Treball fi de carrera

GRAU ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

DISEÑO SUBLINEAL EVOLUTIVO DE CÓDIGOS CORRECTORES DE ERRORES

Miguel Ángel Bautista Martín

Director: Sergio Escalera Guerrero
Xavier Baró Solé
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 18 de febrer de 2010

AGRADECIMIENTOS

Me gustaría para comenzar este estudio, agradecer a todas las personas que han hecho posible que estos 3 años de estudio hayan valido la pena. Me gustaría empezar por mi familia, mis padres Miguel y Ana que tanto me han dado y que han esto apoyándome de manera incansable, moral y económicamente, gracias a ellos estoy donde estoy ahora. No me puedo olvidar de mi hermana, Ana María, muchas veces ella me ha servido de inspiración.

No me olvidaré jamás de mis amigos y compañeros de piso, con los que tanto me he divertido y que han sido los que al fin y al cabo han estado más cerca en los momentos difíciles. Entre otros Adrián S., Pau, Jean Paul, Adrián B. , Javier, etc.

Jamás podría olvidarme de una persona, Lucía, que en este último año ha sido más que un pilar de apoyo en los buenos y en los malos momentos. Sin ella, esto probablemente no habría sido posible, a ti te digo, muchas gracias!

Tampoco podrían faltar los profesores que tanto me han enseñado y de los que tanto he podido aprender, des de el primer año Dra. Anna Puig, Dra. Maite López, Dra. Maria Salamó, pasando por el segundo, Dr. Manel Puig, Dr. Xavier Jarque y finalmente el tercero, Dr. Oriol Pujol, Dra. Petia Radeva y un largo etcétera que no acabaría en esta página.

Para finalizar dar muchas gracias a mis directores de proyecto Dr. Sergio Escalera y Dr. Xavier Baró, han sabido motivarme como al que más y sacar lo mejor de mí. He recibido un trato envidiable por su parte, sólo espero demostrarles mi agradecimiento con el trabajo que se ha llevado a cabo.

Gracias,

Miguel Ángel Bautista Martín

RESUMEN

Una de las principales coyunturas de la inteligencia artificial y el aprendizaje automático es incrementar la interactividad de sistemas inteligentes con el entorno. La mayoría de estrategias actuales suelen hacer uso de sensores para simular la adquisición de datos que nosotros obtenemos a través de nuestros sentidos. Una vez se ha adquirido la información, uno de los grandes retos actuales consiste en el desarrollo de técnicas capaces de procesar grandes conjuntos de datos. La finalidad de dicho proceso es obtener una decisión entre un amplio conjunto de posibilidades. En este estudio proponemos una metodología que es capaz de procesar grandes conjuntos de datos, siendo capaces de discriminar entre un conjunto elevado de clases. En particular, nos centramos en los códigos correctores de errores, definiendo un número logarítmico de clasificadores. Los algoritmos genéticos son usados para definir un subconjunto óptimo de problemas a entrenar, así como para detectar los parámetros óptimos que mejoran la capacidad de generalización de los clasificadores. La estrategia se ha evaluado sobre bases de datos públicas de la comunidad de aprendizaje automático y visión artificial, obteniendo resultados que mejoran los reportados en estado del arte, a la vez que el coste computacional se reduce significativamente.

RESUM

Una de les primeres conjuntures de la intel·ligència artificial i l'aprenentatge automàtic és incrementar la interactivitat dels sistemes intel·ligents amb l'entorn. La majoria d'estratègies actuals solen fer ús de sensors per a simular l'adquisició de dades que nosaltres obtenim mitjançant els nostres sentits. Una vegada s'ha adquirit la informació, un dels grans reptes actuals consisteix en el desenvolupament de tècniques capaces de processar una gran quantitat de dades. La finalitat de dit procés es obtindre una decisió entre un ampli ventall de possibilitats. En aquest estudi proposem una metodologia capaç de processar grans conjunts de dades, sent capaços de discriminar entre un conjunt elevat de classes. En particular, ens centrem en els codis correctors d'errors, definint un nombre logarítmic de classificadors. Els algoritmes genètics són usats per a definir un subconjunt òptim de problemes a entrenar, així com per a detectar el paràmetres òptims que milloren la capacitat de generalització dels classificadors. L'estratègia ha estat avaluada sobre bases de dades públiques de la comunitat d'aprenentatge automàtic i

visió artificial, obtenint resultats que milloren els reportats a l'estat de l'art, a la vegada que redueixen significativament el cost computacional.

ABSTRACT

One of the first junctures in artificial intelligence and machine learning is to increase the interactivity of intelligent systems with their environment. Most of the actual strategies use sensors in order to simulate the data acquisition process that we obtain by means of our senses. Once the information has been extracted, one of the main challenges consists of developing suitable techniques for the processing of huge amount of data. The aim of this process is to take a decision between a wide set of possibilities. In this report, we propose a new methodology capable of processing huge sets of data, being able to discriminate between a wide set of categories. In particular, we focus on the error correcting output codes framework, defining a logarithmic number of classifiers. Genetic algorithms are used to define an optimum subset of problems as well as to find the optimum parameters that increase the generalitzation capability of the classifiers. The approach is tested over a wide number of the public machine learning and computer vision community datasets, getting results that outperform state-of-the-art strategies, at the same time that significantly reduces the computational cost.

Índice general

Capítulo 1

Preámbulo

1.1. Introducción

Uno de los objetivos de la visión y la inteligencia artificial es conseguir programar un agente inteligente que sea capaz de distinguir entre un conjunto de objetos que aparecen en las imágenes y los vídeos capturados del entorno.

Las metodologías propuestas para resolver estos problemas se han denominado 'problemas de clasificación'. El término 'clasificación' se ha utilizado para denotar el reconocimiento y agrupación de objetos que pueden estar representados de distinta forma dentro de un determinado sistema. Dichos objetos agrupados en lo que denominados clases, suelen estar definidos a través de un conjunto de propiedades o características, sobre las cuales se debe hacer un previo aprendizaje para proceder a su posterior reconocimiento.

En un principio podríamos pensar que estos sistemas podrían moverse en un espacio con un conjunto binario de clases (dos categorías a distinguir). De aquí surge toda la teoría y estudio de la clasificación binaria. A simple vista comprenderíamos que se necesitaría alguna herramienta para trabajar con una clasificación que no fuese simplemente binaria, ya que el mundo tiene inherentemente un funcionamiento multi-clase. De esta manera pasaríamos a clasificar un objeto entre N posibles clases definidas en nuestro sistema.

Además, con el gran incremento de la información experimentado en los últimos años y el trepidante aumento del tráfico en la red, nos estamos acercando a la clasificación de miles de clases. Estos problemas de alta cardinalidad de datos son los llamados 'Large Escale Problems', problemas en los

cuales el conjunto de datos a manejar es enormemente amplio y donde la búsqueda exhaustiva o analítica de una solución tiende a ser computacionalmente inviable. Es aquí donde la potencia de la inteligencia artificial y en concreto del aprendizaje automático y de la computación evolutiva se hace palpable.

En concreto, dentro del aprendizaje automático, para el desarrollo de este proyecto nos hemos centrado en el aprendizaje supervisado, el cual se basa en obtener una función que, para cada tipo de entrada (características inherentes de los objetos) retorna un tipo o etiqueta diferente en la salida. Es así como el aprendizaje automático nos ofrece una poderosa herramienta para poder obtener soluciones a los problemas de clasificación.

Por otra parte, la evolución y la genética llevan desde que se estableció la vida en este planeta resolviendo el problema de la adaptación de los seres vivos al medio en el que se desenvuelven. La evolución se podría ver desde un punto de vista computacional como una manera de abordar el proceso resolutivo de un problema, de una manera que no es ni exhaustiva ni analítica. Esta manera de abordar el problema consiste en la proposición que tiempo atrás se propuso en [Dar], en la que se argumentaba que la adaptación al medio de cualquier tipo de ser vivo mejora a medida que transcurren las generaciones. De esta manera podríamos concebir un algoritmo resolutivo como un proceso que parte con un conjunto de soluciones aleatorias válidas, que con el paso de generaciones y su consiguiente apareamiento y mutación, se adaptan de forma más eficiente al medio, es decir, aportan mejores soluciones.

En este proyecto proponemos una aproximación a la solución de los problemas de multiclasificación, que en futuros estudios nos servirá para atacar problemas de muy alta cardinalidad, como los llamados 'Large Escale Problems'.

1.2. Estado del arte

Los problemas de multiclasificación han sido objeto de estudio por varios investigadores en los últimos años, sin lograr una solución clara y viable. En este apartado vamos a revisar los trabajos que abordan este tipo de problemas.

Los primeros sistemas que se desarrollaron con el objetivo de el problema de la multi-clasificación utilizaron diferentes estrategias para aproximar una solución aceptable: agentes inteligentes [AR06], redes neuronales [CPA08],

etc. En la ultima década estudios han demostrado que representar el problema mediante el marco de los códigos correctores de errores (en adelante ECOC) representa significantes mejoras cuando se trabaja con diseños dependientes del problema. Este marco ha sido ampliamente aplicado a problemas clásicos como reconocimiento facial [KGWM01], reconocimiento de texto [Gha02], etc.

Un código corrector de error se entiende como una matriz, las filas de la cual denotan las clases a discriminar en nuestro sistema, y las columnas las hipótesis de las que disponemos para distinguir dichas clases.

El trabajo con los ECOC consiste en dos etapas básicas: en la primera, llamada codificación, a cada clase se le asigna una palabra codificada que servirá para distinguir la clase del resto de posibles clases de forma unívoca; en la segunda etapa, denominada decodificación, cada ejemplo a clasificar (cada nuevo objeto en el espacio de nuestro sistema) se compara con cada palabra codificada previamente, y se obtiene una predicción de clasificación.

Una de las codificaciones más conocidas para solucionar problemas multiclase se denomina *'uno contra todos'*, que consiste en distinguir una clase del resto de clases posibles. La codificación uno contra todos codifica N hipótesis para N clases. Aun así, trabajando con dicho número de hipótesis los resultados obtenidos para problemas de alta cardinalidad muchas veces no han pasado de lo que se denomina como *'nivel de clasificación aleatoria'* que se conoce como *1/N*. Es decir, el margen que transciende de la mera clasificación aleatoria a la clasificación dada como aprobada es *1/N* donde N es el número de clases.

Otra manera de aproximar el problema de clasificación es mediante el marco conocido como *"clasificadores en línea"*. Este tipo de clasificación aprende un objeto a la vez y tiene como punto a favor que implementa un tipo de retroalimentación que hace que el propio sistema readapte su comportamiento a medida que recibe nuevos objetos. Así, podríamos decir que los clasificadores en línea tienen tres pasos básicos: el primero sería la recepción por parte del algoritmo de clasificación del objeto o ejemplo a clasificar; el segundo paso sería la predicción de la etiqueta de dicho objeto; y el tercero seria el innovador en el que el algoritmo recibe la etiqueta real del objeto y utiliza esta información para minimizar el error en futuras pruebas. La contra de dicho sistema es que se le ha de proporcionar la etiqueta real de cada ejemplo a clasificar en un tiempo reducido. Actualmente es difícil diseñar un sistema que aprendiendo en línea de un rendimiento similar a los clasificadores fuera de línea. Esto sugiere que podría ser un buen algoritmo

para situaciones de evolución continua y sistemas en tiempo real en los que en un corto espacio de tiempo se puede saber cuál va a ser la clasificación del ejemplo que se le ha dado al algoritmo. Por ejemplo, en [LM06] se mostró la aplicación satisfactoria de este sistema en la predicción de la dirección del movimiento del mercado financiero.

Otra rama que muchos investigadores han explorado para mejorar el resultado de los problemas de multiclasificación han sido la redes neuronales. Uno de los sistemas de redes neuronales que se utilizan en problemas de multiclasificación es el PMC *(Perceptor multi capa)*. Este sistema se basa en el principio de las redes neuronales comunes, en las que existen tres capas: la capa de entrada, la capa oculta y la capa salida. En este sistema la salida de cada neurona de la capa i es entrada de todas las neuronas de la capa $i+1$.

Destacar que cada capa tiene su tarea en este sistema: la capa de entrada se encarga únicamente de recibir los patrones (ejemplos a clasificar) que se procesaran en el sistema, la segunda capa (la capa oculta) es la capa donde se lleva a cabo todo el procesamiento de dichos patrones y la capa de salida es la capa donde se obtiene las predicciones para los patrones o ejemplos a clasificar. El algoritmo utilizado para el entrenamiento de estas redes se denomina *propagación hacia atrás* .

1.3. Propuesta de resolución

Finalmente, después de comprender y estudiar las anteriores alternativas analizadas en posteriores capítulos , nos declinamos por un tipo de solución, con la que se espera llegar a obtener una serie de resultados que mejoren ampliamente lo existente.

Principalmente el sistema está basado en la potencia de la clasificación binaria que nos ofrece el aprendizaje automático, y más en concreto las máquinas de vectores de soporte (en adelante SVM, del inglés 'Support Vector Machines'). Este mecanismo nos ofrece un alto rendimiento en lo que a clasificación binaria se refiere.

Como habíamos comentado anteriormente, para abordar problemas de multi-clasificación, el marco del los ECOC es uno de los mejores en términos de combinación de clasificadores binarios y de rapidez de cómputo [DB94]. En este sentido hemos diseñado un nuevo ECOC que permite definir un número reducido de hipótesis para abordar problemas de alta cardinalidad. El ECOC-Sublineal contiene $log_2 N$ hipótesis para distinguir N clases de forma

unívoca. Finamente, el último pilar de nuestro sistema será la computación evolutiva, y en concreto los algoritmos genéticos, que nos ofrecen una increíble herramienta en lo que a proceso de búsqueda de soluciones se refiere. Así, finalmente denominaríamos a nuestro sistema como *'diseño sublineal evolutivo para códigos correctores de errores'*.

Para demostrar la fiabilidad y la eficiencia del método, se ha decidido realizar una batería de pruebas sobre algunos conjuntos de datos del *'UCI Machine Learning Repository'*. En concreto han sido seleccionados 12 conjuntos de datos, descritos en el apartado de resultados. Ademas también se han introducido algunos conjuntos de datos privados de mayor cardinalidad (mas de 20 clases), relacionados con problemas de visión artificial. Los resultados obtenidos son comentados y analizados en capítulos posteriores.

1.4. Organización

Este estudio se organiza por capítulos: en el primer capítulo se hace una pequeña introducción histórica y se presentan tanto los problemas a tratar como la solución diseñada.

En el segundo capitulo se tratan profundamente las cuestiones del análisis del problema así como la previsión de los costes. También se tratan los requerimientos tanto funcionales como no funcionales y se analiza toda la metodología a desarrollar, terminando con una primera propuesta de sistema final.

En el tercer capítulo se describe todo el proceso de diseño de la aplicación así como la implementación y los diagramas UML correspondientes.

En el cuarto capítulo se describe la batería de pruebas desarrollada para mostrar la eficiencia del sistema. Se muestran y analizan los resultados obtenidos.

El quinto apartado capítulo engloba las conclusiones del estudio sopesando cuales son los puntos fuertes y débiles del proyecto, así como futuras líneas de investigación.

En los apéndices se incluye un CD con el código del proyecto.

Capítulo 2

Análisis

Este capítulo comprende toda la etapa de análisis del proyecto. En el mismo se estudia el estado del arte de las alternativas de resolución a los problemas de clasificación. También se realiza una revisión de los distintos tipos de clasificadores más utilizados en este ámbito. En lo que a análisis de software se refiere, analizamos los requerimientos funcionales, generando su diagrama de caso de uso. Finalmente terminamos este capítulo con un análisis de los costes generados, así como con una planificación del trabajo a realizar.

2.1. Métodos de multiclasificación

Desde que la inteligencia artificial empezó a adquirir importancia, uno de los problemas que ha intentado resolver han sido los problemas de clasificación automática. Múltiples aproximaciones, provenientes de distintas ramas de la inteligencia artificial han tratado de resolver el problema de forma eficiente ya sea mediante el uso de estimación de probabilidad o mediante la división del problema de multiclasificación en subproblemas de menor complejidad. En este apartado analizamos varios casos específicos.

2.1.1. Redes neuronales artificiales

Un aspecto fundamental que intenta abordar la inteligencia artificial es la reproducción de la metodología de aprendizaje biológica, mediante modelos matemáticos, en algoritmos capaces de ser ejecutados por computadores.

Dentro de esta línea las redes neuronales artificiales son uno de los primeros intentos de esta reproducción.

Un sistema inteligente basado en redes neuronales artificiales (en adelante RNA) intenta copiar el modelo de red neuronal del cerebro humano, en el cual cada una de las neuronas procesa una pequeña parte del problema, para después aportar su solución a las demás neuronas de la red. De esta manera el elemento por excelencia de una RNA es la neurona. Una neurona es un pequeño sistema que genera una salida en función de varias entradas, normalmente en una RNA estas entradas son interconexiones con otras neuronas, es decir, las entradas de unas neuronas son las salidas de otras.

Para generar esta salida la neurona necesita de tres funciones: **la función de excitación, la función de activación y la función de transferencia**.

- La función de excitación es la encargada de caracterizar la neurona como excitatoria o inhibitoria. Existen varias, pero la más común, la función lineal de base, tiene como umbral de activación el signo del valor del sumatorio de las entradas ponderadas con sus correspondientes pesos como se ven en 2.1

$$f_{exc}(x) = sign(\sum x_i \cdot w_i) \tag{2.1}$$

- La función de activación modifica el resultado de la función de excitación a menudo de forma no lineal, una de las más conocidas es la sigmoidal, como podemos observar en 2.2

$$f_{act}(x) = \frac{1}{1 + e^{x/\sigma}} \tag{2.2}$$

- La función de transferencia es la ultima función que se aplica al valor, es la encargada de normalizar el valor de la salida que normalmente tendrá relación con la interpretación que se les quiera dar a dichas neuronas. Una de las funciones de transferencia más conocidas es la sigmoidal (expuesta anteriormente), que acota la salida en el rango $[0, 1]$.

Uno de los modelos más utilizados de red neuronal es el perceptrón multicapa. En este modelo la RNA está formada por distintas capas de neuronas conectadas entre ellas. La virtud de este modelo es que puede solucionar problemas que no son linealmente separables. Las capas del perceptrón pueden

conectarse tanto local como totalmente. Respectivamente en el primer caso la salida de las neurona de la capa i se conectan con una región de neuronas de la capa $i + 1$. En el segundo caso la salida de cada neurona de la capa i esta conectada con la entrada de todas las neuronas de la capa $i + 1$. Normalmente este modelo trabaja con 3 capas, como se puede ver en la imagen 2.1, aunque podría trabajar con más.

Cuando trabajamos con problemas de clasificación normalmente las capas del perceptrón tienen tareas claramente diferenciadas. La capa de entrada sería la que recibe los patrones a clasificar. En la capa o capas ocultas se lleva a cabo todo el procesamiento de dichos patrones para dar en la capa de salida una predicción de clasificación.

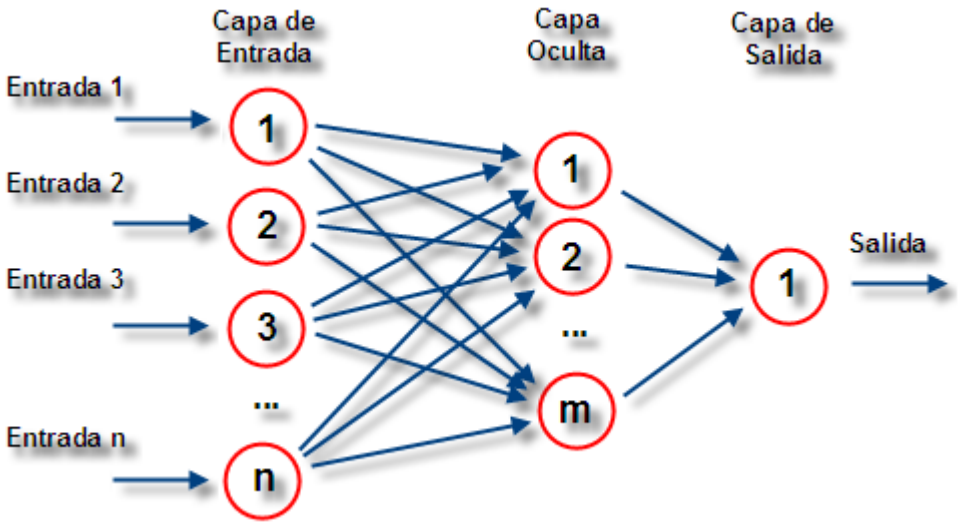


Figura 2.1: Perceptrón multicapa totalmente conectado

Como se ha explicado en capítulos anteriores, para obtener una predicción de clasificación se ha tenido que realizar previamente una etapa de aprendizaje, en la cual el sistema aprende a discriminar los patrones que posteriormente tendrá que clasificar.

