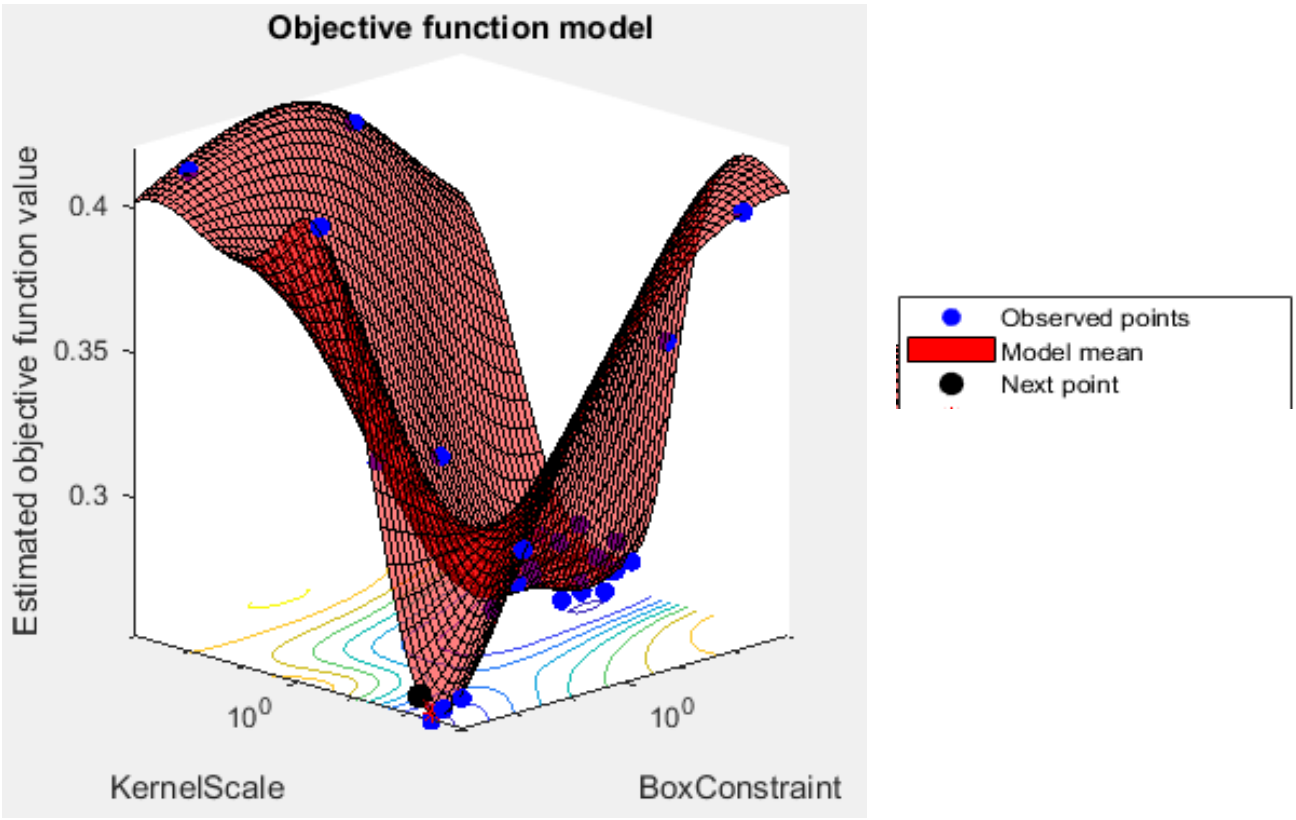


Ilustración LVII: SVM Kernel linear

Vemos como en esta gráfica el algoritmo se adapta mejor a los valores.

- Kernel RBF

  Por último, vamos a usar un Kernel RBF y analizar los resultados.

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 26.9339 seconds.
Total objective function evaluation time: 3.708

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    48.82            6.0292

Observed objective function value = 0.17241
Estimated objective function value = 0.17482
Function evaluation time = 0.12217

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    919.11           1.6205

Estimated objective function value = 0.17482
Estimated function evaluation time = 0.13172
```

Ilustración LVIII: SVM Kernel RBF resumen

Vemos como los valores para las funciones objetivos son mayores usando un kernel RBF que usando un kernel polinomial.

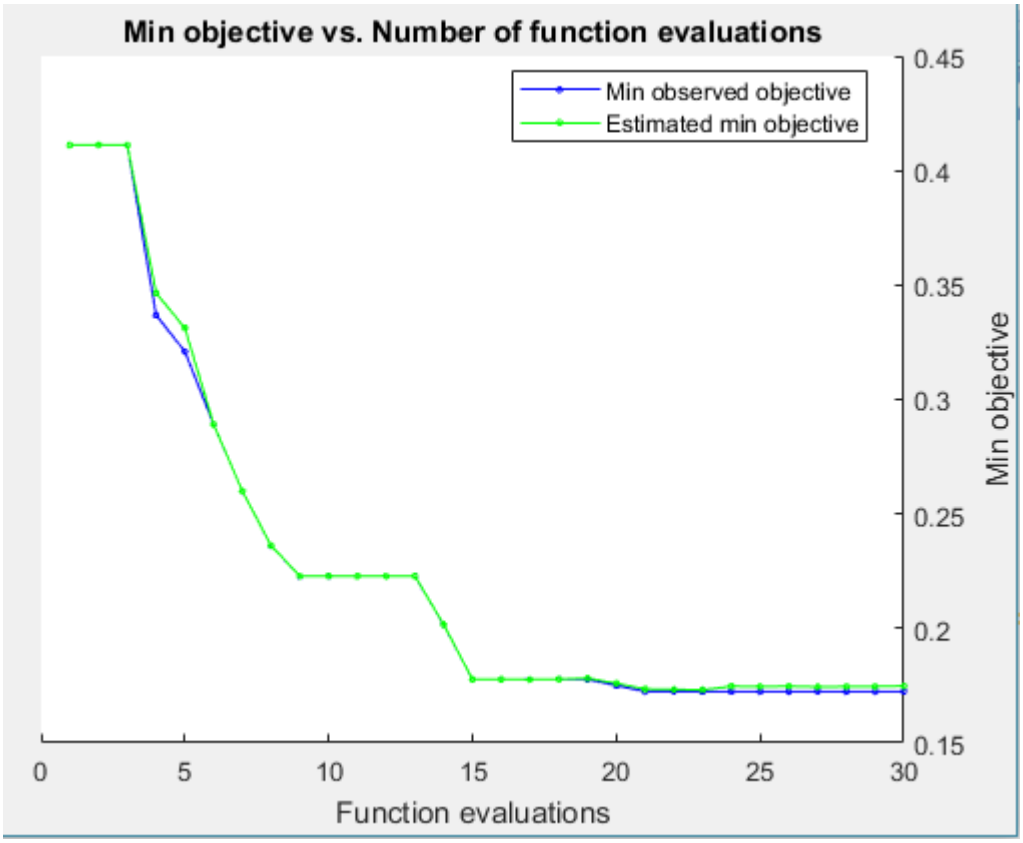


Ilustración LIX: SVM Kernel RBF min e iteraciones

En esta gráfica podemos observar como el número mínimo objetivo se mantiene igual que en el kernel polinomial.

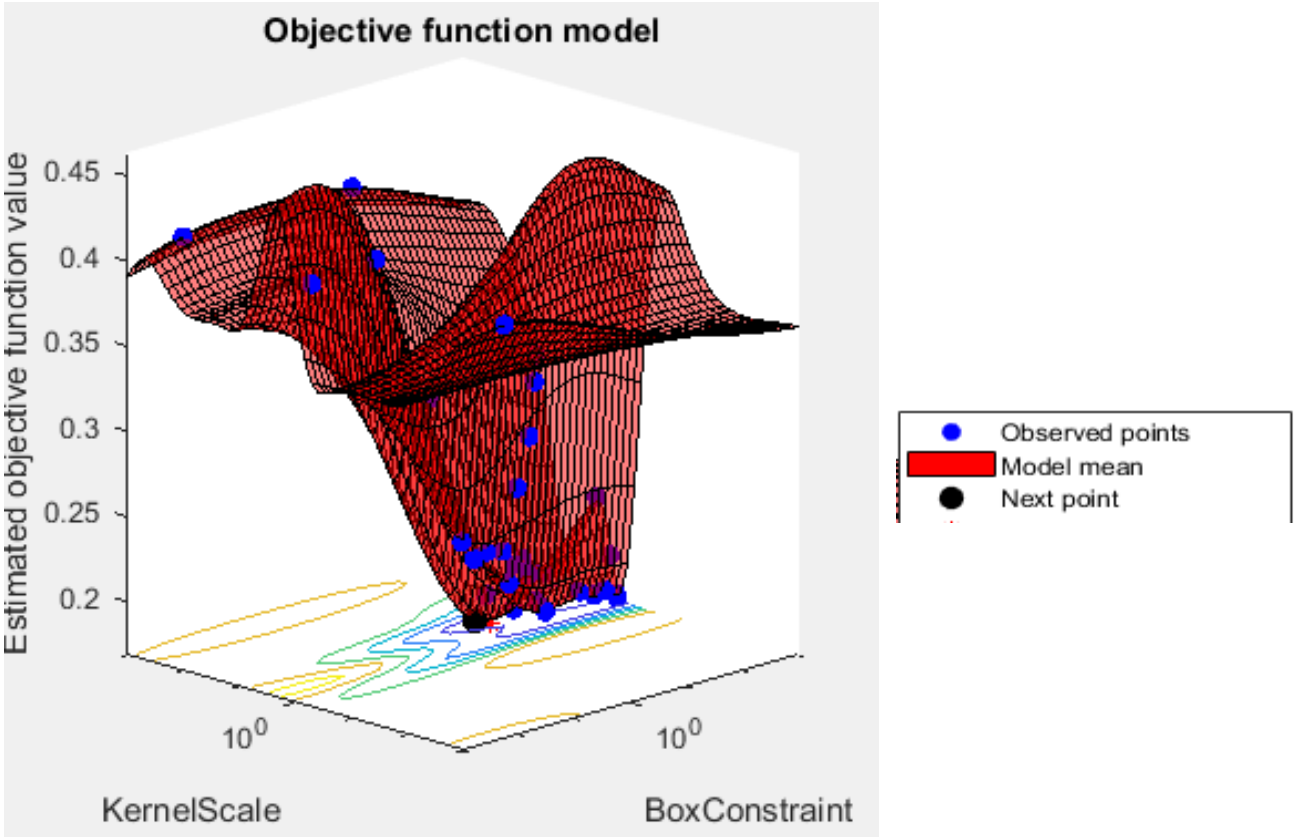


Ilustración LX: SVM Kernel RBF

En esta grafica podemos ver como se adapta el algoritmo a los valores de entrenamiento.

Dado que los resultados obtenidos el mejor kernel es el polinomial de grado dos ya que el valor de su función objetivo es el menor de todos.

2) La rutina de optimización: es un algoritmo para resolver el problema de programación cuadrática (QP) que surge durante el entrenamiento de máquinas de vectores de soporte. Fue inventado por John Platt en 1998 en Microsoft Research. SMO se usa ampliamente para entrenar SVM y se implementa mediante la popular herramienta LIBSVM.

Value	Description
'ISDA'	Iterative Single Data Algorithm (see [30])
'L1QP'	Uses quadprog to implement $L1$ soft-margin minimization by quadratic programming. This option requires an Optimization Toolbox™ license. For more details, see Quadratic Programming Definition (Optimization Toolbox).
'SMO'	Sequential Minimal Optimization (see [17])

Ilustración LXI: SVM rutina de optimización

- Rutina de optimización ISDA

Vamos a ver si mejora el algoritmo si aplicamos una rutina de optimización, vamos a comenzar con ISDA.

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 149.5048 seconds.
Total objective function evaluation time: 123.543

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    0.37138          1.3362

Observed objective function value = 0.16976
Estimated objective function value = 0.17447
Function evaluation time = 0.89526

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    0.20559          1.2956

Estimated objective function value = 0.17447
Estimated function evaluation time = 0.60393
```