El algoritmo por excelencia de aprendizaje del perceptrón multicapa es el de *propagación hacia atrás*. En el que la tarea de aprendizaje se lleva a cabo mediante el ajuste de los pesos de las interconexiones entre capas de neuronas, este algoritmo no es más que una generalización del problema de mínimos cuadrados. Consideremos el error producido en el nodo de salida

k en el n-ésimo dato como $e_k(n) = o_k(n) - p_k(n)$, donde $p_k(n)$ es el valor predicho por el perceptrón y $o_k(n)$ es el valor que realmente tiene dicho dato. Tratamos entonces de minimizar la energía del error en la salida dado por:

$$\varepsilon(n) = \frac{1}{2}\sum e_k^2(n) \tag{2.3}$$

Por la teoría de diferenciales observamos que el cambio en cada peso debería ser:

$$\triangle w_{ki}(n) = -\eta\frac{\partial\varepsilon(n)}{\partial v_k(n)}p_i(n) \tag{2.4}$$

Donde $p_i(n)$ es el valor de la anterior neurona y η es el ratio de aprendizaje. El análisis posterior es más complejo y no se tiene como objeto de este estudio, simplemente comentaremos que en la simplificación final se puede observar como en el peso de una capa k debemos antes cambiar los pesos de la capa $k-1$ en acorde con la derivada de la función de activación, así este algoritmo representa una propagación hacia atrás de la función de activación.

2.1.2. Regresión logística

La regresión logística o modelo logístico se utiliza para modelar la probabilidad de que un hecho tenga lugar. Lo cual se lleva a cabo adaptando los datos a una curva logística. Una de las ventajas de este método de regresión es que puede manejar tanto atributos reales como discretos, mediante el uso de la curva logística (también conocida como sigmoidal). Dicha curva tiene una característica especial y es que puede tomar cualquier valor real $z \in (-\infty, +\infty)$ y dar su probabilidad $f(z)$ .De este manera la curva tendría el siguiente aspecto:

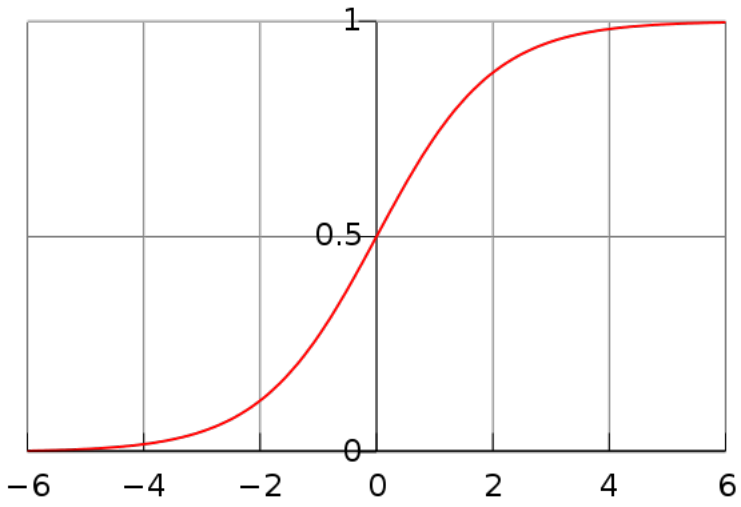


Figura 2.2: Curva logística

De esta manera el problema se reduce a encontrar z que normalmente se hace utilizando el siguiente modelo,conocido como logit:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ... + \beta_n x_n \tag{2.5}$$

Donde el parámetro β_0 es conocido como el interceptor y los parámetros β_i donde $i > 0$ son conocidos como los coeficientes de regresión de las x_i correspondientes. Cada coeficiente es es el peso de ese atributo en el resultado final, si el peso es positivo incrementa la probabilidad de la posible salida mientras que si es negativo la disminuye. Un peso β_i extremadamente alto indicaría que el resultado esta altamente influenciado por el atributo x_i asociado a ese peso. Este método es muy útil para describir las relaciones entre diferentes variables independientes y un resultado binario, comúnmente expresado como una probabilidad.

El método de regresión logística comporta un fuerte componente matemático y estadístico que no se describe ya que no se reconoce como parte del objetivo del trabajo. A modo de pequeña reseña notaremos que la regresión logística asume una distribución binomial de los datos e intenta encontrar las probabilidades para dicha distribución. En el apartado 2.1.1 se describe un poco más a fondo la matemática del método, entendiendo que la regresión logística no es más que un Perceptrón de una única capa.

2.1.3. Árboles de aprendizaje

Un árbol de aprendizaje es un modelo predictivo que analiza observaciones sobre un ítem para llegar a conclusiones sobre el valor del mismo. En concreto nos centraremos en los llamados *árboles de clasificación*. En este tipo de árboles, las hojas representan clasificaciones y las ramas representan conjuntos de características que han conducido a dichas clasificaciones. En concreto estos árboles intentan crear un modelo que prediga el valor de una variable objetivo, basándose en diferentes variables de entrada. Cada hoja del árbol representa un valor de la variable objetivo calculado mediante los valores de las variables de entrada (nodos interiores) desde la hoja hasta la raíz. Otra manera de entender este paradigma es pensar que cada nodo interior del árbol es un clasificador.

Existen varios tipos de árboles dentro de los árboles de aprendizaje:

1. **Árbol de clasificación:** clasificación en valores discretos.
2. **Árbol de regresión:** clasificación en valores reales.
3. **Árbol de clasificación y regresión:** utilizado para las dos tareas anteriores.
4. **Detector automático de interacción χ^2:** genera particiones en distintos niveles de un árbol.

Los algoritmos de construcción de árboles de clasificación se basan en escoger la característica de los datos de entrada que mejor permita la separación del conjunto de datos en varios subconjuntos, también conocida como entropía, aunque también existen otras métricas para su cálculo. La característica será tan buena como la separación que consigna hacer de los datos que en ese momento tienen asignado el mismo valor objetivo. Uno de los algoritmos más usados para este cálculo es la *ganancia de información*:

$$I_e(f) = -\sum f_i \log_2 f_i \tag{2.6}$$

Uno de los algoritmos más utilizados de construcción de árboles de decisión es el dicotomizador iterativo 3, que de manera intuitiva y sin entrar en detalles sería de la forma:

1. Tomar todos los atributos y calcular su entropía con relación al conjunto de testeo.

2. Escoger el atributo de entropía mínima (o análogamente que maximice la ganancia de información).

3. Hacer que el nodo del árbol contenga dicho atributo.

2.1.4. Mezcla de expertos

Un método con alguna semejanza a los árboles de decisión es el modelo de mezcla de expertos. Se basa en la idea que a menudo los humanos llevamos a cabo cuando nos encontramos ante una decisión de una complejidad notable. Por ejemplo, un doctor antes de diagnosticar una enfermedad grave, como podría ser el cáncer, consulta con varios especialistas en la materia como por ejemplo los oncólogos y antes de tomar una decisión también realizara varias pruebas al paciente (resonancias magnéticas, análisis de sangre, tomografías computarizadas, etc.). Análogamente a esta situación actuaría el modelo de mezcla de expertos. Este modelo constaría de un conjunto de clasificadores $C_1, ..., C_t$ que constituirían lo que se denomina el ensamblado. A continuación un clasificador de segundo nivel C_{t+1} asignaría pesos a los distintos clasificadores de primer nivel. Estas salidas ponderadas por los pesos pasarían por un sistema de combinación que daría el resultado final. El clasificador C_{t+1} también se conoce como red de compuertas ya que normalmente suele ser una red neuronal entrenada mediante el algoritmo de *minimización de esperanza*.

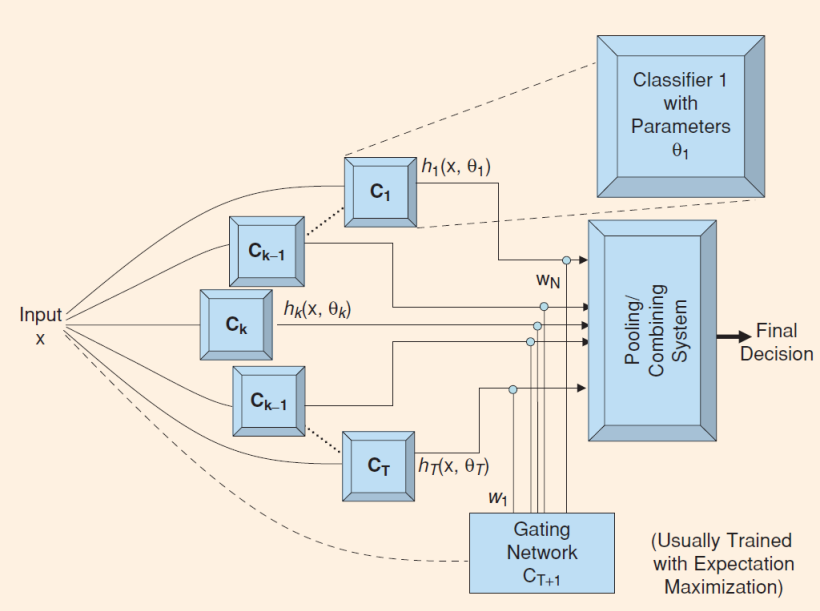


Figura 2.3: Modelo de mezcla de expertos

Como vemos en la figura 2.3, el modelo se podría observar como un algoritmo de clasificación donde cada clasificador, de primer nivel, es un experto sobre un espacio de características o atributos de un ejemplo de entrada x. La red de compuertas elegiría mediante los pesos, el clasificador o la combinación de clasificadores más apropiada para cada ejemplo de entrada. Finalmente para generar la predicción final, el modelo puede combinar los pesos de distintas formas, ya sea eligiendo el clasificador con mayor peso o calculando la suma ponderada de todos los clasificadores, dependiendo del tipo de salida de los expertos (real o discreta).

2.1.5. Códigos correctores de errores

En la literatura que a problemas de multiclasificación se refiere, los códigos correctores de errores tiene una cierta relevancia, por eso serán introducidos en más detalle. Los códigos correctores de errores (en adelante ECOC) se encontrarían encuadrados en la familia de métodos de aprendizaje por ensamblado, dado que es un método que utiliza un conjunto de clasificadores base la respuesta de los cuales es combinada para componer la respuesta final

del método. Los códigos correctores de errores se basan en los principios de la teoría de la comunicación en la cuales un conjunto de bits es enviado por un canal con la agregación de una serie de bits que indican si se ha producido un error en la transmisión. La idea de la aplicación de este principio en los problemas de clasificación es poder resolver los posibles de fallos de los clasificadores base que integran el método.

De esta manera un ECOC se puede visualizar como una matriz, en la cual los clasificadores base están denotados por columnas y las clases que puede discriminar el método están denotadas por filas. De esta manera dadas N clases a aprender un ECOC, rompe dicho problema en n problemas menos complejos (normalmente clasificación binaria) los cuales son resueltos por los clasificadores base.

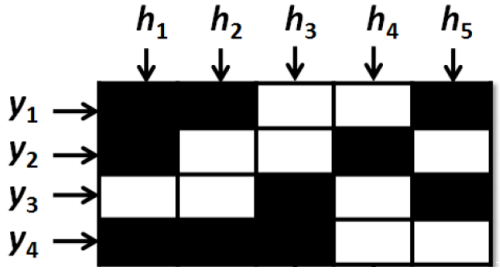


Figura 2.4: Código Corrector de Errores

En la imagen 2.4 podemos ver como los el problema general de N clases se divide en n subproblemas los cuales son resueltos por los clasificadores base, a menudo denominados hipótesis. En la imagen los h_i denotan los clasificadores base del ECOC y las y_i denotan las clases que aprende el sistema. El marco de trabajo de los ECOC se divide en dos partes:

1. Codificación: En la cual se divide el problema multiclase en distintos subproblemas de menor complejidad. Existen varias estrategias de codificación como por ejemplo *uno contra todos* o *uno contra uno*, las cuales serán explicadas en profundidad en capítulos posteriores.

2. Decodificación: En la cual se obtiene cada una de las predicciones llevadas a cabo por los clasificadores base y se ensamblan para llevar a cabo una predicción sobre el problema multiclase original. Análogamente a la codificación también existen varias estrategias de decodificado, por

ejemplo: Decodificación Hamming o Decodificación Euclídea, las cuales también serán analizadas en profundidad en capítulos posteriores

En capítulos posteriores se revisará tanto las idiosincrasias del método como su metodología completa.

2.2. Clasificadores Base

Como hemos visto anteriormente, la idea de unir varias unidades de clasificación base para obtener un herramienta que permita resolver problemas de multiclasificación esta generalmente extendida y es una de las estrategias de aprendizaje más efectivas y robustas, de hecho el *aprendizaje por ensamblado* es un campo de investigación dentro del aprendizaje automático que estudia estos métodos. El objetivo de este apartado es describir el estado del arte de los clasificadores base, es decir, clasificadores capaces de resolver problemas binarios (dos categorías a distinguir), así como analizar sus ventajas e inconvenientes.

2.2.1. Clasificador ingenuo de Bayes

Los clasificadores bayesianos ingenuos son clasificadores simples basados en probabilidades. En concreto estos clasificadores están basados en la teorema de Bayes, asumiendo una fuerte independencia entre las características de los ejemplos de las categorías de aprendizaje. En otras palabras, todas las características de un ejemplo contribuyen independientemente a la probabilidad de que ese ejemplo pertenezca a una categoría o no.

En estos tipos de clasificadores basados en probabilidades, la práctica común es utilizar un algoritmo que adapte los datos a un modelo estadístico determinado. Esta adaptación se lleva a cabo estimando parámetros del modelo estadístico, como podrían ser la media o la desviación. Una vez obtenido dicho modelo estadístico el clasificador puede predecir si un ejemplo pertenecerá a una categoría o no.

Normalmente el clasificador bayesiano ingenuo estima los parámetros de la distribución de los datos mediante el algoritmo de *máxima verosimilitud*. Estos tipos de clasificadores son muy robustos cuando el conjunto de aprendizaje es pequeño dado que al asumir que las variables son independientes, sólo se ha de estimar la varianza de cada categoría, no la matriz de covarianza completa.

2.2.2. Clasificadores Decision Stump y Adaptative boosting

Dentro de el estado del arte de los clasificadores base, los clasificadores Decision Stump combinados mediante Adaptative Boosting (en adelante AdaBoost) son unos de los más utilizados. Demostrando su eficacia en trabajos tan diversos como la detección facial o la detección de spam en el correo electrónico. Adaboost fue formulado por Yoav Freund y Robert Schapire en 1997. Este algoritmo se basa en el paradigma de aprendizaje por ensamblado, en el cual un problema complejo se resuelve combinando soluciones a subproblemas de menor complejidad extraídos del problema original.

La idea de adaboost es la combinación de una serie clasificadores lineales (también conocidos como hipótesis). El objetivo de lo cual es generar un clasificador con una capacidad de generalización mayor. Para obtener dicho clasificador (clasificador fuerte) se requiere de un proceso iterativo, en el que en cada iteración se actualizan unos pesos que corresponden al acierto de los clasificadores débiles con respecto al conjunto de aprendizaje.

El punto fuerte de este algoritmo es que a los clasificadores débiles que fallaron su predicción en la iteración n verán incrementado su peso. De manera que en la iteración $n+1$ se les dará más importancia, haciendo que el algoritmo se concentre en los ejemplos con complejidad de clasificación mayor.

Existen varias versiones de adaboost (discreto, real, gentil, etc) pero entre ellas sólo varia la forma de calcular el peso de cada clasificador base. Por ello sólo analizaremos la primera versión del algoritmo:

1. Comenzar con distribución de pesos uniforme $w_i = 1/N, i = 1, ..., N$.
2. Repetir para $m = 1, 2, ...M$
   - Adaptar el clasificador $f_m(x) \in \{-1, 1\}$ usando los pesos w_i sobre el conjunto de aprendizaje
   - Calcular $err_m = E_w[1_{y \neq f_m(x)}], c_m = \log((1 - err_m)/err_m)$
   - Actualizar $w_i \leftarrow w_i exp[c_m \cdot 1_{y \neq f_m(x)}], i = 1, 2, ..., N$ y renormalizar tal que $\sum_i w_i = 1$
3. Solución del clasificador final $sign[\sum_{m=1}^{M} c_m \dot{f}_m(x)]$

Donde los clasificadores débiles serían los $f_m(x)$ y los pesos serían los c_m. Generalmente los clasificadores débiles son implementados mediante *bases de decisión*, que son algoritmos de clasificación binaria y lineal, los cuales simplemente tienen la restricción de dar una solución mejor que la aleatoria.

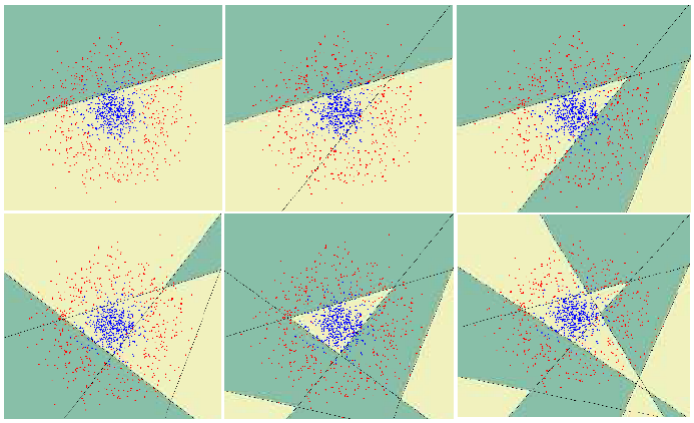


Figura 2.5: Visualización adaptative boosting

Dada su naturaleza, adaboost es sensible a problemas como el ruido en el conjunto de aprendizaje así como a los valores atípicos con relación a la distribución general de los datos. Lo cual es un problema cuando en los problemas de clasificación el número tanto de clases como de ejemplos por clase es arbitrariamente grande.

2.2.3. Clasificadores de máquinas de soporte vectorial

Las máquinas de soporte vectorial (SVM en adelante, del inglés 'Support Vector Machines') fueron formuladas en 1992, Boser, Guyon y Vapnik. La cual fue una generalización de el algoritmo propuesto por el propio Vapnik en 1963 llamado *hiperplano óptimo*. La idea general del estos clasificadores es llevar los ejemplos de las categorías de entrenamiento a un espacio de alta dimensionalidad, donde dichas categorías estén separadas por una distancia cuanto más grande mejor. En este espacio de alta dimensionalidad el SVM construirá un modelo para separar dichas categorías, este modelo será el que más distancia tenga con los ejemplos más cercanos de dichas categorías. Con lo que se reduce el error de generalización del clasificador.

Supongamos entonces que disponemos de dos categorías en nuestro conjunto de aprendizaje. El objetivo de nuestro clasificador será decidir en que categoría queda clasificado cada nuevo punto. De esta manera, los SVM ven cada punto como un vector $p-dimensional$.

La meta del algoritmo es saber si las dos categorías de aprendizaje se pueden separar con un plano $(p-1)-dimensional$, comúnmente conocido como hiperplano o clasificador lineal. Usualmente existirán un numero arbitrariamente grande de hiperplanos que cumplan estos requisitos, una buena técnica para escoger un único plano seria tomar el plano que maximice la distancia con los puntos más cercanos de ambas categorías, es decir, que maximice el margen.