Ilustración LXII: SVM rutina de optimización ISDA resumen

Vemos como con esta rutina empeora el valor de la función objetivo.

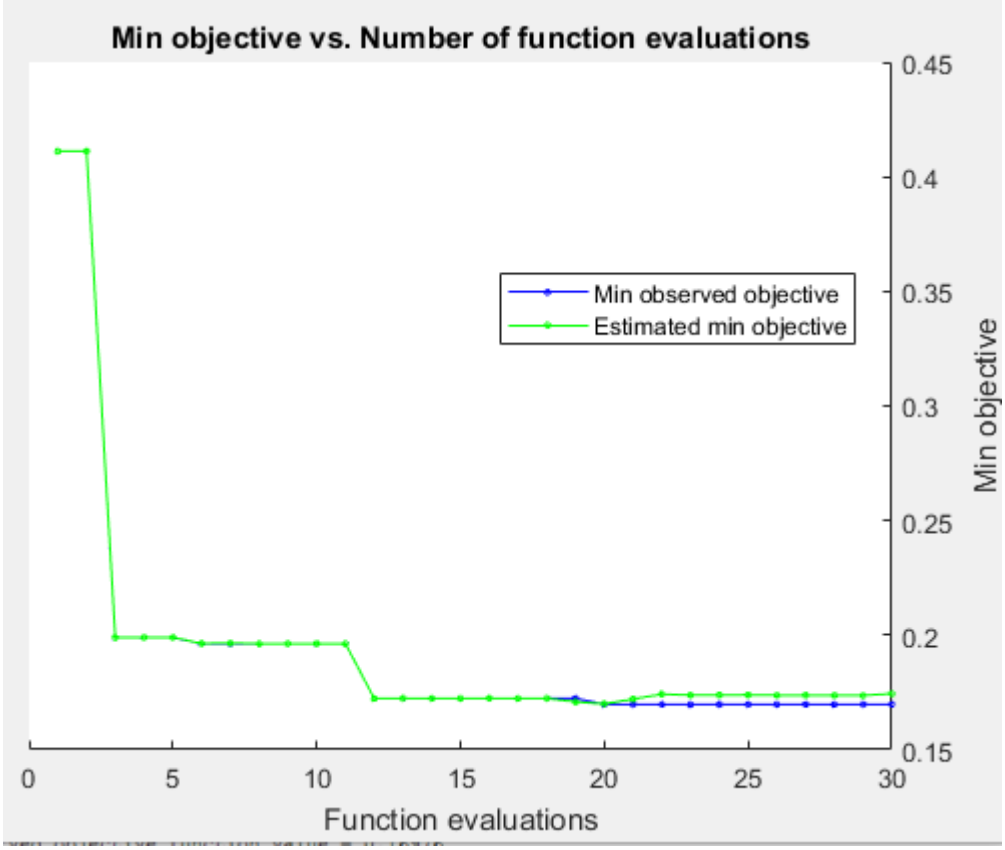


Ilustración LXIII: SVM rutina de optimización ISDA min e iteraciones

Aquí podemos ver como el valor del mínimo objetivo no ha mejorado.

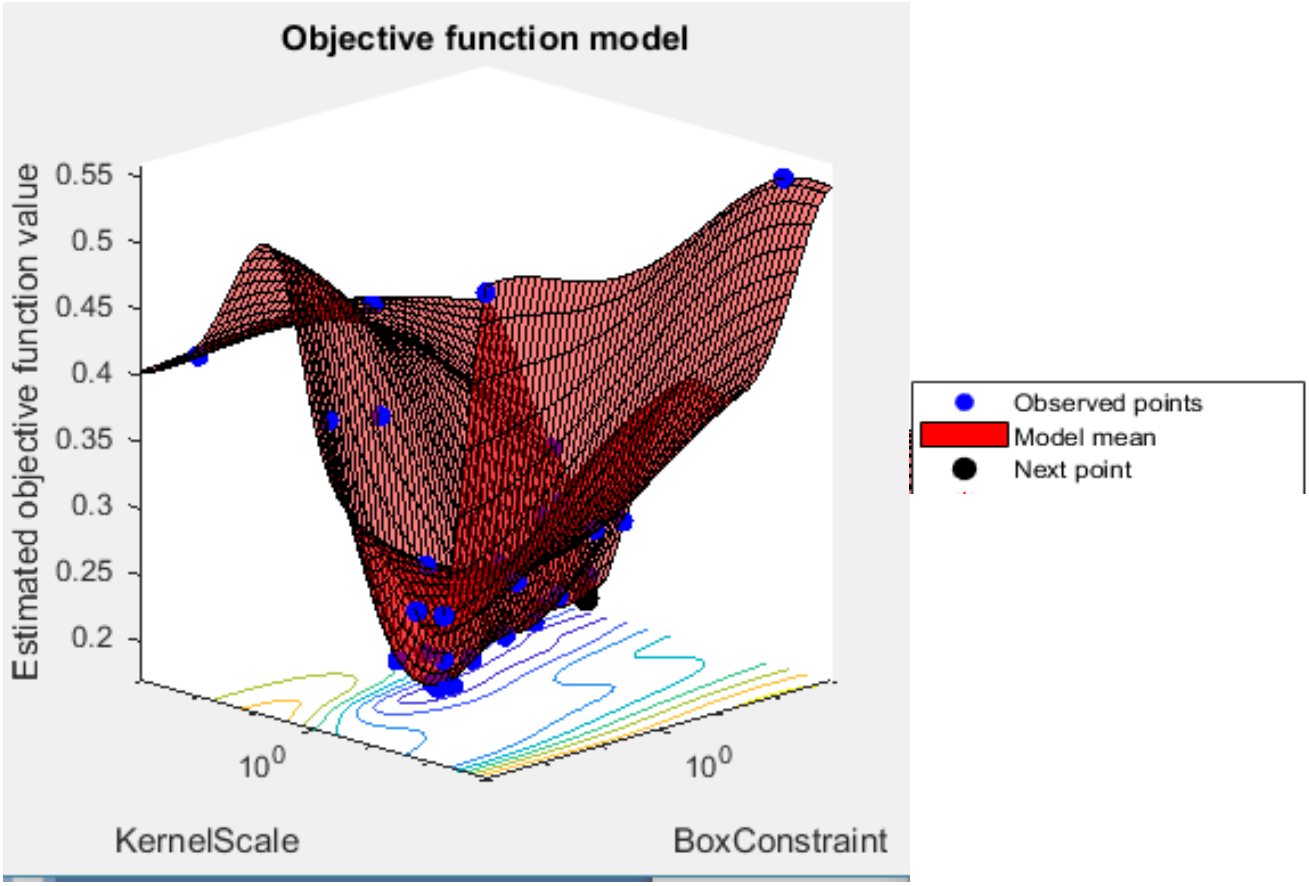


Ilustración LXIV: SVM rutina de optimización ISDA

En esta gráfica podemos ver como se adapta el algoritmo a los datos.