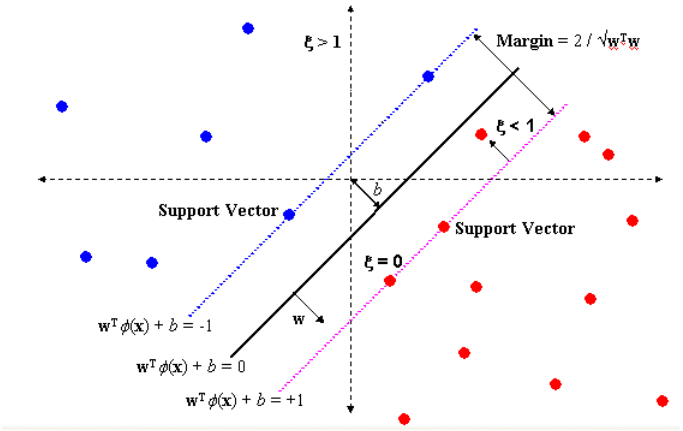


Figura 2.6: Hiperplano de margen óptimo

Más formalmente sea $S = \{(x_1, y_1), ..., (x_l, y_l)\}$ un conjunto de entrenamiento, entonces existe una función del tipo $g(x) = \langle w, \phi(x_i)\rangle + b$ la cual viene determinada por unos pesos w y un umbral b y también existe un $\gamma > 0$ tal que $\xi_i = (\gamma - y_i g(x_i))_+ = 0$ para $1 \leq i \leq l$. Lo que informalmente implica que las dos categorías del conjunto de aprendizaje son linealmente separables con un margen de γ.

En la mayoría de las ocasiones las categorías del conjunto de aprendizaje no serán linealmente separables en el espacio de entrada. Como se ha comentado anteriormente, en estos casos el algoritmo de los SVM mapa los datos en un espacio de alta dimensionalidad, es decir, ve cada punto como un vector $p-dimensional$. Para generar espacios de alta dimensión los SVM

se valen de una transformación vía unas funciones de núcleo, que dan forma al espacio de alta dimensionalidad.

Estas funciones de núcleo fuera de lo que se podría pensar son funciones comunes en el ámbito de las ciencias. Existen entre otros los núcleos polinómicos, los gaussianos y los sigmoidales. En los últimos años los SVM han sido los clasificadores base que más robustez y capacidad de generalización han obtenido, encontrándose así con la atención de la mayoría de la comunidad investigadora.

En posteriores capítulos se analiza en profundidad esta aproximación.

2.3. Requerimientos funcionales

En este apartado enunciamos los requisitos funcionales que deberá tener nuestro sistema.

- **Generar una matriz ECOC**: se considera como requisito que el sistema sea capaz de generar matrices ECOC tanto sublineales, como de codificación 'uno contra uno' o 'uno contra todos' para así poder obtener comparativas de resultados.
- **Evolucionar una matriz ECOC sublineal**: uno de los objetivos del proyecto es evolucionar una matriz ECOC sublineal para así minimizar el error de clasificación que genera.
- Evolucionar parámetros de las máquinas de soporte vectorial.
- **Evaluar una matriz ECOC**: las matrices generadas por el anterior requerimiento se tiene que poder evaluar para obtener una medida de su capacidad de generalización.
- **Generar resultados y estadísticas**: otro requisito es la posibilidad de generar estadísticas y comparativas de la codificación sublineal con las demás.

Los cuales se ven reflejados en el diagrama 2.7.

2.4. Diagrama de casos de uso

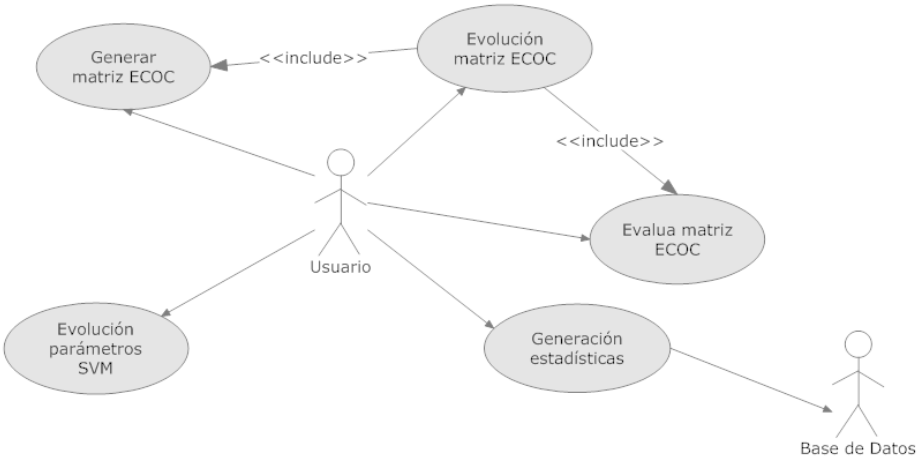


Figura 2.7: Diagrama de casos de uso del sistema

2.5. Planificación y costes

En este apartado describiremos la planificación del desarrollo del sistema así como los costes que se general en la creación del mismo. En el diagrama 2.8 vemos el tiempo que se ha invertido en cada tarea.

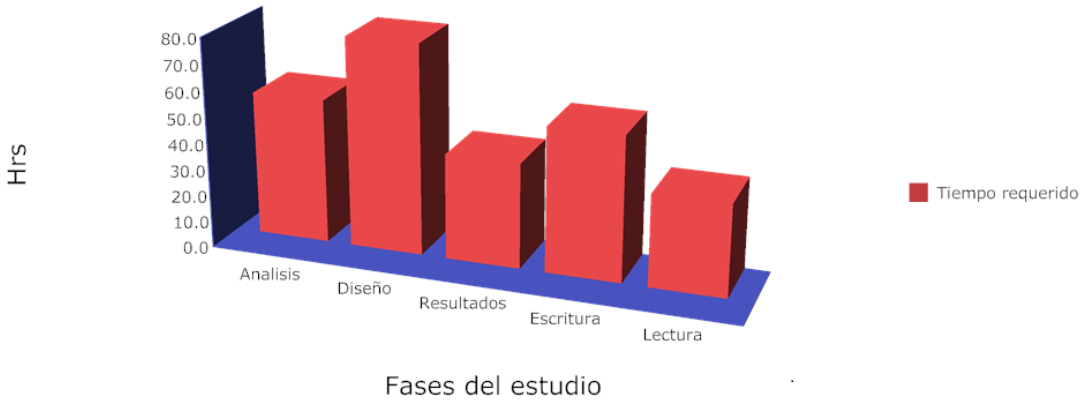


Figura 2.8: Cantidad de tiempo requerido por el estudio

De esta manera, la planificación a seguir es la que se puede ver en la imagen 2.9.

Task	Start	End	Duration	2009				2010	
				September	October	November	December	January	February
Análisis	1/9/2009	17/11/2009	55						
Diseño	1/9/2009	22/12/2009	80						
Resultados	1/12/2009	26/1/2010	40						
Escritura	1/12/2009	16/2/2010	55						
Lectura	1/1/2010	19/2/2010	35						

Figura 2.9: Planificación de tareas a llevar a cabo

En la imagen 2.8 se puede observa como las tareas que más tiempo nos han tomado han sido las de diseño y escritura. Esto ha sido debido a la complejidad del estudio y a la gran carga lectiva que suponía.

A continuación analizaremos los gastos que hubiesen supuesto la realización del proyecto. Tendremos en cuenta el coste de licencias, el de un programador analista por hora y los ítems que se han llevado a cabo, así como el hardware necesario. En la tabla 2.1 se muestran los costes generados.

Ítem/Material	Horas analista/Precio material	Licencia anual (Euros)	Total (Euros)
Codificación ECOC sublineal	15	1000	0
Integración OSU-SVM	20	0	0
Evolución SVM-RBF	20	0	0
Evolución ECOC	20	0	0
Batería Pruebas	25	0	0
Ordenador Personal	800	0	0
Disco duro externo	200	0	0
TOTAL	100×50 + 800 + 200	1000	7000

Cuadro 2.1: Tabla de costes del proyecto

Suponiendo que el precio de un analista programador por hora sea 50 euros, el coste total sería de 7000 euros.

2.6. Conclusión

En este apartado se ha revisado el estado del arte en lo que a problemas de multiclasificación se refiere. Se pretende con ello dar una amplia visión del conjunto de metodologias utilizables y así tener una argumentación sólida para tomar las decisiones de diseño en siguientes apartados.

Se ha mostrado también la planificación y costes del proyecto, y se han descrito los requerimientos funcionales del sistema, mostrando su correspondiente diagrama de casos de uso.

Capítulo 3

Diseño

El objetivo de este capítulo es analizar, en profundidad, las técnicas y métodos escogidos para resolver el problema de clasificación de gran magnitud enunciado en 1.1. Además, se hace un análisis en lo que a desarrollo del software se refiere incluyendo detallados diagramas de clases y secuencia del sistema.

Como se ha visto anteriormente, el objetivo del estudio es encontrar un aproximación viable a los problemas de multiclasificación de muy alta cardinalidad. Donde las aproximaciones anteriores quedaban obsoletas por utilizar un número de clasificadores inviable, porque los resultados obtenidos no eran aceptables o porque el costo computacional era demasiado elevado. La solución que se propone en este trabajo tiene tres puntos clave:

1. Códigos correctores de errores.
2. Clasificadores de máquinas de soporte vectorial.
3. Algoritmos genéticos.

Los cuales son analizados en profundidad a continuación.

3.1. Códigos correctores de errores

3.1.1. Introducción

Los códigos correctores de errores, previamente introducidos en 2.1.5 son un método de aprendizaje por ensamblado que ataca los problemas de multiclasificación desde una perspectiva 'divide y conquista'. Como anteriormente

se ha introducido, los problemas multiclasificación han sido abordados desde múltiples perspectivas. Sin embargo, una de las más potentes ha sido la aproximación vía aprendizaje por ensamblado. En este caso, más concretamente los códigos correctores de errores son ampliamente utilizados en problemas del estado del arte, como por ejemplo la verificación facial [KGWM01].

3.1.2. Motivación

Los problemas de alta cardinalidad nos sugieren una enorme cantidad de datos con distribuciones que probablemente sean muy complejas. En [Pol06] se sugieren aproximaciones basadas en aprendizaje por ensamblado para estas situaciones. Dado que la aproximación de aprendizaje por ensamblado cumplen una serie de características muy favorables a la hora de trabajar con una gran cantidad de datos. Existen varias razones para utilizar un sistema de ensamblado:

Motivos estadísticos

Los lectores familiarizados con redes neuronales u otros sistemas de clasificación automática están muy enterados de que obtener buenos resultados sobre el conjunto de aprendizaje no significa tener una buena capacidad de generalización. Un conjunto de clasificadores con buen resultado sobre el conjunto de aprendizaje podría tener distinta capacidad de generalización.

Incluso un conjunto de clasificadores con una capacidad de generalización similar podrían tener (y muy a menudo tienen) un comportamiento erróneo en el campo, sobretodo si el conjunto de aprendizaje no es suficientemente representativo de la distribución real de los datos.

En casos como estos combinar las salidas de un conjunto de clasificadores debidamente ponderados podría reducir el riesgo de escoger un clasificador con una capacidad de generalización baja.

Grandes volúmenes de datos

En algunas situaciones el volumen de datos a procesar simplemente es muy grande para poder ser analizado por un solo clasificador, por ejemplo, en los problemas de alta cardinalidad. En estos casos, particionar el conjunto de datos en diferentes subconjuntos entrenando para cada uno de ellos un

clasificador, combinando la salida de los mismos de manera inteligente suele ser mucho más eficiente.

Dividir y conquistar

Dejando de lado el volumen de datos a procesar, algunos problemas son simplemente demasiado complejos para ser resueltos por un simple clasificador. En concreto, la frontera que separa 2 categorías puede ser extremadamente difícil de aproximar, o estar fuera del alcance del clasificador utilizado.

Asumamos tener acceso a un clasificador base que nos permite generar fronteras con forma elíptica o circular. Dicho clasificador no podría aprender la frontera que mostramos en el apartado (a) de la imagen 3.1. Consideremos entonces, una frontera generada por un conjunto de clasificadores de un ensamblado como el del apartado (b). Una predicción final basada en una votación por un número suficiente de clasificadores podría aprender una frontera compleja como ésta.

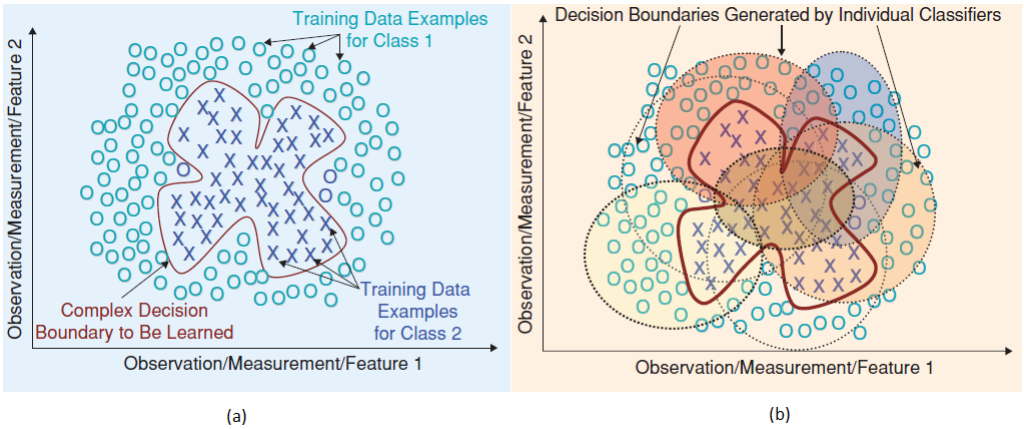


Figura 3.1: Estrategia dividir y conquistar : frontera real (a), frontera aproximada por un ensamblado de clasificadores elípticos (b)

En esencia el sistema de clasificación sigue una estrategia 'divide y conquista' donde en vez de intentar aprender todo el conjunto de datos, se aprenden subconjuntos de menor complejidad. De esta manera la frontera inicial compleja puede ser aproximada por un conjunto de clasificadores combinados apropiadamente.

3.1.3. Metodología

Supongamos pues, un problema de multiclasificación en el cual el sistema debe aprender a discernir N clases. El marco de trabajo de los ECOC atacaría el problema dividiéndolo en n subproblemas (biparticiones de las N clases) que presumiblemente serían de menor complejidad.

Estos n subproblemas de menor complejidad serían entrenados mediante clasificadores base, los más relevantes de los cuales son introducidos en 2.2. Como resultado de las biparticiones generadas para dividir el problema, cada clase recibe una *palabra clave*. Donde en cada posición se encuentra un bit $b \in \{+1, -1\}$ en función del conjunto de categorías a la que pertenece.

Tomando cada palabra clave como fila de una misma matriz, obtenemos la matriz de codificación M donde cada posición $M_{Nn} \in \{-1, +1\}^{Nxn}$.De esta manera, se obtiene una matriz que simplifica el problema de multiclasificación inicial en varios subproblemas de clasificación binaria. Como podemos ver en 3.2, donde para un problema de multiclasificación de 4 categorías se generan 5 biparticiones distintas, cada una de ellas aprendidas un por un clasificador binario h_n. Cada categoría tiene asignado su código de palabra y_N que se determina en función de como se codifique la matriz.

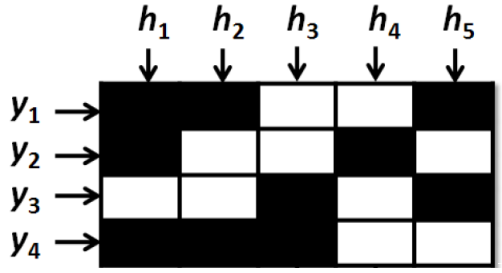


Figura 3.2: Código corrector de errores

De esta manera el marco de trabajo de los ECOC se divide en 2 etapas:

- Codificación : proceso mediante el cual se obtiene la matriz de codificación M.
- Decodificación: procedimiento con el que se obtienen las predicciones nuevas instancias.

Codificación

Existen diferentes alternativas en lo que a metodología de codificación se refiere. En esta sección se explican las más importantes y se introduce la codificación utilizada en el proyecto.

En [DB94] se enuncian las características de una buena codificación:

- Cada palabra clave deberá estar correctamente separada en términos de distancia Hamming (diferencia en los valores de los bits) con el resto de palabras clave.
- No deben existir dos clasificadores binarios que aprendan a discernir entre la misma agrupación de clases.

En lo que a etapa de codificación se refiere, existe varias alternativas a la hora de obtener la matriz de codificación. Se introducen las más conocidas:

- Uno contra todos.
- Uno contra uno.
- Aleatorio denso.

La estrategia 'uno contra todos' es una de las más utilizadas. Consiste en entrenar cada clasificador base para distinguir una categoría del resto. Dadas N clases, esta estrategia establece un número de N clasificadores, es decir, tiene una longitud de palabra clave de N bits. Como se puede observar para problemas de alta dimensionalidad se han de entrenar el mismo número de clasificadores como clases ha de aprender a distinguir. Así, en [CS00] se demuestra que la computación exhaustiva de la matriz de codificación es NP-Hard con el número de clases.

Por otra parte existen las codificaciones aleatorias como, por ejemplo, la 'aleatoria densa', en la que se generan un alto número de matrices de codificación de longitud n, donde los valores binarios $-1, +1$ tienen la misma probabilidad de aparecer. Estudios sobre el rendimiento de dichas matrices de codificación han sugerido una longitud de palabra clave $n = 10\ \lg(N)$, donde N es el número de clases. Así, mediante esta técnica de codificación se consigue una notable mejora en lo que a número de clasificadores se refiere y que repercute en el tiempo de cómputo global. Para el conjunto generado

de matrices, la matriz óptima es la que maximiza la distancia Hamming, teniendo en cuenta que cada columna ha de tener los dos valores $\{+1, -1\}$.

Otra estrategia de codificación es la conocida como 'uno contra uno' en la cual se incorpora un tercer símbolo en el diseño de las matrices M, el símbolo cero, que implica que determinadas clases no son consideradas en las agrupaciones entrenadas por el clasificador base. No obstante, independientemente del rendimiento, el uso de tres símbolos requiere incrementar la longitud de los códigos para obtener resultados robustos. Dado que uno de los objetivos de este proyecto es reducir la longitud de los códigos para conseguir trabajar con grandes cantidades de clases, en este estudio no revisaremos el caso 'ternario' de los ECOC , aunque este tipo de codificación es utilizada en 4 para comparar resultados.

Vistas algunas las alternativas del tipo de codificación del estado del arte, en 3.1.3 nos centramos en la codificación que utilizará nuestro sistema.

Decodificación

En el proceso de decodificación, para cada nuevo dato a clasificar, cada clasificador binario entrenado para discernir entre los grupos de clases designados por el ECOC, predice un valor. Una vez obtenidas las predicciones de cada clasificador base, se forma una palabra clave. Finalmente cada palabra clave obtenida se asigna la clase con una palabra clave más 'parecida', según la métrica escogida, como podemos ver en 3.3.

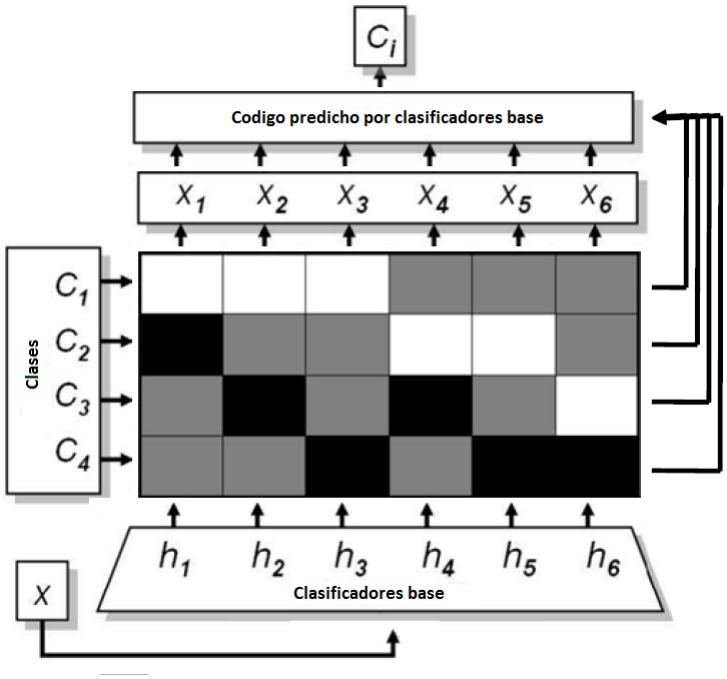


Figura 3.3: Decodificación de un código corrector de errores

Como se comentó con anterioridad, uno de los objetivos de este estudio es utilizar un número reducido de clasificadores, con lo cual la longitud de la palabra clave también será reducida. Consecuentemente deberemos tener especial cuidado a la hora de escoger la alternativa de codificación y decodificación. En este sentido, basándonos en la reglas generales de la etapa de decodificación, las métricas se pueden dividir en dos grupos, dependiendo de como se ataque el problema.