- Rutina de optimización L1QP

Vemos como no hemos mejorado el algoritmo. Vamos a probar ahora con L1QP

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 38.5171 seconds.
Total objective function evaluation time: 11.0807

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    0.021223         0.73417

Observed objective function value = 0.16446
Estimated objective function value = 0.16765
Function evaluation time = 0.43385

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    0.024803         0.74608

Estimated objective function value = 0.16765
Estimated function evaluation time = 0.31484
```

Ilustración LXV: SVM rutina de optimización L1QP resumen

Usando esta rutina podemos ver cómo ha mejorado el valor de la función objetivo con respecto a la rutina anterior.

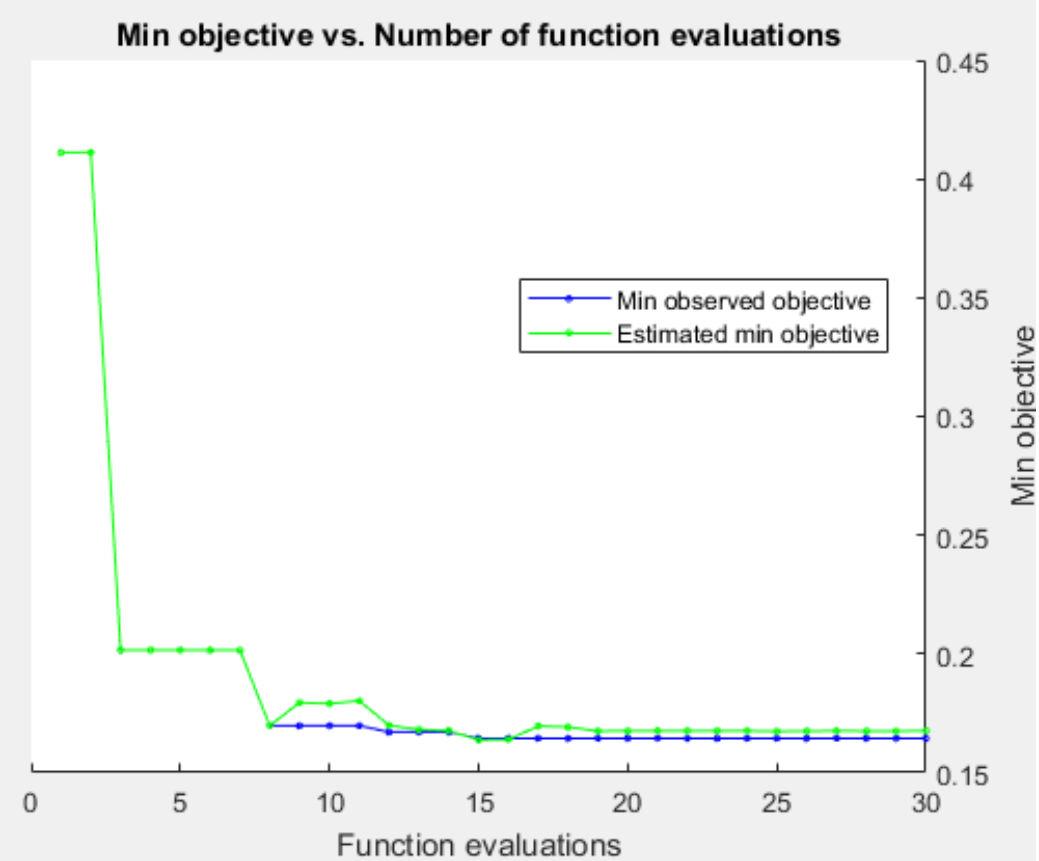


Ilustración LXVI: SVM rutina de optimización L1QP min e iteraciones

En este grafico podemos ver como el mínimo observado y estimado se mantiene con los mismos valores que con la anterior rutina de optimización.

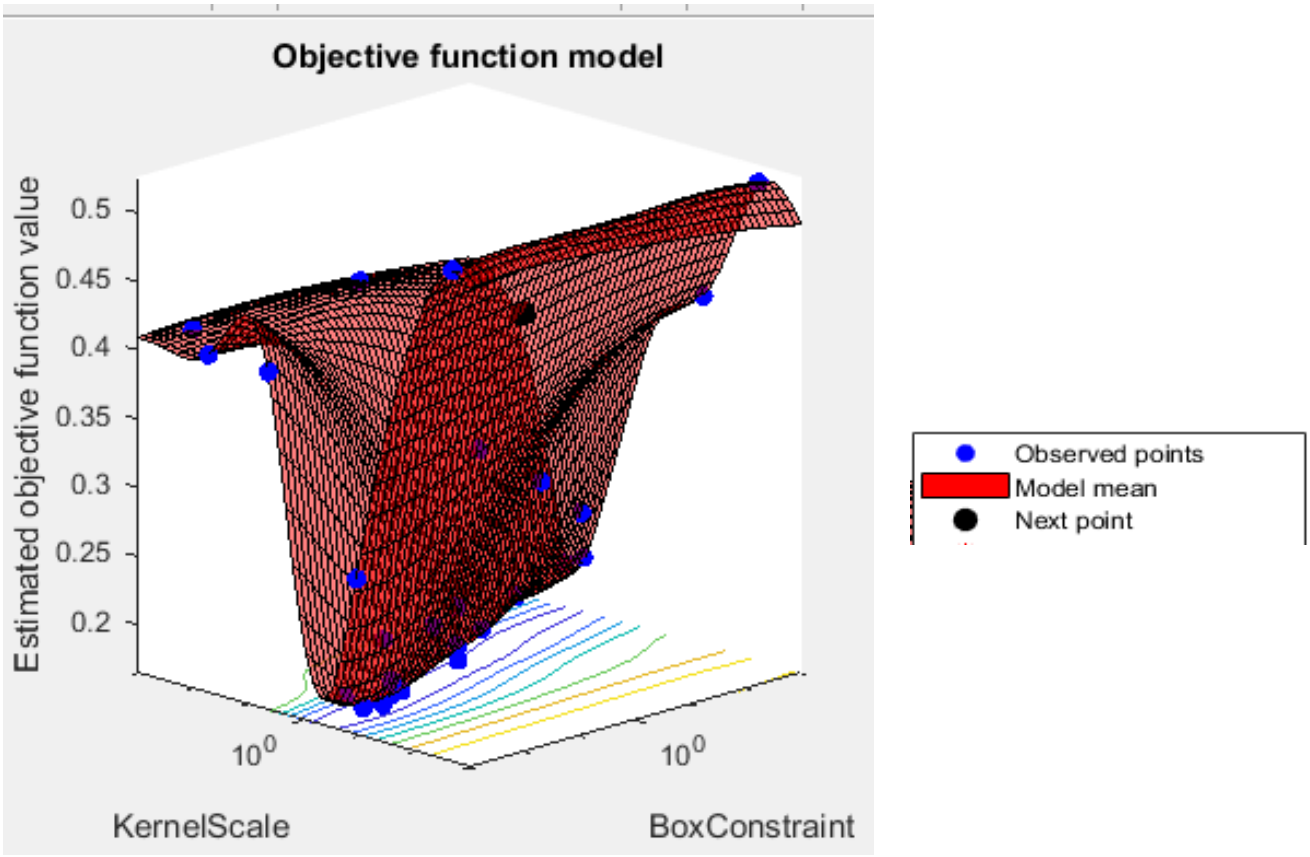


Ilustración LXVII: SVM rutina de optimización L1QP

Podemos ver como con una rutina de optimización L1QP el algoritmo mejora la rutina anterior.

- Rutina SMO

Por último vamos a probar con SMO, que es la rutina que aplica Matlab por defecto.

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 132.8121 seconds.
Total objective function evaluation time: 107.7814

Best observed feasible point:
    BoxConstraint    KernelScale
    _____________    ___________

    0.022282         0.80965

Observed objective function value = 0.16446
Estimated objective function value = 0.16718
Function evaluation time = 0.14253

Best estimated feasible point (according to models):
    BoxConstraint    KernelScale
    _____________    ___________

    0.020828         0.80248

Estimated objective function value = 0.16718
Estimated function evaluation time = 0.14763
```