El primer grupo trataría de resolver el problema desde la perspectiva del cálculo de la distancia entre la predicción llevada a cabo por los clasificadores y la palabra clave destinada a ese ejemplo. El segundo grupo, atacaría el problema desde la probabilidad de cada ejemplo de pertenecer a una clase en concreto. El primer intento de decodificación de un ECOC es la decodificación Hamming. Otra de las mas conocidas es la decodificación Euclidiana. En [WG02] se introduce el estudio de la codificación Hamming invertida así como la decodificación centrada para diseños de ECOC binarios.

Analizamos pues tres alternativas en la etapa de decodificación, del grupo

que trata el problema desde la perspectiva de la distancia entre la palabra predicha y la palabra clave de cada clase. Así pues, analizaremos: la decodificación Hamming, la decodificación Hamming Inversa y la decodificación Euclídea.

- **Decodificación Hamming (HD)**: en este tipo de decodificación se calcula la distancia Hamming entre la palabra predicha por los clasificadores del ECOC y las distintas filas de la matriz M. Donde finalmente se dará como predicción la clase con menor distancia a la palabra clave predicha. Esta métrica es la utilizada en la implementación del sistema.

$$HD(x, y_i) = \sum_{j=1}^{n}(1 - sign(x^j \cdot y_i^j))/2 \tag{3.1}$$

- **Decodificación Hamming Inversa (IHD)**: esta alternativa de decodificación produce una matriz con los valores de proporcionalidad de cada clase en la palabra predicha por los clasificadores de los ECOC. En lo que a la práctica se refiere la IHD tiene un comportamiento muy similar al comportamiento de la estrategia HD.

- **Decodificación Euclídea (ED)**: esta métrica se basa en calcular la distancia Euclídea de la palabra predicha por los clasificadores del ECOC con las diferentes filas de la matriz M.

$$ED(x, y_i) = \sqrt{\sum_{j=1}^{n}(x^j - y_i^j)^2} \tag{3.2}$$

Código corrector de errores sublineal

Los códigos correctores de errores se basan en los principios de la teoría de la comunicación en la cuales un conjunto de bits es enviado por un canal con la agregación de una serie de bits que indican si se ha producido un error en la transmisión. La idea de la aplicación de este principio en los problemas de clasificación es poder resolver los posibles fallos de los clasificadores base que integran el ensamblado.

De esta manera, para identificar 2 palabras distintas que viajen por el canal necesitamos que al menos difieran en un bit. Esto puede ser estimado

computando la distancia Hamming mínima entre todas las parejas mediante 3.3.

$$d_r = min_{i_1,i_2} \{\sum_{j=1}^{n}(1 - sign(y_{i1}^j \cdot y_{i2}^j))/2\} \tag{3.3}$$

Supongamos ahora que tenemos 2 palabras clave que tienen distancia Hamming 3. Esto significa que aunque variemos un bit (en error de clasificador base) aún podremos distinguir las 2 palabras. Esto sugiere que una matriz ECOC M puede corregir $[d_r - 1]/2$ errores de los clasificadores base. Es decir, en el ejemplo anterior, se podría clasificar de forma correcta un dato aunque un clasificador base fallara en su predicción.

De esta manera la literatura sugiere un código de longitud grande para poder corregir todos los errores posibles en el proceso de decodificación. En la últimos años se ha hecho un gran esfuerzo para mejorar la robustez de los clasificadores base y así poder reducir la longitud de la palabra. De esta manera la codificación 'uno contra todos' introducida en 3.1.3 reduce el número de clasificadores base a N. En [RK04] se argumentan resultados similares a las demás estrategias si los clasificadores están correctamente adaptados. En estudios más recientes dicha longitud se ha conseguido reducir a $N - 1$ con la estrategia DECOC [PRV06].

Aunque ambas aproximaciones ofrecen la mínima longitud de palabra clave actual podemos aprovechar los principios de la teoría de la comunicación para obtener una codificación mucho más compacta de los códigos o palabras clave. Suponiendo que la matriz ECOC aprende a distinguir N clases la definición más compacta de palabra clave que podemos utilizar es de longitud B, donde $B = \lceil log_2(N) \rceil$.

Notar que esta codificación representa la codificación de mínima longitud de ECOC existente. En el proyecto se propone codificar la matriz ECOC sublineal binaria, así como la matriz ECOC sublineal gray. En la imagen 3.4 vemos la comparación de dichas codificaciones con la codificación uno contra todos.

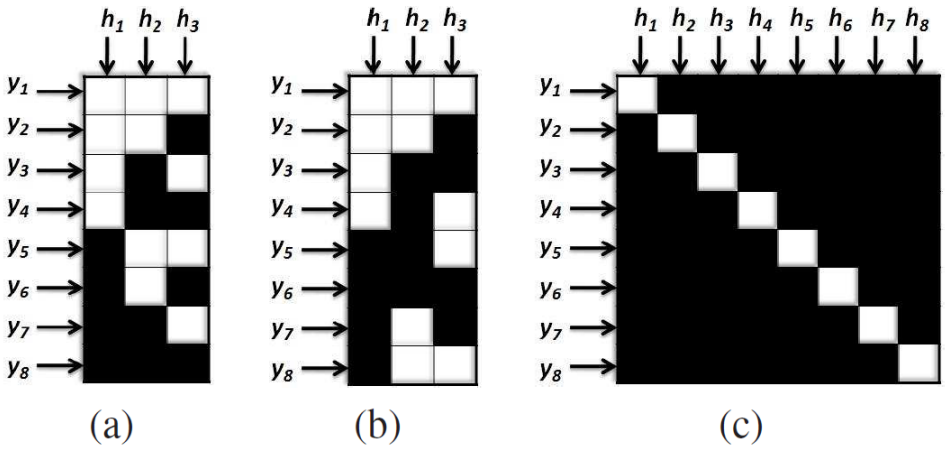


Figura 3.4: Codificaciones ECOC (a) Binario Sublineal, (b) Gray Sublineal, y (c) uno contra todos, para un problema de 8 clases.

Así pues, utilizando este esquema de codificación obtenemos el número mínimo de clasificadores necesarios en nuestro ECOC para poder distinguir las N clases que aprenderá a discriminar el sistema. El proceso para obtener un ECOC sublineal es el siguiente:

- **Entrada**: número de clases del problema a tratar (N)
- Generar una permutación aleatoria de números naturales de 1 a N
- **Repetir**:
- Particionar por el punto central la/s cadena/s de números naturales
- **Hasta**: que no se puedan particionar en la partición i las particiones generadas por la $i-1$ de la estructura de árbol resultante codificar el nivel i como la columna i, donde el hijo izquierdo tendrá valor 1 y el derecho valor -1
- **Salida**: codificación del ECOC Sublineal

En un principio se puede pensar que ésta no es una mejora sustancial cuando se trabaja con un número pequeño de clases. Nada más alejado de la realidad, se estaría en lo correcto, dado que la potencia real de esta alternativa

de codificación reside cuando N es arbitrariamente grande. Es aquí donde la potencia de un codificación sublineal se hace notar como vemos en 3.5.

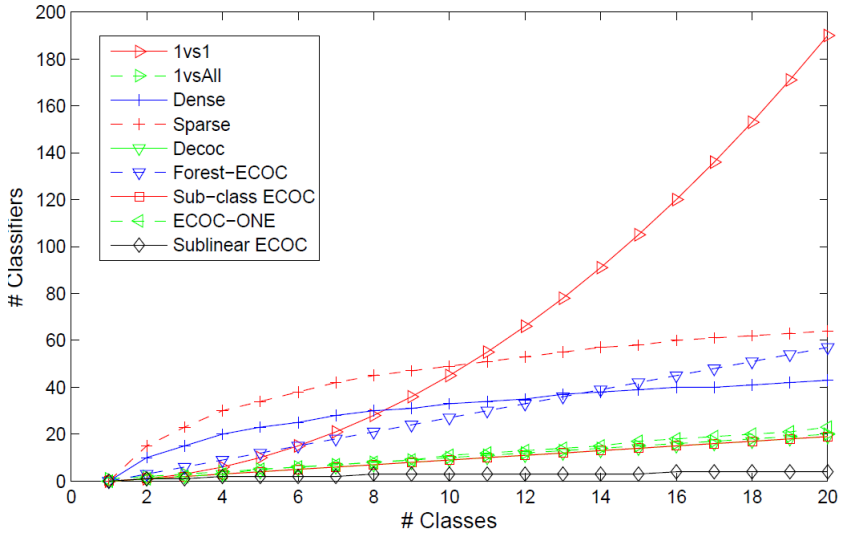


Figura 3.5: número mínimo de clasificadores base requerido por cada codificación ECOC en función del número de clases

Así pues, proponemos esta nueva codificación como solución al problema de escalabilidad de los ECOC.

3.2. Clasificadores de máquinas de soporte vectorial

3.2.1. Introducción

Las máquinas de soporte vectorial (SVM en adelante, del inglés 'Support Vector Machines') fueron formuladas en 1992, por Boser, Guyon y Vapnik. Lo cual fue una generalización de el algoritmo propuesto por el propio Vapnik en 1963 llamado *hiperplano óptimo*. La idea general del estos clasificadores es llevar los ejemplos de las categorías de entrenamiento a un espacio de alta dimensionalidad, donde dichas categorías estén separadas por una distancia cuanto más grande mejor. En este espacio de alta dimensionalidad el SVM

construirá un modelo para separar dichas clases, este modelo será el que más distancia tenga con los ejemplos más cercanos de dichas categorías. Con lo que se reduce el error de generalización del clasificador.

Dichos clasificadores han sido utilizados en múltiples entornos dando en la mayoría de los casos resultados muy satisfactorios. Como por en ejemplo en, [BZ09] donde los autores utilizan estos clasificadores para predecir la posición de proteínas subcelulares. Otro trabajo interesante es la aplicación al reconocimiento de imágenes en [ACSS02].

3.2.2. Motivación

Los problemas de gran cardinalidad y la codificación de los códigos correctores de errores propuesta, nos sugieren la utilización de un número muy compacto de clasificadores base (logarítmico con el número de clases). Esto se puede hacer ya que se asume la utilización de unos clasificadores base muy robustos, que den los mejores resultados posibles, ya que la codificación de ECOC elegida no puede corregir un gran número de errores.

Además, las máquinas de soporte vectorial tienen una dependencia explícita en los datos (mediante los vectores de soporte), de esta manera el modelo es fácilmente interpretable, lo cual es un punto a favor cuando se maneja una gran cantidad de datos. En contrapunto con los redes neuronales en las cuales la dependencia del modelo con los datos no es tan clara.

Por otra parte la tarea de aprendizaje en un SVM involucra la optimización de una función convexa en la que no puede haber mínimos relativos, sólo existe un mínimo absoluto. En contrapartida con las redes neuronales, que entrenadas mediante el algoritmo de propagación hacia atrás 2.3 puede estancarse en un óptimo relativo.

El último punto a favor es que se requieren pocos parámetros a ajustar, normalmente el parámetro del margen suave y el parámetro del kernel. Al contrario que en los sistemas de redes neuronales donde se ha de ajustar la arquitectura y varios parámetros.

3.2.3. Metodología

Las máquinas de soporte vectorial o SVM son una generalización del algoritmo propuesto por Vapnik en 1963 llamado *hiperplano óptimo*. Supongamos que disponemos de dos categorías en nuestro conjunto de aprendizaje. El objetivo de nuestro clasificador será decidir en que categoría queda clasificado

cada nuevo punto. De esta manera, los SVM ven cada punto como un vector $p - dimensional$.

La meta del algoritmo es saber si las dos categorías de aprendizaje se pueden separar con un plano $(p-1) - dimensional$, comúnmente conocido como hiperplano o clasificador lineal. Generalmente, existirán un número arbitrariamente grande de clasificadores lineales que cumplan estos requisitos, una buena técnica para escoger un único plano sería tomar el plano que maximice la distancia con los puntos más cercanos de ambas categorías, es decir, que maximice el margen.

Caso lineal: hiperplano óptimo

Asumamos que las categorías son perfectamente separables en el espacio de entrada. Es decir, se pueden separar mediante un clasificador lineal. Formalmente, sea 3.4 el conjunto de datos x_i y $y_i \in \{-1, +1\}$ sus correspondientes etiquetas, entonces existe una función lineal 3.5 basada en unos pesos w. De esta manera nuestra frontera de decisión seria 3.6, de lo que podemos deducir que permanece invariable a un rescalado tanto de w_i como de b 3.7.

$$S = \{(x_1, y_1), ..., (x_n, y_n)\} \tag{3.4}$$

$$g(x) = \langle w \cdot x_i \rangle + b \tag{3.5}$$

$$D(x_i) = sign(\langle w \cdot x_i \rangle + b) \tag{3.6}$$

$$w \to \lambda w, b \to \lambda b \tag{3.7}$$

Una vez obtenida la frontera de decisión, el siguiente paso es maximizar el margen γ que deja ésta con los puntos más cercanos de las categorías a separar. Geométricamente, el margen viene definido por la proyección del vector $x_1 - x_2$ (donde x_1 es un vector de soporte perteneciente a categoría $+1$ y x_2 es un vector de soporte perteneciente a categoría -1) en la normal al clasificador lineal, como podemos ver en la imagen 3.6.

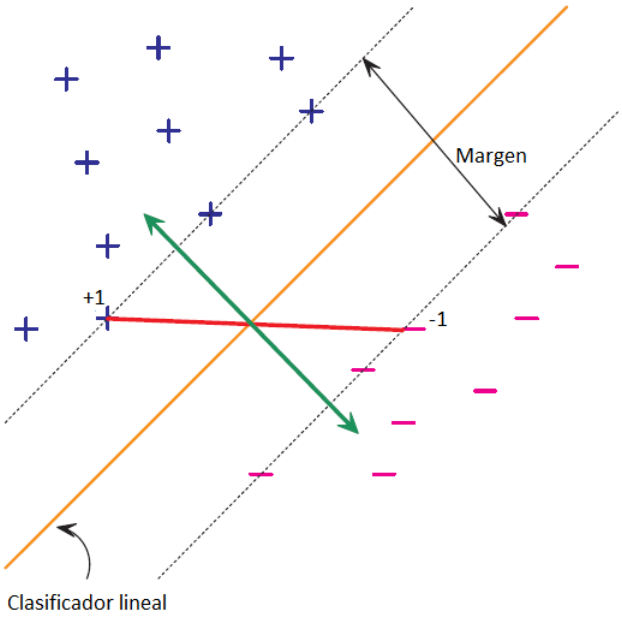


Figura 3.6: Margen de una máquina de vectores de soporte

Formalmente el margen puede ser formulado como:

$$\gamma = 1/\|W\|_2 \tag{3.8}$$

De esta manera la tarea de maximizar el margen formulado en 3.8 es análoga a minimizar 3.9 sujeta a la restricción 3.10.

$$\phi(w) = \frac{1}{2}(w \cdot w) \tag{3.9}$$

$$y_i[(w \cdot x_i) + b] \geq 1 \tag{3.10}$$

Es decir, el problema se traduce en encontrar un óptimo de la función de coste 3.11, donde las $alpha_i$ son los multiplicadores de Lagrange.

$$L(w, b) = \frac{1}{2}(w \cdot w) + \sum_{i=1}^{m}(\alpha_i[y_i((w \cdot x_i) + b) - 1]) \tag{3.11}$$

Uno de los aspectos más interesantes de la optimización de este funcional, es que los datos (vectores x_i) aparecen en producto escalar con los pesos. Con lo cual el proceso de optimización de dicho funcional es independiente de la dimensión que tengan los datos.

Normalmente no nos encontraremos en situaciones donde los datos sean linealmente separables, ya que por ejemplo en nuestro sistema, cada clasificador base de la matriz ECOC contará con 2 conjuntos de categorías muy grandes a distinguir, con lo cual se puede asumir que la distribución será muy compleja. Es aquí las máquinas de soporte vectorial son útiles dado que como se ve en 3.11 la dimensionalidad no afecta en la optimización del margen. En estas situaciones donde una clasificación lineal es inapropiado en el espacio de entrada, los SVM pueden mapear los datos en un espacio de alta dimensión, donde sí sean linealmente separables.

Casos no lineales en el espacio de entrada: mapeado en espacios de alta dimensión

Como se ha explicado anteriormente, muchas serán las situaciones en las que las 2 categorías a distinguir por un SVM no sean perfectamente separables en el espacio de entrada. Por la propiedad vista en 3.11 las máquinas de soporte vectorial pueden mapear esos datos que no son separables linealmente, en un espacio de mayor dimensión (también denominado espacio de características) donde un hiperplano pueda separar perfectamente ambas categorías.

Formalmente, para obtener una mejor representación de los datos cuando éstos no son linealmente separables en el espacio de entrada, podemos proyectarlos en un espacio de alta dimensionalidad mediante 3.12. Es decir, utilizamos un mapeo del tipo 3.13

$$x_i \cdot y_j = \phi(x_i) \cdot \phi(y_j) \tag{3.12}$$

$$x_i \rightarrow \phi(x_i) \tag{3.13}$$

La única condición necesaria para poder realizar este mapeo, es que el espacio de características donde se mapeen los datos ha de ser un espacio de Hilbert (aunque en algunos casos un espacio pre-Hilbert podría ser suficiente). Este tipo de espacio es una generalización $n-dimensional$ de un espacio euclídeo. En el cual un producto vectorial esta completamente definido.

De esta manera la función de mapeo $\phi(x_i) \cdot \phi(x_j)$ será un kernel. Por lo tanto podemos definir un kernel como el producto vectorial de puntos mapeados en el espacio de alta dimensión 3.14.

$$K(x_i, y_j) = \phi(x_i) \cdot \phi(y_j) \tag{3.14}$$

Así, no necesitamos saber que forma tiene el mapeado de datos en el espacio de alta dimensión ya que ésta, estará implícitamente definida en el cálculo del producto vectorial en el espacio de características.

Existen muchos tipos de kernels pero para este estudio nos centraremos en el kernel RBF Gaussiano, el cual será revisado en profundidad en posteriores apartados.

- **Kernel polinomial**: El kernel polinomial es uno de los más usados para modelar un problema de manera no lineal. Este se basa en concebir una función polinomial de grado d (donde d es la dimensión del espacio del kernel) que consiga una clasificación aceptable de los datos de entrada: véase en 3.7, como dicho kernel establece una clasificación mucho más aceptable que cualquier clasificador lineal por sí solo. No obstante en la imagen podemos observar que no es la mejor clasificación posible, sino que se puede refinar.

$$K(x, x') = (\langle x, x' \rangle + 1)^d \tag{3.15}$$

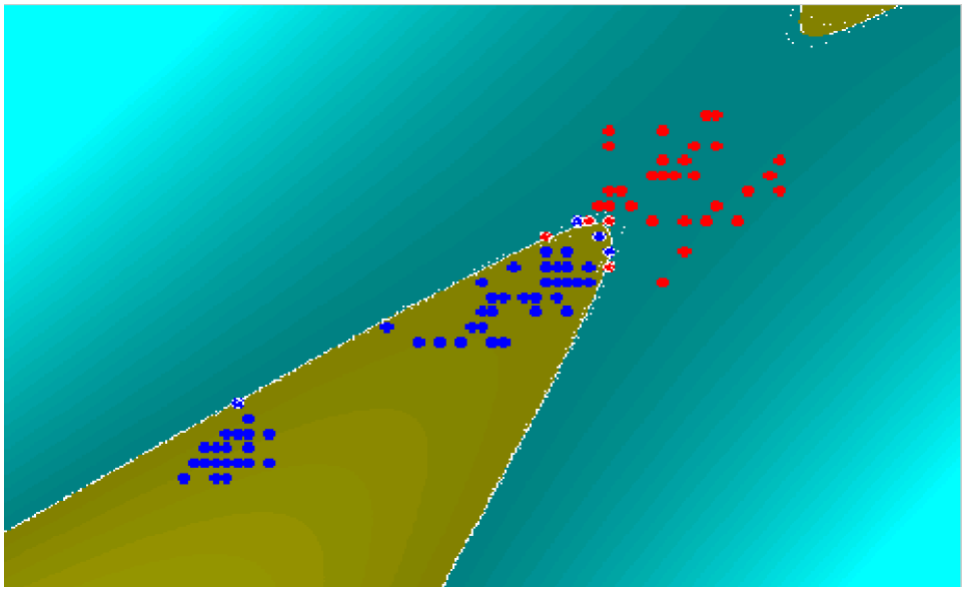


Figura 3.7: Frontera de decisión generada por un kernel polinomial

- **Kernel RBF**: Otro tipo de kernel son los de función de radio base 3.16 (del Inglés 'Radial Basis Function'). Este tipo de kernels han recibido especial atención dado su alto rendimiento incluso en casos de muy alta cardinalidad, como es el nuestro. Nos centraremos en el caso gaussiano, donde se requiere de la adecuación de un parámetro para realizar la clasificación de forma óptima, el denominado parámetro γ. Se puede observar en 3.8 que para la distribución de datos, un kernel RBF tiene un rendimiento similar o ligeramente superior a un kernel polinomial. Generalmente el rendimiento del kernel RBF es mejor que el de un kernel lineal.

$$K(x, x') = \exp(\frac{-\|x - x'\|^2}{2\sigma^2}) \tag{3.16}$$

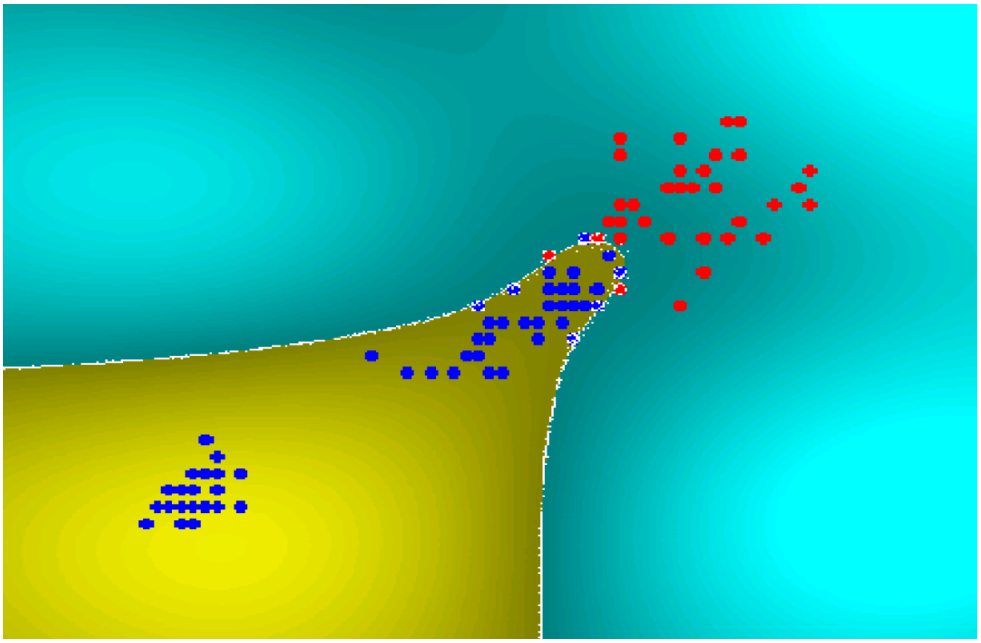


Figura 3.8: Frontera de decisión generada por un kernel RBF gaussiano

Matemáticamente, para que un kernel sea valido tiene que cumplir la condición de Mercer, formalmente sea $K(x, x')$ un kernel, será valido:

$$\int K(x, x')g(x)g(x')\ dxdx' \geq 0 \tag{3.17}$$

En otras palabras, el kernel ha se de ser semidefinido positivo. Ya que si se dan las condiciones 3.18 y 3.19, entonces existe una función del tipo 3.20 la cual genera un espacio de alta dimensión de Hilbert con un producto interior completamente definido.

$$\sum_{i,j} K(x_i, x_j)c_i c_j \geq 0 \tag{3.18}$$

$$\{c_1, ..., c_n\} \ \in \ \mathbb{R} \tag{3.19}$$

$$K(x, y) = \phi(x) \cdot \phi(y) \tag{3.20}$$

Existen algunos kernels, como los sigmoidales, que violan la condición de Mercer. No se consideran dentro del ámbito del estudio y por lo tanto no son introducidos.

Hiperplano de margen suave

Uno de los problemas a resolver en tareas de clasificación automática es el sobreeentrenamiento. Esta situación tiene lugar cuando el sistema es entrenado de forma muy concisa sobre un conjunto de datos, de tal manera que aprende las idiosincrasias de la distribución. Como podemos ver en la imagen 3.9, donde la frontera en color negro representa una buena medida de generalización. Por otra parte la frontera en verde se ajusta tanto a los valores atípicos que hace disminuir su capacidad de generalización.

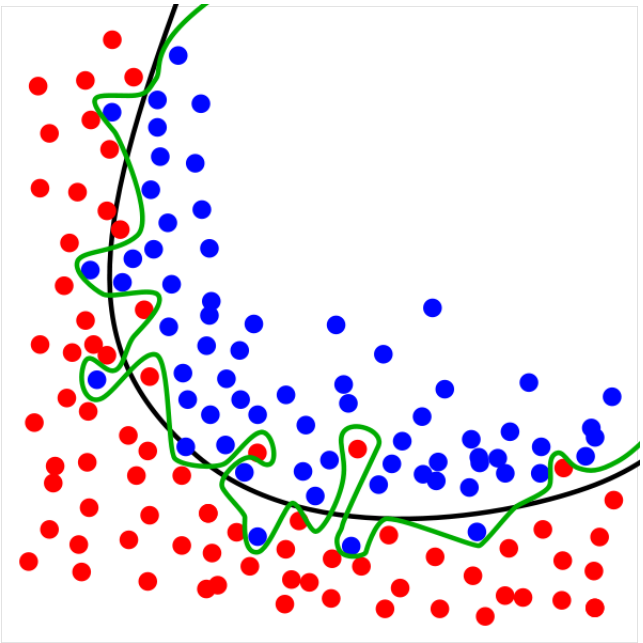


Figura 3.9: Situación de sobreeentrenamiento

El sobreeentrenamiento hace que se pierda una gran parte de la capacidad de generalización del clasificador, llevando a resultados de clasificación generalmente pobres. Normalmente las idiosincrasias de la distribución suelen ser valores atípicos que aparecen debido al ruido de los datos.

La estrategia para resolver este problema en las máquinas de soporte vectorial es la introducción de un margen suave, el cual se puede modificar para minimizar el error de clasificación o en otras palabras, el sobreeentrenamiento.

Formalmente, lo que queremos conseguir es acotar la frontera superior del valor de los multiplicadores de Lagrange (el valor de los cuales se puede interpretar como la importancia que tienen para el hiperplano) α_i por un valor C, como en 3.21. Para ello introducimos una variable débil positiva ξ_i obteniendo 3.22, teniendo que reformular la función de coste para llegar a 3.23 donde $\alpha_i \geq 0$ y $r_i \geq 0$ son los multiplicadores de Lagrange.

$$0 \geq \alpha_i \geq C \tag{3.21}$$

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \tag{3.22}$$

$$L(w, b, \alpha, \xi) = \frac{1}{2}(w \cdot w) + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i[y_i((w \cdot x_i) + b) - 1 + \xi_i]) - \sum_{i=1}^{m} \xi_i r_i \tag{3.23}$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{m} \alpha_i x_i y_i = 0 \tag{3.24}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{3.25}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - r_i = 0 \tag{3.26}$$

Tomando derivadas parciales de la nueva función de coste obtenemos 3.26 de la cual podemos deducir 3.21.

De esta manera conseguimos un hiperplano en el cual se pueden acotar la importancia máxima que tengan algunos puntos sobre el mismo y así controlar el sobreeentrenamiento.

Máquina de soporte vectorial con kernel RBF

En la literatura las máquinas de soporte vectorial con un kernel RBF han obtenido resultados robustos. Más en concreto, los kernel RBF Gaussianos han obtenido los mejores rendimientos en lo que a clasificación binaria se refiere. Esto no es una coincidencia ya que en este tipo de funciones se han venido utilizando en redes neuronales des de hace tiempo.

Una función de base radial o RBF, es una función de valores reales cuyos valores dependen solo de la distancia desde el origen $\phi(x) = \phi(\|x\|)$ o alternativamente desde un punto fuente c llamado el centro, $\phi(x, c) = \phi(\|x - c\|)$. Cualquier función $\phi(x)$ que satisfaga la propiedad $\phi(x) = \phi(\|x\|)$ una función

RBF. La métrica para el cálculo de la distancia suele ser la euclídea, aunque existen otro tipo de métricas. La función RBF mas robusta en las maquinas de soporte vectorial es la gaussiana formulada en 3.27.

$$\varphi(r) = \exp(-\beta r^2) \; para \; \beta > 0 \tag{3.27}$$

3.3. Métodos evolutivos: algoritmos genéticos

3.3.1. Introducción

Como se ha explicado a lo largo de los anteriores apartados, los puntos más importantes del sistema 3.1 o 3.2, tienen una influencia muy grande en la capacidad de generalización del sistema, como se explico previamente los parámetros C y γ de un kernel RBF tienen una importancia vital en la capacidad de generalización del clasificador. En 3.1.3 se analiza el tipo de codificación a utilizar por el código corrector de errores, es comprensible que esta codificación sublineal binaria natural no va a ser la óptima para todos los problemas. Debemos encontrar entonces una codificación binaria que pueda adaptarse a distintos problemas para maximizar la capacidad de generalización.

Esta situación también se da en la elección de los parámetros de los SVM. Donde es comprensible que cada problema binario tendrá una distribución diferente, con lo cual los parámetros del SVM se tendrán que ajustar para que el clasificador SVM obtenga buenos resultados de clasificación.

Estos problemas de ajuste o optimización de parámetros en máquinas de soporte vectorial han sido ampliamente estudiados en la literatura [BZ09] [LM06] [LdC08]. En un principio, el método usado para encontrar dichos valores fue el **descenso del gradiente**, que consiste en un algoritmo de búsqueda de un mínimo de una función. Este método lleva consigo el inherente problema del estancamiento en un mínimo relativo, sin poder llegar al mínimo absoluto 3.10. Además, estos métodos solo se pueden utilizar en casos en que la función a optimizar sea diferenciable. En el caso de los parámetros del SVM es aceptable, pero en el caso del ECOC, hay discontinuidades que limitan la utilización de métodos basados en descenso del gradiente.

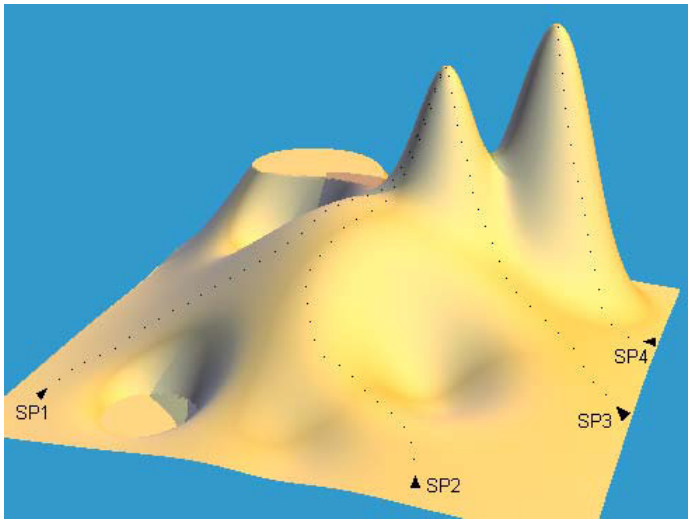


Figura 3.10: Visualización del algoritmo del gradiente descendiente

Una de las propuestas más interesantes se encuentra en [BZ09], donde los autores utilizaron el algoritmo de Levenberg-Marquardt para minimizar el error de clasificación. En su estudio de la optimización de los SVM los autores propone la utilización de dicho algoritmo, obteniendo algunos buenos resultados de optimización. Esto es debido a que dicho algoritmo es una interpolación entre el antes comentado descenso del gradiente y el método de optimización Gauss-Newton. De esta manera, mientras la solución no esté próxima se utiliza el gradiente descendiente y a medida que se aproxima a un mínimo se pone en marcha la minimización Gauss-Newton. [RG04] es otro trabajo donde encontramos la combinación de métodos para reducir el error de clasificación. [RG04] propone un modelo de selección basado en la metodología de minimización de la probabilidad del error estimado en el conjunto de validación. Dicho método incluye un proceso automático de minimización que reduce eficientemente el error empírico, optimizando localmente el conjunto de clasificadores SVM estimando la probabilidad del error en un conjunto de validación.

Vistas las alternativas en el estado del arte de la optimización de los kernels del SVM, en el desarrollo del proyecto se ha optado por una nueva alternativa, delegar la optimización tanto de los parámetros de la maquina de vectores de soporte, como la codificación de la matriz sublineal de ECOC en un algoritmo genético. Dicho algoritmo realizará una computación no

exhaustiva del problema.

3.3.2. Motivación

Como se ha anunciado en capítulos anteriores el computo de una matriz ECOC es NP-Hard con el numero de clases [CS00] . Es decir, haciendo inviable la utilización de métodos analíticos.. A partir de esta situación y teniendo en cuenta que la topología del espacio de soluciones es completamente desconocida una buena alternativa es utilizar algoritmos genéticos para encontrar soluciones.

De esta manera el objetivo de el algoritmo genético será la optimización de la codificación de la matriz ECOC sublineal para ajustarla al tipo de problema que se tiene que resolver. Para proseguir con la explicación de la motivación se ha de tener claro que el numero de clasificadores SVM que tendrá cada matriz de ECOC sublineal es, valga la redundancia, logarítmico con el numero de clases.

En el proceso de evolución (búsqueda de la matriz ECOC mejor ajustada al problema) se generaran múltiples matrices ECOC, las cuales son evaluadas según su grado de adaptación al problema. De esta manera para optimizar los parámetros de los clasificadores SVM de los ECOC se ha de encontrar un método de optimización rápido (ya que se pretende que el método sea viable en términos temporales) y que permita encontrar soluciones validas. Una vez mas, los Algoritmos Genéticos nos permitirán encontrar estos parámetros en un tiempo aceptable.

Aunque en los apartados siguientes se desarrolla más profundamente el método, es importante, comentar que en ningún caso la utilización de algoritmos genéticos asegura encontrar la solución óptima. En ciertas ocasiones se podrán encontrar pero en algunas no se alcanzarán, en cualquier caso en el 4 se analizan más profundamente los resultados obtenidos, los cuales muestran que en ocasiones las soluciones sub-óptimas son suficientemente aceptables.

3.3.3. Metodología

Introducción

El proceso de optimización de los clasificadores SVM así como el de la codificación de un ECOC sublineal se ha delegado a un Algoritmo Genético. Este tipo de algoritmo se basa en un proceso evolutivo, que no es más que

un tipo de algoritmo dentro del conjunto de la computación evolutiva. Es este apartado se analizan, dichos algoritmos profundamente, así como su aplicación en el sistema. La computación evolutiva es un amplio conjunto de métodos inspirados en la teoría de Darwin [Dar]. La evolución natural se puede observar como un proceso de optimización basado en la población. Es decir, un proceso iterativo de adaptación al medio por parte de la población existente. El proceso iterativo consta de diferentes etapas que ocurren una después de otra, conformando lo que se conoce como una *máquina genética de Darwin* 3.11.

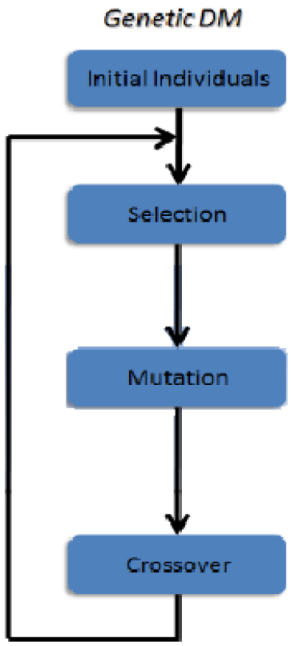


Figura 3.11: Máquina genética de Darwin

En la naturaleza todo ser viviente está 'codificado' en términos de su material genético. Dicho material genético es utilizado como los ingrediente de una receta a la hora de crear un nuevo ser vivo. Así pues, la adaptabilidad de un cierto sujeto al medio vendrá dada por su material genético, y éste a su vez vendrá dado por el material genético de sus predecesores. De esta manera la evolución es el proceso que permite la utilización del material genético y su mejora, con el objetivo de una mejor adaptación al medio. Las bases de los

algoritmos evolutivos recaen sobre la 'selección natural', donde los mejores individuos pueden sobrevivir y reproducirse con el objetivo de perpetuar la especie. La información genética de dichos individuos es transmitida a lo largo de las generaciones, creando cada vez individuos más adaptados al medio y mejorando la especie. En general, en la evolución natural los individuos son clasificados por su adaptación al medio, es decir, los mejores individuos son los más adaptados al medio. A parte del paso de información genética a lo largo de las generaciones, tenemos otras dos fuentes de evolución, el apareamiento y la mutación.

- **Apareamiento o Reproducción**: en el proceso reproductivo de todo ser vivo, dos individuos de la especie intercambian material genético, las cuales son evaluadas según su grado de adaptación al problema. Cuando esa mezcla deriva en una mejor adaptación al medio, ese nuevo individuo sobrevivirá y su información genética se mantendrá a lo largo de la generación. Por el contrario si dicha mezcla deriva en un individuo peor adaptado, no sobrevivirá y esa información se perderá.

- **Mutación**: en contraste con la reproducción, la mutación es el proceso mediante el cual un nuevo material genético puede ser generado. La mutación consiste en pequeños cambios aleatorios en la información genética de un ser vivo. Análogamente a la reproducción, cuando dicha mutación conduce a un individuo mejor adaptado al medio, dicha información perdura a lo largo de las generaciones, en caso contrario, se pierde dado que el individuo no sobrevive. Darwin explica la evolución de las especies como inferencia de dichos métodos, y posteriores estudios en el campo de la biología demuestran que esto se puede generalizar para todos los seres vivos del planeta. Así pues, dichas operaciones son utilizadas para simular un proceso evolutivo desde una perspectiva computacional.

Algoritmos genéticos

Los algoritmos genéticos son probablemente la aplicación más común de la computación evolutiva y de la simulación de la evolución natural. La idea detrás de este tipo de algoritmos es la imitación de la evolución natural mediante programas de ordenador, utilizando una representación del problema basada en la genética e implementando desde un punto de vista funcional

los diferentes métodos de evolución implicados en dicho proceso. Usualmente dichos algoritmos se utilizan en procesos de optimización, dado que los Algoritmos Genéticos pueden realizar tareas de optimización sea cual sea la topología del espacio de soluciones. Normalmente al utilizar este tipo de algoritmos sólo encontramos dos componentes que sean dependientes del problema a tratar. Éstos son, la codificación de los individuos y la función de evaluación. El resto de operadores son estándar con el tipo de codificación que se elija. En los próximos apartados definimos brevemente las distintas alternativas y nos centramos en las utilizadas en el desarrollo del proyecto.

En un algoritmo genético, los individuos que ya han sido codificados son llamados genotipos o alternativamente cromosomas ζ. Y cada característica o parámetro del cromosoma se denomina *alelo*.