Ilustración LXVIII: SVM rutina de optimización SMO

Entre esta rutina y la anterior hay muy poca diferencia, la unica diferencia notable es el tiempo de ejecucion, ya que con esta rutina de optimización el algoritmo tarda menos en ejecutarse.

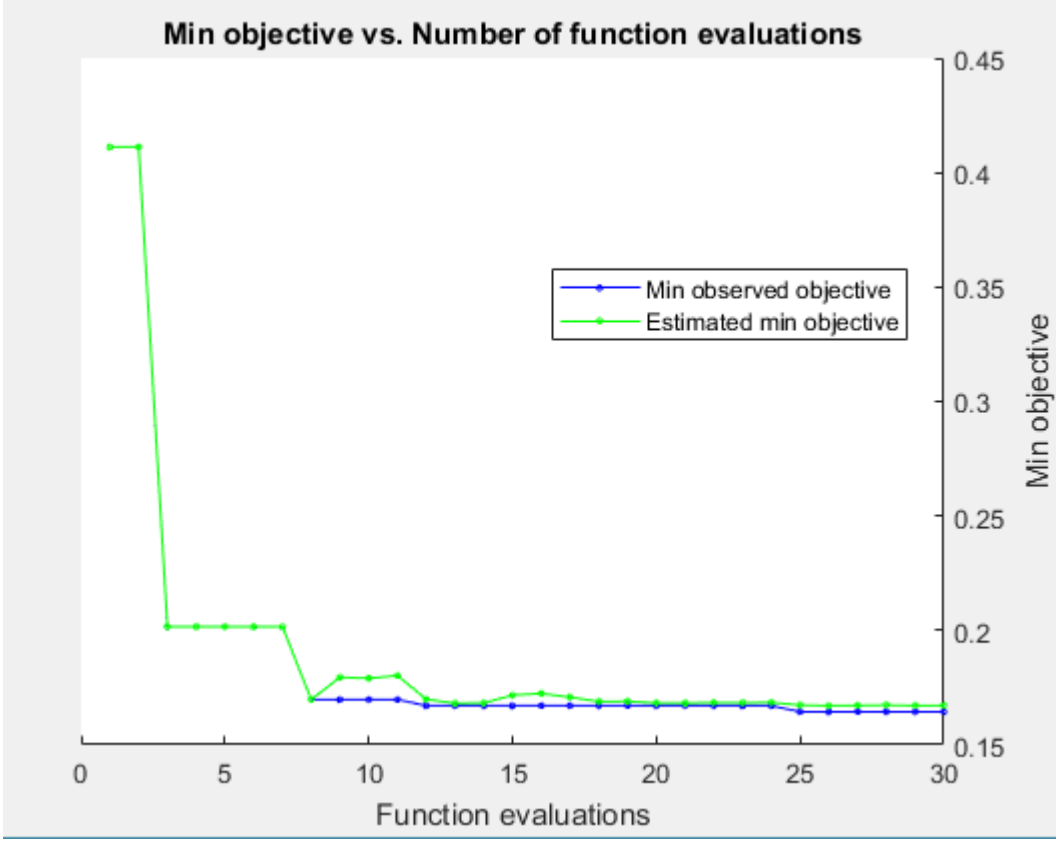


Ilustración LXIX: SVM rutina de optimización SMO min e iteraciones

En este grafico podemos ver como el mínimo observado y estimado se mantiene con los mismos valores que con la anterior rutina de optimización.

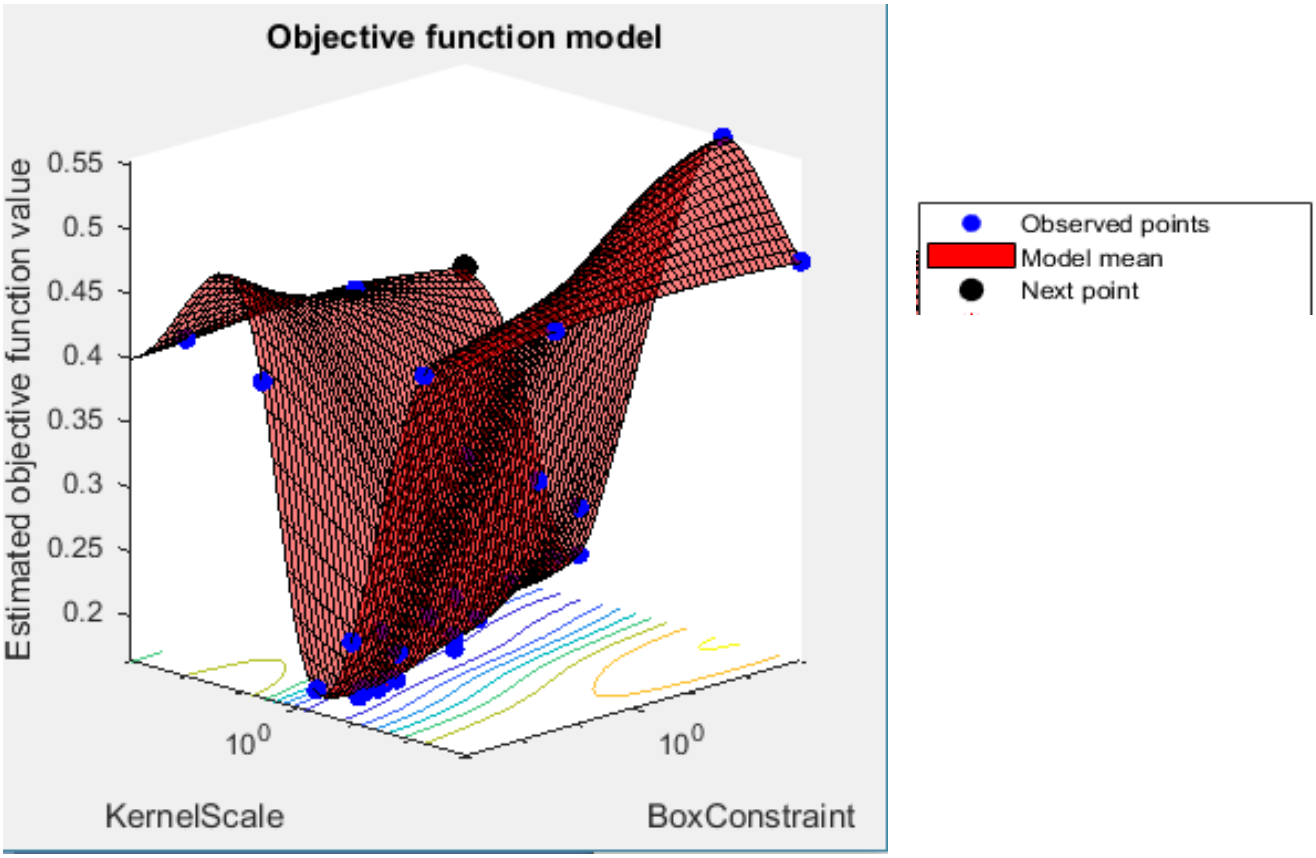


Ilustración LXX: SVM rutina de optimización SMO

Vemos como para este algoritmo la rutina de optimización que mejora un poco los valores de la función objetivo es la L1QP.

4.3.5. Comparación del algoritmo con los distintos parámetros

Después de entrenar nuestro algoritmo finalmente hemos escogido un algoritmo con un Kernel polinomial de grado dos, una rutina de optimización L1QP, ya que mejora los valores de la función objetivo, y con la rutina de optimización mejora el tiempo de ejecución.