Codificación

Este problema viene dado por la elección del tipo de representación de las soluciones o puntos en el espacio de soluciones por medio de genotipos o alternativamente cromosomas. Aunque la codificación depende básicamente del problema a resolver, existen un conjunto de aproximaciones estándar:

- **Codificación Binaria**: es la más utilizada dado que es la primera que se utilizó en este tipo de algoritmos. Consiste en la codificación de una cadena de bits donde cada bit indica un cierto aspecto de la solución en términos del problema. Este tipo de codificación usualmente no es natural para muchos problemas y a veces se han de corregir las operaciones de mutación y cruce. La representación binaria más útil es la conocida como codificación de Gray.

- **Codificación Permutada**: es un tipo de codificación donde cada cromosoma consiste en una cadena de números que representan un número en una secuencia. Este tipo de codificación es muy útil en problemas de ordenación.

- **Codificación por Valor**: este tipo de codificación puede ser utilizada en problemas, donde valores complicados han de ser tratados, como por ejemplo números reales. Ya que la utilización de un codificación binaria podría resultar muy tediosa. Los valores pueden ser cualquier tópico relacionado con el problema. Este tipo de codificación puede ser muy útil para algunos tipos de problemas. Por otra parte, normalmente

es necesario desarrollar las operaciones de mutación y reproducción específicas para el problema.

- **Codificación en árbol**: la codificación en árbol es utilizada para codificación de expresiones o programas, para computación evolutiva. En esta codificación cada cromosoma es un árbol con ciertos objetos. Este tipo de programación es buena para evolucionar programas de ordenador.

Reproducción

La reproducción consiste en la creación de un nuevo individuo mediante la recombinación del material genético de dos individuos. Aunque la reproducción es uno de los procesos estándar en los Algoritmos Genéticos, muchas de las estrategias de reproducción están creadas para codificaciones binarias. La idea que existe detrás de este operador es que un individuo mejor será creado tomando las mejores partes del genoma de sus dos progenitores. El estado del arte de esta estrategia se describe en los siguientes apartados.

- **N-puntos**: es un operador de reproducción que aleatoriamente escoge n puntos en un cromosoma e intercambia el material genético entre esos puntos para producir el nuevo individuo, como podemos ver en 3.12
- **Uniforme**: este operador decide con un cierta probabilidad dada (ratio de combinación) que padre contribuirá con que gen en el nuevo individuo. Esto permite a los progenitores recombinarse a nivel de gen, mientras que en n-puntos la recombinación se produce a nivel de segmento.
- **Heurística**: en este operador se usa el valor de adaptación de cada progenitor para determinar la dirección de la recombinación, que se producirá en la dirección del progenitor con un nivel de adaptación más alto. Dicho nivel de adaptación se multiplica por un ratio aleatorio.

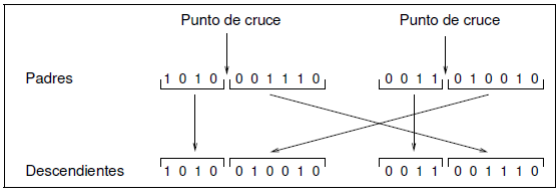


Figura 3.12: Ejemplo de cruce con estrategia 1-punto

Mutación

La mutación consiste en la alteración ocasional y aleatoria de pequeñas partes de un cromosoma. Dicha operación tiene como objetivo la no pérdida de algún bit. Por ejemplo, usando la codificación binaria es posible que después de algunas generaciones la selección lleve a todos los bits en alguna posición a ser 0 o 1. Si esto pasa con el algoritmo convergiendo hacia una solución satisfactoria, entonces el algoritmo ha convergido prematuramente. Esto puede ser un problema especialmente con poblaciones muy pequeñas. Sin la mutación no tendríamos la posibilidad de reintroducir el bit perdido. Los operadores de mutación utilizados usualmente son:

- **Giro de bit**: simplemente invierte el valor de un gen previamente escogido. Este operador solo puede ser usado para codificaciones binarias, como vemos en 3.13.
- **Limitador**: se intercambia el bit con el del final o el del principio de la cadena (escogido aleatoriamente). Este tipo de operador solo se puede utilizar para genes de tipo entero o flotante.
- **Uniforme**: se reemplaza el gen con un gen obtenido con una distribución aleatoria uniforme entre los límites indicados para ese gen. Este tipo de operador solo se puede utilizar para genes de tipo entero o flotante.
- **Gaussiana**: este operador suma al gen elegido una unidad Gaussiana distribuida aleatoriamente. Este tipo de operador solo se puede utilizar para genes de tipo entero o flotante. A parte de la estrategia de mutación elegida es necesario garantizar un ratio de mutación pequeño para evitar que el proceso estocástico pase a ser aleatorio.

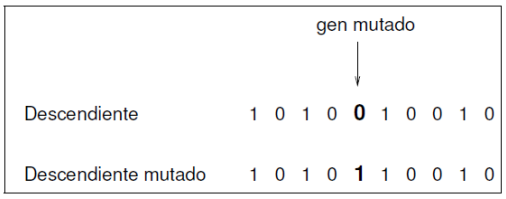


Figura 3.13: Ejemplo de mutación con giro de bit

Selección

Esta etapa se refiere a la simulación de la Selección Natural de forma computacional. En ésta, los individuos mejor adaptados sobreviven mientras que los menos adaptados desaparecen. El valor de adaptación es establecido se calcula utilizando la función de evaluación, que asigna un valor a cada a cada individuo basándose en su adaptación al problema. Son muchas las veces que se utilizan los términos función de evaluación y función de adaptabilidad como sinónimos sin embargo divergen en que una función de evaluación asigna un valor al individuo en términos de adaptación al problema, mientras que la función de adaptabilidad asigna el valor de adaptabilidad a un medida de oportunidades de reproducción. En los siguientes apartados son descritas las funciones de evaluación más utilizadas.

- **Ruleta**: es un proceso de elección de individuos proporcional a su valor de adaptación al medio. Es decir, a mayor valor de adaptación al medio, mayor probabilidad de ser elegido. Una de las desventajas de este proceso es que no se garantiza que el mejor individuos de esa generación pase a la siguiente.

- **Posición**: en este proceso de selección se ordenan los individuos por su valor de adaptación al medio y son escogidos para pasar a la siguiente generación los m primeros individuos, donde $m < I$ (siendo I el número de individuos de una generación).

- **Torneo**: en esta estrategia se escogen n individuos y se guarda el mejor adaptado, el modo de torneo más usado es el binario, donde sólo dos individuos son elegidos. Ya que la mayoría de procesos de selección suele estar basados en probabilidades, a menudo se introduce un valor

de elitismo que garantiza que los n individuos mejor adaptados pasaran a la siguiente generación.

Evolución

Una vez descritas las partes que trabajan conjuntamente en un AG, se describe la evolución simulada como el proceso iterativo de la máquina de estados 3.11, donde los individuos son llevados generación a generación a una adaptación al medio óptima (en el mejor de los casos). Los criterios de parada de la evolución han de ser definidos con conocimiento del problema y a menudo suelen estar relacionados con el tiempo de cómputo o con el número de generaciones que pueden pasar sin mejora de la adaptación media de la población. Un proceso evolutivo se puede describir de la siguiente manera.

- **Entrada**: Función de evaluación y parámetros necesarios
- **Repetir**:
- usar la función de evaluación para calcular la adaptabilidad de la población P_t.
- seleccionar a los padres de la población P_t
- utilizar reproducción para generar la población P_{t+1}
- **Hasta**: el mejor individuo es suficientemente bueno o se llega a un criterio de parada
- **Salida** : el mejor individuo de la última población

En algunos casos, se puede optar por utilizar un método híbrido, añadiendo una búsqueda exhaustiva al final del proceso evolutivo.

3.3.4. Aplicación algoritmos genéticos

El objetivo de este apartado es explicar como se aplican el algoritmo genético a nuestro problema de optimización, analizando las particularidades del diseño del algoritmo.

Aplicación de la codificación

El primer paso en la aplicación del algoritmo genético al problema de clasificación es definir el tipo de codificación que se utilizara. El primer problema de optimización a resolver es la codificación de la matriz ECOC sublineal, que se se codifican mediante una representación binaria como se muestra en 3.28.

$$\zeta =< h_1^{c_1}, \ldots, h_B^{c_1}, h_1^{c_2}, \ldots, h_B^{c_N} > \tag{3.28}$$

$$h_i^{c_j} \in \{0, 1\} \tag{3.29}$$

Donde cada individuo ECOC queda codificado en un cromosoma del tipo 3.28, donde cada h_i es el valor predicho del i-ésimo clasificador para la clase c_j que corresponde al i-ésimo bit de la palabra clave de la clase c_j.

En el caso de la optimización de los parámetros de los clasificadores SVM (los clasificadores base de los ECOC), el proceso es exactamente el mismo, es decir, el cromosoma es del tipo 3.30.

$$\zeta =< C, \gamma > \tag{3.30}$$

En este caso la codificación utilizada también es la binaria, donde cada cada alelo del cromosoma es la representación binaria de un valor en punto flotante.

Aplicación de la función de adaptación

Una vez la codificación queda definida, el siguiente paso es definir la función de evaluación que nos dirá como de adaptados están los individuos en el medio en el que se desenvuelven. Para nuestro problema, dicho valor de adaptación tiene que estar directamente relacionado con el error de clasificación.

Dado un cromosoma $\zeta =< \zeta_0, \zeta_1, \ldots, \zeta_L >$ con $\zeta_i \in \{0, 1\}$, el primer paso es recuperar la matriz ECOC sublineal que representa. Cada problema binario que genera dicha matriz es tratado por un clasificador base (máquina de soporte vectorial). Asumiendo que existe una función $y = f(\mathbf{x})$ que relaciona cada ejemplo $\mathbf{x}$ con su etiqueta real y, optimizar un clasificador significa encontrar los mejores parámetros w^* de una cierta función $y = f'(\mathbf{x}, \mathbf{w})$, de manera que para ningún otro w se producirá un error de clasificación menor.

Los parámetros $\mathbf{w}^*$ son estimados para cada problema binario y su función de adaptación corresponde directamente con su error de clasificación.

Para tener en cuenta el poder de generalización de los clasificadores base, la estimación de $\mathbf{w}^*$ es llevada a cabo en un subconjunto de los ejemplos mientras que los ejemplos restantes son reservados como conjunto de validación y el valor de adaptación es el error producido sobre el conjunto de validación.

Una vez el Algoritmo Genético converge hacia una solución esta se guarda ya que si en el proceso de evolución de la matriz ECOC sublineal, nos volvemos a encontrar una bipartición de datos que genere el mismo problema, no es necesario volver a optimizar los parámetros. Esta pequeña mejora hace que el tiempo de evolución de una matriz ECOC sublineal se reduzca ampliamente.

Aplicación del proceso evolutivo

Habiendo definido previamente la codificación y la función de adaptación de los individuos. Describimos el proceso evolutivo para evolucionar una matriz ECOC sublineal, como el proceso estándar de un Algoritmo Genético. Durante el proceso evolutivo utilizamos un operador de reproducción n-puntos visto en 3.3.3. Mientras que para el operador de mutación se utiliza un operador gaussiano explicado en 3.3.3.

Finalmente, adoptamos un esquema de evolución denominado *modelo de isla*. La idea general de este modelo es dividir la población de K individuos en S subpoblaciones de K/S individuos. Si cada subpoblación es evolucionada independientemente de las otras subpoblaciones cada evolución tendrá una dirección distinta. Introduciendo el concepto de migración, el modelo de isla puede explorar las diferencias entre las diferentes poblaciones y alcanzar un espacio de soluciones más amplio.

3.4. Diagramas de secuencia de sistema

A continuación se muestran los diagramas de secuencia de sistema del software implementado. Cada uno de ellos corresponde a un cierto evento del sistema.

En 3.14 podemos ver como el usuario interactua con la interfaz del sistema. Pidiendo obtener el mejor ECOC para un cierto problema.

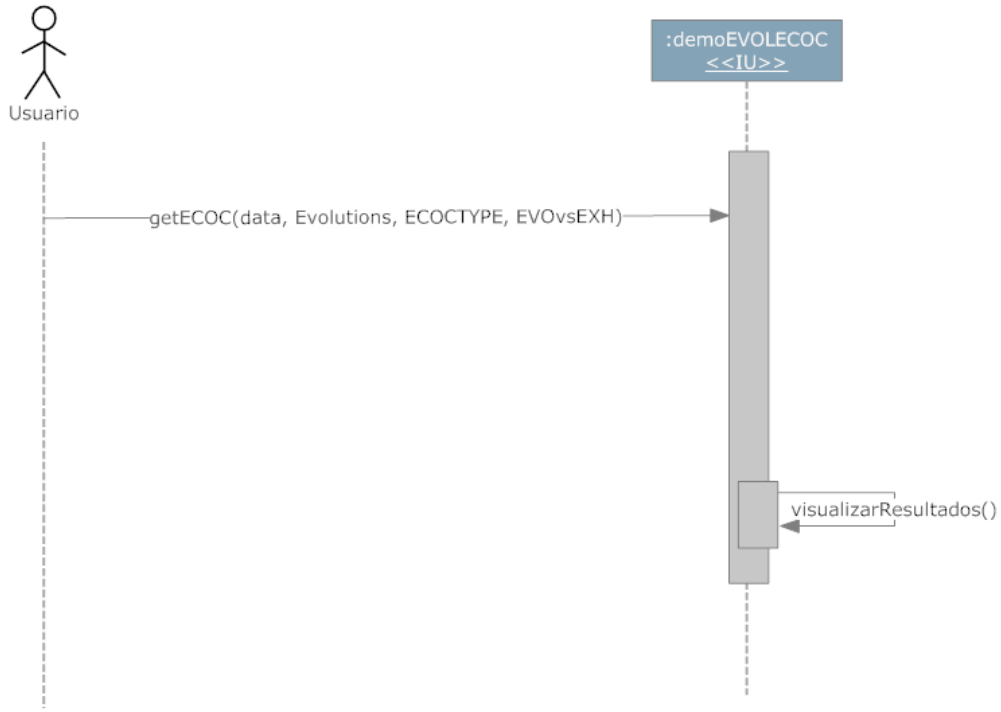


Figura 3.14: DSS_1: Interactuación del usuario con la interfaz del sistema

En 3.15 podemos ver como la interfaz de usuario se comunica con las demás clases y pone en marcha el proceso evolutivo.

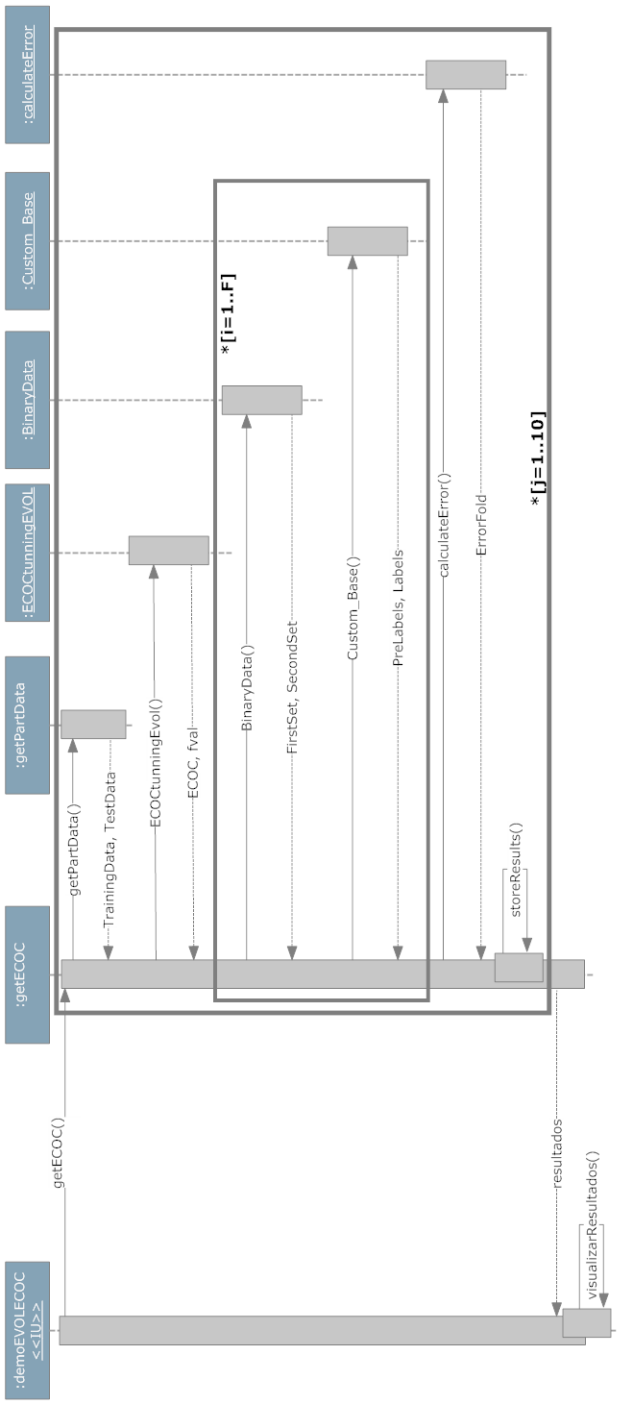


Figura 3.15: DSS_2: Interactuación de la interfaz con las funciones del sistema

En la imagen 3.16 podemos visualizar como evoluciona una matriz mediante la llamada al algoritmo genético (:ga) y como posteriormente se evalúa para poder obtener su error de clasificación.

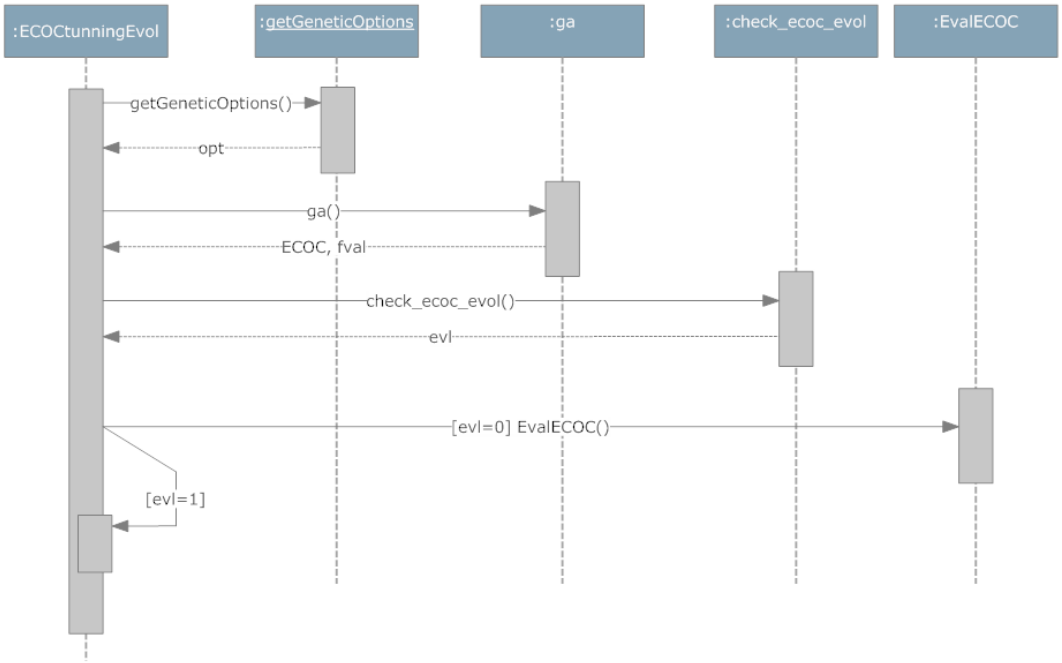


Figura 3.16: DSS_3: Evolución de una matriz ECOC sublineal

En el diagrama 3.17 podemos ver como una matriz ECOC es evaluada, optimizando los parámetros para cada uno de sus clasificadores base (SVM-RBF). Se puede observar la implementación del historial previamente comentado en 3.3.4 mediante las funciones *consult_row*() y *store_row*().

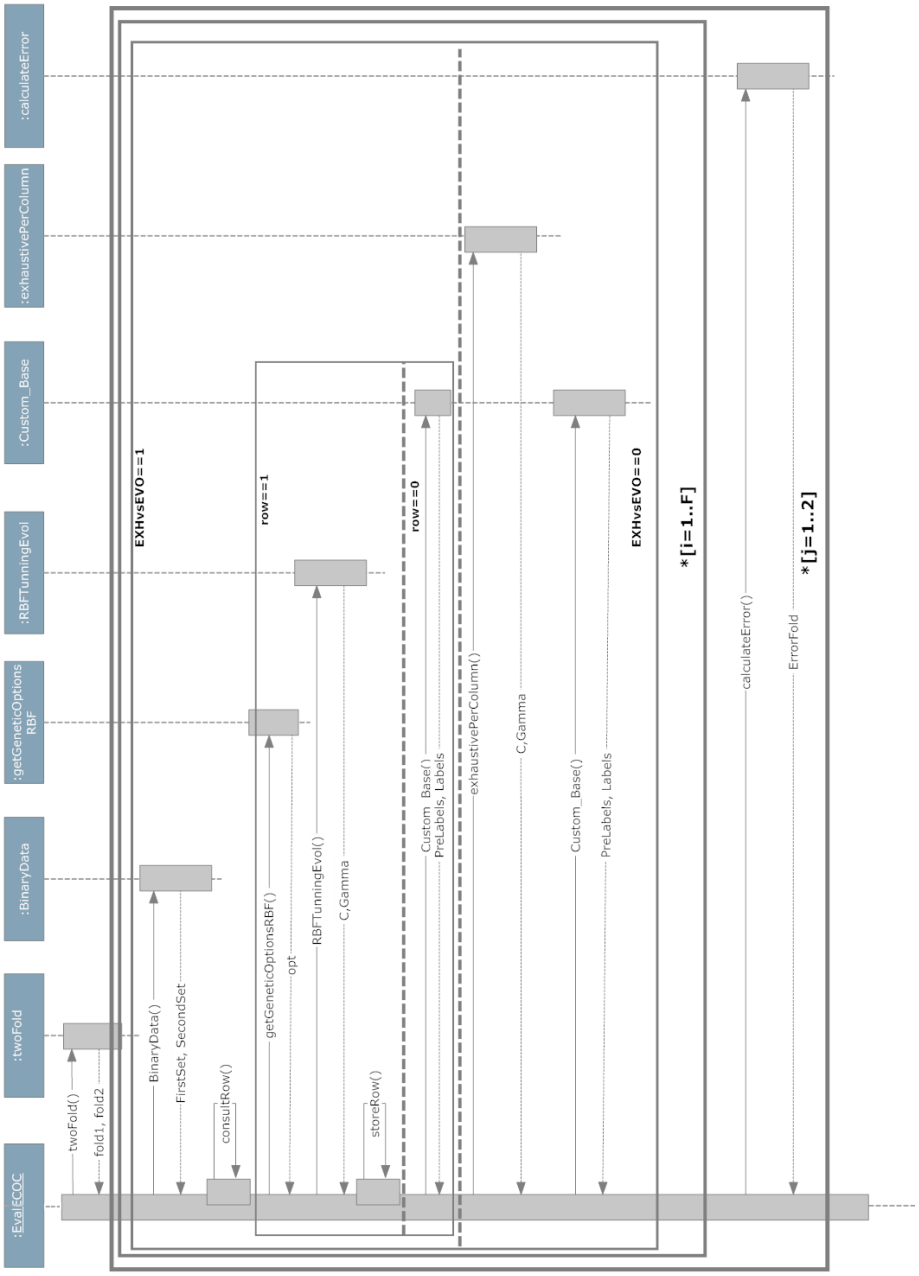


Figura 3.17: DSS_4: Evaluación de una matriz ECOC, con la correspondiente optimización de los parámetros del clasificador SVM

En la imagen 3.18 podemos observar la optimización de los parámetros C y γ mediante un proceso evolutivo.

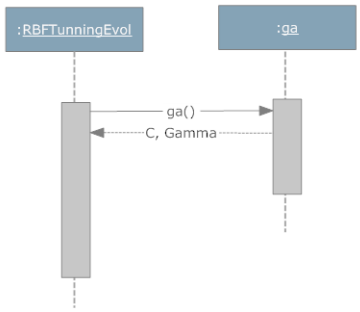


Figura 3.18: DSS_5: Evolución de los parámetros de un clasificador SVM-RBF

En el diagrama 3.19 podemos observar como se entrena un maquina de soporte vectorial y se obtiene sus predicciones, haciendo las llamadas a las funciones *SVMXXXX()* implementadas en la librería OSU-SVM.

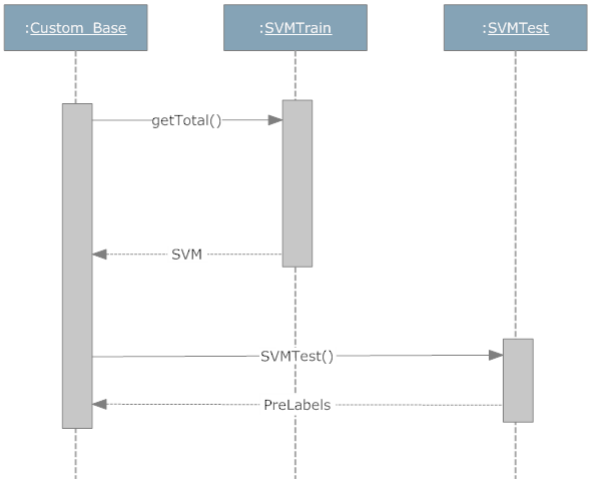


Figura 3.19: DSS_6: Evaluación de un clasificador SVM con unos parámetros fijados

3.5. Diagrama de colaboración

En la imagen 3.20 se muestra el diagrama de colaboración del sistema. Todas las funciones han sido implementadas en Matlab, tanto las correspondientes a los diseños de ECOC, los clasificadores SVM (implementados en una librería), como la programación evolutiva. El diagrama muestra la interacción entre las funciones que implementan el sistema. Como se puede apreciar, las diferentes notaciones se indican a partir de diferentes colores. Las funciones en color rojo corresponden la interfaz de usuario del sistema. Las funciones verdes son 'function handlers' que necesitan las funciones amarillas, ga, (algoritmo genético), implementadas por el propio Matlab. Las funciones azules son funciones de ayuda de su función padre editadas en el mismo script. Cada función en blanco tiene una funcionalidad concreta y está aislada en un script.

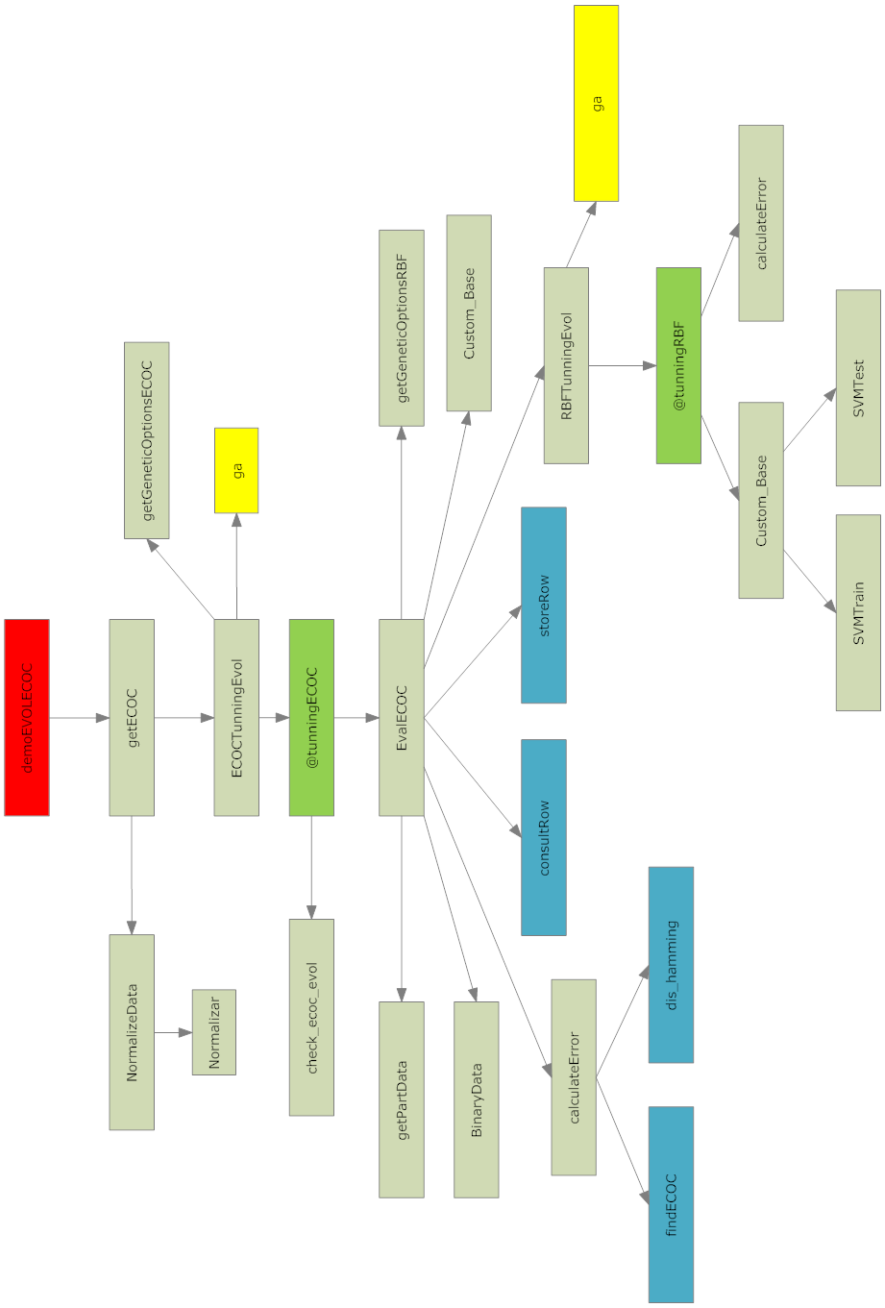


Figura 3.20: Diagrama de colaboración

3.6. Conclusión

En este apartado se hace una descripción de la aproximación utilizada por nuestro estudio para resolver los problemas de clasificación, y como dicha solución puede ser aplicable a los problemas de alta dimensionalidad. No obstante, procederemos a un análisis del estado del arte de las propuestas de resolución en este campo. Para empezar comentaremos que la resolución de los problemas de alta dimensionalidad se ha venido estudiando durante los últimos años sin llegar a una solución eficiente y computacionalmente viable. Nuestra aproximación pretende acabar con este problema dando un marco fiable con el que tratar dicho tipo de problemas.

Inicialmente, uno de los problemas a tratar es que el tiempo de cómputo de entrenamiento de los clasificadores SVM crece rápidamente con el número de ejemplos. La razón de esto es que el problema núcleo de los SVM es un problema de programación cuadrática. Los resolutores de problemas de programación cuadrática tienden normalmente a tener una escalabilidad de $O(n^3)$, así pues, la manera de entrenar un SVM de forma eficiente es aún un problema a resolver por los investigadores. En un principio se optó por una computación paralela de los SVM donde el conjunto de aprendizaje se particionaba en diferentes subconjuntos y cada uno de estos se entrenara paralelamente. En la partición de el conjunto total de aprendizaje se entrena una red neuronal para participar dicho conjunto de forma óptima. Esta estrategia de resolución se dio por obsoleta dada la dependencia de los datos en el entrenamiento de los SVM y su incompatibilidad con los sistemas distribuidos.

Estudios posteriores, proponen marcos de computación paralelas para resolver de forma viable los problemas de programación cuadrática en el entrenamiento de los SVM con un conjunto de entrenamiento de alta dimensionalidad, por ejemplo, el entrenamiento de SVM en cascada. En este sistema se entrenan inicialmente un número reducido de clasificadores, estos resultados son combinados después en un proceso jerárquico, como el descrito en 3.21, donde el resultado del último clasificador es utilizado como retroalimentación del sistema.

Esta estrategia sí que es viable para ser utilizada en sistemas distribuidos, y la retroalimentación hace que el método de cascada mejorada en [P.Z05] muestre en sus experimentos unos resultados eficientes, pero aún tiene problemas a resolver, como la partición óptima del conjunto de entrenamiento.

Otra estrategia para abordar el problema del entrenamiento de los SVM

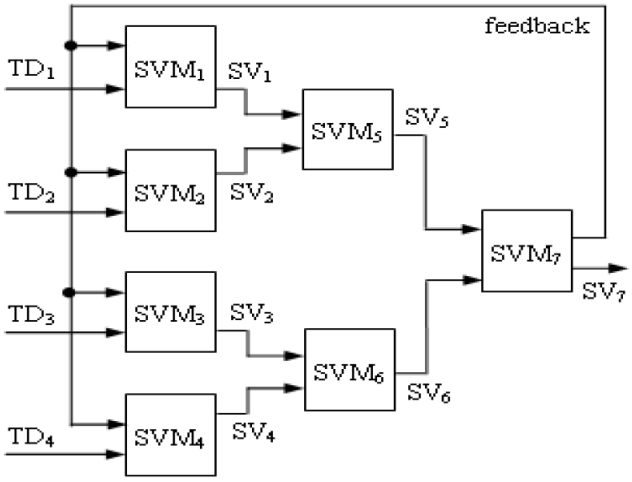


Figura 3.21: Algoritmo de entrenamiento en cascada de SVM

han sido los denominados IRSVM (Incremental Reduced Support Vector Machines), que comienzan su entrenamiento con un conjunto de entrenamiento muy reducido, que se incrementa ligeramente, basándose en una serie de pequeños problemas de programación cuadrática. El tamaño del primer conjunto reducido de aprendizaje es determinado dinámicamente por el algoritmo. Los resultados demostraron que el uso de esta estrategia mejora el rendimiento en generación de clasificadores y kernels no lineales.

Así pues, vistos los estudios en este campo, se propone la alternativa de un *Diseño Evolutivo sublineal del ECOC*, que pretende mejorar los estudios existentes en los siguientes puntos:

- **ECOC** Dicho marco de trabajo presenta mejoras sustanciales, en lo que a problemas de multiclasificación se refiere, a otros marcos como por ejemplo el PMC 2.1. Así pues, será éste el marco para tratar los problemas de multiclasificación.

- **SVM** Los clasificadores SVM y en concreto la utilización de un kernel RBF son los clasificadores que nos han ofrecido un rendimiento y una fiabilidad más grande en problemas de clasificación binaria.

- **Sublineal** Las estrategias previas utilizaban codificaciones 'uno contra

todos' o 'aleatoria dispersa' (entre otras), con lo que el número de clasificadores necesarios para que el problema cumpliese las condiciones ECOC eran enormes para problemas de alta dimensionalidad. Con esta nueva codificación sólo son necesarios $log_2 N$ clasificadores, donde N es el número de clases.

- **Evolutivo** Las estrategias de optimización de los SVM distan mucho de ser procesos que optimicen los parámetros en un tiempo computacionalmente viable y que no se estanquen en mínimos locales. La computación evolutiva nos ofrece una potente arma para delegar la optimización asegurándonos que no se tratara el problema desde una perspectiva exhaustiva.

Capítulo 4

Resultados

En este apartado se presentan los experimentos realizados con el objetivo de demostrar el rendimiento del sistema. Para explicarlos primero discutiremos las bases de datos a clasificar y posteriormente los distintos métodos que se utilizan, para acabar presentando los resultados.

4.1. Bases de datos

En la primera fase de experimentos se han utilizado un conjunto de 12 bases de datos multiclase del repositorio UCI de aprendizaje automático [AN07]. Estas bases de datos son estándar en experimentos de validación en problemas de clasificación. El número de ejemplos, características y clases por base de datos se muestra en 4.1.

Problema	#Ejemplos de entrenamiento	#Características	#Clases
Dermathology	366	34	6
Iris	150	4	3
Ecoli	336	8	8
Vehicle	846	18	4
Wine	178	13	3
Segmentation	2310	19	7
Glass	214	9	7
Thyroid	215	5	3
Vowel	990	10	11
Balance	625	4	3
Shuttle	14500	9	7
Yeast	1484	8	10

Cuadro 4.1: Características de las bases de datos del repositorio UCI.

Después aplicamos la metodología en 4 problemas complejos de visión

por computador. Primero utilizamos secuencias de vídeo obtenidas por un sistema de mapeo móvil para testar los métodos en un problema real de catequización de señales de tráfico consistente en 36 clases. Segundo, 20 clases de la base de datos ARFaces son clasificadas utilizando la metodología explicada anteriormente. Tercero, clasificamos 7 símbolos de documentos musicales antiguos, y cuarto, clasificamos las 70 categorías visuales de la bases de datos publica MPEG.

4.2. Métodos

Para validar los resultados del sistema comparamos las codificaciones de ECOC 'uno contra uno', 'uno contra todos' y ECOC sublineal binario, con la estrategia sublineal evolutiva . En estos experimentos se utiliza una decodificación Hamming 3.1.3. El clasificador base del ECOC es la implementación OSU de las máquinas de soporte vectorial con kernel RBF. Los parámetros C y γ de todos los SVM son ajustados mediante un proceso evolutivo minimizando el error de clasificación de una evaluación 2-fold sobre un subconjunto de entrenamiento.

4.3. Resultados de bases de datos del repositorio de aprendizaje automático UCI

En la tabla 4.2 se comparan los resultados de clasificación de las distintas codificaciones ECOC y el número de clasificadores que se han necesitado para cada una de ellas. Finalmente se calcula el ranking de las codificaciones ECOC así como el número medio de clasificadores usados por cada una de ellas.

Estos resultados confirman que las 4 estrategias de codificación son adecuadas para tratar con problemas de multiclasificación. Este resultado es ampliamente satisfactorio dado que la aproximación sublineal evolutiva alcanza resultados similares (o incluso mejores) que los resultados obtenidos por las demás estrategias, con un número arbitrariamente más pequeño de clasificadores.

	ECOC sub. binario		ECOC sub. evol.		ECOC 1vsTodos		ECOC 1vs1	
Base de Datos	Rend.	Clasf.	Rend.	Clasf.	Rend.	Clasf.	Rend.	Clasf.
Derma	96.0±2.9	3	96.3±2.1	3	95.1±3.3	6	94.7±4.3	15
Iris	96.4±6.3	2	98.2±1.9	2	96.9±6.0	3	96.3±3.1	3
Ecoli	80.5±10.9	3	81.4±10.8	3	79.5±12.2	8	79.2±13.8	28
Vehicle	72.5±14.3	2	76.99±12.4	2	74.2±13.4	4	83.6±10.5	6
Wine	95.5±4.3	2	97.2±2.3	2	95.5±4.3	3	97.2±2.4	3
Segment	96.6±2.3	3	96.6±1.5	3	96.1±1.8	7	97.18±1.3	21
Glass	56.7±23.5	3	50.0±29.7	3	53.85±25.8	6	60.5±26.9	15
Thyroid	96.4±5.3	2	93.8±5.1	2	95.6±7.4	3	96.1±5.4	3
Vowel	57.7±29.4	3	81.78±11.1	3	80.7±11.9	8	78.9±14.2	28
Balance	80.9±11.2	2	87.1±9.2	2	89.9±8.4	3	92.8±6.4	3
Shuttle	80.9±29.1	3	83.4±15.9	3	90.6±11.3	7	86.3±18.1	21
Yeast	50.2±18.2	4	54.7±11.8	4	51.1±18.0	10	52.4±20.8	45
Pos. media & # Clasf.	2.9	**2.7**	**2.0**	**2.7**	2.7	5.7	2.2	15.9

Cuadro 4.2: Resultados de clasificación sobre bases de datos UCI

Además, la versión sublineal evolutiva es la mejor posicionada en el ranking de estrategias, mejorando en la mayoría de ocasiones el rendimiento de la estrategia sublineal binaria. Este resultado sugiere que la estrategia evolutiva es mejor ya que dicha codificaciones minimiza el error sobre el conjunto de aprendizaje, incrementando de esta manera la capacidad de generalización. En particular, la ventaja de la versión evolutiva sublineal sobre la sublineal binaria se hará notar cuanto mayor sea el número de clases en el problema, ya que existirán un número mayor de matrices sublineales.