En cuanto a la optimización de los híper parámetros hemos escogido expected-improvement-plus, ya que disminuía la función objetivo, el box Constraint y el tiempo de ejecución.

4.3.6. Evaluación de los resultados

Hemos entrenado al algoritmo para detectar la doble compresión, únicamente en formatos MPEG. El algoritmo ha sido entrenado primeramente optimizando los híper parámetros. Después se han entrenado los parámetros relacionados con el algoritmo, eligiendo el kernel y la rutina de optimización que minimizaban la función objetivo.

Para la evaluación de los resultados hemos realizado varias pruebas con el algoritmo. Primeramente hemos introducido un vídeo original al que le hemos aplicado el algoritmo y hemos sacado la tabla de predicción comparando la etiqueta real de cada GOP con la predicha por el algoritmo, obteniendo la ilustración que tenemos a continuación.

TrueLabel	PredictedLabel
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'
"ORIGINAL"	'ORIGINAL'

Ilustración LXXI: Video comprimido original

Es decir, el video original tiene 11 GOP de los cuales 11 de ellos tienen la etiqueta "ORIGINAL". Por lo que el modelo ha predicho de forma correcta todos los GOP que forman ese video.

Después hemos introducido el mismo video pero con doble compresión tanto disminuyendo la calidad en VBR, como aumentando o disminuyendo el Bit Rate en CBR y el resultado ha sido el que mostramos en la siguiente ilustración.

TrueLabel	PredictedLabel
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'
"TAMPERING"	'TAMPERING'

Ilustración LXXII: Video con doble compresión

Para el video con doble compresión el modelo ha predicho que todos los GOP que lo forman han sido manipulados. Por lo tanto, podemos concluir en que el video está manipulado.

5. Conclusiones

Proponemos un enfoque novedoso para la detección de compresión de video MPEG. La distribución del primer dígito de coeficientes cuantificados AC MPEG distintos de cero en I, P y B frames sigue la ley de Benford, si el video se comprime en MPEG solo una vez. La distribución se ve perturbada si el video está doblemente comprimido en MPEG.

Para detectar la perturbación, utilizamos las probabilidades del primer dígito y las estadísticas de bondad de ajuste de cada GOP. Los experimentos han demostrado la efectividad del enfoque si el vídeo MPEG está doblemente comprimido, con VBR y con una calidad más baja, y en CBR con un Bit Rate mas alto o más bajo.

El enfoque propuesto se ha intentado que sea lo más general posible, ya que puede detectar la compresión MPEG doble no solo para video VBR sino también para video CBR. En lugar de confiar en la observación específica, utiliza un marco de aprendizaje automático con características distintivas efectivas. Por lo tanto, es más confiable y eficiente.

6. Trabajos futuros

Dado el trabajo que se ha realizado sería interesante poder continuar con la investigación en otros formatos de vídeo, como por ejemplo, H264. También se podría analizar cómo detectar la manipulación de vídeos en streaming.

Como una posible línea de trabajo futuro podríamos detectar en que frame se ha realizado la manipulación, incluso el grupo de GOP que se han alterado.

Para obtener los GOP de un vídeo se realiza una secuencia en serie que debido a que hay que procesar todos los frames y realizar varias operaciones sobre ellos, hace que el proceso sea lento. Una forma más rápida de obtener los GOP es paralelizando el procesado de los frames.

Realizar una comparativa con distintos algoritmos, ya que por alcance y falta de tiempo no se ha podido realizar en este trabajo.

7. Bibliografía

1) Wang, W.H., Farid, H.: Exposing digital forgeries in video by detecting double MPEG compression. In: ACM Multimedia and Security Workshop, Geneva, Switzerland (2006)
2) Lukas, J., Fridrich, J.: Estimation of primary quantization matrix in double compressed JPEG images. In: Digital Forensic Research Workshop, Cleveland, Ohio, USA (2003)
3) Popescu, A.C.: Statistical tools for digital image forensics. Ph.D. Dissertation, Department of Computer Science, Dartmouth College (2005)
4) Fu, D., Shi, Y.Q., Su, W.: A generalized Benford'slawfor JPEG coefficients and its application in image forensics. In: IS&T/SPIE 19TH Annual Symposium, San Jose, California, USA (2007)
5) Mitchell, J.L., Pennebaker, W.B., Fogg, C.E., LeGall, D.J.: MPEG video: compression standard. Chapman& Hall, New York (1997)
6) Leon-Garcia, A.: Probability and Random Processes for Electrical Engineering, 2nd edn. Addison Wesley, Reading (1994)
7) Hsu, C.-W., Chang, C.-C., Lin, C.J.: LIBSVM: a library for support vector machines (2001)
8) Matlab Central File Exchange, http://www.mathworks.com
9) Matlab Fitcsvm https://es.mathworks.com/help/stats/Fitcsvm.html
10) Ffmpeg documentation https://www.ffmpeg.org/documentation.html