Por otra parte, posibles razones por las que el diseño sublineal evolutivo tenga rendimientos similares a las otras estrategias utilizando un número mucho más pequeño de clasificadores base pueden ser:

- El pequeño número de clasificadores que se han de optimizar.
- La resolución de problemas binarios con un número de clases balanceadas.

Cabe recalcar que el proceso evolutivo de la matriz sublineal no es muy rápido, ya que se han de generar cierto número de matrices a evaluar, pero para problemas de muy alta cardinalidad este tiempo es mucho más reducido que el tiempo que se tardaría en entrenar una estrategia 'uno contra todos'. Además, como el tiempo de clasificación es directamente proporcional al número de clasificadores, la estrategia sublineal evolutiva se hace atractiva para tratar problemas en tiempo real, como se puede observar en el número de clasificadores base utilizados por cada estrategia mostrado en la ultima fila de la tabla 4.2. Es importante observar la gran diferencia en términos de cantidad de clasificadores entre la estrategias sublineales y las clásicas. Las aproximaciones sublineales obtienen una mejora de velocidad del 111 % en

comparación con la estrategia 'uno contra todos'. Mientras que en el caso de la estrategia 'uno contra uno' esta mejora es del 489 %.

4.4. Caso particular: base de datos vowel

Para tener una comprensión más profunda de cómo funciona el sistema se propone el análisis de una base de datos en concreto, en este caso, Vowel. En este apartado se hará un análisis del proceso evolutivo de optimización de los SVM para esta base de datos.

Asumamos una columna de ECOC fija F_{ECOC}, aplicaremos al problema de clasificación binaria resultante de la bipartición de dicha columna, una maquina de soporte vectorial lineal y una maquina de soporte vectorial con un kernel RBF. La búsqueda de los parámetros C y γ óptimos se hace de forma exhaustiva acotando el parámetro $C \in [0, 50]$ y el parámetro γ $C \in [0, 5]$, de manera que se obtiene la figura 4.1.

Figura 4.1: Topología del espacio del parámetro C en un SVM

Como podemos ver, en la ilustración 4.1 se pone de manifiesto la linealidad, perdonando la redundancia, de un SVM sin mapeado de alta dimensionalidad. En este ejemplo se puede ver como para distintos grupos de valores

del parámetro C obtenemos varias franjas de error. Cabe recalcar también que dicho error sufre una oscilación del $0,5\,\%$ del parámetro que optimiza la clasificación al parámetro que muestra una menor eficiencia.

Realizando exactamente el mismo proceso para un SVM con un kernel RBF obtenemos el gráfico 4.2:

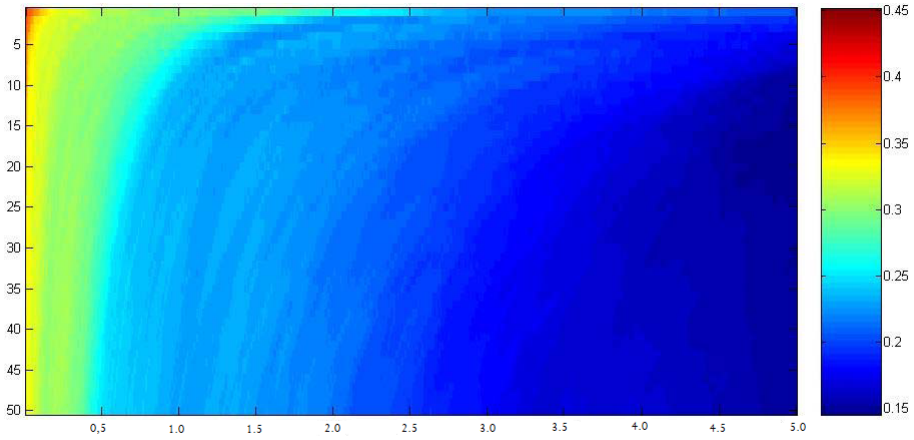


Figura 4.2: Topología del espacio de los parámetros C y γ en un SVM con kernel RBF

Observando la imagen, se hacen notables 2 características. La primera, es el descenso del error de un 33,8 %a un 14,8 %, esto nos hace pensar que la distribución que nos daba la columna F_{ECOC} para este clasificador no era linealmente clasificable, de ahí que obtuviésemos un error tan grande. No obstante, mediante la utilización de un kernel RBF podemos observar cómo se reduce el error. Por otra parte, comentar también el hecho de la suavidad en la variación del error, lo cual hace que este espacio de búsqueda sea propicio para que el evolutivo converja hacia unos parámetros subóptimos en el peor de los casos. Como podemos ver en la figura 4.3 donde se pone de manifiesto la convergencia del proceso evolutivo de optimización hacia soluciones subóptimas.

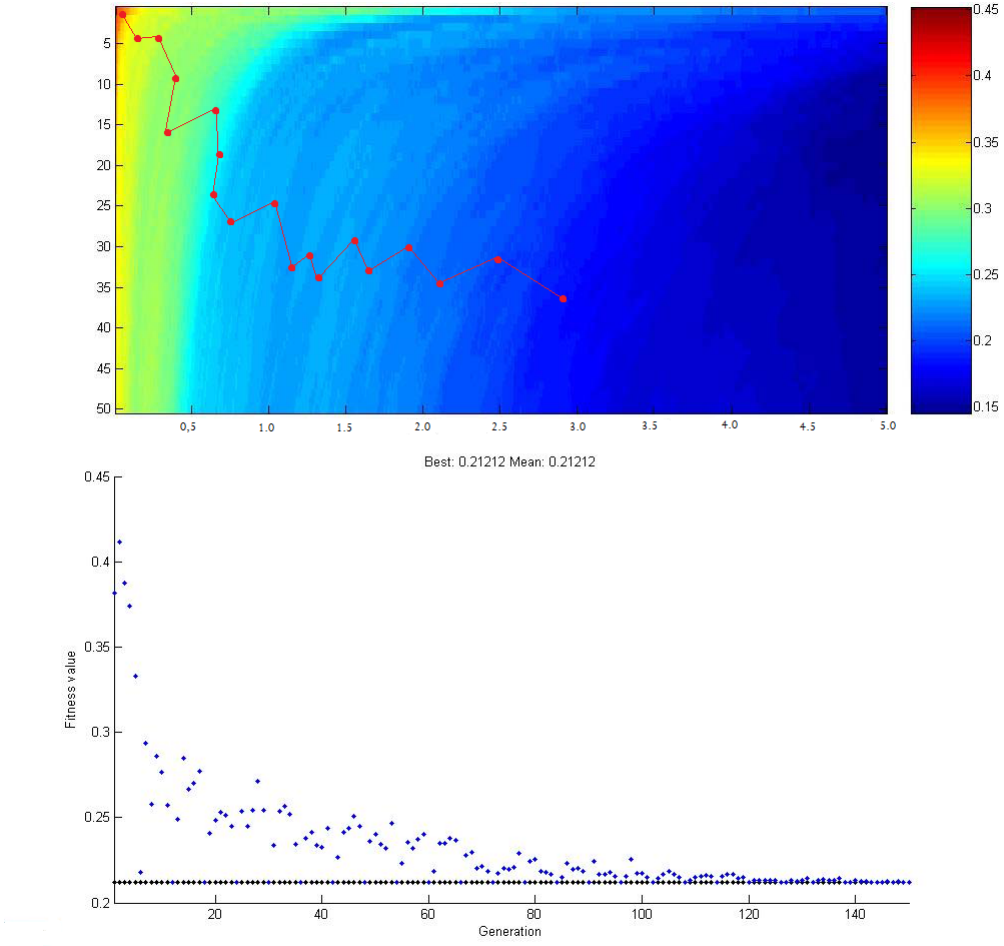


Figura 4.3: Visualización del proceso de optimización del algoritmo genético

De esta manera justificamos la optimización de los parámetros de una maquina de soporte vectorial con kernel RBF mediante la utilización de algoritmos genéticos.

4.5. Resultados de bases de datos de problemas de visión por computador

Una de las aplicaciones más características de los problemas de clasificación automática es la categorización y reconocimiento de objetos visuales mediante técnicas de visión por computador. De esta manera se pretende resolver problemas reales de visión por computador para mostrar la ejecutividad del sistema en entornos donde el número de clases a clasificar empieza a ser poco manejable para las demás estrategias. Se han escogido 4 bases de datos de problemas de visión por computador, las cuales se describen a continuación.

- **Traffic**: esta base de datos contiene 36 clases, cada una de las cuales corresponde a una señal de trafico captada por un sistema de grabación móvil. En general la base de datos contiene 3481 ejemplos (imágenes de 32×32 pixeles), como se puede observar en 4.4

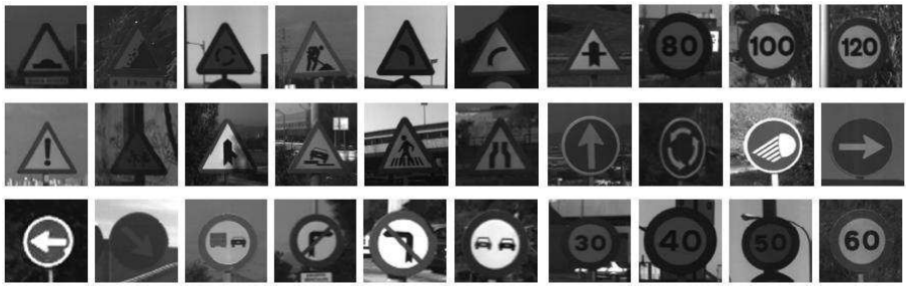
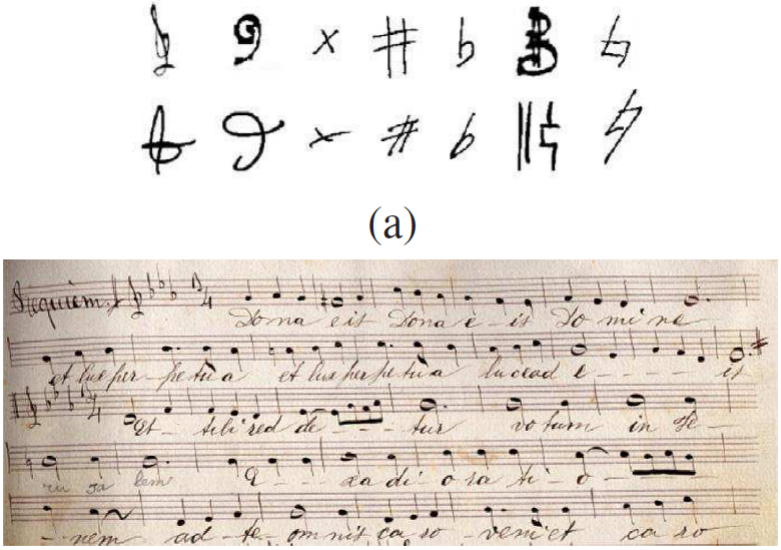


Figura 4.4: Base de datos de imágenes señales de tráfico

- **ARFaces**: este problema de visión por computador esta compuesto por 26 imágenes faciales de 126 individuos (56 mujeres y 70 hombres), todas las imágenes tienen un fondo blanco, como se puede comprobar en 4.5.

Figura 4.5: Base de datos de imágenes faciales

- **Claves**: esta base de datos contiene imágenes de documentos musicales modernos y antiguos, de los cuales se extrajeron las notas escritas mediante un sistema de segmentación. La base de datos contiene un total de 4098 ejemplos divididos en 7 clases diferentes, mostrado en la imagen 4.6.



(a)

(b)

Figura 4.6: Base de datos de claves. En la imagen (a) tenemos las claves y en la (b) los documentos escaneados

- MPEG7: esta base de datos contiene 70 clases, cada una de ellas con 20 ejemplos, lo que hacen un total de 1400 imágenes. Todas las imágenes son descritas mediante un descriptor BSM. En la imagen 4.7 se puede observar un ejemplo de esta base de datos.

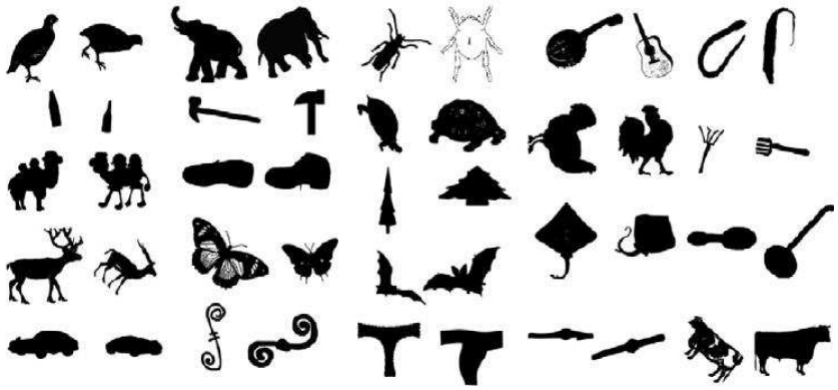


Figura 4.7: Base de datos de imágenes de objetos

En la tabla 4.3 podemos ver los resultados obtenidos por las distintas estrategias y el número de clasificadores necesario para cada una de ellas.

	ECOC sub. binario		ECOC sub. evol.		ECOC 1vsTodos		ECOC 1vs1	
Base de Datos	Rend.	Clasf	Rend.	Clasf	Rend.	Clasf	Rend.	Clasf
Traffic	90.8±4.1	6	90.6±3.4	6	91.8±4.6	36	90.6±4.1	630
ARFaces	76.0±7.2	5	85.84±5.2	5	84.0±6.3	20	96.0±2.5	190
Clefs	81.2±4.2	3	81.8±9.3	3	80.8±11.2	7	84.2±6.8	21
MPEG7	89.29±5.1	7	90.4±4.5	7	87.8±6.4	6.170	92.8±3.7	2415
Pos, media & # Clasf.	3.0	**5.2**	2.2	**5.2**	3.0	33.2	**1.5**	814.0

Cuadro 4.3: Resultados sobre problemas de visión por computador

Estos resultados sugieren que las 4 estrategias son adecuadas para tratar con problemas de multiclasificación en entornos de visión por computador. A su vez, dichos resultados avalan la utilización de una codificación sublineal. Notar que por ejemplo en la base de datos MPEG la aproximación sublineal sería capaz de llevar a cabo tareas de clasificación en tiempo real, mientas que la estrategia 'uno contra uno' con sus más de 2000 clasificadores requeridos, hace este tipo de aplicación inviable.

Los resultados en general sugieren que la aproximación sublineal evolutiva puede ser aplicada en problemas de muy alta cardinalidad donde se tuviese que tratar con miles de categorías. En esos casos incluso bases de datos con 16000 categorías podrían ser aprendidas con solo 14 clasificadores base, en contraposición con los miles de clasificadores requeridos por las estrategias del estado del arte.

Capítulo 5

Conclusiones y trabajo futuro

En este capitulo se reúnen las conclusiones hechas sobre el estudio, así como las posibles líneas de investigación futuras.

5.1. Conclusiones

En este estudio se presenta una metodología general para problemas de clasificación automática de múltiples categorías, que solo requiere $log_2 N$ clasificadores base para un problema de N clases. La metodología es definida en el marco de trabajo de los códigos correctores de errores, en el que se diseña una matriz de codificación mímica que distingue unívocamente N códigos. Para llevar a cabo la codificación de esta matriz sublineal se aplica un algoritmo genético el cual codifica una matriz sublineal que minimiza el error para el conjunto de aprendizaje. Además, el proceso de optimización de los clasificadores base de la matriz ECOC también es delegado en un algoritmo genético.

El sistema se pone a prueba sobre un amplio conjunto de bases de datos del repositorio de aprendizaje automático UCI [AN07] y 4 problemas de clasificación de visión por computador. Los resultados obtenidos muestran que la metodología propuesta obtiene resultados similares e incluso mejores que las estrategias del estado del arte para codificación ECOC con un numero de clasificadores mucho mas pequeño. Por ejemplo, la codificación sublineal entrena 6 clasificadores base para distinguir 50 categorías faciales, mientras que la codificación 'uno contra todos' y 'uno contra uno' requieren 50 y 1225 clasificadores, respectivamente.

En lo que nuestro conocimiento alcanza, es la primera vez que un numero de clasificadores sublineal con el numero de clases es utilizado para categorizar datos de alta cardinalidad.

5.2. Trabajo futuro

Como trabajo futuro, queremos aplicar la estrategia sublineal sobre problemas de muy alta cardinalidad donde un conjunto de 16000 categorías, podría ser aprendido utilizando 14 clasificadores, mientras que las estrategias actuales utilizarían miles.

Una posible actualización del sistema sería la inclusión de nuevas columnas en la matriz de codificación sublineal, que se centrasen en las categorías con un error de clasificación mas alto. De esta manera minimizaríamos el error manteniendo un numero de clasificadores compacto.

Por otra parte se debe estudiar la inclusión de una nueva implementación de las maquinas de soporte vectorial, que reduzcan el tiempo de entrenamiento.

5.3. Anexos

5.3.1. Anexo A : contenido del CD

El contenido del CD consta de una carpeta '/src' que contiene los scripts que implementan el proyecto. Para utilizarlo simplemente se ha de ejecutar el script 'demoEVOLECOC' con el nombre de la base de datos que se quiera probar.

Bibliografía

[ACSS02] N.E. Ayat, M. Cheriet, C.Y. Suen, and M. Cheriet C. Y. Suen. Empirical error based optimization of svm kernels: Application to digit image recognition. In *In the 8 th IWFHR, Niagaraon-the-lake*, pages 105–110. Springer Verlag, 2002.

[Alp04] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[AN07] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[AR06] George Alexander and Anita Raja. The role of problem classification in online meta-cognition. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 218–225, Washington, DC, USA, 2006. IEEE Computer Society.

[BZ09] T Aráz E. Buck and Bin Zhang. Svm kernel optimization: An example in yeast protein subcellular localization prediction. 2009.

[CPA08] Yuehui Chen, Lizhi Peng, and Ajith Abraham. Hierarchical radial basis function neural networks for classification problems. 2008.

[CS00] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *In Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 35–46, 2000.

[Dar] Charles Darwin. *The Origin of Species*. Gramercy.

[DB94] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. Artif. Int. Res.*, 2(1):263–286, 1994.

[EPR08] Sergio Escalera, Oriol Pujol, and Petia Radeva. On the decoding process in ternary error-correcting output codes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):120–134, 2008.

[Gha02] Rayid Ghani. Combining labeled and unlabeled data for multiclass text categorization. In *In Proceedings of the International Conference on Machine Learning*, pages 187–194, 2002.

[KGWM01] J Kittler, R Ghaderi, T Windeatt, and J Matas. Face verification using error correcting output codes. In *In Computer Vision and Pattern Recognition CVPR01*, pages 755–760. IEEE Press, 2001.

[LdC08] Ana Carolina Lorena and André C. P. L. F. de Carvalho. Evolutionary tuning of svm parameter values in multiclass problems. *Neurocomput.*, 71(16-18):3326–3334, 2008.

[LM06] Dalton Lunga and Tshilidzi Marwala. Online forecasting of stock market movement direction using the improved incremental algorithm. In *ICONIP (3)*, pages 440–449, 2006.

[Pol06] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[PRV06] Oriol Pujol, Petia Radeva, and Jordi Vitria. Discriminant ecoc: A heuristic method for application dependent design of error correcting output codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1007–1012, 2006.

[P.Z05] J.Yang P.Zhang, Z.Li. A parallel svm training algorithm on large escale classification problems. 2005.

[RG04] Suju Rajan and Joydeep Ghosh. An empirical comparison of hierarchical vs. two-level approaches to multiclass problems. In *in Lecture Notes in Computer Science*, pages 283–292. Springer, 2004.

[RK04] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, 2004.

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[SVM03] OSU SVM. Osu svm classifier matlab toolbox, 2003.

[WG02] Terry Windeatt and Reza Ghaderi. Coding and decoding strategies for multi-class learning, 2002.